

Contents

1 3-Way QuickSort (Dutch National Flag)	26
Source	31
2 3-way Merge Sort	32
Source	36
3 8086 program for selection sort	37
Source	39
4 A Pancake Sorting Problem	40
Source	45
5 A sorting algorithm that slightly improves on selection sort	46
Source	48
6 Absolute distinct count in a sorted array	49
Source	53
7 Alternate Lower Upper String Sort	54
Source	59
8 Alternate sorting of Linked list	60
Source	65
9 Alternative Sorting	66
Source	71
10 An Insertion Sort time complexity question	72
Source	73
11 Asymptotic Analysis and comparison of sorting algorithms	74
Source	80
12 Bead Sort A Natural Sorting Algorithm	81
Source	84
13 Binary Insertion Sort	85
Source	89

14 Bitonic Sort	90
Source	98
15 BogoSort or Permutation Sort	99
Source	103
16 Bubble Sort	104
Source	118
17 Bubble Sort On Doubly Linked List	119
Source	122
18 Bubble sort using two Stacks	123
Source	125
19 Bucket Sort	126
Source	137
20 Bucket Sort To Sort an Array with Negative Numbers	138
Source	140
21 C Program for Binary Insertion Sort	141
Source	141
22 C Program for Bubble Sort	143
Source	143
23 C Program for Bubble Sort on Linked List	145
Source	148
24 C Program for Recursive Insertion Sort	149
Source	149
25 C Program to Sort an array of names or strings	151
Source	152
26 C program to sort an array of strings using Selection Sort	153
Source	155
27 C qsort() vs C++ sort()	156
Source	159
28 C++ Program for Bubble Sort	160
Source	161
29 C++ Program for Cycle Sort	162
Source	163
30 C++ Program for QuickSort	166
Source	167

31 C++ Program for Recursive Bubble Sort	169
Source	170
32 C++ Program for Stooge Sort	172
Source	172
33 C++ program for Sorting Dates using Selection Sort	174
Source	175
34 Can QuickSort be implemented in O(nLogn) worst case time complexity?	176
Source	180
35 Cartesian Tree Sorting	181
Source	188
36 Check if a grid can become row-wise and column-wise sorted after adjacent swaps	189
Source	191
37 Check if a queue can be sorted into another queue using a stack	192
Source	199
38 Check if any interval completely overlaps the other	200
Source	202
39 Check if any two intervals overlap among a given set of intervals	203
Source	207
40 Check if array contains contiguous integers with duplicates allowed	208
Source	222
41 Check if both halves of the string have at least one different character	223
Source	235
42 Check if given array is almost sorted (elements are at-most one position away)	236
Source	241
43 Check if it is possible to sort an array with conditional swapping of adjacent allowed	242
Source	247
44 Check if linked list is sorted (Iterative and Recursive)	248
Source	251
45 Check if reversing a sub array make the array sorted	252
Source	255
46 Check if the given permutation is a valid DFS of graph	256
Source	260

47 Check if two arrays are equal or not	261
Source	269
48 Check whether Arithmetic Progression can be formed from the given array	270
Source	276
49 Check whether a given array is a k sorted array or not	277
Source	281
50 Check whether a given number is even or odd	282
Source	289
51 Check whether an array can be fit into another array rearranging the elements in the array	290
Source	294
52 Chocolate Distribution Problem	295
Source	303
53 Choose k array elements such that difference of maximum and minimum is minimized	304
Source	309
54 Circle Sort	310
Source	313
55 Closest numbers from a list of unsorted integers	314
Source	319
56 Closest product pair in an array	320
Source	328
57 Cocktail Sort	329
Source	337
58 Comb Sort	338
Source	352
59 Comparator function of qsort() in C	353
Source	356
60 Comparisons involved in Modified Quicksort Using Merge Sort Tree	357
Source	362
61 Convert an array to reduced form Set 1 (Simple and Hashing)	363
Source	367
62 Convert an array to reduced form Set 2 (Using vector of pairs)	368
Source	370

63 Count Inversions in an array Set 1 (Using Merge Sort)	371
Source	379
64 Count all distinct pairs with difference equal to k	380
Source	396
65 Count distinct occurrences as a subsequence	397
Source	402
66 Count minimum number of subsets (or subsequences) with consecutive numbers	403
Source	408
67 Count number of triplets in an array having sum in the range [a, b]	409
Source	423
68 Count number of triplets with product equal to given number Set 2	424
Source	425
69 Count of index pairs with equal elements in an array	429
Source	434
70 Count pairs from two linked lists whose sum is equal to a given value	435
Source	447
71 Count pairs from two sorted arrays whose sum is equal to a given value x	448
Source	467
72 Count pairs from two sorted matrices with given sum	468
Source	490
73 Count pairs in an array that hold $i * arr[i] > j * arr[j]$	491
Source	502
74 Count points covered by given intervals	503
Source	506
75 Count quadruples from four sorted arrays whose sum is equal to a given value x	507
Source	515
76 Counting Sort	516
Source	531
77 Counting cross lines in an array	532
Source	539
78 Cycle Sort	540
Source	548
79 De-arrangements for minimum product sum of two arrays	549

Source	551
80 Delete consecutive same words in a sequence	552
Source	557
81 Difference between highest and least frequencies in an array	558
Source	564
82 Dual pivot Quicksort	565
Source	568
83 Efficiently merging two sorted arrays with O(1) extra space	569
Source	577
84 Elements that occurred only once in the array	578
Source	591
85 Elements to be added so that all elements of a range are present in array	592
Source	597
86 Equally divide into two sets such that one set has maximum distinct elements	598
Source	601
87 External Sorting	602
Source	610
88 Find Sum of all unique sub-array sum for a given array.	611
Source	614
89 Find Surpasser Count of each element in array	615
Source	623
90 Find a pair of elements swapping which makes sum of two arrays same	624
Source	638
91 Find a pair with maximum product in array of Integers	639
Source	649
92 Find a pair with the given difference	650
Source	654
93 Find a permutation that causes worst case of Merge Sort	655
Source	662
94 Find a triplet that sum to a given value	663
Source	677
95 Find all triplets with zero sum	678
Source	689

96 Find all elements in array which have at-least two greater elements	690
Source	702
97 Find array with k number of merge sort calls	703
Source	709
98 Find distinct elements common to all rows of a matrix	710
Source	716
99 Find elements larger than half of the elements in an array	717
Source	721
100 Find first k natural numbers missing in given array	722
Source	727
101 Find four elements that sum to a given value Set 2 (O(n^2Logn) Solution)	728
Source	732
102 Find if k bookings possible with given arrival and departure times	733
Source	735
103 Find k maximum elements of array in original order	736
Source	739
104 Find k-th smallest element in given n ranges	740
Source	743
105 Find maximum height pyramid from the given array of objects	744
Source	753
106 Find memory conflicts among multiple threads	754
Source	757
107 Find minimum difference between any two elements	758
Source	766
108 Find missing elements of a range	767
Source	772
109 Find number of pairs (x, y) in an array such that x^y > y^x	773
Source	780
110 Find number of pairs in an array such that their XOR is 0	781
Source	790
111 Find pair with greatest product in array	791
Source	796
112 Find shortest unique prefix for every word in a given list Set 2 (Using Sorting)	797

Source	802
113 Find sum of non-repeating (distinct) elements in an array	803
Source	805
114 Find the Minimum length Unsorted Subarray, sorting which makes the complete array sorted	806
Source	813
115 Find the Sub-array with sum closest to 0	814
Source	818
116 Find the largest multiple of 3 from array of digits Set 2 (In O(n) time and O(1) space)	819
Source	823
117 Find the largest number that can be formed with the given digits	824
Source	833
118 Find the minimum and maximum amount to buy all N candies	834
Source	841
119 Find the point where maximum intervals overlap	842
Source	852
120 Find whether an array is subset of another array Added Method 3	853
Source	869
121 Find whether it is possible to make array elements same using one external number	870
Source	872
122 Generic Implementation of QuickSort Algorithm in C	873
Source	876
123 Given a number, find the next smallest palindrome	877
Source	892
124 Given a sorted array and a number x, find the pair in array whose sum is closest to x	893
Source	900
125 Given a sorted dictionary of an alien language, find order of characters	901
Source	907
126 Gnome Sort	908
Source	914
127 Heap Sort for decreasing order using min heap	915
Source	920

128 HeapSort	921
Source	932
129 Hoare's vs Lomuto partition scheme in QuickSort	933
Source	945
130 How to efficiently sort a big list dates in 20's	946
Source	949
131 How to make Mergesort to perform O(n) comparisons in best case?	950
Source	952
132 How to sort a big array with many repetitions?	953
Source	960
133 How to sort an array of dates in C/C++?	961
Source	963
134 In-Place Algorithm	964
Source	971
135 Insertion Sort	972
Source	981
136 Insertion Sort by Swapping Elements	982
Source	991
137 Insertion Sort for Doubly Linked List	992
Source	996
138 Insertion sort to sort even and odd positioned elements in different orders	997
Source	1001
139 Insertion sort using C++ STL	1002
Source	1003
140 Intersection of two Sorted Linked Lists	1004
Source	1011
141 Iterative HeapSort	1012
Source	1015
142 Iterative Merge Sort	1016
Source	1031
143 Iterative Quick Sort	1032
Source	1047
144 Java 8 Arrays parallelSort() method with Examples	1048
Source	1052

145 Java Program for Binary Insertion Sort	1053
Source1053
146 Java Program for Bubble Sort	1055
Source1055
147 Java Program for Cycle Sort	1057
Source1058
148 Java Program for Heap Sort	1061
Source1061
149 Java Program for Iterative Merge Sort	1064
Source1064
150 Java Program for QuickSort	1067
Source1068
151 Java Program for Recursive Bubble Sort	1070
Source1070
152 Java Program for Recursive Insertion Sort	1072
Source1072
153 Java Program for Stooge Sort	1074
Source1075
154 Job Selection Problem – Loss Minimization Strategy Set 2	1076
Source1079
155 Job Sequencing Problem – Loss Minimization	1080
Source1081
156 K-th smallest element after removing some integers from natural numbers	1082
Source1084
157 Know Your Sorting Algorithm Set 1 (Sorting Weapons used by Programming Languages)	1085
Source1087
158 Know Your Sorting Algorithm Set 2 (Introsort- C++’s Sorting Weapon)	1088
Source1093
159 Least frequent element in an array	1094
Source1103
160 Lexicographical concatenation of all substrings of a string	1104
Source1105
161 Lexicographically smallest string obtained after concatenating array	1106

Source1110
162 Longest Common Prefix using Sorting	1111
Source1114
163 Longest Consecutive Subsequence	1115
Source1119
164 Loop Invariant Condition with Examples of Sorting Algorithms	1120
Source1122
165 Lower bound for comparison based sorting algorithms	1123
Source1124
166 Making elements of two arrays same with minimum increment/decrement	1125
Source1129
167 Maximise the number of toys that can be purchased with amount K	1130
Source1135
168 Maximize elements using another array	1136
Source1139
169 Maximize sum of consecutive differences in a circular array	1140
Source1144
170 Maximize the profit by selling at-most M products	1145
Source1150
171 Maximize the sum of arr[i]*i	1151
Source1155
172 Maximizing Unique Pairs from two arrays	1156
Source1159
173 Maximum area rectangle by picking four sides from array	1160
Source1166
174 Maximum array from two given arrays keeping order same	1167
Source1169
175 Maximum difference between frequency of two elements such that element having greater frequency is also greater	1170
Source1173
176 Maximum difference between groups of size two	1174
Source1176
177 Maximum in an array that can make another array sorted	1177
Source1179

178 Maximum number of partitions that can be sorted individually to make sorted	1180
Source1184
179 Maximum possible difference of two subsets of an array	1185
Source1197
180 Maximum product of a triplet (subsequnece of size 3) in array	1198
Source1208
181 Maximum product of subsequence of size k	1209
Source1217
182 Maximum sum of absolute difference of an array	1218
Source1225
183 Maximum sum of pairwise product in an array with negative allowed	1226
Source1232
184 Maximum triplet sum in array	1233
Source1246
185 Median after K additional integers	1247
Source1251
186 Median and Mode using Counting Sort	1252
Source1253
187 Merge 3 Sorted Arrays	1254
Source1263
188 Merge Sort	1264
Source1277
189 Merge Sort for Doubly Linked List	1278
Source1286
190 Merge Sort for Linked Lists	1287
Source1294
191 Merge Sort for Linked Lists in JavaScript	1295
Source1299
192 Merge Sort using Multi-threading	1300
Source1304
193 Merge Sort with O(1) extra space merge and O(n lg n) time	1305
Source1309
194 Merge k sorted arrays Set 1	1310
Source1314

195 Merge two sorted arrays	1315
Source1322
196 Merge two sorted arrays with O(1) extra space	1323
Source1328
197 Merging and Sorting Two Unsorted Stacks	1329
Source1335
198 Merging two unsorted arrays in sorted order	1336
Source1346
199 Minimize the sum of product of two arrays with permutations allowed	1347
Source1352
200 Minimum De-arrangements present in array of AP (Arithmetic Progression)	1353
Source1358
201 Minimum cost to sort a matrix of numbers from 0 to $n^2 - 1$	1359
Source1366
202 Minimum difference between groups of size two	1367
Source1369
203 Minimum difference between max and min of all K-size subsets	1370
Source1376
204 Minimum number of distinct elements after removing m items	1377
Source1380
205 Minimum number of elements to add to make median equals x	1381
Source1389
206 Minimum number of subsets with distinct elements	1390
Source1396
207 Minimum number of subtract operation to make an array decreasing	1397
Source1403
208 Minimum number of swaps required to sort an array	1404
Source1410
209 Minimum partitions of maximum size 2 and sum limited by given value	1411
Source1415
210 Minimum product of k integers in an array of positive Integers	1416
Source1418
211 Minimum product pair an array of positive Integers	1419
Source1425

212 Minimum product subset of an array	1426
Source1435
213 Minimum sum of absolute difference of pairs of two arrays	1436
Source1440
214 Minimum sum of two numbers formed from digits of an array	1441
Source1446
215 Minimum swap required to convert binary tree to binary search tree	1447
Source1448
216 Minimum swaps required to Sort Binary array	1449
Source1453
217 Minimum swaps so that binary search can be applied	1454
Source1458
218 Minimum swaps to make two arrays identical	1459
Source1462
219 Minimum swaps to reach permuted array with at most 2 positions left swaps allowed	1463
Source1466
220 Most frequent element in an array	1467
Source1476
221 Most frequent word in an array of strings	1477
Source1480
222 Multiset Equivalence Problem	1481
Source1483
223 No of pairs ($a[j] \geq a[i]$) with k numbers in range ($a[i], a[j]$) that are divisible by x	1484
Source1486
224 Noble integers in an array (count of greater elements is equal to value)	1487
Source1495
225 Number of elements greater than K in the range L to R using Fenwick Tree (Offline queries)	1496
Source1501
226 Number of paths from source to destination in a directed acyclic graph	1502
Source1505
227 Number of sextuplets (or six values) that satisfy an equation	1506
Source1508

228 Number of swaps to sort when only adjacent swapping allowed	1509
Source1516
229 Number of visible boxes after putting one inside another	1517
Source1521
230 Odd-Even Sort / Brick Sort	1522
Source1528
231 PHP Sort array of strings in natural and standard orders	1529
Source1530
232 Pair formation such that maximum pair sum is minimized	1531
Source1533
233 Pairs such that one is a power multiple of other	1534
Source1540
234 Pairs with Difference less than K	1541
Source1549
235 Pancake sorting	1550
Source1558
236 Pandigital Product	1559
Source1563
237 Permute two arrays such that sum of every pair is greater or equal to K	1564
Source1569
238 Pigeonhole Sort	1570
Source1576
239 Position of an element after stable sort	1577
Source1582
240 Possible to form a triangle from array values	1583
Source1589
241 Print All Distinct Elements of a given integer array	1590
Source1598
242 Print Binary Tree levels in sorted order	1599
Source1602
243 Print Binary Tree levels in sorted order Set 2 (Using set)	1603
Source1605
244 Print all the pairs that contains the positive and negative values of an element	1606
Source1610

245 Print all triplets with given sum	1611
Source1622
246 Print array of strings in sorted order without copying one string into another	1623
Source1625
247 Print n smallest elements from given array in their original order	1626
Source1628
248 Print number in ascending order which contains 1, 2 and 3 in their digits.	1629
Source1633
249 Print sorted distinct elements of array	1634
Source1636
250 Print triplets with sum less than or equal to k	1637
Source1642
251 Printing frequency of each character just after its consecutive occurrence	1643
Source1646
252 Program for sorting variables of any data type	1647
Source1650
253 Program to check if an array is sorted or not (Iterative and Recursive)	1651
Source1652
254 Program to print an array in Pendulum Arrangement	1657
Source1664
255 Program to sort string in descending order	1665
Source1668
256 Python Code for time Complexity plot of Heap Sort	1669
Source1671
257 Python Program for Binary Insertion Sort	1672
Source1672
258 Python Program for Bubble Sort	1674
Source1674
259 Python Program for Cycle Sort	1676
Source1677
260 Python Program for Iterative Merge Sort	1679
Source1679
261 Python Program for QuickSort	1681

Source1682
262 Python Program for Recursive Insertion Sort	1684
Source1684
263 Python Program for Stooge Sort	1686
Source1686
264 Python list sort()	1688
Source1689
265 Python Sort Tuples in Increasing Order by any key	1690
Source1691
266 Python Sort a List according to the Length of the Elements	1692
Source1693
267 Python Sort a list according to the second element in sublist	1694
Source1696
268 Python Sort an array according to absolute difference	1697
Source1700
269 Python Sort words of sentence in ascending order	1701
Source1702
270 Quick Sort vs Merge Sort	1703
Source1707
271 QuickSort	1708
Source1722
272 QuickSort Tail Call Optimization (Reducing worst case space to Log n)	1723
Source1725
273 QuickSort on Doubly Linked List	1726
Source1733
274 QuickSort on Singly Linked List	1734
Source1738
275 QuickSort using Random Pivoting	1739
Source1744
276 Quickselect Algorithm	1745
Source1747
277 Radix Sort	1748
Source1760
278 Rank of all elements in an array	1761

Source1769
279 Rearrange an array in maximum minimum form Set 1	1770
Source1776
280 Rearrange an array in maximum minimum form Set 2 (O(1) extra space)	777
Source1787
281 Rearrange an array in order – smallest, largest, 2nd smallest, 2nd largest, ..	1788
Source1794
282 Rearrange an array to minimize sum of product of consecutive pair elements	1795
Source1797
283 Rearrange array such that even positioned are greater than odd	1798
Source1803
284 Rearrange positive and negative numbers using inbuilt sort function	1804
Source1806
285 Rearrange positive and negative numbers with constant extra space	1807
Source1820
286 Recursive Bubble Sort	1821
Source1826
287 Recursive Insertion Sort	1827
Source1833
288 Recursive Selection Sort	1834
Source1838
289 Recursive selection sort for singly linked list Swapping node links	1839
Source1843
290 Replacing an element makes array elements consecutive	1844
Source1850
291 Ropes left after every removal of smallest	1851
Source1857
292 Row wise sorting in 2D array	1858
Source1861
293 Segregate 0s and 1s in an array	1862
Source1877
294 Segregate even and odd numbers Set 3	1878
Source1882

295 Selection Sort	1883
Source1889
296 Serial Sort v/s Parallel Sort in Java	1890
Source1893
297 ShellSort	1894
Source1906
298 Shortest Un-ordered Subarray	1907
Source1913
299 Sleep Sort – The King of Laziness / Sorting while Sleeping	1914
Source1919
300 Smallest Difference Triplet from Three arrays	1920
Source1930
301 Smallest Difference pair of values between two unsorted Arrays	1931
Source1939
302 Smallest element in an array that is repeated exactly ‘k’ times.	1940
Source1953
303 Smallest greater elements in whole array	1954
Source1960
304 Smallest subset with sum greater than all other elements	1961
Source1965
305 Sort 1 to N by swapping adjacent elements	1966
Source1973
306 Sort 3 Integers without using if condition or using only max() function	1974
Source1978
307 Sort 3 numbers	1979
Source1984
308 Sort a Matrix in all way increasing order	1985
Source1987
309 Sort a Rotated Sorted Array	1988
Source1991
310 Sort a binary array using one traversal	1992
Source1996
311 Sort a linked list of 0s, 1s and 2s	1997
Source2004

312 Sort a linked list of 0s, 1s and 2s by changing links	2005
Source	2008
313 Sort a linked list that is sorted alternating ascending and descending orders?	2009
Source	2018
314 Sort a nearly sorted (or K sorted) array	2019
Source	2024
315 Sort a nearly sorted array using STL	2025
Source	2027
316 Sort a stack using a temporary stack	2028
Source	2034
317 Sort all even numbers in ascending order and then sort all odd numbers in descending order	2035
Source	2044
318 Sort an almost sorted array where only two elements are swapped	2045
Source	2048
319 Sort an array according to absolute difference with a given value “using constant extra space”	2049
Source	2051
320 Sort an array according to absolute difference with given value	2052
Source	2054
321 Sort an array according to count of set bits	2055
Source	2064
322 Sort an array according to the order defined by another array	2065
Source	2076
323 Sort an array after applying the given equation	2077
Source	2081
324 Sort an array containing two types of elements	2082
Source	2086
325 Sort an array in wave form	2087
Source	2094
326 Sort an array of 0s, 1s and 2s	2095
Source	2102
327 Sort an array of 0s, 1s and 2s (Simple Counting)	2103
Source	2109

328 Sort an array of large numbers	2110
Source	2112
329 Sort an array of strings according to string lengths	2113
Source	2115
330 Sort an array using socket programming in C	2116
Source	2120
331 Sort an array when two halves are sorted	2121
Source	2128
332 Sort an array which contain 1 to n values	2129
Source	2133
333 Sort an array with swapping only with a special element is allowed	2134
Source	2137
334 Sort array after converting elements to their squares	2138
Source	2146
335 Sort elements by frequency Set 1	2147
Source	2151
336 Sort elements by frequency Set 2	2152
Source	2155
337 Sort elements by frequency Set 4 (Efficient approach using hash)	2156
Source	2158
338 Sort elements on the basis of number of factors	2159
Source	2164
339 Sort even-placed elements in increasing and odd-placed in decreasing order	2165
Source	2168
340 Sort first half in ascending and second half in descending order 1	2169
Source	2175
341 Sort first half in ascending and second half in descending order Set 2	2176
Source	2179
342 Sort first k values in ascending order and remaining n-k values in descending order	2180
Source	2186
343 Sort n numbers in range from 0 to $n^2 - 1$ in linear time	2187
Source	2194
344 Sort numbers stored on different machines	2195

Source2199
345 Sort on the basis of number of factors using STL	2200
Source2202
346 Sort string of characters	2203
Source2206
347 Sort string of characters using Stack	2207
Source2209
348 Sort the bitonic doubly linked list	2210
Source2214
349 Sort the given matrix	2215
Source2221
350 Sort the given string using character search	2222
Source2226
351 Sort the linked list in the order of elements appearing in the array	2227
Source2230
352 Sort the matrix row-wise and column-wise	2231
Source2241
353 Sort the words in lexicographical order in Python	2242
Source2243
354 Sorting 2D Vector in C++ Set 2 (In descending order by row and column)	2244
Source2247
355 Sorting 2D Vector in C++ Set 3 (By number of columns)	2248
Source2252
356 Sorting Array Elements By Frequency Set 3 (Using STL)	2253
Source2255
357 Sorting Big Integers	2256
Source2258
358 Sorting Strings using Bubble Sort	2259
Source2260
359 Sorting Terminology	2261
Source2261
360 Sorting Vector of Pairs in C++ Set 1 (Sort by first and second)	2262
Source2266

361 Sorting Vector of Pairs in C++ Set 2 (Sort in descending order by first and second)	2267
Source2270
362 Sorting a Queue without extra space	2271
Source2278
363 Sorting all array elements except one	2279
Source2283
364 Sorting an array according to another array using pair in STL	2284
Source2286
365 Sorting array except elements in a subarray	2287
Source2292
366 Sorting array of strings (or words) using Trie	2293
Source2295
367 Sorting array of strings (or words) using Trie Set-2 (Handling Duplicate)	2296
Source2302
368 Sorting array using Stacks	2303
Source2308
369 Sorting in Java	2309
Source2312
370 Sorting of Vector of Tuple in C++ (Ascending Order)	2313
Source2317
371 Sorting rows of matrix in ascending order followed by columns in descending order	2318
Source2324
372 Sorting using trivial hash function	2325
Source2329
373 Sorting without comparison of elements	2330
Source2331
374 Sorting Natural Language Programming	2332
Source2336
375 Stability in sorting algorithms	2337
Source2340
376 Stable Selection Sort	2341
Source2345
377 Stable sort for descending order	2346

Source	2352
378 Stooge Sort	2353
Source	2359
379 Strand Sort	2360
Source	2364
380 Structure Sorting (By Multiple Rules) in C++	2365
Source	2368
381 Substring Sort	2369
Source	2373
382 Sum of Areas of Rectangles possible for an array	2374
Source	2385
383 Sum of Manhattan distances between all pairs of points	2386
Source	2395
384 Sum of all elements between k1'th and k2'th smallest elements	2396
Source	2400
385 Sum of minimum absolute difference of each array element	2401
Source	2407
386 Tag Sort (To get both sorted and original)	2408
Source	2412
387 TimSort	2413
Source	2416
388 Time Complexities of all Sorting Algorithms	2417
Source	2418
389 Time complexity of insertion sort when there are O(n) inversions?	2419
Source	2420
390 Tree Sort	2421
Source	2425
391 Triplets in array with absolute difference less than k	2426
Source	2432
392 Union and Intersection of two Linked Lists	2433
Source	2445
393 Union and Intersection of two linked lists Set-2 (Using Merge Sort)	2446
Source	2452
394 Union and Intersection of two linked lists Set-3 (Hashing)	2453

Source2457
395 Water drop problem	2458
Source2460
396 Ways to sort list of dictionaries by values in Python – Using itemgetter	2461
Source2462
397 When does the worst case of Quicksort occur?	2463
Source2463
398 Where is Heap Sort used practically?	2464
Source2464
399 Which sorting algorithm makes minimum number of memory writes?	2465
Source2465
400 Why Quick Sort preferred for Arrays and Merge Sort for Linked Lists	2466
Source2467
401 Why is it faster to process sorted array than an unsorted array ?	2468
Source2470
402 Why quicksort is better than mergesort ?	2471
Source2472
403 k largest(or smallest) elements in an array added Min Heap method	2473
Source2476
404 k smallest elements in same order using O(1) extra space	2477
Source2484
405 sort() in Python	2485
Source2488
406 stable_sort() in C++ STL	2489
Source2492
407 std::sort() in C++ STL	2493
Source2495

Contents

Chapter 1

3-Way QuickSort (Dutch National Flag)

3-Way QuickSort (Dutch National Flag) - GeeksforGeeks

In [simple QuickSort](#) algorithm, we select an element as pivot, partition the array around pivot and recur for subarrays on left and right of pivot.

Consider an array which has many redundant elements. For example, {1, 4, 2, 4, 2, 4, 1, 2, 4, 1, 2, 2, 2, 4, 1, 4, 4, 4}. If 4 is picked as pivot in Simple QuickSort, we fix only one 4 and recursively process remaining occurrences.

The idea of 3 way QuickSort is to process all occurrences of pivot and is based on [Dutch National Flag algorithm](#).

In 3 Way QuickSort, an array arr[1..r] is divided in 3 parts:

- a) arr[1..i] elements less than pivot.
- b) arr[i+1..j-1] elements equal to pivot.
- c) arr[j..r] elements greater than pivot.

Below is C++ implementation of above algorithm.

```
// C++ program for 3-way quick sort
#include <bits/stdc++.h>
using namespace std;

/* This function partitions a[] in three parts
   a) a[1..i] contains all elements smaller than pivot
   b) a[i+1..j-1] contains all occurrences of pivot
   c) a[j..r] contains all elements greater than pivot */
void partition(int a[], int l, int r, int &i, int &j)
{
    i = l-1, j = r;
```

```

int p = l-1, q = r;
int v = a[r];

while (true)
{
    // From left, find the first element greater than
    // or equal to v. This loop will definitely terminate
    // as v is last element
    while (a[++i] < v);

    // From right, find the first element smaller than or
    // equal to v
    while (v < a[--j])
        if (j == l)
            break;

    // If i and j cross, then we are done
    if (i >= j) break;

    // Swap, so that smaller goes on left greater goes on right
    swap(a[i], a[j]);

    // Move all same left occurrence of pivot to beginning of
    // array and keep count using p
    if (a[i] == v)
    {
        p++;
        swap(a[p], a[i]);
    }

    // Move all same right occurrence of pivot to end of array
    // and keep count using q
    if (a[j] == v)
    {
        q--;
        swap(a[j], a[q]);
    }
}

// Move pivot element to its correct index
swap(a[i], a[r]);

// Move all left same occurrences from beginning
// to adjacent to arr[i]
j = i-1;
for (int k = l; k < p; k++, j--)
    swap(a[k], a[j]);

```

```
// Move all right same occurrences from end
// to adjacent to arr[i]
i = i+1;
for (int k = r-1; k > q; k--, i++)
    swap(a[i], a[k]);
}

// 3-way partition based quick sort
void quicksort(int a[], int l, int r)
{
    if (r <= l) return;

    int i, j;

    // Note that i and j are passed as reference
    partition(a, l, r, i, j);

    // Recur
    quicksort(a, l, j);
    quicksort(a, i, r);
}

// A utility function to print an array
void printarr(int a[], int n)
{
    for (int i = 0; i < n; ++i)
        printf("%d ", a[i]);
    printf("\n");
}

// Driver program
int main()
{
    int a[] = {4, 9, 4, 4, 1, 9, 4, 4, 9, 4, 4, 1, 4};
    int size = sizeof(a) / sizeof(int);
    printarr(a, size);
    quicksort(a, 0, size - 1);
    printarr(a, size);
    return 0;
}
```

Output:

```
4 9 4 4 1 9 4 4 9 4 4 1 4
1 1 4 4 4 4 4 4 4 9 9 9
```

Thanks to [Utkarsh](#) for suggesting above implementation.

Another Implementation using Dutch National Flag Algorithm

```
// C++ program for 3-way quick sort
#include <bits/stdc++.h>
using namespace std;

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// A utility function to print an array
void printarr(int a[], int n)
{
    for (int i = 0; i < n; ++i)
        printf("%d ", a[i]);
    printf("\n");
}

/* This function partitions a[] in three parts
a) a[l..i] contains all elements smaller than pivot
b) a[i+1..j-1] contains all occurrences of pivot
c) a[j..r] contains all elements greater than pivot */

//It uses Dutch National Flag Algorithm
void partition(int a[], int low, int high, int &i, int &j)
{
    // To handle 2 elements
    if (high - low <= 1)
    {
        if (a[high] < a[low])
            swap(&a[high], &a[low]);
        i = low;
        j = high;
        return;
    }

    int mid = low;
    int pivot = a[high];
    while (mid <= high)
    {
        if (a[mid]<pivot)
            swap(&a[low++], &a[mid++]);
        else if (a[mid]==pivot)
            mid++;
        else if (a[mid]>pivot)
            swap(&a[mid], &a[high--]);
    }
}
```

```
//update i and j
i = low-1;
j = mid; //or high-1
}

// 3-way partition based quick sort
void quicksort(int a[], int low, int high)
{
    if (low>=high) //1 or 0 elements
        return;

    int i, j;

    // Note that i and j are passed as reference
    partition(a, low, high, i, j);

    // Recur two halves
    quicksort(a, low, i);
    quicksort(a, j, high);
}

// Driver program
int main()
{
    int a[] = {4, 9, 4, 4, 1, 9, 4, 4, 9, 4, 4, 1, 4};
    //int a[] = {4, 39, 54, 14, 31, 89, 44, 34, 59, 64, 64, 11, 41};
    //int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    //int a[] = {91, 82, 73, 64, 55, 46, 37, 28, 19, 10};
    //int a[] = {4, 9, 4, 4, 9, 1, 1, 1};
    int size = sizeof(a) / sizeof(int);

    printarr(a, size);
    quicksort(a, 0, size - 1);
    printarr(a, size);
    return 0;
}
```

Output:

```
4 9 4 4 1 9 4 4 9 4 4 1 4
1 1 4 4 4 4 4 4 4 9 9 9
```

Thanks [Aditya Goel](#) for this implementation.

Reference:

<http://algs4.cs.princeton.edu/lectures/23DemoPartitioning.pdf>
<http://www.sorting-algorithms.com/quick-sort-3-way>

Source

<https://www.geeksforgeeks.org/3-way-quicksort-dutch-national-flag/>

Chapter 2

3-way Merge Sort

3-way Merge Sort - GeeksforGeeks

Prerequisite – [Merge Sort](#)

Merge sort involves recursively splitting the array into 2 parts, sorting and finally merging them. A variant of merge sort is called 3-way merge sort where instead of splitting the array into 2 parts we split it into 3 parts.

Merge sort recursively breaks down the arrays to subarrays of size half. Similarly, 3-way Merge sort breaks down the arrays to subarrays of size one third.

Examples:

```
Input : 45, -2, -45, 78, 30, -42, 10, 19 , 73, 93
Output : -45 -42 -2 10 19 30 45 73 78 93
```

```
Input : 23, -19
Output : -19 23
```

```
// Java program to perform 3 way Merge Sort
public class MergeSort3Way
{
    // Function for 3-way merge sort process
    public static void mergeSort3Way(Integer[] gArray)
    {
        // if array of size is zero returns null
        if (gArray == null)
            return;

        // creating duplicate of given array
        Integer[] fArray = new Integer[gArray.length];
```

```

// copying elements of given array into
// duplicate array
for (int i = 0; i < fArray.length; i++)
    fArray[i] = gArray[i];

// sort function
mergeSort3WayRec(fArray, 0, gArray.length, gArray);

// copy back elements of duplicate array
// to given array
for (int i = 0; i < fArray.length; i++)
    gArray[i] = fArray[i];
}

/* Performing the merge sort algorithm on the
given array of values in the range of indices
[low, high). low is minimum index, high is
maximum index (exclusive) */
public static void mergeSort3WayRec(Integer[] gArray,
                                    int low, int high, Integer[] destArray)
{
    // If array size is 1 then do nothing
    if (high - low < 2)
        return;

    // Splitting array into 3 parts
    int mid1 = low + ((high - low) / 3);
    int mid2 = low + 2 * ((high - low) / 3) + 1;

    // Sorting 3 arrays recursively
    mergeSort3WayRec(destArray, low, mid1, gArray);
    mergeSort3WayRec(destArray, mid1, mid2, gArray);
    mergeSort3WayRec(destArray, mid2, high, gArray);

    // Merging the sorted arrays
    merge(destArray, low, mid1, mid2, high, gArray);
}

/* Merge the sorted ranges [low, mid1),
   [mid1, mid2) and [mid2, high) mid1 is first midpoint
   index in overall range to merge mid2 is second
   midpoint index in overall range to merge*/
public static void merge(Integer[] gArray, int low,
                        int mid1, int mid2, int high,
                        Integer[] destArray)
{
    int i = low, j = mid1, k = mid2, l = low;

```

```

// choose smaller of the smallest in the three ranges
while ((i < mid1) && (j < mid2) && (k < high))
{
    if (gArray[i].compareTo(gArray[j]) < 0)
    {
        if (gArray[i].compareTo(gArray[k]) < 0)
            destArray[l++] = gArray[i++];

        else
            destArray[l++] = gArray[k++];
    }
    else
    {
        if (gArray[j].compareTo(gArray[k]) < 0)
            destArray[l++] = gArray[j++];
        else
            destArray[l++] = gArray[k++];
    }
}

// case where first and second ranges have
// remaining values
while ((i < mid1) && (j < mid2))
{
    if (gArray[i].compareTo(gArray[j]) < 0)
        destArray[l++] = gArray[i++];
    else
        destArray[l++] = gArray[j++];
}

// case where second and third ranges have
// remaining values
while ((j < mid2) && (k < high))
{
    if (gArray[j].compareTo(gArray[k]) < 0)
        destArray[l++] = gArray[j++];
    else
        destArray[l++] = gArray[k++];
}

// case where first and third ranges have
// remaining values
while ((i < mid1) && (k < high))
{
    if (gArray[i].compareTo(gArray[k]) < 0)
        destArray[l++] = gArray[i++];
}

```

```

        else
            destArray[l++] = gArray[k++];
    }

    // copy remaining values from the first range
    while (i < mid1)
        destArray[l++] = gArray[i++];

    // copy remaining values from the second range
    while (j < mid2)
        destArray[l++] = gArray[j++];

    // copy remaining values from the third range
    while (k < high)
        destArray[l++] = gArray[k++];
}

// Driver function
public static void main(String args[])
{
    // test case of values
    Integer[] data = new Integer[] {45, -2, -45, 78,
                                    30, -42, 10, 19, 73, 93};
    mergeSort3Way(data);

    System.out.println("After 3 way merge sort: ");
    for (int i = 0; i < data.length; i++)
        System.out.print(data[i] + " ");
}
}
    
```

Output:

```

After 3 way merge sort:
-45 -42 -2 10 19 30 45 73 78 93
    
```

Here, we first copy the contents of data array to another array called fArray. Then, sort the array by finding midpoints that divide the array into 3 parts and called sort function on each array respectively. The base case of recursion is when size of array is 1 and it returns from the function. Then merging of arrays starts and finally the sorted array will be in fArray which is copied back to gArray.

Time Complexity: In case of 2-way Merge sort we get the equation: $T(n) = 2T(n/2) + O(n)$

Similarly, in case of 3-way Merge sort we get the equation: $T(n) = 3T(n/3) + O(n)$

By solving it using [Master method](#), we get its complexity as $O(n \log_3 n)$. Although time complexity looks less compared to [2 way merge sort](#), the time taken actually may become

higher because number of comparisons in merge function go higher. Please refer [Why is Binary Search preferred over Ternary Search?](#) for details.

Similar article :

[3 way Quick Sort](#)

Source

<https://www.geeksforgeeks.org/3-way-merge-sort/>

Chapter 3

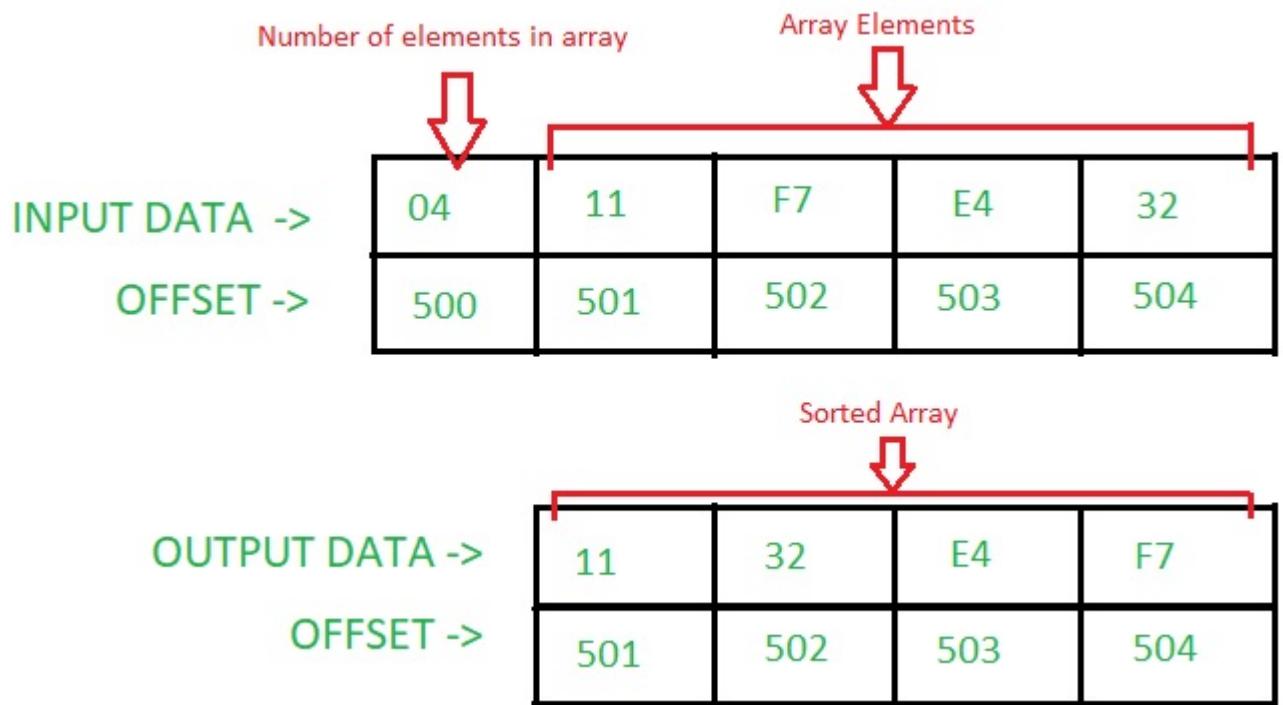
8086 program for selection sort

8086 program for selection sort - GeeksforGeeks

Problem – Write an assembly language program in 8086 microprocessor to sort a given array of n numbers using Selection Sort.

Assumptions – The number of elements in the array is stored at offset 500. The array starts from offset 501.

Example –



Algorithm –

1. We first find the smallest number in the array.
2. Swap the smallest number from the first element of the array.
3. Keep repeating the process till all elements are traversed.

Program –

Offset	Mnemonics	Comment
400	MOV DI, 501	DI < - 501
403	MOV SI, 500	SI < - 500
406	MOV CL, [SI]	CL < - [SI]
408	XOR CH, CH	CH < - CH ^ (XOR) CH
40A	INC SI	SI < - SI+0001
40B	DEC CX	CX < - CX-0001
40C	MOV BX, SI	BX < - SI
40E	MOV AH, CL	AH < - CL
410	INC AH	AH < - AH+01
412	MOV AL, [SI]	AL < - [SI]
414	INC SI	SI < - SI+0001
415	DEC AH	AH < - AH-01
417	CMP AL, [SI]	AL-[SI]
419	JC 41F	If Carry Flag = 1, goto offset 41F
41B	MOV AL, [SI]	AL < - [SI]
41D	MOV BX, SI	BX < - SI
41F	INC SI	SI < - SI+0001
420	DEC AH	AH < - AH-01
422	JNZ 417	If Zero Flag = 0, goto offset 417
424	MOV DL, [BX]	DL < - [BX]
426	XCHG DL, [DI]	DL < - > [DI]
428	XCHG DL, [BX]	DL < - > [BX]
42A	INC DI	DI < - DI+0001
42B	MOV SI, DI	SI < - DI
42D	LOOP 40C	CX < - CX-0001; If Zero Flag = 0, goto offset 40C.
42F	HLT	End of program.

Explanation – Registers AH, AL, BX, CX, DL, SI, DI are used for general purpose:

AL - Stored the smallest number
 AH - Stores the counter for the inner loop
 BX - Stores the offset of the smallest
 number of each iteration of the outer loop
 CX - Stores the counter for the outer loop
 DL - Helps in swapping the elements

SI - Pointer
DI - Pointer

1. **MOV SI, 500:** stores 0500 in SI.
2. **MOV CL, [SI]:** stores the content at offset SI in CL.
3. **XOR CH, CH:** stores the result of logical operation XOR b/w CH and CH in CH.
4. **INC SI:** increase the value of SI by 1.
5. **DEC CX:** decrease the value of CX by 1.
6. **MOV AH, CL:** stores the contents of CL in AH.
7. **CMP AL, [SI]:** compares the content of AL with content at offset SI. If $AL < [SI]$ – Sets Carry Flag(i.e. Carry Flag = 1).
8. **JC 41F:** jumps to offset 041F, if carry flag is set(1).
9. **JNZ 417:** jumps to offset 0417, if zero flag is reset(0).
10. **XCHG DL, [BX]:** swaps the content of DL with content at offset BX.
11. **LOOP 40C:** decrease the value of CX by 1 and check whether Zero Flag is set(1) or not. If Zero Flag is reset(0), then it jumps to offset 040C.
12. **HLT:** terminates the program.

Source

<https://www.geeksforgeeks.org/8086-program-selection-sort/>

Chapter 4

A Pancake Sorting Problem

A Pancake Sorting Problem - GeeksforGeeks

We have discussed [Pancake Sorting](#) in the previous post. Following is a problem based on Pancake Sorting.

Given an unsorted array, sort the given array. You are allowed to do only following operation on array.

```
flip(arr, i): Reverse array from 0 to i
```

Imagine a hypothetical machine where `flip(i)` always takes $O(1)$ time. Write an efficient program for sorting a given array in $O(n \log n)$ time on the given machine. If we apply the same algorithm here, the time taken will be $O(n^2)$ because the algorithm calls `findMax()` in a loop and `findMax()` takes $O(n)$ time even on this hypothetical machine.

We can use insertion sort that uses binary search. The idea is to run a loop from second element to last element (from $i = 1$ to $n-1$), and one by one insert $arr[i]$ in $arr[0..i-1]$ (like [standard insertion sort algorithm](#)). When we insert an element $arr[i]$, we can use binary search to find position of $arr[i]$ in $O(\log i)$ time. Once we have the position, we can use some flip operations to put $arr[i]$ at its new place. Following are abstract steps.

```
// Standard Insertion Sort Loop that starts from second element
for (i=1; i < n; i++) ----> O(n)
{
    int key = arr[i];

    // Find index of ceiling of arr[i] in arr[0..i-1] using binary search
    j = celiSearch(arr, key, 0, i-1); ----> O(log n) (See this)

    // Apply some flip operations to put arr[i] at correct place
}
```

Since flip operation takes $O(1)$ on given hypothetical machine, total running time of above algorithm is $O(n\log n)$. Thanks to [Kumar](#) for suggesting above problem and algorithm.

Let us see how does the above algorithm work. [ceilSearch\(\)](#) actually returns the index of the smallest element which is greater than $\text{arr}[i]$ in $\text{arr}[0..i-1]$. If there is no such element, it returns -1. Let the returned value be j . If j is -1, then we don't need to do anything as $\text{arr}[i]$ is already the greatest element among $\text{arr}[0..i]$. Otherwise we need to put $\text{arr}[i]$ just before $\text{arr}[j]$.

So how to apply flip operations to put $\text{arr}[i]$ just before $\text{arr}[j]$ using values of i and j . Let us take an example to understand this. Let i be 6 and current array be $\{12, 15, 18, 30, 35, 40, \mathbf{20}, 6, 90, 80\}$. To put 20 at its new place, the array should be changed to $\{12, 15, 18, \mathbf{20}, 30, 35, 40, 6, 90, 80\}$. We apply following steps to put 20 at its new place.

- 1) Find j using [ceilSearch](#) (In the above example j is 3).
- 2) $\text{flip}(\text{arr}, j-1)$ (array becomes $\{18, 15, 12, 30, 35, 40, \mathbf{20}, 6, 90, 80\}$)
- 3) $\text{flip}(\text{arr}, i-1);$ (array becomes $\{40, 35, 30, 12, 15, 18, \mathbf{20}, 6, 90, 80\}$)
- 4) $\text{flip}(\text{arr}, i);$ (array becomes $\{\mathbf{20}, 18, 15, 12, 30, 35, 40, 6, 90, 80\}$)
- 5) $\text{flip}(\text{arr}, j);$ (array becomes $\{12, 15, 18, \mathbf{20}, 30, 35, 40, 6, 90, 80\}$)

Following is C implementation of the above algorithm.

C

```
#include <stdlib.h>
#include <stdio.h>

/* A Binary Search based function to get index of ceiling of x in
   arr[low..high] */
int ceilSearch(int arr[], int low, int high, int x)
{
    int mid;

    /* If x is smaller than or equal to the first element,
       then return the first element */
    if(x <= arr[low])
        return low;

    /* If x is greater than the last element, then return -1 */
    if(x > arr[high])
        return -1;

    /* get the index of middle element of arr[low..high]*/
    mid = (low + high)/2; /* low + (high - low)/2 */

    /* If x is same as middle element, then return mid */
    if(arr[mid] == x)
        return mid;

    /* If x is greater than arr[mid], then either arr[mid + 1]
       is ceiling of x, or ceiling lies in arr[mid+1...high] */

```

```

if(arr[mid] < x)
{
    if(mid + 1 <= high && x <= arr[mid+1])
        return mid + 1;
    else
        return ceilSearch(arr, mid+1, high, x);
}

/* If x is smaller than arr[mid], then either arr[mid]
   is ceiling of x or ceiling lies in arr[mid-1...high] */
if (mid - 1 >= low && x > arr[mid-1])
    return mid;
else
    return ceilSearch(arr, low, mid - 1, x);
}

/* Reverses arr[0..i] */
void flip(int arr[], int i)
{
    int temp, start = 0;
    while (start < i)
    {
        temp = arr[start];
        arr[start] = arr[i];
        arr[i] = temp;
        start++;
        i--;
    }
}

/* Function to sort an array using insertion sort, binary search and flip */
void insertionSort(int arr[], int size)
{
    int i, j;

    // Start from the second element and one by one insert arr[i]
    // in already sorted arr[0..i-1]
    for(i = 1; i < size; i++)
    {
        // Find the smallest element in arr[0..i-1] which is also greater than
        // or equal to arr[i]
        int j = ceilSearch(arr, 0, i-1, arr[i]);

        // Check if there was no element greater than arr[i]
        if (j != -1)
        {
            // Put arr[i] before arr[j] using following four flip operations
            flip(arr, j-1);
        }
    }
}

```

```

        flip(arr, i-1);
        flip(arr, i);
        flip(arr, j);
    }
}

/* A utility function to print an array of size n */
void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        printf("%d ", arr[i]);
}

/* Driver program to test insertion sort */
int main()
{
    int arr[] = {18, 40, 35, 12, 30, 35, 20, 6, 90, 80};
    int n = sizeof(arr)/sizeof(arr[0]);
    insertionSort(arr, n);
    printArray(arr, n);
    return 0;
}

```

Python

```

#A Binary Search based function to get index of ceiling of x in arr[low..high]

def ceilSearch(arr,low,high,x):

    #If x is smaller than or equal to the first element,
    #then return the first element
    if x <= arr[low]:
        return low

    #If x is greater than the last element, then return -1
    if x > arr[high]:
        return -1

    #get the index of middle element of arr[low..high]
    mid = (low + high)/2 #low + (high-low)/2

    #If x is same as middle element, then return mid
    if(arr[mid] == x):
        return mid

    #If x is greater than arr[mid], then either arr[mid + 1]

```

```

#is ceiling of x, or ceiling lies in arr[mid+1...high]
if(arr[mid] < x):
    if(mid + 1 <= high and x <= arr[mid+1]):
        return mid + 1
    else:
        return ceilSearch(arr, mid+1, high, x)

#If x is smaller than arr[mid], then either arr[mid]
#is ceiling of x or ceiling lies in arr[mid-1...high]
if (mid - 1 >= low and x > arr[mid-1]):
    return mid
else:
    return ceilSearch(arr, low, mid - 1, x)

#Reverses arr[0..i] */
def flip(arr,i):

    start = 0;
    while (start < i):
        temp = arr[start]
        arr[start] = arr[i]
        arr[i] = temp
        start+=1
        i-=1

#Function to sort an array using insertion sort, binary search and flip
def insertionSort(arr):

    #Start from the second element and one by one insert arr[i]
    #in already sorted arr[0..i-1]
    for i in range(1,len(arr)):
        #Find the smallest element in arr[0..i-1] which is also greater than
        #or equal to arr[i]
        j = ceilSearch(arr, 0, i-1, arr[i])

        #Check if there was no element greater than arr[i]
        if (j != -1):
            #Put arr[i] before arr[j] using following four flip operations
            flip(arr, j-1)
            flip(arr, i-1)
            flip(arr, i)
            flip(arr, j)

    # A utility function to print an array of size n
def printArray(arr):
    for i in range(0,len(arr)):
        print arr[i],

```

```
#Driver function to test above function
arr=[18, 40, 35, 12, 30, 35, 20, 6, 90, 80]
insertionSort(arr)
printArray(arr)

#This code is contributed by Devesh Agrawal
```

Output:

```
6 12 18 20 30 35 35 40 80 90
```

Source

<https://www.geeksforgeeks.org/a-pancake-sorting-question/>

Chapter 5

A sorting algorithm that slightly improves on selection sort

A sorting algorithm that slightly improves on selection sort - GeeksforGeeks

As we know, [selection sort](#) algorithm takes the minimum on every pass on the array, and place it at its correct position.

The idea is to take also the maximum on every pass and place it at its correct position. So in every pass, we keep track of both maximum and minimum and array becomes sorted from both ends.

Examples:

```
First example: 7 8 5 4 9 2
Input :pass 1:|7 8 5 4 9 2|
          pass 2: 2|8 5 4 7|9
          pass 3: 2 4|5 7|8 9
Output :A sorted array: 2 4 5 7 8 9
```

```
second example: 23 78 45 8 32 56 1
Input :pass 1:|23 78 45 8 32 56 1|
          pass 2: 1|23 45 8 32 56 |78
          pass 3: 1 8|45 23 32|56 78
          pass 4: 1 8 23 |32|45 56 78
          in a case of odd elements, so one element
          left for sorting, so sorting stops and the
          array is sorted.
Output : A sorted array: 1 8 23 32 45 56 78
```

```
// C++ program to implement min max selection
// sort.
#include <iostream>
```

```
using namespace std;

void minMaxSelectionSort(int* arr, int n)
{
    for (int i = 0, j = n - 1; i < j; i++, j--) {
        int min = arr[i], max = arr[i];
        int min_i = i, max_i = i;
        for (int k = i; k <= j; k++) {
            if (arr[k] > max) {
                max = arr[k];
                max_i = k;
            } else if (arr[k] < min) {
                min = arr[k];
                min_i = k;
            }
        }
        // shifting the min.
        swap(arr[i], arr[min_i]);

        // Shifting the max. The equal condition
        // happens if we shifted the max to arr[min_i]
        // in the previous swap.
        if (arr[min_i] == max)
            swap(arr[j], arr[min_i]);
        else
            swap(arr[j], arr[max_i]);
    }
}

// Driver code
int main()
{
    int arr[] = { 23, 78, 45, 8, 32, 56, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    minMaxSelectionSort(arr, n);
    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
    return 0;
}
```

Output:

```
Sorted array: 1 8 23 32 45 56 78
```

Source

<https://www.geeksforgeeks.org/sorting-algorithm-slightly-improves-selection-sort/>

Chapter 6

Absolute distinct count in a sorted array

Absolute distinct count in a sorted array - GeeksforGeeks

Given a sorted array of integers, return the number of distinct absolute values among the elements of the array. The input can contain duplicates values.

```
Input: [-3, -2, 0, 3, 4, 5]
Output: 5
There are 5 distinct absolute values
among the elements of this array, i.e.
0, 2, 3, 4 and 5)
```

```
Input: [-1, -1, -1, -1, 0, 1, 1, 1, 1]
Output: 2
```

```
Input: [-1, -1, -1, -1, 0]
Output: 2
```

```
Input: [0, 0, 0]
Output: 1
```

The solution should do only one scan of the input array and should not use any extra space. i.e. expected time complexity is $O(n)$ and auxiliary space is $O(1)$.

One simple solution is to use set. For each element of the input array, we insert its absolute value in the set. As set doesn't support duplicate elements, the element's absolute value will be inserted only once. Therefore, the required count is size of the set.

Below is the implementation of the idea.

C++

```
// Program to find absolute distinct
// count of an array in O(n) time.
#include <bits/stdc++.h>
using namespace std;

// The function returns number of
// distinct absolute values among
// the elements of the array
int distinctCount(int arr[], int n)
{
    unordered_set<int> s;

    // Note that set keeps only one
    // copy even if we try to insert
    // multiple values
    for (int i = 0 ; i < n; i++)
        s.insert(abs(arr[i]));

    return s.size();
}

// Driver code
int main()
{
    int arr[] = {-2, -1, 0, 1, 1};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Count of absolute distinct values : "
         << distinctCount(arr, n);

    return 0;
}
```

Java

```
// java code to find absolute distinct
// count of an array in O(n) time.
import java.util.*;

class GFG
{
    // The function returns number of
    // distinct absolute values among
    // the elements of the array
    static int distinctCount(int arr[], int n)
    {
        Set<Integer> s = new HashSet<Integer> ();

```

```
// Note that set keeps only one
// copy even if we try to insert
// multiple values
for (int i = 0 ; i < n; i++)
    s.add(Math.abs(arr[i]));

return s.size();

}

// Driver code
public static void main(String[] args)
{
    int arr[] = {-2, -1, 0, 1, 1};
    int n = arr.length;

    System.out.println("Count of absolute distinct values : "
        + distinctCount(arr, n));

}
}

// This code is contributed by prerna saini
```

Output :

```
Count of absolute distinct values : 3
```

Time Complexity : O(n)

Auxiliary Space : O(n)

The above implementation takes O(n) extra space, how to do in O(1) extra space?

The idea is to take advantage of the fact that the array is already Sorted. We initialize the count of distinct elements to number of elements in the array. We start with two index variables from two corners of the array and check for pair in the input array with sum as 0. If pair with 0 sum is found or duplicates are encountered, we decrement the count of distinct elements.Finally we return the updated count.

Below is C++ implementation of the idea.

```
// Program to find absolute distinct
// count of an array using O(1) space.
#include <bits/stdc++.h>
using namespace std;
```

```
// The function returns return number
// of distinct absolute values
// among the elements of the array
int distinctCount(int arr[], int n)
{
    // initialize count as number of elements
    int count = n;
    int i = 0, j = n - 1, sum = 0;

    while (i < j)
    {
        // Remove duplicate elements from the
        // left of the current window (i, j)
        // and also decrease the count
        while (i != j && arr[i] == arr[i + 1])
            count--, i++;

        // Remove duplicate elements from the
        // right of the current window (i, j)
        // and also decrease the count
        while (i != j && arr[j] == arr[j - 1])
            count--, j--;

        // break if only one element is left
        if (i == j)
            break;

        // Now look for the zero sum pair
        // in current window (i, j)
        sum = arr[i] + arr[j];

        if (sum == 0)
        {
            // decrease the count if (positive,
            // negative) pair is encountered
            count--;
            i++, j--;
        }
        else if(sum < 0)
            i++;
        else
            j--;
    }

    return count;
}

// Driver code
```

```
int main()
{
    int arr[] = {-2, -1, 0, 1, 1};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Count of absolute distinct values : "
        << distinctCount(arr, n);

    return 0;
}
```

Output :

```
Count of absolute distinct values : 3
```

Time Complexity : $O(n)$
Auxiliary Space : $O(1)$

Source

<https://www.geeksforgeeks.org/absolute-distinct-count-array-sorted-absolute-values/>

Chapter 7

Alternate Lower Upper String Sort

Alternate Lower Upper String Sort - GeeksforGeeks

Given a string containing lowercase and uppercase letters.sort it in such a manner such that the uppercase and lowercase letter comes in an alternate manner but in sorted way.

Examples:

Input : bAwutndekWEdkd

Output :AbEdWddekntuw

Explanation:

Here we can see that letter 'A', 'E', 'W' are sorted as well as letters "b, d, d, d, e, k, k, n, t, u, w" are sorted but both appears alternately in the string as far as possible.

Input :abbfDDhGFBvdFDGBNDasZVDFjkb

Output :BaBaDbDbDbDdFhFjFkGsGvNVZ

- 1) Count lower case characters in an array lCount[]
- 2) Count upper case characters in another array uCount[]
- 3) Modify string using lCount[] and uCount[]

CPP

```
// C++ program for unusul string sorting
#include <bits/stdc++.h>
using namespace std;
#define MAX 26

// Function for alternate sorting of string
void alternateSort(string& s)
```

```
{
    int n = s.length();

    // Count occurrences of individual lower case and
    // upper case characters
    int lCount[MAX] = { 0 }, uCount[MAX] = { 0 };
    for (int i = 0; i < n; i++) {
        if (isupper(s[i]))
            uCount[s[i] - 'A']++;
        else
            lCount[s[i] - 'a']++;
    }

    // Traverse through count arrays and one by one
    // pick characters. Below loop takes O(n) time
    // considering the MAX is constant.
    int i = 0, j = 0, k = 0;
    while (k < n) {
        while (i < MAX && uCount[i] == 0)
            i++;
        if (i < MAX) {
            s[k++] = 'A' + i;
            uCount[i]--;
        }
    }

    while (j < MAX && lCount[j] == 0)
        j++;
    if (j < MAX) {
        s[k++] = 'a' + j;
        lCount[j]--;
    }
}

// Driver code
int main()
{
    string str = "bAwutndekWEdkd";
    alternateSort(str);
    cout << str << "\n";
}
```

Java

```
// Java program for
// unusual string sorting
import java.util.*;
import java.lang.*;
```

```
public class GfG{  
  
    private final static int MAX = 100;  
  
    // Function for alternate  
    // sorting of string  
    public static String alternateSort(String s1)  
    {  
        int n = s1.length();  
  
        char[] s = s1.toCharArray();  
  
        // Count occurrences of  
        // individual lower case and  
        // upper case characters  
        int[] lCount = new int[MAX];  
        int[] uCount = new int[MAX];  
  
        for (int i = 0; i < n; i++) {  
  
            if (Character.isUpperCase(s[i]))  
                uCount[s[i] - 'A']++;  
            else  
                lCount[s[i] - 'a']++;  
        }  
  
        // Traverse through count  
        // arrays and one by one  
        // pick characters.  
        // Below loop takes O(n) time  
        // considering the MAX is constant.  
        int i = 0, j = 0, k = 0;  
        while (k < n)  
        {  
  
            while (i < MAX && uCount[i] == 0)  
                i++;  
  
            if (i < MAX) {  
                s[k++] = (char)('A' + i);  
                uCount[i]--;  
            }  
  
            while (j < MAX && lCount[j] == 0)  
                j++;  
  
            if (j < MAX) {  
                s[k++] = (char)('a' + j);  
                lCount[j]--;  
            }  
        }  
    }  
}
```

```
        s[k++] = (char)('a' + j);
        lCount[j]--;
    }
}

return (new String(s));
}

// Driver function
public static void main(String argc[]){
    String str = "bAwutndekWEdkd";
    System.out.println(alternateSort(str));
}
}

// This code is contributed by Sagar Shukla
```

C#

```
// C# program for unusual string sorting
using System;

public class GFG {

    private static int MAX = 100;

    // Function for alternate
    // sorting of string
    static String alternateSort(String s1)
    {
        int n = s1.Length;
        int l = 0, j = 0, k = 0;

        char[] s = s1.ToCharArray();

        // Count occurrences of
        // individual lower case and
        // upper case characters
        int[] lCount = new int[MAX];
        int[] uCount = new int[MAX];

        for (int i = 0; i < n; i++) {

            if (char.IsUpper(s[i]))
                uCount[s[i] - 'A']++;
        }
```

```

        else
            lCount[s[i] - 'a']++;
    }

    // Traverse through count
    // arrays and one by one
    // pick characters.
    // Below loop takes O(n) time
    // considering the MAX is constant.

    while (k < n)
    {

        while (l < MAX && uCount[l] == 0)
            l++;

        if (l < MAX) {
            s[k++] = (char)('A' + l);
            uCount[l]--;
        }

        while (j < MAX && lCount[j] == 0)
            j++;

        if (j < MAX) {
            s[k++] = (char)('a' + j);
            lCount[j]--;
        }
    }

    return (new String(s));
}

// Driver function
public static void Main()
{
    String str = "bAwutndekWEdkd";

    Console.WriteLine(alternateSort(str));
}
}

// This code is contributed by parashar.

```

Output:

AbEdWddekntuw

Time Complexity : O(n)

Improved By : [parashar](#)

Source

<https://www.geeksforgeeks.org/alternate-lower-upper-string-sort/>

Chapter 8

Alternate sorting of Linked list

Alternate sorting of Linked list - GeeksforGeeks

Given a linked list containing **n** nodes. The problem is to rearrange the nodes of the list in such a way that the data in first node is first minimum, second node is first maximum, third node is second minimum, fourth node is second maximum and so on.

Examples:

Input : list: 4->1->3->5->2
Output : 1->5->2->4->3

Input : list: 10->9->8->7->6->5
Output : 5->10->6->9->7->8

Approach: Following are the steps:

1. Sort the linked list using [Merge Sort](#) technique.
2. Split the list into **front** and **back** lists. Refer **FrontBackProcedure** of [this](#) post.
3. Now, reverse the **back** list. Refer [this](#) post.
4. Finally, merge the nodes of **first** and **back** lists in alternate order.

```
// C++ implementation of alternative sorting
// of the Linked list
#include <bits/stdc++.h>

using namespace std;

// node of a linked list
struct Node {
```

```
int data;
struct Node* next;
};

// function to get a new node
Node* getNode(int data)
{
    // allocate space for node
    Node* newNode = (Node*)malloc(sizeof(Node));

    // put in the data
    newNode->data = data;
    newNode->next = NULL;

    return newNode;
}

// Split the nodes of the given list into front and
// back halves, and return the two lists using the
// reference parameters. If the length is odd, the
// extra node should go in the front list. Uses the
// fast/slow pointer strategy.
void FrontBackSplit(Node* source,
                     Node** frontRef, Node** backRef)
{
    Node* fast;
    Node* slow;
    if (source == NULL || source->next == NULL) {
        // length < 2 cases
        *frontRef = source;
        *backRef = NULL;
    }
    else {
        slow = source;
        fast = source->next;

        // Advance 'fast' two nodes, and
        // advance 'slow' one node
        while (fast != NULL) {
            fast = fast->next;
            if (fast != NULL) {
                slow = slow->next;
                fast = fast->next;
            }
        }
    }

    // 'slow' is before the midpoint in the list,
    // so split it in two at that point.
}
```

```

        *frontRef = source;
        *backRef = slow->next;
        slow->next = NULL;
    }
}

// function to merge two sorted lists in
// sorted order
Node* SortedMerge(Node* a, Node* b)
{
    Node* result = NULL;

    // Base cases
    if (a == NULL)
        return b;
    else if (b == NULL)
        return a;

    // Pick either a or b, and recur
    if (a->data <= b->data) {
        result = a;
        result->next = SortedMerge(a->next, b);
    }
    else {
        result = b;
        result->next = SortedMerge(a, b->next);
    }

    return result;
}

// sorts the linked list by changing
// next pointers (not data)
void MergeSort(Node** headRef)
{
    Node* head = *headRef;
    Node *a, *b;

    // Base case -- length 0 or 1
    if ((head == NULL) || (head->next == NULL))
        return;

    // Split head into 'a' and 'b' sublists
    FrontBackSplit(head, &a, &b);

    // Recursively sort the sublists
    MergeSort(&a);
    MergeSort(&b);
}

```

```

// merge the two sorted lists together
*headRef = SortedMerge(a, b);
}

// function to reverse the linked list
static void reverse(Node** head_ref)
{
    Node* prev = NULL;
    Node* current = *head_ref;
    Node* next;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }

    *head_ref = prev;
}

// function to alternately merge two lists
void alternateMerge(Node* head1, Node* head2)
{
    Node *p, *q;
    while (head1 != NULL && head2 != NULL) {

        // merging nodes alternately by
        // making the desired links
        p = head1->next;
        head1->next = head2;
        head1 = p;
        q = head2->next;
        head2->next = head1;
        head2 = q;
    }
}

// function for alternative sort of the
// linked list
Node* altSortForLinkedList(Node* head)
{
    // sort the linked list
    MergeSort(&head);

    Node *front, *back;

```

```
// Split head into 'front' and 'back' sublists
FrontBackSplit(head, &front, &back);

// reversing the 'back' list
reverse(&back);

// merging the nodes of 'front' and 'back'
// lists alternately
alternateMerge(front, back);

// required head of final list
return front;
}

// Function to print nodes in
// a given linked list
void printList(Node* head)
{
    while (head != NULL) {
        cout << head->data << " ";
        head = head->next;
    }
}

// Driver program to test above
int main()
{
    // linked list: 10->9->8->7->6->5
    Node* head = getNode(10);
    head->next = getNode(9);
    head->next->next = getNode(8);
    head->next->next->next = getNode(7);
    head->next->next->next->next = getNode(6);
    head->next->next->next->next->next = getNode(5);

    cout << "Original list: ";
    printList(head);

    head = altSortForLinkedList(head);

    cout << "\nModified list: ";
    printList(head);

    return 0;
}
```

Output:

Original list: 10 9 8 7 6 5
Modified list: 5 10 6 9 7 8

Time Complexity: $O(n * \log n)$.

Source

<https://www.geeksforgeeks.org/alternate-sorting-linked-list/>

Chapter 9

Alternative Sorting

Alternative Sorting - GeeksforGeeks

Given an array of integers, print the array in such a way that the first element is first maximum and second element is first minimum and so on.

Examples :

```
Input : arr[] = {7, 1, 2, 3, 4, 5, 6}
Output : 7 1 6 2 5 3 4
```

```
Input : arr[] = {1, 6, 9, 4, 3, 7, 8, 2}
Output : 9 1 8 2 7 3 6 4
```

A **simple solution** is to first print maximum element, then minimum, then second maximum, and so on. Time complexity of this approach is $O(n^2)$.

An **efficient solution** involves following steps.

- 1) Sort input array using a $O(n \log n)$ algorithm.
- 2) We maintain two pointers, one from beginning and one from end in sorted array. We alternatively print elements pointed by two pointers and move them toward each other.

C++

```
// C++ program to print an array in alternate
// sorted manner.
#include <bits/stdc++.h>
using namespace std;

// Function to print alternate sorted values
void alternateSort(int arr[], int n)
{
    // Sorting the array
```

```
sort(arr, arr+n);

// Printing the last element of array
// first and then first element and then
// second last element and then second
// element and so on.
int i = 0, j = n-1;
while (i < j) {
    cout << arr[j--] << " ";
    cout << arr[i++] << " ";
}

// If the total element in array is odd
// then print the last middle element.
if (n % 2 != 0)
    cout << arr[i];
}

// Driver code
int main()
{
    int arr[] = {1, 12, 4, 6, 7, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    alternateSort(arr, n);
    return 0;
}
```

Java

```
// Java program to print an array in alternate
// sorted manner
import java.io.*;
import java.util.Arrays;

class AlternativeString
{
    // Function to print alternate sorted values
    static void alternateSort(int arr[], int n)
    {
        Arrays.sort(arr);

        // Printing the last element of array
        // first and then first element and then
        // second last element and then second
        // element and so on.
        int i = 0, j = n-1;
        while (i < j) {
            System.out.print(arr[j--] + " ");
```

```
        System.out.print(arr[i++] + " ");
    }

    // If the total element in array is odd
    // then print the last middle element.
    if (n % 2 != 0)
        System.out.print(arr[i]);
}

/* Driver program to test above functions */
public static void main (String[] args)
{
    int arr[] = {1, 12, 4, 6, 7, 10};
    int n = arr.length;
    alternateSort(arr, n);
}
}
/*This code is contributed by Prakriti Gupta*/
```

Python3

```
# Python 3 program to print an array
# in alternate sorted manner.

# Function to print alternate sorted
# values
def alternateSort(arr, n):

    # Sorting the array
    arr.sort()

    # Printing the last element of array
    # first and then first element and then
    # second last element and then second
    # element and so on.
    i = 0
    j = n-1

    while (i < j):

        print(arr[j], end = " ")
        j-= 1
        print(arr[i], end = " ")
        i+= 1

    # If the total element in array is odd
    # then print the last middle element.
    if (n % 2 != 0):
```

```
print(arr[i])

# Driver code
arr = [1, 12, 4, 6, 7, 10]
n = len(arr)

alternateSort(arr, n)

# This code is contributed by
# Smitha Dinesh Semwal

C#
// C# program to print an array in alternate
// sorted manner
using System;

class AlternativeString {

    // Function to print alternate sorted values
    static void alternateSort(int[] arr, int n)
    {
        Array.Sort(arr);

        // Printing the last element of array
        // first and then first element and then
        // second last element and then second
        // element and so on.
        int i = 0, j = n - 1;
        while (i < j) {
            Console.Write(arr[j--] + " ");
            Console.Write(arr[i++] + " ");
        }

        // If the total element in array is odd
        // then print the last middle element.
        if (n % 2 != 0)
            Console.WriteLine(arr[i]);
    }

    /* Driver program to test above functions */
    public static void Main()
    {
        int[] arr = { 1, 12, 4, 6, 7, 10 };
        int n = arr.Length;
        alternateSort(arr, n);
    }
}
```

```
}
```

```
// This article is contributed by vt_m.
```

PHP

```
<?php
// PHP program to print an array in
// alternate sorted manner.

// Function to print alternate
// sorted values
function alternateSort($arr, $n)
{
    // Sorting the array
    sort($arr);

    // Printing the last element
    // of array first and then
    // first element and then second
    // last element and then second
    // element and so on.
    $i = 0;
    $j = $n - 1;
    while ($i < $j)
    {
        echo $arr[$j--]." ";
        echo $arr[$i++]." ";
    }

    // If the total element in array
    // is odd then print the last
    // middle element.
    if ($n % 2 != 0)
        echo $arr[$i];
}

// Driver code
$arr= array(1, 12, 4, 6, 7, 10);
$n = sizeof($arr) / sizeof($arr[0]);

alternateSort($arr, $n);

// This code is contributed by Mithun Kumar
?>
```

Output :

12 1 10 4 7 6

Time Complexity: $O(n \log n)$

Auxiliary Space : $O(1)$

Improved By : [Varun Bothra](#), Mithun Kumar

Source

<https://www.geeksforgeeks.org/alternative-sorting/>

Chapter 10

An Insertion Sort time complexity question

An Insertion Sort time complexity question - GeeksforGeeks

Question : How much time [Insertion sort](#) takes to sort an array of size n in below form?

arr[] = 2, 1, 4, 3, 6, 5,...i, i-1,n, n-1

Answer : At first look, it seems like Insertion Sort would take $O(n^2)$ time, but it actually takes $O(n)$ time

How? Let us take a closer look at below code.

```
/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
    for (int i = 1; i < n; i++)
    {
        int key = arr[i];
        int j = i-1;

        /* Move elements of arr[0..i-1], that are
           greater than key, to one position ahead
           of their current position */
        while (j >= 0 && arr[j] > key)
        {
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j+1] = key;
    }
}
```

If we analyze the input carefully we see that every element is only one position away from its position in sorted array. The outer for loop will run till 'n' and the inner while loop would take “constant” steps of 1 swap and 2 comparisons. Since, while loop takes constant time and for loop runs for ‘n’ element, so overall complexity is $O(n)$

Alternate Answer : Another way to look at this is, [time taken by Insertion Sort is proportional to number of inversions in an array](#). In above example type, number of inversions is $n/2$, so overall time complexity is $O(n)$

Source

<https://www.geeksforgeeks.org/insertion-sort-time-complexity-question/>

Chapter 11

Asymptotic Analysis and comparison of sorting algorithms

Asymptotic Analysis and comparison of sorting algorithms - GeeksforGeeks

It is a well established fact that merge sort runs faster than insertion sort. Using [asymptotic analysis](#) we can prove that merge sort runs in $O(n \log n)$ time and insertion sort takes $O(n^2)$. It is obvious because merge sort uses a divide-and-conquer approach by recursively solving the problems whereas insertion sort follows an incremental approach.

If we scrutinize the time complexity analysis even further, we'll get to know that insertion sort isn't that bad enough. Surprisingly, insertion sort beats merge sort on smaller input size. This is because there are few constants which we ignore while deducing the time complexity. On larger input sizes of the order 10^4 this doesn't influence the behavior of our function. But when input sizes fall below, say less than 40, then the constants in the equation dominate the input size 'n'.

So far, so good. But I wasn't satisfied with such mathematical analysis. As a computer science undergrad we must believe in writing code. I've written a C program to get a feel of how the algorithms compete against each other for various input sizes. And also, why such rigorous mathematical analysis is done on establishing running time complexities of these sorting algorithms.

Implementation:

```
//C++ code to compare performance of sorting algorithms
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#define MAX_ELEMENT_IN_ARRAY 1000000001
```

```

int cmpfunc (const void * a, const void * b)
{
    // Compare function used by qsort
    return ( *(int*)a - *(int*)b );
}

int* generate_random_array(int n)
{
    srand(time(NULL));
    int *a = malloc(sizeof(int) * n), i;
    for(i = 0; i < n; ++i)
        a[i] = rand() % MAX_ELEMENT_IN_ARRAY;
    return a;
}

int* copy_array(int a[], int n)
{
    int *arr = malloc(sizeof(int) * n);
    int i;
    for(i = 0; i < n ;++i)
        arr[i] = a[i];
    return arr;
}

//Code for Insertion Sort
void insertion_sort_asc(int a[], int start, int end)
{
    int i;
    for(i = start + 1; i <= end ; ++i)
    {
        int key = a[i];
        int j = i - 1;
        while(j >= start && a[j] > key)
        {
            a[j + 1] = a[j];
            --j;
        }
        a[j + 1] = key;
    }
}

//Code for Merge Sort
void merge(int a[], int start, int end, int mid)
{
    int i = start, j = mid + 1, k = 0;
    int *aux = malloc(sizeof(int) * (end - start + 1));
    while(i <= mid && j <= end)
    {

```

```

        if(a[i] <= a[j])
            aux[k++] = a[i++];
        else
            aux[k++] = a[j++];
    }
    while(i <= mid)
        aux[k++] = a[i++];
    while(j <= end)
        aux[k++] = a[j++];
    j = 0;
    for(i = start;i <= end;++i)
        a[i] = aux[j++];
    free(aux);
}

void _merge_sort(int a[],int start,int end)
{
    if(start < end)
    {
        int mid = start + (end - start) / 2;
        _merge_sort(a,start,mid);
        _merge_sort(a,mid + 1,end);
        merge(a,start,end,mid);
    }
}
void merge_sort(int a[],int n)
{
    return _merge_sort(a,0,n - 1);
}

void insertion_and_merge_sort_combine(int a[], int start, int end, int k)
{
    // Performs insertion sort if size of array is less than or equal to k
    // Otherwise, uses mergesort
    if(start < end)
    {
        int size = end - start + 1;

        if(size <= k)
        {
            //printf("Performed insertion sort- start = %d and end = %d\n", start, end);
            return insertion_sort_asc(a,start,end);
        }
        int mid = start + (end - start) / 2;
        insertion_and_merge_sort_combine(a,start,mid,k);
        insertion_and_merge_sort_combine(a,mid + 1,end,k);
        merge(a,start,end,mid);
    }
}

```

```

        }
    }

void test_sorting_runtimes(int size,int num_of_times)
{
    // Measuring the runtime of the sorting algorithms
    int number_of_times = num_of_times;
    int t = number_of_times;
    int n = size;
    double insertion_sort_time = 0, merge_sort_time = 0;
    double merge_sort_and_insertion_sort_mix_time = 0, qsort_time = 0;
    while(t--)
    {
        clock_t start, end;

        int *a = generate_random_array(n);
        int *b = copy_array(a,n);
        start = clock();
        insertion_sort_asc(b,0,n-1);
        end = clock();
        insertion_sort_time += ((double) (end - start)) / CLOCKS_PER_SEC;
        free(b);
        int *c = copy_array(a,n);
        start = clock();
        merge_sort(c,n);
        end = clock();
        merge_sort_time += ((double) (end - start)) / CLOCKS_PER_SEC;
        free(c);
        int *d = copy_array(a,n);
        start = clock();
        insertion_and_merge_sort_combine(d,0,n-1,40);
        end = clock();
        merge_sort_and_insertion_sort_mix_time+=((double) (end - start))/CLOCKS_PER_SEC;
        free(d);
        start = clock();
        qsort(a,n,sizeof(int),cmpfunc);
        end = clock();
        qsort_time += ((double) (end - start)) / CLOCKS_PER_SEC;
        free(a);
    }

    insertion_sort_time /= number_of_times;
    merge_sort_time /= number_of_times;
    merge_sort_and_insertion_sort_mix_time /= number_of_times;
    qsort_time /= number_of_times;
    printf("\nTime taken to sort:\n"
          "%-35s %f\n"
          "%-35s %f\n"

```

```

        "%-35s %f\n"
        "%-35s %f\n\n",
        "(i)Insertion sort: ",
        insertion_sort_time,
        "(ii)Merge sort: ",
        merge_sort_time,
        "(iii)Insertion-mergesort-hybrid: ",
        merge_sort_and_insertion_sort_mix_time,
        "(iv)Qsort library function: ",
        qsort_time);
    }

int main(int argc, char const *argv[])
{
    int t;
    scanf("%d", &t);
    while(t--)
    {
        int size, num_of_times;
        scanf("%d %d", &size, &num_of_times);
        test_sorting_runtimes(size,num_of_times);
    }
    return 0;
}

```

I have compared the running times of the following algorithms:

- **Insertion sort:** The traditional algorithm with no modifications/optimisation. It performs very well for smaller input sizes. And yes, it does beat merge sort
- **Merge sort:** Follows the divide-and-conquer approach. For input sizes of the order 10^5 this algorithm is of the right choice. It renders insertion sort impractical for such large input sizes.
- **Combined version of insertion sort and merge sort:** I have tweaked the logic of merge sort a little bit to achieve a considerably better running time for smaller input sizes. As we know, merge sort splits its input into two halves until it is trivial enough to sort the elements. But here, when the input size falls below a threshold such as ' $n < 40$ ' then this hybrid algorithm makes a call to traditional insertion sort procedure. From the fact that insertion sort runs faster on smaller inputs and merge sort runs faster on larger inputs, this algorithm makes best use both the worlds.
- **Quick sort:** I have not implemented this procedure. This is the library function `qsort()` which is available in . I have considered this algorithm in order to know the significance of implementation. It requires a great deal of programming expertise to minimize the number of steps and make at most use of the underlying language primitives to implement an algorithm in the best way possible. This is the main reason why it is recommended to use library functions. They are written to handle anything

and everything. They optimize to the maximum extent possible. And before I forget, from my analysis qsort() runs blazingly fast on virtually any input size!

The Analysis:

- **Input:** The user has to supply the number of times he/she wants to test the algorithm corresponding to number of test cases. For each test case the user must enter two space separated integers denoting the input size 'n' and the 'num_of_times' denoting the number of times he/she wants to run the analysis and take average. (Clarification: If 'num_of_times' is 10 then each of the algorithm specified above runs 10 times and the average is taken. This is done because the input array is generated randomly corresponding to the input size which you specify. The input array could be all sorted. Our it could correspond to the worst case i.e. descending order. In order to avoid running times of such input arrays. The algorithm is run 'num_of_times' and the average is taken.)
clock() routine and CLOCKS_PER_SEC macro from is used to measure the time taken.
Compilation: I have written the above code in Linux environment (Ubuntu 16.04 LTS). Copy the code snippet above. Compile it using gcc, key in the inputs as specified and admire the power of sorting algorithms!
- **Results:** As you can see for small input sizes, insertion sort beats merge sort by $2 * 10^{-6}$ sec. But this difference in time is not so significant. On the other hand, the hybrid algorithm and qsort() library function, both perform as good as insertion sort.

```
aditya@Aditya-laptop:~/Desktop$ gcc running_times.c
aditya@Aditya-laptop:~/Desktop$ ./a.out
1
30 10

Time taken to sort:
(i)Insertion sort:          0.000001
(ii)Merge sort:            0.000003
(iii)Insertion-mergesort-hybrid: 0.000001
(iv)Qsort library function: 0.000001
```

The input size is now increased by approximately 100 times to $n = 1000$ from $n = 30$. The difference is now tangible. Merge sort runs 10 times faster than insertion sort. There is again a tie between the performance of the hybrid algorithm and the qsort() routine. This suggests that the qsort() is implemented in a way which is more or less similar to our hybrid algorithm i.e., switching between different algorithms to make the best out of them.

```
aditya@Aditya-laptop:~/Desktop$ ./a.out
1
1000 10

Time taken to sort:
(i)Insertion sort:          0.001311
(ii)Merge sort:             0.000279
(iii)Insertion-mergesort-hybrid: 0.000174
(iv)Qsort library function: 0.000171
```

Finally, the input size is increased to 10^5 (1 Lakh!) which is most probably the ideal size used in practical scenario's. Compared to the previous input $n = 1000$ where merge sort beat insertion sort by running 10 times faster, here the difference is even more significant. Merge sort beats insertion sort by 100 times!

The hybrid algorithm which we have written in fact does out perform the traditional merge sort by running 0.01 sec faster. And lastly, qsort() the library function, finally proves us that implementation also plays a crucial role while measuring the running times meticulously by running 3 milliseconds faster!

```
aditya@Aditya-laptop:~/Desktop$ ./a.out
1
100000 10

Time taken to sort:
(i)Insertion sort:          8.127113
(ii)Merge sort:              0.024675
(iii)Insertion-mergesort-hybrid: 0.018627
(iv)Qsort library function: 0.015837
```

Note: Do not run the above program with $n \geq 10^6$ since it will take a lot of computing power. Thank you and Happy coding!

Source

<https://www.geeksforgeeks.org/asymptotic-analysis-comparison-sorting-algorithms/>

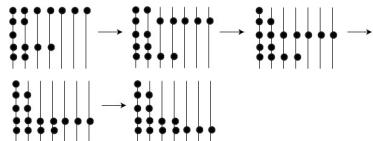
Chapter 12

Bead Sort A Natural Sorting Algorithm

Bead Sort A Natural Sorting Algorithm - GeeksforGeeks

Also known as Gravity sort, this algorithm was inspired from natural phenomenons and was designed keeping in mind objects(or beads) falling under the influence of gravity.

The Idea: Positive numbers are represented by a set of beads like those on an abacus.



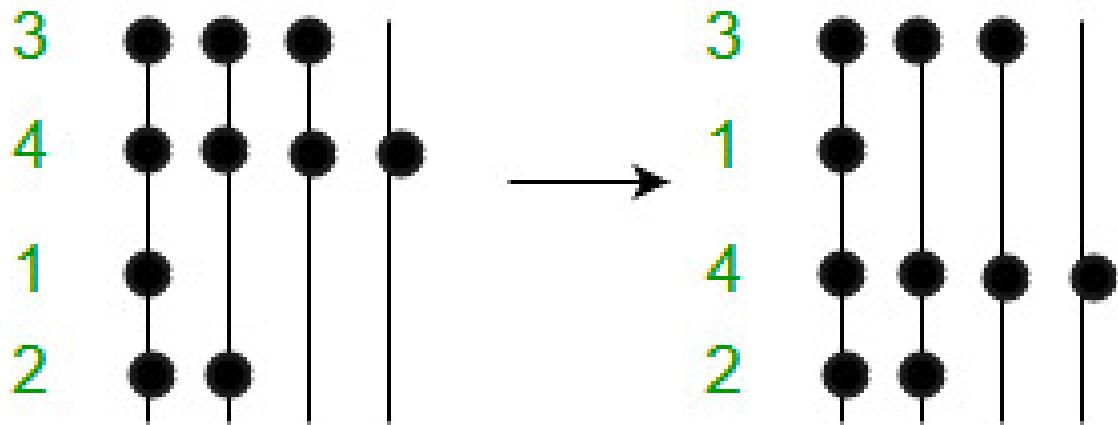
Sorting of {7, 2, 1, 4, 2} using Bead Sort. Beads fall down one by one if there is space below.

1. Like in the image above the beads represent the following numbers from top to bottom : 7, 2, 1, 4, 2. Now, imagine that this is the position of the beads at time $t = 0$ and with every passing second the beads will fall down by one level provided there is no bead already present below them. In such a case, they just rest upon the bead below them.(Rods are numbered from left to right and levels are numbered from the bottom as 1, 2, n.)
2. Now, at time $t = 1$ the bottom 2 beads in the first two rods from the left stay at their positions while the second bead from the top from the second rod comes down by one level to rest upon the bead below it. The beads in the 3rd and 4th rod at level 2 come down to level 1. Simultaneously, the beads in the rods 3 to 7 come down by one level. Now, the numbers from top to bottom become: 2, 6, 2, 2, 4.
3. This goes on till time $t = 4$ where we get the sorted sequence of numbers from top to bottom which is: 1, 2, 2, 4, 7.

Why is it called so?

When one tries to visualize this algorithm it appears as if beads are falling down under the influence of gravity to the bottom-most level they can reach resulting the set of beads being arranged in a descending order from the ground up. If you are having trouble visualizing this visit this [link](#)

Lets say that we have to sort the numbers 3, 4, 1, 2. The above algorithm would work like this.



Bead Sort Working

Below is the code, try to implement it yourself before looking at the code.

The Code:

```
// C++ program to implement gravity/bead sort
#include <bits/stdc++.h>
using namespace std;

#define BEAD(i, j) beads[i * max + j]

// function to perform the above algorithm
void beadSort(int *a, int len)
{
    // Find the maximum element
    int max = a[0];
    for (int i = 1; i < len; i++)
        if (a[i] > max)
            max = a[i];

    // allocating memory
    unsigned char beads[max*len];
    memset(beads, 0, sizeof(beads));

    // mark the beads
    for (int i = 0; i < len; i++)
        for (int j = 0; j < max; j++)
            if (a[i] > j)
                beads[i * max + j] = 1;
}
```

```
for (int i = 0; i < len; i++)
    for (int j = 0; j < a[i]; j++)
        BEAD(i, j) = 1;

for (int j = 0; j < max; j++)
{
    // count how many beads are on each post
    int sum = 0;
    for (int i=0; i < len; i++)
    {
        sum += BEAD(i, j);
        BEAD(i, j) = 0;
    }

    // Move beads down
    for (int i = len - sum; i < len; i++)
        BEAD(i, j) = 1;
}

// Put sorted values in array using beads
for (int i = 0; i < len; i++)
{
    int j;
    for (j = 0; j < max && BEAD(i, j); j++);

    a[i] = j;
}

// driver function to test the algorithm
int main()
{
    int a[] = {5, 3, 1, 7, 4, 1, 1, 20};
    int len = sizeof(a)/sizeof(a[0]);

    beadSort(a, len);

    for (int i = 0; i < len; i++)
        printf("%d ", a[i]);

    return 0;
}
```

Output:

1 1 1 3 4 5 7 20

Time Complexity:

The algorithm's run-time complexity ranges from $O(1)$ to $O(S)$ (S is the sum of the input integers) depending on the user's perspective. Finally, three possible implementations are suggested.

1. **$O(1)$** : Dropping all beads together as a single (simultaneous) operation. This complexity cannot be implemented in practice.
2. **$O(\sqrt{n})$** : In a realistic physical model that uses gravity, the time it takes to let the beads fall is proportional to the square root of the maximum height, which is proportional to n .
3. **$O(n)$** : Dropping the row of beads in the frame (representing a number) as a distinct operation since the number of rows is equal to n .
4. **$O(S)$** : Dropping each and every bead' as a separate operation since S is the sum of all the beads.

Like the [Pigeonhole sort](#), bead sort is unusual in that in worst case it can perform faster than $O(n \log n)$, the fastest performance possible for a comparison sort in worst case. This is possible because the key for a bead sort is always a positive integer and bead sort exploits its structure.

Space Complexity: Bead sort is the record-holder as for waste. The costs for the extra memory exceed the costs for storing the array itself. Its memory complexity is $O(\frac{n^2}{2})$

References:

- https://www.wikiwand.com/en/Bead_sort
- <https://kukuruku.co/post/bead-sort/>
- https://rosettacode.org/wiki/Sorting_algorithms/Bead_sort
- <https://www.cs.auckland.ac.nz/~mjd/misc/BeadSort5.pdf>

Source

<https://www.geeksforgeeks.org/bead-sort-natural-sorting-algorithm/>

Chapter 13

Binary Insertion Sort

Binary Insertion Sort - GeeksforGeeks

We can use binary search to reduce the number of comparisons in [normal insertion sort](#). Binary Insertion Sort uses binary search to find the proper location to insert the selected item at each iteration.

In normal insertion sort, it takes $O(n)$ comparisons(at nth iteration) in worst case. We can reduce it to $O(\log n)$ by using [binary search](#).

C/C++

```
// C program for implementation of binary insertion sort
#include <stdio.h>

// A binary search based function to find the position
// where item should be inserted in a[low..high]
int binarySearch(int a[], int item, int low, int high)
{
    if (high <= low)
        return (item > a[low])? (low + 1): low;

    int mid = (low + high)/2;

    if(item == a[mid])
        return mid+1;

    if(item > a[mid])
        return binarySearch(a, item, mid+1, high);
    return binarySearch(a, item, low, mid-1);
}

// Function to sort an array a[] of size 'n'
void insertionSort(int a[], int n)
{
```

```
int i, loc, j, k, selected;

for (i = 1; i < n; ++i)
{
    j = i - 1;
    selected = a[i];

    // find location where selected could be insereted
    loc = binarySearch(a, selected, 0, j);

    // Move all elements after location to create space
    while (j >= loc)
    {
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = selected;
}

// Driver program to test above function
int main()
{
    int a[] = {37, 23, 0, 17, 12, 72, 31,
               46, 100, 88, 54};
    int n = sizeof(a)/sizeof(a[0]), i;

    insertionSort(a, n);

    printf("Sorted array: \n");
    for (i = 0; i < n; i++)
        printf("%d ",a[i]);

    return 0;
}
```

Java

```
// Java Program implementing
// binary insertion sort

import java.util.Arrays;
class GFG
{
    public static void main(String[] args)
    {
        final int[] arr = {37, 23, 0, 17, 12, 72, 31,
                           46, 100, 88, 54 };
```

```
new GFG().sort(arr);

for(int i=0; i<arr.length; i++)
    System.out.print(arr[i]+" ");
}

public void sort(int array[])
{
    for (int i = 1; i < array.length; i++)
    {
        int x = array[i];

        // Find location to insert using binary search
        int j = Math.abs((Arrays.binarySearch(array, 0, i, x) + 1);

        //Shifting array to one location right
        System.arraycopy(array, j, array, j+1, i-j);

        //Placing element at its correct location
        array[j] = x;
    }
}

// Code contributed by Mohit Gupta_OMG
```

Python

```
# Python Program implementation
# of binary insertion sort

def binary_search(arr, val, start, end):
    # we need to distinguish whether we should insert
    # before or after the left boundary.
    # imagine [0] is the last step of the binary search
    # and we need to decide where to insert -1
    if start == end:
        if arr[start] > val:
            return start
        else:
            return start+1

    # this occurs if we are moving beyond left's boundary
    # meaning the left boundary is the least position to
    # find a number greater than val
    if start > end:
        return start
```

```
mid = (start+end)/2
if arr[mid] < val:
    return binary_search(arr, val, mid+1, end)
elif arr[mid] > val:
    return binary_search(arr, val, start, mid-1)
else:
    return mid

def insertion_sort(arr):
    for i in xrange(1, len(arr)):
        val = arr[i]
        j = binary_search(arr, val, 0, i-1)
        arr = arr[:j] + [val] + arr[j:i] + arr[i+1:]
    return arr

print("Sorted array:")
print insertion_sort([37, 23, 0, 17, 12, 72, 31,
                     46, 100, 88, 54])

# Code contributed by Mohit Gupta_OMG
```

C#

```
// C# Program implementing
// binary insertion sort
using System;

class GFG {

    public static void Main()
    {
        int []arr = {37, 23, 0, 17, 12, 72, 31,
                    46, 100, 88, 54 };

        sort(arr);

        for(int i = 0; i < arr.Length; i++)
            Console.Write(arr[i] + " ");
    }

    public static void sort(int []array)
    {
        for (int i = 1; i < array.Length; i++)
        {
            int x = array[i];

            // Find location to insert using
```

```
// binary search
int j = Math.Abs(Array.BinarySearch(
    array, 0, i, x) + 1);

// Shifting array to one location right
System.Array.Copy(array, j, array,
    j+1, i-j);

// Placing element at its correct
// location
array[j] = x;
}

}

}

// This code is contributed by nitin mittal.
```

Output:

Sorted array:
0 12 17 23 31 37 46 54 72 88 100

Time Complexity: The algorithm as a whole still has a running worst case running time of $O(n^2)$ because of the series of swaps required for each insertion.

This article is contributed by **Amit Auddy**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [nitin mittal](#), [SayanMaiti](#), [Shubhaw Kumar](#)

Source

<https://www.geeksforgeeks.org/binary-insertion-sort/>

Chapter 14

Bitonic Sort

Bitonic Sort - GeeksforGeeks

Background

Bitonic Sort is a classic parallel algorithm for sorting.

- Bitonic sort does $O(n \log^2 n)$ comparisons.
- The number of comparisons done by Bitonic sort are more than popular sorting algorithms like Merge Sort [does $O(n \log n)$ comparisons], but Bitonic sort is better for parallel implementation because we always compare elements in predefined sequence and the sequence of comparison doesn't depend on data. Therefore it is suitable for implementation in hardware and [parallel processor array](#).

To understand Bitonic Sort, we must first understand what is Bitonic Sequence and how to make a given sequence Bitonic.

Bitonic Sequence

A sequence is called Bitonic if it is first increasing, then decreasing. In other words, an array $\text{arr}[0..n-1]$ is Bitonic if there exists an index i where $0 \leq i \leq n-1$ such that

$x_0 \leq x_1 \dots \leq x_i \text{ and } x_i \geq x_{i+1} \dots \geq x_{n-1}$

1. A sequence, sorted in increasing order is considered Bitonic with the decreasing part as empty. Similarly, decreasing order sequence is considered Bitonic with the increasing part as empty.
2. A rotation of Bitonic Sequence is also bitonic.

How to form a Bitonic Sequence from a random input?

We start by forming 4-element bitonic sequences from consecutive 2-element sequence. Consider 4-element in sequence x_0, x_1, x_2, x_3 . We sort x_0 and x_1 in ascending order and x_2

and x_3 in descending order. We then concatenate the two pairs to form a 4 element bitonic sequence.

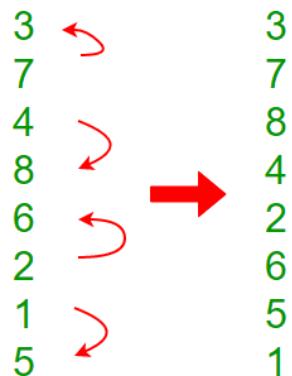
Next, we take two 4 element bitonic sequences, sorting one in ascending order, the other in descending order (using the Bitonic Sort which we will discuss below), and so on, until we obtain the bitonic sequence.

Example:

Convert the following sequence to bitonic sequence: 3, 7, 4, 8, 6, 2, 1, 5

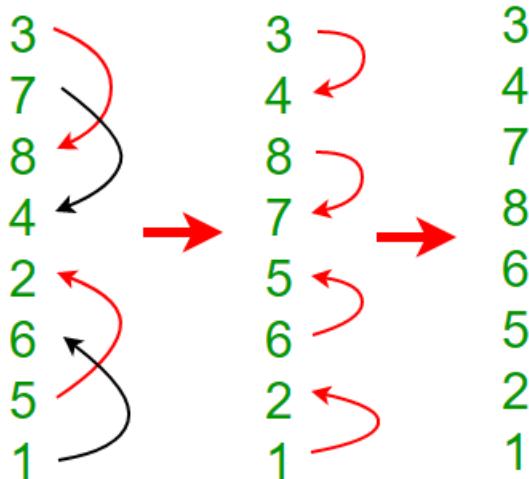
Step 1: Consider each 2-consecutive elements as bitonic sequence and apply bitonic sort on each 2- pair elements. In next step, take two 4 element bitonic sequences and so on.

$$\begin{array}{l} \text{a} \curvearrowleft = \frac{\text{Min}(a,b)}{\text{Max}(a,b)} \\ \text{b} \curvearrowleft = \frac{\text{Max}(a,b)}{\text{Min}(a,b)} \end{array}$$



Note: x_0 and x_1 are sorted in ascending order and x_2 and x_3 in descending order and so on

Step 2: Two 4 element bitonic sequences : **A**(3,7,8,4) and **B**(2,6,5,1) with comparator length as 2



After this step, we'll get Bitonic sequence of length 8.

3, 4, 7, 8, 6, 5, 2, 1

Bitonic Sorting

It mainly involves two steps.

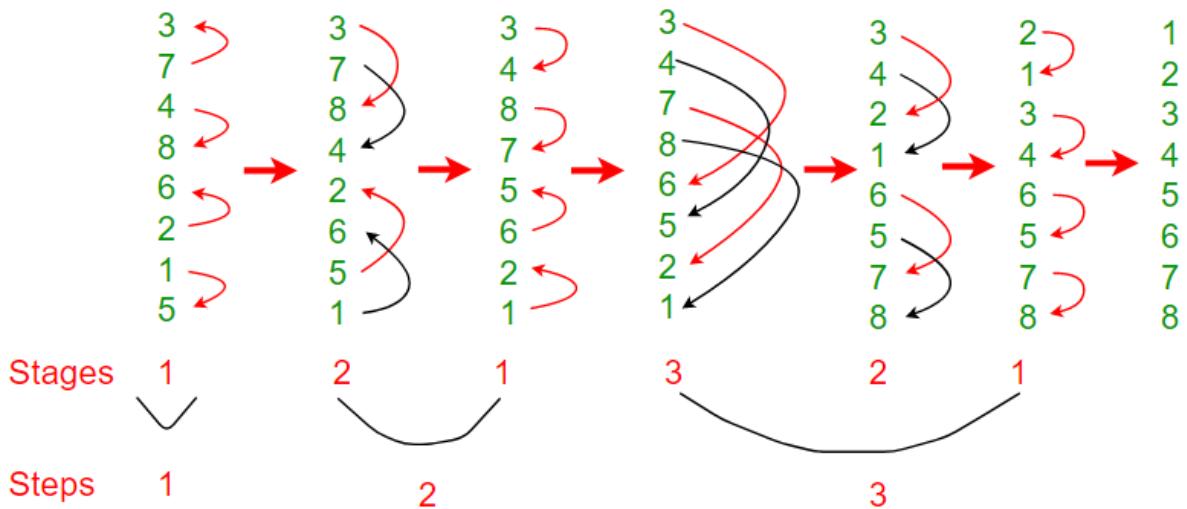
1. Forming a bitonic sequence (discussed above in detail). After this step we reach the fourth stage in below diagram, i.e., the array becomes {3, 4, 7, 8, 6, 5, 2, 1}
2. Creating one sorted sequence from bitonic sequence : After first step, first half is sorted in increasing order and second half in decreasing order.

We compare first element of first half with first element of second half, then second element of first half with second element of second and so on. We exchange elements if an element of first half is smaller.

After above compare and exchange steps, we get two bitonic sequences in array. See fifth stage in below diagram. In the fifth stage, we have {3, 4, 2, 1, 6, 5, 7, 8}. If we take a closer look at the elements, we can notice that there are two bitonic sequences of length $n/2$ such that all elements in first bitonic sequence {3, 4, 2, 1} are smaller than all elements of second bitonic sequence {6, 5, 7, 8}.

We repeat the same process within two bitonic sequences and we get four bitonic sequences of length $n/4$ such that all elements of leftmost bitonic sequence are smaller and all elements of rightmost. See sixth stage in below diagram, arrays is {2, 1, 3, 4, 6, 5, 7, 8}.

If we repeat this process one more time we get 8 bitonic sequences of size $n/8$ which is 1. Since all these bitonic sequence are sorted and every bitonic sequence has one element, we get the sorted array.



Below are implementations of Bitonic Sort.

C++

```
/* C++ Program for Bitonic Sort. Note that this program
   works only when size of input is a power of 2. */
#include<bits/stdc++.h>
using namespace std;

/*The parameter dir indicates the sorting direction, ASCENDING
   or DESCENDING; if (a[i] > a[j]) agrees with the direction,
   then a[i] and a[j] are interchanged.*/
void compAndSwap(int a[], int i, int j, int dir)
{
    if (dir==(a[i]>a[j]))
        swap(a[i],a[j]);
}

/*It recursively sorts a bitonic sequence in ascending order,
   if dir = 1, and in descending order otherwise (means dir=0).
   The sequence to be sorted starts at index position low,
   the parameter cnt is the number of elements to be sorted.*/
void bitonicMerge(int a[], int low, int cnt, int dir)
{
    if (cnt>1)
    {
        int k = cnt/2;
        for (int i=low; i<low+k; i++)
            compAndSwap(a, i, i+k, dir);
        bitonicMerge(a, low, k, dir);
        bitonicMerge(a, low+k, k, dir);
    }
}
```

```
        }
    }

/* This function first produces a bitonic sequence by recursively
   sorting its two halves in opposite sorting orders, and then
   calls bitonicMerge to make them in the same order */
void bitonicSort(int a[],int low, int cnt, int dir)
{
    if (cnt>1)
    {
        int k = cnt/2;

        // sort in ascending order since dir here is 1
        bitonicSort(a, low, k, 1);

        // sort in descending order since dir here is 0
        bitonicSort(a, low+k, k, 0);

        // Will merge wole sequence in ascending order
        // since dir=1.
        bitonicMerge(a,low, cnt, dir);
    }
}

/* Caller of bitonicSort for sorting the entire array of
   length N in ASCENDING order */
void sort(int a[], int N, int up)
{
    bitonicSort(a,0, N, up);
}

// Driver code
int main()
{
    int a[]={3, 7, 4, 8, 6, 2, 1, 5};
    int N = sizeof(a)/sizeof(a[0]);

    int up = 1;    // means sort in ascending order
    sort(a, N, up);

    printf("Sorted array: \n");
    for (int i=0; i<N; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Java

```
/* Java program for Bitonic Sort. Note that this program
   works only when size of input is a power of 2. */
public class BitonicSort
{
    /* The parameter dir indicates the sorting direction,
       ASCENDING or DESCENDING; if (a[i] > a[j]) agrees
       with the direction, then a[i] and a[j] are
       interchanged. */
    void compAndSwap(int a[], int i, int j, int dir)
    {
        if ( (a[i] > a[j] && dir == 1) ||
            (a[i] < a[j] && dir == 0))
        {
            // Swapping elements
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }

    /* It recursively sorts a bitonic sequence in ascending
       order, if dir = 1, and in descending order otherwise
       (means dir=0). The sequence to be sorted starts at
       index position low, the parameter cnt is the number
       of elements to be sorted.*/
    void bitonicMerge(int a[], int low, int cnt, int dir)
    {
        if (cnt>1)
        {
            int k = cnt/2;
            for (int i=low; i<low+k; i++)
                compAndSwap(a,i, i+k, dir);
            bitonicMerge(a,low, k, dir);
            bitonicMerge(a,low+k, k, dir);
        }
    }

    /* This function first produces a bitonic sequence by
       recursively sorting its two halves in opposite sorting
       orders, and then calls bitonicMerge to make them in
       the same order */
    void bitonicSort(int a[], int low, int cnt, int dir)
    {
        if (cnt>1)
        {
            int k = cnt/2;

            // sort in ascending order since dir here is 1
```

```
bitonicSort(a, low, k, 1);

// sort in descending order since dir here is 0
bitonicSort(a, low+k, k, 0);

// Will merge whole sequence in ascending order
// since dir=1.
bitonicMerge(a, low, cnt, dir);
}

}

/* Caller of bitonicSort for sorting the entire array
   of length N in ASCENDING order */
void sort(int a[], int N, int up)
{
    bitonicSort(a, 0, N, up);
}

/* A utility function to print array of size n */
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

// Driver method
public static void main(String args[])
{
    int a[] = {3, 7, 4, 8, 6, 2, 1, 5};
    int up = 1;
    BitonicSort ob = new BitonicSort();
    ob.sort(a, a.length, up);
    System.out.println("\nSorted array");
    printArray(a);
}
}
```

Python

```
# Python program for Bitonic Sort. Note that this program
# works only when size of input is a power of 2.

# The parameter dir indicates the sorting direction, ASCENDING
# or DESCENDING; if (a[i] > a[j]) agrees with the direction,
# then a[i] and a[j] are interchanged.*/
def compAndSwap(a, i, j, dire):
```

```

if (dire==1 and a[i] > a[j]) or (dire==0 and a[i] < a[j]):
    a[i],a[j] = a[j],a[i]

# It recursively sorts a bitonic sequence in ascending order,
# if dir = 1, and in descending order otherwise (means dir=0).
# The sequence to be sorted starts at index position low,
# the parameter cnt is the number of elements to be sorted.
def bitonicMerge(a, low, cnt, dire):
    if cnt > 1:
        k = cnt/2
        for i in range(low , low+k):
            compAndSwap(a, i, i+k, dire)
        bitonicMerge(a, low, k, dire)
        bitonicMerge(a, low+k, k, dire)

# This function first produces a bitonic sequence by recursively
# sorting its two halves in opposite sorting orders, and then
# calls bitonicMerge to make them in the same order
def bitonicSort(a, low, cnt,dire):
    if cnt > 1:
        k = cnt/2
        bitonicSort(a, low, k, 1)
        bitonicSort(a, low+k, k, 0)
        bitonicMerge(a, low, cnt, dire)

# Caller of bitonicSort for sorting the entire array of length N
# in ASCENDING order
def sort(a,N, up):
    bitonicSort(a,0, N, up)

# Driver code to test above
a = [3, 7, 4, 8, 6, 2, 1, 5]
n = len(a)
up = 1

sort(a, n, up)
print ("\n\nSorted array is")
for i in range(n):
    print("%d" %a[i]),

```

Output:

```

Sorted array:
1 2 3 4 5 6 7 8

```

Analysis of Bitonic Sort

To form a sorted sequence of length n from two sorted sequences of length $n/2$, $\log(n)$ comparisons are required (for example: $\log(8) = 3$ when sequence size. Therefore, The number of comparisons $T(n)$ of the entire sorting is given by:

$$T(n) = \log(n) + T(n/2)$$

The solution of this recurrence equation is

$$T(n) = \log(n) + \log(n)-1 + \log(n)-2 + \dots + 1 = \log(n) \cdot (\log(n)+1) / 2$$

As, each stage of the sorting network consists of $n/2$ comparators. Therefore total $\Theta(n \log^2 n)$ comparators.

References:

1. <https://www.youtube.com/watch?v=GEQ8y26blEY>
2. <http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm>
3. https://en.wikipedia.org/wiki/Bitonic_sorter

Source

<https://www.geeksforgeeks.org/bitonic-sort/>

Chapter 15

BogoSort or Permutation Sort

BogoSort or Permutation Sort - GeeksforGeeks

BogoSort also known as permutation sort, stupid sort, slow sort, shotgun sort or monkey sort is a particularly ineffective algorithm based on generate and test paradigm. The algorithm successively generates permutations of its input until it finds one that is sorted.[\(Wiki\)](#)

For example, if bogosort is used to sort a deck of cards, it would consist of checking if the deck were in order, and if it were not, one would throw the deck into the air, pick the cards up at random, and repeat the process until the deck is sorted.

PseudoCode:

```
while not Sorted(list) do
    shuffle (list)
done
```

Example:

Let us consider an example array (3 2 5 1 0 4)

4 5 0 3 2 1 (1st shuffling)

4 1 3 2 5 0 (2nd shuffling)

1 0 3 2 5 4 (3rd shuffling)

3 1 0 2 4 5 (4th shuffling)

1 4 5 0 3 2 (5th shuffling)

.

.

.

0 1 2 3 4 5 (nth shuffling) —— Sorted Array

Here, n is unknown because algorithm doesn't know in which step the resultant permutation will come out to be sorted.

C++

```
// C++ implementation of Bogo Sort
#include<bits/stdc++.h>
using namespace std;

// To check if array is sorted or not
bool isSorted(int a[], int n)
{
    while ( --n > 1 )
        if (a[n] < a[n-1])
            return false;
    return true;
}

// To generate permutation of the array
void shuffle(int a[], int n)
{
    for (int i=0; i < n; i++)
        swap(a[i], a[rand()%n]);
}

// Sorts array a[0..n-1] using Bogo sort
void bogosort(int a[], int n)
{
    // if array is not sorted then shuffle
    // the array again
    while ( !isSorted(a, n) )
        shuffle(a, n);
}

// prints the array
void printArray(int a[], int n)
{
    for (int i=0; i<n; i++)
        printf("%d ", a[i]);
    printf("\n");
}

// Driver code
int main()
{
    int a[] = {3, 2, 5, 1, 0, 4};
    int n = sizeof a/sizeof a[0];
    bogosort(a, n);
    printf("Sorted array :\n");
    printArray(a,n);
    return 0;
}
```

Java

```
// Java Program to implement BogoSort
public class BogoSort
{
    // Sorts array a[0..n-1] using Bogo sort
    void bogoSort(int[] a)
    {
        // if array is not sorted then shuffle the
        // array again
        while (isSorted(a) == false)
            shuffle(a);
    }

    // To generate permutation of the array
    void shuffle(int[] a)
    {
        // Math.random() returns a double positive
        // value, greater than or equal to 0.0 and
        // less than 1.0.
        for (int i=1; i <= n; i++)
            swap(a, i, (int)(Math.random()*i));
    }

    // Swapping 2 elements
    void swap(int[] a, int i, int j)
    {
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }

    // To check if array is sorted or not
    boolean isSorted(int[] a)
    {
        for (int i=1; i < a.length; i++)
            if (a[i] < a[i-1])
                return false;
        return true;
    }

    // Prints the array
    void printArray(int[] arr)
    {
        for (int i=0; i < arr.length; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
}
```

```
public static void main(String[] args)
{
    //Enter array to be sorted here
    int[] a = {3, 2, 5, 1, 0, 4};
    BogoSort ob = new BogoSort();

    ob.bogoSort(a);

    System.out.print("Sorted array: ");
    ob.printArray(a);
}
```

Python

```
# Python program for implementation of Bogo Sort
import random

# Sorts array a[0..n-1] using Bogo sort
def bogoSort(a):
    n = len(a)
    while (is_sorted(a)== False):
        shuffle(a)

# To check if array is sorted or not
def is_sorted(a):
    n = len(a)
    for i in range(0, n-1):
        if (a[i] > a[i+1] ):
            return False
    return True

# To generate permutation of the array
def shuffle(a):
    n = len(a)
    for i in range (0,n):
        r = random.randint(0,n-1)
        a[i], a[r] = a[r], a[i]

# Driver code to test above
a = [3, 2, 4, 1, 0, 5]
bogoSort(a)
print("Sorted array :")
for i in range(len(a)):
    print ("%d" %a[i]),
```

Output:

Sorted array :
0 1 2 3 4 5

Time Complexity:

- Worst Case : $O(\infty)$ (since this algorithm has no upper bound)
- Average Case: $O(n*n!)$
- Best Case : $O(n)$ (when array given is already sorted)

Auxiliary Space : $O(1)$

Source

<https://www.geeksforgeeks.org/bogosort-permutation-sort/>

Chapter 16

Bubble Sort

Bubble Sort - GeeksforGeeks

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example:

First Pass:

(5 1 4 2 8) -> (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps since $5 > 1$.

(1 5 4 2 8) -> (1 4 5 2 8), Swap since $5 > 4$

(1 4 5 2 8) -> (1 4 2 5 8), Swap since $5 > 2$

(1 4 2 5 8) -> (1 4 2 5 8), Now, since these elements are already in order ($8 > 5$), algorithm does not swap them.

Second Pass:

(1 4 2 5 8) -> (1 4 2 5 8)

(1 4 2 5 8) -> (1 2 4 5 8), Swap since $4 > 2$

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

Following is the implementations of Bubble Sort.

C/C++

```
// C program for implementation of Bubble sort
#include <stdio.h>
```

```
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++)

        // Last i elements are already in place
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Java

```
// Java program for implementation of Bubble Sort
class BubbleSort
{
    void bubbleSort(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n-1; i++)
```

```
for (int j = 0; j < n-i-1; j++)
    if (arr[j] > arr[j+1])
    {
        // swap temp and arr[i]
        int temp = arr[j];
        arr[j] = arr[j+1];
        arr[j+1] = temp;
    }
}

/* Prints the array */
void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

// Driver method to test above
public static void main(String args[])
{
    BubbleSort ob = new BubbleSort();
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    ob.bubbleSort(arr);
    System.out.println("Sorted array");
    ob.printArray(arr);
}
}

/* This code is contributed by Rajat Mishra */
```

Python

```
# Python program for implementation of Bubble Sort

def bubbleSort(arr):
    n = len(arr)

    # Traverse through all array elements
    for i in range(n):

        # Last i elements are already in place
        for j in range(0, n-i-1):

            # traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j+1] :
```

```
arr[j], arr[j+1] = arr[j+1], arr[j]

# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print ("Sorted array is:")
for i in range(len(arr)):
    print ("%d" %arr[i]),
```

C#

```
// C# program for implementation
// of Bubble Sort
using System;

class GFG
{
    static void bubbleSort(int []arr)
    {
        int n = arr.Length;
        for (int i = 0; i < n - 1; i++)
            for (int j = 0; j < n - i - 1; j++)
                if (arr[j] > arr[j + 1])
                {
                    // swap temp and arr[i]
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
    }

    /* Prints the array */
    static void printArray(int []arr)
    {
        int n = arr.Length;
        for (int i = 0; i < n; ++i)
            Console.Write(arr[i] + " ");
        Console.WriteLine();
    }

    // Driver method
    public static void Main()
    {
        int []arr = {64, 34, 25, 12, 22, 11, 90};
        bubbleSort(arr);
        Console.WriteLine("Sorted array");
```

```
        printArray(arr);
    }

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP program for implementation
// of Bubble Sort

function bubbleSort(&$arr)
{
    $n = sizeof($arr);

    // Traverse through all array elements
    for($i = 0; $i < $n; $i++)
    {
        // Last i elements are already in place
        for ($j = 0; $j < $n - $i - 1; $j++)
        {
            // traverse the array from 0 to n-i-1
            // Swap if the element found is greater
            // than the next element
            if ($arr[$j] > $arr[$j+1])
            {
                $t = $arr[$j];
                $arr[$j] = $arr[$j+1];
                $arr[$j+1] = $t;
            }
        }
    }

    // Driver code to test above
    $arr = array(64, 34, 25, 12, 22, 11, 90);

    $len = sizeof($arr);
    bubbleSort($arr);

    echo "Sorted array : \n";
    for ($i = 0; $i < $len; $i++)
        echo $arr[$i]. " ";
}

// This code is contributed by ChitraNayal.
```

?>

Output:

Sorted array:
11 12 22 25 34 64 90

<!--Illustration :

| i = 0 | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|---|
| | 0 | 5 | 3 | 1 | 9 | 8 | 2 | 4 | 7 |
| | 1 | 3 | 5 | 1 | 9 | 8 | 2 | 4 | 7 |
| | 2 | 3 | 1 | 5 | 9 | 8 | 2 | 4 | 7 |
| | 3 | 3 | 1 | 5 | 9 | 8 | 2 | 4 | 7 |
| | 4 | 3 | 1 | 5 | 8 | 9 | 2 | 4 | 7 |
| | 5 | 3 | 1 | 5 | 8 | 2 | 9 | 4 | 7 |
| | 6 | 3 | 1 | 5 | 8 | 2 | 4 | 9 | 7 |
| i=1 | 0 | 3 | 1 | 5 | 8 | 2 | 4 | 7 | 9 |
| | 1 | 1 | 3 | 5 | 8 | 2 | 4 | 7 | |
| | 2 | 1 | 3 | 5 | 8 | 2 | 4 | 7 | |
| | 3 | 1 | 3 | 5 | 8 | 2 | 4 | 7 | |
| | 4 | 1 | 3 | 5 | 2 | 8 | 4 | 7 | |
| | 5 | 1 | 3 | 5 | 2 | 4 | 8 | 7 | |
| i=2 | 0 | 1 | 3 | 5 | 2 | 4 | 7 | 8 | |
| | 1 | 1 | 3 | 5 | 2 | 4 | 7 | | |
| | 2 | 1 | 3 | 5 | 2 | 4 | 7 | | |
| | 3 | 1 | 3 | 2 | 5 | 4 | 7 | | |
| | 4 | 1 | 3 | 2 | 4 | 5 | 7 | | |
| i=3 | 0 | 1 | 3 | 2 | 4 | 5 | 7 | | |
| | 1 | 1 | 3 | 2 | 4 | 5 | | | |
| | 2 | 1 | 2 | 3 | 4 | 5 | | | |
| | 3 | 1 | 2 | 3 | 4 | 5 | | | |
| i=4 | 0 | 1 | 2 | 3 | 4 | 5 | | | |
| | 1 | 1 | 2 | 3 | 4 | | | | |
| | 2 | 1 | 2 | 3 | 4 | | | | |
| i=5 | 0 | 1 | 2 | 3 | 4 | | | | |
| | 1 | 1 | 2 | 3 | | | | | |
| i=6 | 0 | 1 | 2 | 3 | | | | | |
| | 1 | 2 | | | | | | | |

—>

Optimized Implementation:

The above function always runs $O(n^2)$ time even if the array is sorted. It can be optimized by stopping the algorithm if inner loop didn't cause any swap.

CPP

```
// Optimized implementation of Bubble sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
```

```
int temp = *xp;
*xp = *yp;
*yp = temp;
}

// An optimized version of Bubble Sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    bool swapped;
    for (i = 0; i < n-1; i++)
    {
        swapped = false;
        for (j = 0; j < n-i-1; j++)
        {
            if (arr[j] > arr[j+1])
            {
                swap(&arr[j], &arr[j+1]);
                swapped = true;
            }
        }

        // IF no two elements were swapped by inner loop, then break
        if (swapped == false)
            break;
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Java

```
// Optimized java implementation
// of Bubble sort
import java.io.*;

class GFG
{
    // An optimized version of Bubble Sort
    static void bubbleSort(int arr[], int n)
    {
        int i, j, temp;
        boolean swapped;
        for (i = 0; i < n - 1; i++)
        {
            swapped = false;
            for (j = 0; j < n - i - 1; j++)
            {
                if (arr[j] > arr[j + 1])
                {
                    // swap arr[j] and arr[j+1]
                    temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }
            if (swapped == false)
                break;
        }
    }

    // Function to print an array
    static void printArray(int arr[], int size)
    {
        int i;
        for (i = 0; i < size; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver program
    public static void main(String args[])
    {
        int arr[] = { 64, 34, 25, 12, 22, 11, 90 };
    }
}
```

```
int n = arr.length;
bubbleSort(arr, n);
System.out.println("Sorted array: ");
printArray(arr, n);
}
}

// This code is contributed
// by Nikita Tiwari.
```

Python3

```
# Python3 Optimized implementation
# of Bubble sort

# An optimized version of Bubble Sort
def bubbleSort(arr):
    n = len(arr)

    # Traverse through all array elements
    for i in range(n):
        swapped = False

        # Last i elements are already
        # in place
        for j in range(0, n-i-1):

            # traverse the array from 0 to
            # n-i-1. Swap if the element
            # found is greater than the
            # next element
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True

        # IF no two elements were swapped
        # by inner loop, then break
        if swapped == False:
            break

# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print ("Sorted array :")
for i in range(len(arr)):
```

```
print ("%d" %arr[i],end=" ")
# This code is contributed by Shreyanshi Arun
```

C#

```
// Optimized C# implementation
// of Bubble sort
using System;

class GFG
{
    // An optimized version of Bubble Sort
    static void bubbleSort(int []arr, int n)
    {
        int i, j, temp;
        bool swapped;
        for (i = 0; i < n - 1; i++)
        {
            swapped = false;
            for (j = 0; j < n - i - 1; j++)
            {
                if (arr[j] > arr[j + 1])
                {
                    // swap arr[j] and arr[j+1]
                    temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }
            if (swapped == false)
                break;
        }
    }

    // Function to print an array
    static void printArray(int []arr, int size)
    {
        int i;
        for (i = 0; i < size; i++)
            Console.Write(arr[i] + " ");
        Console.WriteLine();
    }
}
```

```
// Driver method
public static void Main()
{
    int []arr = {64, 34, 25, 12, 22, 11, 90};
    int n = arr.Length;
    bubbleSort(arr,n);
    Console.WriteLine("Sorted array");
    printArray(arr,n);
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP Optimized implementation
// of Bubble sort

// An optimized version of Bubble Sort
function bubbleSort(&$arr)
{
    $n = sizeof($arr);

    // Traverse through all array elements
    for($i = 0; $i < $n; $i++)
    {
        $swapped = False;

        // Last i elements are already
        // in place
        for ($j = 0; $j < $n - $i - 1; $j++)
        {

            // traverse the array from 0 to
            // n-i-1. Swap if the element
            // found is greater than the
            // next element
            if ($arr[$j] > $arr[$j+1])
            {
                $t = $arr[$j];
                $arr[$j] = $arr[$j+1];
                $arr[$j+1] = $t;
                $swapped = True;
            }
        }

        // IF no two elements were swapped
    }
}
```

```
// by inner loop, then break
if ($swapped == False)
    break;
}
}

// Driver code to test above
$arr = array(64, 34, 25, 12, 22, 11, 90);
$len = sizeof($arr);
bubbleSort($arr);

echo "Sorted array : \n";

for($i = 0; $i < $len; $i++)
    echo $arr[$i]. " ";

// This code is contributed by ChitraNayal.
?>
```

Output:

```
Sorted array:
11 12 22 25 34 64 90
```

Worst and Average Case Time Complexity: $O(n^2)$. Worst case occurs when array is reverse sorted.

Best Case Time Complexity: $O(n)$. Best case occurs when array is already sorted.

Auxiliary Space: $O(1)$

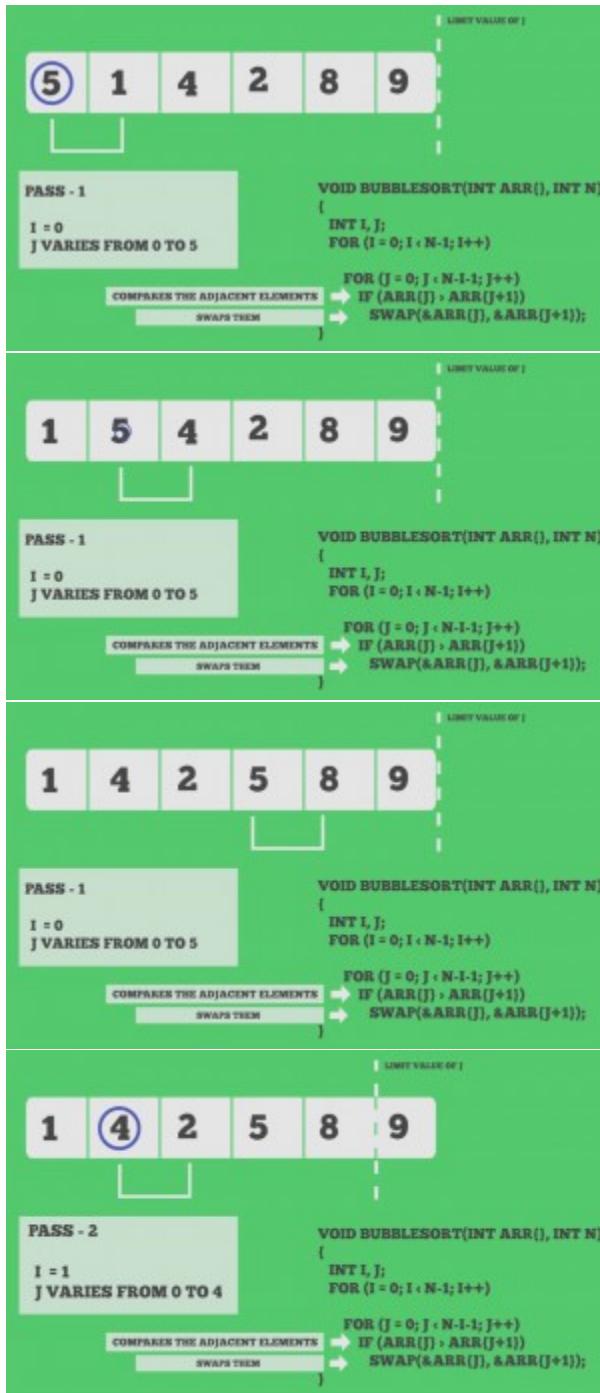
Boundary Cases: Bubble sort takes minimum time (Order of n) when elements are already sorted.

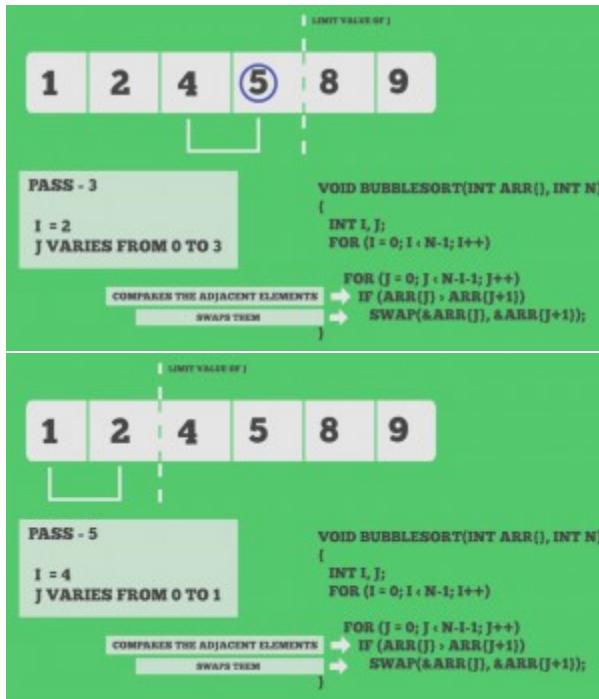
Sorting In Place: Yes

Stable: Yes

Due to its simplicity, bubble sort is often used to introduce the concept of a sorting algorithm. In computer graphics it is popular for its capability to detect a very small error (like swap of just two elements) in almost-sorted arrays and fix it with just linear complexity ($2n$). For example, it is used in a polygon filling algorithm, where bounding lines are sorted by their x coordinate at a specific scan line (a line parallel to x axis) and with incrementing y their order changes (two elements are swapped) only at intersections of two lines (Source: [Wikipedia](#))

Snapshots:





Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

- Selection Sort
- Insertion Sort
- Merge Sort
- Heap Sort
- QuickSort
- Radix Sort
- Counting Sort
- Bucket Sort
- ShellSort

Recursive Bubble Sort

Reference:

- Wikipedia – Bubble Sort
- Image Source

Source

<https://www.geeksforgeeks.org/bubble-sort/>

Chapter 17

Bubble Sort On Doubly Linked List

Bubble Sort On Doubly Linked List - GeeksforGeeks

Sort the given doubly linked list using [bubble sort](#).

Examples:

Input : 5 4 3 2 1
Output : 1 2 3 4 5

Input : 2 1 3 5 4
Output : 1 2 3 4 5

Explanation

As we do in the bubble sort, here also we check elements of two adjacent node whether they are in ascending order or not, if not then we swap the element. We do this until every element get its original position.

In 1st pass the largest element get its original position and in 2nd pass 2nd largest element get its original position and in 3rd pass 3rd largest element get its original position and so on.

And finally whole list get sorted.

Note: If the list is already sorted then it will do only one pass.

```
// CPP program to sort a doubly linked list using
// bubble sort
#include <iostream>
using namespace std;
```

```
// structure of a node
struct Node {
    int data;
    Node* prev;
    Node* next;
};

/* Function to insert a node at the beginning of a linked list */
void insertAtTheBegin(struct Node **start_ref, int data)
{
    struct Node *ptr1 = new Node;
    ptr1->data = data;
    ptr1->next = *start_ref;
    if (*start_ref != NULL)
        (*start_ref)->prev = ptr1;
    *start_ref = ptr1;
}

/* Function to print nodes in a given linked list */
void printList(struct Node *start)
{
    struct Node *temp = start;
    cout << endl;
    while (temp!=NULL)
    {
        cout << temp->data << " ";
        temp = temp->next;
    }
}

/* Bubble sort the given linked list */
void bubbleSort(struct Node *start)
{
    int swapped, i;
    struct Node *ptr1;
    struct Node *lptr = NULL;

    /* Checking for empty list */
    if (start == NULL)
        return;

    do
    {
        swapped = 0;
        ptr1 = start;

        while (ptr1->next != lptr)
        {
```

```
        if (ptr1->data > ptr1->next->data)
        {
            swap(ptr1->data, ptr1->next->data);
            swapped = 1;
        }
        ptr1 = ptr1->next;
    }
    lptr = ptr1;
}
while (swapped);
}

int main()
{
    int arr[] = {12, 56, 2, 11, 1, 90};
    int list_size, i;

    /* start with empty linked list */
    struct Node *start = NULL;

    /* Create linked list from the array arr[] .
     * Created linked list will be 1->11->2->56->12 */
    for (i = 0; i< 6; i++)
        insertAtTheBegin(&start, arr[i]);

    /* print list before sorting */
    printf("\n Linked list before sorting ");
    printList(start);

    /* sort the linked list */
    bubbleSort(start);

    /* print list after sorting */
    printf("\n Linked list after sorting ");
    printList(start);

    return 0;
}
```

Output:

```
Linked list before sorting
90 1 11 2 56 12
Linked list after sorting
1 2 11 12 56 90
```

Source

<https://www.geeksforgeeks.org/bubble-sort-on-doubly-linked-list/>

Chapter 18

Bubble sort using two Stacks

Bubble sort using two Stacks - GeeksforGeeks

Prerequisite : [Bubble Sort](#)

Write a function that sort an array of integers using stacks and also uses bubble sort paradigm.

Algorithm:

1. Push all elements of array in 1st stack
 2. Run a loop for 'n' times(n is size of array)
having the following :
 - 2.a. Keep on pushing elements in the 2nd stack till the top of second stack is smaller than element being pushed from 1st stack.
 - 2.b. If the element being pushed is smaller than top of 2nd stack then swap them (as in bubble sort)
- *Do above steps alternatively

TRICKY STEP: Once a stack is empty, then the top of the next stack will be the largest number so keep it at its position in array i.e arr[len-1-i] and then pop it from that stack.

```
// Java program for bubble sort  
// using stack  
  
import java.util.Arrays;
```

```
import java.util.Stack;

public class Test
{
    // Method for bubble sort using Stack
    static void bubbleSortStack(int arr[], int n)
    {
        Stack<Integer> s1 = new Stack<>();

        // Push all elements of array in 1st stack
        for (int num : arr)
            s1.push(num);

        Stack<Integer> s2 = new Stack<>();

        for (int i = 0; i < n; i++)
        {
            // alternatively
            if (i % 2 == 0)
            {
                while (!s1.isEmpty())
                {
                    int t = s1.pop();

                    if (s2.isEmpty())
                        s2.push(t);
                    else
                    {
                        if (s2.peek() > t)
                        {
                            // swapping
                            int temp = s2.pop();
                            s2.push(t);
                            s2.push(temp);
                        }
                        else
                        {
                            s2.push(t);
                        }
                    }
                }

                // tricky step
                arr[n-1-i] = s2.pop();
            }
            else
            {
                while(!s2.isEmpty())
```

```
{  
    int t = s2.pop();  
  
    if (s1.isEmpty())  
        s1.push(t);  
  
    else  
    {  
        if (s1.peek() > t)  
        {  
            // swapping  
            int temp = s1.pop();  
  
            s1.push(t);  
            s1.push(temp);  
        }  
        else  
            s1.push(t);  
    }  
}  
  
// tricky step  
arr[n-1-i] = s1.pop();  
}  
}  
System.out.println(Arrays.toString(arr));  
}  
  
// Driver Method  
public static void main(String[] args)  
{  
    int arr[] = {15, 12, 44, 2, 5, 10};  
    bubbleSortStack(arr, arr.length);  
}  
}
```

Output:

[2, 5, 10, 12, 15, 44]

Source

<https://www.geeksforgeeks.org/bubble-sort-using-two-stacks/>

Chapter 19

Bucket Sort

Bucket Sort - GeeksforGeeks

Bucket sort is mainly useful when input is uniformly distributed over a range. For example, consider the following problem.

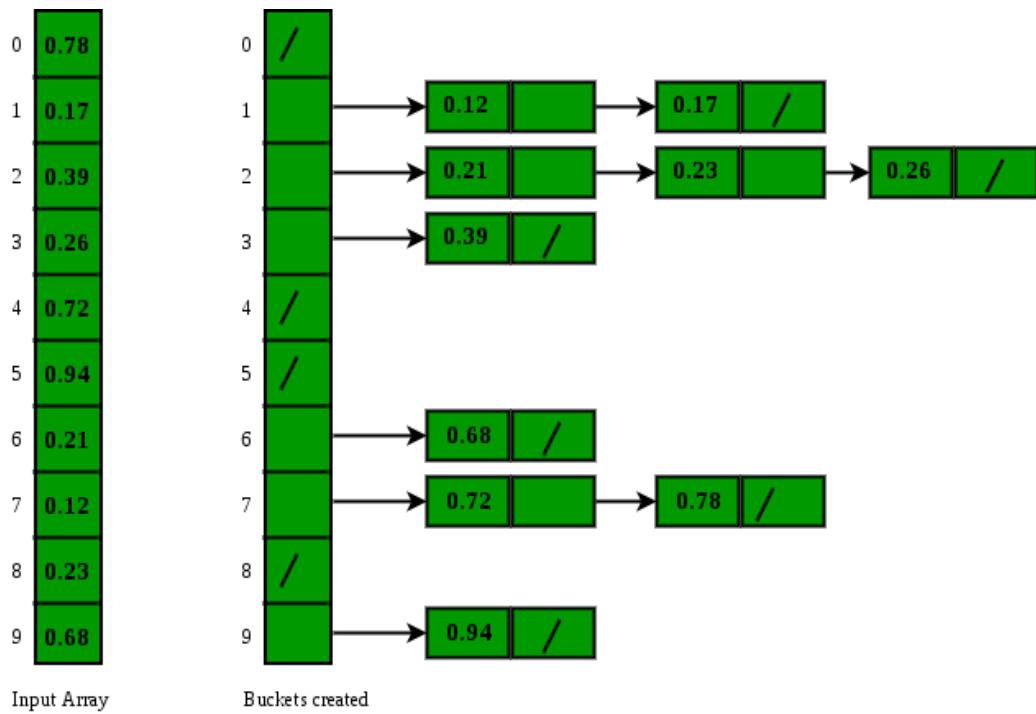
Sort a large set of floating point numbers which are in range from 0.0 to 1.0 and are uniformly distributed across the range. How do we sort the numbers efficiently?

A simple way is to apply a comparison based sorting algorithm. The [lower bound for Comparison based sorting algorithm](#) (Merge Sort, Heap Sort, Quick-Sort .. etc) is $\Omega(n \log n)$, i.e., they cannot do better than $n \log n$.

Can we sort the array in linear time? [Counting sort](#) can not be applied here as we use keys as index in counting sort. Here keys are floating point numbers.

The idea is to use bucket sort. Following is bucket algorithm.

```
bucketSort(arr[], n)
1) Create n empty buckets (Or lists).
2) Do following for every array element arr[i].
.....a) Insert arr[i] into bucket[n*array[i]]
3) Sort individual buckets using insertion sort.
4) Concatenate all sorted buckets.
```



Time Complexity: If we assume that insertion in a bucket takes $O(1)$ time then steps 1 and 2 of the above algorithm clearly take $O(n)$ time. The $O(1)$ is easily possible if we use a linked list to represent a bucket (In the following code, C++ vector is used for simplicity). Step 4 also takes $O(n)$ time as there will be n items in all buckets.

The main step to analyze is step 3. This step also takes $O(n)$ time on average if all numbers are uniformly distributed (please refer [CLRS book](#) for more details)

Following is C++ implementation of the above algorithm.

```
// C++ program to sort an array using bucket sort
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

// Function to sort arr[] of size n using bucket sort
void bucketSort(float arr[], int n)
{
    // 1) Create n empty buckets
    vector<float> b[n];

    // 2) Put array elements in different buckets
    for (int i=0; i<n; i++)
    {
        int bi = n*arr[i]; // Index in bucket
        b[bi].push_back(arr[i]);
    }
}
```

```
}

// 3) Sort individual buckets
for (int i=0; i<n; i++)
    sort(b[i].begin(), b[i].end());

// 4) Concatenate all buckets into arr[]
int index = 0;
for (int i = 0; i < n; i++)
    for (int j = 0; j < b[i].size(); j++)
        arr[index++] = b[i][j];
}

/* Driver program to test above function */
int main()
{
    float arr[] = {0.897, 0.565, 0.656, 0.1234, 0.665, 0.3434};
    int n = sizeof(arr)/sizeof(arr[0]);
    bucketSort(arr, n);

    cout << "Sorted array is \n";
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Output:

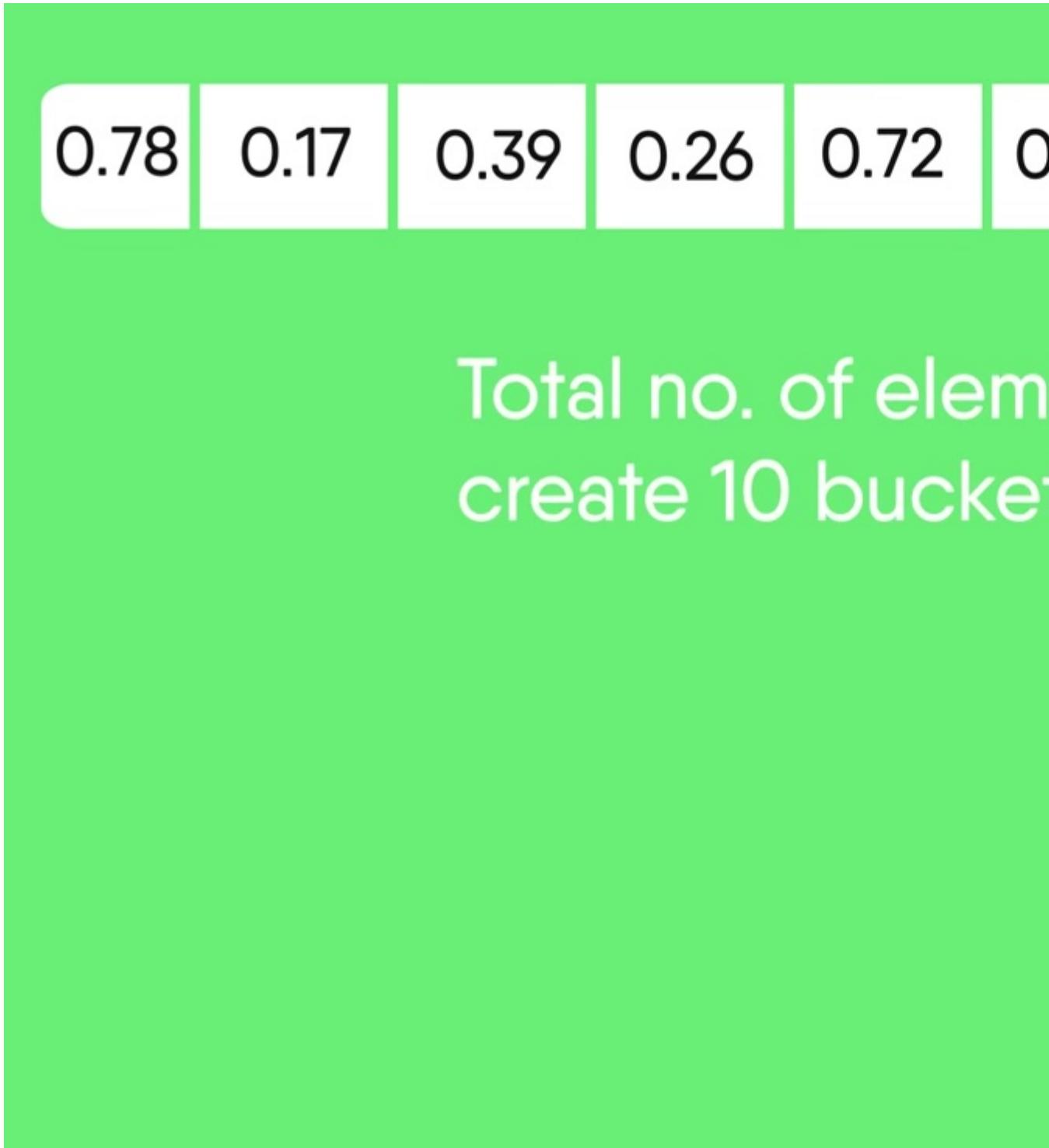
```
Sorted array is
0.1234 0.3434 0.565 0.656 0.665 0.897
```

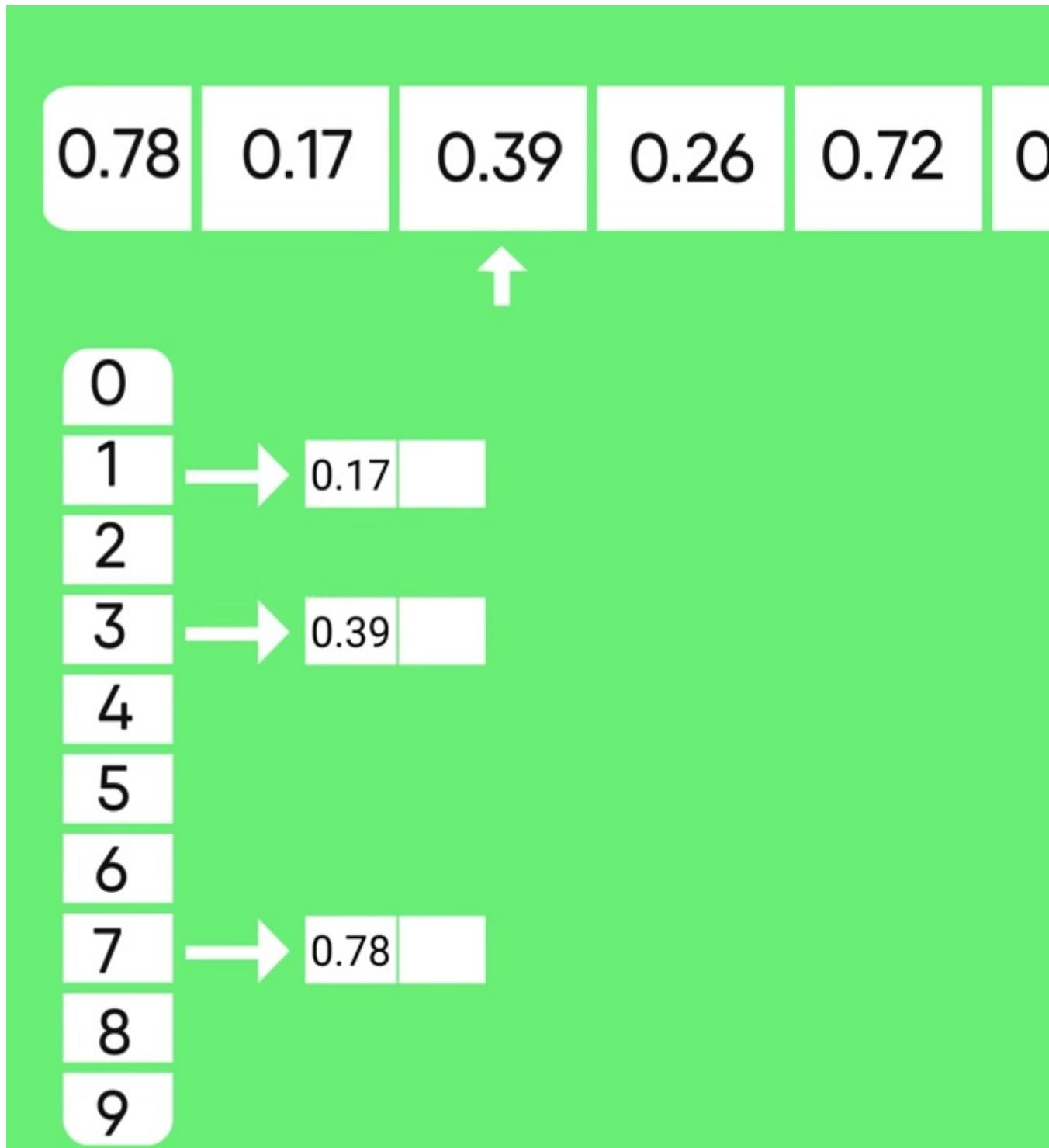
Bucket Sort To Sort an Array with Negative Numbers

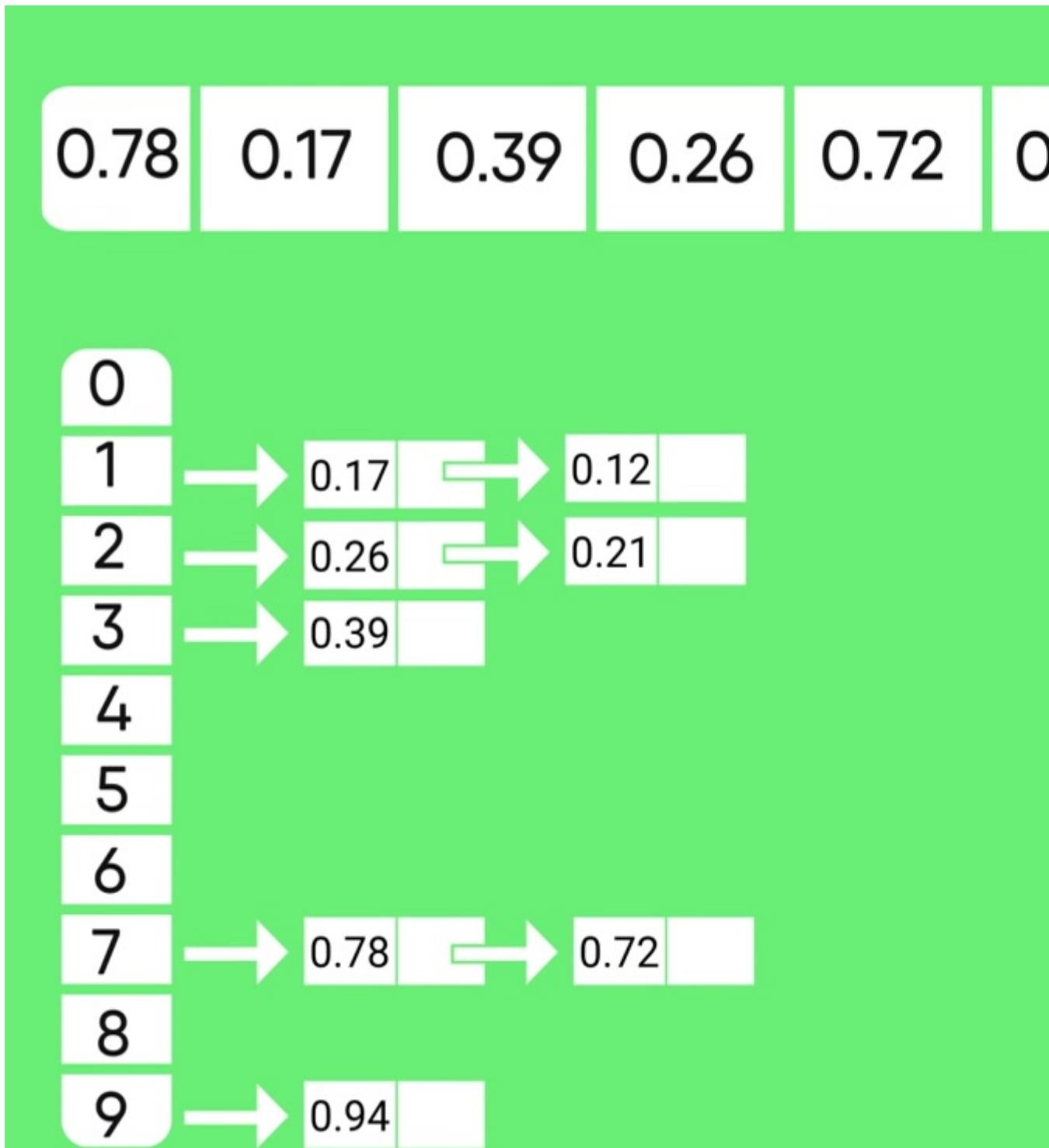
References:

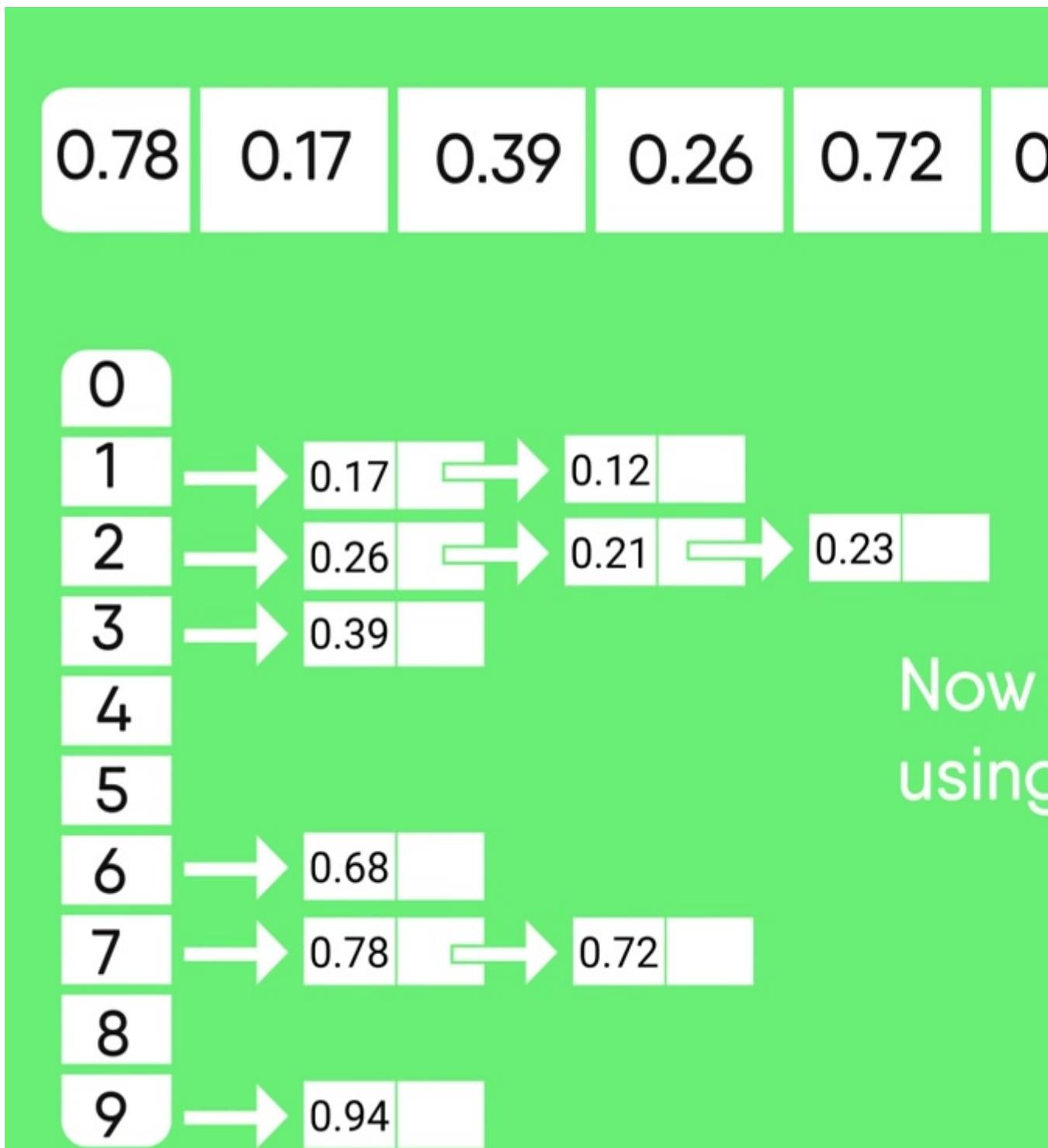
Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest
http://en.wikipedia.org/wiki/Bucket_sort

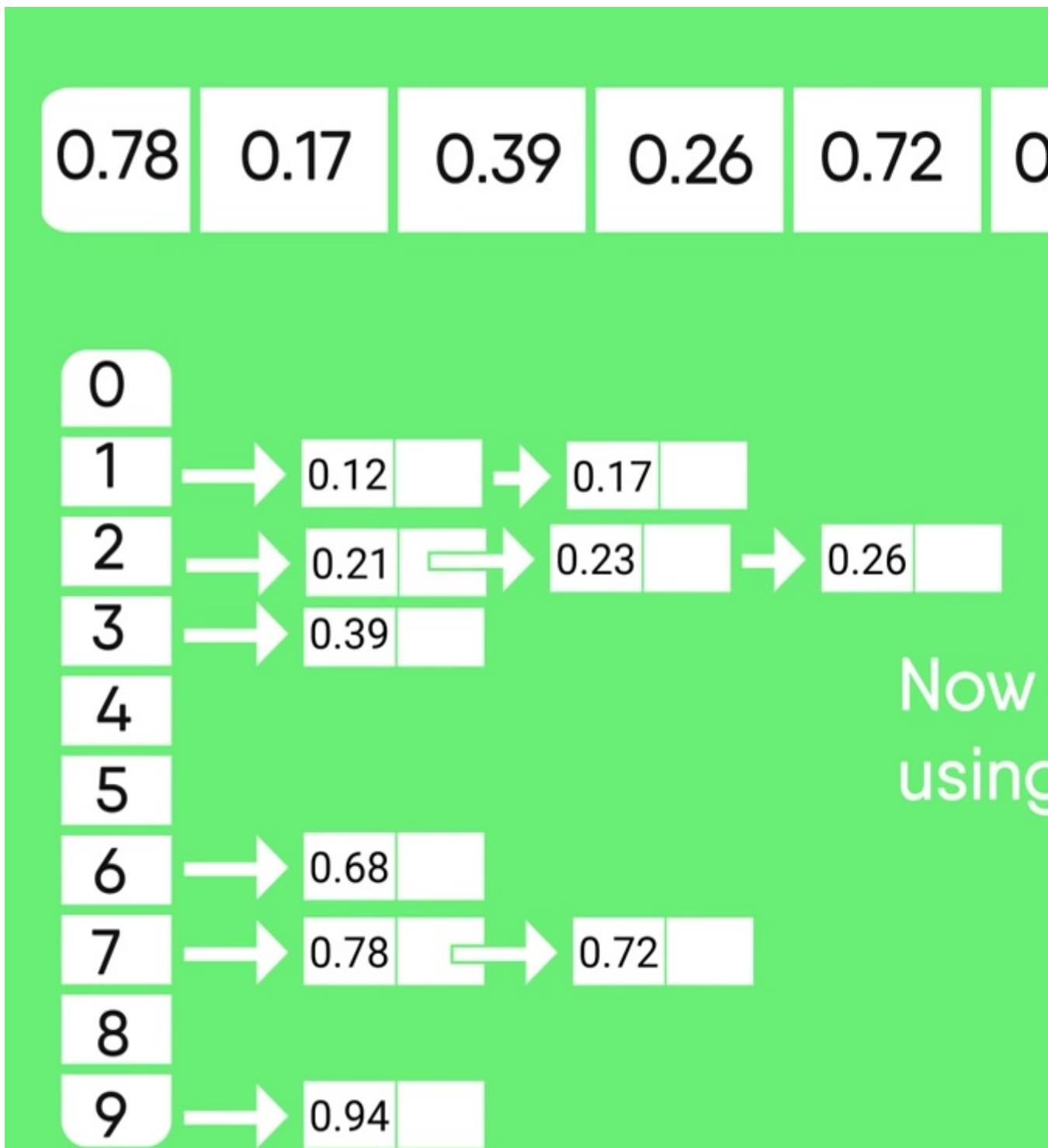
Snapshots:

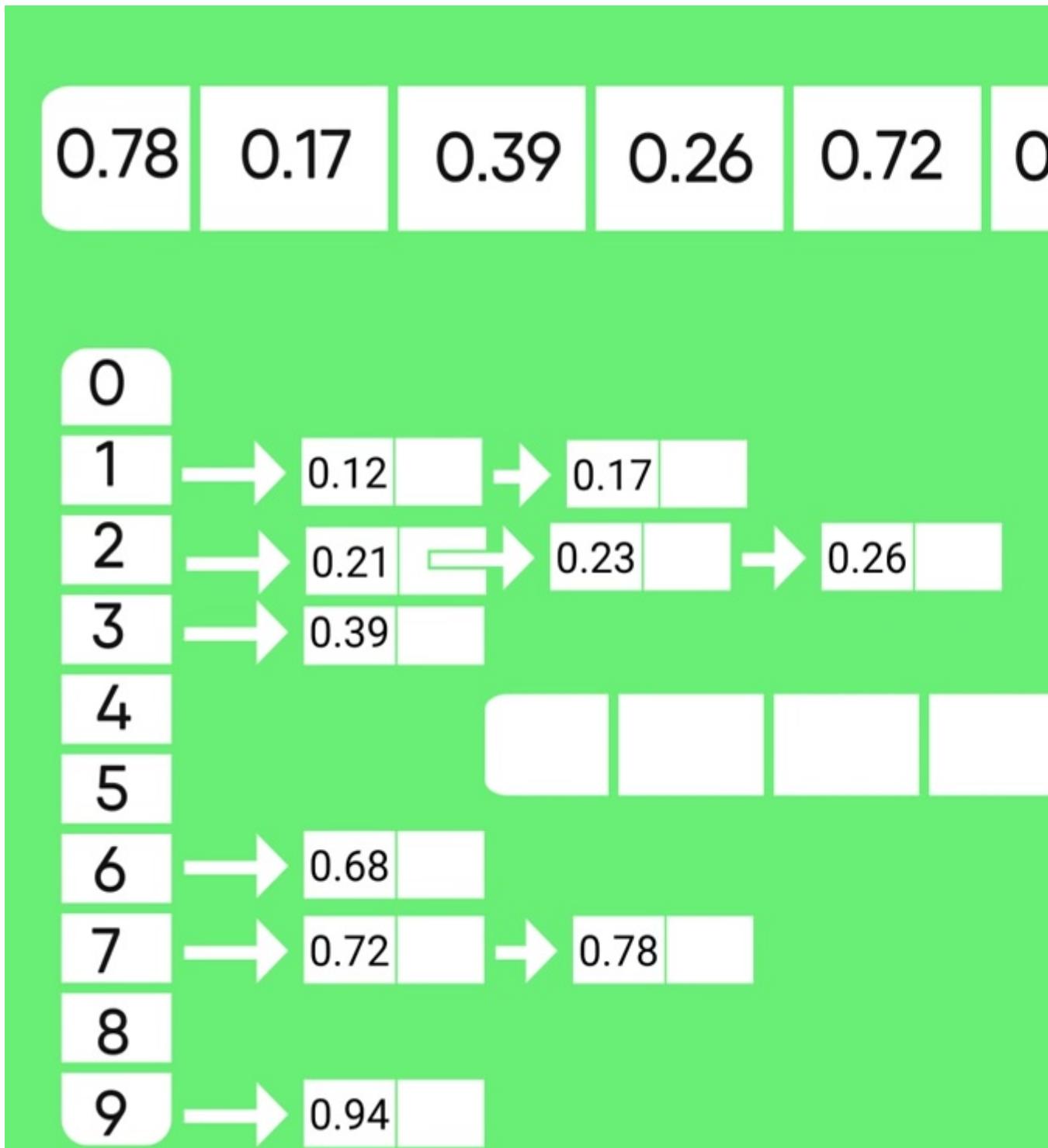


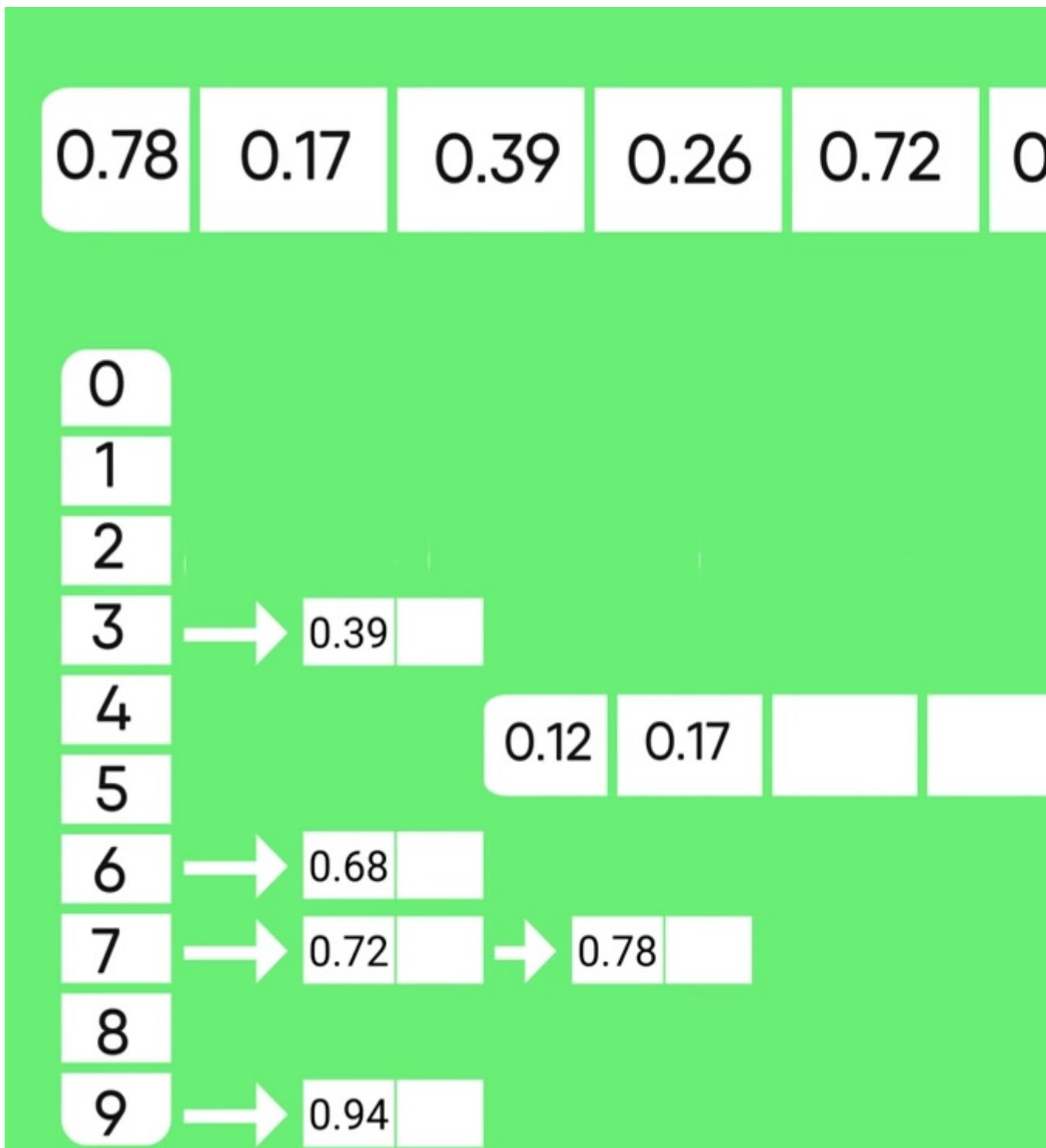


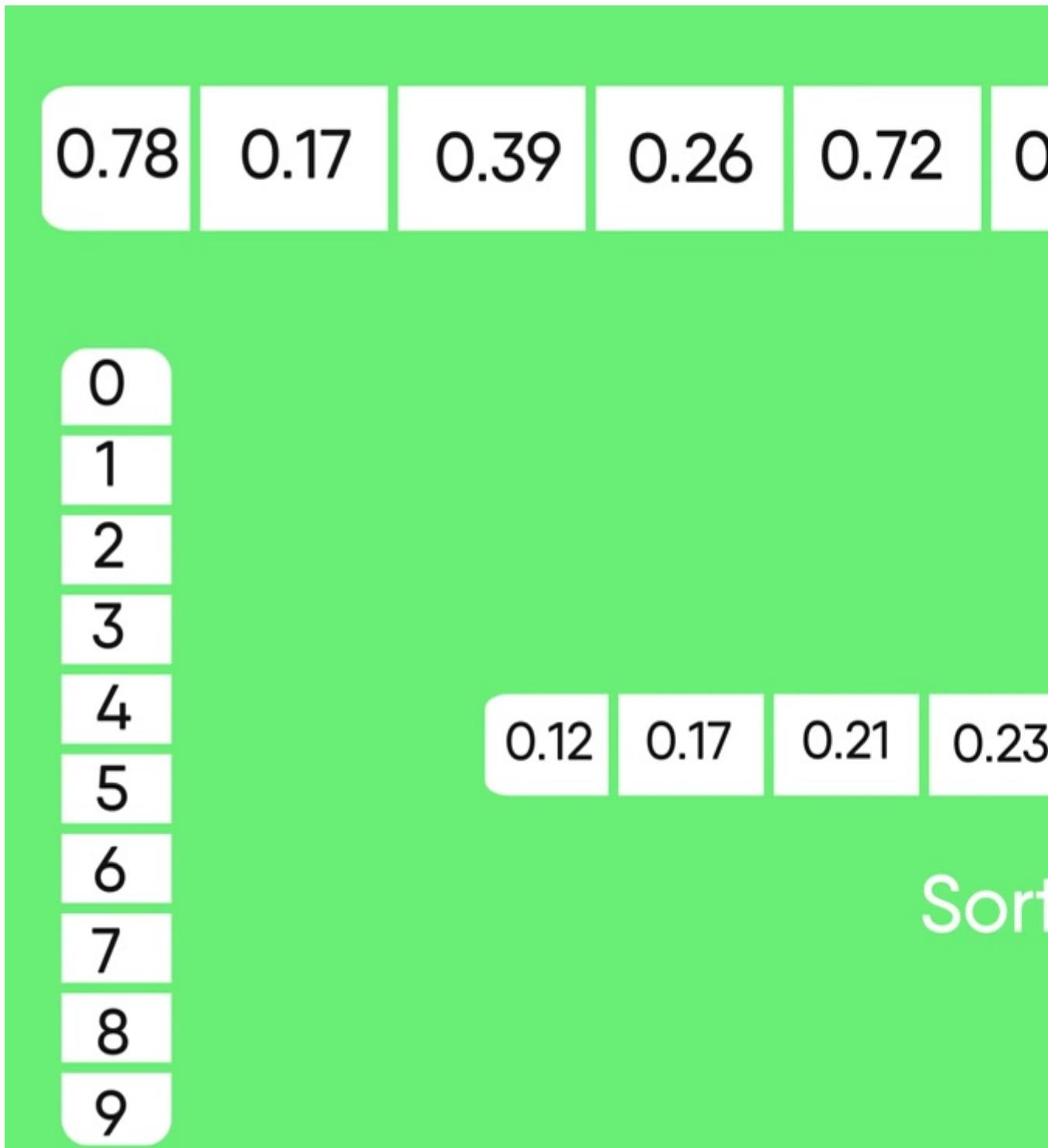












Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

- Selection Sort
- Bubble Sort
- Insertion Sort
- Merge Sort
- Heap Sort
- QuickSort
- Radix Sort
- Counting Sort
- ShellSort

Source

<https://www.geeksforgeeks.org/bucket-sort-2/>

Chapter 20

Bucket Sort To Sort an Array with Negative Numbers

Bucket Sort To Sort an Array with Negative Numbers - GeeksforGeeks

We have discussed bucket sort in the main post on [Bucket Sort](#).

Bucket sort is mainly useful when input is uniformly distributed over a range. For example, consider the problem of sorting a large set of floating point numbers which are in range from 0.0 to 1.0 and are uniformly distributed across the range. In the above post, we have discussed Bucket Sort to sort numbers which are greater than zero.

How to modify Bucket Sort to sort both positive and negative numbers?
Example:

```
Input : arr[] = { -0.897, 0.565, 0.656, -0.1234, 0, 0.3434 }
Output : -0.897 -0.1234 0 0.3434 0.565 0.656
```

Here we considering number is in range -1.0 to 1.0 (floating point number)
Algorithm :

```
sortMixed(arr[], n)
1) Split array into two parts
   create two Empty vector Neg[], Pos[]
   (for negative and positive element respectively)
   Store all negative element in Neg[] by converting
   into positive (Neg[i] = -1 * Arr[i] )
   Store all +ve in pos[] (pos[i] = Arr[i])
2) Call function bucketSortPositive(Pos, pos.size())
   Call function bucketSortPositive(Neg, Neg.size())

bucketSortPositive(arr[], n)
```

- 3) Create n empty buckets (Or lists).
- 4) Do following for every array element arr[i].
 - a) Insert arr[i] into bucket[n*array[i]]
- 5) Sort individual buckets using insertion sort.
- 6) Concatenate all sorted buckets.

Below is c++ implementation of above idea (for floating point number)

```

// C++ program to sort an array of positive
// and negative numbers using bucket sort
#include <bits/stdc++.h>
using namespace std;

// Function to sort arr[] of size n using
// bucket sort
void bucketSort(vector<float> &arr, int n)
{
    // 1) Create n empty buckets
    vector<float> b[n];

    // 2) Put array elements in different
    //     buckets
    for (int i=0; i<n; i++)
    {
        int bi = n*arr[i]; // Index in bucket
        b[bi].push_back(arr[i]);
    }

    // 3) Sort individual buckets
    for (int i=0; i<n; i++)
        sort(b[i].begin(), b[i].end());

    // 4) Concatenate all buckets into arr[]
    int index = 0;
    arr.clear();
    for (int i = 0; i < n; i++)
        for (int j = 0; j < b[i].size(); j++)
            arr.push_back(b[i][j]);
}

// This function mainly splits array into two
// and then calls bucketSort() for two arrays.
void sortMixed(float arr[], int n)
{
    vector<float> Neg ;
    vector<float> Pos;

    // traverse array elements

```

```
for (int i=0; i<n; i++)
{
    if (arr[i] < 0)

        // store -Ve elements by
        // converting into +ve element
        Neg.push_back (-1 * arr[i]) ;
    else
        // store +ve elements
        Pos.push_back (arr[i]) ;
}

bucketSort(Neg, (int)Neg.size());
bucketSort(Pos, (int)Pos.size());

// First store elements of Neg[] array
// by converting into -ve
for (int i=0; i < Neg.size(); i++)
    arr[i] = -1 * Neg[Neg.size() - 1 - i];

// store +ve element
for(int j=Neg.size(); j < n; j++)
    arr[j] = Pos[j - Neg.size()];
}

/* Driver program to test above function */
int main()
{
    float arr[] = {-0.897, 0.565, 0.656,
                   -0.1234, 0, 0.3434};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortMixed(arr, n);

    cout << "Sorted array is \n";
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Output:

```
Sorted array is
-0.897 -0.1234 0 0.3434 0.565 0.656
```

Source

<https://www.geeksforgeeks.org/bucket-sort-to-sort-an-array-with-negative-numbers/>

Chapter 21

C Program for Binary Insertion Sort

C Program for Binary Insertion Sort - GeeksforGeeks

We can use binary search to reduce the number of comparisons in [normal insertion sort](#).
Binary Insertion Sort find use binary search to find the proper location to insert the selected item at each iteration.

In normal insertion, sort it takes $O(i)$ (at ith iteration) in worst case. we can reduce it to $O(\log i)$ by using [binary search](#).

Source

<https://www.geeksforgeeks.org/c-program-for-binary-insertion-sort/>

C/C++

```
// C program for implementation of binary insertion sort
#include <stdio.h>

// A binary search based function to find the position
// where item should be inserted in a[low..high]
int binarySearch(int a[], int item, int low, int high)
{
    if (high <= low)
        return (item > a[low])? (low + 1): low;

    int mid = (low + high)/2;

    if(item == a[mid])
        return mid+1;
```

```
if(item > a[mid])
    return binarySearch(a, item, mid+1, high);
return binarySearch(a, item, low, mid-1);
}

// Function to sort an array a[] of size '\n\
void insertionSort(int a[], int n)
{
    int i, loc, j, k, selected;

    for (i = 1; i < n; ++i)
    {
        j = i - 1;
        selected = a[i];

        // find location where selected could be insereted
        loc = binarySearch(a, selected, 0, j);

        // Move all elements after location to create space
        while (j >= loc)
        {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = selected;
    }
}

// Driver program to test above function
int main()
{
    int a[] = {37, 23, 0, 17, 12, 72, 31,
              46, 100, 88, 54};
    int n = sizeof(a)/sizeof(a[0]), i;

    insertionSort(a, n);

    printf("Sorted array: \n");
    for (i = 0; i < n; i++)
        printf("%d ",a[i]);

    return 0;
}
```

Please refer complete article on [Binary Insertion Sort](#) for more details!

Chapter 22

C Program for Bubble Sort

C Program for Bubble Sort - GeeksforGeeks

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Source

<https://www.geeksforgeeks.org/c-program-for-bubble-sort/>

C/C++

```
// C program for implementation of Bubble sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++)

        // Last i elements are already in place
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
```

```
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Please refer complete article on [Bubble Sort](#) for more details!

Chapter 23

C Program for Bubble Sort on Linked List

C Program for Bubble Sort on Linked List - GeeksforGeeks

Given a singly linked list, sort it using [bubble sort](#).

Input : 10->30->20->5
Output : 5->10->20->30

Input : 20->4->3
Output : 3->4->20

```
// C program to implement Bubble Sort on singly linked list
#include<stdio.h>
#include<stdlib.h>

/* structure for a node */
struct Node
{
    int data;
    struct Node *next;
};

/* Function to insert a node at the beginning of a linked list */
void insertAtTheBegin(struct Node **start_ref, int data);

/* Function to bubble sort the given linked list */
void bubbleSort(struct Node *start);

/* Function to swap data of two nodes a and b*/

```

```
void swap(struct Node *a, struct Node *b);

/* Function to print nodes in a given linked list */
void printList(struct Node *start);

int main()
{
    int arr[] = {12, 56, 2, 11, 1, 90};
    int list_size, i;

    /* start with empty linked list */
    struct Node *start = NULL;

    /* Create linked list from the array arr[].
     * Created linked list will be 1->11->2->56->12 */
    for (i = 0; i < 6; i++)
        insertAtTheBegin(&start, arr[i]);

    /* print list before sorting */
    printf("\n Linked list before sorting ");
    printList(start);

    /* sort the linked list */
    bubbleSort(start);

    /* print list after sorting */
    printf("\n Linked list after sorting ");
    printList(start);

    getchar();
    return 0;
}

/* Function to insert a node at the beginning of a linked list */
void insertAtTheBegin(struct Node **start_ref, int data)
{
    struct Node *ptr1 = (struct Node*)malloc(sizeof(struct Node));
    ptr1->data = data;
    ptr1->next = *start_ref;
    *start_ref = ptr1;
}

/* Function to print nodes in a given linked list */
void printList(struct Node *start)
{
    struct Node *temp = start;
    printf("\n");
```

```
while (temp!=NULL)
{
    printf("%d ", temp->data);
    temp = temp->next;
}
}

/* Bubble sort the given linked list */
void bubbleSort(struct Node *start)
{
    int swapped, i;
    struct Node *ptr1;
    struct Node *lptr = NULL;

    /* Checking for empty list */
    if (start == NULL)
        return;

    do
    {
        swapped = 0;
        ptr1 = start;

        while (ptr1->next != lptr)
        {
            if (ptr1->data > ptr1->next->data)
            {
                swap(ptr1, ptr1->next);
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    }
    while (swapped);
}

/* function to swap data of two nodes a and b*/
void swap(struct Node *a, struct Node *b)
{
    int temp = a->data;
    a->data = b->data;
    b->data = temp;
}
```

Output:

```
Linked list before sorting  
90 1 11 2 56 12  
Linked list after sorting  
1 2 11 12 56 90
```

Improved By : [YugandharTripathi](#)

Source

<https://www.geeksforgeeks.org/c-program-bubble-sort-linked-list/>

Chapter 24

C Program for Recursive Insertion Sort

C Program for Recursive Insertion Sort - GeeksforGeeks

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Below is an iterative algorithm for insertion sort

Algorithm

```
// Sort an arr[] of size n
insertionSort(arr, n)
    Loop from i = 1 to n-1.
        a) Pick element arr[i] and insert
            it into sorted sequence arr[0..i-1]
```

Source

<https://www.geeksforgeeks.org/c-program-for-recursive-insertion-sort/>

C/C++

```
// Recursive C++ program for insertion sort
#include <iostream>
using namespace std;

// Recursive function to sort an array using
// insertion sort
void insertionSortRecursive(int arr[], int n)
{
    // Base case
```

```
if (n <= 1)
    return;

// Sort first n-1 elements
insertionSortRecursive( arr, n-1 );

// Insert last element at its correct position
// in sorted array.
int last = arr[n-1];
int j = n-2;

/* Move elements of arr[0..i-1], that are
   greater than key, to one position ahead
   of their current position */
while (j >= 0 && arr[j] > last)
{
    arr[j+1] = arr[j];
    j--;
}
arr[j+1] = last;
}

// A utility function to print an array of size n
void printArray(int arr[], int n)
{
    for (int i=0; i < n; i++)
        cout << arr[i] << " ";
}

/* Driver program to test insertion sort */
int main()
{
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);

    insertionSortRecursive(arr, n);
    printArray(arr, n);

    return 0;
}
```

Please refer complete article on [Recursive Insertion Sort](#) for more details!

Chapter 25

C Program to Sort an array of names or strings

C Program to Sort an array of names or strings - GeeksforGeeks

Given an array of strings, write a C function to sort them alphabetically.

The idea is to use [qsort\(\)](#) in C and write a comparison function that uses strcmp() to compare two strings.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

static int myCompare (const void * a, const void * b)
{
    return strcmp (*(const char **) a, *(const char **) b);
}

void sort(const char *arr[], int n)
{
    qsort (arr, n, sizeof (const char *), myCompare);
}

int main ()
{
    const char *arr[] = {"GeeksforGeeks", "GeeksQuiz", "CLanguage"};
    int n = sizeof(arr)/sizeof(arr[0]);
    int i;

    printf("Given array is\n");
    for (i = 0; i < n; i++)
        printf("%d: %s \n", i, arr[i]);
```

```
sort(arr, n);

printf("\nSorted array is\n");
for (i = 0; i < n; i++)
    printf("%d: %s \n", i, arr[i]);

return 0;
}
```

Output:

```
Given array is
0: GeeksforGeeks
1: GeeksQuiz
2: CLanguage
```

```
Sorted array is
0: CLanguage
1: GeeksQuiz
2: GeeksforGeekss
```

Source

<https://www.geeksforgeeks.org/c-program-sort-array-names-strings/>

Chapter 26

C program to sort an array of strings using Selection Sort

C program to sort an array of strings using Selection Sort - GeeksforGeeks

Given an array of strings, sort the array using [Selection Sort](#).

Examples:

Input : paper true soap floppy flower
Output : floppy, flower, paper, soap, true

Prerequisite : [Selection Sort](#).

```
// C program to implement selection sort for
// array of strings.
#include <stdio.h>
#include <string.h>
#define MAX_LEN 100

// Sorts an array of strings where length of every
// string should be smaller than MAX_LEN
void selectionSort(char arr[][] [MAX_LEN], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    char minStr[MAX_LEN];
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
```

```
int min_idx = i;
strcpy(minStr, arr[i]);
for (j = i+1; j < n; j++)
{
    // If min is greater than arr[j]
    if (strcmp(minStr, arr[j]) > 0)
    {
        // Make arr[j] as minStr and update min_idx
        strcpy(minStr, arr[j]);
        min_idx = j;
    }
}

// Swap the found minimum element with the first element
if (min_idx != i)
{
    char temp[MAX_LEN];
    strcpy(temp, arr[i]); //swap item[pos] and item[i]
    strcpy(arr[i], arr[min_idx]);
    strcpy(arr[min_idx], temp);
}
}

// Driver code
int main()
{
    char arr[] [MAX_LEN] = {"GeeksforGeeks", "Practice.GeeksforGeeks",
                           "GeeksQuiz"};
    int n = sizeof(arr)/sizeof(arr[0]);
    int i;

    printf("Given array is\n");
    for (i = 0; i < n; i++)
        printf("%d: %s \n", i, arr[i]);

    selectionSort(arr, n);

    printf("\nSorted array is\n");
    for (i = 0; i < n; i++)
        printf("%d: %s \n", i, arr[i]);

    return 0;
}
```

Output :

```
Given array is
0: GeeksforGeeks
1: Practice.GeeksforGeeks
2: GeeksQuiz
```

```
Sorted array is
0: GeeksQuiz
1: GeeksforGeeks
2: Practice.GeeksforGeek
```

Source

<https://www.geeksforgeeks.org/c-program-to-sort-an-array-of-strings-using-selection-sort-2/>

Chapter 27

C `qsort()` vs C++ `sort()`

C `qsort()` vs C++ `sort()` - GeeksforGeeks

Standard C library provides `qsort` function that can be used for sorting an array. Following is the prototype of `qsort()` function.

```
// Sort an array of any type. The parameters are, base
// address of array, size of array and pointer to
// comparator function
void qsort (void* base, size_t num, size_t size,
            int (*comparator)(const void*, const void*));
```

It requires a pointer to the array, the number of elements in the array, the size of each element and a comparator function. We have discussed `qsort` comparator in detail [here](#).

C++ Standard Library provides a similar function `sort()` that originated in the STL. We have discussed C++ `sort` [here](#). Following are prototypes of C++ `sort()` function.

```
// To sort in default or ascending order.
template
void sort(T first, T last);

// To sort according to the order specified
// by comp.
template
void sort(T first, T last, Compare comp);
```

The order of equal elements is not guaranteed to be preserved. C++ provides `std::stable_sort` that can be used to preserve order.

Comparison to `qsort` and `sort()`

1. Implementation details:

As the name suggests, qsort function uses QuickSort algorithm to sort the given array, although the C standard does not require it to implement quicksort.

C++ sort function uses introsort which is a hybrid algorithm. Different implementations use different algorithms. The GNU Standard C++ library, for example, uses a 3-part hybrid sorting algorithm: introsort is performed first (introsort itself being a hybrid of quicksort and heap sort) followed by an insertion sort on the result.

2. Complexity :

The C standard doesn't talk about its complexity of qsort. The new C++11 standard requires that the complexity of sort to be $O(N \log N)$ in the worst case. Previous versions of C++ such as C++03 allow possible worst case scenario of $O(N^2)$. Only average complexity was required to be $O(N \log N)$.

3. Running time:

STL's sort ran faster than C's qsort, because C++'s templates generate optimized code for a particular data type and a particular comparison function.

STL's sort runs 20% to 50% faster than the hand-coded quicksort and 250% to 1000% faster than the C qsort library function. C might be the fastest language but qsort is very slow.

When we tried to sort one million integers on C++14, Time taken by C qsort() was 0.247883 sec and time taken by C++ sort() was only 0.086125 sec

```
// C++ program to demonstrate performance of
// C qsort and C++ sort() algorithm
#include <bits/stdc++.h>
using namespace std;

// Number of elements to be sorted
#define N 1000000

// A comparator function used by qsort
int compare(const void * a, const void * b)
{
    return ( *(int*)a - *(int*)b );
}

// Driver program to test above functions
int main()
{
    int arr[N], dupArr[N];

    // seed for random input
    srand(time(NULL));

    // to measure time taken by qsort and sort
    clock_t begin, end;
    double time_spent;
```

```
// generate random input
for (int i = 0; i < N; i++)
    dupArr[i] = arr[i] = rand()%100000;

begin = clock();
qsort(arr, N, sizeof(int), compare);
end = clock();

// calculate time taken by C qsort function
time_spent = (double)(end - begin) / CLOCKS_PER_SEC;

cout << "Time taken by C qsort() - "
    << time_spent << endl;

time_spent = 0.0;

begin = clock();
sort(dupArr, dupArr + N);
end = clock();

// calculate time taken by C++ sort
time_spent = (double)(end - begin) / CLOCKS_PER_SEC;

cout << "Time taken by C++ sort() - "
    << time_spent << endl;

return 0;
}
```

Output :

```
Time taken by C qsort() - 0.247883
Time taken by C++ sort() - 0.086125
```

C++ sort() is blazingly faster than qsort() on equivalent data due to inlining. sort() on a container of integers will be compiled to use std::less::operator() by default, which will be inlined and sort() will be comparing the integers directly. On the other hand, qsort() will be making an indirect call through a function pointer for every comparison which compilers fails to optimize.

4. Flexibility:

STL's sort works for all data types and for different data containers like C arrays, C++ vectors, C++ deques, etc and other containers that can be written by the user. This kind of flexibility is rather difficult to achieve in C.

5. Safety:

Compared to qsort, the templated sort is more type-safe since it does not require access to data items through unsafe void pointers, as qsort does.

References:

<http://theory.stanford.edu/~amitp/rants/c++-vs-c>
[https://en.wikipedia.org/wiki/Sort_\(C%2B%2B\)](https://en.wikipedia.org/wiki/Sort_(C%2B%2B))

Source

<https://www.geeksforgeeks.org/c-qsort-vs-c-sort/>

Chapter 28

C++ Program for Bubble Sort

C++ Program for Bubble Sort - GeeksforGeeks

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

```
// Optimized implementation of Bubble sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// An optimized version of Bubble Sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    bool swapped;
    for (i = 0; i < n-1; i++)
    {
        swapped = false;
        for (j = 0; j < n-i-1; j++)
        {
            if (arr[j] > arr[j+1])
            {
                swap(&arr[j], &arr[j+1]);
                swapped = true;
            }
        }
    }
}
```

```
// IF no two elements were swapped by inner loop, then break
if (swapped == false)
    break;
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Please refer complete article on [Bubble Sort](#) for more details!

Source

<https://www.geeksforgeeks.org/cpp-program-for-bubble-sort/>

Chapter 29

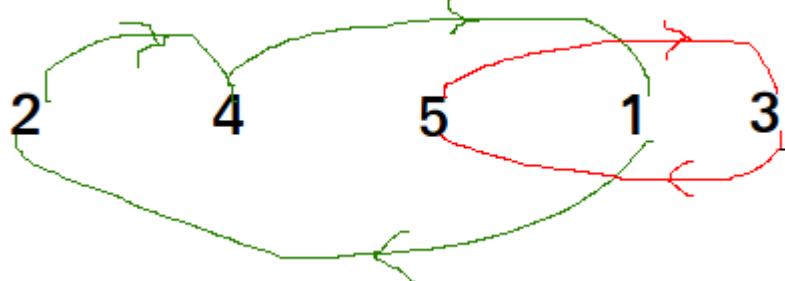
C++ Program for Cycle Sort

C++ Program for Cycle Sort - GeeksforGeeks

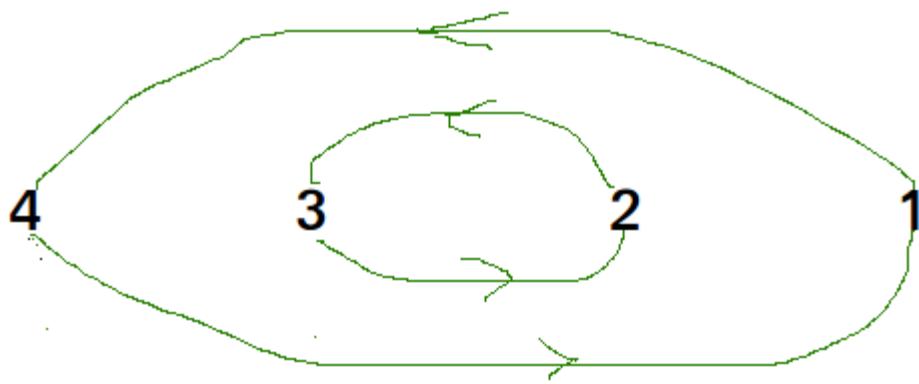
Cycle sort is an in-place sorting Algorithm, [unstable sorting algorithm](#), a comparison sort that is theoretically optimal in terms of the total number of writes to the original array.

- It is optimal in terms of number of memory writes. It [minimizes the number of memory writes](#) to sort (Each value is either written zero times, if it's already in its correct position, or written one time to its correct position.)
- It is based on the idea that array to be sorted can be divided into cycles. Cycles can be visualized as a graph. We have n nodes and an edge directed from node i to node j if the element at i-th index must be present at j-th index in the sorted array.

Cycle in arr[] = {4, 5, 2, 1, 5}



Cycle in arr[] = {4, 3, 2, 1}



We one by one consider all cycles. We first consider the cycle that includes first element. We find correct position of first element, place it at its correct position, say j. We consider old value of arr[j] and find its correct position, we keep doing this till all elements of current cycle are placed at correct position, i.e., we don't come back to cycle starting point.

Source

<https://www.geeksforgeeks.org/cpp-program-for-cycle-sort/>

CPP

```
// C++ program to implement cycle sort
#include <iostream>
using namespace std;

// Function sort the array using Cycle sort
void cycleSort (int arr[], int n)
{
    // count number of memory writes
    int writes = 0;

    // traverse array elements and put it to on
    // the right place
    for (int cycle_start=0; cycle_start<=n-2; cycle_start++)
    {
        // initialize item as starting point
        int item = arr[cycle_start];

        // Find position where we put the item. We basically
        // count all smaller elements on right side of item.
```

```
int pos = cycle_start;
for (int i = cycle_start+1; i<n; i++)
    if (arr[i] < item)
        pos++;

// If item is already in correct position
if (pos == cycle_start)
    continue;

// ignore all duplicate elements
while (item == arr[pos])
    pos += 1;

// put the item to it's right position
if (pos != cycle_start)
{
    swap(item, arr[pos]);
    writes++;
}

// Rotate rest of the cycle
while (pos != cycle_start)
{
    pos = cycle_start;

    // Find position where we put the element
    for (int i = cycle_start+1; i<n; i++)
        if (arr[i] < item)
            pos += 1;

    // ignore all duplicate elements
    while (item == arr[pos])
        pos += 1;

    // put the item to it's right position
    if (item != arr[pos])
    {
        swap(item, arr[pos]);
        writes++;
    }
}

// Number of memory writes or swaps
// cout << writes << endl ;
}

// Driver program to test above function
```

```
int main()
{
    int arr[] = {1, 8, 3, 9, 10, 10, 2, 4 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cycleSort(arr, n) ;

    cout << "After sort : " << endl;
    for (int i =0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Output:

```
After sort :
1 2 3 4 8 9 10 10
```

Please refer complete article on [Cycle Sort](#) for more details!

Chapter 30

C++ Program for QuickSort

C++ Program for QuickSort - GeeksforGeeks

Like [Merge Sort](#), QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Pseudo Code for recursive QuickSort function :

```
/* low --> Starting index, high --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1); // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

Source

<https://www.geeksforgeeks.org/cpp-program-for-quicksort/>

C/C++

```
/* C implementation QuickSort */
#include<stdio.h>

// A utility function to swap two elements
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function takes last element as pivot, places
   the pivot element at its correct position in sorted
   array, and places all smaller (smaller than pivot)
   to left of pivot and all greater elements to right
   of pivot */
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];      // pivot
    int i = (low - 1); // Index of smaller element

    for (int j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;      // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
```

```
{  
    /* pi is partitioning index, arr[p] is now  
       at right place */  
    int pi = partition(arr, low, high);  
  
    // Separately sort elements before  
    // partition and after partition  
    quickSort(arr, low, pi - 1);  
    quickSort(arr, pi + 1, high);  
}  
}  
  
/* Function to print an array */  
void printArray(int arr[], int size)  
{  
    int i;  
    for (i=0; i < size; i++)  
        printf("%d ", arr[i]);  
    printf("\n");  
}  
  
// Driver program to test above functions  
int main()  
{  
    int arr[] = {10, 7, 8, 9, 1, 5};  
    int n = sizeof(arr)/sizeof(arr[0]);  
    quickSort(arr, 0, n-1);  
    printf("Sorted array: \n");  
    printArray(arr, n);  
    return 0;  
}
```

Please refer complete article on [QuickSort](#) for more details!

Chapter 31

C++ Program for Recursive Bubble Sort

C++ Program for Recursive Bubble Sort - GeeksforGeeks

Background :

[Bubble Sort](#) is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Following is iterative Bubble sort algorithm :

```
// Iterative Bubble Sort
bubbleSort(arr[], n)
{
    for (i = 0; i < n-1; i++)
        // Last i elements are already in place
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(arr[j], arr[j+1]);
}
```

Recursion Idea.

1. Base Case: If array size is 1, return.
2. Do One Pass of normal Bubble Sort. This pass fixes last element of current subarray.
3. Recur for all elements except last of current subarray.

Source

<https://www.geeksforgeeks.org/cpp-program-for-recursive-bubble-sort/>

C/C++

```
// C/C++ program for recursive implementation
// of Bubble sort
#include <bits/stdc++.h>
using namespace std;

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    // Base case
    if (n == 1)
        return;

    // One pass of bubble sort. After
    // this pass, the largest element
    // is moved (or bubbled) to end.
    for (int i=0; i<n-1; i++)
        if (arr[i] > arr[i+1])
            swap(arr[i], arr[i+1]);

    // Largest element is fixed,
    // recur for remaining array
    bubbleSort(arr, n-1);
}

/* Function to print an array */
void printArray(int arr[], int n)
{
    for (int i=0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array : \n");
    printArray(arr, n);
    return 0;
}
```

Please refer complete article on [Recursive Bubble Sort](#) for more details!

Chapter 32

C++ Program for Stooge Sort

C++ Program for Stooge Sort - GeeksforGeeks

The Stooge sort is a recursive sorting algorithm. It is defined as below (for ascending order sorting).

- Step 1 : If value at index 0 is greater than value at last index, swap them.
- Step 2: Recursively,
 - a) Stooge sort the initial 2/3rd of the array.
 - b) Stooge sort the last 2/3rd of the array.
 - c) Stooge sort the initial 2/3rd again to confirm.

Source

<https://www.geeksforgeeks.org/cpp-program-for-stooge-sort/>

C++

```
// C++ code to implement stooge sort
#include <iostream>
using namespace std;

// Function to implement stooge sort
void stoogesort(int arr[], int l, int h)
{
    if (l >= h)
        return;

    // If first element is smaller than last,
    // swap them
    if (arr[l] > arr[h])
```

```
swap(arr[l], arr[h]);

// If there are more than 2 elements in
// the array
if(h-l+1>2)
{
    int t = (h-l+1)/3;

    // Recursively sort first 2/3 elements
    stoogesort(arr, l, h-t);

    // Recursively sort last 2/3 elements
    stoogesort(arr, l+t, h);

    // Recursively sort first 2/3 elements
    // again to confirm
    stoogesort(arr, l, h-t);
}

// Driver Code
int main()
{
    int arr[] = {2, 4, 5, 3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Calling Stooge Sort function to sort
    // the array
    stoogesort(arr, 0, n-1);

    // Display the sorted array
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Output:

1 2 3 4 5

Please refer complete article on [Stooge Sort](#) for more details!

Chapter 33

C++ program for Sorting Dates using Selection Sort

C++ program for Sorting Dates using Selection Sort - GeeksforGeeks

```
// C++ program for sorting dates using selection sort
#include<bits/stdc++.h>
using namespace std;
struct date
{
    int day;
    int month;
    int year;
};

int main()
{
    struct date input[5];
    for(int i=0; i<5; i++)
    {
        cin>>input[i].day;
        cin>>input[i].month;
        cin>>input[i].year;
    }
    for (int i=0; i<4; i++)
    {
        for (int j=i+1; j<5; j++)
        {
            if (input[i].year > input[j].year)
            {
                struct date temp = input[i];
```

```
        input[i] = input[j];
        input[j] = temp;
    }
    else if (input[i].year == input[j].year && input[i].month > input[j].month)
    {
        struct date temp = input[i];
        input[i] = input[j];
        input[j] = temp;
    }
    else if (input[i].year == input[j].year && input[i].month == input[j].month && input[i].day > input[j].day)
    {
        struct date temp = input[i];
        input[i] = input[j];
        input[j] = temp;
    }

}
}

for(int i=0; i<5; i++)
{
    cout<<input[i].day<<" "<<input[i].month<<" "<<input[i].year;
    cout<<endl;
}
}
```

This program is contributed by Dinesh T.P.D. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/c-program-for-sorting-dates-using-selection-sort/>

Chapter 34

Can QuickSort be implemented in $O(n\log n)$ worst case time complexity?

Can QuickSort be implemented in $O(n\log n)$ worst case time complexity? - GeeksforGeeks

The worst case time complexity of a typical implementation of [QuickSort](#) is $O(n^2)$. The worst case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted and either first or last element is picked as pivot.

Although randomized QuickSort works well even when the array is sorted, there is still possibility that the randomly picked element is always an extreme. Can the worst case be reduced to $O(n\log n)$?

The answer is yes, we can achieve $O(n\log n)$ worst case. The idea is based on the fact that the [median element of an unsorted array can be found in linear time](#). So we find the median first, then partition the array around the median element.

Following is C++ implementation based on above idea. Most of the functions in below program are copied from [K'th Smallest/Largest Element in Unsorted Array Set 3 \(Worst Case Linear Time\)](#)

```
/* A worst case O(nLogn) implementation of quicksort */
#include<cstring>
#include<iostream>
#include<algorithm>
#include<climits>
using namespace std;

// Following functions are taken from http://goo.gl/ih05BF
int partition(int arr[], int l, int r, int k);
int kthSmallest(int arr[], int l, int r, int k);
```

```
/* A O(nLogn) time complexity function for sorting arr[l..h] */
void quickSort(int arr[], int l, int h)
{
    if (l < h)
    {
        // Find size of current subarray
        int n = h-l+1;

        // Find median of arr[] .
        int med = kthSmallest(arr, l, h, n/2);

        // Partition the array around median
        int p = partition(arr, l, h, med);

        // Recur for left and right of partition
        quickSort(arr, l, p - 1);
        quickSort(arr, p + 1, h);
    }
}

// A simple function to find median of arr[]. This is called
// only for an array of size 5 in this program.
int findMedian(int arr[], int n)
{
    sort(arr, arr+n); // Sort the array
    return arr[n/2]; // Return middle element
}

// Returns k'th smallest element in arr[l..r] in worst case
// linear time. ASSUMPTION: ALL ELEMENTS IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
    // If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1)
    {
        int n = r-l+1; // Number of elements in arr[l..r]

        // Divide arr[] in groups of size 5, calculate median
        // of every group and store it in median[] array.
        int i, median[(n+4)/5]; // There will be floor((n+4)/5) groups;
        for (i=0; i<n/5; i++)
            median[i] = findMedian(arr+l+i*5, 5);
        if (i*5 < n) //For last group with less than 5 elements
        {
            median[i] = findMedian(arr+l+i*5, n%5);
            i++;
        }
    }
}
```

```
// Find median of all medians using recursive call.
// If median[] has only one element, then no need
// of recursive call
int medOfMed = (i == 1)? median[i-1]:
                           kthSmallest(median, 0, i-1, i/2);

// Partition the array around a random element and
// get position of pivot element in sorted array
int pos = partition(arr, l, r, medOfMed);

// If position is same as k
if (pos-1 == k-1)
    return arr[pos];
if (pos-1 > k-1) // If position is more, recur for left
    return kthSmallest(arr, l, pos-1, k);

// Else recur for right subarray
return kthSmallest(arr, pos+1, r, k-pos+l-1);
}

// If k is more than number of elements in array
return INT_MAX;
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// It searches for x in arr[l..r], and partitions the array
// around x.
int partition(int arr[], int l, int r, int x)
{
    // Search for x in arr[l..r] and move it to end
    int i;
    for (i=l; i<r; i++)
        if (arr[i] == x)
            break;
    swap(&arr[i], &arr[r]);

    // Standard partition algorithm
    i = l;
    for (int j = l; j <= r - 1; j++)
    {
        if (arr[j] <= x)
```

```
{  
    swap(&arr[i], &arr[j]);  
    i++;  
}  
}  
swap(&arr[i], &arr[r]);  
return i;  
}  
  
/* Function to print an array */  
void printArray(int arr[], int size)  
{  
    int i;  
    for (i=0; i < size; i++)  
        cout << arr[i] << " ";  
    cout << endl;  
}  
  
// Driver program to test above functions  
int main()  
{  
    int arr[] = {1000, 10, 7, 8, 9, 30, 900, 1, 5, 6, 20};  
    int n = sizeof(arr)/sizeof(arr[0]);  
    quickSort(arr, 0, n-1);  
    cout << "Sorted array is\n";  
    printArray(arr, n);  
    return 0;  
}
```

Output:

```
Sorted array is  
1 5 6 7 8 9 10 20 30 900 1000
```

How is QuickSort implemented in practice – is above approach used?

Although worst case time complexity of the above approach is $O(n\log n)$, it is never used in practical implementations. The hidden constants in this approach are high compared to normal Quicksort. Following are some techniques used in practical implementations of QuickSort.

- 1) Randomly picking up to make worst case less likely to occur (Randomized QuickSort)
- 2) Calling insertion sort for small sized arrays to reduce recursive calls.
- 3) QuickSort is [tail recursive](#), so tail call optimizations is done.

So the approach discussed above is more of a theoretical approach with $O(n\log n)$ worst case time complexity.

This article is compiled by **Shivam**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Chapter 34. Can QuickSort be implemented in $O(n \log n)$ worst case time complexity?

Source

<https://www.geeksforgeeks.org/can-quicksort-implemented-onlogn-worst-case-time-complexity/>

Chapter 35

Cartesian Tree Sorting

Cartesian Tree Sorting - GeeksforGeeks

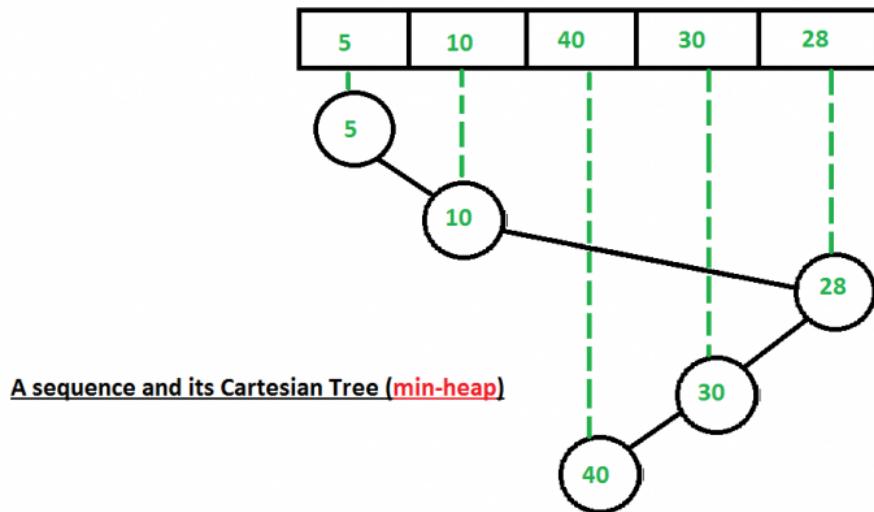
Prerequisites : [Cartesian Tree](#)

Cartesian Sort is an Adaptive Sorting as it sorts the data faster if data is partially sorted. In fact, there are very few sorting algorithms that make use of this fact.

For example consider the array {5, 10, 40, 30, 28}. The input data is partially sorted too as only one swap between “40” and “28” results in a completely sorted order. See how Cartesian Tree Sort will take advantage of this fact below.

Below are steps used for sorting.

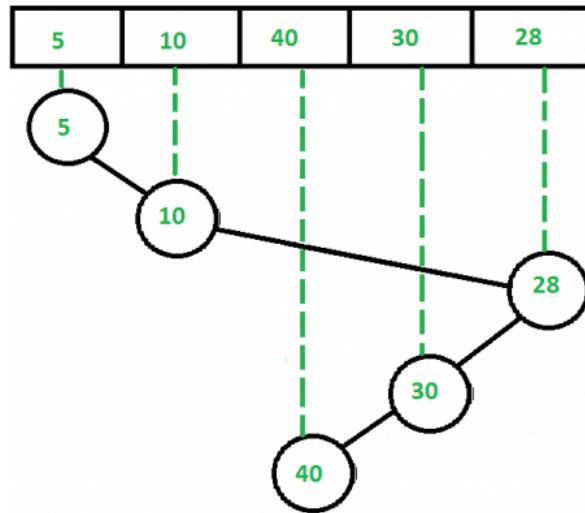
Step 1 : Build a (min-heap) [Cartesian Tree](#) from the given input sequence.



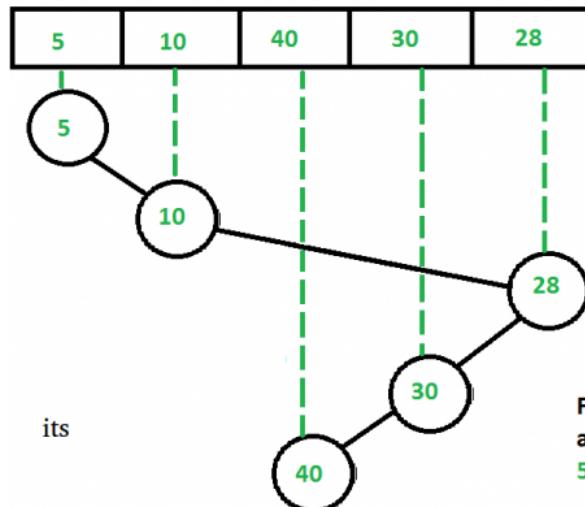
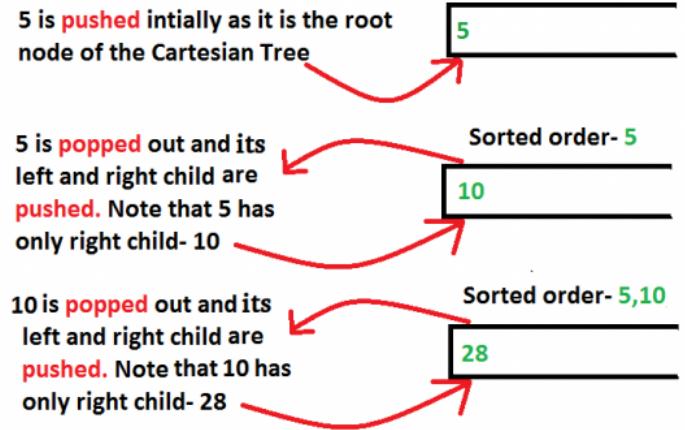
Step 2 : Starting from the root of the built Cartesian Tree, we push the nodes in a priority queue.

Then we pop the node at the top of the priority queue and push the children of the popped node in a pre-order manner.

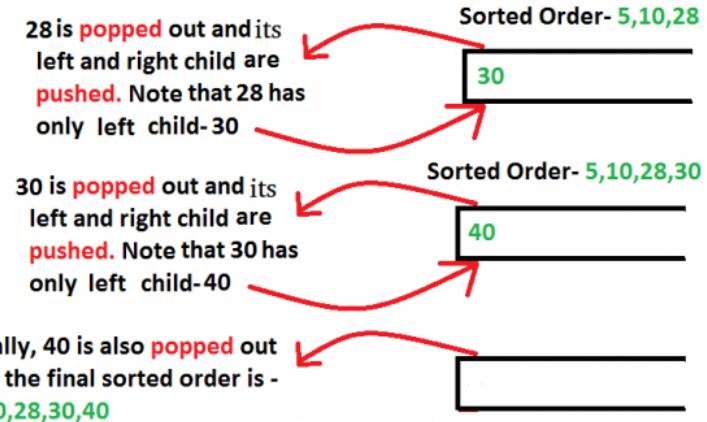
1. Pop the node at the top of the priority queue and add it to a list.
2. Push left child of the popped node first (if present).
3. Push right child of the popped node next (if present).



Inner Operations of Priority Queue



Inner Operations of Priority Queue



How to build (min-heap) Cartesian Tree?

Building min-heap is similar to building a (max-heap) Cartesian Tree (discussed in [previous post](#)), except the fact that now we scan upward from the node's parent up to the root of the tree until a node is found whose value is smaller (and not larger as in the case of a

max-heap Cartesian Tree) than the current one and then accordingly reconfigure links to build the min-heap Cartesian tree.

Why not to use only priority queue?

One might wonder that using priority queue would anyway result in a sorted data if we simply insert the numbers of the input array one by one in the priority queue (i.e- without constructing the Cartesian tree).

But the time taken differs a lot.

Suppose we take the input array – {5, 10, 40, 30, 28}

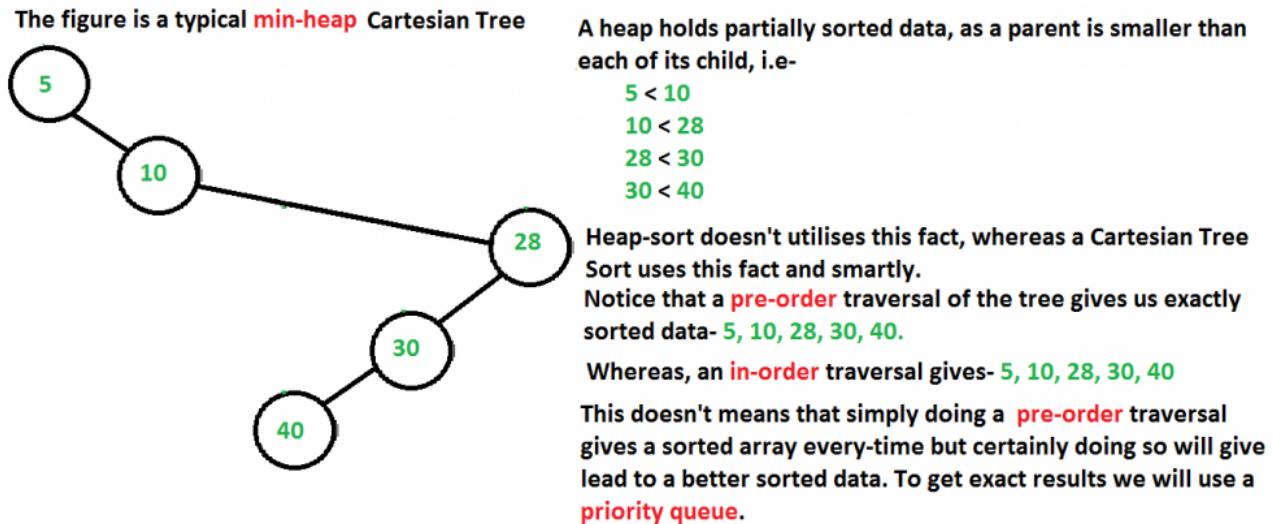
If we simply insert the input array numbers one by one (without using a Cartesian tree), then we may have to waste a lot of operations in adjusting the queue order everytime we insert the numbers (just like a typical heap performs those operations when a new number is inserted, as priority queue is nothing but a heap).

Whereas, here we can see that using a Cartesian tree took only 5 operations (see the above two figures in which we are continuously pushing and popping the nodes of Cartesian tree), which is linear as there are 5 numbers in the input array also. So we see that the best case of Cartesian Tree sort is $O(n)$, a thing where heap-sort will take much more number of operations, because it doesn't make advantage of the fact that the input data is partially sorted.

Why pre-order traversal?

The answer to this is that since Cartesian Tree is basically a heap- data structure and hence follows all the properties of a heap. Thus the root node is always smaller than both of its children. Hence, we use a pre-order fashion popping-and-pushing as in this, the root node is always pushed earlier than its children inside the priority queue and since the root node is always less than both its child, so we don't have to do extra operations inside the priority queue.

Refer to the below figure for better understanding-



```
// A C++ program to implement Cartesian Tree sort
```

```

// Note that in this program we will build a min-heap
// Cartesian Tree and not max-heap.
#include<bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct Node
{
    int data;
    Node *left, *right;
};

// Creating a shortcut for int, Node* pair type
typedef pair<int, Node*> iNPair;

// This function sorts by pushing and popping the
// Cartesian Tree nodes in a pre-order like fashion
void pQBAsedTraversal(Node* root)
{
    // We will use a priority queue to sort the
    // partially-sorted data efficiently.
    // Unlike Heap, Cartesian tree makes use of
    // the fact that the data is partially sorted
    priority_queue <iNPair, vector<iNPair>, greater<iNPair>> pQueue;
    pQueue.push (make_pair (root->data,root));

    // Resembles a pre-order traverse as first data
    // is printed then the left and then right child.
    while (! pQueue.empty())
    {
        iNPair popped_pair = pQueue.top();
        printf("%d ",popped_pair.first);

        pQueue.pop();

        if (popped_pair.second->left != NULL)
            pQueue.push (make_pair(popped_pair.second->left->data,
                                  popped_pair.second->left));

        if (popped_pair.second->right != NULL)
            pQueue.push (make_pair(popped_pair.second->right->data,
                                  popped_pair.second->right));
    }

    return;
}

```

```

Node *buildCartesianTreeUtil(int root, int arr[],
                            int parent[], int leftchild[], int rightchild[])
{
    if (root == -1)
        return NULL;

    Node *temp = new Node;

    temp->data = arr[root];
    temp->left = buildCartesianTreeUtil(leftchild[root],
                                         arr, parent, leftchild, rightchild);

    temp->right = buildCartesianTreeUtil(rightchild[root],
                                         arr, parent, leftchild, rightchild);

    return temp ;
}

// A function to create the Cartesian Tree in O(N) time
Node *buildCartesianTree(int arr[], int n)
{
    // Arrays to hold the index of parent, left-child,
    // right-child of each number in the input array
    int parent[n],leftchild[n],rightchild[n];

    // Initialize all array values as -1
    memset(parent, -1, sizeof(parent));
    memset(leftchild, -1, sizeof(leftchild));
    memset(rightchild, -1, sizeof(rightchild));

    // 'root' and 'last' stores the index of the root and the
    // last processed of the Cartesian Tree.
    // Initially we take root of the Cartesian Tree as the
    // first element of the input array. This can change
    // according to the algorithm
    int root = 0, last;

    // Starting from the second element of the input array
    // to the last on scan across the elements, adding them
    // one at a time.
    for (int i=1; i<=n-1; i++)
    {
        last = i-1;
        rightchild[i] = -1;

        // Scan upward from the node's parent up to
        // the root of the tree until a node is found

```

```

// whose value is smaller than the current one
// This is the same as Step 2 mentioned in the
// algorithm
while (arr[last] >= arr[i] && last != root)
    last = parent[last];

// arr[i] is the smallest element yet; make it
// new root
if (arr[last] >= arr[i])
{
    parent[root] = i;
    leftchild[i] = root;
    root = i;
}

// Just insert it
else if (rightchild[last] == -1)
{
    rightchild[last] = i;
    parent[i] = last;
    leftchild[i] = -1;
}

// Reconfigure links
else
{
    parent[rightchild[last]] = i;
    leftchild[i] = rightchild[last];
    rightchild[last] = i;
    parent[i] = last;
}
}

// Since the root of the Cartesian Tree has no
// parent, so we assign it -1
parent[root] = -1;

return (buildCartesianTreeUtil (root, arr, parent,
                               leftchild, rightchild));
}

// Sorts an input array
int printSortedArr(int arr[], int n)
{
    // Build a cartesian tree
    Node *root = buildCartesianTree(arr, n);
}

```

```

printf("The sorted array is-\n");

// Do pr-order traversal and insert
// in priority queue
pQBasedTraversal(root);
}

/* Driver program to test above functions */
int main()
{
    /* Given input array- {5,10,40,30,28},
       it's corresponding unique Cartesian Tree
       is-

      5
      \
      10
      \
      28
      /
      30
      /
      40
    */
}

int arr[] = {5, 10, 40, 30, 28};
int n = sizeof(arr)/sizeof(arr[0]);

printSortedArr(arr, n);

return(0);
}

```

Output :

```

The sorted array is-
5 10 28 30 40

```

Time Complexity : $O(n)$ best-case behaviour (when the input data is partially sorted),
 $O(n \log n)$ worst-case behavior (when the input data is not partially sorted)

Auxiliary Space : We use a priority queue and a Cartesian tree data structure. Now, at any moment of time the size of the priority queue doesn't exceeds the size of the input array, as we are constantly pushing and popping the nodes. Hence we are using $O(n)$ auxiliary space.

References :

https://en.wikipedia.org/wiki/Adaptive_sort

<http://11011110.livejournal.com/283412.html>
<http://gradbot.blogspot.in/2010/06/cartesian-tree-sort.html>
<http://www.keithschwarz.com/interesting/code/?dir=cartesian-tree-sort>
https://en.wikipedia.org/wiki/Cartesian_tree#Application_in_sorting

Source

<https://www.geeksforgeeks.org/cartesian-tree-sorting/>

Chapter 36

Check if a grid can become row-wise and column-wise sorted after adjacent swaps

Check if a grid can become row-wise and column-wise sorted after adjacent swaps - Geeks-forGeeks

Given a grid of size $n \times len$ filled with lowercase characters. We can swap two adjacent characters in the same row and column. Now we have to check whether it is possible to arrange in such a order that every row and every column in the grid is lexicographically sorted.

Examples:

```
Input : abcde
        fghij
        olmkn
        trpqs
        xywuv
Output : Yes
Explanation :
The grid can be rearranged as
abcde
fghij
klmno
pqrs
uvwxyz
```

The idea to do the above problem is really simple we can simply sort the characters in the same row and then just

check column vise if the new grid is sorted column vise or not. Please note that sorting is possible with adjacent swaps ([Bubble sort](#) for example does only adjacent swaps)

The implementation of the above idea is given below.

C++

```
// C++ program to check if we can make a
// grid of character sorted using adjacent
// swaps.
#include <bits/stdc++.h>
using namespace std;

// v[] is vector of strings. len is length
// of strings in every row.
bool check(vector<string> v, int len)
{
    int n = v.size();
    for (int i = 0; i < n; i++)
        sort(v[i].begin(), v[i].end());

    for (int i = 0; i < len-1; i++)
        for (int j = 0; j < n; j++)
            if (v[i][j] > v[i+1][j])
                return false;
    return true;
}

// Driver code
int main()
{
    vector<string> v = { "ebcda", "ihgfj", "klmno",
                         "pqrst", "yvwvu" };
    int len = 5; // Length of strings
    check(v, len)? cout << "Yes" : cout << "No";
    return 0;
}
```

Python

```
# Python program to check if we can make a
# grid of character sorted using adjacent
# swaps.

# v[] is vector of strings. len is length
# of strings in every row.
def check(v, l):
    n = len(v)
    for i in v:
```

```
i = ''.join(sorted(i))

for i in range(l - 1):
    for j in range(n):
        if (v[i][j] > v[i + 1][j]):
            return False
return True

# Driver code
v = [ "ebcda", "ihgfj", "klmno", "pqrst", "yvwvu" ]
l = 5 # Length of strings
if check(v, l):
    print "Yes"
else:
    print "No"

# This code is contributed by Sachin Bisht
```

Output:

Yes

Source

<https://www.geeksforgeeks.org/check-grid-can-become-row-wise-column-wise-sorted-adjacent-swaps/>

Chapter 37

Check if a queue can be sorted into another queue using a stack

Check if a queue can be sorted into another queue using a stack - GeeksforGeeks

Given a Queue consisting of first **n** natural numbers (in random order). The task is to check whether the given Queue elements can be arranged in increasing order in another Queue using a stack. The operation allowed are:

1. Push and pop elements from the stack
2. Pop (Or enqueue) from the given Queue.
3. Push (Or Dequeue) in the another Queue.

Examples :

Input : Queue[] = { 5, 1, 2, 3, 4 }

Output : Yes

Pop the first element of the given Queue i.e 5.

Push 5 into the stack.

Now, pop all the elements of the given Queue and push them to second Queue.

Now, pop element 5 in the stack and push it to the second Queue.

Input : Queue[] = { 5, 1, 2, 6, 3, 4 }

Output : No

Push 5 to stack.

Pop 1, 2 from given Queue and push it to another Queue.

Pop 6 from given Queue and push to stack.

Pop 3, 4 from given Queue and push to second Queue.

Now, from using any of above operation, we cannot push 5 into the second Queue because it is below the 6 in the stack.

Observe, second Queue (which will contain the sorted element) takes inputs (or enqueue elements) either from given Queue or Stack. So, next expected (which will initially be 1)

element must be present as a front element of given Queue or top element of the Stack. So, simply simulate the process for the second Queue by initializing the expected element as 1. And check if we can get expected element from the front of the given Queue or from the top of the Stack. If we cannot take it from the either of them then pop the front element of given Queue and push it in the Stack.

Also, observe, that the stack must also be sorted at each instance i.e the element at the top of the stack must be smallest in the stack. For eg. let $x > y$, then x will always be expected before y . So, x cannot be pushed before y in the stack. Therefore, we cannot push element with the higher value on the top of the element having lesser value.

Algorithm:

1. Initialize the expected_element = 1
2. Check if either front element of given Queue or top element of the stack have expected_element
 -a) If yes, increment expected_element by 1, repeat step 2.
 -b) Else, pop front of Queue and push it to the stack. If the popped element is greater than top of the Stack, return "No".

Below is the implementation of this approach:

C++

```
// CPP Program to check if a queue of first
// n natural number can be sorted using a stack
#include <bits/stdc++.h>
using namespace std;

// Function to check if given queue element
// can be sorted into another queue using a
// stack.
bool checkSorted(int n, queue<int>& q)
{
    stack<int> st;
    int expected = 1;
    int fnt;

    // while given Queue is not empty.
    while (!q.empty()) {
        fnt = q.front();
        q.pop();

        // if front element is the expected element
        if (fnt == expected)
            expected++;

        else {
            // if stack is empty, push the element
            if (st.empty())
                st.push(fnt);
        }
    }
}
```

```
// if top element is less than element which
// need to be pushed, then return fasle.
else if (!st.empty() && st.top() < fnt) {
    return false;
}

// else push into the stack.
else
    st.push(fnt);
}

// while expected element are coming from
// stack, pop them out.
while (!st.empty() && st.top() == expected) {
    st.pop();
    expected++;
}
}

// if the final expected element value is equal
// to initial Queue size and the stack is empty.
if (expected - 1 == n && st.empty())
    return true;

return false;
}

// Driven Program
int main()
{
    queue<int> q;
    q.push(5);
    q.push(1);
    q.push(2);
    q.push(3);
    q.push(4);

    int n = q.size();

    (checkSorted(n, q) ? (cout << "Yes") :
     (cout << "No"));

    return 0;
}
```

Java

```
// Java Program to check if a queue
// of first n natural number can
// be sorted using a stack
import java.io.*;
import java.util.*;

class GFG
{
    static Queue<Integer> q =
        new LinkedList<Integer>();

    // Function to check if given
    // queue element can be sorted
    // into another queue using a stack.
    static boolean checkSorted(int n)
    {
        Stack<Integer> st =
            new Stack<Integer>();
        int expected = 1;
        int fnt;

        // while given Queue
        // is not empty.
        while (q.size() != 0)
        {
            fnt = q.peek();
            q.poll();

            // if front element is
            // the expected element
            if (fnt == expected)
                expected++;

            else
            {
                // if stack is empty,
                // push the element
                if (st.size() == 0)
                {
                    st.push(fnt);
                }

                // if top element is less than
                // element which need to be
                // pushed, then return fasle.
                else if (st.size() != 0 &&
                         st.peek() < fnt)
                {

```

```
        return false;
    }

    // else push into the stack.
    else
        st.push(fnt);
}

// while expected element are
// coming from stack, pop them out.
while (st.size() != 0 &&
       st.peek() == expected)
{
    st.pop();
    expected++;
}
}

// if the final expected element
// value is equal to initial Queue
// size and the stack is empty.
if (expected - 1 == n &&
    st.size() == 0)
    return true;

return false;
}

// Driver Code
public static void main(String args[])
{
    q.add(5);
    q.add(1);
    q.add(2);
    q.add(3);
    q.add(4);

    int n = q.size();

    if (checkSorted(n))
        System.out.print("Yes");
    else
        System.out.print("No");
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
// C# Program to check if a queue
// of first n natural number can
// be sorted using a stack
using System;
using System.Linq;
using System.Collections.Generic;

class GFG
{
    // Function to check if given
    // queue element can be sorted
    // into another queue using a stack.
    static bool checkSorted(int n,
                           ref Queue<int> q)
    {
        Stack<int> st = new Stack<int>();
        int expected = 1;
        int fnt;

        // while given Queue
        // is not empty.
        while (q.Count != 0)
        {
            fnt = q.Peek();
            q.Dequeue();

            // if front element is
            // the expected element
            if (fnt == expected)
                expected++;

            else
            {
                // if stack is empty,
                // push the element
                if (st.Count != 0)
                {
                    st.Push(fnt);
                }

                // if top element is less than
                // element which need to be
                // pushed, then return fasle.
                else if (st.Count != 0 &&
                         st.Peek() < fnt)
                {

```

```
        return false;
    }

    // else push into the stack.
    else
        st.Push(fnt);
}

// while expected element are
// coming from stack, pop them out.
while (st.Count != 0 &&
       st.Peek() == expected)
{
    st.Pop();
    expected++;
}
}

// if the final expected element
// value is equal to initial Queue
// size and the stack is empty.
if (expected - 1 == n &&
    st.Count == 0)
    return true;

return false;
}

// Driver Code
static void Main()
{
    Queue<int> q = new Queue<int>();
    q.Enqueue(5);
    q.Enqueue(1);
    q.Enqueue(2);
    q.Enqueue(3);
    q.Enqueue(4);

    int n = q.Count;

    if (checkSorted(n, ref q))
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output :

Yes

Video Contributed by [Parul Shandilya](#)

Improved By : [manishshaw1](#), [ParulShandilya](#), [Abdul Mohsin](#)

Source

<https://www.geeksforgeeks.org/check-queue-can-sorted-another-queue-using-stack/>

Chapter 38

Check if any interval completely overlaps the other

Check if any interval completely overlaps the other - GeeksforGeeks

An interval is represented as a combination of start time and end time. Given a set of intervals, we need to write a program to check if any interval **completely** overlaps the other.

Examples:

Input: arr[] = {{1, 3}, {1, 7}, {4, 8}, {2, 5}}

Output: true

The intervals {1, 3} completely overlaps in {1, 7}.

Input: arr[] = {{1, 3}, {7, 9}, {4, 6}, {10, 13}}

Output: false

No pair of intervals overlap.

A **Simple Solution** is to consider every pair of intervals and check if the pair overlaps or not. The time complexity of this solution is $O(n^2)$.

A **better solution** is to use **Sorting**. Following is the complete algorithm.

- 1) Sort all intervals in increasing order of start time. This step takes $O(n \log n)$ time.
- 2) In the sorted array, if the end time of an interval is not more than the end of the previous interval, then there is a complete overlap. This step takes $O(n)$ time.

Given below is an implementation of the above approach:

```
// A C++ program to check if any two intervals
// completely overlap
#include <algorithm>
#include <iostream>
```

```
using namespace std;

// An interval has start time and end time
struct Interval {
    int start;
    int end;
};

// Compares two intervals according to their starting
// time. This is needed for sorting the intervals
// using library function std::sort().
bool compareInterval(Interval i1, Interval i2)
{
    return (i1.start < i2.start) ? true : false;
}

// Function to check if any two intervals
// completely overlap
bool isOverlap(Interval arr[], int n)
{
    // Sort intervals in increasing order of
    // start time
    sort(arr, arr + n - 1, compareInterval);

    // In the sorted array, if end time of an
    // interval is not more than that of
    // end of previous interval, then there
    // is an overlap
    for (int i = 1; i < n; i++)
        if (arr[i].end <= arr[i - 1].end)
            return true;

    // If we reach here, then no overlap
    return false;
}

// Driver code
int main()
{
    // 1st example
    Interval arr1[] = { { 1, 3 }, { 1, 7 }, { 4, 8 },
                        { 2, 5 } };
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    if (isOverlap(arr1, n1))
        cout << "Yes\n";
    else
        cout << "No\n";
}
```

```
// 2nd example
Interval arr2[] = { { 1, 3 }, { 7, 9 }, { 4, 6 },
                     { 10, 13 } };
int n2 = sizeof(arr2) / sizeof(arr2[0]);
if (isOverlap(arr2, n2))
    cout << "Yes\n";
else
    cout << "No\n";

return 0;
}
```

Output:

```
Yes
No
```

Source

<https://www.geeksforgeeks.org/check-interval-completely-overlaps/>

Chapter 39

Check if any two intervals overlap among a given set of intervals

Check if any two intervals overlap among a given set of intervals - GeeksforGeeks

An interval is represented as a combination of start time and end time. Given a set of intervals, check if any two intervals overlap.

Input: arr[] = {{1, 3}, {5, 7}, {2, 4}, {6, 8}}

Output: true

The intervals {1, 3} and {2, 4} overlap

Input: arr[] = {{1, 3}, {7, 9}, {4, 6}, {10, 13}}

Output: false

No pair of intervals overlap.

Expected time complexity is $O(n\log n)$ where n is number of intervals.

We strongly recommend to minimize your browser and try this yourself first.

A **Simple Solution** is to consider every pair of intervals and check if the pair overlaps or not. The time complexity of this solution is $O(n^2)$

Method 1

A better solution is to **Use Sorting**. Following is complete algorithm.

- 1) Sort all intervals in increasing order of start time. This step takes $O(n\log n)$ time.
- 2) In the sorted array, if start time of an interval is less than end of previous interval, then there is an overlap. This step takes $O(n)$ time.

So overall time complexity of the algorithm is $O(n\log n) + O(n)$ which is $O(n\log n)$.

Below is C++ implementation of above idea.

```
// A C++ program to check if any two intervals overlap
#include <algorithm>
#include <iostream>
using namespace std;

// An interval has start time and end time
struct Interval {
    int start;
    int end;
};

// Compares two intervals according to their starting time.
// This is needed for sorting the intervals using library
// function std::sort(). See http:// goo.gl/iGspV
bool compareInterval(Interval i1, Interval i2)
{
    return (i1.start < i2.start) ? true : false;
}

// Function to check if any two intervals overlap
bool isOverlap(Interval arr[], int n)
{
    // Sort intervals in increasing order of start time
    sort(arr, arr + n - 1, compareInterval);

    // In the sorted array, if start time of an interval
    // is less than end of previous interval, then there
    // is an overlap
    for (int i = 1; i < n; i++)
        if (arr[i - 1].end > arr[i].start)
            return true;

    // If we reach here, then no overlap
    return false;
}

// Driver program
int main()
{
    Interval arr1[] = { { 1, 3 }, { 7, 9 }, { 4, 6 }, { 10, 13 } };
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    isOverlap(arr1, n1) ? cout << "Yes\n" : cout << "No\n";

    Interval arr2[] = { { 6, 8 }, { 1, 3 }, { 2, 4 }, { 4, 7 } };
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    isOverlap(arr2, n2) ? cout << "Yes\n" : cout << "No\n";
}
```

```
    return 0;
}
```

Output:

No
Yes

Method 2: This approach is suggested by **Anjali Agarwal**. Following are the steps:

1. Find the overall maximum element. Let it be **max_ele**
2. Initialize an array of size **max_ele** with 0.
3. For every interval [start, end], increment the value at index start, i.e. **arr[start]++** and decrement the value at index (end + 1), i.e. **arr[end + 1]- -**.
4. Compute the prefix sum of this array (**arr[]**).
5. Every index, i of this prefix sum array will tell how many times i has occurred in all the intervals taken together. If this value is greater than 1, then it occurs in 2 or more intervals.
6. So, simply initialize the result variable as false and while traversing the prefix sum array, change the result variable to true whenever the value at that index is greater than 1.

Below is the implementation of this (Method 2) approach.

```
// A C++ program to check if any two intervals overlap
#include <algorithm>
#include <iostream>
using namespace std;

// An interval has start time and end time
struct Interval {
    int start;
    int end;
};

// Function to check if any two intervals overlap
bool isOverlap(Interval arr[], int n)
{

    int max_ele = 0;

    // Find the overall maximum element
    for (int i = 0; i < n; i++) {
        if (max_ele < arr[i].end)
            max_ele = arr[i].end;
```

```
}

// Initialize an array of size max_ele
int aux[max_ele + 1] = { 0 };
for (int i = 0; i < n; i++) {

    // starting point of the interval
    int x = arr[i].start;

    // end point of the interval
    int y = arr[i].end;
    aux[x]++, aux[y + 1]--;
}
for (int i = 1; i <= max_ele; i++) {
    // Calculating the prefix Sum
    aux[i] += aux[i - 1];

    // Overlap
    if (aux[i] > 1)
        return true;
}

// If we reach here, then no Overlap
return false;
}

// Driver program
int main()
{
    Interval arr1[] = { { 1, 3 }, { 7, 9 }, { 4, 6 }, { 10, 13 } };
    int n1 = sizeof(arr1) / sizeof(arr1[0]);

    isOverlap(arr1, n1) ? cout << "Yes\n" : cout << "No\n";

    Interval arr2[] = { { 6, 8 }, { 1, 3 }, { 2, 4 }, { 4, 7 } };
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    isOverlap(arr2, n2) ? cout << "Yes\n" : cout << "No\n";

    return 0;
}
// This Code is written by Anjali Agarwal
```

Output:

```
No
Yes
```

Time Complexity : O(max_ele + n)

Note: This method is more efficient than Method 1 if there are more number of intervals and at the same time maximum value among all intervals should be low, since time complexity is directly proportional to $O(\max_ele)$.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/check-if-any-two-intervals-overlap-among-a-given-set-of-intervals/>

Chapter 40

Check if array contains contiguous integers with duplicates allowed

Check if array contains contiguous integers with duplicates allowed - GeeksforGeeks

Given an array of **n** integers(duplicates allowed). Print “Yes” if it is a set of contiguous integers else print “No”.

Examples:

```
Input : arr[] = {5, 2, 3, 6, 4, 4, 6, 6}
Output : Yes
The elements form a contiguous set of integers
which is {2, 3, 4, 5, 6}.
```

```
Input : arr[] = {10, 14, 10, 12, 12, 13, 15}
Output : No
```

Source: [Amazon interview Experience Set 416](#).

We have discussed different solutions for distinct elements in below post.

[Check if array elements are consecutive](#)

A **simple solution** is to first sort the array. Then traverse the array to check if all consecutive elements differ at most by one.

C

```
// Sorting based C++ implementation
// to check whether the array
// contains a set of contiguous
```

```
// integers
#include <bits/stdc++.h>
using namespace std;

// function to check whether
// the array contains a set
// of contiguous integers
bool areElementsContiguous(int arr[], int n)
{
    // Sort the array
    sort(arr, arr+n);

    // After sorting, check if
    // current element is either
    // same as previous or is
    // one more.
    for (int i = 1; i < n; i++)
        if (arr[i] - arr[i-1] > 1)
            return false;

    return true;
}

// Driver program to test above
int main()
{
    int arr[] = { 5, 2, 3, 6,
                 4, 4, 6, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    if (areElementsContiguous(arr, n))
        cout << "Yes";

    else
        cout << "No";

    return 0;
}
```

Java

```
// Sorting based Java implementation
// to check whether the array
// contains a set of contiguous
// integers
import java.util.*;

class GFG {
```

```
// function to check whether
// the array contains a set
// of contiguous integers
static boolean areElementsContiguous(int arr[],
                                    int n)
{
    // Sort the array
    Arrays.sort(arr);

    // After sorting, check if
    // current element is either
    // same as previous or is
    // one more.
    for (int i = 1; i < n; i++)
        if (arr[i] - arr[i-1] > 1)
            return false;

    return true;
}

/* Driver program to test above function */
public static void main(String[] args)
{
    int arr[] = { 5, 2, 3, 6,
                 4, 4, 6, 6 };
    int n = arr.length;

    if (areElementsContiguous(arr, n))
        System.out.println("Yes");

    else
        System.out.println("No");
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
# Sorting based Python implementation
# to check whether the array
# contains a set of contiguous integers

def areElementsContiguous(arr, n):
    # Sort the array
```

```
arr.sort()

# After sorting, check if
# current element is either
# same as previous or is
# one more.
for i in range(1,n):
    if (arr[i] - arr[i-1] > 1) :
        return 0
return 1

# Driver code
arr = [ 5, 2, 3, 6, 4, 4, 6, 6 ]
n = len(arr)
if areElementsContiguous(arr, n): print("Yes")
else: print("No")

# This code is contributed by 'Ansu Kumari'.
```

C#

```
// Sorting based C# implementation
// to check whether the array
// contains a set of contiguous
// integers
using System;

class GFG {

    // function to check whether
    // the array contains a set
    // of contiguous integers
    static bool areElementsContiguous(int []arr,
                                      int n)
    {
        // Sort the array
        Array.Sort(arr);

        // After sorting, check if
        // current element is either
        // same as previous or is
        // one more.
        for (int i = 1; i < n; i++)
            if (arr[i] - arr[i - 1] > 1)
                return false;

        return true;
    }
}
```

```
// Driver program
public static void Main()
{
    int []arr = { 5, 2, 3, 6,
                  4, 4, 6, 6 };
    int n = arr.Length;

    if (areElementsContiguous(arr, n))
        Console.WriteLine("Yes");

    else
        Console.WriteLine("No");

}

// This code is contributed by Vt_m.
```

Output:

Yes

Time Complexity : $O(n \log n)$

Efficient solution using visited array

- 1) Find minimum and maximum elements.
- 2) Create a visited array of size $\max - \min + 1$. Initialize this array as false.
- 3) Traverse the given array and mark $\text{visited}[arr[i] - \min]$ as true for every element $arr[i]$.
- 4) Traverse visited array and return true if all values are true. Else return false.

C++

```
// C++ implementation to
// check whether the array
// contains a set of
// contiguous integers
#include <bits/stdc++.h>
using namespace std;

// function to check
// whether the array
// contains a set of
// contiguous integers
bool areElementsContiguous(int arr[], int n)
{
```

```
// Find maximum and
// minimum elements.
int max = *max_element(arr, arr + n);
int min = *min_element(arr, arr + n);

int m = max - min + 1;

// There should be at least
// m elements in array to
// make them contiguous.
if (m > n)
    return false;

// Create a visited array
// and initialize false.
bool visited[m];
memset(visited, false, sizeof(visited));

// Mark elements as true.
for (int i=0; i<n; i++)
    visited[arr[i] - min] = true;

// If any element is not
// marked, all elements
// are not contiguous.
for (int i=0; i<m; i++)
    if (visited[i] == false)
        return false;

return true;
}

// Driver program
int main()
{
    int arr[] = { 5, 2, 3, 6,
                  4, 4, 6, 6 };

    int n = sizeof(arr) / sizeof(arr[0]);

    if (areElementsContiguous(arr, n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
// Java implementation to
// check whether the array
// contains a set of
// contiguous integers
import java.util.*;

class GFG {

    // function to check
    // whether the array
    // contains a set of
    // contiguous integers
    static boolean areElementsContiguous(int arr[],
                                         int n)
    {
        // Find maximum and
        // minimum elements.
        int max = Integer.MIN_VALUE;
        int min = Integer.MAX_VALUE;

        for(int i = 0; i < n; i++)
        {
            max = Math.max(max, arr[i]);
            min = Math.min(min, arr[i]);
        }

        int m = max - min + 1;

        // There should be at least
        // m elements in array to
        // make them contiguous.
        if (m > n)
            return false;

        // Create a visited array
        // and initialize false.
        boolean visited[] = new boolean[n];

        // Mark elements as true.
        for (int i = 0; i < n; i++)
            visited[arr[i] - min] = true;

        // If any element is not
        // marked, all elements
        // are not contiguous.
```

```
for (int i = 0; i < m; i++)
    if (visited[i] == false)
        return false;

    return true;
}

/* Driver program */
public static void main(String[] args)
{
    int arr[] = { 5, 2, 3, 6,
                  4, 4, 6, 6 };

    int n = arr.length;

    if (areElementsContiguous(arr, n))
        System.out.println("Yes");

    else
        System.out.println("No");
}
```

Python3

```
# Python3 implementation to
# check whether the array
# contains a set of
# contiguous integers

# function to check
# whether the array
# contains a set of
# contiguous integers
def areElementsContiguous(arr, n):

    # Find maximum and
    # minimum elements.
    max1 = max(arr)
    min1 = min(arr)

    m = max1 - min1 + 1

    # There should be at least
    # m elements in array to
    # make them contiguous.
    if (m > n):
```

```
        return False

    # Create a visited array
    # and initialize fals

    visited = [0] * m

    # Mark elements as true.
    for i in range(0,n) :
        visited[arr[i] - min1] = True

    # If any element is not
    # marked, all elements
    # are not contiguous.
    for i in range(0, m):
        if (visited[i] == False):
            return False

    return True

# Driver program
arr = [5, 2, 3, 6, 4, 4, 6, 6 ]
n = len(arr)

if (areElementsContiguous(arr, n)):
    print("Yes")
else:
    print("No")

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// C# implementation to check whether
// the array contains a set of
// contiguous integers
using System;

class GFG {

    // function to check whether the
    // array contains a set of
    // contiguous integers
    static bool areElementsContiguous(
                    int []arr, int n)
    {

        // Find maximum and
```

```
// minimum elements.
int max = int.MinValue;
int min = int.MaxValue;

for(int i = 0; i < n; i++)
{
    max = Math.Max(max, arr[i]);
    min = Math.Min(min, arr[i]);
}

int m = max - min + 1;

// There should be at least
// m elements in array to
// make them contiguous.
if (m > n)
    return false;

// Create a visited array
// and initialize false.
bool []visited = new bool[n];

// Mark elements as true.
for (int i = 0; i < n; i++)
    visited[arr[i] - min] = true;

// If any element is not
// marked, all elements
// are not contiguous.
for (int i = 0; i < m; i++)
    if (visited[i] == false)
        return false;

return true;
}

/* Driver program */
public static void Main()
{
    int []arr = { 5, 2, 3, 6,
                 4, 4, 6, 6 };

    int n = arr.Length;

    if (areElementsContiguous(arr, n))
        Console.WriteLine("Yes");
    else
```

```
        Console.WriteLine("No");
    }
}

// This code is contributed by nitin mittal.
```

Output:

Yes

Time Complexity : O(n)

Efficient solution using hash table

Insert all the elements in the hash table. Now pick the first element and keep on incrementing in its value by 1 till you find a value not present in the hash table. Again pick the first element and keep on decrementing in its value by 1 till you find a value not present in the hash table. Get the **count** of elements (obtained by this process) which are present in the hash table. If the count equals hash size print “Yes” else “No”.

C++

```
// C++ implementation to check whether the array
// contains a set of contiguous integers
#include <bits/stdc++.h>
using namespace std;

// Function to check whether the array contains
// a set of contiguous integers
bool areElementsContiguous(int arr[], int n)
{
    // Storing elements of 'arr[]' in a hash
    // table 'us'
    unordered_set<int> us;
    for (int i = 0; i < n; i++)
        us.insert(arr[i]);

    // as arr[0] is present in 'us'
    int count = 1;

    // starting with previous smaller element
    // of arr[0]
    int curr_ele = arr[0] - 1;

    // if 'curr_ele' is present in 'us'
    while (us.find(curr_ele) != us.end()) {

        // increment count
        count++;
        curr_ele--;
    }

    // if count is equal to size of array
    // then all elements are contiguous
    if (count == n)
        return true;
    else
        return false;
}
```

```
count++;

// update 'curr_ele'
curr_ele--;
}

// starting with next greater element
// of arr[0]
curr_ele = arr[0] + 1;

// if 'curr_ele' is present in 'us'
while (us.find(curr_ele) != us.end()) {

    // increment count
    count++;

    // update 'curr_ele'
    curr_ele++;
}

// returns true if array contains a set of
// contiguous integers else returns false
return (count == (int)(us.size()));
}

// Driver program to test above
int main()
{
    int arr[] = { 5, 2, 3, 6, 4, 4, 6, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    if (areElementsContiguous(arr, n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
// Java implementation to check whether the array
// contains a set of contiguous integers
import java.io.*;
import java.util.*;

class GFG {
    // Function to check whether the array
    // contains a set of contiguous integers
```

```
static Boolean areElementsContiguous(int arr[], int n)
{
    // Storing elements of 'arr[]' in
    // a hash table 'us'
    HashSet<Integer> us = new HashSet<Integer>();

    for (int i = 0; i < n; i++)
        us.add(arr[i]);

    // As arr[0] is present in 'us'
    int count = 1;

    // Starting with previous smaller
    // element of arr[0]
    int curr_ele = arr[0] - 1;

    // If 'curr_ele' is present in 'us'
    while (us.contains(curr_ele) == true) {

        // increment count
        count++;

        // update 'curr_ele'
        curr_ele--;
    }

    // Starting with next greater
    // element of arr[0]
    curr_ele = arr[0] + 1;

    // If 'curr_ele' is present in 'us'
    while (us.contains(curr_ele) == true) {

        // increment count
        count++;

        // update 'curr_ele'
        curr_ele++;
    }

    // Returns true if array contains a set of
    // contiguous integers else returns false
    return (count == (us.size()));
}

// Driver Code
public static void main(String[] args)
{
```

```
int arr[] = { 5, 2, 3, 6, 4, 4, 6, 6 };
int n = arr.length;

if (areElementsContiguous(arr, n))
    System.out.println("Yes");
else
    System.out.println("No");
}

}

// This code is contributed by 'Gitanjali'.
```

Python

```
# Python implementation to check whether the array
# contains a set of contiguous integers

# Function to check whether the array
# contains a set of contiguous integers
def areElementsContiguous(arr):
    # Storing elements of 'arr[]' in a hash table 'us'
    us = set()
    for i in arr: us.add(i)

    # As arr[0] is present in 'us'
    count = 1

    # Starting with previous smaller element of arr[0]
    curr_ele = arr[0] - 1

    # If 'curr_ele' is present in 'us'
    while curr_ele in us:

        # Increment count
        count += 1

        # Update 'curr_ele'
        curr_ele -= 1

    # Starting with next greater element of arr[0]
    curr_ele = arr[0] + 1

    # If 'curr_ele' is present in 'us'
    while curr_ele in us:

        # Increment count
        count += 1
```

```
# Update 'curr_ele"  
curr_ele += 1  
  
# Returns true if array contains a set of  
# contiguous integers else returns false  
return (count == len(us))  
  
# Driver code  
arr = [ 5, 2, 3, 6, 4, 4, 6, 6 ]  
if areElementsContiguous(arr): print("Yes")  
else: print("No")  
  
# This code is contributed by 'Ansul Kumari'
```

Output :

Yes

Time Complexity: O(n).

Auxiliary Space: O(n).

This method requires only one traversal of given array. It traverses hash table after array traversal (hash table contains only distinct elements).

Improved By : nitin mittal

Source

<https://www.geeksforgeeks.org/check-array-contains-contiguous-integers-duplicates-allowed/>

Chapter 41

Check if both halves of the string have at least one different character

Check if both halves of the string have at least one different character - GeeksforGeeks

Earlier we have discussed [on how to check if both halves of the string have same set of characters](#). Now, we further extend our problem on checking if both halves of the string have at least one different character.

Examples:

```
Input : baaaab
Output: No, both halves do not differ at all
The two halves contain the same characters
and their frequencies match so no different
character exists
```

```
Input : abccpb
Output : Yes, both halves differ by at least one character
```

Method 1: (Two counter arrays)

- Split the string into two halves
- Traverse two different halves separately and count the occurrence of each character into two different counter array
- Now, traverse these arrays and if these arrays differ at a point, we get the answer as “Yes”

C++

```
// C++ implementation to check if
// both halves of the string have
// at least one different character
#include <cstring>
#include <iostream>
#include <string>
using namespace std;
#define MAX 26

// Function which break string into two halves
// Counts frequency of characters in each half
// Compares the two counter array and returns
// true if these counter arrays differ
bool function(string str)
{
    int l = str.length();

    // Declaration and initialization
    // of counter array
    int counter1[MAX];
    int counter2[MAX];
    memset(counter1, 0, sizeof(counter1));
    memset(counter2, 0, sizeof(counter2));

    for (int i = 0; i < l / 2; i++)
        counter1[str[i] - 'a']++;
    for (int i = l / 2; i < l; i++)
        counter2[str[i] - 'a']++;
    for (int i = 0; i < MAX; i++)
        if (counter2[i] != counter1[i])
            return true;

    return false;
}

// Driver function
int main()
{
    string str = "abcasdsabcae";
    if (function(str))
        cout << "Yes, both halves differ"
            << " by at least one character";
    else
        cout << "No, both halves do "
            << "not differ at all";
    return 0;
}
```

}

Java

```
// Java implementaion of the problem
import java.util.*;
import java.lang.*;

class GeeksforGeeks {
    final static int MAX = 26;

    // Function which break string into two halves
    // Counts frequency of characters in each half
    // Compares the two counter array and returns
    // true if these counter arrays differ
    static boolean function(String str)
    {
        int l = str.length();

        // Declaration and initialization
        // of counter array
        int counter1[] = new int[MAX];
        int counter2[] = new int[MAX];
        for (int i = 0; i < MAX; i++) {
            counter1[i] = 0;
            counter2[i] = 0;
        }

        for (int i = 0; i < l / 2; i++)
            counter1[str.charAt(i) - 'a']++;
        for (int i = l / 2; i < l; i++)
            counter2[str.charAt(i) - 'a']++;
        for (int i = 0; i < MAX; i++) {
            if (counter2[i] != counter1[i])
                return true;
        }
        return false;
    }

    // Driver function
    public static void main(String args[])
    {
        String str = "abcasdsabcae";
        if (function(str))
            System.out.print("Yes, both halves "+
                            "differ by at least one character");
        else
            System.out.print("No, both halves "+

```

```
        "do not differ at all");
    }
}
```

Python3

```
# Python implementation to check if
# both halves of the string have
# at least one different character

MAX = 26

# Function which break string into two halves
# Counts frequency of characters in each half
# Compares the two counter array and returns
# true if these counter arrays differ
def function(st):
    global MAX
    l = len(st)

    # Declaration and initialization
    # of counter array
    counter1, counter2 = [0] * MAX, [0] * MAX

    for i in range(l//2):
        counter1[ord(st[i]) - ord('a')] += 1

    for i in range(l//2, l):
        counter2[ord(st[i]) - ord('a')] += 1

    for i in range(MAX):
        if (counter2[i] != counter1[i]):
            return True
    return False

# Driver function
st = "abcasdsabcae"
if function(st): print("Yes, both halves differ ",
                      "by at least one character")
else: print("No, both halves do not differ at all")

# This code is contributed by Ansu Kumari
```

C#

```
// C# implementation to check if
```

```
// both halves of the string have
// at least one different character

using System;

class GeeksforGeeks {
    static int MAX = 26;

    // Function which break string into two halves
    // Counts frequency of characters in each half
    // Compares the two counter array and returns
    // true if these counter arrays differ
    static bool function(String str)
    {
        int l = str.Length;

        // Declaration and initialization
        // of counter array
        int []counter1 = new int[MAX];
        int []counter2 = new int[MAX];
        for (int i = 0; i < MAX; i++)
        {
            counter1[i] = 0;
            counter2[i] = 0;
        }

        for (int i = 0; i < l / 2; i++)
            counter1[str[i] - 'a']++;
        for (int i = l / 2; i < l; i++)
            counter2[str[i] - 'a']++;
        for (int i = 0; i < MAX; i++) {
            if (counter2[i] != counter1[i])
                return true;
        }
        return false;
    }

    // Driver function
    public static void Main()
    {
        String str = "abcasdsabcae";
        if (function(str))
            Console.WriteLine("Yes, both halves "+
                "differ by at least one character");
        else
            Console.WriteLine("No, both halves "+
                "do not differ at all");
    }
}
```

```
    }
}

//This code is contributed by vt_m.
```

Output:

```
Yes, both halves differ by at least one character
```

Method 2:(One counter arrays)

- This method uses only a single array of length 26.
- For the first half, we increment the characters in the counter array of length 26.
- For second array, we decrement the character in that same counter array.
- Now, if for an index corresponding to a character has non-zero value, that is the distinct character present
- The positive ones are ones present in the first half and the negative ones are characters in the second half

C++

```
// C++ implementation to check if
// both halves of the string have
// at least one different character
#include <cstring>
#include <iostream>
#include <string>
using namespace std;
#define MAX 26

// Function which break string into two halves
// Increments frequency of characters for first half
// Decrements frequency of characters for second half
// true if any index has non-zero value
bool function(string str)
{
    int l = str.length();

    // Declaration and initialization
    // of counter array
    int counter[MAX];
    memset(counter, 0, sizeof(counter));
    for (int i = 0; i < l / 2; i++)
```

```
        counter[str[i] - 'a']++;
    for (int i = l / 2; i < l; i++)
        counter[str[i] - 'a']--;
    for (int i = 0; i < MAX; i++)
        if (counter[i] != 0)
            return true;

    return false;
}

// Driver function
int main()
{
    string str = "abcasdsabcae";
    if (function(str))
        cout << "Yes, both halves differ"
            <<" by at least one character";
    else
        cout << "No, both halves do"
            <<" not differ at all";
    return 0;
}
```

Java

```
// Java implementaion of the problem
import java.util.*;
import java.lang.*;

class GeeksforGeeks {

    final static int MAX = 26;

    // Function which break string into two halves
    // Increments frequency of characters for first half
    // Decrements frequency of characters for second half
    // true if any index has non-zero value
    static boolean function(String str)
    {
        int l = str.length();

        // Declaration and initialization
        // of counter array
        int counter[] = new int[MAX];
        for (int i = 0; i < MAX; i++)
            counter[i] = 0;
        for (int i = 0; i < l / 2; i++)
            counter[str.charAt(i) - 'a']++;
```

```
        for (int i = l / 2; i < l; i++)
            counter[str.charAt(i) - 'a']--;
        for (int i = 0; i < MAX; i++)
            if (counter[i] != 0)
                return true;

        return false;
    }

// Driver function
public static void main(String args[])
{
    String str = "abcasdsabcae";
    if (function(str))
        System.out.print("Yes, both halves"
+ " differ by at least one character");
    else
        System.out.print("No, both halves"
+ " do not differ at all");
}
```

Python3

```
# Python3 implementation to check if
# both halves of the string have
# at least one different character
MAX = 26

# Function which break string into two
# halves Increments frequency of characters
# for first half Decrements frequency of
# characters for second half true if any
# index has non-zero value
def function(st):
    global MAX
    l = len(st)

    # Declaration and initialization
    # of counter array
    counter = [0] * MAX

    for i in range(l // 2):
        counter[ord(st[i]) - ord('a')] += 1

    for i in range(l // 2, l):
        counter[ord(st[i]) - ord('a')] -= 1
```

```
for i in range(MAX):
    if (counter[i] != 0):
        return True

    return False

# Driver function
st = "abcasdabcae"
if function(st):
    print("Yes, both halves differ by at ",
          "least one character")
else:
    print("No, both halves do not differ at all")

# This code is contributed by Ansu Kumari
```

C#

```
// C# implementaion of the problem
using System;

class GFG {

    static int MAX = 26;

    // Function which break string into
    // two halves Increments frequency
    // of characters for first half
    // Decrements frequency of characters
    // for second half true if any index
    // has non-zero value
    static bool function(String str)
    {
        int l = str.Length;

        // Declaration and initialization
        // of counter array
        int []counter = new int[MAX];
        for (int i = 0; i < MAX; i++)
            counter[i] = 0;
        for (int i = 0; i < l / 2; i++)
            counter[str[i] - 'a']++;
        for (int i = l / 2; i < l; i++)
            counter[str[i] - 'a']--;
        for (int i = 0; i < MAX; i++)
            if (counter[i] != 0)
                return true;
```

```
        return false;
    }

    // Driver function
    public static void Main()
    {
        string str = "abcasdsabcae";
        if (function(str))
            Console.WriteLine("Yes, both halves"
                + " differ by at least one "
                + "character");
        else
            Console.WriteLine("No, both halves"
                + " do not differ at all");
    }
}

// This code is contributed by anuj_67.
```

The time complexity of both the approaches is linear, i.e. $O(\text{len})$

Method 3: (No extra space)

- Input the string in form of character array
- Sort the two strings separately
- Traverse the two halves together, if these differ at any point, return true to the calling function

Follow the code below.

C++

```
// C++ implementation to check if
// both halves of the string have
// at least one different character
#include <algorithm>
#include <cstring>
#include <iostream>
#include <string>
using namespace std;

// Function which break string into two halves
// Sorts the two halves separately
// Compares the two halves
// return true if any index has non-zero value
bool function(char str[])
{
    int l = strlen(str);
```

```
// Declaration and initialization
// of counter array
sort(str, str + (l / 2));
sort(str + (l / 2), str + l);
for (int i = 0; i < l / 2; i++)
    if (str[i] != str[l / 2 + i])
        return true;
return false;
}

// Driver function
int main()
{
    char str[] = "abcasdsabcae";
    if (function(str))
        cout << "Yes, both halves differ by"
            <<" at least one character";
    else
        cout << "No, both halves do"
            <<" not differ at all";
    return 0;
}
```

Java

```
// Java implementation to check if
// both halves of the string have
// at least one different character

import java.io.*;
import java.util.*;

class GFG {

    // Function which break string into two halves
    // Sorts the two halves separately
    // Compares the two halves
    // return true if any index has non-zero value
    static Boolean function(char str[])
    {
        int l = str.length;

        // Declaration and initialization
        // of counter array
        Arrays.sort(str, 0, (l / 2));
        Arrays.sort(str,(l / 2), l);
        for (int i = 0; i < l / 2; i++)
```

```
        if (str[i] != str[l / 2 + i])
            return true;
    return false;
}

public static void main (String[] args) {
    char str[] = ("abcasdsabcae").toCharArray();
    if (function(str))
        System.out.println("Yes, both halves differ"
+ " by at least one character");
    else
        System.out.println("No, both halves do"
+ " not differ at all");
}
}

// This code is contributed by Gitanjali.
```

Python3

```
# Python implementation to check if
# both halves of the string have
# at least one different character

# Function which break string into two halves
# Sorts the two halves separately
# Compares the two halves
# return true if any index has non-zero value
def function(st):
    st = list(st)
    l = len(st)

    # Declaration and initialization
    # of counter array
    st[:l//2] = sorted(st[:l//2])
    st[l//2:] = sorted(st[l//2:])
    for i in range(l//2):
        if (st[i] != st[l//2 + i]):
            return True
    return False

# Driver function
st = "abcasdsabcae"
if function(st): print("Yes, both halves differ ",
                      "by at least one character")
else: print("No, both halves do not differ at all")

# This code is contributed by Ansu Kumari
```

The complexity of the above approach is $O(\text{len} \log (\text{len}))$

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/check-halves-string-least-one-different-character/>

Chapter 42

Check if given array is almost sorted (elements are at-most one position away)

Check if given array is almost sorted (elements are at-most one position away) - GeeksforGeeks

Given an array with n distinct elements. An array is said to be almost sorted (non-decreasing) if any of its elements can occurs maximum of 1 distance away from their original places in sorted array. We need to find whether the given array is almost sorted or not.

Examples:

Input : arr[] = {1, 3, 2, 4}
Output : Yes
Explanation : All elements are either at original place or at most a unit away.

Input : arr[] = {1, 4, 2, 3}
Output : No
Explanation : 4 is 2 unit away from its original place.

Sorting Approach : With the help of sorting we can predict whether our given array is almost sorted or not. Idea behind that is first sort the input array say A[] and then if array will be almost sorted then each element A_i of given array must be equal to any of B_{i-1} , B_i or B_{i+1} of sorted array B[].

Time Complexity : $O(n \log n)$

Chapter 42. Check if given array is almost sorted (elements are at-most one position away)

```
// suppose B[] is copy of A[]
sort(B, B+n);

// check first element
if ((A[0]!=B[0]) && (A[0]!=B[1]) )
    return 0;
// iterate over array
for(int i=1; i<n-1; i++)
{
    if (A[i]!=B[i-1]) && (A[i]!=B[i]) && (A[i]!=B[i+1]) )
        return false;
}
// check for last element
if ((A[i]!=B[i-1]) && (A[i]!=B[i]) )
    return 0;

// finally return true
return true;
```

Time complexity : $O(n \log n)$

Efficient Approach: The idea is based on [Bubble Sort](#). Like Bubble Sort, we compare adjacent elements and swap them if they are not in order. Here after swapping we move the index one position extra so that bubbling is limited to one place. So after one iteration if resultant array is sorted then we can say that our input array was almost sorted otherwise not almost sorted.

```
// perform bubble sort tech once
for (int i=0; i<n-1; i++)
    if (A[i+1]<A[i])
        swap(A[i], A[i+1]);
    i++;

// check whether resultant is sorted or not
for (int i=0; i<n-1; i++)
    if (A[i+1]<A[i])
        return false;

// If resultant is sorted return true
return true;
```

C++

```
// CPP program to find whether given array
// almost sorted or not
#include <bits/stdc++.h>
using namespace std;
```

```
// function for checking almost sort
bool almostSort(int A[], int n)
{
    // One by one compare adjacents.
    for (int i = 0; i < n - 1; i++) {
        if (A[i] > A[i + 1]) {
            swap(A[i], A[i + 1]);
            i++;
        }
    }

    // check whether resultant is sorted or not
    for (int i = 0; i < n - 1; i++)
        if (A[i] > A[i + 1])
            return false;

    // is resultant is sorted return true
    return true;
}

// driver function
int main()
{
    int A[] = { 1, 3, 2, 4, 6, 5 };
    int n = sizeof(A) / sizeof(A[0]);
    if (almostSort(A, n))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
// JAVA Code to check if given array is almost
// sorted or not
import java.util.*;

class GFG {

    // function for checking almost sort
    public static boolean almostSort(int A[], int n)
    {
        // One by one compare adjacents.
        for (int i = 0; i < n - 1; i++) {
            if (A[i] > A[i + 1]) {
                int temp = A[i];
```

Chapter 42. Check if given array is almost sorted (elements are at-most one position away)

```
A[i] = A[i+1];
A[i+1] = temp;
i++;
}
}

// check whether resultant is sorted or not
for (int i = 0; i < n - 1; i++)
    if (A[i] > A[i + 1])
        return false;

// is resultant is sorted return true
return true;
}

/* Driver program to test above function */
public static void main(String[] args)
{
    int A[] = { 1, 3, 2, 4, 6, 5 };
    int n = A.length;
    if (almostSort(A, n))
        System.out.print("Yes");
    else
        System.out.print("No");

    }
}

// This code is contributed by Arnav Kr. Mandal.
```

C#

```
// C# Code to check if given array
// is almost sorted or not
using System;

class GFG {

    // function for checking almost sort
    public static bool almostSort(int []A, int n)
    {

        // One by one compare adjacents.
        for (int i = 0; i < n - 1; i++)
        {
            if (A[i] > A[i + 1])
            {
                int temp = A[i];
```

Chapter 42. Check if given array is almost sorted (elements are at-most one position away)

```
        A[i] = A[i + 1];
        A[i + 1] = temp;
        i++;
    }
}

// Check whether resultant is
// sorted or not
for (int i = 0; i < n - 1; i++)
    if (A[i] > A[i + 1])
        return false;

// is resultant is sorted return true
return true;
}

// Driver Code
public static void Main()
{
    int []A = {1, 3, 2, 4, 6, 5};
    int n = A.Length;
    if (almostSort(A, n))
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");

}
}

// This code is contributed by Nitin Mittal.
```

PHP

```
<?php
// PHP program to find
// whether given array
// almost sorted or not

// function for checking
// almost sort
function almostSort($A, $n)
{
    // One by one compare adjacents.
    for ($i = 0; $i < $n - 1; $i++)
    {
        if ($A[$i] > $A[$i + 1])
        {
            list($A[$i],
```

Chapter 42. Check if given array is almost sorted (elements are at-most one position away)

```
$A[$i + 1]) = array($A[$i + 1],  
                      $A[$i] );  
  
                $i++;  
            }  
        }  
  
        // check whether resultant  
        // is sorted or not  
        for ($i = 0; $i <$n - 1; $i++)  
            if ($A[$i] > $A[$i + 1])  
                return false;  
  
        // is resultant is  
        // sorted return true  
        return true;  
    }  
  
// Driver Code  
$A = array (1, 3, 2,  
            4, 6, 5);  
$n = sizeof($A) ;  
if (almostSort($A, $n))  
    echo "Yes", "\n";  
else  
    echo "Yes", "\n";  
  
// This code is contributed by ajit  
?>
```

Output:

Yes

Improved By : [nitin mittal, jit_t](#)

Source

<https://www.geeksforgeeks.org/check-given-array-almost-sorted-elements-one-position-away/>

Chapter 43

Check if it is possible to sort an array with conditional swapping of adjacent allowed

Check if it is possible to sort an array with conditional swapping of adjacent allowed - GeeksforGeeks

We are given an unsorted array of integers in the range from 0 to n-1. We are allowed to swap adjacent elements in array many number of times but only if the absolute difference between these element is 1. Check if it is possible to sort the array. If yes then print “yes” else “no”.

Examples:

```
Input : arr[] = {1, 0, 3, 2}
Output : yes
Explanation:- We can swap arr[0] and arr[1].
Again we swap arr[2] and arr[3].
Final arr[] = {0, 1, 2, 3}.

Input : arr[] = {2, 1, 0}
Output : no
```

Although the problem looks complex at first look, there is a simple solution to it. If we traverse array from left to right and we make sure elements before an index i are sorted before we reach i, we must have maximum of arr[0..i-1] just before i. And this maximum must be either smaller than arr[i] or just one greater than arr[i]. In first case, we simply move ahead. In second case, we swap and move ahead.

Compare the current element with the next element in array. If current element is greater than next element then do following:-

...a) Check if difference between two numbers is 1 then swap it.

...b) else Return false.

If we reach end of array, we return true.

C++

```
// C++ program to check if we can sort
// an array with adjacent swaps allowed
#include<bits/stdc++.h>
using namespace std;

// Returns true if it is possible to sort
// else false/
bool checkForSorting(int arr[], int n)
{
    for (int i=0; i<n-1; i++)
    {
        // We need to do something only if
        // previousl element is greater
        if (arr[i] > arr[i+1])
        {
            if (arr[i] - arr[i+1] == 1)
                swap(arr[i], arr[i+1]);

            // If difference is more than
            // one, then not possible
            else
                return false;
        }
    }
    return true;
}

// Driver code
int main()
{
    int arr[] = {1,0,3,2};
    int n = sizeof(arr)/sizeof(arr[0]);
    if (checkForSorting(arr, n))
        cout << "Yes";
    else
        cout << "No";
}
```

Java

```
class Main
{
```

```
// Returns true if it is possible to sort
// else false/
static boolean checkForSorting(int arr[], int n)
{
    for (int i=0; i<n-1; i++)
    {
        // We need to do something only if
        // previousl element is greater
        if (arr[i] > arr[i+1])
        {
            if (arr[i] - arr[i+1] == 1)
            {
                // swapping
                int temp = arr[i];
                arr[i] = arr[i+1];
                arr[i+1] = temp;
            }

            // If difference is more than
            // one, then not possible
            else
                return false;
        }
    }
    return true;
}

// Driver function
public static void main(String args[])
{
    int arr[] = {1,0,3,2};
    int n = arr.length;
    if (checkForSorting(arr, n))
        System.out.println("Yes");
    else
        System.out.println("No");
}
```

Python3

```
# Python 3 program to
# check if we can sort
# an array with adjacent
# swaps allowed

# Returns true if it
# is possible to sort
```

```
# else false/
def checkForSorting(arr, n):

    for i in range(0,n-1):

        # We need to do something only if
        # previousl element is greater
        if (arr[i] > arr[i+1]):

            if (arr[i] - arr[i+1] == 1):
                arr[i], arr[i+1] = arr[i+1], arr[i]

            # If difference is more than
            # one, then not possible
            else:
                return False

    return True

# Driver code
arr = [1,0,3,2]
n = len(arr)
if (checkForSorting(arr, n)):
    print("Yes")
else:
    print("No")

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
// C# program to check if we can sort
// an array with adjacent swaps allowed
using System;

class GFG
{
    // Returns true if it is
    // possible to sort else false
    static bool checkForSorting(int []arr, int n)
    {
        for (int i=0; i<n-1; i++)
        {
            // We need to do something only if
            // previousl element is greater
            if (arr[i] > arr[i+1])
            {
```

```
if (arr[i] - arr[i+1] == 1)
{
    // swapping
    int temp = arr[i];
    arr[i] = arr[i+1];
    arr[i+1] = temp;
}

// If difference is more than
// one, then not possible
else
    return false;
}
return true;
}

// Driver function
public static void Main()
{
    int []arr = {1, 0, 3, 2};
    int n = arr.Length;
    if (checkForSorting(arr, n))
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
}
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// PHP program to check if we can sort
// an array with adjacent swaps allowed
// Returns true if it is possible to sort
// else false

function checkForSorting($arr, $n)
{
    $temp = 0;
    for ($i = 0; $i < $n - 1; $i++)
    {

        // We need to do something only if
        // previous element is greater
        if ($arr[$i] > $arr[$i + 1])
```

```
{  
    if ($arr[$i] - $arr[$i + 1] == 1)  
    {  
        // swapping  
        $temp = $arr[$i];  
        $arr[$i] = $arr[$i + 1];  
        $arr[$i + 1] = $temp;  
    }  
  
    // If difference is more than  
    // one, then not possible  
    else  
        return false;  
}  
}  
return true;  
}  
  
// Driver Code  
$arr = array(1,0,3,2);  
$n = sizeof($arr);  
if (checkForSorting($arr, $n))  
    echo "Yes";  
else  
    echo "No";  
  
// This code is contributed  
// by nitin mittal.  
?>
```

Output:

Yes

Time Complexity=O(n)

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/check-possible-sort-array-conditional-swapping-adjacent-allowed/>

Chapter 44

Check if linked list is sorted (Iterative and Recursive)

Check if linked list is sorted (Iterative and Recursive) - GeeksforGeeks

Given a Linked List, task is to check whether the Linked List is sorted in Descending order or not?

Examples :

```
Input  : 8 -> 7 -> 5 -> 2 -> 1
Output : Yes
Explanation :
In given linked list, starting from head,
8 > 7 > 5 > 2 > 1. So, it is sorted in reverse order

Input  : 24 -> 12 -> 9 -> 11 -> 8 -> 2
Output : No
```

Iterative Approach : Traverse the linked list from head to end. For every newly encountered element, check **node -> data > node -> next -> data**. If True, do same for each node else return 0 and Print “No”.

```
// C++ program to check Linked List is sorted
// in descending order or not
#include <bits/stdc++.h>
using namespace std;

/* Linked list node */
struct Node
{
```

```
int data;
struct Node* next;
};

// function to Check Linked List is
// sorted in descending order or not
bool isSortedDesc(struct Node *head)
{
    if (head == NULL)
        return true;

    // Traverse the list till last node and return
    // false if a node is smaller than or equal
    // its next.
    for (Node *t=head; t->next != NULL; t=t->next)
        if (t->data <= t->next->data)
            return false;
    return true;
}

Node *newNode(int data)
{
    Node *temp = new Node;
    temp->next = NULL;
    temp->data = data;
}

// Driver program to test above
int main()
{
    struct Node *head = newNode(7);
    head->next = newNode(5);
    head->next->next = newNode(4);
    head->next->next->next = newNode(3);

    isSortedDesc(head) ? cout << "Yes" :
                        cout << "No";

    return 0;
}
```

Output:

Yes

Time Complexity : O(N), where N is the length of linked list.

Recursive Approach :

Check Recursively that **node -> data > node -> next -> data**, If not, return 0 that is our terminated condition to come out from recursion else Call Check_List Function Recursively for next node.

```
// C++ program to recursively check Linked List
// is sorted in descending order or not
#include <bits/stdc++.h>
using namespace std;

/* Linked list node */
struct Node
{
    int data;
    struct Node* next;
};

// function to Check Linked List is
// sorted in descending order or not
bool isSortedDesc(struct Node *head)
{
    // Base cases
    if (head == NULL || head->next == NULL)
        return true;

    // Check first two nodes and recursively
    // check remaining.
    return (head->data > head->next->data &&
            isSortedDesc(head->next));
}

Node *newNode(int data)
{
    Node *temp = new Node;
    temp->next = NULL;
    temp->data = data;
}

// Driver program to test above
int main()
{
    struct Node *head = newNode(7);
    head->next = newNode(5);
    head->next->next = newNode(4);
    head->next->next->next = newNode(3);

    isSortedDesc(head) ? cout << "Yes" :
                        cout << "No";
}
```

```
    return 0;  
}
```

Output:

Yes

Source

<https://www.geeksforgeeks.org/check-linked-list-sorting-order/>

Chapter 45

Check if reversing a sub array make the array sorted

Check if reversing a sub array make the array sorted - GeeksforGeeks

Given an array of distinct n integers. The task is to check whether reversing one sub-array make the array sorted or not. If the array is already sorted or by reversing a subarray once make it sorted, print “Yes”, else print “No”.

Examples:

```
Input : arr [] = {1, 2, 5, 4, 3}
Output : Yes
By reversing the subarray {5, 4, 3},
the array will be sorted.
```

```
Input : arr [] = { 1, 2, 4, 5, 3 }
Output : No
```

Method 1 (Simple : $O(n^2)$)

A simple solution is to consider every subarray one by one. Try reversing every subarray and check if reversing the subarray makes the whole array sorted. If yes, return true. If reversing any subarray doesn't make the array sorted, then return false.

Method 2 (Sorting : $O(n \log n)$):

The idea is to compare the given array with the sorted array. Make a copy of the given array and sort it. Now, find the first index and last index which do not match with sorted array. If no such indices are found, print “Yes”. Else check if the elements between the indices are in decreasing order.

```
// C++ program to check whether reversing a
```

```
// sub array make the array sorted or not
#include<bits/stdc++.h>
using namespace std;

// Return true, if reversing the subarray will
// sort the array, else return false.
bool checkReverse(int arr[], int n)
{
    // Copying the array.
    int temp[n];
    for (int i = 0; i < n; i++)
        temp[i] = arr[i];

    // Sort the copied array.
    sort(temp, temp + n);

    // Finding the first mismatch.
    int front;
    for (front = 0; front < n; front++)
        if (temp[front] != arr[front])
            break;

    // Finding the last mismatch.
    int back;
    for (back = n - 1; back >= 0; back--)
        if (temp[back] != arr[back])
            break;

    // If whole array is sorted
    if (front >= back)
        return true;

    // Checking subarray is decreasing or not.
    do
    {
        front++;
        if (arr[front - 1] < arr[front])
            return false;
    } while (front != back);

    return true;
}

// Driven Program
int main()
{
    int arr[] = { 1, 2, 5, 4, 3 };
    int n = sizeof(arr)/sizeof(arr[0]);
```

```
    checkReverse(arr, n)? (cout << "Yes" << endl):
                           (cout << "No" << endl);
    return 0;
}
```

Output:

Yes

Time Complexity: O(nlogn).

Method 3 (Linear : O(n)):

Observe, answer will be “Yes” when the array is sorted or when the array consist of three parts. First part is increasing subarray, then decreasing subarray and then again increasing subarray. So, we need to check that array contain increasing elements then some decreasing elements and then increasing elements. In all other case, answer will be “No”.

Below is the C++ implementation of this approach:

```
// C++ program to check whether reversing a sub array
// make the array sorted or not
#include<bits/stdc++.h>
using namespace std;

// Return true, if reversing the subarray will sort t
// he array, else return false.
bool checkReverse(int arr[], int n)
{
    if (n == 1)
        return true;

    // Find first increasing part
    int i;
    for (i=1; i < n && arr[i-1] < arr[i]; i++);
    if (i == n)
        return true;

    // Find reversed part
    int j = i;
    while (arr[j] < arr[j-1])
    {
        if (i > 1 && arr[j] < arr[i-2])
            return false;
        j++;
    }
}
```

```
if (j == n)
    return true;

// Find last increasing part
int k = j;

// To handle cases like {1,2,3,4,20,9,16,17}
if (arr[k] < arr[i-1])
    return false;

while (k > 1 && k < n)
{
    if (arr[k] < arr[k-1])
        return false;
    k++;
}
return true;
}

// Driven Program
int main()
{
    int arr[] = {1, 3, 4, 10, 9, 8};
    int n = sizeof(arr)/sizeof(arr[0]);
    checkReverse(arr, n)? cout << "Yes" : cout << "No";
    return 0;
}
```

Output:

Yes

Time Complexity: O(n).

Source

<https://www.geeksforgeeks.org/check-reversing-sub-array-make-array-sorted/>

Chapter 46

Check if the given permutation is a valid DFS of graph

Check if the given permutation is a valid DFS of graph - GeeksforGeeks

Given a graph with N nodes numbered from 1 to N and M edges and an array of numbers from 1 to N. Check if it is possible to obtain any permutation of array by applying DFS (Depth First Traversal) on given graph.

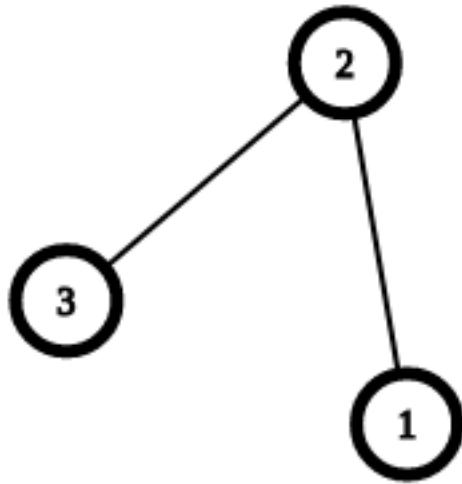
Prerequisites : [DFS Map in CPP](#)

Examples :

```
Input : N = 3, M = 2
        Edges are:
        1) 1-2
        2) 2-3
        P = {1, 2, 3}
Output : YES
Explanation :
Since there are edges between
1-2 and 2-3, therefore we can
have DFS in the order 1-2-3
```

```
Input : N = 3, M = 2
        Edges are:
        1) 1-2
        2) 2-3
        P = {1, 3, 2}
Output : NO
Explanation :
Since there is no edge between 1 and 3,
the DFS traversal is not possible
```

in the order of given permutation.



Possible Permutations in which whole graph can be traversed are:

- 1) 1 2 3
- 2) 3 2 1

Approach : We assume that the input graph is represented as adjacency list. The idea is to first sort all adjacency lists according to input order, then traverse the given graph starting from first node in given given permutation. If we visit all vertices in same order, then given permutation is a valid DFS.

1. Store the indexes of each number in the given permutation in a Hash map.
2. Sort every adjacency list according to the indexes of permutation since there is need to maintain the order.
3. Perform the Depth First Traversal Search with source node as 1st number of given permutation.
4. Keep a counter variable and at every recursive call, check if the counter has reached the number of nodes, i.e. N and set the flag as 1. If the flag is 0 after complete DFS, answer is ‘NO’ otherwise ‘YES’

Below is the implementation of above approach :

```
// CPP program to check if given  
// permutation can be obtained  
// upon DFS traversal on given graph  
#include <bits/stdc++.h>  
using namespace std;  
  
// To track of DFS is valid or not.
```

```
bool flag = false;

// HashMap to store the indexes
// of given permutation
map<int, int> mp;

// Comparator function for sort
bool cmp(int a, int b)
{
    // Sort according ascending
    // order of indexes
    return mp[a] < mp[b];
}

// Graph class represents a undirected
// using adjacency list representation
class Graph
{
    int V; // No. of vertices
    int counter; // Counter variable

public:
    // Pointer to an array containing
    // adjacency lists
    list<int>*> adj;

    Graph(int V); // Constructor

    // function to add an edge to graph
    void addEdge(int u, int v);

    // DFS traversal of the vertices
    // reachable from v
    void DFS(int v, int Perm[]);
};

Graph::Graph(int V)
{
    this->V = V;
    this->counter = 0;
    adj = new list<int>[V + 1];
}

void Graph::addEdge(int u, int v)
{
    adj[u].push_back(v); // Add v to u's list.
    adj[v].push_back(u); // Add u to v's list
}
```

```
// DFS traversal of the
// vertices reachable from v.
void Graph::DFS(int v, int Perm[])
{
    // Increment counter for
    // every node being traversed
    counter++;

    // Check if counter has
    // reached number of vertices
    if (counter == V) {

        // Set flag to 1
        flag = 1;
        return;
    }

    // Recur for all vertices adjacent
    // to this vertex only if it
    // lies in the given permutation
    list<int>::iterator i;
    for (i = adj[v].begin();
         i != adj[v].end(); i++)
    {

        // if the current node equals to
        // current element of permutation
        if (*i == Perm[counter])
            DFS(*i, Perm);
    }
}

// Returns true if P[] is a valid DFS of given
// graph. In other words P[] can be obtained by
// doing a DFS of the graph.
bool checkPermutation(int N, int M,
                      vector<pair<int, int> > V, int P[])
{
    // Create the required graph with
    // N vertices and M edges
    Graph G(N);

    // Add Edges to Graph G
    for (int i = 0; i < M; i++)
        G.addEdge(V[i].first, V[i].second);

    for (int i = 0; i < N; i++)
```

```
mp[P[i]] = i;

// Sort every adjacency
// list according to HashMap
for (int i = 1; i <= N; i++)
    G.adj[i].sort(cmp);

// Call DFS with source node as P[0]
G.DFS(P[0], P);

// If Flag has been set to 1, means
// given permutation is obtained
// by DFS on given graph
return flag;
}

// Driver code
int main()
{
    // Number of vertices and number of edges
    int N = 3, M = 2;

    // Vector of pair to store edges
    vector<pair<int, int> > V;

    V.push_back(make_pair(1, 2));
    V.push_back(make_pair(2, 3));

    int P[] = { 1, 2, 3 };

    // Return the answer
    if (checkPermutation(N, M, V, P))
        cout << "YES" << endl;
    else
        cout << "NO" << endl;

    return 0;
}
```

Output:

YES

Source

<https://www.geeksforgeeks.org/check-given-permutation-valid-dfs-graph/>

Chapter 47

Check if two arrays are equal or not

Check if two arrays are equal or not - GeeksforGeeks

Given two given arrays of equal length, the task is to find if given arrays are equal or not. Two arrays are said to be equal if both of them contain same set of elements, arrangements (or permutation) of elements may be different though.

Note : If there are repetitions, then counts of repeated elements must also be same for two array to be equal.

Examples :

```
Input  : arr1[] = {1, 2, 5, 4, 0};  
         arr2[] = {2, 4, 5, 0, 1};  
Output : Yes
```

```
Input  : arr1[] = {1, 2, 5, 4, 0, 2, 1};  
         arr2[] = {2, 4, 5, 0, 1, 1, 2};  
Output : Yes
```

```
Input : arr1[] = {1, 7, 1};  
        arr2[] = {7, 7, 1};  
Output : No
```

A **simple solution** is to sort both array and then linearly compare elements.
C/C++

```
// C++ program to find given two array  
// are equal or not  
#include<bits/stdc++.h>
```

```
using namespace std;

// Returns true if arr1[0..n-1] and arr2[0..m-1]
// contain same elements.
bool areEqual(int arr1[], int arr2[], int n, int m)
{
    // If lengths of array are not equal means
    // array are not equal
    if (n != m)
        return false;

    // Sort both arrays
    sort(arr1, arr1+n);
    sort(arr2, arr2+m);

    // Linearly compare elements
    for (int i=0; i<n; i++)
        if (arr1[i] != arr2[i])
            return false;

    // If all elements were same.
    return true;
}

// Driver Code
int main()
{
    int arr1[] = { 3, 5, 2, 5, 2};
    int arr2[] = { 2, 3, 5, 5, 2};
    int n = sizeof(arr1)/sizeof(int);
    int m = sizeof(arr2)/sizeof(int);

    if (areEqual(arr1, arr2, n, m))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
// Java program to find given two array
// are equal or not
import java.io.*;
import java.util.*;

class GFG
{
```

```
// Returns true if arr1[0..n-1] and arr2[0..m-1]
// contain same elements.
public static boolean areEqual(int arr1[], int arr2[])
{
    int n = arr1.length;
    int m = arr2.length;

    // If lengths of array are not equal means
    // array are not equal
    if (n != m)
        return false;

    // Sort both arrays
    Arrays.sort(arr1);
    Arrays.sort(arr2);

    // Linearly compare elements
    for (int i=0; i<n; i++)
        if (arr1[i] != arr2[i])
            return false;

    // If all elements were same.
    return true;
}

//Driver code
public static void main (String[] args)
{
    int arr1[] = { 3, 5, 2, 5, 2};
    int arr2[] = { 2, 3, 5, 5, 2};

    if (areEqual(arr1, arr2))
        System.out.println("Yes");
    else
        System.out.println("No");

}
```

C#

```
// C# program to find given two array
// are equal or not
using System;

class GFG {

    // Returns true if arr1[0..n-1] and
```

```
// arr2[0..m-1] contain same elements.
public static bool areEqual(int []arr1,
                            int []arr2)
{
    int n = arr1.Length;
    int m = arr2.Length;

    // If lengths of array are not
    // equal means array are not equal
    if (n != m)
        return false;

    // Sort both arrays
    Array.Sort(arr1);
    Array.Sort(arr2);

    // Linearly compare elements
    for (int i = 0; i < n; i++)
        if (arr1[i] != arr2[i])
            return false;

    // If all elements were same.
    return true;
}

// Driver code
public static void Main ()
{
    int []arr1 = { 3, 5, 2, 5, 2};
    int []arr2 = { 2, 3, 5, 5, 2};

    if (areEqual(arr1, arr2))
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP program to find given
// two array are equal or not

// Returns true if arr1[0..n-1]
// and arr2[0..m-1] contain same elements.
```

```
function areEqual( $arr1, $arr2, $n, $m)
{
    // If lengths of array
    // are not equal means
    // array are not equal
    if ($n != $m)
        return false;

    // Sort both arrays
    sort($arr1);
    sort($arr2);

    // Linearly compare elements
    for ( $i = 0; $i < $n; $i++)
        if ($arr1[$i] != $arr2[$i])
            return false;

    // If all elements were same.
    return true;
}

// Driver Code
$arr1 = array( 3, 5, 2, 5, 2);
$arr2 = array( 2, 3, 5, 5, 2);
$n = count($arr1);
$m = count($arr2);

if (areEqual($arr1, $arr2, $n, $m))
    echo "Yes";
else
    echo "No";

// This code is contributed by anuj_67.
?>
```

Output :

Yes

Time Complexity : $O(n \log n)$

Auxiliary Space : $O(1)$

An **Efficient solution** of this approach is to use hashing. We store all elements of arr1[] and their counts in a hash table. Then we traverse arr2[] and check if count of every element in arr2[] matches with count in arr1[].

Below is C++ implementation of above idea. We use [unordered_map](#) to store counts.

C/C++

```
// C++ program to find given two array
// are equal or not using hashing technique
#include<bits/stdc++.h>
using namespace std;

// Returns true if arr1[0..n-1] and arr2[0..m-1]
// contain same elements.
bool areEqual(int arr1[], int arr2[], int n, int m)
{
    // If lengths of arrays are not equal
    if (n != m)
        return false;

    // Store arr1[] elements and their counts in
    // hash map
    unordered_map<int, int> mp;
    for (int i=0; i<n; i++)
        mp[arr1[i]]++;

    // Traverse arr2[] elements and check if all
    // elements of arr2[] are present same number
    // of times or not.
    for (int i=0; i<n; i++)
    {
        // If there is an element in arr2[], but
        // not in arr1[]
        if (mp.find(arr2[i]) == mp.end())
            return false;

        // If an element of arr2[] appears more
        // times than it appears in arr1[]
        if (mp[arr2[i]] == 0)
            return false;

        mp[arr2[i]]--;
    }

    return true;
}

// Driver Code
int main()
{
    int arr1[] = {3, 5, 2, 5, 2};
    int arr2[] = {2, 3, 5, 5, 2};
    int n = sizeof(arr1)/sizeof(int);
    int m = sizeof(arr2)/sizeof(int);
```

```
if (areEqual(arr1, arr2, n, m))
    cout << "Yes";
else
    cout << "No";
return 0;
}
```

Java

```
// Java program to find given two array
// are equal or not using hashing technique
import java.util.*;
import java.io.*;

class GFG
{
    // Returns true if arr1[0..n-1] and arr2[0..m-1]
    // contain same elements.
    public static boolean areEqual(int arr1[], int arr2[])
    {
        int n = arr1.length;
        int m = arr2.length;

        // If lengths of arrays are not equal
        if (n != m)
            return false;

        // Store arr1[] elements and their counts in
        // hash map
        Map<Integer, Integer> map = new HashMap<Integer, Integer>();
        int count = 0;
        for (int i = 0; i < n; i++)
        {
            if(map.get(arr1[i]) == null)
                map.put(arr1[i], 1);
            else
            {
                count = map.get(arr1[i]);
                count++;
                map.put(arr1[i], count);
            }
        }

        // Traverse arr2[] elements and check if all
        // elements of arr2[] are present same number
        // of times or not.
        for (int i = 0; i < n; i++)
        {
```

```
// If there is an element in arr2[], but
// not in arr1[]
if (!map.containsKey(arr2[i]))
    return false;

// If an element of arr2[] appears more
// times than it appears in arr1[]
if (map.get(arr2[i]) == 0)
    return false;

count = map.get(arr2[i]);
--count;
map.put(arr2[i], count);
}

// again traverse arr2 to ensure that count
// for all elements become zero.
for(int i = 0; i < n; i++)
{
    if(map.get(arr2[i]) > 0)
        return false;
}
return true;
}

//Driver code
public static void main (String[] args)
{
    int arr1[] = { 3, 5, 2, 5, 2};
    int arr2[] = { 2, 3, 5, 5, 2};

    if (areEqual(arr1, arr2))
        System.out.println("Yes");
    else
        System.out.println("No");

}
```

Output :

Yes

Time Complexity : O(n)
Auxiliary Space : O(n)

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/check-if-two-arrays-are-equal-or-not/>

Chapter 48

Check whether Arithmetic Progression can be formed from the given array

Check whether Arithmetic Progression can be formed from the given array - GeeksforGeeks

Given an array of **n** integers. The task is to check whether an arithmetic progression can be formed using all the given elements. If possible print “Yes”, else print “No”.

Examples:

```
Input : arr[] = {0, 12, 4, 8}
Output : Yes
Rearrange given array as {0, 4, 8, 12}
which forms an arithmetic progression.
```

```
Input : arr[] = {12, 40, 11, 20}
Output : No
```

Method 1 (Simple)

A simple solution is to first find the smallest element, then find second smallest element and find the difference between these two. Let this difference be d. After finding the difference, find third smallest, fourth smallest and so on. After finding every i-th smallest (from third onward), find the difference between value of current element and value of previous element. If difference is not same as d, return false. If all elements have same difference, return true. Time complexity of this solution is $O(n^2)$

Method 2(Use Sorting)

The idea is to sort the given array. After sorting, check if differences between consecutive elements are same or not. If all differences are same, Arithmetic Progression is possible.

Below is the implementation of this approach:

C++

```
// C++ program to check if a given array
// can form arithmetic progression
#include<bits/stdc++.h>
using namespace std;

// Returns true if a permutation of arr[0..n-1]
// can form arithmetic progression
bool checkIsAP(int arr[], int n)
{
    if (n == 1)
        return true;

    // Sort array
    sort(arr, arr + n);

    // After sorting, difference between
    // consecutive elements must be same.
    int d = arr[1] - arr[0];
    for (int i=2; i<n; i++)
        if (arr[i] - arr[i-1] != d)
            return false;

    return true;
}

// Driven Program
int main()
{
    int arr[] = { 20, 15, 5, 0, 10 };
    int n = sizeof(arr)/sizeof(arr[0]);

    (checkIsAP(arr, n)) ? (cout << "Yes" << endl) :
                           (cout << "No" << endl);

    return 0;
}
```

Java

```
// Java program to check if a given array
// can form arithmetic progression
import java.util.Arrays;

class GFG {
```

```
// Returns true if a permutation of
// arr[0..n-1] can form arithmetic
// progression
static boolean checkIsAP(int arr[], int n)
{
    if (n == 1)
        return true;

    // Sort array
    Arrays.sort(arr);

    // After sorting, difference between
    // consecutive elements must be same.
    int d = arr[1] - arr[0];
    for (int i = 2; i < n; i++)
        if (arr[i] - arr[i-1] != d)
            return false;

    return true;
}

//driver code
public static void main (String[] args)
{
    int arr[] = { 20, 15, 5, 0, 10 };
    int n = arr.length;

    if(checkIsAP(arr, n))
        System.out.println("Yes");
    else
        System.out.println("No");
}
```

// This code is contributed by Anant Agarwal.

Python3

```
# Python3 program to check if a given
# array can form arithmetic progression

# Returns true if a permutation of arr[0..n-1]
# can form arithmetic progression
def checkIsAP(arr, n):
    if (n == 1): return True

    # Sort array
```

```
arr.sort()

# After sorting, difference between
# consecutive elements must be same.
d = arr[1] - arr[0]
for i in range(2, n):
    if (arr[i] - arr[i-1] != d):
        return False

return True

# Driver code
arr = [ 20, 15, 5, 0, 10 ]
n = len(arr)
print("Yes") if(checkIsAP(arr, n)) else print("No")

# This code is contributed by Anant Agarwal.
```

C#

```
// C# program to check if a given array
// can form arithmetic progression
using System;

class GFG {

    // Returns true if a permutation of
    // arr[0..n-1] can form arithmetic
    // progression
    static bool checkIsAP(int []arr, int n)
    {
        if (n == 1)
            return true;

        // Sort array
        Array.Sort(arr);

        // After sorting, difference between
        // consecutive elements must be same.
        int d = arr[1] - arr[0];
        for (int i = 2; i < n; i++)
            if (arr[i] - arr[i - 1] != d)
                return false;

        return true;
    }

    //Driver Code
}
```

```
public static void Main ()
{
    int []arr = {20, 15, 5, 0, 10};
    int n = arr.Length;

    if(checkIsAP(arr, n))
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to check if
// a given array can form
// arithmetic progression

// Returns true if a permutation
// of arr[0..n-1] can form
// arithmetic progression
function checkIsAP($arr, $n)
{
    if ($n == 1)
        return true;

    // Sort array
    sort($arr);

    // After sorting, difference
    // between consecutive elements
    // must be same.
    $d = $arr[1] - $arr[0];
    for ($i = 2; $i < $n; $i++)
        if ($arr[$i] -
            $arr[$i - 1] != $d)
            return false;

    return true;
}

// Driver Code
$arr = array(20, 15, 5, 0, 10);
$n = count($arr);
```

```
if(checkIsAP($arr, $n))
echo "Yes";
else
echo "No";

// This code is contributed
// by Sam007
?>
```

Output:

Yes

Time Complexity: $O(n \log n)$.

Method 3(Use Hashing)

1. Find out the smallest and second smallest elements
2. Find difference between the two elements. $d = \text{second_smallest} - \text{smallest}$
3. Store all elements in a hashmap and return “NO” if duplicate element found (can be done together with step 1).
4. Now start from “second smallest element + d” and one by one check $n-2$ terms of Arithmetic Progression in hashmap. If any value of progression is missing, return false.
5. Return “YES” after end of the loop.

Time Complexity : $O(n)$

Auxiliary Space : $O(n)$

Thanks to **Chenna Rao** for suggesting this method,

Method 4(Using counting sort)

We can reduce space required in method 3 if given array can be modified.

1. Find smallest and second smallest elements.
2. Find $d = \text{second_smallest} - \text{smallest}$
3. Subtract smallest element from all elements.
4. Now if given array represent AP, all elements should be of form $i*d$ where i varies from 0 to $n-1$.
5. One by one divide all reduced elements with d . If any element is not divisible by d , return false.

Chapter 48. Check whether Arithmetic Progression can be formed from the given array

6. Now if array represents AP, it must be a permutation of numbers from 0 to n-1. We can easily check this using counting sort.

Improved By : [vt_m](#), [Sam007](#)

Source

<https://www.geeksforgeeks.org/check-whether-arithmetic-progression-can-formed-given-array/>

Chapter 49

Check whether a given array is a k sorted array or not

Check whether a given array is a k sorted array or not - GeeksforGeeks

Given an array of **n** distinct elements. Check whether the given array is a **k** sorted array or not. A **k** sorted array is an array where each element is at most **k** distance away from its target position in the sorted array.

For example, let us consider **k** is 2, an element at index 7 in the sorted array, can be at indexes 5, 6, 7, 8, 9 in the given array.

Examples :

Input : arr[] = {3, 2, 1, 5, 6, 4}, k = 2
Output : Yes
Every element is at most 2 distance away
from its target position in the sorted array.

Input : arr[] = {13, 8, 10, 7, 15, 14, 12}, k = 3
Output : No
13 is more than k = 3 distance away
from its target position in the sorted array.

Copy elements of original array **arr[]** to an auxiliary array **aux[]**. Sort **aux[]**. Now, for each element at index **i** in **arr[]**, find its index **j** in **aux[]** using Binary Search. If for any element **k < abs(i-j)**, then **arr[]** is not a **k** sorted array. Else it is a **k** sorted array. Here **abs** is the absolute value.

C++

```
// C++ implementation to check whether the given array
// is a k sorted array or not
```

```
#include <bits / stdc++.h>
using namespace std;

// function to find index of element 'x' in sorted 'arr'
// uses binary search technique
int binarySearch(int arr[], int low, int high, int x)
{
    while (low <= high)
    {
        int mid = (low + high) / 2;

        if (arr[mid] == x)
            return mid;
        else if (arr[mid] > x)
            high = mid - 1;
        else
            low = mid + 1;
    }
}

// function to check whether the given array is
// a 'k' sorted array or not
string isKSortedArray(int arr[], int n, int k)
{
    // auxiliary array 'aux'
    int aux[n];

    // copy elements of 'arr' to 'aux'
    for (int i = 0; i < n; i++)
        aux[i] = arr[i];

    // sort 'aux'
    sort(aux, aux + n);

    // for every element of 'arr' at index 'i',
    // find its index 'j' in 'aux'
    for (int i = 0; i < n; i++)
    {
        // index of arr[i] in sorted array 'aux'
        int j = binarySearch(aux, 0, n-1, arr[i]);

        // if abs(i-j) > k, then that element is
        // not at-most k distance away from its
        // target position. Thus, 'arr' is not a
        // k sorted array
        if (abs(i - j) > k)
            return "No";
    }
}
```

```
// 'arr' is a k sorted array
return "Yes";
}

// Driver program to test above
int main()
{
    int arr[] = {3, 2, 1, 5, 6, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    cout << "Is it a k sorted array?: "
         << isKSortedArray(arr, n, k);
    return 0;
}
```

Java

```
// Java implementation to check whether the given array
// is a k sorted array or not

import java.util.Arrays;

class Test
{
    // Method to check whether the given array is
    // a 'k' sorted array or not
    static String isKSortedArray(int arr[], int n, int k)
    {
        // auxiliary array 'aux'
        int aux[] = new int[n];

        // copy elements of 'arr' to 'aux'
        for (int i = 0; i<n; i++)
            aux[i] = arr[i];

        // sort 'aux'
        Arrays.sort(aux);

        // for every element of 'arr' at index 'i',
        // find its index 'j' in 'aux'
        for (int i = 0; i<n; i++)
        {
            // index of arr[i] in sorted array 'aux'
            int j = Arrays.binarySearch(aux, arr[i]);

            // if abs(i-j) > k, then that element is
            // not at-most k distance away from its
```

```
// target position. Thus, 'arr' is not a
// k sorted array
if (Math.abs(i - j) > k)
    return "No";
}

// 'arr' is a k sorted array
return "Yes";
}

// Driver method
public static void main(String args[])
{
    int arr[] = {3, 2, 1, 5, 6, 4};
    int k = 2;

    System.out.println("Is it a k sorted array ?: " +
                       isKSortedArray(arr, arr.length, k));
}
}
```

C#

```
// C# implementation to check
// whether the given array is a
// k sorted array or not
using System;
using System.Collections;

class GFG {

    // Method to check whether the given
    // array is a 'k' sorted array or not
    static String isKSortedArray(int []arr, int n, int k)
    {
        // auxiliary array 'aux'
        int []aux = new int[n];

        // copy elements of 'arr' to 'aux'
        for (int i = 0; i < n; i++)
            aux[i] = arr[i];

        // sort 'aux'
        Array.Sort(aux);

        // for every element of 'arr' at index
        // 'i', find its index 'j' in 'aux'
        for (int i = 0; i < n; i++)
```

```
{  
    // index of arr[i] in sorted array 'aux'  
    int j = Array.BinarySearch(aux, arr[i]);  
  
    // if abs(i-j) > k, then that element is  
    // not at-most k distance away from its  
    // target position. Thus, 'arr' is not a  
    // k sorted array  
    if (Math.Abs(i - j) > k)  
        return "No";  
}  
  
// 'arr' is a k sorted array  
return "Yes";  
}  
  
// Driver method  
public static void Main()  
{  
    int []arr = {3, 2, 1, 5, 6, 4};  
    int k = 2;  
  
    Console.WriteLine("Is it a k sorted array ?: " +  
                      isKSortedArray(arr, arr.Length, k));  
}  
}  
  
// This code is contributed by Sam007
```

Output:

```
Is it a k sorted array?: Yes
```

Time Complexity: O(nlogn)
Auxiliary space: O(n)

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/check-whether-given-array-k-sorted-array-not/>

Chapter 50

Check whether a given number is even or odd

Check whether a given number is even or odd - GeeksforGeeks

Given a number, check whether it is even or odd.

A number is called even if it is divisible by 2. All other numbers are odd (not divisible by 2)

First few even numbers : 0, 2, 4, 6,

First few odd numbers : 1, 3, 5, 7,

Input : n = 12

Output : Even

Input : n = 13

Output : Odd

Examples :

```
input: 2
output: even
```

```
input: 5
output: odd
```

One **simple solution** is to find remainder after dividing by 2.
C++

```
// A simple C++ program to
// check for even or odd
#include <iostream>
using namespace std;

// Returns true if n is
// even, else odd
bool isEven(int n)
{
    return (n % 2 == 0);

// Driver code
int main()
{
    int n = 101;
    isEven(n)? cout << "Even" :
                cout << "Odd";

    return 0;
}
```

Java

```
// Java program program to
// check for even or odd

class GFG
{
    // Returns true if n is even, else odd
    public static boolean isEven(int n)
    {
        return (n % 2 == 0);
    }

    // Driver code
    public static void main(String[] args)
    {
```

```
int n = 101;
if(isEven(n) == true)
    System.out.print("Even");
else
    System.out.print("Odd");
}

// This code is contributed by rishabh_jain
```

Python3

```
# A simple Python3 code
# to check for even or odd

# Returns true if n is even, else odd
def isEven(n):
    return (n % 2 == 0)

# Driver code
n = 101
print("Even" if isEven(n) else "Odd")

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# program program to
// check for even or odd
using System;

class GFG
{
    // Returns true if n is even, else odd
    public static bool isEven(int n)
    {
        return (n % 2 == 0);
    }

    // Driver code
    public static void Main()
    {
        int n = 101;
        if(isEven(n) == true)
            Console.WriteLine("Even");
        else
            Console.WriteLine("Odd");
```

```
    }
}

// This code is contributed by vt_m
```

PHP

```
<?php
// A simple PHP program to
// check for even or odd

// Returns true if n is
// even, else odd
function isEven($n)
{
    return ($n % 2 == 0);
}

// Driver code
$n = 101;
if(isEven != true)
    echo "Even";
else
    echo "Odd";

// This code is contributed by Ajit
?>
```

Output :

Odd

A **better solution** is to use bitwise operators. We need to check whether last bit is 1 or not. If last bit is 1 then number is odd, otherwise always even.

Explanation:

```
input : 5      // odd
      00000101
& 00000001
-----
      00000001
-----

input : 8      //even
```

```
00001000
& 00000001
-----
00000000
-----
```

Below is the implementation of the idea.

C++

```
// A simple C++ program to
// check for even or odd
#include <iostream>
using namespace std;

// Returns true if n is
// even, else odd
bool isEven(int n)
{

// n & 1 is 1, then
// odd, else even
return !(n & 1);
}

// Driver code
int main()
{
int n = 101;
isEven(n)? cout << "Even" :
            cout << "Odd";

return 0;
}
```

Java

```
// Java program program to
// check for even or odd

class GFG
{
    // Returns true if n
    // is even, else odd
    public static boolean isEven(int n)
    {
        if((n & 1) == 0)
            return true;
```

```
        else
            return false;
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 101;
        if(isEven(n) == true)
            System.out.print("Even");
        else
            System.out.print("Odd");
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
# A Python3 code program
# to check for even or odd

# Returns true if n is even, else odd
def isEven(n):

    # n&1 is 1, then odd, else even
    return (not(n & 1))

# Driver code
n = 101;
print("Even" if isEven(n) else "Odd")

# This code is contributed by Sharad_Bhardwaj>.
```

C#

```
// C# program program to
// check for even or odd
using System;

class GFG
{
    // Returns true if n
    // is even, else odd
    public static bool isEven(int n)
    {
        if((n & 1) == 0)
```

```
        return true;
    else
        return false;
}

// Driver code
public static void Main()
{
    int n = 101;
    if(isEven(n) == true)
        Console.WriteLine("Even");
    else
        Console.WriteLine("Odd");
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// A simple PHP program to
// check for even or odd

// Returns true if n is
// even, else odd
function isEven($n)
{
    return !( $n & 1 );
}

// Driver code
$n = 101;
if(isEven($n) == true)
    echo "Even";
else
    echo "Odd";

// This code is contributed by Smitha
?>
```

Output :

Odd

Improved By : [vt_m](#), [jit_t](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/check-whether-given-number-even-odd/>

Chapter 51

Check whether an array can be fit into another array rearranging the elements in the array

Check whether an array can be fit into another array rearranging the elements in the array
- GeeksforGeeks

Given two arrays A and B of same size N. Check whether array A can be fit into array B. An array is said to fit into another array if by arranging the elements of both arrays, there exists a solution such that the i^{th} element of the first array is less than or equal to i^{th} element of the second array.

Examples:

```
Input : A[] = { 7, 5, 3, 2 }, B[] = { 5, 4, 8, 7 }
Output : YES
Rearrange the first array to {3, 2, 7, 5}
Do not rearrange the second array's element.
After rearranging, all Ai<=Bi.
```

```
Input : A[] = { 7, 5, 3, 2, 5, 105, 45, 10 }, B[] = { 2, 4, 0, 5, 6, 9, 75, 84 }
Output : NO
```

Approach : Sort both the arrays and check whether A_i is less than or equal to B_i for all $0 \leq i < N$. If at any i^{th} position A_i is greater than B_i return false, otherwise return true.

Below is the implementation of the above approach:

C++

```
// C++ Program to check whether an array
```

```
// can be fit into another array with given
// condition.
#include <bits/stdc++.h>
using namespace std;

// Returns true if the array A can be fit into
// array B, otherwise false
bool checkFittingArrays(int A[], int B[], int N)
{
    // Sort both the arrays
    sort(A, A + N);
    sort(B, B + N);

    // Iterate over the loop and check whether every
    // array element of A is less than or equal to
    // its corresponding array element of B
    for (int i = 0; i < N; i++)
        if (A[i] > B[i])
            return false;

    return true;
}

// Driver Code
int main()
{
    int A[] = { 7, 5, 3, 2 };
    int B[] = { 5, 4, 8, 7 };
    int N = sizeof(A) / sizeof(A[0]);

    if (checkFittingArrays(A, B, N))
        cout << "YES";
    else
        cout << "NO";
    return 0;
}
```

Java

```
// Java Program to check
// whether an array can
// be fit into another
// array with given
// condition.
import java.io.*;
import java.util.*;
import java.lang.*;
```

```
class GFG
{
    // Returns true if the
    // array A can be fit
    // into array B,
    // otherwise false
    static boolean checkFittingArrays(int []A,
                                      int []B,
                                      int N)
    {
        // Sort both the arrays
        Arrays.sort(A);
        Arrays.sort(B);

        // Iterate over the loop
        // and check whether every
        // array element of A is
        // less than or equal to
        // its corresponding array
        // element of B
        for (int i = 0; i < N; i++)
            if (A[i] > B[i])
                return false;

        return true;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int A[] = {7, 5, 3, 2};
        int B[] = {5, 4, 8, 7};
        int N = A.length;

        if (checkFittingArrays(A, B, N))
            System.out.print("YES");
        else
            System.out.print("NO");
    }
}
```

C#

```
// C# Program to check
// whether an array can
// be fit into another
```

```
// array with given
// condition.
using System;

class GFG
{

    // Returns true if the
    // array A can be fit
    // into array B,
    // otherwise false
    static bool checkFittingArrays(int []A,
                                    int []B,
                                    int N)
    {

        // Sort both the arrays
        Array.Sort(A);
        Array.Sort(B);

        // Iterate over the loop
        // and check whether every
        // array element of A is
        // less than or equal to
        // its corresponding array
        // element of B
        for (int i = 0; i < N; i++)
            if (A[i] > B[i])
                return false;

        return true;
    }

    // Driver Code
    public static void Main ()
    {
        int []A = {7, 5, 3, 2};
        int []B = {5, 4, 8, 7};
        int N = A.Length;

        if (checkFittingArrays(A, B, N))
            Console.WriteLine("YES");
        else
            Console.WriteLine("NO");
    }
}

// This code is contributed
```

// by anuj_67.

Output:

YES

Time Complexity : $O(N * \log N)$, where N is the size of array.

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/check-whether-an-array-can-be-fit-into-another-array-rearranging-the-elements-in-the-array/>

Chapter 52

Chocolate Distribution Problem

Chocolate Distribution Problem - GeeksforGeeks

Given an array of n integers where each value represents number of chocolates in a packet. Each packet can have variable number of chocolates. There are m students, the task is to distribute chocolate packets such that:

1. Each student gets one packet.
2. The difference between the number of chocolates in packet with maximum chocolates and packet with minimum chocolates given to the students is minimum.

Examples:

```
Input : arr[] = {7, 3, 2, 4, 9, 12, 56}
        m = 3
Output: Minimum Difference is 2
We have seven packets of chocolates and
we need to pick three packets for 3 students
If we pick 2, 3 and 4, we get the minimum
difference between maximum and minimum packet
sizes.
```

```
Input : arr[] = {3, 4, 1, 9, 56, 7, 9, 12}
        m = 5
Output: Minimum Difference is 6
The set goes like 3,4,7,9,9 and the output
is 9-3 = 6
```

```
Input : arr[] = {12, 4, 7, 9, 2, 23, 25, 41,
                30, 40, 28, 42, 30, 44, 48,
                43, 50}
```

```
m = 7
Output: Minimum Difference is 10
We need to pick 7 packets. We pick 40, 41,
42, 44, 48, 43 and 50 to minimize difference
between maximum and minimum.
```

Source: [Flipkart Interview Experience](#)

A **simple solution** is to generate all subsets of size m of arr[0..n-1]. For every subset, find the difference between maximum and minimum elements in it. Finally return minimum difference.

An **efficient solution** is based on the observation that, to minimize difference, we must choose consecutive elements from a sorted packets. We first sort the array arr[0..n-1], then find the subarray of size m with minimum difference between last and first elements.

C++

```
// C++ program to solve chocolate distribution
// problem
#include<bits/stdc++.h>
using namespace std;

// arr[0..n-1] represents sizes of packets
// m is number of students.
// Returns minimum difference between maximum
// and minimum values of distribution.
int findMinDiff(int arr[], int n, int m)
{
    // if there are no chocolates or number
    // of students is 0
    if (m==0 || n==0)
        return 0;

    // Sort the given packets
    sort(arr, arr+n);

    // Number of students cannot be more than
    // number of packets
    if (n < m)
        return -1;

    // Largest number of chocolates
    int min_diff = INT_MAX;

    // Find the subarray of size m such that
    // difference between last (maximum in case
    // of sorted) and first (minimum in case of
    // sorted) elements of subarray is minimum.
```

```
int first = 0, last = 0;
for (int i=0; i+m-1<n; i++)
{
    int diff = arr[i+m-1] - arr[i];
    if (diff < min_diff)
    {
        min_diff = diff;
        first = i;
        last = i + m - 1;
    }
}
return (arr[last] - arr[first]);
}

int main()
{
    int arr[] = {12, 4, 7, 9, 2, 23, 25, 41,
                 30, 40, 28, 42, 30, 44, 48,
                 43, 50};
    int m = 7; // Number of students
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Minimum difference is "
         << findMinDiff(arr, n, m);
    return 0;
}
```

Java

```
// JAVA Code For Chocolate Distribution
// Problem
import java.util.*;

class GFG {

    // arr[0..n-1] represents sizes of
    // packets. m is number of students.
    // Returns minimum difference between
    // maximum and minimum values of
    // distribution.
    static int findMinDiff(int arr[], int n,
                           int m)
    {
        // if there are no chocolates or
        // number of students is 0
        if (m == 0 || n == 0)
            return 0;

        // Sort the given packets
```

```
Arrays.sort(arr);

// Number of students cannot be
// more than number of packets
if (n < m)
    return -1;

// Largest number of chocolates
int min_diff = Integer.MAX_VALUE;

// Find the subarray of size m
// such that difference between
// last (maximum in case of
// sorted) and first (minimum in
// case of sorted) elements of
// subarray is minimum.
int first = 0, last = 0;
for (int i = 0; i + m - 1 < n; i++)
{
    int diff = arr[i+m-1] - arr[i];
    if (diff < min_diff)
    {
        min_diff = diff;
        first = i;
        last = i + m - 1;
    }
}
return (arr[last] - arr[first]);
}

/* Driver program to test above function */
public static void main(String[] args)
{
    int arr[] = {12, 4, 7, 9, 2, 23,
                25, 41, 30, 40, 28,
                42, 30, 44, 48, 43,
                50};

    int m = 7; // Number of students

    int n = arr.length;
    System.out.println("Minimum difference is "
        + findMinDiff(arr, n, m));

}
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
# Python3 program to solve
# chocolate distribution
# problem

import sys;

# arr[0..n-1] represents sizes of packets
# m is number of students.
# Returns minimum difference between maximum
# and minimum values of distribution.
def findMinDiff(arr, n, m):

    # if there are no chocolates or number
    # of students is 0
    if (m==0 or n==0):
        return 0

    # Sort the given packets
    arr.sort()

    # Number of students cannot be more than
    # number of packets
    if (n < m):
        return -1

    # Largest number of chocolates
    min_diff = sys.maxsize

    # Find the subarray of size m such that
    # difference between last (maximum in case
    # of sorted) and first (minimum in case of
    # sorted) elements of subarray is minimum.
    first = 0
    last = 0
    i=0
    while(i+m-1<n ):

        diff = arr[i+m-1] - arr[i]
        if (diff < min_diff):

            min_diff = diff
            first = i
            last = i + m - 1

        i+=1
```

```
return (arr[last] - arr[first])

# Driver Code
if __name__ == "__main__":
    arr = [12, 4, 7, 9, 2, 23, 25, 41,
           30, 40, 28, 42, 30, 44, 48,
           43, 50]
    m = 7 # Number of students
    n = len(arr)
    print("Minimum difference is", findMinDiff(arr, n, m))

#This code is contributed by Smitha
```

C#

```
// C# Code For Chocolate Distribution
// Problem
using System;

class GFG {

    // arr[0..n-1] represents sizes of
    // packets. m is number of students.
    // Returns minimum difference between
    // maximum and minimum values of
    // distribution.
    static int findMinDiff(int []arr, int n,
                           int m)
    {

        // if there are no chocolates or
        // number of students is 0
        if (m == 0 || n == 0)
            return 0;

        // Sort the given packets
        Array.Sort(arr);

        // Number of students cannot be
        // more than number of packets
        if (n < m)
            return -1;

        // Largest number of chocolates
        int min_diff = int.MaxValue;

        // Find the subarray of size m
        // such that difference between
```

```
// last (maximum in case of
// sorted) and first (minimum in
// case of sorted) elements of
// subarray is minimum.
int first = 0, last = 0;
for (int i = 0; i + m - 1 < n; i++)
{
    int diff = arr[i+m-1] - arr[i];

    if (diff < min_diff)
    {
        min_diff = diff;
        first = i;
        last = i + m - 1;
    }
}

return (arr[last] - arr[first]);
}

/* Driver program to test above function */
public static void Main()
{
    int []arr = {12, 4, 7, 9, 2, 23,
                25, 41, 30, 40, 28,
                42, 30, 44, 48, 43,
                50};

    int m = 7; // Number of students

    int n = arr.Length;

    Console.WriteLine("Minimum difference is "
                      + findMinDiff(arr, n, m));
}

}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to solve
// chocolate distribution
// problem

// arr[0..n-1] represents
```

```
// sizes of packets m is
// number of students.
// Returns minimum difference
// between maximum and minimum
// values of distribution.
function findMinDiff($arr, $n, $m)
{
    // if there are no
    // chocolates or number
    // of students is 0
    if ($m == 0 || $n == 0)
        return 0;

    // Sort the given packets
    sort($arr);

    // Number of students
    // cannot be more than
    // number of packets
    if ($n < $m)
        return -1;

    // Largest number
    // of chocolates
    $min_diff = PHP_INT_MAX;

    // Find the subarray of size
    // m such that difference
    // between last (maximum in
    // case of sorted) and first
    // (minimum in case of sorted)
    // elements of subarray is minimum.
    $first = 0; $last = 0;
    for ($i = 0;
         $i + $m - 1 < $n; $i++)
    {
        $diff = $arr[$i + $m - 1] -
               $arr[$i];
        if ($diff < $min_diff)
        {
            $min_diff = $diff;
            $first = $i;
            $last = $i + $m - 1;
        }
    }
    return ($arr[$last] -
            $arr[$first]);
}
```

```
// Driver Code
$arr = array(12, 4, 7, 9, 2, 23,
            25, 41, 30, 40, 28,
            42, 30, 44, 48, 43, 50);

$m = 7; // Number of students
$n = sizeof($arr);
echo "Minimum difference is ",
     findMinDiff($arr, $n, $m);

// This code is contributed by ajit
?>
```

Output:

```
Minimum difference is 10
```

Time Complexity : $O(n \log n)$ as we apply sorting before subarray search.

Improved By : [jit_t](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/chocolate-distribution-problem/>

Chapter 53

Choose k array elements such that difference of maximum and minimum is minimized

Choose k array elements such that difference of maximum and minimum is minimized -
GeeksforGeeks

Given an array of **n** integers and a positive number **k**. We are allowed to take any **k** integers from the given array. The task is to find the minimum possible value of the difference between maximum and minimum of K numbers.

Examples:

```
Input : arr[] = {10, 100, 300, 200, 1000, 20, 30}
        k = 3
Output : 20
20 is the minimum possible difference between any
maximum and minimum of any k numbers.
Given k = 3, we get the result 20 by selecting
integers {10, 20, 30}.
max(10, 20, 30) - max(10, 20, 30) = 30 - 10 = 20.
```

```
Input : arr[] = {1, 2, 3, 4, 10, 20, 30, 40,
                  100, 200}.
        k = 4
Output : 3
```

The idea is to sort the array and choose k continuous integers. Why continuous? let the chosen k integers be arr[0], arr[1],..arr[r], arr[r+x]..., arr[k-1], all in increasing order but not continuous in the sorted array. This means there exists an integer p which lies between

arr[r] and arr[r+x].. So if p is included and arr[0] is removed, then the new difference will be arr[r] – arr[1] whereas old difference was arr[r] – arr[0]. And we know arr[0] <= arr[1] <= .. <= arr[k-1] so minimum difference reduces or remain same. If we perform same procedure for other p like number, we get the minimum difference.

Algorithm to solve the problem:

1. Sort the Array.
2. Calculate the maximum(k numbers) – minimum(k numbers) for each group of k consecutive integers.
3. Return minimum of all values obtained in step 2.

Below is the implementation of above idea :

C++

```
// C++ program to find minimum difference of maximum
// and minimum of K number.
#include<bits/stdc++.h>
using namespace std;

// Return minimum difference of maximum and minimum
// of k elements of arr[0..n-1].
int minDiff(int arr[], int n, int k)
{
    int result = INT_MAX;

    // Sorting the array.
    sort(arr, arr + n);

    // Find minimum value among all K size subarray.
    for (int i=0; i<=n-k; i++)
        result = min(result, arr[i+k-1] - arr[i]);

    return result;
}

// Driven Program
int main()
{
    int arr[] = {10, 100, 300, 200, 1000, 20, 30};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;

    cout << minDiff(arr, n, k) << endl;
    return 0;
}
```

Java

```
// Java program to find minimum difference
// of maximum and minimum of K number.
import java.util.*;

class GFG {

    // Return minimum difference of
    // maximum and minimum of k
    // elements of arr[0..n-1].
    static int minDiff(int arr[], int n, int k) {
        int result = Integer.MAX_VALUE;

        // Sorting the array.
        Arrays.sort(arr);

        // Find minimum value among
        // all K size subarray.
        for (int i = 0; i <= n - k; i++)
            result = Math.min(result, arr[i + k - 1] - arr[i]);

        return result;
    }

    // Driver code
    public static void main(String[] args) {
        int arr[] = {10, 100, 300, 200, 1000, 20, 30};
        int n = arr.length;
        int k = 3;

        System.out.println(minDiff(arr, n, k));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python program to find minimum
# difference of maximum
# and minimum of K number.

# Return minimum difference
# of maximum and minimum
# of k elements of arr[0..n-1].
def minDiff(arr,n,k):
```

```
result = +2147483647

# Sorting the array.
arr.sort()

# Find minimum value among
# all K size subarray.
for i in range(n-k+1):
    result = int(min(result, arr[i+k-1] - arr[i]))

return result

# Driver code

arr= [10, 100, 300, 200, 1000, 20, 30]
n =len(arr)
k = 3

print(minDiff(arr, n, k))

# This code is contributed
# by Anant Agarwal.
```

C#

```
// C# program to find minimum
// difference of maximum and
// minimum of K number.
using System;

class GFG {

    // Return minimum difference of
    // maximum and minimum of k
    // elements of arr[0..n - 1].
    static int minDiff(int []arr, int n,
                      int k)
    {
        int result = int.MaxValue;

        // Sorting the array.
        Array.Sort(arr);

        // Find minimum value among
        // all K size subarray.
        for (int i = 0; i <= n - k; i++)
            result = Math.Min(result, arr[i + k - 1] - arr[i]);
    }
}
```

```
        return result;
    }

// Driver code
public static void Main() {
    int []arr = {10, 100, 300, 200, 1000, 20, 30};
    int n = arr.Length;
    int k = 3;

    Console.WriteLine(minDiff(arr, n, k));
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find minimum
// difference of maximum and
// minimum of K number.

// Return minimum difference
// of maximum and minimum
// of k elements of arr[0..n-1].
function minDiff($arr, $n, $k)
{
    $INT_MAX = 2147483647;
    $result = $INT_MAX ;

    // Sorting the array.
    sort($arr , $n);
    sort($arr);

    // Find minimum value among
    // all K size subarray.
    for ($i = 0; $i <= $n - $k; $i++)
        $result = min($result, $arr[$i + $k - 1] -
                      $arr[$i]);
    return $result;
}

// Driver Code
$arr = array(10, 100, 300, 200, 1000, 20, 30);
$n = sizeof($arr);
$k = 3;
echo minDiff($arr, $n, $k);
```

```
// This code is contributed by nitin mittal.  
?>
```

Output:

20

Time Complexity: $O(n \log n)$.

Improved By : [vt_m](#), nitin mittal

Source

<https://www.geeksforgeeks.org/k-numbers-difference-maximum-minimum-k-number-minimized/>

Chapter 54

Circle Sort

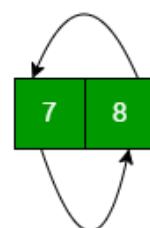
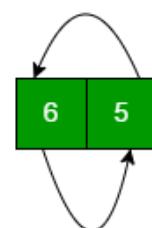
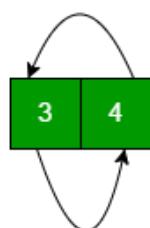
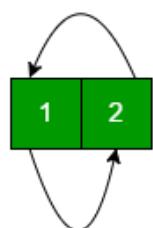
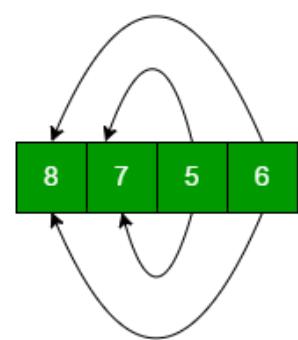
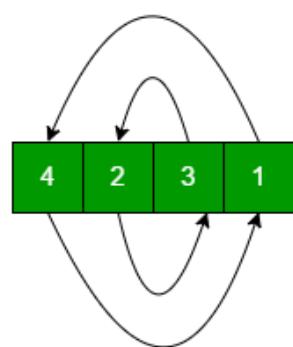
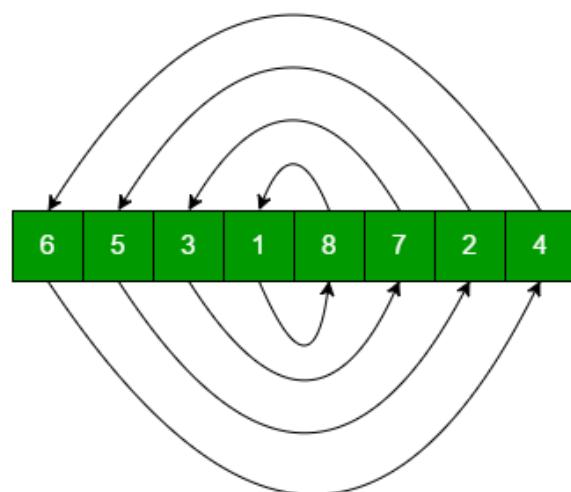
Circle Sort - GeeksforGeeks

Circle sort algorithm can be visualized by drawing concentric circles on an array of integers. The elements of the array lying on the same circle diametrically opposite to each other are compared and if found in the wrong order they are swapped. This goes on in a recursive fashion in which the array is divided into sub-arrays on which the above process is repeated until we get pairs of sorted elements which when put together form a sorted array.

In short below two steps are repeated while there are swap operations involved in the steps.

- Compare the first element to the last element, then the second element to the second last element, etc.
- Then split the array in two and recurse until there is only one single element in the array.

It can be better explained by the image below.



Below is the implementation of above algorithm.

```
// CPP implementation of Circle Sort
#include<bits/stdc++.h>
using namespace std;

// Function to perform circular swaps recursively
// This function returns true if there was a swap
// operation performed.
bool circleSortRec(int a[], int low, int high)
{
    bool swapped = false;

    // base case
    if (low == high)
        return false;

    // storing the upper and lower bounds
    // of list to be used later in the
    // recursive case
    int lo = low, hi = high;

    while (lo < hi)
    {
        // swaps the pair of elements
        // if true
        if (a[lo] > a[hi])
        {
            swap(a[lo], a[hi]);
            swapped = true;
        }
        lo++;
        hi--;
    }

    // special case arises only for list
    // of odd size
    if (lo == hi)
        if (a[lo] > a[hi + 1])
    {
        swap(a[low], a[hi+1]);
        swapped = true;
    }

    // recursive case to check the
    // traverse lists as sub lists
    int mid = (high - low) / 2;
    bool firstHalf = circleSortRec(a, low, low+mid);
```

```
bool secondHalf = circleSortRec(a, low+mid+1, high);

return swapped || firstHalf || secondHalf;
}

// This function mainly calls circleSortRec
void circleSort(int a[], int n)
{
    // Keep calling circleSortRec while
    // there is a swap operation.
    while (circleSortRec(a, 0, n-1))
    {
        ;
    }
}

// Driver program
int main()
{
    int a[] = {7, 5, 3, 1, 2, 4, 6, 8};
    int n = sizeof(a)/sizeof(a[0]);

    printf("\nUnsorted : ");
    for (int i=0; i<n; i++)
        cout << a[i] << " ";

    circleSort(a, n);

    printf("\nSorted : ");
    for (int i=0; i<n; i++)
        cout << a[i] << " ";

    return 0;
}
```

Output :

```
Unsorted : [6, 5, 3, 1, 8, 7, 2, 4]
Sorted : [1, 2, 3, 4, 5, 6, 7, 8]
```

References :

[SourceForge](#)

Source

<https://www.geeksforgeeks.org/circle-sort/>

Chapter 55

Closest numbers from a list of unsorted integers

Closest numbers from a list of unsorted integers - GeeksforGeeks

Given a list of distinct unsorted integers, find the pair of elements that have the smallest absolute difference between them? If there are multiple pairs, find them all.

Examples:

```
Input : arr[] = {10, 50, 12, 100}
Output : (10, 12)
The closest elements are 10 and 12

Input : arr[] = {5, 4, 3, 2}
Output : (2, 3), (3, 4), (4, 5)
```

This problem is mainly an extension of [Find minimum difference between any two elements](#).

1. Sort the given array.
2. Find minimum difference of all pairs in linear time in sorted array.
3. Traverse sorted array one more time to print all pairs with minimum difference obtained in step 2.

C++

```
// CPP program to find minimum difference
// an unsorted array.
#include<bits/stdc++.h>
using namespace std;
```

```
// Returns minimum difference between any
// two pair in arr[0..n-1]
void printMinDiffPairs(int arr[], int n)
{
    if (n <= 1)
        return;

    // Sort array elements
    sort(arr, arr+n);

    // Compare differences of adjacent
    // pairs to find the minimum difference.
    int minDiff = arr[1] - arr[0];
    for (int i = 2 ; i < n ; i++)
        minDiff = min(minDiff, arr[i] - arr[i-1]);

    // Traverse array again and print all pairs
    // with difference as minDiff.
    for (int i = 1; i < n; i++)
        if ((arr[i] - arr[i-1]) == minDiff)
            cout << "(" << arr[i-1] << ", "
                << arr[i] << "), ";
}

// Driver code
int main()
{
    int arr[] = {5, 3, 2, 4, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    printMinDiffPairs(arr, n);
    return 0;
}
```

Java

```
// Java program to find minimum
// difference an unsorted array.
import java.util.*;

class GFG
{

    // Returns minimum difference between
    // any two pair in arr[0..n-1]
    static void printMinDiffPairs(int arr[], int n)
    {
        if (n <= 1)
```

```
        return;

    // Sort array elements
    Arrays.sort(arr);

    // Compare differences of adjacent
    // pairs to find the minimum difference.
    int minDiff = arr[1] - arr[0];
    for (int i = 2; i < n; i++)
        minDiff = Math.min(minDiff, arr[i] - arr[i-1]);

    // Traverse array again and print all pairs
    // with difference as minDiff.
    for ( int i = 1; i < n; i++)
    {
        if ((arr[i] - arr[i-1]) == minDiff)
        {
            System.out.print("(" + arr[i-1] + " , "
                            + arr[i] + ")");
        }
    }
}

// Driver code
public static void main (String[] args)
{
    int arr[] = {5, 3, 2, 4, 1};
    int n = arr.length;
    printMinDiffPairs(arr, n);
}
```

// This code is contributed by Ansu Kumari

Python3

```
# Python3 program to find minimum
# difference in an unsorted array.

# Returns minimum difference between
# any two pair in arr[0..n-1]
def printMinDiffPairs(arr , n):
    if n <= 1: return

    # Sort array elements
    arr.sort()

    # Compare differences of adjacent
```

```
# pairs to find the minimum difference.
minDiff = arr[1] - arr[0]
for i in range(2 , n):
    minDiff = min(minDiff, arr[i] - arr[i-1])

# Traverse array again and print all
# pairs with difference as minDiff.
for i in range(1 , n):
    if (arr[i] - arr[i-1]) == minDiff:
        print( "(" + str(arr[i-1]) + ", "
               + str(arr[i]) + ")"), ", end = '')

# Driver code
arr = [5, 3, 2, 4, 1]
n = len(arr)
printMinDiffPairs(arr , n)

# This code is contributed by Ansu Kumari
```

C#

```
// C# program to find minimum
// difference an unsorted array.
using System;

class GFG
{

    // Returns minimum difference between
    // any two pair in arr[0..n-1]
    static void printMinDiffPairs(int []arr, int n)
    {
        if (n <= 1)
            return;

        // Sort array elements
        Array.Sort(arr);

        // Compare differences of adjacent
        // pairs to find the minimum difference.
        int minDiff = arr[1] - arr[0];
        for (int i = 2; i < n; i++)
            minDiff = Math.Min(minDiff, arr[i] - arr[i-1]);

        // Traverse array again and print all pairs
        // with difference as minDiff.
        for ( int i = 1; i < n; i++)
        {
```

```
        if ((arr[i] - arr[i-1]) == minDiff)
    {
        Console.WriteLine(" (" + arr[i-1] + ", "
                           + arr[i] + "), " );
    }
}

// Driver code
public static void Main ()
{
    int []arr = {5, 3, 2, 4, 1};
    int n = arr.Length;
    printMinDiffPairs(arr, n);
}
}

// This code is contributed by vt_m
```

PHP

```
<?php
//PHP program to find minimum difference
// an unsorted array.

// Returns minimum difference between any
// two pair in arr[0..n-1]
function printMinDiffPairs($arr, $n)
{
    if ($n <= 1)
        return;

    // Sort array elements
    sort($arr);

    // Compare differences of adjacent
    // pairs to find the minimum
    // difference.
    $minDiff = $arr[1] - $arr[0];

    for ($i = 2 ; $i < $n ; $i++)
        $minDiff = min($minDiff, $arr[$i]
                      - $arr[$i-1]);

    // Traverse array again and print all
    // pairs with difference as minDiff.
    for ($i = 1; $i < $n; $i++)
        if (($arr[$i] - $arr[$i-1]) ==
```

```
        $minDiff)
echo "(" , $arr[$i-1] , ", ",
       $arr[$i] , ")");
}

// Driver code
$arr = array(5, 3, 2, 4, 1);
$n = sizeof($arr);
printMinDiffPairs($arr, $n);

// This code is contributed by ajit.
?>
```

Output:

```
(1, 2), (2, 3), (3, 4), (4, 5),
```

Does above program handle duplicates?

The cases like {x, x, x} are not handled by above program. For this case, the expected output (x, x), (x, x), (x, x), but above program prints (x, x), (x, x)

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/closest-numbers-list-unsorted-integers/>

Chapter 56

Closest product pair in an array

Closest product pair in an array - GeeksforGeeks

Given an array of non negative integers and a number x, find a pair in array whose product is closest to x.

Examples :

```
Input : arr[] = [2, 3, 5, 9]
        x = 47
Output : {5, 9}
```

```
Input : arr[] = [2, 3, 5, 9]
        x = 8
Output : {2, 5}
```

Method 1

A simple solution is to consider every pair and keep track of closest pair (absolute difference between pair product and x is minimum). Finally print the closest pair. Time complexity

of this solution is $O(n^2)$

Method 2 $O(n \log n)$

1. Sort the array
2. Initialize a variable diff as infinite (Diff is used to store the difference between pair and x). We need to find the minimum diff.
3. Traverse the array and for each i, do the following :
 - Find the lower bound for $x/arr[i]$ in the sub array on right of $arr[i]$, i.e., in sub array $arr[i+1..n-1]$. Let it be denoted by l.

- Find the upper bound for $x/arr[i]$ in the sub array on right of $arr[i]$, i.e., in sub array $arr[i+1..n-1]$. Let it be denoted be u .
- If $\min(\text{abs}((arr[i] * l) - x), \text{abs}((arr[i] * u) - x)) < \text{diff}$ then update diff and result

Method 3 O(n for sorted)

An efficient solution can find the pair in $O(n)$ time. Following is detailed algorithm.

- 1) Initialize a variable diff as infinite
(Diff is used to store the difference between pair and x). We need to find the minimum diff.
- 2) Initialize two index variables l and r in the given sorted array.
 - (a) Initialize first to the leftmost index:
 $l = 0$
 - (b) Initialize second the rightmost index:
 $r = n-1$
- 3) Loop while $l < r$.
 - (a) If $\text{abs}((arr[l] * arr[r]) - x) < \text{diff}$
then update diff and result
 - (b) Else if $((arr[l] * arr[r]) < x)$ then
 $l++$
 - (c) Else $r--$

Following is the implementation of above algorithm.

C++

```
// Simple C++ program to find the pair with
// product closest to a given no.
#include <bits/stdc++.h>
using namespace std;

// Prints the pair with product closest to x
void printClosest(int arr[], int n, int x)
{
    int res_l, res_r; // To store indexes of result pair

    // Initialize left and right indexes and
    // difference between pair product and x
    int l = 0, r = n - 1, diff = INT_MAX;

    // While there are elements between l and r
    while (r > l) {

        // Check if this pair is closer than
        // the closest pair so far
```

```
if (abs(arr[l] * arr[r] - x) < diff) {
    res_l = l;
    res_r = r;
    diff = abs(arr[l] * arr[r] - x);
}

// If this pair has more product,
// move to smaller values.
if (arr[l] * arr[r] > x)
    r--;

else // Move to larger values
    l++;
}

cout << " The closest pair is "
    << arr[res_l] << " and " << arr[res_r];
}

// Driver program to test above functions
int main()
{
    int arr[] = { 2, 3, 5, 9 }, x = 8;
    int n = sizeof(arr) / sizeof(arr[0]);
    printClosest(arr, n, x);
    return 0;
}
```

Java

```
// Simple Java program to find
// the pair with product closest
// to a given no.
import java.io.*;

class GFG
{
// Prints the pair with
// product closest to x
static void printClosest(int arr[],
                        int n, int x)
{
    // To store indexes of result pair
    int res_l = 0, res_r = 0;

    // Initialize left and right
    // indexes and difference
    // between pair product and x
```

```
int l = 0, r = n - 1, diff = Integer.MAX_VALUE;

// While there are
// elements between l and r
while (r > l)
{
    // Check if this pair is closer
    // than the closest pair so far
    if (Math.abs(arr[l] * arr[r] - x) < diff)
    {
        res_l = l;
        res_r = r;
        diff = Math.abs(arr[l] * arr[r] - x);
    }

    // If this pair has more product,
    // move to smaller values.
    if (arr[l] * arr[r] > x)
        r--;
    else
        l++;
}

System.out.print("The closest pair is ");
System.out.print (arr[res_l] +
                  " and " +
                  arr[res_r]);
}

// Driver Code
public static void main (String[] args)
{
    int arr[] = {2, 3, 5, 9};
    int x = 8;
    int n = arr.length;
    printClosest(arr, n, x);
}
}

// This code is contributed by anuj_67.
```

Python3

```
# Simple Python3 program to find
# the pair with product closest
```

```
# to a given no.
import sys

# Prints the pair with
# product closest to x
def printClosest(arr, n, x):

    # To store indexes
    # of result pair
    res_l = 0;
    res_r = 0;

    # Initialize left and right
    # indexes and difference
    # between pair product and x
    l = 0;
    r = n - 1;
    diff = sys.maxsize;

    # While there are elements
    # between l and r
    while (r > l):

        # Check if this pair is
        # closer than the closest
        # pair so far
        if (abs(arr[l] *
               arr[r] - x) < diff):
            res_l = l;
            res_r = r;
            diff = abs(arr[l] *
                       arr[r] - x);

        # If this pair has more
        # product, move to smaller
        # values.
        if (arr[l] * arr[r] > x):
            r = r - 1;

        # Move to larger values
        else:
            l = l + 1;

    print("The closest pair is", arr[res_l] ,
          "and", arr[res_r]);

# Driver Code
arr = [2, 3, 5, 9];
```

```
x = 8;
n = len(arr);
printClosest(arr, n, x);

# This code is contributed
# by rahul
```

C#

```
// Simple C# program to find
// the pair with product closest
// to a given no.
using System;

class GFG
{
    // Prints the pair with
    // product closest to x
    static void printClosest(int []arr,
                            int n, int x)
    {
        // To store indexes of result pair
        int res_l = 0, res_r = 0;

        // Initialize left and right
        // indexes and difference
        // between pair product and x
        int l = 0, r = n - 1,
            diff = int.MaxValue;

        // While there are
        // elements between l and r
        while (r > l)
        {

            // Check if this pair is closer
            // than the closest pair so far
            if (Math.Abs(arr[l] *
                        arr[r] - x) < diff)
            {
                res_l = l;
                res_r = r;
                diff = Math.Abs(arr[l] *
                                arr[r] - x);
            }

            // If this pair has more product,
            // move to smaller values.
        }
    }
}
```

```
        if (arr[l] * arr[r] > x)
            r--;
        else
            l++;
    }

    Console.WriteLine("The closest pair is ");
    Console.Write(arr[res_l] +
                  " and " +
                  arr[res_r]);
}

// Driver Code
public static void Main ()
{
    int []arr = {2, 3, 5, 9};
    int x = 8;
    int n = arr.Length;
    printClosest(arr, n, x);
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// Simple PHP program to find
// the pair with product closest
// to a given no.

// Prints the pair with
// product closest to x
function printClosest($arr, $n, $x)
{
    // To store indexes
    // of result pair
    $res_l; $res_r;

    // Initialize left and right
    // indexes and difference
    // between pair product and x
    $l = 0; $r = $n - 1; $diff = PHP_INT_MAX;

    // While there are elements
    // between l and r
```

```
while ($r > $l)
{
    // Check if this pair is
    // closer than the closest
    // pair so far
    if (abs($arr[$l] *
            $arr[$r] - $x) < $diff)
    {
        $res_l = $l;
        $res_r = $r;
        $diff = abs($arr[$l] *
                    $arr[$r] - $x);
    }

    // If this pair has more
    // product, move to smaller
    // values.
    if ($arr[$l] * $arr[$r] > $x)
        $r--;

    // Move to larger values
    else
        $l++;
}

echo " The closest pair is " ,
      $arr[$res_l] , " and " ,
      $arr[$res_r];
}

// Driver Code
$arr = array(2, 3, 5, 9);
$x = 8;
$n = count($arr);
printClosest($arr, $n, $x);

// This code is contributed by anuj_67.
?>
```

Output :

The closest pair is 2 and 5

Improved By : [vt_m](#), [mithunkumarmnnit321](#)

Source

<https://www.geeksforgeeks.org/closest-product-pair-array/>

Chapter 57

Cocktail Sort

Cocktail Sort - GeeksforGeeks

Cocktail Sort is a variation of [Bubble sort](#). The Bubble sort algorithm always traverses elements from left and moves the largest element to its correct position in first iteration and second largest in second iteration and so on. Cocktail Sort traverses through a given array in both directions alternatively.

Algorithm:

Each iteration of the algorithm is broken up into 2 stages:

1. The first stage loops through the array from left to right, just like the Bubble Sort. During the loop, adjacent items are compared and if value on the left is greater than the value on the right, then values are swapped. At the end of first iteration, largest number will reside at the end of the array.
2. The second stage loops through the array in opposite direction- starting from the item just before the most recently sorted item, and moving back to the start of the array. Here also, adjacent items are compared and are swapped if required.

Example : Let us consider an example array (5 1 4 2 8 0 2)

First Forward Pass:

(**5** 1 4 2 8 0 2) ? (**1** 5 4 2 8 0 2), Swap since $5 > 1$
(**1** 5 4 2 8 0 2) ? (**1** 4 **5** 2 8 0 2), Swap since $5 > 4$
(**1** 4 **5** **2** 8 0 2) ? (**1** 4 **2** **5** 8 0 2), Swap since $5 > 2$
(**1** 4 2 **5** **8** 0 2) ? (**1** 4 2 **5** **8** 0 2)
(**1** 4 2 5 **8** **0** 2) ? (**1** 4 2 5 **0** **8** 2), Swap since $8 > 0$
(**1** 4 2 5 0 **8** **2**) ? (**1** 4 2 5 0 **2** **8**), Swap since $8 > 2$

After first forward pass, greatest element of the array will be present at the last index of array.

First Backward Pass:

(1 4 2 5 **0** **2** 8) ? (1 4 2 5 **0** **2** 8)

(1 4 2 5 0 2 8) ? (1 4 2 0 5 2 8), Swap since 5 > 0
(1 4 2 0 5 2 8) ? (1 4 0 2 5 2 8), Swap since 2 > 0
(1 4 0 2 5 2 8) ? (1 0 4 2 5 2 8), Swap since 4 > 0
(1 0 4 2 5 2 8) ? (0 1 4 2 5 2 8), Swap since 1 > 0

After first backward pass, smallest element of the array will be present at the first index of the array.

Second Forward Pass:

(0 1 4 2 5 2 8) ? (0 1 4 2 5 2 8)
(0 1 4 2 5 2 8) ? (0 1 2 4 5 2 8), Swap since 4 > 2
(0 1 2 4 5 2 8) ? (0 1 2 4 5 2 8)
(0 1 2 4 5 2 8) ? (0 1 2 4 2 5 8), Swap since 5 > 2

Second Backward Pass:

(0 1 2 4 2 5 8) ? (0 1 2 2 4 5 8), Swap since 4 > 2

Now, the array is already sorted, but our algorithm doesn't know if it is completed. The algorithm needs to complete this whole pass without any **swap** to know it is sorted.

(0 1 2 2 4 5 8) ? (0 1 2 2 4 5 8)
(0 1 2 2 4 5 8) ? (0 1 2 2 4 5 8)

Below is the implementation of the above algorithm :

C++

```
// C++ implementation of Cocktail Sort
#include <bits/stdc++.h>
using namespace std;

// Sorts arrar a[0..n-1] using Cocktail sort
void CocktailSort(int a[], int n)
{
    bool swapped = true;
    int start = 0;
    int end = n - 1;

    while (swapped) {
        // reset the swapped flag on entering
        // the loop, because it might be true from
        // a previous iteration.
        swapped = false;

        // loop from left to right same as
        // the bubble sort
        for (int i = start; i < end; ++i) {
            if (a[i] > a[i + 1]) {
                swap(a[i], a[i + 1]);
                swapped = true;
            }
        }
    }
}
```

```
// if nothing moved, then array is sorted.
if (!swapped)
    break;

// otherwise, reset the swapped flag so that it
// can be used in the next stage
swapped = false;

// move the end point back by one, because
// item at the end is in its rightful spot
--end;

// from right to left, doing the
// same comparison as in the previous stage
for (int i = end - 1; i >= start; --i) {
    if (a[i] > a[i + 1]) {
        swap(a[i], a[i + 1]);
        swapped = true;
    }
}

// increase the starting point, because
// the last stage would have moved the next
// smallest number to its rightful spot.
++start;
}

/* Prints the array */
void printArray(int a[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}

// Driver code
int main()
{
    int arr[] = { 5, 1, 4, 2, 8, 0, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    CocktailSort(a, n);
    printf("Sorted array :\n");
    printArray(a, n);
    return 0;
}
```

Java

```
// Java program for implementation of Cocktail Sort
public class CocktailSort {
    void cocktailSort(int a[])
    {
        boolean swapped = true;
        int start = 0;
        int end = a.length;

        while (swapped == true) {
            // reset the swapped flag on entering the
            // loop, because it might be true from a
            // previous iteration.
            swapped = false;

            // loop from bottom to top same as
            // the bubble sort
            for (int i = start; i < end - 1; ++i) {
                if (a[i] > a[i + 1]) {
                    int temp = a[i];
                    a[i] = a[i + 1];
                    a[i + 1] = temp;
                    swapped = true;
                }
            }

            // if nothing moved, then array is sorted.
            if (swapped == false)
                break;

            // otherwise, reset the swapped flag so that it
            // can be used in the next stage
            swapped = false;

            // move the end point back by one, because
            // item at the end is in its rightful spot
            end = end - 1;

            // from top to bottom, doing the
            // same comparison as in the previous stage
            for (int i = end - 1; i >= start; i--) {
                if (a[i] > a[i + 1]) {
                    int temp = a[i];
                    a[i] = a[i + 1];
                    a[i + 1] = temp;
                    swapped = true;
                }
            }
        }
    }
}
```

```
}

    // increase the starting point, because
    // the last stage would have moved the next
    // smallest number to its rightful spot.
    start = start + 1;
}
}

/* Prints the array */
void printArray(int a[])
{
    int n = a.length;
    for (int i = 0; i < n; i++)
        System.out.print(a[i] + " ");
    System.out.println();
}

// Driver method
public static void main(String[] args)
{
    CocktailSort ob = new CocktailSort();
    int a[] = { 5, 1, 4, 2, 8, 0, 2 };
    ob.cocktailSort(a);
    System.out.println("Sorted array");
    ob.printArray(a);
}
}
```

Python

```
# Python program for implementation of Cocktail Sort

def cocktailSort(a):
    n = len(a)
    swapped = True
    start = 0
    end = n-1
    while (swapped == True):

        # reset the swapped flag on entering the loop,
        # because it might be true from a previous
        # iteration.
        swapped = False

        # loop from left to right same as the bubble
        # sort
        for i in range (start, end):
```

```

if (a[i] > a[i + 1]) :
    a[i], a[i + 1] = a[i + 1], a[i]
    swapped = True

# if nothing moved, then array is sorted.
if (swapped == False):
    break

# otherwise, reset the swapped flag so that it
# can be used in the next stage
swapped = False

# move the end point back by one, because
# item at the end is in its rightful spot
end = end-1

# from right to left, doing the same
# comparison as in the previous stage
for i in range(end-1, start-1, -1):
    if (a[i] > a[i + 1]):
        a[i], a[i + 1] = a[i + 1], a[i]
        swapped = True

# increase the starting point, because
# the last stage would have moved the next
# smallest number to its rightful spot.
start = start + 1

# Driver code to test above
a = [5, 1, 4, 2, 8, 0, 2]
cocktailSort(a)
print("Sorted array is:")
for i in range(len(a)):
    print ("% d" % a[i]),

```

C#

```

// C# program for implementation of Cocktail Sort
using System;

class GFG {

    static void cocktailSort(int[] a)
    {
        bool swapped = true;
        int start = 0;
        int end = a.Length;

```

```
while (swapped == true) {

    // reset the swapped flag on entering the
    // loop, because it might be true from a
    // previous iteration.
    swapped = false;

    // loop from bottom to top same as
    // the bubble sort
    for (int i = start; i < end - 1; ++i) {
        if (a[i] > a[i + 1]) {
            int temp = a[i];
            a[i] = a[i + 1];
            a[i + 1] = temp;
            swapped = true;
        }
    }

    // if nothing moved, then array is sorted.
    if (swapped == false)
        break;

    // otherwise, reset the swapped flag so that it
    // can be used in the next stage
    swapped = false;

    // move the end point back by one, because
    // item at the end is in its rightful spot
    end = end - 1;

    // from top to bottom, doing the
    // same comparison as in the previous stage
    for (int i = end - 1; i >= start; i--) {
        if (a[i] > a[i + 1]) {
            int temp = a[i];
            a[i] = a[i + 1];
            a[i + 1] = temp;
            swapped = true;
        }
    }

    // increase the starting point, because
    // the last stage would have moved the next
    // smallest number to its rightful spot.
    start = start + 1;
}

}
```

```
/* Prints the array */
static void printArray(int[] a)
{
    int n = a.Length;
    for (int i = 0; i < n; i++)
        Console.Write(a[i] + " ");
    Console.WriteLine();
}

// Driver method
public static void Main()
{
    int[] a = { 5, 1, 4, 2, 8, 0, 2 };
    cocktailSort(a);
    Console.WriteLine("Sorted array ");
    printArray(a);
}
}

// This code is contributed by Sam007
```

Output:

```
Sorted array is:
0 1 2 2 4 5 8
```

Worst and Average Case Time Complexity: $O(n^2)$.

Best Case Time Complexity: $O(n)$. Best case occurs when array is already sorted.

Auxiliary Space: $O(1)$

Sorting In Place: Yes

Stable: Yes

Comparison with Bubble Sort:

Time complexities are same, but Cocktail performs better than Bubble Sort. Typically cocktail sort is less than two times faster than bubble sort. Consider the example (2, 3, 4, 5, 1). Bubble sort requires four traversals of array for this example, while Cocktail sort requires only two traversals. (Source [Wiki](#))

References:

- https://en.wikipedia.org/wiki/Cocktail_shaker_sort
- http://will.thimbleby.net/algorithms/doku.php?id=cocktail_sort
- <http://www.programming-algorithms.net/article/40270/Shaker-sort>

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/cocktail-sort/>

Chapter 58

Comb Sort

Comb Sort - GeeksforGeeks

Comb Sort is mainly an improvement over Bubble Sort. Bubble sort always compares adjacent values. So all [inversions](#) are removed one by one. Comb Sort improves on Bubble Sort by using gap of size more than 1. The gap starts with a large value and shrinks by a factor of 1.3 in every iteration until it reaches the value 1. Thus Comb Sort removes more than one [inversion counts](#) with one swap and performs better than Bubble Sort.

The shrink factor has been empirically found to be 1.3 (by testing Combsort on over 200,000 random lists) [Source: [Wiki](#)]

Although, it works better than Bubble Sort on average, worst case remains $O(n^2)$.

Below is the implementation.

C++

```
// C++ implementation of Comb Sort
#include<bits/stdc++.h>
using namespace std;

// To find gap between elements
int getNextGap(int gap)
{
    // Shrink gap by Shrink factor
    gap = (gap*10)/13;

    if (gap < 1)
        return 1;
    return gap;
}

// Function to sort a[0..n-1] using Comb Sort
void combSort(int a[], int n)
```

```
{
    // Initialize gap
    int gap = n;

    // Initialize swapped as true to make sure that
    // loop runs
    bool swapped = true;

    // Keep running while gap is more than 1 and last
    // iteration caused a swap
    while (gap != 1 || swapped == true)
    {
        // Find next gap
        gap = getNextGap(gap);

        // Initialize swapped as false so that we can
        // check if swap happened or not
        swapped = false;

        // Compare all elements with current gap
        for (int i=0; i<n-gap; i++)
        {
            if (a[i] > a[i+gap])
            {
                swap(a[i], a[i+gap]);
                swapped = true;
            }
        }
    }

    // Driver program
    int main()
    {
        int a[] = {8, 4, 1, 56, 3, -44, 23, -6, 28, 0};
        int n = sizeof(a)/sizeof(a[0]);

        combSort(a, n);

        printf("Sorted array: \n");
        for (int i=0; i<n; i++)
            printf("%d ", a[i]);

        return 0;
    }
}
```

Java

```
// Java program for implementation of Comb Sort
class CombSort
{
    // To find gap between elements
    int getNextGap(int gap)
    {
        // Shrink gap by Shrink factor
        gap = (gap*10)/13;
        if (gap < 1)
            return 1;
        return gap;
    }

    // Function to sort arr[] using Comb Sort
    void sort(int arr[])
    {
        int n = arr.length;

        // initialize gap
        int gap = n;

        // Initialize swapped as true to make sure that
        // loop runs
        boolean swapped = true;

        // Keep running while gap is more than 1 and last
        // iteration caused a swap
        while (gap != 1 || swapped == true)
        {
            // Find next gap
            gap = getNextGap(gap);

            // Initialize swapped as false so that we can
            // check if swap happened or not
            swapped = false;

            // Compare all elements with current gap
            for (int i=0; i<n-gap; i++)
            {
                if (arr[i] > arr[i+gap])
                {
                    // Swap arr[i] and arr[i+gap]
                    int temp = arr[i];
                    arr[i] = arr[i+gap];
                    arr[i+gap] = temp;

                    // Set swapped
                    swapped = true;
                }
            }
        }
    }
}
```

```
        }
    }
}

// Driver method
public static void main(String args[])
{
    CombSort ob = new CombSort();
    int arr[] = {8, 4, 1, 56, 3, -44, 23, -6, 28, 0};
    ob.sort(arr);

    System.out.println("sorted array");
    for (int i=0; i<arr.length; ++i)
        System.out.print(arr[i] + " ");

}
/* This code is contributed by Rajat Mishra */
```

Python

```
# Python program for implementation of CombSort

# To find next gap from current
def getNextGap(gap):

    # Shrink gap by Shrink factor
    gap = (gap * 10)/13
    if gap < 1:
        return 1
    return gap

# Function to sort arr[] using Comb Sort
def combSort(arr):
    n = len(arr)

    # Initialize gap
    gap = n

    # Initialize swapped as true to make sure that
    # loop runs
    swapped = True

    # Keep running while gap is more than 1 and last
    # iteration caused a swap
    while gap != 1 or swapped == 1:
```

```
# Find next gap
gap = getNextGap(gap)

# Initialize swapped as false so that we can
# check if swap happened or not
swapped = False

# Compare all elements with current gap
for i in range(0, n-gap):
    if arr[i] > arr[i + gap]:
        arr[i], arr[i + gap]=arr[i + gap], arr[i]
        swapped = True

# Driver code to test above
arr = [ 8, 4, 1, 3, -44, 23, -6, 28, 0]
combSort(arr)

print ("Sorted array:")
for i in range(len(arr)):
    print (arr[i]),

# This code is contributed by Mohit Kumra
```

C#

```
// C# program for implementation of Comb Sort
using System;

class GFG
{
    // To find gap between elements
    static int getNextGap(int gap)
    {
        // Shrink gap by Shrink factor
        gap = (gap*10)/13;
        if (gap < 1)
            return 1;
        return gap;
    }

    // Function to sort arr[] using Comb Sort
    static void sort(int []arr)
    {
        int n = arr.Length;

        // initialize gap
```

```
int gap = n;

// Initialize swapped as true to
// make sure that loop runs
bool swapped = true;

// Keep running while gap is more than
// 1 and last iteration caused a swap
while (gap != 1 || swapped == true)
{
    // Find next gap
    gap = getNextGap(gap);

    // Initialize swapped as false so that we can
    // check if swap happened or not
    swapped = false;

    // Compare all elements with current gap
    for (int i=0; i<n-gap; i++)
    {
        if (arr[i] > arr[i+gap])
        {
            // Swap arr[i] and arr[i+gap]
            int temp = arr[i];
            arr[i] = arr[i+gap];
            arr[i+gap] = temp;

            // Set swapped
            swapped = true;
        }
    }
}

// Driver method
public static void Main()
{
    int []arr = {8, 4, 1, 56, 3, -44, 23, -6, 28, 0};
    sort(arr);

    Console.WriteLine("sorted array");
    for (int i=0; i<arr.Length; ++i)
        Console.Write(arr[i] + " ");

    }
}

// This code is contributed by Sam007
```

Output :

Sorted array:
-44 -6 0 1 3 4 8 23 28 56

Illustration:

Let the array elements be

8, 4, 1, 56, 3, -44, 23, -6, 28, 0

Initially gap value = 10

After shrinking gap value => $10/1.3 = 7$;

8 4 1 56 3 -44 23 -6 28 0
-6 4 1 56 3 -44 23 8 28 0
-6 4 0 56 3 -44 23 8 28 1

New gap value => $7/1.3 = 5$;

-44 4 0 56 3 -6 23 8 28 1
-44 4 0 28 3 -6 23 8 56 1
-44 4 0 28 1 -6 23 8 56 3

New gap value => $5/1.3 = 3$;

-44 1 0 28 4 -6 23 8 56 3
-44 1 -6 28 4 0 23 8 56 3
-44 1 -6 23 4 0 28 8 56 3
-44 1 -6 23 4 0 3 8 56 28

New gap value => $3/1.3 = 2$;

-44 1 -6 0 4 23 3 8 56 28
-44 1 -6 0 3 23 4 8 56 28
-44 1 -6 0 3 8 4 23 56 28

New gap value => $2/1.3 = 1$;

-44 -6 1 0 3 8 4 23 56 28
-44 -6 0 1 3 8 4 23 56 28
-44 -6 0 1 3 4 8 23 56 28
-44 -6 0 1 3 4 8 23 28 56

no more swaps required (Array sorted)

Time Complexity : Worst case complexity of this algorithm is $O(n^2)$ and the Best Case complexity is $O(n)$.

Auxiliary Space : $O(1)$.

Snapshots:

Let the array elements be

| | | | | |
|---|---|---|----|---|
| 8 | 4 | 1 | 56 | 3 |
|---|---|---|----|---|

Gap value

10

Run N

1

Let the array elements be

| | | | | |
|---|---|---|----|---|
| 8 | 4 | 1 | 56 | 3 |
|---|---|---|----|---|

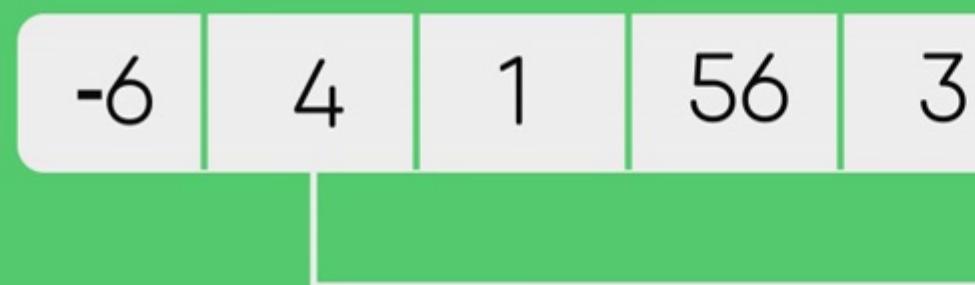
Gap value

$$10/1.3 = 7$$

Run N

2

Let the array elements be



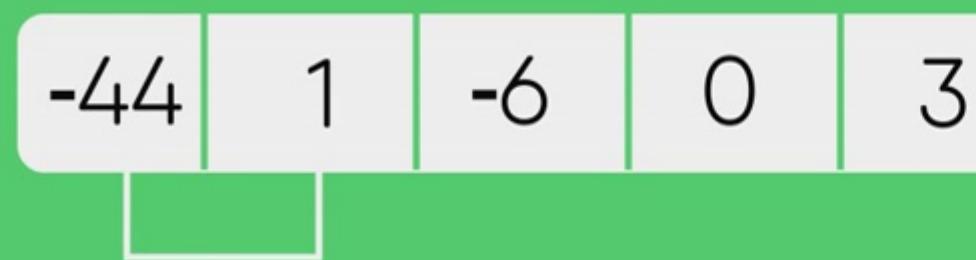
Gap value

$$10/1.3 = 7$$

Run N

2

Let the array elements be



Gap value

$$2/1.3 = 1$$

Run N

6

Let the array elements be



Gap value

$$2/1.3 = 1$$

Run N

6

Let the array elements be

| | | | | |
|-----|----|---|---|---|
| -44 | -6 | 0 | 1 | 3 |
|-----|----|---|---|---|

Array is now

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz

[Selection Sort](#), [Bubble Sort](#), [Insertion Sort](#), [Merge Sort](#), [Heap Sort](#), [QuickSort](#), [Radix Sort](#),
[Counting Sort](#), [Bucket Sort](#), [ShellSort](#), [Pigeonhole Sort](#)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/comb-sort/>

Chapter 59

Comparator function of qsort() in C

Comparator function of qsort() in C - GeeksforGeeks

Standard C library provides [qsort\(\)](#) that can be used for sorting an array. As the name suggests, the function uses QuickSort algorithm to sort the given array. Following is prototype of qsort()

```
void qsort (void* base, size_t num, size_t size,
            int (*comparator)(const void*,const void*));
```

The key point about qsort() is comparator function *comparator*. The comparator function takes two arguments and contains logic to decide their relative order in sorted output. The idea is to provide flexibility so that qsort() can be used for any type (including user defined types) and can be used to obtain any desired order (increasing, decreasing or any other).

The comparator function takes two pointers as arguments (both type-casted to const void*) and defines the order of the elements by returning (in a stable and transitive manner

```
int comparator(const void* p1, const void* p2);
Return value meaning
<0 The element pointed by p1 goes before the element pointed by p2
0 The element pointed by p1 is equivalent to the element pointed by p2
>0 The element pointed by p1 goes after the element pointed by p2
```

Source: <http://www.cplusplus.com/reference/cstdlib/qsort/>

For example, let there be an array of students where following is type of student.

```
struct Student
```

```
{  
    int age, marks;  
    char name[20];  
};
```

Lets say we need to sort the students based on marks in ascending order. The comparator function will look like:

```
int comparator(const void *p, const void *q)  
{  
    int l = ((struct Student *)p)->marks;  
    int r = ((struct Student *)q)->marks;  
    return (l - r);  
}
```

See following posts for more sample uses of `qsort()`.

[Given a sequence of words, print all anagrams together](#)

[Box Stacking Problem](#)

[Closest Pair of Points](#)

Following is an interesting problem that can be easily solved with the help of `qsort()` and comparator function.

Given an array of integers, sort it in such a way that the odd numbers appear first and the even numbers appear later. The odd numbers should be sorted in descending order and the even numbers should be sorted in ascending order.

The simple approach is to first modify the input array such that the even and odd numbers are segregated followed by applying some sorting algorithm on both parts(odd and even) separately.

However, there exists an interesting approach with a little modification in comparator function of Quick Sort. The idea is to write a comparator function that takes two addresses p and q as arguments. Let l and r be the number pointed by p and q. The function uses following logic:

- 1) If both (l and r) are odd, put the greater of two first.
- 2) If both (l and r) are even, put the smaller of two first.
- 3) If one of them is even and other is odd, put the odd number first.

Following is C implementation of the above approach.

```
#include <stdio.h>  
#include <stdlib.h>  
  
// This function is used in qsort to decide the relative order  
// of elements at addresses p and q.  
int comparator(const void *p, const void *q)  
{  
    // Get the values at given addresses  
    int l = *(const int *)p;
```

```
int r = *(const int *)q;

// both odd, put the greater of two first.
if ((l&1) && (r&1))
    return (r-l);

// both even, put the smaller of two first
if ( !(l&1) && !(r&1) )
    return (l-r);

// l is even, put r first
if ( !(l&1) )
    return 1;

// l is odd, put l first
return -1;
}

// A utility function to print an array
void printArr(int arr[], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        printf("%d ", arr[i]);
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 6, 5, 2, 3, 9, 4, 7, 8};

    int size = sizeof(arr) / sizeof(arr[0]);
    qsort((void*)arr, size, sizeof(arr[0]), comparator);

    printf("Output array is\n");
    printArr(arr, size);

    return 0;
}
```

Output:

```
Output array is
9 7 5 3 1 2 4 6 8
```

Exercise:

Given an array of integers, sort it in alternate fashion. Alternate fashion means that the

elements at even indices are sorted separately and elements at odd indices are sorted separately.

This article is compiled by [Aashish Barnwal](#). Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

<https://www.geeksforgeeks.org/comparator-function-of-qsort-in-c/>

Chapter 60

Comparisons involved in Modified Quicksort Using Merge Sort Tree

Comparisons involved in Modified Quicksort Using Merge Sort Tree - GeeksforGeeks

In [QuickSort](#), ideal situation is when median is always chosen as pivot as this results in minimum time. In this article, [Merge Sort Tree](#) is used to find median for different ranges in QuickSort and number of comparisons are analyzed.

Examples:

```
Input : arr = {4, 3, 5, 1, 2}
Output : 11
Explanation
We have to make 11 comparisons when we
apply quick sort to the array.
```

If we carefully analyze the quick sort algorithm then every time the array is given to the quick sort function, the array always consists of a permutation of the numbers in some range L to R. Initially, it's [1 to N], then its [1 to pivot – 1] and [pivot + 1 to N] and so on. Also it's not easy to observe that the relative ordering of numbers in every possible array does not change. Now in order to get the pivot element we just need to get the middle number i.e the $(r - 1 + 2)/2^{\text{th}}$ number among the array.

To do this efficiently we can use a [Persistent Segment Tree](#), a [Fenwick Tree](#), or a [Merge Sort Tree](#). This Article focuses on the Merge Sort Tree Implementation.

In the Modified Quick Sort Algorithm where we chose the pivot element of the array as the median of the array. Now, determining the median requires us to find the middle element considered, after sorting the array which is in itself a $O(n * \log(n))$ operation where n is the size of the array.

Let's say we have a range L to R then the median of this range is calculated as:

$$\begin{aligned}\text{Median of } A[L; R] &= \text{Middle element of sorted}(A[L; R]) \\ &= (R - L + 1)/2^{\text{th}} \text{ element of} \\ &\quad \text{sorted}(A[L; R])\end{aligned}$$

Let's consider we have P partitions during the quick sort algorithm which means we have to find the pivot by sorting the array range from L to R where L and R are the starting and ending points of each partition. This is costly.

But, we have a permutation of numbers from L to R in every partition, so we can just find the $\text{ceil}((R - L + 1)/2)^{\text{th}}$ smallest number in this range as we know when we would have sorted this partition then it would always have been this element that would have ended up as being the median element as a result also the pivot. Now the elements less than pivot go to the left subtree and the ones greater than it go in the right subtree also maintaining their order.

We repeat this procedure for all partitions P and find the comparisons involved in each partition. Since in the Current Partition all the elements from L to R of that partition are compared to the pivot, we have $(R - L + 1)$ comparisons in the current partition. We also need to consider, by recursively calculating, the total comparisons in the left and right subtrees formed too. Thus we conclude,

$$\begin{aligned}\text{Total Comparisons} &= \text{Comparisons in Current Partition} + \\ &\quad \text{Comparisons in Left partition} + \\ &\quad \text{Comparisons in right partition}\end{aligned}$$

We discussed above the approach to be used to find the pivot element efficiently here the [Kth order statistics using merge sort tree](#) can be used to find the same as discussed.

```
// CPP program to implement number of comparisons
// in modified quick sort
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1000;

// Constructs a segment tree and stores tree[]
void buildTree(int treeIndex, int l, int r, int arr[],
               vector<int> tree[])
{
    /* l => start of range,
       r => ending of a range
       treeIndex => index in the Segment Tree/Merge
       Sort Tree */

```

```

/* leaf node */
if (l == r) {
    tree[treeIndex].push_back(arr[l - 1]);
    return;
}

int mid = (l + r) / 2;

/* building left subtree */
buildTree(2 * treeIndex, l, mid, arr, tree);

/* building left subtree */
buildTree(2 * treeIndex + 1, mid + 1, r, arr, tree);

/* merging left and right child in sorted order */
merge(tree[2 * treeIndex].begin(),
      tree[2 * treeIndex].end(),
      tree[2 * treeIndex + 1].begin(),
      tree[2 * treeIndex + 1].end(),
      back_inserter(tree[treeIndex]));
}

// Returns the Kth smallest number in query range
int queryRec(int segmentStart, int segmentEnd,
             int queryStart, int queryEnd, int treeIndex,
             int K, vector<int> tree[])
{
    /* segmentStart => start of a Segment,
       segmentEnd   => ending of a Segment,
       queryStart   => start of a query range,
       queryEnd     => ending of a query range,
       treeIndex     => index in the Segment
                           Tree/Merge Sort Tree,
       K   => kth smallest number to find */
    if (segmentStart == segmentEnd)
        return tree[treeIndex][0];

    int mid = (segmentStart + segmentEnd) / 2;

    // finds the last index in the segment
    // which is <= queryEnd
    int last_in_query_range =
        (upper_bound(tree[2 * treeIndex].begin(),
                    tree[2 * treeIndex].end(), queryEnd)
         - tree[2 * treeIndex].begin());

    // finds the first index in the segment
    // which is >= queryStart
}

```

```

int first_in_query_range =
    (lower_bound(tree[2 * treeIndex].begin(),
        tree[2 * treeIndex].end(), queryStart)
     - tree[2 * treeIndex].begin());

int M = last_in_query_range - first_in_query_range;

if (M >= K) {

    // Kth smallest is in left subtree,
    // so recursively call left subtree for Kth
    // smallest number
    return queryRec(segmentStart, mid, queryStart,
                    queryEnd, 2 * treeIndex, K, tree);
}

else {

    // Kth smallest is in right subtree,
    // so recursively call right subtree for the
    // (K-M)th smallest number
    return queryRec(mid + 1, segmentEnd, queryStart,
                    queryEnd, 2 * treeIndex + 1, K - M, tree);
}

// A wrapper over query()
int query(int queryStart, int queryEnd, int K, int n, int arr[],
          vector<int> tree[])
{

    return queryRec(1, n, queryStart, queryEnd,
                    1, K, tree);
}

/* Calculates total Comparisons Involved in Quick Sort
   Has the following parameters:

   start => starting index of array
   end   => ending index of array
   n     => size of array
   tree  => Merge Sort Tree */

int quickSortComparisons(int start, int end, int n, int arr[],
                         vector<int> tree[])
{
    /* Base Case */
    if (start >= end)

```

```

        return 0;

    // Compute the middle point of range and the pivot
    int middlePoint = (end - start + 2) / 2;
    int pivot = query(start, end, middlePoint, n, arr, tree);

    /* Total Comparisons = (Comparisons in Left part +
                           Comparisons of right +
                           Comparisons in parent) */

    // count comparisons in parent array
    int comparisons_in_parent = (end - start + 1);

    // count comparisons involved in left partition
    int comparisons_in_left_part =
        quickSortComparisons(start, pivot - 1, n, arr, tree);

    // count comparisons involved in right partition
    int comparisons_in_right_part =
        quickSortComparisons(pivot + 1, end, n, arr, tree);

    // Return Total Comparisons
    return comparisons_in_left_part +
           comparisons_in_parent +
           comparisons_in_right_part;
}

// Driver code
int main()
{
    int arr[] = { 4, 3, 5, 1, 2 };

    int n = sizeof(arr) / sizeof(arr[0]);

    // Construct segment tree in tree[]
    vector<int> tree[MAX];
    buildTree(1, 1, n, arr, tree);

    cout << "Number of Comparisons = "
        << quickSortComparisons(1, n, n, arr, tree);;

    return 0;
}

```

Output:

Number of Comparisons = 11

Complexity is $O(\log^2(n))$ per query for computing pivot

Source

<https://www.geeksforgeeks.org/comparisons-involved-modified-quicksort-using-merge-sort-tree/>

Chapter 61

Convert an array to reduced form Set 1 (Simple and Hashing)

Convert an array to reduced form Set 1 (Simple and Hashing) - GeeksforGeeks

Given an array with n distinct elements, convert the given array to a form where all elements are in range from 0 to n-1. The order of elements is same, i.e., 0 is placed in place of smallest element, 1 is placed for second smallest element, ... n-1 is placed for largest element.

Input: arr[] = {10, 40, 20}
Output: arr[] = {0, 2, 1}

Input: arr[] = {5, 10, 40, 30, 20}
Output: arr[] = {0, 1, 4, 3, 2}

Expected time complexity is $O(n \log n)$.

Method 1 (Simple)

A Simple Solution is to first find minimum element replace it with 0, consider remaining array and find minimum in the remaining array and replace it with 1 and so on. Time complexity of this solution is $O(n^2)$

Method 2 (Efficient)

The idea is to use hashing and sorting. Below are steps.

- 1) Create a temp array and copy contents of given array to temp[]. This takes $O(n)$ time.
- 2) Sort temp[] in ascending order. This takes $O(n \log n)$ time.
- 3) Create an empty hash table. This takes $O(1)$ time.
- 4) Traverse temp[] from left to right and store mapping of numbers and their values (in converted array) in hash table. This takes $O(n)$ time on average.

5) Traverse given array and change elements to their positions using hash table. This takes $O(n)$ time on average.

Overall time complexity of this solution is $O(n \log n)$.

Below are implementations of above idea.

C++

```
// C++ program to convert an array in reduced
// form
#include <bits/stdc++.h>
using namespace std;

void convert(int arr[], int n)
{
    // Create a temp array and copy contents
    // of arr[] to temp
    int temp[n];
    memcpy(temp, arr, n*sizeof(int));

    // Sort temp array
    sort(temp, temp + n);

    // Create a hash table. Refer
    // http://tinyurl.com/zp5wgef
    unordered_map<int, int> umap;

    // One by one insert elements of sorted
    // temp[] and assign them values from 0
    // to n-1
    int val = 0;
    for (int i = 0; i < n; i++)
        umap[temp[i]] = val++;

    // Convert array by taking positions from
    // umap
    for (int i = 0; i < n; i++)
        arr[i] = umap[arr[i]];
}

void printArr(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}

// Driver program to test above method
int main()
```

```
{  
    int arr[] = {10, 20, 15, 12, 11, 50};  
    int n = sizeof(arr)/sizeof(arr[0]);  
  
    cout << "Given Array is \n";  
    printArr(arr, n);  
  
    convert(arr , n);  
  
    cout << "\n\nConverted Array is \n";  
    printArr(arr, n);  
  
    return 0;  
}
```

Java

```
// Java Program to convert an Array  
// to reduced form  
import java.util.*;  
  
class GFG  
{  
    public static void convert(int arr[], int n)  
    {  
        // Create a temp array and copy contents  
        // of arr[] to temp  
        int temp[] = arr.clone();  
  
        // Sort temp array  
        Arrays.sort(temp);  
  
        // Create a hash table.  
        HashMap<Integer, Integer> umap = new HashMap<>();  
  
        // One by one insert elements of sorted  
        // temp[] and assign them values from 0  
        // to n-1  
        int val = 0;  
        for (int i = 0; i < n; i++)  
            umap.put(temp[i], val++);  
  
        // Convert array by taking positions from  
        // umap  
        for (int i = 0; i < n; i++)  
            arr[i] = umap.get(arr[i]);  
    }  
}
```

```
public static void printArr(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}

// Driver code
public static void main(String[] args)
{

    int arr[] = {10, 20, 15, 12, 11, 50};
    int n = arr.length;

    System.out.println("Given Array is ");
    printArr(arr, n);

    convert(arr , n);

    System.out.println("\n\nConverted Array is ");
    printArr(arr, n);

}

}

// This code is contributed by Abhishek Panwar
```

Python3

```
# Python3 program to convert an array
# in reduced form
def convert(arr, n):
    # Create a temp array and copy contents
    # of arr[] to temp
    temp = [arr[i] for i in range (n) ]

    # Sort temp array
    temp.sort()

    # create a map
    umap = {}

    # One by one insert elements of sorted
    # temp[] and assign them values from 0
    # to n-1
    val = 0
    for i in range (n):
        umap[temp[i]] = val
```

```
val += 1

# Convert array by taking positions from umap
for i in range (n):
    arr[i] = umap[arr[i]]

def printArr(arr, n):
    for i in range(n):
        print(arr[i], end = " ")

# Driver Code
if __name__ == "__main__":
    arr = [10, 20, 15, 12, 11, 50]
    n = len(arr)
    print("Given Array is ")
    printArr(arr, n)
    convert(arr , n)
    print("\n\nConverted Array is ")
    printArr(arr, n)

# This code is contributed by Abhishek Gupta
```

Output :

```
Given Array is
10 20 15 12 11 50
```

```
Converted Array is
0 4 3 2 1 5
```

Convert an array to reduced form Set 2 (Using vector of pairs)

This article is contributed by **Dheeraj Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [panwarabhishek345](#), [ab_gupta](#)

Source

<https://www.geeksforgeeks.org/convert-an-array-to-reduced-form-set-1-simple-and-hashing/>

Chapter 62

Convert an array to reduced form Set 2 (Using vector of pairs)

Convert an array to reduced form Set 2 (Using vector of pairs) - GeeksforGeeks

Given an array with n distinct elements, convert the given array to a form where all elements are in range from 0 to n-1. The order of elements is same, i.e., 0 is placed in place of smallest element, 1 is placed for second smallest element, ... n-1 is placed for largest element.

Input: arr[] = {10, 40, 20}
Output: arr[] = {0, 2, 1}

Input: arr[] = {5, 10, 40, 30, 20}
Output: arr[] = {0, 1, 4, 3, 2}

We have discussed [simple and hashing based solutions](#).

In this post, a new solution is discussed. The idea is to create a vector of pairs. Every element of pair contains element and index. We sort vector by array values. After sorting, we copy indexes to original array.

```
// C++ program to convert an array in reduced
// form
#include <bits/stdc++.h>
using namespace std;

// Converts arr[0..n-1] to reduced form.
void convert(int arr[], int n)
{
```

```
// A vector of pairs. Every element of
// pair contains array element and its
// index
vector <pair<int, int> > v;

// Put all elements and their index in
// the vector
for (int i = 0; i < n; i++)
    v.push_back(make_pair(arr[i], i));

// Sort the vector by array values
sort(v.begin(), v.end());

// Put indexes of modified vector in arr[]
for (int i=0; i<n; i++)
    arr[v[i].second] = i;
}

// Utility function to print an array.
void printArr(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}

// Driver program to test above method
int main()
{
    int arr[] = {10, 20, 15, 12, 11, 50};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Given Array is \n";
    printArr(arr, n);

    convert(arr , n);

    cout << "\n\nConverted Array is \n";
    printArr(arr, n);

    return 0;
}
```

Output :

```
Given Array is
10 20 15 12 11 50
```

```
Converted Array is
```

0 4 3 2 1 5

Time Complexity : $O(n \log n)$

Auxiliary Space : $O(n)$

Improved By : [aganjali10](#)

Source

<https://www.geeksforgeeks.org/convert-array-reduced-form-set-2-using-vector-pairs/>

Chapter 63

Count Inversions in an array Set 1 (Using Merge Sort)

Count Inversions in an array Set 1 (Using Merge Sort) - GeeksforGeeks

Inversion Count for an array indicates – how far (or close) the array is from being sorted. If array is already sorted then inversion count is 0. If array is sorted in reverse order that inversion count is the maximum.

Formally speaking, two elements $a[i]$ and $a[j]$ form an inversion if $a[i] > a[j]$ and $i < j$

Example:

The sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3).

METHOD 1 (Simple)

For each element, count number of elements which are on right side of it and are smaller than it.

C

```
#include <bits/stdc++.h>
int getInvCount(int arr[], int n)
{
    int inv_count = 0;
    for (int i = 0; i < n - 1; i++)
        for (int j = i+1; j < n; j++)
            if (arr[i] > arr[j])
                inv_count++;

    return inv_count;
}

/* Driver program to test above functions */
int main(int argc, char** args)
```

```
{  
    int arr[] = {1, 20, 6, 4, 5};  
    int n = sizeof(arr)/sizeof(arr[0]);  
    printf(" Number of inversions are %d \n", getInvCount(arr, n));  
    return 0;  
}
```

Java

```
// Java program to count  
// inversions in an array  
class Test  
{  
    static int arr[] = new int[]{1, 20, 6, 4, 5};  
  
    static int getInvCount(int n)  
    {  
        int inv_count = 0;  
        for (int i = 0; i < n - 1; i++)  
            for (int j = i+1; j < n; j++)  
                if (arr[i] > arr[j])  
                    inv_count++;  
  
        return inv_count;  
    }  
  
    // Driver method to test the above function  
    public static void main(String[] args)  
    {  
        System.out.println("Number of inversions are "  
                           + getInvCount(arr.length));  
  
    }  
}
```

Python3

```
# Python3 program to count  
# inversions in an array  
  
def getInvCount(arr, n):  
  
    inv_count = 0  
    for i in range(n):  
        for j in range(i+1, n):  
            if (arr[i] > arr[j]):  
                inv_count += 1
```

```
    return inv_count

# Driver Code
arr = [1, 20, 6, 4, 5]
n = len(arr)
print("Number of inversions are",
      getInvCount(arr, n))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// C# program to count inversions
// in an array
using System;
using System.Collections.Generic;

class GFG {

    static int []arr =
        new int[]{1, 20, 6, 4, 5};

    static int getInvCount(int n)
    {
        int inv_count = 0;

        for (int i = 0; i < n - 1; i++)
            for (int j = i+1; j < n; j++)
                if (arr[i] > arr[j])
                    inv_count++;

        return inv_count;
    }

    // Driver code
    public static void Main()
    {
        Console.WriteLine("Number of "
                          + "inversions are "
                          + getInvCount(arr.Length));
    }
}

// This code is contributed by Sam007
```

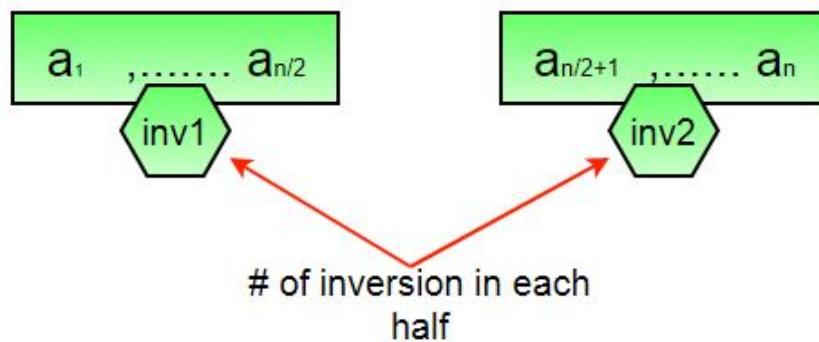
Output:

Number of inversions are 5

Time Complexity: $O(n^2)$

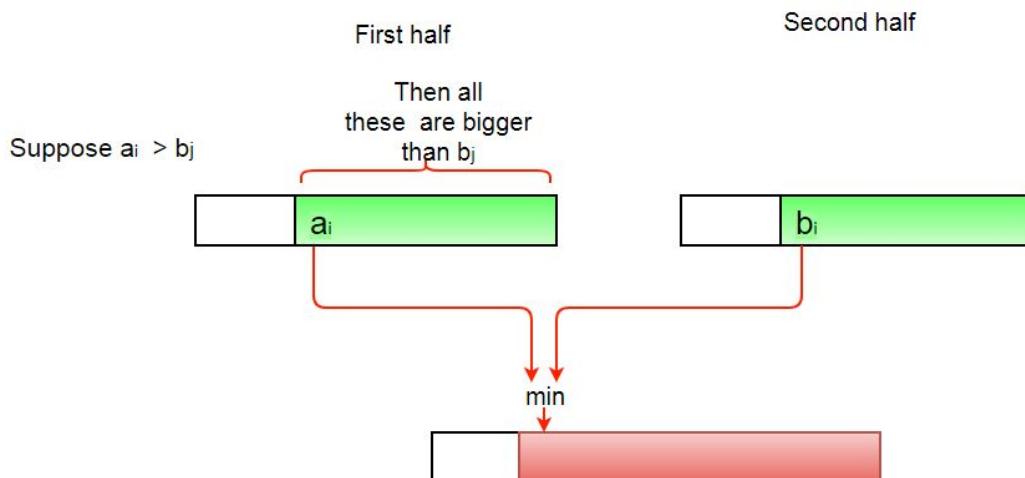
METHOD 2(Enhance Merge Sort)

Suppose we know the number of inversions in the left half and right half of the array (let be $inv1$ and $inv2$), what kinds of inversions are not accounted for in $Inv1 + Inv2$? The answer is – the inversions we have to count during the merge step. Therefore, to get number of inversions, we need to add number of inversions in left subarray, right subarray and merge().

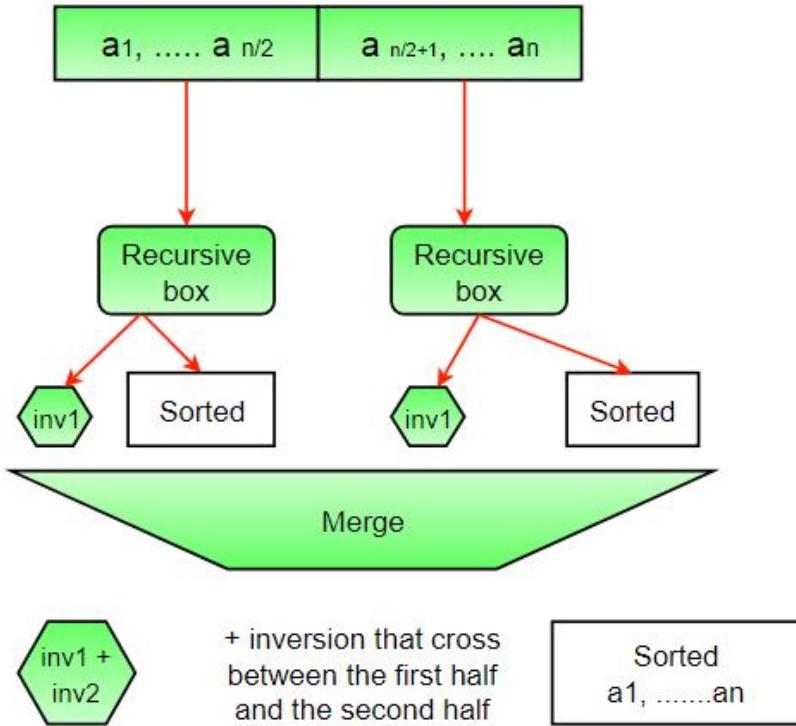


How to get number of inversions in merge()?

In merge process, let i is used for indexing left sub-array and j for right sub-array. At any step in merge(), if $a[i]$ is greater than $a[j]$, then there are $(mid - i)$ inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray ($a[i+1], a[i+2] \dots a[mid]$) will be greater than $a[j]$



The complete picture:



Implementation:
C

```
#include <bits/stdc++.h>

int _mergeSort(int arr[], int temp[], int left, int right);
int merge(int arr[], int temp[], int left, int mid, int right);

/* This function sorts the input array and returns the
   number of inversions in the array */
int mergeSort(int arr[], int array_size)
{
    int *temp = (int *)malloc(sizeof(int)*array_size);
    return _mergeSort(arr, temp, 0, array_size - 1);
}

/* An auxiliary recursive function that sorts the input array and
   returns the number of inversions in the array. */
int _mergeSort(int arr[], int temp[], int left, int right)
{
    int mid, inv_count = 0;
    if (right > left)
```

```

{
    /* Divide the array into two parts and call _mergeSortAndCountInv()
       for each of the parts */
    mid = (right + left)/2;

    /* Inversion count will be sum of inversions in left-part, right-part
       and number of inversions in merging */
    inv_count = _mergeSort(arr, temp, left, mid);
    inv_count += _mergeSort(arr, temp, mid+1, right);

    /*Merge the two parts*/
    inv_count += merge(arr, temp, left, mid+1, right);
}
return inv_count;
}

/* This funt merges two sorted arrays and returns inversion count in
   the arrays.*/
int merge(int arr[], int temp[], int left, int mid, int right)
{
    int i, j, k;
    int inv_count = 0;

    i = left; /* i is index for left subarray*/
    j = mid; /* j is index for right subarray*/
    k = left; /* k is index for resultant merged subarray*/
    while ((i <= mid - 1) && (j <= right))
    {
        if (arr[i] <= arr[j])
        {
            temp[k++] = arr[i++];
        }
        else
        {
            temp[k++] = arr[j++];

            /*this is tricky -- see above explanation/diagram for merge()*/
            inv_count = inv_count + (mid - i);
        }
    }

    /* Copy the remaining elements of left subarray
       (if there are any) to temp*/
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    /* Copy the remaining elements of right subarray
       (if there are any) to temp*/
}

```

```
while (j <= right)
    temp[k++] = arr[j++];

/*Copy back the merged elements to original array*/
for (i=left; i <= right; i++)
    arr[i] = temp[i];

return inv_count;
}

/* Driver program to test above functions */
int main(int argc, char** args)
{
    int arr[] = {1, 20, 6, 4, 5};
    printf(" Number of inversions are %d \n", mergeSort(arr, 5));
    getchar();
    return 0;
}
```

Java

```
// Java implementation of counting the
// inversion using merge sort

class Test
{

    /* This method sorts the input array and returns the
       number of inversions in the array */
    static int mergeSort(int arr[], int array_size)
    {
        int temp[] = new int[array_size];
        return _mergeSort(arr, temp, 0, array_size - 1);
    }

    /* An auxiliary recursive method that sorts the input array and
       returns the number of inversions in the array. */
    static int _mergeSort(int arr[], int temp[], int left, int right)
    {
        int mid, inv_count = 0;
        if (right > left)
        {
            /* Divide the array into two parts and call _mergeSortAndCountInv()
               for each of the parts */
            mid = (right + left)/2;

            /* Inversion count will be sum of inversions in left-part, right-part
               and number of inversions in merging */

```

```

inv_count = _mergeSort(arr, temp, left, mid);
inv_count += _mergeSort(arr, temp, mid+1, right);

/*Merge the two parts*/
inv_count += merge(arr, temp, left, mid+1, right);
}
return inv_count;
}

/* This method merges two sorted arrays and returns inversion count in
   the arrays.*/
static int merge(int arr[], int temp[], int left, int mid, int right)
{
    int i, j, k;
    int inv_count = 0;

    i = left; /* i is index for left subarray*/
    j = mid; /* j is index for right subarray*/
    k = left; /* k is index for resultant merged subarray*/
    while ((i <= mid - 1) && (j <= right))
    {
        if (arr[i] <= arr[j])
        {
            temp[k++] = arr[i++];
        }
        else
        {
            temp[k++] = arr[j++];

            /*this is tricky -- see above explanation/diagram for merge()*/
            inv_count = inv_count + (mid - i);
        }
    }

    /* Copy the remaining elements of left subarray
       (if there are any) to temp*/
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    /* Copy the remaining elements of right subarray
       (if there are any) to temp*/
    while (j <= right)
        temp[k++] = arr[j++];

    /*Copy back the merged elements to original array*/
    for (i=left; i <= right; i++)
        arr[i] = temp[i];
}

```

```
    return inv_count;
}

// Driver method to test the above function
public static void main(String[] args)
{
    int arr[] = new int[]{1, 20, 6, 4, 5};
    System.out.println("Number of inversions are " + mergeSort(arr, 5));

}
}
```

Output:

```
Number of inversions are 5
```

Note that above code modifies (or sorts) the input array. If we want to count only inversions then we need to create a copy of original array and call mergeSort() on copy.

Time Complexity: O(nlogn)

Algorithmic Paradigm: Divide and Conquer

You may like to see.

[Count inversions in an array Set 2 \(Using Self-Balancing BST\)](#)

[Counting Inversions using Set in C++ STL](#)

[Count inversions in an array Set 3 \(Using BIT\)](#)

References:

<http://www.cs.umd.edu/class/fall2009/cmsc451/lectures/Lec08-inversions.pdf>

<http://www.cp.eng.chula.ac.th/~piak/teaching/algo/algo2008/count-inv.htm>

Improved By : Sam007

Source

<https://www.geeksforgeeks.org/counting-inversions/>

Chapter 64

Count all distinct pairs with difference equal to k

Count all distinct pairs with difference equal to k - GeeksforGeeks

Given an integer array and a positive integer k, count all distinct pairs with difference equal to k.

Examples:

Input: arr[] = {1, 5, 3, 4, 2}, k = 3

Output: 2

There are 2 pairs with difference 3, the pairs are {1, 4} and {5, 2}

Input: arr[] = {8, 12, 16, 4, 0, 20}, k = 4

Output: 5

There are 5 pairs with difference 4, the pairs are {0, 4}, {4, 8}, {8, 12}, {12, 16} and {16, 20}

Method 1 (Simple)

A simple solution is to consider all pairs one by one and check difference between every pair. Following program implements the simple solution. We run two loops: the outer loop picks the first element of pair, the inner loop looks for the other element. This solution doesn't work if there are duplicates in array as the requirement is to count only distinct pairs.

C++

```
/* A simple program to count pairs with difference k*/
#include<iostream>
using namespace std;

int countPairsWithDiffK(int arr[], int n, int k)
```

```
{  
    int count = 0;  
  
    // Pick all elements one by one  
    for (int i = 0; i < n; i++)  
    {  
        // See if there is a pair of this picked element  
        for (int j = i+1; j < n; j++)  
            if (arr[i] - arr[j] == k || arr[j] - arr[i] == k )  
                count++;  
    }  
    return count;  
}  
  
// Driver program to test above function  
int main()  
{  
    int arr[] = {1, 5, 3, 4, 2};  
    int n = sizeof(arr)/sizeof(arr[0]);  
    int k = 3;  
    cout << "Count of pairs with given diff is "  
         << countPairsWithDiffK(arr, n, k);  
    return 0;  
}
```

Java

```
// A simple Java program to  
//count pairs with difference k  
import java.util.*;  
import java.io.*;  
  
class GFG {  
  
    static int countPairsWithDiffK(int arr[],  
                                  int n, int k)  
    {  
        int count = 0;  
  
        // Pick all elements one by one  
        for (int i = 0; i < n; i++)  
        {  
            // See if there is a pair  
            // of this picked element  
            for (int j = i + 1; j < n; j++)  
                if (arr[i] - arr[j] == k ||  
                    arr[j] - arr[i] == k)  
                    count++;  
    }  
}
```

```
        }
        return count;
    }

// Driver code
public static void main(String args[])
{
    int arr[] = { 1, 5, 3, 4, 2 };
    int n = arr.length;
    int k = 3;
    System.out.println("Count of pairs with given diff is "
                       + countPairsWithDiffK(arr, n, k));
}
}

// This code is contributed
// by Sahil_Bansall
```

Python3

```
# A simple program to count pairs with difference k

def countPairsWithDiffK(arr, n, k):
    count = 0

    # Pick all elements one by one
    for i in range(0, n):

        # See if there is a pair of this picked element
        for j in range(i+1, n) :

            if arr[i] - arr[j] == k or arr[j] - arr[i] == k:
                count += 1

    return count

# Driver program
arr = [1, 5, 3, 4, 2]

n = len(arr)
k = 3
print ("Count of pairs with given diff is ",
       countPairsWithDiffK(arr, n, k))
```

C#

```
// A simple C# program to count pairs with
```

```
// difference k
using System;

class GFG {

    static int countPairsWithDiffK(int []arr,
                                  int n, int k)
    {
        int count = 0;

        // Pick all elements one by one
        for (int i = 0; i < n; i++)
        {
           

            // See if there is a pair
            // of this picked element
            for (int j = i + 1; j < n; j++)
                if (arr[i] - arr[j] == k ||
                    arr[j] - arr[i] == k)
                    count++;

        }

        return count;
    }

    // Driver code
    public static void Main()
    {
        int []arr = { 1, 5, 3, 4, 2 };
        int n = arr.Length;
        int k = 3;

        Console.WriteLine("Count of pairs with "
                        + " given diff is "
                        + countPairsWithDiffK(arr, n, k));
    }
}

// This code is contributed by Sam007.
```

PHP

```
<?php
// A simple PHP program to count
// pairs with difference k

function countPairsWithDiffK($arr, $n,
                            $k)
```

```
{  
    $count = 0;  
  
    // Pick all elements one by one  
    for($i = 0; $i < $n; $i++)  
    {  
  
        // See if there is a pair of  
        // this picked element  
        for($j = $i + 1; $j < $n; $j++)  
            if ($arr[$i] - $arr[$j] == $k or  
                $arr[$j] - $arr[$i] == $k)  
                $count++;  
    }  
    return $count;  
}  
  
// Driver Code  
$arr = array(1, 5, 3, 4, 2);  
$n = count($arr);  
$k = 3;  
echo "Count of pairs with given diff is "  
     , countPairsWithDiffK($arr, $n, $k);  
  
// This code is contributed by anuj_67.  
?>
```

Output :

Count of pairs with given diff is 2

Time Complexity of $O(n^2)$

Method 2 (Use Sorting)

We can find the count in $O(n \log n)$ time using a $O(n \log n)$ sorting algorithm like [Merge Sort](#), [Heap Sort](#), etc. Following are the detailed steps.

- 1) Initialize count as 0
- 2) Sort all numbers in increasing order.
- 3) Remove duplicates from array.
- 4) Do following for each element arr[i]
 - a) Binary Search for arr[i] + k in subarray from i+1 to n-1.
 - b) If arr[i] + k found, increment count.
- 5) Return count.

C++

```
/* A sorting based program to count pairs with difference k*/
#include <iostream>
#include <algorithm>
using namespace std;

/* Standard binary search function */
int binarySearch(int arr[], int low, int high, int x)
{
    if (high >= low)
    {
        int mid = low + (high - low)/2;
        if (x == arr[mid])
            return mid;
        if (x > arr[mid])
            return binarySearch(arr, (mid + 1), high, x);
        else
            return binarySearch(arr, low, (mid -1), x);
    }
    return -1;
}

/* Returns count of pairs with difference k in arr[] of size n. */
int countPairsWithDiffK(int arr[], int n, int k)
{
    int count = 0, i;
    sort(arr, arr+n); // Sort array elements

    /* code to remove duplicates from arr[] */

    // Pick a first element point
    for (i = 0; i < n-1; i++)
        if (binarySearch(arr, i+1, n-1, arr[i] + k) != -1)
            count++;

    return count;
}

// Driver program
int main()
{
    int arr[] = {1, 5, 3, 4, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    cout << "Count of pairs with given diff is "
         << countPairsWithDiffK(arr, n, k);
    return 0;
}
```

}

Java

```
// A sorting base java program to
// count pairs with difference k
import java.util.*;
import java.io.*;

class GFG {

    // Standard binary search function //
    static int binarySearch(int arr[], int low,
                           int high, int x)
    {
        if (high >= low)
        {
            int mid = low + (high - low) / 2;
            if (x == arr[mid])
                return mid;
            if (x > arr[mid])
                return binarySearch(arr, (mid + 1),
                                   high, x);
            else
                return binarySearch(arr, low,
                                   (mid - 1), x);
        }
        return -1;
    }

    // Returns count of pairs with
    // difference k in arr[] of size n.
    static int countPairsWithDiffK(int arr[], int n, int k)
    {
        int count = 0, i;

        // Sort array elements
        Arrays.sort(arr);

        // code to remove duplicates from arr[]

        // Pick a first element point
        for (i = 0; i < n - 1; i++)
            if (binarySearch(arr, i + 1, n - 1,
                            arr[i] + k) != -1)
                count++;

        return count;
    }
}
```

```
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 1, 5, 3, 4, 2 };
    int n = arr.length;
    int k = 3;
    System.out.println("Count of pairs with given diff is "
                       + countPairsWithDiffK(arr, n, k));
}
}

// This code is contributed by Sahil_Bansall
```

Python

```
# A sorting based program to
# count pairs with difference k

# Standard binary search function
def binarySearch(arr, low, high, x):

    if (high >= low):

        mid = low + (high - low)//2
        if x == arr[mid]:
            return (mid)
        elif(x > arr[mid]):
            return binarySearch(arr, (mid + 1), high, x)
        else:
            return binarySearch(arr, low, (mid -1), x)

    return -1

# Returns count of pairs with
# difference k in arr[] of size n.
def countPairsWithDiffK(arr, n, k):

    count = 0
    arr.sort() # Sort array elements

    # code to remove
    # duplicates from arr[]

    # Pick a first element point
    for i in range (0, n - 2):
```

```
if (binarySearch(arr, i + 1, n - 1,
                  arr[i] + k) != -1):
    count += 1

return count

# Driver Code
arr= [1, 5, 3, 4, 2]
n = len(arr)
k = 3
print ("Count of pairs with given diff is ",
       countPairsWithDiffK(arr, n, k))

# This code is contributed
# by Shivi_Aggarwal
```

C#

```
// A sorting base C# program to
// count pairs with difference k
using System;

class GFG {

    // Standard binary search function
    static int binarySearch(int []arr, int low,
                           int high, int x)
    {
        if (high >= low)
        {
            int mid = low + (high - low) / 2;
            if (x == arr[mid])
                return mid;
            if (x > arr[mid])
                return binarySearch(arr, (mid + 1),
                                   high, x);
            else
                return binarySearch(arr, low,
                                   (mid - 1), x);
        }
        return -1;
    }

    // Returns count of pairs with
    // difference k in arr[] of size n.
    static int countPairsWithDiffK(int []arr,
```

```
int n, int k)
{

    int count = 0, i;

    // Sort array elements
    Array.Sort(arr);

    // code to remove duplicates from arr[]

    // Pick a first element point
    for (i = 0; i < n - 1; i++)
        if (binarySearch(arr, i + 1, n - 1,
                          arr[i] + k) != -1)
            count++;

    return count;
}

// Driver code
public static void Main()
{
    int []arr = { 1, 5, 3, 4, 2 };
    int n = arr.Length;
    int k = 3;

    Console.WriteLine("Count of pairs with"
                      + " given diff is "
                      + countPairsWithDiffK(arr, n, k));
}
}

// This code is contributed by Sam007.
```

PHP

```
<?php
// A sorting based PHP program to
// count pairs with difference k

// Standard binary search function
function binarySearch($arr, $low,
                      $high, $x)
{
    if ($high >= $low)
    {
        $mid = $low + ($high - $low)/2;
        if ($x == $arr[$mid])
```

```
        return $mid;

    if ($x > $arr[$mid])
        return binarySearch($arr, ($mid + 1),
                            $high, $x);
    else
        return binarySearch($arr, $low,
                            ($mid -1), $x);
    }
    return -1;
}

/* Returns count of pairs with
   difference k in arr[] of size n. */
function countPairsWithDiffK($arr, $n, $k)
{
    $count = 0;
    $i;

    // Sort array elements
    sort($arr);

    // Code to remove duplicates
    // from arr[]

    // Pick a first element point
    for ($i = 0; $i < $n - 1; $i++)
        if (binarySearch($arr, $i + 1, $n - 1,
                        $arr[$i] + $k) != -1)
            $count++;

    return $count;
}

// Driver Code
$arr = array(1, 5, 3, 4, 2);
$n = count($arr);
$k = 3;
echo "Count of pairs with given diff is "
     , countPairsWithDiffK($arr, $n, $k);

// This code is contributed by anuj-67.
?>
```

Output:

Count of pairs with given diff is 2

Time complexity: The first step (sorting) takes $O(n\log n)$ time. The second step runs binary search n times, so the time complexity of second step is also $O(n\log n)$. Therefore, overall time complexity is $O(n\log n)$. The second step can be optimized to $O(n)$, see [this](#).

Method 3 (Use Self-balancing BST)

We can also use a self-balancing BST like [AVL tree](#) or Red Black tree to solve this problem. Following is detailed algorithm.

- 1) Initialize count as 0.
- 2) Insert all elements of arr[] in an AVL tree. While inserting, ignore an element if already present in AVL tree.
- 3) Do following for each element arr[i].
 - a) Search for arr[i] + k in AVL tree, if found then increment count.
 - b) Search for arr[i] - k in AVL tree, if found then increment count.
 - c) Remove arr[i] from AVL tree.

Time complexity of above solution is also $O(n\log n)$ as search and delete operations take $O(\log n)$ time for a self-balancing binary search tree.

Method 4 (Use Hashing)

We can also use hashing to achieve the average time complexity as $O(n)$ for many cases.

- 1) Initialize count as 0.
- 2) Insert all distinct elements of arr[] in a hash map. While inserting, ignore an element if already present in the hash map.
- 3) Do following for each element arr[i].
 - a) Look for arr[i] + k in the hash map, if found then increment count.
 - b) Look for arr[i] - k in the hash map, if found then increment count.
 - c) Remove arr[i] from hash table.

A very simple case where hashing works in $O(n)$ time is the case where range of values is very small. For example, in the following implementation, range of numbers is assumed to be 0 to 99999. A simple hashing technique to use values as index can be used.

```
/* An efficient program to count pairs with difference k when the range
   numbers is small */
#define MAX 100000
int countPairsWithDiffK(int arr[], int n, int k)
{
    int count = 0; // Initialize count

    // Initialize empty hashmap.
    bool hashmap[MAX] = {false};

    // Insert array elements to hashmap
    for (int i = 0; i < n; i++)
        hashmap[arr[i]] = true;

    // Count pairs
    for (int i = 0; i < n; i++) {
        if (hashmap[i + k])
            count++;
        if (hashmap[i - k])
            count++;
    }
}
```

```
for (int i = 0; i < n; i++)
{
    int x = arr[i];
    if (x - k >= 0 && hashmap[x - k])
        count++;
    if (x + k < MAX && hashmap[x + k])
        count++;
    hashmap[x] = false;
}
return count;
}
```

Method 5 (Use Sorting)

Sort the array arr

Take two pointers, l and r, both pointing to 1st element

Take the difference arr[r] – arr[l]

- If value diff is K, increment count and move both pointers to next element
- if value diff > k, move l to next element
- if value diff < k, move r to next element

return count

C++

```
/* A sorting based program to count pairs with difference k*/
#include <iostream>
#include <algorithm>
using namespace std;

/* Returns count of pairs with difference k in arr[] of size n. */
int countPairsWithDiffK(int arr[], int n, int k)
{
    int count = 0;
    sort(arr, arr+n); // Sort array elements

    int l = 0;
    int r = 0;
    while(r < n)
    {
        if(arr[r] - arr[l] == k)
        {
            count++;
            l++;
            r++;
        }
    }
}
```

```
        }
        else if(arr[r] - arr[l] > k)
            l++;
        else // arr[r] - arr[l] < sum
            r++;
    }
    return count;
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 5, 3, 4, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    cout << "Count of pairs with given diff is "
         << countPairsWithDiffK(arr, n, k);
    return 0;
}
```

Java

```
// A sorting based Java program to
// count pairs with difference k
import java.util.*;

class GFG {

    /* Returns count of pairs with
    difference k in arr[] of size n. */
    static int countPairsWithDiffK(int arr[], int n,
                                  int k)
    {
        int count = 0;
        Arrays.sort(arr); // Sort array elements

        int l = 0;
        int r = 0;
        while(r < n)
        {
            if(arr[r] - arr[l] == k)
            {
                count++;
                l++;
                r++;
            }
            else if(arr[r] - arr[l] > k)
                l++;
        }
    }
}
```

```
        else // arr[r] - arr[l] < sum
            r++;
    }
    return count;
}

// Driver program to test above function
public static void main(String[] args)
{
    int arr[] = {1, 5, 3, 4, 2};
    int n = arr.length;
    int k = 3;
    System.out.println("Count of pairs with given diff is " +
        countPairsWithDiffK(arr, n, k));
}
}

// This code is contributed by Prerna Saini
```

C#

```
// A sorting based C# program to count
// pairs with difference k
using System;

class GFG {

    /* Returns count of pairs with
    difference k in arr[] of size n. */
    static int countPairsWithDiffK(int []arr,
                                int n, int k)
    {
        int count = 0;

        // Sort array elements
        Array.Sort(arr);

        int l = 0;
        int r = 0;
        while(r < n)
        {
            if(arr[r] - arr[l] == k)
            {
                count++;
                l++;
                r++;
            }
            else if(arr[r] - arr[l] > k)
```

```
        l++;
    else // arr[r] - arr[l] < sum
        r++;
}
return count;
}

// Driver program to test above function
public static void Main()
{
    int []arr = {1, 5, 3, 4, 2};
    int n = arr.Length;
    int k = 3;
    Console.Write("Count of pairs with "
                  + "given diff is " +
                  countPairsWithDiffK(arr, n, k));
}
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// A sorting based program to count
// pairs with difference k

// Returns count of pairs with
// difference k in arr[] of size n.
function countPairsWithDiffK( $arr, $n, $k)
{
    $count = 0;

    // Sort array elements
    sort($arr);

    $l = 0;
    $r = 0;
    while($r < $n)
    {
        if($arr[$r] - $arr[$l] == $k)
        {
            $count++;
            $l++;
            $r++;
        }
        else if($arr[$r] - $arr[$l] > $k)
            $l++;
    }
}
```

```
// arr[r] - arr[l] < sum
else
    $r++;
}
return $count;
}

// Driver Code
$arr = array(1, 5, 3, 4, 2);
$n = count($arr);
$k = 3;
echo "Count of pairs with given diff is "
, countPairsWithDiffK($arr, $n, $k);

// This code is contributed by anuj_67,
?>
```

Output:

Count of pairs with given diff is 2

Time Complexity: O(nlogn)

Improved By : nitin mittal, vt_m, Shivi_Aggarwal

Source

<https://www.geeksforgeeks.org/count-pairs-difference-equal-k/>

Chapter 65

Count distinct occurrences as a subsequence

Count distinct occurrences as a subsequence - GeeksforGeeks

Given two strings S and T, find count of distinct occurrences of T in S as a subsequence.

Examples:

```
Input : S = banana, T = ban
Output : 3
T appears in S as below three subsequences.
[ban], [ba n], [b an]
```

```
Input : S = geeksforgeeks, T = ge
Output : 6
T appears in S as below three subsequences.
[ge], [ g e], [g e], [g e] [g e]
and [ g e]
```

This problem can be recursively defined as below.

```
// Returns count of subsequences of S that match T
// m is length of T and n is length of S
subsequenceCount(S, T, n, m)

// An empty string is subsequence of all.
1) If length of T is 0, return 1.

// Else no string can be a sequence of empty S.
2) Else if S is empty, return 0.
```

- 3) Else if last characters of S and T don't match,
remove last character of S and recur for remaining
return subsequenceCount(S, T, n-1, m)
- 4) Else (Last characters match), the result is sum
of two counts.

 // Remove last character of S and recur.
 a) subsequenceCount(S, T, n-1, m) +

 // Remove last characters of S and T, and recur.
 b) subsequenceCount(S, T, n-1, m-1)

Since there are overlapping subproblems in above recurrence result, we can apply dynamic programming approach to solve above problem. We create a 2D array mat[m+1][n+1] where m is length of string T and n is length of string S. mat[i][j] denotes the number of distinct subsequence of substring S(1..i) and substring T(1..j) so mat[m][n] contains our solution.

C++

```
/* C/C++ program to count number of times S appears
   as a subsequence in T */
#include <bits/stdc++.h>
using namespace std;

int findSubsequenceCount(string S, string T)
{
    int m = T.length(), n = S.length();

    // T can't appear as a subsequence in S
    if (m > n)
        return 0;

    // mat[i][j] stores the count of occurrences of
    // T(1..i) in S(1..j).
    int mat[m + 1][n + 1];

    // Initializing first column with all 0s. An empty
    // string can't have another string as subsequence
    for (int i = 1; i <= m; i++)
        mat[i][0] = 0;

    // Initializing first row with all 1s. An empty
    // string is subsequence of all.
    for (int j = 0; j <= n; j++)
        mat[0][j] = 1;

    // Fill mat[][] in bottom up manner
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            if (T[i-1] == S[j-1])
                mat[i][j] = mat[i-1][j-1] + mat[i-1][j];
            else
                mat[i][j] = mat[i-1][j];
}
```

```

for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
        // If last characters don't match, then value
        // is same as the value without last character
        // in S.
        if (T[i - 1] != S[j - 1])
            mat[i][j] = mat[i][j - 1];

        // Else value is obtained considering two cases.
        // a) All substrings without last character in S
        // b) All substrings without last characters in
        // both.
        else
            mat[i][j] = mat[i][j - 1] + mat[i - 1][j - 1];
    }
}

/* uncomment this to print matrix mat
for (int i = 1; i <= m; i++, cout << endl)
    for (int j = 1; j <= n; j++)
        cout << mat[i][j] << " ";
return mat[m][n];
}

// Driver code to check above method
int main()
{
    string T = "ge";
    string S = "geeksforgeeks";
    cout << findSubsequenceCount(S, T) << endl;
    return 0;
}

```

Java

```

// Java program to count number of times
// S appears as a subsequence in T
import java.io.*;

class GFG {
    static int findSubsequenceCount(String S, String T)
    {
        int m = T.length();
        int n = S.length();

        // T can't appear as a subsequence in S
        if (m > n)
            return 0;

```

```

// mat[i][j] stores the count of
// occurrences of T(1..i) in S(1..j).
int mat[][] = new int[m + 1][n + 1];

// Initializing first column with
// all 0s. An emptystring can't have
// another string as subsequence
for (int i = 1; i <= m; i++)
    mat[i][0] = 0;

// Initializing first row with all 1s.
// An empty string is subsequence of all.
for (int j = 0; j <= n; j++)
    mat[0][j] = 1;

// Fill mat[][] in bottom up manner
for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
        // If last characters don't match,
        // then value is same as the value
        // without last character in S.
        if (T.charAt(i - 1) != S.charAt(j - 1))
            mat[i][j] = mat[i][j - 1];

        // Else value is obtained considering two cases.
        // a) All substrings without last character in S
        // b) All substrings without last characters in
        // both.
        else
            mat[i][j] = mat[i][j - 1] + mat[i - 1][j - 1];
    }
}

/* uncomment this to print matrix mat
for (int i = 1; i <= m; i++, cout << endl)
    for (int j = 1; j <= n; j++)
        System.out.println ( mat[i][j] + " ");
return mat[m][n];
}

// Driver code to check above method
public static void main(String[] args)
{
    String T = "ge";
    String S = "geeksforgeeks";
    System.out.println(findSubsequenceCount(S, T));
}

```

```
}
```

```
// This code is contributed by vt_m
```

C#

```
// C# program to count number of times
// S appears as a subsequence in T
using System;

class GFG {

    static int findSubsequenceCount(string S, string T)
    {
        int m = T.Length;
        int n = S.Length;

        // T can't appear as a subsequence in S
        if (m > n)
            return 0;

        // mat[i][j] stores the count of
        // occurrences of T(1..i) in S(1..j).
        int[,] mat = new int[m + 1, n + 1];

        // Initializing first column with
        // all 0s. An emptystring can't have
        // another string as subsequence
        for (int i = 1; i <= m; i++)
            mat[i, 0] = 0;

        // Initializing first row with all 1s.
        // An empty string is subsequence of all.
        for (int j = 0; j <= n; j++)
            mat[0, j] = 1;

        // Fill mat[][] in bottom up manner
        for (int i = 1; i <= m; i++) {

            for (int j = 1; j <= n; j++) {

                // If last characters don't match,
                // then value is same as the value
                // without last character in S.
                if (T[i - 1] != S[j - 1])
                    mat[i, j] = mat[i, j - 1];

                // Else value is obtained considering two cases.
                // a) All substrings without last character in S
```

```
// b) All substrings without last characters in
// both.
else
    mat[i, j] = mat[i, j - 1] + mat[i - 1, j - 1];
}
}

/* uncomment this to print matrix mat
for (int i = 1; i <= m; i++, cout << endl)
    for (int j = 1; j <= n; j++)
        System.out.println ( mat[i][j] +" ");
return mat[m, n];
}

// Driver code to check above method
public static void Main()
{
    string T = "ge";
    string S = "geeksforgeeks";
    Console.WriteLine(findSubsequenceCount(S, T));
}
}

// This code is contributed by vt_m
```

Output:

6

Time Complexity : $O(m*n)$
Auxiliary Space : $O(m*n)$

Since $mat[i][j]$ accesses elements of current row and previous row only, we can optimize auxiliary space just by using two rows only reducing space from $m*n$ to $2*n$.

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/count-distinct-occurrences-as-a-subsequence/>

Chapter 66

Count minimum number of subsets (or subsequences) with consecutive numbers

Count minimum number of subsets (or subsequences) with consecutive numbers - Geeks-forGeeks

Given an array of distinct positive numbers, the task is to calculate the number of subsets (or subsequences) from the array such that each subset contains consecutive numbers.

Examples:

```
Input : arr[] = {100, 56, 5, 6, 102, 58,
                 101, 57, 7, 103, 59}
Output : 3
{5, 6, 7}, {56, 57, 58, 59}, {100, 101, 102, 103}
are 3 subset in which numbers are consecutive.
```

```
Input : arr[] = {10, 100, 105}
Output : 3
{10}, {100} and {105} are 3 subset in which
numbers are consecutive.
```

The idea is to sort the array and traverse the sorted array to count the number of such subsets. To count the number of such subsets, we need to count the consecutive numbers such that difference between them is not equal to one.

Following is the algorithm for the finding number of subset containing consecutive numbers:

1. Sort the array arr[] and count = 1.

2. Traverse the sorted array and for each element arr[i].
If arr[i] + 1 != arr[i+1],
then increment the count by one.
3. Return the count.

Below is the implementation of this approach :

C++

```
// C++ program to find number of subset containing
// consecutive numbers
#include <bits/stdc++.h>
using namespace std;

// Returns count of subsets with consecutive numbers
int numofsubset(int arr[], int n)
{
    // Sort the array so that elements which are
    // consecutive in nature became consecutive
    // in the array.
    sort(arr, arr + n);

    int count = 1; // Initialize result
    for (int i = 0; i < n - 1; i++) {
        // Check if there is beginning of another
        // subset of consecutive number
        if (arr[i] + 1 != arr[i + 1])
            count++;
    }

    return count;
}

// Driven Program
int main()
{
    int arr[] = { 100, 56, 5, 6, 102, 58, 101,
                 57, 7, 103, 59 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << numofsubset(arr, n) << endl;
    return 0;
}
```

Java

```
// Java program to find number of subset
// containing consecutive numbers
import java.util.*;
```

```
class GFG {

    // Returns count of subsets with consecutive numbers
    static int numofsubset(int arr[], int n)
    {
        // Sort the array so that elements
        // which are consecutive in nature
        // became consecutive in the array.
        Arrays.sort(arr);

        // Initialize result
        int count = 1;
        for (int i = 0; i < n - 1; i++) {
            // Check if there is beginning
            // of another subset of
            // consecutive number
            if (arr[i] + 1 != arr[i + 1])
                count++;
        }

        return count;
    }

    // Driven Program
    public static void main(String[] args)
    {
        int arr[] = { 100, 56, 5, 6, 102, 58, 101,
                     57, 7, 103, 59 };
        int n = arr.length;
        System.out.println(numofsubset(arr, n));
    }
}

// This code is contributed by prerna saini.
```

Python

```
# Python program to find number of subset containing
# consecutive numbers
def numofsubset(arr, n):

    # Sort the array so that elements which are consecutive
    # in nature became consecutive in the array.
    x = sorted(arr)

    count = 1

    for i in range(0, n-1):
```

```
# Check if there is beginning of another subset of
# consecutive number
if (x[i] + 1 != x[i + 1]):
    count = count + 1

return count

# Driven Program
arr = [ 100, 56, 5, 6, 102, 58, 101, 57, 7, 103, 59 ]
n = len(arr)
print numofsubset(arr, n)

# This code is contributed by Afzal Ansari.
```

C#

```
// C# program to find number of subset
// containing consecutive numbers
using System;

class GFG {

    // Returns count of subsets with
    // consecutive numbers
    static int numofsubset(int[] arr, int n)
    {
        // Sort the array so that elements
        // which are consecutive in nature
        // became consecutive in the array.
        Array.Sort(arr);

        // Initialize result
        int count = 1;
        for (int i = 0; i < n - 1; i++) {

            // Check if there is beginning
            // of another subset of
            // consecutive number
            if (arr[i] + 1 != arr[i + 1])
                count++;
        }

        return count;
    }

    // Driven Program
    public static void Main()
    {
```

```
{  
    int[] arr = { 100, 56, 5, 6, 102, 58, 101,  
                57, 7, 103, 59 };  
    int n = arr.Length;  
    Console.WriteLine(numofsubset(arr, n));  
}  
}  
  
// This code is contributed by vt_m.
```

PHP

```
<?php  
// PHP program to find number  
// of subset containing  
// consecutive numbers  
  
// Returns count of subsets  
// with consecutive numbers  
function numofsubset( $arr, $n)  
{  
  
    // Sort the array so that  
    // elements which are  
    // consecutive in nature  
    // became consecutive  
    // in the array.  
    sort($arr);  
  
    // Initialize result  
    $count = 1;  
    for ( $i = 0; $i < $n - 1; $i++)  
    {  
  
        // Check if there is  
        // beginning of another  
        // subset of consecutive  
        // number  
        if ( $arr[$i] + 1 != $arr[$i + 1] )  
            $count++;  
    }  
  
    return $count;  
}  
  
// Driver Code  
$arr = array(100, 56, 5, 6, 102, 58, 101,  
            57, 7, 103, 59 );
```

```
$n = sizeof($arr);  
echo numofsubset($arr, $n);  
  
// This code is contributed by Anuj_67  
?>
```

Output:

3

Time Complexity : $O(n \log n)$

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/count-minimum-number-subsets subsequences-consecutive-numbers/>

Chapter 67

Count number of triplets in an array having sum in the range [a, b]

Count number of triplets in an array having sum in the range [a, b] - GeeksforGeeks

Given an array of distinct integers and a range [a, b], the task is to count the number of triplets having a sum in the range [a, b].

Examples:

```
Input : arr[] = {8, 3, 5, 2}
        range = [7, 11]
Output : 1
There is only one triplet {2, 3, 5}
having sum 10 in range [7, 11].
```

```
Input : arr[] = {2, 7, 5, 3, 8, 4, 1, 9}
        range = [8, 16]
Output : 36
```

A **naive** approach is to run three loops to consider all the triplets one by one. Find the sum of each triplet and increment the count if the sum lies in a given range [a, b].

Below is the implementation of above approach:

C++

```
// C++ program to count triplets with
// sum that lies in given range [a, b].
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to count triplets
int countTriplets(int arr[], int n, int a, int b)
{
    // Initialize result
    int ans = 0;

    // Fix the first element as A[i]
    for (int i = 0; i < n - 2; i++) {

        // Fix the second element as A[j]
        for (int j = i + 1; j < n - 1; j++) {

            // Now look for the third number
            for (int k = j + 1; k < n; k++)

                if (arr[i] + arr[j] + arr[k] >= a
                    && arr[i] + arr[j] + arr[k] <= b)
                    ans++;

        }
    }

    return ans;
}

// Driver Code
int main()
{
    int arr[] = { 2, 7, 5, 3, 8, 4, 1, 9 };
    int n = sizeof arr / sizeof arr[0];
    int a = 8, b = 16;
    cout << countTriplets(arr, n, a, b) << endl;
    return 0;
}
```

Java

```
// Java program to count triplets
// with sum that lies in given
// range [a, b].
import java.util.*;

class GFG
{

    // Function to count triplets
```

```
public static int countTriplets(int []arr, int n,
                               int a, int b)
{
    // Initialize result
    int ans = 0;

    // Fix the first
    // element as A[i]
    for (int i = 0; i < n - 2; i++)
    {

        // Fix the second
        // element as A[j]
        for (int j = i + 1; j < n - 1; j++)
        {

            // Now look for the
            // third number
            for (int k = j + 1; k < n; k++)
            {
                if (arr[i] + arr[j] + arr[k] >= a &&
                    arr[i] + arr[j] + arr[k] <= b)
                    {ans++;}
            }
        }
    }

    return ans;
}

// Driver Code
public static void main(String[] args)
{
    int[] arr = { 2, 7, 5, 3, 8, 4, 1, 9 };
    int n = arr.length;
    int a = 8, b = 16;
    System.out.println(" " + countTriplets(arr, n,
                                            a, b));
}
}

// This code is contributed
// by Harshit Saini
```

Python3

```
# Python3 program to count
# triplets with sum that
```

```
# lies in given range [a, b].  
  
# Function to count triplets  
def countTriplets(arr, n, a, b):  
  
    # Initialize result  
    ans = 0  
  
    # Fix the first  
    # element as A[i]  
    for i in range(0, n - 2):  
  
        # Fix the second  
        # element as A[j]  
        for j in range(i + 1, n - 1):  
  
            # Now look for  
            # the third number  
            for k in range(j + 1, n):  
  
                if ((arr[i] + arr[j] + arr[k] >= a)  
                    and (arr[i] + arr[j] + arr[k] <= b)):  
                    ans += 1  
  
    return ans
```

```
# Driver code  
if __name__ == "__main__":  
  
    arr = [ 2, 7, 5, 3, 8, 4, 1, 9 ]  
    n = len(arr)  
    a = 8; b = 16  
    print(countTriplets(arr, n, a, b))
```

```
# This code is contributed  
# by Harshit Saini
```

C#

```
// C# program to count triplets  
// with sum that lies in given  
// range [a, b].  
using System;  
  
class GFG  
{  
  
    // Function to count triplets
```

```
public static int countTriplets(int []arr, int n,
                               int a, int b)
{
    // Initialize result
    int ans = 0;

    // Fix the first
    // element as A[i]
    for (int i = 0;
         i < n - 2; i++)
    {

        // Fix the second
        // element as A[j]
        for (int j = i + 1;
             j < n - 1; j++)
        {

            // Now look for the
            // third number
            for (int k = j + 1;
                 k < n; k++)
            {
                if (arr[i] + arr[j] + arr[k] >= a &&
                    arr[i] + arr[j] + arr[k] <= b)
                    {ans++;}
            }
        }
    }

    return ans;
}

// Driver Code
public static void Main()
{
    int[] arr = {2, 7, 5, 3, 8, 4, 1, 9};
    int n = arr.Length;
    int a = 8, b = 16;
    Console.WriteLine(" " + countTriplets(arr, n,
                                         a, b));
}
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

```
<?php
// PHP program to count triplets with
// sum that lies in given range [a, b] .

// Function to count triplets
function countTriplets($arr, $n, $a, $b)
{
    // Initialize result
    $ans = 0;

    // Fix the first element as A[i]
    for ($i = 0; $i < $n - 2; $i++)
    {

        // Fix the second element as A[j]
        for ($j = $i + 1; $j < $n - 1; $j++)
        {

            // Now look for the third number
            for ($k = $j + 1; $k < $n; $k++)

                if ($arr[$i] + $arr[$j] + $arr[$k] >= $a &&
                    $arr[$i] + $arr[$j] + $arr[$k] <= $b)
                    $ans++;

        }
    }

    return $ans;
}

// Driver Code
$arr = array( 2, 7, 5, 3, 8, 4, 1, 9 );
$n = sizeof($arr);
$a = 8; $b = 16;
echo countTriplets($arr, $n, $a, $b) . "\n";

// This code is contributed
// by Akanksha Rai(Addy_akku)
?>
```

Output:

36

Time complexity: $O(n^3)$

An **efficient** solution is to first find the count of triplets having a sum less than or equal to upper limit b in the range [a, b]. This count of triplets will also include triplets having

a sum less than the lower limit a. Subtract the count of triplets having a sum less than a. The final result is the count of triplets having a sum in the range [a, b].

The algorithm is as follows:

- Find count of triplets having a sum less than or equal to b. Let this count be x.
- Find count of triplets having a sum less than a. Let this count be y.
- Final result is x-y.

To find the count of triplets having a sum less than or equal to given value, refer [Count triplets with sum smaller than a given value](#)

Below is the implementation of the above approach:

C++

```
// C++ program to count triplets with
// sum that lies in given range [a, b].
#include <bits/stdc++.h>

using namespace std;

// Function to find count of triplets having
// sum less than or equal to val.
int countTripletsLessThan(int arr[], int n, int val)
{
    // sort the input array.
    sort(arr, arr + n);

    // Initialize result
    int ans = 0;

    int j, k;

    // to store sum
    int sum;

    // Fix the first element
    for (int i = 0; i < n - 2; i++) {

        // Initialize other two elements as
        // corner elements of subarray arr[j+1..k]
        j = i + 1;
        k = n - 1;

        // Use Meet in the Middle concept.
        while (j != k) {
```

```
sum = arr[i] + arr[j] + arr[k];

// If sum of current triplet
// is greater, then to reduce it
// decrease k.
if (sum > val)
    k--;

// If sum is less than or equal
// to given value, then add
// possible triplets (k-j) to result.
else {
    ans += (k - j);
    j++;
}
}

return ans;
}

// Function to return count of triplets having
// sum in range [a, b].
int countTriplets(int arr[], int n, int a, int b)
{

    // to store count of triplets.
    int res;

    // Find count of triplets having sum less
    // than or equal to b and subtract count
    // of triplets having sum less than or
    // equal to a-1.
    res = countTripletsLessThan(arr, n, b) -
        countTripletsLessThan(arr, n, a - 1);

    return res;
}

// Driver Code
int main()
{
    int arr[] = { 2, 7, 5, 3, 8, 4, 1, 9 };
    int n = sizeof arr / sizeof arr[0];
    int a = 8, b = 16;
    cout << countTriplets(arr, n, a, b) << endl;
    return 0;
}
```

Java

```
// Java program to count triplets
// with sum that lies in given
// range [a, b].
import java.util.*;

class GFG
{
// Function to find count of
// triplets having sum less
// than or equal to val.
public static int countTripletsLessThan(int []arr,
                                         int n, int val)
{
    // sort the input array.
    Arrays.sort(arr);

    // Initialize result
    int ans = 0;

    int j, k;

    // to store sum
    int sum;

    // Fix the first element
    for (int i = 0; i < n - 2; i++)
    {

        // Initialize other two elements
        // as corner elements of subarray
        // arr[j+1..k]
        j = i + 1;
        k = n - 1;

        // Use Meet in the
        // Middle concept.
        while (j != k)
        {
            sum = arr[i] + arr[j] + arr[k];

            // If sum of current triplet
            // is greater, then to reduce it
            // decrease k.
            if (sum > val)
                k--;
            else
                ans++;
        }
    }
}
```

```
// If sum is less than or
// equal to given value,
// then add possible
// triplets (k-j) to result.
else
{
    ans += (k - j);
    j++;
}
}

return ans;
}

// Function to return count
// of triplets having sum
// in range [a, b].
public static int countTriplets(int arr[], int n,
                                int a, int b)
{
    // to store count
    // of triplets.
    int res;

    // Find count of triplets
    // having sum less than or
    // equal to b and subtract
    // count of triplets having
    // sum less than or equal
    // to a-1.
    res = countTripletsLessThan(arr, n, b) -
          countTripletsLessThan(arr, n, a - 1);

    return res;
}

// Driver Code
public static void main(String[] args)
{
    int[] arr = {2, 7, 5, 3,
                 8, 4, 1, 9};
    int n = arr.length;
    int a = 8, b = 16;
    System.out.println(" " + countTriplets(arr, n,
                                             a, b));
}
```

```
}
```

```
// This code is contributed
// by Harshit Saini
```

Python3

```
# Python program to count
# triplets with sum that
# lies in given range [a, b].

# Function to find count of
# triplets having sum less
# than or equal to val.
def countTripletsLessThan(arr, n, val):

    # sort the input array.
    arr.sort()

    # Initialize result
    ans = 0

    j = 0; k = 0

    # to store sum
    sum = 0

    # Fix the first element
    for i in range(0,n-2):

        # Initialize other two
        # elements as corner
        # elements of subarray
        # arr[j+1..k]
        j = i + 1
        k = n - 1

        # Use Meet in the
        # Middle concept.
        while j != k :
            sum = arr[i] + arr[j] + arr[k]

            # If sum of current triplet
            # is greater, then to reduce it
            # decrease k.
            if sum > val:
                k-=1
```

```
# If sum is less than or
# equal to given value,
# then add possible
# triplets (k-j) to result.
else :
    ans += (k - j)
    j += 1
return ans

# Function to return
# count of triplets having
# sum in range [a, b].
def countTriplets(arr, n, a, b):

    # to store count of triplets.
    res = 0

    # Find count of triplets
    # having sum less than or
    # equal to b and subtract
    # count of triplets having
    # sum less than or equal to a-1.
    res = (countTripletsLessThan(arr, n, b) -
           countTripletsLessThan(arr, n, a - 1))

return res
```

```
# Driver code
if __name__ == "__main__":
    arr = [ 2, 7, 5, 3, 8, 4, 1, 9 ]
    n = len(arr)
    a = 8; b = 16
    print(countTriplets(arr, n, a, b))
```

```
# This code is contributed by
# Harshit Saini
```

C#

```
// C# program to count triplets
// with sum that lies in given
// range [a, b].
using System;

class GFG
{
    // Function to find count of
```

```
// triplets having sum less
// than or equal to val.
public static int countTripletsLessThan(int[] arr,
                                         int n, int val)
{
    // sort the input array.
    Array.Sort(arr);

    // Initialize result
    int ans = 0;

    int j, k;

    // to store sum
    int sum;

    // Fix the first element
    for (int i = 0; i < n - 2; i++)
    {
        // Initialize other two elements
        // as corner elements of subarray
        // arr[j+1..k]
        j = i + 1;
        k = n - 1;

        // Use Meet in the
        // Middle concept.
        while (j != k)
        {
            sum = arr[i] + arr[j] + arr[k];

            // If sum of current triplet
            // is greater, then to reduce it
            // decrease k.
            if (sum > val)
                k--;

            // If sum is less than or
            // equal to given value,
            // then add possible
            // triplets (k-j) to result.
            else
            {
                ans += (k - j);
                j++;
            }
        }
    }
}
```

```
}

    return ans;
}

// Function to return count
// of triplets having sum
// in range [a, b].
public static int countTriplets(int[] arr, int n,
                                int a, int b)
{
    // to store count
    // of triplets.
    int res;

    // Find count of triplets
    // having sum less than or
    // equal to b and subtract
    // count of triplets having
    // sum less than or equal
    // to a-1.
    res = countTripletsLessThan(arr, n, b) -
          countTripletsLessThan(arr, n, a - 1);

    return res;
}

// Driver Code
public static void Main()
{
    int[] arr = {2, 7, 5, 3,
                 8, 4, 1, 9};
    int n = arr.Length;
    int a = 8, b = 16;
    Console.WriteLine(" " + countTriplets(arr, n,
                                         a, b));
}
}

// This code is contributed
// by Akanksha Rai(Addy_akku)
```

Output:

Time complexity: $O(n^2)$

Auxiliary space: $O(1)$

Improved By : [Harshit Saini](#), [Abby_akku](#)

Source

<https://www.geeksforgeeks.org/count-number-of-triplets-in-an-array-having-sum-in-the-range-a-b/>

Chapter 68

Count number of triplets with product equal to given number Set 2

Count number of triplets with product equal to given number Set 2 - GeeksforGeeks

Given an array of distinct integers(considering only positive numbers) and a number ‘m’, find the number of triplets with the product equal to ‘m’.

Examples:

Input: arr[] = { 1, 4, 6, 2, 3, 8}
m = 24

Output: 3

Input: arr[] = { 0, 4, 6, 2, 3, 8}
m = 18

Output: 0

An approach with O(n) extra space has already been discussed in previous [post](#). In this post an approach with O(1) space complexity will be discussed.

Approach: The idea is to use Three-pointer technique:

1. Sort the input array.
2. Fix the first element as A[i] where i is from 0 to array size – 2.
3. After fixing the first element of triplet, find the other two elements using [2 pointer technique](#).

Below is the implementation of above approach:

Source

<https://www.geeksforgeeks.org/count-number-of-triplets-with-product-equal-to-given-number-set-2-2/>

C++

```
// C++ implemntation of above approach
#include <bits/stdc++.h>
using namespace std;

// Function to count such triplets
int countTriplets(int arr[], int n, int m)
{
    int count = 0;

    // Sort the array
    sort(arr, arr + n);
    int end, start, mid;

    // three pointer technique
    for (end = n - 1; end >= 2; end--) {
        int start = 0, mid = end - 1;
        while (start < mid) {

            // Calculate the product of a triplet
            long int prod = arr[end] * arr[start] * arr[mid];

            // Check if that product is greater than m,
            // decrement mid
            if (prod > m)
                mid--;

            // Check if that product is greater than m,
            // increment start
            else if (prod < m)
                start++;

            // Check if that product is greater than m,
            // decrement mid, increment start and
            // increment the count of pairs
            else if (prod == m) {
                count++;
                mid--;
                start++;
            }
        }
    }
}
```

```
    return count;
}

// Drivers code
int main()
{
    int arr[] = { 1, 1, 1, 1, 1, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int m = 1;

    cout << countTriplets(arr, n, m);

    return 0;
}
```

Java

```
// Java implemntation of
// above approach
import java.io.*;
import java.util.*;

class GFG
{

// Function to count such triplets
static int countTriplets(int arr[],
                         int n, int m)
{
    int count = 0;

    // Sort the array
    Arrays.sort(arr);
    int end, start, mid;

    // three pointer technique
    for (end = n - 1; end >= 2; end--)
    {
        start = 0; mid = end - 1;
        while (start < mid)
        {

            // Calculate the product
            // of a triplet
            long prod = arr[end] *
                        arr[start] *
                        arr[mid];
```

```
// Check if that product
// is greater than m,
// decrement mid
if (prod > m)
    mid--;

// Check if that product
// is greater than m,
// increment start
else if (prod < m)
    start++;

// Check if that product
// is greater than m,
// decrement mid, increment
// start and increment the
// count of pairs
else if (prod == m)
{
    count++;
    mid--;
    start++;
}
}

return count;
}

// Driver code
public static void main (String[] args)
{
    int []arr = { 1, 1, 1, 1, 1, 1 };
    int n = arr.length;
    int m = 1;

    System.out.println(countTriplets(arr, n, m));
}
}

// This code is contributed
// by inder_verma.
```

C#

```
// C# implementation of above approach
using System;
class GFG
```

```
{  
    // Function to count such triplets  
    static int countTriplets(int []arr,  
                           int n, int m)  
    {  
        int count = 0;  
  
        // Sort the array  
        Array.Sort(arr);  
        int end, start, mid;  
  
        // three pointer technique  
        for (end = n - 1; end >= 2; end--)  
        {  
            start = 0; mid = end - 1;  
            while (start < mid) { // Calculate the product // of a triplet long prod = arr[end] * arr[start]  
                * arr[mid]; // Check if that product // is greater than m, // decrement mid if (prod > m)  
                mid--;  
  
                // Check if that product  
                // is greater than m,  
                // increment start  
                else if (prod < m) start++; // Check if that product // is greater than m, // decrement mid,  
                increment // start and increment the // count of pairs else if (prod == m) { count++; mid-;  
                start++; } } } return count; } // Driver code public static void Main (String []args) { int  
[]arr = { 1, 1, 1, 1, 1, 1 }; int n = arr.Length; int m = 1; Console.WriteLine(countTriplets(arr,  
n, m)); } } // This code is contributed // by Arnab Kundu [tabbyending]
```

Output:

6

Time complexity: O(N^2)

Space Complexity: O(1)

Improved By : [inderDuMCA](#), [andrew1234](#)

Chapter 69

Count of index pairs with equal elements in an array

Count of index pairs with equal elements in an array - GeeksforGeeks

Given an array of **n** elements. The task is to count the total number of indices (i, j) such that arr[i] = arr[j] and i != j

Examples :

Input : arr[] = {1, 1, 2}

Output : 1

As arr[0] = arr[1], the pair of indices is (0, 1)

Input : arr[] = {1, 1, 1}

Output : 3

As arr[0] = arr[1], the pair of indices is (0, 1),
(0, 2) and (1, 2)

Input : arr[] = {1, 2, 3}

Output : 0

Method 1 (Brute Force):

For each index i, find element after it with same value as arr[i]. Below is the implementation of this approach:

C++

```
// C++ program to count of pairs with equal
// elements in an array.
#include<bits/stdc++.h>
using namespace std;
```

```
// Return the number of pairs with equal
// values.
int countPairs(int arr[], int n)
{
    int ans = 0;

    // for each index i and j
    for (int i = 0; i < n; i++)
        for (int j = i+1; j < n; j++)

            // finding the index with same
            // value but different index.
            if (arr[i] == arr[j])
                ans++;

    return ans;
}

// Driven Program
int main()
{
    int arr[] = { 1, 1, 2 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << countPairs(arr, n) << endl;
    return 0;
}
```

Java

```
// Java program to count of pairs with equal
// elements in an array.
class GFG {

    // Return the number of pairs with equal
    // values.
    static int countPairs(int arr[], int n)
    {
        int ans = 0;

        // for each index i and j
        for (int i = 0; i < n; i++)
            for (int j = i+1; j < n; j++)

                // finding the index with same
                // value but different index.
                if (arr[i] == arr[j])
                    ans++;

        return ans;
    }
}
```

```
}

//driver code
public static void main (String[] args)
{
    int arr[] = { 1, 1, 2 };
    int n = arr.length;

    System.out.println(countPairs(arr, n));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program to
# count of pairs with equal
# elements in an array.

# Return the number of
# pairs with equal values.
def countPairs(arr, n):

    ans = 0

    # for each index i and j
    for i in range(0 , n):
        for j in range(i + 1, n):

            # finding the index
            # with same value but
            # different index.
            if (arr[i] == arr[j]):
                ans += 1
    return ans

# Driven Code
arr = [1, 1, 2 ]
n = len(arr)
print(countPairs(arr, n))

# This code is contributed
# by Smitha
```

C#

```
// C# program to count of pairs with equal
```

```
// elements in an array.  
using System;  
  
class GFG {  
  
    // Return the number of pairs with equal  
    // values.  
    static int countPairs(int []arr, int n)  
    {  
        int ans = 0;  
  
        // for each index i and j  
        for (int i = 0; i < n; i++)  
            for (int j = i+1; j < n; j++)  
  
                // finding the index with same  
                // value but different index.  
                if (arr[i] == arr[j])  
                    ans++;  
        return ans;  
    }  
  
    // Driver code  
    public static void Main ()  
    {  
        int []arr = { 1, 1, 2 };  
        int n = arr.Length;  
  
        Console.WriteLine(countPairs(arr, n));  
    }  
}  
  
// This code is contributed by anuj_67.
```

PHP

```
<?php  
// PHP program to count of  
// pairs with equal elements  
// in an array.  
  
// Return the number of pairs  
// with equal values.  
function countPairs( $arr, $n)  
{  
    $ans = 0;  
  
    // for each index i and j
```

```
for ( $i = 0; $i < $n; $i++)
    for ( $j = $i + 1; $j < $n; $j++)

        // finding the index with same
        // value but different index.
        if ($arr[$i] == $arr[$j])
            $ans++;
    return $ans;
}

// Driven Code
$arr = array( 1, 1, 2 );
$n = count($arr);
echo countPairs($arr, $n) ;

// This code is contributed by anuj_67.
?>
```

Output :

1

Time Complexity : $O(n^2)$

Method 2 (Efficient approach):

The idea is to count the frequency of each number and then find the number of pairs with equal elements. Suppose, a number x appears k times at index i_1, i_2, \dots, i_k . Then pick any two indexes i_x and i_y which will be counted as 1 pair. Similarly, i_y and i_x can also be pair. So, choose nC_2 is the number of pairs such that $\text{arr}[i] = \text{arr}[j] = x$.

Below is the implementation of this approach:

C++

```
// C++ program to count of index pairs with
// equal elements in an array.
#include<bits/stdc++.h>
using namespace std;

// Return the number of pairs with equal
// values.
int countPairs(int arr[], int n)
{
    unordered_map<int, int> mp;

    // Finding frequency of each number.
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;
}
```

```
// Calculating pairs of each value.  
int ans = 0;  
for (auto it=mp.begin(); it!=mp.end(); it++)  
{  
    int count = it->second;  
    ans += (count * (count - 1))/2;  
}  
  
return ans;  
}  
  
// Driven Program  
int main()  
{  
    int arr[] = {1, 1, 2};  
    int n = sizeof(arr)/sizeof(arr[0]);  
    cout << countPairs(arr, n) << endl;  
    return 0;  
}
```

Output :

1

Time Complexity : O(n)

Source:

http://stackoverflow.com/questions/26772364/efficient-algorithm-for-counting-number-of-pairs-of-identical-elements-in-an-array#comment42124861_26772516

Improved By : [ash_maurya](#), [vt_m](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/count-index-pairs-equal-elements-array/>

Chapter 70

Count pairs from two linked lists whose sum is equal to a given value

Count pairs from two linked lists whose sum is equal to a given value - GeeksforGeeks

Given two linked lists(can be sorted or unsorted) of size **n1** and **n2** of distinct elements. Given a value **x**. The problem is to count all pairs from both lists whose sum is equal to the given value **x**.

Note: The pair has an element from each linked list.

Examples:

```
Input : list1 = 3->1->5->7
        list2 = 8->2->5->3
        x = 10
Output : 2
The pairs are:
(5, 5) and (7, 3)

Input : list1 = 4->3->5->7->11->2->1
        list2 = 2->3->4->5->6->8-12
        x = 9
Output : 5
```

Method 1 (Naive Approach): Using two loops pick elements from both the linked lists and check whether the sum of the pair is equal to **x** or not.

C/C++

```
// C++ implementation to count pairs from both linked
// lists whose sum is equal to a given value
#include <bits/stdc++.h>
using namespace std;

/* A Linked list node */
struct Node
{
    int data;
    struct Node* next;
};

// function to insert a node at the
// beginning of the linked list
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list to the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

// function to count all pairs from both the linked lists
// whose sum is equal to a given value
int countPairs(struct Node* head1, struct Node* head2, int x)
{
    int count = 0;

    struct Node *p1, *p2;

    // traverse the 1st linked list
    for (p1 = head1; p1 != NULL; p1 = p1->next)

        // for each node of 1st list
        // traverse the 2nd list

        for (p2 = head2; p2 != NULL; p2 = p2->next)

            // if sum of pair is equal to 'x'
            // increment count
```

```
        if ((p1->data + p2->data) == x)
            count++;

        // required count of pairs
        return count;
    }

// Driver program to test above
int main()
{
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;

    // create linked list1 3->1->5->7
    push(&head1, 7);
    push(&head1, 5);
    push(&head1, 1);
    push(&head1, 3);

    // create linked list2 8->2->5->3
    push(&head2, 3);
    push(&head2, 5);
    push(&head2, 2);
    push(&head2, 8);

    int x = 10;

    cout << "Count = "
        << countPairs(head1, head2, x);
    return 0;
}
```

Java

```
// Java implementation to count pairs from both linked
// lists whose sum is equal to a given value

// Note : here we use java.util.LinkedList for
// linked list implementation

import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;

class GFG
{
    // method to count all pairs from both the linked lists
    // whose sum is equal to a given value
```

```
static int countPairs(LinkedList<Integer> head1, LinkedList<Integer> head2, int x)
{
    int count = 0;

    // traverse the 1st linked list
    Iterator<Integer> itr1 = head1.iterator();
    while(itr1.hasNext())
    {
        // for each node of 1st list
        // traverse the 2nd list
        Iterator<Integer> itr2 = head2.iterator();

        while(itr2.hasNext())
        {
            // if sum of pair is equal to 'x'
            // increment count
            if ((itr1.next() + itr2.next()) == x)
                count++;
        }
    }

    // required count of pairs
    return count;
}

// Driver method
public static void main(String[] args)
{
    Integer arr1[] = {3, 1, 5, 7};
    Integer arr2[] = {8, 2, 5, 3};

    // create linked list1 3->1->5->7
    LinkedList<Integer> head1 = new LinkedList<>(Arrays.asList(arr1));

    // create linked list2 8->2->5->3
    LinkedList<Integer> head2 = new LinkedList<>(Arrays.asList(arr2));

    int x = 10;

    System.out.println("Count = " + countPairs(head1, head2, x));
}
```

Output:

Count = 2

Time Complexity: $O(n_1 \cdot n_2)$

Auxiliary Space: $O(1)$

Method 2 (Sorting): Sort the 1st linked list in ascending order and the 2nd linked list in descending order using merge sort technique. Now traverse both the lists from left to right in the following way:

Algorithm:

```
countPairs(list1, list2, x)
    Initialize count = 0
    while list != NULL and list2 != NULL
        if (list1->data + list2->data) == x
            list1 = list1->next
            list2 = list2->next
            count++
        else if (list1->data + list2->data) > x
            list2 = list2->next
        else
            list1 = list1->next

    return count
```

For simplicity, the implementation given below assumes that list1 is sorted in ascending order and list2 is sorted in descending order.

C/C++

```
// C++ implementation to count pairs from both linked
// lists whose sum is equal to a given value
#include <bits/stdc++.h>
using namespace std;

/* A Linked list node */
struct Node
{
    int data;
    struct Node* next;
};

// function to insert a node at the
// beginning of the linked list
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =

```

```
(struct Node*) malloc(sizeof(struct Node));

/* put in the data */
new_node->data = new_data;

/* link the old list to the new node */
new_node->next = (*head_ref);

/* move the head to point to the new node */
(*head_ref) = new_node;
}

// function to count all pairs from both the linked
// lists whose sum is equal to a given value
int countPairs(struct Node* head1, struct Node* head2,
               int x)
{
    int count = 0;

    // sort head1 in ascending order and
    // head2 in descending order
    // sort (head1), sort (head2)
    // For simplicity both lists are considered to be
    // sorted in the respective orders

    // traverse both the lists from left to right
    while (head1 != NULL && head2 != NULL)
    {
        // if this sum is equal to 'x', then move both
        // the lists to next nodes and increment 'count'
        if ((head1->data + head2->data) == x)
        {
            head1 = head1->next;
            head2 = head2->next;
            count++;
        }

        // if this sum is greater than x, then
        // move head2 to next node
        else if ((head1->data + head2->data) > x)
            head2 = head2->next;

        // else move head1 to next node
        else
            head1 = head1->next;
    }

    // required count of pairs
}
```

```
    return count;
}

// Driver program to test above
int main()
{
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;

    // create linked list1 1->3->5->7
    // assumed to be in ascending order
    push(&head1, 7);
    push(&head1, 5);
    push(&head1, 3);
    push(&head1, 1);

    // create linked list2 8->5->3->2
    // assumed to be in descending order
    push(&head2, 2);
    push(&head2, 3);
    push(&head2, 5);
    push(&head2, 8);

    int x = 10;

    cout << "Count = "
        << countPairs(head1, head2, x);
    return 0;
}
```

Java

```
// Java implementation to count pairs from both linked
// lists whose sum is equal to a given value

// Note : here we use java.util.LinkedList for
// linked list implementation

import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedList;

class GFG
{
    // method to count all pairs from both the linked lists
    // whose sum is equal to a given value
    static int countPairs(LinkedList<Integer> head1, LinkedList<Integer> head2, int x)
```

```
{  
    int count = 0;  
  
    // sort head1 in ascending order and  
    // head2 in descending order  
    Collections.sort(head1);  
    Collections.sort(head2,Collections.reverseOrder());  
  
    // traverse both the lists from left to right  
    Iterator<Integer> itr1 = head1.iterator();  
    Iterator<Integer> itr2 = head2.iterator();  
  
    Integer num1 = itr1.hasNext() ? itr1.next() : null;  
    Integer num2 = itr2.hasNext() ? itr2.next() : null;  
  
    while(num1 != null && num2 != null)  
    {  
  
        // if this sum is equal to 'x', then move both  
        // the lists to next nodes and increment 'count'  
  
        if ((num1 + num2) == x)  
        {  
            num1 = itr1.hasNext() ? itr1.next() : null;  
            num2 = itr2.hasNext() ? itr2.next() : null;  
  
            count++;  
        }  
  
        // if this sum is greater than x, then  
        // move itr2 to next node  
        else if ((num1 + num2) > x)  
            num2 = itr2.hasNext() ? itr2.next() : null;  
  
        // else move itr1 to next node  
        else  
            num1 = itr1.hasNext() ? itr1.next() : null;  
  
    }  
  
    // required count of pairs  
    return count;  
}  
  
// Driver method  
public static void main(String[] args)  
{  
    Integer arr1[] = {3, 1, 5, 7};
```

```
Integer arr2[] = {8, 2, 5, 3};

// create linked list1 3->1->5->7
LinkedList<Integer> head1 = new LinkedList<>(Arrays.asList(arr1));

// create linked list2 8->2->5->3
LinkedList<Integer> head2 = new LinkedList<>(Arrays.asList(arr2));

int x = 10;

System.out.println("Count = " + countPairs(head1, head2, x));
}
}
```

Output:

```
Count = 2
```

Time Complexity: $O(n_1 \log n_1) + O(n_2 \log n_2)$

Auxiliary Space: $O(1)$

Sorting will change the order of nodes. If order is important, then copy of the linked lists can be created and used.

Method 3 (Hashing): Hash table is implemented using [unordered_set in C++](#). We store all first linked list elements in hash table. For elements of second linked list, we subtract every element from x and check the result in hash table. If result is present, we increment the **count**.

C++

```
// C++ implementation to count pairs from both linked
// lists whose sum is equal to a given value
#include <bits/stdc++.h>
using namespace std;

/* A Linked list node */
struct Node
{
    int data;
    struct Node* next;
};

// function to insert a node at the
// beginning of the linked list
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */

```

```
struct Node* new_node =
    (struct Node*) malloc(sizeof(struct Node));

/* put in the data */
new_node->data = new_data;

/* link the old list to the new node */
new_node->next = (*head_ref);

/* move the head to point to the new node */
(*head_ref) = new_node;
}

// function to count all pairs from both the linked
// lists whose sum is equal to a given value
int countPairs(struct Node* head1, struct Node* head2,
               int x)
{
    int count = 0;

    unordered_set<int> us;

    // insert all the elements of 1st list
    // in the hash table(unordered_set 'us')
    while (head1 != NULL)
    {
        us.insert(head1->data);

        // move to next node
        head1 = head1->next;
    }

    // for each element of 2nd list
    while (head2 != NULL)
    {
        // find (x - head2->data) in 'us'
        if (us.find(x - head2->data) != us.end())
            count++;

        // move to next node
        head2 = head2->next;
    }
    // required count of pairs
    return count;
}

// Driver program to test above
int main()
```

```
{  
    struct Node* head1 = NULL;  
    struct Node* head2 = NULL;  
  
    // create linked list1 3->1->5->7  
    push(&head1, 7);  
    push(&head1, 5);  
    push(&head1, 1);  
    push(&head1, 3);  
  
    // create linked list2 8->2->5->3  
    push(&head2, 3);  
    push(&head2, 5);  
    push(&head2, 2);  
    push(&head2, 8);  
  
    int x = 10;  
  
    cout << "Count = "  
        << countPairs(head1, head2, x);  
    return 0;  
}
```

Java

```
// Java implementation to count pairs from both linked  
// lists whose sum is equal to a given value  
  
// Note : here we use java.util.LinkedList for  
// linked list implementation  
  
import java.util.Arrays;  
import java.util.HashSet;  
import java.util.Iterator;  
import java.util.LinkedList;  
  
class GFG  
{  
    // method to count all pairs from both the linked lists  
    // whose sum is equal to a given value  
    static int countPairs(LinkedList<Integer> head1, LinkedList<Integer> head2, int x)  
    {  
        int count = 0;  
  
        HashSet<Integer> us = new HashSet<Integer>();  
  
        // insert all the elements of 1st list  
        // in the hash table(unordered_set 'us')  
}
```

```
Iterator<Integer> itr1 = head1.iterator();
while (itr1.hasNext())
{
    us.add(itr1.next());
}

Iterator<Integer> itr2 = head2.iterator();
// for each element of 2nd list
while (itr2.hasNext())
{
    // find (x - head2->data) in 'us'
    if (us.add(x - itr2.next()))
        count++;

}

// required count of pairs
return count;
}

// Driver method
public static void main(String[] args)
{
    Integer arr1[] = {3, 1, 5, 7};
    Integer arr2[] = {8, 2, 5, 3};

    // create linked list1 3->1->5->7
    LinkedList<Integer> head1 = new LinkedList<>(Arrays.asList(arr1));

    // create linked list2 8->2->5->3
    LinkedList<Integer> head2 = new LinkedList<>(Arrays.asList(arr2));

    int x = 10;

    System.out.println("Count = " + countPairs(head1, head2, x));
}
```

Output:

Count = 2

Time Complexity: O(n1 + n2)

Auxiliary Space: O(n1), hash table should be created of the array having smaller size so as to reduce the space complexity.

Source

<https://www.geeksforgeeks.org/count-pairs-two-linked-lists-whose-sum-equal-given-value/>

Chapter 71

Count pairs from two sorted arrays whose sum is equal to a given value x

Count pairs from two sorted arrays whose sum is equal to a given value x - GeeksforGeeks

Given two sorted arrays of size **m** and **n** of distinct elements. Given a value **x**. The problem is to count all pairs from both arrays whose sum is equal to **x**.

Note: The pair has an element from each array.

Examples :

```
Input : arr1[] = {1, 3, 5, 7}  
        arr2[] = {2, 3, 5, 8}  
        x = 10
```

```
Output : 2  
The pairs are:  
(5, 5) and (7, 3)
```

```
Input : arr1[] = {1, 2, 3, 4, 5, 7, 11}  
        arr2[] = {2, 3, 4, 5, 6, 8, 12}  
        x = 9
```

```
Output : 5
```

Method 1 (Naive Approach): Using two loops pick elements from both the arrays and check whether the sum of the pair is equal to **x** or not.

C++

```
// C++ implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value
#include <bits/stdc++.h>
using namespace std;

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
int countPairs(int arr1[], int arr2[],
               int m, int n, int x)
{
    int count = 0;

    // generating pairs from
    // both the arrays
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)

            // if sum of pair is equal
            // to 'x' increment count
            if ((arr1[i] + arr2[j]) == x)
                count++;

    // required count of pairs
    return count;
}

// Driver Code
int main()
{
    int arr1[] = {1, 3, 5, 7};
    int arr2[] = {2, 3, 5, 8};
    int m = sizeof(arr1) / sizeof(arr1[0]);
    int n = sizeof(arr2) / sizeof(arr2[0]);
    int x = 10;
    cout << "Count = "
         << countPairs(arr1, arr2, m, n, x);
    return 0;
}
```

Java

```
// Java implementation to count pairs from
// both sorted arrays whose sum is equal
// to a given value
```

```
import java.io.*;

class GFG {

    // function to count all pairs
    // from both the sorted arrays
    // whose sum is equal to a given
    // value
    static int countPairs(int []arr1,
                          int []arr2, int m, int n, int x)
    {
        int count = 0;

        // generating pairs from
        // both the arrays
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)

                // if sum of pair is equal
                // to 'x' increment count
                if ((arr1[i] + arr2[j]) == x)
                    count++;

        // required count of pairs
        return count;
    }

    // Driver Code

    public static void main (String[] args)
    {
        int arr1[] = {1, 3, 5, 7};
        int arr2[] = {2, 3, 5, 8};
        int m = arr1.length;
        int n = arr2.length;
        int x = 10;

        System.out.println( "Count = "
        + countPairs(arr1, arr2, m, n, x));
    }
}

// This code is contributed by anuj_67.
```

Python3

```
# python implementation to count
# pairs from both sorted arrays
```

```
# whose sum is equal to a given
# value

# function to count all pairs from
# both the sorted arrays whose sum
# is equal to a given value
def countPairs(arr1, arr2, m, n, x):
    count = 0

    # generating pairs from both
    # the arrays
    for i in range(m):
        for j in range(n):

            # if sum of pair is equal
            # to 'x' increment count
            if arr1[i] + arr2[j] == x:
                count = count + 1

    # required count of pairs
    return count

# Driver Program
arr1 = [1, 3, 5, 7]
arr2 = [2, 3, 5, 8]
m = len(arr1)
n = len(arr2)
x = 10
print("Count = ",
      countPairs(arr1, arr2, m, n, x))

# This code is contributed by Shrikant13.
```

C#

```
// C# implementation to count pairs from
// both sorted arrays whose sum is equal
// to a given value
using System;

class GFG {

    // function to count all pairs
    // from both the sorted arrays
    // whose sum is equal to a given
    // value
    static int countPairs(int []arr1,
                          int []arr2, int m, int n, int x)
```

```
{  
    int count = 0;  
  
    // generating pairs from  
    // both the arrays  
    for (int i = 0; i < m; i++)  
        for (int j = 0; j < n; j++)  
  
            // if sum of pair is equal  
            // to 'x' increment count  
            if ((arr1[i] + arr2[j]) == x)  
                count++;  
  
    // required count of pairs  
    return count;  
}  
  
// Driver Code  
  
public static void Main ()  
{  
    int []arr1 = {1, 3, 5, 7};  
    int []arr2 = {2, 3, 5, 8};  
    int m = arr1.Length;  
    int n = arr2.Length;  
    int x = 10;  
  
    Console.WriteLine( "Count = "  
        + countPairs(arr1, arr2, m, n, x));  
}  
}  
  
// This code is contributed by anuj_67.
```

PHP

```
<?php  
// PHP implementation to count  
// pairs from both sorted arrays  
// whose sum is equal to a given  
// value  
  
// function to count all pairs  
// from both the sorted arrays  
// whose sum is equal to a given  
// value  
function countPairs( $arr1, $arr2,
```

```
$m, $n, $x)
{
    $count = 0;

    // generating pairs from
    // both the arrays
    for ( $i = 0; $i < $m; $i++)
        for ( $j = 0; $j < $n; $j++)

            // if sum of pair is equal
            // to 'x' increment count
            if (($arr1[$i] + $arr2[$j]) == $x)
                $count++;

    // required count of pairs
    return $count;
}

// Driver Code
$arr1 = array(1, 3, 5, 7);
$arr2 = array(2, 3, 5, 8);
$m = count($arr1);
$n = count($arr2);
$x = 10;
echo "Count = ",
     countPairs($arr1, $arr2,
                $m,$n, $x);

// This code is contributed by anuj_67.
?>
```

Output :

```
Count = 2
```

Time Complexity : O(mn)

Auxiliary space : O(1)

Method 2 (Binary Search): For each element **arr1[i]**, where $1 \leq i \leq m$, search the value $(x - arr1[i])$ in **arr2[]**. If search is successful, increment the **count**.

C++

```
// C++ implementation to count
// pairs from both sorted arrays
```

```
// whose sum is equal to a given
// value
#include <bits/stdc++.h>
using namespace std;

// function to search 'value'
// in the given array 'arr[]'
// it uses binary search technique
// as 'arr[]' is sorted
bool isPresent(int arr[], int low,
               int high, int value)
{
    while (low <= high)
    {
        int mid = (low + high) / 2;

        // value found
        if (arr[mid] == value)
            return true;

        else if (arr[mid] > value)
            high = mid - 1;
        else
            low = mid + 1;
    }

    // value not found
    return false;
}

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
int countPairs(int arr1[], int arr2[],
               int m, int n, int x)
{
    int count = 0;
    for (int i = 0; i < m; i++)
    {
        // for each arr1[i]
        int value = x - arr1[i];

        // check if the 'value'
        // is present in 'arr2[]'
        if (isPresent(arr2, 0, n - 1, value))
            count++;
    }
}
```

```
// required count of pairs
return count;
}

// Driver Code
int main()
{
    int arr1[] = {1, 3, 5, 7};
    int arr2[] = {2, 3, 5, 8};
    int m = sizeof(arr1) / sizeof(arr1[0]);
    int n = sizeof(arr2) / sizeof(arr2[0]);
    int x = 10;
    cout << "Count = "
        << countPairs(arr1, arr2, m, n, x);
    return 0;
}
```

Java

```
// Java implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value
import java.io.*;
class GFG {

    // function to search 'value'
    // in the given array 'arr[]'
    // it uses binary search technique
    // as 'arr[]' is sorted
    static boolean isPresent(int arr[], int low,
                            int high, int value)
    {
        while (low <= high)
        {
            int mid = (low + high) / 2;

            // value found
            if (arr[mid] == value)
                return true;

            else if (arr[mid] > value)
                high = mid - 1;
            else
                low = mid + 1;
        }
    }
}
```

```
// value not found
return false;
}

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
static int countPairs(int arr1[], int arr2[],
                      int m, int n, int x)
{
    int count = 0;
    for (int i = 0; i < m; i++)
    {
        // for each arr1[i]
        int value = x - arr1[i];

        // check if the 'value'
        // is present in 'arr2[]'
        if (isPresent(arr2, 0, n - 1, value))
            count++;
    }

    // required count of pairs
    return count;
}

// Driver Code
public static void main (String[] args)
{
    int arr1[] = {1, 3, 5, 7};
    int arr2[] = {2, 3, 5, 8};
    int m = arr1.length;
    int n = arr2.length;
    int x = 10;
    System.out.println("Count = "
                       + countPairs(arr1, arr2, m, n, x));
}
}

// This code is contributed by anuj_67.
```

C#

```
// C# implementation to count pairs from both
// sorted arrays whose sum is equal to a given
// value
```

```
using System;

class GFG {

    // function to search 'value' in the given
    // array 'arr[]' it uses binary search
    // technique as 'arr[]' is sorted
    static bool isPresent(int []arr, int low,
                          int high, int value)
    {
        while (low <= high)
        {
            int mid = (low + high) / 2;

            // value found
            if (arr[mid] == value)
                return true;

            else if (arr[mid] > value)
                high = mid - 1;
            else
                low = mid + 1;
        }

        // value not found
        return false;
    }

    // function to count all pairs
    // from both the sorted arrays
    // whose sum is equal to a given
    // value
    static int countPairs(int []arr1, int []arr2,
                          int m, int n, int x)
    {
        int count = 0;

        for (int i = 0; i < m; i++)
        {

            // for each arr1[i]
            int value = x - arr1[i];

            // check if the 'value'
            // is present in 'arr2[]'
            if (isPresent(arr2, 0, n - 1, value))
                count++;
        }
    }
}
```

```
// required count of pairs
return count;
}

// Driver Code
public static void Main ()
{
    int []arr1 = {1, 3, 5, 7};
    int []arr2 = {2, 3, 5, 8};
    int m = arr1.Length;
    int n = arr2.Length;
    int x = 10;
    Console.WriteLine("Count = "
        + countPairs(arr1, arr2, m, n, x));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value

// function to search 'value'
// in the given array 'arr[]'
// it uses binary search technique
// as 'arr[]' is sorted
function isPresent($arr, $low,
                  $high, $value)
{
    while ($low <= $high)
    {
        $mid = ($low + $high) / 2;

        // value found
        if ($arr[$mid] == $value)
            return true;

        else if ($arr[$mid] > $value)
            $high = $mid - 1;
        else
            $low = $mid + 1;
    }
}
```

```
// value not found
return false;
}

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
function countPairs($arr1, $arr2,
                     $m, $n, $x)
{
    $count = 0;
    for ($i = 0; $i < $m; $i++)
    {
        // for each arr1[i]
        $value = $x - $arr1[$i];

        // check if the 'value'
        // is present in 'arr2[]'
        if (isPresent($arr2, 0,
                      $n - 1, $value))
            $count++;
    }

    // required count of pairs
    return $count;
}

// Driver Code
$arr1 = array(1, 3, 5, 7);
$arr2 = array(2, 3, 5, 8);
$m = count($arr1);
$n = count($arr2);
$x = 10;
echo "Count = "
     , countPairs($arr1, $arr2, $m, $n, $x);

// This code is contributed by anuj_67.
?>
```

Output :

Count = 2

Time Complexity : O(mlogn), searching should be applied on the array which is of greater

size so as to reduce the time complexity.

Auxiliary space : O(1)

Method 3 (Hashing): Hash table is implemented using [unordered_set in C++](#). We store all first array elements in hash table. For elements of second array, we subtract every element from x and check the result in hash table. If result is present, we increment the count.

C++

```
// C++ implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value
#include <bits/stdc++.h>
using namespace std;

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
int countPairs(int arr1[], int arr2[],
               int m, int n, int x)
{
    int count = 0;

    unordered_set<int> us;

    // insert all the elements
    // of 1st array in the hash
    // table(unordered_set 'us')
    for (int i = 0; i < m; i++)
        us.insert(arr1[i]);

    // for each element of 'arr2[]'
    for (int j = 0; j < n; j++)

        // find (x - arr2[j]) in 'us'
        if (us.find(x - arr2[j]) != us.end())
            count++;

    // required count of pairs
    return count;
}

// Driver Code
int main()
{
    int arr1[] = {1, 3, 5, 7};
```

```
int arr2[] = {2, 3, 5, 8};  
int m = sizeof(arr1) / sizeof(arr1[0]);  
int n = sizeof(arr2) / sizeof(arr2[0]);  
int x = 10;  
cout << "Count = "  
     << countPairs(arr1, arr2, m, n, x);  
return 0;  
}
```

Output :

```
Count = 2
```

Time Complexity : O(m+n)

Auxiliary space : O(m), hash table should be created of the array having smaller size so as to reduce the space complexity.

Method 4 (Efficient Approach): This approach uses the concept of two pointers, one to traverse 1st array from left to right and another to traverse the 2nd array from right to left.

Algorithm :

```
countPairs(arr1, arr2, m, n, x)  
  
    Initialize l = 0, r = n - 1  
    Initialize count = 0  
  
    loop while l = 0  
        if (arr1[l] + arr2[r]) == x  
            l++, r--  
            count++  
        else if (arr1[l] + arr2[r]) < x  
            l++  
        else  
            r--  
  
    return count
```

C++

```
// C++ implementation to count  
// pairs from both sorted arrays  
// whose sum is equal to a given  
// value
```

```
#include <bits/stdc++.h>
using namespace std;

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
int countPairs(int arr1[], int arr2[],
               int m, int n, int x)
{
    int count = 0;
    int l = 0, r = n - 1;

    // traverse 'arr1[]' from
    // left to right
    // traverse 'arr2[]' from
    // right to left
    while (l < m && r >= 0)
    {
        // if this sum is equal
        // to 'x', then increment 'l',
        // decrement 'r' and
        // increment 'count'
        if ((arr1[l] + arr2[r]) == x)
        {
            l++; r--;
            count++;
        }

        // if this sum is less
        // than x, then increment l
        else if ((arr1[l] + arr2[r]) < x)
            l++;

        // else decrement 'r'
        else
            r--;
    }

    // required count of pairs
    return count;
}

// Driver Code
int main()
{
    int arr1[] = {1, 3, 5, 7};
    int arr2[] = {2, 3, 5, 8};
```

```
int m = sizeof(arr1) / sizeof(arr1[0]);
int n = sizeof(arr2) / sizeof(arr2[0]);
int x = 10;
cout << "Count = "
     << countPairs(arr1, arr2, m, n, x);
return 0;
}
```

Java

```
// Java implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value
import java.io.*;

class GFG {

    // function to count all pairs
    // from both the sorted arrays
    // whose sum is equal to a given
    // value
    static int countPairs(int arr1[],
                          int arr2[], int m, int n, int x)
    {
        int count = 0;
        int l = 0, r = n - 1;

        // traverse 'arr1[]' from
        // left to right
        // traverse 'arr2[]' from
        // right to left
        while (l < m && r >= 0)
        {

            // if this sum is equal
            // to 'x', then increment 'l',
            // decrement 'r' and
            // increment 'count'
            if ((arr1[l] + arr2[r]) == x)
            {
                l++; r--;
                count++;
            }

            // if this sum is less
            // than x, then increment l
            else if ((arr1[l] + arr2[r]) < x)

```

```
        l++;

        // else decrement 'r'
        else
            r--;
    }

    // required count of pairs
    return count;
}

// Driver Code
public static void main (String[] args)
{
    int arr1[] = {1, 3, 5, 7};
    int arr2[] = {2, 3, 5, 8};
    int m = arr1.length;
    int n = arr2.length;
    int x = 10;
    System.out.println( "Count = "
        + countPairs(arr1, arr2, m, n, x));
}
}

// This code is contributed by anuj_67.
```

C#

```
// C# implementation to count
// pairs from both sorted arrays
// whose sum is equal to a given
// value
using System;

class GFG {

    // function to count all pairs
    // from both the sorted arrays
    // whose sum is equal to a given
    // value
    static int countPairs(int []arr1,
        int []arr2, int m, int n, int x)
    {
        int count = 0;
        int l = 0, r = n - 1;

        // traverse 'arr1[]' from
        // left to right
```

```
// traverse 'arr2[]' from
// right to left
while (l < m && r >= 0)
{
    // if this sum is equal
    // to 'x', then increment 'l',
    // decrement 'r' and
    // increment 'count'
    if ((arr1[l] + arr2[r]) == x)
    {
        l++; r--;
        count++;
    }

    // if this sum is less
    // than x, then increment l
    else if ((arr1[l] + arr2[r]) < x)
        l++;

    // else decrement 'r'
    else
        r--;
}

// required count of pairs
return count;
}

// Driver Code
public static void Main ()
{
    int []arr1 = {1, 3, 5, 7};
    int []arr2 = {2, 3, 5, 8};
    int m = arr1.Length;
    int n = arr2.Length;
    int x = 10;
    Console.WriteLine( "Count = "
    + countPairs(arr1, arr2, m, n, x));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP implementation to count
```

```
// pairs from both sorted arrays
// whose sum is equal to a given
// value

// function to count all pairs
// from both the sorted arrays
// whose sum is equal to a given
// value
function countPairs( $arr1, $arr2,
                     $m, $n, $x)
{
    $count = 0;
    $l = 0; $r = $n - 1;

    // traverse 'arr1[]' from
    // left to right
    // traverse 'arr2[]' from
    // right to left
    while ($l < $m and $r >= 0)
    {
        // if this sum is equal
        // to 'x', then increment 'l',
        // decrement 'r' and
        // increment 'count'
        if (($arr1[$l] + $arr2[$r]) == $x)
        {
            $l++; $r--;
            $count++;
        }

        // if this sum is less
        // than x, then increment l
        else if (($arr1[$l] + $arr2[$r]) < $x)
            $l++;

        // else decrement 'r'
        else
            $r--;
    }

    // required count of pairs
    return $count;
}

// Driver Code
$arr1 = array(1, 3, 5, 7);
$arr2 = array(2, 3, 5, 8);
```

```
$m = count($arr1);
$n = count($arr2);
$x = 10;
echo "Count =
, countPairs($arr1, $arr2, $m, $n, $x);
// This code is contributed by anuj_67

?>
```

Output :

Count = 2

Time Complexity : $O(m + n)$
Auxiliary space : $O(1)$

Improved By : [shrikanth13](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/count-pairs-two-sorted-arrays-whose-sum-equal-given-value-x/>

Chapter 72

Count pairs from two sorted matrices with given sum

Count pairs from two sorted matrices with given sum - GeeksforGeeks

Given two sorted matrices **mat1** and **mat2** of size **n x n** of distinct elements. Given a value **x**. The problem is to count all pairs from both matrices whose sum is equal to **x**.

Note: The pair has an element from each matrix. Matrices are strictly sorted which means that matrices are sorted in a way such that all elements in a row are sorted in increasing order and for row 'i', where $1 \leq i \leq n-1$, first element of row 'i' is greater than the last element of row 'i-1'.

Examples:

```
Input : mat1[] [] = { {1, 5, 6},  
                      {8, 10, 11},  
                      {15, 16, 18} }  
  
        mat2[] [] = { {2, 4, 7},  
                      {9, 10, 12},  
                      {13, 16, 20} }  
  
        x = 21  
Output : 4  
The pairs are:  
(1, 20), (5, 16), (8, 13) and (11, 10).
```

Method 1 (Naive Approach): For each element **ele** of **mat1[][]** linearly search $(x - ele)$ in **mat2[][]**.

C++

```
// C++ implementation to count pairs from two
```

```
// sorted matrices whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function to search 'val' in mat[] []
// returns true if 'val' is present
// else false
bool valuePresent(int mat[] [SIZE], int n, int val)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (mat[i][j] == val)

                // 'val' found
                return true;

    // 'val' not found
    return false;
}

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[] [SIZE], int mat2[] [SIZE],
               int n, int x)
{
    int count = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)

            // if value (x-mat1[i][j]) is found in mat2[] []
            if (valuePresent(mat2, n, x - mat1[i][j]))
                count++;

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    int mat1[] [SIZE] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };
}
```

```
int mat2[] [SIZE] = { { 2, 4, 7 },
                      { 9, 10, 12 },
                      { 13, 16, 20 } };

int n = 3;
int x = 21;

cout << "Count = "
     << countPairs(mat1, mat2, n, x);

return 0;
}
```

Java

```
// java implementation to count
// pairs from twosorted matrices
// whose sum is equal to a given value
import java.io.*;

class GFG
{
    int SIZE= 10;

    // function to search 'val' in mat[] []
    // returns true if 'val' is present
    // else false
    static boolean valuePresent(int mat[][], int n,
                                int val)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (mat[i][j] == val)

                    // 'val' found
                    return true;

        // 'val' not found
        return false;
    }

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int mat1[][], int mat2[][],
                          int n, int x)
    {
```

```
int count = 0;

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
    {
        // if value (x-mat1[i][j]) is
        // found in mat2[][][]
        if (valuePresent(mat2, n, x - mat1[i][j]))
            count++;
    }
    // required count of pairs
    return count;
}

// Driver program
public static void main (String[] args)
{
    int mat1[][] = { { 1, 5, 6 },
                    { 8, 10, 11 },
                    { 15, 16, 18 } };

    int mat2[][] = { { 2, 4, 7 },
                    { 9, 10, 12 },
                    { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    System.out.println ("Count = " +
                        countPairs(mat1, mat2, n, x));
}

// This article is contributed by vt_m
```

Python3

```
# Python3 implementation to count pairs
# from two sorted matrices whose sum is
# equal to a given value x

# function to search 'val' in mat[][]
# returns true if 'val' is present else
# false
def valuePresent(mat, n, val):

    for i in range(0, n):
```

```
for j in range(0, n):

    if mat[i][j] == val:

        # 'val' found
        return True

    # 'val' not found
    return False

# function to count pairs from two sorted
# matrices whose sum is equal to a given
# value x
def countPairs(mat1, mat2, n, x):

    count = 0

    for i in range(0, n):
        for j in range(0, n):

            # if value (x-mat1[i][j]) is found
            # in mat2[] []
            if valuePresent(mat2, n, x - mat1[i][j]):
                count += 1

    # required count of pairs
    return count

# Driver program
mat1 = [[ 1, 5, 6 ],
        [ 8, 10, 11 ],
        [ 15, 16, 18 ] ]

mat2 = [ [ 2, 4, 7 ],
        [ 9, 10, 12 ],
        [ 13, 16, 20 ] ]

n = 3
x = 21

print( "Count = "),
print(countPairs(mat1, mat2, n, x))

# This code is contributed by upendra bartwal

C#
//C# implementation to count
```

```
// pairs from twosorted matrices
// whose sum is equal to a given value
using System;

class GFG
{
    // int SIZE= 10;

    // function to search 'val' in mat[] []
    // returns true if 'val' is present
    // else false
    static bool valuePresent(int[,] mat, int n,
                                int val)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (mat[i, j] == val)

                    // 'val' found
                    return true;

        // 'val' not found
        return false;
    }

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int [,]mat1, int [,]mat2,
                          int n, int x)
    {
        int count = 0;

        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
            {
                // if value (x-mat1[i][j]) is
                // found in mat2[] []
                if (valuePresent(mat2, n, x - mat1[i, j]))
                    count++;

            }
        // required count of pairs
        return count;
    }

    // Driver program
    public static void Main ()
    {
```

```
int [,]mat1 = { { 1, 5, 6 },
                { 8, 10, 11 },
                { 15, 16, 18 } };

int [,]mat2 = { { 2, 4, 7 },
                { 9, 10, 12 },
                { 13, 16, 20 } };

int n = 3;
int x = 21;

Console.WriteLine("Count = " +
                  countPairs(mat1, mat2, n, x));

}

}

// This article is contributed by vt_m
```

Output:

```
Count = 4
```

Time Complexity: $O(n^4)$.

Auxiliary Space: $O(1)$.

Method 2 (Binary Search): As matrix is strictly sorted, use the concept of binary search technique. For each element **ele** of **mat1[][],** apply the binary search technique on the elements of the first column of **mat2[][],** to find the row index number of the largest element smaller than equal to $(x - ele)$. Let it be **row_no.** If no such row exists then no pair can be formed with element **ele.** Else apply the concept of binary search technique to find the value $(x - ele)$ in the row represented by **row_no** in **mat2[][],** If value found then increment **count.**

C++

```
// C++ implementation to count pairs from two
// sorted matrices whose sum is equal to a given
// value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function returns the row index no of largest
// element smaller than equal to 'x' in first
```

```
// column of mat[][] . If no such element exists
// then it returns -1.
int binarySearchOnRow(int mat[SIZE][SIZE],
                      int l, int h, int x)
{
    while (l <= h) {
        int mid = (l + h) / 2;

        // if 'x' is greater than or equal to mat[mid][0],
        // then search in mat[mid+1...h][0]
        if (mat[mid][0] <= x)
            l = mid + 1;

        // else search in mat[l...mid-1][0]
        else
            h = mid - 1;
    }

    // required row index number
    return h;
}

// function to search 'val' in mat[row] []
bool binarySearchOnCol(int mat[] [SIZE], int l, int h,
                      int val, int row)
{
    while (l <= h) {
        int mid = (l + h) / 2;

        // 'val' found
        if (mat[row][mid] == val)
            return true;

        // search in mat[row][mid+1...h]
        else if (mat[row][mid] < val)
            l = mid + 1;

        // search in mat[row][l...mid-1]
        else
            h = mid - 1;
    }

    // 'val' not found
    return false;
}

// function to search 'val' in mat[] []
// returns true if 'val' is present
```

```
// else false
bool searchValue(int mat[] [SIZE],
                 int n, int val)
{
    // to get the row index number of the largest element
    // smaller than equal to 'val' in mat[] []
    int row_no = binarySearchOnRow(mat, 0, n - 1, val);

    // if no such row exists, then
    // 'val' is not present
    if (row_no == -1)
        return false;

    // to search 'val' in mat[row_no] []
    return binarySearchOnCol(mat, 0, n - 1, val, row_no);
}

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[] [SIZE], int mat2[] [SIZE],
               int n, int x)
{
    int count = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            // if value (x-mat1[i][j]) is found in mat2[] []
            if (searchValue(mat2, n, x - mat1[i][j]))
                count++;

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    int mat1[] [SIZE] = { { 1, 5, 6 },
                         { 8, 10, 11 },
                         { 15, 16, 18 } };

    int mat2[] [SIZE] = { { 2, 4, 7 },
                         { 9, 10, 12 },
                         { 13, 16, 20 } };

    int n = 3;
    int x = 21;
```

```
    cout << "Count = "
        << countPairs(mat1, mat2, n, x);

    return 0;
}
```

Java

```
// java implementation to count
// pairs from two sorted matrices
// whose sum is equal to a given
// value x
import java.io.*;

class GFG {
    int SIZE= 10;

    // function returns the row index no of largest
    // element smaller than equal to 'x' in first
    // column of mat[][] . If no such element exists
    // then it returns -1.
    static int binarySearchOnRow(int mat[][], int l,
                                 int h, int x)
    {
        while (l <= h)
        {
            int mid = (l + h) / 2;

            // if 'x' is greater than or
            // equal to mat[mid][0] , then
            // search in mat[mid+1...h][0]
            if (mat[mid][0] <= x)
                l = mid + 1;

            // else search in mat[l...mid-1][0]
            else
                h = mid - 1;
        }

        // required row index number
        return h;
    }

    // function to search 'val' in mat[row] []
    static boolean binarySearchOnCol(int mat[][], int l, int h,
                                    int val, int row)
    {
        while (l <= h)
```

```
{  
    int mid = (l + h) / 2;  
  
    // 'val' found  
    if (mat[row][mid] == val)  
        return true;  
  
    // search in mat[row][mid+1...h]  
    else if (mat[row][mid] < val)  
        l = mid + 1;  
  
    // search in mat[row][l...mid-1]  
    else  
        h = mid - 1;  
}  
  
// 'val' not found  
return false;  
}  
  
// function to search 'val' in mat[][]  
// returns true if 'val' is present  
// else false  
static boolean searchValue(int mat[][],  
                           int n, int val)  
{  
    // to get the row index number  
    // of the largest element smaller  
    // than equal to 'val' in mat[][]  
    int row_no = binarySearchOnRow(mat, 0, n - 1, val);  
  
    // if no such row exists, then  
    // 'val' is not present  
    if (row_no == -1)  
        return false;  
  
    // to search 'val' in mat[row_no][]  
    return binarySearchOnCol(mat, 0, n - 1, val, row_no);  
}  
  
// function to count pairs from  
// two sorted matrices whose sum  
// is equal to a given value x  
static int countPairs(int mat1[][], int mat2[][],  
                      int n, int x)  
{  
    int count = 0;
```

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
    {
        // if value (x-mat1[i][j]) is found in mat2[] []
        if (searchValue(mat2, n, x - mat1[i][j]))
            count++;
    }
    // required count of pairs
    return count;
}

// Driver program
public static void main (String[] args)
{
    int mat1[][] = { { 1, 5, 6 },
                    { 8, 10, 11 },
                    { 15, 16, 18 } };

    int mat2[][] = { { 2, 4, 7 },
                    { 9, 10, 12 },
                    { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    System.out.println ( "Count = " +
                        countPairs(mat1, mat2, n, x));
}

}

// This code is contributed by vt_m
```

C#

```
// C# implementation to count
// pairs from two sorted matrices
// whose sum is equal to a given
// value x
using System;

class GFG
{
    //int SIZE= 10;

    // function returns the row index no of largest
    // element smaller than equal to 'x' in first
    // column of mat[] []. If no such element exists
```

```
// then it returns -1.
static int binarySearchOnRow(int [,]mat, int l,
                             int h, int x)
{
    while (l <= h)
    {
        int mid = (l + h) / 2;

        // if 'x' is greater than or
        // equal to mat[mid][0], then
        // search in mat[mid+1...h][0]
        if (mat[mid,0] <= x)
            l = mid + 1;

        // else search in mat[l...mid-1][0]
        else
            h = mid - 1;
    }

    // required row index number
    return h;
}

// function to search 'val' in mat[row] []
static bool binarySearchOnCol(int [,]mat, int l, int h,
                             int val, int row)
{
    while (l <= h)
    {
        int mid = (l + h) / 2;

        // 'val' found
        if (mat[row,mid] == val)
            return true;

        // search in mat[row][mid+1...h]
        else if (mat[row,mid] < val)
            l = mid + 1;

        // search in mat[row][l...mid-1]
        else
            h = mid - 1;
    }

    // 'val' not found
    return false;
}
```

```
// function to search 'val' in mat[] []
// returns true if 'val' is present
// else false
static bool searchValue(int [,]mat,
                       int n, int val)
{
    // to get the row index number
    // of the largest element smaller
    // than equal to 'val' in mat[] []
    int row_no = binarySearchOnRow(mat, 0, n - 1, val);

    // if no such row exists, then
    // 'val' is not present
    if (row_no == -1)
        return false;

    // to search 'val' in mat[row_no] []
    return binarySearchOnCol(mat, 0, n - 1, val, row_no);
}

// function to count pairs from
// two sorted matrices whose sum
// is equal to a given value x
static int countPairs(int [,]mat1, int [,]mat2,
                      int n, int x)
{
    int count = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
    {
        // if value (x-mat1[i][j]) is found in mat2[] []
        if (searchValue(mat2, n, x - mat1[i,j]))
            count++;
    }
    // required count of pairs
    return count;
}

// Driver program
public static void Main ()
{
    int [,]mat1 = { { 1, 5, 6 },
                   { 8, 10, 11 },
                   { 15, 16, 18 } };

    int [,]mat2 = { { 2, 4, 7 },
                   { 9, 10, 12 },
                   { 11, 13, 14 } };
}
```

```
{ 13, 16, 20 }};

int n = 3;
int x = 21;

Console.WriteLine ( "Count = " +
                    countPairs(mat1, mat2, n, x));

}

}

// This code is contributed by vt_m
```

Output:

```
Count = 4
```

Time Complexity: $(n^2 \log_2 n)$.
Auxiliary Space: $O(1)$.

Method 3 (Hashing): Create a hash table and insert all the elements of $\text{mat2}[][]$ in it. Now for each element ele of $\text{mat1}[][]$ find $(x - \text{ele})$ in the hash table.

```
// C++ implementation to count pairs from two
// sorted matrices whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[] [SIZE], int mat2[] [SIZE],
               int n, int x)
{
    int count = 0;

    // unordered_set 'us' implemented as hash table
    unordered_set<int> us;

    // insert all the elements of mat2[] [] in 'us'
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            us.insert(mat2[i][j]);
```

```

// for each element of mat1[] []
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)

        // if (x-mat1[i][j]) is in 'us'
        if (us.find(x - mat1[i][j]) != us.end())
            count++;

    // required count of pairs
    return count;
}

// Driver program to test above
int main()
{
    int mat1[] [SIZE] = { { 1, 5, 6 },
                        { 8, 10, 11 },
                        { 15, 16, 18 } };

    int mat2[] [SIZE] = { { 2, 4, 7 },
                        { 9, 10, 12 },
                        { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    cout << "Count = "
        << countPairs(mat1, mat2, n, x);

    return 0;
}

```

Output:

Count = 4

Time complexity: $O(n^2)$.

Auxiliary Space: $O(n^2)$.

Method 4 (Efficient Approach): From the top leftmost element traverse mat1[][] in forward direction (i.e., from the topmost row up to last, each row is being traversed from left to right) and from the bottom rightmost element traverse mat2[][] in backward direction (i.e., from the bottom row up to first, each row is being traversed from right to left). For each element e1 of mat1[][] and e2 of mat2[][] encountered, calculate val = (e1 + e2). If val == x, increment count. Else if val is less than x, move to next element of mat1[][] in forward direction. Else move to next element of mat2[][] in backward direction. Continue this process until either of the two matrices gets completely traversed.

C++

```
// C++ implementation to count pairs from two
// sorted matrices whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

#define SIZE 10

// function to count pairs from two sorted matrices
// whose sum is equal to a given value x
int countPairs(int mat1[][] [SIZE], int mat2[][] [SIZE],
               int n, int x)
{
    // 'r1' and 'c1' for pointing current element
    // of mat1[] []
    // 'r2' and 'c2' for pointing current element
    // of mat2[] []
    int r1 = 0, c1 = 0;
    int r2 = n - 1, c2 = n - 1;

    // while there are more elements
    // in both the matrices
    int count = 0;
    while ((r1 < n) && (r2 >= -1)) {
        int val = mat1[r1][c1] + mat2[r2][c2];

        // if true
        if (val == x) {

            // increment 'count'
            count++;

            // move mat1[] [] column 'c1' to right
            // move mat2[] [] column 'c2' to left
            c1++;
            c2--;
        }

        // if true, move mat1[] [] column 'c1' to right
        else if (val < x)
            c1++;

        // else move mat2[] [] column 'c2' to left
        else
            c2--;

        // if 'c1' crosses right boundary
    }
}
```

```
if (c1 == n) {  
  
    // reset 'c1'  
    c1 = 0;  
  
    // increment row 'r1'  
    r1++;  
}  
  
// if 'c2' crosses left boundary  
if (c2 == -1) {  
  
    // reset 'c2'  
    c2 = n - 1;  
  
    // decrement row 'r2'  
    r2--;  
}  
}  
  
// required count of pairs  
return count;  
}  
  
// Driver program to test above  
int main()  
{  
    int mat1[] [SIZE] = { { 1, 5, 6 },  
                         { 8, 10, 11 },  
                         { 15, 16, 18 } };  
  
    int mat2[] [SIZE] = { { 2, 4, 7 },  
                         { 9, 10, 12 },  
                         { 13, 16, 20 } };  
  
    int n = 3;  
    int x = 21;  
  
    cout << "Count = "  
        << countPairs(mat1, mat2, n, x);  
  
    return 0;  
}
```

Java

```
// java implementation to count  
// pairs from two sorted
```

```
// matrices whose sum is
// equal to a given value x
import java.io.*;

class GFG
{
    int SIZE = 10;

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int mat1[][] , int mat2[][] ,
                           int n, int x)
    {
        // 'r1' and 'c1' for pointing current
        // element of mat1[][]
        // 'r2' and 'c2' for pointing current
        // element of mat2[][]
        int r1 = 0, c1 = 0;
        int r2 = n - 1, c2 = n - 1;

        // while there are more elements
        // in both the matrices
        int count = 0;
        while ((r1 < n) && (r2 >= -1))
        {
            int val = mat1[r1][c1] + mat2[r2][c2];

            // if true
            if (val == x) {

                // increment 'count'
                count++;

                // move mat1[][] column 'c1' to right
                // move mat2[][] column 'c2' to left
                c1++;
                c2--;
            }

            // if true, move mat1[][] []
            // column 'c1' to right
            else if (val < x)
                c1++;

            // else move mat2[][] column
            // 'c2' to left
            else

```

```
c2--;

// if 'c1' crosses right boundary
if (c1 == n) {

    // reset 'c1'
    c1 = 0;

    // increment row 'r1'
    r1++;
}

// if 'c2' crosses left boundary
if (c2 == -1) {

    // reset 'c2'
    c2 = n - 1;

    // decrement row 'r2'
    r2--;
}
}

// required count of pairs
return count;
}

// Driver code
public static void main (String[] args)
{
    int mat1[][] = { { 1, 5, 6 },
                    { 8, 10, 11 },
                    { 15, 16, 18 } };

    int mat2[][] = { { 2, 4, 7 },
                    { 9, 10, 12 },
                    { 13, 16, 20 } };

    int n = 3;
    int x = 21;

    System.out.println ( "Count = " +
                        countPairs(mat1, mat2, n, x));
}
}

// This article is contributed by vt_m
```

C#

```
// C# implementation to count pairs
// from two sorted matrices whose
// sum is equal to a given value x
using System;

class GFG {

    // function to count pairs from
    // two sorted matrices whose sum
    // is equal to a given value x
    static int countPairs(int [,]mat1,
                          int [,]mat2, int n, int x)
    {

        // 'r1' and 'c1' for pointing
        // current element of mat1[] []
        // 'r2' and 'c2' for pointing
        // current element of mat2[] []
        int r1 = 0, c1 = 0;
        int r2 = n - 1, c2 = n - 1;

        // while there are more elements
        // in both the matrices
        int count = 0;
        while ((r1 < n) && (r2 >= -1))
        {
            int val = mat1[r1,c1]
                      + mat2[r2,c2];

            // if true
            if (val == x) {

                // increment 'count'
                count++;

                // move mat1[] [] column
                // 'c1' to right
                // move mat2[] [] column
                // 'c2' to left
                c1++;
                c2--;
            }
        }

        // if true, move mat1[] []
        // column 'c1' to right
        else if (val < x)
```

```
c1++;

// else move mat2[][] column
// 'c2' to left
else
    c2--;

// if 'c1' crosses right
// boundary
if (c1 == n) {

    // reset 'c1'
    c1 = 0;

    // increment row 'r1'
    r1++;
}

// if 'c2' crosses left
// boundary
if (c2 == -1) {

    // reset 'c2'
    c2 = n - 1;

    // decrement row 'r2'
    r2--;
}

// required count of pairs
return count;
}

// Driver code
public static void Main ()
{
    int [,]mat1 = { { 1, 5, 6 },
                    { 8, 10, 11 },
                    { 15, 16, 18 } };

    int [,]mat2 = { { 2, 4, 7 },
                    { 9, 10, 12 },
                    { 13, 16, 20 } };

    int n = 3;
    int x = 21;
```

```
Console.Write ( "Count = " +
    countPairs(mat1, mat2, n, x));

}

// This code is contributed by
// nitin mittal
```

Output:

```
Count = 4
```

Time Complexity: $O(n^2)$.
Auxiliary Space: $O(1)$.

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/count-pairs-two-sorted-matrices-given-sum/>

Chapter 73

Count pairs in an array that hold $i * arr[i] > j * arr[j]$

Count pairs in an array that hold $i * arr[i] > j * arr[j]$ - GeeksforGeeks

Given an array of integers $arr[0..n-1]$, count all pairs $(arr[i], arr[j])$ in the such that $i * arr[i] > j * arr[j]$, $0 \leq i < j \leq n$.

Examples :

```
Input : arr[] = {5 , 0, 10, 2, 4, 1, 6}
Output: 5
Pairs which hold condition i*arr[i] > j*arr[j]
are (10, 2) (10, 4) (10, 1) (2, 1) (4, 1)
```

```
Input : arr[] = {8, 4, 2, 1}
Output : 2
```

A **Simple solution** is to run two loops. Pick each element of array one-by-one and for each element find element on right side of array that hold condition, then increment counter and last return counter value.

Below is the implementation of above idea:

C++

```
// C++ program to count all pair that
// hold condition i*arr[i] > j*arr[j]
#include<iostream>
using namespace std;

// Return count of pair in given array
// such that i*arr[i] > j*arr[j]
```

```
int CountPair(int arr[] , int n )
{
    int result = 0; // Initialize result

    for (int i=0; i<n; i++)
    {
        // Generate all pair and increment
        // counter if the hold given condition
        for (int j = i + 1; j < n; j++)
            if (i*arr[i] > j*arr[j] )
                result++;
    }
    return result;
}

// Driver code
int main()
{
    int arr[] = {5 , 0, 10, 2, 4, 1, 6} ;
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Count of Pairs : "
         << CountPair(arr, n);
    return 0;
}
```

Java

```
// Java Code for Count pairs in an
// array that hold  $i*arr[i] > j*arr[j]$ 
class GFG {

    // Return count of pair in given array
    // such that  $i*arr[i] > j*arr[j]$ 
    public static int CountPair(int arr[] , int n )
    {
        int result = 0; // Initialize result

        for (int i = 0; i < n; i++)
        {
            // Generate all pair and increment
            // counter if the hold given condition
            for (int j = i + 1; j < n; j++)
                if (i*arr[i] > j*arr[j] )
                    result++;
        }
        return result;
    }
}
```

```
/* Driver program to test above function */
public static void main(String[] args)
{
    int arr[] = {5, 0, 10, 2, 4, 1, 6} ;
    int n = arr.length;
    System.out.println("Count of Pairs : " +
                        CountPair(arr, n));
}
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
# C# Code to Count pairs in an
# array that hold  $i * arr[i] > j * arr[j]$ 

# Return count of pair in given array
# such that  $i * arr[i] > j * arr[j]$ 
def CountPair(arr, n):

    # Initialize result
    result = 0;

    for i in range(0, n):

        # Generate all pair and increment
        # counter if the hold given condition
        j = i + 1
        while(j < n):
            if (i * arr[i] > j * arr[j]):
                result = result +1
            j = j + 1
    return result;

# Driver program to test above function */

arr = [5, 0, 10, 2, 4, 1, 6]
n = len(arr)
print("Count of Pairs : " , CountPair(arr, n))

# This code is contributed by Sam007.
```

C#

```
// C# Code to Count pairs in an
// array that hold  $i * arr[i] > j * arr[j]$ 
using System;
```

```
class GFG
{
    // Return count of pair in given array
    // such that i*arr[i] > j*arr[j]
    public static int CountPair(int []arr , int n )
    {
        // Initialize result
        int result = 0;

        for (int i = 0; i < n; i++)
        {
            // Generate all pair and increment
            // counter if the hold given condition
            for (int j = i + 1; j < n; j++)
                if (i*arr[i] > j*arr[j] )
                    result++;
        }
        return result;
    }

    /* Driver program to test above function */
    public static void Main()
    {
        int []arr = {5, 0, 10, 2, 4, 1, 6};
        int n = arr.Length;
        Console.WriteLine("Count of Pairs : " +
                           CountPair(arr, n));
    }
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP program to count all pair that
// hold condition i*arr[i] > j*arr[j]

// Return count of pair in given array
// such that i*arr[i] > j*arr[j]
function CountPair($arr , $n )
{

    // Initialize result
    $result = 0;
```

```
for($i = 0; $i < $n; $i++)
{
    // Generate all pair and increment
    // counter if the hold given condition
    for ($j = $i + 1; $j < $n; $j++)
        if ($i * $arr[$i] > $j * $arr[$j] )
            $result++;
}
return $result;
}

// Driver code
$arr = array(5, 0, 10, 2, 4, 1, 6) ;
$n = sizeof($arr);
echo "Count of Pairs : ",
CountPair($arr, $n);

// This code is contributed by m_kit
?>
```

Output:

Count of Pairs : 5

Time Complexity: $O(n^2)$

An **efficient solution** of this problem takes $O(n \log n)$ time. The idea is based on an interesting fact about this problem that after modifying the array such that every element is multiplied with its index, this problem convert into [Count Inversions in an array](#).
Algorithm :

- Given an array 'arr' and it's size 'n'
- 1) First traversal array element, i goes from 0 to n-1
 - a) Multiple each element with its index $arr[i] = arr[i] * i$
 - 2) After that step 1. whole process is similar to Count Inversions in an array.

Below the implementation of above idea

C++

```
// C++ program to count all pair that
// hold condition  $i * arr[i] > j * arr[j]$ 
#include <bits/stdc++.h>
using namespace std;
```

```
/* This function merges two sorted arrays and
   returns inversion count in the arrays.*/
int merge(int arr[], int temp[], int left,
          int mid, int right)
{
    int inv_count = 0;

    int i = left; /* index for left subarray*/
    int j = mid; /* index for right subarray*/
    int k = left; /* index for resultant subarray*/
    while ((i <= mid - 1) && (j <= right))
    {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
        {
            temp[k++] = arr[j++];
            inv_count = inv_count + (mid - i);
        }
    }

    /* Copy the remaining elements of left
       subarray (if there are any) to temp*/
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    /* Copy the remaining elements of right
       subarray (if there are any) to temp*/
    while (j <= right)
        temp[k++] = arr[j++];

    /* Copy back the merged elements to original
       array*/
    for (i=left; i <= right; i++)
        arr[i] = temp[i];

    return inv_count;
}

/* An auxiliary recursive function that sorts
   the input array and returns the number of
   inversions in the array. */
int _mergeSort(int arr[], int temp[], int left,
               int right)
{
    int mid, inv_count = 0;
    if (right > left)
```

```
{  
    /* Divide the array into two parts and call  
       _mergeSortAndCountInv() for each of  
       the parts */  
    mid = (right + left)/2;  
  
    /* Inversion count will be sum of inversions in  
       left-part, right-part and number of inversions  
       in merging */  
    inv_count = _mergeSort(arr, temp, left, mid);  
    inv_count += _mergeSort(arr, temp, mid+1, right);  
  
    /*Merge the two parts*/  
    inv_count += merge(arr, temp, left, mid+1, right);  
}  
  
return inv_count;  
}  
  
/* This function sorts the input array and  
   returns the number of inversions in the  
   array */  
int countPairs(int arr[], int n)  
{  
    // Modify the array so that problem reduces to  
    // count inversion problem.  
    for (int i=0; i<n; i++)  
        arr[i] = i*arr[i];  
  
    // Count inversions using same logic as  
    // below post  
    // https://www.geeksforgeeks.org/counting-inversions/  
    int temp[n];  
    return _mergeSort(arr, temp, 0, n - 1);  
}  
  
// Driver code  
int main()  
{  
    int arr[] = {5, 0, 10, 2, 4, 1, 6};  
    int n = sizeof(arr)/sizeof(arr[0]);  
    cout << "Count of Pairs : "  
        << countPairs(arr, n);  
    return 0;  
}
```

Java

```
// Java program to count all pair that
// hold condition i*arr[i] > j*arr[j]
import java.io.*;

class GFG
{
    // This function merges two sorted arrays and
    // returns inversion count in the arrays.
    static int merge(int arr[], int temp[], int left,
                     int mid, int right)
    {
        int inv_count = 0;

        /* index for left subarray*/
        int i = left;

        /* index for right subarray*/
        int j = mid;
        /* index for resultant subarray*/
        int k = left;

        while ((i <= mid - 1) && (j <= right))
        {
            if (arr[i] <= arr[j])
                temp[k++] = arr[i++];
            else
            {
                temp[k++] = arr[j++];

                inv_count = inv_count + (mid - i);
            }
        }

        /* Copy the remaining elements of left
        subarray (if there are any) to temp*/
        while (i <= mid - 1)
            temp[k++] = arr[i++];

        /* Copy the remaining elements of right
        subarray (if there are any) to temp*/
        while (j <= right)
            temp[k++] = arr[j++];

        // Copy back the merged elements
        // to original array
        for (i = left; i <= right; i++)
            arr[i] = temp[i];
    }
}
```

```
        return inv_count;
    }

/* An auxiliary recursive function
that sorts the input array and
returns the number of inversions
in the array. */
static int _mergeSort(int arr[], int temp[],
                      int left,int right)
{
    int mid, inv_count = 0;
    if (right > left)
    {
        /* Divide the array into two parts and call
        _mergeSortAndCountInv() for each of
        the parts */
        mid = (right + left) / 2;

        // Inversion count will be sum of inversions in
        // left-part, right-part and number of inversions
        // in merging
        inv_count = _mergeSort(arr, temp, left, mid);
        inv_count += _mergeSort(arr, temp, mid+1, right);

        /*Merge the two parts*/
        inv_count += merge(arr, temp, left, mid+1, right);
    }

    return inv_count;
}

// This function sorts the input array and
// returns the number of inversions in the
// array
static int countPairs(int arr[], int n)
{
    // Modify the array so that problem reduces to
    // count inversion problem.
    for (int i = 0; i < n; i++)
        arr[i] = i * arr[i];

    // Count inversions using same logic as
    // below post
    // https://www.geeksforgeeks.org/counting-inversions/
    int temp[] = new int [n];
    return _mergeSort(arr, temp, 0, n - 1);
}
```

```
// Driver code

public static void main (String[] args)
{
    int arr[] = {5, 0, 10, 2, 4, 1, 6};
    int n = arr.length;
    System.out.print( "Count of Pairs : "
                      + countPairs(arr, n));

}

// This code is contributed by vt_m
```

C#

```
// C# program to count all pair that
// hold condition  $i * arr[i] > j * arr[j]$ 
using System;

class GFG
{
    // This function merges two sorted arrays and
    // returns inversion count in the arrays.
    static int merge(int []arr, int []temp, int left,
                     int mid, int right)
    {
        int inv_count = 0;

        /* index for left subarray*/
        int i = left;

        /* index for right subarray*/
        int j = mid;
        /* index for resultant subarray*/
        int k = left;

        while ((i <= mid - 1) && (j <= right))
        {
            if (arr[i] <= arr[j])
                temp[k++] = arr[i++];
            else
            {
                temp[k++] = arr[j++];

                inv_count = inv_count + (mid - i);
            }
        }
    }
```

```
/* Copy the remaining elements of left
subarray (if there are any) to temp*/
while (i <= mid - 1)
    temp[k++] = arr[i++];

/* Copy the remaining elements of right
subarray (if there are any) to temp*/
while (j <= right)
    temp[k++] = arr[j++];

// Copy back the merged elements
// to original array
for (i = left; i <= right; i++)
    arr[i] = temp[i];

return inv_count;
}

/* An auxiliary recursive function
that sorts the input array and
returns the number of inversions
in the array. */
static int _mergeSort(int []arr, int []temp,
                      int left,int right)
{
    int mid, inv_count = 0;
    if (right > left)
    {
        /* Divide the array into two parts and call
        _mergeSortAndCountInv() for each of
        the parts */
        mid = (right + left) / 2;

        // Inversion count will be sum of inversions in
        // left-part, right-part and number of inversions
        // in merging
        inv_count = _mergeSort(arr, temp, left, mid);
        inv_count += _mergeSort(arr, temp, mid+1, right);

        /*Merge the two parts*/
        inv_count += merge(arr, temp, left, mid+1, right);
    }

    return inv_count;
}

// This function sorts the input array and
```

```
// returns the number of inversions in the
// array
static int countPairs(int []arr, int n)
{
    // Modify the array so that problem reduces to
    // count inversion problem.
    for (int i = 0; i < n; i++)
        arr[i] = i * arr[i];

    // Count inversions using same logic as
    // below post
    // https://www.geeksforgeeks.org/counting-inversions/
    int []temp = new int [n];
    return _mergeSort(arr, temp, 0, n - 1);
}

// Driver code

public static void Main ()
{
    int []arr = {5, 0, 10, 2, 4, 1, 6};
    int n = arr.Length;
    Console.WriteLine( "Count of Pairs : "
                      + countPairs(arr, n));

}

// This code is contributed by anuj_67.
```

Output:

Count of Pairs : 5

Time Complexity: $O(n \log n)$

Improved By : [Sam007](#), [jit_t](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/count-pairs-array-hold-iarri-jarri/>

Chapter 74

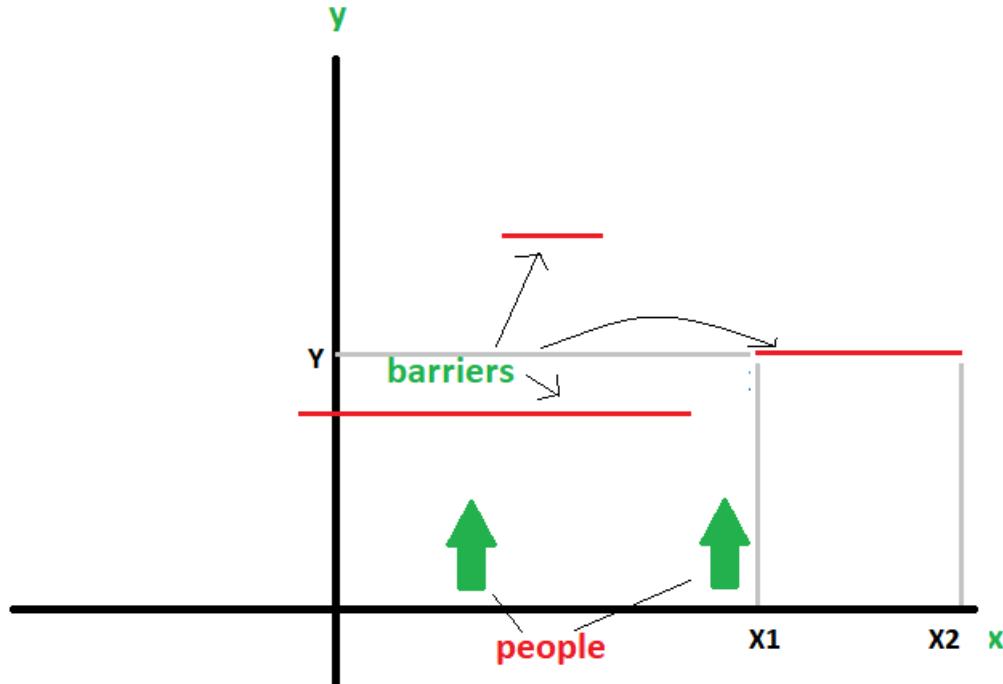
Count points covered by given intervals

Count points covered by given intervals - GeeksforGeeks

Consider an infinite x-y plane. Infinite people walk on the plane in upward or +ve Y direction. At each integer point on the x-axis, only one person walks. Suppose, several barriers exist parallel to the x-axis. Barriers are provided as three triplets

- X1 – the x-point at which the barrier starts
- X2 – the x-point at which the barrier ends.
- Y – the point on the Y axis where the barrier lies.

Calculate how many people will get stuck at any point due to the barriers. Use of extra space to keep track of different points on x axis is not allowed.



Examples:

```

Input : barriers[] = {{3 6 2}, {-7 4 3}}
Output : 14
The barrier from 3 to 6 blocks 3, 4, 5, 6,
so 4 persons blocked till now.
The barrier from -7 to 4 blocks -7,-6,-5,-4,-
3, -2, -1, 0, 1, 2, 3, 4. But, 3 and 4 have
already been blocked.
So, total persons blocked is 14.

```

Asked in: Microsoft IDC Internship.

A simple approach is to use a very long array initialized to zero. Then we can mark those values by 1 which are in the barrier by looping through each barrier. This would solve the case of overlapping of barriers.

But we cannot use another array as mentioned before. Thus, we use sorting and simple math. Following are the steps:

1. Sort all barriers by x1 (Starting point)
2. After sorting, 3 cases arrive:
.....I. The next barrier does not overlap with the previous. In this case, we simply add

count of points covered by current barrier.

.....II. The next barrier partly overlaps the previous one. In this point we add non-overlapping points covered by current barrier.

.....III. The next barrier completely overlaps the previous barrier. In this case, we simply ignore current barrier.

Below is the implementation of above approach.

```
// CPP program to find number of people
// that are stopped by barriers
#include <bits/stdc++.h>
using namespace std;

struct Barrier
{
    int x1, x2, y;
};

// Compares two Barriers according to x1
bool compareBarrier(Barrier b1, Barrier b2)
{
    return (b1.x1 < b2.x1);
}

// Returns number of people blocked by
// array of barriers arr[0..n-1]
int countStopped(Barrier arr[], int n)
{
    // Sort barriers according to x1.
    sort(arr, arr+n, compareBarrier);

    // End point of previous barrier
    // Initializing with some value
    // smaller than lowest x1.
    int prev_end = arr[0].x1 - 1;

    // Traverse through all barriers
    int count = 0;
    for (int i=0; i<n; i++)
    {
        // If current barrier doesn't overlap
        // with previous
        if (prev_end < arr[i].x1)
        {
            count += (arr[i].x2 - arr[i].x1 + 1);
            prev_end = arr[i].x2;
        }
    }

    // If current barrier overlaps and
```

```
// blocks some more people
else if (prev_end < arr[i].x2)
{
    count += (arr[i].x2 - prev_end);
    prev_end = arr[i].x2;
}
}

return count;
}

// Driver code
int main()
{
    Barrier arr[] = {{3, 6, 2}, {-7, 4, 3}};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << countStopped(arr, n);
    return 0;
}
```

Output :

14

We can easily note that there is no importance of y in the question, so it may not be stored.

Time Complexity : $O(n \log n)$

Source

<https://www.geeksforgeeks.org/count-points-covered-by-given-intervals/>

Chapter 75

Count quadruples from four sorted arrays whose sum is equal to a given value x

Count quadruples from four sorted arrays whose sum is equal to a given value x - Geeks-forGeeks

Given four sorted arrays each of size n of distinct elements. Given a value x . The problem is to count all **quadruples**(group of four numbers) from all the four arrays whose sum is equal to x .

Note: The quadruple has an element from each of the four arrays.

Examples:

```
Input : arr1 = {1, 4, 5, 6},  
        arr2 = {2, 3, 7, 8},  
        arr3 = {1, 4, 6, 10},  
        arr4 = {2, 4, 7, 8}  
n = 4, x = 30
```

```
Output : 4  
The quadruples are:  
(4, 8, 10, 8), (5, 7, 10, 8),  
(5, 8, 10, 7), (6, 7, 10, 7)
```

```
Input : For the same above given fours arrays  
       x = 25  
Output : 14
```

Method 1 (Naive Approach): Using four nested loops generate all quadruples and check whether elements in the quadruple sum up to x or not.

C++

```
// C++ implementation to count quadruples from four sorted arrays
// whose sum is equal to a given value x
#include <bits/stdc++.h>

using namespace std;

// function to count all quadruples from
// four sorted arrays whose sum is equal
// to a given value x
int countQuadruples(int arr1[], int arr2[],
                     int arr3[], int arr4[], int n, int x)
{
    int count = 0;

    // generate all possible quadruples from
    // the four sorted arrays
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                for (int l = 0; l < n; l++)
                    // check whether elements of
                    // quadruple sum up to x or not
                    if ((arr1[i] + arr2[j] + arr3[k] + arr4[l]) == x)
                        count++;

    // required count of quadruples
    return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };

    int n = sizeof(arr1) / sizeof(arr1[0]);
    int x = 30;
    cout << "Count = "
        << countQuadruples(arr1, arr2, arr3,
                           arr4, n, x);
    return 0;
}
```

Python3

```
# A Python implementation to count
# quadruples from four sorted arrays
# whose sum is equal to a given value x

# function to count all quadruples
# from four sorted arrays whose sum
# is equal to a given value x
def countQuuadruples(arr1, arr2,
                     arr3, arr4, n, x):
    count = 0

    # generate all possible
    # quadruples from the four
    # sorted arrays
    for i in range(n):
        for j in range(n):
            for k in range(n):
                for l in range(n):

                    # check whether elements of
                    # quadruple sum up to x or not
                    if (arr1[i] + arr2[j] +
                        arr3[k] + arr4[l] == x):
                        count += 1

    # required count of quadruples
    return count

# Driver Code
arr1 = [1, 4, 5, 6]
arr2 = [2, 3, 7, 8]
arr3 = [1, 4, 6, 10]
arr4 = [2, 4, 7, 8 ]
n = len(arr1)
x = 30
print("Count = ", countQuuadruples(arr1, arr2,
                                    arr3, arr4, n, x))

# This code is contributed
# by Shrikant13
```

Output:

Count = 4

Time Complexity: $O(n^4)$

Auxiliary Space: $O(1)$

Method 2 (Binary Search): Generate all triplets from the 1st three arrays. For each triplet so generated, find the sum of elements in the triplet. Let it be T . Now, search the value $(x - T)$ in the 4th array. If value found in the 4th array, then increment **count**. This process is repeated for all the triplets generated from the 1st three arrays.

```

// C++ implementation to count quadruples from
// four sorted arrays whose sum is equal to a
// given value x
#include <bits/stdc++.h>

using namespace std;

// find the 'value' in the given array 'arr[]'
// binary search technique is applied
bool isPresent(int arr[], int low, int high, int value)
{
    while (low <= high) {
        int mid = (low + high) / 2;

        // 'value' found
        if (arr[mid] == value)
            return true;
        else if (arr[mid] > value)
            high = mid - 1;
        else
            low = mid + 1;
    }

    // 'value' not found
    return false;
}

// function to count all quadruples from four
// sorted arrays whose sum is equal to a given value x
int countQuadruples(int arr1[], int arr2[], int arr3[],
                     int arr4[], int n, int x)
{
    int count = 0;

    // generate all triplets from the 1st three arrays
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++) {

                // calculate the sum of elements in
                // the triplet so generated

```

```

int T = arr1[i] + arr2[j] + arr3[k];

// check if 'x-T' is present in 4th
// array or not
if (isPresent(arr4, 0, n, x - T))

    // increment count
    count++;
}

// required count of quadruples
return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };

    int n = sizeof(arr1) / sizeof(arr1[0]);
    int x = 30;
    cout << "Count = "
        << countQuadruples(arr1, arr2, arr3, arr4, n, x);
    return 0;
}

```

Output:

Count = 4

Time Complexity: $O(n^3 \log n)$

Auxiliary Space: $O(1)$

Method 3 (Use of two pointers): Generate all pairs from the 1st two arrays. For each pair so generated, find the sum of elements in the pair. Let it be **p_sum**. For each **p_sum**, **count pairs from the 3rd and 4th sorted array with sum equal to $(x - p_sum)$** . Accumulate these count in the **total_count** of quadruples.

```

// C++ implementation to count quadruples from
// four sorted arrays whose sum is equal to a
// given value x
#include <bits/stdc++.h>

```

```
using namespace std;

// count pairs from the two sorted array whose sum
// is equal to the given 'value'
int countPairs(int arr1[], int arr2[], int n, int value)
{
    int count = 0;
    int l = 0, r = n - 1;

    // traverse 'arr1[]' from left to right
    // traverse 'arr2[]' from right to left
    while (l < n & r >= 0) {
        int sum = arr1[l] + arr2[r];

        // if the 'sum' is equal to 'value', then
        // increment 'l', decrement 'r' and
        // increment 'count'
        if (sum == value) {
            l++;
            r--;
            count++;
        }

        // if the 'sum' is greater than 'value', then
        // decrement r
        else if (sum > value)
            r--;

        // else increment l
        else
            l++;
    }

    // required count of pairs
    return count;
}

// function to count all quadruples from four sorted arrays
// whose sum is equal to a given value x
int countQuadruples(int arr1[], int arr2[], int arr3[],
                     int arr4[], int n, int x)
{
    int count = 0;

    // generate all pairs from arr1[] and arr2[]
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            // calculate the sum of elements in
            // the pair so generated
```

```

        int p_sum = arr1[i] + arr2[j];

        // count pairs in the 3rd and 4th array
        // having value 'x-p_sum' and then
        // accumulate it to 'count'
        count += countPairs(arr3, arr4, n, x - p_sum);
    }

    // required count of quadruples
    return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };

    int n = sizeof(arr1) / sizeof(arr1[0]);
    int x = 30;
    cout << "Count = "
        << countQuadruples(arr1, arr2, arr3,
                           arr4, n, x);
    return 0;
}

```

Output:

Count = 4

Time Complexity: $O(n^3)$

Auxiliary Space: $O(1)$

Method 4 Efficient Approach(Hashing): Create a hash table where (**key, value**) tuples are represented as (**sum, frequency**) tuples. Here the sum are obtained from the pairs of 1st and 2nd array and their frequency count is maintained in the hash table. Hash table is implemented using [unordered_map in C++](#). Now, generate all pairs from the 3rd and 4th array. For each pair so generated, find the sum of elements in the pair. Let it be **p_sum**. For each **p_sum**, check whether (**x - p_sum**) exists in the hash table or not. If it exists, then add the frequency of (**x - p_sum**) to the **count** of quadruples.

```

// C++ implementation to count quadruples from
// four sorted arrays whose sum is equal to a
// given value x

```

```
#include <bits/stdc++.h>

using namespace std;

// function to count all quadruples from four sorted
// arrays whose sum is equal to a given value x
int countQuadruples(int arr1[], int arr2[], int arr3[],
                     int arr4[], int n, int x)
{
    int count = 0;

    // unordered_map 'um' implemented as hash table
    // for <sum, frequency> tuples
    unordered_map<int, int> um;

    // count frequency of each sum obtained from the
    // pairs of arr1[] and arr2[] and store them in 'um'
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            um[arr1[i] + arr2[j]]++;

    // generate pair from arr3[] and arr4[]
    for (int k = 0; k < n; k++)
        for (int l = 0; l < n; l++) {

            // calculate the sum of elements in
            // the pair so generated
            int p_sum = arr3[k] + arr4[l];

            // if 'x-p_sum' is present in 'um' then
            // add frequency of 'x-p_sum' to 'count'
            if (um.find(x - p_sum) != um.end())
                count += um[x - p_sum];
        }

    // required count of quadruples
    return count;
}

// Driver program to test above
int main()
{
    // four sorted arrays each of size 'n'
    int arr1[] = { 1, 4, 5, 6 };
    int arr2[] = { 2, 3, 7, 8 };
    int arr3[] = { 1, 4, 6, 10 };
    int arr4[] = { 2, 4, 7, 8 };
```

```
int n = sizeof(arr1) / sizeof(arr1[0]);
int x = 30;
cout << "Count = "
    << countQuadruples(arr1, arr2, arr3, arr4, n, x);
return 0;
}
```

Output:

Count = 4

Time Complexity: $O(n^2)$
Auxiliary Space: $O(n^2)$

Improved By : [shrikanth13](#)

Source

<https://www.geeksforgeeks.org/count-quadruples-four-sorted-arrays-whose-sum-equal-given-value-x/>

Chapter 76

Counting Sort

Counting Sort - GeeksforGeeks

[Counting sort](#) is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values (kind of hashing). Then doing some arithmetic to calculate the position of each object in the output sequence.

Let us understand it with the help of an example.

For simplicity, consider the data in the range 0 to 9.

Input data: 1, 4, 1, 2, 7, 5, 2

1) Take a count array to store the count of each unique object.

Index: 0 1 2 3 4 5 6 7 8 9

Count: 0 2 2 0 1 1 0 1 0 0

2) Modify the count array such that each element at each index stores the sum of previous counts.

Index: 0 1 2 3 4 5 6 7 8 9

Count: 0 2 4 4 5 6 6 7 7 7

The modified count array indicates the position of each object in the output sequence.

3) Output each object from the input sequence followed by decreasing its count by 1.

Process the input data: 1, 4, 1, 2, 7, 5, 2. Position of 1 is 2.

Put data 1 at index 2 in output. Decrease count by 1 to place next data 1 at an index 1 smaller than this index.

Following is implementation of counting sort.

C

```
// C Program for counting sort
#include <stdio.h>
#include <string.h>
#define RANGE 255

// The main function that sort the given string arr[] in
// alphabetical order
void countSort(char arr[])
{
    // The output character array that will have sorted arr
    char output[strlen(arr)];

    // Create a count array to store count of individual
    // characters and initialize count array as 0
    int count[RANGE + 1], i;
    memset(count, 0, sizeof(count));

    // Store count of each character
    for(i = 0; arr[i]; ++i)
        ++count[arr[i]];

    // Change count[i] so that count[i] now contains actual
    // position of this character in output array
    for (i = 1; i <= RANGE; ++i)
        count[i] += count[i-1];

    // Build the output character array
    for (i = 0; arr[i]; ++i)
    {
        output[count[arr[i]]-1] = arr[i];
        --count[arr[i]];
    }

    // Copy the output array to arr, so that arr now
    // contains sorted characters
    for (i = 0; arr[i]; ++i)
        arr[i] = output[i];
}

// Driver program to test above function
int main()
{
    char arr[] = "geeksforgeeks";//"applepp";

    countSort(arr);

    printf("Sorted character array is %sn", arr);
    return 0;
}
```

}

Java

```
// Java implementation of Counting Sort
class CountingSort
{
    void sort(char arr[])
    {
        int n = arr.length;

        // The output character array that will have sorted arr
        char output[] = new char[n];

        // Create a count array to store count of individual
        // characters and initialize count array as 0
        int count[] = new int[256];
        for (int i=0; i<256; ++i)
            count[i] = 0;

        // store count of each character
        for (int i=0; i<n; ++i)
            ++count[arr[i]];

        // Change count[i] so that count[i] now contains actual
        // position of this character in output array
        for (int i=1; i<=255; ++i)
            count[i] += count[i-1];

        // Build the output character array
        // To make it stable we are operating in reverse order.
        for (int i = n-1; i>=0; i--)
        {
            output[count[arr[i]]-1] = arr[i];
            --count[arr[i]];
        }

        // Copy the output array to arr, so that arr now
        // contains sorted characters
        for (int i = 0; i<n; ++i)
            arr[i] = output[i];
    }

    // Driver method
    public static void main(String args[])
    {
        CountingSort ob = new CountingSort();
        char arr[] = {'g', 'e', 'e', 'k', 's', 'f', 'o',
                     'g', 'e', 'e', 'k', 's', 'f', 'o'};
    }
}
```

```
'r', 'g', 'e', 'e', 'k', 's'  
};  
  
ob.sort(arr);  
  
System.out.print("Sorted character array is ");  
for (int i=0; i<arr.length; ++i)  
    System.out.print(arr[i]);  
}  
}  
/*This code is contributed by Rajat Mishra */
```

Python

```
# Python program for counting sort  
  
# The main function that sort the given string arr[] in  
# alphabetical order  
def countSort(arr):  
  
    # The output character array that will have sorted arr  
    output = [0 for i in range(256)]  
  
    # Create a count array to store count of individual  
    # characters and initialize count array as 0  
    count = [0 for i in range(256)]  
  
    # For storing the resulting answer since the  
    # string is immutable  
    ans = [" " for _ in arr]  
  
    # Store count of each character  
    for i in arr:  
        count[ord(i)] += 1  
  
    # Change count[i] so that count[i] now contains actual  
    # position of this character in output array  
    for i in range(256):  
        count[i] += count[i-1]  
  
    # Build the output character array  
    for i in range(len(arr)):  
        output[count[ord(arr[i])]-1] = arr[i]  
        count[ord(arr[i])] -= 1  
  
    # Copy the output array to arr, so that arr now  
    # contains sorted characters  
    for i in range(len(arr)):
```

```
    ans[i] = output[i]
    return ans

# Driver program to test above function
arr = "geeksforgeeks"
ans = countSort(arr)
print "Sorted character array is %s" %("".join(ans))

# This code is contributed by Nikhil Kumar Singh
```

C#

```
// C# implementation of Counting Sort
using System;

class GFG {

    static void countsort(char []arr)
    {
        int n = arr.Length;

        // The output character array that
        // will have sorted arr
        char []output = new char[n];

        // Create a count array to store
        // count of individual characters
        // and initialize count array as 0
        int []count = new int[256];

        for (int i=0; i<256; ++i)
            count[i] = 0;

        // store count of each character
        for (int i=0; i<n; ++i)
            ++count[arr[i]];

        // Change count[i] so that count[i]
        // now contains actual position of
        // this character in output array
        for (int i=1; i<=255; ++i)
            count[i] += count[i-1];

        // Build the output character array
        // To make it stable we are operating in reverse order.
        for (int i = n-1; i>=0; i--)
        {
            output[count[arr[i]]-1] = arr[i];
```

```
--count[arr[i]];
}

// Copy the output array to arr, so
// that arr now contains sorted
// characters
for (int i = 0; i<n; ++i)
    arr[i] = output[i];
}

// Driver method
public static void Main()
{
    char []arr = {'g', 'e', 'e', 'k', 's', 'f', 'o',
                  'r', 'g', 'e', 'e', 'k', 's'
                };

    countsort(arr);

    Console.Write("Sorted character array is ");
    for (int i=0; i<arr.Length; ++i)
        Console.Write(arr[i]);
}
}

// This code is contributed by Sam007.
```

PHP

```
<?php
// PHP Program for counting sort

$RANGE = 255;

// The main function that sort
// the given string arr[] in
// alphabetical order
function countSort($arr)
{
    global $RANGE;

    // The output character array
    // that will have sorted arr
    $output = array(strlen($arr));
    $len = strlen($arr);

    // Create a count array to
```

```
// store count of individual
// characters and initialize
// count array as 0
$count = array_fill(0, $RANGE + 1, 0);

// Store count of
// each character
for($i = 0; $i < $len; ++$i)
    ++$count[ord($arr[$i])];

// Change count[i] so that
// count[i] now contains
// actual position of this
// character in output array
for ($i = 1; $i <= $RANGE; ++$i)
    $count[$i] += $count[$i - 1];

// Build the output
// character array
// To make it stable we are operating
// in reverse order.
for ($i = $len-1; $i >= 0 ; $i--)
{
    $output[$count[ord($arr[$i])] - 1] = $arr[$i];
    --$count[ord($arr[$i])];
}

// Copy the output array to
// arr, so that arr now
// contains sorted characters
for ($i = 0; $i < $len; ++$i)
    $arr[$i] = $output[$i];
return $arr;
}

// Driver Code
$arr = "geeksforgeeks"; //applepp";

$arr = countSort($arr);

echo "Sorted character array is " . $arr;

// This code is contributed by mits
?>
```

Output:

Sorted character array is eeeefggkkorss

Time Complexity: $O(n+k)$ where n is the number of elements in input array and k is the range of input.

Auxiliary Space: $O(n+k)$

Points to be noted:

1. Counting sort is efficient if the range of input data is not significantly greater than the number of objects to be sorted. Consider the situation where the input sequence is between range 1 to 10K and the data is 10, 5, 10K, 5K.
2. It is not a comparison based sorting. Its running time complexity is $O(n)$ with space proportional to the range of data.
3. It is often used as a sub-routine to another sorting algorithm like radix sort.
4. Counting sort uses a partial hashing to count the occurrence of the data object in $O(1)$.
5. Counting sort can be extended to work for negative inputs also.

Exercise:

1. Modify above code to sort the input data in the range from M to N.
2. Modify above code to sort negative input data.
3. Is counting sort stable and online?
4. Thoughts on parallelizing the counting sort algorithm.

Snapshots:

For simplicity, consider data



| Index : | 0 | 1 | 2 | 3 | 4 |
|---------|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 |

Count each element in the given array and place the count at the appropriate index.

For simplicity, consider data

1 4 1



Index : 0 1 2 3 4

0 2 0 0 1

For simplicity, consider data

| Index : | 0 | 1 | 2 | 3 | 4 |
|---------|---|---|---|---|---|
| | 0 | 2 | 2 | 0 | 1 |

Modify the count array by ac-

For simplicity, consider data

| | | | | |
|---------|---|---|---|---|
| | 1 | 4 | 1 | |
| Index : | 0 | 1 | 2 | 3 |
| | 0 | 2 | 4 | 0 |
| | | | | 1 |

$2 + 2$

For simplicity, consider data

| | | | | |
|---------|---|---|---|---|
| | 1 | 4 | 1 | |
| Index : | 0 | 1 | 2 | 3 |
| | 0 | 2 | 4 | 4 |
| | | | | 1 |

4 + 0

For simplicity, consider data

1

4

1

Index : 0

1

2

3

4

0

0

4

4

4

Places : 1

2

3

4

1

1

For simplicity, consider data



[Selection Sort](#), [Bubble Sort](#), [Insertion Sort](#), [Merge Sort](#), [Heap Sort](#), [QuickSort](#), [Radix Sort](#), [Counting Sort](#), [Bucket Sort](#), [ShellSort](#), [Comb Sort](#), [PegionHole Sorting](#)

This article is compiled by [Aashish Barnwal](#). Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : [Mithun Kumar](#), [spattk](#)

Source

<https://www.geeksforgeeks.org/counting-sort/>

Chapter 77

Counting cross lines in an array

Counting cross lines in an array - GeeksforGeeks

Given an unsorted array of distinct elements. Task is to count number of cross lines formed in an array elements after sorting the array elements.

Note: Draw a line between same array elements before sorting and after sorting the array elements.

Examples :

Input : arr[] = { 3, 2, 1, 4, 5 }

Output : 3

```
before sort: 3 2 1 4 5
          \ | /   | |
          \|/   | |
          / | \   | |
```

After sort : 1 2 3 4 5

line (1 to 1) cross line (2 to 2)

line (1 to 1) cross line (3 to 3)

line (2 to 2) cross line (3 to 3)

Note: the line between two 4s and the line
between two 5s don't cross any other lines;

Input : arr[] = { 5, 4, 3, 1 }

Output : 6

Simple solution of this problem is based on the [insertion sort](#). we simply pick each array elements one-by-one and try to find it's proper position in the sorted array.during finding it's appropriate position of an element we have to cross all the element_line whose value is greater than current element.

Below is the implementation of above idea :

C++

```
// c++ program to count cross line in array
#include <bits/stdc++.h>
using namespace std;

// function return count of cross line in an array
int countCrossLine(int arr[], int n)
{
    int count_crossline = 0;
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        /* Move elements of arr[0..i-1], that are
           greater than key, to one position ahead
           of their current position */
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;

            // increment cross line by one
            count_crossline++;
        }
        arr[j + 1] = key;
    }
    return count_crossline;
}

// driver program to test above function
int main()
{
    int arr[] = { 4, 3, 1, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << countCrossLine(arr, n) << endl;
    return 0;
}
```

Java

```
// Java program to count
// cross line in array

class GFG
{
    static int countCrossLine(int arr[],
                           int n)
    {
        int count_crossline = 0;
```

```
int i, key, j;
for (i = 1; i < n; i++)
{
    key = arr[i];
    j = i - 1;

    // Move elements of arr[0..i-1],
    // that are greater than key,
    // to one position ahead of
    // their current position
    while (j >= 0 && arr[j] > key)
    {
        arr[j + 1] = arr[j];
        j = j - 1;

        // increment cross
        // line by one
        count_crossline++;
    }
    arr[j + 1] = key;
}

return count_crossline;
}

// Driver Code
public static void main(String args[])
{
    int arr[] = new int[]{ 4, 3, 1, 2 };
    int n = arr.length;
    System.out.print(countCrossLine(arr, n));
}
}

// This code is contributed by Sam007
```

C#

```
// C# program to count cross line in array
using System;

class GFG {

    // function return count of cross line
    // in an array
    static int countCrossLine(int []arr, int n)
    {
        int count_crossline = 0;
```

```
int i, key, j;
for (i = 1; i < n; i++) {
    key = arr[i];
    j = i - 1;

    /* Move elements of arr[0..i-1],
       that are greater than key, to one
       position ahead of their current
       position */
    while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j];
        j = j - 1;

        // increment cross line by one
        count_crossline++;
    }
    arr[j + 1] = key;
}

return count_crossline;
}

// Driver code
public static void Main()
{
    int []arr = new int[]{ 4, 3, 1, 2 };
    int n = arr.Length;
    Console.WriteLine(countCrossLine(arr, n));
}
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP program to count
// cross line in array

// Function return count
// of cross line in an array
function countCrossLine($arr, $n)
{

$count_crossline = 0;
$i; $key; $j;
for ($i = 1; $i < $n; $i++)
{
```

```

$key = $arr[$i];
$j = $i - 1;

/* Move elements of arr[0..i-1], that are
   greater than key, to one position ahead
   of their current position */
while ($j >= 0 and $arr[$j] > $key)
{
    $arr[$j + 1] = $arr[$j];
    $j = $j - 1;

    // increment cross line by one
    $count_crossline++;
}
$arr[$j + 1] = $key;
}
return $count_crossline;
}

// Driver Code
$arr = array( 4, 3, 1, 2 );
$n = count($arr);
echo countCrossLine($arr, $n);

// This code is contributed by anuj_67.
?>

```

5

Time complexity : $O(n^2)$
 Auxiliary space: $O(1)$

Efficient solution based on the merge sort and [inversions count](#).

```

lets we have arr[] { 2, 4, 1, 3 }
                  \   \ /   /
                  \   / \ /
                  / \   / \
After sort    arr[] { 1, 2, 3, 4 }
and here all inversion are (2, 1), (4, 1), (4, 3)
that mean line 1 : cross line 4, 2
                  line 2 : cross line 1
                  line 4 : cross line 3, 1
                  line 3 : cross line 3
so total unique cross_line are: 3

```

During mer

Below c++ implementation of above idea.

```
// c++ program to count cross line in array
#include <bits/stdc++.h>
using namespace std;

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r, int* count_crossline)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];

            //=====
            //===== MAIN PORTION OF CODE =====//
            //===== ===== //
            // add all line which is cross by current element
            *count_crossline += (n1 - i);
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
       are any */
}
```

```

are any */
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy the remaining elements of R[], if there
are any */
while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
}
}

/* l is for left index and r is right index of the
sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r, int* count_crossline)
{
    if (l < r) {

        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - 1) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m, count_crossline);
        mergeSort(arr, m + 1, r, count_crossline);

        merge(arr, l, m, r, count_crossline);
    }
}

// function return count of cross line in an array
int countCrossLine(int arr[], int n)
{
    int count_crossline = 0;
    mergeSort(arr, 0, n - 1, &count_crossline);

    return count_crossline;
}

// driver program to test above function
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
}

```

```
    cout << countCrossLine(arr, n) << endl;
    return 0;
}
```

Output:

10

Time complexity : $O(n \log n)$

Improved By : [vt_m](#), [Sam007](#)

Source

<https://www.geeksforgeeks.org/counting-cross-lines-array/>

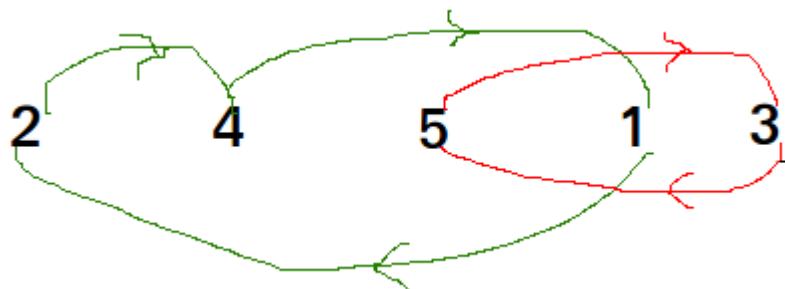
Chapter 78

Cycle Sort

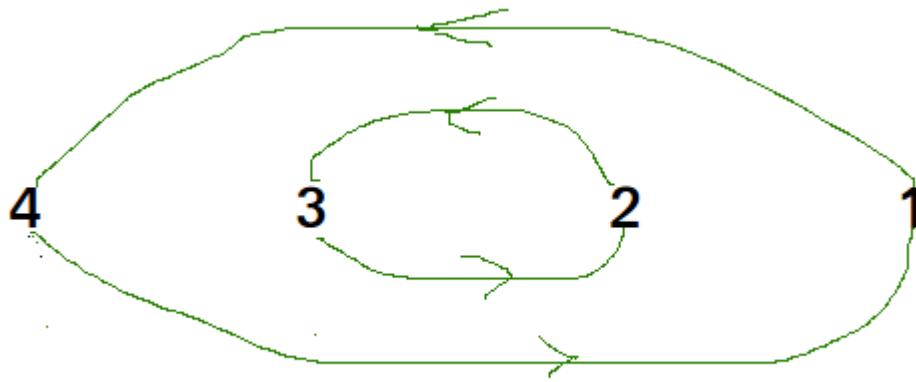
Cycle Sort - GeeksforGeeks

Cycle sort is an in-place sorting Algorithm, [unstable sorting algorithm](#), a comparison sort that is theoretically optimal in terms of the total number of writes to the original array.

- It is optimal in terms of number of memory writes. It [minimizes the number of memory writes](#) to sort (Each value is either written zero times, if it's already in its correct position, or written one time to its correct position.)
- It is based on the idea that array to be sorted can be divided into cycles. Cycles can be visualized as a graph. We have n nodes and an edge directed from node i to node j if the element at i-th index must be present at j-th index in the sorted array.
Cycle in arr[] = {4, 5, 2, 1, 5}



Cycle in arr[] = {4, 3, 2, 1}



We one by one consider all cycles. We first consider the cycle that includes first element. We find correct position of first element, place it at its correct position, say j. We consider old value of arr[j] and find its correct position, we keep doing this till all elements of current cycle are placed at correct position, i.e., we don't come back to cycle starting point.

Explanation :

```
arr[] = {10, 5, 2, 3}
index = 0 1 2 3
cycle_start = 0
item = 10 = arr[0]
```

Find position where we put the item
`pos = cycle_start`
`while (arr[i] < item)`
 `pos++;`

We put 10 at arr[3] and change item to
old value of arr[3].
`arr[] = {10, 5, 2, 10}`
`item = 3`

Again rotate rest cycle that start with index '0'
Find position where we put the item = 3
we swap item with element at arr[1] now
`arr[] = {10, 3, 2, 10}`
`item = 5`

Again rotate rest cycle that start with index '0' and item = 5
we swap item with element at arr[2].
`arr[] = {10, 3, 5, 10 }`

```
item = 2
```

```
Again rotate rest cycle that start with index '0' and item = 2
arr[] = {2, 3, 5, 10}
```

```
Above is one iteration for cycle_stat = 0.
Repeat above steps for cycle_start = 1, 2, ..n-2
```

CPP

```
// C++ program to implement cycle sort
#include <iostream>
using namespace std;

// Function sort the array using Cycle sort
void cycleSort(int arr[], int n)
{
    // count number of memory writes
    int writes = 0;

    // traverse array elements and put it to on
    // the right place
    for (int cycle_start = 0; cycle_start <= n - 2; cycle_start++) {
        // initialize item as starting point
        int item = arr[cycle_start];

        // Find position where we put the item. We basically
        // count all smaller elements on right side of item.
        int pos = cycle_start;
        for (int i = cycle_start + 1; i < n; i++)
            if (arr[i] < item)
                pos++;

        // If item is already in correct position
        if (pos == cycle_start)
            continue;

        // ignore all duplicate elements
        while (item == arr[pos])
            pos += 1;

        // put the item to it's right position
        if (pos != cycle_start) {
            swap(item, arr[pos]);
            writes++;
        }
    }

    // Rotate rest of the cycle
```

```

        while (pos != cycle_start) {
            pos = cycle_start;

            // Find position where we put the element
            for (int i = cycle_start + 1; i < n; i++)
                if (arr[i] < item)
                    pos += 1;

            // ignore all duplicate elements
            while (item == arr[pos])
                pos += 1;

            // put the item to it's right position
            if (item != arr[pos]) {
                swap(item, arr[pos]);
                writes++;
            }
        }
    }

    // Number of memory writes or swaps
    // cout << writes << endl ;
}

// Driver program to test above function
int main()
{
    int arr[] = { 1, 8, 3, 9, 10, 10, 2, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cycleSort(arr, n);

    cout << "After sort : " << endl;
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}

```

Java

```

// Java program to implement cycle sort

import java.util.*;
import java.lang.*;

class GFG {
    // Function sort the array using Cycle sort
    public static void cycleSort(int arr[], int n)
    {

```

```
// count number of memory writes
int writes = 0;

// traverse array elements and put it to on
// the right place
for (int cycle_start = 0; cycle_start <= n - 2; cycle_start++) {
    // initialize item as starting point
    int item = arr[cycle_start];

    // Find position where we put the item. We basically
    // count all smaller elements on right side of item.
    int pos = cycle_start;
    for (int i = cycle_start + 1; i < n; i++)
        if (arr[i] < item)
            pos++;

    // If item is already in correct position
    if (pos == cycle_start)
        continue;

    // ignore all duplicate elements
    while (item == arr[pos])
        pos += 1;

    // put the item to it's right position
    if (pos != cycle_start) {
        int temp = item;
        item = arr[pos];
        arr[pos] = temp;
        writes++;
    }

    // Rotate rest of the cycle
    while (pos != cycle_start) {
        pos = cycle_start;

        // Find position where we put the element
        for (int i = cycle_start + 1; i < n; i++)
            if (arr[i] < item)
                pos += 1;

        // ignore all duplicate elements
        while (item == arr[pos])
            pos += 1;

        // put the item to it's right position
        if (item != arr[pos]) {
            int temp = item;
```

```

        item = arr[pos];
        arr[pos] = temp;
        writes++;
    }
}
}

// Driver program to test above function
public static void main(String[] args)
{
    int arr[] = { 1, 8, 3, 9, 10, 10, 2, 4 };
    int n = arr.length;
    cycleSort(arr, n);

    System.out.println("After sort : ");
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}
}

// Code Contributed by Mohit Gupta_OMG <(0_o)>

```

Python3

```

# Python program to implement cycle sort

def cycleSort(array):
    writes = 0

    # Loop through the array to find cycles to rotate.
    for cycleStart in range(0, len(array) - 1):
        item = array[cycleStart]

        # Find where to put the item.
        pos = cycleStart
        for i in range(cycleStart + 1, len(array)):
            if array[i] < item:
                pos += 1

        # If the item is already there, this is not a cycle.
        if pos == cycleStart:
            continue

        # Otherwise, put the item there or right after any duplicates.
        while item == array[pos]:
            pos += 1
        array[pos], item = item, array[pos]

```

```

writes += 1

# Rotate the rest of the cycle.
while pos != cycleStart:

    # Find where to put the item.
    pos = cycleStart
    for i in range(cycleStart + 1, len(array)):
        if array[i] < item:
            pos += 1

    # Put the item there or right after any duplicates.
    while item == array[pos]:
        pos += 1
    array[pos], item = item, array[pos]
    writes += 1

return writes

# driver code
arr = [1, 8, 3, 9, 10, 10, 2, 4 ]
n = len(arr)
cycleSort(arr)

print("After sort : ")
for i in range(0, n) :
    print(arr[i], end = ' ')

```

Code Contributed by Mohit Gupta_OMG <(0_o)>

C#

```

// C# program to implement cycle sort
using System;

class GFG {

    // Function sort the array using Cycle sort
    public static void cycleSort(int[] arr, int n)
    {
        // count number of memory writes
        int writes = 0;

        // traverse array elements and
        // put it to on the right place
        for (int cycle_start = 0; cycle_start <= n - 2; cycle_start++)
        {
            // initialize item as starting point

```

```
int item = arr[cycle_start];

// Find position where we put the item.
// We basically count all smaller elements
// on right side of item.
int pos = cycle_start;
for (int i = cycle_start + 1; i < n; i++)
    if (arr[i] < item)
        pos++;

// If item is already in correct position
if (pos == cycle_start)
    continue;

// ignore all duplicate elements
while (item == arr[pos])
    pos += 1;

// put the item to it's right position
if (pos != cycle_start) {
    int temp = item;
    item = arr[pos];
    arr[pos] = temp;
    writes++;
}

// Rotate rest of the cycle
while (pos != cycle_start) {
    pos = cycle_start;

    // Find position where we put the element
    for (int i = cycle_start + 1; i < n; i++)
        if (arr[i] < item)
            pos += 1;

    // ignore all duplicate elements
    while (item == arr[pos])
        pos += 1;

    // put the item to it's right position
    if (item != arr[pos]) {
        int temp = item;
        item = arr[pos];
        arr[pos] = temp;
        writes++;
    }
}
}
```

```
}

// Driver program to test above function
public static void Main()
{
    int[] arr = { 1, 8, 3, 9, 10, 10, 2, 4 };
    int n = arr.Length;

    // Function calling
    cycleSort(arr, n);

    Console.Write("After sort : ");
    for (int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
}
}

// This code is contributed by Nitin Mittal
```

Output:

```
After sort :
1 2 3 4 8 9 10 10
```

Time Complexity : $O(n^2)$

Worst Case : $O(n^2)$

Average Case: $O(n^2)$

Best Case : $O(n^2)$

This sorting algorithm is best suited for situations where memory write or swap operations are costly.

Reference:

https://en.wikipedia.org/wiki/Cycle_sort

Improved By : [nitin mittal, rahul sholapurkar](#)

Source

<https://www.geeksforgeeks.org/cycle-sort/>

Chapter 79

De-arrangements for minimum product sum of two arrays

De-arrangements for minimum product sum of two arrays - GeeksforGeeks

Given two arrays A[] and B[] of same size n. We need to first permute any of arrays such that the sum of product of pairs(1 element from each) is minimum. That is **SUM (Ai*Bi)** for all i is minimum. We also need to count number of de-arrangements present in original array as compared to permuted array.

Examples:

```
Input : A[] = {4, 3, 2},  
        B[] = {7, 12, 5}  
Output : 3  
Explanation : A[] = {4, 3, 2} and B[] = {5, 7, 12}  
results in minimum product sum. B[] = {7, 12, 5}  
is 3-position different from new B[].
```

```
Input : A[] = {4, 3, 2},  
        B[] = { 1, 2, 3}  
Output : 0  
Explanation : A[] = {4, 3, 2} and B[] = {1, 2, 3}  
results in minimum product sum. B[] = {1, 2, 3}  
is exactly same as new one.
```

Idea behind finding the minimum sum of product from two array is to sort both array one in increasing and other in decreasing manner. These type of arrays will always produce minimum sum of pair product. Sorting both array will give the pair value i.e. which element from A is paired to which element from B[]. After that count the de-arrangement from original arrays.

Algorithm :

1. make a copy of both array.
2. sort copy_A[] in increasing, copy_B[] in decreasing order.
3. Iterate for all Ai, find Ai in copy_A[] as copy_A[j] and check whether copy_B[j] == B[i] or not. Increment count if not equal.
4. Return Count Value. That will be our answer.

```

// CPP program to count de-arrangements for
// minimum product.
#include<bits/stdc++.h>
using namespace std;

// function for finding de-arrangement
int findDearrange (int A[], int B[], int n)
{
    // create copy of array
    vector <int> copy_A (A, A+n);
    vector <int> copy_B (B, B+n);

    // sort array in inc & dec way
    sort(copy_A.begin(), copy_A.end());
    sort(copy_B.begin(), copy_B.end(), greater<int>());

    // count no. of de arrangements
    int count = 0;
    for (int i=0; i<n;i++)
    {
        vector<int>::iterator itA;

        // find position of A[i] in sorted array
        itA = lower_bound(copy_A.begin(),
                           copy_A.end(), A[i]);

        // check whether B[i] is same as required or not
        if (B[i] != copy_B[itA-copy_A.begin()])
            count++;
    }

    // return count
    return count;
}

// driver function
int main()
{
    int A[] = {1, 2, 3, 4};
    int B[] = {6, 3, 4, 5};

```

```
int n = sizeof(A) /sizeof(A[0]);;
cout << findDearrange(A,B,n);
return 0;
}
```

Output:

2

Source

<https://www.geeksforgeeks.org/de-arrangements-for-minimum-product-sum-of-two-arrays/>

Chapter 80

Delete consecutive same words in a sequence

Delete consecutive same words in a sequence - GeeksforGeeks

Given a sequence of n strings, the task is to check if any two similar words come together then they destroy each other then print the number of words left in the sequence after this pairwise destruction.

Examples:

Input : ab aa aa bcd ab

Output : 3

As aa, aa destroys each other so, ab bcd ab is the new sequence.

Input : tom jerry jerry tom

Output : 0

As first both jerry will destroy each other.

Then sequence will be tom, tom they will also destroy each other. So, the final sequence doesn't contain any word.

Method 1:

1- Start traversing the sequence by storing it in vector.

a) Check if the current string is equal to next string
if yes, erase both from the vector.

b) And check the same till last.

2- Return vector size.

C++

```
// C++ program to remove consecutive same words
#include<bits/stdc++.h>
using namespace std;

// Function to find the size of manipulated sequence
int removeConsecutiveSame(vector <string > v)
{
    int n = v.size();

    // Start traversing the sequence
    for (int i=0; i<n-1; )
    {
        // Compare the current string with next one
        // Erase both if equal
        if (v[i].compare(v[i+1]) == 0)
        {
            // Erase function delete the element and
            // also shifts other element that's why
            // i is not updated
            v.erase(v.begin()+i);
            v.erase(v.begin()+i);

            // Update i, as to check from previous
            // element again
            if (i > 0)
                i--;
        }

        // Reduce sequence size
        n = n-2;
    }

    // Increment i, if not equal
    else
        i++;
}

// Return modified size
return v.size();
}

// Driver code
int main()
{
    vector<string> v = { "tom", "jerry", "jerry", "tom"};
    cout << removeConsecutiveSame(v);
    return 0;
}
```

}

Java

```
// Java program to remove consecutive same words

import java.util.Vector;

class Test
{
    // Method to find the size of manipulated sequence
    static int removeConsecutiveSame(Vector <String > v)
    {
        int n = v.size();

        // Start traversing the sequence
        for (int i=0; i<n-1; )
        {
            // Compare the current string with next one
            // Erase both if equal
            if (v.get(i).equals(v.get(i+1)))
            {
                // Erase function delete the element and
                // also shifts other element that's why
                // i is not updated
                v.remove(i);
                v.remove(i);

                // Update i, as to check from previous
                // element again
                if (i > 0)
                    i--;
            }

            // Reduce sequence size
            n = n-2;
        }

        // Increment i, if not equal
        else
            i++;
    }

    // Return modified size
    return v.size();
}

// Driver method
public static void main(String[] args)
```

```
{  
    Vector<String> v = new Vector<>();  
  
    v.addElement("tom"); v.addElement("jerry");  
    v.addElement("jerry");v.addElement("tom");  
  
    System.out.println(removeConsecutiveSame(v));  
}  
}
```

Output:

0

Method 2:(Using Stack)

- 1- Start traversing the strings and push into stack.
 - a) Check if the current string is same as the string at the top of the stack
 - i) If yes, pop the string from top.
 - ii) Else push the current string.
- 2- Return stack size if the whole sequence is traversed.

C++

```
// C++ implementation of above method  
#include<bits/stdc++.h>  
using namespace std;  
  
// Function to find the size of manipulated sequence  
int removeConsecutiveSame(vector <string> v)  
{  
    stack<string> st;  
  
    // Start traversing the sequence  
    for (int i=0; i<v.size(); i++)  
    {  
        // Push the current string if the stack  
        // is empty  
        if (st.empty())  
            st.push(v[i]);  
        else  
        {
```

```
        string str = st.top();

        // compare the current string with stack top
        // if equal, pop the top
        if (str.compare(v[i]) == 0)
            st.pop();

        // Otherwise push the current string
        else
            st.push(v[i]);
    }
}

// Return stack size
return st.size();
}

// Driver code
int main()
{
    vector<string> V = { "ab", "aa", "aa", "bcd", "ab"};
    cout << removeConsecutiveSame(V);
    return 0;
}
```

Java

```
// Java implementation of above method

import java.util.Stack;
import java.util.Vector;

class Test
{
    // Method to find the size of manipulated sequence
    static int removeConsecutiveSame(Vector <String> v)
    {
        Stack<String> st = new Stack<>();

        // Start traversing the sequence
        for (int i=0; i<v.size(); i++)
        {
            // Push the current string if the stack
            // is empty
            if (st.empty())
                st.push(v.get(i));
            else
            {
```

```
String str = st.peek();

// compare the current string with stack top
// if equal, pop the top
if (str.equals(v.get(i)))
    st.pop();

// Otherwise push the current string
else
    st.push(v.get(i));
}

}

// Return stack size
return st.size();
}

// Driver method
public static void main(String[] args)
{
    Vector<String> v = new Vector<>();

    v.addElement("ab"); v.addElement("aa");
    v.addElement("aa");v.addElement("bcd");
    v.addElement("ab");

    System.out.println(removeConsecutiveSame(v));
}
}
```

Output:

3

Source

<https://www.geeksforgeeks.org/delete-consecutive-words-sequence/>

Chapter 81

Difference between highest and least frequencies in an array

Difference between highest and least frequencies in an array - GeeksforGeeks

Given an array, find the difference between highest occurrence and least occurrence of any number in an array

Examples:

```
Input : arr[] = [7, 8, 4, 5, 4, 1, 1, 7, 7, 2, 5]
```

```
Output : 2
```

```
Lowest occurring element (5) occurs once.
```

```
Highest occurring element (1 or 7) occurs 3 times
```

```
Input : arr[] = [1, 1, 1, 3, 3, 3]
```

```
Output : 0
```

A **simple solution** is to use two loops to count frequency of every element and keep track of maximum and minimum frequencies.

A **better solution** is to sort the array in $O(n \log n)$ and check consecutive element's occurrence and compare their count respectively.

CPP

```
// CPP code to find the difference between highest
// and least frequencies
#include <bits/stdc++.h>
using namespace std;

int findDiff(int arr[], int n)
{

```

```
// sort the array
sort(arr, arr + n);

int count = 0, max_count = 0, min_count = n;
for (int i = 0; i < (n - 1); i++) {

    // checking consecutive elements
    if (arr[i] == arr[i + 1]) {
        count += 1;
        continue;
    }
    else {
        max_count = max(max_count, count);
        min_count = min(min_count, count);
        count = 0;
    }
}

return (max_count - min_count);
}

// Driver
int main()
{
    int arr[] = { 7, 8, 4, 5, 4, 1, 1, 7, 7, 2, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << findDiff(arr, n) << "\n";
    return 0;
}
```

Java

```
// JAVA Code for Difference between
// highest and least frequencies
// in an array
import java.util.*;

class GFG {

    static int findDiff(int arr[], int n)
    {
        // sort the array
        Arrays.sort(arr);

        int count = 0, max_count = 0,
            min_count = n;
```

```
for (int i = 0; i < (n - 1); i++) {

    // checking consecutive elements
    if (arr[i] == arr[i + 1]) {
        count += 1;
        continue;
    }
    else {
        max_count = Math.max(max_count,
                             count);

        min_count = Math.min(min_count,
                             count);
        count = 0;
    }
}

return (max_count - min_count);
}

// Driver program to test above function
public static void main(String[] args)
{
    int arr[] = { 7, 8, 4, 5, 4, 1,
                 1, 7, 7, 2, 5 };
    int n = arr.length;

    System.out.println(findDiff(arr, n));
}
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
# Python3 code to find the difference
# between highest nd least frequencies

def findDiff(arr, n):

    # sort the array
    arr.sort()

    count = 0; max_count = 0; min_count = n
    for i in range(0, (n-1)):

        # checking consecutive elements
```

```
if arr[i] == arr[i + 1]:
    count += 1
    continue
else:
    max_count = max(max_count, count)
    min_count = min(min_count, count)
    count = 0
return max_count - min_count

# Driver Code
arr = [ 7, 8, 4, 5, 4, 1, 1, 7, 7, 2, 5 ]
n = len(arr)
print (findDiff(arr, n))

# This code is contributed by Shreyanshi Arun.
```

C#

```
// C# Code for Difference between
// highest and least frequencies
// in an array
using System;

class GFG {

    static int findDiff(int[] arr, int n)
    {

        // sort the array
        Array.Sort(arr);

        int count = 0, max_count = 0,
              min_count = n;

        for (int i = 0; i < (n - 1); i++) {

            // checking consecutive elements
            if (arr[i] == arr[i + 1]) {
                count += 1;
                continue;
            }
            else {
                max_count = Math.Max(max_count,
                                      count);

                min_count = Math.Min(min_count,
                                      count);
                count = 0;
            }
        }
        return max_count - min_count;
    }
}
```

```
        }
    }

    return (max_count - min_count);
}

// Driver program to test above function
public static void Main()
{
    int[] arr = { 7, 8, 4, 5, 4, 1,
                  1, 7, 7, 2, 5 };
    int n = arr.Length;

    Console.WriteLine(findDiff(arr, n));
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP code to find the
// difference between highest
// and least frequencies

// function that
// returns difference
function findDiff($arr, $n)
{

    // sort the array
    sort($arr);

    $count = 0; $max_count = 0;
    $min_count = $n;
    for ($i = 0; $i < ($n - 1); $i++)
    {

        // checking consecutive elements
        if ($arr[$i] == $arr[$i + 1])
        {
            $count += 1;
            continue;
        }
        else
        {
```

```
$max_count = max($max_count, $count);
$min_count = min($min_count, $count);
$count = 0;
}
}

return ($max_count - $min_count);
}

// Driver Code
$arr = array(7, 8, 4, 5, 4, 1,
             1, 7, 7, 2, 5);
$n = sizeof($arr);

echo(findDiff($arr, $n) . "\n");

// This code is contributed by Ajit.
?>
```

Output:

2

An **efficient solution** is to use hashing. We count frequencies of all elements and finally traverse the hash table to find maximum and minimum.

Below is c++ implementation

```
// CPP code to find the difference between highest
// and least frequencies
#include <bits/stdc++.h>
using namespace std;

int findDiff(int arr[], int n)
{
    // Put all elements in a hash map
    unordered_map<int, int> hm;
    for (int i = 0; i < n; i++)
        hm[arr[i]]++;

    // Find counts of maximum and minimum
    // frequent elements
    int max_count = 0, min_count = n;
    for (auto x : hm) {
        max_count = max(max_count, x.second);
        min_count = min(min_count, x.second);
    }
}
```

```
    return (max_count - min_count);
}

// Driver
int main()
{
    int arr[] = { 7, 8, 4, 5, 4, 1, 1, 7, 7, 2, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << findDiff(arr, n) << "\n";
    return 0;
}
```

Output:

2

Time Complexity : O(n)

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/difference-between-highest-and-least-frequencies-in-an-array/>

Chapter 82

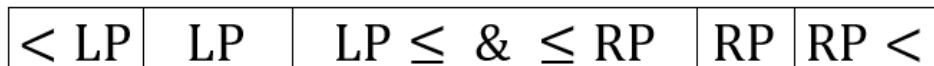
Dual pivot Quicksort

Dual pivot Quicksort - GeeksforGeeks

As we know, the single pivot [quick sort](#) takes a pivot from one of the ends of the array and partitioning the array, so that all elements are left to the pivot are less than or equal to the pivot, and all elements that are right to the pivot are greater than the pivot.

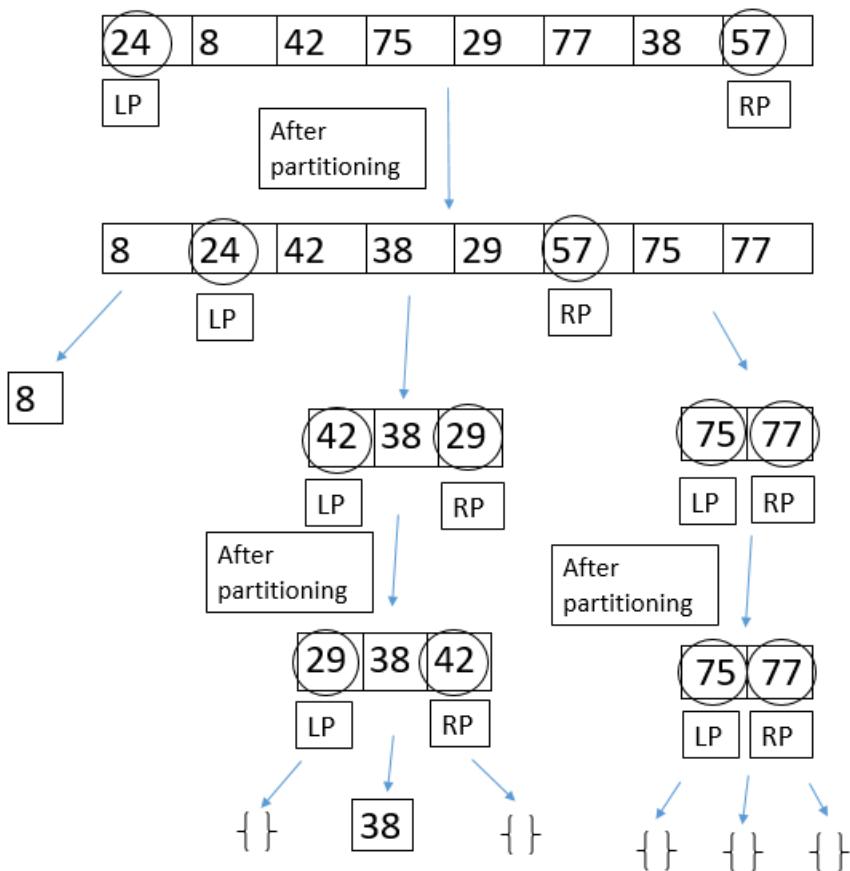
The idea of dual pivot quick sort is to take two pivots, one in the left end of the array and the second, in the right end of the array. The left pivot must be less than or equal to the right pivot, so we swap them if necessary.

Then, we begin partitioning the array into three parts: in the first part, all elements will be less than the left pivot, in the second part all elements will be greater or equal to the left pivot and also will be less than or equal to the right pivot, and in the third part all elements will be greater than the right pivot. Then, we shift the two pivots to their appropriate positions as we see in the below bar, and after that we begin quicksorting these three parts recursively, using this method.



Dual pivot quick sort is a little bit faster than the original single pivot quicksort. But still, the worst case will remain $O(n^2)$ when the array is already sorted in an increasing or decreasing order.

An example:



```

// C program to implement dual pivot QuickSort
#include <stdio.h>

int partition(int* arr, int low, int high, int* lp);

void swap(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void DualPivotQuickSort(int* arr, int low, int high)
{
    if (low < high) {
        // lp means left pivot, and rp means right pivot.
        int lp, rp;
        rp = partition(arr, low, high, &lp);
        DualPivotQuickSort(arr, low, lp - 1);
    }
}

```

```

        DualPivotQuickSort(arr, lp + 1, rp - 1);
        DualPivotQuickSort(arr, rp + 1, high);
    }
}

int partition(int* arr, int low, int high, int* lp)
{
    if (arr[low] > arr[high])
        swap(&arr[low], &arr[high]);
    // p is the left pivot, and q is the right pivot.
    int j = low + 1;
    int g = high - 1, k = low + 1, p = arr[low], q = arr[high];
    while (k <= g) {

        // if elements are less than the left pivot
        if (arr[k] < p) {
            swap(&arr[k], &arr[j]);
            j++;
        }

        // if elements are greater than or equal
        // to the right pivot
        else if (arr[k] >= q) {
            while (arr[g] > q && k < g)
                g--;
            swap(&arr[k], &arr[g]);
            g--;
            if (arr[k] < p) {
                swap(&arr[k], &arr[j]);
                j++;
            }
        }
        k++;
    }
    j--;
    g++;

    // bring pivots to their appropriate positions.
    swap(&arr[low], &arr[j]);
    swap(&arr[high], &arr[g]);

    // returning the indeces of the pivots.
    *lp = j; // because we cannot return two elements
             // from a function.

    return g;
}

```

```
// Driver code
int main()
{
    int arr[] = { 24, 8, 42, 75, 29, 77, 38, 57 };
    DualPivotQuickSort(arr, 0, 7);
    printf("Sorted array: ");
    for (int i = 0; i < 8; i++)
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

Output:

```
Sorted array: 8 24 29 38 42 57 75 77
```

Source

<https://www.geeksforgeeks.org/dual-pivot-quicksort/>

Chapter 83

Efficiently merging two sorted arrays with O(1) extra space

Efficiently merging two sorted arrays with O(1) extra space - GeeksforGeeks

Given two sorted arrays, we need to merge them in $O((n+m)*\log(n+m))$ time with $O(1)$ extra space into a sorted array, where n is the size of the first array, and m is the size of the second array.

Example:

```
Input: ar1[] = {10};  
       ar2[] = {2, 3};  
Output: ar1[] = {2}  
       ar2[] = {3, 10}  
  
Input: ar1[] = {1, 5, 9, 10, 15, 20};  
       ar2[] = {2, 3, 8, 13};  
Output: ar1[] = {1, 2, 3, 5, 8, 9}  
       ar2[] = {10, 13, 15, 20}
```

We have discussed a quadratic time solution in below post.

[Merge two sorted arrays with O\(1\) extra space](#)

In this post a better solution is discussed.

The idea: we start comparing elements that are far from each other rather than adjacent. For every pass, we calculate the gap and compare the elements towards the right of the gap. Every pass, the gap reduces to the ceiling value of dividing by 2.

Examples:

First example: $a1[] = \{3 27 38 43\}$, $a2[] = \{9 10 82\}$
Start with gap = ceiling of $n/2 = 4$ [This gap is for
whole merged array]

```
3 27 38 43  9 10 82
3 27 38 43  9 10 82
3 10 38 43  9 27 82
gap = 2:
3 10 38 43  9 27 82
3 10 38 43  9 27 82
3 10 38 43  9 27 82
3 27 9 10   38 43 82
3 27 9 10   38 43 82
gap = 1:
3 27 9 10   38 43 82
3 27 9 10   38 43 82
3 9 27 10   38 43 82
3 9 10 27   38 43 82
3 9 10 27   38 43 82
3 9 10 27   38 43 82
Output : 3 9 10 27 38 43 82
```

Second Example: $a1[] = \{10 27 38 43 82\}$, $a2[] = \{3 9\}$
Start with gap = ceiling of $n/2 (4)$:

```
10 27 38 43 82  3 9
10 27 38 43 82  3 9
10 3 38 43 82   27 9
10 3 9 43 82   27 38
gap = 2:
10 3 9 43 82   27 38
9 3 10 43 82   27 38
9 3 10 43 82   27 38
9 3 10 43 82   27 38
9 3 10 27 82   43 38
9 3 10 27 38   43 82
gap = 1
9 3 10 27 38   43 82
3 9 10 27 38   43 82
3 9 10 27 38   43 82
3 9 10 27 38   43 82
3 9 10 27 38   43 82
Output : 3 9 10 27 38   43 82
```

C++

```
// Merging two sorted arrays with O(1)
// extra space
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to find next gap.
int nextGap(int gap)
{
    if (gap <= 1)
        return 0;
    return (gap / 2) + (gap % 2);
}

void merge(int *arr1, int *arr2, int n, int m)
{
    int i, j, gap = n + m;
    for (gap = nextGap(gap); gap > 0; gap = nextGap(gap))
    {
        // comparing elements in the first array.
        for (i = 0; i + gap < n; i++)
            if (arr1[i] > arr1[i + gap])
                swap(arr1[i], arr1[i + gap]);

        //comparing elements in both arrays.
        for (j = gap > n ? gap - n : 0; i < n && j < m; i++, j++)
            if (arr1[i] > arr2[j])
                swap(arr1[i], arr2[j]);

        if (j < m)
        {
            //comparing elements in the second array.
            for (j = 0; j + gap < m; j++)
                if (arr2[j] > arr2[j + gap])
                    swap(arr2[j], arr2[j + gap]);
        }
    }
}

// Driver code
int main()
{
    int a1[] = { 10, 27, 38, 43, 82 };
    int a2[] = { 3, 9 };
    int n = sizeof(a1) / sizeof(int);
    int m = sizeof(a2) / sizeof(int);

    merge(a1, a2, n, m);

    printf("First Array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", a1[i]);
}
```

```
    printf("\nSecond Array: ");
    for (int i = 0; i < m; i++)
        printf("%d ", a2[i]);
    printf("\n");
    return 0;
}
```

Java

```
// Java program for Merging two sorted arrays
// with O(1) extra space

public class MergeTwoSortedArrays {

    // Function to find next gap.
    private static int nextGap(int gap)
    {
        if (gap <= 1)
            return 0;
        return (gap / 2) + (gap % 2);
    }

    private static void merge(int[] arr1, int[] arr2,
                             int n, int m) {
        int i, j, gap = n + m;
        for (gap = nextGap(gap); gap > 0;
             gap = nextGap(gap))
        {
            // comparing elements in the first
            // array.
            for (i = 0; i + gap < n; i++)
                if (arr1[i] > arr1[i + gap]) {
                    int temp = arr1[i];
                    arr1[i] = arr1[i + gap];
                    arr1[i+gap] = temp;
                }

            // comparing elements in both arrays.
            for (j = gap > n ? gap - n : 0 ;
                 i < n && j < m; i++, j++)
                if (arr1[i] > arr2[j]) {
                    int temp = arr1[i];
                    arr1[i] = arr2[j];
                    arr2[j] = temp;
                }
        }

        if (j < m)
```

```
{  
    // comparing elements in the  
    // second array.  
    for (j = 0; j + gap < m; j++)  
        if (arr2[j] > arr2[j + gap]) {  
            int temp = arr2[j];  
            arr2[j] = arr2[j + gap];  
            arr2[j+gap] = temp;  
        }  
    }  
}  
  
public static void main(String[] args) {  
    int[] a1 = { 10, 27, 38, 43 ,82 };  
    int[] a2 = { 3,9 };  
  
    merge(a1, a2, a1.length, a2.length);  
  
    System.out.print("First Array: ");  
    for (int i = 0; i < a1.length; i++) {  
        System.out.print(a1[i] + " ");  
    }  
  
    System.out.println();  
  
    System.out.print("Second Array: ");  
    for (int i = 0; i < a2.length; i++) {  
        System.out.print(a2[i] + " ");  
    }  
}  
}  
  
// This code is contributed by Vinisha Shah
```

C#

```
// C# program for Merging two sorted arrays  
// with  $O(1)$  extra space  
using System;  
  
class GFG {  
  
    // Function to find next gap.  
    static int nextGap(int gap)  
    {  
        if (gap <= 1)  
            return 0;
```

```

        return (gap / 2) + (gap % 2);
    }

private static void merge(int[] arr1,
                         int[] arr2, int n, int m)
{
    int i, j, gap = n + m;
    for (gap = nextGap(gap); gap > 0;
         gap = nextGap(gap))
    {
        // comparing elements in the first
        // array.
        for (i = 0; i + gap < n; i++)
            if (arr1[i] > arr1[i + gap]) {
                int temp = arr1[i];
                arr1[i] = arr1[i + gap];
                arr1[i+gap] = temp;
            }

        // comparing elements in both arrays.
        for (j = gap > n ? gap - n : 0 ;
             i < n && j < m; i++, j++)
            if (arr1[i] > arr2[j]) {
                int temp = arr1[i];
                arr1[i] = arr2[j];
                arr2[j] = temp;
            }

        if (j < m)
        {
            // comparing elements in the
            // second array.
            for (j = 0; j + gap < m; j++)
                if (arr2[j] > arr2[j + gap])
                {
                    int temp = arr2[j];
                    arr2[j] = arr2[j + gap];
                    arr2[j+gap] = temp;
                }
        }
    }

    // Driver code
    public static void Main()
    {
        int[] a1 = { 10, 27, 38, 43 ,82 };
        int[] a2 = { 3,9 };
    }
}

```

```
    merge(a1, a2, a1.Length, a2.Length);

    Console.WriteLine("First Array: ");
    for (int i = 0; i < a1.Length; i++) {
        Console.Write(a1[i] + " ");
    }

    Console.WriteLine();

    Console.WriteLine("Second Array: ");
    for (int i = 0; i < a2.Length; i++) {
        Console.Write(a2[i] + " ");
    }
}

// This code is contributed by Sam007.
```

PHP

```
<?php
// Merging two sorted arrays
// with O(1) extra space

// Function to find next gap.
function nextGap($gap)
{
    if ($gap <= 1)
        return 0;
    return ($gap / 2) +
        ($gap % 2);
}

function merge($arr1, $arr2,
              $n, $m)
{
    $i;
    $j;
    $gap = $n + $m;
    for ($gap = nextGap($gap);
         $gap > 0; $gap = nextGap($gap))
    {
        // comparing elements
        // in the first array.
        for ($i = 0; $i + $gap < $n; $i++)
            if ($arr1[$i] > $arr1[$i + $gap])
            {
```

```

$tmp = $arr1[$i];
$arr1[$i] = $arr1[$i + $gap];
$arr1[$i + $gap] = $tmp;
}

// comparing elements
// in both arrays.
for ($j = $gap > $n ? $gap - $n : 0 ;
     $i < $n && $j < $m; $i++, $j++)
    if ($arr1[$i] > $arr2[$j])
    {
        $tmp = $arr1[$i];
        $arr1[$i] = $arr2[$j];
        $arr2[$j] = $tmp;
    }

if ($j < $m)
{
    // comparing elements in
    // the second array.
    for ($j = 0; $j + $gap < $m; $j++)
        if ($arr2[$j] > $arr2[$j + $gap])
        {
            $tmp = $arr2[$j];
            $arr2[$j] = $arr2[$j + $gap];
            $arr2[$j + $gap] = $tmp;
        }
    }
}

echo "First Array: ";
for ($i = 0; $i < $n; $i++)
    echo $arr1[$i]. " ";

echo "\nSecond Array: ";
for ($i = 0; $i < $m; $i++)
    echo $arr2[$i]. " ";
echo"\n";

}

// Driver code
$a1 = array(10, 27, 38, 43 ,82);
$a2 = array(3,9);
$n = sizeof($a1);
$m = sizeof($a2);

merge($a1, $a2, $n, $m);

```

```
// This code is contributed  
// by mits.  
?>
```

Output:

```
First Array: 3 9 10 27 38  
Second Array: 43 82
```

Improved By : [Sam007, Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/efficiently-merging-two-sorted-arrays-with-o1-extra-space/>

Chapter 84

Elements that occurred only once in the array

Elements that occurred only once in the array - GeeksforGeeks

Given an array arr that has numbers appearing twice or once. The task is to identify numbers that occurred only once in the array.

Note: Duplicates appear side by side every time. Might be few numbers can occur one time and just assume this is a right rotating array (just say an array can rotate k times towards right). Order of the elements in the output doesn't matter.

Examples:

Input: $arr[] = \{ 7, 7, 8, 8, 9, 1, 1, 4, 2, 2 \}$
Output: 9 4

Input: $arr[] = \{-9, -8, 4, 4, 5, 5, -1\}$
Output: -9 -8 -1

Method-1: Using *Sorting*.

- Sort the array.
- Check for each element at index i (except the first and last element), if

$arr[i] != arr[i-1] \&& arr[i] != arr[i+1]$

- For the first element, check if $arr[0] != arr[1]$.
- For the last element, check if $arr[n-1] != arr[n-2]$.

Below is the implementation of above approach:

C++

```
// C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// Function to find the elements that
// appeared only once in the array
void occurredOnce(int arr[], int n)
{
    // Sort the array
    sort(arr, arr + n);

    // Check for first element
    if (arr[0] != arr[1])
        cout << arr[0] << " ";

    // Check for all the elements if it is different
    // its adjacent elements
    for (int i = 1; i < n - 1; i++)
        if (arr[i] != arr[i + 1] && arr[i] != arr[i - 1])
            cout << arr[i] << " ";

    // Check for the last element
    if (arr[n - 2] != arr[n - 1])
        cout << arr[n - 1] << " ";
}

// Driver code
int main()
{
    int arr[] = { 7, 7, 8, 8, 9, 1, 1, 4, 2, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    occurredOnce(arr, n);

    return 0;
}
```

Java

```
// Java implementation
// of above approach
import java.util.*;
```

```
class GFG
{
    // Function to find the elements that
    // appeared only once in the array
    static void occurredOnce(int arr[], int n)
    {
        // Sort the array
        Arrays.sort(arr);

        // Check for first element
        if (arr[0] != arr[1])
            System.out.println(arr[0] + " ");

        // Check for all the elements
        // if it is different
        // its adjacent elements
        for (int i = 1; i < n - 1; i++)
            if (arr[i] != arr[i + 1] &&
                arr[i] != arr[i - 1])
                System.out.print(arr[i] + " ");

        // Check for the last element
        if (arr[n - 2] != arr[n - 1])
            System.out.print(arr[n - 1] + " ");
    }

    // Driver code
    public static void main(String args[])
    {
        int arr[] = {7, 7, 8, 8, 9,
                    1, 1, 4, 2, 2};
        int n = arr.length;

        occurredOnce(arr, n);
    }
}

// This code is contributed
// by Arnab Kundu
```

Python 3

```
# Python 3 implementation
# of above approach

# Function to find the elements
# that appeared only once in
```

```
# the array
def occurredOnce(arr, n):

    # Sort the array
    arr.sort()

    # Check for first element
    if arr[0] != arr[1]:
        print(arr[0], end = " ")

    # Check for all the elements
    # if it is different its
    # adjacent elements
    for i in range(1, n - 1):
        if (arr[i] != arr[i + 1] and
            arr[i] != arr[i - 1]):
            print( arr[i], end = " ")

    # Check for the last element
    if arr[n - 2] != arr[n - 1]:
        print(arr[n - 1], end = " ")

# Driver code
if __name__ == "__main__":
    arr = [ 7, 7, 8, 8, 9,
            1, 1, 4, 2, 2 ]
    n = len(arr)
    occurredOnce(arr, n)

# This code is contributed
# by ChitraNayal
```

C#

```
// C# implementation
// of above approach
using System;

class GFG
{

    // Function to find the elements that
    // appeared only once in the array
    static void occurredOnce(int[] arr, int n)
    {
        // Sort the array
        Array.Sort(arr);
```

```
// Check for first element
if (arr[0] != arr[1])
    Console.WriteLine(arr[0] + " ");

// Check for all the elements
// if it is different
// its adjacent elements
for (int i = 1; i < n - 1; i++)
    if (arr[i] != arr[i + 1] &&
        arr[i] != arr[i - 1])
        Console.WriteLine(arr[i] + " ");

// Check for the last element
if (arr[n - 2] != arr[n - 1])
    Console.WriteLine(arr[n - 1] + " ");
}

// Driver code
public static void Main()
{
    int[] arr = {7, 7, 8, 8, 9,
                1, 1, 4, 2, 2};
    int n = arr.Length;

    occurredOnce(arr, n);
}
}

// This code is contributed
// by ChitraNayal
```

PHP

```
<?php
// PHP implementation
// of above approach

// Function to find the elements
// that appeared only once in
// the array
function occurredOnce(&$arr, $n)
{
    // Sort the array
    sort($arr);

    // Check for first element
    if ($arr[0] != $arr[1])
```

```
echo $arr[0]." ";

// Check for all the elements
// if it is different its
// adjacent elements
for ($i = 1; $i < $n - 1; $i++)
    if ($arr[$i] != $arr[$i + 1] &&
        $arr[$i] != $arr[$i - 1])
        echo $arr[$i]." ";

// Check for the last element
if ($arr[$n - 2] != $arr[$n - 1])
    echo $arr[$n - 1]." ";
}

// Driver code
$arr = array(7, 7, 8, 8, 9,
            1, 1, 4, 2, 2);
$n = sizeof($arr);
occurredOnce($arr, $n);

// This code is contributed
// by ChitraNayal
?>
```

Output:

4 9

Time Complexity: O(Nlogn)
Space Complexity: O(1)

Method-2: (*Using Hashing*): In C++, `unordered_map` can be used for hashing.

- Traverse the array.
- Store each element with its occurrence in the `unordered_map`.
- Traverse the `unordered_map` and print all the elements with occurrence 1.

Below is the implementation of above approach:

C++

```
// C++ implementation to find elements
// that appeared only once
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to find the elements that
// appeared only once in the array
void occurredOnce(int arr[], int n)
{
    unordered_map<int, int> mp;

    // Store all the elements in the map with
    // their occurrence
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // Traverse the map and print all the
    // elements with occurrence 1
    for (auto it = mp.begin(); it != mp.end(); it++)
        if (it->second == 1)
            cout << it->first << " ";
}

// Driver code
int main()
{
    int arr[] = { 7, 7, 8, 8, 9, 1, 1, 4, 2, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    occurredOnce(arr, n);

    return 0;
}
```

Output:

4 9

Time Complexity: O(N)

Space Complexity: O(N)

Method-3: Using given assumptions.

It is given that an array can be rotated any times and duplicates will appear side by side every time. So after rotating the first and last element will appear side by side.

- Check if the first and last element is equal. If yes then start traversing the elements between them.
- Check if the current element is equal to the element at immediate previous index. If yes, check the same for next element.

- If not, print the current element.

C++

```
// C++ implementation to find elements
// that appeared only once
#include <bits/stdc++.h>
using namespace std;

// Function to find the elements that
// appeared only once in the array
void occurredOnce(int arr[], int n)
{
    int i = 1, len = n;

    // Check if the first and last element is equal
    // If yes, remove those elements
    if (arr[0] == arr[len - 1]) {
        i = 2;
        len--;
    }

    // Start traversing the remaining elements
    for (; i < n; i++) {

        // Check if current element is equal to
        // the element at immediate previous index
        // If yes, check the same for next element
        if (arr[i] == arr[i - 1])
            i++;

        // Else print the current element
        else
            cout << arr[i - 1] << " ";

        // Check for the last element
        if (arr[n - 1] != arr[0] && arr[n - 1] != arr[n - 2])
            cout << arr[n - 1];
    }
}

// Driver code
int main()
{

    int arr[] = { 7, 7, 8, 8, 9, 1, 1, 4, 2, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    occurredOnce(arr, n);
}
```

```
        return 0;
    }

Java

// Java implementation to find
// elements that appeared only once
class GFG
{
// Function to find the elements that
// appeared only once in the array
static void occurredOnce(int arr[], int n)
{
    int i = 1, len = n;

    // Check if the first and last
    // element is equal. If yes,
    // remove those elements
    if (arr[0] == arr[len - 1])
    {
        i = 2;
        len--;
    }

    // Start traversing the
    // remaining elements
    for (; i < n; i++)

        // Check if current element is
        // equal to the element at
        // immediate previous index
        // If yes, check the same
        // for next element
        if (arr[i] == arr[i - 1])
            i++;

        // Else print the current element
        else
            System.out.print(arr[i - 1] + " ");

    // Check for the last element
    if (arr[n - 1] != arr[0] &&
        arr[n - 1] != arr[n - 2])
        System.out.print(arr[n - 1]);
}

// Driver code
```

```
public static void main(String args[])
{
    int arr[] = {7, 7, 8, 8, 9,
                 1, 1, 4, 2, 2};
    int n = arr.length;

    occurredOnce(arr, n);
}
}

// This code is contributed
// by Arnab Kundu
```

Python 3

```
# Python 3 implementation to find
# elements that appeared only once

# Function to find the elements that
# appeared only once in the array
def occurredOnce(arr, n):
    i = 1
    len = n

    # Check if the first and
    # last element is equal
    # If yes, remove those elements
    if arr[0] == arr[len - 1]:
        i = 2
        len -= 1

    # Start traversing the
    # remaining elements
    while i < n:

        # Check if current element is
        # equal to the element at
        # immediate previous index
        # If yes, check the same for
        # next element
        if arr[i] == arr[i - 1]:
            i += 1

        # Else print the current element
        else:
            print(arr[i - 1], end = " ")

        i += 1
```

```
# Check for the last element
if (arr[n - 1] != arr[0] and
    arr[n - 1] != arr[n - 2]):
    print(arr[n - 1])

# Driver code
if __name__ == "__main__":
    arr = [ 7, 7, 8, 8, 9, 1, 1, 4, 2, 2 ]
    n = len(arr)

    occurredOnce(arr, n)

# This code is contributed
# by ChitraNayal

C#
// C# implementation to find
// elements that appeared only once
using System;

class GFG
{
// Function to find the elements that
// appeared only once in the array
static void occurredOnce(int[] arr, int n)
{
    int i = 1, len = n;

    // Check if the first and last
    // element is equal. If yes,
    // remove those elements
    if (arr[0] == arr[len - 1])
    {
        i = 2;
        len--;
    }

    // Start traversing the
    // remaining elements
    for (; i < n; i++)

        // Check if current element is
        // equal to the element at
        // immediate previous index
        // If yes, check the same
        // for next element
```

```
if (arr[i] == arr[i - 1])
    i++;

// Else print the current element
else
    Console.WriteLine(arr[i - 1] + " ");

// Check for the last element
if (arr[n - 1] != arr[0] &&
    arr[n - 1] != arr[n - 2])
    Console.WriteLine(arr[n - 1]);
}

// Driver code
public static void Main()
{
    int[] arr = {7, 7, 8, 8, 9,
                 1, 1, 4, 2, 2};
    int n = arr.Length;

    occurredOnce(arr, n);
}
}

// This code is contributed
// by ChitraNayal
```

PHP

```
<?php
// PHP implementation to find
// elements that appeared only once

// Function to find the elements that
// appeared only once in the array
function occurredOnce(&$arr, $n)
{
    $i = 1;
    $len = $n;

    // Check if the first and last
    // element is equal. If yes,
    // remove those elements
    if ($arr[0] == $arr[$len - 1])
    {
        $i = 2;
        $len--;
    }
}
```

```
// Start traversing the
// remaining elements
for (; $i < $n; $i++)

    // Check if current element is
    // equal to the element at
    // immediate previous index
    // If yes, check the same for
    // next element
    if ($arr[$i] == $arr[$i - 1])
        $i++;

    // Else print the current element
    else
        echo $arr[$i - 1] . " ";

    // Check for the last element
    if ($arr[$n - 1] != $arr[0] &&
        $arr[$n - 1] != $arr[$n - 2])
        echo $arr[$n - 1];
}

// Driver code
$arr = array(7, 7, 8, 8, 9,
            1, 1, 4, 2, 2);
$n = sizeof($arr);

occurredOnce($arr, $n);

// This code is contributed
// by ChitraNayal
?>
```

Output:

9 4

Time Complexity: O(N)
Space Complexity: O(1)

Improved By : [andrew1234](#), [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/elements-that-occurred-only-once-in-the-array/>

Chapter 85

Elements to be added so that all elements of a range are present in array

Elements to be added so that all elements of a range are present in array - GeeksforGeeks

Given an array of size N. Let A and B be the minimum and maximum in the array respectively. Task is to find how many number should be added to the given array such that all the element in the range [A, B] occur at-least once in the array.

Examples:

```
Input : arr[] = {4, 5, 3, 8, 6}
Output : 1
Only 7 to be added in the list.
```

```
Input : arr[] = {2, 1, 3}
Output : 0
```

Method 1 (Sorting)

- 1- Sort the array.
- 2- Compare $\text{arr}[i] == \text{arr}[i+1]-1$ or not. If not, update count = $\text{arr}[i+1]-\text{arr}[i]-1$.
- 3- Return count.

C++

```
// C++ program for above implementation
#include <bits/stdc++.h>
using namespace std;

// Function to count numbers to be added
```

```
int countNum(int arr[], int n)
{
    int count = 0;

    // Sort the array
    sort(arr, arr + n);

    // Check if elements are consecutive
    // or not. If not, update count
    for (int i = 0; i < n - 1; i++)
        if (arr[i] != arr[i+1] &&
            arr[i] != arr[i + 1] - 1)
            count += arr[i + 1] - arr[i] - 1;

    return count;
}

// Drivers code
int main()
{
    int arr[] = { 3, 5, 8, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << countNum(arr, n) << endl;
    return 0;
}
```

Java

```
// java program for above implementation
import java.io.*;
import java.util.*;

public class GFG {

    // Function to count numbers to be added
    static int countNum(int []arr, int n)
    {
        int count = 0;

        // Sort the array
        Arrays.sort(arr);

        // Check if elements are consecutive
        // or not. If not, update count
        for (int i = 0; i < n - 1; i++)
            if (arr[i] != arr[i+1] &&
                arr[i] != arr[i + 1] - 1)
                count += arr[i + 1] - arr[i] - 1;
    }
}
```

```
        return count;
    }

// Drivers code
static public void main (String[] args)
{
    int []arr = { 3, 5, 8, 6 };
    int n = arr.length;

    System.out.println(countNum(arr, n));
}
}

// This code is contributed by vt_m.
```

Python3

```
# python program for above implementation

# Function to count numbers to be added
def countNum(arr, n):

    count = 0

    # Sort the array
    arr.sort()

    # Check if elements are consecutive
    # or not. If not, update count
    for i in range(0, n-1):
        if (arr[i] != arr[i+1] and
            arr[i] != arr[i + 1] - 1):
            count += arr[i + 1] - arr[i] - 1;

    return count

# Drivers code
arr = [ 3, 5, 8, 6 ]
n = len(arr)
print(countNum(arr, n))

# This code is contributed by Sam007
```

C#

```
// C# program for above implementation
```

```
using System;

public class GFG {

    // Function to count numbers to be added
    static int countNum(int []arr, int n)
    {
        int count = 0;

        // Sort the array
        Array.Sort(arr);

        // Check if elements are consecutive
        // or not. If not, update count
        for (int i = 0; i < n - 1; i++)
            if (arr[i] != arr[i+1] &&
                arr[i] != arr[i + 1] - 1)
                count += arr[i + 1] - arr[i] - 1;

        return count;
    }

    // Drivers code
    static public void Main ()
    {

        int []arr = { 3, 5, 8, 6 };
        int n = arr.Length;

        Console.WriteLine(countNum(arr, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program for
// above implementation

// Function to count
// numbers to be added
function countNum($arr, $n)
{

    $count = 0;
```

```
// Sort the array
sort($arr);

// Check if elements are
// consecutive or not.
// If not, update count
for ($i = 0; $i < $n - 1; $i++)
    if ($arr[$i] != $arr[$i + 1] &&
        $arr[$i] != $arr[$i + 1] - 1)
        $count += $arr[$i + 1] -
                  $arr[$i] - 1;

return $count;
}

// Driver code
$arr = array(3, 5, 8, 6);
$n = count($arr);
echo countNum($arr, $n) ;

// This code is contributed
// by anuj_67.
?>
```

Output:

2

Time Complexity: $O(n \log n)$

Method 2 (Use Hashing)

- 1- Maintain a hash of array elements.
- 2- Store minimum and maximum element.
- 3- Traverse from minimum to maximum element in hash
And count if element is not in hash.
- 4- Return count.

C++

```
// C++ program for above implementation
#include <bits/stdc++.h>
using namespace std;

// Function to count numbers to be added
int countNum(int arr[], int n)
{
    unordered_set<int> s;
    int count = 0, maxm = INT_MIN, minm = INT_MAX;
```

```
// Make a hash of elements
// and store minimum and maximum element
for (int i = 0; i < n; i++) {
    s.insert(arr[i]);
    if (arr[i] < minm)
        minm = arr[i];
    if (arr[i] > maxm)
        maxm = arr[i];
}

// Traverse all elements from minimum
// to maximum and count if it is not
// in the hash
for (int i = minm; i <= maxm; i++)
    if (s.find(arr[i]) == s.end())
        count++;
return count;
}

// Drivers code
int main()
{
    int arr[] = { 3, 5, 8, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << countNum(arr, n) << endl;
    return 0;
}
```

Output:

2

Time Complexity- $O(\max - \min + 1)$

Improved By : [vt_m](#), [Sam007](#)

Source

<https://www.geeksforgeeks.org/elements-to-be-added-so-that-all-elements-of-a-range-are-present-in-array/>

Chapter 86

Equally divide into two sets such that one set has maximum distinct elements

Equally divide into two sets such that one set has maximum distinct elements - Geeks-forGeeks

There are two processes P1 and P2, and N resources where N is an even number. There is an array of N size and arr[i] represents the type of ith resource. There may be more than one instance of a resource. You are to divide these resources equally between P1 and P2 such that maximum number of distinct number of resources are allocated to P2. Print maximum number of distinct resources allocated to P2.

Examples:

Input : arr[] = [1, 1, 2, 2, 3, 3]

Output: 3

Explanation:

There are three different kinds of resources (1, 2 and 3), and two for each kind.

Optimal distribution: Process P1 has resources [1, 2, 3] and the process P2 has gifts [1, 2, 3], too. Process p2 has 3 distinct resources.

arr[] = [1, 1, 2, 1, 3, 4]

Output: 3

Explanation:

There are three different kinds of resources (1, 2, 3, 4), 3 instances of 1 and single single instances of resource 2, 3, 4. Optimal distribution: Process P1 has resources [1, 1, 1] and the process P2 has gifts [2, 3, 4].

Process p2 has 3 distinct resources.

Approach 1 (Using sorting):

1. Sort the Array of resources.

2. Find out the elements which are unique by comparing the adjacent elements of the sorted array.suppose count holds the distinct number of resources in array.
3. Return the minimum of count and N/2.

Time complexity- O(N log N)

C++

```
// C++ program to equally divide n elements
// into two sets such that second set has
// maximum distinct elements.
#include <algorithm>
#include <iostream>
using namespace std;

int distribution(int arr[], int n)
{
    sort(arr, arr + n);
    int count = 1;
    for (int i = 1; i < n; i++)
        if (arr[i] > arr[i - 1])
            count++;

    return min(count, n / 2);
}

// Driver code
int main()
{
    int arr[] = { 1, 1, 2, 1, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << distribution(arr, n) << endl;
    return 0;
}
```

Java

```
// Java program to equally divide n elements
// into two sets such that second set has
// maximum distinct elements.
import java.util.*;
class Geeks {

    static int distribution(int arr[], int n)
    {
        Arrays.sort(arr);
        int count = 1;
        for (int i = 1; i < n; i++)
```

```
        if (arr[i] > arr[i - 1])
            count++;

    return Math.min(count, n / 2);
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 1, 1, 2, 1, 3, 4 };
    int n = arr.length;
    System.out.println(distribution(arr, n));
}
}

// This code is contributed by ankita_saini
```

C#

```
// C# program to equally divide
// n elements into two sets such
// that second set has maximum
// distinct elements.
using System;

class GFG
{
static int distribution(int []arr, int n)
{
    Array.Sort(arr);
    int count = 1;
    for (int i = 1; i < n; i++)
        if (arr[i] > arr[i - 1])
            count++;

    return Math.Min(count, n / 2);
}

// Driver code
public static void Main(String []args)
{
    int []arr= { 1, 1, 2, 1, 3, 4 };
    int n = arr.Length;
    Console.WriteLine(distribution(arr, n));
}
}

// This code is contributed
```

```
// by ankita_saini
```

Ouput:

```
3
```

Approach 2(using hash set)

Another way to find out distinct element is set, insert all the element in the set. By the property of a set, it will contain only unique elements. At the end, we can count the number of elements in the set, given by, say count. The value to be returned will again be given by $\min(\text{count}, n/2)$.

C++

```
// C++ program to equally divide n elements
// into two sets such that second set has
// maximum distinct elements.
#include <bits/stdc++.h>
using namespace std;

int distribution(int arr[], int n)
{
    unordered_set<int, greater<int> > resources;

    // Insert all the resources in the set
    // There will be unique resources in the set
    for (int i = 0; i < n; i++)
        resources.insert(arr[i]);

    // return minimum of distinct resources
    // and n/2
    return min(resources.size(), n / 2);
}

// Driver code
int main()
{
    int arr[] = { 1, 1, 2, 1, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << distribution(arr, n) << endl;
    return 0;
}
```

Improved By : [ankita_saini](#)

Source

<https://www.geeksforgeeks.org/equally-divide-into-two-sets-such-that-one-set-has-maximum-distinct-elements/>

Chapter 87

External Sorting

External Sorting - GeeksforGeeks

External sorting is a term for a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually RAM) and instead they must reside in the slower external memory (usually a hard drive). External sorting typically uses a hybrid sort-merge strategy. In the sorting phase, chunks of data small enough to fit in main memory are read, sorted, and written out to a temporary file. In the merge phase, the sorted sub-files are combined into a single larger file.

One example of external sorting is the external merge sort algorithm, which sorts chunks that each fit in RAM, then merges the sorted chunks together. We first divide the file into **runs** such that the size of a run is small enough to fit into main memory. Then sort each run in main memory using merge sort sorting algorithm. Finally merge the resulting runs together into successively bigger runs, until the file is sorted.

Prerequisite for the algorithm/code:

[MergeSort](#) : Used for sort individual runs (a run is part of file that is small enough to fit in main memory)

[Merge K Sorted Arrays](#) : Used to merge sorted runs.

Below are the steps used in C++ implementation.

Inputs:

```
input_file  : Name of input file. input.txt
output_file : Name of output file, output.txt
run_size   : Size of a run (can fit in RAM)
num_ways   : Number of runs to be merged
```

Output:

- 1) Read `input_file` such that at most '`run_size`' elements are read at a time. Do following for the every run read in an array.
 - a) Sort the run using `MergeSort`.

- b) Store the sorted run in a temporary file, say 'i' for i'th run.
- 2) Merge the sorted files using the approach discussed here

Following is C++ implementation of above steps.

```
// C++ program to implement external sorting using
// merge sort
#include <bits/stdc++.h>
using namespace std;

struct MinHeapNode
{
    // The element to be stored
    int element;

    // index of the array from which the element is taken
    int i;
};

// Prototype of a utility function to swap two min heap nodes
void swap(MinHeapNode* x, MinHeapNode* y);

// A class for Min Heap
class MinHeap
{
    MinHeapNode* harr; // pointer to array of elements in heap
    int heap_size;     // size of min heap

public:
    // Constructor: creates a min heap of given size
    MinHeap(MinHeapNode a[], int size);

    // to heapify a subtree with root at given index
    void MinHeapify(int);

    // to get index of left child of node at index i
    int left(int i) { return (2 * i + 1); }

    // to get index of right child of node at index i
    int right(int i) { return (2 * i + 2); }

    // to get the root
    MinHeapNode getMin() { return harr[0]; }

    // to replace root with new node x and heapify()
    // new root
    void replaceMin(MinHeapNode x)
```

```

{
    harr[0] = x;
    MinHeapify(0);
}
};

// Constructor: Builds a heap from a given array a[]
// of given size
MinHeap::MinHeap(MinHeapNode a[], int size)
{
    heap_size = size;
    harr = a; // store address of array
    int i = (heap_size - 1) / 2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// A recursive method to heapify a subtree with root
// at given index. This method assumes that the
// subtrees are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && harr[l].element < harr[i].element)
        smallest = l;
    if (r < heap_size && harr[r].element < harr[smallest].element)
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}

// A utility function to swap two elements
void swap(MinHeapNode* x, MinHeapNode* y)
{
    MinHeapNode temp = *x;
    *x = *y;
    *y = temp;
}

// Merges two subarrays of arr[].

```

```

// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for(i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for(j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
            arr[k++] = L[i++];
        else
            arr[k++] = R[j++];
    }

    /* Copy the remaining elements of L[], if there
       are any */
    while (i < n1)
        arr[k++] = L[i++];

    /* Copy the remaining elements of R[], if there
       are any */
    while(j < n2)
        arr[k++] = R[j++];
}

/* l is for left index and r is right index of the
   sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h

```

```

        int m = l + (r - 1) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

FILE* openFile(char* fileName, char* mode)
{
    FILE* fp = fopen(fileName, mode);
    if (fp == NULL)
    {
        perror("Error while opening the file.\n");
        exit(EXIT_FAILURE);
    }
    return fp;
}

// Merges k sorted files. Names of files are assumed
// to be 1, 2, 3, ... k
void mergeFiles(char *output_file, int n, int k)
{
    FILE* in[k];
    for (int i = 0; i < k; i++)
    {
        char fileName[2];

        // convert i to string
        sprintf(fileName, sizeof(fileName), "%d", i);

        // Open output files in read mode.
        in[i] = openFile(fileName, "r");
    }

    // FINAL OUTPUT FILE
    FILE *out = openFile(output_file, "w");

    // Create a min heap with k heap nodes. Every heap node
    // has first element of scratch output file
    MinHeapNode* harr = new MinHeapNode[k];
    int i;
    for (i = 0; i < k; i++)
    {
        // break if no output file is empty and
        // index i will be no. of input files

```

```

        if (fscanf(in[i], "%d ", &harr[i].element) != 1)
            break;

        harr[i].i = i; // Index of scratch output file
    }
    MinHeap hp(harr, i); // Create the heap

    int count = 0;

    // Now one by one get the minimum element from min
    // heap and replace it with next element.
    // run till all filled input files reach EOF
    while (count != i)
    {
        // Get the minimum element and store it in output file
        MinHeapNode root = hp.getMin();
        fprintf(out, "%d ", root.element);

        // Find the next element that will replace current
        // root of heap. The next element belongs to same
        // input file as the current min element.
        if (fscanf(in[root.i], "%d ", &root.element) != 1 )
        {
            root.element = INT_MAX;
            count++;
        }

        // Replace root with next element of input file
        hp.replaceMin(root);
    }

    // close input and output files
    for (int i = 0; i < k; i++)
        fclose(in[i]);

    fclose(out);
}

// Using a merge-sort algorithm, create the initial runs
// and divide them evenly among the output files
void createInitialRuns(char *input_file, int run_size,
                      int num_ways)
{
    // For big input file
    FILE *in = openFile(input_file, "r");

    // output scratch files
    FILE* out[num_ways];

```

```

char fileName[2];
for (int i = 0; i < num_ways; i++)
{
    // convert i to string
    sprintf(fileName, sizeof(fileName), "%d", i);

    // Open output files in write mode.
    out[i] = openFile(fileName, "w");
}

// allocate a dynamic array large enough
// to accommodate runs of size run_size
int* arr = (int*)malloc(run_size * sizeof(int));

bool more_input = true;
int next_output_file = 0;

int i;
while (more_input)
{
    // write run_size elements into arr from input file
    for (i = 0; i < run_size; i++)
    {
        if (fscanf(in, "%d ", &arr[i]) != 1)
        {
            more_input = false;
            break;
        }
    }

    // sort array using merge sort
    mergeSort(arr, 0, i - 1);

    // write the records to the appropriate scratch output file
    // can't assume that the loop runs to run_size
    // since the last run's length may be less than run_size
    for (int j = 0; j < i; j++)
        fprintf(out[next_output_file], "%d ", arr[j]);

    next_output_file++;
}

// close input and output files
for (int i = 0; i < num_ways; i++)
    fclose(out[i]);

fclose(in);
}

```

```
// For sorting data stored on disk
void externalSort(char* input_file, char *output_file,
                  int num_ways, int run_size)
{
    // read the input file, create the initial runs,
    // and assign the runs to the scratch output files
    createInitialRuns(input_file, run_size, num_ways);

    // Merge the runs using the K-way merging
    mergeFiles(output_file, run_size, num_ways);
}

// Driver program to test above
int main()
{
    // No. of Partitions of input file.
    int num_ways = 10;

    // The size of each partition
    int run_size = 1000;

    char input_file[] = "input.txt";
    char output_file[] = "output.txt";

    FILE* in = openFile(input_file, "w");

    srand(time(NULL));

    // generate input
    for (int i = 0; i < num_ways * run_size; i++)
        fprintf(in, "%d ", rand());

    fclose(in);

    externalSort(input_file, output_file, num_ways,
                 run_size);

    return 0;
}
```

This code won't work on online compiler as it requires file creation permissions. When run local machine, it produces sample input file “input.txt” with 10000 random numbers. It sorts the numbers and puts the sorted numbers in a file “output.txt”. It also generates files with names 1, 2, .. to store sorted runs.

References :

https://en.wikipedia.org/wiki/External_sorting

<http://web.eecs.utk.edu/~leparkr/Courses/CS302-Fall06/Notes/external-sorting2.html>

This article is contributed by **Aditya Goel**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/external-sorting/>

Chapter 88

Find Sum of all unique sub-array sum for a given array.

Find Sum of all unique sub-array sum for a given array. - GeeksforGeeks

Given an array of n-positive elements. Sub-array sum is defined as the sum of all elements of a particular sub-array, the task is to find the sum of all unique sub-array sum.

Note: Unique Sub-array sum means no other sub-array will have the same sum value.

Examples:

Input : arr[] = {3, 4, 5}

Output : 36

Explanation: All possible unique sub-array with their sum are as:

(3), (4), (5), (3+4), (4+5), (3+4+5). Here all are unique so required sum = 36

Input : arr[] = {2, 4, 2}

Output : 12

Explanation: All possible unique sub-array with their sum are as:

(2), (4), (2), (2+4), (4+2), (2+4+2). Here only (4) and (2+4+2) are unique.

Method 1 (Sorting Based)

- 1- Calculate cumulative sum of array.
- 2- Store all sub-array sum in vector.
- 3- Sort the vector.
- 4- Mark all duplicate sub-array sum to zero
- 5- Calculate and return totalSum.

```
// CPP for finding sum of all unique subarray sum
#include <bits/stdc++.h>
using namespace std;

// function for finding grandSum
```

```
long long int findSubarraySum(int arr[], int n)
{
    int i, j;

    // calculate cumulative sum of array
    // cArray[0] will store sum of zero elements
    long long int cArray[n + 1] = { 0 };
    for (i = 0; i < n; i++)
        cArray[i + 1] = cArray[i] + arr[i];

    vector<long long int> subArrSum;

    // store all subarray sum in vector
    for (i = 1; i <= n; i++)
        for (j = i; j <= n; j++)
            subArrSum.push_back(cArray[j] - cArray[i - 1]);

    // sort the vector
    sort(subArrSum.begin(), subArrSum.end());

    // mark all duplicate sub-array sum to zero
    long long totalSum = 0;
    for (i = 0; i < subArrSum.size() - 1; i++) {
        if (subArrSum[i] == subArrSum[i + 1]) {
            j = i + 1;
            while (subArrSum[j] == subArrSum[i] && j < subArrSum.size()) {
                subArrSum[j] = 0;
                j++;
            }
            subArrSum[i] = 0;
        }
    }

    // calculate total sum
    for (i = 0; i < subArrSum.size(); i++)
        totalSum += subArrSum[i];

    // return totalSum
    return totalSum;
}

// Drivers code
int main()
{
    int arr[] = { 3, 2, 3, 1, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findSubarraySum(arr, n);
    return 0;
}
```

}

Output:

41

Method 2 (Hashing Based) The idea is make an empty hash table. We generate all subarrays. For every subarray, we compute its sum and increment count of sum in hash table. Finally we add all those sums whose count is 1.

```
// CPP for finding sum of all unique subarray sum
#include <bits/stdc++.h>
using namespace std;

// function for finding grandSum
long long int findSubarraySum(int arr[], int n)
{
    int res = 0;

    // Go through all subarrays, compute sums
    // and count occurrences of sums.
    unordered_map<int, int> m;
    for (int i = 0; i < n; i++) {
        int sum = 0;
        for (int j = i; j < n; j++) {
            sum += arr[j];
            m[sum]++;
        }
    }

    // Print all those sums that appear
    // once.
    for (auto x : m)
        if (x.second == 1)
            res += x.first;

    return res;
}

// Driver code
int main()
{
    int arr[] = { 3, 2, 3, 1, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findSubarraySum(arr, n);
    return 0;
}
```

Output:

41

Source

<https://www.geeksforgeeks.org/find-sum-unique-sub-array-sum-given-array/>

Chapter 89

Find Surpasser Count of each element in array

Find Surpasser Count of each element in array - GeeksforGeeks

A surpasser of an element of an array is a greater element to its right, therefore $x[j]$ is a surpasser of $x[i]$ if $i < j$ and $x[i] < x[j]$. The surpasser count of an element is the number of surpassers. Given an array of distinct integers, for each element of the array find its surpasser count i.e. count the number of elements to the right that are greater than that element.

Examples :

Input: [2, 7, 5, 3, 0, 8, 1]
Output: [4, 1, 1, 1, 2, 0, 0]

Method 1 (Naive)

The naive solution would be to run two loops. For each element of the array, we count all elements greater than it to its right. The complexity of this solution is $O(n^2)$

C++

```
// Naive C++ program to find surpasser count of
// each element in array
#include <bits/stdc++.h>
using namespace std;

// Function to find surpasser count of each element
// in array
void findSurpasser(int arr[], int n)
{
```

```
for (int i = 0; i < n; i++)
{
    // stores surpasser count for element arr[i]
    int count = 0;
    for (int j = i + 1; j < n; j++)
        if (arr[j] > arr[i])
            count++;

    cout << count << " ";
}
}

/* Function to print an array */
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

/* Driver program to test above functions */
int main()
{
    int arr[] = { 2, 7, 5, 3, 0, 8, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, n);

    printf("Surpasser Count of array is \n");
    findSurpasser(arr, n);

    return 0;
}
```

Java

```
// Naive Java program to find surpasser count
// of each element in array
import java.io.*;

class GFG {

    // Function to find surpasser count of
    // each element in array
    static void findSurpasser(int arr[], int n)
    {
        for (int i = 0; i < n; i++)
```

```
{

    // stores surpasser count for
    // element arr[i]
    int count = 0;
    for (int j = i + 1; j < n; j++)
        if (arr[j] > arr[i])
            count++;

    System.out.print(count + " ");
}
}

/* Function to print an array */
static void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        System.out.print( arr[i] + " ");

    System.out.println();
}

// Driver program to test above functions
public static void main (String[] args)
{
    int arr[] = { 2, 7, 5, 3, 0, 8, 1 };
    int n = arr.length;

    System.out.println("Given array is ");
    printArray(arr, n);

    System.out.println("Surpasser Count of"
                       + " array is ");
    findSurpasser(arr, n);
}
}

// This code is contributed by Anuj_67.
```

Python3

```
# Naive Python3 program to find
# surpasser count of each element in array

# Function to find surpasser count of
# each element in array
def findSurpasser(arr, n):
```

```
for i in range(0, n):

    # stores surpasser count for element
    # arr[i]
    count = 0;

    for j in range (i + 1, n):
        if (arr[j] > arr[i]):
            count += 1

    print(count, end = " ")

# Function to print an array
def printArray(arr, n):

    for i in range(0, n):
        print(arr[i], end = " ")

# Driver program to test above functions
arr = [2, 7, 5, 3, 0, 8, 1]
n = len(arr)

print("Given array is")
printArray(arr , n)

print("\nSurpasser Count of array is");
findSurpasser(arr , n)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// Naive C# program to find surpasser count
// of each element in array
using System;

class GFG {

    // Function to find surpasser count of
    // each element in array
    static void findSurpasser(int []arr, int n)
    {
        for (int i = 0; i < n; i++)
        {

            // stores surpasser count for
            // element arr[i]
```

```
int count = 0;
for (int j = i + 1; j < n; j++)
    if (arr[j] > arr[i])
        count++;

Console.WriteLine(count + " ");
}

}

/* Function to print an array */
static void printArray(int []arr, int n)
{
    for (int i = 0; i < n; i++)
        Console.WriteLine( arr[i] + " ");

    Console.WriteLine();
}

// Driver program to test above functions
public static void Main ()
{
    int []arr = { 2, 7, 5, 3, 0, 8, 1 };
    int n = arr.Length;

    Console.WriteLine("Given array is ");
    printArray(arr, n);

    Console.WriteLine("Surpasser Count of"
                    + " array is ");
    findSurpasser(arr, n);
}
}

// This code is contributed by Anuj_67.
```

PHP

```
<?php
// Naive PHP program to find
// surpasser count of each
// element in array

// Function to find surpasser
// count of each element in array
function findSurpasser($arr, $n)
{
    for ( $i = 0; $i < $n; $i++)
    {
```

```
// stores surpasser count
// for element arr[i]
$count = 0;
for ( $j = $i + 1; $j < $n; $j++)
    if ($arr[$j] > $arr[$i])
        $count++;

echo $count , " ";
}

/*
Function to print an array */
function printArray( $arr, $n)
{
    for ( $i = 0; $i < $n; $i++)
        echo $arr[$i], " ";
        echo "\n";
}

// Driver Code
$arr = array( 2, 7, 5, 3, 0, 8, 1 );
$n = count($arr);

echo "Given array is \n";
printArray($arr, $n);

echo "Surpasser Count of array is \n";
findSurpasser($arr, $n);

// This code is contributed by Anuj_67.
?>
```

Output :

```
Given array is
2 7 5 3 0 8 1
Surpasser Count of array is
4 1 1 1 2 0 0
```

Time Complexity : $O(n^2)$

Method 2 (Uses Merge Sort)

For any element of the array, we can easily find out number of elements to the right that are greater than that element if we know number of elements to its right that are less than that element. The idea is to count the number of inversions for each element of the array using merge sort. So, surpasser count of an element at position i will be equal to " $n - i - \text{inversion-count}$ " at that position where n is the size of the array.

We have already discussed how to find inversion count of complete array [here](#). We have modified the discussed approach to find number of inversions for each element of the array instead of returning inversion count of whole array. Also, as all elements of the array are distinct, we maintain a map that stores inversion count for each element of the array.

Below is C++ implementation of above idea –

C++

```
// C++ program to find surpasser count of each element
// in array
#include <bits/stdc++.h>
using namespace std;

/* Function to merge the two halves arr[l..m] and
   arr[m+1..r] of array arr[] */
int merge(int arr[], int l, int m, int r,
          unordered_map<int, int> &hm)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];

    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0, j = 0, k = l;
    int c = 0;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            // increment inversion count of L[i]
            hm[L[i]] += c;
            arr[k++] = L[i++];
        }
        else
        {
            arr[k++] = R[j++];
            // inversion found
            c++;
        }
    }
}
```

```
        }

    }

/* Copy the remaining elements of L[], if
there are any */
while (i < n1)
{
    hm[L[i]] += c;
    arr[k++] = L[i++];
}

/* Copy the remaining elements of R[], if
there are any */
while (j < n2)
    arr[k++] = R[j++];
}

/* l is for left index and r is right index of
the sub-array of arr to be sorted */
int mergeSort(int arr[], int l, int r,
              unordered_map<int, int> &hm)
{
    if (l < r)
    {
        int m = l + (r - 1) / 2;
        mergeSort(arr, l, m, hm);
        mergeSort(arr, m + 1, r, hm);
        merge(arr, l, m, r, hm);
    }
}

/* Function to print an array */
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

void findSurpasser(int arr[], int n)
{
    // To store inversion count for elements
    unordered_map<int, int> hm;

    // To store copy of array
    int dup[n];
    memcpy(dup, arr, n*sizeof(arr[0]));
}
```

```
// Sort the copy and store inversion count
// for each element.
mergeSort(dup, 0, n - 1, hm);

printf("Surpasser Count of array is \n");
for (int i = 0; i < n; i++)
    printf("%d ", (n - 1) - i - hm[arr[i]]);

/* Driver program to test above functions */
int main()
{
    int arr[] = { 2, 7, 5, 3, 0, 8, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, n);

    findSurpasser(arr, n);

    return 0;
}
```

Output :

```
Given array is
2 7 5 3 0 8 1
Surpasser Count of array is
4 1 1 1 2 0 0
```

Time complexity of above solution is O(nlogn).

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-surpasser-count-of-each-element-in-array/>

Chapter 90

Find a pair of elements swapping which makes sum of two arrays same

Find a pair of elements swapping which makes sum of two arrays same - GeeksforGeeks

Given two arrays of integers, find a pair of values (one value from each array) that you can swap to give the two arrays the same sum.

Examples:

```
Input : A[] = {4, 1, 2, 1, 1, 2}
        B[] = {3, 6, 3, 3)
Output : {1, 3}
Sum of elements in A[] = 11
Sum of elements in B[] = 15
To get same sum from both arrays, we
can swap following values:
1 from A[] and 3 from B[]

Input: A[] = {5, 7, 4, 6}
       B[] = {1, 2, 3, 8}
Output: 6 2
```

Method 1 (Naive Implementation)

Iterate through the arrays and check all pairs of values. Compare new sums or look for a pair with that difference.

C/C++

```
// CPP code naive solution to find a pair swapping
```

```
// which makes sum of arrays sum.
#include <iostream>
using namespace std;

// Function to calculate sum of elements of array
int getSum(int X[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += X[i];
    return sum;
}

void findSwapValues(int A[], int n, int B[], int m)
{
    // Calculation of sums from both arrays
    int sum1 = getSum(A, n);
    int sum2 = getSum(B, m);

    // Look for val1 and val2, such that
    // sumA - val1 + val2 = sumB - val2 + val1
    int newsum1, newsum2, val1, val2;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            newsum1 = sum1 - A[i] + B[j];
            newsum2 = sum2 - B[j] + A[i];
            if (newsum1 == newsum2) {
                val1 = A[i];
                val2 = B[j];
            }
        }
    }

    cout << val1 << " " << val2;
}

// Driver code
int main()
{
    int A[] = { 4, 1, 2, 1, 1, 2 };
    int n = sizeof(A) / sizeof(A[0]);
    int B[] = { 3, 6, 3, 3 };
    int m = sizeof(B) / sizeof(B[0]);

    // Call to function
    findSwapValues(A, n, B, m);
    return 0;
}
```

Java

```
// Java program to find a pair swapping
// which makes sum of arrays sum
import java.io.*;

class GFG
{
    // Function to calculate sum of elements of array
    static int getSum(int X[], int n)
    {
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum += X[i];
        return sum;
    }

    // Function to prints elements to be swapped
    static void findSwapValues(int A[], int n, int B[], int m)
    {
        // Calculation of sums from both arrays
        int sum1 = getSum(A, n);
        int sum2 = getSum(B, m);

        // Look for val1 and val2, such that
        // sumA - val1 + val2 = sumB - val2 + val1
        int newsum1, newsum2, val1 = 0, val2 = 0;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                newsum1 = sum1 - A[i] + B[j];
                newsum2 = sum2 - B[j] + A[i];
                if (newsum1 == newsum2)
                {
                    val1 = A[i];
                    val2 = B[j];
                }
            }
        }

        System.out.println(val1+" "+val2);
    }

    // driver program
    public static void main (String[] args)
    {
        int A[] = { 4, 1, 2, 1, 1, 2 };
    }
}
```

```
int n = A.length;
int B[] = { 3, 6, 3, 3 };
int m = B.length;

// Call to function
findSwapValues(A, n, B, m);
}

}

// Contributed by Pramod Kumar
```

Python

```
# Python code naive solution to find a pair swapping
# which makes sum of lists sum.

# Function to calculate sum of elements of list
def getSum(X):
    sum=0
    for i in X:
        sum+=i
    return sum

# Function to prints elements to be swapped
def findSwapValues(A,B):
    # Calculation if sums from both lists
    sum1=getSum(A)
    sum2=getSum(B)

    # Boolean variable used to reduce further iterations
    # after the pair is found
    k=False

    # Lool for val1 and val2, such that
    # sumA - val1 + val2 = sumB -val2 + val1
    val1,val2=0,0
    for i in A:
        for j in B:
            newsum1=sum1-i+j
            newsum2=sum2-j+i

            if newsum1 ==newsum2:
                val1=i
                val2=j
                # Set to True when pair is found
                k=True
                break
    # If k is True, it means pair is found.
```

```
# So, no further iterations.  
if k==True:  
    break  
print val1,val2  
return  
  
# Driver code  
A=[4,1,2,1,1,2]  
B=[3,6,3,3]  
  
# Call to function  
findSwapValues(A,B)  
  
# code contributed by sachin bisht
```

Output :

1 3

Time Complexity :- O(n*m)

Method 2 -> Other Naive implementation

We are looking for two values, a and b, such that:

$$\begin{aligned} \text{sumA} - a + b &= \text{sumB} - b + a \\ 2a - 2b &= \text{sumA} - \text{sumB} \\ a - b &= (\text{sumA} - \text{sumB}) / 2 \end{aligned}$$

Therefore, we're looking for two values that have a specific target difference: $(\text{sumA} - \text{sumB}) / 2$.

C/C++

```
// CPP code for naive implementation  
#include <iostream>  
using namespace std;  
  
// Function to calculate sum of elements of array  
int getSum(int X[], int n)  
{  
    int sum = 0;  
    for (int i = 0; i < n; i++)  
        sum += X[i];  
    return sum;
```

```
}

// Function to calculate : a - b = (sumA - sumB) / 2
int getTarget(int A[], int n, int B[], int m)
{
    // Calculation of sums from both arrays
    int sum1 = getSum(A, n);
    int sum2 = getSum(B, m);

    // because that the target must be an integer
    if ((sum1 - sum2) % 2 != 0)
        return 0;
    return ((sum1 - sum2) / 2);
}

void findSwapValues(int A[], int n, int B[], int m)
{
    int target = getTarget(A, n, B, m);
    if (target == 0)
        return;

    // Look for val1 and val2, such that
    // val1 - val2 = (sumA - sumB) / 2
    int val1, val2;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (A[i] - B[j] == target) {
                val1 = A[i];
                val2 = B[j];
            }
        }
    }

    cout << val1 << " " << val2;
}

// Driver code
int main()
{
    int A[] = { 4, 1, 2, 1, 1, 2 };
    int n = sizeof(A) / sizeof(A[0]);
    int B[] = { 3, 6, 3, 3 };
    int m = sizeof(B) / sizeof(B[0]);

    // Call to function
    findSwapValues(A, n, B, m);
    return 0;
}
```

Java

```
// Java program to find a pair swapping
// which makes sum of arrays sum
import java.io.*;

class GFG
{
    // Function to calculate sum of elements of array
    static int getSum(int X[], int n)
    {
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum += X[i];
        return sum;
    }

    // Function to calculate : a - b = (sumA - sumB) / 2
    static int getTarget(int A[], int n, int B[], int m)
    {
        // Calculation of sums from both arrays
        int sum1 = getSum(A, n);
        int sum2 = getSum(B, m);

        // because that the target must be an integer
        if ((sum1 - sum2) % 2 != 0)
            return 0;
        return ((sum1 - sum2) / 2);
    }

    // Function to prints elements to be swapped
    static void findSwapValues(int A[], int n, int B[], int m)
    {
        int target = getTarget(A, n, B, m);
        if (target == 0)
            return;

        // Look for val1 and val2, such that
        // val1 - val2 = (sumA - sumB) / 2
        int val1 = 0, val2 = 0;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                if (A[i] - B[j] == target)
                {
                    val1 = A[i];
                    val2 = B[j];
                }
            }
        }
    }
}
```

```
        }
    }
}
System.out.println(val1+" "+val2);
}

// driver program
public static void main (String[] args)
{
    int A[] = { 4, 1, 2, 1, 1, 2 };
    int n = A.length;
    int B[] = { 3, 6, 3, 3 };
    int m = B.length;

    // Call to function
    findSwapValues(A, n, B, m);
}
}

// Contributed by Pramod Kumar
```

Python

```
# Python Code for naive implementation

# Function to calculate sum of elements of list
def getSum(X):
    sum=0
    for i in X:
        sum+=i
    return sum

# Function to calculate : a-b = (sumA - sumB) / 2
def getTarget(A,B):

    #Calculations of sums from both lists
    sum1=getSum(A)
    sum2=getSum(B)

    # Because the target must be an integer
    if( (sum1-sum2)%2!=0):
        return 0
    return (sum1-sum2)/2

def findSwapValues(A,B):
    target=getTarget(A,B)
    if target==0:
```

```
return

# Boolean variable used to reduce further iterations
# after the pair is found
flag=False

# Look for val1 and val2, such that
# val1 - val2 = (sumA -sumB) /2
val1,val2=0,0
for i in A:
    for j in B:

        if i-j == target:
            val1=i
            val2=j
            # Set to True when pair is found
            flag=True
            break
        if flag==True:
            break
    print val1,val2
return

# Driver code
A=[4,1,2,1,1,2]
B=[3,6,3,3]

# Call to function
findSwapValues(A,B)

# code contributed by sachin bisht
```

Output:

1 3

Time Complexity :- $O(n*m)$

Method 3 -> Optimized Solution :-

- 1) Sort the arrays.
- 2) Traverse both array simultaneously and do following for every pair.
 - a) If difference is too small then, make it bigger by moving 'a' to a

- bigger value.
- b) If it is too big then, make it smaller by moving b to a bigger value.
 - c) If it's just right, return this pair.

C/C++

```
// CPP code for optimized implementation
#include <bits/stdc++.h>
using namespace std;

// Returns sum of elements in X[]
int getSum(int X[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += X[i];
    return sum;
}

// Finds value of
// a - b = (sumA - sumB) / 2
int getTarget(int A[], int n, int B[], int m)
{
    // Calculation of sums from both arrays
    int sum1 = getSum(A, n);
    int sum2 = getSum(B, m);

    // because that the target must be an integer
    if ((sum1 - sum2) % 2 != 0)
        return 0;
    return ((sum1 - sum2) / 2);
}

// Prints elements to be swapped
void findSwapValues(int A[], int n, int B[], int m)
{
    // Call for sorting the arrays
    sort(A, A + n);
    sort(B, B + m);

    // Note that target can be negative
    int target = getTarget(A, n, B, m);

    // target 0 means, answer is not possible
    if (target == 0)
        return;
}
```

```
int i = 0, j = 0;
while (i < n && j < m) {
    int diff = A[i] - B[j];
    if (diff == target) {
        cout << A[i] << " " << B[j];
        return;
    }

    // Look for a greater value in A[]
    else if (diff < target)
        i++;
    else
        j++;
}
}

// Driver code
int main()
{
    int A[] = { 4, 1, 2, 1, 1, 2 };
    int n = sizeof(A) / sizeof(A[0]);

    int B[] = { 1, 6, 3, 3 };
    int m = sizeof(B) / sizeof(B[0]);

    findSwapValues(A, n, B, m);
    return 0;
}
```

Java

```
// Java code for optimized implementation
import java.io.*;
import java.util.*;

class GFG
{
    // Function to calculate sum of elements of array
    static int getSum(int X[], int n)
    {
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum += X[i];
        return sum;
    }
}
```

```
// Function to calculate : a - b = (sumA - sumB) / 2
static int getTarget(int A[], int n, int B[], int m)
{
    // Calculation of sums from both arrays
    int sum1 = getSum(A, n);
    int sum2 = getSum(B, m);

    // because that the target must be an integer
    if ((sum1 - sum2) % 2 != 0)
        return 0;
    return ((sum1 - sum2) / 2);
}

// Function to prints elements to be swapped
static void findSwapValues(int A[], int n, int B[], int m)
{
    // Call for sorting the arrays
    Arrays.sort(A);
    Arrays.sort(B);

    // Note that target can be negative
    int target = getTarget(A, n, B, m);

    // target 0 means, answer is not possible
    if (target == 0)
        return;

    int i = 0, j = 0;
    while (i < n && j < m)
    {
        int diff = A[i] - B[j];
        if (diff == target)
        {
            System.out.println(A[i]+" "+B[i]);
            return;
        }

        // Look for a greater value in A[]
        else if (diff < target)
            i++;

        // Look for a greater value in B[]
        else
            j++;
    }
}

// driver program
```

```
public static void main (String[] args)
{
    int A[] = { 4, 1, 2, 1, 1, 2 };
    int n = A.length;
    int B[] = { 3, 6, 3, 3 };
    int m = B.length;

    // Call to function
    findSwapValues(A, n, B, m);
}
}

// Contributed by Pramod Kumar
```

Python

```
# Python code for optimized implementation

# Returns sum of elements in list
def getSum(X):
    sum=0
    for i in X:
        sum+=i
    return sum

# Finds value of
# a - b = (sumA - sumB) / 2
def getTarget(A,B):
    # Calculations of sumd from both lists
    sum1=getSum(A)
    sum2=getSum(B)

    # Because that target must be an integer
    if( (sum1-sum2)%2!=0):
        return 0
    return (sum1-sum2)/2

# Prints elements to be swapped
def findSwapValues(A,B):
    # Call for sorting the lists
    A.sort()
    B.sort()

    # Note that target can be negative
    target=getTarget(A,B)

    # target 0 means, answer is not possible
    if(target==0):
```

```
        return
i,j=0,0
while(i<len(A) and j<len(B)):
    diff=A[i]-B[j]
    if diff == target:
        print A[i],B[j]
        return
    # Look for a greater value in list A
    elif diff <target:
        i+=1
    # Look for a greater value in list B
    else:
        j+=1

A=[4,1,2,1,1,2]
B=[3,6,3,3]

findSwapValues(A,B)

#code contibuted by sachin bisht
```

Output:

2 3

Time Complexity :-

If arrays are sorted : $O(n + m)$

If arrays aren't sorted : $O(n\log(n) + m\log(m))$

Method 4 (Hashing)

We can solve this problem in $O(m+n)$ time and $O(m)$ auxiliary space. Below are algorithmic steps.

```
// assume array1 is small i.e. (m < n)
// where m is array1.length and n is array2.length
1. Find sum1(sum of small array elements) ans sum2
   (sum of larger array elements). // time O(m+n)
2. Make a hashset for small array(here array1).
3. Calculate diff as (sum1-sum2)/2.
4. Run a loop for array2
   for (int i equal to 0 to n-1)
       if (hashset contains (array2[i]+diff))
           print array2[i]+diff and array2[i]
           set flag and break;
```

5. If flag is unset then there is no such kind of pair.

Thanks to nicky khan for suggesting method 4.

Source

<https://www.geeksforgeeks.org/find-a-pair-swapping-which-makes-sum-of-two-arrays-same/>

Chapter 91

Find a pair with maximum product in array of Integers

Find a pair with maximum product in array of Integers - GeeksforGeeks

Given an array with both +ive and -ive integers, return a pair with highest product.

Examples :

Input: arr[] = {1, 4, 3, 6, 7, 0}
Output: {6,7}

Input: arr[] = {-1, -3, -4, 2, 0, -5}
Output: {-4,-5}

A **Simple Solution** is to consider every pair and keep track maximum product. Below is the implementation of this simple solution.

C++

```
// A simple C++ program to find max product pair in
// an array of integers
#include<bits/stdc++.h>
using namespace std;

// Function to find maximum product pair in arr[0..n-1]
void maxProduct(int arr[], int n)
{
    if (n < 2)
    {
        cout << "No pairs exists\n";
        return;
    }
```

```
}

// Initialize max product pair
int a = arr[0], b = arr[1];

// Traverse through every possible pair
// and keep track of max product
for (int i=0; i<n; i++)
    for (int j=i+1; j<n; j++)
        if (arr[i]*arr[j] > a*b)
            a = arr[i], b = arr[j];

cout << "Max product pair is {" << a << ", "
       << b << "}";

}

// Driver program to test
int main()
{
    int arr[] = {1, 4, 3, 6, 7, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    maxProduct(arr, n);
    return 0;
}
```

Java

```
// JAVA Code to Find a pair with maximum
// product in array of Integers
import java.util.*;

class GFG {

    // Function to find maximum product pair
    // in arr[0..n-1]
    static void maxProduct(int arr[], int n)
    {
        if (n < 2)
        {
            System.out.println("No pairs exists");
            return;
        }

        // Initialize max product pair
        int a = arr[0], b = arr[1];

        // Traverse through every possible pair
        // and keep track of max product
```

```
for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
        if (arr[i] * arr[j] > a * b){
            a = arr[i];
            b = arr[j];
        }

    System.out.println("Max product pair is {" +
                        a + ", " + b + "}");
}

/* Driver program to test above function */
public static void main(String[] args)
{
    int arr[] = {1, 4, 3, 6, 7, 0};
    int n = arr.length;
    maxProduct(arr, n);

}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
# A simple Python3 program to find max
# product pair in an array of integers

# Function to find maximum
# product pair in arr[0..n-1]
def maxProduct(arr, n):

    if (n < 2):
        print("No pairs exists")
        return

    # Initialize max product pair
    a = arr[0]; b = arr[1]

    # Traverse through every possible pair
    # and keep track of max product
    for i in range(0, n):

        for j in range(i + 1, n):
            if (arr[i] * arr[j] > a * b):
                a = arr[i]; b = arr[j]

    print("Max product pair is {" , a, ", " , b, "}" ,
```

```
        sep = "")  
  
# Driver Code  
arr = [1, 4, 3, 6, 7, 0]  
n = len(arr)  
maxProduct(arr, n)  
  
# This code is contributed by Smitha Dinesh Semwal.  
  
C#  
  
// C# Code to Find a pair with maximum  
// product in array of Integers  
using System;  
  
class GFG  
{  
  
    // Function to find maximum  
    // product pair in arr[0..n-1]  
    static void maxProduct(int []arr, int n)  
    {  
        if (n < 2)  
        {  
            Console.WriteLine("No pairs exists");  
            return;  
        }  
  
        // Initialize max product pair  
        int a = arr[0], b = arr[1];  
  
        // Traverse through every possible pair  
        // and keep track of max product  
        for (int i = 0; i < n; i++)  
            for (int j = i + 1; j < n; j++)  
                if (arr[i] * arr[j] > a * b)  
                {  
                    a = arr[i];  
                    b = arr[j];  
                }  
  
        Console.WriteLine("Max product pair is {" +  
                         a + ", " + b + "});  
    }  
  
    // Driver Code  
    public static void Main()  
    {
```

```
        int []arr = {1, 4, 3, 6, 7, 0};
        int n = arr.Length;
        maxProduct(arr, n);
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// A simple PHP program to
// find max product pair in
// an array of integers

// Function to find maximum
// product pair in arr[0..n-1]
function maxProduct( $arr, $n)
{
    if ($n < 2)
    {
        echo "No pairs exists\n";
        return;
    }

    // Initialize max product pair
    $a = $arr[0];
    $b = $arr[1];

    // Traverse through every possible pair
    // and keep track of max product
    for ($i = 0; $i < $n; $i++)
        for ($j = $i + 1; $j < $n; $j++)
    {
        if ($arr[$i] * $arr[$j] > $a * $b)
        {
            $a = $arr[$i];
            $b = $arr[$j];
        }
    }

    echo "Max product pair is {" , $a , ", ";
    echo $b , "}";
}

// Driver Code
$arr = array(1, 4, 3, 6, 7, 0);
$n = count($arr);
```

```
maxProduct($arr, $n);  
  
// This code is contributed by anuj_67.  
?>
```

Output :

```
Max product pair is {6, 7}
```

Time Complexity : $O(n^2)$

A **Better Solution** is to use sorting. Below are detailed steps.

- 1) Sort input array in increasing order.
- 2) If all elements are positive, then return product of last two numbers.
- 3) Else return maximum of products of first two and last two numbers.

Time complexity of this solution is $O(n \log n)$. Thanks to Rahul Jain for suggesting this method.

An **Efficient Solution** can solve the above problem in single traversal of input array. The idea is to traverse the input array and keep track of following four values.

- a) Maximum positive value
- b) Second maximum positive value
- c) Maximum negative value i.e., a negative value with maximum absolute value
- d) Second maximum negative value.

At the end of the loop, compare the products of first two and last two and print the maximum of two products. Below is the implementation of this idea.

C++

```
// A O(n) C++ program to find maximum product pair in an array  
#include<bits/stdc++.h>  
using namespace std;  
  
// Function to find maximum product pair in arr[0..n-1]  
void maxProduct(int arr[], int n)  
{  
    if (n < 2)  
    {  
        cout << "No pairs exists\n";  
        return;  
    }  
  
    if (n == 2)  
    {  
        cout << arr[0] << " " << arr[1] << endl;  
        return;  
    }  
}
```

```
// Initialize maximum and second maximum
int posa = INT_MIN, posb = INT_MIN;

// Initialize minimum and second minimum
int nega = INT_MIN, negb = INT_MIN;

// Traverse given array
for (int i = 0; i < n; i++)
{
    // Update maximum and second maximum if needed
    if (arr[i] > posa)
    {
        posb = posa;
        posa = arr[i];
    }
    else if (arr[i] > posb)
        posb = arr[i];

    // Update minimum and second minimum if needed
    if (arr[i] < 0 && abs(arr[i]) > abs(nega))
    {
        negb = nega;
        nega = arr[i];
    }
    else if (arr[i] < 0 && abs(arr[i]) > abs(negb))
        negb = arr[i];
}

if (nega*negb > posa*posb)
    cout << "Max product pair is {" << nega << ", "
        << negb << "}";
else
    cout << "Max product pair is {" << posa << ", "
        << posb << "}";
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 4, 3, 6, 7, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    maxProduct(arr, n);
    return 0;
}
```

Java

```
// JAVA Code to Find a pair with maximum
// product in array of Integers
import java.util.*;

class GFG {

    // Function to find maximum product pair
    // in arr[0..n-1]
    static void maxProduct(int arr[], int n)
    {
        if (n < 2)
        {
            System.out.println("No pairs exists");
            return;
        }

        if (n == 2)
        {
            System.out.println(arr[0] + " " + arr[1]);
            return;
        }

        // Initialize maximum and second maximum
        int posa = Integer.MIN_VALUE,
            posb = Integer.MIN_VALUE;

        // Initialize minimum and second minimum
        int nega = Integer.MIN_VALUE,
            negb = Integer.MIN_VALUE;

        // Traverse given array
        for (int i = 0; i < n; i++)
        {
            // Update maximum and second maximum
            // if needed
            if (arr[i] > posa)
            {
                posb = posa;
                posa = arr[i];
            }
            else if (arr[i] > posb)
                posb = arr[i];

            // Update minimum and second minimum
            // if needed
            if (arr[i] < 0 && Math.abs(arr[i]) >
                Math.abs(nega))
            {

```

```
        negb = nega;
        nega = arr[i];
    }
    else if(arr[i] < 0 && Math.abs(arr[i])
            > Math.abs(negb))
        negb = arr[i];
}

if (nega * negb > posa * posb)
    System.out.println("Max product pair is {"
                        + nega + ", " + negb + "}");
else
    System.out.println("Max product pair is {"
                        + posa + ", " + posb + "}");
}

/* Driver program to test above function */
public static void main(String[] args)
{
    int arr[] = {1, 4, 3, 6, 7, 0};
    int n = arr.length;
    maxProduct(arr, n);

}
}

// This code is contributed by Arnav Kr. Mandal.
```

C#

```
// C# Code to Find a pair with maximum
// product in array of Integers
using System;
class GFG {

    // Function to find maximum
    // product pair in arr[0..n-1]
    static void maxProduct(int []arr, int n)
    {
        if (n < 2)
        {
            Console.WriteLine("No pairs exists");
            return;
        }

        if (n == 2)
        {
            Console.WriteLine(arr[0] + " " + arr[1]);
        }
    }
}
```

```
        return;
    }

    // Initialize maximum and
    // second maximum
    int posa = int.MinValue;
    int posb = int.MaxValue;

    // Initialize minimum and
    // second minimum
    int nega = int.MinValue;
    int negb = int.MaxValue;

    // Traverse given array
    for (int i = 0; i < n; i++)
    {

        // Update maximum and
        // second maximum
        // if needed
        if (arr[i] > posa)
        {
            posb = posa;
            posa = arr[i];
        }
        else if (arr[i] > posb)
            posb = arr[i];

        // Update minimum and
        // second minimum if
        // needed
        if (arr[i] < 0 && Math.Abs(arr[i]) >
            Math.Abs(nega))
        {
            negb = nega;
            nega = arr[i];
        }
        else if (arr[i] < 0 &&
            Math.Abs(arr[i]) >
            Math.Abs(negb))
            negb = arr[i];
    }

    if (nega * negb > posa * posb)
        Console.WriteLine("Max product pair is {"
            + nega + ", " + negb + "}");
    else
        Console.WriteLine("Max product pair is {"
```

```
        + posa + ", " + posb + "}");
    }

    // Driver Code
    public static void Main()
    {
        int []arr = {1, 4, 3, 6, 7, 0};
        int n = arr.Length;
        maxProduct(arr, n);
    }
}

// This code is contributed by anuj_67.
```

Output:

Max product pair is {7, 6}

Time complexity: O(n)

Auxiliary Space: O(1)

Thanks to Gaurav Ahirwar for suggesting this method.

Improved By : [nitin mittal, vt_m](#)

Source

<https://www.geeksforgeeks.org/return-a-pair-with-maximum-product-in-array-of-integers/>

Chapter 92

Find a pair with the given difference

Find a pair with the given difference - GeeksforGeeks

Given an unsorted array and a number n, find if there exists a pair of elements in the array whose difference is n.

Examples:

Input: arr[] = {5, 20, 3, 2, 50, 80}, n = 78

Output: Pair Found: (2, 80)

Input: arr[] = {90, 70, 20, 80, 50}, n = 45

Output: No Such Pair

The simplest method is to run two loops, the outer loop picks the first element (smaller element) and the inner loop looks for the element picked by outer loop plus n. Time complexity of this method is $O(n^2)$.

We can use sorting and Binary Search to improve time complexity to $O(n \log n)$. The first step is to sort the array in ascending order. Once the array is sorted, traverse the array from left to right, and for each element arr[i], binary search for arr[i] + n in arr[i+1..n-1]. If the element is found, return the pair.

Both first and second steps take $O(n \log n)$. So overall complexity is $O(n \log n)$.

The second step of the above algorithm can be improved to $O(n)$. The first step remain same. The idea for second step is take two index variables i and j, initialize them as 0 and 1 respectively. Now run a linear loop. If arr[j] – arr[i] is smaller than n, we need to look for greater arr[j], so increment j. If arr[j] – arr[i] is greater than n, we need to look for greater arr[i], so increment i. Thanks to Aashish Barnwal for suggesting this approach.

The following code is only for the second step of the algorithm, it assumes that the array is already sorted.

C/C++

```
// C/C++ program to find a pair with the given difference
#include <stdio.h>

// The function assumes that the array is sorted
bool findPair(int arr[], int size, int n)
{
    // Initialize positions of two elements
    int i = 0;
    int j = 1;

    // Search for a pair
    while (i<size && j<size)
    {
        if (i != j && arr[j]-arr[i] == n)
        {
            printf("Pair Found: (%d, %d)", arr[i], arr[j]);
            return true;
        }
        else if (arr[j]-arr[i] < n)
            j++;
        else
            i++;
    }

    printf("No such pair");
    return false;
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 8, 30, 40, 100};
    int size = sizeof(arr)/sizeof(arr[0]);
    int n = 60;
    findPair(arr, size, n);
    return 0;
}
```

Java

```
// Java program to find a pair with the given difference
import java.io.*;

class PairDifference
{
    // The function assumes that the array is sorted
    static boolean findPair(int arr[], int n)
    {
```

```
int size = arr.length;

// Initialize positions of two elements
int i = 0, j = 1;

// Search for a pair
while (i < size && j < size)
{
    if (i != j && arr[j]-arr[i] == n)
    {
        System.out.print("Pair Found: "+
                          "( "+arr[i]+", "+ arr[j]+" )");
        return true;
    }
    else if (arr[j] - arr[i] < n)
        j++;
    else
        i++;
}

System.out.print("No such pair");
return false;
}

// Driver program to test above function
public static void main (String[] args)
{
    int arr[] = {1, 8, 30, 40, 100};
    int n = 60;
    findPair(arr,n);
}
}
/*This code is contributed by Devesh Agrawal*/
```

Python

```
# Python program to find a pair with the given difference

# The function assumes that the array is sorted
def findPair(arr,n):

    size = len(arr)

    # Initialize positions of two elements
    i,j = 0,1

    # Search for a pair
    while i < size and j < size:
```

```
if i != j and arr[j]-arr[i] == n:  
    print "Pair found (",arr[i],",",arr[j],")"  
    return True  
  
elif arr[j] - arr[i] < n:  
    j+=1  
else:  
    i+=1  
print "No pair found"  
return False  
  
# Driver function to test above function  
arr = [1, 8, 30, 40, 100]  
n = 60  
findPair(arr, n)  
  
# This code is contributed by Devesh Agrawal
```

C#

```
// C# program to find a pair with the given difference  
using System;  
  
class GFG {  
  
    // The function assumes that the array is sorted  
    static bool findPair(int []arr, int n)  
    {  
        int size = arr.Length;  
  
        // Initialize positions of two elements  
        int i = 0, j = 1;  
  
        // Search for a pair  
        while (i < size && j < size)  
        {  
            if (i != j && arr[j] - arr[i] == n)  
            {  
                Console.Write("Pair Found: "  
                + "( " + arr[i] + ", " + arr[j] +" )");  
  
                return true;  
            }  
            else if (arr[j] - arr[i] < n)  
                j++;  
            else  
                i++;  
        }  
    }  
}
```

```
}

Console.WriteLine("No such pair");

return false;
}

// Driver program to test above function
public static void Main ()
{
    int []arr = {1, 8, 30, 40, 100};
    int n = 60;

    findPair(arr, n);
}
}

// This code is contributed by Sam007.
```

Output:

Pair Found: (40, 100)

Hashing can also be used to solve this problem. Create an empty hash table HT. Traverse the array, use array elements as hash keys and enter them in HT. Traverse the array again look for value $n + arr[i]$ in HT.

Source

<https://www.geeksforgeeks.org/find-a-pair-with-the-given-difference/>

Chapter 93

Find a permutation that causes worst case of Merge Sort

Find a permutation that causes worst case of Merge Sort - GeeksforGeeks

Given a set of elements, find which permutation of these elements would result in worst case of Merge Sort?

Asymptotically, merge sort always takes $\Theta(n \log n)$ time, but the cases that require more comparisons generally take more time in practice. We basically need to find a permutation of input elements that would lead to maximum number of comparisons when sorted using a typical Merge Sort algorithm.

Example:

Consider the below set of elements
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16}

Below permutation of the set causes 153
comparisons.
{1, 9, 5, 13, 3, 11, 7, 15, 2, 10, 6,
14, 4, 12, 8, 16}

And an already sorted permutation causes
30 comparisons.

See this for a program that counts
comparisons and shows above results.

Now how to get worst case input for merge sort for an input set?

Lets us try to build the array in bottom up manner

Let the sorted array be {1,2,3,4,5,6,7,8}.

In order to generate the worst case of merge sort, the merge operation that resulted in above sorted array should result in maximum comparisons. In order to do so, the left and right subarray involved in merge operation should store alternate elements of sorted array. i.e. left sub-array should be {1,3,5,7} and right sub-array should be {2,4,6,8}. Now every element of array will be compared at-least once and that will result in maximum comparisons. We apply the same logic for left and right sub-array as well. For array {1,3,5,7}, the worst case will be when its left and right sub-array are {1,5} and {3,7} respectively and for array {2,4,6,8} the worst case will occur for {2,4} and {6,8}.

Complete Algorithm –

GenerateWorstCase(arr[])

1. 1. Create two auxillary arrays left and right and store alternate array elements in them.
2. Call GenerateWorstCase for left subarray: GenerateWorstCase (left)
3. Call GenerateWorstCase for right subarray: GenerateWorstCase (right)
4. Copy all elements of left and right subarrays back to original array.

Below is implementation of the idea

C/C++

```
// C/C++ program to generate Worst Case of Merge Sort
#include <stdlib.h>
#include <stdio.h>

// Function to print an array
void printArray(int A[], int size)
{
    for (int i = 0; i < size; i++)
        printf("%d ", A[i]);

    printf("\n");
}

// Function to join left and right subarray
int join(int arr[], int left[], int right[],
         int l, int m, int r)
{
    int i; // Used in second loop
    for (i = 0; i <= m - l; i++)
        arr[i] = left[i];

    for (int j = 0; j < r - m; j++)
        arr[i + j] = right[j];
}
```

```
// Function to store alternate elements in left
// and right subarray
int split(int arr[], int left[], int right[],
          int l, int m, int r)
{
    for (int i = 0; i <= m - 1; i++)
        left[i] = arr[i * 2];

    for (int i = 0; i < r - m; i++)
        right[i] = arr[i * 2 + 1];
}

// Function to generate Worst Case of Merge Sort
int generateWorstCase(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;

        // create two auxillary arrays
        int left[m - l + 1];
        int right[r - m];

        // Store alternate array elements in left
        // and right subarray
        split(arr, left, right, l, m, r);

        // Recurse first and second halves
        generateWorstCase(left, l, m);
        generateWorstCase(right, m + 1, r);

        // join left and right subarray
        join(arr, left, right, l, m, r);
    }
}

// Driver code
int main()
{
    // Sorted array
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9,
                  10, 11, 12, 13, 14, 15, 16 };
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Sorted array is \n");
    printArray(arr, n);
```

```
// generate Worst Case of Merge Sort
generateWorstCase(arr, 0, n - 1);

printf("\nInput array that will result in "
      "worst case of merge sort is \n");
printArray(arr, n);

return 0;
}
```

Java

```
// Java program to generate Worst Case of Merge Sort

import java.util.Arrays;

class GFG
{
    // Function to join left and right subarray
    static void join(int arr[], int left[], int right[],
                     int l, int m, int r)
    {
        int i;
        for (i = 0; i <= m - 1; i++)
            arr[i] = left[i];

        for (int j = 0; j < r - m; j++)
            arr[i + j] = right[j];
    }

    // Function to store alternate elements in left
    // and right subarray
    static void split(int arr[], int left[], int right[],
                     int l, int m, int r)
    {
        for (int i = 0; i <= m - 1; i++)
            left[i] = arr[i * 2];

        for (int i = 0; i < r - m; i++)
            right[i] = arr[i * 2 + 1];
    }

    // Function to generate Worst Case of Merge Sort
    static void generateWorstCase(int arr[], int l, int r)
    {
        if (l < r)
        {
            int m = l + (r - l) / 2;
```

```
// create two auxillary arrays
int[] left = new int[m - l + 1];
int[] right = new int[r - m];

// Store alternate array elements in left
// and right subarray
split(arr, left, right, l, m, r);

// Recurse first and second halves
generateWorstCase(left, l, m);
generateWorstCase(right, m + 1, r);

// join left and right subarray
join(arr, left, right, l, m, r);
}

}

// driver program
public static void main (String[] args)
{
    // sorted array
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9,
                  10, 11, 12, 13, 14, 15, 16 };
    int n = arr.length;
    System.out.println("Sorted array is");
    System.out.println(Arrays.toString(arr));

    // generate Worst Case of Merge Sort
    generateWorstCase(arr, 0, n - 1);

    System.out.println("\nInput array that will result in \n"+
                      "worst case of merge sort is \n");
    System.out.println(Arrays.toString(arr));
}

}

// Contributed by Pramod Kumar

C#
// C# program to generate Worst Case of
// Merge Sort
using System;

class GFG {
```

```
// Function to join left and right subarray
static void join(int []arr, int []left,
                 int []right, int l, int m, int r)
{
    int i;
    for (i = 0; i <= m - l; i++)
        arr[i] = left[i];

    for (int j = 0; j < r - m; j++)
        arr[i + j] = right[j];
}

// Function to store alternate elements in
// left and right subarray
static void split(int []arr, int []left,
                  int []right, int l, int m, int r)
{
    for (int i = 0; i <= m - l; i++)
        left[i] = arr[i * 2];

    for (int i = 0; i < r - m; i++)
        right[i] = arr[i * 2 + 1];
}

// Function to generate Worst Case of
// Merge Sort
static void generateWorstCase(int []arr,
                               int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;

        // create two auxillary arrays
        int[] left = new int[m - l + 1];
        int[] right = new int[r - m];

        // Store alternate array elements
        // in left and right subarray
        split(arr, left, right, l, m, r);

        // Recurse first and second halves
        generateWorstCase(left, l, m);
        generateWorstCase(right, m + 1, r);

        // join left and right subarray
        join(arr, left, right, l, m, r);
    }
}
```

```
}

// driver program
public static void Main ()
{

    // sorted array
    int []arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9,
                  10, 11, 12, 13, 14, 15, 16 };

    int n = arr.Length;
    Console.WriteLine("Sorted array is\n");

    for(int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");

    // generate Worst Case of Merge Sort
    generateWorstCase(arr, 0, n - 1);

    Console.WriteLine("\nInput array that will "
                     + "result in \n worst case of"
                     + " merge sort is \n");

    for(int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
}

// This code is contributed by Smitha
```

Output:

```
Sorted array is
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Input array that will result in worst
case of merge sort is
1 9 5 13 3 11 7 15 2 10 6 14 4 12 8 16
```

References – [Stack Overflow](#)

This article is contributed by **Aditya Goel**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [Smitha Dinesh Semwal](#)

Source

<https://www.geeksforgeeks.org/find-a-permutation-that-causes-worst-case-of-merge-sort/>

Chapter 94

Find a triplet that sum to a given value

Find a triplet that sum to a given value - GeeksforGeeks

Given an array and a value, find if there is a triplet in array whose sum is equal to the given value. If there is such a triplet present in array, then print the triplet and return true. Else return false. For example, if the given array is {12, 3, 4, 1, 6, 9} and given sum is 24, then there is a triplet (12, 3 and 9) present in array whose sum is 24.

Method 1 (Naive)

A simple method is to generate all possible triplets and compare the sum of every triplet with the given value. The following code implements this simple method using three nested loops.

C

```
#include <stdio.h>

// returns true if there is triplet with sum equal
// to 'sum' present in A[]. Also, prints the triplet
bool find3Numbers(int A[], int arr_size, int sum)
{
    int l, r;

    // Fix the first element as A[i]
    for (int i = 0; i < arr_size - 2; i++) {

        // Fix the second element as A[j]
        for (int j = i + 1; j < arr_size - 1; j++) {

            // Now look for the third number
            for (int k = j + 1; k < arr_size; k++) {
                if (A[i] + A[j] + A[k] == sum) {
```

```
        printf("Triplet is %d, %d, %d",
               A[i], A[j], A[k]);
        return true;
    }
}
}

// If we reach here, then no triplet was found
return false;
}

/* Driver program to test above function */
int main()
{
    int A[] = { 1, 4, 45, 6, 10, 8 };
    int sum = 22;
    int arr_size = sizeof(A) / sizeof(A[0]);
    find3Numbers(A, arr_size, sum);
    return 0;
}
```

Java

```
class FindTriplet {

    // returns true if there is triplet with sum equal
    // to 'sum' present in A[]. Also, prints the triplet
    boolean find3Numbers(int A[], int arr_size, int sum)
    {
        int l, r;

        // Fix the first element as A[i]
        for (int i = 0; i < arr_size - 2; i++) {

            // Fix the second element as A[j]
            for (int j = i + 1; j < arr_size - 1; j++) {

                // Now look for the third number
                for (int k = j + 1; k < arr_size; k++) {
                    if (A[i] + A[j] + A[k] == sum) {
                        System.out.print("Triplet is " + A[i] +
                                         ", " + A[j] + ", " + A[k]);
                        return true;
                    }
                }
            }
        }
    }
}
```

```
// If we reach here, then no triplet was found
return false;
}

// Driver program to test above functions
public static void main(String[] args)
{
    FindTriplet triplet = new FindTriplet();
    int A[] = { 1, 4, 45, 6, 10, 8 };
    int sum = 22;
    int arr_size = A.length;

    triplet.find3Numbers(A, arr_size, sum);
}
}
```

Python3

```
# Python3 program to find a triplet
# that sum to a given value

# returns true if there is triplet with
# sum equal to 'sum' present in A[].
# Also, prints the triplet
def find3Numbers(A, arr_size, sum):

    # Fix the first element as A[i]
    for i in range( 0, arr_size-2):

        # Fix the second element as A[j]
        for j in range(i + 1, arr_size-1):

            # Now look for the third number
            for k in range(j + 1, arr_size):
                if A[i] + A[j] + A[k] == sum:
                    print("Triplet is", A[i],
                          ", ", A[j], ", ", A[k])
                    return True

    # If we reach here, then no
    # triplet was found
    return False

# Driver program to test above function
A = [1, 4, 45, 6, 10, 8]
sum = 22
arr_size = len(A)
```

```
find3Numbers(A, arr_size, sum)

# This code is contributed by Smitha Dinesh Semwal

C#

// C# program to find a triplet
// that sum to a given value
using System;

class GFG
{
    // returns true if there is
    // triplet with sum equal
    // to 'sum' present in A[].
    // Also, prints the triplet
    static bool find3Numbers(int []A,
                            int arr_size,
                            int sum)
    {
        // Fix the first
        // element as A[i]
        for (int i = 0;
             i < arr_size - 2; i++)
        {

            // Fix the second
            // element as A[j]
            for (int j = i + 1;
                 j < arr_size - 1; j++)
            {

                // Now look for
                // the third number
                for (int k = j + 1;
                     k < arr_size; k++)
                {
                    if (A[i] + A[j] + A[k] == sum)
                    {
                        Console.WriteLine("Triplet is " + A[i] +
                                         ", " + A[j] +
                                         ", " + A[k]);
                        return true;
                    }
                }
            }
        }
    }
}
```

```
// If we reach here,  
// then no triplet was found  
return false;  
}  
  
// Driver Code  
static public void Main ()  
{  
    int []A = {1, 4, 45, 6, 10, 8};  
    int sum = 22;  
    int arr_size = A.Length;  
  
    find3Numbers(A, arr_size, sum);  
}  
}  
  
// This code is contributed by m_kit
```

PHP

```
<?php  
// PHP program to find a triplet  
// that sum to a given value  
  
// returns true if there is  
// triplet with sum equal to  
// 'sum' present in A[].  
// Also, prints the triplet  
function find3Numbers($A, $arr_size, $sum)  
{  
    $l; $r;  
  
    // Fix the first  
    // element as A[i]  
    for ($i = 0;  
         $i < $arr_size - 2; $i++)  
    {  
        // Fix the second  
        // element as A[j]  
        for ($j = $i + 1;  
             $j < $arr_size - 1; $j++)  
        {  
            // Now look for the  
            // third number  
            for ($k = $j + 1;  
                 $k < $arr_size; $k++)  
            {  
                if ($A[$i] + $A[$j] +
```

```
$A[$k] == $sum)
{
    echo "Triplet is", " ", $A[$i],
          ", ", $A[$j],
          ", ", $A[$k];
    return true;
}
}
}

// If we reach here, then
// no triplet was found
return false;
}
// Driver Code
$A = array(1, 4, 45,
           6, 10, 8);
$sum = 22;
$arr_size = sizeof($A);

find3Numbers($A, $arr_size, $sum);

// This code is contributed by ajit
?>
```

Output :

```
Triplet is 4, 10, 8
```

Time Complexity : $O(n^3)$

Method 2 (Use Sorting)

Time complexity of the method 1 is $O(n^3)$. The complexity can be reduced to $O(n^2)$ by sorting the array first, and then using method 1 of [this](#) post in a loop.

- 1) Sort the input array.
- 2) Fix the first element as $A[i]$ where i is from 0 to array size - 2. After fixing the first element of triplet, find the other two elements using method 1 of [this](#) post.

C++

```
// C++ program to find a triplet
#include <bits/stdc++.h>
using namespace std;

// returns true if there is triplet with sum equal
// to 'sum' present in A[]. Also, prints the triplet
bool find3Numbers(int A[], int arr_size, int sum)
```

```
{  
    int l, r;  
  
    /* Sort the elements */  
    sort(A, A + arr_size);  
  
    /* Now fix the first element one by one and find the  
       other two elements */  
    for (int i = 0; i < arr_size - 2; i++) {  
  
        // To find the other two elements, start two index  
        // variables from two corners of the array and move  
        // them toward each other  
        l = i + 1; // index of the first element in the  
        // remaining elements  
  
        r = arr_size - 1; // index of the last element  
        while (l < r) {  
            if (A[i] + A[l] + A[r] == sum) {  
                printf("Triplet is %d, %d, %d", A[i],  
                       A[l], A[r]);  
                return true;  
            }  
            else if (A[i] + A[l] + A[r] < sum)  
                l++;  
            else // A[i] + A[l] + A[r] > sum  
                r--;  
        }  
    }  
  
    // If we reach here, then no triplet was found  
    return false;  
}  
  
/* Driver program to test above function */  
int main()  
{  
    int A[] = { 1, 4, 45, 6, 10, 8 };  
    int sum = 22;  
    int arr_size = sizeof(A) / sizeof(A[0]);  
  
    find3Numbers(A, arr_size, sum);  
  
    return 0;  
}
```

Java

```
class FindTriplet {

    // returns true if there is triplet with sum equal
    // to 'sum' present in A[]. Also, prints the triplet
    boolean find3Numbers(int A[], int arr_size, int sum)
    {
        int l, r;

        /* Sort the elements */
        quickSort(A, 0, arr_size - 1);

        /* Now fix the first element one by one and find the
           other two elements */
        for (int i = 0; i < arr_size - 2; i++) {

            // To find the other two elements, start two index variables
            // from two corners of the array and move them toward each
            // other
            l = i + 1; // index of the first element in the remaining elements
            r = arr_size - 1; // index of the last element
            while (l < r) {
                if (A[i] + A[l] + A[r] == sum) {
                    System.out.print("Triplet is " + A[i] +
                                     ", " + A[l] + ", " + A[r]);
                    return true;
                }
                else if (A[i] + A[l] + A[r] < sum)
                    l++;
                else // A[i] + A[l] + A[r] > sum
                    r--;
            }
        }

        // If we reach here, then no triplet was found
        return false;
    }

    int partition(int A[], int si, int ei)
    {
        int x = A[ei];
        int i = (si - 1);
        int j;

        for (j = si; j <= ei - 1; j++) {
            if (A[j] <= x) {
                i++;
                int temp = A[i];
```

```
        A[i] = A[j];
        A[j] = temp;
    }
}
int temp = A[i + 1];
A[i + 1] = A[ei];
A[ei] = temp;
return (i + 1);
}

/* Implementation of Quick Sort
A[] --> Array to be sorted
si --> Starting index
ei --> Ending index
*/
void quickSort(int A[], int si, int ei)
{
    int pi;

    /* Partitioning index */
    if (si < ei) {
        pi = partition(A, si, ei);
        quickSort(A, si, pi - 1);
        quickSort(A, pi + 1, ei);
    }
}

// Driver program to test above functions
public static void main(String[] args)
{
    FindTriplet triplet = new FindTriplet();
    int A[] = { 1, 4, 45, 6, 10, 8 };
    int sum = 22;
    int arr_size = A.length;

    triplet.find3Numbers(A, arr_size, sum);
}
```

Python3

```
# Python3 program to find a triplet

# returns true if there is triplet
# with sum equal to 'sum' present
# in A[]. Also, prints the triplet
def find3Numbers(A, arr_size, sum):
```

```
# Sort the elements
A.sort()

# Now fix the first element
# one by one and find the
# other two elements
for i in range(0, arr_size-2):

    # To find the other two elements,
    # start two index variables from
    # two corners of the array and
    # move them toward each other

    # index of the first element
    # in the remaining elements
    l = i + 1

    # index of the last element
    r = arr_size-1
    while (l < r):

        if( A[i] + A[l] + A[r] == sum):
            print("Triplet is", A[i],
                  ', ', A[l], ', ', A[r]);
            return True

        elif (A[i] + A[l] + A[r] < sum):
            l += 1
        else: # A[i] + A[l] + A[r] > sum
            r -= 1

    # If we reach here, then
    # no triplet was found
    return False

# Driver program to test above function
A = [1, 4, 45, 6, 10, 8]
sum = 22
arr_size = len(A)

find3Numbers(A, arr_size, sum)

# This is contributed by Smitha Dinesh Semwal

C#
// C# program to find a triplet
```

```
using System;

class GFG
{

    // returns true if there is triplet
    // with sum equal to 'sum' present
    // in A[]. Also, prints the triplet
    bool find3Numbers(int[] A, int arr_size,
                      int sum)
    {
        int l, r;

        /* Sort the elements */
        quickSort(A, 0, arr_size - 1);

        /* Now fix the first element
        one by one and find the
        other two elements */
        for (int i = 0; i < arr_size - 2; i++)
        {

            // To find the other two elements,
            // start two index variables from
            // two corners of the array and
            // move them toward each other
            l = i + 1; // index of the first element
                        // in the remaining elements
            r = arr_size - 1; // index of the last element
            while (l < r)
            {
                if (A[i] + A[l] + A[r] == sum)
                {
                    Console.Write("Triplet is " + A[i] +
                                ", " + A[l] + ", " + A[r]);
                    return true;
                }
                else if (A[i] + A[l] + A[r] < sum)
                    l++;
                else // A[i] + A[l] + A[r] > sum
                    r--;
            }
        }

        // If we reach here, then
        // no triplet was found
        return false;
    }
}
```

```
}  
  
int partition(int[] A, int si, int ei)  
{  
    int x = A[ei];  
    int i = (si - 1);  
    int j;  
  
    for (j = si; j <= ei - 1; j++)  
    {  
        if (A[j] <= x)  
        {  
            i++;  
            int temp = A[i];  
            A[i] = A[j];  
            A[j] = temp;  
        }  
    }  
    int temp1 = A[i + 1];  
    A[i + 1] = A[ei];  
    A[ei] = temp1;  
    return (i + 1);  
}  
  
/* Implementation of Quick Sort  
A[] --> Array to be sorted  
si --> Starting index  
ei --> Ending index  
*/  
void quickSort(int[] A, int si, int ei)  
{  
    int pi;  
  
    /* Partitioning index */  
    if (si < ei)  
    {  
        pi = partition(A, si, ei);  
        quickSort(A, si, pi - 1);  
        quickSort(A, pi + 1, ei);  
    }  
}  
  
// Driver Code  
static void Main()  
{  
    GFG triplet = new GFG();  
    int[] A = new int[] { 1, 4, 45, 6, 10, 8 };  
    int sum = 22;
```

```
    int arr_size = A.Length;  
  
    triplet.find3Numbers(A, arr_size, sum);  
}  
}  
  
// This code is contributed by mits
```

PHP

```
<?php  
// PHP program to find a triplet  
  
// returns true if there is  
// triplet with sum equal to  
// 'sum' present in A[]. Also,  
// prints the triplet  
function find3Numbers($A, $arr_size, $sum)  
{  
    $l; $r;  
  
    /* Sort the elements */  
    sort($A);  
  
    /* Now fix the first element  
    one by one and find the  
    other two elements */  
    for ($i = 0; $i < $arr_size - 2; $i++)  
    {  
  
        // To find the other two elements,  
        // start two index variables from  
        // two corners of the array and  
        // move them toward each other  
        $l = $i + 1; // index of the first element  
                    // in the remaining elements  
  
        // index of the last element  
        $r = $arr_size - 1;  
        while ($l < $r)  
        {  
            if ($A[$i] + $A[$l] +  
                $A[$r] == $sum)  
            {  
                echo "Triplet is ", $A[$i], " ",  
                     $A[$l], " ",  
                     $A[$r], "\n";  
                return true;  
            }  
        }  
    }  
}
```

```
        }
        else if ($A[$i] + $A[$l] +
                 $A[$r] < $sum)
            $l++;
        else // A[i] + A[l] + A[r] > sum
            $r--;
    }
}

// If we reach here, then
// no triplet was found
return false;
}

// Driver Code
$A = array (1, 4, 45, 6, 10, 8);
$sum = 22;
$arr_size = sizeof($A);

find3Numbers($A, $arr_size, $sum);

// This code is contributed by ajit
?>
```

Output :

Triplet is 4, 8, 10

Time Complexity : $O(n^2)$

Method 3 (Hashing Based Solution)
C++

```
#include <bits/stdc++.h>
using namespace std;

// returns true if there is triplet with sum equal
// to 'sum' present in A[]. Also, prints the triplet
bool find3Numbers(int A[], int arr_size, int sum)
{
    // Fix the first element as A[i]
    for (int i = 0; i < arr_size - 2; i++) {

        // Find pair in subarray A[i+1..n-1]
        // with sum equal to sum - A[i]
        unordered_set<int> s;
        int curr_sum = sum - A[i];
```

```
for (int j = i + 1; j < arr_size; j++) {  
    if (s.find(curr_sum - A[j]) != s.end())  
        return true;  
    s.insert(A[j]);  
}  
}  
  
// If we reach here, then no triplet was found  
return false;  
}  
  
/* Driver program to test above function */  
int main()  
{  
    int A[] = { 1, 4, 45, 6, 10, 8 };  
    int sum = 22;  
    int arr_size = sizeof(A) / sizeof(A[0]);  
  
    if (find3Numbers(A, arr_size, sum))  
        cout << "Yes";  
    else  
        cout << "No";  
  
    return 0;  
}
```

How to print all triplets with given sum?

Please refer [Find all triplets with zero sum](#)

Improved By : [jit_t](#), Mithun Kumar

Source

<https://www.geeksforgeeks.org/find-a-triplet-that-sum-to-a-given-value/>

Chapter 95

Find all triplets with zero sum

Find all triplets with zero sum - GeeksforGeeks

Given an array of distinct elements. The task is to find triplets in array whose sum is zero.

Examples :

```
Input : arr[] = {0, -1, 2, -3, 1}
Output : 0 -1 1
         2 -3 1
```

```
Input : arr[] = {1, -2, 1, 0, 5}
Output : 1 -2 1
```

Method 1 (Simple : $O(n^3)$)

The naive approach is that run three loops and check one by one that sum of three elements is zero or not if sum of three elements is zero then print elements other wise print not found.

C++

```
// A simple C++ program to find three elements
// whose sum is equal to zero
#include<bits/stdc++.h>
using namespace std;

// Prints all triplets in arr[] with 0 sum
void findTriplets(int arr[], int n)
{
    bool found = true;
    for (int i=0; i<n-2; i++)
    {
        for (int j=i+1; j<n-1; j++)
        {
            for (int k=j+1; k<n; k++)
            {
                if (arr[i] + arr[j] + arr[k] == 0)
                    cout << arr[i] << " " << arr[j] << " " << arr[k] << endl;
            }
        }
    }
}
```

```
for (int k=j+1; k<n; k++)
{
    if (arr[i]+arr[j]+arr[k] == 0)
    {
        cout << arr[i] << " "
            << arr[j] << " "
            << arr[k] << endl;
        found = true;
    }
}
}

// If no triplet with 0 sum found in array
if (found == false)
    cout << " not exist " << endl;

}

// Driver code
int main()
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    findTriplets(arr, n);
    return 0;
}
```

Java

```
// A simple Java program to find three elements
// whose sum is equal to zero
class num{
// Prints all triplets in arr[] with 0 sum
static void findTriplets(int[] arr, int n)
{
    boolean found = true;
    for (int i=0; i<n-2; i++)
    {
        for (int j=i+1; j<n-1; j++)
        {
            for (int k=j+1; k<n; k++)
            {
                if (arr[i]+arr[j]+arr[k] == 0)
                {
                    System.out.print(arr[i]);
                    System.out.print(" ");
                    System.out.print(arr[j]);
                    System.out.print(" ");
                    System.out.print(arr[k]);
                    System.out.println();
                }
            }
        }
    }
}
```

```
        System.out.print(" ");
        System.out.print(arr[k]);
        System.out.print("\n");
        found = true;
    }
}
}

// If no triplet with 0 sum found in array
if (found == false)
    System.out.println(" not exist ");

}

// Driver code
public static void main(String[] args)
{
    int arr[] = {0, -1, 2, -3, 1};
    int n =arr.length;
    findTriplets(arr, n);

}
}

//This code is contributed by
//Smitha Dinesh Semwal
```

Python3

```
# A simple Python 3 program
# to find three elements whose
# sum is equal to zero

# Prints all triplets in
# arr[] with 0 sum
def findTriplets(arr, n):

    found = True
    for i in range(0, n-2):

        for j in range(i+1, n-1):

            for k in range(j+1, n):

                if (arr[i] + arr[j] + arr[k] == 0):
                    print(arr[i], arr[j], arr[k])
                    found = True
```

```
# If no triplet with 0 sum
# found in array
if (found == False):
    print(" not exist ")

# Driver code
arr = [0, -1, 2, -3, 1]
n = len(arr)
findTriplets(arr, n)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// A simple C# program to find three elements
// whose sum is equal to zero
using System;

class GFG {

    // Prints all triplets in arr[] with 0 sum
    static void findTriplets(int []arr, int n)
    {
        bool found = true;
        for (int i = 0; i < n-2; i++)
        {
            for (int j = i+1; j < n-1; j++)
            {
                for (int k = j+1; k < n; k++)
                {
                    if (arr[i] + arr[j] + arr[k]
                        == 0)
                    {
                        Console.Write(arr[i]);
                        Console.Write(" ");
                        Console.Write(arr[j]);
                        Console.Write(" ");
                        Console.Write(arr[k]);
                        Console.Write("\n");
                        found = true;
                    }
                }
            }
        }

        // If no triplet with 0 sum found in
        // array
    }
}
```

```
        if (found == false)
            Console.WriteLine(" not exist ");
    }

// Driver code
public static void Main()
{
    int []arr = {0, -1, 2, -3, 1};
    int n = arr.Length;
    findTriplets(arr, n);
}
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// A simple PHP program to
// find three elements whose
// sum is equal to zero

// Prints all triplets
// in arr[] with 0 sum
function findTriplets($arr, $n)
{
    $found = true;
    for ($i = 0; $i < $n - 2; $i++)
    {
        for ($j = $i + 1; $j < $n - 1; $j++)
        {
            for ($k = $j + 1; $k < $n; $k++)
            {
                if ($arr[$i] + $arr[$j] +
                    $arr[$k] == 0)
                {
                    echo $arr[$i] , " ",
                        $arr[$j] , " ",
                        $arr[$k] , "\n";
                    $found = true;
                }
            }
        }
    }
}

// If no triplet with 0
// sum found in array
if ($found == false)
```

```
echo " not exist ", "\n";
}

// Driver Code
$arr = array (0, -1, 2, -3, 1);
$n = sizeof($arr);
findTriplets($arr, $n);

// This code is contributed by m_kit
?>

0 -1 1
2 -3 1
```

Time Complexity : $O(n^3)$
Auxiliary Space : $O(1)$

Method 2 (Hashing : $O(n^2)$)

We iterate through every element. For every element $arr[i]$, we find a pair with sum “ $-arr[i]$ ”. This problem reduces to pairs sum and can be solved in $O(n)$ time using hashing.

```
Run a loop from i=0 to n-2
    Create an empty hash table
    Run inner loop from j=i+1 to n-1
        If -(arr[i] + arr[j]) is present in hash table
            print arr[i], arr[j] and -(arr[i]+arr[j])
        Else
            Insert arr[j] in hash table.
```

C++

```
// C++ program to find triplets in a given
// array whose sum is zero
#include<bits/stdc++.h>
using namespace std;

// function to print triplets with 0 sum
void findTriplets(int arr[], int n)
{
    bool found = false;

    for (int i=0; i<n-1; i++)
    {
        // Find all pairs with sum equals to
```

```

// "-arr[i]"
unordered_set<int> s;
for (int j=i+1; j<n; j++)
{
    int x = -(arr[i] + arr[j]);
    if (s.find(x) != s.end())
    {
        printf("%d %d %d\n", x, arr[i], arr[j]);
        found = true;
    }
    else
        s.insert(arr[j]);
}
}

if (found == false)
    cout << " No Triplet Found" << endl;
}

// Driver code
int main()
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    findTriplets(arr, n);
    return 0;
}

-1 0 1
-3 2 1

```

Time Complexity : $O(n^2)$

Auxiliary Space : $O(n)$

Method 3 (Sorting : $O(n^2)$)

The above method requires extra space. We can solve in $O(1)$ extra space. The idea is based on method 2 of [this](#) post.

1. Sort all element of array
2. Run loop from $i=0$ to $n-2$.
 - Initialize two index variables $l=i+1$ and $r=n-1$
4. while ($l < r$)
 - Check sum of $arr[i]$, $arr[l]$, $arr[r]$ is zero or not if sum is zero then print the triplet and do $l++$ and $r--$.

5. If sum is less than zero then l++
6. If sum is greater than zero then r--
7. If not exist in array then print not found.

C++

```
// C++ program to find triplets in a given
// array whose sum is zero
#include<bits/stdc++.h>
using namespace std;

// function to print triplets with 0 sum
void findTriplets(int arr[], int n)
{
    bool found = false;

    // sort array elements
    sort(arr, arr+n);

    for (int i=0; i<n-1; i++)
    {
        // initialize left and right
        int l = i + 1;
        int r = n - 1;
        int x = arr[i];
        while (l < r)
        {
            if (x + arr[l] + arr[r] == 0)
            {
                // print elements if it's sum is zero
                printf("%d %d %d\n", x, arr[l], arr[r]);
                l++;
                r--;
                found = true;
            }

            // If sum of three elements is less
            // than zero then increment in left
            else if (x + arr[l] + arr[r] < 0)
                l++;

            // if sum is greater than zero than
            // decrement in right side
            else
                r--;
        }
    }
}
```

```
    if (found == false)
        cout << " No Triplet Found" << endl;
}

// Driven source
int main()
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    findTriplets(arr, n);
    return 0;
}
```

Python3

```
# python program to find triplets in a given
# array whose sum is zero

# function to print triplets with 0 sum
def findTriplets(arr, n):

    found = False

    # sort array elements
    arr.sort()

    for i in range(0, n-1):

        # initialize left and right
        l = i + 1
        r = n - 1
        x = arr[i]
        while (l < r):

            if (x + arr[l] + arr[r] == 0):
                # print elements if it's sum is zero
                print(x, arr[l], arr[r])
                l+=1
                r-=1
                found = True

            # If sum of three elements is less
            # than zero then increment in left
            elif (x + arr[l] + arr[r] < 0):
                l+=1

            # if sum is greater than zero than
```

```
# decrement in right side
else:
    r-=1

if (found == False):
    print(" No Triplet Found")

# Driven source
arr = [0, -1, 2, -3, 1]
n = len(arr)
findTriplets(arr, n)

# This code is contributed by Smitha Dinesh Semwal
```

PHP

```
<?php
// PHP program to find
// triplets in a given
// array whose sum is zero

// function to print
// triplets with 0 sum
function findTriplets($arr, $n)
{
    $found = false;

    // sort array elements
    sort($arr);

    for ($i = 0; $i < $n - 1; $i++)
    {
        // initialize left
        // and right
        $l = $i + 1;
        $r = $n - 1;
        $x = $arr[$i];
        while ($l < $r)
        {
            if ($x + $arr[$l] +
                $arr[$r] == 0)
            {
                // print elements if
                // it's sum is zero
                echo $x, " ", $arr[$l],
                    " ", $arr[$r], "\n";
                $l++;
            }
        }
    }
}
```

```
$r--;
$found = true;
}

// If sum of three elements
// is less than zero then
// increment in left
else if ($x + $arr[$l] +
          $arr[$r] < 0)
    $l++;

// if sum is greater than
// zero then decrement
// in right side
else
    $r--;
}
}

if ($found == false)
    echo " No Triplet Found" ,"\n";
}

// Driver Code
$arr = array (0, -1, 2, -3, 1);
$n = sizeof($arr);
findTriplets($arr, $n);

// This code is contributed by ajit
?>
```

Output :

```
-3 1 2
-1 0 1
```

Time Complexity : $O(n^2)$
Auxiliary Space : $O(1)$

Improved By : [nitin mittal, jit_t](#)

Source

<https://www.geeksforgeeks.org/find-triplets-array-whose-sum-equal-zero/>

Chapter 96

Find all elements in array which have at-least two greater elements

Find all elements in array which have at-least two greater elements - GeeksforGeeks

Given an array of n distinct elements, the task is to find all elements in array which have at-least two greater elements than themselves.

Examples :

```
Input : arr[] = {2, 8, 7, 1, 5};  
Output : 2 1 5  
The output three elements have two or  
more greater elements
```

```
Input : arr[] = {7, -2, 3, 4, 9, -1};  
Output : -2 3 4 -1
```

Method 1 (Simple)

The naive approach is to run two loops and check one by one element of array check that array elements have at-least two elements greater than itself or not. If its true then print array element.

C++

```
// Simple C++ program to find  
// all elements in array which  
// have at-least two greater  
// elements itself.  
#include<bits/stdc++.h>
```

```
using namespace std;

void findElements(int arr[], int n)
{
    // Pick elements one by one and
    // count greater elements. If
    // count is more than 2, print
    // that element.
    for (int i = 0; i < n; i++)
    {
        int count = 0;
        for (int j = 0; j < n; j++)
            if (arr[j] > arr[i])
                count++;

        if (count >= 2)
            cout << arr[i] << " ";
    }
}

// Driver code
int main()
{
    int arr[] = { 2, -6, 3, 5, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    findElements(arr, n);
    return 0;
}
```

Java

```
// Java program to find all
// elements in array which
// have at-least two greater
// elements itself.
import java.util.*;
import java.io.*;

class GFG
{

static void findElements(int arr[],
                        int n)
{
    // Pick elements one by one
    // and count greater elements.
    // If count is more than 2,
    // print that element.
```

```
for (int i = 0; i < n; i++)
{
    int count = 0;

    for (int j = 0; j < n; j++)
        if (arr[j] > arr[i])
            count++;

    if (count >= 2)
        System.out.print(arr[i] + " ");
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 2, -6 ,3 , 5, 1};
    int n = arr.length;
    findElements(arr, n);
}
}

// This code is contributed by Sahil_Bansall
```

Python3

```
# Python3 program to find
# all elements in array
# which have at-least two
# greater elements itself.

def findElements( arr, n):

    # Pick elements one by
    # one and count greater
    # elements. If count
    # is more than 2, print
    # that element.

    for i in range(n):
        count = 0
        for j in range(0, n):
            if arr[j] > arr[i]:
                count = count + 1

        if count >= 2 :
```

```
print(arr[i], end=" ")

# Driver code
arr = [ 2, -6 ,3 , 5, 1]
n = len(arr)
findElements(arr, n)

# This code is contributed by sunnysingh

C#

// C# program to find all elements in
// array which have at least two greater
// elements itself.
using System;

class GFG
{

static void findElements(int []arr, int n)
{
    // Pick elements one by one and count
    // greater elements. If count is more
    // than 2, print that element.
    for (int i = 0; i < n; i++)
    {
        int count = 0;

        for (int j = 0; j < n; j++)
            if (arr[j] > arr[i])
                count++;

        if (count >= 2)
Console.Write(arr[i] + " ");
    }
}

// Driver code
public static void Main(String []args)
{
    int []arr = {2, -6 ,3 , 5, 1};
    int n = arr.Length;
    findElements(arr, n);

}
```

```
// This code is contributed by Parashar.
```

PHP

```
<?php
// Simple PHP program to find
// all elements in array which
// have at-least two greater
// elements itself.

function findElements($arr, $n)
{
    // Pick elements one by one and
    // count greater elements. If
    // count is more than 2,
    // print that element.
    for ($i = 0; $i < $n; $i++)
    {
        $count = 0;
        for ($j = 0; $j < $n; $j++)
            if ($arr[$j] > $arr[$i])
                $count++;

        if ($count >= 2)
            echo $arr[$i]." ";
    }
}

// Driver code
$arr = array( 2, -6 ,3 , 5, 1);
$n = sizeof($arr);
findElements($arr, $n);

?>
```

Output :

```
2 -6 1
```

Time Complexity : $O(n^2)$

Method 2 (Use Sorting)

We sort the array first in increasing order, then we print first $n-2$ elements where n is size of array.

C++

```
// Sorting based C++ program to
// find all elements in array
// which have atleast two greater
// elements itself.
#include<bits/stdc++.h>
using namespace std;

void findElements(int arr[], int n)
{
    sort(arr, arr + n);

    for (int i = 0; i < n - 2; i++)
        cout << arr[i] << " ";
}

// Driver Code
int main()
{
    int arr[] = { 2, -6 ,3 , 5, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    findElements(arr, n);
    return 0;
}
```

Java

```
// Sorting based Java program to find
// all elements in array which have
// atleast two greater elements itself.
import java.util.*;
import java.io.*;

class GFG
{

static void findElements(int arr[], int n)
{
    Arrays.sort(arr);

    for (int i = 0; i < n - 2; i++)
        System.out.print(arr[i] + " ");
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 2, -6 ,3 , 5, 1};
    int n = arr.length;
```

```
    findElements(arr, n);  
}  
}  
  
// This code is contributed by Sahil_Bansall
```

Python3

```
# Sorting based Python 3 program  
# to find all elements in array  
# which have atleast two greater  
# elements itself.  
  
def findElements(arr, n):  
  
    arr.sort()  
  
    for i in range(0, n-2):  
        print(arr[i], end = " ")  
  
# Driven source  
arr = [2, -6, 3, 5, 1]  
n = len(arr)  
findElements(arr, n)  
  
# This code is contributed  
# by Smitha Dinesh Semwal
```

Output :

-6 1 2

C#

```
// Sorting based C# program to find  
// all elements in array which have  
// atleast two greater elements itself.  
using System;  
  
class GFG  
{  
  
    static void findElements(int []arr, int n)  
    {  
        Array.Sort(arr);
```

```
for (int i = 0; i < n-2; i++)
    Console.Write(arr[i] + " ");
}

// Driver code
public static void Main(String []args)
{
    int []arr = { 2, -6 ,3 , 5, 1};
    int n = arr.Length;
    findElements(arr, n);

}
}

// This code is contributed by parashar
```

PHP

```
<?php
// Sorting based PHP program to
// find all elements in array
// which have atleast two greater
// elements itself.

function findElements( $arr, $n)
{
    sort($arr);

    for ($i = 0; $i < $n - 2; $i++)
        echo $arr[$i] , " ";
}

// Driver Code
$arr = array( 2, -6 ,3 , 5, 1);
$n = count($arr);
findElements($arr, $n);

// This code is contributed by anuj_67.
?>;
```

Output :

-6 1 2

Time Complexity : O(n Log n)

Method 3 (Efficient)

In second method we simply calculate second maximum element of array and print all element which is less than or equal to second maximum.

C++

```
// C++ program to find all elements
// in array which have atleast two
// greater elements itself.
#include<bits/stdc++.h>
using namespace std;

void findElements(int arr[], int n)
{
    int first = INT_MIN,
        second = INT_MIN;
    for (int i = 0; i < n; i++)
    {
        /* If current element is smaller
        than first then update both first
        and second */
        if (arr[i] > first)
        {
            second = first;
            first = arr[i];
        }

        /* If arr[i] is in between first
        and second then update second */
        else if (arr[i] > second)
            second = arr[i];
    }

    for (int i = 0; i < n; i++)
        if (arr[i] < second)
            cout << arr[i] << " ";
}

// Driver code
int main()
{
    int arr[] = { 2, -6, 3, 5, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    findElements(arr, n);
    return 0;
}
```

Java

```
// Java program to find all elements
// in array which have atleast
// two greater elements itself.
import java.util.*;
import java.io.*;

class GFG
{

static void findElements(int arr[], int n)
{
    int first = Integer.MIN_VALUE;
    int second = Integer.MAX_VALUE;

    for (int i = 0; i < n; i++)
    {
        // If current element is smaller
        // than first then update both
        // first and second
        if (arr[i] > first)
        {
            second = first;
            first = arr[i];
        }

        /* If arr[i] is in between
        first and second
        then update second */
        else if (arr[i] > second)
            second = arr[i];
    }

    for (int i = 0; i < n; i++)
        if (arr[i] < second)
            System.out.print(arr[i] + " ");
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 2, -6, 3, 5, 1};
    int n = arr.length;
    findElements(arr, n);
}
}

// This code is contributed by Sahil_Bansall
```

Python3

```
# Python 3 program to find all elements
# in array which have atleast two
# greater elements itself.
import sys

def findElements(arr, n):

    first = -sys.maxsize
    second = -sys.maxsize

    for i in range(0, n):

        # If current element is smaller
        # than first then update both
        # first and second
        if (arr[i] > first):

            second = first
            first = arr[i]

        # If arr[i] is in between first
        # and second then update second
        elif (arr[i] > second):
            second = arr[i]

    for i in range(0, n):
        if (arr[i] < second):
            print(arr[i], end = " ")

# Driver code
arr = [2, -6, 3, 5, 1]
n = len(arr)
findElements(arr, n)

# This code is contributed
# by Smitha Dinesh Semwal
```

C#

```
// C# program to find all elements
// in array which have atleast
// two greater elements itself.
using System;

class GFG
{
    static void findElements(int []arr,
```

```
        int n)
{
    int first = int.MinValue;
    int second = int.MaxValue;

    for (int i = 0; i < n; i++)
    {
        // If current element is smaller
        // than first then update both
        // first and second
        if (arr[i] > first)
        {
            second = first;
            first = arr[i];
        }

        /* If arr[i] is in between
        first and second
        then update second */
        else if (arr[i] > second)
            second = arr[i];
    }

    for (int i = 0; i < n; i++)
        if (arr[i] < second)
            Console.Write(arr[i] + " ");
}

// Driver code
public static void Main(String []args)
{
    int []arr = { 2, -6, 3, 5, 1};
    int n = arr.Length;
    findElements(arr, n);
}
}

// This code is contributed by parashar...
```

PHP

```
<?php
// PHP program to find all elements
// in array which have atleast two
// greater elements itself.

function findElements($arr, $n)
{
```

```
$first = PHP_INT_MIN;
$second = PHP_INT_MIN;
for ($i = 0; $i < $n; $i++)
{
    /* If current element is smaller
       than first then update both first
       and second */
    if ($arr[$i] > $first)
    {
        $second = $first;
        $first = $arr[$i];
    }

    /* If arr[i] is in between first
       and second then update second */
    else if ($arr[$i] > $second)
        $second = $arr[$i];
}

for($i = 0; $i < $n; $i++)
    if ($arr[$i] < $second)
        echo $arr[$i] , " ";
}

// Driver code
$arr = array(2, -6, 3, 5, 1);
$n = count($arr);
findElements($arr, $n);

// This code is contributed by vishal tripathi.
?>
```

Output :

```
2 -6 1
```

Time Complexity : O(n)

Improved By : [parashar](#), [jit_t](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-elements-array-least-two-greater-elements/>

Chapter 97

Find array with k number of merge sort calls

Find array with k number of merge sort calls - GeeksforGeeks

Given two numbers n and k, find an array containing values in [1, n] and requires exactly k calls of [recursive merge sort function](#).

Examples:

```
Input : n = 3
        k = 3
Output : a[] = {2, 1, 3}
Explanation:
Here, a[] = {2, 1, 3}
First of all, mergesort(0, 3) will be called,
which then sets mid = 1 and calls mergesort(0,
1) and mergesort(1, 3), which do not perform
any recursive calls because segments (0, 1)
and (1, 3) are sorted.
Hence, total mergesort calls are 3.
```

```
Input : n = 4
        k = 1
Output : a[] = {1, 2, 3, 4}
Explanation:
Here, a[] = {1, 2, 3, 4} then there will be
1 mergesort call - mergesort(0, 4), which
will check that the array is sorted and then
end.
```

If we have value of k even, then there is no solution, since the number of calls is always odd (one call in the beginning, and each call makes either 0 or 2 recursive calls).

If k is odd, let's try to start with a sorted permutation and try to "unsort" it. Let's make a function $\text{unsort}(l, r)$ that will do it. When we "unsort" a segment, we can either keep it sorted (if we already made enough calls), or make it non-sorted and then call $\text{unsort}(l, \text{mid})$ and $\text{unsort}(\text{mid}, r)$, if we need more calls. When we make a segment non-sorted, it's better to keep its both halves sorted; an easy way to handle this is to swap two middle element.

It's easy to see that the number of unsort calls is equal to the number of mergesort calls to sort the resulting permutation, so we can use this approach to try getting exactly k calls.

Below is the code for the above problem.

CPP

```
// C++ program to find an array that can be
// sorted with k merge sort calls.
#include <iostream>
using namespace std;

void unsort(int l, int r, int a[], int& k)
{
    if (k < 1 || l + 1 == r)
        return;

    // We make two recursive calls, so
    // reduce k by 2.
    k -= 2;

    int mid = (l + r) / 2;
    swap(a[mid - 1], a[mid]);
    unsort(l, mid, a, k);
    unsort(mid, r, a, k);
}

void arrayWithKCalls(int n, int k)
{
    if (k % 2 == 0) {
        cout << " NO SOLUTION ";
        return;
    }

    // Create an array with values
    // in [1, n]
    int a[n+1];
    a[0] = 1;
    for (int i = 1; i < n; i++)
        a[i] = i + 1;
    k--;

    // calling unsort function
    unsort(0, n, a, k);
}
```

```
for (int i = 0; i < n; ++i)
    cout << a[i] << ' ';
}

// Driver code
int main()
{
    int n = 10, k = 17;
    arrayWithKCalls(n, k);
    return 0;
}

Java

// Java program to find an array that can be
// sorted with k merge sort calls.
class GFG {

    static void unsort(int l, int r, int a[], int k)
    {

        if (k < 1 || l + 1 == r)
            return;

        // We make two recursive calls, so
        // reduce k by 2.
        k -= 2;

        int mid = (l + r) / 2;
        int temp = a[mid - 1];
        a[mid - 1] = a[mid];
        a[mid] = temp;

        unsort(l, mid, a, k);
        unsort(mid, r, a, k);
    }

    static void arrayWithKCalls(int n, int k)
    {
        if (k % 2 == 0) {
            System.out.print("NO SOLUTION");
            return;
        }

        // Create an array with values
        // in [1, n]
        int a[] = new int[n + 1];
```

```
a[0] = 1;

for (int i = 1; i < n; i++)
    a[i] = i + 1;
k--;

// calling unsort function
unsort(0, n, a, k);

for (int i = 0; i < n; ++i)
    System.out.print(a[i] + " ");
}

// Driver code
public static void main(String[] args)
{
    int n = 10, k = 17;

    arrayWithKCalls(n, k);
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python program to find
# an array that can be
# sorted with k merge
# sort calls.

def unsort(l,r,a,k):

    if (k < 1 or l + 1 == r):
        return

    # We make two recursive calls, so
    # reduce k by 2.
    k -= 2

    mid = (l + r) // 2
    temp = a[mid - 1]
    a[mid-1] = a[mid]
    a[mid] = temp

    unsort(l, mid, a, k)
    unsort(mid, r, a, k)
```

```
def arrayWithKCalls(n,k):

    if (k % 2 == 0):
        print("NO SOLUTION")
        return

    # Create an array with values
    # in [1, n]
    a = [0 for i in range(n + 2)]
    a[0] = 1
    for i in range(1, n):
        a[i] = i + 1
    k-=1

    # calling unsort function
    unsort(0, n, a, k)

    for i in range(n):
        print(a[i] , " ",end="")

# Driver code

n = 10
k = 17
arrayWithKCalls(n, k)

# This code is contributed
# by Anant Agarwal.
```

C#

```
// C# program to find an array that can
// be sorted with k merge sort calls.
using System;

class GFG {

    static void unsort(int l, int r,
                      int []a, int k)
    {
        if (k < 1 || l + 1 == r)
            return;

        // We make two recursive calls,
        // so reduce k by 2.
        k -= 2;
```

```
int mid = (l + r) / 2;
int temp = a[mid - 1];
a[mid - 1] = a[mid];
a[mid] = temp;

unsort(l, mid, a, k);
unsort(mid, r, a, k);
}

static void arrayWithKCalls(int n, int k)
{
    if (k % 2 == 0)
    {
        Console.WriteLine("NO SOLUTION");
        return;
    }

    // Create an array with
    // values in [1, n]
    int []a = new int[n + 1];
    a[0] = 1;

    for (int i = 1; i < n; i++)
        a[i] = i + 1;
    k--;

    // calling unsort function
    unsort(0, n, a, k);

    for (int i = 0; i < n; ++i)
        Console.Write(a[i] + " ");
}

// Driver code
public static void Main()
{
    int n = 10, k = 17;

    arrayWithKCalls(n, k);
}
}

// This code is contributed by vt_m.
```

Output:

3 1 4 6 2 8 5 9 7 10

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-array-k-number-merge-sort-calls/>

Chapter 98

Find distinct elements common to all rows of a matrix

Find distinct elements common to all rows of a matrix - GeeksforGeeks

Given a $n \times n$ matrix. The problem is to find all the distinct elements common to all rows of the matrix. The elements can be printed in any order.

Examples:

```
Input : mat[][] = { {2, 1, 4, 3},  
                   {1, 2, 3, 2},  
                   {3, 6, 2, 3},  
                   {5, 2, 5, 3} }  
Output : 2 3  
  
Input : mat[][] = { {12, 1, 14, 3, 16},  
                   {14, 2, 1, 3, 35},  
                   {14, 1, 14, 3, 11},  
                   {14, 25, 3, 2, 1},  
                   {1, 18, 3, 21, 14} }  
Output : 1 3 14
```

Method 1: Using three nested loops. Check if an element of 1st row is present in all the subsequent rows. Time Complexity of $O(n^3)$. Extra space could be required to handle the duplicate elements.

Method 2: Sort all the rows of the matrix individually in increasing order. Then apply a modified approach of the problem of [finding common elements in 3 sorted arrays](#). Below an implementation for the same is given.

C++

```
// C++ implementation to find distinct elements
// common to all rows of a matrix
#include <bits/stdc++.h>
using namespace std;
const int MAX = 100;

// function to individually sort
// each row in increasing order
void sortRows(int mat[][] [MAX], int n)
{
    for (int i=0; i<n; i++)
        sort(mat[i], mat[i] + n);
}

// function to find all the common elements
void findAndPrintCommonElements(int mat[][] [MAX], int n)
{
    // sort rows individually
    sortRows(mat, n);

    // current column index of each row is stored
    // from where the element is being searched in
    // that row
    int curr_index[n];
    memset(curr_index, 0, sizeof(curr_index));
    int f = 0;

    for (; curr_index[0]<n; curr_index[0]++)
    {
        // value present at the current column index
        // of 1st row
        int value = mat[0][curr_index[0]];

        bool present = true;

        // 'value' is being searched in all the
        // subsequent rows
        for (int i=1; i<n; i++)
        {
            // iterate through all the elements of
            // the row from its current column index
            // till an element greater than the 'value'
            // is found or the end of the row is
            // encountered
            while (curr_index[i] < n &&
                   mat[i][curr_index[i]] <= value)
                curr_index[i]++;
        }
    }
}
```

```
// if the element was not present at the column
// before to the 'curr_index' of the row
if (mat[i][curr_index[i]-1] != value)
    present = false;

// if all elements of the row have
// been traversed
if (curr_index[i] == n)
{
    f = 1;
    break;
}
}

// if the 'value' is common to all the rows
if (present)
    cout << value << " ";

// if any row have been completely traversed
// then no more common elements can be found
if (f == 1)
    break;
}
}

// Driver program to test above
int main()
{
    int mat[][] [MAX] = { {12, 1, 14, 3, 16},
    {14, 2, 1, 3, 35},
    {14, 1, 14, 3, 11},
    {14, 25, 3, 2, 1},
    {1, 18, 3, 21, 14}
};

    int n = 5;
    findAndPrintCommonElements(mat, n);
    return 0;
}
```

Java

```
// JAVA Code to find distinct elements
// common to all rows of a matrix
import java.util.*;

class GFG {
```

```
// function to individually sort
// each row in increasing order
public static void sortRows(int mat[][] , int n)
{
    for (int i=0; i<n; i++)
        Arrays.sort(mat[i]);
}

// function to find all the common elements
public static void findAndPrintCommonElements(int mat[][] ,
                                              int n)
{
    // sort rows individually
    sortRows(mat, n);

    // current column index of each row is stored
    // from where the element is being searched in
    // that row
    int curr_index[] = new int[n];

    int f = 0;

    for (; curr_index[0]<n; curr_index[0]++)
    {
        // value present at the current column index
        // of 1st row
        int value = mat[0][curr_index[0]];

        boolean present = true;

        // 'value' is being searched in all the
        // subsequent rows
        for (int i=1; i<n; i++)
        {
            // iterate through all the elements of
            // the row from its current column index
            // till an element greater than the 'value'
            // is found or the end of the row is
            // encountered
            while (curr_index[i] < n &&
                   mat[i][curr_index[i]] <= value)
                curr_index[i]++;
        }

        // if the element was not present at the
        // column before to the 'curr_index' of the
        // row
        if (mat[i][curr_index[i]-1] != value)
            present = false;
    }
}
```

```
// if all elements of the row have
// been traversed
if (curr_index[i] == n)
{
    f = 1;
    break;
}
}

// if the 'value' is common to all the rows
if (present)
    System.out.print(value+" ");

// if any row have been completely traversed
// then no more common elements can be found
if (f == 1)
    break;
}

/* Driver program to test above function */
public static void main(String[] args)
{
    int mat[][] = { {12, 1, 14, 3, 16},
                    {14, 2, 1, 3, 35},
                    {14, 1, 14, 3, 11},
                    {14, 25, 3, 2, 1},
                    {1, 18, 3, 21, 14}
                };

    int n = 5;
    findAndPrintCommonElements(mat, n);
}
}

// This code is contributed by Arnav Kr. Mandal.
```

Output:

1 3 14

Time Complexity: $O(n^2 \log n)$, each row of size n requires $O(n \log n)$ for sorting and there are total n rows.

Auxiliary Space : $O(n)$ to store current column indexes for each row.

Method 3: It uses the concept of hashing. The following steps are:

1. Map the element of 1st row in a hash table. Let it be **hash**.
2. Fow row = 2 to n
3. Map each element of the current row into a temporary hash table. Let it be **temp**.
4. Iterate through the elements of **hash** and check that the elements in **hash** are present in **temp**. If not present then delete those elements from **hash**.
5. When all the rows are being processed in this manner, then the elements left in **hash** are the required common elements.

```
// C++ program to find distinct elements
// common to all rows of a matrix
#include <bits/stdc++.h>
using namespace std;

const int MAX = 100;

// function to individually sort
// each row in increasing order
void findAndPrintCommonElements(int mat[][] [MAX], int n)
{
    unordered_set<int> us;

    // map elements of first row
    // into 'us'
    for (int i=0; i<n; i++)
        us.insert(mat[0][i]);

    for (int i=1; i<n; i++)
    {
        unordered_set<int> temp;
        // mapping elements of current row
        // in 'temp'
        for (int j=0; j<n; j++)
            temp.insert(mat[i][j]);

        unordered_set<int>:: iterator itr;

        // iterate through all the elements
        // of 'us'
        for (itr=us.begin(); itr!=us.end(); itr++)

            // if an element of 'us' is not present
            // into 'temp', then erase that element
            // from 'us'
            if (temp.find(*itr) == temp.end())
                us.erase(*itr);
    }
}
```

```
// if size of 'us' becomes 0,
// then there are no common elements
if (us.size() == 0)
    break;
}

// print the common elements
unordered_set<int>:: iterator itr;
for (itr=us.begin(); itr!=us.end(); itr++)
    cout << *itr << " ";
}

// Driver program to test above
int main()
{
    int mat[][] [MAX] = { {2, 1, 4, 3},
                         {1, 2, 3, 2},
                         {3, 6, 2, 3},
                         {5, 2, 5, 3} };
    int n = 4;
    findAndPrintCommonElements(mat, n);
    return 0;
}
```

Output:

3 2

Time Complexity: $O(n^2)$
Space Complexity: $O(n)$

Source

<https://www.geeksforgeeks.org/find-distinct-elements-common-rows-matrix/>

Chapter 99

Find elements larger than half of the elements in an array

Find elements larger than half of the elements in an array - GeeksforGeeks

Given an array of n elements, the task is to find the elements that are greater than half of elements in an array. In case of odd elements, we need to print elements larger than $\text{floor}(n/2)$ elements where n is total number of elements in array.

Examples :

Input : arr[] = {1, 6, 3, 4}
Output : 4 6

Input : arr[] = {10, 4, 2, 8, 9}
Output : 10 9 8

A **naive approach** is to take an element and compare with all other elements and if it is greater then increment the count and then check if count is greater than $n/2$ elements then print.

An **efficient method** is to sort the array in ascending order and then print last $\text{ceil}(n/2)$ elements from sorted array.

Below is C++ implementation of this sorting based approach.

C/C++

```
// C++ program to find elements that are larger than
// half of the elements in array
#include <bits/stdc++.h>
using namespace std;
```

```
// Prints elements larger than n/2 element
void findLarger(int arr[], int n)
{
    // Sort the array in ascending order
    sort(arr, arr + n);

    // Print last ceil(n/2) elements
    for (int i = n-1; i >= n/2; i--)
        cout << arr[i] << " ";
}

// Driver program
int main()
{
    int arr[] = {1, 3, 6, 1, 0, 9};
    int n = sizeof(arr)/sizeof(arr[0]);
    findLarger(arr, n);
    return 0;
}
```

Java

```
// Java program to find elements that are
// larger than half of the elements in array
import java.util.*;

class Gfg
{
    // Prints elements larger than n/2 element
    static void findLarger(int arr[], int n)
    {
        // Sort the array in ascending order
        Arrays.sort(arr);

        // Print last ceil(n/2) elements
        for (int i = n-1; i >= n/2; i--)
            System.out.print(arr[i] + " ");
    }

    // Driver program
    public static void main(String[] args)
    {
        int arr[] = {1, 3, 6, 1, 0, 9};
        int n = arr.length;
        findLarger(arr, n);
    }
}
```

```
// This code is contributed by Raghav Sharma
```

Python

```
# Python program to find elements that are larger than
# half of the elements in array
# Prints elements larger than n/2 element
def findLarger(arr,n):

    # Sort the array in ascending order
    x = sorted(arr)

    # Print last ceil(n/2) elements
    for i in range(n/2,n):
        print(x[i]),

# Driver program
arr = [1, 3, 6, 1, 0, 9]
n = len(arr);
findLarger(arr,n)
```

This code is contributed by Afzal Ansari

C#

```
// C# program to find elements
// that are larger than half
// of the elements in array
using System;

class GFG
{
    // Prints elements larger
    // than n/2 element
    static void findLarger(int []arr,
                          int n)
    {
        // Sort the array
        // in ascending order
        Array.Sort(arr);

        // Print last ceil(n/2) elements
        for (int i = n - 1; i >= n / 2; i--)
            Console.Write(arr[i] + " ");
    }

    // Driver Code
}
```

```
public static void Main()
{
    int []arr = {1, 3, 6, 1, 0, 9};
    int n = arr.Length;
    findLarger(arr, n);
}

// This code is contributed
// by nitin mittal.
```

PHP

```
<?php
// PHP program to find elements
// that are larger than half of
// the elements in array

// Prints elements larger
// than n/2 element
function findLarger($arr, $n)
{
    // Sort the array in
    // ascending order
    sort($arr);

    // Print last ceil(n/2) elements
    for ($i = $n - 1; $i >= $n / 2; $i--)
        echo $arr[$i] , " ";
}

// Driver Code
$arr = array(1, 3, 6, 1, 0, 9);
$n = count($arr);
findLarger($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output :

9 6 3

Improved By : [nitin mittal](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-elements-larger-half-elements-array/>

Chapter 100

Find first k natural numbers missing in given array

Find first k natural numbers missing in given array - GeeksforGeeks

Given an array of size n and a number k, we need to print first k natural numbers that are not there in given array.

Examples:

```
Input : [2 3 4]
        k = 3
Output : [1 5 6]
```

```
Input : [-2 -3 4]
        k = 2
Output : [1 2]
```

- 1) Sort the given array.
- 2) After sorting, we find position of first positive number in array.
- 3) Now we traverse the array and keep printing elements in gaps between two consecutive array elements.
- 4) If gaps don't cover k missing numbers, we print numbers greater than the largest array element.

C++

```
// C++ program to find missing k numbers
// in an array.
#include <bits/stdc++.h>
using namespace std;
```

```
// Prints first k natural numbers in
// arr[0..n-1]
void printKMissing(int arr[], int n, int k)
{
    sort(arr, arr + n);

    // Find first positive number
    int i = 0;
    while (i < n && arr[i] <= 0)
        i++;

    // Now find missing numbers
    // between array elements
    int count = 0, curr = 1;
    while (count < k && i < n) {
        if (arr[i] != curr) {
            cout << curr << " ";
            count++;
        }
        else
            i++;

        curr++;
    }

    // Find missing numbers after
    // maximum.
    while (count < k) {
        cout << curr << " ";
        curr++;
        count++;
    }
}

// Driver code
int main()
{
    int arr[] = { 2, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    printKMissing(arr, n, k);
    return 0;
}
```

Java

```
// Java program to find missing k numbers
// in an array.
```

```
import java.util.Arrays;

class GFG {
    // Prints first k natural numbers in
    // arr[0..n-1]
    static void printKMissing(int[] arr, int n, int k)
    {
        Arrays.sort(arr);

        // Find first positive number
        int i = 0;
        while (i < n && arr[i] <= 0)
            i++;

        // Now find missing numbers
        // between array elements
        int count = 0, curr = 1;
        while (count < k && i < n) {
            if (arr[i] != curr) {
                System.out.print(curr + " ");
                count++;
            }
            else
                i++;
            curr++;
        }

        // Find missing numbers after
        // maximum.
        while (count < k) {
            System.out.print(curr + " ");
            curr++;
            count++;
        }
    }

    // Driver code
    public static void main(String[] args)
    {
        int[] arr = { 2, 3, 4 };
        int n = arr.length;
        int k = 3;
        printKMissing(arr, n, k);
    }
}
/* This code is contributed by Mr. Somesh Awasthi */
```

C#

```
// C# program to find missing
// k numbers in an array.
using System;

class GFG {
    // Prints first k natural numbers
    // in arr[0..n-1]
    static void printKMissing(int[] arr,
                              int n,
                              int k)
    {
        Array.Sort(arr);

        // Find first positive number
        int i = 0;
        while (i < n && arr[i] <= 0)
            i++;

        // Now find missing numbers
        // between array elements
        int count = 0, curr = 1;
        while (count < k && i < n) {
            if (arr[i] != curr) {
                Console.Write(curr + " ");
                count++;
            }
            else
                i++;
            curr++;
        }
    }

    // Find missing numbers
    // after maximum.
    while (count < k) {
        Console.Write(curr + " ");
        curr++;
        count++;
    }
}

// Driver code
public static void Main()
{
    int[] arr = {2, 3, 4};
    int n = arr.Length;
    int k = 3;
    printKMissing(arr, n, k);
}
```

```
}
```

```
// This code is contributed by Nitin Mittal
```

PHP

```
<?php
// PHP program to find missing k numbers
// in an array.

// Prints first k natural numbers in
// arr[0..n-1]
function printKMissing($arr, $n, $k)
{
    sort($arr); sort($arr , $n);

    // Find first positive number
    $i = 0;
    while ($i < $n && $arr[$i] <= 0)
        $i++;

    // Now find missing numbers
    // between array elements
    $count = 0; $curr = 1;
    while ($count < $k && $i < $n) {
        if ($arr[$i] != $curr) {
            echo $curr , " ";
            $count++;
        }
        else
            $i++;

        $curr++;
    }

    // Find missing numbers after
    // maximum.
    while ($count < $k) {
        echo $curr , " ";
        $curr++;
        $count++;
    }
}

// Driver code
$arr =array ( 2, 3, 4 );
$n = sizeof($arr);
$k = 3;
```

```
printKMissing($arr, $n, $k);  
// This code is contributed by Nitin Mittal.  
?>
```

Output:

1 5 6

Time Complexity : $O(n \log n)$

Improved By : nitin mittal

Source

<https://www.geeksforgeeks.org/find-first-k-natural-numbers-missing-given-array/>

Chapter 101

Find four elements that sum to a given value Set 2 (O($n^2 \log n$) Solution)

Find four elements that sum to a given value Set 2 (O($n^2 \log n$) Solution) - GeeksforGeeks

Given an array of integers, find any one combination of four elements in the array whose sum is equal to a given value X.

For example, if the given array is {10, 2, 3, 4, 5, 9, 7, 8} and X = 23, then your function should print “3 5 7 8” (3 + 5 + 7 + 8 = 23).

We have discussed a O(n^3) algorithm in [the previous post](#) on this topic. The problem can be solved in O($n^2 \log n$) time with the help of auxiliary space.

Thanks to itsnimish for suggesting this method. Following is the detailed process.

Let the input array be A[].

1) Create an auxiliary array aux[] and store sum of all possible pairs in aux[]. The size of aux[] will be $n*(n-1)/2$ where n is the size of A[].

2) Sort the auxiliary array aux[].

3) Now the problem reduces to find two elements in aux[] with sum equal to X. We can use method 1 of [this post](#) to find the two elements efficiently. There is following important point to note though. An element of aux[] represents a pair from A[]. While picking two elements from aux[], we must check whether the two elements have an element of A[] in common. For example, if first element sum of A[1] and A[2], and second element is sum of A[2] and A[4], then these two elements of aux[] don't represent four distinct elements of input array A[].

Following is C implementation of this method.

```
#include <stdio.h>
#include <stdlib.h>
```

```
// The following structure is needed to store pair sums in aux[]
struct pairSum
{
    int first; // index (int A[]) of first element in pair
    int sec; // index of second element in pair
    int sum; // sum of the pair
};

// Following function is needed for library function qsort()
int compare (const void *a, const void * b)
{
    return ( (*(pairSum *)a).sum - (*(pairSum*)b).sum );
}

// Function to check if two given pairs have any common element or not
bool noCommon(struct pairSum a, struct pairSum b)
{
    if (a.first == b.first || a.first == b.sec ||
        a.sec == b.first || a.sec == b.sec)
        return false;
    return true;
}

// The function finds four elements with given sum X
void findFourElements (int arr[], int n, int X)
{
    int i, j;

    // Create an auxiliary array to store all pair sums
    int size = (n*(n-1))/2;
    struct pairSum aux[size];

    /* Generate all possible pairs from A[] and store sums
       of all possible pairs in aux[] */
    int k = 0;
    for (i = 0; i < n-1; i++)
    {
        for (j = i+1; j < n; j++)
        {
            aux[k].sum = arr[i] + arr[j];
            aux[k].first = i;
            aux[k].sec = j;
            k++;
        }
    }
}
```

```
// Sort the aux[] array using library function for sorting
qsort (aux, size, sizeof(aux[0]), compare);

// Now start two index variables from two corners of array
// and move them toward each other.
i = 0;
j = size-1;
while (i < size && j >=0 )
{
    if ((aux[i].sum + aux[j].sum == X) && noCommon(aux[i], aux[j]))
    {
        printf ("%d, %d, %d, %d\n", arr[aux[i].first], arr[aux[i].sec],
               arr[aux[j].first], arr[aux[j].sec]);
        return;
    }
    else if (aux[i].sum + aux[j].sum < X)
        i++;
    else
        j--;
}
}

// Driver program to test above function
int main()
{
    int arr[] = {10, 20, 30, 40, 1, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    int X = 91;
    findFourElements (arr, n, X);
    return 0;
}
```

Output:

20, 1, 30, 40

Please note that the above code prints only one quadruple. If we remove the return statement and add statements “`i++; j-;`”, then it prints same quadruple five times. The code can modified to print all quadruples only once. It has been kept this way to keep it simple.

Time complexity: The step 1 takes $O(n^2)$ time. The second step is sorting an array of size $O(n^2)$. Sorting can be done in $O(n^2\log n)$ time using merge sort or heap sort or any other $O(n\log n)$ algorithm. The third step takes $O(n^2)$ time. So overall complexity is $O(n^2\log n)$.

Auxiliary Space: $O(n^2)$. The big size of auxiliary array can be a concern in this method.

1. Store sums of all pairs in a hash table

2. Traverse through all pairs again and search for $X - (\text{current pair sum})$ in the hash table.
3. If a pair is found with the required sum, then make sure that all elements are distinct array elements and an element is not considered more than once.

```
// A hashing based CPP program to find if there are
// four elements with given sum.
#include <bits/stdc++.h>
using namespace std;

// The function finds four elements with given sum X
void findFourElements (int arr[], int n, int X)
{
    // Store sums of all pairs in a hash table
    unordered_map<int, pair<int, int>> mp;
    for (int i = 0; i < n-1; i++)
        for (int j = i+1; j < n; j++)
            mp[arr[i] + arr[j]] = {i, j};

    // Traverse through all pairs and search
    // for  $X - (\text{current pair sum})$ .
    for (int i = 0; i < n-1; i++)
    {
        for (int j = i+1; j < n; j++)
        {
            int sum = arr[i] + arr[j];

            // If  $X - sum$  is present in hash table,
            if (mp.find(X - sum) != mp.end())
            {

                // Making sure that all elements are
                // distinct array elements and an element
                // is not considered more than once.
                pair<int, int> p = mp[X - sum];
                if (p.first != i && p.first != j &&
                    p.second != i && p.second != j)
                {
                    cout << arr[i] << ", " << arr[j] << ", "
                        << arr[p.first] << ", "
                        << arr[p.second];
                    return;
                }
            }
        }
    }
}
```

```
// Driver program to test above function
int main()
{
    int arr[] = {10, 20, 30, 40, 1, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    int X = 91;
    findFourElements(arr, n, X);
    return 0;
}
```

Improved By : [Mrinal Tak](#), [ImStark](#)

Source

<https://www.geeksforgeeks.org/find-four-elements-that-sum-to-a-given-value-set-2/>

Chapter 102

Find if k bookings possible with given arrival and departure times

Find if k bookings possible with given arrival and departure times - GeeksforGeeks

A hotel manager has to process N advance bookings of rooms for the next season. His hotel has K rooms. Bookings contain an arrival date and a departure date. He wants to find out whether there are enough rooms in the hotel to satisfy the demand.

The idea is to sort the arrays and keep track of overlaps.

Examples:

```
Input : Arrivals : [1 3 5]
        Departures : [2 6 8]
        K : 1
Output : False
Hotel manager needs at least two
rooms as the second and third
intervals overlap.
```

The idea is store arrival and departure times in an auxiliary array with an additional marker to indicate whether time is arrival or departure. Now sort the array. Process the sorted array, for every arrival increment active bookings. And for every departure, decrement. Keep track of maximum active bookings. If count of active bookings at any moment is more than k, then return false. Else return true.

Below is the c++ code for above approach.

```
#include <bits/stdc++.h>
```

```
using namespace std;

bool areBookingsPossible(int arrival[],
                         int departure[], int n, int k)
{
    vector<pair<int, int>> ans;

    // create a common vector both arrivals
    // and departures.
    for (int i = 0; i < n; i++) {
        ans.push_back(make_pair(arrival[i], 1));
        ans.push_back(make_pair(departure[i], 0));
    }

    // sort the vector
    sort(ans.begin(), ans.end());

    int curr_active = 0, max_active = 0;

    for (int i = 0; i < ans.size(); i++) {

        // if new arrival, increment current
        // guests count and update max active
        // guests so far
        if (ans[i].second == 1) {
            curr_active++;
            max_active = max(max_active,
                             curr_active);
        }

        // if a guest departs, decrement
        // current guests count.
        else
            curr_active--;
    }

    // if max active guests at any instant
    // were more than the available rooms,
    // return false. Else return true.
    return (k >= max_active);
}

int main()
{
    int arrival[] = { 1, 3, 5 };
    int departure[] = { 2, 6, 8 };
    int n = sizeof(arrival) / sizeof(arrival[0]);
    cout << (areBookingsPossible(arrival,
```

```
        departure, n, 1) ? "Yes\n" : "No\n");
    return 0;
}
```

Output:

No

Time Complexity : $O(n \log n)$

Auxiliary Space : $O(n)$

Source

<https://www.geeksforgeeks.org/find-k-bookings-possible-given-arrival-departure-times/>

Chapter 103

Find k maximum elements of array in original order

Find k maximum elements of array in original order - GeeksforGeeks

Given an array arr[] and an integer k, we need to print k maximum elements of given array. The elements should printed in the order of the input.

Note : k is always less than or equal to n.

Examples:

```
Input : arr[] = {10 50 30 60 15}
        k = 2
Output : 50 60
The top 2 elements are printed
as per their appearance in original
array.
```

```
Input : arr[] = {50 8 45 12 25 40 84}
        k = 3
Output : 50 45 84
```

Method 1: We search for the maximum element k times in the given array. Each time we find one maximum element, we print it and repalce it with minus infinite (INT_MIN in C) in the array. The time complexity of this method is O(n*k).

Method 2: In this method, we store the original array in a new array and will sort the new array in descending order. After sorting, we iterate the original array from 0 to n and print all those elements that appear in first k elements of new array. For searching, we can do [Binary Search](#).

C++

```
// CPP program to find k maximum elements
// of array in original order
#include <bits/stdc++.h>
using namespace std;

// Function to print m Maximum elements
void printMax(int arr[], int k, int n)
{
    // vector to store the copy of the
    // original array
    vector<int> brr(arr, arr + n);

    // Sorting the vector in descending
    // order. Please refer below link for
    // details
    // https://www.geeksforgeeks.org/sort-c-stl/
    sort(brr.begin(), brr.end(), greater<int>());

    // Traversing through original array and
    // printing all those elements that are
    // in first k of sorted vector.
    // Please refer https://goo.gl/44Rwgt
    // for details of binary_search()
    for (int i = 0; i < n; ++i)
        if (binary_search(brr.begin(),
                          brr.begin() + k, arr[i],
                          greater<int>()))
            cout << arr[i] << " ";
}

// Driver code
int main()
{
    int arr[] = { 50, 8, 45, 12, 25, 40, 84 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    printMax(arr, k, n);
    return 0;
}
```

Java

```
// Java program to find k maximum
// elements of array in original order
import java.util.Arrays;
import java.util.Collections;

public class GfG {
```

```
// Function to print m Maximum elements
public static void printMax(int arr[], int k, int n)
{
    // Array to store the copy
    // of the original array
    Integer[] brr = new Integer[n];

    for (int i = 0; i < n; i++)
        brr[i] = arr[i];

    // Sorting the array in
    // descending order
    Arrays.sort(brr, Collections.reverseOrder());

    // Traversing through original array and
    // printing all those elements that are
    // in first k of sorted array.
    // Please refer https://goo.gl/uj5RCD
    // for details of Arrays.binarySearch()
    for (int i = 0; i < n; ++i)
        if (Arrays.binarySearch(brr, arr[i],
            Collections.reverseOrder()) >= 0
            && Arrays.binarySearch(brr, arr[i],
            Collections.reverseOrder()) < k)

            System.out.print(arr[i]+ " ");

}

// Driver code
public static void main(String args[])
{
    int arr[] = { 50, 8, 45, 12, 25, 40, 84 };
    int n = arr.length;
    int k = 3;
    printMax(arr, k, n);
}

// This code is contributed by Swetank Modi
```

Output :

50 45 84

Time Complexity : $O(n \log n)$ for sorting.
Auxiliary Space : $O(n)$

Source

<https://www.geeksforgeeks.org/find-k-maximum-elements-array-original-order/>

Chapter 104

Find k-th smallest element in given n ranges

Find k-th smallest element in given n ranges - GeeksforGeeks

Given n and q, i.e, the number of ranges and number of queries, find the kth smallest element for each query (assume k>1).Print the value of kth smallest element if it exists, else print -1.

Examples :

```
Input : arr[] = {{1, 4}, {6, 8}}
        queries[] = {2, 6, 10};
Output : 2
         7
         -1
```

After combining the given ranges, the numbers become 1 2 3 4 6 7 8. As here 2nd element is 2, so we print 2. As 6th element is 7, so we print 7 and as 10th element doesn't exist, so we print -1.

```
Input : arr[] = {{2, 6}, {5, 7}}
        queries[] = {5, 8};
Output : 6
         -1
```

After combining the given ranges, the numbers become 2 3 4 5 6 7. As here 5th element is 6, so we print 6 and as 8th element doesn't exist, so we print -1.

The idea is to first Prerequisite : [Merge Overlapping Intervals](#) and keep all intervals sorted

in ascending order of start time. After merging in an array merged[], we use linear search to find kth smallest element. Below is the implementation of the above approach :

```

// C++ implementation to solve k queries
// for given n ranges
#include <bits/stdc++.h>
using namespace std;

// Structure to store the
// start and end point
struct Interval
{
    int s;
    int e;
};

// Comparison function for sorting
bool comp(Interval a, Interval b)
{
    return a.s < b.s;
}

// Function to find Kth smallest number in a vector
// of merged intervals
int kthSmallestNum(vector<Interval> merged, int k)
{
    int n = merged.size();

    // Traverse merged[] to find
    // Kth smallest element using Linear search.
    for (int j = 0; j < n; j++)
    {
        if (k <= abs(merged[j].e -
                     merged[j].s + 1))
            return (merged[j].s + k - 1);

        k = k - abs(merged[j].e -
                    merged[j].s + 1);
    }

    if (k)
        return -1;
}

// To combined both type of ranges,
// overlapping as well as non-overlapping.
void mergeIntervals(vector<Interval> &merged,
                     Interval arr[], int n)

```

```
{  
    // Sorting intervals according to start  
    // time  
    sort(arr, arr + n, comp);  
  
    // Merging all intervals into merged  
    merged.push_back(arr[0]);  
    for (int i = 1; i < n; i++)  
    {  
        // To check if starting point of next  
        // range is lying between the previous  
        // range and ending point of next range  
        // is greater than the Ending point  
        // of previous range then update ending  
        // point of previous range by ending  
        // point of next range.  
        Interval prev = merged.back();  
        Interval curr = arr[i];  
        if ((curr.s >= prev.s &&  
            curr.s <= prev.e) &&  
            (curr.e > prev.e))  
  
            merged.back().e = curr.e;  
  
        else  
        {  
            // If starting point of next range  
            // is greater than the ending point  
            // of previous range then store next range  
            // in merged[].  
            if (curr.s > prev.e)  
                merged.push_back(curr);  
        }  
    }  
  
    // Driver\'s Function  
    int main()  
    {  
        Interval arr[] = {{2, 6}, {4, 7}};  
        int n = sizeof(arr)/sizeof(arr[0]);  
        int query[] = {5, 8};  
        int q = sizeof(query)/sizeof(query[0]);  
  
        // Merge all intervals into merged[]  
        vector<Interval>merged;  
        mergeIntervals(merged, arr, n);
```

```
// Processing all queries on merged
// intervals
for (int i = 0; i < q; i++)
    cout << kthSmallestNum(merged, query[i])
    << endl;

return 0;
}
```

Output:

```
6
-1
```

Time Complexity : $O(n \log(n))$

Source

<https://www.geeksforgeeks.org/find-k-th-smallest-element-in-given-n-ranges/>

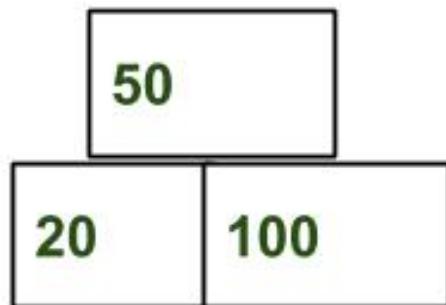
Chapter 105

Find maximum height pyramid from the given array of objects

Find maximum height pyramid from the given array of objects - GeeksforGeeks

Given n objects, with each object has width w_i . We need to arrange them in a pyramidal way such that :

1. Total width of i^{th} is less than $(i + 1)^{\text{th}}$.
2. Total number of objects in the i^{th} is less than $(i + 1)^{\text{th}}$.



The task is to find the maximum height that can be achieved from given objects.

Examples :

```
Input : arr[] = {40, 100, 20, 30}
Output : 2
Top level : 30.
Lower (or bottom) level : 20, 40 and 100
Other possibility can be placing
20 on the top, and at second level any
other 4 objects. Another possibility is
to place 40 at top and other three at the
bottom.
```

```
Input : arr[] = {10, 20, 30, 50, 60, 70}
Output : 3
```

The idea is to use greedy approach by placing the object with the lowest width at the top, the next object at the level right below and so on.

To find the maximum number of levels, sort the given array and try to form pyramid from top to bottom. Find the smallest element of array i.e first element of array after sorting, place it on the top. Then try to build levels below it with greater number of objects and greater width.

Below is the implementation of this approach:

C++

```
// C++ program to find maximum height pyramid
// from the given object width.
#include<bits/stdc++.h>
using namespace std;

// Returns maximum number of pyramidal levels
// n boxes of given widths.
int maxLevel(int boxes[], int n)
{
    // Sort objects in increasing order of widths
    sort(boxes, boxes + n);

    int ans = 1; // Initialize result

    // Total width of previous level and total
    // number of objects in previous level
    int prev_width = boxes[0];
    int prev_count = 1;

    // Number of object in current level.
```

```
int curr_count = 0;

// Width of current level.
int curr_width = 0;
for (int i=1; i<n; i++)
{
    // Picking the object. So increase current
    // width and number of object.
    curr_width += boxes[i];
    curr_count += 1;

    // If current width and number of object
    // are greater than previous.
    if (curr_width > prev_width &&
        curr_count > prev_count)
    {
        // Update previous width, number of
        // object on previous level.
        prev_width = curr_width;
        prev_count = curr_count;

        // Reset width of current level, number
        // of object on current level.
        curr_count = 0;
        curr_width = 0;

        // Increment number of level.
        ans++;
    }
}

return ans;
}

// Driver Program
int main()
{
    int boxes[] = {10, 20, 30, 50, 60, 70};
    int n = sizeof(boxes)/sizeof(boxes[0]);
    cout << maxLevel(boxes, n) << endl;
    return 0;
}
```

Java

```
// Java program to find maximum height pyramid
// from the given object width.
import java.io.*;
```

```
import java.util.Arrays;

class GFG {

    // Returns maximum number of pyramidcal
    // levels n boxes of given widths.
    static int maxLevel(int []boxes, int n)
    {

        // Sort objects in increasing order
        // of widths
        Arrays.sort(boxes);

        int ans = 1; // Initialize result

        // Total width of previous level
        // and total number of objects in
        // previous level
        int prev_width = boxes[0];
        int prev_count = 1;

        // Number of object in current
        // level.
        int curr_count = 0;

        // Width of current level.
        int curr_width = 0;
        for (int i = 1; i < n; i++)
        {
            // Picking the object. So
            // increase current width
            // and number of object.
            curr_width += boxes[i];
            curr_count += 1;

            // If current width and
            // number of object
            // are greater than previous.
            if (curr_width > prev_width &&
                curr_count > prev_count)
            {

                // Update previous width,
                // number of object on
                // previous level.
                prev_width = curr_width;
                prev_count = curr_count;
            }
        }
    }
}
```

```
// Reset width of current
// level, number of object
// on current level.
curr_count = 0;
curr_width = 0;

// Increment number of
// level.
ans++;
}
}

return ans;
}

// Driver Program
static public void main (String[] args)
{
    int []boxes = {10, 20, 30, 50, 60, 70};
    int n = boxes.length;
    System.out.println(maxLevel(boxes, n));
}
}

// This code is contributed by anuj_67.
```

Python 3

```
# Python 3 program to find
# maximum height pyramid from
# the given object width.

# Returns maximum number
# of pyramidal levels n
# boxes of given widths.
def maxLevel(boxes, n):

    # Sort objects in increasing
    # order of widths
    boxes.sort()

    ans = 1 # Initialize result

    # Total width of previous
    # level and total number of
    # objects in previous level
    prev_width = boxes[0]
    prev_count = 1
```

```
# Number of object in
# current level.
curr_count = 0

# Width of current level.
curr_width = 0
for i in range(1, n):

    # Picking the object. So
    # increase current width
    # and number of object.
    curr_width += boxes[i]
    curr_count += 1

    # If current width and
    # number of object are
    # greater than previous.
    if (curr_width > prev_width and
        curr_count > prev_count):

        # Update previous width,
        # number of object on
        # previous level.
        prev_width = curr_width
        prev_count = curr_count

        # Reset width of current
        # level, number of object
        # on current level.
        curr_count = 0
        curr_width = 0

    # Increment number of level.
    ans += 1
return ans

# Driver Code
if __name__ == "__main__":
    boxes= [10, 20, 30, 50, 60, 70]
    n = len(boxes)
    print(maxLevel(boxes, n))

# This code is contributed
# by ChitraNayal
```

C#

```
// C# program to find maximum height pyramid
// from the given object width.
using System;

public class GFG {

    // Returns maximum number of pyramidcal
    // levels n boxes of given widths.
    static int maxLevel(int []boxes, int n)
    {
        // Sort objects in increasing order
        // of widths
        Array.Sort(boxes);

        int ans = 1; // Initialize result

        // Total width of previous level
        // and total number of objects in
        // previous level
        int prev_width = boxes[0];
        int prev_count = 1;

        // Number of object in current
        // level.
        int curr_count = 0;

        // Width of current level.
        int curr_width = 0;
        for (int i=1; i<n; i++)
        {
            // Picking the object. So
            // increase current width
            // and number of object.
            curr_width += boxes[i];
            curr_count += 1;

            // If current width and
            // number of object
            // are greater than previous.
            if (curr_width > prev_width &&
                curr_count > prev_count)
            {

                // Update previous width,
                // number of object on
                // previous level.
                prev_width = curr_width;
                prev_count = curr_count;
            }
        }
    }
}
```

```
// Reset width of current
// level, number of object
// on current level.
curr_count = 0;
curr_width = 0;

// Increment number of
// level.
ans++;
}
}

return ans;
}

// Driver Program
static public void Main ()
{
    int []boxes = {10, 20, 30, 50, 60, 70};
    int n = boxes.Length;
    Console.WriteLine(maxLevel(boxes, n));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP program to find maximum
// height pyramid from the
// given object width.

// Returns maximum number of
// pyramidcal levels n boxes
// of given widths.
function maxLevel($boxes, $n)
{
    // Sort objects in increasing
    // order of widths
    sort($boxes);

    // Initialize result
    $ans = 1;

    // Total width of previous
    // level and total number
```

```
// of objects in previous level
$prev_width = $boxes[0];
$prev_count = 1;

// Number of object
// in current level.
$curren_count = 0;

// Width of current level.
$curren_width = 0;
for ( $i = 1; $i < $n; $i++)
{
    // Picking the object. So
    // increase current width
    // and number of object.
    $curr_width += $boxes[$i];
    $curr_count += 1;

    // If current width and number
    // of object are greater
    // than previous.
    if ($curr_width > $prev_width and
        $curr_count > $prev_count)
    {
        // Update previous width, number
        // of object on previous level.
        $prev_width = $curr_width;
        $prev_count = $curr_count;

        // Reset width of current
        // level, number of object
        // on current level.
        $curr_count = 0;
        $curr_width = 0;

        // Increment number of level.
        $ans++;
    }
}
return $ans;
}

// Driver Code
$boxes = array(10, 20, 30, 50, 60, 70);
$n = count($boxes);
echo maxLevel($boxes, $n) ;

// This code is contributed by anuj_67.
```

?>

Output :

3

Time Complexity : $O(n \log n)$.

Please see below article for more efficient solutions.

[Maximum height of triangular arrangement of array values](#)

Improved By : [vt_m](#), ChitraNayal

Source

<https://www.geeksforgeeks.org/find-maximum-height-pyramid-from-the-given-array-of-objects/>

Chapter 106

Find memory conflicts among multiple threads

Find memory conflicts among multiple threads - GeeksforGeeks

Consider a RAM organized in blocks. There are multiple processes running on the system. Every application gets below information.

(Thread T, Memory Block M, time t, R/W) which essentially tells that the thread T was using memory block M at time t and operation could be read or write.

Memory conflict is defined as –

- Multiple read operations at the same location are not cause of conflict.
- One write operation between $x+5$ to $x-5$ to location M, will be cause of conflict for a thread accessing location M at time x where x is some time in standard unit of time measurement.
- Example – If thread T1 accessed memory location M at time $x+1$ and if a thread T2 accesses location M before time $x+6$, then T1 and T2 are candidate of conflict given one of them does write operation.

You are given with the list of threads accessing memory locations, you have to find all conflicts.

Example–

Input:

(1, 512, 1, R)
(2, 432, 2, W)
(3, 512, 3, R)
(4, 932, 4, R)
(5, 512, 5, W)
(6, 932, 6, R)
(7, 835, 7, R)
(8, 432, 8, R)

Output:

```
Thread 1 & 3 conflict with thread 5
All other operations are safe.
```

We strongly recommend you to minimize your browser and try this yourself first.

The idea is to sort all threads by memory block and if memory block is same, then by time. Once we have all threads sorted, we can traverse all threads one by one. For every thread being traversed, we simply need to check previous adjacent threads of same block as threads are sorted by time.

Below is C++ implementation of this idea.

```
// C++ program to find memory conflicts among threads
#include<bits/stdc++.h>
using namespace std;

/* Structure for details of thread */
struct Thread
{
    int id, memblk, time;
    char access;
};

/* Compare function needed for sorting threads
   We first sort threads on the basis of memory block,
   If they are same, then we sort on the basis of time */
bool compare(const Thread& x, const Thread& y)
{
    if (x.memblk == y.memblk)
        return x.time < y.time;
    else return x.memblk < y.memblk;
}

// Function to print all conflicts among threads
void printConflicts(Thread arr[], int n)
{
    /* Sort the threads first by memblock, then by
       time */
    sort(arr, arr+n, compare);

    /*start from second thread till last thread*/
    for (int i = 1; i < n; i++)
    {
        /* As threads are sorted, We check further
           for conflict possibility only if there
           memblock is same*/
        if(arr[i].memblk == arr[i-1].memblk)
```

```

{

/* As threads with same block are sorted in increasing order
   of access time. So we check possibility conflict from last
   thread to all previous threads which access the same block
   of memory such that the current thread access under the
   arr[i-1].time + 5.. and if any of them does read operation
   than conflict occurs. at any point memblock becomes different
   or current thread moves out of vulnerable time of latest
   previous processed thread, the loop breaks.
*/
if (arr[i].time <= arr[i-1].time+5)
{
    int j = i-1; /* start with previous thread */

    // Find all previous conflicting threads
    while (arr[i].memblk == arr[j].memblk &&
           arr[i].time <= arr[j].time+5 &&
           j >= 0)
    {
        if (arr[i].access == 'W' || arr[j].access == 'W')
        {
            cout << "threads " << arr[j].id << " and "
                  << arr[i].id << " conflict.\n";
        }
        j--;
    }
}
}

cout << "All other operations are same\n";
}

// Driver program to test above function
int main()
{
    Thread arr[] = { {1, 512, 1, 'R'}, {2, 432, 2, 'W'},
                     {3, 512, 3, 'R'}, {4, 932, 4, 'R'},
                     {5, 512, 5, 'W'}, {6, 932, 6, 'R'},
                     {7, 835, 7, 'R'}, {8, 432, 8, 'R'}
                 };
    int n = sizeof(arr)/sizeof(arr[0]);
    printConflicts(arr, n);
    return 0;
}

```

Output:

```
threads 3 and 5 conflict.  
threads 1 and 5 conflict.  
All other operations are same
```

Time complexity: The above solution uses sorting to sort threads. Sorting can be done in $O(n\log n)$ time. Then it prints all conflicts. Printing all conflicts takes $O(n + m)$ time where m is number of conflicts. So overall time complexity is $O(n\log n + m)$.

This article is contributed by [Gaurav Ahirwar](#). Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

<https://www.geeksforgeeks.org/find-memory-conflicts-among-multiple-threads/>

Chapter 107

Find minimum difference between any two elements

Find minimum difference between any two elements - GeeksforGeeks

Given an unsorted array, find the minimum difference between any pair in given array.

Examples :

```
Input  : {1, 5, 3, 19, 18, 25};  
Output : 1  
Minimum difference is between 18 and 19
```

```
Input  : {30, 5, 20, 9};  
Output : 4  
Minimum difference is between 5 and 9
```

```
Input  : {1, 19, -4, 31, 38, 25, 100};  
Output : 5  
Minimum difference is between 1 and -4
```

Method 1 (Simple: $O(n^2)$)

A simple solution is to use two loops.

C/C++

```
// C++ implementation of simple method to find  
// minimum difference between any pair  
#include <bits/stdc++.h>  
using namespace std;  
  
// Returns minimum difference between any pair
```

```
int findMinDiff(int arr[], int n)
{
    // Initialize difference as infinite
    int diff = INT_MAX;

    // Find the min diff by comparing difference
    // of all possible pairs in given array
    for (int i=0; i<n-1; i++)
        for (int j=i+1; j<n; j++)
            if (abs(arr[i] - arr[j]) < diff)
                diff = abs(arr[i] - arr[j]);

    // Return min diff
    return diff;
}

// Driver code
int main()
{
    int arr[] = {1, 5, 3, 19, 18, 25};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Minimum difference is " << findMinDiff(arr, n);
    return 0;
}
```

Java

```
// Java implementation of simple method to find
// minimum difference between any pair

class GFG
{
    // Returns minimum difference between any pair
    static int findMinDiff(int[] arr, int n)
    {
        // Initialize difference as infinite
        int diff = Integer.MAX_VALUE;

        // Find the min diff by comparing difference
        // of all possible pairs in given array
        for (int i=0; i<n-1; i++)
            for (int j=i+1; j<n; j++)
                if (Math.abs((arr[i] - arr[j])) < diff)
                    diff = Math.abs((arr[i] - arr[j]));

        // Return min diff
        return diff;
    }
}
```

```
// Driver method to test the above function
public static void main(String[] args)
{
    int arr[] = new int[]{1, 5, 3, 19, 18, 25};
    System.out.println("Minimum difference is "+
        findMinDiff(arr, arr.length));

}
}
```

Python

```
# Python implementation of simple method to find
# minimum difference between any pair

# Returns minimum difference between any pair
def findMinDiff(arr, n):
    # Initialize difference as infinite
    diff = 10**20

    # Find the min diff by comparing difference
    # of all possible pairs in given array
    for i in range(n-1):
        for j in range(i+1,n):
            if abs(arr[i]-arr[j]) < diff:
                diff = abs(arr[i] - arr[j])

    # Return min diff
    return diff

# Driver code
arr = [1, 5, 3, 19, 18, 25]
n = len(arr)
print("Minimum difference is " + str(findMinDiff(arr, n)))

# This code is contributed by Pratik Chhajer
```

C#

```
// C# implementation of simple method to find
// minimum difference between any pair
using System;

class GFG {

    // Returns minimum difference between any pair
```

```
static int findMinDiff(int []arr, int n)
{
    // Initialize difference as infinite
    int diff = int.MaxValue;

    // Find the min diff by comparing difference
    // of all possible pairs in given array
    for (int i = 0; i < n-1; i++)
        for (int j = i+1; j < n; j++)
            if (Math.Abs((arr[i] - arr[j])) < diff)
                diff = Math.Abs((arr[i] - arr[j]));

    // Return min diff
    return diff;
}

// Driver method to test the above function
public static void Main()
{
    int []arr = new int[]{1, 5, 3, 19, 18, 25};
    Console.Write("Minimum difference is " +
                  findMinDiff(arr, arr.Length));
}
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// PHP implementation of simple
// method to find minimum
// difference between any pair

// Returns minimum difference
// between any pair
function findMinDiff($arr, $n)
{
    // Initialize difference
    // as infinite
    $diff = PHP_INT_MAX;

    // Find the min diff by comparing
    // difference of all possible
    // pairs in given array
    for ($i = 0; $i < $n - 1; $i++)
        for ($j = $i + 1; $j < $n; $j++)
```

```
if (abs($arr[$i] - $arr[$j]) < $diff)
    $diff = abs($arr[$i] - $arr[$j]);

// Return min diff
return $diff;
}

// Driver code
$arr = array(1, 5, 3, 19, 18, 25);
$n = sizeof($arr);
echo "Minimum difference is " ,
      findMinDiff($arr, $n);

// This code is contributed by ajit
?>
```

Output :

```
Minimum difference is 1
```

Method 2 (Efficient: O(n Log n)

The idea is to use sorting. Below are steps.

- 1) Sort array in ascending order. This step takes $O(n \log n)$ time.
- 2) Initialize difference as infinite. This step takes $O(1)$ time.
- 3) Compare all adjacent pairs in sorted array and keep track of minimum difference. This step takes $O(n)$ time.

Below is implementation of above idea.

C++

```
// C++ program to find minimum difference between
// any pair in an unsorted array
#include <bits/stdc++.h>
using namespace std;

// Returns minimum difference between any pair
int findMinDiff(int arr[], int n)
{
    // Sort array in non-decreasing order
    sort(arr, arr+n);

    // Initialize difference as infinite
    int diff = INT_MAX;

    // Find the min diff by comparing adjacent
    // pairs in sorted array
```

```
for (int i=0; i<n-1; i++)
    if (arr[i+1] - arr[i] < diff)
        diff = arr[i+1] - arr[i];

    // Return min diff
    return diff;
}

// Driver code
int main()
{
    int arr[] = {1, 5, 3, 19, 18, 25};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Minimum difference is " << findMinDiff(arr, n);
    return 0;
}
```

Java

```
// Java program to find minimum difference between
// any pair in an unsorted array

import java.util.Arrays;

class GFG
{
    // Returns minimum difference between any pair
    static int findMinDiff(int[] arr, int n)
    {
        // Sort array in non-decreasing order
        Arrays.sort(arr);

        // Initialize difference as infinite
        int diff = Integer.MAX_VALUE;

        // Find the min diff by comparing adjacent
        // pairs in sorted array
        for (int i=0; i<n-1; i++)
            if (arr[i+1] - arr[i] < diff)
                diff = arr[i+1] - arr[i];

        // Return min diff
        return diff;
    }

    // Driver method to test the above function
    public static void main(String[] args)
    {
```

```
    int arr[] = new int[]{1, 5, 3, 19, 18, 25};
    System.out.println("Minimum difference is "+
                        findMinDiff(arr, arr.length));

}
}
```

Python

```
# Python program to find minimum difference between
# any pair in an unsorted array

# Returns minimum difference between any pair
def findMinDiff(arr, n):

    # Sort array in non-decreasing order
    arr = sorted(arr)

    # Initialize difference as infinite
    diff = 10**20

    # Find the min diff by comparing adjacent
    # pairs in sorted array
    for i in range(n-1):
        if arr[i+1] - arr[i] < diff:
            diff = arr[i+1] - arr[i]

    # Return min diff
    return diff

# Driver code
arr = [1, 5, 3, 19, 18, 25]
n = len(arr)
print("Minimum difference is " + str(findMinDiff(arr, n)))

# This code is contributed by Pratik Chhajer
```

C#

```
// C# program to find minimum
// difference between any pair
// in an unsorted array
using System;

class GFG
{
    // Returns minimum difference
```

```
// between any pair
static int findMinDiff(int[] arr,
                      int n)
{
    // Sort array in
    // non-decreasing order
    Array.Sort(arr);

    // Initialize difference
    // as infinite
    int diff = int.MaxValue;

    // Find the min diff by
    // comparing adjacent pairs
    // in sorted array
    for (int i = 0; i < n - 1; i++)
        if (arr[i + 1] - arr[i] < diff)
            diff = arr[i + 1] - arr[i];

    // Return min diff
    return diff;
}

// Driver Code
public static void Main()
{
    int []arr = new int[]{1, 5, 3, 19, 18, 25};
    Console.WriteLine("Minimum difference is " +
                      findMinDiff(arr, arr.Length));

}
}

//This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP program to find minimum
// difference between any pair
// in an unsorted array

// Returns minimum difference
// between any pair
function findMinDiff($arr, $n)
{

// Sort array in
```

```
// non-decreasing order
sort($arr);

// Initialize difference
// as infinite
$diff = PHP_INT_MAX;

// Find the min diff by
// comparing adjacent
// pairs in sorted array
for ($i = 0; $i < $n - 1; $i++)
    if ($arr[$i + 1] - $arr[$i] < $diff)
        $diff = $arr[$i + 1] - $arr[$i];

// Return min diff
return $diff;
}

// Driver code
$arr = array(1, 5, 3, 19, 18, 25);
$n = sizeof($arr);
echo "Minimum difference is " ,
      findMinDiff($arr, $n);

// This code is contributed ajit
?>
```

Output :

```
Minimum difference is 1
```

This article is contributed by **Harshit Agrawal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : [nitin mittal](#), [vt_m](#), [jit_t](#)

Source

<https://www.geeksforgeeks.org/find-minimum-difference-pair/>

Chapter 108

Find missing elements of a range

Find missing elements of a range - GeeksforGeeks

Given an array arr[0..n-1] of distinct elements and a range [low, high], find all numbers that are in range, but not in array. The missing elements should be printed in sorted order.

Examples:

```
Input: arr[] = {10, 12, 11, 15},  
       low = 10, hight = 15  
Output: 13, 14
```

```
Input: arr[] = {1, 14, 11, 51, 15},  
       low = 50, hight = 55  
Output: 50, 52, 53, 54
```

There can be following two approaches to solve the problem.

1. **Use Sorting :** Sort the array, then do binary search for ‘low’. Once location of low is found, start traversing array from that location and keep printing all missing numbers.
C++

```
// A sorting based C++ program to find missing  
// elements from an array  
#include<bits/stdc++.h>  
using namespace std;  
  
// Print all elements of range [low, high] that  
// are not present in arr[0..n-1]
```

```
void printMissing(int arr[], int n, int low,
                  int high)
{
    // Sort the array
    sort(arr, arr+n);

    // Do binary search for 'low' in sorted
    // array and find index of first element
    // which either equal to or greater than
    // low.
    int *ptr = lower_bound(arr, arr+n, low);
    int index = ptr - arr;

    // Start from the found index and linearly
    // search every range element x after this
    // index in arr[]
    int i = index, x = low;
    while (i < n && x<=high)
    {
        // If x doesn't math with current element
        // print it
        if (arr[i] != x)
            cout << x << " ";

        // If x matches, move to next element in arr[]
        else
            i++;

        // Move to next element in range [low, high]
        x++;
    }

    // Print range elements thar are greater than the
    // last element of sorted array.
    while (x <= high)
        cout << x++ << " ";
}

// Driver program
int main()
{
    int arr[] = {1, 3, 5, 4};
    int n = sizeof(arr)/sizeof(arr[0]);
    int low = 1, high = 10;
    printMissing(arr, n, low, high);
    return 0;
}
```

Java

```
// A sorting based Java program to find missing
// elements from an array

import java.util.Arrays;

public class PrintMissing
{
    // Print all elements of range [low, high] that
    // are not present in arr[0..n-1]
    static void printMissing(int ar[], int low, int high)
    {
        Arrays.sort(ar);
        // Do binary search for 'low' in sorted
        // array and find index of first element
        // which either equal to or greater than
        // low.
        int index = ceilindex(ar, low, 0, ar.length - 1);
        int x = low;

        // Start from the found index and linearly
        // search every range element x after this
        // index in arr[]
        while (index < ar.length && x <= high)
        {
            // If x doesn't match with current element
            // print it
            if (ar[index] != x)
            {
                System.out.print(x + " ");
            }

            // If x matches, move to next element in arr[]
            else
                index++;
            // Move to next element in range [low, high]
            x++;
        }

        // Print range elements that are greater than the
        // last element of sorted array.
        while (x <= high)
        {
            System.out.print(x + " ");
            x++;
        }
    }
}
```

```
}

// Utility function to find ceil index of given element
static int ceilindex(int ar[], int val, int low, int high)
{

    if (val < ar[0])
        return 0;
    if (val > ar[ar.length - 1])
        return ar.length;

    int mid = (low + high) / 2;
    if (ar[mid] == val)
        return mid;
    if (ar[mid] < val)
    {
        if (mid + 1 < high && ar[mid + 1] >= val)
            return mid + 1;
        return ceilindex(ar, val, mid + 1, high);
    }
    else
    {
        if (mid - 1 >= low && ar[mid - 1] < val)
            return mid;
        return ceilindex(ar, val, low, mid - 1);
    }
}

// Driver program to test above function
public static void main(String[] args)
{
    int arr[] = { 1, 3, 5, 4 };
    int low = 1, high = 10;
    printMissing(arr, low, high);
}
}

// This code is contributed by Rishabh Mahrsee
```

Output:

2 6 7 8 9 10

2. **Use Hashing :** Create a hash table and insert all array items into the hash table. Once all items are in hash table, traverse through the range and print all missing elements.

C++

```
// A hashing based C++ program to find missing
// elements from an array
#include<bits/stdc++.h>
using namespace std;

// Print all elements of range [low, high] that
// are not present in arr[0..n-1]
void printMissing(int arr[], int n, int low,
                  int high)
{
    // Insert all elements of arr[] in set
    unordered_set<int> s;
    for (int i=0; i<n; i++)
        s.insert(arr[i]);

    // Traverse through the range and print all
    // missing elements
    for (int x=low; x<=high; x++)
        if (s.find(x) == s.end())
            cout << x << " ";
}

// Driver program
int main()
{
    int arr[] = {1, 3, 5, 4};
    int n = sizeof(arr)/sizeof(arr[0]);
    int low = 1, high = 10;
    printMissing(arr, n, low, high);
    return 0;
}
```

Java

```
// A hashing based Java program to find missing
// elements from an array

import java.util.Arrays;
import java.util.HashSet;

public class Print
{
    // Print all elements of range [low, high] that
    // are not present in arr[0..n-1]
    static void printMissing(int ar[], int low, int high)
    {
        HashSet<Integer> hs = new HashSet<>();
```

```
// Insert all elements of arr[] in set
for (int i = 0; i < ar.length; i++)
    hs.add(ar[i]);

// Traverse through the range and print all
// missing elements
for (int i = low; i <= high; i++)
{
    if (!hs.contains(i))
    {
        System.out.print(i + " ");
    }
}
}

// Driver program to test above function
public static void main(String[] args)
{
    int arr[] = { 1, 3, 5, 4 };
    int low = 1, high = 10;
    printMissing(arr, low, high);
}
}

// This code is contributed by Rishabh Mahrsee
```

Output:

2 6 7 8 9 10

Which approach is better?

Time complexity of first approach is $O(n \log n + k)$ where k is number of missing elements (Note that k may be more than $n \log n$ if array is small and range is big)

Time complexity of second solution is $O(n + (\text{high-low}+1))$.

If the given array has almost elements of range i.e., n is close to value of $(\text{high-low}+1)$, then second approach is definitely better as there is no $\log n$ factor. But if n is much smaller than range, then first approach is better as it doesn't require extra space for hashing. We can also modify first approach to print adjacent missing elements as range to save time. For example if 50, 51, 52, 53, 54, 59 are missing, we can print them as 50-54, 59 in first method. And if printing this way is allowed, the first approach takes only $O(n \log n)$ time.

This article is contributed by **Piyush Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/find-missing-elements-of-a-range/>

Chapter 109

Find number of pairs (x, y) in an array such that $x^y > y^x$

Find number of pairs (x, y) in an array such that $x^y > y^x$ - GeeksforGeeks

Given two arrays X[] and Y[] of positive integers, find number of pairs such that $x^y > y^x$ where x is an element from X[] and y is an element from Y[].

Examples:

```
Input: X[] = {2, 1, 6}, Y = {1, 5}
Output: 3
// There are total 3 pairs where pow(x, y) is greater than pow(y, x)
// Pairs are (2, 1), (2, 5) and (6, 1)
```

```
Input: X[] = {10, 19, 18}, Y[] = {11, 15, 9};
Output: 2
// There are total 2 pairs where pow(x, y) is greater than pow(y, x)
// Pairs are (10, 11) and (10, 15)
```

The **brute force solution** is to consider each element of X[] and Y[], and check whether the given condition satisfies or not. Time Complexity of this solution is $O(m*n)$ where m and n are sizes of given arrays.

Following is C++ code based on brute force solution.

```
int countPairsBruteForce(int X[], int Y[], int m, int n)
{
    int ans = 0;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
```

```

    if (pow(X[i], Y[j]) > pow(Y[j], X[i]))
        ans++;
    return ans;
}

```

Efficient Solution:

The problem can be solved in $O(n \log n + m \log n)$ time. The trick here is, if $y > x$ then $x^y > y^x$ with some exceptions. Following are simple steps based on this trick.

- 1) Sort array $Y[]$.
- 2) For every x in $X[]$, find the index idx of smallest number greater than x (also called ceil of x) in $Y[]$ using binary search or we can use the inbuilt function `upper_bound()` in algorithm library.
- 3) All the numbers after idx satisfy the relation so just add $(n - idx)$ to the count.

Base Cases and Exceptions:

Following are exceptions for x from $X[]$ and y from $Y[]$

If $x = 0$, then the count of pairs for this x is 0.

If $x = 1$, then the count of pairs for this x is equal to count of 0s in $Y[]$.

The following cases must be handled separately as they don't follow the general rule that x smaller than y means x^y is greater than y^x .

- a) $x = 2, y = 3$ or 4
- b) $x = 3, y = 2$

Note that the case where $x = 4$ and $y = 2$ is not there

Following diagram shows all exceptions in tabular form. The value 1 indicates that the corresponding (x, y) form a valid pair.

	0	1	2	3	4
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	1
4	1	1	0	0	0

Following is implementation. In the following implementation, we pre-process the Y array and count 0, 1, 2, 3 and 4 in it, so that we can handle all exceptions in constant time. The array $NoOfY[]$ is used to store the counts.

C++

```

// Java program to finds number of pairs (x, y)
// in an array such that x^y > y^x

```

```
#include<iostream>
#include<algorithm>
using namespace std;

// This function return count of pairs with x as one element
// of the pair. It mainly looks for all values in Y[] where
//  $x^Y[i] > Y[i]^x$ 
int count(int x, int Y[], int n, int NoOfY[])
{
    // If x is 0, then there cannot be any value in Y such that
    //  $x^Y[i] > Y[i]^x$ 
    if (x == 0) return 0;

    // If x is 1, then the number of pairs is equal to number of
    // zeroes in Y[]
    if (x == 1) return NoOfY[0];

    // Find number of elements in Y[] with values greater than x
    // upper_bound() gets address of first greater element in Y[0..n-1]
    int* idx = upper_bound(Y, Y + n, x);
    int ans = (Y + n) - idx;

    // If we have reached here, then x must be greater than 1,
    // increase number of pairs for y=0 and y=1
    ans += (NoOfY[0] + NoOfY[1]);

    // Decrease number of pairs for x=2 and (y=4 or y=3)
    if (x == 2) ans -= (NoOfY[3] + NoOfY[4]);

    // Increase number of pairs for x=3 and y=2
    if (x == 3) ans += NoOfY[2];

    return ans;
}

// The main function that returns count of pairs (x, y) such that
// x belongs to X[], y belongs to Y[] and  $x^y > y^x$ 
int countPairs(int X[], int Y[], int m, int n)
{
    // To store counts of 0, 1, 2, 3 and 4 in array Y
    int NoOfY[5] = {0};
    for (int i = 0; i < n; i++)
        if (Y[i] < 5)
            NoOfY[Y[i]]++;

    // Sort Y[] so that we can do binary search in it
    sort(Y, Y + n);
```

```
int total_pairs = 0; // Initialize result

// Take every element of X and count pairs with it
for (int i=0; i<m; i++)
    total_pairs += count(X[i], Y, n, NoOfY);

return total_pairs;
}

// Driver program to test above functions
int main()
{
    int X[] = {2, 1, 6};
    int Y[] = {1, 5};

    int m = sizeof(X)/sizeof(X[0]);
    int n = sizeof(Y)/sizeof(Y[0]);

    cout << "Total pairs = " << countPairs(X, Y, m, n);

    return 0;
}
```

Java

```
// Java program to finds number of pairs (x, y)
// in an array such that  $x^y > y^x$ 

import java.util.Arrays;

class Test
{
    // This function return count of pairs with x as one element
    // of the pair. It mainly looks for all values in Y[] where
    //  $x^Y[i] > Y[i]^x$ 
    static int count(int x, int Y[], int n, int NoOfY[])
    {
        // If x is 0, then there cannot be any value in Y such that
        //  $x^Y[i] > Y[i]^x$ 
        if (x == 0) return 0;

        // If x is 1, then the number of pais is equal to number of
        // zeroes in Y[]
        if (x == 1) return NoOfY[0];

        // Find number of elements in Y[] with values greater than x
        // getting upperbound of x with binary search
```

```
int idx = Arrays.binarySearch(Y, x);
int ans;
if(idx < 0){
    idx = Math.abs(idx+1);
    ans = Y.length - idx;
}
else{
    while (Y[idx]==x) {
        idx++;
    }
    ans = Y.length - idx;
}

// If we have reached here, then x must be greater than 1,
// increase number of pairs for y=0 and y=1
ans += (NoOfY[0] + NoOfY[1]);

// Decrease number of pairs for x=2 and (y=4 or y=3)
if (x == 2) ans -= (NoOfY[3] + NoOfY[4]);

// Increase number of pairs for x=3 and y=2
if (x == 3) ans += NoOfY[2];

return ans;
}

// The main function that returns count of pairs (x, y) such that
// x belongs to X[], y belongs to Y[] and  $x^y > y^x$ 
static int countPairs(int X[], int Y[], int m, int n)
{
    // To store counts of 0, 1, 2, 3 and 4 in array Y
    int NoOfY[] = new int[5];
    for (int i = 0; i < n; i++)
        if (Y[i] < 5)
            NoOfY[Y[i]]++;

    // Sort Y[] so that we can do binary search in it
    Arrays.sort(Y);

    int total_pairs = 0; // Initialize result

    // Take every element of X and count pairs with it
    for (int i=0; i<m; i++)
        total_pairs += count(X[i], Y, n, NoOfY);

    return total_pairs;
}
```

```
// Driver method
public static void main(String args[])
{
    int X[] = {2, 1, 6};
    int Y[] = {1, 5};

    System.out.println("Total pairs = " + countPairs(X, Y, X.length, Y.length));
}
```

C#

```
// C# program to finds number of pairs (x, y)
// in an array such that  $x^y > y^x$ 
using System;

class GFG {

    // This function return count of pairs
    // with x as one element of the pair.
    // It mainly looks for all values in Y[]
    // where  $x^Y[i] > Y[i]^x$ 
    static int count(int x, int[] Y, int n, int[] NoOfY)
    {
        // If x is 0, then there cannot be any
        // value in Y such that  $x^Y[i] > Y[i]^x$ 
        if (x == 0)
            return 0;

        // If x is 1, then the number of pairs
        // is equal to number of zeroes in Y[]
        if (x == 1)
            return NoOfY[0];

        // Find number of elements in Y[] with
        // values greater than x getting
        // upperbound of x with binary search
        int idx = Array.BinarySearch(Y, x);
        int ans;
        if (idx < 0) {
            idx = Math.Abs(idx + 1);
            ans = Y.Length - idx;
        }

        else {
            while (Y[idx] == x) {
                idx++;
            }
        }
    }
}
```

```
ans = Y.Length - idx;
}

// If we have reached here, then x
// must be greater than 1, increase
// number of pairs for y = 0 and y = 1
ans += (NoOfY[0] + NoOfY[1]);

// Decrease number of pairs
// for x = 2 and (y = 4 or y = 3)
if (x == 2)
    ans -= (NoOfY[3] + NoOfY[4]);

// Increase number of pairs for x = 3 and y = 2
if (x == 3)
    ans += NoOfY[2];

return ans;
}

// The main function that returns count
// of pairs (x, y) such that x belongs
// to X[], y belongs to Y[] and  $x^y > y^x$ 
static int countPairs(int[] X, int[] Y, int m, int n)
{
    // To store counts of 0, 1, 2, 3 and 4 in array Y
    int[] NoOfY = new int[5];
    for (int i = 0; i < n; i++)
        if (Y[i] < 5)
            NoOfY[Y[i]]++;

    // Sort Y[] so that we can do binary search in it
    Array.Sort(Y);

    int total_pairs = 0; // Initialize result

    // Take every element of X and count pairs with it
    for (int i = 0; i < m; i++)
        total_pairs += count(X[i], Y, n, NoOfY);

    return total_pairs;
}

// Driver method
public static void Main()
{
    int[] X = { 2, 1, 6 };
    int[] Y = { 1, 5 };
```

```
Console.WriteLine("Total pairs = " +
                  countPairs(X, Y, X.Length, Y.Length));
}
}

// This code is contributed by Sam007
```

Output:

```
Total pairs = 3
```

Time Complexity : Let m and n be the sizes of arrays X[] and Y[] respectively. The sort step takes O(nLogn) time. Then every element of X[] is searched in Y[] using binary search. This step takes O(mLogn) time. Overall time complexity is O(nLogn + mLogn).

This article is contributed by **Shubham Mittal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

<https://www.geeksforgeeks.org/find-number-pairs-xy-yx/>

Chapter 110

Find number of pairs in an array such that their XOR is 0

Find number of pairs in an array such that their XOR is 0 - GeeksforGeeks

Given an array $A[]$ of size N. Find the number of pairs (i, j) such that $A_i \text{ XOR } A_j = 0$, and $1 \leq i < j \leq N$.

Examples :

Input : $A[] = \{1, 3, 4, 1, 4\}$

Output : 2

Explanation : Index (0, 3) and (2, 4)

Input : $A[] = \{2, 2, 2\}$

Output : 3

First Approach : Sorting

$A_i \text{ XOR } A_j = 0$ is only satisfied when $A_i = A_j$. Therefore, we will first sort the array and then count the frequency of each element. By combinatorics, we can observe that if frequency of some element is $Count$ then, it will contribute $\frac{Count \times (Count - 1)}{2}$ to the answer.

Below is the implementation of above approach :

C++

```
// C++ program to find number
```

```
// of pairs in an array such
// that their XOR is 0
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the
// count
int calculate(int a[], int n)
{
    // Sorting the list using
    // built in function
    sort(a, a + n);

    int count = 1;
    int answer = 0;

    // Traversing through the
    // elements
    for (int i = 1; i < n; i++) {

        if (a[i] == a[i - 1]){

            // Counting frequency of each
            // elements
            count += 1;

        }
        else
        {
            // Adding the contribution of
            // the frequency to the answer
            answer = answer + (count * (count - 1)) / 2;
            count = 1;
        }
    }

    answer = answer + (count * (count - 1)) / 2;

    return answer;
}

// Driver Code
int main()
{

    int a[] = { 1, 2, 1, 2, 4 };
    int n = sizeof(a) / sizeof(a[0]);
}
```

```
// Print the count
cout << calculate(a, n);
return 0;
}

// This article is contributed by Sahil_Bansall.
```

Java

```
// Java program to find number
// of pairs in an array such
// that their XOR is 0
import java.util.*;

class GFG
{
    // Function to calculate
    // the count
    static int calculate(int a[], int n)
    {
        // Sorting the list using
        // built in function
        Arrays.sort(a);

        int count = 1;
        int answer = 0;

        // Traversing through the
        // elements
        for (int i = 1; i < n; i++)
        {

            if (a[i] == a[i - 1])
            {
                // Counting frequency of each
                // elements
                count += 1;

            }
            else
            {
                // Adding the contribution of
                // the frequency to the answer
                answer = answer + (count * (count - 1)) / 2;
                count = 1;
            }
        }
    }
}
```

```
        answer = answer + (count * (count - 1)) / 2;

    return answer;
}

// Driver Code
public static void main (String[] args)
{
    int a[] = { 1, 2, 1, 2, 4 };
    int n = a.length;

    // Print the count
    System.out.println(calculate(a, n));
}
}

// This code is contributed by Ansu Kumari.
```

Python3

```
# Python3 program to find number of pairs
# in an array such that their XOR is 0

# Function to calculate the count
def calculate(a) :

    # Sorting the list using
    # built in function
    a.sort()

    count = 1
    answer = 0

    # Traversing through the elements
    for i in range(1, len(a)) :

        if a[i] == a[i - 1] :

            # Counting frequency of each elements
            count += 1

        else :

            # Adding the contribution of
            # the frequency to the answer
            answer = answer + count * (count - 1) // 2
            count = 1
```

```
answer = answer + count * (count - 1) // 2

return answer

# Driver Code
if __name__ == '__main__':
    a = [1, 2, 1, 2, 4]

    # Print the count
    print(calculate(a))
```

C#

```
// Java program to find number
// of pairs in an array such
// that their XOR is 0
using System;

class GFG
{
    // Function to calculate
    // the count
    static int calculate(int []a, int n)
    {
        // Sorting the list using
        // built in function
        Array.Sort(a);

        int count = 1;
        int answer = 0;

        // Traversing through the
        // elements
        for (int i = 1; i < n; i++)
        {

            if (a[i] == a[i - 1])
            {
                // Counting frequency of each
                // elements
                count += 1;

            }
            else
            {
                // Adding the contribution of
```

```
// the frequency to the answer
answer = answer + (count * (count - 1)) / 2;
count = 1;
}
}

answer = answer + (count * (count - 1)) / 2;

return answer;
}

// Driver Code
public static void Main ()
{
    int []a = { 1, 2, 1, 2, 4 };
    int n = a.Length;

    // Print the count
    Console.WriteLine(calculate(a, n));
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find number
// of pairs in an array such
// that their XOR is 0

// Function to calculate
// the count
function calculate($a, $n)
{

    // Sorting the list using
    // built in function
    sort($a);

    $count = 1;
    $answer = 0;

    // Traversing through the
    // elements
    for ($i = 1; $i < $n; $i++)
    {
```

```
if ($a[$i] == $a[$i - 1])
{
    // Counting frequency of
    // each elements
    $count += 1;

}
else
{
    // Adding the contribution of
    // the frequency to the answer
    $answer = $answer + ($count *
        ($count - 1)) / 2;
    $count = 1;
}
}

$answer = $answer + ($count *
    ($count - 1)) / 2;

return $answer;
}

// Driver Code
$a = array(1, 2, 1, 2, 4);
$n = count($a);

// Print the count
echo calculate($a, $n);

// This code is contributed by anuj_67.
?>
```

Output :

2

Time Complexity : $O(N \log N)$

Second Approach : Hashing (Index Mapping)

Solution is handy, if we can count the frequency of each element in the array. Index mapping technique can be used to count the frequency of each element.

Below is the implementation of above approach :

C++

```
// C++ program to find number of pairs
// in an array such that their XOR is 0
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the answer
int calculate(int a[]){
    // Finding the maximum of the array
    int *maximum = max_element(a, a + 5);

    // Creating frequency array
    // With initial value 0
    int frequency[*maximum + 1] = {0};

    // Traversing through the array
    for(int i = 0; i < (*maximum)+1; i++)
    {
        // Counting frequency
        frequency[a[i]] += 1;
    }
    int answer = 0;

    // Traversing through the frequency array
    for(int i = 0; i < (*maximum)+1; i++)
    {
        // Calculating answer
        answer = answer + frequency[i] * (frequency[i] - 1) ;
    }
    return answer/2;
}

// Driver Code
int main()
{
    int a[] = {1, 2, 1, 2, 4};

    // Function calling
    cout << (calculate(a));
}

// This code is contributed by Smitha
```

Python 3

```
# Python3 program to find number of pairs
# in an array such that their XOR is 0

# Function to calculate the answer
def calculate(a) :

    # Finding the maximum of the array
    maximum = max(a)

    # Creating frequency array
    # With initial value 0
    frequency = [0 for x in range(maximum + 1)]

    # Traversing through the array
    for i in a :

        # Counting frequency
        frequency[i] += 1

    answer = 0

    # Traversing through the frequency array
    for i in frequency :

        # Calculating answer
        answer = answer + i * (i - 1) // 2

    return answer

# Driver Code
a = [1, 2, 1, 2, 4]
print(calculate(a))
```

PHP

Output :

2

Time Complexity : O(N)

Note : Index Mapping method can only be used when the numbers in the array are not large. In such cases, sorting method can be used.

Improved By : [vt_m, Smitha Dinesh Semwal](#)

Source

<https://www.geeksforgeeks.org/find-number-pairs-array-xor-0/>

Chapter 111

Find pair with greatest product in array

Find pair with greatest product in array - GeeksforGeeks

Given an array of n elements, the task is to find the greatest number such that it is product of two elements of given array. If no such element exists, print -1. Elements are within the range of 1 to 10^5 .

Examples :

Input : arr[] = {10, 3, 5, 30, 35}
Output: 30
Explanation: 30 is the product of 10 and 3.

Input : arr[] = {2, 5, 7, 8}
Output: -1
Explanation: Since, no such element exists.

Input : arr[] = {10, 2, 4, 30, 35}
Output: -1

Input : arr[] = {10, 2, 2, 4, 30, 35}
Output: 4

Input : arr[] = {17, 2, 1, 35, 30}
Output : 35

A **naive approach** is to pick an element and then check for each pair product if equal to that number and update the max if the number is maximum, repeat until whole array gets traversed takes $O(n^3)$ time.

C++

```
// C++ program to find a pair with product
// in given array.
#include<bits/stdc++.h>
using namespace std;

// Function to find greatest number that us
int findGreatest( int arr[] , int n)
{
    int result = -1;
    for (int i = 0; i < n ; i++)
        for (int j = 0; j < n-1; j++)
            for (int k = j+1 ; k < n ; k++)
                if (arr[j] * arr[k] == arr[i])
                    result = max(result, arr[i]);
    return result;
}

// Driver code
int main()
{
    // Your C++ Code
    int arr[] = {30, 10, 9, 3, 35};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << findGreatest(arr, n);

    return 0;
}
```

Java

```
// Java program to find a pair
// with product in given array.
import java.io.*;

class GFG{

static int findGreatest( int []arr , int n)
{
    int result = -1;
    for (int i = 0; i < n ; i++)
        for (int j = 0; j < n-1; j++)
            for (int k = j+1 ; k < n ; k++)
                if (arr[j] * arr[k] == arr[i])
                    result = Math.max(result, arr[i]);
    return result;
}
```

```
// Driver code
static public void main (String[] args)
{
    int []arr = {30, 10, 9, 3, 35};
    int n = arr.length;

    System.out.println(findGreatest(arr, n));
}
}

//This code is contributed by vt_m.
```

C#

```
// C# program to find a pair with product
// in given array.
using System;

class GFG{

static int findGreatest( int []arr , int n)
{
    int result = -1;
    for (int i = 0; i < n ; i++)
        for (int j = 0; j < n-1; j++)
            for (int k = j+1 ; k < n ; k++)
                if (arr[j] * arr[k] == arr[i])
                    result = Math.Max(result, arr[i]);
    return result;
}

// Driver code
static public void Main ()
{
    int []arr = {30, 10, 9, 3, 35};
    int n = arr.Length;

    Console.WriteLine(findGreatest(arr, n));
}
}

//This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find a pair
```

```
// with product in given array.

// Function to find
// greatest number
function findGreatest($arr , $n)
{
    $result = -1;
    for ($i = 0; $i < $n ; $i++)
        for ($j = 0; $j < $n-1; $j++)
            for ($k = $j+1 ; $k < $n ; $k++)
                if ($arr[$j] * $arr[$k] == $arr[$i])
                    $result = max($result, $arr[$i]);
    return $result;
}

// Driver code
$arr = array(30, 10, 9, 3, 35);
$n = count($arr);

echo findGreatest($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output :

30

An **efficient method** follows below implementation:-

1. Create an empty hash table and store all array elements in it.
2. Sort the array in ascending order.
3. Pick elements one by one from end of the array.
4. And check if there exists a pair whose product is equal to that number. In this efficiency can be achieved. The idea is to reach till \sqrt{n} of that number. If we don't get the pair till \sqrt{n} that means no such pair exists. We use hash table to make sure that we can find other element of pair in $O(1)$ time.
5. Repeat steps 2 to 3 until we get the element or whole array gets traversed.

Below is C++ implementation.

C++

```
// C++ program to find the largest product number
#include<bits/stdc++.h>
using namespace std;

// Function to find greatest number
int findGreatest(int arr[], int n)
{
    // Store occurrences of all elements in hash
    // array
    unordered_map<int, int> m;
    for (int i = 0 ; i < n; i++)
        m[arr[i]]++;

    // Sort the array and traverse all elements from
    // end.
    sort(arr, arr+n);

    for (int i=n-1; i>1; i--)
    {
        // For every element, check if there is another
        // element which divides it.
        for (int j=0; j<i && arr[j]<=sqrt(arr[i]); j++)
        {
            if (arr[i] % arr[j] == 0)
            {
                int result = arr[i]/arr[j];

                // Check if the result value exists in array
                // or not if yes the return arr[i]
                if (result != arr[j] && m[result] > 0)
                    return arr[i];

                // To handle the case like arr[i] = 4 and
                // arr[j] = 2
                else if (result == arr[j] && m[result] > 1)
                    return arr[i];
            }
        }
    }
    return -1;
}

// Drivers code
int main()
{
    int arr[] = {17, 2, 1, 15, 30};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << findGreatest(arr, n);
```

```
    return 0;  
}
```

Output :

30

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-pair-with-greatest-product-in-array/>

Chapter 112

Find shortest unique prefix for every word in a given list Set 2 (Using Sorting)

Find shortest unique prefix for every word in a given list Set 2 (Using Sorting) - GeeksforGeeks

Given an array of words, find all shortest unique prefixes to represent each word in the given array. Assume that no word is a prefix of another. Output the shortest unique prefixes in sorted order.

```
Input : {"zebra", "dog", "duck", "dove"}  
Output : z, dog, dov, du  
Explanation: dog => dog  
            dove = dov  
            duck = du  
            z    => zebra
```

```
Input: {"geeksgeeks", "geeksquiz", "geeksforgeeks"}  
Output: geeksf, geeksg, geeksq
```

We have discussed a Trie based approach in below post.

[Find shortest unique prefix for every word in a given list Set 1 \(Using Trie\)](#)

In this post a sorting based approach is discussed. On comparing the string with 2 other most similar strings in the array, we can find its shortest unique prefix. For example, if we sort the array {"zebra", "dog", "duck", "dove"}, we get {"dog", "dove", "duck", "zebra"}. The shortest unique prefix for the string "dove" can be found as:

Compare "dove" to "dog" -> unique prefix for dove is "dov"
Compare "dove" to "duck" -> unique prefix for dove is "do"

Now, the shortest unique prefix for “dove” is the one with greater length between “dov” and “do”. So, it is “dov”.

The shortest unique prefix for first and last string can be found by comparing them with only 1 most similar neighbor on right and left, respectively.

We can sort the array of strings and keep on doing this for every string of the array.

```
// Java program to print shortest unique prefixes
// for every word.
import java.io.*;
import java.util.*;

class GFG
{
    public String[] uniquePrefix(String[] a)
    {
        int size = a.length;

        /* create an array to store the results */
        String[] res = new String[size];

        /* sort the array of strings */
        Arrays.sort(a);

        /* compare the first string with its only right
           neighbor */
        int j = 0;
        while (j < Math.min(a[0].length()-1, a[1].length()-1))
        {
            if (a[0].charAt(j)==a[1].charAt(j))
                j++;
            else
                break;
        }

        int ind = 0;
        res[ind++] = a[0].substring(0, j+1);

        /* Store the unique prefix of a[1] from its left neighbor */
        String temp_prefix = a[1].substring(0, j+1);
        for (int i = 1; i < size-1; i++)
        {
            /* compute common prefix of a[i] unique from
               its right neighbor */
            j = 0;
            while (j < Math.min( a[i].length()-1, a[i+1].length()-1 ))
            {
                if (a[i].charAt(j) == a[i+1].charAt(j))
```

```
        j++;
    else
        break;
}

String new_prefix = a[i].substring(0, j+1);

/* compare the new prefix with previous prefix */
if (temp_prefix.length() > new_prefix.length() )
    res[ind++] = temp_prefix;
else
    res[ind++] = new_prefix;

/* store the prefix of a[i+1] unique from its
   left neighbour */
temp_prefix = a[i+1].substring(0, j+1);
}

/* compute the unique prefix for the last string
   in sorted array */
j = 0;
String sec_last = a[size-2] ;

String last = a[size-1];

while (j < Math.min( sec_last.length()-1, last.length()-1))
{
    if (sec_last.charAt(j) == last.charAt(j))
        j++;
    else
        break;
}

res[ind] = last.substring(0, j+1);
return res;
}

/* Driver Function to test other function */
public static void main(String[] args)
{
    GFG gfg = new GFG();

    String[] input = {"zebra", "dog", "duck", "dove"};

    String[] output = gfg.uniquePrefix(input);
    System.out.println( "The shortest unique prefixes" +
                        " in sorted order are :");
}
```

```
        for (int i=0; i < output.length; i++)
            System.out.print( output[i] + " ");
    }
}
```

Output:

```
The shortest unique prefixes in sorted order are :
dog dov du z
```

If we want to output the prefixes as the order of strings in the input array, we can store the string and its corresponding index in the hashmap. While adding the prefix to result array, we can get the index of the corresponding string from hashmap and add the prefix to that index.

```
// Java program to print shortest unique prefixes
// for every word in order of appearance of words
import java.io.*;
import java.util.*;

class GFG
{
    public String[] uniquePrefix(String[] a)
    {
        int size = a.length;

        /* create an array to store the results */
        String[] res = new String[size];

        Arrays.fill(res, "");

        /* hashmap to store the indexes */
        HashMap<String, Integer> hm =
            new HashMap<String, Integer>();

        for (int i = 0; i < size; i++)
            hm.put(a[i], i);

        /* sort the array of strings */
        Arrays.sort(a);

        /* compare the first string with its only right neighbor */
        int j = 0;
        while (j < Math.min(a[0].length()-1, a[1].length()-1))
        {
            if (a[0].charAt(j) == a[1].charAt(j))
```

```
        j++;
    else
        break;
}

/* get the index of a[0] from HashMap */
res[ hm.get(a[0]) ] = a[0].substring(0, j+1);

/* Store the unique prefix of a[1] from its
   left neighbor */
String temp_prefix= a[1].substring(0, j+1);

for (int i = 1; i < size-1; i++)
{
    /* compute prefix of a[i] unique from its right neighbor */
    j = 0;
    while (j < Math.min(a[i].length()-1, a[i+1].length()-1))
    {
        if (a[i].charAt(j) == a[i+1].charAt(j) )
            j++;
        else
            break;
    }

    String new_prefix = a[i].substring(0, j+1);

    /* compare the new prefix with previous prefix */
    if (temp_prefix.length() > new_prefix.length() )
        res[ hm.get(a[i]) ] = temp_prefix;
    else
        res[ hm.get(a[i]) ] = new_prefix;

    /* store the prefix of a[i+1] unique from its
       left neighbour */
    temp_prefix = a[i+1].substring(0, j+1);
}

/* compute the unique prefix for the last string
   in sorted array */
String sec_last = a[size-2];
String last = a[size-1];

j = 0;
while (j < Math.min( sec_last.length()-1, last.length()-1))
{
    if (sec_last.charAt(j) == last.charAt(j) )
        j++;
```

```
        else
            break;
    }

    res[ hm.get(last) ] = last.substring(0, j+1);

    return res;
}

/* Driver Function to test other function */
public static void main(String[] args)
{
    GFG gfg = new GFG();
    String[] input = {"zebra", "dog", "duck", "dove"};
    String[] output = gfg.uniquePrefix(input);
    System.out.println( "The shortest unique prefixes are :");
    for (int i=0; i < output.length; i++)
        System.out.print( output[i] + " ");
}
```

Output:

```
The shortest unique prefixes are :
z dog du dov
```

For a more efficient solution, we can use Trie as discussed in this [post](#).

Source

<https://www.geeksforgeeks.org/find-shortest-unique-prefix-every-word-given-list-set-2-using-sorting/>

Chapter 113

Find sum of non-repeating (distinct) elements in an array

Find sum of non-repeating (distinct) elements in an array - GeeksforGeeks

Given an integer array with repeated elements, the task is to find sum of all distinct elements in array.

Examples:

```
Input : arr[] = {12, 10, 9, 45, 2, 10, 10, 45,10};  
Output : 78
```

Here we take 12, 10, 9, 45, 2 for sum
because it's distinct elements

```
Input : arr[] = {1, 10, 9, 4, 2, 10, 10, 45 , 4};  
Output : 71
```

A **Simple Solution** is to use two nested loops. The outer loop picks an element one by one starting from the leftmost element. The inner loop checks if the element is present on left side of it. If present, then ignores the element.

Time Complexity : $O(n^2)$
Auxiliary Space : $O(1)$

A **Better Solution** of this problem is that using sorting technique we firstly sort all elements of array in ascending order and and find one by one distinct elements in array.

```
// C++ Find the sum of all non-repeated  
// elements in an array  
#include<bits/stdc++.h>  
using namespace std;
```

```
// Find the sum of all non-repeated elements
// in an array
int findSum(int arr[], int n)
{
    // sort all elements of array
    sort(arr, arr + n);

    int sum = 0;
    for (int i=0; i<n; i++)
    {
        if (arr[i] != arr[i+1])
            sum = sum + arr[i];
    }

    return sum;
}

// Driver code
int main()
{
    int arr[] = {1, 2, 3, 1, 1, 4, 5, 6};
    int n = sizeof(arr)/sizeof(int);
    cout << findSum(arr, n);
    return 0;
}
```

Output:

21

Time Complexity : $O(n \log n)$

Space Complexity : $O(1)$

An **Efficient solution** of this problem is that using `unordered_set` we run a single for loop and which value comes first time its add in sum variable and store in hash table that for next time we not use this value.

```
// C++ Find the sum of all non- repeated
// elements in an array
#include<bits/stdc++.h>
using namespace std;

// Find the sum of all non-repeated elements
// in an array
void findSum(int arr[], int n)
{
```

```
int sum = 0;

// Hash to store all element of array
unordered_set< int > s;
for (int i=0; i<n; i++)
{
    if (s.find(arr[i]) == s.end())
    {
        sum += arr[i];
        s.insert(arr[i]);
    }
}

return sum;
}

// Driver code
int main()
{
    int arr[] = {1, 2, 3, 1, 1, 4, 5, 6};
    int n = sizeof(arr)/sizeof(int);
    cout << findSum(arr, n);
    return 0;
}
```

Output:

21

Time Complexity : $O(n)$
Auxiliary Space : $O(n)$

Source

<https://www.geeksforgeeks.org/find-sum-non-repeating-distinct-elements-array/>

Chapter 114

Find the Minimum length Unsorted Subarray, sorting which makes the complete array sorted

Find the Minimum length Unsorted Subarray, sorting which makes the complete array sorted
- GeeksforGeeks

Given an unsorted array $\text{arr}[0..n-1]$ of size n , find the minimum length subarray $\text{arr}[s..e]$ such that sorting this subarray makes the whole array sorted.

Examples:

- 1) If the input array is [10, 12, 20, 30, 25, 40, 32, 31, 35, 50, 60], your program should be able to find that the subarray lies between the indexes 3 and 8.
- 2) If the input array is [0, 1, 15, 25, 6, 7, 30, 40, 50], your program should be able to find that the subarray lies between the indexes 2 and 5.

Solution:

1) Find the candidate unsorted subarray

- a) Scan from left to right and find the first element which is greater than the next element. Let s be the index of such an element. In the above example 1, s is 3 (index of 30).
- b) Scan from right to left and find the first element (first in right to left order) which is smaller than the next element (next in right to left order). Let e be the index of such an element. In the above example 1, e is 7 (index of 31).

2) Check whether sorting the candidate unsorted subarray makes the complete array sorted or not. If not, then include more elements in the subarray.

- a) Find the minimum and maximum values in $\text{arr}[s..e]$. Let minimum and maximum values be min and max . min and max for [30, 25, 40, 32, 31] are 25 and 40 respectively.
- b) Find the first element (if there is any) in $\text{arr}[0..s-1]$ which is greater than min , change s

to index of this element. There is no such element in above example 1.

c) Find the last element (if there is any) in $arr[e+1..n-1]$ which is smaller than max, change e to index of this element. In the above example 1, e is changed to 8 (index of 35)

3) Print s and e.

Implementation:

C

```
#include<stdio.h>

void printUnsorted(int arr[], int n)
{
    int s = 0, e = n-1, i, max, min;

    // step 1(a) of above algo
    for (s = 0; s < n-1; s++)
    {
        if (arr[s] > arr[s+1])
            break;
    }
    if (s == n-1)
    {
        printf("The complete array is sorted");
        return;
    }

    // step 1(b) of above algo
    for(e = n - 1; e > 0; e--)
    {
        if(arr[e] < arr[e-1])
            break;
    }

    // step 2(a) of above algo
    max = arr[s]; min = arr[s];
    for(i = s + 1; i <= e; i++)
    {
        if(arr[i] > max)
            max = arr[i];
        if(arr[i] < min)
            min = arr[i];
    }

    // step 2(b) of above algo
    for( i = 0; i < s; i++)
```

```
{  
    if(arr[i] > min)  
    {  
        s = i;  
        break;  
    }  
}  
  
// step 2(c) of above algo  
for( i = n -1; i >= e+1; i--)  
{  
    if(arr[i] < max)  
    {  
        e = i;  
        break;  
    }  
}  
  
// step 3 of above algo  
printf(" The unsorted subarray which makes the given array "  
      " sorted lies between the indees %d and %d", s, e);  
return;  
}  
  
int main()  
{  
    int arr[] = {10, 12, 20, 30, 25, 40, 32, 31, 35, 50, 60};  
    int arr_size = sizeof(arr)/sizeof(arr[0]);  
    printUnsorted(arr, arr_size);  
    getchar();  
    return 0;  
}
```

Java

```
class Main  
{  
    static void printUnsorted(int arr[], int n)  
    {  
        int s = 0, e = n-1, i, max, min;  
  
        // step 1(a) of above algo  
        for (s = 0; s < n-1; s++)  
        {  
            if (arr[s] > arr[s+1])  
                break;  
        }  
        if (s == n-1)
```

```
{  
    System.out.println("The complete array is sorted");  
    return;  
}  
  
// step 1(b) of above algo  
for(e = n - 1; e > 0; e--)  
{  
    if(arr[e] < arr[e-1])  
        break;  
}  
  
// step 2(a) of above algo  
max = arr[s]; min = arr[s];  
for(i = s + 1; i <= e; i++)  
{  
    if(arr[i] > max)  
        max = arr[i];  
    if(arr[i] < min)  
        min = arr[i];  
}  
  
// step 2(b) of above algo  
for( i = 0; i < s; i++)  
{  
    if(arr[i] > min)  
    {  
        s = i;  
        break;  
    }  
}  
  
// step 2(c) of above algo  
for( i = n - 1; i >= e+1; i--)  
{  
    if(arr[i] < max)  
    {  
        e = i;  
        break;  
    }  
}  
  
// step 3 of above algo  
System.out.println(" The unsorted subarray which"+  
                    " makes the given array sorted lies"+  
                    " between the indices "+s+" and "+e);  
return;  
}
```

```
public static void main(String args[])
{
    int arr[] = {10, 12, 20, 30, 25, 40, 32, 31, 35, 50, 60};
    int arr_size = arr.length;
    printUnsorted(arr, arr_size);
}
```

Python3

```
def printUnsorted(arr, n):
    e = n-1
    # step 1(a) of above algo
    for s in range(0,n-1):
        if arr[s] > arr[s+1]:
            break

    if s == n-1:
        print ("The complete array is sorted")
        exit()

    # step 1(b) of above algo
    e= n-1
    while e > 0:
        if arr[e] < arr[e-1]:
            break
        e -= 1

    # step 2(a) of above algo
    max = arr[s]
    min = arr[s]
    for i in range(s+1,e+1):
        if arr[i] > max:
            max = arr[i]
        if arr[i] < min:
            min = arr[i]

    # step 2(b) of above algo
    for i in range(s):
        if arr[i] > min:
            s = i
            break

    # step 2(c) of above algo
    i = n-1
    while i >= e+1:
        if arr[i] < max:
```

```
e = i
break
i -= 1

# step 3 of above algo
print ("The unsorted subarray which makes the given array")
print ("sorted lies between the indexes %d and %d"%( s, e))

arr = [10, 12, 20, 30, 25, 40, 32, 31, 35, 50, 60]
arr_size = len(arr)
printUnsorted(arr, arr_size)

# This code is contributed by Shreyanshi Arun

C#

using System;

class GFG
{
    static void printUnsorted(int []arr, int n)
    {
        int s = 0, e = n-1, i, max, min;

        // step 1(a) of above algo
        for (s = 0; s < n-1; s++)
        {
            if (arr[s] > arr[s+1])
                break;
        }
        if (s == n-1)
        {
            Console.WriteLine("The complete " +
                               "array is sorted");
            return;
        }

        // step 1(b) of above algo
        for(e = n - 1; e > 0; e--)
        {
            if(arr[e] < arr[e-1])
                break;
        }

        // step 2(a) of above algo
        max = arr[s]; min = arr[s];

        for(i = s + 1; i <= e; i++)
    }
```

```
{  
    if(arr[i] > max)  
        max = arr[i];  
  
    if(arr[i] < min)  
        min = arr[i];  
}  
  
// step 2(b) of above algo  
for( i = 0; i < s; i++)  
{  
    if(arr[i] > min)  
    {  
        s = i;  
        break;  
    }  
}  
  
// step 2(c) of above algo  
for( i = n -1; i >= e+1; i--)  
{  
    if(arr[i] < max)  
    {  
        e = i;  
        break;  
    }  
}  
  
// step 3 of above algo  
Console.Write(" The unsorted subarray which"+  
            " makes the given array sorted lies \n"+  
            " between the indices "+s+" and "+e);  
return;  
}  
  
public static void Main()  
{  
    int []arr = {10, 12, 20, 30, 25, 40,  
                32, 31, 35, 50, 60};  
    int arr_size = arr.Length;  
  
    printUnsorted(arr, arr_size);  
}  
}  
  
// This code contributed by Sam007
```

Output :

Chapter 114. Find the Minimum length Unsorted Subarray, sorting which makes the complete array sorted

The unsorted subarray which makes the given array sorted lies between the indees 3 and 8

Time Complexity: $O(n)$

Source

<https://www.geeksforgeeks.org/minimum-length-unsorted-subarray-sorting-which-makes-the-complete-array-sorted/>

Chapter 115

Find the Sub-array with sum closest to 0

Find the Sub-array with sum closest to 0 - GeeksforGeeks

Given an array of both positive and negative numbers, the task is to find out the subarray whose sum is closest to 0.

There can be multiple such subarrays, we need to output just 1 of them.

Examples:

```
Input : arr[] = {-1, 3, 2, -5, 4}
Output : 1, 3
Subarray from index 1 to 3 has sum closest to 0 i.e.
3 + 2 + -5 = 0
```

```
Input : {2, -5, 4, -6, 3}
Output : 0, 2
2 + -5 + 4 = 1 closest to 0
```

Asked in : Microsoft

A **Naive approach** is to consider all subarrays one by one and update indexes of subarray with sum closest to 0.

```
// C++ program to find subarray with
// sum closest to 0
#include <bits/stdc++.h>
using namespace std;

// Function to find the subarray
pair<int, int> findSubArray(int arr[], int n)
{
```

```
int start, end, min_sum = INT_MAX;

// Pick a starting point
for (int i = 0; i < n; i++) {

    // Consider current starting point
    // as a subarray and update minimum
    // sum and subarray indexes
    int curr_sum = arr[i];
    if (min_sum > abs(curr_sum)) {
        min_sum = abs(curr_sum);
        start = i;
        end = i;
    }

    // Try all subarrays starting with i
    for (int j = i + 1; j < n; j++) {
        curr_sum = curr_sum + arr[j];

        // update minimum sum
        // and subarray indexes
        if (min_sum > abs(curr_sum)) {
            min_sum = abs(curr_sum);
            start = i;
            end = j;
        }
    }
}

// Return starting and ending indexes
pair<int, int> p = make_pair(start, end);
return p;
}

// Drivers code
int main()
{
    int arr[] = { 2, -5, 4, -6, -3 };
    int n = sizeof(arr) / sizeof(arr[0]);

    pair<int, int> point = findSubArray(arr, n);
    cout << "Subarray starting from ";
    cout << point.first << " to " << point.second;
    return 0;
}
```

Output:

Subarray starting from 0 to 2

Time Complexity: $O(n^2)$

An **Efficient method** is to perform following steps:-

1. Maintain a [Prefix sum array](#). Also maintain indexes in the prefix sum array.
2. Sort the prefix sum array on the basis of sum.
3. Find the two elements in a prefix sum array with minimum difference.

i.e. `Find min(pre_sum[i] - pre_sum[i-1])`

4. Return indexes of pre_sum with minimum difference.
5. Subarray with $(lower_index+1, upper_index)$ will have the sum closest to 0.
6. Taking $lower_index+1$ because on subtracting value at $lower_index$ we get the sum closest to 0. That's why $lower_index$ need not to be included.

```
// C++ program to find subarray with sum
// closest to 0
#include <bits/stdc++.h>
using namespace std;

struct prefix {
    int sum;
    int index;
};

// Sort on the basis of sum
bool comparison(prefix a, prefix b)
{
    return a.sum < b.sum;
}

// Returns subarray with sum closest to 0.
pair<int, int> findSubArray(int arr[], int n)
{
    int start, end, min_diff = INT_MAX;

    prefix pre_sum[n + 1];

    // To consider the case of subarray starting
    // from beginning of the array
    pre_sum[0].sum = 0;
    pre_sum[0].index = -1;
```

```
// Store prefix sum with index
for (int i = 1; i <= n; i++) {
    pre_sum[i].sum = pre_sum[i-1].sum + arr[i-1];
    pre_sum[i].index = i - 1;
}

// Sort on the basis of sum
sort(pre_sum, pre_sum + (n + 1), comparison);

// Find two consecutive elements with minimum difference
for (int i = 1; i <= n; i++) {
    int diff = pre_sum[i].sum - pre_sum[i-1].sum;

    // Update minimum difference
    // and starting and ending indexes
    if (min_diff > diff) {
        min_diff = diff;
        start = pre_sum[i-1].index;
        end = pre_sum[i].index;
    }
}

// Return starting and ending indexes
pair<int, int> p = make_pair(start + 1, end);
return p;
}

// Drivers code
int main()
{
    int arr[] = { 2, 3, -4, -1, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    pair<int, int> point = findSubArray(arr, n);
    cout << "Subarray starting from ";
    cout << point.first << " to " << point.second;

    return 0;
}
```

Output:

```
Subarray starting from 0 to 3
```

Time Complexity: $O(n \log n)$

Reference:

<https://www.careercup.com/question?id=14583859>

Source

<https://www.geeksforgeeks.org/find-sub-array-sum-closest-0/>

Chapter 116

Find the largest multiple of 3 from array of digits Set 2 (In O(n) time and O(1) space)

Find the largest multiple of 3 from array of digits Set 2 (In O(n) time and O(1) space) - GeeksforGeeks

Given an array of digits (contain elements from 0 to 9). Find the largest number that can be made from **some or all digits** of array and is divisible by 3. The same element may appear multiple times in the array, but each element in the array may only be used once.

Examples:

Input : arr[] = {5, 4, 3, 1, 1}
Output : 4311

Input : Arr[] = {5, 5, 5, 7}
Output : 555

Asked In : [Google Interview](#)

We have discussed a [queue based solution](#). Both solutions (discussed in previous and this posts) are based on the fact that **a number is divisible by 3 if and only if sum of digits of the number is divisible by 3**.

For example, let us consider 555, it is divisible by 3 because sum of digits is $5 + 5 + 5 = 15$, which is divisible by 3. If a sum of digits is not divisible by 3 then the remainder should be either 1 or 2.

If we get remainder either ‘1’ or ‘2’, we have to remove maximum two digits to make a number that is divisible by 3:

1. If remainder is ‘1’ : We have to remove single digit that have remainder ‘1’ or we have to remove two digit that have remainder ‘2’ ($2 + 2 \Rightarrow 4 \% 3 \Rightarrow '1'$)

2. If remainder is '2' : .We have to remove single digit that have remainder '2' or we have to remove two digit that have remainder '1' ($1 + 1 \Rightarrow 2 \% 3 \Rightarrow 2$).

Examples :

```
Input : arr[] = 5, 5, 5, 7
Sum of digits = 5 + 5 + 7 = 22
Reminder = 22 % 3 = 1
We remove smallest single digit that
has remainder '1'. We remove 7 % 3 = 1
So largest number divisible by 3 is : 555
```

```
Let's take an another example :
Input : arr[] = 4 , 4 , 1 , 1 , 1 , 3
Sum of digits = 4 + 4 + 1 + 1 + 1 + 3 = 14
Reminder = 14 % 3 = 2
We have to remove the smallest digit that
has remainder ' 2 ' or two digits that have
remainder '1'. Here there is no digit with
remainder '2', so we have to remove two smallest
digits that have remainder '1'. The digits are :
1, 1. So largest number divisible by 3 is 4 4 3 1
```

Below is C++ implementation of above idea.

```
// C++ program to find the largest number
// that can be mode from elements of the
// array and is divisible by 3
#include<bits/stdc++.h>
using namespace std;

// Number of digits
#define MAX_SIZE 10

// function to sort array of digits using
// counts
void sortArrayUsingCounts(int arr[], int n)
{
    // Store count of all elements
    int count[MAX_SIZE] = {0};
    for (int i = 0; i < n; i++)
        count[arr[i]]++;

    // Store
    int index = 0;
    for (int i = 0; i < MAX_SIZE; i++)
```

```
        while (count[i] > 0)
            arr[index++] = i, count[i]--;
    }

// Remove elements from arr[] at indexes ind1 and ind2
bool removeAndPrintResult(int arr[], int n, int ind1,
                           int ind2 = -1)
{
    for (int i = n-1; i >=0; i--)
        if (i != ind1 && i != ind2)
            cout << arr[i] ;
}

// Returns largest multiple of 3 that can be formed
// using arr[] elements.
bool largest3Multiple(int arr[], int n)
{
    // Sum of all array element
    int sum = accumulate(arr, arr+n, 0);

    // Sum is divisible by 3 , no need to
    // delete an element
    if (sum%3 == 0)
        return true ;

    // Sort array element in increasing order
    sortArrayUsingCounts(arr, n);

    // Find remainder
    int remainder = sum % 3;

    // If remainder is '1', we have to delete either
    // one element of remainder '1' or two elements
    // of remainder '2'
    if (remainder == 1)
    {
        int rem_2[2];
        rem_2[0] = -1, rem_2[1] = -1;

        // Traverse array elements
        for (int i = 0 ; i < n ; i++)
        {
            // Store first element of remainder '1'
            if (arr[i]%3 == 1)
            {
                removeAndPrintResult(arr, n, i);
                return true;
            }
        }
    }
}
```

```
if (arr[i] % 3 == 2)
{
    // If this is first occurrence of remainder 2
    if (rem_2[0] == -1)
        rem_2[0] = i;

    // If second occurrence
    else if (rem_2[1] == -1)
        rem_2[1] = i;
}
}

if (rem_2[0] != -1 && rem_2[1] != -1)
{
    removeAndPrintResult(arr, n, rem_2[0], rem_2[1]);
    return true;
}
}

// If remainder is '2', we have to delete either
// one element of remainder '2' or two elements
// of remainder '1'
else if (remainder == 2)
{
    int rem_1[2];
    rem_1[0] = -1, rem_1[1] = -1;

    // traverse array elements
    for (int i = 0; i < n; i++)
    {
        // store first element of remainder '2'
        if (arr[i] % 3 == 2)
        {
            removeAndPrintResult(arr, n, i);
            return true;
        }

        if (arr[i] % 3 == 1)
        {
            // If this is first occurrence of remainder 1
            if (rem_1[0] == -1)
                rem_1[0] = i;

            // If second occurrence
            else if (rem_1[1] == -1)
                rem_1[1] = i;
        }
    }
}
```

```
}

if (rem_1[0] != -1 && rem_1[1] != -1)
{
    removeAndPrintResult(arr, n, rem_1[0], rem_1[1]);
    return true;
}
}

cout << "Not possible";
return false;
}

// Driver code
int main()
{
    int arr[] = {4, 4, 1, 1, 1, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    largest3Multiple(arr, n);
    return 0;
}
```

Output:

555

Source

<https://www.geeksforgeeks.org/find-largest-multiple-3-array-digits-set-2-time-o1-space/>

Chapter 117

Find the largest number that can be formed with the given digits

Find the largest number that can be formed with the given digits - GeeksforGeeks

Given an array of integers arr[] representing digits of a number. The task is to write a program to generate the largest number possible using these digits.

Note: The digits in the array are in between 0 and 9. That is, $0 < \text{arr}[i] < 9$.

Examples:

Input : arr[] = {4, 7, 9, 2, 3}
Output : Largest number: 97432

Input : arr[] = {8, 6, 0, 4, 6, 4, 2, 7}
Output : Largest number: 87664420

Naive Approach: The naive approach is to sort the given array of digits in **descending order** and then form the number using the digits in array keeping the order of digits in the number same as that of the sorted array.

Time Complexity: $O(N \log N)$, where N is the number of digits.

Below is the implementation of above idea:

C++

```
// C++ program to generate largest possible
// number with given digits
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to generate largest possible
// number with given digits
int findMaxNum(int arr[], int n)
{
    // sort the given array in
    // descending order
    sort(arr, arr+n, greater<int>());

    int num = arr[0];

    // generate the number
    for(int i=1; i<n; i++)
    {
        num = num*10 + arr[i];
    }

    return num;
}

// Driver code
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 0};

    int n = sizeof(arr)/sizeof(arr[0]);

    cout<<findMaxNum(arr,n);

    return 0;
}
```

Java

```
// Java program to generate largest
// possible number with given digits
import java.*;
import java.util.Arrays;

class GFG
{
    // Function to generate largest
    // possible number with given digits
    static int findMaxNum(int arr[], int n)
    {
        // sort the given array in
        // ascending order and then
        // traverse into descending
```

```
Arrays.sort(arr);
int num = arr[0];
// generate the number
for(int i = n - 1; i >= 0; i--)
{
    num = num * 10 + arr[i];
}
return num;
}

// Driver code
public static void main(String[] args)
{
    int arr[] = {1, 2, 3, 4, 5, 0};
    int n = arr.length;
    System.out.println(findMaxNum(arr, n));
}
```

// This code is contributed by mits

Python3

```
# Python3 program to generate largest possible
# number with given digits

# Function to generate largest possible
# number with given digits
def findMaxNum(arr,n) :

    # sort the given array in
    # descending order
    arr.sort(reverse = True)

    # initialize num with starting
    # element of an arr
    num = arr[0]

    # generate the number
    for i in range(1,n) :
        num = num * 10 + arr[i]

    return num

# Driver code
if __name__ == "__main__" :
```

```
arr = [1,2,3,4,5,0]
n = len(arr)
print(findMaxNum(arr,n))
```

PHP

```
<?php
// PHP program to generate
// largest possible number
// with given digits

// Function to generate
// largest possible number
// with given digits
function findMaxNum(&$arr, $n)
{
    // sort the given array
    // in descending order
    rsort($arr);

    $num = $arr[0];

    // generate the number
    for($i = 1; $i < $n; $i++)
    {
        $num = $num * 10 + $arr[$i];
    }

    return $num;
}

// Driver code
$arr = array(1, 2, 3, 4, 5, 0);
$n = sizeof($arr);
echo findMaxNum($arr,$n);

// This code is contributed
// by ChitraNayal
?>
```

Output:

543210

Efficient Approach: An efficient approach is to observe that we have to form the number using only digits from 0-9. Hence we can create a hash of size 10 to store *the number of occurrences of the digits in the given array* into the hash table. Where the key in the hash table will be digits from 0 to 9 and their values will be the count of their occurrences in the array.

Finally, print the digits the number of times they occur in descending order starting from the digit 9.

Below is the implementation of above approach:

C++

```
// C++ program to generate largest possible
// number with given digits
#include <bits/stdc++.h>

using namespace std;

// Function to generate largest possible
// number with given digits
void findMaxNum(int arr[], int n)
{
    // Declare a hash array of size 10
    // and initialize all the elements to zero
    int hash[10] = {0};

    // store the number of occurrences of the digits
    // in the given array into the hash table
    for(int i=0; i<n; i++)
    {
        hash[arr[i]]++;
    }

    // Traverse the hash in descending order
    // to print the required number
    for(int i=9; i>=0; i--)
    {
        // Print the number of times a digits occurs
        for(int j=0; j<hash[i]; j++)
            cout<<i;
    }
}

// Driver code
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 0};

    int n = sizeof(arr)/sizeof(arr[0]);
```

```
    findMaxNum(arr,n);

    return 0;
}

Java

// Java program to generate
// largest possible number
// with given digits
class GFG
{

    // Function to generate
    // largest possible number
    // with given digits
    static void findMaxNum(int arr[],
                           int n)
    {
        // Declare a hash array of
        // size 10 and initialize
        // all the elements to zero
        int []hash = new int[10];

        // store the number of occurrences
        // of the digits in the given array
        // into the hash table
        for(int i = 0; i < n; i++)
        {
            hash[arr[i]]++;
        }

        // Traverse the hash in descending
        // order to print the required number
        for(int i = 9; i >= 0; i--)
        {
            // Print the number of
            // times a digits occurs
            for(int j = 0; j < hash[i]; j++)
                System.out.print(i);
        }
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = {1, 2, 3, 4, 5, 0};
```

```
int n = arr.length;  
  
findMaxNum(arr,n);  
}  
}  
  
// This code is contributed  
// by ChitraNayal
```

Python 3

```
# Python 3 program to generate  
# largest possible number  
# with given digits  
  
# Function to generate  
# largest possible number  
# with given digits  
def findMaxNum(arr, n):  
  
    # Declare a hash array of  
    # size 10 and initialize  
    # all the elements to zero  
    hash = [0] * 10  
  
    # store the number of occurrences  
    # of the digits in the given array  
    # into the hash table  
    for i in range(n):  
        hash[arr[i]] += 1  
  
    # Traverse the hash in  
    # descending order to  
    # print the required number  
    for i in range(9, -1, -1):  
  
        # Print the number of  
        # times a digits occurs  
        for j in range(hash[i]):  
            print(i, end = "")  
  
# Driver code  
if __name__ == "__main__":  
    arr = [1, 2, 3, 4, 5, 0]  
    n = len(arr)  
    findMaxNum(arr, n)
```

```
# This code is contributed
# by ChitraNayal

C#

// C# program to generate
// largest possible number
// with given digits
using System;

class GFG
{

    // Function to generate
    // largest possible number
    // with given digits
    static void findMaxNum(int[] arr,
                           int n)
    {
        // Declare a hash array of
        // size 10 and initialize
        // all the elements to zero
        int[] hash = new int[10];

        // store the number of
        // occurrences of the
        // digits in the given
        // array into the hash table
        for(int i = 0; i < n; i++)
        {
            hash[arr[i]]++;
        }

        // Traverse the hash in
        // descending order to
        // print the required number
        for(int i = 9; i >= 0; i--)
        {
            // Print the number of
            // times a digits occurs
            for(int j = 0; j < hash[i]; j++)
                Console.Write(i);
        }
    }

    // Driver code
    public static void Main()
    {
```

```
int[] arr = {1, 2, 3, 4, 5, 0};

int n = arr.Length;

findMaxNum(arr, n);
}

}

// This code is contributed
// by ChitraNayal
```

PHP

```
<?php
// PHP program to generate
// largest possible number
// with given digits

// Function to generate
// largest possible number
// with given digits
function findMaxNum($arr, $n)
{
    // Declare a hash array of
    // size 10 and initialize
    // all the elements to zero
    $hash = array_fill(0, 10, 0);

    // store the number of occurrences
    // of the digits in the given array
    // into the hash table
    for($i = 0; $i < $n; $i++)
        $hash[$arr[$i]] += 1;

    // Traverse the hash in
    // descending order to
    // print the required number
    for($i = 9; $i >= 0; $i--)

        // Print the number of
        // times a digits occurs
        for($j = 0; $j < $hash[$i]; $j++)
            echo $i;
}

// Driver code
$arr = array(1, 2, 3, 4, 5, 0);
$n = sizeof($arr);
```

```
findMaxNum($arr,$n);  
  
// This code is contributed  
// by mits  
?>
```

Output:

543210

Time Complexity: O(N), where N is the number of digits.

Auxiliary Space: O(1), size of hash is only 10 which is a constant.

Improved By : [ANKITRAI1](#), [ChitraNayal](#), [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/find-the-largest-number-that-can-be-formed-with-the-given-digits/>

Chapter 118

Find the minimum and maximum amount to buy all N candies

Find the minimum and maximum amount to buy all N candies - GeeksforGeeks

In a candy store there are N different types of candies available and the prices of all the N different types of candies are provided. There is also an attractive offer by candy store. We can buy a single candy from the store and get at-most K other candies (all are different types) for free.

1. Find minimum amount of money we have to spend to buy all the N different candies.
2. Find maximum amount of money we have to spend to buy all the N different candies.

In both the cases we must utilize the offer and get maximum possible candies back. If k or more candies are available, we must take k candies for every candy purchase. If less than k candies are available, we must take all candies for a candy purchase.

Examples:

```
Input : price[] = {3, 2, 1, 4}
        k = 2
Output : Min = 3, Max = 7
Since k is 2, if we buy one candy we can take
atmost two more for free.
So in the first case we buy the candy which
costs 1 and take candies worth 3 and 4 for
free, also you buy candy worth 2 as well.
So min cost = 1 + 2 = 3.
```

In the second case we buy the candy which costs 4 and take candies worth 1 and 2 for free, also We buy candy worth 3 as well.
So max cost = $3 + 4 = 7$.

One important thing to note is, we must use the offer and get maximum candies back for every candy purchase. So if we want to minimize the money, we must buy candies of minimum cost and get candies of maximum costs for free. To maximize the money, we must do reverse. Below is algorithm based on this.

First Sort the price array.

For finding minimum amount :

Start purchasing candies from starting and reduce k free candies from last with every single purchase.

For finding maximum amount :

Start purchasing candies from the end and reduce k free candies from starting in every single purchase.

C++

```
// C++ implementation to find the minimum
// and maximum amount
#include<bits/stdc++.h>
using namespace std;

// Function to find the minimum amount
// to buy all candies
int findMinimum(int arr[], int n, int k)
{
    int res = 0;
    for (int i=0; i<n ; i++)
    {
        // Buy current candy
        res += arr[i];

        // And take k candies for free
        // from the last
        n = n-k;
    }
    return res;
}
```

```
// Function to find the maximum amount
// to buy all candies
int findMaximum(int arr[], int n, int k)
{
    int res = 0, index = 0;

    for (int i=n-1; i>=index; i--)
    {
        // Buy candy with maximum amount
        res += arr[i];

        // And get k candies for free from
        // the starting
        index += k;
    }
    return res;
}

// Driver code
int main()
{
    int arr[] = {3, 2, 1, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;
    sort(arr, arr+n);

    cout << findMinimum(arr, n, k) << " "
        << findMaximum(arr, n, k) << endl;
    return 0;
}
```

Java

```
// Java implementation to find the
// minimum and maximum amount
import java.util.*;

class GFG {

    // Function to find the minimum
    // amount to buy all candies
    static int findMinimum(int arr[], int n, int k)
    {
        int res = 0;
        for (int i = 0; i < n; i++)
        {
            // Buy current candy
            res += arr[i];
```

```
// And take k candies for free
// from the last
n = n - k;
}
return res;
}

// Function to find the maximum
// amount to buy all candies
static int findMaximum(int arr[], int n, int k)
{
    int res = 0, index = 0;

    for (int i = n - 1; i >= index; i--)
    {
        // Buy candy with maximum amount
        res += arr[i];

        // And get k candies for free from
        // the starting
        index += k;
    }
    return res;
}

// Driver code
public static void main(String[] args)
{
    int arr[] = { 3, 2, 1, 4 };
    int n = arr.length;
    int k = 2;
    Arrays.sort(arr);

    System.out.println(findMinimum(arr, n, k) +
                       " " + findMaximum(arr, n, k));
}

// This code is contributed by prerna saini
```

Python3

```
# Python implementation
# to find the minimum
# and maximum amount

# Function to find
```

```
# the minimum amount
# to buy all candies
def findMinimum(arr,n,k):

    res = 0
    i=0
    while(n):

        # Buy current candy
        res += arr[i]

        # And take k
        # candies for free
        # from the last
        n = n-k
        i+=1
    return res

# Function to find
# the maximum amount
# to buy all candies
def findMaximum(arr, n, k):

    res = 0
    index = 0
    i=n-1
    while(i>=index):

        # Buy candy with
        # maximum amount
        res += arr[i]

        # And get k candies
        # for free from
        # the starting
        index += k
        i -= 1

    return res

# Driver code

arr = [3, 2, 1, 4]
n = len(arr)
k = 2

arr.sort()
```

```
print(findMinimum(arr, n, k), " ",
      findMaximum(arr, n, k))

# This code is contributed
# by Anant Agarwal.

C#

// C# implementation to find the
// minimum and maximum amount
using System;

public class GFG {

    // Function to find the minimum
    // amount to buy all candies
    static int findMinimum(int []arr,
                           int n, int k)
    {
        int res = 0;
        for (int i = 0; i < n; i++)
        {

            // Buy current candy
            res += arr[i];

            // And take k candies for
            // free from the last
            n = n - k;
        }

        return res;
    }

    // Function to find the maximum
    // amount to buy all candies
    static int findMaximum(int []arr,
                           int n, int k)
    {
        int res = 0, index = 0;

        for (int i = n - 1; i >= index; i--)
        {
            // Buy candy with maximum
            // amount
            res += arr[i];

            // And get k candies for free
        }
    }
}
```

```
// from the starting
index += k;
}

return res;
}

// Driver code
public static void Main()
{
    int []arr = { 3, 2, 1, 4 };
    int n = arr.Length;
    int k = 2;
    Array.Sort(arr);

    Console.WriteLine(
        findMinimum(arr, n, k) + " "
        + findMaximum(arr, n, k));
}
}

// This code is contributed by Sam007.
```

PHP

```
<?php
// PHP implementation to find the minimum
// and maximum amount

// Function to find the minimum amount
// to buy all candies
function findMinimum($arr, $n,$k)
{
    $res = 0;
    for ($i = 0; $i < $n ; $i++)
    {

        // Buy current candy
        $res += $arr[$i];

        // And take k candies for free
        // from the last
        $n = $n - $k;
    }
    return $res;
}

// Function to find the maximum amount
```

```
// to buy all candies
function findMaximum($arr, $n, $k)
{
    $res = 0;
    $index = 0;

    for ($i = $n - 1; $i >= $index; $i--)
    {

        // Buy candy with maximum amount
        $res += $arr[$i];

        // And get k candies
        // for free from
        // the starting
        $index += $k;
    }
    return $res;
}

// Driver Code
$arr = array(3, 2, 1, 4);
$n = sizeof($arr);
$k = 2;
sort($arr); sort($arr,$n);

echo findMinimum($arr, $n, $k), " "
      ,findMaximum($arr, $n, $k);
return 0;

// This code is contributed by nitin mittal.
?>
```

Output:

3 7

Time Complexity : $O(n \log n)$

Improved By : [Sam007](#), [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/find-minimum-maximum-amount-buy-n-candies/>

Chapter 119

Find the point where maximum intervals overlap

Find the point where maximum intervals overlap - GeeksforGeeks

Consider a big party where a log register for guest's entry and exit times is maintained. Find the time at which there are maximum guests in the party. Note that entries in register are not in any order.

Example :

```
Input: arr1[] = {1, 2, 9, 5, 5}
       exit[] = {4, 5, 12, 9, 12}
First guest in array arrives at 1 and leaves at 4,
second guest arrives at 2 and leaves at 5, and so on.
```

```
Output: 5
There are maximum 3 guests at time 5.
```

Below is a **Simple Method** to solve this problem.

- 1) Traverse all intervals and find min and max time (time at which first guest arrives and time at which last guest leaves)
- 2) Create a count array of size 'max - min + 1'. Let the array be count[].
- 3) For each interval [x, y], run a loop for i = x to y and do following in loop.
count[i - min]++;
- 4) Find the index of maximum element in count array. Let this index be 'max_index', return max_index + min.

Above solution requires $O(\text{max-min}+1)$ extra space. Also time complexity of above solution depends on lengths of intervals. In worst case, if all intervals are from 'min' to 'max', then time complexity becomes $O((\text{max-min}+1)*n)$ where n is number of intervals.

An **Efficient Solution** is to use sorting in $O(n \log n)$ time. The idea is to consider all events (all arrivals and exits) in sorted order. Once we have all events in sorted order, we can trace the number of guests at any time keeping track of guests that have arrived, but not exited.

Consider the above example.

```
arr[] = {1, 2, 10, 5, 5}
dep[] = {4, 5, 12, 9, 12}
```

Below are all events sorted by time. Note that in sorting, if two events have same time, then arrival is preferred over exit.

Time	Event Type	Total Number of Guests Present
1	Arrival	1
2	Arrival	2
4	Exit	1
5	Arrival	2
5	Arrival	3 // Max Guests
5	Exit	2
9	Exit	1
10	Arrival	2
12	Exit	1
12	Exit	0

Total number of guests at any time can be obtained by subtracting total exits from total arrivals by that time.

So maximum guests are three at time 5.

Following is the implementation of above approach. Note that the implementation doesn't create a single sorted list of all events, rather it individually sorts arr[] and dep[] arrays, and then uses merge process of merge sort to process them together as a single sorted array.

C++

```
// Program to find maximum guest at any time in a party
#include<iostream>
#include<algorithm>
using namespace std;

void findMaxGuests(int arr[], int exit[], int n)
{
    // Sort arrival and exit arrays
    sort(arr, arr+n);
    sort(exit, exit+n);

    // guests_in indicates number of guests at a time
    int guests_in = 1, max_guests = 1, time = arr[0];
    int i = 1, j = 0;
```

```
// Similar to merge in merge sort to process
// all events in sorted order
while (i < n && j < n)
{
    // If next event in sorted order is arrival,
    // increment count of guests
    if (arrl[i] <= exit[j])
    {
        guests_in++;

        // Update max_guests if needed
        if (guests_in > max_guests)
        {
            max_guests = guests_in;
            time = arrl[i];
        }
        i++; //increment index of arrival array
    }
    else // If event is exit, decrement count
    {
        // of guests.
        guests_in--;
        j++;
    }
}
cout << "Maximum Number of Guests = " << max_guests
     << " at time " << time;
}

// Driver program to test above function
int main()
{
    int arrl[] = {1, 2, 10, 5, 5};
    int exit[] = {4, 5, 12, 9, 12};
    int n = sizeof(arrl)/sizeof(arrl[0]);
    findMaxGuests(arrl, exit, n);
    return 0;
}
```

Java

```
// Java Program to find maximum guest
// at any time in a party
import java.util.*;

class GFG {
```

```
static void findMaxGuests(int arrl[], int exit[],
                         int n)
{
    // Sort arrival and exit arrays
    Arrays.sort(arrl);
    Arrays.sort(exit);

    // guests_in indicates number of guests at a time
    int guests_in = 1, max_guests = 1, time = arrl[0];
    int i = 1, j = 0;

    // Similar to merge in merge sort to process
    // all events in sorted order
    while (i < n && j < n)
    {
        // If next event in sorted order is arrival,
        // increment count of guests
        if (arrl[i] <= exit[j])
        {
            guests_in++;

            // Update max_guests if needed
            if (guests_in > max_guests)
            {
                max_guests = guests_in;
                time = arrl[i];
            }
            i++; //increment index of arrival array
        }
        else // If event is exit, decrement count
        { // of guests.
            guests_in--;
            j++;
        }
    }

    System.out.println("Maximum Number of Guests = "+
                       max_guests + " at time " + time);
}

// Driver program to test above function
public static void main(String[] args)
{
    int arrl[] = {1, 2, 10, 5, 5};
    int exit[] = {4, 5, 12, 9, 12};
    int n = arrl.length;
    findMaxGuests(arrl, exit, n);
}
```

```
}
```

```
// This code is contributed by Prerna Saini
```

C#

```
// C# Program to find maximum guest
// at any time in a party
using System;
class GFG
{
    static void findMaxGuests(int []arrl,
                            int []exit,
                            int n)
    {
        // Sort arrival and exit arrays
        Array.Sort(arrl);
        Array.Sort(exit);

        // guests_in indicates number
        // of guests at a time
        int guests_in = 1,
            max_guests = 1,
            time = arrl[0];
        int i = 1, j = 0;

        // Similar to merge in merge sort
        // to process all events in sorted order
        while (i < n && j < n)
        {
            // If next event in sorted
            // order is arrival,
            // increment count of guests
            if (arrl[i] <= exit[j])
            {
                guests_in++;

                // Update max_guests if needed
                if (guests_in > max_guests)
                {
                    max_guests = guests_in;
                    time = arrl[i];
                }

                //increment index of arrival array
                i++;
            }
            // If event is exit, decrement
        }
    }
}
```

```
// count of guests.  
else  
{  
    guests_in--;  
    j++;  
}  
}  
  
Console.WriteLine("Maximum Number of Guests = "+  
                  max_guests +  
                  " at time " + time);  
}  
  
// Driver Code  
public static void Main()  
{  
    int []arrl = {1, 2, 10, 5, 5};  
    int []exit = {4, 5, 12, 9, 12};  
    int n = arrl.Length;  
    findMaxGuests(arrl, exit, n);  
}  
}  
  
// This code is contributed by nitin mittal.
```

PHP

```
<?php  
// PHP Program to find maximum  
// guest at any time in a party  
  
function findMaxGuests($arrl, $exit, $n)  
{  
  
    // Sort arrival and exit arrays  
    sort($arrl);  
    sort($exit);  
  
    // guests_in indicates number  
    // of guests at a time  
    $guests_in = 1;  
    $max_guests = 1;  
    $time = $arrl[0];  
    $i = 1;  
    $j = 0;  
  
    // Similar to merge in merge  
    // sort to process all events
```

```
// in sorted order
while ($i < $n and $j < $n)
{
    // If next event in sorted
    // order is arrival,
    // increment count of guests
    if ($arr1[$i] <= $exit[$j])
    {
        $guests_in++;

        // Update max_guests if needed
        if ($guests_in > $max_guests)
        {
            $max_guests = $guests_in;
            $time = $arr1[$i];
        }

        // increment index of
        // arrival array
        $i++;
    }

    // If event is exit, decrement
    // count of guests.
    else
    {
        $guests_in--;
        $j++;
    }
}

echo "Maximum Number of Guests = " , $max_guests
      , " at time " , $time;
}

// Driver Code
$arr1 = array(1, 2, 10, 5, 5);
$exit = array(4, 5, 12, 9, 120);
$n = count($arr1);
findMaxGuests($arr1, $exit, $n);

// This code is contributed by anuj_67.
?>
```

Output :

```
Maximum Number of Guests = 3 at time 5
```

Time Complexity of this method is O(nLogn).

Thanks to Gaurav Ahirwar for suggesting this method.

Another Efficient Solution :

Approach :

- 1). Create an auxiliary array used for storing dynamic data of starting and ending points.
- 2). Loop through the whole array of elements and increase the value at the starting point by 1 and similarly decrease the value after ending point by 1.
[Here we use the expressions “x[start[i]]-=1” and “x[end[i]+1]-=1”]
- 4). While looping, after calculating the auxiliary array: permanently add the value at current index and check for the maximum valued index traversing from left to right.

C++

```
#include<bits/stdc++.h>
using namespace std;

void maxOverlap(vector<int>& start, vector<int>& end ){

    int n= start.size();
    // Finding maximum starting time O(n)
    int maxa=*max_element(start.begin(),start.end());

    //Finding maximum ending time O(n)
    int maxb=*max_element(end.begin(),end.end());

    int maxc=max(maxa,maxb);

    int x[maxc+2];memset(x,0,sizeof x);

    int cur=0, idx;
    for(int i=0;i<n;i++)// Creating and auxiliary array O(n)
    { //Lazy addition
        ++x[start[i]];
        --x[end[i]+1];
    }

    int maxy=INT_MIN;
    //Lazily Calculating value at index i O(n)
    for(int i=0;i<=maxc;i++)
    {
        cur+=x[i];if(maxy<cur){maxy=cur;idx=i;}
    }
    cout<<"Maximum value is "<<maxy<<" at position "<<idx<<endl;
}

// DRIVER FUNCTION
```

```
int main()
{
    vector<int> start={13,28,29,14,40,17,3},
          end={107,95,111,105,70,127,74};

    maxOverlap(start,end);
    return 0;
}
```

Java

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {

    public static void maxOverlap(int []start,int [] end ,int n)
    {
        // Finding maximum starting time
        int maxa = Arrays.stream(start).max().getAsInt();

        // Finding maximum ending time
        int maxb = Arrays.stream(end).max().getAsInt();

        int maxc=Math.max(maxa,maxb);

        int []x = new int[maxc+2];
        Arrays.fill(x, 0);

        int cur=0,idx=0;
        // CREATING AN AUXILIARY ARRAY
        for(int i=0;i<n;i++)
        { // Lazy addition
            ++x[start[i]];
            --x[end[i]+1];
        }

        int maxy=Integer.MIN_VALUE;
        // Lazily Calculating value at index i
        for(int i=0;i<=maxc;i++)
        {
            cur+=x[i];if(maxy<cur){maxy=cur;idx=i;}
        }
        System.out.println("Maximum value is:"+maxy+" at position: "+idx+"");
    }
}
```

```
}

public static void main(String[] args) { // Driver function

    int [] start = new int[]{13, 28, 29, 14, 40, 17, 3 };
    int [] end    = new int[]{107, 95, 111, 105, 70, 127, 74};
    int n=start.length;

    maxOverlap(start,end,n);
}
}
```

Python3

```
import sys

def maxOverlap(start,end):

    n= len(start)
    maxa = max(start)# Finding maximum starting time
    maxb = max(end) # Finding maximum ending time
    maxc=max(maxa,maxb)
    x =(maxc+2)*[0]
    cur=0; idx=0

    for i in range(0,n) :# CREATING AN AUXILIARY ARRAY
        x[start[i]]+=1 # Lazy addition
        x[end[i]+1]-=1

    maxy=-1
    #Lazily Calculating value at index i
    for i in range(0,maxc+1):
        cur+=x[i]
        if maxy<cur :
            maxy=cur
            idx=i
    print("Maximum value is: {0:d}".format(maxy),
          " at position: {0:d}".format(idx))
if __name__ == "__main__":
    start=[13,28,29,14,40,17,3]
    end=[107,95,111,105,70,127,74]

    maxOverlap(start,end)
```

Time Complexity : O(n)

Auxiliary Space : $O(n)$

Thanks to Harshit Saini for suggesting this method.

Improved By : [nitin mittal](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-the-point-where-maximum-intervals-overlap/>

Chapter 120

Find whether an array is subset of another array Added Method 3

Find whether an array is subset of another array Added Method 3 - GeeksforGeeks

Given two arrays: arr1[0..m-1] and arr2[0..n-1]. Find whether arr2[] is a subset of arr1[] or not. Both the arrays are not in sorted order. It may be assumed that elements in both array are distinct.

Examples:

Input: arr1[] = {11, 1, 13, 21, 3, 7}, arr2[] = {11, 3, 7, 1}

Output: arr2[] is a subset of arr1[]

Input: arr1[] = {1, 2, 3, 4, 5, 6}, arr2[] = {1, 2, 4}

Output: arr2[] is a subset of arr1[]

Input: arr1[] = {10, 5, 2, 23, 19}, arr2[] = {19, 5, 3}

Output: arr2[] is not a subset of arr1[]

Method 1 (Simple)

Use two loops: The outer loop picks all the elements of arr2[] one by one. The inner loop linearly searches for the element picked by outer loop. If all elements are found then return 1, else return 0.

C++

```
#include<bits/stdc++.h>

/* Return 1 if arr2[] is a subset of
arr1[] */
bool isSubset(int arr1[], int arr2[],
              int m, int n)
{
```

```
int i = 0;
int j = 0;
for (i = 0; i < n; i++)
{
    for (j = 0; j < m; j++)
    {
        if(arr2[i] == arr1[j])
            break;
    }

    /* If the above inner loop was
    not broken at all then arr2[i]
    is not present in arr1[] */
    if (j == m)
        return 0;
}

/* If we reach here then all
elements of arr2[] are present
in arr1[] */
return 1;
}

// Driver code
int main()
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);

    if(isSubset(arr1, arr2, m, n))
        printf("arr2[] is subset of arr1[] ");
    else
        printf("arr2[] is not a subset of arr1[] ");

    getchar();
    return 0;
}
```

Java

```
// Java program to find whether an array
// is subset of another array

class GFG {
```

```
/* Return true if arr2[] is a subset
of arr1[] */
static boolean isSubset(int arr1[],
                      int arr2[], int m, int n)
{
    int i = 0;
    int j = 0;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
            if(arr2[i] == arr1[j])
                break;

        /* If the above inner loop
        was not broken at all then
        arr2[i] is not present in
        arr1[] */
        if (j == m)
            return false;
    }

    /* If we reach here then all
    elements of arr2[] are present
    in arr1[] */
    return true;
}

// Driver code
public static void main(String args[])
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = arr1.length;
    int n = arr2.length;

    if(isSubset(arr1, arr2, m, n))
        System.out.print("arr2[] is "
                        + "subset of arr1[] ");
    else
        System.out.print("arr2[] is "
                        + "not a subset of arr1[] ");
}
}

// C# program to find whether an array
```

```
// is subset of another array
using System;

class GFG {

    /* Return true if arr2[] is a
    subset of arr1[] */
    static bool isSubset(int []arr1,
                        int []arr2, int m, int n)
    {
        int i = 0;
        int j = 0;
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < m; j++)
                if(arr2[i] == arr1[j])
                    break;

            /* If the above inner loop
            was not broken at all then
            arr2[i] is not present in
            arr1[] */
            if (j == m)
                return false;
        }

        /* If we reach here then all
        elements of arr2[] are present
        in arr1[] */
        return true;
    }

    // Driver function
    public static void Main()
    {
        int []arr1 = {11, 1, 13, 21, 3, 7};
        int []arr2 = {11, 3, 7, 1};

        int m = arr1.Length;
        int n = arr2.Length;

        if(isSubset(arr1, arr2, m, n))
            Console.WriteLine("arr2[] is subset"
                            + " of arr1[] ");
        else
            Console.WriteLine("arr2[] is not a "
                            + "subset of arr1[] ");
    }
}
```

```
}
```

```
// This code is contributed by Sam007
```

PHP

```
<?php
/* Return 1 if arr2[] is a subset of
arr1[] */
function isSubset($arr1, $arr2, $m, $n)
{
    $i = 0;
    $j = 0;
    for ($i = 0; $i < $n; $i++)
    {
        for ($j = 0; $j < $m; $j++)
        {
            if($arr2[$i] == $arr1[$j])
                break;
        }

        /* If the above inner loop was
        not broken at all then arr2[i]
        is not present in arr1[] */
        if ($j == $m)
            return 0;
    }

    /* If we reach here then all
    elements of arr2[] are present
    in arr1[] */
    return 1;
}

/* Driver code
$arr1 = array(11, 1, 13, 21, 3, 7);
$arr2 = array(11, 3, 7, 1);

$m = count($arr1);
$n = count($arr2);

if(isSubset($arr1, $arr2, $m, $n))
    echo "arr2[] is subset of arr1[] ";
else
    echo "arr2[] is not a subset of arr1[] ";

// This code is contributed by anuj_67.
?>
```

Output:

```
arr2[] is subset of arr1[]
```

Time Complexity: $O(m*n)$

Method 2 (Use Sorting and Binary Search)

- 1) Sort arr1[] $O(m \log m)$
- 2) For each element of arr2[], do binary search for it in sorted arr1[].
 - a) If the element is not found then return 0.
- 3) If all elements are present then return 1.

C

```
#include<stdio.h>

/* Function prototypes */
void quickSort(int *arr, int si, int ei);
int binarySearch(int arr[], int low, int high, int x);

/* Return 1 if arr2[] is a subset of arr1[] */
bool isSubset(int arr1[], int arr2[], int m, int n)
{
    int i = 0;

    quickSort(arr1, 0, m-1);
    for (i=0; i<n; i++)
    {
        if (binarySearch(arr1, 0, m-1, arr2[i]) == -1)
            return 0;
    }

    /* If we reach here then all elements of arr2[]
       are present in arr1[] */
    return 1;
}

/* FOLLOWING FUNCTIONS ARE ONLY FOR SEARCHING AND SORTING PURPOSE */
/* Standard Binary Search function*/
int binarySearch(int arr[], int low, int high, int x)
{
    if(high >= low)
    {
        int mid = (low + high)/2; /*low + (high - low)/2;*/
        if(arr[mid] == x)
            return mid;
        else if(arr[mid] < x)
            return binarySearch(arr, mid+1, high, x);
        else
            return binarySearch(arr, low, mid-1, x);
    }
    return -1;
}
```

```
/* Check if arr[mid] is the first occurrence of x.  
   arr[mid] is first occurrence if x is one of the following  
   is true:  
   (i) mid == 0 and arr[mid] == x  
   (ii) arr[mid-1] < x and arr[mid] == x  
*/  
if(( mid == 0 || x > arr[mid-1]) && (arr[mid] == x))  
    return mid;  
else if(x > arr[mid])  
    return binarySearch(arr, (mid + 1), high, x);  
else  
    return binarySearch(arr, low, (mid -1), x);  
}  
return -1;  
}  
  
void exchange(int *a, int *b)  
{  
    int temp;  
    temp = *a;  
    *a    = *b;  
    *b    = temp;  
}  
  
int partition(int A[], int si, int ei)  
{  
    int x = A[ei];  
    int i = (si - 1);  
    int j;  
  
    for (j = si; j <= ei - 1; j++)  
    {  
        if(A[j] <= x)  
        {  
            i++;  
            exchange(&A[i], &A[j]);  
        }  
    }  
    exchange (&A[i + 1], &A[ei]);  
    return (i + 1);  
}  
  
/* Implementation of Quick Sort  
A[] --> Array to be sorted  
si --> Starting index  
ei --> Ending index  
*/
```

```
void quickSort(int A[], int si, int ei)
{
    int pi; /* Partitioning index */
    if(si < ei)
    {
        pi = partition(A, si, ei);
        quickSort(A, si, pi - 1);
        quickSort(A, pi + 1, ei);
    }
}

/*Driver program to test above functions */
int main()
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);

    if(isSubset(arr1, arr2, m, n))
        printf("arr2[] is subset of arr1[] ");
    else
        printf("arr2[] is not a subset of arr1[] ");

    return 0;
}
```

Java

```
class Main
{
    /* Return true if arr2[] is a subset of arr1[] */
    static boolean isSubset(int arr1[], int arr2[], int m, int n)
    {
        int i = 0;

        sort(arr1, 0, m-1);
        for (i=0; i<n; i++)
        {
            if (binarySearch(arr1, 0, m-1, arr2[i]) == -1)
                return false;
        }

        /* If we reach here then all elements of arr2[]
           are present in arr1[] */
        return true;
    }
}
```

```
/* FOLLOWING FUNCTIONS ARE ONLY FOR SEARCHING AND SORTING PURPOSE */
/* Standard Binary Search function*/
static int binarySearch(int arr[], int low, int high, int x)
{
    if(high >= low)
    {
        int mid = (low + high)/2; /*low + (high - low)/2;*/

        /* Check if arr[mid] is the first occurrence of x.
           arr[mid] is first occurrence if x is one of the following
           is true:
           (i) mid == 0 and arr[mid] == x
           (ii) arr[mid-1] < x and arr[mid] == x
        */
        if(( mid == 0 || x > arr[mid-1]) && (arr[mid] == x))
            return mid;
        else if(x > arr[mid])
            return binarySearch(arr, (mid + 1), high, x);
        else
            return binarySearch(arr, low, (mid -1), x);
    }
    return -1;
}

/* This function takes last element as pivot,
   places the pivot element at its correct
   position in sorted array, and places all
   smaller (smaller than pivot) to left of
   pivot and all greater elements to right
   of pivot */
static int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low-1); // index of smaller element
    for (int j=low; j<high; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;

            // swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
}
```

```
}

// swap arr[i+1] and arr[high] (or pivot)
int temp = arr[i+1];
arr[i+1] = arr[high];
arr[high] = temp;

return i+1;
}

/* The main function that implements QuickSort()
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
static void sort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is
           now at right place */
        int pi = partition(arr, low, high);

        // Recursively sort elements before
        // partition and after partition
        sort(arr, low, pi-1);
        sort(arr, pi+1, high);
    }
}

public static void main(String args[])
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = arr1.length;
    int n = arr2.length;

    if(isSubset(arr1, arr2, m, n))
        System.out.print("arr2[] is subset of arr1[] ");
    else
        System.out.print("arr2[] is not a subset of arr1[] ");
}
}
```

Time Complexity: $O(m \log m + n \log m)$. Please note that this will be the complexity if an $m \log m$ algorithm is used for sorting which is not the case in above code. In above code Quick Sort is used and worst case time complexity of Quick Sort is $O(m^2)$

Method 3 (Use Sorting and Merging)

- 1) Sort both arrays: arr1[] and arr2[] O(mLogm + nLogn)
- 2) Use Merge type of process to see if all elements of sorted arr2[] are present in sorted arr1[].

Thanks to **Parthsarthi** for suggesting this method.

C++

```
#include <bits/stdc++.h>
using namespace std;

/* Return 1 if arr2[] is a subset of arr1[] */
bool isSubset(int arr1[], int arr2[], int m, int n)
{
    int i = 0, j = 0;

    if (m < n)
        return 0;

    sort(arr1, arr1 + m);
    sort(arr2, arr2 + n);
    while (i < n && j < m)
    {
        if( arr1[j] <arr2[i] )
            j++;
        else if( arr1[j] == arr2[i] )
        {
            j++;
            i++;
        }
        else if( arr1[j] > arr2[i] )
            return 0;
    }

    return (i < n)? false : true;
}

/*Driver program to test above functions */
int main()
{
    int arr1[] = {11, 1, 13, 21, 3, 7};
    int arr2[] = {11, 3, 7, 1};

    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);

    if(isSubset(arr1, arr2, m, n))
        printf("arr2[] is subset of arr1[] ");
    else
```

```
    printf("arr2[] is not a subset of arr1[] ");

    return 0;
}
```

Java

```
// Java code to find whether an array is subset of
// another array
import java.util.Arrays;
class GFG
{
    /* Return true if arr2[] is a subset of arr1[] */
    static boolean isSubset(int arr1[], int arr2[], int m,
                           int n)
    {
        int i = 0, j = 0;

        if(m < n)
            return false;

        Arrays.sort(arr1); //sorts arr1
        Arrays.sort(arr2); // sorts arr2

        while( i < n && j < m )
        {
            if( arr1[j] < arr2[i] )
                j++;
            else if( arr1[j] == arr2[i] )
            {
                j++;
                i++;
            }
            else if( arr1[j] > arr2[i] )
                return false;
        }

        if( i < n )
            return false;
        else
            return true;
    }

    public static void main(String[] args)
    {
        int arr1[] = {11, 1, 13, 21, 3, 7};
        int arr2[] = {11, 3, 7, 1};
```

```
int m = arr1.length;
int n = arr2.length;

if(isSubset(arr1, arr2, m, n))
    System.out.println("arr2 is a subset of arr1");
else
    System.out.println("arr2 is not a subset of arr1");
}
}

// This code is contributed by Kamal Rawal
```

C#

```
// C# code to find whether an array
// is subset of another array
using System;
class GFG {

    // Return true if arr2[] is
    // a subset of arr1[] */
    static bool isSubset(int []arr1,
                        int []arr2,
                        int m,
                        int n)
    {
        int i = 0, j = 0;

        if(m < n)
            return false;

        //sorts arr1
        Array.Sort(arr1);

        // sorts arr2
        Array.Sort(arr2);

        while( i < n && j < m )
        {
            if( arr1[j] < arr2[i] )
                j++;
            else if( arr1[j] == arr2[i] )
            {
                j++;
                i++;
            }
            else if( arr1[j] > arr2[i] )
                return false;
        }
    }
}
```

```
if( i < n )
    return false;
else
    return true;
}

// Driver Code
public static void Main()
{
    int []arr1 = {11, 1, 13, 21, 3, 7};
    int []arr2 = {11, 3, 7, 1};

    int m = arr1.Length;
    int n = arr2.Length;

    if(isSubset(arr1, arr2, m, n))
        Console.WriteLine("arr2 is a subset of arr1");
    else
        Console.WriteLine("arr2 is not a subset of arr1");
}
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php

/* Return 1 if arr2[] is a subset of arr1[] */
function isSubset( $arr1, $arr2, $m, $n)
{
    $i = 0; $j = 0;

    if ( $m < $n)
        return 0;

    sort($arr1);
    sort($arr2);

    while ( $i < $n and $j < $m )
    {
        if( $arr1[$j] <$arr2[$i] )
            $j++;
        else if( $arr1[$j] == $arr2[$i] )
        {
            $j++;
            $i++;
        }
    }
}
```

```

        }
        else if( $arr1[$j] > $arr2[$i] )
            return 0;
    }

    return ($i < $n) ? false : true;
}

/*Driver program to test above functions */

$arr1 = array(11, 1, 13, 21, 3, 7);
$arr2 = array(11, 3, 7, 1);

$m = count($arr1);
$n = count($arr2);

if(isSubset($arr1, $arr2, $m, $n))
    echo "arr2[] is subset of arr1[] ";
else
    echo "arr2[] is not a subset of arr1[] ";

// This code is contributed by anuj_67.
?>

```

Output:

```
arr2 is a subset of arr1
```

Time Complexity: $O(m\log m + n\log n)$ which is better than method 2. Please note that this will be the complexity if an $n\log n$ algorithm is used for sorting both arrays which is not the case in above code. In above code Quick Sort is used and worst case time complexity of Quick Sort is $O(n^2)$

Method 4 (Use Hashing)

- 1) Create a Hash Table for all the elements of $\text{arr1}[]$.
- 2) Traverse $\text{arr2}[]$ and search for each element of $\text{arr2}[]$ in the Hash Table. If element is not found then return 0.
- 3) If all elements are found then return 1.

```

// Java code to find whether an array is subset of
// another array
import java.util.HashSet;
class GFG
{
    /* Return true if arr2[] is a subset of arr1[] */
    static boolean isSubset(int arr1[], int arr2[], int m,
                           int n)

```

```
{  
    HashSet<Integer> hset= new HashSet<>();  
  
    // hset stores all the values of arr1  
    for(int i = 0; i < m; i++)  
    {  
        if(!hset.contains(arr1[i]))  
            hset.add(arr1[i]);  
    }  
  
    // loop to check if all elements of arr2 also  
    // lies in arr1  
    for(int i = 0; i < n; i++)  
    {  
        if(!hset.contains(arr2[i]))  
            return false;  
    }  
    return true;  
}  
  
public static void main(String[] args)  
{  
    int arr1[] = {11, 1, 13, 21, 3, 7};  
    int arr2[] = {11, 3, 7, 1};  
  
    int m = arr1.length;  
    int n = arr2.length;  
  
    if(isSubset(arr1, arr2, m, n))  
        System.out.println("arr2 is a subset of arr1");  
    else  
        System.out.println("arr2 is not a subset of arr1");  
}  
}  
// This code is contributed by Kamal Rawal
```

Note that method 1, method 2 and method 4 don't handle the cases when we have duplicates in arr2[]. For example, {1, 4, 4, 2} is not a subset of {1, 4, 2}, but these methods will print it as a subset.

Improved By : [Sam007](#), [nitin mittal](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-whether-an-array-is-subset-of-another-array-set-1/>

Chapter 121

Find whether it is possible to make array elements same using one external number

Find whether it is possible to make array elements same using one external number - Geeks-forGeeks

Given an Array, three operations can be performed using any external number x.

Add x to an element once

Subtract x from an element once

Perform no operation on the element

Find whether there exists a number X, such that if the above operations are performed with the number X, the resulting array has equal elements.

If the number exists, print “YES” and the value, space separated, else print “NO”

Examples:

Input : [1, 1, 3, 5, 5]

Output : YES, x = 2

Explanation : The number 2 can be added to the first two elements and can be subtracted from the last two elements to obtain a common element 3 throughout the array

Input : [1, 3, 5, 7, 9]

Output : NO

The idea is to form groups of unique elements from given array. Following cases arise :

1. Count of unique elements is 1. Answer is YES with $x = 0$
2. Count of unique elements is 2. Answer is YES with $x = \text{Difference of two unique elements.}$
3. Count of unique elements is 3.
 - If difference between mid and max is same as difference between mid and min, answer is YES with $x = \text{difference between mid and max or mid and min.}$
 - Otherwise answer is NO.

In Python, we can quickly find unique elements using [set in Python](#).

```
# Program in python 2.x to find an element X
# that can be used to operate on an array and
# get equal elements

# Prints "YES" and an element x if we can
# equalize array using x. Else prints "NO"
def canEqualise(array):

    # We all the unique elements (using set
    # function). Then we sort unique elements.
    uniques = sorted(set(array))

    # if there are only 1 or 2 unique elements,
    # then we can add or subtract x from one of them
    # to get the other element
    if len(uniques) == 1:
        print("YES " + "0")
    elif len(uniques) == 2:
        print("YES " + str(uniques[1] - uniques[0]))

    # If count of unique elements is three, then
    # difference between the middle and minimum
    # should be same as difference between maximum
    # and middle
    elif len(uniques) == 3:
        if uniques[2] - uniques[1] == uniques[1] - uniques[0]:
            X = uniques[2] - uniques[1]
            print("YES " + str(X))
        else:
            print("NO")

    # if there are more than three unique elements, then
    # we cannot add or subtract the same value from all
    # the elements.
    else:
```

```
print("NO")  
  
# Driver code  
array = [55, 52, 52, 49, 52]  
canEqualise(array)
```

Output:

```
YES 3
```

This code has complexity **O(n log n)**

The same problem could be extended to ask for two numbers required to equalize the array. Following the same process, we would require 5 unique elements in the array to require two numbers to equalize the array. So to require n numbers to equalize an array, we would require $(2n + 1)$ unique elements in the array.

Source

<https://www.geeksforgeeks.org/find-whether-possible-make-array-elements-using-one-external-number/>

Chapter 122

Generic Implementation of QuickSort Algorithm in C

Generic Implementation of QuickSort Algorithm in C - GeeksforGeeks

Write a function to implement [quicksort](#) algorithm that will work for all types of data i.e ints, floats, chars etc.

It should accept all types of data and show the sorted data as output.

Note: This function is similar to C standard library function `qsort()`.

Examples:

First Input as a string.

```
Input :abc cad bcd xyz bsd  
Output :abc bcd bsd cad xyz
```

Second input as integer

```
Input :5 6 4 2 3  
Output :2 3 4 5 6
```

We use `void*` to implement generic quicksort function in C. `void*` does not know how much bytes of memory it has to occupy in memory space. It must be casted to any other data type like `int*`, `char*` before doing any operation on it.

Example: when we declare `int var;` compiler knows that it has occupy 4 bytes of memory but `void` does not know how much bytes of memory it has to occupy.

We will also use a pointer to function that will point to a function which is dependent to different types of data i.e and this function will be defined by the user according to there need.

Below is the image representation of `void*` in memory before and after casting it to any particular data type for better understanding.

`Void* pt` in Memory :

```
void* pt casted to char* :  
  
// C Program to illustrate Generic Quicksort Function  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
// function for comparing two strings. This function  
// is passed as a parameter to _quickSort() when we  
// want to sort  
int cmpstr(void* v1, void* v2)  
{  
    // casting v1 to char** and then assigning it to  
    // pointer to v1 as v1 is array of characters i.e  
    // strings.  
    char *a1 = *(char**)v1;  
    char *a2 = *(char**)v2;  
    return strcmp(a1, a2);  
}  
  
// function for comparing two strings  
int cmpnum(void* s1, void* s2)  
{  
    // casting s1 to int* so it can be  
    // copied in variable a.  
    int *a = (int*)s1;  
    int *b = (int*)s2;  
    if ((*a) > (*b))  
        return 1;  
    else if ((*a) < (*b))  
        return -1;  
    else  
        return 0;  
}  
  
/* you can also write compare function for floats,  
   chars, double similarly as integer. */  
// function for swap two elements  
void swap(void* v1, void* v2, int size)  
{  
    // buffer is array of characters which will  
    // store element byte by byte  
    char buffer[size];  
  
    // memcpy will copy the contents from starting  
    // address of v1 to length of size in buffer  
    // byte by byte.  
    memcpy(buffer, v1, size);
```

```
    memcpy(v1, v2, size);
    memcpy(v2, buffer, size);
}

// v is an array of elements to sort.
// size is the number of elements in array
// left and right is start and end of array
// (*comp)(void*, void*) is a pointer to a function
// which accepts two void* as its parameter
void _qsort(void* v, int size, int left, int right,
            int (*comp)(void*, void*))
{
    void *vt, *v3;
    int i, last, mid = (left + right) / 2;
    if (left >= right)
        return;

    // casting void* to char* so that operations
    // can be done.
    void* vl = (char*)(v + (left * size));
    void* vr = (char*)(v + (mid * size));
    swap(vl, vr, size);
    last = left;
    for (i = left + 1; i <= right; i++) {

        // vl and vt will have the starting address
        // of the elements which will be passed to
        // comp function.
        vt = (char*)(v + (i * size));
        if ((*comp)(vl, vt) > 0) {
            ++last;
            v3 = (char*)(v + (last * size));
            swap(vt, v3, size);
        }
    }
    v3 = (char*)(v + (last * size));
    swap(vl, v3, size);
    _qsort(v, size, left, last - 1, comp);
    _qsort(v, size, last + 1, right, comp);
}

int main()
{
    // Your C Code
    char* a[] = {"bbc", "xcd", "ede", "def",
                 "afg", "hello", "hmmm", "okay", "how" };

    int b[] = { 45, 78, 89, 65, 70, 23, 44 };
}
```

```
int* p = b;
_qsort(a, sizeof(char*), 0, 8, (int (*)(void*, void*))(cmpstr));
_qsort(p, sizeof(int), 0, 6, (int (*)(void*, void*))(cmpnum));

for (int i = 0; i < 9; i++)
    printf("%s ", a[i]);
printf("\n");

for (int i = 0; i < 7; i++)
    printf("%d ", b[i]);
return 0;
}
```

Output:

```
afg bbc def ede hello hmmm how okay xcd
23 44 45 65 70 78 89
```

Source

<https://www.geeksforgeeks.org/generic-implementation-of-quicksort-algorithm-in-c/>

Chapter 123

Given a number, find the next smallest palindrome

Given a number, find the next smallest palindrome - GeeksforGeeks

Given a number, find the next smallest palindrome larger than this number. For example, if the input number is “2 3 5 4 5”, the output should be “2 3 6 3 2”. And if the input number is “9 9 9”, the output should be “1 0 0 1”.

The input is assumed to be an array. Every entry in array represents a digit in input number. Let the array be ‘num[]’ and size of array be ‘n’

There can be three different types of inputs that need to be handled separately.

1) The input number is palindrome and has all 9s. For example “9 9 9”. Output should be “1 0 0 1”

2) The input number is not palindrome. For example “1 2 3 4”. Output should be “1 3 3 1”

3) The input number is palindrome and doesn’t have all 9s. For example “1 2 2 1”. Output should be “1 3 3 1”.

Solution for input type 1 is easy. The output contains $n + 1$ digits where the corner digits are 1, and all digits between corner digits are 0.

Now let us first talk about input type 2 and 3. How to convert a given number to a greater palindrome? To understand the solution, let us first define the following two terms:

Left Side: The left half of given number. Left side of “1 2 3 4 5 6” is “1 2 3” and left side of “1 2 3 4 5” is “1 2”

Right Side: The right half of given number. Right side of “1 2 3 4 5 6” is “4 5 6” and right side of “1 2 3 4 5” is “4 5”

To convert to palindrome, we can either take the mirror of its left side or take mirror of its right side. However, if we take the mirror of the right side, then the palindrome so formed is not guaranteed to be next larger palindrome. So, we must take the mirror of left side and copy it to right side. But there are some cases that must be handled in different ways. See the following steps.

We will start with two indices i and j. i pointing to the two middle elements (or pointing

to two elements around the middle element in case of n being odd). We one by one move i and j away from each other.

Step 1. Initially, ignore the part of left side which is same as the corresponding part of right side. For example, if the number is “8 3 4 2 2 4 6 9 , we ignore the middle four elements. i now points to element 3 and j now points to element 6.

Step 2. After step 1, following cases arise:

Case 1: Indices i & j cross the boundary.

This case occurs when the input number is palindrome. In this case, we just add 1 to the middle digit (or digits in case n is even) propagate the carry towards MSB digit of left side and simultaneously copy mirror of the left side to the right side.

For example, if the given number is “1 2 9 2 1”, we increment 9 to 10 and propagate the carry. So the number becomes “1 3 0 3 1”

Case 2: There are digits left between left side and right side which are not same. So, we just mirror the left side to the right side & try to minimize the number formed to guarantee the next smallest palindrome.

In this case, there can be **two sub-cases**.

2.1) Copying the left side to the right side is sufficient, we don't need to increment any digits and the result is just mirror of left side. Following are some examples of this sub-case.

Next palindrome for “7 8 3 3 2 2 is “7 8 3 3 8 7”

Next palindrome for “1 2 5 3 2 2 is “1 2 5 5 2 1”

Next palindrome for “1 4 5 8 7 6 7 8 3 2 2 is “1 4 5 8 7 6 7 8 5 4 1”

How do we check for this sub-case? All we need to check is the digit just after the ignored part in step 1. This digit is highlighted in above examples. If this digit is greater than the corresponding digit in right side digit, then copying the left side to the right side is sufficient and we don't need to do anything else.

2.2) Copying the left side to the right side is NOT sufficient. This happens when the above defined digit of left side is smaller. Following are some examples of this case.

Next palindrome for “7 1 3 3 2 2 is “7 1 4 4 1 7”

Next palindrome for “1 2 3 4 6 2 8 is “1 2 3 5 3 2 1”

Next palindrome for “9 4 1 8 7 9 7 8 3 2 2 is “9 4 1 8 8 0 8 8 1 4 9”

We handle this subcase like Case 1. We just add 1 to the middle digit (or digits in case n is even) propagate the carry towards MSB digit of left side and simultaneously copy mirror of the left side to the right side.

C++

```
#include <stdio.h>

// A utility function to print an array
void printArray (int arr[], int n);

// A utility function to check if num has all 9s
int AreAll9s (int num[], int n );

// Returns next palindrome of a given number num[] .
```

```
// This function is for input type 2 and 3
void generateNextPalindromeUtil (int num[], int n )
{
    // find the index of mid digit
    int mid = n/2;

    // A bool variable to check if copy of left side to right is sufficient or not
    bool leftsmaller = false;

    // end of left side is always 'mid -1'
    int i = mid - 1;

    // Begining of right side depends if n is odd or even
    int j = (n % 2)? mid + 1 : mid;

    // Initially, ignore the middle same digits
    while (i >= 0 && num[i] == num[j])
        i--,j++;

    // Find if the middle digit(s) need to be incremented or not (or copying left
    // side is not sufficient)
    if ( i < 0 || num[i] < num[j])
        leftsmaller = true;

    // Copy the mirror of left to right
    while (i >= 0)
    {
        num[j] = num[i];
        j++;
        i--;
    }

    // Handle the case where middle digit(s) must be incremented.
    // This part of code is for CASE 1 and CASE 2.2
    if (leftsmaller == true)
    {
        int carry = 1;
        i = mid - 1;

        // If there are odd digits, then increment
        // the middle digit and store the carry
        if (n%2 == 1)
        {
            num[mid] += carry;
            carry = num[mid] / 10;
            num[mid] %= 10;
            j = mid + 1;
        }
    }
}
```

```
    else
        j = mid;

    // Add 1 to the rightmost digit of the left side, propagate the carry
    // towards MSB digit and simultaneously copying mirror of the left side
    // to the right side.
    while (i >= 0)
    {
        num[i] += carry;
        carry = num[i] / 10;
        num[i] %= 10;
        num[j++] = num[i--]; // copy mirror to right
    }
}

// The function that prints next palindrome of a given number num[]
// with n digits.
void generateNextPalindrome( int num[], int n )
{
    int i;

    printf("Next palindrome is:");

    // Input type 1: All the digits are 9, simply o/p 1
    // followed by n-1 0's followed by 1.
    if( AreAll9s( num, n ) )
    {
        printf( "1 " );
        for( i = 1; i < n; i++ )
            printf( "0 " );
        printf( "1" );
    }

    // Input type 2 and 3
    else
    {
        generateNextPalindromeUtil ( num, n );

        // print the result
        printArray (num, n);
    }
}

// A utility function to check if num has all 9s
int AreAll9s( int* num, int n )
{
    int i;
```

```
for( i = 0; i < n; ++i )
    if( num[i] != 9 )
        return 0;
return 1;
}

/* Utility that prints out an array on a line */
void printArray(int arr[], int n)
{
    int i;
    for (i=0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver Program to test above function
int main()
{
    int num[] = {9, 4, 1, 8, 7, 9, 7, 8, 3, 2, 2};

    int n = sizeof (num)/ sizeof(num[0]);

    generateNextPalindrome( num, n );

    return 0;
}
```

Java

```
// Java program to find next smallest
// palindrome

public class nextplaindrome
{
    // Returns next palindrome of a given
    // number num[]. This function is for
    // input type 2 and 3
    static void generateNextPalindromeUtil(int num[], int n)
    {
        int mid = n / 2;

        // end of left side is always 'mid -1'
        int i = mid - 1;

        // Begining of right side depends
        // if n is odd or even
        int j = (n % 2 == 0) ? mid : mid + 1;
```

```
// A bool variable to check if copy of left
// side to right
// is sufficient or not
boolean leftsmaller = false;

// Initially, ignore the middle same digits
while (i >= 0 && num[i] == num[j])
{
    i--;
    j++;
}

// Find if the middle digit(s) need to
// be incremented or not (or copying left
// side is not sufficient)
if (i < 0 || num[i] < num[j])
{
    leftsmaller = true;
}

// Copy the mirror of left to tight
while (i >= 0)
{
    num[j++] = num[i--];
}

// Handle the case where middle digit(s)
// must be incremented. This part of code
// is for CASE 1 and CASE 2.2
if (leftsmaller)
{
    int carry = 1;

    // If there are odd digits, then increment
    // the middle digit and store the carry
    if (n % 2 == 1) {
        num[mid] += 1;
        carry = num[mid] / 10;
        num[mid] %= 10;
    }
    i = mid - 1;
    j = (n % 2 == 0 ? mid : mid + 1);

    // Add 1 to the rightmost digit of the left
    // side, propagate the carry towards MSB digit
    // and simultaneously copying mirror of the
    // left side to the right side.
    while (i >= 0)
```

```
{  
    num[i] = num[i] + carry;  
    carry = num[i] / 10;  
    num[i] %= 10;  
    num[j] = num[i];// copy mirror to right  
    i--;  
    j++;  
}  
  
}  
  
// The function that prints next palindrome  
// of a given number num[] with n digits.  
static void generateNextPalindrome(int num[], int n)  
{  
    System.out.println("Next Palindrome is:");  
  
    // Input type 1: All the digits are 9,  
    // simply o/p 1 followed by n-1 0's  
    // followed by 1.  
    if (isAll9(num, n)) {  
        System.out.print("1");  
        for (int i = 0; i < n - 1; i++)  
            System.out.print("0");  
        System.out.println("1");  
    }  
  
    // Input type 2 and 3  
    else {  
        generateNextPalindromeUtil(num, n);  
        printarray(num);  
    }  
}  
  
// A utility function to check if num has all 9s  
static boolean isAll9(int num[], int n) {  
    for (int i = 0; i < n; i++)  
        if (num[i] != 9)  
            return false;  
    return true;  
}  
  
/* Utility that prints out an array on a line */  
static void printarray(int num[]) {  
    for (int i = 0; i < num.length; i++)  
        System.out.print(num[i]);
```

```
        System.out.println();
    }

    public static void main(String[] args)
    {
        int num[] = { 9, 4, 1, 8, 7, 9, 7, 8, 3, 2, 2 };
        generateNextPalindrome(num, num.length);
    }
}
```

Python3

```
# Returns next palindrome of a given number num[] .
# This function is for input type 2 and 3
def generateNextPalindromeUtil (num, n) :

    # find the index of mid digit
    mid = int(n/2 )

    # A bool variable to check if copy of left
    # side to right is sufficient or not
    leftsmaller = False

    # end of left side is always 'mid -1'
    i = mid - 1

    # Begining of right side depends
    # if n is odd or even
    j = mid + 1 if (n % 2) else mid

    # Initially, ignore the middle same digits
    while (i >= 0 and num[i] == num[j]) :
        i-=1
        j+=1

    # Find if the middle digit(s) need to be
    # incremented or not (or copying left
    # side is not sufficient)
    if ( i < 0 or num[i] < num[j]):
        leftsmaller = True

    # Copy the mirror of left to right
    while (i >= 0) :

        num[j] = num[i]
        j+=1
        i-=1
```

```
# Handle the case where middle
# digit(s) must be incremented.
# This part of code is for CASE 1 and CASE 2.2
if (leftsmaller == True) :

    carry = 1
    i = mid - 1

    # If there are odd digits, then increment
    # the middle digit and store the carry
    if (n%2 == 1) :

        num[mid] += carry
        carry = int(num[mid] / 10 )
        num[mid] %= 10
        j = mid + 1

    else:
        j = mid

    # Add 1 to the rightmost digit of the
    # left side, propagate the carry
    # towards MSB digit and simultaneously
    # copying mirror of the left side
    # to the right side.
    while (i >= 0) :

        num[i] += carry
        carry = num[i] / 10
        num[i] %= 10
        num[j] = num[i] # copy mirror to right
        j+=1
        i-=1

# The function that prints next
# palindrome of a given number num[]
# with n digits.
def generateNextPalindrome(num, n ) :

    print("\nNext palindrome is:")

    # Input type 1: All the digits are 9, simply o/p 1
    # followed by n-1 0's followed by 1.
    if( AreAll9s( num, n ) == True) :

        print( "1")
        for i in range(1, n):
```

```
        print( "0" )
        print( "1" )

# Input type 2 and 3
else:

    generateNextPalindromeUtil ( num, n )

    # print the result
    printArray (num, n)

# A utility function to check if num has all 9s
def AreAll9s(num, n):
    for i in range(1, n):
        if( num[i] != 9 ) :
            return 0
    return 1

# Utility that prints out an array on a line
def printArray(arr, n):

    for i in range(0, n):
        print(int(arr[i]),end=" ")
    print()

# Driver Program to test above function
if __name__ == "__main__":
    num = [9, 4, 1, 8, 7, 9, 7, 8, 3, 2, 2]
    n = len(num)
    generateNextPalindrome( num, n )

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// C# program to find next smallest palindrome
using System;
public class GFG {

    // Returns next palindrome of a given
    // number num[]. This function is for
    // input type 2 and 3
    static void generateNextPalindromeUtil(int []num, int n)
    {
        int mid = n / 2;
```

```
// end of left side is always 'mid -1'  
int i = mid - 1;  
  
// Begining of right side depends  
// if n is odd or even  
int j = (n % 2 == 0) ? mid : mid + 1;  
  
// A bool variable to check if copy of left  
// side to right  
// is sufficient or not  
bool leftsmaller = false;  
  
// Initially, ignore the middle same digits  
while (i >= 0 && num[i] == num[j])  
{  
    i--;  
    j++;  
}  
  
// Find if the middle digit(s) need to  
// be incremented or not (or copying left  
// side is not sufficient)  
if (i < 0 || num[i] < num[j])  
{  
    leftsmaller = true;  
}  
  
// Copy the mirror of left to right  
while (i >= 0)  
{  
    num[j++] = num[i--];  
}  
  
// Handle the case where middle digit(s)  
// must be incremented. This part of code  
// is for CASE 1 and CASE 2.2  
if (leftsmaller)  
{  
    int carry = 1;  
  
    // If there are odd digits, then increment  
    // the middle digit and store the carry  
    if (n % 2 == 1) {  
        num[mid] += 1;  
        carry = num[mid] / 10;  
        num[mid] %= 10;  
    }  
}
```

```
i = mid - 1;
j = (n % 2 == 0 ? mid : mid + 1);

// Add 1 to the rightmost digit of the left
// side, propagate the carry towards MSB digit
// and simultaneously copying mirror of the
// left side to the right side.
while (i >= 0)
{
    num[i] = num[i] + carry;
    carry = num[i] / 10;
    num[i] %= 10;
    num[j] = num[i];// copy mirror to right
    i--;
    j++;
}

}

// The function that prints next palindrome
// of a given number num[] with n digits.
static void generateNextPalindrome(int []num, int n)
{
    Console.WriteLine("Next Palindrome is:");

    // Input type 1: All the digits are 9,
    // simply o/p 1 followed by n-1 0's
    // followed by 1.
    if (isAll9(num, n)) {
        Console.Write("1");
        for (int i = 0; i < n - 1; i++)
            Console.Write("0");
        Console.Write("1");
    }

    // Input type 2 and 3
    else {
        generateNextPalindromeUtil(num, n);
        printarray(num);
    }
}

// A utility function to check if num has all 9s
static bool isAll9(int[] num, int n) {
    for (int i = 0; i < n; i++)
        if (num[i] != 9)
```

```
        return false;
    return true;
}

/* Utility that prints out an array on a line */
static void printarray(int []num) {
    for (int i = 0; i < num.Length; i++)
        Console.Write(num[i]+ " ");
    Console.WriteLine(" ");
}

// Driver code
public static void Main()
{
    int []num = { 9, 4, 1, 8, 7, 9, 7, 8, 3, 2, 2 };
    generateNextPalindrome(num, num.Length);
}
}

// This code is contributed by Smitha.
```

PHP

```
<?php
// PHP program to find next
// smallest palindrome

// Returns next palindrome
// of a given number num[].
// This function is for
// input type 2 and 3
function generateNextPalindromeUtil($num, $n)
{
    $mid = (int)($n / 2);

    // end of left side
    // is always 'mid -1'
    $i = $mid - 1;

    // Begining of right
    // side depends if n
    // is odd or even
    $j = ($n % 2 == 0) ?
        $mid : ($mid + 1);

    // A bool variable to check
    // if copy of left side to
    // right is sufficient or not
```

```
$leftsmaller = false;

// Initially, ignore the
// middle same digits
while ($i >= 0 &&
       $num[$i] == $num[$j])
{
    $i--;
    $j++;
}

// Find if the middle digit(s)
// need to be incremented or
// not (or copying left side
// is not sufficient)
if ($i < 0 || $num[$i] < $num[$j])
{
    $leftsmaller = true;
}

// Copy the mirror
// of left to tight
while ($i >= 0)
{
    $num[$j++] = $num[$i--];
}

// Handle the case where
// middle digit(s) must be
// incremented. This part
// of code is for CASE 1
// and CASE 2.2
if ($leftsmaller)
{
    $carry = 1;

    // If there are odd digits,
    // then increment the middle
    // digit and store the carry
    if ($n % 2 == 1)
    {
        $num[$mid] += 1;
        $carry = (int)($num[$mid] / 10);
        $num[$mid] %= 10;
    }
    $i = $mid - 1;
    $j = ($n % 2 == 0 ?
          $mid : $mid + 1);
}
```

```
// Add 1 to the rightmost digit
// of the left side, propagate
// the carry towards MSB digit
// and simultaneously copying
// mirror of the left side to
// the right side.
while ($i >= 0)
{
    $num[$i] = $num[$i] + $carry;
    $carry = (int)($num[$i] / 10);
    $num[$i] %= 10;

    // copy mirror to right
    $num[$j] = $num[$i];
    $i--;
    $j++;
}

return $num;
}

// The function that prints
// next palindrome of a given
// number num[] with n digits.
function generateNextPalindrome($num, $n)
{
    echo "Next Palindrome is:\n";

    // Input type 1: All the
    // digits are 9, simply
    // o/p 1 followed by n-1
    // 0's followed by 1.
    if (isAll9($num, $n))
    {
        echo "1";
        for ($i = 0; $i < $n - 1; $i++)
            echo "0";
        echo "1";
    }

    // Input type 2 and 3
    else
    {
        $num = generateNextPalindromeUtil($num, $n);
        printarray($num);
```

```
    }
}

// A utility function to
// check if num has all 9s
function isAll9($num, $n)
{
    for ($i = 0; $i < $n; $i++)
        if ($num[$i] != 9)
            return false;
    return true;
}

/* Utility that prints out
an array on a line */
function printarray($num)
{
    for ($i = 0;
        $i < count($num); $i++)
        echo $num[$i];
    echo "\n";
}

// Driver code
$num = array(9, 4, 1, 8, 7,
            9, 7, 8, 3, 2, 2);
generateNextPalindrome($num,
                      count($num));

// This code is contributed by mits.
?>
```

Output:

```
Next palindrome is:
9 4 1 8 8 0 8 8 1 4 9
```

Improved By : [Smitha Dinesh Semwal](#), [Mithun Kumar](#), [hritikgupta](#), [ankush_007](#)

Source

<https://www.geeksforgeeks.org/given-a-number-find-next-smallest-palindrome-larger-than-this-number/>

Chapter 124

Given a sorted array and a number x, find the pair in array whose sum is closest to x

Given a sorted array and a number x, find the pair in array whose sum is closest to x - GeeksforGeeks

Given a sorted array and a number x, find a pair in array whose sum is closest to x.

Examples:

Input: arr[] = {10, 22, 28, 29, 30, 40}, x = 54
Output: 22 and 30

Input: arr[] = {1, 3, 4, 7, 10}, x = 15
Output: 4 and 10

A simple solution is to consider every pair and keep track of closest pair (absolute difference between pair sum and x is minimum). Finally print the closest pair. Time complexity of this solution is $O(n^2)$

An efficient solution can find the pair in $O(n)$ time. The idea is similar to method 2 of [this post](#). Following is detailed algorithm.

- 1) Initialize a variable diff as infinite (Diff is used to store the difference between pair and x). We need to find the minimum diff.
- 2) Initialize two index variables l and r in the given sorted array.
 - (a) Initialize first to the leftmost index: l = 0
 - (b) Initialize second the rightmost index: r = n-1
- 3) Loop while l < r.

- (a) If $\text{abs}(\text{arr}[l] + \text{arr}[r] - \text{sum}) < \text{diff}$ then
update diff and result
- (b) Else if($\text{arr}[l] + \text{arr}[r] < \text{sum}$) then $l++$
- (c) Else $r--$

Following is the implementation of above algorithm.

C++

```
// Simple C++ program to find the pair with sum closest to a given no.
#include <iostream>
#include <climits>
#include <cstdlib>
using namespace std;

// Prints the pair with sum closest to x
void printClosest(int arr[], int n, int x)
{
    int res_l, res_r; // To store indexes of result pair

    // Initialize left and right indexes and difference between
    // pair sum and x
    int l = 0, r = n-1, diff = INT_MAX;

    // While there are elements between l and r
    while (r > l)
    {
        // Check if this pair is closer than the closest pair so far
        if (abs(arr[l] + arr[r] - x) < diff)
        {
            res_l = l;
            res_r = r;
            diff = abs(arr[l] + arr[r] - x);
        }

        // If this pair has more sum, move to smaller values.
        if (arr[l] + arr[r] > x)
            r--;
        else // Move to larger values
            l++;
    }

    cout << " The closest pair is " << arr[res_l] << " and " << arr[res_r];
}

// Driver program to test above functions
int main()
{
```

```
int arr[] = {10, 22, 28, 29, 30, 40}, x = 54;
int n = sizeof(arr)/sizeof(arr[0]);
printClosest(arr, n, x);
return 0;
}
```

Java

```
// Java program to find pair with sum closest to x
import java.io.*;
import java.util.*;
import java.lang.Math;

class CloseSum {

    // Prints the pair with sum closest to x
    static void printClosest(int arr[], int n, int x)
    {
        int res_l=0, res_r=0; // To store indexes of result pair

        // Initialize left and right indexes and difference between
        // pair sum and x
        int l = 0, r = n-1, diff = Integer.MAX_VALUE;

        // While there are elements between l and r
        while (r > l)
        {
            // Check if this pair is closer than the closest pair so far
            if (Math.abs(arr[l] + arr[r] - x) < diff)
            {
                res_l = l;
                res_r = r;
                diff = Math.abs(arr[l] + arr[r] - x);
            }

            // If this pair has more sum, move to smaller values.
            if (arr[l] + arr[r] > x)
                r--;
            else // Move to larger values
                l++;
        }

        System.out.println(" The closest pair is "+arr[res_l]+" and "+ arr[res_r]);
    }

    // Driver program to test above function
    public static void main(String[] args)
```

```
{  
    int arr[] = {10, 22, 28, 29, 30, 40}, x = 54;  
    int n = arr.length;  
    printClosest(arr, n, x);  
}  
/*This code is contributed by Devesh Agrawal*/
```

Python3

```
# Python3 program to find the pair  
# with sum  
# closest to a given no.  
  
# A sufficiently large value greater  
# than any  
# element in the input array  
MAX_VAL = 1000000000  
  
#Prints the pair with sum closest to x  
  
def printClosest(arr, n, x):  
  
    # To store indexes of result pair  
    res_l, res_r = 0, 0  
  
    #Initialize left and right indexes  
    # and difference between  
    # pair sum and x  
    l, r, diff = 0, n-1, MAX_VAL  
  
    # While there are elements between l and r  
    while r > l:  
        # Check if this pair is closer than the  
        # closest pair so far  
        if abs(arr[l] + arr[r] - x) < diff:  
            res_l = l  
            res_r = r  
            diff = abs(arr[l] + arr[r] - x)  
  
        if arr[l] + arr[r] > x:  
            # If this pair has more sum, move to  
            # smaller values.  
            r -= 1  
        else:  
            # Move to larger values  
            l += 1
```

```
print('The closest pair is {} and {}'
      .format(arr[res_l], arr[res_r]))\n\n# Driver code to test above
if __name__ == "__main__":
    arr = [10, 22, 28, 29, 30, 40]
    n = len(arr)
    x=54
    printClosest(arr, n, x)\n\n# This code is contributed by Tuhin Patra
```

C#

```
// C# program to find pair with sum closest to x
using System;\n\nclass GFG {\n\n    // Prints the pair with sum closest to x
    static void printClosest(int []arr, int n, int x)
    {\n\n        // To store indexes of result pair
        int res_l = 0, res_r = 0;\n\n        // Initialize left and right indexes and
        // difference between pair sum and x
        int l = 0, r = n-1, diff = int.MaxValue;\n\n        // While there are elements between l and r
        while (r > l)
        {\n\n            // Check if this pair is closer than the
            // closest pair so far
            if (Math.Abs(arr[l] + arr[r] - x) < diff)
            {
                res_l = l;
                res_r = r;
                diff = Math.Abs(arr[l] + arr[r] - x);
            }\n\n            // If this pair has more sum, move to
            // smaller values.
            if (arr[l] + arr[r] > x)
```

```
        r--;
        else // Move to larger values
        l++;
    }

    Console.WriteLine(" The closest pair is " +
                      arr[res_l] + " and " + arr[res_r]);
}

// Driver program to test above function
public static void Main()
{
    int []arr = {10, 22, 28, 29, 30, 40};
    int x = 54;
    int n = arr.Length;

    printClosest(arr, n, x);
}
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// Simple PHP program to find the
// pair with sum closest to a
// given no.

// Prints the pair with
// sum closest to x
function printClosest($arr, $n, $x)
{

    // To store indexes
    // of result pair
    $res_l;
    $res_r;

    // Initialize left and right
    // indexes and difference between
    // pair sum and x
    $l = 0;
    $r = $n - 1;
    $diff = PHP_INT_MAX;

    // While there are elements
    // between l and r
```

```
while ($r > $l)
{
    // Check if this pair is closer
    // than the closest pair so far
    if (abs($arr[$l] + $arr[$r] - $x) <
        $diff)
    {
        $res_l = $l;
        $res_r = $r;
        $diff = abs($arr[$l] + $arr[$r] - $x);
    }

    // If this pair has more sum,
    // move to smaller values.
    if ($arr[$l] + $arr[$r] > $x)
        $r--;
    else
        $l++;
}

echo " The closest pair is "
    , $arr[$res_l] , " and "
    , $arr[$res_r];
}

// Driver Code
$arr = array(10, 22, 28, 29, 30, 40);
$x = 54;
$n = count($arr);
printClosest($arr, $n, $x);

// This code is contributed by anuj_67.
?>
```

Output:

```
The closest pair is 22 and 30
```

This article is contributed by **Harsh**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [nitin mittal](#), [vt_m](#), [TuhinPatra](#)

Chapter 124. Given a sorted array and a number x, find the pair in array whose sum is closest to x

Source

<https://www.geeksforgeeks.org/given-sorted-array-number-x-find-pair-array-whose-sum-closest-x/>

Chapter 125

Given a sorted dictionary of an alien language, find order of characters

Given a sorted dictionary of an alien language, find order of characters - GeeksforGeeks

Given a sorted dictionary (array of words) of an alien language, find order of characters in the language.

Examples:

Input: words[] = {"baa", "abcd", "abca", "cab", "cad"}

Output: Order of characters is 'b', 'd', 'a', 'c'

Note that words are sorted and in the given language "baa" comes before "abcd", therefore 'b' is before 'a' in output.
Similarly we can find other orders.

Input: words[] = {"caa", "aaa", "aab"}

Output: Order of characters is 'c', 'a', 'b'

The idea is to create a graph of characters and then find [topological sorting](#) of the created graph. Following are the detailed steps.

- 1) Create a graph g with number of vertices equal to the size of alphabet in the given alien language. For example, if the alphabet size is 5, then there can be 5 characters in words. Initially there are no edges in graph.
- 2) Do following for every pair of adjacent words in given sorted array.
 -a) Let the current pair of words be $word1$ and $word2$. One by one compare characters of both words and find the first mismatching characters.
 -b) Create an edge in g from mismatching character of $word1$ to that of $word2$.
- 3) Print [topological sorting](#) of the above created graph.

Following is the implementation of the above algorithm.

C++

```
// A C++ program to order of characters in an alien language
#include<iostream>
#include <list>
#include <stack>
#include <cstring>
using namespace std;

// Class to represent a graph
class Graph
{
    int V;      // No. of vertices'

    // Pointer to an array containing adjacency listsList
    list<int> *adj;

    // A function used by topologicalSort
    void topologicalSortUtil(int v, bool visited[], stack<int> &Stack);
public:
    Graph(int V);    // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w);

    // prints a Topological Sort of the complete graph
    void topologicalSort();
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

// A recursive function used by topologicalSort
void Graph::topologicalSortUtil(int v, bool visited[], stack<int> &Stack)
{
    // Mark the current node as visited.
    visited[v] = true;

    // Recur for all the vertices adjacent to this vertex
```

```
list<int>::iterator i;
for (i = adj[v].begin(); i != adj[v].end(); ++i)
    if (!visited[*i])
        topologicalSortUtil(*i, visited, Stack);

// Push current vertex to stack which stores result
Stack.push(v);
}

// The function to do Topological Sort. It uses recursive topologicalSortUtil()
void Graph::topologicalSort()
{
    stack<int> Stack;

    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    // Call the recursive helper function to store Topological Sort
    // starting from all vertices one by one
    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            topologicalSortUtil(i, visited, Stack);

    // Print contents of stack
    while (Stack.empty() == false)
    {
        cout << (char) ('a' + Stack.top()) << " ";
        Stack.pop();
    }
}

int min(int x, int y)
{
    return (x < y)? x : y;
}

// This function finds and prints order of character from a sorted
// array of words. n is size of words[]. alpha is set of possible
// alphabets.
// For simplicity, this function is written in a way that only
// first 'alpha' characters can be there in words array. For
// example if alpha is 7, then words[] should have only 'a', 'b',
// 'c', 'd', 'e', 'f', 'g'
void printOrder(string words[], int n, int alpha)
{
    // Create a graph with 'alpha' edges
```

```
Graph g(alpha);

// Process all adjacent pairs of words and create a graph
for (int i = 0; i < n-1; i++)
{
    // Take the current two words and find the first mismatching
    // character
    string word1 = words[i], word2 = words[i+1];
    for (int j = 0; j < min(word1.length(), word2.length()); j++)
    {
        // If we find a mismatching character, then add an edge
        // from character of word1 to that of word2
        if (word1[j] != word2[j])
        {
            g.addEdge(word1[j]-'a', word2[j]-'a');
            break;
        }
    }
}

// Print topological sort of the above created graph
g.topologicalSort();
}

// Driver program to test above functions
int main()
{
    string words[] = {"caa", "aaa", "aab"};
    printOrder(words, 3, 3);
    return 0;
}
```

Java

```
// A Java program to order of
// characters in an alien language
import java.util.*;

// Class to represent a graph
class Graph
{

    // An array representing the graph as an adjacency list
    private final LinkedList<Integer>[] adjacencyList;

    Graph(int nVertices)
    {
        adjacencyList = new LinkedList[nVertices];
```

```
for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++)
{
    adjacencyList[vertexIndex] = new LinkedList<>();
}
}

// function to add an edge to graph
void addEdge(int startVertex, int endVertex)
{
    adjacencyList[startVertex].add(endVertex);
}

private int getNo0fVertices()
{
    return adjacencyList.length;
}

// A recursive function used by topologicalSort
private void topologicalSortUtil(int currentVertex, boolean[] visited,
                                 Stack<Integer> stack)
{
    // Mark the current node as visited.
    visited[currentVertex] = true;

    // Recur for all the vertices adjacent to this vertex
    for (int adjacentVertex : adjacencyList[currentVertex])
    {
        if (!visited[adjacentVertex])
        {
            topologicalSortUtil(adjacentVertex, visited, stack);
        }
    }

    // Push current vertex to stack which stores result
    stack.push(currentVertex);
}

// prints a Topological Sort of the complete graph
void topologicalSort()
{
    Stack<Integer> stack = new Stack<>();

    // Mark all the vertices as not visited
    boolean[] visited = new boolean[getNo0fVertices()];
    for (int i = 0; i < getNo0fVertices(); i++)
    {
        visited[i] = false;
    }
}
```

```
// Call the recursive helper function to store Topological
// Sort starting from all vertices one by one
for (int i = 0; i < getNoOfVertices(); i++)
{
    if (!visited[i])
    {
        topologicalSortUtil(i, visited, stack);
    }
}

// Print contents of stack
while (!stack.isEmpty())
{
    System.out.print((char)('a' + stack.pop()) + " ");
}
}

public class OrderOfCharacters
{
    // This function finds and prints order
    // of character from a sorted array of words.
    // alpha is number of possible alphabets
    // starting from 'a'. For simplicity, this
    // function is written in a way that only
    // first 'alpha' characters can be there
    // in words array. For example if alpha
    // is 7, then words[] should contain words
    // having only 'a', 'b', 'c', 'd', 'e', 'f', 'g'
    private static void printOrder(String[] words, int alpha)
    {
        // Create a graph with 'alpha' edges
        Graph graph = new Graph(alpha);

        for (int i = 0; i < words.length - 1; i++)
        {
            // Take the current two words and find the first mismatching
            // character
            String word1 = words[i];
            String word2 = words[i+1];
            for (int j = 0; j < Math.min(word1.length(), word2.length()); j++)
            {
                // If we find a mismatching character, then add an edge
                // from character of word1 to that of word2
                if (word1.charAt(j) != word2.charAt(j))
                {
                    graph.addEdge(word1.charAt(j) - 'a', word2.charAt(j)- 'a');
```

```
        break;
    }
}
}

// Print topological sort of the above created graph
graph.topologicalSort();
}

// Driver program to test above functions
public static void main(String[] args)
{
    String[] words = {"caa", "aaa", "aab"};
    printOrder(words, 3);
}
}

//Contributed by Harikrishnan Rajan
```

Output:

c a b

Time Complexity: The first step to create a graph takes $O(n + \alpha)$ time where n is number of given words and α is number of characters in given alphabet. The second step is also topological sorting. Note that there would be α vertices and at-most $(n-1)$ edges in the graph. The time complexity of [topological sorting](#) is $O(V+E)$ which is $O(n + \alpha)$ here. So overall time complexity is $O(n + \alpha) + O(n + \alpha)$ which is $O(n + \alpha)$.

Exercise:

The above code doesn't work when the input is not valid. For example {"aba", "bba", "aaa"} is not valid, because from first two words, we can deduce 'a' should appear before 'b', but from last two words, we can deduce 'b' should appear before 'a' which is not possible. Extend the above program to handle invalid inputs and generate the output as "Not valid".

This article is contributed by **Piyush Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

<https://www.geeksforgeeks.org/given-sorted-dictionary-find-precedence-characters/>

Chapter 126

Gnome Sort

Gnome Sort - GeeksforGeeks

Gnome Sort also called Stupid sort is based on the concept of a Garden Gnome sorting his flower pots. A garden gnome sorts the flower pots by the following method-

- He looks at the flower pot next to him and the previous one; if they are in the right order he steps one pot forward, otherwise he swaps them and steps one pot backwards.
- If there is no previous pot (he is at the starting of the pot line), he steps forwards; if there is no pot next to him (he is at the end of the pot line), he is done.

Input –

```
Array- arr[]
Total elements - n
```

Algorithm Steps

1. If you are at the start of the array then go to the right element (from arr[0] to arr[1]).
2. If the current array element is larger or equal to the previous array element then go one step right

```
if (arr[i] >= arr[i-1])
    i++;
```

3. If the current array element is smaller than the previous array element then swap these two elements and go one step backwards

```
if (arr[i] < arr[i-1])
{
    swap(arr[i], arr[i-1]);
    i--;
}
```

4. Repeat steps 2) and 3) till ‘i’ reaches the end of the array (i.e- ‘n-1’)
5. If the end of the array is reached then stop and the array is sorted.

Example-

34 2 10 -9

- **Underlined elements** are the pair under consideration.
- “**Red**” colored are the pair which needs to be swapped.
- Result of the swapping is colored as “**blue**”

34 2 10 -9
2 34 10 -9
2 34 10 -9
2 10 34 -9
2 10 34 -9
2 10 34 -9
2 10 -9 34
2 10 -9 34
2 -9 10 34
2 -9 10 34
-9 2 10 34
-9 2 10 34
-9 2 10 34
-9 2 10 34(Sorted output)

Below is the implementation of the algorithm.

C++

```
// A C++ Program to implement Gnome Sort
#include <iostream>
using namespace std;

// A function to sort the algorithm using gnome sort
void gnomeSort(int arr[], int n)
{
    int index = 0;

    while (index < n) {
        if (index == 0)
            index++;
        if (arr[index] >= arr[index - 1])
            index++;
        else {
            swap(arr[index], arr[index - 1]);
            index--;
        }
    }
    return;
}

// A utility function ot print an array of size n
void printArray(int arr[], int n)
{
    cout << "Sorted sequence after Gnome sort: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << "\n";
}

// Driver program to test above functions.
int main()
{
    int arr[] = { 34, 2, 10, -9 };
    int n = sizeof(arr) / sizeof(arr[0]);

    gnomeSort(arr, n);
    printArray(arr, n);

    return (0);
}
```

Java

```
// Java Program to implement Gnome Sort

import java.util.Arrays;
public class GFG {
    static void gnomeSort(int arr[], int n)
    {
        int index = 0;

        while (index < n) {
            if (index == 0)
                index++;
            if (arr[index] >= arr[index - 1])
                index++;
            else {
                int temp = 0;
                temp = arr[index];
                arr[index] = arr[index - 1];
                arr[index - 1] = temp;
                index--;
            }
        }
        return;
    }

    // Driver program to test above functions.
    public static void main(String[] args)
    {
        int arr[] = { 34, 2, 10, -9 };

        gnomeSort(arr, arr.length);

        System.out.print("Sorted sequence after applying Gnome sort: ");
        System.out.println(Arrays.toString(arr));
    }
}

// Code Contributed by Mohit Gupta_OMG
```

Python

```
# Python program to implement Gnome Sort

# A function to sort the given list using Gnome sort
def gnomeSort( arr, n):
    index = 0
    while index < n:
        if index == 0:
            index = index + 1
```

```
if arr[index] >= arr[index - 1]:
    index = index + 1
else:
    arr[index], arr[index-1] = arr[index-1], arr[index]
    index = index - 1

return arr

# Driver Code
arr = [ 34, 2, 10, -9]
n = len(arr)

arr = gnomeSort(arr, n)
print "Sorted sequence after applying Gnome Sort :",
for i in arr:
    print i,
```

Contributed By Harshit Agrawal

C#

```
// C# Program to implement Gnome Sort
using System;

class GFG {

    static void gnomeSort(int[] arr, int n)
    {
        int index = 0;

        while (index < n)
        {
            if (index == 0)
                index++;
            if (arr[index] >= arr[index - 1])
                index++;
            else {
                int temp = 0;
                temp = arr[index];
                arr[index] = arr[index - 1];
                arr[index - 1] = temp;
                index--;
            }
        }
        return;
    }

    // Driver program to test above functions.
```

```
public static void Main()
{
    int[] arr = { 34, 2, 10, -9 };

    // Function calling
    gnomeSort(arr, arr.Length);

    Console.WriteLine("Sorted sequence after applying Gnome sort: ");

    for (int i = 0; i < arr.Length; i++)
        Console.Write(arr[i] + " ");
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP Program to implement
// Gnome Sort

// A function to sort the
// algorithm using gnome sort
function gnomeSort($arr, $n)
{
    $index = 0;

    while ($index < $n)
    {
        if ($index == 0)
            $index++;
        if ($arr[$index] >= $arr[$index - 1])
            $index++;
        else
        {
            $temp = 0;
            $temp = $arr[$index];
            $arr[$index] = $arr[$index - 1];
            $arr[$index - 1] = $temp;
            $index--;
        }
    }
    echo "Sorted sequence ",
        "after Gnome sort: ";
    for ($i = 0; $i < $n; $i++)
        echo $arr[$i] . " ";
    echo "\n";
```

```
}
```

```
// Driver Code
$arr = array(34, 2, 10, -9);
$n = count($arr);

gnomeSort($arr, $n);

// This code is contributed
// by Sam007
?>
```

Output:

```
Sorted sequence after applying Gnome sort: -9 2 10 34
```

Time Complexity – As there are no nested loop (only one while) it may seem that this is a linear $O(N)$ time algorithm. But the time complexity is $O(N^2)$. This is because the variable – ‘index’ in our program doesn’t always gets incremented, it gets decremented too. *However this sorting algorithm is adaptive and performs better if the array is already/partially sorted.*

Auxiliary Space – This is an in-place algorithm. So $O(1)$ auxiliary space is needed.

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/gnome-sort-a-stupid-one/>

Chapter 127

Heap Sort for decreasing order using min heap

Heap Sort for decreasing order using min heap - GeeksforGeeks

Given an array of elements, sort the array in decreasing order using min heap.

Input : arr[] = {5, 3, 10, 1}
Output : arr[] = {10, 5, 3, 1}

Input : arr[] = {1, 50, 100, 25}
Output : arr[] = {100, 50, 25, 1}

Prerequisite : [Heap sort using min heap](#).

Algorithm :

1. Build a min heap from the input data.
2. At this point, the smallest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
3. Repeat above steps while size of heap is greater than 1.

Note :Heap Sort using min heap sorts in descending order where as max heap sorts in ascending order

C++

```
// C++ program for implementation of Heap Sort
#include <iostream>
using namespace std;

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
```

```
void heapify(int arr[], int n, int i)
{
    int smallest = i; // Initialize smalles as root
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2

    // If left child is smaller than root
    if (l < n && arr[l] < arr[smallest])
        smallest = l;

    // If right child is smaller than smallest so far
    if (r < n && arr[r] < arr[smallest])
        smallest = r;

    // If smallest is not root
    if (smallest != i) {
        swap(arr[i], arr[smallest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, smallest);
    }
}

// main function to do heap sort
void heapSort(int arr[], int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i = n - 1; i >= 0; i--) {
        // Move current root to end
        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

/* A utility function to print array of size n */
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}
```

```
// Driver program
int main()
{
    int arr[] = { 4, 6, 3, 2, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is \n";
    printArray(arr, n);
}
```

Java

```
// Java program for implementation of Heap Sort

import java.io.*;

class GFG {

    // To heapify a subtree rooted with node i which is
    // an index in arr[]. n is size of heap
    static void heapify(int arr[], int n, int i)
    {
        int smallest = i; // Initialize smalles as root
        int l = 2 * i + 1; // left = 2*i + 1
        int r = 2 * i + 2; // right = 2*i + 2

        // If left child is smaller than root
        if (l < n && arr[l] < arr[smallest])
            smallest = l;

        // If right child is smaller than smallest so far
        if (r < n && arr[r] < arr[smallest])
            smallest = r;

        // If smallest is not root
        if (smallest != i) {
            int temp = arr[i];
            arr[i] = arr[smallest];
            arr[smallest] = temp;

            // Recursively heapify the affected sub-tree
            heapify(arr, n, smallest);
        }
    }

    // main function to do heap sort
```

```
static void heapSort(int arr[], int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i = n - 1; i >= 0; i--) {

        // Move current root to end
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

/* A utility function to print array of size n */
static void printArray(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

// Driver program
public static void main(String[] args)
{
    int arr[] = { 4, 6, 3, 2, 9 };
    int n = arr.length;

    heapSort(arr, n);

    System.out.println("Sorted array is ");
    printArray(arr, n);
}
}

// This code is contributed by vt_m.
```

C#

```
// C# program for implementation of Heap Sort
using System;

class GFG {
```

```
// To heapify a subtree rooted with
// node i which is an index in arr[],
// n is size of heap
static void heapify(int[] arr, int n, int i)
{
    int smallest = i; // Initialize smalles as root
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2

    // If left child is smaller than root
    if (l < n && arr[l] < arr[smallest])
        smallest = l;

    // If right child is smaller than smallest so far
    if (r < n && arr[r] < arr[smallest])
        smallest = r;

    // If smallest is not root
    if (smallest != i) {
        int temp = arr[i];
        arr[i] = arr[smallest];
        arr[smallest] = temp;

        // Recursively heapify the affected sub-tree
        heapify(arr, n, smallest);
    }
}

// main function to do heap sort
static void heapSort(int[] arr, int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i = n - 1; i >= 0; i--) {

        // Move current root to end
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}
```

```
/* A utility function to print array of size n */
static void printArray(int[] arr, int n)
{
    for (int i = 0; i < n; ++i)
        Console.Write(arr[i] + " ");
    Console.WriteLine();
}

// Driver program
public static void Main()
{
    int[] arr = { 4, 6, 3, 2, 9 };
    int n = arr.Length;

    heapSort(arr, n);

    Console.WriteLine("Sorted array is ");
    printArray(arr, n);
}
}

// This code is contributed by vt_m.
```

Output:

```
Sorted array is
9 6 4 3 2
```

Time complexity:It takes $O(\log n)$ for heapify and $O(n)$ for constructing a heap. Hence, the overall time complexity of heap sort using min heap or max heap is $O(n \log n)$

Source

<https://www.geeksforgeeks.org/heap-sort-for-decreasing-order-using-min-heap/>

Chapter 128

HeapSort

HeapSort - GeeksforGeeks

Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

What is [Binary Heap](#)?

Let us first define a Complete Binary Tree. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible (Source [Wikipedia](#))

A [Binary Heap](#) is a Complete Binary Tree where items are stored in a special order such that value in a parent node is greater(or smaller) than the values in its two children nodes. The former is called as max heap and the latter is called min heap. The heap can be represented by binary tree or array.

Why array based representation for [Binary Heap](#)?

Since a Binary Heap is a Complete Binary Tree, it can be easily represented as array and array based representation is space efficient. If the parent node is stored at index I, the left child can be calculated by $2 * I + 1$ and right child by $2 * I + 2$ (assuming the indexing starts at 0).

Heap Sort Algorithm for sorting in increasing order:

1. Build a max heap from the input data.
2. At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
3. Repeat above steps while size of heap is greater than 1.

How to build the heap?

Heapify procedure can be applied to a node only if its children nodes are heapified. So the heapification must be performed in the bottom up order.

Lets understand with the help of an example:

```
Input data: 4, 10, 3, 5, 1
        4(0)
        /   \
    10(1)   3(2)
    /   \
5(3)   1(4)
```

The numbers in bracket represent the indices in the array representation of data.

Applying heapify procedure to index 1:

```
        4(0)
        /   \
    10(1)   3(2)
    /   \
5(3)   1(4)
```

Applying heapify procedure to index 0:

```
        10(0)
        /   \
    5(1)   3(2)
    /   \
4(3)   1(4)
```

The heapify procedure calls itself recursively to build heap in top down manner.

C++

```
// C++ program for implementation of Heap Sort
#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
    int largest = i; // Initialize largest as root
    int l = 2*i + 1; // left = 2*i + 1
    int r = 2*i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;
```

```
// If largest is not root
if (largest != i)
{
    swap(arr[i], arr[largest]);

    // Recursively heapify the affected sub-tree
    heapify(arr, n, largest);
}
}

// main function to do heap sort
void heapSort(int arr[], int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i=n-1; i>=0; i--)
    {
        // Move current root to end
        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

/* A utility function to print array of size n */
void printArray(int arr[], int n)
{
    for (int i=0; i<n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

// Driver program
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is \n";
    printArray(arr, n);
}
```

Java

```
// Java program for implementation of Heap Sort
public class HeapSort
{
    public void sort(int arr[])
    {
        int n = arr.length;

        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i=n-1; i>=0; i--)
        {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i, 0);
        }
    }

    // To heapify a subtree rooted with node i which is
    // an index in arr[]. n is size of heap
    void heapify(int arr[], int n, int i)
    {
        int largest = i; // Initialize largest as root
        int l = 2*i + 1; // left = 2*i + 1
        int r = 2*i + 2; // right = 2*i + 2

        // If left child is larger than root
        if (l < n && arr[l] > arr[largest])
            largest = l;

        // If right child is larger than largest so far
        if (r < n && arr[r] > arr[largest])
            largest = r;

        // If largest is not root
        if (largest != i)
        {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
        }
    }
}
```

```

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

/* A utility function to print array of size n */
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i]+" ");
    System.out.println();
}

// Driver program
public static void main(String args[])
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = arr.length;

    HeapSort ob = new HeapSort();
    ob.sort(arr);

    System.out.println("Sorted array is");
    printArray(arr);
}
}

```

Python

```

# Python program for implementation of heap Sort

# To heapify subtree rooted at index i.
# n is size of heap
def heapify(arr, n, i):
    largest = i # Initialize largest as root
    l = 2 * i + 1      # left = 2*i + 1
    r = 2 * i + 2      # right = 2*i + 2

    # See if left child of root exists and is
    # greater than root
    if l < n and arr[i] < arr[l]:
        largest = l

    # See if right child of root exists and is
    # greater than root
    if r < n and arr[largest] < arr[r]:

```

```

largest = r

# Change root, if needed
if largest != i:
    arr[i],arr[largest] = arr[largest],arr[i] # swap

# Heapify the root.
heapify(arr, n, largest)

# The main function to sort an array of given size
def heapSort(arr):
    n = len(arr)

    # Build a maxheap.
    for i in range(n, -1, -1):
        heapify(arr, n, i)

    # One by one extract elements
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i] # swap
        heapify(arr, i, 0)

# Driver code to test above
arr = [ 12, 11, 13, 5, 6, 7]
heapSort(arr)
n = len(arr)
print ("Sorted array is")
for i in range(n):
    print ("%d" %arr[i]),
# This code is contributed by Mohit Kumra

```

C#

```

// C# program for implementation of Heap Sort
using System;

public class HeapSort
{
    public void sort(int[] arr)
    {
        int n = arr.Length;

        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i=n-1; i>=0; i--)

```

```

{
    // Move current root to end
    int temp = arr[0];
    arr[0] = arr[i];
    arr[i] = temp;

    // call max heapify on the reduced heap
    heapify(arr, i, 0);
}
}

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int[] arr, int n, int i)
{
    int largest = i; // Initialize largest as root
    int l = 2*i + 1; // left = 2*i + 1
    int r = 2*i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i)
    {
        int swap = arr[i];
        arr[i] = arr[largest];
        arr[largest] = swap;

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

/* A utility function to print array of size n */
static void printArray(int[] arr)
{
    int n = arr.Length;
    for (int i=0; i<n; ++i)
        Console.Write(arr[i]+ " ");
    Console.Read();
}

```

```

// Driver program
public static void Main()
{
    int[] arr = {12, 11, 13, 5, 6, 7};
    int n = arr.Length;

    HeapSort ob = new HeapSort();
    ob.sort(arr);

    Console.WriteLine("Sorted array is");
    printArray(arr);
}
}

// This code is contributed
// by Akanksha Rai(Addy_akku)

```

PHP

```

<?php

// Php program for implementation of Heap Sort

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
function heapify(&$arr, $n, $i)
{
    $largest = $i; // Initialize largest as root
    $l = 2*$i + 1; // left = 2*i + 1
    $r = 2*$i + 2; // right = 2*i + 2

    // If left child is larger than root
    if ($l < $n && $arr[$l] > $arr[$largest])
        $largest = $l;

    // If right child is larger than largest so far
    if ($r < $n && $arr[$r] > $arr[$largest])
        $largest = $r;

    // If largest is not root
    if ($largest != $i)
    {
        $swap = $arr[$i];
        $arr[$i] = $arr[$largest];
        $arr[$largest] = $swap;

        // Recursively heapify the affected sub-tree
        heapify($arr, $n, $largest);
    }
}

```

```

        }

    }

// main function to do heap sort
function heapSort(&$arr, $n)
{
    // Build heap (rearrange array)
    for ($i = $n / 2 - 1; $i >= 0; $i--)
        heapify($arr, $n, $i);

    // One by one extract an element from heap
    for ($i = $n-1; $i >= 0; $i--)
    {
        // Move current root to end
        $temp = $arr[0];
        $arr[0] = $arr[$i];
        $arr[$i] = $temp;

        // call max heapify on the reduced heap
        heapify($arr, $i, 0);
    }
}

/* A utility function to print array of size n */
function printArray(&$arr, $n)
{
    for ($i = 0; $i < $n; ++$i)
        echo ($arr[$i]. " ");

}

// Driver program
$arr = array(12, 11, 13, 5, 6, 7);
$n = sizeof($arr)/sizeof($arr[0]);

heapSort($arr, $n);

echo 'Sorted array is ' . "\n";
printArray($arr , $n);

// This code is contributed by Shivi_Agarwal
?>

```

Output:

Sorted array is

5 6 7 11 12 13

[Here](#) is previous C code for reference.

Notes:

Heap sort is an in-place algorithm.

Its typical implementation is not stable, but can be made stable (See [this](#))

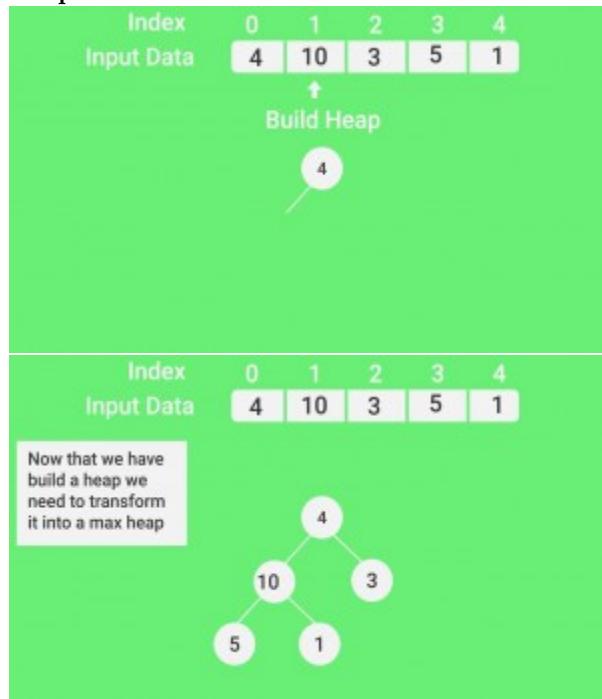
Time Complexity: Time complexity of heapify is $O(\log n)$. Time complexity of createAndBuildHeap() is $O(n)$ and overall time complexity of Heap Sort is $O(n \log n)$.

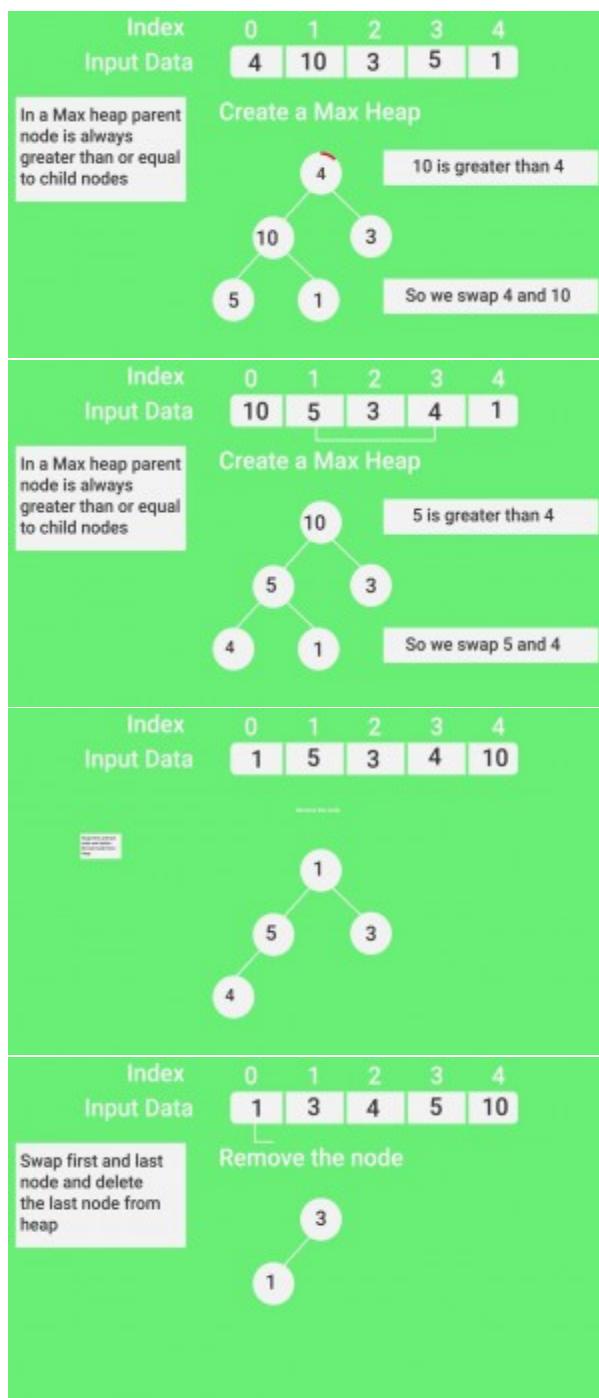
Applications of HeapSort

1. Sort a nearly sorted (or K sorted) array
2. k largest(or smallest) elements in an array

Heap sort algorithm has limited uses because Quicksort and Mergesort are better in practice. Nevertheless, the Heap data structure itself is enormously used. See [Applications of Heap Data Structure](#)

Snapshots:





Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

[QuickSort](#), [Selection Sort](#), [Bubble Sort](#), [Insertion Sort](#), [Merge Sort](#), [Heap Sort](#), [QuickSort](#), [Radix Sort](#), [Counting Sort](#), [Bucket Sort](#), [ShellSort](#), [Comb Sort](#), [Pigeonhole Sort](#)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : [Shivi_Aggarwal](#), [Abby_akku](#)

Source

<https://www.geeksforgeeks.org/heap-sort/>

Chapter 129

Hoare's vs Lomuto partition scheme in QuickSort

Hoare's vs Lomuto partition scheme in QuickSort - GeeksforGeeks

We have discussed implementation of [QuickSort using Lomuto partition scheme](#). Lomuto's partition scheme is easy to implement as compare to Hoare scheme.

Lomuto's Partition Scheme

```
partition(arr[], lo, hi)
    pivot = arr[hi]
    i = lo      // place for swapping
    for j := lo to hi - 1 do
        if arr[j] <= pivot then
            swap arr[i] with arr[j]
            i = i + 1
    swap arr[i] with arr[hi]
    return i
```

Refer [QuickSort](#) for details of this partitioning scheme.

Below are implementations of this approach:-

C++

```
/* C++ implementation QuickSort using Lomuto's partition
   Scheme.*/
#include<bits/stdc++.h>
using namespace std;

/* This function takes last element as pivot, places
   the pivot element at its correct position in sorted
```

```

array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition(int arr[], int low, int high)
{
    int pivot = arr[high];      // pivot
    int i = (low - 1); // Index of smaller element

    for (int j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;      // increment index of smaller element
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

```

```
}
```

```
// Driver program to test above functions
int main()
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    quickSort(arr, 0, n-1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Java

```
// Java implementation QuickSort
// using Lomuto's partition Scheme
import java.io.*;

class GFG
{
    static void Swap(int[] array,
                    int position1,
                    int position2)
    {
        // Swaps elements in an array

        // Copy the first position's element
        int temp = array[position1];

        // Assign to the second element
        array[position1] = array[position2];

        // Assign to the first element
        array[position2] = temp;
    }

    /* This function takes last element as
    pivot, places the pivot element at its
    correct position in sorted array, and
    places all smaller (smaller than pivot)
    to left of pivot and all greater elements
    to right of pivot */
    static int partition(int []arr, int low,
                        int high)
    {
        int pivot = arr[high];
```

```
// Index of smaller element
int i = (low - 1);

for (int j = low; j <= high - 1; j++)
{
    // If current element is smaller
    // than or equal to pivot
    if (arr[j] <= pivot)
    {
        i++; // increment index of
              // smaller element
        Swap(arr, i, j);
    }
}
Swap(arr, i + 1, high);
return (i + 1);
}

/* The main function that
   implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
static void quickSort(int []arr, int low,
                      int high)
{
    if (low < high)
    {
        /* pi is partitioning index,
           arr[p] is now at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
static void printArray(int []arr, int size)
{
    int i;
    for (i = 0; i < size; i++)
        System.out.print(" " + arr[i]);
    System.out.println();
}
```

```
// Driver Code
static public void main (String[] args)
{
    int []arr = {10, 7, 8, 9, 1, 5};
    int n = arr.length;
    quickSort(arr, 0, n-1);
    System.out.println("Sorted array: ");
    printArray(arr, n);
}
}

// This code is contributed by vt_m.
```

C#

```
// C# implementation QuickSort
// using Lomuto's partition Scheme
using System;

class GFG
{
    static void Swap(int[] array,
                    int position1,
                    int position2)
    {
        // Swaps elements in an array

        // Copy the first position's element
        int temp = array[position1];

        // Assign to the second element
        array[position1] = array[position2];

        // Assign to the first element
        array[position2] = temp;
    }

    /* This function takes last element as
    pivot, places the pivot element at its
    correct position in sorted array, and
    places all smaller (smaller than pivot)
    to left of pivot and all greater elements
    to right of pivot */
    static int partition(int []arr, int low,
                        int high)
    {
        int pivot = arr[high];
```

```
// Index of smaller element
int i = (low - 1);

for (int j = low; j <= high - 1; j++)
{
    // If current element is smaller
    // than or equal to pivot
    if (arr[j] <= pivot)
    {
        i++; // increment index of
              // smaller element
        Swap(arr, i, j);
    }
}
Swap(arr, i + 1, high);
return (i + 1);
}

/* The main function that
   implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
static void quickSort(int []arr, int low,
                      int high)
{
    if (low < high)
    {
        /* pi is partitioning index,
           arr[p] is now at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
static void printArray(int []arr, int size)
{
    int i;
    for (i = 0; i < size; i++)
        Console.Write(" " + arr[i]);
    Console.WriteLine();
}
```

```
// Driver Code
static public void Main()
{
    int []arr = {10, 7, 8, 9, 1, 5};
    int n = arr.Length;
    quickSort(arr, 0, n-1);
    Console.WriteLine("Sorted array: ");
    printArray(arr, n);
}
}

// This code is contributed by vt_m.
```

Output:

```
Sorted array:
1 5 7 8 9 10
```

Hoare's Partition Scheme

Hoare's Partition Scheme works by initializing two indexes that start at two ends, the two indexes move toward each other until an inversion is (A smaller value on left side and greater value on right side) found. When an inversion is found, two values are swapped and process is repeated.

It is implemented in below manner:

```
partition(arr[], lo, hi)
pivot = arr[lo]
i = lo - 1 // Initialize left index
j = hi + 1 // Initialize right index

// Find a value in left side greater
// than pivot
do
    i = i + 1
    while arr[i] < pivot

    if i >= j then
        return j

    swap arr[i] with arr[j]
```

Below are implementations of this approach:-
C++

```
/* C++ implementation of QuickSort using Hoare's
partition scheme. */
#include<bits/stdc++.h>
using namespace std;

/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition(int arr[], int low, int high)
{
    int pivot = arr[low];
    int i = low - 1, j = high + 1;

    while (true)
    {
        // Find leftmost element greater than
        // or equal to pivot
        do
        {
            i++;
        } while (arr[i] < pivot);

        // Find rightmost element smaller than
        // or equal to pivot
        do
        {
            j--;
        } while (arr[j] > pivot);

        // If two pointers met.
        if (i >= j)
            return j;

        swap(arr[i], arr[j]);
    }
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
```

```
        at right place */
int pi = partition(arr, low, high);

// Separately sort elements before
// partition and after partition
quickSort(arr, low, pi);
quickSort(arr, pi + 1, high);
}

}

/* Function to print an array */
void printArray(int arr[], int n)
{
    for (int i=0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    quickSort(arr, 0, n-1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Java

```
// Java implementation of QuickSort
// using Hoare's partition scheme
import java.io.*;

class GFG
{

/* This function takes last element
as pivot, places the pivot element
at its correct position in sorted
array, and places all smaller
(smaller than pivot) to left of pivot
and all greater elements to right
of pivot */
static int partition(int []arr, int low,
                     int high)
{
```

```
int pivot = arr[low];
int i = low - 1, j = high + 1;

while (true)
{
    // Find leftmost element greater
    // than or equal to pivot
    do
    {
        i++;
    } while (arr[i] < pivot);

    // Find rightmost element smaller
    // than or equal to pivot
    do
    {
        j--;
    } while (arr[j] > pivot);

    // If two pointers met.
    if (i >= j)
        return j;
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    //swap(arr[i], arr[j]);
}

/* The main function that
   implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
static void quickSort(int []arr, int low,
                      int high)
{
    if (low < high)
    {
        /* pi is partitioning index,
           arr[p] is now at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi);
        quickSort(arr, pi + 1, high);
    }
}
```

```
}

/* Function to print an array */
static void printArray(int []arr, int n)
{
    for (int i=0; i < n; i++)
        System.out.print(" " + arr[i]);
    System.out.println();
}

// Driver Code
static public void main (String[] args)
{
    int []arr = {10, 7, 8, 9, 1, 5};
    int n = arr.length;
    quickSort(arr, 0, n - 1);
    System.out.println("Sorted array: ");
    printArray(arr, n);
}
}

// This code is contributed by vt_m.
```

C#

```
// C# implementation of QuickSort
// using Hoare's partition scheme
using System;

class GFG
{

    /* This function takes last element
    as pivot, places the pivot element
    at its correct position in sorted
    array, and places all smaller
    (smaller than pivot) to left of pivot
    and all greater elements to right
    of pivot */
    static int partition(int []arr, int low,
                        int high)
    {
        int pivot = arr[low];
        int i = low - 1, j = high + 1;

        while (true)
        {
            // Find leftmost element greater
```

```

// than or equal to pivot
do
{
    i++;
} while (arr[i] < pivot);

// Find rightmost element smaller
// than or equal to pivot
do
{
    j--;
} while (arr[j] > pivot);

// If two pointers met.
if (i >= j)
    return j;
int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
//swap(arr[i], arr[j]);
}

/*
 * The main function that
 * implements QuickSort
 * arr[] --> Array to be sorted,
 * low --> Starting index,
 * high --> Ending index */
static void quickSort(int []arr, int low,
                      int high)
{
    if (low < high)
    {
        /* pi is partitioning index,
         * arr[p] is now at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
static void printArray(int []arr, int n)
{
    for (int i=0; i < n; i++)

```

```
        Console.Write(" " + arr[i]);
        Console.WriteLine();
    }

// Driver Code
static public void Main()
{
    int []arr = {10, 7, 8, 9, 1, 5};
    int n = arr.Length;
    quickSort(arr, 0, n - 1);
    Console.WriteLine("Sorted array: ");
    printArray(arr, n);
}
}

// This code is contributed by vt_m.
```

Output:

```
Sorted array:
1 5 7 8 9 10
```

Comparison:

1. Hoare's scheme is more efficient than Lomuto's partition scheme because it does three times fewer swaps on average, and it creates efficient partitions even when all values are equal.
2. Like Lomuto's partition scheme, Hoare partitioning also causes Quicksort to degrade to $O(n^2)$ when the input array is already sorted, it also doesn't produce a stable sort.
3. Note that in this scheme, the pivot's final location is not necessarily at the index that was returned, and the next two segments that the main algorithm recurs on are (lo..p) and (p+1..hi) as opposed to (lo..p-1) and (p+1..hi) as in Lomuto's scheme.

Source : https://en.wikipedia.org/wiki/Quicksort#Hoare_partition_scheme

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/>

Chapter 130

How to efficiently sort a big list dates in 20's

How to efficiently sort a big list dates in 20's - GeeksforGeeks

Given a big list of dates in 20's, how to efficiently sort the list.

Example:

Input:

```
Date arr[] = {{20, 1, 2014},  
              {25, 3, 2010},  
              {3, 12, 2000},  
              {18, 11, 2001},  
              {19, 4, 2015},  
              {9, 7, 2005}}
```

Output:

```
Date arr[] = {{3, 12, 2000},  
              {18, 11, 2001},  
              {9, 7, 2005},  
              {25, 3, 2010},  
              {20, 1, 2014},  
              {19, 4, 2015}}
```

We strongly recommend to minimize your browser and try this yourself first.

A Simple Solution is to use a $O(n\log n)$ algorithm like [Merge Sort](#). We can sort the list in $O(n)$ time using [Radix Sort](#). In a typical Radix Sort implementation, we first sort by last digit, then by second last digit, and so on. Here we sort in following order.

- 1) First sort by day using counting sort
- 2) Then sort by month using counting sort
- 3) Finally sort by year using counting sort

As the number of days, months and years are fixed, all three steps take $O(n)$ time. Therefore, overall time complexity is $O(n)$.

Below is C++ implementation of above idea. Note that the implementation doesn't do any error handing to keep the code simple.

```
// C++ program to sort an array of dates using Radix Sort
#include <bits/stdc++.h>
struct Date
{
    int d, m, y;
};

// Prototypes
void countSortDay(Date arr[], int n);
void countSortMonth(Date arr[], int n);
void countSortYear(Date arr[], int n);

// The main function that sorts array of dates
// using Radix Sort
void radixSortDates(Date arr[], int n)
{
    // First sort by day
    countSortDay(arr, n);

    // Then by month
    countSortMonth(arr, n);

    // Finally by year
    countSortYear(arr, n);
}

// A function to do counting sort of arr[] according to
// day
void countSortDay(Date arr[], int n)
{
    Date output[n]; // output array
    int i, count[31] = {0};

    // Store count of occurrences in count[]
    for (i=0; i<n; i++)
        count[arr[i].d - 1]++;

    // Change count[i] so that count[i] now contains
    // actual position of this day in output[]
    for (i=1; i<31; i++)
        count[i] += count[i-1];

    // Build the output array
    for (i=n-1; i>=0; i--)
    {
        output[count[arr[i].d - 1] - 1] = arr[i];
        count[arr[i].d - 1]--;
    }
}
```

```
// Copy the output array to arr[], so that arr[] now
// contains sorted numbers according to current digit
for (i=0; i<n; i++)
arr[i] = output[i];
}

// A function to do counting sort of arr[] according to
// month.
void countSortMonth(Date arr[], int n)
{
Date output[n]; // output array
int i, count[12] = {0};

for (i = 0; i < n; i++)
count[arr[i].m - 1]++;
for (i = 1; i < 12; i++)
count[i] += count[i - 1];
for (i=n-1; i>=0; i--)
{
output[count[arr[i].m - 1] - 1] = arr[i];
count[arr[i].m - 1]--;
}
for (i = 0; i < n; i++)
arr[i] = output[i];
}

// A function to do counting sort of arr[] according to
// year.
void countSortYear(Date arr[], int n)
{
Date output[n]; // output array
int i, count[1000] = {0};
for (i = 0; i < n; i++)
count[arr[i].y - 2000]++;
for (i = 1; i < 1000; i++)
count[i] += count[i - 1];
for (i = n - 1; i >= 0; i--)
{
output[count[arr[i].y - 2000] - 1] = arr[i];
count[arr[i].y - 2000]--;
}
for (i = 0; i < n; i++)
arr[i] = output[i];
}

// A utility function to print an array
void print(Date arr[], int n)
{
int i;
for (i=0; i<n; i++)
```

```
printf("%2d, %2d, %d\n",
arr[i].d, arr[i].m, arr[i].y);
}

// Driver program to test above functions
int main()
{
Date arr[] = {{20, 1, 2014}, {25, 3, 2010},
{3, 12, 2000}, {18, 11, 2001},
{19, 4, 2015}, {9, 7, 2005}};
int n = sizeof(arr)/sizeof(arr[0]);
printf("Input Dates\n");
print(arr, n);
radixSortDates(arr, n);
printf("\nSorted Dates\n");
print(arr, n);
return 0;
}
```

Output:

```
Input Dates
{20, 1, 2014}
{25, 3, 2010}
{3, 12, 2000}
{18, 11, 2001}
{19, 4, 2015}
{9, 7, 2005}
```

```
Sorted Dates
{3, 12, 2000}
{18, 11, 2001}
{9, 7, 2005}
{25, 3, 2010}
{20, 1, 2014}
{19, 4, 2015}
```

This article is contributed by **Rajeev**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/how-to-efficiently-sort-a-big-list-dates-in-20s/>

Chapter 131

How to make Mergesort to perform O(n) comparisons in best case?

How to make Mergesort to perform O(n) comparisons in best case? - GeeksforGeeks

As we know, [Mergesort](#) is a [divide and conquer algorithm](#) that splits the array to halves recursively until it reaches an array of the size of 1, and after that it merges sorted subarrays until the original array is fully sorted. [Typical implementation of merge sort](#) works in $O(n \log n)$ time in all three cases (best, average and worst).

We need to reduce the best case performance from $O(n \log n)$ to $O(n)$.

The idea is to consider the case when array is already sorted. Before merging, just check if $\text{arr}[\text{mid}] > \text{arr}[\text{mid}+1]$, because we are dealing with sorted subarrays. This will lead us to the recursive relation $T(n) = 2*T(n/2) + 1$ which can be resolved by the [master's theorem](#), so $T(n) = n$.

Examples:

```
Input : 1 2 3 4
Subarrays with size of 1: |1| |2| |3| |4|
Subarrays with size of 2: |1 2| |3 4|
Output : 1 2 3 4
```

```
Input : 1 2 3 4 5 6 7 8
        Subarrays with size of 1: |1| |2| |3| |4| |5| |6| |7| |8|
        Subarrays with size of 2: |1 2| |3 4| |5 6| |7 8|
        Subarrays with size of 4: |1 2 3 4| |5 6 7 8|
Output : 1 2 3 4 5 6 7 8
```

```
// C program to implement merge sort that works
```

```
// in  $O(n)$  time in best case.
#include <stdio.h>
#include <stdlib.h>

void merge(int* arr, int low, int mid, int high);

void mergesort(int* arr, int low, int high)
{
    if (low < high) {
        int mid = (low + high) / 2;
        mergesort(arr, low, mid);
        mergesort(arr, mid + 1, high);

        // This is where we optimize for best
        // case.
        if (arr[mid] > arr[mid + 1])
            merge(arr, low, mid, high);
    }
}

void merge(int* arr, int low, int mid, int high)
{
    int i = low, j = mid + 1, k = 0;
    int* temp = (int*)calloc(high - low + 1, sizeof(int));
    while ((i <= mid) && (j <= high))
        if (arr[i] < arr[j])
            temp[k++] = arr[i++];
        else
            temp[k++] = arr[j++];
    while (j <= high) // if( i>mid )
        temp[k++] = arr[j++];
    while (i <= mid) // j>high
        temp[k++] = arr[i++];

    // copy temp[] to arr[]
    for (i = low, k = 0; i <= high; i++, k++)
        arr[i] = temp[k];
    free(temp);
}

int main()
{
    int a[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
    mergesort(a, 0, 7);
    for (int i = 0; i < 8; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

1 2 3 4 5 6 7 8

Source

<https://www.geeksforgeeks.org/make-mergesort-perform-comparisons-best-case/>

Chapter 132

How to sort a big array with many repetitions?

How to sort a big array with many repetitions? - GeeksforGeeks

Consider a big array where elements are from a small set and in any range, i.e. there are many repetitions. How to efficiently sort the array?

Example:

Input: arr[] = {100, 12, 100, 1, 1, 12, 100, 1, 12, 100, 1, 1}
Output: arr[] = {1, 1, 1, 1, 1, 12, 12, 12, 100, 100, 100}

We strongly recommend you to minimize your browser and try this yourself first.

A **Basic Sorting** algorithm like [MergeSort](#), [HeapSort](#) would take $O(n \log n)$ time where n is number of elements, can we do better?

A **Better Solution** is to use Self-Balancing Binary Search Tree like [AVL](#) or [Red-Black](#) to sort in $O(n \log m)$ time where m is number of distinct elements. The idea is to extend tree node to have count of keys also.

```
struct Node
{
    int key;
    struct Node *left, *right;
    int count; // Added to handle duplicates

    // Other tree node info for balancing like height in AVL
}
```

Below is complete algorithm using AVL tree.

- 1) Create an empty AVL Tree with count as an additional field.

- 2) Traverse input array and do following for every element ‘arr[i]’
 -a) If arr[i] is not present in tree, then insert it and initialize count as 1
 -b) Else increment its count in tree.
- 3) Do Inorder Traversal of tree. While doing inorder put every key its count times in arr[].

The 2nd step takes O(n Log m) time and 3rd step takes O(n) time. So overall time complexity is O(n Log m)

Below is C++ implementation of above idea.

```
// C++ program to sort an array using AVL tree
#include<iostream>
using namespace std;

// An AVL tree Node
struct Node
{
    int key;
    struct Node *left, *right;
    int height, count;
};

// Function to insert a key in AVL Tree, if key is already present,
// then it increments count in key's node.
struct Node* insert(struct Node* Node, int key);

// This function puts inorder traversal of AVL Tree in arr[]
void inorder(int arr[], struct Node *root, int *index_ptr);

// An AVL tree based sorting function for sorting an array with
// duplicates
void sort(int arr[], int n)
{
    // Create an empty AVL Tree
    struct Node *root = NULL;

    // Insert all nodes one by one in AVL tree. The insert function
    // increments count if key is already present
    for (int i=0; i<n; i++)
        root = insert(root, arr[i]);

    // Do inorder traversal to put elements back in sorted order
    int index = 0;
    inorder(arr, root, &index);
}

// This function puts inorder traversal of AVL Tree in arr[]
void inorder(int arr[], struct Node *root, int *index_ptr)
{
```

```
if (root != NULL)
{
    // Recur for left child
    inorder(arr, root->left, index_ptr);

    // Put all occurrences of root's key in arr[]
    for (int i=0; i<root->count; i++)
    {
        arr[*index_ptr] = root->key;
        (*index_ptr)++;
    }

    // Recur for right child
    inorder(arr, root->right, index_ptr);
}

// A utility function to get height of the tree
int height(struct Node *N)
{
    if (N == NULL)
        return 0;
    return N->height;
}

// Helper function that allocates a new Node
struct Node* newNode(int key)
{
    struct Node* node = new Node;
    node->key    = key;
    node->left   = node->right = NULL;
    node->height = node->count = 1;
    return(node);
}

// A utility function to right rotate subtree rooted
// with y.
struct Node *rightRotate(struct Node *y)
{
    struct Node *x = y->left;
    struct Node *T2 = x->right;

    // Perform rotation
    x->right = y;
    y->left = T2;

    // Update heights
    y->height = max(height(y->left), height(y->right))+1;
}
```

```
x->height = max(height(x->left), height(x->right))+1;

// Return new root
return x;
}

// A utility function to left rotate subtree rooted with x
struct Node *leftRotate(struct Node *x)
{
    struct Node *y = x->right;
    struct Node *T2 = y->left;

    // Perform rotation
    y->left = x;
    x->right = T2;

    // Update heights
    x->height = max(height(x->left), height(x->right))+1;
    y->height = max(height(y->left), height(y->right))+1;

    // Return new root
    return y;
}

// Get Balance factor of Node N
int getBalance(struct Node *N)
{
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}

// Function to insert a key in AVL Tree, if key is already
// present, then it increments count in key's node.
struct Node* insert(struct Node* Node, int key)
{
    /* 1. Perform the normal BST rotation */
    if (Node == NULL)
        return (newNode(key));

    // If key already exists in BST, increment count and return
    if (key == Node->key)
    {
        (Node->count)++;
        return Node;
    }

    /* Otherwise, recur down the tree */
}
```

```
if (key < Node->key)
    Node->left = insert(Node->left, key);
else
    Node->right = insert(Node->right, key);

/* 2. Update height of this ancestor Node */
Node->height = max(height(Node->left), height(Node->right)) + 1;

/* 3. Get the balance factor of this ancestor Node to
   check whether this Node became unbalanced */
int balance = getBalance(Node);

// If this Node becomes unbalanced, then there are 4 cases

// Left Left Case
if (balance > 1 && key < Node->left->key)
    return rightRotate(Node);

// Right Right Case
if (balance < -1 && key > Node->right->key)
    return leftRotate(Node);

// Left Right Case
if (balance > 1 && key > Node->left->key)
{
    Node->left = leftRotate(Node->left);
    return rightRotate(Node);
}

// Right Left Case
if (balance < -1 && key < Node->right->key)
{
    Node->right = rightRotate(Node->right);
    return leftRotate(Node);
}

/* return the (unchanged) Node pointer */
return Node;
}

// A utility function to print an array
void printArr(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << ", ";
    cout << endl;
}
```

```
/* Driver program to test above function*/
int main()
{
    int arr[] = {100, 12, 100, 1, 1, 12, 100, 1, 12, 100, 1, 1};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Input array is\n";
    printArr(arr, n);

    sort(arr, n);

    cout << "Sorted array is\n";
    printArr(arr, n);
}
```

Output:

```
Input array is
100, 12, 100, 1, 1, 12, 100, 1, 12, 100, 1, 1,
Sorted array is
1, 1, 1, 1, 1, 12, 12, 12, 100, 100, 100,
```

We can also use [Binary Heap](#) to solve in $O(n \log m)$ time.

We can also use [Hashing](#) to solve above problem in $O(n + m \log m)$ time.

- 1) Create an empty hash table. Input array values are stores as key and their counts are stored as value in hash table.
- 2) For every element 'x' of arr[], do following
 -a) If x is present in hash table, increment its value
 -b) Else insert x with value equals to 1.
- 3) Consider all keys of hash table and sort them.
- 4) Traverse all sorted keys and print every key its value times.

Time complexity of 2nd step is $O(n)$ under the assumption that hash search and insert take $O(1)$ time. Step 3 takes $O(m \log m)$ time where m is total number of distinct keys in input array. Step 4 takes $O(n)$ time. So overall time complexity is $O(n + m \log m)$.

Program implementation using Hash Table

```
// A C++ program to sort a big array with many repetitions

#include <iostream>
#include <algorithm>
#include <map>
using namespace std;

void sort(int arr[], int n)
{
    //1. Create an empty hash table.
```

```
map<int, int> count;

//2. Input array values are stores as key and their
//counts are stored as value in hash table.
for (int i=0; i<n; i++)
    count[arr[i]]++;

map<int, int>::iterator it;
int index = 0;

//3. Consider all keys of hash table and sort them.
//In std::map, keys are already sorted.

//4. Traverse all sorted keys and print every key its value times.
for (it=count.begin(); it!=count.end(); ++it)
{
    while(it->second--)
        arr[index++]=it->first;
}

// Utility function to print an array
void printArray(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver program to test above function.
int main()
{
    int arr[] = {100, 12, 100, 1, 1, 12, 100, 1, 12, 100, 1, 1};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Input array is\n";
    printArray(arr, n);

    sort(arr, n);

    cout << "Sorted array is\n";
    printArray(arr, n);

    return 0;
}
// Contributed by Aditya Goel
```

Output:

```
Input array is  
100 12 100 1 1 12 100 1 12 100 1 1  
Sorted array is  
1 1 1 1 1 12 12 12 100 100 100
```

This article is contributed by Ankur. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

<https://www.geeksforgeeks.org/how-to-sort-a-big-array-with-many-repetitions/>

Chapter 133

How to sort an array of dates in C/C++?

How to sort an array of dates in C/C++? - GeeksforGeeks

Given an array of dates, how to sort them.

Example:

Input:

```
Date arr[] = {{20, 1, 2014},  
              {25, 3, 2010},  
              {3, 12, 1676},  
              {18, 11, 1982},  
              {19, 4, 2015},  
              {9, 7, 2015}}}
```

Output:

```
Date arr[] = {{3, 12, 1676},  
              {18, 11, 1982},  
              {25, 3, 2010},  
              {20, 1, 2014},  
              {19, 4, 2015},  
              {9, 7, 2015}}}
```

We strongly recommend you to minimize your browser and try this yourself first

The idea is to use in-built function to [sort function in C++](#). We can write our own compare function that first compares years, then months, then days.

Below is complete C++ program.

```
// C++ program to sort an array of dates
#include<bits/stdc++.h>
using namespace std;

// Structure for date
struct Date
{
    int day, month, year;
};

// This is the compare function used by in-built sort
// function to sort the array of dates.
// It takes two Dates as parameters (const is
// given to tell the compiler that the value won't be
// changed during the compare - this is for optimisation..)

// Returns true if dates have to be swapped and returns
// false if not. Since we want ascending order, we return
// true if first Date is less than second date
bool compare(const Date &d1, const Date &d2)
{
    // All cases when true should be returned
    if (d1.year < d2.year)
        return true;
    if (d1.year == d2.year && d1.month < d2.month)
        return true;
    if (d1.year == d2.year && d1.month == d2.month &&
        d1.day < d2.day)
        return true;

    // If none of the above cases satisfy, return false
    return false;
}

// Function to sort array arr[0..n-1] of dates
void sortDates(Date arr[], int n)
{
    // Calling in-built sort function.
    // First parameter array beginning,
    // Second parameter - array ending,
    // Third is the custom compare function
    sort(arr, arr+n, compare);
}

// Driver Program
int main()
{
    Date arr[] = {{20, 1, 2014},
```

```
{25, 3, 2010},  
{ 3, 12, 1676},  
{18, 11, 1982},  
{19, 4, 2015},  
{ 9, 7, 2015}};  
int n = sizeof(arr)/sizeof(arr[0]);  
  
sortDates(arr, n);  
  
cout << "Sorted dates are\n";  
for (int i=0; i<n; i++)  
{  
    cout << arr[i].day << " " << arr[i].month  
        << " " << arr[i].year;  
    cout << endl;  
}  
}
```

Output:

```
Sorted dates are  
3 12 1676  
18 11 1982  
25 3 2010  
20 1 2014  
19 4 2015  
9 7 2015
```

Similarly in C, we can use [qsort\(\)](#) function.

Related Problem:

[How to efficiently sort a big list dates in 20's](#)

This article is contributed by [Dinesh T.P.D.](#). Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/how-to-sort-an-array-of-dates-in-c/>

Chapter 134

In-Place Algorithm

In-Place Algorithm - GeeksforGeeks

In-place has more than one definitions. One **strict definition** is,

An in-place algorithm is an algorithm that does not need an extra space and produces an output in the same memory that contains the data by transforming the input ‘in-place’. However, a small constant extra space used for variables is allowed.

A more **broad definition** is,

In-place means that the algorithm does not use extra space for manipulating the input but may require a small though nonconstant extra space for its operation. Usually, this space is $O(\log n)$, though sometimes anything in $o(n)$ (Smaller than linear) is allowed [Source : [Wikipedia](#)]

A **Not In-Place** Implementation of reversing an array

C++

```
// An in-place C++ program to reverse an array
#include <bits/stdc++.h>
using namespace std;

/* Function to reverse arr[] from start to end*/
void revereseArray(int arr[], int n)
{
    // Create a copy array and store reversed
    // elements
    int rev[n];
    for (int i=0; i<n; i++)
        rev[n-i-1] = arr[i];
```

```
// Now copy reversed elements back to arr[]
for (int i=0; i<n; i++)
    arr[i] = rev[i];
}

/* Utility function to print an array */
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

/* Driver function to test above functions */
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);
    printArray(arr, n);
    revereseArray(arr, n);
    cout << "Reversed array is" << endl;
    printArray(arr, n);
    return 0;
}
```

Java

```
// An in-place Java program
// to reverse an array
import java.util.*;

class GFG
{
    /* Function to reverse arr[]
       from start to end*/
    public static void revereseArray(int []arr,
                                    int n)
    {
        // Create a copy array
        // and store reversed
        // elements
        int []rev = new int[n];
        for (int i = 0; i < n; i++)
            rev[n - i - 1] = arr[i];

        // Now copy reversed
        // elements back to arr[]
```

```
        for (int i = 0; i < n; i++)
            arr[i] = rev[i];
    }

/* Utility function to
   print an array */
public static void printArray(int []arr,
                               int size)
{
    for (int i = 0; i < size; i++)
        System.out.print(arr[i] + " ");
    System.out.println("");
}

// Driver code
public static void main(String[] args)
{
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = arr.length;
    printArray(arr, n);
    revereseArray(arr, n);
    System.out.println("Reversed array is");
    printArray(arr, n);
}
}

// This code is contributed
// by Harshit Saini
```

Python3

```
# An in-place Python program
# to reverse an array

''' Function to reverse arr[]
   from start to end '''
def revereseArray(arr, n):

    # Create a copy array
    # and store reversed
    # elements
    rev = n * [0]
    for i in range(0, n):
        rev[n - i - 1] = arr[i]

    # Now copy reversed
    # elements back to arr[]
    for i in range(0, n):
```

```

arr[i] = rev[i]

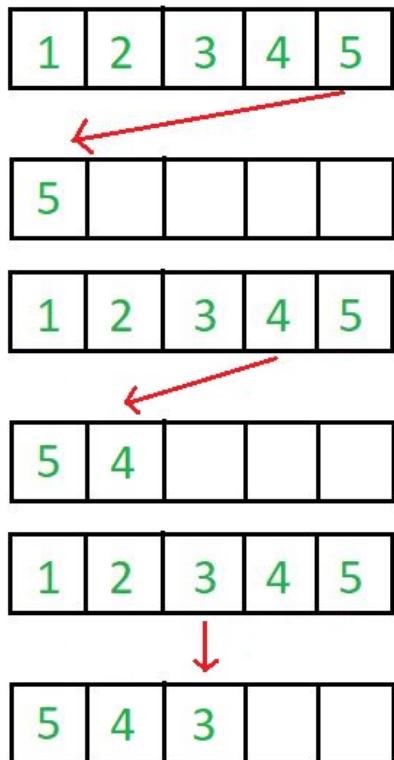
# Driver code
if __name__ == "__main__":
    arr = [1, 2, 3, 4, 5, 6]
    n = len(arr)
    print(*arr)
    revereseArray(arr, n);
    print("Reversed array is")
    print(*arr)

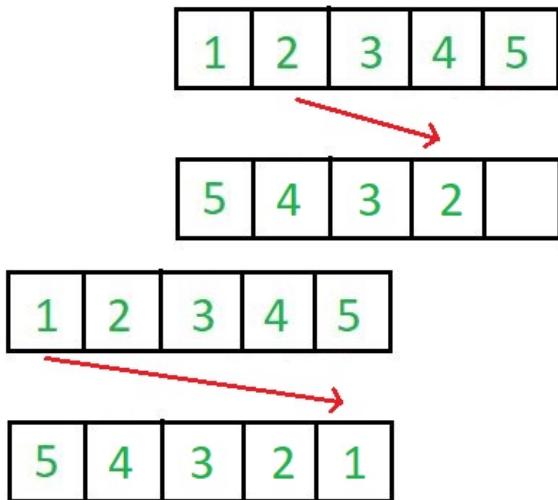
# This code is contributed
# by Harshit Saini

```

Output:

1 2 3 4 5 6
 Reversed array is
 6 5 4 3 2 1





This needs $O(n)$ extra space and is an example of not-in-place algorithm.

An **In-Place** Implementation of Reversing an array.
C++

```
// An in-place C++ program to reverse an array
#include <bits/stdc++.h>
using namespace std;

/* Function to reverse arr[] from start to end*/
void revereseArray(int arr[], int n)
{
    for (int i=0; i<n/2; i++)
        swap(arr[i], arr[n-i-1]);
}

/* Utility function to print an array */
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

/* Driver function to test above functions */
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);
    printArray(arr, n);
    revereseArray(arr, n);
    cout << "Reversed array is" << endl;
}
```

```
    printArray(arr, n);
    return 0;
}

Java

// An in-place Java program
// to reverse an array
import java.util.*;

class GFG
{
    public static int __(int x, int y) {return x;}

    /* Function to reverse arr[]
       from start to end*/
    public static void revereseArray(int []arr,
                                    int n)
    {
        for (int i = 0; i < n / 2; i++)
            arr[i] = __(arr[n - i - 1],
                        arr[n - i - 1] = arr[i]);
    }

    /* Utility function to
       print an array */
    public static void printArray(int []arr,
                                int size)
    {
        for (int i = 0; i < size; i++)
            System.out.print(Integer.toString(arr[i]) + " ");
        System.out.println("");
    }

    // Driver code
    public static void main(String[] args)
    {
        int []arr = new int[]{1, 2, 3, 4, 5, 6};
        int n = arr.length;
        printArray(arr, n);
        revereseArray(arr, n);
        System.out.println("Reversed array is");
        printArray(arr, n);
    }
}

// This code is contributed
// by Harshit Saini
```

Python3

```

# An in-place Python program
# to reverse an array

''' Function to reverse arr[]
   from start to end'''
def revereseArray(arr, n):

    for i in range(0, int(n / 2)):
        arr[i], arr[n - i - 1] = arr[n - i - 1], arr[i]

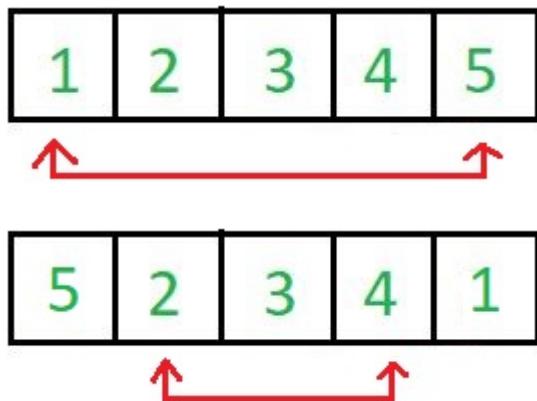
# Driver code
if __name__ == "__main__":
    arr = [1, 2, 3, 4, 5, 6]
    n = len(arr)
    print(*arr)
    revereseArray(arr, n)
    print("Reversed array is")
    print(*arr)

# This code is contributed
# by Harshit Saini

```

Output:

1 2 3 4 5 6
 Reversed array is
 6 5 4 3 2 1





This needs $O(1)$ extra space for exchanging elements and is an example of in-place algorithm.

Which Sorting Algorithms are In-Place and which are not?

In Place : [Bubble sort](#), [Selection Sort](#), [Insertion Sort](#), [Heapsort](#).

Not In-Place : [Merge Sort](#). Note that merge sort requires $O(n)$ extra space.

What about QuickSort? Why is it called In-Place?

QuickSort uses extra space for recursive function calls. It is called in-place according to broad definition as extra space required is not used to manipulate input, but only for recursive calls.

Improved By : [Harshit Saini](#)

Source

<https://www.geeksforgeeks.org/in-place-algorithm/>

Chapter 135

Insertion Sort

Insertion Sort - GeeksforGeeks

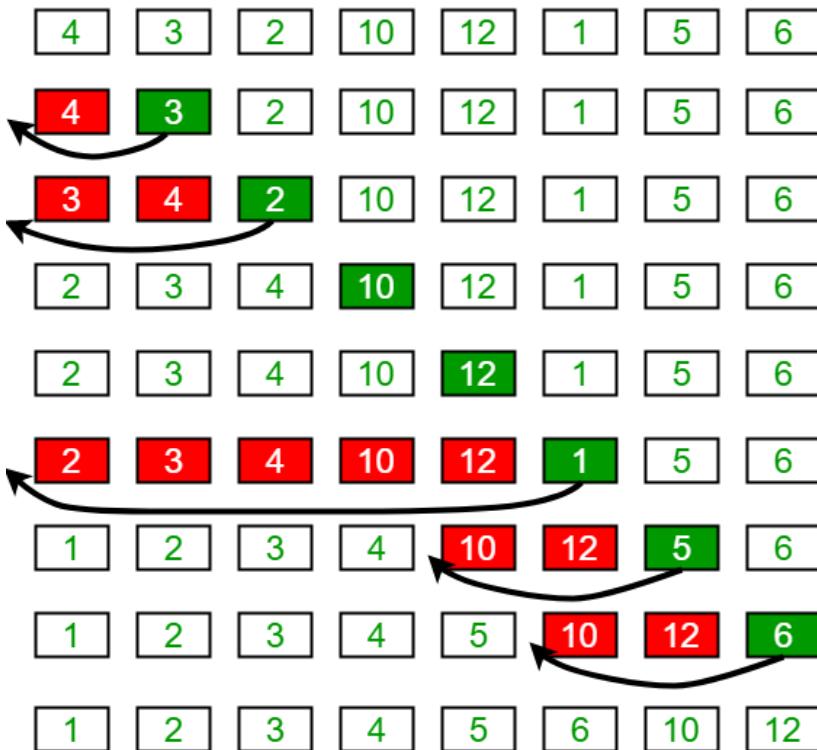
Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Algorithm

```
// Sort an arr[] of size n  
insertionSort(arr, n)  
Loop from i = 1 to n-1.  
.....a) Pick element arr[i] and insert it into sorted sequence arr[0...i-1]
```

Example:

Insertion Sort Execution Example



Another Example:

12, 11, 13, 5, 6

Let us loop for $i = 1$ (second element of the array) to 5 (Size of input array)

$i = 1$. Since 11 is smaller than 12, move 12 and insert 11 before 12

11, 12, 13, 5, 6

$i = 2$. 13 will remain at its position as all elements in $A[0..I-1]$ are smaller than 13

11, 12, 13, 5, 6

$i = 3$. 5 will move to the beginning and all other elements from 11 to 13 will move one position ahead of their current position.

5, 11, 12, 13, 6

$i = 4$. 6 will move to position after 5, and elements from 11 to 13 will move one position ahead of their current position.

5, 6, 11, 12, 13

C/C++

```
// C program for insertion sort
#include <stdio.h>
#include <math.h>
```

```
/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i-1;

        /* Move elements of arr[0..i-1], that are
           greater than key, to one position ahead
           of their current position */
        while (j >= 0 && arr[j] > key)
        {
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j+1] = key;
    }
}

// A utility function to print an array of size n
void printArray(int arr[], int n)
{
    int i;
    for (i=0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

/* Driver program to test insertion sort */
int main()
{
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);

    insertionSort(arr, n);
    printArray(arr, n);

    return 0;
}
```

Java

```
// Java program for implementation of Insertion Sort
class InsertionSort
```

```
{  
    /*Function to sort array using insertion sort*/  
    void sort(int arr[])  
    {  
        int n = arr.length;  
        for (int i=1; i<n; ++i)  
        {  
            int key = arr[i];  
            int j = i-1;  
  
            /* Move elements of arr[0..i-1], that are  
               greater than key, to one position ahead  
               of their current position */  
            while (j>=0 && arr[j] > key)  
            {  
                arr[j+1] = arr[j];  
                j = j-1;  
            }  
            arr[j+1] = key;  
        }  
    }  
  
    /* A utility function to print array of size n*/  
    static void printArray(int arr[])  
    {  
        int n = arr.length;  
        for (int i=0; i<n; ++i)  
            System.out.print(arr[i] + " ");  
  
        System.out.println();  
    }  
  
    // Driver method  
    public static void main(String args[])  
    {  
        int arr[] = {12, 11, 13, 5, 6};  
  
        InsertionSort ob = new InsertionSort();  
        ob.sort(arr);  
  
        printArray(arr);  
    }  
} /* This code is contributed by Rajat Mishra. */
```

Python

```
# Python program for implementation of Insertion Sort
```

```
# Function to do insertion sort
def insertionSort(arr):

    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):

        key = arr[i]

        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j = i-1
        while j >=0 and key < arr[j] :
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key

# Driver code to test above
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
for i in range(len(arr)):
    print ("%d" %arr[i])

# This code is contributed by Mohit Kumra
```

C#

```
// C# program for implementation of Insertion Sort
using System;

class InsertionSort
{

    // Function to sort array
    // using insertion sort/
    void sort(int[] arr)
    {
        int n = arr.Length;
        for (int i = 1; i < n; ++i)
        {
            int key = arr[i];
            int j = i - 1;

            // Move elements of arr[0..i-1],
            // that are greater than key,
            // to one position ahead of
            // their current position
```

```
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }

// A utility function to print
// array of size n
static void printArray(int[] arr)
{
    int n = arr.Length;
    for (int i = 0; i < n; ++i)
        Console.Write(arr[i] + " ");

    Console.WriteLine("\n");
}

// Driver Code
public static void Main()
{
    int[] arr = {12, 11, 13, 5, 6};
    InsertionSort ob = new InsertionSort();
    ob.sort(arr);
    printArray(arr);
}

// This code is contributed by ChitraNayal.
```

PHP

```
<?php
// PHP program for insertion sort

// Function to sort an array
// using insertion sort
function insertionSort(&$arr, $n)
{
    for ($i = 1; $i < $n; $i++)
    {
        $key = $arr[$i];
        $j = $i-1;

        // Move elements of arr[0..i-1],
        // that are greater than key, to
```

```
// one position ahead of their
// current position
while ($j >= 0 && $arr[$j] > $key)
{
    $arr[$j + 1] = $arr[$j];
    $j = $j - 1;
}

$arr[$j + 1] = $key;
}

// A utility function to
// print an array of size n
function printArray(&$arr, $n)
{
    for ($i = 0; $i < $n; $i++)
        echo $arr[$i]." ";
    echo "\n";
}

// Driver Code
$arr = array(12, 11, 13, 5, 6);
$n = sizeof($arr);
insertionSort($arr, $n);
printArray($arr, $n);

// This code is contributed by ChitraNayal.
?>
```

Output:

5 6 11 12 13

Time Complexity: $O(n^2)$

Auxiliary Space: $O(1)$

Boundary Cases: Insertion sort takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.

Algorithmic Paradigm: Incremental Approach

Sorting In Place: Yes

Stable: Yes

Online: Yes

Uses: Insertion sort is used when number of elements is small. It can also be useful when input array is almost sorted, only few elements are misplaced in complete big array.

What is Binary Insertion Sort?

We can use binary search to reduce the number of comparisons in normal insertion sort. Binary Insertion Sort find use binary search to find the proper location to insert the selected item at each iteration. In normal insertion, sort it takes $O(i)$ (at i th iteration) in worst case. we can reduce it to $O(\log i)$ by using binary search. The algorithm as a whole still has a running worst case running time of $O(n^2)$ because of the series of swaps required for each insertion. Refer [this](#) for implementation.

How to implement Insertion Sort for Linked List?

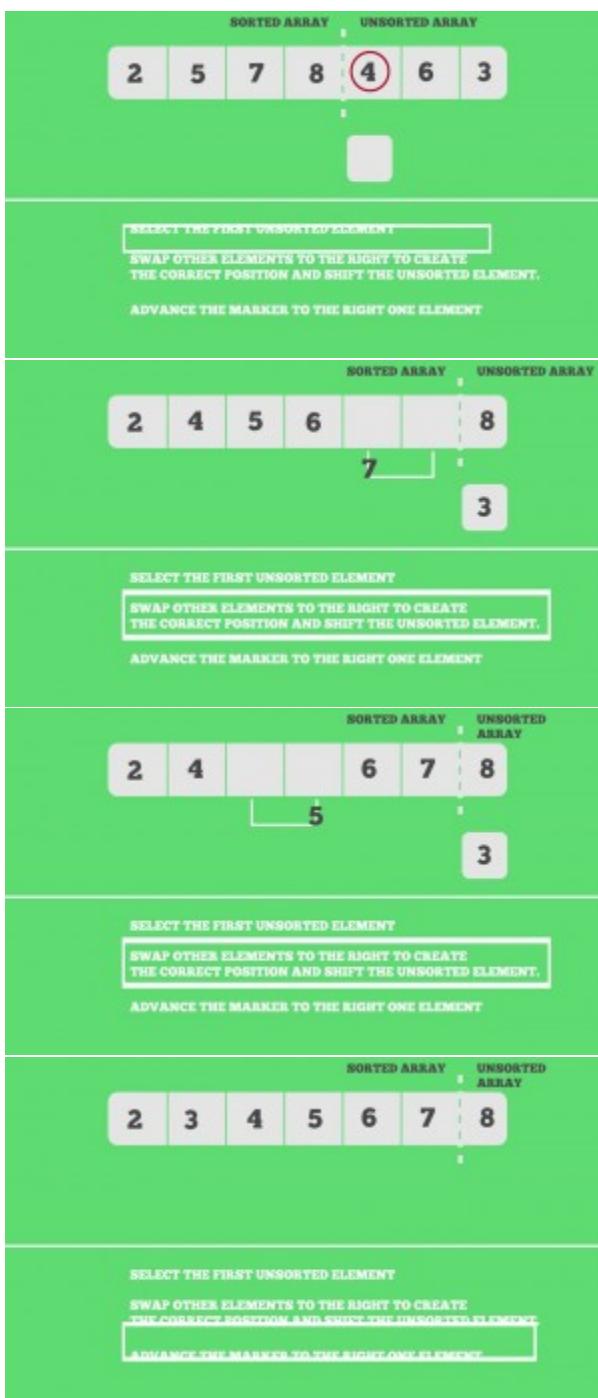
Below is simple insertion sort algorithm for linked list.

- 1) Create an empty sorted (or result) list
- 2) Traverse the given list, do following for every node.
 -a) Insert current node in sorted way in sorted or result list.
- 3) Change head of given linked list to head of sorted (or result) list.

Refer [this](#) for implementation.

Snapshots:





Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz
[Selection Sort](#), [Bubble Sort](#), [Insertion Sort](#), [Merge Sort](#), [Heap Sort](#), [QuickSort](#), [Radix Sort](#),
[Counting Sort](#), [Bucket Sort](#), [ShellSort](#), [Comb Sort](#),

Improved By : [ChitraNayal](#), [PIYUSHKUMAR19](#)

Source

<https://www.geeksforgeeks.org/insertion-sort/>

Chapter 136

Insertion Sort by Swapping Elements

Insertion Sort by Swapping Elements - GeeksforGeeks

Insertion Sort is suitable for arrays of small size. It also achieves best-case complexity of $O(n)$ if the arrays are already sorted. We have discussed at both [Iterative Insertion Sort](#) and [Recursive Insertion Sort](#). In this article slightly different implementations for both iterative and recursive versions are discussed.

Iterative Insertion Sort

Let us look at the algorithm for the iterative insertion sort

```
function insertionSort(V)
    i, j, k
    for i from 1..length(V)
        k = V[i]
        j = i-1
        while j > 0 and k < V[j]
            V[j+1] = V[j]
            j -= 1
        V[j] = k
    return V
```

Inside the while loop, we shift all values larger than k by one position and then insert k into the first position where k is larger than the array value. The same effect is obtained if we swap consecutive array elements. By repeated swapping k will travel to its correct position.

Let's take an example to illustrate this

Insert 3 in A = {1, 2, 4, 5, 6}

```
Put 3 at the end of list.  
A = {1, 2, 4, 5, 6, 3}  
3 < 6, swap 3 and 6  
A = {1, 2, 4, 5, 3, 6}  
3 < 5 swap 3 and 5  
A = {1, 2, 4, 3, 5, 6}  
3 < 4 swap 3 and 4  
A = {1, 2, 3, 4, 5, 6}  
3 > 2 so stop
```

By repeatedly swapping 3 travels to its proper position in the list

Therefore the above algorithm can be modified as

```
function insertionSort(V)  
    for i in 1...length(V)  
        j = i  
        while ( j > 0 and V[j] < V[j-1])  
            Swap V[j] and V[j-1]  
            j -= 1  
    return V
```

The CPP code for this algorithm is given below

C++

```
// Iterative CPP program to sort  
// an array by swapping elements  
#include <iostream>  
#include <vector>  
using namespace std;  
using Vector = vector<int>;  
  
// Utility function to print a Vector  
void printVector(const Vector& V)  
{  
    for (auto e : V) {  
        cout << e << " ";  
    }  
    cout << endl;  
}  
  
// Function performs insertion sort on  
// vector V  
void insertionSort(Vector& V)  
{  
    int N = V.size();
```

```
int i, j, key;

for (i = 1; i < N; i++) {
    j = i;

    // Insert V[i] into list 0..i-1
    while (j > 0 and V[j] < V[j - 1]) {

        // Swap V[j] and V[j-1]
        swap(V[j], V[j - 1]);

        // Decrement j by 1
        j -= 1;
    }
}

// Driver Code
int main()
{
    Vector A = { 9, 8, 7, 5, 2, 1, 2, 3 };

    cout << "Array: " << endl;
    printVector(A);

    cout << "After Sorting :" << endl;
    insertionSort(A);
    printVector(A);

    return 0;
}
```

Java

```
// Iterative Java program to sort
// an array by swapping elements
import java.io.*;
import java.util.*;

class GFG
{
    // Utility function to print a Vector
    static void printVector( Vector<Integer> V)
    {
        for (int i = 0; i < V.size(); i++) {
            System.out.print(V.get(i)+" ");
        }
    }
}
```

```
        System.out.println();
    }

    // Function performs insertion sort on
    // vector V
    static void insertionSort(Vector<Integer> V)
    {
        int N = V.size();
        int i, j, key;

        for (i = 1; i < N; i++) {
            j = i;

            // Insert V[i] into list 0..i-1
            while (j > 0 && V.get(j) < V.get(j - 1)) {

                // Swap V[j] and V[j-1]
                int temp= V.get(j);
                V.set(j, V.get(j - 1));
                V.set(j - 1, temp);

                // Decrement j by 1
                j -= 1;
            }
        }
    }

    public static void main (String[] args)
    {
        Vector<Integer> A = new Vector<Integer> ();
        A.add(0, 9);
        A.add(1, 8);
        A.add(2, 7);
        A.add(3, 5);
        A.add(4, 2);
        A.add(5, 1);
        A.add(6, 2);
        A.add(7, 3);
        System.out.print("Array: ");
        printVector(A);
        System.out.print("After Sorting :");
        insertionSort(A);
        printVector(A);
    }
}

//This code is contributed by Gitanjali.
```

Python3

```
# Iterative python program to sort
# an array by swapping elements
import math

# Utility function to print a Vector
def printVector( V):

    for i in V:
        print(i ,end= " ")
    print (" ")

def insertionSort( V):

    N = len(V)

    for i in range(1,N):
        j = i

        # Insert V[i] into list 0..i-1
        while (j > 0 and V[j] < V[j - 1]) :

            # Swap V[j] and V[j-1]
            temp = V[j];
            V[j] = V[j - 1];
            V[j-1] = temp;

            # Decrement j
            j -= 1


    # Driver method
    A = [ 9, 8, 7, 5, 2, 1, 2, 3 ]
    n = len(A)
    print("Array")
    printVector(A)
    print( "After Sorting :")
    insertionSort(A)
    printVector(A)

# This code is contributed by Gitanjali.
```

Output:

Array:

```
9 8 7 5 2 1 2 3
After Sorting :
1 2 2 3 5 7 8 9
```

Recursive Insertion Sort

Consider an Array A of size N

- First recursively sort the sublist of A which is of size N-1
- Insert the last element of A into the sorted sublist.

To perform the insertion step use repeated swapping as discussed above.

Algorithm

```
function insertionSortRecursive(A, N)
    if N >= 1
        insertionSortRecursive(A, N-1)
        j = N-1
        while j > 0 and A[j] < A[j-1]
            Swap A[j] and A[j-1]
            j = j-1
        [end of while]
    [end of if]
```

The CPP implementation is given below

CPP

```
// Recursive CPP program to sort an array
// by swapping elements
#include <iostream>
#include <vector>

using namespace std;
using Vector = vector<int>

// Utility function to print a Vector
void printVector(const Vector& V)
{
    for (auto e : V) {
        cout << e << " ";
    }
    cout << endl;
}

// Function to perform Insertion Sort recursively
```

```
void insertionSortRecursive(Vector& V, int N)
{
    if (N <= 1)
        return;

    // General Case
    // Sort V till second last element and
    // then insert last element into V
    insertionSortRecursive(V, N - 1);

    // Insertion step
    int j = N - 1;
    while (j > 0 and V[j] < V[j - 1]) {

        // Swap V[j] and V[j-1]
        swap(V[j], V[j - 1]);

        // Decrement j
        j -= 1;
    }
}

// Driver Code
int main()
{

    // Declare a vector of size 10
    Vector A = { 9, 8, 7, 5, 2, 1, 2, 3 };

    cout << "Array: " << endl;
    printVector(A);

    cout << "After Sorting :" << endl;
    insertionSortRecursive(A, A.size());
    printVector(A);
    return 0;
}
```

Java

```
// Recursive Java program to sort
// an array by swapping elements
import java.io.*;
import java.util.*;

class GFG
{
    // Utility function to print a Vector
```

```
static void printVector( Vector<Integer> V)
{
    for (int i = 0; i < V.size(); i++) {
        System.out.print(V.get(i) + " ");
    }
    System.out.println();
}

// Function performs insertion sort on
// vector V
static void insertionSortRecursive(Vector<Integer> V,int N)
{
    if (N <= 1)
        return;

    // General Case
    // Sort V till second last element and
    // then insert last element into V
    insertionSortRecursive(V, N - 1);

    // Insertion step
    int j = N - 1;

    // Insert V[i] into list 0..i-1
    while (j > 0 && V.get(j) < V.get(j - 1))
    {

        // Swap V[j] and V[j-1]
        int temp= V.get(j);
        V.set(j, V.get(j - 1));
        V.set(j - 1, temp);

        // Decrement j by 1
        j -= 1;
    }

}

// Driver code
public static void main (String[] args)
{
    Vector<Integer> A = new Vector<Integer> ();
    A.add(0, 9);
    A.add(1, 8);
    A.add(2, 7);
    A.add(3, 5);
```

```
A.add(4, 2);
A.add(5, 1);
A.add(6, 2);
A.add(7, 3);
System.out.print("Array: ");
printVector(A);
System.out.print("After Sorting :");
insertionSortRecursive(A,A.size());
printVector(A);
}
}

// This code is contributed by Gitanjali.
```

Python3

```
# Recursive python program
# to sort an array
# by swapping elements
import math

# Utility function to print
# a Vector
def printVector( V):

    for i in V:
        print(i, end = " ")
    print (" ")

# Function to perform Insertion
# Sort recursively
def insertionSortRecursive(V, N):

    if (N <= 1):
        return 0

    # General Case
    # Sort V till second
    # last element and
    # then insert last element
    # into V
    insertionSortRecursive(V, N - 1)

    # Insertion step
    j = N - 1
    while (j > 0 and V[j] < V[j - 1]) :

        # Swap V[j] and V[j-1]
```

```
temp = V[j];
V[j] = V[j - 1];
V[j-1] = temp;

# Decrement j
j -= 1

# Driver method
A = [ 9, 8, 7, 5, 2, 1, 2, 3 ]
n=len(A)
print("Array")
printVector(A)
print( "After Sorting :")

insertionSortRecursive(A,n)
printVector(A)

# This code is contributed
# by Gitanjali.
```

Output

```
Array:
9 8 7 5 2 1 2 3
After Sorting :
1 2 2 3 5 7 8 9
```

Note

The Time Complexity of the algorithm is still $O(N^2)$ in the worst case. Moreover, these versions are potentially slower since repeated swapping requires more operations. However, these versions are discussed because of their implementation simplicity and ease of understanding.

References

[Stack Overflow – Insertion Sort by Swapping](#)

Source

<https://www.geeksforgeeks.org/insertion-sort-swapping-elements/>

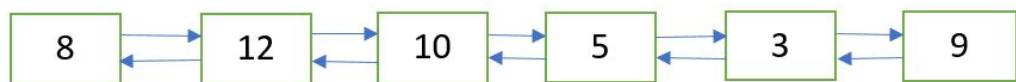
Chapter 137

Insertion Sort for Doubly Linked List

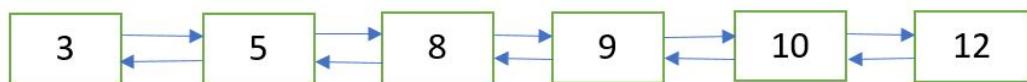
Insertion Sort for Doubly Linked List - GeeksforGeeks

Sort the doubly linked list using insertion sort technique.

Initial doubly linked list



Doubly Linked List after applying insertion sort



Algorithm:

Below is a simple insertion sort algorithm for doubly linked list.

- 1) Create an empty **sorted (or result)** doubly linked list.
- 2) Traverse the given doubly linked list, do following for every node.
 -a) Insert current node in sorted way in **sorted(or result)** doubly linked list.
 - 3) Change head of given linked list to head of **sorted (or result)** list.

The main step is (2.a) which has been covered in below post.

[Sorted Insert for Doubly Linked List](#)

```
// C++ implementation for insertion Sort
// on a doubly linked list
#include <bits/stdc++.h>
```

```
using namespace std;

// Node of a doubly linked list
struct Node {
    int data;
    struct Node* prev, *next;
};

// function to create and return a new node
// of a doubly linked list
struct Node* getNode(int data)
{
    // allocate node
    struct Node* newNode =
        (struct Node*)malloc(sizeof(struct Node));

    // put in the data
    newNode->data = data;
    newNode->prev = newNode->next = NULL;
    return newNode;
}

// function to insert a new node in sorted way in
// a sorted doubly linked list
void sortedInsert(struct Node** head_ref, struct Node* newNode)
{
    struct Node* current;

    // if list is empty
    if (*head_ref == NULL)
        *head_ref = newNode;

    // if the node is to be inserted at the beginning
    // of the doubly linked list
    else if ((*head_ref)->data >= newNode->data) {
        newNode->next = *head_ref;
        newNode->next->prev = newNode;
        *head_ref = newNode;
    }

    else {
        current = *head_ref;

        // locate the node after which the new node
        // is to be inserted
        while (current->next != NULL &&
               current->next->data < newNode->data)
            current = current->next;
    }
}
```

```
/*Make the appropriate links */

newNode->next = current->next;

// if the new node is not inserted
// at the end of the list
if (current->next != NULL)
    newNode->next->prev = newNode;

current->next = newNode;
newNode->prev = current;
}

}

// function to sort a doubly linked list using insertion sort
void insertionSort(struct Node** head_ref)
{
    // Initialize 'sorted' - a sorted doubly linked list
    struct Node* sorted = NULL;

    // Traverse the given doubly linked list and
    // insert every node to 'sorted'
    struct Node* current = *head_ref;
    while (current != NULL) {

        // Store next for next iteration
        struct Node* next = current->next;

        // removing all the links so as to create 'current'
        // as a new node for insertion
        current->prev = current->next = NULL;

        // insert current in 'sorted' doubly linked list
        sortedInsert(&sorted, current);

        // Update current
        current = next;
    }

    // Update head_ref to point to sorted doubly linked list
    *head_ref = sorted;
}

// function to print the doubly linked list
void printList(struct Node* head)
{
    while (head != NULL) {
```

```
        cout << head->data << " ";
        head = head->next;
    }
}

// function to insert a node at the beginning of
// the doubly linked list
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*)malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* Make next of new node as head and previous as NULL */
    new_node->next = (*head_ref);
    new_node->prev = NULL;

    /* change prev of head node to new node */
    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

// Driver program to test above
int main()
{
    /* start with the empty doubly linked list */
    struct Node* head = NULL;

    // insert the following data
    push(&head, 9);
    push(&head, 3);
    push(&head, 5);
    push(&head, 10);
    push(&head, 12);
    push(&head, 8);

    cout << "Doubly Linked List Before Sortingn";
    printList(head);

    insertionSort(&head);

    cout << "nDoubly Linked List After Sortingn";
}
```

```
    printList(head);  
  
    return 0;  
}
```

Output:

```
Doubly Linked List Before Sorting  
8 12 10 5 3 9  
Doubly Linked List After Sorting  
3 5 8 9 10 12
```

Time Complexity: $O(n^2)$

Source

<https://www.geeksforgeeks.org/insertion-sort-doubly-linked-list/>

Chapter 138

Insertion sort to sort even and odd positioned elements in different orders

Insertion sort to sort even and odd positioned elements in different orders - GeeksforGeeks

We are given an array. We need to sort the even positioned elements in the ascending order and the odd positioned elements in the descending order. We must apply [insertion sort](#) to sort them.

Examples:

```
Input : a[] = {7, 10, 11, 3, 6, 9, 2, 13, 0}
Output :      11 3 7 9 6 10 2 13 0
Even positioned elements after sorting int
ascending order : 3 9 10 13
Odd positioned elements after sorting int
descending order : 11 7 6 2 0
```

We separately apply the insertion sort technique on the even positioned elements and the odd positioned elements separately but within the same array. The loop starts for the odd positioned from the 0th index(1st element) and for the even from the 1st index(2nd element) and keep on increasing by 2 since every alternate is odd/even positioned.

We now simply apply the insertion sort procedure on the odd positioned and even positioned. the odd positioned elements are A[0, 2, 4, ...] and even are A[1, 3, 5, 7...]. So they are considered as separate sub-arrays but within the same array.

Explanation for the odd positioned:

The 0th element already sorted. Now the 2nd element compared with the 0th and inserted and so on the (i+2)th element compared with the previous ones until the end of the array.

The same approach is applied for the even positioned elements in the array.(This is same as the insertion sort technique).

C++

```
// C++ program to sort even positioned elements
// in ascending order and odd positioned
// elements in descending order.
#include<stdio.h>
#include<stdlib.h>

// Function to calculate the given problem.
void evenOddInsertionSort(int arr[], int n)
{
    for (int i = 2; i < n; i++)
    {
        int j = i-2;
        int temp = arr[i];

        /* checking whether the position is even
           or odd. And after checking applying the
           insertion sort to the given
           positioned elements.*/

        // checking for odd positioned.
        if ((i+1) & 1 == 1)
        {
            // Inserting even positioned elements
            // in ascending order.
            while (temp >= arr[j] && j >= 0)
            {
                arr[j+2] = arr[j];
                j -= 2;
            }
            arr[j+2] = temp;
        }

        // sorting the even positioned.
        else {

            // Inserting odd positioned elements
            // in descending order.
            while (temp <= arr[j] && j >= 0)
            {
                arr[j+2] = arr[j];
                j -= 2;
            }
            arr[j+2] = temp;
        }
    }
}
```

```
    }
}

// A utility function to print an array of size n
void printArray(int arr[], int n)
{
    for (int i=0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

/* Driver program to test insertion sort */
int main()
{
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);

    evenOddInsertionSort(arr, n);
    printArray(arr, n);

    return 0;
}
```

Python3

```
# Python3 program to sort even
# positioned elements in ascending
# order and odd positionedelements
# in descending order.

# Function to calculate
# the given problem.
def evenOddInsertionSort(arr, n):

    for i in range(2, n):

        j = i - 2
        temp = arr[i]

        # checking whether the position
        # is even or odd. And after
        # checking applying the insertion
        # sort to the given
        # positioned elements.

        # checking for odd positioned.
        if ((i + 1) & 1 == 1) :
```

```
# Inserting even positioned elements
# in ascending order.
while (temp >= arr[j] and j >= 0):

    arr[j + 2] = arr[j]
    j -= 2

arr[j + 2] = temp

# sorting the even positioned.
else :

    # Inserting odd positioned elements
    # in descending order.
    while (temp <= arr[j] and j >= 0) :

        arr[j + 2] = arr[j]
        j -= 2

arr[j + 2] = temp

# A utility function to print an array of size n
def printArray(arr, n):

    for i in range(0, n):
        print(arr[i], end=" ")

# Driver program
arr = [12, 11, 13, 5, 6]
n = len(arr)
evenOddInsertionSort(arr, n)
printArray(arr, n)

# This code is contributed by
# Smitha Dinesh Semwal
```

Output:13 5 12 11 6

There exist better approaches to solve this problem without insertion sort. Please refer [Sort even-placed elements in increasing and odd-placed in decreasing order](#)for details.

Source

<https://www.geeksforgeeks.org/insertion-sortheven-odd-positioned-elements/>

Chapter 139

Insertion sort using C++ STL

Insertion sort using C++ STL - GeeksforGeeks

Implementation of [Insertion Sort](#) using STL functions.

Pre-requisites : [Insertion Sort](#), [std::rotate](#), [std::upper_bound](#), [C++ Iterators](#).

The idea is to use [std::upper_bound](#) to find an element making the array unsorted. Then we can rotate the unsorted part so that it ends up sorted. We can traverse the array doing these operations and the result will be a sorted array.

The code is offline on purpose to show hard-coded and easy to understand code.

```
// C++ program to implement insertion sort using STL.
#include <bits/stdc++.h>

// Function to sort the array
void insertionSort(std::vector<int> &vec)
{
    for (auto it = vec.begin(); it != vec.end(); it++)
    {
        // Searching the upper bound, i.e., first
        // element greater than *it from beginning
        auto const insertion_point =
            std::upper_bound(vec.begin(), it, *it);

        // Shifting the unsorted part
        std::rotate(insertion_point, it, it+1);
    }
}

// Function to print the array
void print(std::vector<int> vec)
{
    for( int x : vec)
```

```
        std::cout << x << " ";
        std::cout << '\n';
    }

// Driver code
int main()
{
    std::vector<int> arr = {2, 1, 5, 3, 7, 5, 4, 6};
    insertionSort(arr);
    print(arr);
}
```

Output:

```
1 2 3 4 5 5 6 7
```

Source

<https://www.geeksforgeeks.org/insertion-sort-using-c-stl/>

Chapter 140

Intersection of two Sorted Linked Lists

Intersection of two Sorted Linked Lists - GeeksforGeeks

Given two lists sorted in increasing order, create and return a new list representing the intersection of the two lists. The new list should be made with its own memory — the original lists should not be changed.

For example, let the first linked list be 1->2->3->4->6 and second linked list be 2->4->6->8, then your function should create and return a third list as 2->4->6.

Method 1 (Using Dummy Node)

The strategy here uses a temporary dummy node as the start of the result list. The pointer tail always points to the last node in the result list, so appending new nodes is easy. The dummy node gives tail something to point to initially when the result list is empty. This dummy node is efficient, since it is only temporary, and it is allocated in the stack. The loop proceeds, removing one node from either ‘a’ or ‘b’, and adding it to tail. When we are done, the result is in dummy.next.

```
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

void push(struct Node** head_ref, int new_data);

/*This solution uses the temporary dummy to build up the result list */
struct Node* sortedIntersect(struct Node* a, struct Node* b)
```

```
{  
    struct Node dummy;  
    struct Node* tail = &dummy;  
    dummy.next = NULL;  
  
    /* Once one or the other list runs out -- we're done */  
    while (a != NULL && b != NULL)  
    {  
        if (a->data == b->data)  
        {  
            push((&tail->next), a->data);  
            tail = tail->next;  
            a = a->next;  
            b = b->next;  
        }  
        else if (a->data < b->data) /* advance the smaller list */  
            a = a->next;  
        else  
            b = b->next;  
    }  
    return(dummy.next);  
}  
  
/* UTILITY FUNCTIONS */  
/* Function to insert a node at the beginning of the linked list */  
void push(struct Node** head_ref, int new_data)  
{  
    /* allocate node */  
    struct Node* new_node =  
        (struct Node*) malloc(sizeof(struct Node));  
  
    /* put in the data */  
    new_node->data = new_data;  
  
    /* link the old list off the new node */  
    new_node->next = (*head_ref);  
  
    /* move the head to point to the new node */  
    (*head_ref) = new_node;  
}  
  
/* Function to print nodes in a given linked list */  
void printList(struct Node *node)  
{  
    while (node != NULL)  
    {  
        printf("%d ", node->data);  
        node = node->next;  
    }
```

```
    }

}

/* Drier program to test above functions*/
int main()
{
    /* Start with the empty lists */
    struct Node* a = NULL;
    struct Node* b = NULL;
    struct Node *intersect = NULL;

    /* Let us create the first sorted linked list to test the functions
     Created linked list will be 1->2->3->4->5->6 */
    push(&a, 6);
    push(&a, 5);
    push(&a, 4);
    push(&a, 3);
    push(&a, 2);
    push(&a, 1);

    /* Let us create the second sorted linked list
     Created linked list will be 2->4->6->8 */
    push(&b, 8);
    push(&b, 6);
    push(&b, 4);
    push(&b, 2);

    /* Find the intersection two linked lists */
    intersect = sortedIntersect(a, b);

    printf("\n Linked list containing common items of a & b \n ");
    printList(intersect);

    getchar();
}
```

Time Complexity: $O(m+n)$ where m and n are number of nodes in first and second linked lists respectively.

Method 2 (Using Local References)

This solution is structurally very similar to the above, but it avoids using a dummy node. Instead, it maintains a `struct node**` pointer, `lastPtrRef`, that always points to the last pointer of the result list. This solves the same case that the dummy node did — dealing with the result list when it is empty. If you are trying to build up a list at its tail, either the dummy node or the `struct node**` “reference” strategy can be used.

```
#include<stdio.h>
#include<stdlib.h>
```

```
/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

void push(struct Node** head_ref, int new_data);

/* This solution uses the local reference */
struct Node* sortedIntersect(struct Node* a, struct Node* b)
{
    struct Node* result = NULL;
    struct Node** lastPtrRef = &result;

    /* Advance comparing the first nodes in both lists.
       When one or the other list runs out, we're done. */
    while (a!=NULL && b!=NULL)
    {
        if (a->data == b->data)
        {
            /* found a node for the intersection */
            push(lastPtrRef, a->data);
            lastPtrRef = &((*lastPtrRef)->next);
            a = a->next;
            b = b->next;
        }
        else if (a->data < b->data)
            a=a->next;           /* advance the smaller list */
        else
            b=b->next;
    }
    return(result);
}

/* UTILITY FUNCTIONS */
/* Function to insert a node at the beginning of the linked list */
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
}
```

```
new_node->next = (*head_ref);

/* move the head to point to the new node */
(*head_ref)      = new_node;
}

/* Function to print nodes in a given linked list */
void printList(struct Node *node)
{
    while(node != NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

/* Driver program to test above functions*/
int main()
{
    /* Start with the empty lists */
    struct Node* a = NULL;
    struct Node* b = NULL;
    struct Node *intersect = NULL;

    /* Let us create the first sorted linked list to test the functions
     * Created linked list will be 1->2->3->4->5->6 */
    push(&a, 6);
    push(&a, 5);
    push(&a, 4);
    push(&a, 3);
    push(&a, 2);
    push(&a, 1);

    /* Let us create the second sorted linked list
     * Created linked list will be 2->4->6->8 */
    push(&b, 8);
    push(&b, 6);
    push(&b, 4);
    push(&b, 2);

    /* Find the intersection two linked lists */
    intersect = sortedIntersect(a, b);

    printf("\n Linked list containing common items of a & b \n ");
    printList(intersect);

    getchar();
}
```

Time Complexity: $O(m+n)$ where m and n are number of nodes in first and second linked lists respectively.

Method 3 (Recursive)

Below is the recursive implementation of sortedIntersect().

```
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

struct Node *sortedIntersect(struct Node *a, struct Node *b)
{
    /* base case */
    if (a == NULL || b == NULL)
        return NULL;

    /* If both lists are non-empty */

    /* advance the smaller list and call recursively */
    if (a->data < b->data)
        return sortedIntersect(a->next, b);

    if (a->data > b->data)
        return sortedIntersect(a, b->next);

    // Below lines are executed only when a->data == b->data
    struct Node *temp = (struct Node *)malloc(sizeof(struct Node));
    temp->data = a->data;

    /* advance both lists and call recursively */
    temp->next = sortedIntersect(a->next, b->next);
    return temp;
}

/* UTILITY FUNCTIONS */
/* Function to insert a node at the beginning of the linked list */
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;
```

```
/* link the old list off the new node */
new_node->next = (*head_ref);

/* move the head to point to the new node */
(*head_ref)      = new_node;
}

/* Function to print nodes in a given linked list */
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

/* Drier program to test above functions*/
int main()
{
    /* Start with the empty lists */
    struct Node* a = NULL;
    struct Node* b = NULL;
    struct Node *intersect = NULL;

    /* Let us create the first sorted linked list to test the functions
     * Created linked list will be 1->2->3->4->5->6 */
    push(&a, 6);
    push(&a, 5);
    push(&a, 4);
    push(&a, 3);
    push(&a, 2);
    push(&a, 1);

    /* Let us create the second sorted linked list
     * Created linked list will be 2->4->6->8 */
    push(&b, 8);
    push(&b, 6);
    push(&b, 4);
    push(&b, 2);

    /* Find the intersection two linked lists */
    intersect = sortedIntersect(a, b);

    printf("\n Linked list containing common items of a & b \n ");
    printList(intersect);
```

```
    return 0;  
}
```

Time Complexity: $O(m+n)$ where m and n are number of nodes in first and second linked lists respectively.

References:

cslibrary.stanford.edu/105/LinkedListProblems.pdf

Source

<https://www.geeksforgeeks.org/intersection-of-two-sorted-linked-lists/>

Chapter 141

Iterative HeapSort

Iterative HeapSort - GeeksforGeeks

HeapSort is a comparison based sorting technique where we first build Max Heap and then swaps the root element with last element (size times) and maintains the heap property each time to finally make it sorted.

Examples:

```
Input : 10 20 15 17 9 21
Output : 9 10 15 17 20 21
```

```
Input: 12 11 13 5 6 7 15 5 19
Output: 5 5 6 7 11 12 13 15 19
```

In first Example, first we have to build Max Heap.

So, we will start from 20 as child and check for its parent. Here 10 is smaller, so we will swap these two.

Now, 20 10 15 17 9 21

Now, child 17 is greater than its parent 10. So, both will be swapped and order will be
20 17 15 10 9 21

Now, child 21 is greater than parent 15. So, both will be swapped.

20 17 21 10 9 15

Now, again 21 is bigger than parent 20. So,

21 17 20 10 9 15

This is Max Heap.

Now, we have to apply sorting. Here, we have to swap first element with last one and we have to maintain Max Heap property.

So, after first swapping : 15 17 20 10 9 21

It clearly violates Max Heap property. So, we have to maintain it. So, order will be
20 17 15 10 9 21

17 10 15 9 20 21

15 10 9 17 20 21

10 9 15 17 20 21

9 10 15 17 20 21

Here, underlined part is sorted part.

```

// C++ program for implementation
// of Iterative Heap Sort
#include <bits/stdc++.h>
using namespace std;

// function build Max Heap where value
// of each child is always smaller
// than value of their parent
void buildMaxHeap(int arr[], int n)
{
    for (int i = 1; i < n; i++)
    {
        // if child is bigger than parent
        if (arr[i] > arr[(i - 1) / 2])
        {
            int j = i;

            // swap child and parent until
            // parent is smaller
            while (arr[j] > arr[(j - 1) / 2])
            {
                swap(arr[j], arr[(j - 1) / 2]);
                j = (j - 1) / 2;
            }
        }
    }
}

void heapSort(int arr[], int n)
{
    buildMaxHeap(arr, n);

    for (int i = n - 1; i > 0; i--)
    {
        // swap value of first indexed
        // with last indexed
        swap(arr[0], arr[i]);

        // maintaining heap property
        // after each swapping
        int j = 0, index;

        do

```

```

{
    index = (2 * j + 1);

    // if left child is smaller than
    // right child point index variable
    // to right child
    if (arr[index] < arr[index + 1] &&
        index < (i - 1))
        index++;

    // if parent is smaller than child
    // then swapping parent with child
    // having higher value
    if (arr[j] < arr[index] && index < i)
        swap(arr[j], arr[index]);

    j = index;

} while (index < i);
}
}

// Driver Code to test above
int main()
{
    int arr[] = {10, 20, 15, 17, 9, 21};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Given array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    printf("\n\n");

    heapSort(arr, n);

    // print array after sorting
    printf("Sorted array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}

```

Output :

Given array: 10 20 15 17 9 21

Sorted array: 9 10 15 17 20 21

Here, both function buildMaxHeap and heapSort runs in $O(n\log n)$ time. So, overall time complexity is $O(n\log n)$

Source

<https://www.geeksforgeeks.org/iterative-heap-sort/>

Chapter 142

Iterative Merge Sort

Iterative Merge Sort - GeeksforGeeks

Following is a typical recursive implementation of [Merge Sort](#)

C/C++

```
/* Recursive C program for merge sort */
#include<stdlib.h>
#include<stdio.h>

/* Function to merge the two halves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r);

/* l is for left index and r is right index of the sub-array
   of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l+(r-l)/2; //Same as (l+r)/2 but avoids overflow for large l & h
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}

/* Function to merge the two halves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
```

```
/* create temp arrays */
int L[n1], R[n2];

/* Copy data to temp arrays L[] and R[] */
for (i = 0; i < n1; i++)
    L[i] = arr[l + i];
for (j = 0; j < n2; j++)
    R[j] = arr[m + 1+ j];

/* Merge the temp arrays back into arr[l..r]*/
i = 0;
j = 0;
k = l;
while (i < n1 && j < n2)
{
    if (L[i] <= R[j])
    {
        arr[k] = L[i];
        i++;
    }
    else
    {
        arr[k] = R[j];
        j++;
    }
    k++;
}

/* Copy the remaining elements of L[], if there are any */
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy the remaining elements of R[], if there are any */
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}

/* Function to print an array */
void printArray(int A[], int size)
{
```

```
int i;
for (i=0; i < size; i++)
    printf("%d ", A[i]);
printf("\n");
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

Java

```
// Recursive Java Program for merge sort

import java.util.Arrays;
public class GFG
{
    public static void mergeSort(int[] array)
    {
        if(array == null)
        {
            return;
        }

        if(array.length > 1)
        {
            int mid = array.length / 2;

            // Split left part
            int[] left = new int[mid];
            for(int i = 0; i < mid; i++)
            {
                left[i] = array[i];
            }

            // Split right part
```

```
int[] right = new int[array.length - mid];
for(int i = mid; i < array.length; i++)
{
    right[i - mid] = array[i];
}
mergeSort(left);
mergeSort(right);

int i = 0;
int j = 0;
int k = 0;

// Merge left and right arrays
while(i < left.length && j < right.length)
{
    if(left[i] < right[j])
    {
        array[k] = left[i];
        i++;
    }
    else
    {
        array[k] = right[j];
        j++;
    }
    k++;
}
// Collect remaining elements
while(i < left.length)
{
    array[k] = left[i];
    i++;
    k++;
}
while(j < right.length)
{
    array[k] = right[j];
    j++;
    k++;
}
}

// Driver program to test above functions.
public static void main(String[] args)
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int i=0;
```

```
System.out.println("Given array is");

for(i=0; i<arr.length; i++)
    System.out.print(arr[i]+" ");

mergeSort(arr);

System.out.println("\n");
System.out.println("Sorted array is");

for(i=0; i<arr.length; i++)
    System.out.print(arr[i]+" ");
}

}

// Code Contributed by Mohit Gupta_OMG
```

Python

```
# Recursive Python Program for merge sort

def merge(left, right):
    if not len(left) or not len(right):
        return left or right

    result = []
    i, j = 0, 0
    while (len(result) < len(left) + len(right)):
        if left[i] < right[j]:
            result.append(left[i])
            i+= 1
        else:
            result.append(right[j])
            j+= 1
        if i == len(left) or j == len(right):
            result.extend(left[i:] or right[j:])
            break

    return result

def mergesort(list):
    if len(list) < 2:
        return list

    middle = len(list)/2
    left = mergesort(list[:middle])
    right = mergesort(list[middle:])
```

```
return merge(left, right)

seq = [12, 11, 13, 5, 6, 7]
print("Given array is")
print(seq);
print("\n")
print("Sorted array is")
print(mergesort(seq))

# Code Contributed by Mohit Gupta_OMG
```

C#

```
/* Iterative C# program for merge
sort */
using System;

class GFG {

    /* l is for left index and r
       is right index of the sub-array
       of arr to be sorted */
    static void mergeSort(int[] arr,
                          int l, int r)
    {
        if (l < r)
        {

            // Same as (l+r)/2 but avoids
            // overflow for large l & h
            int m = l + (r - l) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m+1, r);
            merge(arr, l, m, r);
        }
    }

    /* Function to merge the two halves
    arr[l..m] and arr[m+1..r] of array
    arr[] */
    static void merge(int[] arr, int l,
                      int m, int r)
    {
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;

        /* create temp arrays */

```

```
int []L = new int[n1];
int []R = new int[n2];

/* Copy data to temp arrays
L[] and R[] */
for (i = 0; i < n1; i++)
    L[i] = arr[l + i];
for (j = 0; j < n2; j++)
    R[j] = arr[m + 1+ j];

/* Merge the temp arrays back
into arr[l..r]*/
i = 0;
j = 0;
k = l;
while (i < n1 && j < n2)
{
    if (L[i] <= R[j])
    {
        arr[k] = L[i];
        i++;
    }
    else
    {
        arr[k] = R[j];
        j++;
    }
    k++;
}

/* Copy the remaining elements of
L[], if there are any */
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy the remaining elements of
R[], if there are any */
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}
}
```

```

/* Function to print an array */
static void printArray(int []A, int size)
{
    int i;
    for (i=0; i < size; i++)
        Console.Write(A[i]+" ");
    Console.Write("\n");
}

/* Driver program to test above functions */
public static void Main()
{
    int []arr = {12, 11, 13, 5, 6, 7};
    int arr_size = arr.Length;

    Console.WriteLine("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    Console.WriteLine("\nSorted array is \n");
    printArray(arr, arr_size);
}
}

// This code is contributed by Smitha

```

Output:

```
Given array is
12 11 13 5 6 7
```

```
Sorted array is
5 6 7 11 12 13
```

Iterative Merge Sort:

The above function is recursive, so uses [function call stack](#) to store intermediate values of l and h. The function call stack stores other bookkeeping information together with parameters. Also, function calls involve overheads like storing activation record of the caller function and then resuming execution. Unlike [Iterative QuickSort](#), the iterative MergeSort doesn't require explicit auxiliary stack.

The above function can be easily converted to iterative version. Following is iterative Merge Sort.

C

```
/* Iterative C program for merge sort */
```

```

#include<stdlib.h>
#include<stdio.h>

/* Function to merge the two halves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r);

// Utility function to find minimum of two integers
int min(int x, int y) { return (x < y)? x : y; }

/* Iterative mergesort function to sort arr[0...n-1] */
void mergeSort(int arr[], int n)
{
    int curr_size; // For current size of subarrays to be merged
                   // curr_size varies from 1 to n/2
    int left_start; // For picking starting index of left subarray
                    // to be merged

    // Merge subarrays in bottom up manner. First merge subarrays of
    // size 1 to create sorted subarrays of size 2, then merge subarrays
    // of size 2 to create sorted subarrays of size 4, and so on.
    for (curr_size=1; curr_size<=n-1; curr_size = 2*curr_size)
    {
        // Pick starting point of different subarrays of current size
        for (left_start=0; left_start<n-1; left_start += 2*curr_size)
        {
            // Find ending point of left subarray. mid+1 is starting
            // point of right
            int mid = left_start + curr_size - 1;

            int right_end = min(left_start + 2*curr_size - 1, n-1);

            // Merge Subarrays arr[left_start...mid] & arr[mid+1...right_end]
            merge(arr, left_start, mid, right_end);
        }
    }
}

/* Function to merge the two halves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

```

```
/* Copy data to temp arrays L[] and R[] */
for (i = 0; i < n1; i++)
    L[i] = arr[1 + i];
for (j = 0; j < n2; j++)
    R[j] = arr[m + 1+ j];

/* Merge the temp arrays back into arr[l..r]*/
i = 0;
j = 0;
k = l;
while (i < n1 && j < n2)
{
    if (L[i] <= R[j])
    {
        arr[k] = L[i];
        i++;
    }
    else
    {
        arr[k] = R[j];
        j++;
    }
    k++;
}

/* Copy the remaining elements of L[], if there are any */
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy the remaining elements of R[], if there are any */
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}

/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
```

```
    printf("\n");
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, n);

    mergeSort(arr, n);

    printf("\nSorted array is \n");
    printArray(arr, n);
    return 0;
}
```

Java

```
/* Iterative Java program for merge sort */
import java.lang.Math.*;

class GFG {

    /* Iterative mergesort function to sort
     * arr[0...n-1] */
    static void mergeSort(int arr[], int n)
    {

        // For current size of subarrays to
        // be merged curr_size varies from
        // 1 to n/2
        int curr_size;

        // For picking starting index of
        // left subarray to be merged
        int left_start;

        // Merge subarrays in bottom up
        // manner. First merge subarrays
        // of size 1 to create sorted
        // subarrays of size 2, then merge
        // subarrays of size 2 to create
        // sorted subarrays of size 4, and
        // so on.
    }
}
```

```

for (curr_size = 1; curr_size <= n-1;
     curr_size = 2*curr_size)
{
    // Pick starting point of different
    // subarrays of current size
    for (left_start = 0; left_start < n-1;
         left_start += 2*curr_size)
    {
        // Find ending point of left
        // subarray. mid+1 is starting
        // point of right
        int mid = left_start + curr_size - 1;

        int right_end = Math.min(left_start
                               + 2*curr_size - 1, n-1);

        // Merge Subarrays arr[left_start...mid]
        // & arr[mid+1...right_end]
        merge(arr, left_start, mid, right_end);
    }
}

/* Function to merge the two halves arr[l..m] and
arr[m+1..r] of array arr[] */
static void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[] = new int[n1];
    int R[] = new int[n2];

    /* Copy data to temp arrays L[]
and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into
arr[l..r]*/
    i = 0;
    j = 0;
    k = l;
}

```

```
while (i < n1 && j < n2)
{
    if (L[i] <= R[j])
    {
        arr[k] = L[i];
        i++;
    }
    else
    {
        arr[k] = R[j];
        j++;
    }
    k++;
}

/* Copy the remaining elements of
L[], if there are any */
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy the remaining elements of
R[], if there are any */
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}

/* Function to print an array */
static void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        System.out.printf("%d ", A[i]);
    System.out.printf("\n");
}

/* Driver program to test above functions */
public static void main(String[] args)
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = arr.length;
```

```
System.out.printf("Given array is \n");
printArray(arr, n);

mergeSort(arr, n);

System.out.printf("\nSorted array is \n");
printArray(arr, n);
}

}

// This code is contributed by Smitha
```

Python 3

```
# Iterative Merge sort (Bottom Up)

# Iterative mergesort function to
# sort arr[0...n-1]
def mergeSort(a):

    current_size = 1

    # Outer loop for traversing Each
    # sub array of current_size
    while current_size < len(a) - 1:

        left = 0
        # Inner loop for merge call
        # in a sub array
        # Each complete Iteration sorts
        # the iterating sub array
        while left < len(a)-1:

            # mid index = left index of
            # sub array + current sub
            # array size - 1
            mid = left + current_size - 1

            # (False result,True result)
            # [Condition] Can use current_size
            # if 2 * current_size < len(a)-1
            # else len(a)-1
            right = ((2 * current_size + left - 1,
                      len(a) - 1)[2 * current_size
                                   + left - 1 > len(a)-1])

            # Merge call for each sub array
```

```
merge(a, left, mid, right)
    left = left + current_size*2

    # Increasing sub array size by
    # multiple of 2
    current_size = 2 * current_size

# Merge Function
def merge(a, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    L = [0] * n1
    R = [0] * n2
    for i in range(0, n1):
        L[i] = a[l + i]
    for i in range(0, n2):
        R[i] = a[m + i + 1]

    i, j, k = 0, 0, l
    while i < n1 and j < n2:
        if L[i] > R[j]:
            a[k] = R[j]
            j += 1
        else:
            a[k] = L[i]
            i += 1
        k += 1

    while i < n1:
        a[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        a[k] = R[j]
        j += 1
        k += 1

# Driver code
a = [12, 11, 13, 5, 6, 7]
print("Given array is ")
print(a)

mergeSort(a)

print("Sorted array is ")
print(a)
```

```
# Contributed by Madhur Chhangani [RCOEM]
```

Output:

```
Given array is  
12 11 13 5 6 7
```

```
Sorted array is  
5 6 7 11 12 13
```

Time complexity of above iterative function is same as recursive, i.e., $\Theta(n\log n)$.

References:

<http://csg.sph.umich.edu/abecasis/class/2006/615.09.pdf>

This article is contributed by **Shivam Agrawal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [MADHURCHHANGANI](#), Smitha Dinesh Semwal, parv

Source

<https://www.geeksforgeeks.org/iterative-merge-sort/>

Chapter 143

Iterative Quick Sort

Iterative Quick Sort - GeeksforGeeks

Following is a typical recursive implementation of [Quick Sort](#) that uses last element as pivot.
C++

```
// CPP code for recursive function of Quicksort
#include<bits/stdc++.h>

using namespace std;

// Function to swap numbers
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

/* This function takes last element as pivot,
   places the pivot element at its correct
   position in sorted array, and places
   all smaller (smaller than pivot) to left
   of pivot and all greater elements to
   right of pivot */
int partition (int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);

    for (int j = l; j <= h- 1; j++)
    {
        if (arr[j] <= x)
        {
```

```
i++;
    swap (&arr[i], &arr[j]);
}
}
swap (&arr[i + 1], &arr[h]);
return (i + 1);
}

/* A[] --> Array to be sorted,
l --> Starting index,
h --> Ending index */
void quickSort(int A[], int l, int h)
{
    if (l < h)
    {
        /* Partitioning index */
        int p = partition(A, l, h);
        quickSort(A, l, p - 1);
        quickSort(A, p + 1, h);
    }
}

// Driver code
int main(){

    int n = 5;
    int arr[n] = {4, 2, 6, 9, 2};

    quickSort(arr, 0, n-1);

    for(int i = 0; i < n; i++){
        cout << arr[i] << " ";
    }

    return 0;
}
```

Java

```
// Java program for implementation of QuickSort
import java.util.*;

class QuickSort
{
    /* This function takes last element as pivot,
    places the pivot element at its correct
    position in sorted array, and places all
```

```
smaller (smaller than pivot) to left of
pivot and all greater elements to right
of pivot */
static int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low-1); // index of smaller element
    for (int j=low; j<=high-1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;
            // swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    // swap arr[i+1] and arr[high] (or pivot)
    int temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;

    return i+1;
}

/* The main function that implements QuickSort()
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
static void qSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is
        now at right place */
        int pi = partition(arr, low, high);

        // Recursively sort elements before
        // partition and after partition
        qSort(arr, low, pi-1);
        qSort(arr, pi+1, high);
    }
}
```

```
// Driver code
public static void main(String args[])
{
    int n = 5;
    int arr[] = {4, 2, 6, 9, 2};

    qSort(arr, 0, n-1);

    for(int i =0;i<n;i++){
        System.out.print(arr[i]+" ");
    }

}
```

Python3

```
# A typical recursive Python
# implementation of QuickSort

# Function takes last element as pivot,
# places the pivot element at its correct
# position in sorted array, and places all
# smaller (smaller than pivot) to left of
# pivot and all greater elements to right
# of pivot
def partition(arr, low, high):
    i = (low - 1)          # index of smaller element
    pivot = arr[high]       # pivot

    for j in range(low, high):

        # If current element is smaller
        # than or equal to pivot
        if arr[j] <= pivot:

            # increment index of
            # smaller element
            i += 1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return (i + 1)

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
```

```
# low --> Starting index,
# high --> Ending index

# Function to do Quick sort
def quickSort(arr,low,high):
    if low < high:

        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr, low, high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)

# Driver Code
if __name__ == '__main__':
    arr = [4, 2, 6, 9, 2]
    n = len(arr)

    # Calling quickSort function
    quickSort(arr, 0, n - 1)

    for i in range(n):
        print(arr[i], end = " ")
```

C#

```
// C# program for implementation of
// QuickSort
using System;

class GFG {

    /* This function takes last element
    as pivot, places the pivot element
    at its correct position in sorted
    array, and places all smaller
    (smaller than pivot) to left of
    pivot and all greater elements to
    right of pivot */
    static int partition(int []arr,
                        int low, int high)
    {
        int temp;
        int pivot = arr[high];
```

```
// index of smaller element
int i = (low-1);
for (int j = low; j <= high-1; j++)
{
    // If current element is
    // smaller than or
    // equal to pivot
    if (arr[j] <= pivot)
    {
        i++;
        // swap arr[i] and arr[j]
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

// swap arr[i+1] and arr[high]
// (or pivot)
temp = arr[i+1];
arr[i+1] = arr[high];
arr[high] = temp;

return i+1;
}

/* The main function that implements
QuickSort() arr[] --> Array to be
sorted,
low --> Starting index,
high --> Ending index */
static void qSort(int []arr, int low,
                  int high)
{
    if (low < high)
    {
        /* pi is partitioning index,
        arr[pi] is now at right place */
        int pi = partition(arr, low, high);

        // Recursively sort elements
        // before partition and after
        // partition
        qSort(arr, low, pi-1);
        qSort(arr, pi+1, high);
    }
}
```

```
        }
    }

// Driver code
public static void Main()
{
    int n = 5;
    int []arr = {4, 2, 6, 9, 2};

    qSort(arr, 0, n-1);

    for(int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
}
}

// This code is contributed by nitin mittal.
```

Output:

2 2 4 6 9

The above implementation can be optimized in many ways

- 1) The above implementation uses last index as pivot. This causes worst-case behavior on already sorted arrays, which is a commonly occurring case. The problem can be solved by choosing either a random index for the pivot, or choosing the middle index of the partition or choosing the median of the first, middle and last element of the partition for the pivot. (See [this](#)for details)
- 2) To reduce the recursion depth, recur first for the smaller half of the array, and use a tail call to recurse into the other.
- 3) Insertion sort works better for small subarrays. Insertion sort can be used for invocations on such small arrays (i.e. where the length is less than a threshold t determined experimentally). For example, [this](#) library implementation of qsort uses insertion sort below size 7.

Despite above optimizations, the function remains recursive and uses [function call stack](#) to store intermediate values of l and h. The function call stack stores other bookkeeping information together with parameters. Also, function calls involve overheads like storing activation record of the caller function and then resuming execution.

The above function can be easily converted to iterative version with the help of an auxiliary stack. Following is an iterative implementation of the above recursive code.

C/C++

```
// An iterative implementation of quick sort
```

```

#include <stdio.h>

// A utility function to swap two elements
void swap ( int* a, int* b )
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function is same in both iterative and recursive*/
int partition (int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);

    for (int j = l; j <= h- 1; j++)
    {
        if (arr[j] <= x)
        {
            i++;
            swap (&arr[i], &arr[j]);
        }
    }
    swap (&arr[i + 1], &arr[h]);
    return (i + 1);
}

/* A[] --> Array to be sorted,
   l --> Starting index,
   h --> Ending index */
void quickSortIterative (int arr[], int l, int h)
{
    // Create an auxiliary stack
    int stack[ h - l + 1 ];

    // initialize top of stack
    int top = -1;

    // push initial values of l and h to stack
    stack[ ++top ] = l;
    stack[ ++top ] = h;

    // Keep popping from stack while is not empty
    while ( top >= 0 )
    {
        // Pop h and l
        h = stack[ top-- ];

```

```
l = stack[ top-- ];

// Set pivot element at its correct position
// in sorted array
int p = partition( arr, l, h );

// If there are elements on left side of pivot,
// then push left side to stack
if ( p-1 > l )
{
    stack[ ++top ] = l;
    stack[ ++top ] = p - 1;
}

// If there are elements on right side of pivot,
// then push right side to stack
if ( p+1 < h )
{
    stack[ ++top ] = p + 1;
    stack[ ++top ] = h;
}
}

// A utility function to print contents of arr
void printArr( int arr[], int n )
{
    int i;
    for ( i = 0; i < n; ++i )
        printf( "%d ", arr[i] );
}

// Driver program to test above functions
int main()
{
    int arr[] = {4, 3, 5, 2, 1, 3, 2, 3};
    int n = sizeof( arr ) / sizeof( *arr );
    quickSortIterative( arr, 0, n - 1 );
    printArr( arr, n );
    return 0;
}
```

Java

```
// Java program for implementation of QuickSort
import java.util.*;

class QuickSort
```

```

{
    /* This function takes last element as pivot,
    places the pivot element at its correct
    position in sorted array, and places all
    smaller (smaller than pivot) to left of
    pivot and all greater elements to right
    of pivot */
    static int partition(int arr[], int low, int high)
    {
        int pivot = arr[high];

        // index of smaller element
        int i = (low-1);
        for (int j = low; j <= high-1; j++)
        {
            // If current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;

                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }

    /* A[] --> Array to be sorted,
       l --> Starting index,
       h --> Ending index */
    static void quickSortIterative (int arr[], int l, int h)
    {
        // Create an auxiliary stack
        int[] stack = new int[h-l+1];

        // initialize top of stack
        int top = -1;

        // push initial values of l and h to stack

```

```
stack[++top] = l;
stack[++top] = h;

// Keep popping from stack while is not empty
while (top >= 0)
{
    // Pop h and l
    h = stack[top--];
    l = stack[top--];

    // Set pivot element at its correct position
    // in sorted array
    int p = partition(arr, l, h);

    // If there are elements on left side of pivot,
    // then push left side to stack
    if (p-1 > l)
    {
        stack[++top] = l;
        stack[++top] = p - 1;
    }

    // If there are elements on right side of pivot,
    // then push right side to stack
    if (p+1 < h)
    {
        stack[++top] = p + 1;
        stack[++top] = h;
    }
}

// Driver code
public static void main(String args[])
{
    int arr[] = {4, 3, 5, 2, 1, 3, 2, 3};
    int n = 8;

    // Function calling
    quickSortIterative(arr, 0, n-1);

    for(int i = 0; i < n; i++){
        System.out.print(arr[i] + " ");
    }
}
```

Python

```
# Python program for implementation of Quicksort

# This function is same in both iterative and recursive
def partition(arr,l,h):
    i = ( l - 1 )
    x = arr[h]

    for j in range(l , h):
        if arr[j] <= x:

            # increment index of smaller element
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[h] = arr[h],arr[i+1]
    return (i+1)

# Function to do Quick sort
# arr[] --> Array to be sorted,
# l --> Starting index,
# h --> Ending index
def quickSortIterative(arr,l,h):

    # Create an auxiliary stack
    size = h - l + 1
    stack = [0] * (size)

    # initialize top of stack
    top = -1

    # push initial values of l and h to stack
    top = top + 1
    stack[top] = l
    top = top + 1
    stack[top] = h

    # Keep popping from stack while is not empty
    while top >= 0:

        # Pop h and l
        h = stack[top]
        top = top - 1
        l = stack[top]
        top = top - 1

        # Set pivot element at its correct position in
        # sorted array
        p = partition( arr, l, h )
```

```
# If there are elements on left side of pivot,
# then push left side to stack
if p-1 > l:
    top = top + 1
    stack[top] = l
    top = top + 1
    stack[top] = p - 1

# If there are elements on right side of pivot,
# then push right side to stack
if p+1 < h:
    top = top + 1
    stack[top] = p + 1
    top = top + 1
    stack[top] = h

# Driver code to test above
arr = [4, 3, 5, 2, 1, 3, 2, 3]
n = len(arr)
quickSortIterative(arr, 0, n-1)
print ("Sorted array is:")
for i in range(n):
    print ("%d" %arr[i]),

# This code is contributed by Mohit Kumra
```

C#

```
// C# program for implementation of QuickSort
using System;

class GFG {

    /* This function takes last element as pivot,
    places the pivot element at its correct
    position in sorted array, and places all
    smaller (smaller than pivot) to left of
    pivot and all greater elements to right
    of pivot */
    static int partition(int []arr, int low,
                        int high)
    {
        int temp;
        int pivot = arr[high];

        // index of smaller element
        int i = (low-1);

        for (int j=low; j<high; j++)
```

```

for (int j = low; j <= high-1; j++)
{
    // If current element is smaller
    // than or equal to pivot
    if (arr[j] <= pivot)
    {
        i++;

        // swap arr[i] and arr[j]
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

// swap arr[i+1] and arr[high]
// (or pivot)

temp = arr[i+1];
arr[i+1] = arr[high];
arr[high] = temp;

return i+1;
}

/* A[] --> Array to be sorted,
l --> Starting index,
h --> Ending index */
static void quickSortIterative (int []arr,
                                int l, int h)
{
    // Create an auxiliary stack
    int [] stack = new int[h-l+1];

    // initialize top of stack
    int top = -1;

    // push initial values of l and h to
    // stack
    stack[++top] = l;
    stack[++top] = h;

    // Keep popping from stack while
    // is not empty
    while (top >= 0)
    {
        // Pop h and l
        h = stack[top--];

```

```
l = stack[top--];

// Set pivot element at its
// correct position in
// sorted array
int p = partition(arr, l, h);

// If there are elements on
// left side of pivot, then
// push left side to stack
if (p-1 > l)
{
    stack[++top] = l;
    stack[++top] = p - 1;
}

// If there are elements on
// right side of pivot, then
// push right side to stack
if (p+1 < h)
{
    stack[++top] = p + 1;
    stack[++top] = h;
}
}

// Driver code
public static void Main()
{
    int []arr = {4, 3, 5, 2, 1, 3, 2, 3};
    int n = 8;

    // Function calling
    quickSortIterative(arr, 0, n-1);

    for(int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
}
}

// This code is contributed by anuj_67.
```

Output:

```
1 2 2 3 3 3 4 5
```

The above mentioned optimizations for recursive quick sort can also be applied to iterative

version.

- 1) Partition process is same in both recursive and iterative. The same techniques to choose optimal pivot can also be applied to iterative version.
- 2) To reduce the stack size, first push the indexes of smaller half.
- 3) Use insertion sort when the size reduces below a experimentally calculated threshold.

References:

<http://en.wikipedia.org/wiki/Quicksort>

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : [nitin mittal](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/iterative-quick-sort/>

Chapter 144

Java 8 Arrays parallelSort() method with Examples

Java 8 Arrays parallelSort() method with Examples - GeeksforGeeks

Java 8 introduced a new method called as **parallelSort()** in **java.util.Arrays** Class. It uses Parallel Sorting of array elements

Algorithm of parallelSort()

1. The array is divided into sub-arrays and that sub-arrays is again divided into their sub-arrays, until the minimum level of detail in a set of array.
2. Arrays are sorted individually by multiple thread.
3. The parallel sort uses Fork/Join Concept for sorting.
4. Sorted sub-arrays are then merged.

Syntax :

1. For sorting data in ascending order :

```
public static void parallelSort(Object obj[])
```

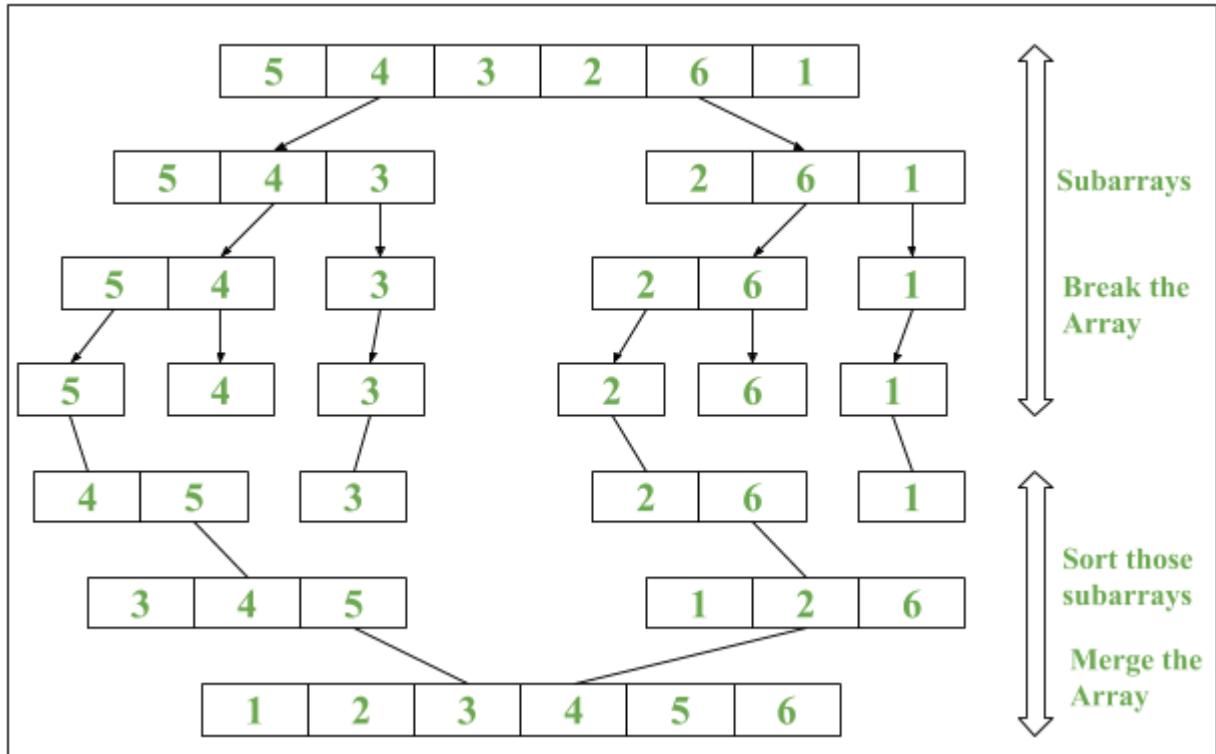
2. For sorting data in specified range in ascending order :

```
public static void parallelSort(Object obj[], int from, int to)
```

Advantage :

parallelSort() method uses concept of **MultiThreading** which makes the sorting **faster** as compared to normal sorting method.

Example



Below are the program that will illustrate the use of Arrays.parallelSort():

Program 1: To demonstrate use of Parallel Sort

```
// Java program to demonstrate
// Arrays.parallelSort() method

import java.util.Arrays;

public class ParallelSort {
    public static void main(String[] args)
    {
        // Creating an array
        int numbers[] = { 9, 8, 7, 6, 3, 1 };

        // Printing unsorted Array
        System.out.print("Unsorted Array: ");
        // Iterating the Elements using stream
        Arrays.stream(numbers)
            .forEach(n -> System.out.print(n + " "));
        System.out.println();
    }
}
```

```
// Using Arrays.parallelSort()
Arrays.parallelSort(numbers);

// Printing sorted Array
System.out.print("Sorted Array: ");
// Iterating the Elements using stream
Arrays.stream(numbers)
    .forEach(n -> System.out.print(n + " "));
}
}
```

Output:

```
Unsorted Array: 9 8 7 6 3 1
Sorted Array: 1 3 6 7 8 9
```

Time Complexity is $O(n \log n)$

Program 2: To demonstrate use of Parallel Sort w.r.t. Series Sort (Normal Sort)

```
// Java program to demonstrate impact
// of Parallel Sort vs Serial Sort

import java.util.Arrays;
import java.util.Random;

public class ParallelSort {
    public static void main(String[] args)
    {
        // Creating an array
        int numbers[] = new int[100];

        // Iterating Loop till i = 1000
        // with interval of 10
        for (int i = 0; i < 1000; i += 10) {

            System.out.println("\nFor iteration number: "
                + (i / 10 + 1));

            // Random Int Array Generation
            Random rand = new Random();

            for (int j = 0; j < 100; j++) {
                numbers[j] = rand.nextInt();
            }

            // Start and End Time of Arrays.sort()
        }
    }
}
```

```
long startTime = System.nanoTime();

// Performing Serial Sort
Arrays.sort(numbers);

long endTime = System.nanoTime();

// Printing result of Serial Sort
System.out.println("Start and End Time in Serial (in ns): "
+ startTime + ":" + endTime);
System.out.println("Time taken by Serial Sort(in ns): "
+ (endTime - startTime));

// Start and End Time of Arrays.parallelSort()
startTime = System.nanoTime();

// Performing Parallel Sort
Arrays.parallelSort(numbers);

endTime = System.nanoTime();

// Printing result of Parallel Sort
System.out.println("Start and End Time in parallel (in ns): "
+ startTime + ":" + endTime);
System.out.println("Time taken by Parallel Sort(in ns): "
+ (endTime - startTime));
System.out.println();
}

}
}
```

Output:

```
For iteration number: 1
Start and End Time in Serial (in ns): 3951000637977:3951000870361
Time taken by Serial Sort(in ns): 232384
Start and End Time in parallel (in ns): 3951000960823:3951000971044
Time taken by Parallel Sort(in ns): 10221
```

```
For iteration number: 2
Start and End Time in Serial (in ns): 3951001142284:3951001201757
Time taken by Serial Sort(in ns): 59473
Start and End Time in parallel (in ns): 3951001256643:3951001264039
Time taken by Parallel Sort(in ns): 7396
.
.
```

```
.  
For iteration number: 99  
Start and End Time in Serial (in ns): 3951050723541:3951050731520  
Time taken by Serial Sort(in ns): 7979  
Start and End Time in parallel (in ns): 3951050754238:3951050756130  
Time taken by Parallel Sort(in ns): 1892
```

```
For iteration number: 100  
Start and End Time in Serial (in ns): 3951050798392:3951050804741  
Time taken by Serial Sort(in ns): 6349  
Start and End Time in parallel (in ns): 3951050828544:3951050830582  
Time taken by Parallel Sort(in ns): 2038
```

Note : Different time intervals will be printed But parallel sort will be done before normal sort.

Environment: 2.6 GHz Intel Core i7, java version 8

Source

<https://www.geeksforgeeks.org/java-8-arrays-parallelsort-method-with-examples/>

Chapter 145

Java Program for Binary Insertion Sort

Java Program for Binary Insertion Sort - GeeksforGeeks

We can use binary search to reduce the number of comparisons in [normal insertion sort](#).
Binary Insertion Sort find use binary search to find the proper location to insert the selected item at each iteration.
In normal insertion, sort it takes $O(i)$ (at ith iteration) in worst case. we can reduce it to $O(\log i)$ by using [binary search](#).

Source

<https://www.geeksforgeeks.org/java-program-for-binary-insertion-sort/>

Java

```
// Java Program implementing
// binary insertion sort

import java.util.Arrays;
class GFG
{
    public static void main(String[] args)
    {
        final int[] arr = {37, 23, 0, 17, 12, 72, 31,
                           46, 100, 88, 54 };

        new GFG().sort(arr);

        for(int i=0; i<arr.length; i++)
            System.out.print(arr[i]+" ");
    }

    void sort(int arr[])
    {
        int n = arr.length;
        for (int i = 1; i < n; ++i)
        {
            int key = arr[i];
            int j = i - 1;

            /* Move elements of arr[0..i-1], that are
               greater than key, to one position ahead
               of their current position */
            while (j >= 0 && arr[j] > key)
                arr[j + 1] = arr[j],
                j = j - 1;
            arr[j + 1] = key;
        }
    }
}
```

```
}

public void sort(int array[])
{
    for (int i = 1; i < array.length; i++)
    {
        int x = array[i];

        // Find location to insert using binary search
        int j = Math.abs(Arrays.binarySearch(array, 0, i, x) + 1);

        //Shifting array to one location right
        System.arraycopy(array, j, array, j+1, i-j);

        //Placing element at its correct location
        array[j] = x;
    }
}

// Code contributed by Mohit Gupta_OMG
```

Please refer complete article on [Binary Insertion Sort](#) for more details!

Chapter 146

Java Program for Bubble Sort

Java Program for Bubble Sort - GeeksforGeeks

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Source

<https://www.geeksforgeeks.org/java-program-for-bubble-sort/>

Java

```
// Java program for implementation of Bubble Sort
class BubbleSort
{
    void bubbleSort(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n-i-1; j++)
                if (arr[j] > arr[j+1])
                {
                    // swap temp and arr[i]
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
    }

    /* Prints the array */
    void printArray(int arr[])
    {
```

```
int n = arr.length;
for (int i=0; i<n; ++i)
    System.out.print(arr[i] + " ");
System.out.println();
}

// Driver method to test above
public static void main(String args[])
{
    BubbleSort ob = new BubbleSort();
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    ob.bubbleSort(arr);
    System.out.println("Sorted array");
    ob.printArray(arr);
}
/* This code is contributed by Rajat Mishra */
```

Please refer complete article on [Bubble Sort](#) for more details!

Chapter 147

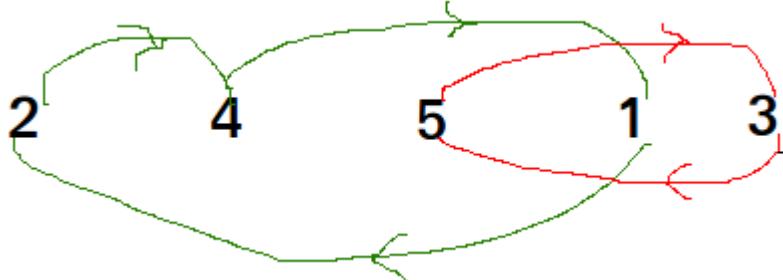
Java Program for Cycle Sort

Java Program for Cycle Sort - GeeksforGeeks

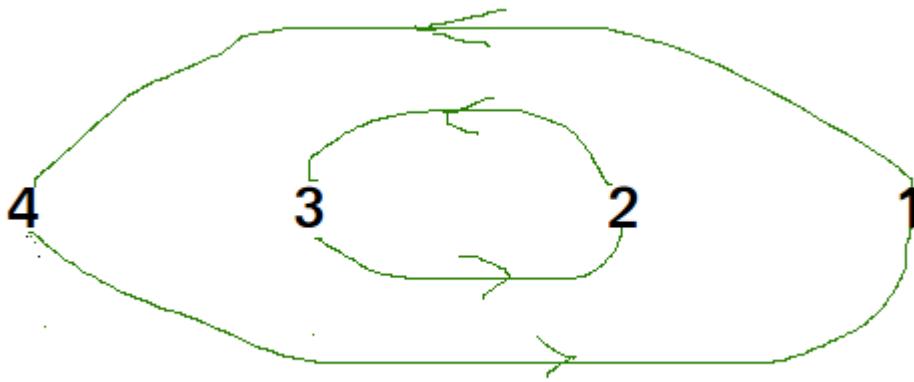
Cycle sort is an in-place sorting Algorithm, [unstable sorting algorithm](#), a comparison sort that is theoretically optimal in terms of the total number of writes to the original array.

- It is optimal in terms of number of memory writes. It [minimizes the number of memory writes](#) to sort (Each value is either written zero times, if it's already in its correct position, or written one time to its correct position.)
- It is based on the idea that array to be sorted can be divided into cycles. Cycles can be visualized as a graph. We have n nodes and an edge directed from node i to node j if the element at i-th index must be present at j-th index in the sorted array.

Cycle in arr[] = {4, 5, 2, 1, 5}



Cycle in arr[] = {4, 3, 2, 1}



We one by one consider all cycles. We first consider the cycle that includes first element. We find correct position of first element, place it at its correct position, say j. We consider old value of arr[j] and find its correct position, we keep doing this till all elements of current cycle are placed at correct position, i.e., we don't come back to cycle starting point.

Source

<https://www.geeksforgeeks.org/java-program-for-cycle-sort/>

Java

```
// Java program to implement cycle sort

import java.util.*;
import java.lang.*;

class GFG
{
// Function sort the array using Cycle sort
    public static void cycleSort (int arr[], int n)
    {
        // count number of memory writes
        int writes = 0;

        // traverse array elements and put it to on
        // the right place
        for (int cycle_start=0; cycle_start<=n-2; cycle_start++)
        {
            // initialize item as starting point
            int item = arr[cycle_start];
```

```
// Find position where we put the item. We basically
// count all smaller elements on right side of item.
int pos = cycle_start;
for (int i = cycle_start+1; i<n; i++)
    if (arr[i] < item)
        pos++;

// If item is already in correct position
if (pos == cycle_start)
    continue;

// ignore all duplicate elements
while (item == arr[pos])
    pos += 1;

// put the item to it\'s right position
if (pos != cycle_start)
{
    int temp = item;
    item = arr[pos];
    arr[pos] = temp;
    writes++;
}

// Rotate rest of the cycle
while (pos != cycle_start)
{
    pos = cycle_start;

    // Find position where we put the element
    for (int i = cycle_start+1; i<n; i++)
        if (arr[i] < item)
            pos += 1;

    // ignore all duplicate elements
    while (item == arr[pos])
        pos += 1;

    // put the item to it\'s right position
    if (item != arr[pos])
    {
        int temp = item;
        item = arr[pos];
        arr[pos] = temp;
        writes++;
    }
}
```

```
        }
    }

// Driver program to test above function
public static void main(String[] args)
{
    int arr[] = {1, 8, 3, 9, 10, 10, 2, 4 };
    int n = arr.length;
    cycleSort(arr, n) ;

    System.out.println("After sort : ");
    for (int i =0; i<n; i++)
        System.out.print(arr[i] + " ");
}
}

// Code Contributed by Mohit Gupta_OMG <(0_o)>
```

Output:

```
After sort :
1 2 3 4 8 9 10 10
```

Please refer complete article on [Cycle Sort](#) for more details!

Chapter 148

Java Program for Heap Sort

Java Program for Heap Sort - GeeksforGeeks

Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

Source

<https://www.geeksforgeeks.org/java-program-for-heap-sort/>

Java

```
// Java program for implementation of Heap Sort
public class HeapSort
{
    public void sort(int arr[])
    {
        int n = arr.length;

        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i=n-1; i>=0; i--)
        {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
```

```
// call max heapify on the reduced heap
heapify(arr, i, 0);
}

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
    int largest = i; // Initialize largest as root
    int l = 2*i + 1; // left = 2*i + 1
    int r = 2*i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i)
    {
        int swap = arr[i];
        arr[i] = arr[largest];
        arr[largest] = swap;

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

/* A utility function to print array of size n */
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i]+" ");
    System.out.println();
}

// Driver program
public static void main(String args[])
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = arr.length;
```

```
HeapSort ob = new HeapSort();
ob.sort(arr);

System.out.println("Sorted array is");
printArray(arr);
}

}
```

Please refer complete article on [Heap Sort](#) for more details!

Chapter 149

Java Program for Iterative Merge Sort

Java Program for Iterative Merge Sort - GeeksforGeeks

Following is a typical recursive implementation of [Merge Sort](#) that uses last element as pivot.

Source

<https://www.geeksforgeeks.org/java-program-for-iterative-merge-sort/>

Java

```
// Recursive Java Program for merge sort

import java.util.Arrays;
public class GFG
{
    public static void mergeSort(int[] array)
    {
        if(array == null)
        {
            return;
        }

        if(array.length > 1)
        {
            int mid = array.length / 2;

            // Split left part
            int[] left = new int[mid];
            int[] right = new int[array.length - mid];

            mergeSort(left);
            mergeSort(right);

            merge(left, right, array);
        }
    }
}
```

```
for(int i = 0; i < mid; i++)
{
    left[i] = array[i];
}

// Split right part
int[] right = new int[array.length - mid];
for(int i = mid; i < array.length; i++)
{
    right[i - mid] = array[i];
}
mergeSort(left);
mergeSort(right);

int i = 0;
int j = 0;
int k = 0;

// Merge left and right arrays
while(i < left.length && j < right.length)
{
    if(left[i] < right[j])
    {
        array[k] = left[i];
        i++;
    }
    else
    {
        array[k] = right[j];
        j++;
    }
    k++;
}
// Collect remaining elements
while(i < left.length)
{
    array[k] = left[i];
    i++;
    k++;
}
while(j < right.length)
{
    array[k] = right[j];
    j++;
    k++;
}
}
```

```
// Driver program to test above functions.  
public static void main(String[] args)  
{  
    int arr[] = {12, 11, 13, 5, 6, 7};  
    int i=0;  
    System.out.println("Given array is");  
  
    for(i=0; i<arr.length; i++)  
        System.out.print(arr[i]+" ");  
  
    mergeSort(arr);  
  
    System.out.println("\n");  
    System.out.println("Sorted array is");  
  
    for(i=0; i<arr.length; i++)  
        System.out.print(arr[i]+" ");  
}  
}  
  
// Code Contributed by Mohit Gupta_OMG
```

Please refer complete article on [Iterative Merge Sort](#) for more details!

Chapter 150

Java Program for QuickSort

Java Program for QuickSort - GeeksforGeeks

Like [Merge Sort](#), QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Pseudo Code for recursive QuickSort function :

```
/* low --> Starting index, high --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1); // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

Source

<https://www.geeksforgeeks.org/java-program-for-quicksort/>

Java

```
// Java program for implementation of QuickSort
class QuickSort
{
    /* This function takes last element as pivot,
       places the pivot element at its correct
       position in sorted array, and places all
       smaller (smaller than pivot) to left of
       pivot and all greater elements to right
       of pivot */
    int partition(int arr[], int low, int high)
    {
        int pivot = arr[high];
        int i = (low-1); // index of smaller element
        for (int j=low; j<high; j++)
        {
            // If current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;
                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }

    /* The main function that implements QuickSort()
       arr[] --> Array to be sorted,
       low --> Starting index,
       high --> Ending index */
    void sort(int arr[], int low, int high)
```

```
{  
    if (low < high)  
    {  
        /* pi is partitioning index, arr[pi] is  
           now at right place */  
        int pi = partition(arr, low, high);  
  
        // Recursively sort elements before  
        // partition and after partition  
        sort(arr, low, pi-1);  
        sort(arr, pi+1, high);  
    }  
}  
  
/* A utility function to print array of size n */  
static void printArray(int arr[])  
{  
    int n = arr.length;  
    for (int i=0; i<n; ++i)  
        System.out.print(arr[i]+" ");  
    System.out.println();  
}  
  
// Driver program  
public static void main(String args[])  
{  
    int arr[] = {10, 7, 8, 9, 1, 5};  
    int n = arr.length;  
  
    QuickSort ob = new QuickSort();  
    ob.sort(arr, 0, n-1);  
  
    System.out.println("sorted array");  
    printArray(arr);  
}  
}  
/*This code is contributed by Rajat Mishra */
```

Please refer complete article on [QuickSort](#) for more details!

Chapter 151

Java Program for Recursive Bubble Sort

Java Program for Recursive Bubble Sort - GeeksforGeeks

Background :

[Bubble Sort](#) is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Following is iterative Bubble sort algorithm :

```
// Iterative Bubble Sort
bubbleSort(arr[], n)
{
    for (i = 0; i < n-1; i++)

        // Last i elements are already in place
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(arr[j], arr[j+1]);
}
```

Recursion Idea.

1. Base Case: If array size is 1, return.
2. Do One Pass of normal Bubble Sort. This pass fixes last element of current subarray.
3. Recur for all elements except last of current subarray.

Source

<https://www.geeksforgeeks.org/java-program-for-recursive-bubble-sort/>

Java

```
// Java program for recursive implementation
// of Bubble sort

import java.util.Arrays;

public class GFG
{
    // A function to implement bubble sort
    static void bubbleSort(int arr[], int n)
    {
        // Base case
        if (n == 1)
            return;

        // One pass of bubble sort. After
        // this pass, the largest element
        // is moved (or bubbled) to end.
        for (int i=0; i<n-1; i++)
            if (arr[i] > arr[i+1])
            {
                // swap arr[i], arr[i+1]
                int temp = arr[i];
                arr[i] = arr[i+1];
                arr[i+1] = temp;
            }

        // Largest element is fixed,
        // recur for remaining array
        bubbleSort(arr, n-1);
    }

    // Driver Method
    public static void main(String[] args)
    {
        int arr[] = {64, 34, 25, 12, 22, 11, 90};

        bubbleSort(arr, arr.length);

        System.out.println("Sorted array : ");
        System.out.println(Arrays.toString(arr));
    }
}
```

Please refer complete article on [Recursive Bubble Sort](#) for more details!

Chapter 152

Java Program for Recursive Insertion Sort

Java Program for Recursive Insertion Sort - GeeksforGeeks

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Below is an iterative algorithm for insertion sort

Algorithm

```
// Sort an arr[] of size n
insertionSort(arr, n)
    Loop from i = 1 to n-1.
        a) Pick element arr[i] and insert
            it into sorted sequence arr[0..i-1]
```

Source

<https://www.geeksforgeeks.org/java-program-for-recursive-insertion-sort/>

Java

```
// Recursive Java program for insertion sort

import java.util.Arrays;

public class GFG
{
    // Recursive function to sort an array using
    // insertion sort
    static void insertionSortRecursive(int arr[], int n)
```

```
{  
    // Base case  
    if (n <= 1)  
        return;  
  
    // Sort first n-1 elements  
    insertionSortRecursive( arr, n-1 );  
  
    // Insert last element at its correct position  
    // in sorted array.  
    int last = arr[n-1];  
    int j = n-2;  
  
    /* Move elements of arr[0..i-1], that are  
       greater than key, to one position ahead  
       of their current position */  
    while (j >= 0 && arr[j] > last)  
    {  
        arr[j+1] = arr[j];  
        j--;  
    }  
    arr[j+1] = last;  
}  
  
// Driver Method  
public static void main(String[] args)  
{  
    int arr[] = {12, 11, 13, 5, 6};  
  
    insertionSortRecursive(arr, arr.length);  
  
    System.out.println(Arrays.toString(arr));  
}  
}
```

Please refer complete article on [Recursive Insertion Sort](#) for more details!

Chapter 153

Java Program for Stooge Sort

Java Program for Stooge Sort - GeeksforGeeks

The Stooge sort is a recursive sorting algorithm. It is defined as below (for ascending order sorting).

Step 1 : If value at index 0 is greater than
value at last index, swap them.
Step 2: Recursively,
a) Stooge sort the initial 2/3rd of the array.
b) Stooge sort the last 2/3rd of the array.
c) Stooge sort the initial 2/3rd again to confirm.

```
// Java program to implement stooge sort
import java.io.*;

public class stooge
{
    // Function to implement stooge sort
    static void stoogesort(int arr[], int l, int h)
    {
        if (l >= h)
            return;

        // If first element is smaller
        // than last,swap them
        if (arr[l] > arr[h])
        {
            int t = arr[l];
            arr[l] = arr[h];
            arr[h] = t;
        }
    }
}
```

```
// If there are more than 2 elements in
// the array
if (h-l+1 > 2)
{
    int t = (h-l+1) / 3;

    // Recursively sort first 2/3 elements
    stoogesort(arr, l, h-t);

    // Recursively sort last 2/3 elements
    stoogesort(arr, l+t, h);

    // Recursively sort first 2/3 elements
    // again to confirm
    stoogesort(arr, l, h-t);
}

// Driver Code
public static void main(String args[])
{
    int arr[] = {2, 4, 5, 3, 1};
    int n = arr.length;

    stoogesort(arr, 0, n-1);

    for (int i=0; i < n; i++)
        System.out.print(arr[i] + " ");
}
// Code Contributed by Mohit Gupta_OMG <(0_o)>
```

Output:

1 2 3 4 5

Please refer complete article on [Stooge Sort](#) for more details!

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/java-program-for-stooge-sort/>

Chapter 154

Job Selection Problem – Loss Minimization Strategy Set 2

Job Selection Problem - Loss Minimization Strategy Set 2 - GeeksforGeeks

We have discussed one loss minimization strategy before: [Job Sequencing Problem – Loss Minimization](#). In this article, we will look at another strategy that applies to a slightly different problem.

We are given a sequence of N goods of production numbered from 1 to N. Each good has a volume denoted by (V_i). The constraint is that once a good has been completed its volume starts decaying at a fixed percentage (P) per day. All goods decay at the same rate and further each good take one day to complete.

We are required to find the order in which the goods should be produced so that overall volume of goods is maximized.

Example-1:

Input: 4, 2, 151, 15, 1, 52, 12 and $P = 10\%$

Output: 222.503

Solution: In the optimum sequence of jobs, the total volume of goods left at the end of all jobs is 222.503

Example-2:

Input: 3, 1, 41, 52, 15, 4, 1, 63, 12 and $P = 20\%$

Output: 145.742

Solution: In the optimum sequence of jobs the total volume of goods left at the end of all jobs is 145.72

Explanation –

Since this is an optimization problem, we can try to solve this problem by using a greedy algorithm. On each day we make a selection from among the goods that are yet to be produced. Thus all we need is a local selection criteria or heuristic, which when applied to select the jobs will give us the optimum result.

Instead of trying to maximize the volume, we can also try to minimize the losses. Since the total volume that can be obtained from all goods is also constant, if we minimize the losses we are guaranteed to get the optimum answer.

Now consider any good having volume V

Loss after Day 1: PV

Loss after Day 2: $PV + P(1-P)V$ or $V(2P - P^2)$

Loss after Day 3: $V(2P - P^2) + P(1 - 2P + P^2)V$ or $V(3P - 3P^2 + P^3)$

As the day increases the losses too increase. So the trick would be to ensure that the goods are not kept idle after production. Further, since we are required to produce at least one job per day, we should perform low volume jobs, and then perform the high volume jobs. This strategy works due to two factors.

1. High Volume goods are not kept idle after production.
2. As the volume decreases the loss per day too decreases, so for low volume goods the losses become negligible after a few days.

So in order to obtain the optimum solution we produce the larger volume goods later on. For the first day select the good with least volume and produce it. Remove the produced good from the list of goods. For the next day repeat the same. Keep repeating while there are goods left to be produced.

When calculating the total volume at the end of production, keep in mind the the

good produced on day i, will have $\frac{V_i}{A_i}$ times its volume left. Ev-
idently, the good produced on day N (last day) will have its volume intact since
 $\frac{V_N}{A_N} = \frac{V_N}{1 - P} = 1$

Algorithm –

```

Step 1: Add all the goods to a min-heap
Step 2: Repeat following steps while Queue is not empty
        Extract the good at the head of the heap
        Print the good
        Remove the good from the heap
        [END OF LOOP]
Step 4: End
    
```

Complexity –

We perform exactly N push() and pop() operations each of which takes log (N) time. Hence time complexity is O(N * log(N)).

Below is the Cpp implementation of the solution.

```
#include <bits/stdc++.h>
using namespace std;

void optimum_sequence_jobs(vector<int>& V, double P)
{
    int j = 1, N = V.size() - 1;
    double result = 0;

    // Create a min-heap (priority queue)
    priority_queue<int, vector<int>, greater<int> > Queue;

    // Add all goods to the the Queue
    for (int i = 1; i <= N; i++)
        Queue.push(V[i]);

    // Pop Goods from Queue as long as it is not empty
    while (!Queue.empty()) {

        // Print the good
        cout << Queue.top() << " ";

        // Add the Queue to the vector
        // so that total volume can be calculated
        V[j++] = Queue.top();
        Queue.pop();
    }

    // Calculating volume of goods left when all
    // are produced. Move from right to left of
    // sequence multiplying each volume by
    // increasing powers of 1 - P starting from 0
    for (int i = N; i >= 1; i--)
        result += pow((1 - P), N - i) * V[i];

    // Print result
    cout << endl << result << endl;
}

// Driver code
int main()
{
    // For implementation simplicity days are numbered
    // from 1 to N. Hence 1 based indexing is used
    vector<int> V{ -1, 3, 5, 4, 1, 2, 7, 6, 8, 9, 10 };

    // 10% loss per day
```

```
double P = 0.10;  
  
optimum_sequence_jobs(V, P);  
  
return 0;  
}
```

Output –

```
1 2 3 4 5 6 7 8 9 10  
41.3811
```

Source

<https://www.geeksforgeeks.org/job-selection-problem-loss-minimization-strategy-set-2/>

Chapter 155

Job Sequencing Problem – Loss Minimization

Job Sequencing Problem - Loss Minimization - GeeksforGeeks

We are given N jobs numbered 1 to N. For each activity, let T_i denotes the number of days required to complete the job. For each day of delay before starting to work for job i , a loss of L_i is incurred.

We are required to find a sequence to complete the jobs so that overall loss is minimized. We can only work on one job at a time.

If multiple such solutions are possible, then we are required to give the lexicographically least permutation (i.e earliest in dictionary order).

Examples:

```
Input : L = {3, 1, 2, 4} and
        T = {4, 1000, 2, 5}
Output : 3, 4, 1, 2
Explanation: We should first complete
job 3, then jobs 4, 1, 2 respectively.
```

```
Input : L = {1, 2, 3, 5, 6}
        T = {2, 4, 1, 3, 2}
Output : 3, 5, 4, 1, 2
Explanation: We should complete jobs
3, 5, 4, 1 and then 2 in this order.
```

Let us consider two extreme cases and we shall deduce the general case solution from them.

All jobs take same time to finish, i.e $T_i = k$ for all i . Since all jobs take same time to finish we should first select jobs which have large Loss (L_i). We should select jobs which have the

highest losses and finish them as early as possible.

Thus this is a greedy algorithm. Sort the jobs in descending order based on L_i only.

All jobs have the same penalty. Since all jobs have the same penalty we will do those jobs first which will take less amount of time to finish. This will minimize the total delay, and hence also the total loss incurred.

This is also a greedy algorithm. Sort the jobs in ascending order based on T_i . Or we can also sort in descending order of $1/T_i$.

Source

<https://www.geeksforgeeks.org/job-sequencing-problem-loss-minimization/>

Chapter 156

K-th smallest element after removing some integers from natural numbers

K-th smallest element after removing some integers from natural numbers - GeeksforGeeks

Given an array **arr[]** of size ‘n’ and a positive integer **k**. Consider series of natural numbers and remove arr[0], arr[1], arr[2], ..., arr[p] from it. Now the task is to find k-th smallest number in the remaining set of natural numbers. If no such number exists print “-1”.

Examples :

```
Input : arr[] = { 1 } and k = 1.  
Output: 2  
Natural numbers are {1, 2, 3, 4, .... }  
After removing {1}, we get {2, 3, 4, ... }.  
Now, K-th smallest element = 2.
```

```
Input : arr[] = {1, 3}, k = 4.  
Output : 6  
First 5 Natural number {1, 2, 3, 4, 5, 6, ... }  
After removing {1, 3}, we get {2, 4, 5, 6, ... }.
```

Method 1 (Simple):

Make an auxiliary array **b[]** for presence/absence of natural numbers and initialize all with 0. Make all the integer equal to 1 which are present in array **arr[]** i.e **b[arr[i]] = 1**. Now, run a loop and decrement **k** whenever unmarked cell is encountered. When the value of **k** is 0, we get the answer.

Below is C++ implementation of this approach:

C++

```
// C++ program to find the K-th smallest element
// after removing some integers from natural number.
#include<bits/stdc++.h>
#define MAX 1000000
using namespace std;

// Return the K-th smallest element.
int ksmallest(int arr[], int n, int k)
{
    // Making an array, and mark all number as unmarked.
    int b[MAX];
    memset(b, 0, sizeof b);

    // Marking the number present in the given array.
    for (int i = 0; i < n; i++)
        b[arr[i]] = 1;

    for (int j=1; j<MAX; j++)
    {
        // If j is unmarked, reduce k by 1.
        if (b[j] != 1)
            k--;
    }

    // If k is 0 return j.
    if (!k)
        return j;
}

// Driven Program
int main()
{
    int k = 1;
    int arr[] = { 1 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << ksmallest(arr, n, k);
    return 0;
}
```

Output :

2

Time Complexity : O(n).

Method 2 (Efficient):

First, sort the array arr[]. Observe, there will be $\text{arr}[0] - 1$ numbers between 0 and $\text{arr}[0]$, similarly, $\text{arr}[1] - \text{arr}[0] - 1$ numbers between $\text{arr}[0]$ and $\text{arr}[1]$ and so on. So, if k lies between $\text{arr}[i] - \text{arr}[i+1] - 1$, then return K-th smallest element in the range. Else reduce k by $\text{arr}[i] - \text{arr}[i+1] - 1$ i.e., $k = k - (\text{arr}[i] - \text{arr}[i+1] - 1)$.

Algorithm to solve the problem:

1. Sort the array arr[].
2. For $i = 1$ to k . Find $c = \text{arr}[i+1] - \text{arr}[i] - 1$.
 - a) if $k - c \leq 0$, return $\text{arr}[i-1] + k$.
 - b) else $k = k - c$.

Below is implementation of this approach:

C++

```
// C++ program to find the Kth smallest element // after removing some integer from first n //
```

Java

```
// Java program to find the // Kth smallest element after // removing some integer from // f
```

C#

```
// C# program to find the // Kth smallest element after // removing some integer from // fir
```

PHP

```
<?php // PHP program to find the Kth // smallest element after // removing some integer from
```

Output :

2

More efficient method : [K-th smallest element after removing given integers from natural numbers Set 2](#)

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/k-th-smallest-element-removing-integers-natural-numbers/>

Chapter 157

Know Your Sorting Algorithm Set 1 (Sorting Weapons used by Programming Languages)

Know Your Sorting Algorithm Set 1 (Sorting Weapons used by Programming Languages) - GeeksforGeeks

Ever wondered how `sort()` function we use in C++/Java or `sorted()` in Python work internally?

Here is a list of all the inbuilt sorting algorithms of different programming languages and the algorithm they use internally.

1. C's `qsort()` – [Quicksort](#)

- Best Case Time Complexity- $O(N \log N)$
- Average Case Time Complexity- $O(N \log N)$
- Worse Case Time Complexity- $O(N^2)$
- Auxiliary Space- $O(\log N)$
- Stable- Depends on the implementation of the comparator function
- Adaptive- No

2. C++'s `sort()` – [Introsort](#) (Hybrid of [Quicksort](#), [Heap Sort](#) and [Insertion Sort](#))

- Best Case Time Complexity- $O(N \log N)$
- Average Case Time Complexity- $O(N \log N)$
- Worse Case Time Complexity- $O(N \log N)$
- Auxiliary Space- $O(\log N)$

- Stable- No
- Adaptive- No

3. C++'s `stable_sort()` – Mergesort

- Best Case Time Complexity- $O(N \log N)$
- Average Case Time Complexity- $O(N \log N)$
- Worse Case Time Complexity- $O(N \log N)$
- Auxiliary Space- $O(N)$
- Stable- Yes
- Adaptive- Yes

4. Java 6's `Arrays.sort()` – Quicksort

- Best Case Time Complexity- $O(N \log N)$
- Average Case Time Complexity- $O(N \log N)$
- Worse Case Time Complexity- $O(N^2)$
- Auxiliary Space- $O(\log N)$
- Stable- Depends
- Adaptive- No

5. Java 7's `Arrays.sort()` – Timsort (Hybrid of Mergesort and Insertion Sort)

- Best Case Time Complexity- $O(N)$
- Average Case Time Complexity- $O(N \log N)$
- Worse Case Time Complexity- $O(N \log N)$
- Auxiliary Space- $O(N)$
- Stable- Yes
- Adaptive- Yes

6. Java's `Collections.sort()` – Mergesort

- Best Case Time Complexity- $O(N \log N)$
- Average Case Time Complexity- $O(N \log N)$
- Worse Case Time Complexity- $O(N \log N)$
- Auxiliary Space- $O(N)$
- Stable- Yes
- Adaptive- Yes

7. Python's `sorted()` – Timsort (Hybrid of Mergesort and Insertion Sort)

- Best Case Time Complexity- $O(N)$
- Average Case Time Complexity- $O(N \log N)$

- Worse Case Time Complexity- $O(N \log N)$
- Auxiliary Space- $O(N)$
- Stable- Yes
- Adaptive- Yes

8. Python's `sort()` – Timsort (Hybrid of Mergesort and Insertion Sort)

- Best Case Time Complexity- $O(N)$
- Average Case Time Complexity- $O(N \log N)$
- Worse Case Time Complexity- $O(N \log N)$
- Auxiliary Space- $O(N)$
- Stable- Yes
- Adaptive- Yes

In the next sets we will implement Introsort (C++'s sorting weapon) and Sleep sort, Gnome Sort and other unconventional sorting algorithms.

Source

<https://www.geeksforgeeks.org/know-sorting-algorithm-set-1-sorting-weapons-used-programming-languages/>

Chapter 158

Know Your Sorting Algorithm Set 2 (Introsort- C++'s Sorting Weapon)

Know Your Sorting Algorithm Set 2 (Introsort- C++'s Sorting Weapon) - GeeksforGeeks

We have discussed [sorting weapons used by different languages](#) in previous article. In this article, C++'s Sorting Weapon, Introsort is discussed.

What is Introsort?

Simply putting, it is the best sorting algorithm around. It is a hybrid sorting algorithm, which means that it uses more than one sorting algorithms as a routine.

Which standard sorting algorithms are used in Introsort

Introsort being a hybrid sorting algorithm uses three sorting algorithm to minimise the running time, [Quicksort](#), [Heapsort](#) and [Insertion Sort](#)

How does it work?

Introsort begins with quicksort and if the recursion depth goes more than a particular limit it switches to Heapsort to avoid Quicksort's worse case $O(N^2)$ time complexity. It also uses insertion sort when the number of elements to sort is quite less.

So first it creates a partition. Three cases arises from here.

1. If the partition size is such that there is a possibility to exceed the maximum depth limit then the Introsort switches to Heapsort. We define the maximum depth limit as $2*\log(N)$
2. If the partition size is too small then Quicksort decays to Insertion Sort. We define this cutoff as 16 (due to research). So if the partition size is less than 16 then we will do insertion sort.
3. If the partition size is under the limit and not too small (i.e- between 16 and $2*\log(N)$), then it performs a simple quicksort.

Why is it better than simple Quicksort or Why the need of Introsort?

Since Quicksort can have a worse case $O(N^2)$ time complexity and it also increases the recursion stack space ($O(\log N)$ if tail recursion applied), so to avoid all these, we need to switch the algorithm from Quicksort to another if there is a chance of worse case. So Introsort solves this problem by switching to Heapsort.

Also due to larger constant factor, quicksort can perform even worse than $O(N^2)$ sorting algorithm when N is small enough. So it switches to insertion sort to decrease the running time of sorting.

Also if a bad pivot-selection is done then the quicksort does no better than the bubble-sort.

Why is Insertion Sort used (and not Bubble Sort, etc)?

Insertion sort offers following advantages.

1. It is a known and established fact that insertion sort is the most optimal comparison-based sorting algorithm for small arrays.
2. It has a good locality of reference
3. It is an adaptive sorting algorithm, i.e- it outperforms all the other algorithms if the array elements are partially sorted.

Why is Heapsort used (and not Mergesort etc)?

This is solely because of memory requirements. Merge sort requires $O(N)$ space whereas Heapsort is an in-place $O(1)$ space algorithm.

Why is Heapsort not used in place of Quicksort when the partition size is under the limit ?

This question is same as why Quicksort generally outperforms Heapsort ?

The answer is, although Heapsort also being $O(N \log N)$ in average as well as worse case and $O(1)$ space also, we still don't use it when the partition size is under the limit because the extra hidden constant factor in Heapsort is quite larger than that of Quicksort.

Why is cut-off 16 for switching from quick sort to insertion sort, and $2*\log N$ for switching from quick sort to heap sort ?

These values are chosen empirically as an approximate because of various tests and researches conducted.

```
/* A Program to sort the array using Introsort.  
The most popular C++ STL Algorithm- sort()  
uses Introsort. */  
  
#include<bits/stdc++.h>  
using namespace std;  
  
// A utility function to swap the values pointed by  
// the two pointers  
void swapValue(int *a, int *b)
```

```
{  
    int *temp = a;  
    a = b;  
    b = temp;  
    return;  
}  
  
/* Function to sort an array using insertion sort*/  
void InsertionSort(int arr[], int *begin, int *end)  
{  
    // Get the left and the right index of the subarray  
    // to be sorted  
    int left = begin - arr;  
    int right = end - arr;  
  
    for (int i = left+1; i <= right; i++)  
    {  
        int key = arr[i];  
        int j = i-1;  
  
        /* Move elements of arr[0..i-1], that are  
           greater than key, to one position ahead  
           of their current position */  
        while (j >= left && arr[j] > key)  
        {  
            arr[j+1] = arr[j];  
            j = j-1;  
        }  
        arr[j+1] = key;  
    }  
  
    return;  
}  
  
// A function to partition the array and return  
// the partition point  
int* Partition(int arr[], int low, int high)  
{  
    int pivot = arr[high]; // pivot  
    int i = (low - 1); // Index of smaller element  
  
    for (int j = low; j <= high- 1; j++)  
    {  
        // If current element is smaller than or  
        // equal to pivot  
        if (arr[j] <= pivot)  
        {  
            // increment index of smaller element
```

```
i++;

    swap(arr[i], arr[j]);
}
}

swap(arr[i + 1], arr[high]);
return (arr + i + 1);
}

// A function that find the middle of the
// values pointed by the pointers a, b, c
// and return that pointer
int *MedianOfThree(int * a, int * b, int * c)
{
    if (*a < *b && *b < *c)
        return (b);

    if (*a < *c && *c <= *b)
        return (c);

    if (*b <= *a && *a < *c)
        return (a);

    if (*b < *c && *c <= *a)
        return (c);

    if (*c <= *a && *a < *b)
        return (a);

    if (*c <= *b && *b <= *c)
        return (b);
}

// A Utility function to perform intro sort
void IntrosortUtil(int arr[], int * begin,
                    int * end, int depthLimit)
{
    // Count the number of elements
    int size = end - begin;

    // If partition size is low then do insertion sort
    if (size < 16)
    {
        InsertionSort(arr, begin, end);
        return;
    }
}
```

```
// If the depth is zero use heapsort
if (depthLimit == 0)
{
    make_heap(begin, end+1);
    sort_heap(begin, end+1);
    return;
}

// Else use a median-of-three concept to
// find a good pivot
int * pivot = MedianOfThree(begin, begin+size/2, end);

// Swap the values pointed by the two pointers
swapValue(pivot, end);

// Perform Quick Sort
int * partitionPoint = Partition(arr, begin-arr, end-arr);
IntrosortUtil(arr, begin, partitionPoint-1, depthLimit - 1);
IntrosortUtil(arr, partitionPoint + 1, end, depthLimit - 1);

return;
}

/* Implementation of introsort*/
void Introsort(int arr[], int *begin, int *end)
{
    int depthLimit = 2 * log(end-begin);

    // Perform a recursive Introsort
    IntrosortUtil(arr, begin, end, depthLimit);

    return;
}

// A utility function ot print an array of size n
void printArray(int arr[], int n)
{
    for (int i=0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test Introsort
int main()
{
    int arr[] = {3, 1, 23, -9, 233, 23, -313, 32, -9};
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
// Pass the array, the pointer to the first element and  
// the pointer to the last element  
Introsort(arr, arr, arr+n-1);  
printArray(arr, n);  
  
    return(0);  
}
```

Output:

5 6 11 12 13

Is Introsort stable ?-

Since Quicksort is also not stable so Introsort is also not stable.

Time Complexity

Best Case – $O(N \log N)$

Average Case- $O(N \log N)$

Worse Case- $O(N \log N)$

where, N = number of elements to be sorted.

Auxiliary Space

Just like quicksort, it may use $O(\log N)$ auxiliary recursion stack space.

[Know Your Sorting Algorithm Set 2 \(Introsort- C++'s Sorting Weapon\)](#)

References

<https://en.wikipedia.org/wiki/Introsort>

Improved By : [callmeHK](#)

Source

<https://www.geeksforgeeks.org/know-your-sorting-algorithm-set-2-introsort-cs-sorting-weapon/>

Chapter 159

Least frequent element in an array

Least frequent element in an array - GeeksforGeeks

Given an array, find the least frequent element in it. If there are multiple elements that appear least number of times, print any one of them.

Examples :

```
Input : arr[] = {1, 3, 2, 1, 2, 2, 3, 1}
Output : 3
3 appears minimum number of times in given
array.
```

```
Input : arr[] = {10, 20, 30}
Output : 10 or 20 or 30
```

A **simple solution** is to run two loops. The outer loop picks all elements one by one. The inner loop finds frequency of the picked element and compares with the minimum so far. Time complexity of this solution is $O(n^2)$.

A **better solution** is to do sorting. We first sort the array, then linearly traverse the array.

C++

```
// CPP program to find the least frequent element
// in an array.
#include <bits/stdc++.h>
using namespace std;

int leastFrequent(int arr[], int n)
{
```

```
// Sort the array
sort(arr, arr + n);

// find the min frequency using linear traversal
int min_count = n+1, res = -1, curr_count = 1;
for (int i = 1; i < n; i++) {
    if (arr[i] == arr[i - 1])
        curr_count++;
    else {
        if (curr_count < min_count) {
            min_count = curr_count;
            res = arr[i - 1];
        }
        curr_count = 1;
    }
}

// If last element is least frequent
if (curr_count < min_count)
{
    min_count = curr_count;
    res = arr[n - 1];
}

return res;
}

// driver program
int main()
{
    int arr[] = {1, 3, 2, 1, 2, 2, 3, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << leastFrequent(arr, n);
    return 0;
}
```

Java

```
// Java program to find the least frequent element
// in an array.
import java.io.*;
import java.util.*;

class GFG {

    static int leastFrequent(int arr[], int n)
    {
```

```
// Sort the array
Arrays.sort(arr);

// find the min frequency using
// linear traversal
int min_count = n+1, res = -1;
int curr_count = 1;

for (int i = 1; i < n; i++) {
    if (arr[i] == arr[i - 1])
        curr_count++;
    else {
        if (curr_count < min_count) {
            min_count = curr_count;
            res = arr[i - 1];
        }
        curr_count = 1;
    }
}

// If last element is least frequent
if (curr_count < min_count)
{
    min_count = curr_count;
    res = arr[n - 1];
}

return res;
}

// driver program
public static void main(String args[])
{
    int arr[] = {1, 3, 2, 1, 2, 2, 3, 1};
    int n = arr.length;
    System.out.print(leastFrequent(arr, n));

}
}

/*This code is contributed by Nikita Tiwari.*/
```

Python3

```
# Python 3 program to find the least
# frequent element in an array.
```

```
def leastFrequent(arr, n) :  
  
    # Sort the array  
    arr.sort()  
  
    # find the min frequency using  
    # linear traversal  
    min_count = n + 1  
    res = -1  
    curr_count = 1  
    for i in range(1, n) :  
        if (arr[i] == arr[i - 1]) :  
            curr_count = curr_count + 1  
        else :  
            if (curr_count < min_count) :  
                min_count = curr_count  
                res = arr[i - 1]  
  
            curr_count = 1  
  
    # If last element is least frequent  
    if (curr_count < min_count) :  
        min_count = curr_count  
        res = arr[n - 1]  
  
    return res
```

```
# Driver program  
arr = [1, 3, 2, 1, 2, 2, 3, 1]  
n = len(arr)  
print(leastFrequent(arr, n))
```

```
# This code is contributed  
# by Nikita Tiwari.
```

C#

```
// C# program to find the least  
// frequent element in an array.  
using System;  
  
class GFG {  
  
    static int leastFrequent(int[] arr, int n)
```

```
{  
    // Sort the array  
    Array.Sort(arr);  
  
    // find the min frequency  
    // using linear traversal  
    int min_count = n + 1, res = -1;  
    int curr_count = 1;  
  
    for (int i = 1; i < n; i++)  
    {  
        if (arr[i] == arr[i - 1])  
            curr_count++;  
        else  
        {  
            if (curr_count < min_count)  
            {  
                min_count = curr_count;  
                res = arr[i - 1];  
            }  
  
            curr_count = 1;  
        }  
    }  
  
    // If last element is least frequent  
    if (curr_count < min_count)  
    {  
        min_count = curr_count;  
        res = arr[n - 1];  
    }  
  
    return res;  
}  
  
// Driver code  
static public void Main ()  
{  
    int[] arr = {1, 3, 2, 1, 2, 2, 3, 1};  
    int n = arr.Length;  
  
    // Function calling  
    Console.WriteLine(leastFrequent(arr, n));  
}  
}  
  
// This code is contributed by Shrikant13
```

PHP

```
<?php
// PHP program to find the
// least frequent element
// in an array.

function leastFrequent($arr, $n)
{

    // Sort the array
    sort($arr);
    sort($arr , $n);

    // find the min frequency
    // using linear traversal
    $min_count = $n + 1;
    $res = -1;
    $curr_count = 1;
    for($i = 1; $i < $n; $i++)
    {
        if ($arr[$i] == $arr[$i - 1])
            $curr_count++;
        else
        {
            if ($curr_count < $min_count)
            {
                $min_count = $curr_count;
                $res = $arr[$i - 1];
            }
            $curr_count = 1;
        }
    }

    // If last element is
    // least frequent
    if ($curr_count < $min_count)
    {
        $min_count = $curr_count;
        $res = $arr[$n - 1];
    }

    return $res;
}

// Driver Code
{
    $arr = array(1, 3, 2, 1, 2, 2, 3, 1);
```

```
$n = sizeof($arr) / sizeof($arr[0]);
echo leastFrequent($arr, $n);
return 0;
}

// This code is contributed by nitin mittal
?>
```

Output:

3

Time Complexity : $O(n \log n)$

Auxiliary Space : $O(1)$

An **efficient solution** is to use hashing. We create a hash table and store elements and their frequency counts as key value pairs. Finally we traverse the hash table and print the key with minimum value.

C++

```
// CPP program to find the least frequent element
// in an array.
#include <bits/stdc++.h>
using namespace std;

int leastFrequent(int arr[], int n)
{
    // Insert all elements in hash.
    unordered_map<int, int> hash;
    for (int i = 0; i < n; i++)
        hash[arr[i]]++;

    // find the min frequency
    int min_count = n+1, res = -1;
    for (auto i : hash) {
        if (min_count >= i.second) {
            res = i.first;
            min_count = i.second;
        }
    }

    return res;
}

// driver program
int main()
```

```
{  
    int arr[] = {1, 3, 2, 1, 2, 2, 3, 1};  
    int n = sizeof(arr) / sizeof(arr[0]);  
    cout << leastFrequent(arr, n);  
    return 0;  
}
```

Java

```
//Java program to find the least frequent element  
//in an array  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Map.Entry;  
  
class GFG {  
  
    static int leastFrequent(int arr[],int n)  
    {  
  
        // Insert all elements in hash.  
        Map<Integer,Integer> count =  
            new HashMap<Integer,Integer>();  
  
        for(int i = 0; i < n; i++)  
        {  
            int key = arr[i];  
            if(count.containsKey(key))  
            {  
                int freq = count.get(key);  
                freq++;  
                count.put(key,freq);  
            }  
            else  
                count.put(key,1);  
        }  
  
        // find min frequency.  
        int min_count = n+1, res = -1;  
        for(Entry<Integer,Integer> val : count.entrySet())  
        {  
            if (min_count >= val.getValue())  
            {  
                res = val.getKey();  
                min_count = val.getValue();  
            }  
        }  
    }  
}
```

```
        return res;
    }

// driver program
public static void main (String[] args) {

    int arr[] = {1, 3, 2, 1, 2, 2, 3, 1};
    int n = arr.length;

    System.out.println(leastFrequent(arr,n));
}
}

// This code is contributed by Akash Singh.
```

C#

```
// C# program to find the
// least frequent element
// in an array.
using System;
using System.Collections.Generic;

class GFG
{
    static int leastFrequent(int []arr,
                           int n)
    {
        // Insert all elements in hash.
        Dictionary<int, int> count =
            new Dictionary<int,
                           int>();
        for (int i = 0; i < n; i++)
        {
            int key = arr[i];
            if(count.ContainsKey(key))
            {
                int freq = count[key];
                freq++;
                count[key] = freq;
            }
            else
                count.Add(key, 1);
        }

        // find the min frequency
        int min_count = n + 1, res = -1;
        foreach (KeyValuePair<int,
```

```
int> pair in count)
{
    if (min_count >= pair.Value)
    {
        res = pair.Key;
        min_count = pair.Value;
    }
}
return res;
}

// Driver Code
static void Main()
{
    int []arr = new int[]{1, 3, 2, 1,
                        2, 2, 3, 1};
    int n = arr.Length;
    Console.WriteLine(leastFrequent(arr, n));
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

3

Time Complexity : O(n)
Auxiliary Space : O(n)

Improved By : [shrikanth13](#), [nitin mittal](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/least-frequent-element-array/>

Chapter 160

Lexicographical concatenation of all substrings of a string

Lexicographical concatenation of all substrings of a string - GeeksforGeeks

Given a string, find concatenation of all substrings in lexicographic order.

Examples:

Input : s = “abc”

Output : aababcbbcc

The **substrings** of s in lexicographic order are “a”, “b”, “c”, “ab”, “abc”, “bc”. Concatenation of substrings is “a”+”ab”+”abc”+”b”+”bc”+”c” = “aababcbbcc”

Input : s = “cba”

Output : abbaccbcba

1. Find all the substrings of string and store it in a string array. The size of array would be $n*(n+1)/2$ where n is length of input string.
2. Sort the string array to make them all in lexicographical order.
3. Concatenate the strings of string array in another empty string.

```
// CPP Program to create concatenation of all
// substrings in lexicographic order.
#include <bits/stdc++.h>
using namespace std;

string lexicographicSubConcat(string s)
{
    int n = s.length();

    // Creating an array to store substrings
```

```
int sub_count = n*(n+1)/2;
string arr[sub_count];

// finding all substrings of string
int index = 0;
for (int i = 0; i < n; i++)
    for (int len = 1; len <= n - i; len++)
        arr[index++] = s.substr(i, len);

// Sort all substrings in lexicographic
// order
sort(arr, arr + sub_count);

// Concatenating all substrings
string res = "";
for (int i = 0; i < sub_count; i++)
    res += arr[i];

return res;
}

int main()
{
    string s = "cba";
    cout << lexicographicSubConcat(s);
    return 0;
}
```

Output:

aababcbbcc

Source

<https://www.geeksforgeeks.org/lexicographical-concatenation-substrings-string/>

Chapter 161

Lexicographically smallest string obtained after concatenating array

Lexicographically smallest string obtained after concatenating array - GeeksforGeeks

Given n strings, concatenate them in an order that produces the lexicographically smallest possible string.

Examples:

```
Input : a[] = ["c", "cb", "cba"]
Output : cbacbc
Possible strings are ccdbcba, ccbacab,
cbccba, cbcbac, cbacbc and cbaccb.
Among all these strings, cbacbc is
the lexicographically smallest.
```

```
Input : a[] = ["aa", "ab", "aaa"]
Output : aaaaaab
```

One might think that sorting the given strings in the lexicographical order and then concatenating them produces the correct output. This approach produces the correct output for inputs like ["a", "ab", "abc"]. However, applying this method on ["c", "cb", "cba"] produces the wrong input and hence this approach is incorrect.

The correct approach is to use a regular sorting algorithm. When two strings a and b are compared to decide if they have to be swapped or not, do not check if a is lexicographically smaller than b or not. Instead check if appending b at the end of a produces a lexicographically smaller string or appending a at the end of b does. This approach works because we want the concatenated string to be lexicographically small, not the individual strings to be

in the lexicographical order.

C++

```
// CPP code to find the lexicographically
// smallest string
#include <bits/stdc++.h>
using namespace std;

// Compares two strings by checking if
// which of the two concatenations causes
// lexicographically smaller string.
bool compare(string a, string b)
{
    return (a+b < b+a);
}

string lexSmallest(string a[], int n)
{
    // Sort strings using above compare()
    sort(a, a+n, compare);

    // Concatenating sorted strings
    string answer = "";
    for (int i = 0; i < n; i++)
        answer += a[i];

    return answer;
}

// Driver code
int main()
{
    string a[] = { "c", "cb", "cba" };
    int n = sizeof(a)/sizeof(a[0]);
    cout << lexSmallest(a, n);
    return 0;
}
```

Java

```
// Java code to find the lexicographically
// smallest string

class GFG {

    // function to sort the
    // array of string
```

```
static void sort(String a[], int n)
{
    //sort the array
    for(int i = 0;i < n;i++)
    {
        for(int j = i + 1;j < n;j++)
        {

            // comparing which of the
            // two concatenation causes
            // lexicographically smaller
            // string
            if((a[i] + a[j]).compareTo(a[j] + a[i]) > 0)
            {
                String s = a[i];
                a[i] = a[j];
                a[j] = s;
            }
        }
    }
}

static String lexsmallest(String a[], int n)
{
    // Sort strings
    sort(a,n);

    // Concatenating sorted strings
    String answer = "";
    for (int i = 0; i < n; i++)
        answer += a[i];

    return answer;
}

// Driver code
public static void main(String args[])
{
    String a[] = {"c", "cb", "cba"};
    int n = 3;
    System.out.println("lexicographically smallest string = "
                      + lexsmallest(a, n));
}
```

```
// This code is contributed by Arnab Kundu
```

C#

```
// C# code to find
// the lexicographically
// smallest string
using System;

class GFG {

    // function to sort the
    // array of string
    static void sort(String []a, int n)
    {

        //sort the array
        for(int i = 0;i < n;i++)
        {
            for(int j = i + 1;j < n;j++)
            {

                // comparing which of the
                // two concatenation causes
                // lexiographically smaller
                // string
                if((a[i] + a[j]).CompareTo(a[j] +
                                            a[i]) > 0)
                {
                    String s = a[i];
                    a[i] = a[j];
                    a[j] = s;
                }
            }
        }
    }

    static String lexsmallest(String []a, int n)
    {

        // Sort strings
        sort(a,n);

        // Concatenating sorted
        // strings
        String answer = "";
        for (int i = 0; i < n; i++)
            answer += a[i];
    }
}
```

```
        return answer;
    }

// Driver code
public static void Main()
{
    String []a = {"c", "cb", "cba"};
    int n = 3;
    Console.WriteLine("lexicographically smallest string = "
                      + lexsmallest(a, n));

}
}

// This code is contributed by nitin mittal
```

Output:

cbacbc

Time complexity : The above code runs in $O(M * N * \log N)$ where N is number of strings and M is maximum length of a string.

Improved By : [andrew1234](#), [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/lexicographically-smallest-string-obtained-concatenating-array/>

Chapter 162

Longest Common Prefix using Sorting

Longest Common Prefix using Sorting - GeeksforGeeks

Problem Statement: Given a set of strings, find the longest common prefix.

Examples:

Input: {"geeksforgeeks", "geeks", "geek", "geezer"}
Output: "gee"

Input: {"apple", "ape", "april"}
Output: "ap"

The longest common prefix for an array of strings is the common prefix between 2 most dissimilar strings. For example, in the given array {"apple", "ape", "zebra"}, there is no common prefix because the 2 most dissimilar strings of the array "ape" and "zebra" do not share any starting characters.

We have discussed five different approaches in below posts.

1. [Word by Word Matching](#)
2. [Character by Character Matching](#)
3. [Divide and Conquer](#)
4. [Binary Search.](#)
5. [Using Trie\)](#)

In this post a new method based on sorting is discussed. The idea is to sort the array of strings and find the common prefix of the first and last string of the sorted array.

Java

```
// Java program to find longest common prefix of
// given array of words.
import java.util.*;

public class GFG
{
    public String longestCommonPrefix(String[] a)
    {
        int size = a.length;

        /* if size is 0, return empty string */
        if (size == 0)
            return "";

        if (size == 1)
            return a[0];

        /* sort the array of strings */
        Arrays.sort(a);

        /* find the minimum length from first and last string */
        int end = Math.min(a[0].length(), a[size-1].length());

        /* find the common prefix between the first and
           last string */
        int i = 0;
        while (i < end && a[0].charAt(i) == a[size-1].charAt(i) )
            i++;

        String pre = a[0].substring(0, i);
        return pre;
    }

    /* Driver Function to test other function */
    public static void main(String[] args)
    {
        GFG gfg = new GFG();
        String[] input = {"geeksforgeeks", "geeks", "geek", "geezer"};
        System.out.println("The longest Common Prefix is : " +
                           gfg.longestCommonPrefix(input));
    }
}
```

C#

```
// C# program to find longest common prefix of
// given array of words.
using System;
```

```
public class GFG {

    static string longestCommonPrefix(String[] a)
    {
        int size = a.Length;

        /* if size is 0, return empty string */
        if (size == 0)
            return "";

        if (size == 1)
            return a[0];

        /* sort the array of strings */
        Array.Sort(a);

        /* find the minimum length from first
        and last string */
        int end = Math.Min(a[0].Length,
                           a[size-1].Length);

        /* find the common prefix between the
        first and last string */
        int i = 0;
        while (i < end && a[0][i] == a[size-1][i] )
            i++;

        string pre = a[0].Substring(0, i);
        return pre;
    }

    /* Driver Function to test other function */
    public static void Main()
    {

        string[] input = {"geeksforgeeks", "geeks",
                         "geek", "geezer"};

        Console.WriteLine( "The longest Common"
                          + " Prefix is : "
                          + longestCommonPrefix(input));
    }
}

// This code is contributed by Sam007.
```

Output:

The longest common prefix is : gee

Time Complexity: $O(\text{MAX} * n * \log n)$ where n is the number of strings in the array and MAX is maximum number of characters in any string.

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/longest-common-prefix-using-sorting/>

Chapter 163

Longest Consecutive Subsequence

Longest Consecutive Subsequence - GeeksforGeeks

Given an array of integers, find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

Examples

Input: arr[] = {1, 9, 3, 10, 4, 20, 2};
Output: 4

The subsequence 1, 3, 4, 2 is the longest subsequence
of consecutive elements

Input: arr[] = {36, 41, 56, 35, 44, 33, 34, 92, 43, 32, 42}
Output: 5
The subsequence 36, 35, 33, 34, 32 is the longest subsequence
of consecutive elements.

One Solution is to first sort the array and find the longest subarray with consecutive elements. Time complexity of this solution is $O(n\log n)$. Thanks to Hao.W for suggesting this solution.

We can solve this problem in $O(n)$ time using an **Efficient Solution**. The idea is to use [Hashing](#). We first insert all elements in a Hash. Then check all the possible starts of consecutive subsequences. Below is complete algorithm.

- 1) Create an empty hash.
- 2) Insert all array elements to hash.
- 3) Do following for every element arr[i]

.....a) Check if this element is the starting point of a subsequence. To check this, we simply look for $\text{arr}[i] - 1$ in hash, if not found, then this is the first element a subsequence.

If this element is a first element, then count number of elements in the consecutive starting with this element.

If count is more than current res, then update res.

Below is C++ implementation of above algorithm.

C/C++

```
// C++ program to find longest contiguous subsequence
#include<bits/stdc++.h>
using namespace std;

// Returns length of the longest contiguous subsequence
int findLongestConseqSubseq(int arr[], int n)
{
    unordered_set<int> S;
    int ans = 0;

    // Hash all the array elements
    for (int i = 0; i < n; i++)
        S.insert(arr[i]);

    // check each possible sequence from the start
    // then update optimal length
    for (int i=0; i<n; i++)
    {
        // if current element is the starting
        // element of a sequence
        if (S.find(arr[i]-1) == S.end())
        {
            // Then check for next elements in the
            // sequence
            int j = arr[i];
            while (S.find(j) != S.end())
                j++;

            // update optimal length if this length
            // is more
            ans = max(ans, j - arr[i]);
        }
    }
}
```

```
        }
        return ans;
    }

// Driver program
int main()
{
    int arr[] = {1, 9, 3, 10, 4, 20, 2};
    int n = sizeof arr/ sizeof arr[0];
    cout << "Length of the Longest contiguous subsequence is "
         << findLongestConseqSubseq(arr, n);
    return 0;
}
```

Java

```
// Java program to find longest consecutive subsequence
import java.io.*;
import java.util.*;

class ArrayElements
{
    // Returns length of the longest consecutive subsequence
    static int findLongestConseqSubseq(int arr[],int n)
    {
        HashSet<Integer> S = new HashSet<Integer>();
        int ans = 0;

        // Hash all the array elements
        for (int i=0; i<n; ++i)
            S.add(arr[i]);

        // check each possible sequence from the start
        // then update optimal length
        for (int i=0; i<n; ++i)
        {
            // if current element is the starting
            // element of a sequence
            if (!S.contains(arr[i]-1))
            {
                // Then check for next elements in the
                // sequence
                int j = arr[i];
                while (S.contains(j))
                    j++;

                // update optimal length if this length
                // is more
            }
        }
    }
}
```

```
        if (ans < j - arr[i])
            ans = j - arr[i];
    }
}
return ans;
}

// Testing program
public static void main(String args[])
{
    int arr[] = {1, 9, 3, 10, 4, 20, 2};
    int n = arr.length;
    System.out.println("Length of the Longest consecutive subsequence is " +
                        findLongestConseqSubseq(arr, n));
}
}
// This code is contributed by Aakash Hasija
```

Python

```
# Python program to find longest contiguous subsequence

from sets import Set
def findLongestConseqSubseq(arr, n):

    s = Set()
    ans=0

    # Hash all the array elements
    for ele in arr:
        s.add(ele)

    # check each possible sequence from the start
    # then update optimal length
    for i in range(n):

        # if current element is the starting
        # element of a sequence
        if (arr[i]-1) not in s:

            # Then check for next elements in the
            # sequence
            j=arr[i]
            while(j in s):
                j+=1

            # update optimal length if this length
            # is more
```

```
ans=max(ans, j-arr[i])
return ans

# Driver function
if __name__=='__main__':
    n = 7
    arr = [1, 9, 3, 10, 4, 20, 2]
    print "Length of the Longest contiguous subsequence is ",
    print findLongestConseqSubseq(arr, n)

# Contributed by: Harshit Sidhwa
```

Output:

```
Length of the Longest contiguous subsequence is 4
```

Time Complexity: At first look, time complexity looks more than $O(n)$. If we take a closer look, we can notice that it is $O(n)$ under the assumption that hash insert and search take $O(1)$ time. The function S.find() inside the while loop is called at most twice for every element. For example, consider the case when all array elements are consecutive. In this case, the outer find is called for every element, but we go inside the if condition only for the smallest element. Once we are inside the if condition, we call find() one more time for every other element.

Thanks to [Gaurav Ahirwar](#) for above solution.

Source

<https://www.geeksforgeeks.org/longest-consecutive-subsequence/>

Chapter 164

Loop Invariant Condition with Examples of Sorting Algorithms

Loop Invariant Condition with Examples of Sorting Algorithms - GeeksforGeeks

Loop Invariant Condition:

Loop invariant condition is a condition about the relationship between the variables of our program which is definitely true immediately before and immediately after each iteration of the loop.

For example: Consider an array A{7, 5, 3, 10, 2, 6} with 6 elements and we have to find maximum element max in the array.

```
max = -INF (minus infinite)
for (i = 0 to n-1)
    if (A[i] > max)
        max = A[i]
```

In the above example after 3rd iteration of the loop max value is 7, which holds true for first 3 elements of array A. Here, the loop invariant condition is that max is always maximum among the first i elements of array A.

Loop Invariant condition of various algorithms:

Prerequisite: [insertion sort](#), [selection sort](#), [quick sort](#), [bubblesort](#),

Selection Sort:

In [selection sort](#) algorithm we find the minimum element from the unsorted part and put it at the beginning.

```
min_idx = 0
for (i = 0; i < n-1; i++)
```

```
{
    min_idx = i;
    for (j = i+1 to n-1)
        if (arr[j] < arr[min_idx])
            min_idx = j;

    swap(&arr[min_idx], &arr[i]);
}
```

In the above pseudo code there are two loop invariant condition:

1. In the outer loop, array is sorted for first i elements.
2. In the inner loop, min is always the minimum value in A[i to j].

Insertion Sort:

In [insertion sort](#), loop invariant condition is that the subarray A[0 to i-1] is always sorted.

```
for (i = 1 to n-1)
{
    key = arr[i];
    j = i-1;
    while (j >= 0 and arr[j] > key)
    {
        arr[j+1] = arr[j];
        j = j-1;
    }
    arr[j+1] = key;
}
```

Quicksort:

In [quicksort](#) algorithm, after every partition call array is divided into 3 regions:

1. Pivot element is placed at its correct position.
2. Elements less than pivot element lie on the left side of pivot element.
3. Elements greater than pivot element lie on the right side of pivot element.

```
quickSort(arr[], low, high)
{
    if (low < high)
    {
        pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1); // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}

partition (arr[], low, high)
{
```

```
pivot = arr[high];
i = (low - 1)
for (j = low; j <= high- 1; j++)
    if (arr[j] <= pivot)
        i++;
    swap arr[i] and arr[j]
swap arr[i + 1] and arr[high])
return (i + 1)
}
```

Bubble Sort:

In [bubble sort](#) algorithm, after each iteration of the loop largest element of the array is always placed at right most position. Therefore, the loop invariant condition is that at the end of i iteration right most i elements are sorted and in place.

```
for (i = 0 to n-1)
    for (j = 0 to j arr[j+1])
        swap(&arr[j], &arr[j+1]);
```

Source

<https://www.geeksforgeeks.org/loop-invariant-condition-examples-sorting-algorithms/>

Chapter 165

Lower bound for comparison based sorting algorithms

Lower bound for comparison based sorting algorithms - GeeksforGeeks

The problem of sorting can be viewed as following.

Input: A sequence of n numbers $\langle a_1, a_2, \dots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \dots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \dots \leq a'_n$.

A sorting algorithm is comparison based if it uses comparison operators to find the order between two numbers. Comparison sorts can be viewed abstractly in terms of decision trees. A decision tree is a [full binary tree](#) that represents the comparisons between elements that are performed by a particular sorting algorithm operating on an input of a given size. The execution of the sorting algorithm corresponds to tracing a path from the root of the decision tree to a leaf. At each internal node, a comparison $a_i \leq a_j$ is made. The left subtree then dictates subsequent comparisons for $a_i \leq a_j$, and the right subtree dictates subsequent comparisons for $a_i > a_j$. When we come to a leaf, the sorting algorithm has established the ordering. So we can say following about the decision tree.

1) Each of the $n!$ permutations on n elements must appear as one of the leaves of the decision tree for the sorting algorithm to sort properly.

2) Let x be the maximum number of comparisons in a sorting algorithm. The maximum height of the decision tree would be x . A tree with maximum height x has at most 2^x leaves.

After combining the above two facts, we get following relation.

$$n! \leq 2^x$$

Taking Log on both sides.

$$\log_2(n!) \leq x$$

Since $\log_2(n!) = \Theta(n \log n)$, we can say

$$x = \Omega(n \log 2n)$$

Therefore, any comparison based sorting algorithm must make at least $n \log_2 n$ comparisons to sort the input array, and Heapsort and merge sort are asymptotically optimal comparison sorts.

References:

Introduction to Algorithms, by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein

Source

<https://www.geeksforgeeks.org/lower-bound-on-comparison-based-sorting-algorithms/>

Chapter 166

Making elements of two arrays same with minimum increment/decrement

Making elements of two arrays same with minimum increment/decrement - GeeksforGeeks

Given two arrays of same size, we need to convert the first array into another with minimum operations. In an operation, we can either increment or decrement an element by one. Note that orders of appearance of elements do not need to be same.

Here to convert one number into another we can add or subtract 1 from it.

Examples :

Input : a = { 3, 1, 1 }, b = { 1, 2, 2 }

Output : 2

Explanation : Here we can increase any 1 into 2 by 1 operation and 3 to 2 in one decrement operation. So a[] becomes {2, 2, 1} which is a permutation of b[].

Input : a = { 3, 1, 1 }, b = { 1, 1, 2 }

Output : 1

Algorithm :

1. First sort both the arrays.
2. After sorting we will run a loop in which we compare the first and second array elements and calculate the required operation needed to make first array equal to second.

Below is implementation of the above approach

C++

```
// CPP program to find minimum increment/decrement  
// operations to make array elements same.
```

```
#include <bits/stdc++.h>
using namespace std;

int MinOperation(int a[], int b[], int n)
{
    // sorting both arrays in
    // ascending order
    sort(a, a + n);
    sort(b, b + n);

    // variable to store the
    // final result
    int result = 0;

    // After sorting both arrays
    // Now each array is in non-
    // decreasing order. Thus,
    // we will now compare each
    // element of the array and
    // do the increment or decrement
    // operation depending upon the
    // value of array b[].
    for (int i = 0; i < n; ++i) {
        if (a[i] > b[i])
            result = result + abs(a[i] - b[i]);

        else if (a[i] < b[i])
            result = result + abs(a[i] - b[i]);
    }

    return result;
}

// Driver code
int main()
{
    int a[] = { 3, 1, 1 };
    int b[] = { 1, 2, 2 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << MinOperation(a, b, n);
    return 0;
}
```

Java

```
// Java program to find minimum
// increment/decrement operations
// to make array elements same.
```

```
import java.util.Arrays;
import java.io.*;

class GFG
{
    static int MinOperation(int a[] ,
                           int b[] ,
                           int n)
    {
        // sorting both arrays
        // in ascending order
        Arrays.sort(a);
        Arrays.sort(b);

        // variable to store
        // the final result
        int result = 0;

        // After sorting both arrays
        // Now each array is in non-
        // decreasing order. Thus,
        // we will now compare each
        // element of the array and
        // do the increment or decrement
        // operation depending upon the
        // value of array b[] .
        for (int i = 0; i < n; ++i)
        {
            if (a[i] > b[i])
                result = result +
                    Math.abs(a[i] - b[i]);

            else if (a[i] < b[i])
                result = result +
                    Math.abs(a[i] - b[i]);
        }

        return result;
    }

    // Driver code
    public static void main (String[] args)
    {
        int a[] = {3, 1, 1};
        int b[] = {1, 2, 2};
        int n = a.length;
        System.out.println(MinOperation(a, b, n));
    }
}
```

```
}
```

```
}
```

```
// This code is contributed
```

```
// by akt_mit
```

PHP

```
<?php
// PHP program to find minimum
// increment/decrement operations
// to make array elements same.
function MinOperation($a, $b, $n)
{
    // sorting both arrays in
    // ascending order

    sort($a);
    sort($b);

    // variable to store
    // the final result
    $result = 0;

    // After sorting both arrays
    // Now each array is in non-
    // decreasing order. Thus,
    // we will now compare each
    // element of the array and
    // do the increment or decrement
    // operation depending upon the
    // value of array b[].
    for ($i = 0; $i < $n; ++$i)
    {
        if ($a[$i] > $b[$i])
            $result = $result + abs($a[$i] -
                $b[$i]);

        else if ($a[$i] < $b[$i])
            $result = $result + abs($a[$i] -
                $b[$i]);
    }

    return $result;
}

// Driver code
$a = array ( 3, 1, 1 );
```

```
$b = array ( 1, 2, 2 );
$n = sizeof($a);
echo MinOperation($a, $b, $n);

// This code is contributed by ajit
?>
```

Output :

2

Time Complexity : $O(n \log n)$

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/making-elements-of-two-arrays-same-with-minimum-incrementdecrement/>

Chapter 167

Maximise the number of toys that can be purchased with amount K

Maximise the number of toys that can be purchased with amount K - GeeksforGeeks

Given an array consisting of cost of toys. Given an integer K depicting the amount of money available to purchase toys. Write a program to find the maximum number of toys one can buy with the amount K.

Note: One can buy only 1 quantity of a particular toy.

Examples :

Input: N = 10, K = 50
cost = { 1, 12, 5, 111, 200, 1000, 10, 9, 12, 15 }
Output: 6

Explanation: Toys with amount 1, 5, 9, 10, 12, and 12 can be purchased resulting in a total amount of 49. Hence, maximum number of toys is 6.

Input: N = 7, K = 50
cost = { 1, 12, 5, 111, 200, 1000, 10 }
Output: 4

The idea to solve this problem is to first sort the *cost* array in ascending order. This will arrange the toys in increasing order of the cost. Now iterate over the cost array and keep calculating the sum of costs until the sum is less than or equal to K. Finally return the number of toys used to calculate the sum which is just less than or equals to K.

Below is the implementation of above approach:

C++

```
// C++ Program to maximize the
// number of toys with K amount
#include <bits/stdc++.h>
using namespace std;

// This functions returns the required
// number of toys
int maximum_toys(int cost[], int N, int K)
{
    int count = 0, sum = 0;

    // sort the cost array
    sort(cost, cost + N);
    for (int i = 0; i < N; i++) {

        // If the array element is less than K
        // add it to prefix sum and check this prefix
        // sum is less than given K
        if (cost[i] < K && sum <= K) {
            sum = sum + cost[i];

            // Increment the count variable
            if (sum <= K)
                count++;
        }
    }
    return count;
}

// Driver Code
int main()
{
    int K = 50;
    int cost[] = { 1, 12, 5, 111, 200, 1000, 10, 9, 12, 15 };
    int N = sizeof(cost) / sizeof(cost[0]);

    cout << maximum_toys(cost, N, K) << endl;
    return 0;
}
```

Java

```
// Java Program to maximize the
// number of toys with K amount
import java.io.*;
import java.util.*;

class GFG
```

```
{  
// This functions returns  
// the required number of toys  
static int maximum_toys(int cost[],  
                        int N, int K)  
{  
    int count = 0, sum = 0;  
  
    // sort the cost array  
    Arrays.sort(cost);  
    for (int i = 0; i < N; i++)  
    {  
  
        // If the array element is less  
        // then K add it to prefix sum  
        // and check this prefix sum is  
        // less then given K  
        if (cost[i] < K && sum <= K)  
        {  
            sum = sum + cost[i];  
  
            // Increment the  
            // count variable  
            if (sum <= K)  
                count++;  
        }  
    }  
    return count;  
}  
  
// Driver Code  
public static void main (String[] args)  
{  
    int K = 50;  
    int cost[] = {1, 12, 5, 111, 200,  
                 1000, 10, 9, 12, 15};  
    int N = cost.length;  
  
    System.out.print( maximum_toys(cost, N, K));  
}  
}  
  
// This code is contributed by anuj_67.
```

C#

```
// C# Program to maximize the  
// number of toys with K amount
```

```
using System;

class GFG
{
// This functions returns
// the required number of toys
static int maximum_toys(int []cost,
                        int N, int K)
{
    int count = 0, sum = 0;

    // sort the cost array
    Array.Sort(cost);
    for (int i = 0; i < N; i++)
    {

        // If the array element is less
        // then K add it to prefix sum
        // and check this prefix sum is
        // less then given K
        if (cost[i] < K && sum <= K)
        {
            sum = sum + cost[i];

            // Increment the
            // count variable
            if (sum <= K)
                count++;
        }
    }
    return count;
}

// Driver Code
public static void Main ()
{
    int K = 50;
    int []cost = {1, 12, 5, 111, 200,
                  1000, 10, 9, 12, 15};
    int N = cost.Length;

    Console.WriteLine( maximum_toys(cost, N, K));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP Program to maximize the
// number of toys with K amount

// This functions returns
// the required number of toys
function maximum_toys($cost, $N, $K)
{
    $count = 0; $sum = 0;

    // sort the cost array
    sort($cost);
    for ($i = 0; $i < $N; $i++)
    {
        // If the array element is less
        // then K add it to prefix sum
        // and check this prefix sum is
        // less then given K
        if ($cost[$i] < $K and $sum <= $K)
        {
            $sum = $sum + $cost[$i];

            // Increment the
            // count variable
            if ($sum <= $K)
                $count++;
        }
    }
    return $count;
}

// Driver Code
$K = 50;
$cost = array(1, 12, 5, 111, 200,
             1000, 10, 9, 12, 15 );
$N = count($cost);

echo maximum_toys($cost, $N, $K), "\n";

// This code is contributed by anuj_67
?>
```

Output :

Time Complexity : $O(N * \log N)$, where N is the size of cost array.

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/maximise-the-number-of-toys-that-can-be-purchased-with-amount-k/>

Chapter 168

Maximize elements using another array

Maximize elements using another array - GeeksforGeeks

Given two arrays with size n, maximize the first array by using the elements from the second array such that the new array formed contains n greatest but unique elements of both the arrays giving the second array priority (All elements of second array appear before first array). The order of appearance of elements is kept same in output as in input.

Examples:

Input : arr1[] = {2, 4, 3}

arr2[] = {5, 6, 1}

Output : 5 6 4

As 5, 6 and 4 are maximum elements from two arrays giving second array higher priority. Order of elements is same in output as in input.

Input : arr1[] = {7, 4, 8, 0, 1}

arr2[] = {9, 7, 2, 3, 6}

Output : 9 7 6 4 8

Approach : We create an auxiliary array of size 2*n and store the elements of 2nd array in auxiliary array, and then we will store elements of 1st array in it. After that we will sort auxiliary array in decreasing order. To keep the order of elements according to input arrays we will use hash table. We will store 1st n largest unique elements of auxiliary array in hash table. Now we traverse the second array and store that elements of second array in auxiliary array that are present in hash table. Similarly we will traverse first array and store the elements that are present in hash table. In this way we get n unique and largest elements from both the arrays in auxiliary array while keeping the order of appearance of elements same.

Below is the implementation of above approach :

```
// CPP program to print the maximum elements
// giving second array higher priority
#include <bits/stdc++.h>
using namespace std;

// Compare function used to sort array
// in decreasing order
bool compare(int a, int b)
{
    return a > b;
}

// Function to maximize array elements
void maximizeArray(int arr1[], int arr2[],
                   int n)
{
    // auxiliary array arr3 to store
    // elements of arr1 & arr2
    int arr3[2*n], k = 0;
    for (int i = 0; i < n; i++)
        arr3[k++] = arr1[i];
    for (int i = 0; i < n; i++)
        arr3[k++] = arr2[i];

    // hash table to store n largest
    // unique elements
    unordered_set<int> hash;

    // sorting arr3 in decreasing order
    sort(arr3, arr3 + 2 * n, compare);

    // finding n largest unique elements
    // from arr3 and storing in hash
    int i = 0;
    while (hash.size() != n) {

        // if arr3 element not present in hash,
        // then store this element in hash
        if (hash.find(arr3[i]) == hash.end())
            hash.insert(arr3[i]);

        i++;
    }

    // store those elements of arr2 in arr3
    // that are present in hash
    k = 0;
    for (int i = 0; i < n; i++) {
```

```
// if arr2 element is present in hash,
// store it in arr3
if (hash.find(arr2[i]) != hash.end()) {
    arr3[k++] = arr2[i];
    hash.erase(arr2[i]);
}
}

// store those elements of arr1 in arr3
// that are present in hash
for (int i = 0; i < n; i++) {

    // if arr1 element is present in hash,
    // store it in arr3
    if (hash.find(arr1[i]) != hash.end()) {
        arr3[k++] = arr1[i];
        hash.erase(arr1[i]);
    }
}

// copying 1st n elements of arr3 to arr1
for (int i = 0; i < n; i++)
    arr1[i] = arr3[i];
}

// Function to print array elements
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver Code
int main()
{
    int array1[] = { 7, 4, 8, 0, 1 };
    int array2[] = { 9, 7, 2, 3, 6 };
    int size = sizeof(array1) / sizeof(array1[0]);
    maximizeArray(array1, array2, size);
    printArray(array1, size);
}
```

Output:

9 7 6 4 8

Time complexity: $O(n * \log n)$.

Source

<https://www.geeksforgeeks.org/maximize-elements-using-another-array/>

Chapter 169

Maximize sum of consecutive differences in a circular array

Maximize sum of consecutive differences in a circular array - GeeksforGeeks

Given an array of n elements. Consider array as circular array i.e element after a_n is a_1 . The task is to find maximum sum of the difference between consecutive elements with rearrangement of array element allowed i.e after rearrangement of element find $a_1 - a_2 + a_2 - a_3 + \dots + a_{n-1} - a_n + a_n - a_1$.

Examples:

```
Input : arr[] = { 4, 2, 1, 8 }
Output : 18
Rearrange given array as : { 1, 8, 2, 4 }
Sum of difference between consecutive element
= |1 - 8| + |8 - 2| + |2 - 4| + |4 - 1|
= 7 + 6 + 2 + 3
= 18.
```

```
Input : arr[] = { 10, 12, 15 }
Output : 10
```

The idea is to use Greedy Approach and try to bring elements having greater difference closer.

Consider the sorted permutation of the given array $a_1, a_1, a_2, \dots, a_{n-1}, a_n$ such that $a_1 < a_2 < a_3 \dots < a_{n-1} < a_n$.

Now, to obtain the answer having maximum sum of difference between consecutive element, arrange element in following manner:

$a_1, a_n, a_2, a_{n-1}, \dots, a_{n/2}, a_{(n/2)+1}$

We can observe that the arrangement produces the optimal answer, as all $a_1, a_2, a_3, \dots, a_{(n/2)-1}, a_{n/2}$ are subtracted twice while $a_{(n/2)+1}, a_{(n/2)+2}, a_{(n/2)+3}, \dots, a_{n-1}, a_n$ are added

twice.

C++

```
// C++ program to maximize the sum of difference
// between consecutive elements in circular array
#include <bits/stdc++.h>
using namespace std;

// Return the maximum Sum of difference between
// consecutive elements.
int maxSum(int arr[], int n)
{
    int sum = 0;

    // Sorting the array.
    sort(arr, arr + n);

    // Subtracting a1, a2, a3,....., a(n/2)-1, an/2
    // twice and adding a(n/2)+1, a(n/2)+2, a(n/2)+3, .
    // ...., an - 1, an twice.
    for (int i = 0; i < n/2; i++)
    {
        sum -= (2 * arr[i]);
        sum += (2 * arr[n - i - 1]);
    }

    return sum;
}

// Driver Program
int main()
{
    int arr[] = { 4, 2, 1, 8 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << maxSum(arr, n) << endl;
    return 0;
}
```

Java

```
// Java program to maximize the sum of difference
// between consecutive elements in circular array
import java.io.*;
import java.util.Arrays;

class MaxSum
{
```

```
// Return the maximum Sum of difference between
// consecutive elements.
static int maxSum(int arr[], int n)
{
    int sum = 0;

    // Sorting the array.
    Arrays.sort(arr);

    // Subtracting a1, a2, a3,....., a(n/2)-1,
    // an/2 twice and adding a(n/2)+1, a(n/2)+2,
    // a(n/2)+3,....., an - 1, an twice.
    for (int i = 0; i < n/2; i++)
    {
        sum -= (2 * arr[i]);
        sum += (2 * arr[n - i - 1]);
    }

    return sum;
}

// Driver Program
public static void main (String[] args)
{
    int arr[] = { 4, 2, 1, 8 };
    int n = arr.length;
    System.out.println(maxSum(arr, n));
}
}
/*This code is contributed by Prakriti Gupta*/
```

Python3

```
# Python3 program to maximize the sum of difference
# between consecutive elements in circular array

# Return the maximum Sum of difference
# between consecutive elements
def maxSum(arr, n):
    sum = 0

    # Sorting the array
    arr.sort()

    # Subtracting a1, a2, a3,....., a(n/2)-1, an/2
    # twice and adding a(n/2)+1, a(n/2)+2, a(n/2)+3,.
    # ...., an - 1, an twice.
    for i in range(0, int(n / 2)) :
```

```
sum -= (2 * arr[i])
sum += (2 * arr[n - i - 1])

return sum

# Driver Program
arr = [4, 2, 1, 8]
n = len(arr)
print (maxSum(arr, n))

# This code is contributed by Shreyanshi Arun.
```

C#

```
// C# program to maximize the sum of difference
// between consecutive elements in circular array
using System;

class MaxSum {

    // Return the maximum Sum of difference
    // between consecutive elements.
    static int maxSum(int[] arr, int n)
    {
        int sum = 0;

        // Sorting the array.
        Array.Sort(arr);

        // Subtracting a1, a2, a3, ...., a(n/2)-1,
        // an/2 twice and adding a(n/2)+1, a(n/2)+2,
        // a(n/2)+3, ...., an - 1, an twice.
        for (int i = 0; i < n / 2; i++) {
            sum -= (2 * arr[i]);
            sum += (2 * arr[n - i - 1]);
        }

        return sum;
    }

    // Driver Program
    public static void Main()
    {
        int[] arr = { 4, 2, 1, 8 };
        int n = arr.Length;
        Console.WriteLine(maxSum(arr, n));
    }
}
```

```
}
```

```
//This Code is contributed by vt_m.
```

Output :

18

Time Complexity: $O(n \log n)$.

Auxiliary Space : $O(1)$

Source

<https://www.geeksforgeeks.org/maximize-sum-consecutive-differences-circular-array/>

Chapter 170

Maximize the profit by selling at-most M products

Maximize the profit by selling at-most M products - GeeksforGeeks

Given two lists that contains cost prices $CP[]$ and selling prices $SP[]$ of products respectively. The task is to maximize the profit by selling at-most ‘M’ products.

Examples:

Input: N = 5, M = 3

$CP[] = \{5, 10, 35, 7, 23\}$

$SP[] = \{11, 10, 0, 9, 19\}$

Output: 8

Profit on 0th product i.e. $11-5 = 6$

Profit on 3rd product i.e. $9-7 = 2$

Selling any other product will not give profit.

So, total profit = $6+2 = 8$.

Input: N = 4, M = 2

$CP[] = \{17, 9, 8, 20\}$

$SP[] = \{10, 9, 8, 27\}$

Output: 7

Approach:

1. Store the profit/loss on buying and selling of each product i.e. $SP[i]-CP[i]$ in an array.
2. Sort that array in descending order.
3. Add the positive values up to M values as positive values denote profit.
4. Return Sum.

Below is the implementation of above approach:

C++

```
// C++ implementation of above approach:  
#include <bits/stdc++.h>  
using namespace std;  
  
// Function to find profit  
int solve(int N, int M, int cp[], int sp[])  
{  
    int profit[N];  
  
    // Calculating profit for each gadget  
    for (int i = 0; i < N; i++)  
        profit[i] = sp[i] - cp[i];  
  
    // sort the profit array in decending order  
    sort(profit, profit + N, greater<int>());  
  
    // variable to calculate total profit  
    int sum = 0;  
  
    // check for best M profits  
    for (int i = 0; i < M; i++) {  
        if (profit[i] > 0)  
            sum += profit[i];  
        else  
            break;  
    }  
  
    return sum;  
}  
  
// Driver Code  
int main()  
{  
  
    int N = 5, M = 3;  
    int CP[] = { 5, 10, 35, 7, 23 };  
    int SP[] = { 11, 10, 0, 9, 19 };  
  
    cout << solve(N, M, CP, SP);  
  
    return 0;  
}
```

Java

```
// Java implementation of above approach:  
import java.util.*;  
import java.lang.*;  
import java.io.*;  
  
class GFG  
{  
  
    // Function to find profit  
    static int solve(int N, int M,  
                    int cp[], int sp[])  
    {  
        Integer []profit = new Integer[N];  
  
        // Calculating profit for each gadget  
        for (int i = 0; i < N; i++)  
            profit[i] = sp[i] - cp[i];  
  
        // sort the profit array  
        // in decending order  
        Arrays.sort(profit, Collections.reverseOrder());  
  
        // variable to calculate total profit  
        int sum = 0;  
  
        // check for best M profits  
        for (int i = 0; i < M; i++)  
        {  
            if (profit[i] > 0)  
                sum += profit[i];  
            else  
                break;  
        }  
  
        return sum;  
    }  
  
    // Driver Code  
    public static void main(String args[])  
    {  
        int N = 5, M = 3;  
        int CP[] = { 5, 10, 35, 7, 23 };  
        int SP[] = { 11, 10, 0, 9, 19 };  
  
        System.out.println(solve(N, M, CP, SP));  
    }  
}
```

```
// This code is contributed  
// by Subhadeep Gupta
```

Python3

```
# Python3 implementation  
# of above approach  
  
# Function to find profit  
def solve(N, M, cp, sp) :  
  
    # take empty list  
    profit = []  
  
    # Calculating profit  
    # for each gadget  
    for i in range(N) :  
        profit.append(sp[i] - cp[i])  
  
    # sort the profit array  
    # in decending order  
    profit.sort(reverse = True)  
  
    sum = 0  
  
    # check for best M profits  
    for i in range(M) :  
        if profit[i] > 0 :  
            sum += profit[i]  
        else :  
            break  
  
    return sum  
  
# Driver Code  
if __name__ == "__main__" :  
  
    N, M = 5, 3  
    CP = [5, 10, 35, 7, 23]  
    SP = [11, 10, 0, 9, 19]  
  
    # function calling  
    print(solve(N, M, CP, SP))  
  
# This code is contributed  
# by ANKITRAI1
```

C#

```
// C# implementation of above approach:  
using System;  
  
class GFG  
{  
  
    // Function to find profit  
    static int solve(int N, int M,  
    int[] cp, int[] sp)  
    {  
        int[] profit = new int[N];  
  
        // Calculating profit for each gadget  
        for (int i = 0; i < N; i++) profit[i] = sp[i] - cp[i]; // sort the profit array // in descending  
        order Array.Sort(profit); Array.Reverse(profit); // variable to calculate total profit int sum  
        = 0; // check for best M profits for (int i = 0; i < M; i++) { if (profit[i] > 0)  
        sum += profit[i];  
        else  
        break;  
    }  
  
    return sum;  
}  
  
// Driver Code  
public static void Main()  
{  
    int N = 5, M = 3;  
    int[] CP = { 5, 10, 35, 7, 23 };  
    int[] SP = { 11, 10, 0, 9, 19 };  
  
    Console.WriteLine(solve(N, M, CP, SP));  
}  
}  
  
// This code is contributed  
// by ChitraNayal
```

PHP

```
0)  
$sum += $profit[$i];  
else  
break;  
}  
  
return $sum;  
}  
  
// Driver Code  
$N = 5;  
$M = 3;  
$CP = array( 5, 10, 35, 7, 23 );  
$SP = array( 11, 10, 0, 9, 19 );
```

```
echo solve($N, $M, $CP, $SP);  
// This code is contributed  
// by ChitraNayal  
?>
```

Output:

8

Improved By : [ANKITRAI1](#), [tufan_gupta2000](#), [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/maximize-the-profit-by-selling-at-most-m-products/>

Chapter 171

Maximize the sum of arr[i]*i

Maximize the sum of arr[i]*i - GeeksforGeeks

Given an array of N integers. You are allowed to rearrange the element of the array. The task is to find the maximum value of $\sum arr[i]*i$, where $i = 0, 1, 2, \dots, n-1$.

Examples:

```
Input : N = 4, arr[] = { 3, 5, 6, 1 }
Output : 31
If we arrange arr[] as { 1, 3, 5, 6 }.
Sum of arr[i]*i is 1*0 + 3*1 + 5*2 + 6*3
= 31, which is maximum
```

```
Input : N = 2, arr[] = { 19, 20 }
Output : 20
```

A **simple solution** is to [generate all permutations of given array](#). For every permutation, compute the value of $\sum arr[i]*i$ and finally return the maximum value.

An **efficient solution** is based on the fact that the largest value should be scaled maximum and smallest value should be scaled minimum. So we multiply minimum value of i with minimum value of $arr[i]$. So, sort the given array in increasing order and compute the sum of $arr[i]*i$, where $i = 0$ to $n-1$.

Below is the implementation of this approach:

C++

```
// CPP program to find the maximum value
// of i*arr[i]
#include<bits/stdc++.h>
using namespace std;
```

```
int maxSum(int arr[], int n)
{
    // Sort the array
    sort(arr, arr + n);

    // Finding the sum of arr[i]*i
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += (arr[i]*i);

    return sum;
}

// Driven Program
int main()
{
    int arr[] = { 3, 5, 6, 1 };
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << maxSum(arr, n) << endl;
    return 0;
}
```

Java

```
// Java program to find the
// maximum value of i*arr[i]
import java.util.*;

class GFG {

    static int maxSum(int arr[], int n)
    {
        // Sort the array
        Arrays.sort(arr);

        // Finding the sum of arr[i]*i
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum += (arr[i] * i);

        return sum;
    }

    // Driven Program
    public static void main(String[] args)
    {
        int arr[] = { 3, 5, 6, 1 };
```

```
int n = arr.length;

System.out.println(maxSum(arr, n));

}

// This code is contributed by Prerna Saini
```

Python3

```
# Python program to find the
# maximum value of i*arr[i]
def maxSum(arr,n):

    # Sort the array
    arr.sort()

    # Finding the sum of
    # arr[i]*i
    sum = 0
    for i in range(n):
        sum += arr[i] * i

    return sum

# Driver Program
arr = [3,5,6,1]
n = len(arr)
print(maxSum(arr,n))

# This code is contributed
# by Shrikant13
```

C#

```
// C# program to find the
// maximum value of i*arr[i]
using System;

class GFG {

    // Function to find the
    // maximum value of i*arr[i]
    static int maxSum(int[] arr, int n)
    {

        // Sort the array
```

```
        Array.Sort(arr);

        // Finding the sum of arr[i]*i
        int sum = 0;
        for (int i = 0; i < n; i++)
            sum += (arr[i] * i);

        return sum;
    }

    // Driver code
    static public void Main()
    {
        int[] arr = {3, 5, 6, 1};
        int n = arr.Length;

        Console.WriteLine(maxSum(arr, n));
    }
}

// This code is contributed by Ajit.
```

PHP

```
<?php
// PHP program to find the
// maximum value of i*arr[i]

// function returns the
// maximum value of i*arr[i]
function maxSum($arr, $n)
{
    // Sort the array
    sort($arr);

    // Finding the sum
    // of arr[i]*i
    $sum = 0;
    for ($i = 0; $i < $n; $i++)
        $sum += ($arr[$i] * $i);

    return $sum;
}

// Driver Code
$arr = array( 3, 5, 6, 1 );
$n = count($arr);
```

```
echo maxSum($arr, $n);  
// This code is contributed by anuj_67.  
?>
```

Output:

31

Time Complexity : O(n Log n)

Improved By : [shrikanth13](#), [jit_t](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/maximize-sum-arrii/>

Chapter 172

Maximizing Unique Pairs from two arrays

Maximizing Unique Pairs from two arrays - GeeksforGeeks

Given two arrays of equal size N, form maximum number of pairs by using their elements, one from the first array and second from the second array, such that an element from each array is used at-most once and the absolute difference between the selected elements used for forming a pair is less than or equal to a given element K.

Examples:

```
Input : a[] = {3, 4, 5, 2, 1}
        b[] = {6, 5, 4, 7, 15}
        k = 3
Output : 4
The maximum number of pairs that can be formed
using the above 2 arrays is 4 and the corresponding
pairs are [1, 4], [2, 5], [3, 6], [4, 7], we can't
pair the remaining elements.
Other way of pairing under given constraint is
[2, 5], [3, 6], [4, 4], but count of pairs here
is 3 which is less than the result 4.
```

Simple Approach: By taking few examples, we can observe that if we sort both array. Then one by pick closest feasible element for every element, we get the optimal answer.

In this approach we first sort both the arrays and then compare each element of the first array with each element of the second array for the possible pair, if it's possible to form a pair, we form the pair and move to check for the next possible pair for the next element of the first array.

```
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

// Returns count of maximum pairs that can
// be formed from a[] and b[] under given
// constraints.
ll findMaxPairs(ll a[], ll b[], ll n, ll k)
{
    sort(a, a+n); // Sorting the first array.
    sort(b, b+n); // Sorting the second array.

    // To keep track of visited elements of b[]
    bool flag[n];
    memset(flag, false, sizeof(flag));

    // For every element of a[], find a pair
    // for it and break as soon as a pair is
    // found.
    int result = 0;
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j++)
        {
            if (abs(a[i]-b[j])<=k && flag[j]==false)
            {
                // Increasing the count if a pair is formed.
                result++;

                /* Making the corresponding flag array
                   element as 1 indicating the element
                   in the second array element has
                   been used. */
                flag[j] = true;

                // We break the loop to make sure an
                // element of a[] is used only once.
                break;
            }
        }
    }
    return result;
}

// Driver code
int main()
{
    ll a[] = {10, 15, 20}, b[] = {17, 12, 24};
```

```
int n = sizeof(a)/sizeof(a[0]);
int k = 3;
cout << findMaxPairs(a, b, n, k);
return 0;
}
```

Output:

2

Time complexity : $O(n^2)$

Auxiliary Space : $O(n)$

Efficient Approach: In this approach, rather than checking all the possible combination of pairs, we optimize our code by checking only the feasible combination of pairs using the 2 pointer approach.

```
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

// Returns count of maximum pairs that can
// be formed from a[] and b[] under given
// constraints.
ll findMaxPairs(ll a[], ll b[], ll n, ll k)
{
    sort(a, a+n); // Sorting the first array.
    sort(b, b+n); // Sorting the second array.

    int result = 0;
    for (int i=0, j=0; i<n && j<n;)
    {
        if (abs(a[i] - b[j]) <= k)
        {
            result++;

            // Increasing array pointer of
            // both the first and the second array.
            i++;
            j++;
        }

        // Increasing array pointer of the second array.
        else if(a[i] > b[j])
            j++;

        // Increasing array pointer of the first array.
    }
}
```

```
        else
            i++;
    }
    return result;
}

// Driver code
int main()
{
    ll a[] = {10, 15, 20};
    ll b[] = {17, 12, 24};
    int n = sizeof(a)/sizeof(a[0]);
    int k = 3;
    cout << findMaxPairs(a, b, n, k);
    return 0;
}
```

Output:

2

Time complexity : $O(n \log n)$

Auxiliary Space : $O(1)$

Source

<https://www.geeksforgeeks.org/maximizing-unique-pairs-two-arrays/>

Chapter 173

Maximum area rectangle by picking four sides from array

Maximum area rectangle by picking four sides from array - GeeksforGeeks

Given an array of n positive integers that represent lengths. Find out the maximum possible area whose four sides are picked from given array. Note that a rectangle can only be formed if there are two pairs of equal values in given array.

Examples:

Input : arr[] = {2, 1, 2, 5, 4, 4}
Output : 8
Explanation : Dimension will be 4 * 2

Input : arr[] = {2, 1, 3, 5, 4, 4}
Output : 0
Explanation : No rectangle possible

Method 1 (Sorting)

The task basically reduces to finding two pairs of equal values in array. If there are more than two pairs, then pick the two pairs with maximum values. A simple solution is to do following.

- 1) Sort the given array.
- 2) Traverse array from largest to smallest value and return two pairs with maximum values.

C++

```
// CPP program for finding maximum area possible
// of a rectangle
#include <bits/stdc++.h>
using namespace std;
```

```
// function for finding max area
int findArea(int arr[], int n)
{
    // sort array in non-increasing order
    sort(arr, arr + n, greater<int>());

    // Initialize two sides of rectangle
    int dimension[2] = { 0, 0 };

    // traverse through array
    for (int i = 0, j = 0; i < n - 1 && j < 2; i++)

        // if any element occurs twice
        // store that as dimension
        if (arr[i] == arr[i + 1])
            dimension[j++] = arr[i++];

    // return the product of dimensions
    return (dimension[0] * dimension[1]);
}

// driver function
int main()
{
    int arr[] = { 4, 2, 1, 4, 6, 6, 2, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findArea(arr, n);
    return 0;
}
```

Java

```
// Java program for finding maximum area
// possible of a rectangle
import java.util.Arrays;
import java.util.Collections;

public class GFG
{
    // function for finding max area
    static int findArea(Integer arr[], int n)
    {
        // sort array in non-increasing order
        Arrays.sort(arr, Collections.reverseOrder());

        // Initialize two sides of rectangle
        int[] dimension = { 0, 0 };
```

```
// traverse through array
for (int i = 0, j = 0; i < n - 1 && j < 2;
     i++)
{
    // if any element occurs twice
    // store that as dimension
    if (arr[i] == arr[i + 1])
        dimension[j++] = arr[i++];

    // return the product of dimensions
    return (dimension[0] * dimension[1]);
}

// driver function
public static void main(String args[])
{
    Integer arr[] = { 4, 2, 1, 4, 6, 6, 2, 5 };
    int n = arr.length;
    System.out.println(findArea(arr, n));
}
}

// This code is contributed by Sumit Ghosh
```

Python3

```
# Python3 program for finding
# maximum area possible of
# a rectangle

# function for finding
# max area
def findArea(arr, n):

    # sort array in
    # non-increasing order
    arr.sort(reverse = True)

    # Initialize two
    # sides of rectangle
    dimension = [0, 0]

    # traverse through array
    i = 0
    j = 0
    while(i < n - 1 and j < 2):

        # if any element occurs twice
```

```
# store that as dimension
if (arr[i] == arr[i + 1]):
    dimension[j] = arr[i]
    j += 1
    i += 1
i += 1

# return the product
# of dimensions
return (dimension[0] *
        dimension[1])

# Driver code
arr = [4, 2, 1, 4, 6, 6, 2, 5]
n = len(arr)
print(findArea(arr, n))

# This code is contributed
# by Smitha
```

PHP

```
<?php
// PHP program for finding maximum area possible
// of a rectangle

// function for finding max area
function findArea($arr, $n)
{

    // sort array in non-
    // increasing order
    rsort($arr);

    // Initialize two sides
    // of rectangle
    $dimension = array( 0, 0 );

    // traverse through array
    for( $i = 0, $j = 0; $i < $n - 1 &&
         $j < 2; $i++)

        // if any element occurs twice
        // store that as dimension
        if ($arr[$i] == $arr[$i + 1])
            $dimension[$j++] = $arr[$i++];

    // return the product
```

```
// of dimensions
return ($dimension[0] *
        $dimension[1]);
}

// Driver Code
$arr = array(4, 2, 1, 4, 6, 6, 2, 5);
$n = count($arr);
echo findArea($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

24

Time Complexity : $O(n \log n)$

Method 2 (Hashing)

The idea is to insert all first occurrences of elements in a hash set. For second occurrences, keep track of maximum two values.

C++

```
// CPP program for finding maximum area possible
// of a rectangle
#include <bits/stdc++.h>
using namespace std;

// function for finding max area
int findArea(int arr[], int n)
{
    unordered_set<int> s;

    // traverse through array
    int first = 0, second = 0;
    for (int i = 0; i < n; i++) {

        // If this is first occurrence of arr[i],
        // simply insert and continue
        if (s.find(arr[i]) == s.end()) {
            s.insert(arr[i]);
            continue;
        }
    }
```

```
// If this is second (or more) occurrence,
// update first and second maximum values.
if (arr[i] > first) {
    second = first;
    first = arr[i];
} else if (arr[i] > second)
    second = arr[i];
}

// return the product of dimensions
return (first * second);
}

// driver function
int main()
{
    int arr[] = { 4, 2, 1, 4, 6, 6, 2, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findArea(arr, n);
    return 0;
}
```

Java

```
// Java program for finding maximum
// area possible of a rectangle
import java.util.HashSet;
import java.util.Set;

public class GFG
{
    // function for finding max area
    static int findArea(int arr[], int n)
    {
        //unordered_set<int> s;

        Set<Integer> s = new HashSet<>();

        // traverse through array
        int first = 0, second = 0;
        for (int i = 0; i < n; i++) {

            // If this is first occurrence of
            // arr[i], simply insert and continue
            if (!s.contains(arr[i])) {
                s.add(arr[i]);
                continue;
            }
        }
    }
}
```

```
// If this is second (or more)
// occurrence, update first and
// second maximum values.
if (arr[i] > first) {
    second = first;
    first = arr[i];
} else if (arr[i] > second)
    second = arr[i];
}

// return the product of dimensions
return (first * second);
}

// driver function
public static void main(String args[])
{
    int arr[] = { 4, 2, 1, 4, 6, 6, 2, 5 };
    int n = arr.length;
    System.out.println(findArea(arr, n));
}
}

// This code is contributed by Sumit Ghosh
```

Output:

24

Time Complexity : O(n)

Improved By : [vt_m](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/maximum-area-rectangle-picking-four-sides-array/>

Chapter 174

Maximum array from two given arrays keeping order same

Maximum array from two given arrays keeping order same - GeeksforGeeks

Given two same sized arrays A[] and B[] (both arrays contain distinct elements individually but may have some common elements), task is to form a third (or result) array of same size. The result array should have maximum n elements from both array. It should have chosen elements of A[] first, then chosen elements of B[] in same order as they appear in original arrays. If there are common elements, then only one element should be present in res[] and priority should be given to A[].

Examples:

```
Input : A[] = [ 9 7 2 3 6 ]
        B[] = [ 7 4 8 0 1 ]
Output : res[] = [9 7 6 4 8]
res[] has maximum n elements of both A[]
and B[] such that elements of A[] appear
first (in same order), then elements of B[] .
Also 7 is common and priority is given to
A's 7.

Input : A[] = [ 6 7 5 3 ]
        B[] = [ 5 6 2 9 ]
Output : res[] = [ 6 7 5 9 ]
```

- 1) Create copies of both arrays and sort the copies in decreasing order.
- 2) Use a hash to pick unique n maximum elements of both arrays, giving priority to A[].
- 3) Initialize result array as empty.
- 4) Traverse through A[], copy those elements of A[] that are present in the hash. This is done to keep order of elements same.

5) Repeat step 4 for B[]. This time we only consider those elements that are not present in A[] (Do not appear twice in hash).

Below c++ implementation of above idea.

```
// Make a set of maximum elements from two
// arrays A[] and B[]
#include <bits/stdc++.h>
using namespace std;

void maximizeTheFirstArray(int A[], int B[],
                           int n)
{
    // Create copies of A[] and B[] and sort
    // the copies in descending order.
    vector<int> temp1(A, A+n);
    vector<int> temp2(B, B+n);
    sort(temp1.begin(), temp1.end(), greater<int>());
    sort(temp2.begin(), temp2.end(), greater<int>());

    // Put maximum n distinct elements of
    // both sorted arrays in a map.
    unordered_map<int, int> m;
    int i = 0, j = 0;
    while (m.size() < n)
    {
        if (temp1[i] >= temp2[j])
        {
            m[temp1[i]]++;
            i++;
        }
        else
        {
            m[temp2[j]]++;
            j++;
        }
    }

    // Copy elements of A[] to that
    // are present in hash m.
    vector<int> res;
    for (int i = 0; i < n; i++)
        if (m.find(A[i]) != m.end())
            res.push_back(A[i]);

    // Copy elements of B[] to that
    // are present in hash m. This time
    // we also check if the element did
    // not appear twice.
```

```
for (int i = 0; i < n; i++)
    if (m.find(B[i]) != m.end() &&
        m[B[i]] == 1)
        res.push_back(B[i]);

// print result
for (int i = 0; i < n; i++)
    cout << res[i] << " ";
}

// driver program
int main()
{
    int A[] = { 9, 7, 2, 3, 6 };
    int B[] = { 7, 4, 8, 0, 1 };
    int n = sizeof(A) / sizeof(A[0]);
    maximizeTheFirstArray(A, B, n);
    return 0;
}
```

Output:

9 7 6 4 8

Time complexity: $O(n \log n)$

Source

<https://www.geeksforgeeks.org/maximum-array-from-two-given-arrays-keeping-order-same/>

Chapter 175

Maximum difference between frequency of two elements such that element having greater frequency is also greater

Maximum difference between frequency of two elements such that element having greater frequency is also greater - GeeksforGeeks

Given an array of n positive integers with many repeating elements. The task is to find maximum difference between the frequency of any two different elements, such that the element with greater frequency is also greater in value than the second integer.

Examples:

```
Input : arr[] = { 3, 1, 3, 2, 3, 2 }.
Output : 2
Frequency of 3 = 3.
Frequency of 2 = 2.
Frequency of 1 = 1.
Here difference of frequency of element 3 and 1 is = 3 - 1 = 2.
Also 3 > 1.
```

Method 1 (Use Hashing):

The naive approach can be, find the frequency of each element and for each element find the element having lesser value and lesser frequency than the current element.

Below is C++ implementation of this approach:

```
// C++ program to find maximum difference
```

```
// between frequency of any two element
// such that element with greater frequency
// is also greater in value.
#include<bits/stdc++.h>
using namespace std;

// Return the maximum difference between
// frequencies of any two elements such that
// element with greater frequency is also
// greater in value.
int maxdiff(int arr[], int n)
{
    unordered_map<int, int> freq;

    // Finding the frequency of each element.
    for (int i = 0; i < n; i++)
        freq[arr[i]]++;

    int ans = 0;
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j++)
        {
            // finding difference such that element
            // having greater frequency is also
            // greater in value.
            if (freq[arr[i]] > freq[arr[j]] &&
                arr[i] > arr[j] )
                ans = max(ans, freq[arr[i]]-freq[arr[j]]);
            else if (freq[arr[i]] < freq[arr[j]] &&
                      arr[i] < arr[j] )
                ans = max(ans, freq[arr[j]]-freq[arr[i]]);
        }
    }

    return ans;
}

// Driven Program
int main()
{
    int arr[] = { 3, 1, 3, 2, 3, 2 };
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << maxdiff(arr, n) << endl;
    return 0;
}
```

Output:

2

Time Complexity: $O(n^2)$.

Method 2 (Use Hashing and Sorting):

The idea is to find all the distinct elements and store in an array, say dist[]. Sort the distinct element array dist[] in increasing order. Now for any distinct element at index i, for all index j such that $i > j > 0$, find the element between index 0 to i-1 having minimum frequency. We can find frequency of an element in same way as method 1, i.e., storing frequencies in a hash table.

So do this for all i and find the maximum difference. To find the minimum frequency for all i maintain a prefix minimum.

Below is C++ representation of this approach:

```
// Efficient C++ program to find maximum
// difference between frequency of any two
// elements such that element with greater
// frequency is also greater in value.
#include<bits/stdc++.h>
using namespace std;

// Return the maximum difference between
// frequencies of any two elements such that
// element with greater frequency is also
// greater in value.
int maxdiff(int arr[], int n)
{
    unordered_map<int, int> freq;

    int dist[n];

    // Finding the frequency of each element.
    int j = 0;
    for (int i = 0; i < n; i++)
    {
        if (freq.find(arr[i]) == freq.end())
            dist[j++] = arr[i];

        freq[arr[i]]++;
    }

    // Sorting the distinct element
    sort(dist, dist + j);
```

```
int min_freq = n+1;

// Iterate through all sorted distinct elements.
// For each distinct element, maintaining the
// element with minimum frequency than that
// element and also finding the maximum
// frequency difference
int ans = 0;
for (int i=0; i<j; i++)
{
    int cur_freq = freq[dist[i]];
    ans = max(ans, cur_freq - min_freq);
    min_freq = min(min_freq, cur_freq);
}

return ans;
}

// Driven Program
int main()
{
    int arr[] = { 3, 1, 3, 2, 3, 2 };
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << maxdiff(arr, n) << endl;
    return 0;
}
```

Output:

2

Time Complexity : $O(n \log n)$.

Source

<https://www.geeksforgeeks.org/maximum-difference-between-frequency-of-two-elements-such-that-element-having>

Chapter 176

Maximum difference between groups of size two

Maximum difference between groups of size two - GeeksforGeeks

Given an array of even number of elements, form groups of 2 using these array elements such that the difference between the group with highest sum and the one with lowest sum is maximum.

Note: An element can be a part of one group only and it has to be a part of at least 1 group.

Examples:

```
Input : arr[] = {1, 4, 9, 6}
Output : 10
Groups formed will be (1, 4) and (6, 9),
the difference between highest sum group
(6, 9) i.e 15 and lowest sum group (1, 4)
i.e 5 is 10.
```

```
Input : arr[] = {6, 7, 1, 11}
Output : 11
Groups formed will be (1, 6) and (7, 11),
the difference between highest sum group
(7, 11) i.e 18 and lowest sum group (1, 6)
i.e 7 is 11.
```

Simple Approach: We can solve this problem by making all possible combinations and checking each set of combination difference between the group with highest sum and with the lowest sum. A total of $n*(n-1)/2$ such groups would be formed ($nC2$).

Time Complexity: $O(n^3)$, because it will take $O(n^2)$ to generate groups and to check against each group n iterations will be needed thus overall it takes $O(n^3)$ time.

Efficient Approach: We can use the greedy approach. Sort the whole array and our result is sum of last two elements minus sum of first two elements.

C++

```
// CPP program to find minimum difference
// between groups of highest and lowest
// sums.
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

ll CalculateMax(ll arr[], int n)
{
    // Sorting the whole array.
    sort(arr, arr + n);

    int min_sum = arr[0] + arr[1];
    int max_sum = arr[n-1] + arr[n-2];

    return abs(max_sum - min_sum);
}

// Driver code
int main()
{
    ll arr[] = { 6, 7, 1, 11 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << CalculateMax(arr, n) << endl;
    return 0;
}
```

PHP

```
<?php
// PHP program to find minimum
// difference between groups of
// highest and lowest sums.
function CalculateMax($arr, $n)
{
    // Sorting the whole array.
    sort($arr);

    $min_sum = $arr[0] +
               $arr[1];
```

```
$max_sum = $arr[$n - 1] +  
$arr[$n - 2];  
  
return abs($max_sum -  
$min_sum);  
}  
  
// Driver code  
$arr = array (6, 7, 1, 11 );  
$n = sizeof($arr);  
echo CalculateMax($arr, $n), "\n" ;  
  
// This code is contributed by ajit  
?>
```

Output:

11

Time Complexity: O (n * log n)

Further Optimization :

Instead of sorting, we can find maximum two and minimum two in linear time and reduce time complexity to O(n).

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/maximum-difference-groups-size-two/>

Chapter 177

Maximum in an array that can make another array sorted

Maximum in an array that can make another array sorted - GeeksforGeeks

Given two arrays among which one is almost sorted with one element being in the wrong position making the array unsorted, the task is to swap that element with the maximum element from the second array which can be used to make the first array sorted.

Examples:

Input: arr1 = {1, 3, 7, 4, 10},
arr2 = {2, 1, 5, 8, 9}
Output: 1 3 7 9 10
Swap 4 with 9.
Input: arr1 = {20, 1, 23},
arr2 = {50, 26, 7}
Output: Not Possible

Approach:

1. Get the index of the element which is making the array unsorted.
2. Get the maximum element from the second array satisfying the neighboring conditions of the element with wrong index i.e
 - (i) max element \geq arr[wrong index-1]
 - (ii)max element \leq arr[wrong index+1] if wrong index+1 exists

```
// C++ program to make array sorted
#include <bits/stdc++.h>
using namespace std;

// Function to check whether there is any
// swappable element present to make the first
```

```
// array sorted
bool swapElement(int arr1[], int arr2[], int n)
{
    // wrongIdx is the index of the element
    // which is making the first array unsorted
    int wrongIdx = 0;
    for (int i = 1; i < n; i++) {
        if (arr1[i] < arr1[i - 1])
            wrongIdx = i;

    int maximum = INT_MIN;
    int maxIdx = -1;
    bool res = false;

    // Find the maximum element which satisfies the
    // the above mentioned neighboring conditions
    for (int i = 0; i < n; i++) {
        if (arr2[i] > maximum && arr2[i] >= arr1[wrongIdx - 1]) {
            if (wrongIdx + 1 <= n - 1 &&
                arr2[i] <= arr1[wrongIdx + 1]) {
                maximum = arr2[i];
                maxIdx = i;
                res = true;
            }
        }
    }

    // if res is true then swap the element
    // and make the first array sorted
    if (res)
        swap(arr1[wrongIdx], arr2[maxIdx]);

    return res;
}

// Function to print the sorted array if elements
// are swapped.
void getSortedArray(int arr1[], int arr2[], int n)
{
    if (swapElement(arr1, arr2, n))
        for (int i = 0; i < n; i++)
            cout << arr1[i] << " ";
    else
        cout << "Not Possible" << endl;
}

// Drivers code
```

```
int main()
{
    int arr1[] = { 1, 3, 7, 4, 10 };
    int arr2[] = { 2, 1, 6, 8, 9 };

    int n = sizeof(arr1) / sizeof(arr1[0]);
    getSortedArray(arr1, arr2, n);
}
```

Output:

1 3 7 9 10

Source

<https://www.geeksforgeeks.org/maximum-in-an-array-that-can-make-another-array-sorted/>

Chapter 178

Maximum number of partitions that can be sorted individually to make sorted

Maximum number of partitions that can be sorted individually to make sorted - Geeks-forGeeks

Given an array arr[] of size n such that elements of arr[] in range [0, 1, ..n-1]. Our task is to divide the array into maximum number of partitions that can be sorted individually, then concatenated to make the whole array sorted.

Examples :

Input : arr[] = [2, 1, 0, 3]
Output : 2
If divide arr[] into two partitions
{2, 1, 0} and {3}, sort them and concatenate
them, we get the whole array sorted.

Input : arr[] = [2, 1, 0, 3, 4, 5]
Output : 4
The maximum number of partitions are four, we
get these partitions as {2, 1, 0}, {3}, {4}
and {5}

The idea is based on the fact that if an element arr[i] is maximum of prefix arr[0..i], then we can make a partition ending with arr[i].

C++

```
// CPP program to find Maximum number of partitions
```

```
// such that we can get a sorted array.
#include <bits/stdc++.h>
using namespace std;

// Function to find maximum partitions.
int maxPartitions(int arr[], int n)
{
    int ans = 0, max_so_far = 0;
    for (int i = 0; i < n; ++i) {

        // Find maximum in prefix arr[0..i]
        max_so_far = max(max_so_far, arr[i]);

        // If maximum so far is equal to index,
        // we can make a new partition ending at
        // index i.
        if (max_so_far == i)
            ans++;
    }
    return ans;
}

// Driver code
int main()
{
    int arr[] = { 1, 0, 2, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << maxPartitions(arr, n);
    return 0;
}
```

Java

```
// java program to find Maximum number of partitions
// such that we can get a sorted array

import java.io.*;

class GFG
{
    // Function to find maximum partitions.
    static int maxPartitions(int arr[], int n)
    {
        int ans = 0, max_so_far = 0;
        for (int i = 0; i < n; ++i) {

            // Find maximum in prefix arr[0..i]
            max_so_far = Math.max(max_so_far, arr[i]);
```

```
// If maximum so far is equal to index,
// we can make a new partition ending at
// index i.
if (max_so_far == i)
    ans++;
}
return ans;
}

// Driver code
public static void main (String[] args)
{
    int arr[] = { 1, 0, 2, 3, 4 };
    int n = arr.length;
    System.out.println (maxPartitions(arr, n));

}
}

// This code is contributed by vt_m.
```

Python3

```
# Python3 program to find Maximum
# number of partitions such that
# we can get a sorted array.

# Function to find maximum partitions.
def maxPartitions(arr, n):

    ans = 0; max_so_far = 0
    for i in range(0, n):

        # Find maximum in prefix arr[0..i]
        max_so_far = max(max_so_far, arr[i])

        # If maximum so far is equal to
        # index, we can make a new partition
        # ending at index i.
        if (max_so_far == i):
            ans += 1

    return ans

# Driver code
arr = [1, 0, 2, 3, 4]
n = len(arr)
```

```
print(maxPartitions(arr, n))

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
// C# program to find Maximum number of partitions
// such that we can get a sorted array
using System;

class GFG
{
    // Function to find maximum partitions.
    static int maxPartitions(int []arr, int n)
    {
        int ans = 0, max_so_far = 0;
        for (int i = 0; i < n; ++i)
        {

            // Find maximum in prefix arr[0..i]
            max_so_far = Math.Max(max_so_far, arr[i]);

            // If maximum so far is equal to index,
            // we can make a new partition ending at
            // index i.
            if (max_so_far == i)
                ans++;

        }
        return ans;
    }

    // Driver code
    public static void Main ()
    {
        int []arr = { 1, 0, 2, 3, 4 };
        int n = arr.Length;
        Console.Write (maxPartitions(arr, n));

    }
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// PHP program to find Maximum
```

```
// number of partitions such
// that we can get a sorted array.

// Function to find maximum partitions.
function maxPartitions($arr, $n)
{
    $ans = 0;
    $max_so_far = 0;
    for ($i = 0; $i < $n; ++$i) {

        // Find maximum in prefix arr[0..i]
        $max_so_far = max($max_so_far, $arr[$i]);

        // If maximum so far is equal to index,
        // we can make a new partition ending at
        // index i.
        if ($max_so_far == $i)
            $ans++;
    }
    return $ans;
}

// Driver code
{
    $arr = array(1, 0, 2, 3, 4);
    $n = sizeof($arr) / sizeof($arr[0]);
    echo maxPartitions($arr, $n);
    return 0;
}

// This code is contributed by nitin mittal
?>
```

Output:

4

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/maximum-number-partitions-can-sorted-individually-make-sorted/>

Chapter 179

Maximum possible difference of two subsets of an array

Maximum possible difference of two subsets of an array - GeeksforGeeks

Given an array of n-integers. Array may contain repetitive elements but the highest frequency of any elements must not exceed two. You have to make two subsets such that difference of their elements sum is maximum and both of them jointly contains all of elements of given array along with the most important condition, no subset should contain repetitive elements.

Examples:

```
Input : arr[] = {5, 8, -1, 4}
Output : Maximum Difference = 18
Explanation :
Let Subset A = {5, 8, 4} & Subset B = {-1}
Sum of elements of subset A = 17, of subset B = -1
Difference of Sum of Both subsets = 17 - (-1) = 18
```

```
Input : arr[] = {5, 8, 5, 4}
Output : Maximum Difference = 12
Explanation :
Let Subset A = {5, 8, 4} & Subset B = {5}
Sum of elements of subset A = 17, of subset B = 5
Difference of Sum of Both subsets = 17 - 5 = 12
```

Before solving this question we have to take care of some given conditions and they are listed as:

- While building up the subsets, take care that no subset should contain repetitive elements. And for this we can conclude that all such elements whose frequency are

2, going to be part of both subsets and hence overall they don't have any impact on difference of subset sum. So, we can easily ignore them.

- For making the difference of sum of elements of both subset maximum we have to make subset in such a way that all positive elements belongs to one subset and negative ones to other subset.

Algorithm with time complexity O(n²):

```
for i=0 to n-1
    isSingleOccurance = true;
    for j= i+1 to n-1

        // if frequency of any element is two
        // make both equal to zero
        if arr[i] equals arr[j]
            arr[i] = arr[j] = 0
            isSingleOccurance = false;
            break;

        if isSingleOccurance == true
            if (arr[i] > 0)
                SubsetSum_1 += arr[i];
            else
                SubsetSum_2 += arr[i];
    return abs(SubsetSum_1 - SubsetSum2)
```

C++

```
// CPP find maximum difference of subset sum
#include <bits/stdc++.h>
using namespace std;

// function for maximum subset diff
int maxDiff(int arr[], int n)
{
    int SubsetSum_1 = 0, SubsetSum_2 = 0;
    for (int i = 0; i <= n - 1; i++) {

        bool isSingleOccurance = true;
        for (int j = i + 1; j <= n - 1; j++) {

            // if frequency of any element is two
            // make both equal to zero
            if (arr[i] == arr[j]) {
                isSingleOccurance = false;
```

```
        arr[i] = arr[j] = 0;
        break;
    }
}
if (isSingleOccurance) {
    if (arr[i] > 0)
        SubsetSum_1 += arr[i];
    else
        SubsetSum_2 += arr[i];
}
}
return abs(SubsetSum_1 - SubsetSum_2);
}

// driver program
int main()
{
    int arr[] = { 4, 2, -3, 3, -2, -2, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Maximum Difference = " << maxDiff(arr, n);
    return 0;
}
```

Java

```
// java find maximum difference
// of subset sum
import java .io.*;

public class GFG {

    // function for maximum subset diff
    static int maxDiff(int []arr, int n)
    {
        int SubsetSum_1 = 0, SubsetSum_2 = 0;
        for (int i = 0; i <= n - 1; i++)
        {
            boolean isSingleOccurance = true;
            for (int j = i + 1; j <= n - 1; j++)
            {

                // if frequency of any element
                // is two make both equal to
                // zero
                if (arr[i] == arr[j])
                {
                    isSingleOccurance = false;
                    arr[i] = arr[j] = 0;
```

```
        break;
    }
}
if (isSingleOccurance)
{
    if (arr[i] > 0)
        SubsetSum_1 += arr[i];
    else
        SubsetSum_2 += arr[i];
}
}

return Math.abs(SubsetSum_1 - SubsetSum_2);
}

// driver program
static public void main (String[] args)
{
    int []arr = { 4, 2, -3, 3, -2, -2, 8 };
    int n = arr.length;

    System.out.println("Maximum Difference = "
                       + maxDiff(arr, n));
}
}

// This code is contributed by vt_m.
```

Python3

```
# Python3 find maximum difference
# of subset sum

import math

# function for maximum subset diff
def maxDiff(arr, n) :
    SubsetSum_1 = 0
    SubsetSum_2 = 0
    for i in range(0, n) :

        isSingleOccurance = True
        for j in range(i + 1, n) :

            # if frequency of any element
            # is two make both equal to
            # zero
            if (arr[i] == arr[j]) :
```

```
isSingleOccurance = False
arr[i] = arr[j] = 0
break

if (isSingleOccurance == True) :
    if (arr[i] > 0) :
        SubsetSum_1 += arr[i]
    else :
        SubsetSum_2 += arr[i]

return abs(SubsetSum_1 - SubsetSum_2)

# Driver Code
arr = [4, 2, -3, 3, -2, -2, 8]
n = len(arr)
print ("Maximum Difference = {}"
       . format(maxDiff(arr, n)))

# This code is contributed by Manish Shaw
# (manishshaw1)
```

C#

```
// C# find maximum difference of
// subset sum
using System;

public class GFG {

    // function for maximum subset diff
    static int maxDiff(int []arr, int n)
    {
        int SubsetSum_1 = 0, SubsetSum_2 = 0;
        for (int i = 0; i <= n - 1; i++)
        {

            bool isSingleOccurance = true;
            for (int j = i + 1; j <= n - 1; j++)
            {

                // if frequency of any element
                // is two make both equal to
                // zero
                if (arr[i] == arr[j])
                {
                    isSingleOccurance = false;
                    arr[i] = arr[j] = 0;
                    break;
                }
            }
        }
    }
}
```

```
        }
    }
    if (isSingleOccurance)
    {
        if (arr[i] > 0)
            SubsetSum_1 += arr[i];
        else
            SubsetSum_2 += arr[i];
    }
}

return Math.Abs(SubsetSum_1 - SubsetSum_2);
}

// driver program
static public void Main ()
{
    int []arr = { 4, 2, -3, 3, -2, -2, 8 };
    int n = arr.Length;

    Console.WriteLine("Maximum Difference = "
                      + maxDiff(arr, n));
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP find maximum difference
// of subset sum

// function for maximum subset diff
function maxDiff($arr, $n)
{
    $SubsetSum_1 = 0;
    $SubsetSum_2 = 0;
    for ($i = 0; $i <= $n - 1; $i++)
    {

        $isSingleOccurance = true;
        for ($j = $i + 1; $j <= $n - 1; $j++)
        {

            // if frequency of any element is two
            // make both equal to zero
            if ($arr[$i] == $arr[$j])
```

```
{  
    $isSingleOccurance = false;  
    $arr[$i] = $arr[$j] = 0;  
    break;  
}  
}  
if ($isSingleOccurance)  
{  
    if ($arr[$i] > 0)  
        $SubsetSum_1 += $arr[$i];  
    else  
        $SubsetSum_2 += $arr[$i];  
}  
}  
}  
return abs($SubsetSum_1 - $SubsetSum_2);  
}  
  
// Driver Code  
$arr = array(4, 2, -3, 3, -2, -2, 8);  
$n = sizeof($arr);  
echo "Maximum Difference = " , maxDiff($arr, $n);  
  
// This code is contributed by nitin mittal  
?>
```

Output:

Maximum Difference = 20

Algorithm with time complexity O(n log n):

```
-> sort the array  
-> for i =0 to n-2  
    // consecutive two elements are not equal  
    // add absolute arr[i] to result  
    if arr[i] != arr[i+1]  
        result += abs(arr[i])  
    // else skip next element too  
    else  
        i++;  
  
// special check for last two elements  
-> if (arr[n-2] != arr[n-1])  
    result += arr[n-1]  
  
-> return result;
```

C++

```
// CPP find maximum difference of subset sum
#include <bits/stdc++.h>
using namespace std;

// function for maximum subset diff
int maxDiff(int arr[], int n)
{
    int result = 0;

    // sort the array
    sort(arr, arr + n);

    // calculate the result
    for (int i = 0; i < n - 1; i++) {
        if (arr[i] != arr[i + 1])
            result += abs(arr[i]);
        else
            i++;
    }

    // check for last element
    if (arr[n - 2] != arr[n - 1])
        result += abs(arr[n - 1]);

    // return result
    return result;
}

// driver program
int main()
{
    int arr[] = { 4, 2, -3, 3, -2, -2, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Maximum Difference = " << maxDiff(arr, n);
    return 0;
}
```

Java

```
// java find maximum difference of
// subset sum
import java.io.*;
import java.util.*;

public class GFG {
```

```
// function for maximum subset diff
static int maxDiff(int []arr, int n)
{
    int result = 0;

    // sort the array
    Arrays.sort(arr);

    // calculate the result
    for (int i = 0; i < n - 1; i++)
    {
        if (arr[i] != arr[i + 1])
            result += Math.abs(arr[i]);
        else
            i++;
    }

    // check for last element
    if (arr[n - 2] != arr[n - 1])
        result += Math.abs(arr[n - 1]);

    // return result
    return result;
}

// driver program
static public void main (String[] args)
{
    int[] arr = { 4, 2, -3, 3, -2, -2, 8 };
    int n = arr.length;

    System.out.println("Maximum Difference = "
                       + maxDiff(arr, n));
}
}

// This code is contributed by vt_m.

C#

// C# find maximum difference
// of subset sum
using System;

public class GFG {

    // function for maximum subset diff
```

```
static int maxDiff(int []arr, int n)
{
    int result = 0;

    // sort the array
    Array.Sort(arr);

    // calculate the result
    for (int i = 0; i < n - 1; i++)
    {
        if (arr[i] != arr[i + 1])
            result += Math.Abs(arr[i]);
        else
            i++;
    }

    // check for last element
    if (arr[n - 2] != arr[n - 1])
        result += Math.Abs(arr[n - 1]);

    // return result
    return result;
}

// driver program
static public void Main ()
{
    int[] arr = { 4, 2, -3, 3, -2, -2, 8 };
    int n = arr.Length;

    Console.WriteLine("Maximum Difference = "
                      + maxDiff(arr, n));
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP find maximum difference of subset sum

// function for maximum subset diff
function maxDiff( $arr, $n)
{
    $result = 0;

    // sort the array
```

```
sort($arr);

// calculate the result
for ( $i = 0; $i < $n - 1; $i++)
{
    if ($arr[$i] != $arr[$i + 1])
        $result += abs($arr[$i]);
    else
        $i++;
}

// check for last element
if ($arr[$n - 2] != $arr[$n - 1])
    $result += abs($arr[$n - 1]);

// return result
return $result;
}

// Driver Code
$arr = array( 4, 2, -3, 3, -2, -2, 8 );
$n = count($arr);
echo "Maximum Difference = "
    , maxDiff($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
Maximum Difference = 20
```

Algorithm with time complexity O(n):

```
make hash table for positive elements:
    for all positive elements(arr[i])
        if frequency == 1
            SubsetSum_1 += arr[i];
make hash table for negative elements:
    for all negative elements
        if frequency == 1
            SubsetSum_2 += arr[i];
return abs(SubsetSum_1 - SubsetSum2)

// CPP find maximum difference of subset sum
```

```
#include <bits/stdc++.h>
using namespace std;

// function for maximum subset diff
int maxDiff(int arr[], int n)
{
    unordered_map<int, int> hashPositive;
    unordered_map<int, int> hashNegative;

    int SubsetSum_1 = 0, SubsetSum_2 = 0;

    // construct hash for positive elements
    for (int i = 0; i <= n - 1; i++)
        if (arr[i] > 0)
            hashPositive[arr[i]]++;

    // calculate subset sum for positive elements
    for (int i = 0; i <= n - 1; i++)
        if (arr[i] > 0 && hashPositive[arr[i]] == 1)
            SubsetSum_1 += arr[i];

    // construct hash for negative elements
    for (int i = 0; i <= n - 1; i++)
        if (arr[i] < 0)
            hashNegative[abs(arr[i])]++;

    // calculate subset sum for negative elements
    for (int i = 0; i <= n - 1; i++)
        if (arr[i] < 0 &&
            hashNegative[abs(arr[i])] == 1)
            SubsetSum_2 += arr[i];

    return abs(SubsetSum_1 - SubsetSum_2);
}

// driver program
int main()
{
    int arr[] = { 4, 2, -3, 3, -2, -2, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Maximum Difference = " << maxDiff(arr, n);
    return 0;
}
```

Output:

```
Maximum Difference = 20
```

Improved By : [vt_m](#), [nitin mittal](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/maximum-possible-difference-two-subsets-array/>

Chapter 180

Maximum product of a triplet (subsequence of size 3) in array

Maximum product of a triplet (subsequence of size 3) in array - GeeksforGeeks

Given an integer array, find a maximum product of a triplet in array.

Examples:

Input: [10, 3, 5, 6, 20]

Output: 1200

Multiplication of 10, 6 and 20

Input: [-10, -3, -5, -6, -20]

Output: -90

Input: [1, -4, 3, -6, 7, 0]

Output: 168

Approach 1 (Naive, $O(n^3)$ time, $O(1)$ Space)

A simple solution is to check for every triplet using three nested loops. Below is its implementation –

C++

```
// A C++ program to find a maximum product of a
// triplet in array of integers
#include <bits/stdc++.h>
using namespace std;

/* Function to find a maximum product of a triplet
```

```
    in array of integers of size n */
int maxProduct(int arr[], int n)
{
    // if size is less than 3, no triplet exists
    if (n < 3)
        return -1;

    // will contain max product
    int max_product = INT_MIN;

    for (int i = 0; i < n - 2; i++)
        for (int j = i + 1; j < n - 1; j++)
            for (int k = j + 1; k < n; k++)
                max_product = max(max_product,
                                   arr[i] * arr[j] * arr[k]);

    return max_product;
}

// Driver program to test above functions
int main()
{
    int arr[] = { 10, 3, 5, 6, 20 };
    int n = sizeof(arr) / sizeof(arr[0]);

    int max = maxProduct(arr, n);

    if (max == -1)
        cout << "No Triplet Exists";
    else
        cout << "Maximum product is " << max;

    return 0;
}
```

Python3

```
# Python program to find a maximum
# product of a triplet in array
# of integers
import sys

# Function to find a maximum
# product of a triplet in array
# of integers of size n
def maxProduct(arr, n):

    # if size is less than 3,
```

```
# no triplet exists
if n < 3:
    return -1

# will contain max product
max_product = -(sys.maxsize - 1)

for i in range(0, n - 2):
    for j in range(i + 1, n - 1):
        for k in range(j + 1, n):
            max_product = max(
                max_product, arr[i]
                * arr[j] * arr[k])

return max_product

# Driver Program
arr = [10, 3, 5, 6, 20]
n = len(arr)

max = maxProduct(arr, n)

if max == -1:
    print("No Triplet Exists")
else:
    print("Maximum product is", max)

# This code is contributed by Shrikant13
```

C#

```
// A C# program to find a
// maximum product of a
// triplet in array of integers
using System;
class GFG {

    // Function to find a maximum
    // product of a triplet in array
    // of integers of size n
    static int maxProduct(int []arr, int n)
    {

        // if size is less than
        // 3, no triplet exists
        if (n < 3)
            return -1;


```

```
// will contain max product
int max_product = int.MinValue;

for (int i = 0; i < n - 2; i++)
    for (int j = i + 1; j < n - 1; j++)
        for (int k = j + 1; k < n; k++)
            max_product = Math.Max(max_product,
                arr[i] * arr[j] * arr[k]);

return max_product;
}

// Driver Code
public static void Main ()
{
    int []arr = { 10, 3, 5, 6, 20 };
    int n = arr.Length;

    int max = maxProduct(arr, n);

    if (max == -1)
        Console.WriteLine("No Triplet Exists");
    else
        Console.WriteLine("Maximum product is " + max);
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// A PHP program to find a
// maximum product of a
// triplet in array of integers

// Function to find a maximum
// product of a triplet
// in array of integers of
// size n
function maxProduct($arr, $n)
{
    $INT_MIN = 0;

    // if size is less than
    // 3, no triplet exists
    if ($n < 3)
        return -1;
```

```
// will contain max product
$max_product = $INT_MIN;

for ($i = 0; $i < $n - 2; $i++)
    for ($j = $i + 1; $j < $n - 1; $j++)
        for ($k = $j + 1; $k < $n; $k++)
            $max_product = max($max_product,
                $arr[$i] * $arr[$j] * $arr[$k]);

return $max_product;
}

// Driver Code
$arr = array(10, 3, 5, 6, 20 );
$n = sizeof($arr);
$max = maxProduct($arr, $n);
if ($max == -1)
    echo "No Triplet Exists";
else
    echo "Maximum product is " ,$max;

// This code is contributed by nitin mittal.
?>
```

Output :

```
Maximum product is 1200
```

Approach 2: O(n) Time, O(n) Space

1. Construct four auxiliary arrays leftMax[], rightMax[], leftMin[] and rightMin[] of same size as input array.
2. Fill leftMax[], rightMax[], leftMin[] and rightMin[] in below manner.
 - leftMax[i] will contain maximum element on left of arr[i] excluding arr[i]. For index 0, left will contain -1.
 - leftMin[i] will contain minimum element on left of arr[i] excluding arr[i]. For index 0, left will contain -1.
 - rightMax[i] will contain maximum element on right of arr[i] excluding arr[i]. For index n-1, right will contain -1.
 - rightMin[i] will contain minimum element on right of arr[i] excluding arr[i]. For index n-1, right will contain -1.
3. For all array indexes i except first and last index, compute maximum of arr[i]*x*y where x can be leftMax[i] or leftMin[i] and y can be rightMax[i] or rightMin[i].

4. Return the maximum from step 3.

Below is its C++ implementation –

```
// A C++ program to find a maximum product of a triplet
// in array of integers
#include <bits/stdc++.h>
using namespace std;

/* Function to find a maximum product of a triplet
in array of integers of size n */
int maxProduct(int arr[], int n)
{
    // if size is less than 3, no triplet exists
    if (n < 3)
        return -1;

    // Construct four auxiliary vectors
    // of size n and initailize them by -1
    vector<int> leftMin(n, -1);
    vector<int> rightMin(n, -1);
    vector<int> leftMax(n, -1);
    vector<int> rightMax(n, -1);

    // will contain max product
    int max_product = INT_MIN;

    // to store maximum element on left of array
    int max_sum = arr[0];

    // to store minimum element on left of array
    int min_sum = arr[0];

    // leftMax[i] will contain max element
    // on left of arr[i] excluding arr[i].
    // leftMin[i] will contain min element
    // on left of arr[i] excluding arr[i].
    for (int i = 1; i < n; i++)
    {
        leftMax[i] = max_sum;
        if (arr[i] > max_sum)
            max_sum = arr[i];

        leftMin[i] = min_sum;
        if (arr[i] < min_sum)
            min_sum = arr[i];
    }
}
```

```
// reset max_sum to store maximum element on
// right of array
max_sum = arr[n - 1];

// reset min_sum to store minimum element on
// right of array
min_sum = arr[n - 1];

// rightMax[i] will contain max element
// on right of arr[i] excluding arr[i].
// rightMin[i] will contain min element
// on right of arr[i] excluding arr[i].
for (int j = n - 2; j >= 0; j--)
{
    rightMax[j] = max_sum;
    if (arr[j] > max_sum)
        max_sum = arr[j];

    rightMin[j] = min_sum;
    if (arr[j] < min_sum)
        min_sum = arr[j];
}

// For all array indexes i except first and
// last, compute maximum of arr[i]*x*y where
// x can be leftMax[i] or leftMin[i] and
// y can be rightMax[i] or rightMin[i].
for (int i = 1; i < n - 1; i++)
{
    int max1 = max(arr[i] * leftMax[i] * rightMax[i],
                   arr[i] * leftMin[i] * rightMin[i]);

    int max2 = max(arr[i] * leftMax[i] * rightMin[i],
                   arr[i] * leftMin[i] * rightMax[i]);

    max_product = max(max_product, max(max1, max2));
}

return max_product;
}

// Driver program to test above functions
int main()
{
    int arr[] = { 1, 4, 3, -6, -7, 0 };
    int n = sizeof(arr) / sizeof(arr[0]);

    int max = maxProduct(arr, n);
```

```
if (max == -1)
    cout << "No Triplet Exists";
else
    cout << "Maximum product is " << max;

return 0;
}
```

Output :

```
Maximum product is 168
```

Approach 3: O(nlogn) Time, O(1) Space

Sort the array using some efficient in-place sorting algorithm in ascending order.

Return the maximum of product of last three elements of the array and product of first two elements and last element.

Below is its C++ implementation –

```
// A C++ program to find a maximum product of a
// triplet in array of integers
#include <bits/stdc++.h>
using namespace std;

/* Function to find a maximum product of a triplet
   in array of integers of size n */
int maxProduct(int arr[], int n)
{
    // if size is less than 3, no triplet exists
    if (n < 3)
        return -1;

    // Sort the array in ascending order
    sort(arr, arr + n);

    // Return the maximum of product of last three
    // elements and product of first two elements
    // and last element
    return max(arr[0] * arr[1] * arr[n - 1],
               arr[n - 1] * arr[n - 2] * arr[n - 3]);
}

// Driver program to test above functions
int main()
{
```

```
int arr[] = { -10, -3, 5, 6, -20 };
int n = sizeof(arr) / sizeof(arr[0]);

int max = maxProduct(arr, n);

if (max == -1)
    cout << "No Triplet Exists";
else
    cout << "Maximum product is " << max;

return 0;
}
```

Output :

```
Maximum product is 1200
```

Approach 4: O(n) Time, O(1) Space

1. Scan the array and compute Maximum, second maximum and third maximum element present in the array.
2. Scan the array and compute Minimum and second minimum element present in the array.
3. Return the maximum of product of Maximum, second maximum and third maximum and product of Minimum, second minimum and Maximum element.

Note – Step 1 and Step 2 can be done in single traversal of the array.

Below is its C++ implementation –

```
// A O(n) C++ program to find maximum product pair in
// an array.
#include <bits/stdc++.h>
using namespace std;

/* Function to find a maximum product of a triplet
   in array of integers of size n */
int maxProduct(int arr[], int n)
{
    // if size is less than 3, no triplet exists
    if (n < 3)
        return -1;

    // Initialize Maximum, second maximum and third
```

```
// maximum element
int maxA = INT_MIN, maxB = INT_MIN, maxC = INT_MIN;

// Initialize Minimum and second mimimum element
int minA = INT_MAX, minB = INT_MAX;

for (int i = 0; i < n; i++)
{
    // Update Maximum, second maximum and third
    // maximum element
    if (arr[i] > maxA)
    {
        maxC = maxB;
        maxB = maxA;
        maxA = arr[i];
    }

    // Update second maximum and third maximum element
    else if (arr[i] > maxB)
    {
        maxC = maxB;
        maxB = arr[i];
    }

    // Update third maximum element
    else if (arr[i] > maxC)
        maxC = arr[i];

    // Update Minimum and second mimimum element
    if (arr[i] < minA)
    {
        minB = minA;
        minA = arr[i];
    }

    // Update second mimimum element
    else if(arr[i] < minB)
        minB = arr[i];
}

return max(minA * minB * maxA,
           maxA * maxB * maxC);
}

// Driver program to test above function
int main()
{
    int arr[] = { 1, -4, 3, -6, 7, 0 };
```

```
int n = sizeof(arr) / sizeof(arr[0]);  
  
int max = maxProduct(arr, n);  
  
if (max == -1)  
    cout << "No Triplet Exists";  
else  
    cout << "Maximum product is " << max;  
  
return 0;  
}
```

Output :

Maximum product is 168

Exercise:

1. Print the triplet that has maximum product.
2. Find a minimum product of a triplet in array.

This article is contributed by **Aditya Goel**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [shrikanth13](#), [nitin mittal](#), [vt_m](#), [Rasheed60](#)

Source

<https://www.geeksforgeeks.org/find-maximum-product-of-a-triplet-in-array/>

Chapter 181

Maximum product of subsequence of size k

Maximum product of subsequence of size k - GeeksforGeeks

Given an array A[] of n integers, the task is to find a subsequence of size k whose product is maximum among all possible k sized subsequences of given array.

Constraints

```
1 <= n <= 10^5  
1 <= k <= n
```

Examples:

```
Input : A[] = {1, 2, 0, 3},  
        k = 2  
Output : 6  
Explanation : Subsequence containing elements  
{2, 3} gives maximum product : 2*3 = 6
```

```
Input : A[] = {1, 2, -1, -3, -6, 4},  
        k = 4  
Output : 144  
Explanation : Subsequence containing {2, -3,  
-6, 4} gives maximum product : 2*(-3)*(-6)*4  
= 144
```

Following are different cases that arise in this problem.

- **CASE I: if maximum element of A is 0 and k is odd** Here if we don't include 0 in subsequence then product will be less than 0, Since the product of an odd number of negative integers gives a negative integer. Hence 0 must be included in the subsequence. Since 0 is present in subsequence, product of subsequence is 0. Answer = 0.
- **CASE II: if maximum element of A is negative and k is odd.** Here product will be less than 0,
Since the product of an odd number of negative integers gives a negative integer. So to get maximum product, we take product of smallest (absolute value wise) k elements. Since absolute value wise : $A[n-1] > A[n-2] \dots > A[0]$. Hence we take product of $A[n-1], A[n-2], A[n-3], \dots, A[n-k]$
Answer = $A[n-1] * A[n-2] * \dots * A[n-k]$
- **CASE III: if maximum element of A is positive and k is odd.** Here in subsequence of k size if all elements are < 0 then product will be less than 0, Since the product of an odd number of negative integers gives a negative integer. Hence, atleast one element must be a positive integer in the subsequence. To get max product max positive number should be present in the subsequence. Now we need to add $k-1$ more elements to the subsequence. Since k is odd, $k-1$ becomes even. So problem boils down to case IV. Answer = $A[n-1] * \text{Answer from CASE IV}$.
- **CASE IV: if k is even.** Since k is even, we always add pair in subsequence. So total pairs required to be added in subsequence is $k/2$. So for simplicity, our new k is $k/2$. Now since A is sorted, pair with maximum product will always be either $A[0]*A[1]$ OR $A[n-1]*A[n-2]$. In case of doubt about the previous statement think about negative numbers

Now,

```

if A[0]*A[1] > A[n-1]*A[n-2] ,
second max product pair will be
either A[2]*A[3] OR A[n-1]*[n-2].
else second max product pair will be
either A[0]*A[1] OR A[n-3]*[n-4].
```

Here is implementation of above solution

C/C++

```

// C++ code to find maximum possible product of
// sub-sequence of size k from given array of n
// integers
#include <algorithm> // for sorting
#include <iostream>
using namespace std;

// Required function
int maxProductSubarrayOfSizeK(int A[], int n, int k)
{
    // sorting given input array
```

```
sort(A, A + n);

// variable to store final product of all element
// of sub-sequence of size k
int product = 1;

// CASE I
// If max element is 0 and
// k is odd then max product will be 0
if (A[n - 1] == 0 && (k & 1))
    return 0;

// CASE II
// If all elements are negative and
// k is odd then max product will be
// product of rightmost-subarray of size k
if (A[n - 1] <= 0 && (k & 1)) {
    for (int i = n - 1; i >= n - k; i--)
        product *= A[i];
    return product;
}

// else
// i is current left pointer index
int i = 0;

// j is current right pointer index
int j = n - 1;

// CASE III
// if k is odd and rightmost element in
// sorted array is positive then it
// must come in subsequence
// Multiplying A[j] with product and
// correspondingly changing j
if (k & 1) {
    product *= A[j];
    j--;
    k--;
}

// CASE IV
// Now k is even
// Now we deal with pairs
// Each time a pair is multiplied to product
// ie.. two elements are added to subsequence each time
// Effectively k becomes half
// Hence, k >= 1 means k /= 2
```

```
k >>= 1;

// Now finding k corresponding pairs
// to get maximum possible value of product
for (int itr = 0; itr < k; itr++) {

    // product from left pointers
    int left_product = A[i] * A[i + 1];

    // product from right pointers
    int right_product = A[j] * A[j - 1];

    // Taking the max product from two choices
    // Correspondingly changing the pointer's position
    if (left_product > right_product) {
        product *= left_product;
        i += 2;
    }
    else {
        product *= right_product;
        j -= 2;
    }
}

// Finally return product
return product;
}

// Driver Code to test above function
int main()
{
    int A[] = { 1, 2, -1, -3, -6, 4 };
    int n = sizeof(A) / sizeof(A[0]);
    int k = 4;
    cout << maxProductSubarrayOfSizeK(A, n, k);

    return 0;
}
```

Java

```
// Java program to find maximum possible product of
// sub-sequence of size k from given array of n
// integers
import java.io.*;
import java.util.*;

class GFG {
```

```
// Function to find maximum possible product
static int maxProductSubarrayOfSizeK(int A[], int n, int k)
{
    // sorting given input array
    Arrays.sort(A);

    // variable to store final product of all element
    // of sub-sequence of size k
    int product = 1;

    // CASE I
    // If max element is 0 and
    // k is odd then max product will be 0
    if (A[n - 1] == 0 && k % 2 != 0)
        return 0;

    // CASE II
    // If all elements are negative and
    // k is odd then max product will be
    // product of rightmost-subarray of size k
    if (A[n - 1] <= 0 && k % 2 != 0) {
        for (int i = n - 1; i >= n - k; i--)
            product *= A[i];
        return product;
    }

    // else
    // i is current left pointer index
    int i = 0;

    // j is current right pointer index
    int j = n - 1;

    // CASE III
    // if k is odd and rightmost element in
    // sorted array is positive then it
    // must come in subsequence
    // Multiplying A[j] with product and
    // correspondingly changing j
    if (k % 2 != 0) {
        product *= A[j];
        j--;
        k--;
    }

    // CASE IV
    // Now k is even
    // Now we deal with pairs
```

```
// Each time a pair is multiplied to product
// ie.. two elements are added to subsequence each time
// Effectively k becomes half
// Hence, k >= 1 means k /= 2
k >>= 1;

// Now finding k corresponding pairs
// to get maximum possible value of product
for (int itr = 0; itr < k; itr++) {
    // product from left pointers
    int left_product = A[i] * A[i + 1];

    // product from right pointers
    int right_product = A[j] * A[j - 1];

    // Taking the max product from two choices
    // Correspondingly changing the pointer's position
    if (left_product > right_product) {
        product *= left_product;
        i += 2;
    }
    else {
        product *= right_product;
        j -= 2;
    }
}

// Finally return product
return product;
}

// driver program
public static void main(String[] args)
{
    int A[] = { 1, 2, -1, -3, -6, 4 };
    int n = A.length;
    int k = 4;
    System.out.println(maxProductSubarrayOfSizeK(A, n, k));
}
}

// Contributed by Pramod Kumar

C#
// C# program to find maximum possible
// product of sub-sequence of size k
// from given array of n integers
```

```
using System;

class GFG {

    // Function to find maximum possible product
    static int maxProductSubarrayOfSizeK(int[] A, int n,
                                         int k)
    {
        // sorting given input array
        Array.Sort(A);

        // variable to store final product of
        // all element of sub-sequence of size k
        int product = 1;
        int i;

        // CASE I
        // If max element is 0 and
        // k is odd then max product will be 0
        if (A[n - 1] == 0 && k % 2 != 0)
            return 0;

        // CASE II
        // If all elements are negative and
        // k is odd then max product will be
        // product of rightmost-subarray of size k
        if (A[n - 1] <= 0 && k % 2 != 0) {
            for (i = n - 1; i >= n - k; i--)
                product *= A[i];
            return product;
        }

        // else
        // i is current left pointer index
        i = 0;

        // j is current right pointer index
        int j = n - 1;

        // CASE III
        // if k is odd and rightmost element in
        // sorted array is positive then it
        // must come in subsequence
        // Multiplying A[j] with product and
        // correspondingly changing j
        if (k % 2 != 0) {
            product *= A[j];
            j--;
        }
    }
}
```

```
        k--;
    }

    // CASE IV
    // Now k is even
    // Now we deal with pairs
    // Each time a pair is multiplied to
    // product i.e.. two elemnts are added to
    // subsequence each time Effectively k becomes half
    // Hence, k >= 1 means k /= 2
    k >>= 1;

    // Now finding k corresponding pairs
    // to get maximum possible value of product
    for (int itr = 0; itr < k; itr++) {

        // product from left pointers
        int left_product = A[i] * A[i + 1];

        // product from right pointers
        int right_product = A[j] * A[j - 1];

        // Taking the max product from two choices
        // Correspondingly changing the pointer's position
        if (left_product > right_product) {
            product *= left_product;
            i += 2;
        }
        else {
            product *= right_product;
            j -= 2;
        }
    }

    // Finally return product
    return product;
}

// driver program
public static void Main()
{
    int[] A = { 1, 2, -1, -3, -6, 4 };
    int n = A.Length;
    int k = 4;
    Console.WriteLine(maxProductSubarrayOfSizeK(A, n, k));
}
```

```
// This code is contributed by vt_m.
```

Output:

144

Time Complexity : $O(n * \log n)$ $O(n * \log n)$ from sorting + $O(k)$ from one traversal in array = $O(n * \log n)$

Auxiliary Space : $O(1)$

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/maximum-product-subsequence-size-k/>

Chapter 182

Maximum sum of absolute difference of an array

Maximum sum of absolute difference of an array - GeeksforGeeks

Given an array, we need to find the maximum sum of absolute difference of the array elements for a sequence of this array.

Examples:

```
Input : { 1, 2, 4, 8 }
Output : 18
Explanation : For the given array there are
several sequence possible
like : {2, 1, 4, 8}
        {4, 2, 1, 8} and some more.
Now, the absolute difference of an array sequence will be
like for this array sequence {1, 2, 4, 8}, the absolute
difference sum is
= |1-2| + |2-4| + |4-8| + |8-1|
= 14
For the given array, we get the maximum value for
the sequence {1, 8, 2, 4}
= |1-8| + |8-2| + |2-4| + |4-1|
= 18
```

To solve this problem, we have to think greedily that how can we maximize the difference value of the elements so that we can have a maximum sum. This is possible only if we calculate the difference between some very high values and some very low values like (highest - smallest). This is the idea which we have to use to solve this problem. Let us see the above example, we will have maximum difference possible for sequence {1, 8, 2, 4} because in this sequence we will get some high difference values, (1-8 = 7, 8-2 = 6 ..). Here, by

placing 8(highest element) in place of 1 and 2 we get two high difference values. Similarly, for the other values, we will place next highest values in between other, as we have only one left i.e 4 which is placed at last.

Algorithm: *To get the maximum sum, we should have a sequence in which small and large elements comes alternate. This is done to get maximum difference.*

For the implementation of the above algorithm ->

1. We will sort the array.
2. Calculate the final sequence by taking one smallest element and largest element from the sorted array and make one vector array of this final sequence.
3. Finally, calculate the sum of absolute difference between the elements of the array.

Below is the implementation of above idea :

C++

```
// CPP implementation of
// above algorithm
#include <bits/stdc++.h>
using namespace std;

int MaxSumDifference(int a[], int n)
{
    // final sequence stored in the vector
    vector<int> finalSequence;

    // sort the original array
    // so that we can retrieve
    // the large elements from
    // the end of array elements
    sort(a, a + n);

    // In this loop first we will insert
    // one smallest element not entered
    // till that time in final sequence
    // and then enter a highest element
    // (not entered till that time) in
    // final sequence so that we
    // have large difference value. This
    // process is repeated till all array
    // has completely entered in sequence.
    // Here, we have loop till n/2 because
    // we are inserting two elements at a
    // time in loop.
    for (int i = 0; i < n / 2; ++i) {
        finalSequence.push_back(a[i]);
        finalSequence.push_back(a[n - i - 1]);
    }
}
```

```
// variable to store the
// maximum sum of absolute
// difference
int MaximumSum = 0;

// In this loop absolute difference
// of elements for the final sequence
// is calculated.
for (int i = 0; i < n - 1; ++i) {
    MaximumSum = MaximumSum + abs(finalSequence[i] -
                                    finalSequence[i + 1]);
}

// absolute difference of last element
// and 1st element
MaximumSum = MaximumSum + abs(finalSequence[n - 1] -
                                finalSequence[0]);

// return the value
return MaximumSum;
}

// Driver function
int main()
{
    int a[] = { 1, 2, 4, 8 };
    int n = sizeof(a) / sizeof(a[0]);

    cout << MaxSumDifference(a, n) << endl;
}
```

Java

```
// Java implementation of
// above algorithm
import java.io.*;
import java.util.*;

public class GFG {

    static int MaxSumDifference(Integer []a, int n)
    {

        // final sequence stored in the vector
        List<Integer> finalSequence =
            new ArrayList<Integer>();

        // sort the original array
```

```
// so that we can retrieve
// the large elements from
// the end of array elements
Arrays.sort(a);

// In this loop first we will insert
// one smallest element not entered
// till that time in final sequence
// and then enter a highest element
// (not entered till that time) in
// final sequence so that we
// have large difference value. This
// process is repeated till all array
// has completely entered in sequence.
// Here, we have loop till n/2 because
// we are inserting two elements at a
// time in loop.
for (int i = 0; i < n / 2; ++i) {
    finalSequence.add(a[i]);
    finalSequence.add(a[n - i - 1]);
}

// variable to store the
// maximum sum of absolute
// difference
int MaximumSum = 0;

// In this loop absolute difference
// of elements for the final sequence
// is calculated.
for (int i = 0; i < n - 1; ++i) {
    MaximumSum = MaximumSum +
        Math.abs(finalSequence.get(i)
            - finalSequence.get(i + 1));
}

// absolute difference of last element
// and 1st element
MaximumSum = MaximumSum +
    Math.abs(finalSequence.get(n - 1)
        - finalSequence.get(0));

// return the value
return MaximumSum;
}

// Driver Code
public static void main(String args[])

```

```
{  
    Integer []a = { 1, 2, 4, 8 };  
    int n = a.length;  
  
    System.out.print(MaxSumDifference(a, n));  
}  
}  
  
// This code is contributed by  
// Manish Shaw (manishshaw1)
```

Python3

```
import numpy as np  
class GFG:  
  
    def MaxSumDifference(a,n):  
        # sort the original array  
        # so that we can retrieve  
        # the large elements from  
        # the end of array elements  
        np.sort(a);  
  
        # In this loop first we will  
        # insert one smallest element  
        # not entered till that time  
        # in final sequence and then  
        # enter a highest element(not  
        # entered till that time) in  
        # final sequence so that we  
        # have large difference value.  
        # This process is repeated till  
        # all array has completely  
        # entered in sequence. Here,  
        # we have loop till n/2 because  
        # we are inserting two elements  
        # at a time in loop.  
        j = 0  
        finalSequence = [0 for x in range(n)]  
        for i in range(0, int(n / 2)):  
            finalSequence[j] = a[i]  
            finalSequence[j + 1] = a[n - i - 1]  
            j = j + 2  
  
        # variable to store the  
        # maximum sum of absolute  
        # difference  
        MaximumSum = 0
```

```
# In this loop absolute
# difference of elements
# for the final sequence
# is calculated.
for i in range(0, n - 1):
    MaximumSum = (MaximumSum +
                   abs(finalSequence[i] -
                        finalSequence[i + 1]))

# absolute difference of last
# element and 1st element
MaximumSum = (MaximumSum +
              abs(finalSequence[n - 1] -
                  finalSequence[0]));

# return the value
print (MaximumSum)

# Driver Code
a = [ 1, 2, 4, 8 ]
n = len(a)
GFG.MaxSumDifference(a, n);

# This code is contributed
# by Prateek Bajaj
```

C#

```
// C# implementation of
// above algorithm
using System;
using System.Collections.Generic;
class GFG {

    static int MaxSumDifference(int []a, int n)
    {

        // final sequence stored in the vector
        List<int> finalSequence = new List<int>();

        // sort the original array
        // so that we can retrieve
        // the large elements from
        // the end of array elements
        Array.Sort(a);

        // In this loop first we will insert
```

```
// one smallest element not entered
// till that time in final sequence
// and then enter a highest element
// (not entered till that time) in
// final sequence so that we
// have large difference value. This
// process is repeated till all array
// has completely entered in sequence.
// Here, we have loop till n/2 because
// we are inserting two elements at a
// time in loop.
for (int i = 0; i < n / 2; ++i) {
    finalSequence.Add(a[i]);
    finalSequence.Add(a[n - i - 1]);
}

// variable to store the
// maximum sum of absolute
// difference
int MaximumSum = 0;

// In this loop absolute difference
// of elements for the final sequence
// is calculated.
for (int i = 0; i < n - 1; ++i) {
    MaximumSum = MaximumSum + Math.Abs(finalSequence[i] -
        finalSequence[i + 1]);
}

// absolute difference of last element
// and 1st element
MaximumSum = MaximumSum + Math.Abs(finalSequence[n - 1] -
    finalSequence[0]);

// return the value
return MaximumSum;
}

// Driver Code
public static void Main()
{
    int []a = { 1, 2, 4, 8 };
    int n = a.Length;

    Console.WriteLine(MaxSumDifference(a, n));
}
```

```
// This code is contributed by  
// Manish Shaw (manishshaw1)
```

Output :

18

Improved By : [manishshaw1](#), Prateek Bajaj

Source

<https://www.geeksforgeeks.org/maximum-sum-absolute-difference-array/>

Chapter 183

Maximum sum of pairwise product in an array with negative allowed

Maximum sum of pairwise product in an array with negative allowed - GeeksforGeeks

Given an array of n elements. Find maximum sum of pairwise multiplications. Sum can be larger so take mod with 10^9+7 . If there are odd elements, then we can add any one element (without forming a pair) to the sum.

Examples:

```
Input : arr[] = {-1, 4, 5, -7, -4, 9, 0}
Output : 77
So to get the maximum sum, the arrangement will
be {-7, -4}, {-1, 0}, {9, 5} and {4}.
So the answer is (-7*(-4))+((-1)*0)+(9*5)+(4) ={77}.
```

```
Input : arr[] = {8, 7, 9}
Output : 79
Answer is (9*8) +(7) = 79.
```

- 1- Sort the given array.
- 2- First, multiply the negative numbers pairwise from the starting and add to the total_sum.
- 3- Second, multiply the positive numbers pairwise from the last and to the total_sum.
- 4- Check if negative and positive both counts are odd, then add the product of last pair i.e. last negative and positive left.
- 5- Or if any of the one counts is odd, then add that element left.
- 6- Return sum.

C++

```
// C++ program for above implementation
#include <bits/stdc++.h>
#define Mod 1000000007
using namespace std;

// Function to find the maximum sum
long long int findSum(int arr[], int n)
{
    long long int sum = 0;

    // Sort the array first
    sort(arr, arr + n);

    // First multiply negative numbers pairwise
    // and sum up from starting as to get maximum
    // sum.
    int i = 0;
    while (i < n && arr[i] < 0) {
        if (i != n - 1 && arr[i + 1] <= 0) {
            sum = (sum + (arr[i] * arr[i + 1]) % Mod) % Mod;
            i += 2;
        }
        else
            break;
    }

    // Second multiply positive numbers pairwise
    // and summed up from the last as to get maximum
    // sum.
    int j = n - 1;
    while (j >= 0 && arr[j] > 0) {
        if (j != 0 && arr[j - 1] > 0) {
            sum = (sum + (arr[j] * arr[j - 1]) % Mod) % Mod;
            j -= 2;
        }
        else
            break;
    }

    // To handle case if positive and negative
    // numbers both are odd in counts.
    if (j > i)
        sum = (sum + (arr[i] * arr[j]) % Mod) % Mod;

    // If one of them occurs odd times
    else if (i == j)
        sum = (sum + arr[i]) % Mod;
```

```
    return sum;
}

// Drivers code
int main()
{
    int arr[] = { -1, 9, 4, 5, -4, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findSum(arr, n);
    return 0;
}
```

Java

```
// Java program for above implementation
import java.io.*;
import java.util.*;

class GFG {

    static int Mod = 1000000007;

    // Function to find the maximum sum
    static long findSum(int arr[], int n) {
        long sum = 0;

        // Sort the array first
        Arrays.sort(arr);

        // First multiply negative numbers
        // pairwise and sum up from starting
        // as to get maximum sum.
        int i = 0;
        while (i < n && arr[i] < 0) {
            if (i != n - 1 && arr[i + 1] <= 0) {
                sum = (sum + (arr[i] * arr[i + 1]) % Mod) % Mod;
                i += 2;
            }
            else
                break;
        }

        // Second multiply positive numbers
        // pairwise and summed up from the
        // last as to get maximum sum.
        int j = n - 1;
        while (j >= 0 && arr[j] > 0) {
            if (j != 0 && arr[j - 1] > 0) {
```

```
        sum = (sum + (arr[j] * arr[j - 1]) % Mod) % Mod;
        j -= 2;
    } else
        break;
}

// To handle case if positive and negative
// numbers both are odd in counts.
if (j > i)
    sum = (sum + (arr[i] * arr[j])) % Mod;

// If one of them occurs odd times
else if (i == j)
    sum = (sum + arr[i]) % Mod;

return sum;
}

// Drivers code
public static void main(String args[]) {
    int arr[] = {-1, 9, 4, 5, -4, 7};
    int n = arr.length;
    System.out.println(findSum(arr, n));
}
}

/*This code is contributed by Nikita Tiwari.*/
```

Python3

```
# Python3 code for above implementation
Mod= 1000000007

# Function to find the maximum sum
def findSum(arr, n):
    sum = 0

    # Sort the array first
    arr.sort()

    # First multiply negative numbers
    # pairwise and sum up from starting
    # as to get maximum sum.
    i = 0
    while i < n and arr[i] < 0:
        if i != n - 1 and arr[i + 1] <= 0:
            sum = (sum + (arr[i] * arr[i + 1]))
                % Mod) % Mod
```

```
i += 2
else:
    break

# Second multiply positive numbers
# pairwise and summed up from the
# last as to get maximum sum.
j = n - 1
while j >= 0 and arr[j] > 0:
    if j != 0 and arr[j - 1] > 0:
        sum = (sum + (arr[j] * arr[j - 1])
               % Mod) % Mod
    j -= 2
else:
    break

# To handle case if positive
# and negative numbers both
# are odd in counts.
if j > i:
    sum = (sum + (arr[i] * arr[j])) % Mod
    % Mod

# If one of them occurs odd times
elif i == j:
    sum = (sum + arr[i]) % Mod

return sum

# Driver code
arr = [ -1, 9, 4, 5, -4, 7 ]
n = len(arr)
print(findSum(arr, n))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# program for above implementation
using System;

class GFG {

    static int Mod = 1000000007;

    // Function to find the maximum sum
    static long findSum(int[] arr, int n)
    {
```

```
long sum = 0;

// Sort the array first
Array.Sort(arr);

// First multiply negative numbers
// pairwise and sum up from starting
// as to get maximum sum.
int i = 0;
while (i < n && arr[i] < 0) {
    if (i != n - 1 && arr[i + 1] <= 0) {
        sum = (sum + (arr[i] * arr[i + 1]) % Mod) % Mod;
        i += 2;
    }
    else
        break;
}

// Second multiply positive numbers
// pairwise and summed up from the
// last as to get maximum sum.
int j = n - 1;
while (j >= 0 && arr[j] > 0) {
    if (j != 0 && arr[j - 1] > 0) {
        sum = (sum + (arr[j] * arr[j - 1]) % Mod) % Mod;
        j -= 2;
    }
    else
        break;
}

// To handle case if positive and negative
// numbers both are odd in counts.
if (j > i)
    sum = (sum + (arr[i] * arr[j]) % Mod) % Mod;

// If one of them occurs odd times
else if (i == j)
    sum = (sum + arr[i]) % Mod;

return sum;
}

// Drivers code
public static void Main()
{
    int[] arr = { -1, 9, 4, 5, -4, 7 };
    int n = arr.Length;
```

```
        Console.WriteLine(findSum(arr, n));
    }
}

/*This code is contributed by vt_m.*/
```

Output:

87

Source

<https://www.geeksforgeeks.org/maximum-sum-of-pairwise-product-in-an-array-with-negative-allowed/>

Chapter 184

Maximum triplet sum in array

Maximum triplet sum in array - GeeksforGeeks

Given an array, the task is to find maximum triplet sum in the array.

Examples :

```
Input : arr[] = {1, 2, 3, 0, -1, 8, 10}
Output : 21
10 + 8 + 3 = 21
```

```
Input : arr[] = {9, 8, 20, 3, 4, -1, 0}
Output : 37
20 + 9 + 8 = 37
```

Naive approach : In this method we simply run three loop and one by one add three element and compare with previous sum if the sum of three element is greater then store in previous sum .

C++

```
// C++ code to find maximum triplet sum
#include <bits/stdc++.h>
using namespace std;

int maxTripletSum(int arr[], int n)
{
    // Initialize sum with INT_MIN
    int sum = INT_MIN;

    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            for (int k = j + 1; k < n; k++)
```

```
        if (sum < arr[i] + arr[j] + arr[k])
            sum = arr[i] + arr[j] + arr[k];
    return sum;
}

// Driven code
int main()
{
    int arr[] = { 1, 0, 8, 6, 4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << maxTripletSum(arr, n);
    return 0;
}
```

Java

```
// Java code to find maximum triplet sum
import java.io.*;

class GFG {

    static int maxTripletSum(int arr[], int n)
    {
        // Initialize sum with INT_MIN
        int sum = -1000000;

        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                for (int k = j + 1; k < n; k++)
                    if (sum < arr[i] + arr[j] + arr[k])
                        sum = arr[i] + arr[j] + arr[k];
        return sum;
    }

    // Driven code
    public static void main(String args[])
    {
        int arr[] = { 1, 0, 8, 6, 4, 2 };
        int n = arr.length;
        System.out.println(maxTripletSum(arr, n));
    }
}

// This code is contributed by Nikita Tiwari.
```

Python3

```
# Python 3 code to find
```

```
# maximum triplet sum

def maxTripletSum(arr, n) :

    # Initialize sum with
    # INT_MIN
    sm = -1000000

    for i in range(0, n) :
        for j in range(i + 1, n) :
            for k in range(j + 1, n) :

                if (sm < (arr[i] + arr[j] + arr[k])) :
                    sm = arr[i] + arr[j] + arr[k]

    return sm

# Driven code
arr = [ 1, 0, 8, 6, 4, 2 ]
n = len(arr)

print(maxTripletSum(arr, n))

# This code is contributed by Nikita Tiwari.
```

C#

```
// C# code to find maximum triplet sum
using System;

class GFG {

    static int maxTripletSum(int[] arr, int n)
    {
        // Initialize sum with INT_MIN
        int sum = -1000000;

        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                for (int k = j + 1; k < n; k++)
                    if (sum < arr[i] + arr[j] + arr[k])
                        sum = arr[i] + arr[j] + arr[k];
        return sum;
    }

    // Driven code
    public static void Main()
    {
        int[] arr = { 1, 0, 8, 6, 4, 2 };
```

```
        int n = arr.Length;
        Console.WriteLine(maxTripletSum(arr, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP code to find maximum triplet sum

function maxTripletSum( $arr, $n)
{

    // Initialize sum with INT_MIN
    $sum = PHP_INT_MIN;

    for($i = 0; $i < $n; $i++)
        for($j = $i + 1; $j < $n; $j++)
            for($k = $j + 1; $k < $n; $k++)
                if ($sum < $arr[$i] +
                    $arr[$j] +
                    $arr[$k])

                    $sum = $arr[$i] +
                        $arr[$j] +
                        $arr[$k];

    return $sum;
}

// Driver Code
$arr = array(1, 0, 8, 6, 4, 2);
$n = count($arr);
echo maxTripletSum($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output :

18

Time complexity : $O(n^3)$
Space complexity : $O(1)$

Another approach : In this, we first need to sort the whole array and after that when we add last three element of the array then we find maximum sum of triplates.

C++

```
// C++ code to find maximum triplet sum
#include <bits/stdc++.h>
using namespace std;

// This function assumes that there are at least
// three elements in arr[].
int maxTripletSum(int arr[], int n)
{
    // sort the given array
    sort(arr, arr + n);

    // After sorting the array.
    // Add last three element of the given array
    return arr[n - 1] + arr[n - 2] + arr[n - 3];
}

// Driven code
int main()
{
    int arr[] = { 1, 0, 8, 6, 4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << maxTripletSum(arr, n);
    return 0;
}
```

Java

```
// Java code to find maximum triplet sum
import java.io.*;
import java.util.*;

class GFG {

    // This function assumes that there are
    // at least three elements in arr[].
    static int maxTripletSum(int arr[], int n)
    {
        // sort the given array
        Arrays.sort(arr);

        // After sorting the array.
        // Add last three element
```

```
// of the given array
return arr[n - 1] + arr[n - 2] + arr[n - 3];
}

// Driven code
public static void main(String args[])
{
    int arr[] = { 1, 0, 8, 6, 4, 2 };
    int n = arr.length;
    System.out.println(maxTripletSum(arr, n));
}
}
```

// This code is contributed by Nikita Tiwari.

Python3

```
# Python 3 code to find
# maximum triplet sum

# This function assumes
# that there are at least
# three elements in arr[] .
def maxTripletSum(arr, n) :

    # sort the given array
    arr.sort()

    # After sorting the array.
    # Add last three element
    # of the given array
    return (arr[n - 1] + arr[n - 2] + arr[n - 3])
```

```
# Driven code
arr = [ 1, 0, 8, 6, 4, 2 ]
n = len(arr)

print(maxTripletSum(arr, n))

# This code is contributed by Nikita Tiwari.
```

C#

```
// C# code to find maximum triplet sum
using System;
```

```
class GFG {

    // This function assumes that there are
    // at least three elements in arr[].
    static int maxTripletSum(int[] arr, int n)
    {
        // sort the given array
        Array.Sort(arr);

        // After sorting the array.
        // Add last three element
        // of the given array
        return arr[n - 1] + arr[n - 2] + arr[n - 3];
    }

    // Driven code
    public static void Main()
    {
        int[] arr = { 1, 0, 8, 6, 4, 2 };
        int n = arr.Length;
        Console.WriteLine(maxTripletSum(arr, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP code to find
// maximum triplet sum

// This function assumes that
// there are at least
// three elements in arr[].
function maxTripletSum( $arr, $n)
{
    // sort the given array
    sort($arr);

    // After sorting the array.
    // Add last three element
    // of the given array
    return $arr[$n - 1] + $arr[$n - 2] +
           $arr[$n - 3];
}
```

```
// Driver code
$arr = array( 1, 0, 8, 6, 4, 2 );
$n = count($arr);
echo maxTripletSum($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output :

18

Time complexity : O(nlogn)

Space complexity : O(1)

Efficient approach : Scan the array and compute Maximum, second maximum and third maximum element present in the array and return the sum of its and it would be maximum sum .

C++

```
// C++ code to find maximum triplet sum
#include <bits/stdc++.h>
using namespace std;

// This function assumes that there are at least
// three elements in arr[].
int maxTripletSum(int arr[], int n)
{
    // Initialize Maximum, second maximum and third
    // maximum element
    int maxA = INT_MIN, maxB = INT_MIN, maxC = INT_MIN;

    for (int i = 0; i < n; i++) {

        // Update Maximum, second maximum and third
        // maximum element
        if (arr[i] > maxA) {
            maxC = maxB;
            maxB = maxA;
            maxA = arr[i];
        }

        // Update second maximum and third maximum
        // element
        else if (arr[i] > maxB) {
            maxC = maxB;
            maxB = arr[i];
        }
    }

    // Return sum of maximum, second maximum and third maximum
    return maxA + maxB + maxC;
}
```

```
    maxB = arr[i];
}

// Update third maximum element
else if (arr[i] > maxC)
    maxC = arr[i];
}

return (maxA + maxB + maxC);
}

// Driven code
int main()
{
    int arr[] = { 1, 0, 8, 6, 4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << maxTripletSum(arr, n);
    return 0;
}
```

Java

```
// Java code to find maximum triplet sum
import java.io.*;
import java.util.*;

class GFG {

    // This function assumes that there
    // are at least three elements in arr[].
    static int maxTripletSum(int arr[], int n)
    {
        // Initialize Maximum, second maximum and third
        // maximum element
        int maxA = -1000000000, maxB = -1000000000;
        int maxC = -1000000000;

        for (int i = 0; i < n; i++) {

            // Update Maximum, second maximum
            // and third maximum element
            if (arr[i] > maxA)
            {
                maxC = maxB;
                maxB = maxA;
                maxA = arr[i];
            }
        }
    }
}
```

```
// Update second maximum and third maximum
// element
else if (arr[i] > maxB)
{
    maxC = maxB;
    maxB = arr[i];
}

// Update third maximum element
else if (arr[i] > maxC)
    maxC = arr[i];
}

return (maxA + maxB + maxC);
}

// Driven code
public static void main(String args[])
{
    int arr[] = { 1, 0, 8, 6, 4, 2 };
    int n = arr.length;
    System.out.println(maxTripletSum(arr, n));
}
}

// This code is contributed by Nikita Tiwari.
```

Python3

```
# Python 3 code to find
# maximum triplet sum

# This function assumes
# that there are at least
# three elements in arr[] .
def maxTripletSum(arr, n) :

    # Initialize Maximum, second
    # maximum and third maximum
    # element
    maxA = -100000000
    maxB = -100000000
    maxC = -100000000

    for i in range(0, n) :
```

```
# Update Maximum, second maximum
# and third maximum element
if (arr[i] > maxA) :
    maxC = maxB
    maxB = maxA
    maxA = arr[i]

# Update second maximum and
# third maximum element
elif (arr[i] > maxB) :
    maxC = maxB
    maxB = arr[i]

# Update third maximum element
elif (arr[i] > maxC) :
    maxC = arr[i]

return (maxA + maxB + maxC)

# Driven code
arr = [ 1, 0, 8, 6, 4, 2 ]
n = len(arr)

print(maxTripletSum(arr, n))

# This code is contributed by Nikita Tiwari.
```

C#

```
// C# code to find maximum triplet sum
using System;

class GFG {

    // This function assumes that there
    // are at least three elements in arr[].
    static int maxTripletSum(int[] arr, int n)
    {
        // Initialize Maximum, second maximum
        // and third maximum element
        int maxA = -100000000, maxB = -100000000;
        int maxC = -100000000;

        for (int i = 0; i < n; i++) {

            // Update Maximum, second maximum
            // and third maximum element
            if (arr[i] > maxA) {
```

```
        maxC = maxB;
        maxB = maxA;
        maxA = arr[i];
    }

    // Update second maximum and third
    // maximum element
    else if (arr[i] > maxB) {
        maxC = maxB;
        maxB = arr[i];
    }

    // Update third maximum element
    else if (arr[i] > maxC)
        maxC = arr[i];
}

return (maxA + maxB + maxC);
}

// Driven code
public static void Main()
{
    int[] arr = { 1, 0, 8, 6, 4, 2 };
    int n = arr.Length;
    Console.WriteLine(maxTripletSum(arr, n));
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP code to find
// maximum triplet sum

// This function assumes that
// there are at least three
// elements in arr[].
function maxTripletSum($arr, $n)
{
    // Initialize Maximum,
    // second maximum and
    // third maximum element
    $maxA = PHP_INT_MIN;
    $maxB = PHP_INT_MIN;
    $maxC = PHP_INT_MIN;
```

```
for ( $i = 0; $i < $n; $i++)
{
    // Update Maximum,
    // second maximum and
    // third maximum element
    if ($arr[$i] > $maxA)
    {
        $maxC = $maxB;
        $maxB = $maxA;
        $maxA = $arr[$i];
    }

    // Update second maximum and
    // third maximum element
    else if ($arr[$i] > $maxB)
    {
        $maxC = $maxB;
        $maxB = $arr[$i];
    }

    // Update third maximum element
    else if ($arr[$i] > $maxC)
        $maxC = $arr[$i];
}

return ($maxA + $maxB + $maxC);
}

// Driven code
$arr = array( 1, 0, 8, 6, 4, 2 );
$n = count($arr);
echo maxTripletSum($arr, $n);

// This code is contributed by anuj_67.
?>
```

Output :

18

Time complexity : O(n)
Space complexity : O(1)

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/maximum-triplet-sum-array/>

Chapter 185

Median after K additional integers

Median after K additional integers - GeeksforGeeks

Given an array of n integers. We are allowed to add k additional integer in the array and then find the median of the resultant array. We can choose any k values to be added.

Constraints:

k < n
n + k is always odd.

Examples :

Input : arr[] = { 4, 7 }
k = 1
Output : 7
Explanation : One of the possible solutions
is to add 8 making the array [4, 7, 8], whose
median is 7

Input : arr[] = { 6, 1, 1, 1, 1 }
k = 2
Output : 1
Explanation : No matter what elements we add
to this array, the median will always be 1

We first sort the array in increasing order. Since value of k is less than n and n+k is always odd, we can always choose to add k elements which are greater than the largest element of array and $(n+k)/2$ th element is always a median of the array.

C++

```
// CPP program to find median of an array when
// we are allowed to add additional K integers
// to it.
#include <bits/stdc++.h>
using namespace std;

// Find median of array after adding k elements
void printMedian(int arr[], int n, int K)
{
    // sorting the array in increasing order.
    sort(arr, arr + n);

    // printing the median of array.
    // Since n + K is always odd and K < n,
    // so median of array always lies in
    // the range of n.
    cout << arr[(n + K) / 2];
}

// driver function
int main()
{
    int arr[] = { 5, 3, 2, 8 };
    int k = 3;
    int n = sizeof(arr) / sizeof(arr[0]);
    printMedian(arr, n, k);
    return 0;
}
```

Java

```
// Java program to find median of an array when
// we are allowed to add additional K integers
// to it.
import java.util.Arrays;

class GFG {

    // Find median of array after adding k elements
    static void printMedian(int arr[], int n, int K)
    {

        // sorting the array in increasing order.
        Arrays.sort(arr);

        // printing the median of array.
        // Since n + K is always odd and K < n,
        // so median of array always lies in
```

```
// the range of n.  
System.out.print(arr[(n + K) / 2]);  
}  
  
//Driver code  
public static void main (String[] args)  
{  
  
    int arr[] = { 5, 3, 2, 8 };  
    int k = 3;  
    int n = arr.length;  
  
    printMedian(arr, n, k);  
}  
}  
  
// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 code to find median of an  
# array when we are allowed to add  
# additional K integers to it.  
  
# Find median of array after  
# adding k elements  
def printMedian (arr, n, K):  
  
    # sorting the array in  
    # increasing order.  
    arr.sort()  
  
    # printing the median of array.  
    # Since n + K is always odd  
    # and K < n, so median of  
    # array always lies in  
    # the range of n.  
    print( arr[int((n + K) / 2)])  
  
# driver function  
arr = [ 5, 3, 2, 8 ]  
k = 3  
n = len(arr)  
printMedian(arr, n, k)  
  
# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# program to find median of an array when
// we are allowed to add additional K integers
// to it.
using System;

class GFG
{
    // Find median of array after adding k elements
    static void printMedian(int []arr, int n, int K)
    {
        // sorting the array in increasing order.
        Array.Sort(arr);

        // printing the median of array.
        // Since n + K is always odd and K < n,
        // so median of array always lies in
        // the range of n.
        Console.WriteLine(arr[(n + K) / 2]);
    }

    //Driver code
    public static void Main ()
    {
        int []arr = { 5, 3, 2, 8 };
        int k = 3;
        int n = arr.Length;
        printMedian(arr, n, k);
    }
}

// This code is contributed by anant321.
```

PHP

```
<?php
// PHP program to find median
// of an array when we are allowed
// to add additional K integers to it.

// Find median of array
// after adding k elements
function printMedian($arr, $n, $K)
{
    // sorting the array
    // in increasing order.
    sort($arr);

    // printing the median of
```

```
// array. Since n + K is
// always odd and K < n,
// so median of array always
// lies in the range of n.
echo $arr[($n + $K) / 2];
}

// Driver Code
$arr = array( 5, 3, 2, 8 );
$k = 3;
$n = count($arr);
printMedian($arr, $n, $k);

// This code is contributed by Sam007
?>
```

Output :

8

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/median-k-additional-integers/>

Chapter 186

Median and Mode using Counting Sort

Median and Mode using Counting Sort - GeeksforGeeks

Given an n sized unsorted array, find median and mode using **counting sort** technique. This can be useful when array elements are in limited range.

Examples:

```
Input : array a[] = {1, 1, 1, 2, 7, 1}
Output : Mode = 1
         Median = 1.5
```

```
Input : array a[] = {9, 9, 9, 9, 9}
Output : Mode = 9
         Median = 9
```

Prerequisites: [Count Sort](#), [Median of Array](#), [Mode \(Most frequent element in array\)](#)

1. Auxiliary(count) array before summing its previous counts, $c[]$:

Index: 0 1 2 3 4 5 6 7 8 9 10
count: 0 4 1 0 0 0 0 1 0 0 0

2. Mode = index with maximum value of count.
Mode = 1(for above example)

3. count array is modified similarly as it is done while performing count sort.
Index: 0 1 2 3 4 5 6 7 8 9 10
count: 0 3 5 6 7 8 9 10 10 10 10

4. output array is calculated normally as in count sort, $b[]$:
output array $b[] = \{1, 1, 1, 2, 2, 3, 4, 5, 6, 7\}$

5. If size of array $b[]$ is odd, Median = $b[n/2]$
Else Median = $(b[(n-1)/2] + b[n/2])/2$
6. For above example size of $b[]$ is even hence, Median = $(b[4] + b[5])/2$.
Median = $(2 + 3)/2 = 2.5$

Basic Approach to be followed :

Assuming size of input array is n :

Step1: Take the count array before summing its previous counts into next index.

Step2: The index with maximum value stored in it is the mode of given data.

Step3: In case there are more than one indexes with maximum value in it, all are results for mode so we can take any.

Step4: Store the value at that index in a separate variable called mode.

Step5: Continue with the normal processing of the count sort.

Step6: In the resultant(sorted) array, if n is odd then median = middle-most element of the

sorted array, And if n is even the median = average of two middle-most elements of the sorted array.

Step7: Store the result in a separate variable called median.

Following is the implementation of problem discussed above:

Time Complexity = $O(N + P)$, where N is the time for input array and P is time for count array.

Space Complexity = $O(P)$, where P is the size of auxiliary array.

Source

<https://www.geeksforgeeks.org/median-and-mode-using-counting-sort/>

Chapter 187

Merge 3 Sorted Arrays

Merge 3 Sorted Arrays - GeeksforGeeks

Given 3 arrays (A, B, C) which are sorted in ascending order, we are required to merge them together in ascending order and output the array D.

Examples:

Input : A = [1, 2, 3, 4, 5]
B = [2, 3, 4]
C = [4, 5, 6, 7]
Output : D = [1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 6, 7]

Input : A = [1, 2, 3, 5]
B = [6, 7, 8, 9]
C = [10, 11, 12]
Output: D = [1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12]

Method 1 (Two Arrays at a time)

We have discussed at [Merging 2 Sorted arrays](#). So we can first merge two arrays and then merge the resultant with the third array. Time Complexity for merging two arrays $O(m+n)$. So for merging the third array, the time complexity will become $O(m+n+o)$. Note that this is indeed the best time complexity that can be achieved for this problem.

Space Complexity: Since we merge two arrays at a time, we need another array to store the result of the first merge. This raises the space complexity to $O(m+n)$. Note that space required to hold the result of 3 arrays is ignored while calculating complexity.

Algorithm

```
function merge(A, B)
    Let m and n be the sizes of A and B
```

Let D be the array to store result

```
// Merge by taking smaller element from A and B
while i < m and j < n
    if A[i] <= B[j]
        Add A[i] to D and increment i by 1
    else Add B[j] to D and increment j by 1

// If array A has exhausted, put elements from B
while j < n
    Add B[j] to D and increment j by 1

// If array B has exhausted, put elements from A
while i < n
    Add A[i] to D and increment i by 1

Return D

function merge_three(A, B, C)
    T = merge(A, B)
    return merge(T, C)
```

The C++ and Python Implementation is given below

CPP

```
// C++ program to merge three sorted arrays
// by merging two at a time.
#include <iostream>
#include <vector>
using namespace std;

using Vector = vector<int>;

void printVector(const Vector& a)
{
    cout << "[";
    for (auto e : a)
        cout << e << " ";
    cout << "]" << endl;
}

Vector mergeTwo(Vector& A, Vector& B)
{
    // Get sizes of vectors
    int m = A.size();
    int n = B.size();
```

```
// Vector for storing Result
Vector D;
D.reserve(m + n);

int i = 0, j = 0;
while (i < m && j < n) {

    if (A[i] <= B[j])
        D.push_back(A[i++]);
    else
        D.push_back(B[j++]);
}

// B has exhausted
while (i < m)
    D.push_back(A[i++]);

// A has exhausted
while (j < n)
    D.push_back(B[j++]);

return D;
}

int main()
{
    Vector A = { 1, 2, 3, 5 };
    Vector B = { 6, 7, 8, 9 };
    Vector C = { 10, 11, 12 };

    // First Merge A and B
    Vector T = mergeTwo(A, B);

    // Print Result after merging T with C
    printVector(mergeTwo(T, C));
    return 0;
}
```

Python

```
# Python program to merge three sorted arrays
# by merging two at a time.

def merge_two(a, b):
    (m, n) = (len(a), len(b))
    i = j = 0

    # Destination Array
```

```
d = []

# Merge from a and b together
while i < m and j < n:
    if a[i] <= b[j]:
        d.append(a[i])
        i += 1
    else:
        d.append(b[j])
        j += 1

# Merge from a if b has run out
while i < m:
    d.append(a[i])
    i += 1

# Merge from b if a has run out
while j < n:
    d.append(b[j])
    j += 1

return d

def merge(a, b, c):
    t = merge_two(a, b)
    return merge_two(t, c)

if __name__ == "__main__":
    A = [1, 2, 3, 5]
    B = [6, 7, 8, 9]
    C = [10, 11, 12]
    print(merge(A, B, C))
```

Output:

```
[1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12]
```

Method 2 (Three arrays at a time)

The Space complexity of method 1 can be improved we merge the three arrays together.

```
function merge-three(A, B, C)
    Let m, n, o be size of A, B, and C
    Let D be the array to store the result

    // Merge three arrays at the same time
```

```

while i < m and j < n and k < o
    Get minimum of A[i] , B[j] , C[i]
    if the minimum is from A, add it to
        D and advance i
    else if the minimum is from B add it
        to D and advance j
    else if the minimum is from C add it
        to D and advance k

    // After above step at least 1 array has
    // exhausted. Only C has exhausted
    while i < m and j < n
        put minimum of A[i] and B[j] into D
        Advance i if minimum is from A else advance j

    // Only B has exhausted
    while i < m and k < o
        Put minimum of A[i] and C[k] into D
        Advance i if minimum is from A else advance k

    // Only A has exhausted
    while j < n and k < o
        Put minimum of B[j] and C[k] into D
        Advance j if minimum is from B else advance k

    // After above steps at least 2 arrays have
    // exhausted
    if A and B have exhausted take elements from C
    if B and C have exhausted take elements from A
    if A and C have exhausted take elements from B

    return D

```

Complexity: The Time Complexity is $O(m+n+o)$ since we process each element from the three arrays once. We only need one array to store the result of merging and so ignoring this array, the space complexity is $O(1)$.

The C++ and Python Implementation of the algorithm is given below:

C++

```

// C++ program to merger three sorted arrays
// by merging three simultaneously.
#include <iostream>
#include <vector>
using namespace std;

using Vector = vector<int>;

```

```
void printVector(const Vector& a)
{
    cout << "[";
    for (auto e : a) {
        cout << e << " ";
    }
    cout << "]" << endl;
}

Vector mergeThree(Vector& A, Vector& B,
                  Vector& C)
{
    int m, n, o, i, j, k;
    // Get Sizes of three vectors
    m = A.size();
    n = B.size();
    o = C.size();

    // Vector for storing output
    Vector D;
    D.reserve(m + n + o);

    i = j = k = 0;

    while (i < m && j < n && k < o) {

        // Get minimum of a, b, c
        int m = min(min(A[i], B[j]), C[k]);

        // Put m in D
        D.push_back(m);

        // Increment i, j, k
        if (m == A[i])
            i++;
        else if (m == B[j])
            j++;
        else
            k++;
    }

    // C has exhausted
    while (i < m && j < n) {
        if (A[i] <= B[j]) {
            D.push_back(A[i]);
            i++;
        }
    }
    else {
```

```

        D.push_back(B[j]);
        j++;
    }
}

// B has exhausted
while (i < m && k < n) {
    if (A[i] <= C[j]) {
        D.push_back(A[i]);
        i++;
    }
    else {
        D.push_back(C[j]);
        k++;
    }
}

// A has exhausted
while (j < n && k < o) {
    if (B[j] <= C[k]) {
        D.push_back(B[j]);
        j++;
    }
    else {
        D.push_back(C[j]);
        k++;
    }
}

// A and B have exhausted
while (k < o)
    D.push_back(C[k++]);

// B and C have exhausted
while (i < m)
    D.push_back(A[i++]);

// A and C have exhausted
while (j < n)
    D.push_back(B[j++]);

return D;
}

int main()
{
    Vector A = { 1, 2, 41, 52, 84 };
    Vector B = { 1, 2, 41, 52, 67 };
}

```

```
Vector C = { 1, 2, 41, 52, 67, 85 };

// Print Result
printVector(mergeThree(A, B, C));
return 0;
}
```

Python

```
# Python program to merge three sorted arrays
# simultaneously.

def merge_three(a, b, c):
    (m, n, o) = (len(a), len(b), len(c))
    i = j = k = 0

    # Destination array
    d = []

    while i < m and j < n and k < o:

        # Get Minimum element
        m = min(a[i], b[j], c[k])

        # Add m to D
        d.append(m)

        # Increment the source pointer which
        # gives m
        if a[i] == m:
            i += 1
        elif b[j] == m:
            j += 1
        elif c[k] == m:
            k += 1

    # Merge a and b in c has exhausted
    while i < m and j < n:
        if a[i] <= b[j]:
            d.append(a[i])
            i += 1
        else:
            d.append(b[j])
            j += 1

    # Merge b and c if a has exhausted
    while j < n and k < o:
        if b[j] <= c[k]:
```

```
d.append(b[j])
j += 1
else:
    d.append(c[k])
    k += 1

# Merge a and c if b has exhausted
while i < m and k < o:
    if a[i] <= c[k]:
        d.append(a[i])
        i += 1
    else:
        d.append(c[k])
        k += 1

# Take elements from a if b and c
# have exhausted
while i < m:
    d.append(a[i])
    i += 1

# Take elements from b if a and c
# have exhausted
while j < n:
    d.append(b[j])
    j += 1

# Take elements from c if a and
# b have exhausted
while k < o:
    d.append(c[k])
    k += 1

return d

if __name__ == "__main__":
    a = [1, 2, 41, 52, 84]
    b = [1, 2, 41, 52, 67]
    c = [1, 2, 41, 52, 67, 85]

    print(merge_three(a, b, c))
```

Output

```
[1, 1, 1, 2, 2, 2, 41, 41, 41, 52, 52, 52, 67, 67, 84, 85]
```

Note: While it is relatively easy to implement direct procedures to merge two or three

arrays, the process becomes cumbersome if we want to merge 4 or more arrays. In such cases, we should follow the procedure shown in [Merge K Sorted Arrays](#).

Source

<https://www.geeksforgeeks.org/merge-3-sorted-arrays/>

Chapter 188

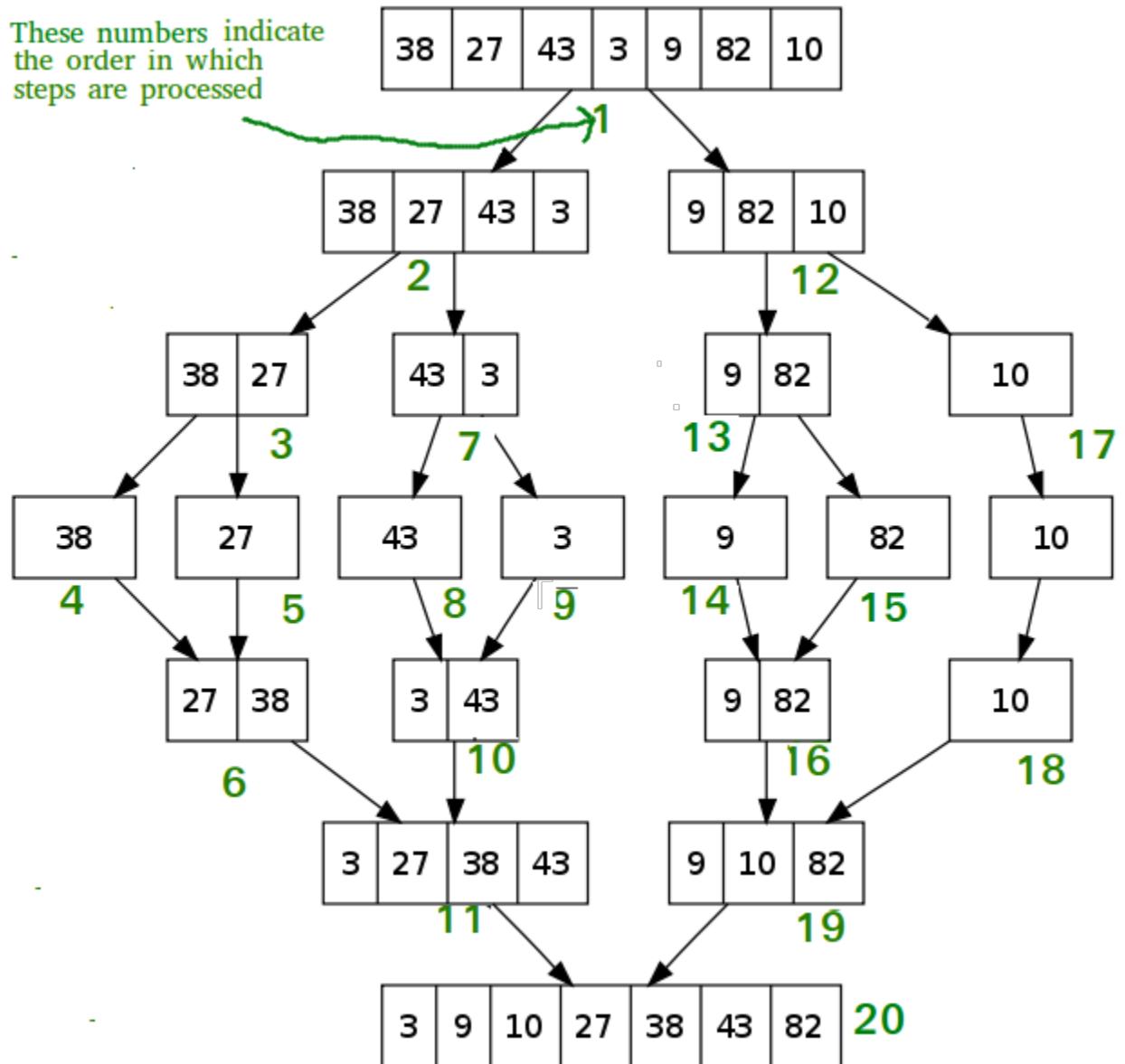
Merge Sort

Merge Sort - GeeksforGeeks

Like [QuickSort](#), Merge Sort is a [Divide and Conquer](#) algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one. See following C implementation for details.

```
MergeSort(arr[], l, r)
If r > l
    1. Find the middle point to divide the array into two halves:
        middle m = (l+r)/2
    2. Call mergeSort for first half:
        Call mergeSort(arr, l, m)
    3. Call mergeSort for second half:
        Call mergeSort(arr, m+1, r)
    4. Merge the two halves sorted in step 2 and 3:
        Call merge(arr, l, m, r)
```

The following diagram from [wikipedia](#) shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}. If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge processes comes into action and starts merging arrays back till the complete array is merged.



C/C++

```
/* C program for Merge Sort */
#include<stdlib.h>
#include<stdio.h>

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
```

```

{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
       are any */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
       are any */
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
    }
}

```

```
        k++;
    }
}

/* l is for left index and r is right index of the
   sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}

/* UTILITY FUNCTIONS */
/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

Java

```
/* Java program for Merge Sort */
class MergeSort
{
    // Merges two subarrays of arr[].
    // First subarray is arr[1..m]
    // Second subarray is arr[m+1..r]
    void merge(int arr[], int l, int m, int r)
    {
        // Find sizes of two subarrays to be merged
        int n1 = m - l + 1;
        int n2 = r - m;

        /* Create temp arrays */
        int L[] = new int [n1];
        int R[] = new int [n2];

        /*Copy data to temp arrays*/
        for (int i=0; i<n1; ++i)
            L[i] = arr[l + i];
        for (int j=0; j<n2; ++j)
            R[j] = arr[m + 1+ j];

        /* Merge the temp arrays */

        // Initial indexes of first and second subarrays
        int i = 0, j = 0;

        // Initial index of merged subarry array
        int k = l;
        while (i < n1 && j < n2)
        {
            if (L[i] <= R[j])
            {
                arr[k] = L[i];
                i++;
            }
            else
            {
                arr[k] = R[j];
                j++;
            }
            k++;
        }

        /* Copy remaining elements of L[] if any */
    }
}
```

```
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy remaining elements of R[] if any */
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}
}

// Main function that sorts arr[l..r] using
// merge()
void sort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Find the middle point
        int m = (l+r)/2;

        // Sort first and second halves
        sort(arr, l, m);
        sort(arr , m+1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

/* A utility function to print array of size n */
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

// Driver method
public static void main(String args[])
{
    int arr[] = {12, 11, 13, 5, 6, 7};
```

```
System.out.println("Given Array");
printArray(arr);

MergeSort ob = new MergeSort();
ob.sort(arr, 0, arr.length-1);

System.out.println("\nSorted array");
printArray(arr);
}

/*
 * This code is contributed by Rajat Mishra */

```

Python

```
# Python program for implementation of MergeSort

# Merges two subarrays of arr[].
# First subarray is arr[l..m]
# Second subarray is arr[m+1..r]
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m

    # create temp arrays
    L = [0] * (n1)
    R = [0] * (n2)

    # Copy data to temp arrays L[] and R[]
    for i in range(0, n1):
        L[i] = arr[l + i]

    for j in range(0, n2):
        R[j] = arr[m + 1 + j]

    # Merge the temp arrays back into arr[l..r]
    i = 0      # Initial index of first subarray
    j = 0      # Initial index of second subarray
    k = l      # Initial index of merged subarray

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1
```

```

# Copy the remaining elements of L[], if there
# are any
while i < n1:
    arr[k] = L[i]
    i += 1
    k += 1

# Copy the remaining elements of R[], if there
# are any
while j < n2:
    arr[k] = R[j]
    j += 1
    k += 1

# l is for left index and r is right index of the
# sub-array of arr to be sorted
def mergeSort(arr,l,r):
    if l < r:

        # Same as (l+r)/2, but avoids overflow for
        # large l and h
        m = (l+(r-1))/2

        # Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)

# Driver code to test above
arr = [12, 11, 13, 5, 6, 7]
n = len(arr)
print ("Given array is")
for i in range(n):
    print ("%d" %arr[i]),

mergeSort(arr,0,n-1)
print ("\n\nSorted array is")
for i in range(n):
    print ("%d" %arr[i]),

# This code is contributed by Mohit Kumra
C#
// C# program for Merge Sort
using System;

```

```
class MergeSort {

    // Merges two subarrays of arr[].
    // First subarray is arr[l..m]
    // Second subarray is arr[m+1..r]
    void merge(int[] arr, int l,
               int m, int r)
    {

        // Find sizes of two subarrays
        // to be merged
        int n1 = m - l + 1;
        int n2 = r - m;

        // Create temp arrays
        int[] L = new int [n1];
        int[] R = new int [n2];

        // Copy data to temp arrays
        int i, j;
        for (i = 0; i < n1; ++i)
            L[i] = arr[l + i];

        for (j = 0; j < n2; ++j)
            R[j] = arr[m + 1+ j];

        // Merge the temp arrays

        // Initial indexes of first
        // and second subarrays
        i = 0;
        j = 0;

        // Initial index of merged
        // subarry array
        int k = l;
        while (i < n1 && j < n2)
        {
            if (L[i] <= R[j])
            {
                arr[k] = L[i];
                i++;
            }

            else
            {
                arr[k] = R[j];
                j++;
            }
            k++;
        }
    }
}
```

```
        j++;
    }
    k++;
}

// Copy remaining elements
// of L[] if any
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

// Copy remaining elements
// of R[] if any
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}
}

// Main function that sorts
// arr[l..r] using merge()
void sort(int[] arr, int l, int r)
{
    if (l < r)
    {
        // Find the middle point
        int m = (l + r) / 2;

        // Sort first and
        // second halves
        sort(arr, l, m);
        sort(arr, m + 1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

// A utility function to print
// array of size n
static void printArray(int[] arr)
{
    int n = arr.Length;
```

```

        for (int i = 0; i < n; ++i)
            Console.Write(arr[i] + " ");

        Console.WriteLine("\n");
    }

    // Driver Code
    public static void Main()
    {
        int[] arr = {12, 11, 13, 5, 6, 7};

        Console.WriteLine("Given Array\n");
        printArray(arr);

        MergeSort ob = new MergeSort();
        ob.sort(arr, 0, arr.Length - 1);

        Console.WriteLine("\nSorted array\n");
        printArray(arr);
    }
}

// This code is contributed by ChitraNayal.

```

Output:

Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13

Time Complexity: Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.

$$T(n) = 2T(n/2) + \Theta(n)$$

The above recurrence can be solved either using Recurrence Tree method or Master method.

It falls in case II of Master Method and solution of the recurrence is $\Theta(n \log n)$.

Time complexity of Merge Sort is $\Theta(n \log n)$ in all 3 cases (worst, average and best) as merge sort always divides the array in two halves and take linear time to merge two halves.

Auxiliary Space: $O(n)$

Algorithmic Paradigm: Divide and Conquer

Sorting In Place: No in a typical implementation

Stable: Yes

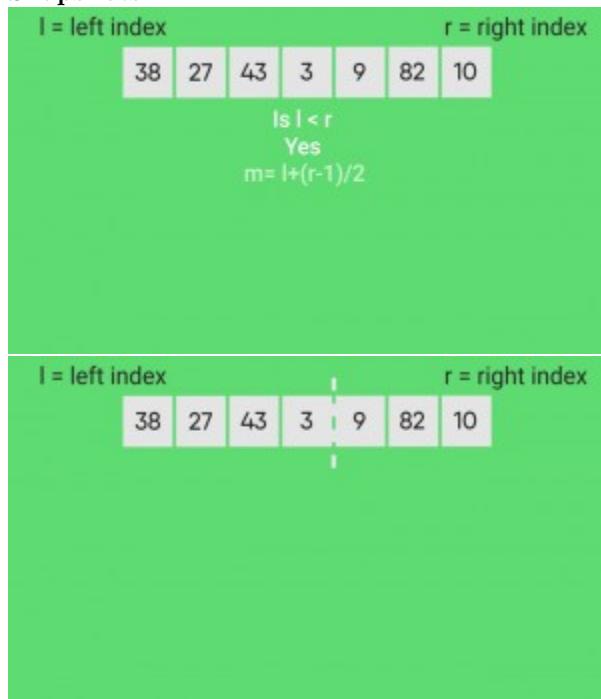
Applications of Merge Sort

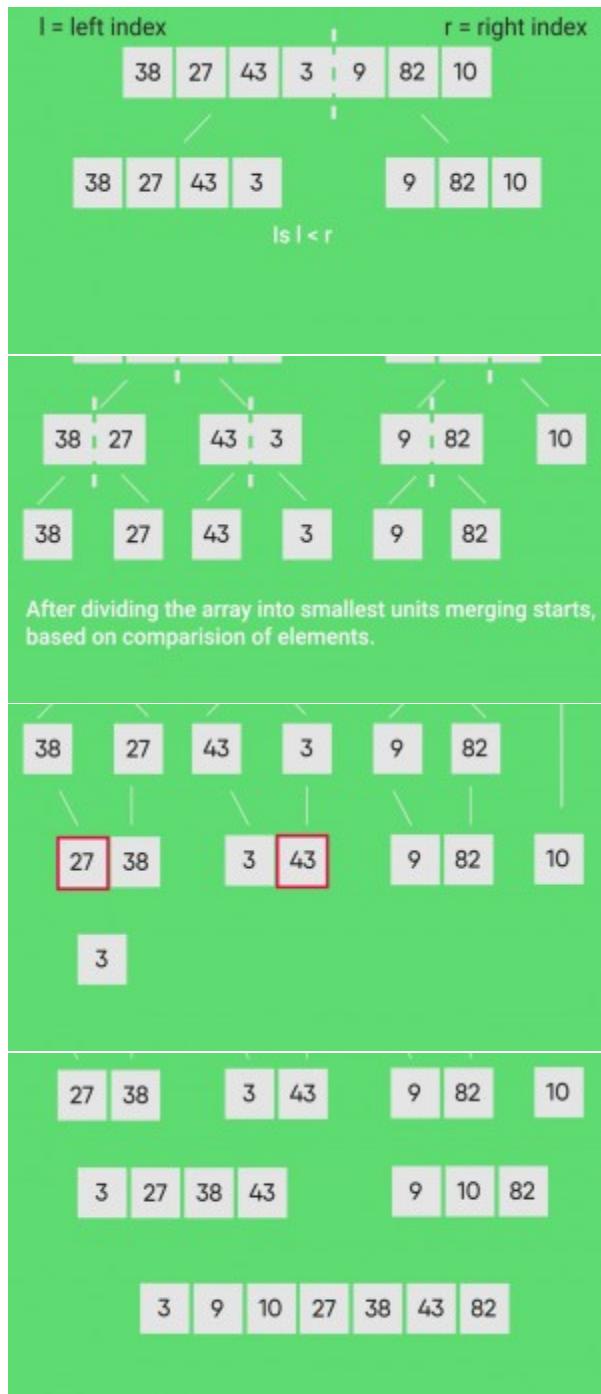
1. Merge Sort is useful for sorting linked lists in $O(n\log n)$ time. In case of linked lists the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike array, in linked list, we can insert items in the middle in $O(1)$ extra space and $O(1)$ time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.

In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of $A[0]$ be x then to access $A[i]$, we can directly access the memory at $(x + i*4)$. Unlike arrays, we can not do random access in linked list. Quick Sort requires a lot of this kind of access. In linked list to access i 'th index, we have to travel each and every node from the head to i 'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

2. Inversion Count Problem
3. Used in External Sorting

Snapshots:





- Recent Articles on Merge Sort
- Coding practice for sorting.

- Quiz on Merge Sort

Other Sorting Algorithms on GeeksforGeeks:

3-way Merge Sort, Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket Sort, ShellSort, Comb Sort

Improved By : ChitraNayal

Source

<https://www.geeksforgeeks.org/merge-sort/>

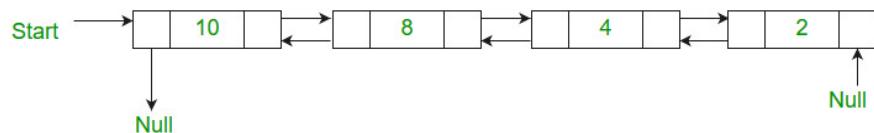
Chapter 189

Merge Sort for Doubly Linked List

Merge Sort for Doubly Linked List - GeeksforGeeks

Given a doubly linked list, write a function to sort the doubly linked list in increasing order using merge sort.

For example, the following doubly linked list should be changed to 24810



[Merge sort for singly linked list](#) is already discussed. The important change here is to modify the previous pointers also when merging two lists.

Below is the implementation of merge sort for doubly linked list.

C

```
// C program for merge sort on doubly linked list
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int data;
    struct Node *next, *prev;
};

struct Node *split(struct Node *head);

// Function to merge two linked lists
struct Node *merge(struct Node *first, struct Node *second)
```

```
{  
    // If first linked list is empty  
    if (!first)  
        return second;  
  
    // If second linked list is empty  
    if (!second)  
        return first;  
  
    // Pick the smaller value  
    if (first->data < second->data)  
    {  
        first->next = merge(first->next,second);  
        first->next->prev = first;  
        first->prev = NULL;  
        return first;  
    }  
    else  
    {  
        second->next = merge(first,second->next);  
        second->next->prev = second;  
        second->prev = NULL;  
        return second;  
    }  
}  
  
// Function to do merge sort  
struct Node *mergeSort(struct Node *head)  
{  
    if (!head || !head->next)  
        return head;  
    struct Node *second = split(head);  
  
    // Recur for left and right halves  
    head = mergeSort(head);  
    second = mergeSort(second);  
  
    // Merge the two sorted halves  
    return merge(head,second);  
}  
  
// A utility function to insert a new node at the  
// beginning of doubly linked list  
void insert(struct Node **head, int data)  
{  
    struct Node *temp =  
        (struct Node *)malloc(sizeof(struct Node));  
    temp->data = data;
```

```
temp->next = temp->prev = NULL;
if (!(*head))
    (*head) = temp;
else
{
    temp->next = *head;
    (*head)->prev = temp;
    (*head) = temp;
}
}

// A utility function to print a doubly linked list in
// both forward and backward directions
void print(struct Node *head)
{
    struct Node *temp = head;
    printf("Forward Traversal using next pointer\n");
    while (head)
    {
        printf("%d ", head->data);
        temp = head;
        head = head->next;
    }
    printf("\nBackward Traversal using prev pointer\n");
    while (temp)
    {
        printf("%d ", temp->data);
        temp = temp->prev;
    }
}

// Utility function to swap two integers
void swap(int *A, int *B)
{
    int temp = *A;
    *A = *B;
    *B = temp;
}

// Split a doubly linked list (DLL) into 2 DLLs of
// half sizes
struct Node *split(struct Node *head)
{
    struct Node *fast = head,*slow = head;
    while (fast->next && fast->next->next)
    {
        fast = fast->next->next;
        slow = slow->next;
```

```
}

struct Node *temp = slow->next;
slow->next = NULL;
return temp;
}

// Driver program
int main(void)
{
    struct Node *head = NULL;
    insert(&head,5);
    insert(&head,20);
    insert(&head,4);
    insert(&head,3);
    insert(&head,30);
    insert(&head,10);
    head = mergeSort(head);
    printf("\n\nLinked List after sorting\n");
    print(head);
    return 0;
}
```

Java

```
// Java program to implement merge sort in singly linked list

// Linked List Class
class LinkedList {

    static Node head; // head of list

    /* Node Class */
    static class Node {

        int data;
        Node next, prev;

        // Constructor to create a new node
        Node(int d) {
            data = d;
            next = prev = null;
        }
    }

    void print(Node node) {
        Node temp = node;
        System.out.println("Forward Traversal using next pointer");
        while (node != null) {
```

```
        System.out.print(node.data + " ");
        temp = node;
        node = node.next;
    }
    System.out.println("\nBackward Traversal using prev pointer");
    while (temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.prev;
    }
}

// Split a doubly linked list (DLL) into 2 DLLs of
// half sizes
Node split(Node head) {
    Node fast = head, slow = head;
    while (fast.next != null && fast.next.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }
    Node temp = slow.next;
    slow.next = null;
    return temp;
}

Node mergeSort(Node node) {
    if (node == null || node.next == null) {
        return node;
    }
    Node second = split(node);

    // Recur for left and right halves
    node = mergeSort(node);
    second = mergeSort(second);

    // Merge the two sorted halves
    return merge(node, second);
}

// Function to merge two linked lists
Node merge(Node first, Node second) {
    // If first linked list is empty
    if (first == null) {
        return second;
    }

    // If second linked list is empty
    if (second == null) {
        return first;
```

```
}

// Pick the smaller value
if (first.data < second.data) {
    first.next = merge(first.next, second);
    first.next.prev = first;
    first.prev = null;
    return first;
} else {
    second.next = merge(first, second.next);
    second.next.prev = second;
    second.prev = null;
    return second;
}
}

// Driver program to test above functions
public static void main(String[] args) {

    LinkedList list = new LinkedList();
    list.head = new Node(10);
    list.head.next = new Node(30);
    list.head.next.next = new Node(3);
    list.head.next.next.next = new Node(4);
    list.head.next.next.next.next = new Node(20);
    list.head.next.next.next.next.next = new Node(5);

    Node node = null;
    node = list.mergeSort(head);
    System.out.println("Linked list after sorting :");
    list.print(node);

}

}

// This code has been contributed by Mayank Jaiswal
```

Python

```
# Program for merge sort on doubly linked list

# A node of the doubly linked list
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
```

```
        self.next = None
        self.prev = None

class DoublyLinkedList:

    # Constructor for empty Doubly Linked List
    def __init__(self):
        self.head = None

    # Function to merge two linked list
    def merge(self, first, second):

        # If first linked list is empty
        if first is None:
            return second

        # If secon linked list is empty
        if second is None:
            return first

        # Pick the smaller value
        if first.data < second.data:
            first.next = self.merge(first.next, second)
            first.next.prev = first
            first.prev = None
            return first
        else:
            second.next = self.merge(first, second.next)
            second.next.prev = second
            second.prev = None
            return second

    # Function to do merge sort
    def mergeSort(self, tempHead):
        if tempHead is None:
            return tempHead
        if tempHead.next is None:
            return tempHead

        second = self.split(tempHead)

        # Recur for left and righ halves
        tempHead = self.mergeSort(tempHead)
        second = self.mergeSort(second)

        # Merge the two sorted halves
        return self.merge(tempHead, second)
```

```
# Split the doubly linked list (DLL) into two DLLs
# of half sizes
def split(self, tempHead):
    fast = slow = tempHead
    while(True):
        if fast.next is None:
            break
        if fast.next.next is None:
            break
        fast = fast.next.next
        slow = slow.next

    temp = slow.next
    slow.next = None
    return temp

# Given a reference to the head of a list and an
# integer, inserts a new node on the front of list
def push(self, new_data):

    # 1. Allocates node
    # 2. Put the data in it
    new_node = Node(new_data)

    # 3. Make next of new node as head and
    # previous as None (already None)
    new_node.next = self.head

    # 4. change prev of head node to new_node
    if self.head is not None:
        self.head.prev = new_node

    # 5. move the head to point to the new node
    self.head = new_node

def printList(self, node):
    temp = node
    print "Forward Traversal using next pointer"
    while(node is not None):
        print node.data,
        temp = node
        node = node.next
    print "\nBackward Traversal using prev pointer"
    while(temp):
        print temp.data,
        temp = temp.prev
```

```
# Driver program to test the above functions
dll = DoublyLinkedList()
dll.push(5)
dll.push(20);
dll.push(4);
dll.push(3);
dll.push(30)
dll.push(10);
dll.head = dll.mergeSort(dll.head)
print "Linked List after sorting"
dll.printList(dll.head)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
Linked List after sorting
Forward Traversal using next pointer
3 4 5 10 20 30
Backward Traversal using prev pointer
30 20 10 5 4 3
```

Thanks to Goku for providing above implementation in a comment [here](#).

Time Complexity: Time complexity of the above implementation is same as time complexity of [MergeSort for arrays](#). It takes $\Theta(n\log n)$ time.

You may also like to see [QuickSort for doubly linked list](#)

Source

<https://www.geeksforgeeks.org/merge-sort-for-doubly-linked-list/>

Chapter 190

Merge Sort for Linked Lists

Merge Sort for Linked Lists - GeeksforGeeks

[Merge sort](#) is often preferred for sorting a linked list. The slow random-access performance of a linked list makes some other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible.

Let head be the first node of the linked list to be sorted and headRef be the pointer to head. Note that we need a reference to head in MergeSort() as the below implementation changes next links to sort the linked lists (not data at the nodes), so head node has to be changed if the data at original head is not the smallest value in linked list.

```
MergeSort(headRef)
1) If head is NULL or there is only one element in the Linked List
   then return.
2) Else divide the linked list into two halves.
   FrontBackSplit(head, &a, &b); /* a and b are two halves */
3) Sort the two halves a and b.
   MergeSort(a);
   MergeSort(b);
4) Merge the sorted a and b (using SortedMerge() discussed here)
   and update the head pointer using headRef.
   *headRef = SortedMerge(a, b);
```

C

```
// C code for linked list merged sort
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct Node
```

```
{  
    int data;  
    struct Node* next;  
};  
  
/* function prototypes */  
struct Node* SortedMerge(struct Node* a, struct Node* b);  
void FrontBackSplit(struct Node* source,  
                     struct Node** frontRef, struct Node** backRef);  
  
/* sorts the linked list by changing next pointers (not data) */  
void MergeSort(struct Node** headRef)  
{  
    struct Node* head = *headRef;  
    struct Node* a;  
    struct Node* b;  
  
    /* Base case -- length 0 or 1 */  
    if ((head == NULL) || (head->next == NULL))  
    {  
        return;  
    }  
  
    /* Split head into 'a' and 'b' sublists */  
    FrontBackSplit(head, &a, &b);  
  
    /* Recursively sort the sublists */  
    MergeSort(&a);  
    MergeSort(&b);  
  
    /* answer = merge the two sorted lists together */  
    *headRef = SortedMerge(a, b);  
}  
  
/* See https://www.geeksforgeeks.org/?p=3622 for details of this  
function */  
struct Node* SortedMerge(struct Node* a, struct Node* b)  
{  
    struct Node* result = NULL;  
  
    /* Base cases */  
    if (a == NULL)  
        return(b);  
    else if (b==NULL)  
        return(a);  
  
    /* Pick either a or b, and recur */  
    if (a->data <= b->data)
```

```

{
    result = a;
    result->next = SortedMerge(a->next, b);
}
else
{
    result = b;
    result->next = SortedMerge(a, b->next);
}
return(result);
}

/* UTILITY FUNCTIONS */
/* Split the nodes of the given list into front and back halves,
   and return the two lists using the reference parameters.
   If the length is odd, the extra node should go in the front list.
   Uses the fast/slow pointer strategy. */
void FrontBackSplit(struct Node* source,
                     struct Node** frontRef, struct Node** backRef)
{
    struct Node* fast;
    struct Node* slow;
    slow = source;
    fast = source->next;

    /* Advance 'fast' two nodes, and advance 'slow' one node */
    while (fast != NULL)
    {
        fast = fast->next;
        if (fast != NULL)
        {
            slow = slow->next;
            fast = fast->next;
        }
    }

    /* 'slow' is before the midpoint in the list, so split it in two
       at that point. */
    *frontRef = source;
    *backRef = slow->next;
    slow->next = NULL;
}

/* Function to print nodes in a given linked list */
void printList(struct Node *node)
{
while(node!=NULL)
{

```

```
printf("%d ", node->data);
node = node->next;
}
}

/* Function to insert a node at the beginning of the linked list */
void push(struct Node** head_ref, int new_data)
{
/* allocate node */
struct Node* new_node =
    (struct Node*) malloc(sizeof(struct Node));

/* put in the data */
new_node->data = new_data;

/* link the old list off the new node */
new_node->next = (*head_ref);

/* move the head to point to the new node */
(*head_ref) = new_node;
}

/* Driver program to test above functions*/
int main()
{
/* Start with the empty list */
struct Node* res = NULL;
struct Node* a = NULL;

/* Let us create a unsorted linked lists to test the functions
Created lists shall be a: 2->3->20->5->10->15 */
push(&a, 15);
push(&a, 10);
push(&a, 5);
push(&a, 20);
push(&a, 3);
push(&a, 2);

/* Sort the above created Linked List */
MergeSort(&a);

printf("Sorted Linked List is: \n");
printList(a);

getchar();
return 0;
}
```

Java

```
// Java program to illustrate merge sorted
// of linkedList

public class linkedList
{
    node head = null;
    // node a,b;
    static class node
    {
        int val;
        node next;

        public node(int val)
        {
            this.val = val;
        }
    }

    node sortedMerge(node a, node b)
    {
        node result = null;
        /* Base cases */
        if (a == null)
            return b;
        if (b == null)
            return a;

        /* Pick either a or b, and recur */
        if (a.val <= b.val)
        {
            result = a;
            result.next = sortedMerge(a.next, b);
        }
        else
        {
            result = b;
            result.next = sortedMerge(a, b.next);
        }
        return result;
    }

    node mergeSort(node h)
    {
        // Base case : if head is null
        if (h == null || h.next == null)
```

```
{  
    return h;  
}  
  
// get the middle of the list  
node middle = getMiddle(h);  
node nextofmiddle = middle.next;  
  
// set the next of middle node to null  
middle.next = null;  
  
// Apply mergeSort on left list  
node left = mergeSort(h);  
  
// Apply mergeSort on right list  
node right = mergeSort(nextofmiddle);  
  
// Merge the left and right lists  
node sortedlist = sortedMerge(left, right);  
return sortedlist;  
}  
  
// Utility function to get the middle of the linked list  
node getMiddle(node h)  
{  
    //Base case  
    if (h == null)  
        return h;  
    node fastptr = h.next;  
    node slowptr = h;  
  
    // Move fastptr by two and slow ptr by one  
    // Finally slowptr will point to middle node  
    while (fastptr != null)  
    {  
        fastptr = fastptr.next;  
        if(fastptr!=null)  
        {  
            slowptr = slowptr.next;  
            fastptr=fastptr.next;  
        }  
    }  
    return slowptr;  
}  
  
void push(int new_data)  
{  
    /* allocate node */
```

```
node new_node = new node(new_data);

/* link the old list off the new node */
new_node.next = head;

/* move the head to point to the new node */
head = new_node;
}

// Utility function to print the linked list
void printList(node headref)
{
    while (headref != null)
    {
        System.out.print(headref.val + " ");
        headref = headref.next;
    }
}

public static void main(String[] args)
{
    linkedList li = new linkedList();
    /*
     * Let us create a unsorted linked lists to test the functions Created
     * lists shall be a: 2->3->20->5->10->15
     */
    li.push(15);
    li.push(10);
    li.push(5);
    li.push(20);
    li.push(3);
    li.push(2);
    System.out.println("Linked List without sorting is :");
    li.printList(li.head);

    // Apply merge Sort
    li.head = li.mergeSort(li.head);
    System.out.print("\n Sorted Linked List is: \n");
    li.printList(li.head);
}

// This code is contributed by Rishabh Mahrsee
```

Time Complexity: O(n Log n)

Sources:

http://en.wikipedia.org/wiki/Merge_sort

<http://cslibrary.stanford.edu/105/LinkedListProblems.pdf>

Source

<https://www.geeksforgeeks.org/merge-sort-for-linked-list/>

Chapter 191

Merge Sort for Linked Lists in JavaScript

Merge Sort for Linked Lists in JavaScript - GeeksforGeeks

Prerequisite:[Merge Sort for Linked Lists](#)

Merge sort is often preferred for sorting a linked list. The slow random-access performance of a linked list makes some other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible.

In this post, Merge sort for linked list is implemented using JavaScript.

Examples:

```
Input : 5 -> 4 -> 3 -> 2 -> 1  
Output :1 -> 2 -> 3 -> 4 -> 5  
  
Input : 10 -> 20 -> 3 -> 2 -> 1  
Output : 1 -> 2 -> 3 -> 10 -> 20
```

```
<script>  
  
// Create Node of LinkedList  
function Node(data) {  
    this.node = data;  
    this.next = null;  
}  
  
// To initialize a linkedlist  
function LinkedList(list) {  
    this.head = list || null  
}  
  
// Function to insert The new Node into the linkedList
```

```
LinkedList.prototype.insert = function(data) {

    // Check if the linked list is empty
    // so insert first node and lead head
    // points to generic node
    if (this.head === null)
        this.head = new Node(data);

    else {

        // If linked list is not empty, insert the node
        // at the end of the linked list
        let list = this.head;
        while (list.next) {
            list = list.next;
        }

        // Now here list pointer points to last
        // node let's insert out new node in it
        list.next = new Node(data)
    }
}

// Function to print linkedList
LinkedList.prototype.iterate = function() {

    // First we will check whether our
    // linked list is empty or node
    if (this.head === null)
        return null;

    // If linked list is not empty we will
    // iterate from each Node and prints
    // it's value store in "data" property

    let list = this.head;

    // we will iterate until our list variable
    // contains the "Next" value of the last Node
    // i.e-> null
    while (list) {
        document.write(list.node)
        if (list.next)
            document.write(' -> ')
        list = list.next
    }
}
```

```
// Function to mergesort a linked list
LinkedList.prototype.mergeSort = function(list) {

    if (list.next === null)
        return list;

    let count = 0;
    let countList = list
    let leftPart = list;
    let leftPointer = list;
    let rightPart = null;
    let rightPointer = null;

    // Counting the nodes in the received linkedlist
    while (countList.next !== null) {
        count++;
        countList = countList.next;
    }

    // counting the mid of the linked list
    let mid = Math.floor(count / 2)
    let count2 = 0;

    // separating the left and right part with
    // respect to mid node in the linked list
    while (count2 < mid) {
        count2++;
        leftPointer = leftPointer.next;
    }

    rightPart = new LinkedList(leftPointer.next);
    leftPointer.next = null;

    // Here are two linked list which
    // contains the left most nodes and right
    // most nodes of the mid node
    return this._mergeSort(this.mergeSort(leftPart),
                           this.mergeSort(rightPart.head))
}

// Merging both lists in sorted manner
LinkedList.prototype._mergeSort = function(left, right) {

    // Create a new empty linked list
    let result = new LinkedList()

    let resultPointer = result.head;
    let pointerLeft = left;
```

```
let pointerRight = right;

// If true then add left most node value in result,
// increment left pointer else do the same in
// right linked list.
// This loop will be executed until pointer's of
// a left node or right node reached null
while (pointerLeft && pointerRight) {
    let tempNode = null;

    // Check if the right node's value is greater than
    // left node's value
    if (pointerLeft.node > pointerRight.node) {
        tempNode = pointerRight.node
        pointerRight = pointerRight.next;
    }
    else {
        tempNode = pointerLeft.node
        pointerLeft = pointerLeft.next;
    }

    if (result.head == null) {
        result.head = new Node(tempNode)
        resultPointer = result.head
    }
    else {
        resultPointer.next = new Node(tempNode)
        resultPointer = resultPointer.next
    }
}

// Add the remaining elements in the last of resultant
// linked list
resultPointer.next = pointerLeft;
while (resultPointer.next)
    resultPointer = resultPointer.next

resultPointer.next = pointerRight

// Result is the new sorted linked list
return result.head;
}

// Initialize the object
let l = new LinkedList();
l.insert(10)
l.insert(20)
```

```
l.insert(3)
l.insert(2)
l.insert(1)
// Print the linked list
l.iterate()

// Sort the linked list
l.head = LinkedList.prototype.mergeSort(l.head)

document.write('<br> After sorting : ');

// Print the sorted linked list
l.iterate()
</script>
```

Output

```
10 -> 20 -> 3 -> 2 -> 1
After sorting : 1 -> 2 -> 3 -> 10 -> 20
```

Source

<https://www.geeksforgeeks.org/merge-sort-linked-lists-javascript/>

Chapter 192

Merge Sort using Multi-threading

Merge Sort using Multi-threading - GeeksforGeeks

Merge Sort is a popular sorting technique which divides an array or list into two halves and then start merging them when sufficient depth is reached. Time complexity of merge sort is $O(n\log n)$.

Threads are lightweight processes and threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.

Multi-threading is way to improve parallelism by running the threads simultaneously in different cores of your processor. In this program, we'll use 4 threads but you may change it according to the number of cores your processor has.

Examples:

```
Input : 83, 86, 77, 15, 93, 35, 86, 92, 49, 21,  
        62, 27, 90, 59, 63, 26, 40, 26, 72, 36  
Output : 15, 21, 26, 26, 27, 35, 36, 40, 49, 59,  
        62, 63, 72, 77, 83, 86, 86, 90, 92, 93
```

```
Input : 6, 5, 4, 3, 2, 1  
Output : 1, 2, 3, 4, 5, 6
```

Note* It is better to execute the program in linux based system.
To compile in linux System :

```
g++ -pthread program_name.cpp
```

```
// CPP Program to implement merge sort using
// multi-threading
#include <iostream>
#include <pthread.h>
#include <time.h>

// number of elements in array
#define MAX 20

// number of threads
#define THREAD_MAX 4

using namespace std;

// array of size MAX
int a[MAX];
int part = 0;

// merge function for merging two parts
void merge(int low, int mid, int high)
{
    int* left = new int[mid - low + 1];
    int* right = new int[high - mid];

    // n1 is size of left part and n2 is size
    // of right part
    int n1 = mid - low + 1, n2 = high - mid, i, j;

    // storing values in left part
    for (i = 0; i < n1; i++)
        left[i] = a[i + low];

    // storing values in right part
    for (i = 0; i < n2; i++)
        right[i] = a[i + mid + 1];

    int k = low;
    i = j = 0;

    // merge left and right in ascending order
    while (i < n1 && j < n2) {
        if (left[i] <= right[j])
            a[k++] = left[i++];
        else
            a[k++] = right[j++];
    }

    // insert remaining values from left
}
```

```
while (i < n1) {
    a[k++] = left[i++];
}

// insert remaining values from right
while (j < n2) {
    a[k++] = right[j++];
}
}

// merge sort function
void merge_sort(int low, int high)
{
    // calculating mid point of array
    int mid = low + (high - low) / 2;
    if (low < high) {

        // calling first half
        merge_sort(low, mid);

        // calling second half
        merge_sort(mid + 1, high);

        // merging the two halves
        merge(low, mid, high);
    }
}

// thread function for multi-threading
void* merge_sort(void* arg)
{
    // which part out of 4 parts
    int thread_part = part++;

    // calculating low and high
    int low = thread_part * (MAX / 4);
    int high = (thread_part + 1) * (MAX / 4) - 1;

    // evaluating mid point
    int mid = low + (high - low) / 2;
    if (low < high) {
        merge_sort(low, mid);
        merge_sort(mid + 1, high);
        merge(low, mid, high);
    }
}

// Driver Code
```

```
int main()
{
    // generating random values in array
    for (int i = 0; i < MAX; i++)
        a[i] = rand() % 100;

    // t1 and t2 for calculating time for
    // merge sort
    clock_t t1, t2;

    t1 = clock();
    pthread_t threads[THREAD_MAX];

    // creating 4 threads
    for (int i = 0; i < THREAD_MAX; i++)
        pthread_create(&threads[i], NULL, merge_sort,
                      (void*)NULL);

    // joining all 4 threads
    for (int i = 0; i < 4; i++)
        pthread_join(threads[i], NULL);

    // merging the final 4 parts
    merge(0, (MAX / 2 - 1) / 2, MAX / 2 - 1);
    merge(MAX / 2, MAX/2 + (MAX-1-MAX/2)/2, MAX - 1);
    merge(0, (MAX - 1)/2, MAX - 1);

    t2 = clock();

    // displaying sorted array
    cout << "Sorted array: ";
    for (int i = 0; i < MAX; i++)
        cout << a[i] << " ";

    // time taken by merge sort in seconds
    cout << "Time taken: " << (t2 - t1) /
        (double)CLOCKS_PER_SEC << endl;

    return 0;
}</div>
```

Output:

```
Sorted array: 15 21 26 26 27 35 36 40 49 59 62 63 72 77 83 86 86 90 92 93
Time taken: 0.001023
```

Source

<https://www.geeksforgeeks.org/merge-sort-using-multi-threading/>

Chapter 193

Merge Sort with O(1) extra space merge and O(n lg n) time

Merge Sort with O(1) extra space merge and O(n lg n) time - GeeksforGeeks

We have discussed [Merge sort](#). How to modify the algorithm so that merge works in O(1) extra space and algorithm still works in O(n Log n) time. We may assume that the input values are integers only.

Examples:

Input : 5 4 3 2 1
Output : 1 2 3 4 5

Input : 999 612 589 856 56 945 243
Output : 56 243 589 612 856 945 999

For integer types, merge sort can be made inplace using some mathematics trick of modulus and division. That means storing two elements value at one index and can be extracted using modulus and division.

First we have to find a value greater than all the elements of the array. Now we can store the original value as modulus and the second value as division. Suppose we want to store **arr[i]** and **arr[j]** both at index i(means in arr[i]). First we have to find a '**maxval**' greater than both arr[i] and arr[j]. Now we can store as **arr[i] = arr[i] + arr[j]*maxval**. Now **arr[i]%maxval** will give the original value of arr[i] and **arr[i]/maxval** will give the value of arr[j]. So below is the implementation on merge sort.

C++

```
// C++ program to sort an array using merge sort such
// that merge operation takes O(1) extra space.
#include <bits/stdc++.h>
```

```
using namespace std;
void merge(int arr[], int beg, int mid, int end, int maxele)
{
    int i = beg;
    int j = mid + 1;
    int k = beg;
    while (i <= mid && j <= end) {
        if (arr[i] % maxele <= arr[j] % maxele) {
            arr[k] = arr[k] + (arr[i] % maxele) * maxele;
            k++;
            i++;
        }
        else {
            arr[k] = arr[k] + (arr[j] % maxele) * maxele;
            k++;
            j++;
        }
    }
    while (i <= mid) {
        arr[k] = arr[k] + (arr[i] % maxele) * maxele;
        k++;
        i++;
    }
    while (j <= end) {
        arr[k] = arr[k] + (arr[j] % maxele) * maxele;
        k++;
        j++;
    }
}

// Obtaining actual values
for (int i = beg; i <= end; i++)
    arr[i] = arr[i] / maxele;
}

// Recursive merge sort with extra parameter, naxele
void mergeSortRec(int arr[], int beg, int end, int maxele)
{
    if (beg < end) {
        int mid = (beg + end) / 2;
        mergeSortRec(arr, beg, mid, maxele);
        mergeSortRec(arr, mid + 1, end, maxele);
        merge(arr, beg, mid, end, maxele);
    }
}

// This functions finds max element and calls recursive
// merge sort.
void mergeSort(int arr[], int n)
```

```
{  
    int maxele = *max_element(arr, arr+n) + 1;  
    mergeSortRec(arr, 0, n-1, maxele);  
}  
  
int main()  
{  
    int arr[] = { 999, 612, 589, 856, 56, 945, 243 };  
    int n = sizeof(arr) / sizeof(arr[0]);  
  
    mergeSort(arr, n);  
  
    cout << "Sorted array \n";  
    for (int i = 0; i < n; i++)  
        cout << arr[i] << " ";  
    return 0;  
}
```

C#

```
// C# program to sort an array  
// using merge sort such that  
// merge operation takes O(1)  
// extra space.  
using System;  
using System.Linq;  
  
class GFG  
{  
    static void merge(int []arr, int beg,  
                      int mid, int end,  
                      int maxele)  
    {  
        int i = beg;  
        int j = mid + 1;  
        int k = beg;  
        while (i <= mid && j <= end)  
        {  
            if (arr[i] %  
                maxele <= arr[j] % maxele)  
            {  
                arr[k] = arr[k] + (arr[i] %  
                                    maxele) * maxele;  
                k++;  
                i++;  
            }  
            else  
            {
```

```
        arr[k] = arr[k] +
                  (arr[j] % maxele) *
                      maxele;
                k++;
                j++;
            }
        }
    while (i <= mid)
    {
        arr[k] = arr[k] + (arr[i] %
                           maxele) * maxele;
        k++;
        i++;
    }
    while (j <= end)
    {
        arr[k] = arr[k] + (arr[j] %
                           maxele) * maxele;
        k++;
        j++;
    }

    // Obtaining actual values
    for ( i = beg; i <= end; i++)
        arr[i] = arr[i] / maxele;
}

// Recursive merge sort
// with extra parameter, naxele
static void mergeSortRec(int []arr, int beg,
                         int end, int maxele)
{
    if (beg < end)
    {
        int mid = (beg + end) / 2;
        mergeSortRec(arr, beg,
                     mid, maxele);
        mergeSortRec(arr, mid + 1,
                     end, maxele);
        merge(arr, beg, mid,
              end, maxele);
    }
}

// This functions finds
// max element and calls
// recursive merge sort.
static void mergeSort(int []arr, int n)
```

```
{  
    int maxele = arr.Max() + 1;  
    mergeSortRec(arr, 0, n - 1, maxele);  
}  
  
//Driver code  
public static void Main ()  
{  
    int []arr = {999, 612, 589,  
                856, 56, 945, 243};  
    int n = arr.Length;  
  
    mergeSort(arr, n);  
  
    Console.WriteLine("Sorted array ");  
    for (int i = 0; i < n; i++)  
        Console.Write( arr[i] + " ");  
}  
}  
  
// This code is contributed  
// by inder_verma.
```

Output:

```
Sorted array  
56 243 589 612 856 945 999
```

Improved By : [inderDuMCA](#)

Source

<https://www.geeksforgeeks.org/merge-sort-with-o1-extra-space-merge-and-on-lg-n-time/>

Chapter 194

Merge k sorted arrays Set 1

Merge k sorted arrays Set 1 - GeeksforGeeks

Given k sorted arrays of size n each, merge them and print the sorted output.

Example:

Input:

```
k = 3, n = 4
arr[][] = {{1, 3, 5, 7},
            {2, 4, 6, 8},
            {0, 9, 10, 11}};
```

Output: 0 1 2 3 4 5 6 7 8 9 10 11

A **simple solution** is to create an output array of size $n*k$ and one by one copy all arrays to it. Finally, sort the output array using any $O(n \log n)$ sorting algorithm. This approach takes $O(nk \log nk)$ time.

One efficient solution is to first merge arrays into groups of 2. After first merging, we have $k/2$ arrays. We again merge arrays in groups, now we have $k/4$ arrays. We keep doing it until we have one array left. The time complexity of this solution would be $O(nk \log k)$. How? Every merging in first iteration would take $2n$ time (merging two arrays of size n). Since there are total $k/2$ merging, total time in first iteration would be $O(nk)$. Next iteration would also take $O(nk)$. There will be total $O(\log k)$ iterations, hence time complexity is $O(nk \log k)$.

Another efficient solution is to use [Min Heap](#). This Min Heap based solution has same time complexity which is $O(nk \log k)$. But for [different sized arrays](#), this solution works much better.

Following is detailed algorithm.

1. Create an output array of size $n*k$.
2. Create a min heap of size k and insert 1st element in all the arrays into the heap

3. Repeat following steps $n*k$ times.

- a) Get minimum element from heap (minimum is always at root) and store it in output array.
- b) Replace heap root with next element from the array from which the element is extracted. If the array doesn't have any more elements, then replace root with infinite. After replacing the root, heapify the tree.

Following is C++ implementation of the above algorithm.

```
// C++ program to merge k sorted arrays of size n each.
#include<iostream>
#include<limits.h>
using namespace std;

#define n 4

// A min heap node
struct MinHeapNode
{
    int element; // The element to be stored
    int i; // index of the array from which the element is taken
    int j; // index of the next element to be picked from array
};

// Prototype of a utility function to swap two min heap nodes
void swap(MinHeapNode *x, MinHeapNode *y);

// A class for Min Heap
class MinHeap
{
    MinHeapNode *harr; // pointer to array of elements in heap
    int heap_size; // size of min heap
public:
    // Constructor: creates a min heap of given size
    MinHeap(MinHeapNode a[], int size);

    // to heapify a subtree with root at given index
    void MinHeapify(int);

    // to get index of left child of node at index i
    int left(int i) { return (2*i + 1); }

    // to get index of right child of node at index i
    int right(int i) { return (2*i + 2); }

    // to get the root
    MinHeapNode getMin() { return harr[0]; }

    // to replace root with new node x and heapify() new root
}
```

```

void replaceMin(MinHeapNode x) { harr[0] = x; MinHeapify(0); }

// This function takes an array of arrays as an argument and
// All arrays are assumed to be sorted. It merges them together
// and prints the final sorted output.
int *mergeKArrays(int arr[][][n], int k)
{
    int *output = new int[n*k]; // To store output array

    // Create a min heap with k heap nodes. Every heap node
    // has first element of an array
    MinHeapNode *harr = new MinHeapNode[k];
    for (int i = 0; i < k; i++)
    {
        harr[i].element = arr[i][0]; // Store the first element
        harr[i].i = i; // index of array
        harr[i].j = 1; // Index of next element to be stored from array
    }
    MinHeap hp(harr, k); // Create the heap

    // Now one by one get the minimum element from min
    // heap and replace it with next element of its array
    for (int count = 0; count < n*k; count++)
    {
        // Get the minimum element and store it in output
        MinHeapNode root = hp.getMin();
        output[count] = root.element;

        // Find the next element that will replace current
        // root of heap. The next element belongs to same
        // array as the current root.
        if (root.j < n)
        {
            root.element = arr[root.i][root.j];
            root.j += 1;
        }
        // If root was the last element of its array
        else root.element = INT_MAX; //INT_MAX is for infinite

        // Replace root with next element of array
        hp.replaceMin(root);
    }

    return output;
}

// FOLLOWING ARE IMPLEMENTATIONS OF STANDARD MIN HEAP METHODS

```

```

// FROM CORMEN BOOK
// Constructor: Builds a heap from a given array a[] of given size
MinHeap::MinHeap(MinHeapNode a[], int size)
{
    heap_size = size;
    harr = a; // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && harr[l].element < harr[i].element)
        smallest = l;
    if (r < heap_size && harr[r].element < harr[smallest].element)
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}

// A utility function to swap two elements
void swap(MinHeapNode *x, MinHeapNode *y)
{
    MinHeapNode temp = *x; *x = *y; *y = temp;
}

// A utility function to print array elements
void printArray(int arr[], int size)
{
    for (int i=0; i < size; i++)
        cout << arr[i] << " ";
}

// Driver program to test above functions
int main()
{

```

```
// Change n at the top to change number of elements
// in an array
int arr[][][n] = {{2, 6, 12, 34},
                  {1, 9, 20, 1000},
                  {23, 34, 90, 2000}};
int k = sizeof(arr)/sizeof(arr[0]);

int *output = mergeKArrays(arr, k);

cout << "Merged array is " << endl;
printArray(output, n*k);

return 0;
}
```

Output:

```
Merged array is
1 2 6 9 12 20 23 34 34 90 1000 2000
```

Time Complexity: The main step is 3rd step, the loop runs $n*k$ times. In every iteration of loop, we call heapify which takes $O(\log k)$ time. Therefore, the time complexity is $O(nk \log k)$.

Merge k sorted arrays Set 2 (Different Sized Arrays)

Thanks to [vignesh](#)for suggesting this problem and initial solution. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [maxflex](#)

Source

<https://www.geeksforgeeks.org/merge-k-sorted-arrays/>

Chapter 195

Merge two sorted arrays

Merge two sorted arrays - GeeksforGeeks

Given two sorted arrays, the task is to merge them in a sorted manner.

Examples:

```
Input : arr1[] = { 1, 3, 4, 5}
        arr2[] = {2, 4, 6, 8}
Output : arr3[] = {1, 2, 3, 4, 5, 6, 7, 8}
```

```
Input : arr1[] = { 5, 8, 9}
        arr2[] = {4, 7, 8}
Output : arr3[] = {4, 5, 7, 8, 8, 9}
```

Method 1 ($O(n_1 * n_2)$ Time and $O(1)$ Extra Space)

1. Create an array arr3[] of size $n_1 + n_2$.
2. Copy all n_1 elements of arr1[] to arr3[]
3. Traverse arr2[] and one by one insert elements (like [insertion sort](#)) of arr3[] to arr1[].
This step take $O(n_1 * n_2)$ time.

We have discussed implementation of above method in [Merge two sorted arrays with \$O\(1\)\$ extra space](#)

Method 2 ($O(n_1 + n_2)$ Time and $O(n_1 + n_2)$ Extra Space)

The idea is to use Merge function of [Merge sort](#).

1. Create an array arr3[] of size $n_1 + n_2$.
2. Simultaneously traverse arr1[] and arr2[].

- Pick smaller of current elements in arr1[] and arr2[], copy this smaller element to next position in arr3[] and move ahead in arr3[] and the array whose element is picked.
3. If there are remaining elements in arr1[] or arr2[], copy them also in arr3[].

C++

```
// C++ program to merge two sorted arrays/
#include<iostream>
using namespace std;

// Merge arr1[0..n1-1] and arr2[0..n2-1] into
// arr3[0..n1+n2-1]
void mergeArrays(int arr1[], int arr2[], int n1,
                 int n2, int arr3[])
{
    int i = 0, j = 0, k = 0;

    // Traverse both array
    while (i < n1 && j < n2)
    {
        // Check if current element of first
        // array is smaller than current element
        // of second array. If yes, store first
        // array element and increment first array
        // index. Otherwise do same with second array
        if (arr1[i] < arr2[j])
            arr3[k++] = arr1[i++];
        else
            arr3[k++] = arr2[j++];
    }

    // Store remaining elements of first array
    while (i < n1)
        arr3[k++] = arr1[i++];

    // Store remaining elements of second array
    while (j < n2)
        arr3[k++] = arr2[j++];
}

// Driver code
int main()
{
    int arr1[] = {1, 3, 5, 7};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
```

```
int arr2[] = {2, 4, 6, 8};  
int n2 = sizeof(arr2) / sizeof(arr2[0]);  
  
int arr3[n1+n2];  
mergeArrays(arr1, arr2, n1, n2, arr3);  
  
cout << "Array after merging" << endl;  
for (int i=0; i < n1+n2; i++)  
    cout << arr3[i] << " ";  
  
return 0;  
}
```

Java

```
// Java program to merge two sorted arrays  
import java.util.*;  
import java.lang.*;  
import java.io.*;  
  
class MergeTwoSorted  
{  
    // Merge arr1[0..n1-1] and arr2[0..n2-1]  
    // into arr3[0..n1+n2-1]  
    public static void mergeArrays(int[] arr1, int[] arr2, int n1,  
                                    int n2, int[] arr3)  
    {  
        int i = 0, j = 0, k = 0;  
  
        // Traverse both array  
        while (i < n1 && j < n2)  
        {  
            // Check if current element of first  
            // array is smaller than current element  
            // of second array. If yes, store first  
            // array element and increment first array  
            // index. Otherwise do same with second array  
            if (arr1[i] < arr2[j])  
                arr3[k++] = arr1[i++];  
            else  
                arr3[k++] = arr2[j++];  
        }  
  
        // Store remaining elements of first array  
        while (i < n1)  
            arr3[k++] = arr1[i++];  
  
        // Store remaining elements of second array
```

```
        while (j < n2)
            arr3[k++] = arr2[j++];
    }

    public static void main (String[] args)
    {
        int[] arr1 = {1, 3, 5, 7};
        int n1 = arr1.length;

        int[] arr2 = {2, 4, 6, 8};
        int n2 = arr2.length;

        int[] arr3 = new int[n1+n2];

        mergeArrays(arr1, arr2, n1, n2, arr3);

        System.out.println("Array after merging");
        for (int i=0; i < n1+n2; i++)
            System.out.print(arr3[i] + " ");
    }
}

/* This code is contributed by Mr. Somesh Awasthi */
```

Python 3

```
# Python program to merge
# two sorted arrays

# Merge arr1[0..n1-1] and
# arr2[0..n2-1] into
# arr3[0..n1+n2-1]
def mergeArrays(arr1, arr2, n1, n2):
    arr3 = [None] * (n1 + n2)
    i = 0
    j = 0
    k = 0

    # Traverse both array
    while i < n1 and j < n2:

        # Check if current element
        # of first array is smaller
        # than current element of
        # second array. If yes,
        # store first array element
        # and increment first array
        # index. Otherwise do same
```

```
# with second array
if arr1[i] < arr2[j]:
    arr3[k] = arr1[i]
    k = k + 1
    i = i + 1
else:
    arr3[k] = arr2[j]
    k = k + 1
    j = j + 1

# Store remaining elements
# of first array
while i < n1:
    arr3[k] = arr1[i];
    k = k + 1
    i = i + 1

# Store remaining elements
# of second array
while j < n2:
    arr3[k] = arr2[j];
    k = k + 1
    j = j + 1
print("Array after merging")
for i in range(n1 + n2):
    print(str(arr3[i]), end = " ")

# Driver code
arr1 = [1, 3, 5, 7]
n1 = len(arr1)

arr2 = [2, 4, 6, 8]
n2 = len(arr2)
mergeArrays(arr1, arr2, n1, n2);

# This code is contributed
# by ChitraNayal
```

C#

```
// C# program to merge
// two sorted arrays
using System;

class GFG
{
    // Merge arr1[0..n1-1] and
```

```
// arr2[0..n2-1] into
// arr3[0..n1+n2-1]
public static void mergeArrays(int[] arr1, int[] arr2,
                               int n1, int n2, int[] arr3)
{
    int i = 0, j = 0, k = 0;

    // Traverse both array
    while (i < n1 && j < n2)
    {
        // Check if current element
        // of first array is smaller
        // than current element
        // of second array. If yes,
        // store first array element
        // and increment first array
        // index. Otherwise do same
        // with second array
        if (arr1[i] < arr2[j])
            arr3[k++] = arr1[i++];
        else
            arr3[k++] = arr2[j++];
    }

    // Store remaining
    // elements of first array
    while (i < n1)
        arr3[k++] = arr1[i++];

    // Store remaining elements
    // of second array
    while (j < n2)
        arr3[k++] = arr2[j++];
}

// Driver code
public static void Main()
{
    int[] arr1 = {1, 3, 5, 7};
    int n1 = arr1.Length;

    int[] arr2 = {2, 4, 6, 8};
    int n2 = arr2.Length;

    int[] arr3 = new int[n1+n2];

    mergeArrays(arr1, arr2, n1, n2, arr3);
```

```
Console.WriteLine("Array after merging\n");
for (int i = 0; i < n1 + n2; i++)
    Console.Write(arr3[i] + " ");
}

// This code is contributed
// by ChitraNayal
```

PHP

```
<?php
// PHP program to merge
// two sorted arrays

// Merge $arr1[0..$n1-1] and
//         $arr2[0..$n2-1] into
//         $arr3[0..$n1+$n2-1]
function mergeArrays(&$arr1, &$arr2,
                     $n1, $n2, &$arr3)
{
    $i = 0;
    $j = 0;
    $k = 0;

    // Traverse both array
    while ($i < $n1 && $j < $n2)
    {
        // Check if current element
        // of first array is smaller
        // than current element of
        // second array. If yes,
        // store first array element
        // and increment first array
        // index. Otherwise do same
        // with second array
        if ($arr1[$i] < $arr2[$j])
            $arr3[$k++] = $arr1[$i++];
        else
            $arr3[$k++] = $arr2[$j++];
    }

    // Store remaining elements
    // of first array
    while ($i < $n1)
        $arr3[$k++] = $arr1[$i++];

    // Store remaining elements
```

```
// of second array
while ($j < $n2)
    $arr3[$k++] = $arr2[$j++];
}

// Driver code
$arr1 = array(1, 3, 5, 7);
$n1 = sizeof($arr1);

$arr2 = array(2, 4, 6, 8);
$n2 = sizeof($arr2);

$arr3[$n1 + $n2] = array();
mergeArrays($arr1, $arr2, $n1,
            $n2, $arr3);

echo "Array after merging \n" ;
for ($i = 0; $i < $n1 + $n2; $i++)
    echo $arr3[$i] . " ";

// This code is contributed
// by ChitraNayal
?>
```

Output:

```
Array after merging
1 2 3 4 5 6 7 8
```

Time Complexity : $O(n_1 + n_2)$
Auxiliary Space : $O(n_1 + n_2)$

[Brocade](#), [Goldman-Sachs](#), [Juniper](#), [Linkedin](#), [Microsoft](#), [Quikr](#), [Snapdeal](#), [Synopsys](#), [Zoho](#)

Related Articles :

[Merge two sorted arrays with \$O\(1\)\$ extra space](#)
[Merge k sorted arrays Set 1](#)

Improved By : [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/merge-two-sorted-arrays/>

Chapter 196

Merge two sorted arrays with O(1) extra space

Merge two sorted arrays with O(1) extra space - GeeksforGeeks

We are given two sorted array. We need to merge these two arrays such that the initial numbers (after complete sorting) are in the first array and the remaining numbers are in the second array. Extra space allowed in O(1).

Example:

```
Input: ar1[] = {10};  
       ar2[] = {2, 3};  
Output: ar1[] = {2}  
       ar2[] = {3, 10}  
  
Input: ar1[] = {1, 5, 9, 10, 15, 20};  
       ar2[] = {2, 3, 8, 13};  
Output: ar1[] = {1, 2, 3, 5, 8, 9}  
       ar2[] = {10, 13, 15, 20}
```

This task is simple and O(m+n) if we are allowed to use extra space. But it becomes really complicated when extra space is not allowed and doesn't look possible in less than O(m*n) worst case time.

The idea is to begin from last element of ar2[] and search it in ar1[]. If there is a greater element in ar1[], then we move last element of ar1[] to ar2[]. To keep ar1[] and ar2[] sorted, we need to place last element of ar2[] at correct place in ar1[]. We can use [Insertion Sort](#) type of insertion for this. Below is algorithm:

- 1) Iterate through every element of ar2[] starting from last

```

element. Do following for every element ar2[i]
a) Store last element of ar1[i]: last = ar1[i]
b) Loop from last element of ar1[] while element ar1[j] is
   smaller than ar2[i].
      ar1[j+1] = ar1[j] // Move element one position ahead
      j--
c) If any element of ar1[] was moved or (j != m-1)
   ar1[j+1] = ar2[i]
   ar2[i] = last

```

In above loop, elements in ar1[] and ar2[] are always kept sorted.

Below is C++ and Java implementation of above algorithm.

C++

```

// C++ program to merge two sorted arrays with O(1) extra space.
#include <bits/stdc++.h>
using namespace std;

// Merge ar1[] and ar2[] with O(1) extra space
void merge(int ar1[], int ar2[], int m, int n)
{
    // Iterate through all elements of ar2[] starting from
    // the last element
    for (int i=n-1; i>=0; i--)
    {
        /* Find the smallest element greater than ar2[i]. Move all
           elements one position ahead till the smallest greater
           element is not found */
        int j, last = ar1[m-1];
        for (j=m-2; j >= 0 && ar1[j] > ar2[i]; j--)
            ar1[j+1] = ar1[j];

        // If there was a greater element
        if (j != m-2 || last > ar2[i])
        {
            ar1[j+1] = ar2[i];
            ar2[i] = last;
        }
    }
}

// Driver program
int main(void)
{
    int ar1[] = {1, 5, 9, 10, 15, 20};
    int ar2[] = {2, 3, 8, 13};
    int m = sizeof(ar1)/sizeof(ar1[0]);

```

```
int n = sizeof(ar2)/sizeof(ar2[0]);
merge(ar1, ar2, m, n);

cout << "After Merging nFirst Array: ";
for (int i=0; i<m; i++)
    cout << ar1[i] << " ";
cout << "nSecond Array: ";
for (int i=0; i<n; i++)
    cout << ar2[i] << " ";
return 0;
}
```

Java

```
// Java program program to merge two
// sorted arrays with O(1) extra space.

import java.util.Arrays;

class Test
{
    static int arr1[] = new int[]{1, 5, 9, 10, 15, 20};
    static int arr2[] = new int[]{2, 3, 8, 13};

    static void merge(int m, int n)
    {
        // Iterate through all elements of arr2[] starting from
        // the last element
        for (int i=n-1; i>=0; i--)
        {
            /* Find the smallest element greater than arr2[i]. Move all
               elements one position ahead till the smallest greater
               element is not found */
            int j, last = arr1[m-1];
            for (j=m-2; j >= 0 && arr1[j] > arr2[i]; j--)
                arr1[j+1] = arr1[j];

            // If there was a greater element
            if (j != m-2 || last > arr2[i])
            {
                arr1[j+1] = arr2[i];
                arr2[i] = last;
            }
        }
    }

    // Driver method to test the above function
    public static void main(String[] args)
```

```
{  
    merge(arr1.length,arr2.length);  
    System.out.print("After Merging nFirst Array: ");  
    System.out.println(Arrays.toString(arr1));  
    System.out.print("Second Array: ");  
    System.out.println(Arrays.toString(arr2));  
}  
}  
}
```

Python3

```
# Python program to merge  
# two sorted arrays  
# with O(1) extra space.  
  
# Merge ar1[] and ar2[]  
# with O(1) extra space  
def merge(ar1, ar2, m, n):  
  
    # Iterate through all  
    # elements of ar2[] starting from  
    # the last element  
    for i in range(n-1, -1, -1):  
  
        # Find the smallest element  
        # greater than ar2[i]. Move all  
        # elements one position ahead  
        # till the smallest greater  
        # element is not found  
        last = ar1[m-1]  
        j=m-2  
        while(j >= 0 and ar1[j] > ar2[i]):  
            ar1[j+1] = ar1[j]  
            j-=1  
  
        # If there was a greater element  
        if (j != m-2 or last > ar2[i]):  
  
            ar1[j+1] = ar2[i]  
            ar2[i] = last  
  
    # Driver program  
  
ar1 = [1, 5, 9, 10, 15, 20]  
ar2 = [2, 3, 8, 13]  
m = len(ar1)  
n = len(ar2)
```

```
merge(ar1, ar2, m, n)

print("After Merging \nFirst Array:", end="")
for i in range(m):
    print(ar1[i] , " ", end="")

print("\nSecond Array: ", end="")
for i in range(n):
    print(ar2[i] , " ", end="")

# This code is contributed
# by Anant Agarwal.
```

Output:

```
After Merging
First Array: 1 2 3 5 8 9
Second Array: 10 13 15 20
```

Time Complexity: The worst case time complexity of code/algorithm is $O(m*n)$. The worst case occurs when all elements of $ar1[]$ are greater than all elements of $ar2[]$.

Illustration:

Input:

ar1[]	1	5	9	10	15	20
ar2[]	2	3	8	13		



Ist iteration:

ar1[]	1	5	9	10	13	15
ar2[]	2	3	8	20		



IIInd Iteration:

ar1[]	1	5	8	9	10	13
ar2[]	2	3	15	20		



IIIrd Iteration:

ar1[]	1	3	5	8	9	10
ar2[]	2	13	15	20		



IVth Iteration:

ar1[]	1	2	3	5	8	9
ar2[]	10	13	15	20		

Source

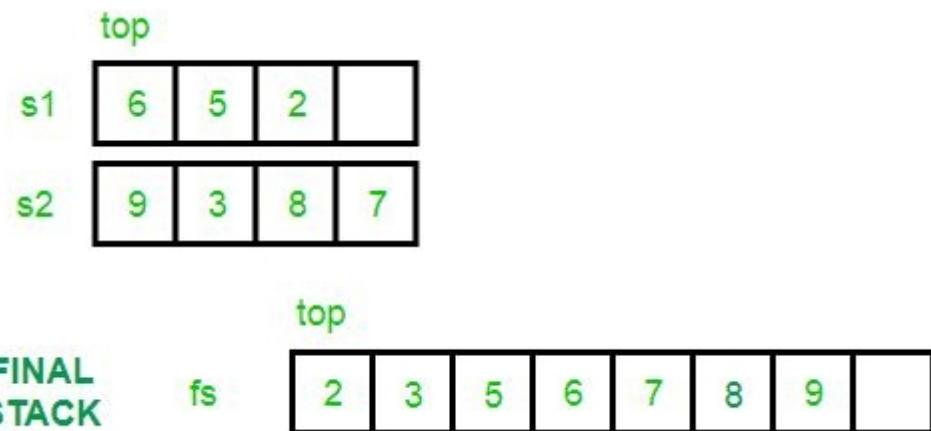
<https://www.geeksforgeeks.org/merge-two-sorted-arrays-o1-extra-space/>

Chapter 197

Merging and Sorting Two Unsorted Stacks

Merging and Sorting Two Unsorted Stacks - GeeksforGeeks

Given 2 input stacks with elements in an unsorted manner. Problem is to merge them into a new final stack, such that the elements become arranged in a sorted manner.



Examples:

Input : s1 : 9 4 2 1
s2: 8 17 3 10
Output : final stack: 1 2 3 4 8 9 10 17

Input : s1 : 5 7 2 6 4
s2 : 12 9 3

Output : final stack: 2 3 4 5 6 7 9 12

Create an empty stack to store result. We first insert elements of both stacks into the result. Then we [sort the result stack](#).

C++

```
// C++ program to merge two unsorted stacks
// into a third stack in sorted way.
#include <bits/stdc++.h>
using namespace std;

// Sorts input stack and returns sorted stack.
stack<int> sortStack(stack<int>& input)
{
    stack<int> tmpStack;

    while (!input.empty()) {
        // pop out the first element
        int tmp = input.top();
        input.pop();

        // while temporary stack is not empty and top
        // of stack is greater than temp
        while (!tmpStack.empty() && tmpStack.top() > tmp) {

            // pop from temporary stack and push
            // it to the input stack
            input.push(tmpStack.top());
            tmpStack.pop();
        }

        // push temp in temporary stack
        tmpStack.push(tmp);
    }

    return tmpStack;
}

stack<int> sortedMerge(stack<int>& s1, stack<int>& s2)
{
    // Push contents of both stacks in result
    stack<int> res;
    while (!s1.empty()) {
        res.push(s1.top());
        s1.pop();
    }
    while (!s2.empty()) {
```

```
        res.push(s2.top());
        s2.pop();
    }

    // Sort the result stack.
    return sortStack(res);
}

// main function
int main()
{
    stack<int> s1, s2;
    s1.push(34);
    s1.push(3);
    s1.push(31);

    s2.push(1);
    s2.push(12);
    s2.push(23);

    // This is the temporary stack
    stack<int> tmpStack = sortedMerge(s1, s2);
    cout << "Sorted and merged stack :\n";

    while (!tmpStack.empty()) {
        cout << tmpStack.top() << " ";
        tmpStack.pop();
    }
}
```

Java

```
// Java program to merge two unsorted stacks
// into a third stack in sorted way.
import java.io.*;
import java.util.*;

public class GFG {

    // This is the temporary stack
    static Stack<Integer> res = new Stack<Integer>();
    static Stack<Integer> tmpStack = new Stack<Integer>();

    // Sorts input stack and returns
    // sorted stack.
    static void sortStack(Stack<Integer> input)
    {
        while (input.size() != 0)
```

```
{  
    // pop out the first element  
    int tmp = input.peek();  
    input.pop();  
  
    // while temporary stack is not empty and  
    // top of stack is greater than temp  
    while (tmpStack.size() != 0 &&  
          tmpStack.peek() > tmp)  
    {  
  
        // pop from temporary stack and push  
        // it to the input stack  
        input.push(tmpStack.peek());  
        tmpStack.pop();  
    }  
  
    // push temp in temporary stack  
    tmpStack.push(tmp);  
}  
}  
  
static void sortedMerge(Stack<Integer> s1,  
                      Stack<Integer> s2)  
{  
    // Push contents of both stacks in result  
    while (s1.size() != 0) {  
        res.push(s1.peek());  
        s1.pop();  
    }  
  
    while (s2.size() != 0) {  
        res.push(s2.peek());  
        s2.pop();  
    }  
  
    // Sort the result stack.  
    sortStack(res);  
}  
  
// main function  
public static void main(String args[])  
{  
    Stack<Integer> s1 = new Stack<Integer>();  
    Stack<Integer> s2 = new Stack<Integer>();  
    s1.push(34);  
    s1.push(3);  
    s1.push(31);
```

```
s2.push(1);
s2.push(12);
s2.push(23);

sortedMerge(s1, s2);
System.out.println("Sorted and merged stack :");

while (tmpStack.size() != 0) {
    System.out.print(tmpStack.peek() + " ");
    tmpStack.pop();
}
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

C#

```
// C# program to merge two unsorted stacks
// into a third stack in sorted way.
using System;
using System.Collections.Generic;

class GFG {

    // Sorts input stack and returns
    // sorted stack.
    static Stack<int> sortStack(ref Stack<int> input)
    {
        Stack<int> tmpStack = new Stack<int>();

        while (input.Count != 0)
        {
            // pop out the first element
            int tmp = input.Peek();
            input.Pop();

            // while temporary stack is not empty and
            // top of stack is greater than temp
            while (tmpStack.Count != 0 &&
                    tmpStack.Peek() > tmp)
            {

                // pop from temporary stack and push
                // it to the input stack
                input.Push(tmpStack.Peek());
            }
        }
    }
}
```

```
        tmpStack.Pop();
    }

    // push temp in temporary of stack
    tmpStack.Push(tmp);
}

return tmpStack;
}

static Stack<int> sortedMerge(ref Stack<int> s1,
                               ref Stack<int> s2)
{
    // Push contents of both stacks in result
    Stack<int> res = new Stack<int>();
    while (s1.Count!=0) {
        res.Push(s1.Peek());
        s1.Pop();
    }
    while (s2.Count!=0) {
        res.Push(s2.Peek());
        s2.Pop();
    }

    // Sort the result stack.
    return sortStack(ref res);
}

// main function
static void Main()
{
    Stack<int> s1 = new Stack<int>();
    Stack<int> s2 = new Stack<int>();
    s1.Push(34);
    s1.Push(3);
    s1.Push(31);

    s2.Push(1);
    s2.Push(12);
    s2.Push(23);

    // This is the temporary stack
    Stack<int> tmpStack = new Stack<int>();
    tmpStack = sortedMerge(ref s1,ref s2);
    Console.WriteLine("Sorted and merged stack :\n");

    while (tmpStack.Count!=0) {
        Console.Write(tmpStack.Peek() + " ");
    }
}
```

```
        tmpStack.Pop();
    }
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

Output:

Sorted and merged stack :
34 31 23 12 3 1

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/merging-sorting-two-unsorted-stacks/>

Chapter 198

Merging two unsorted arrays in sorted order

Merging two unsorted arrays in sorted order - GeeksforGeeks

Write a **SortedMerge()** function that takes two lists, each of which is unsorted, and merges the two together into one new list which is in sorted (increasing) order. **SortedMerge()** should return the new list.

Examples :

```
Input : a[] = {10, 5, 15}
        b[] = {20, 3, 2}
Output : Merge List :
        {2, 3, 5, 10, 15, 20}

Input : a[] = {1, 10, 5, 15}
        b[] = {20, 0, 2}
Output : Merge List :
        {0, 1, 2, 5, 10, 15, 20}
```

There are many cases to deal with: either ‘a’ or ‘b’ may be empty, during processing either ‘a’ or ‘b’ may run out first, and finally there’s the problem of starting the result list empty, and building it up while going through ‘a’ and ‘b’.

Method 1 (first Concatenate then Sort)

In this case, we first append the two unsorted lists. Then we simply sort the concatenated list.

C++

```
// CPP program to merge two unsorted lists
```

```
// in sorted order
#include <bits/stdc++.h>
using namespace std;

// Function to merge array in sorted order
void sortedMerge(int a[], int b[], int res[],
                 int n, int m)
{
    // Concatenate two arrays
    int i = 0, j = 0, k = 0;
    while (i < n) {
        res[k] = a[i];
        i += 1;
        k += 1;
    }
    while (j < m) {
        res[k] = b[j];
        j += 1;
        k += 1;
    }

    // sorting the res array
    sort(res, res + n + m);
}

// Driver code
int main()
{
    int a[] = { 10, 5, 15 };
    int b[] = { 20, 3, 2, 12 };
    int n = sizeof(a) / sizeof(a[0]);
    int m = sizeof(b) / sizeof(b[0]);

    // Final merge list
    int res[n + m];
    sortedMerge(a, b, res, n, m);

    cout << "Sorted merged list :";
    for (int i = 0; i < n + m; i++)
        cout << " " << res[i];
    cout << "n";
}

return 0;
}
```

Java

```
// Java Code for Merging two unsorted
```

```
// arrays in sorted order
import java.util.*;

class GFG {

    // Function to merge array in sorted order
    public static void sortedMerge(int a[], int b[],
                                   int res[], int n,
                                   int m)
    {
        // Concatenate two arrays
        int i = 0, j = 0, k = 0;
        while (i < n) {
            res[k] = a[i];
            i++;
            k++;
        }

        while (j < m) {
            res[k] = b[j];
            j++;
            k++;
        }

        // sorting the res array
        Arrays.sort(res);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int a[] = { 10, 5, 15 };
        int b[] = { 20, 3, 2, 12 };
        int n = a.length;
        int m = b.length;

        // Final merge list
        int res[] = new int[n + m];
        sortedMerge(a, b, res, n, m);

        System.out.print("Sorted merged list :");
        for (int i = 0; i < n + m; i++)
            System.out.print(" " + res[i]);
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python

```
# Python program to merge two unsorted lists
# in sorted order

# Function to merge array in sorted order
def sortedMerge(a, b, res, n, m):
    # Concatenate two arrays
    i, j, k = 0, 0, 0
    while (i < n):
        res[k] = a[i]
        i += 1
        k += 1
    while (j < m):
        res[k] = b[j]
        j += 1
        k += 1

    # sorting the res array
    res.sort()

# Driver code
a = [ 10, 5, 15 ]
b = [ 20, 3, 2, 12 ]
n = len(a)
m = len(b)

# Final merge list
res = [0 for i in range(n + m)]
sortedMerge(a, b, res, n, m)
print "Sorted merged list :"
for i in range(n + m):
    print res[i],
```

This code is contributed by Sachin Bisht

C#

```
// C# Code for Merging two
// unsorted arrays in sorted order
using System;

class GFG {

    // Function to merge array in sorted order
    public static void sortedMerge(int []a, int []b,
                                int []res, int n,
```

```
        int m)
{
    // Concatenate two arrays
    int i = 0, j = 0, k = 0;
    while (i < n) {
        res[k] = a[i];
        i++;
        k++;
    }

    while (j < m) {
        res[k] = b[j];
        j++;
        k++;
    }

    // sorting the res array
    Array.Sort(res);
}

/* Driver program to test above function */
public static void Main()
{
    int []a = {10, 5, 15};
    int []b = {20, 3, 2, 12};
    int n = a.Length;
    int m = b.Length;

    // Final merge list
    int []res=new int[n + m];
    sortedMerge(a, b, res, n, m);

    Console.WriteLine("Sorted merged list :");
    for (int i = 0; i < n + m; i++)
        Console.Write(" " + res[i]);
}
}

// This code is contributed by nitin mittal.
```

Output :

Sorted merged list : 2 3 5 10 12 15 20

Time Complexity : O ((n + m) (log(n + m)))
Auxiliary Space : O ((n + m))

Method 2 (First Sort then Merge)

We first sort both the given arrays separately. Then we simply merge two sorted arrays.

C++

```
// CPP program to merge two unsorted lists
// in sorted order
#include <bits/stdc++.h>
using namespace std;

// Function to merge array in sorted order
void sortedMerge(int a[], int b[], int res[],
                 int n, int m)
{
    // Sorting a[] and b[]
    sort(a, a + n);
    sort(b, b + m);

    // Merge two sorted arrays into res[]
    int i = 0, j = 0, k = 0;
    while (i < n && j < m) {
        if (a[i] <= b[j]) {
            res[k] = a[i];
            i += 1;
            k += 1;
        } else {
            res[k] = b[j];
            j += 1;
            k += 1;
        }
    }
    while (i < n) { // Merging remaining
                    // elements of a[] (if any)
        res[k] = a[i];
        i += 1;
        k += 1;
    }
    while (j < m) { // Merging remaining
                    // elements of b[] (if any)
        res[k] = b[j];
        j += 1;
        k += 1;
    }
}

// Driver code
```

```
int main()
{
    int a[] = { 10, 5, 15 };
    int b[] = { 20, 3, 2, 12 };
    int n = sizeof(a) / sizeof(a[0]);
    int m = sizeof(b) / sizeof(b[0]);

    // Final merge list
    int res[n + m];

    sortedMerge(a, b, res, n, m);

    cout << "Sorted merge list :";
    for (int i = 0; i < n + m; i++)
        cout << " " << res[i];
    cout << "\n";

    return 0;
}
```

Java

```
// JAVA Code for Merging two unsorted
// arrays in sorted order
import java.util.*;

class GFG {

    // Function to merge array in sorted order
    public static void sortedMerge(int a[], int b[],
                                   int res[], int n,
                                   int m)
    {
        // Sorting a[] and b[]
        Arrays.sort(a);
        Arrays.sort(b);

        // Merge two sorted arrays into res[]
        int i = 0, j = 0, k = 0;
        while (i < n && j < m) {
            if (a[i] <= b[j]) {
                res[k] = a[i];
                i += 1;
                k += 1;
            } else {
                res[k] = b[j];
                j += 1;
                k += 1;
            }
        }
    }
}
```

```
        }
    }

    while (i < n) { // Merging remaining
                    // elements of a[] (if any)
        res[k] = a[i];
        i += 1;
        k += 1;
    }
    while (j < m) { // Merging remaining
                    // elements of b[] (if any)
        res[k] = b[j];
        j += 1;
        k += 1;
    }
}

/* Driver program to test above function */
public static void main(String[] args)
{
    int a[] = { 10, 5, 15 };
    int b[] = { 20, 3, 2, 12 };
    int n = a.length;
    int m = b.length;

    // Final merge list
    int res[] = new int[n + m];
    sortedMerge(a, b, res, n, m);

    System.out.print( "Sorted merged list :");
    for (int i = 0; i < n + m; i++)
        System.out.print(" " + res[i]);
}
}

// This code is contributed by Arnav Kr. Mandal.
```

Python

```
# Python program to merge two unsorted lists
# in sorted order

# Function to merge array in sorted order
def sortedMerge(a, b, res, n, m):
    # Sorting a[] and b[]
    a.sort()
    b.sort()

    # Merge two sorted arrays into res[]
```

```
i, j, k = 0, 0, 0
while (i < n and j < m):
    if (a[i] <= b[j]):
        res[k] = a[i]
        i += 1
        k += 1
    else:
        res[k] = b[j]
        j += 1
        k += 1

while (i < n): # Merging remaining
    # elements of a[] (if any)
    res[k] = a[i]
    i += 1
    k += 1

while (j < m): # Merging remaining
    # elements of b[] (if any)
    res[k] = b[j]
    j += 1
    k += 1

# Driver code
a = [ 10, 5, 15 ]
b = [ 20, 3, 2, 12 ]
n = len(a)
m = len(b)

# Final merge list
res = [0 for i in range(n + m)]
sortedMerge(a, b, res, n, m)
print "Sorted merged list :"
for i in range(n + m):
    print res[i],
```

This code is contributed by Sachin Bisht

C#

```
// C# Code for Merging two unsorted
// arrays in sorted order
using System;

class GFG {

    // Function to merge array in
    // sorted order
```

```
static void sortedMerge(int []a, int []b,
                      int []res, int n, int m)
{
    // Sorting a[] and b[]
    Array.Sort(a);
    Array.Sort(b);

    // Merge two sorted arrays into res[]
    int i = 0, j = 0, k = 0;

    while (i < n && j < m)
    {
        if (a[i] <= b[j])
        {
            res[k] = a[i];
            i += 1;
            k += 1;
        }
        else
        {
            res[k] = b[j];
            j += 1;
            k += 1;
        }
    }

    while (i < n)
    {

        // Merging remaining
        // elements of a[] (if any)
        res[k] = a[i];
        i += 1;
        k += 1;
    }

    while (j < m)
    {

        // Merging remaining
        // elements of b[] (if any)
        res[k] = b[j];
        j += 1;
        k += 1;
    }
}

/* Driver program to test
```

```
above function */
public static void Main()
{
    int []a = { 10, 5, 15 };
    int []b = { 20, 3, 2, 12 };
    int n = a.Length;
    int m = b.Length;

    // Final merge list
    int []res = new int[n + m];
    sortedMerge(a, b, res, n, m);

    Console.WriteLine("Sorted merged list :");
    for (int i = 0; i < n + m; i++)
        Console.Write(" " + res[i]);
}
}

// This code is contributed by nitin mittal.
```

Output :

```
Sorted merge list : 2 3 5 10 12 15 20
```

Time Complexity : $O(n \log n + m \log m + (n + m))$
Space Complexity : $O(n + m)$

It is obvious from above time complexities that **method 2 is better than method 1.**

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/merging-two-unsorted-arrays-sorted-order/>

Chapter 199

Minimize the sum of product of two arrays with permutations allowed

Minimize the sum of product of two arrays with permutations allowed - GeeksforGeeks

Given two arrays, A and B, of equal size n, the task is to find the minimum value of $A[0] * B[0] + A[1] * B[1] + \dots + A[n-1] * B[n-1]$. Shuffling of elements of arrays A and B is allowed.

Examples :

Input : A[] = {3, 1, 1} and B[] = {6, 5, 4}.

Output : 23

Minimum value of S = $1*6 + 1*5 + 3*4 = 23$.

Input : A[] = { 6, 1, 9, 5, 4 } and B[] = { 3, 4, 8, 2, 4 }

Output : 80.

Minimum value of S = $1*8 + 4*4 + 5*4 + 6*3 + 9*2 = 80$.

The idea is to multiply minimum element of one array to maximum element of another array.
Algorithm to solve this problem:

1. Sort both the arrays A and B.
2. Traverse the array and for each element, multiply $A[i]$ and $B[n - i - 1]$ and add to the total.

Below is the implementation of this approach:

C/C++

```
// C++ program to calculate minimum sum of product
// of two arrays.
#include <bits/stdc++.h>
using namespace std;

// Returns minimum sum of product of two arrays
// with permutations allowed
int minValue(int A[], int B[], int n)
{
    // Sort A and B so that minimum and maximum
    // value can easily be fetched.
    sort(A, A + n);
    sort(B, B + n);

    // Multiplying minimum value of A and maximum
    // value of B
    int result = 0;
    for (int i = 0; i < n; i++)
        result += (A[i] * B[n - i - 1]);

    return result;
}

// Driven Program
int main()
{
    int A[] = { 3, 1, 1 };
    int B[] = { 6, 5, 4 };
    int n = sizeof(A) / sizeof(A[0]);
    cout << minValue(A, B, n) << endl;
    return 0;
}
```

Java

```
// java program to calculate minimum
// sum of product of two arrays.
import java.io.*;
import java.util.*;

class GFG {

    // Returns minimum sum of product of two arrays
    // with permutations allowed
    static int minValue(int A[], int B[], int n)
    {
        // Sort A and B so that minimum and maximum
        // value can easily be fetched.
```

```
Arrays.sort(A);
Arrays.sort(B);

// Multiplying minimum value of A
// and maximum value of B
int result = 0;
for (int i = 0; i < n; i++)
    result += (A[i] * B[n - i - 1]);

return result;
}

// Driven Program
public static void main(String[] args)
{
    int A[] = { 3, 1, 1 };
    int B[] = { 6, 5, 4 };
    int n = A.length;
    ;
    System.out.println(minValue(A, B, n));
}
}

// This code is contributed by vt_m
```

Python

```
# Python program to calculate minimum sum of product
# of two arrays.

# Returns minimum sum of product of two arrays
# with permutations allowed
def minValue(A, B, n):
    # Sort A and B so that minimum and maximum
    # value can easily be fetched.
    sorted(A)
    sorted(B)

    # Multiplying minimum value of A and maximum
    # value of B
    result = 0
    for i in range(n):
        result += (A[i] * B[n - i - 1])

    return result

# Driven Program
A = [3, 1, 1]
```

```
B = [6, 5, 4]
n = len(A)
print minValue(A, B, n)

# Contributed by: Afzal Ansari
```

C#

```
// C# program to calculate minimum
// sum of product of two arrays.
using System;

class GFG {

    // Returns minimum sum of product
    // of two arrays with permutations
    // allowed
    static int minValue(int[] a, int[] b,
                        int n)
    {

        // Sort A and B so that minimum
        // and maximum value can easily
        // be fetched.
        Array.Sort(a);
        Array.Sort(b);

        // Multiplying minimum value of
        // A and maximum value of B
        int result = 0;

        for (int i = 0; i < n; i++)
            result += (a[i] * b[n - i - 1]);

        return result;
    }

    // Driven Program
    public static void Main()
    {

        int[] a = { 3, 1, 1 };
        int[] b = { 6, 5, 4 };
        int n = a.Length;

        Console.WriteLine(minValue(a, b, n));
    }
}
```

```
// This code is contributed by nitin mittal.
```

PHP

```
<?php
// PHP program to calculate minimum
// sum of product of two arrays.

// Returns minimum sum of
// product of two arrays
// with permutations allowed
function minValue($A, $B, $n)
{
    // Sort A and B so that minimum
    // and maximum value can easily
    // be fetched.
    sort($A); sort($A , $n);
    sort($B); sort($B , $n);

    // Multiplying minimum value of
    // A and maximum value of B
    $result = 0;
    for ($i = 0; $i < $n; $i++)
        $result += ($A[$i] *
                    $B[$n - $i - 1]);

    return $result;
}

// Driver Code
$A = array( 3, 1, 1 );
$B = array( 6, 5, 4 );
$n = sizeof($A) / sizeof($A[0]);
echo minValue($A, $B, $n) ;

// This code is contributed by nitin mittal.
?>
```

Output :

23

Time Complexity : $O(n \log n)$.

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/minimize-sum-product-two-arrays-permutations-allowed/>

Chapter 200

Minimum De-arrangements present in array of AP (Arithmetic Progression)

Minimum De-arrangements present in array of AP (Arithmetic Progression) - GeeksforGeeks

Given an array of n-elements. Given array is a permutation of some Arithmetic Progression. Find the minimum number of De-arrangements present in that array so as to make that array an Arithmetic progression.

Examples:

Input : arr[] = [8, 6, 10 ,4, 2]
Output : Minimum De-arrangement = 3
Explanation : arr[] = [10, 8, 6, 4, 2] is permutation
which forms an AP and has minimum de-arrangements.

Input : arr[] = [5, 10, 15, 25, 20]
Output : Minimum De-arrangement = 2
Explanation : arr[] = [5, 10, 15, 20, 25] is permutation
which forms an AP and has minimum de-arrangements.

As per property of Arithmetic Progression our sequence will be either in increasing or decreasing manner. Also, we know that reverse of any Arithmetic Progression also form another Arithmetic Progression. So, we create a copy of original array and then once sort our given array in increase order and find total count of mismatch again after that we will reverse our sorted array and found new count of mismatch. Comparing both the counts of mismatch we can find the minimum number of de-arrangements. Time Complexity = O(nlogn).

C++

```
// CPP for counting minimum de-arrangements present
// in an array.
#include<bits/stdc++.h>
using namespace std;

// function to count Dearrangement
int countDe (int arr[], int n)
{
    // create a copy of original array
    vector <int> v (arr, arr+n);

    // sort the array
    sort(arr, arr+n);

    // traverse sorted array for counting mismatches
    int count1 = 0;
    for (int i=0; i<n; i++)
        if (arr[i] != v[i])
            count1++;

    // reverse the sorted array
    reverse(arr,arr+n);

    // traverse reverse sorted array for counting
    // mismatches
    int count2 = 0;
    for (int i=0; i<n; i++)
        if (arr[i] != v[i])
            count2++;

    // return minimum mismatch count
    return (min (count1, count2));
}

// driver program
int main()
{
    int arr[] = {5, 9, 21, 17, 13};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Minimum Dearrangement = " << countDe(arr, n);
    return 0;
}
```

Java

```
// Java code for counting minimum
// de-arrangements present in an array.
import java.util.*;
```

```
import java.lang.*;
import java.util.Arrays;

public class GeeksforGeeks{

    // function to count Dearrangement
    public static int countDe(int arr[], int n){
        int v[] = new int[n];

        // create a copy of original array
        for(int i = 0; i < n; i++)
            v[i] = arr[i];

        // sort the array
        Arrays.sort(arr);

        // traverse sorted array for
        // counting mismatches
        int count1 = 0;
        for (int i = 0; i < n; i++)
            if (arr[i] != v[i])
                count1++;

        // reverse the sorted array
        Collections.reverse(Arrays.asList(arr));

        // traverse reverse sorted array
        // for counting mismatches
        int count2 = 0;
        for (int i = 0; i < n; i++)
            if (arr[i] != v[i])
                count2++;

        // return minimum mismatch count
        return (Math.min (count1, count2));
    }

    // driver code
    public static void main(String argc[]){
        int arr[] = {5, 9, 21, 17, 13};
        int n = 5;
        System.out.println("Minimum Dearrangement = "+
                           countDe(arr, n));
    }
}

/*This code is contributed by Sagar Shukla.*/
```

Python3

```
# Python3 code for counting minimum
# de-arrangements present in an array.

# function to count Dearrangement
def countDe(arr, n):
    v = [None] * n

    i=0

    # create a copy of
    # original array
    while(i < n):
        v[i] = arr[i]
        i = i + 1

    # sort the array
    arr.sort()

    # traverse sorted array for
    # counting mismatches
    count1 = 0
    i = 0
    while( i < n ):
        if (arr[i] != v[i]):
            count1 = count1 + 1
        i = i + 1

    # reverse the sorted array
    arr.sort(reverse=True)

    # traverse reverse sorted array
    # for counting mismatches
    count2 = 0
    i = 0
    while( i < n ):
        if (arr[i] != v[i]):
            count2 = count2 + 1
        i = i + 1

    # return minimum mismatch count
    return (min (count1, count2))

# Driven code
arr = [5, 9, 21, 17, 13]
n = 5
print ("Minimum Dearrangement =",countDe(arr, n))
```

```
# This code is contributed by "rishabh_jain".
```

C#

```
// C# code for counting
// minimum de-arrangements
// present in an array.
using System;

class GFG
{

    // function to count
    // Dearrangement
    public static int countDe(int[] arr,
                           int n)
    {
        int[] v = new int[n];

        // create a copy
        // of original array
        for(int i = 0; i < n; i++)
            v[i] = arr[i];

        // sort the array
        Array.Sort(arr);

        // traverse sorted array for
        // counting mismatches
        int count1 = 0;
        for (int i = 0; i < n; i++)
            if (arr[i] != v[i])
                count1++;

        // reverse the sorted array
        Array.Reverse(arr);

        // traverse reverse sorted array
        // for counting mismatches
        int count2 = 0;
        for (int i = 0; i < n; i++)
            if (arr[i] != v[i])
                count2++;

        // return minimum
        // mismatch count
        return (Math.Min (count1, count2));
    }
}
```

```
}

// Driver code
public static void Main()
{
    int[] arr = new int[]{5, 9, 21, 17, 13};
    int n = 5;
    Console.WriteLine("Minimum Dearrangement = " +
                      countDe(arr, n));
}

// This code is contributed by mits
```

Output:

Minimum Dearrangement = 2

Improved By : [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/minimum-de-arrangements-present-array-ap-arithmetic-progression/>

Chapter 201

Minimum cost to sort a matrix of numbers from 0 to $n^2 - 1$

Minimum cost to sort a matrix of numbers from 0 to $n^2 - 1$ - GeeksforGeeks

Given an $n \times n$ matrix containing all the numbers in the range 0 to $n^2 - 1$. The problem is to calculate the total energy required for rearranging all the numbers in the matrix in strictly increasing order, i.e., after the rearrangement the 1st row contains 'n' numbers from 0 to $n-1$, then 2nd row from n to $2n-1$ and so on up to the last or nth row. A number can be moved in any of the four directions left, right, top or bottom from its current position to reach its destination in the final modified matrix. The number of steps moved in transferring a number from its current location to its required destination is the energy required by the number for its movement. For example, in a 4×4 matrix, number '6' is present at location (2, 3). Its destination location in the modified matrix is (1, 1). So '6' is moved 2 steps towards left and 1 step up to reach location (1, 1). Total 3 steps moved and thus energy required by '6' is 3 units. In this way we have to sum up all the energies required in the movement / rearrangement of all the numbers.

Examples :

```
Input : mat[][] = { {3, 0},  
                   {2, 1} }  
Output : 4 units  
For number '3':  
Move it one step right and one step down.  
Total 2 steps thus energy required = 2 units.  
  
For number '0':  
Move it one step left.  
Total 1 step thus energy required = 1 unit.  
  
For number '1':
```

Move it one step up.

Total 1 step thus energy required = 1 unit.

Total energy required = 4 units.

```
Input : mat[] [] = { {4, 7, 10, 3},
                     {8, 5, 6, 1},
                     {9, 11, 10, 2},
                     {15, 13, 14, 12} }
```

Output : 22 units

Algorithm:

```
calculateEnergy(mat, n)
    Declare i_des, j_des, q
    Initialize tot_energy = 0
    for i = 0 to n-1
        for j = 0 to n-1
            q = mat[i][j] / n
            i_des = q
            j_des = mat[i][j] - (n * q)
            tot_energy += abs(i_des - i) + abs(j_des - j)
    return tot_energy
```

C++

```
// C++ implementation to find the total energy
// required to rearrange the numbers
#include <bits/stdc++.h>

using namespace std;

#define SIZE 100

// function to find the total energy
// required to rearrange the numbers
int calculateEnergy(int mat[SIZE][SIZE], int n)
{
    int i_des, j_des, q;
    int tot_energy = 0;

    // nested loops to access the elements
    // of the given matrix
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            // store quotient
```

```
q = mat[i][j] / n;

// final destination location (i_des, j_des) of
// the element mat[i][j] is being calculated
i_des = q;
j_des = mat[i][j] - (n * q);

// energy required for the movement of the
// element mat[i][j] is calculated and then
// accumulated in the 'tot_energy'
tot_energy += abs(i_des - i) + abs(j_des - j);
}

}

// required total energy
return tot_energy;
}

// Driver program to test above
int main()
{
    int mat[SIZE][SIZE] = { { 4, 7, 0, 3 },
                           { 8, 5, 6, 1 },
                           { 9, 11, 10, 2 },
                           { 15, 13, 14, 12 } };
    int n = 4;

    cout << "Total energy required = "
         << calculateEnergy(mat, n) << " units";

    return 0;
}
```

Java

```
// Java implementation to find
// the total energy required
// to rearrange the numbers

import java.util.*;
import java.lang.*;

public class GfG{

    private final static int SIZE = 100;

    // function to find the total energy
    // required to rearrange the numbers
```

```
public static int calculateEnergy(int mat[][],
int n)
{
    int i_des, j_des, q;
    int tot_energy = 0;

    // nested loops to access the elements
    // of the given matrix
    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            // store quotient
            q = mat[i][j] / n;

            // final destination location
            // (i_des, j_des) of
            // the element mat[i][j] is
            // being calculated
            i_des = q;
            j_des = mat[i][j] - (n * q);

            // energy required for the
            // movement of the
            // element mat[i][j] is
            // calculated and then
            // accumulated in the 'tot_energy'
            tot_energy += Math.abs(i_des - i) +
            Math.abs(j_des - j);
        }
    }

    // required total energy
    return tot_energy;
}

// Driver function
public static void main(String argc[]){

    int[][] mat = new int[][] {{ 4, 7, 0, 3 },
                               { 8, 5, 6, 1 },
                               { 9, 11, 10, 2 },
                               { 15, 13, 14, 12 }};

    int n = 4;

    System.out.println("Total energy required = "
        + calculateEnergy(mat, n) + " units");
}
```

```
}
```

```
}
```

```
// This code is contributed by Sagar Shukla
```

Python3

```
# implementation to find the total
# energy required to rearrange the
# numbers
n = 4

# function to find the total energy
# required to rearrange the numbers
def calculateEnergy(mat,n):

    tot_energy = 0

    # nested loops to access the
    # elements of the given matrix
    for i in range(n):
        for j in range(n):

            #store quotient
            q = mat[i][j]//n

            # final destination location
            # (i_des, j_des) of the
            # element mat[i][j] is being
            # calculated
            i_des = q
            j_des = mat[i][j]- (n*q)

            # energy required for the
            # movement of the element
            # mat[i][j] is calculated
            # and then accumulated in
            # the 'tot_energy'
            tot_energy += (abs(i_des-i)
                           + abs(j_des-j))

    # required total energy
    return tot_energy

# Driver Program
mat = [[4, 7, 0, 3],
       [8, 5, 6, 1],
```

```
[9, 11, 10, 2],  
[15, 13, 14, 12]]  
print("Total energy required = ",  
      calculateEnergy(mat,n), "units")  
  
# This code is contributed by Shrikant13.
```

C#

```
// C# implementation to find  
// the total energy required  
// to rearrange the numbers  
using System;  
  
class GFG {  
  
    // function to find the total energy  
    // required to rearrange the numbers  
    public static int calculateEnergy(int[, ] mat,  
                                      int n)  
    {  
        int i_des, j_des, q;  
        int tot_energy = 0;  
  
        // nested loops to access the elements  
        // of the given matrix  
        for (int i = 0; i < n; i++) {  
  
            for (int j = 0; j < n; j++) {  
  
                // store quotient  
                q = mat[i, j] / n;  
  
                // final destination location  
                // (i_des, j_des) of  
                // the element mat[i][j] is  
                // being calculated  
                i_des = q;  
                j_des = mat[i, j] - (n * q);  
  
                // energy required for the  
                // movement of the  
                // element mat[i][j] is  
                // calculated and then  
                // accumulated in the 'tot_energy'  
                tot_energy += Math.Abs(i_des - i) +  
                              Math.Abs(j_des - j);  
            }  
        }  
    }  
}
```

```
}

// required total energy
return tot_energy;
}

// Driver function
public static void Main()
{
    int[, ] mat = new int[, ]{ { 4, 7, 0, 3 },
                                { 8, 5, 6, 1 },
                                { 9, 11, 10, 2 },
                                { 15, 13, 14, 12 } };

    int n = 4;

    Console.WriteLine("Total energy required = " +
                      calculateEnergy(mat, n) + " units");
}
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP implementation to find
// the total energy required
// to rearrange the numbers

// function to find the total energy
// required to rearrange the numbers
function calculateEnergy($mat, $n)
{
    $i_des; $j_des; $q;
    $tot_energy = 0;

    // nested loops to access the
    // elements of the given matrix
    for ($i = 0; $i < $n; $i++)
    {
        for ($j = 0; $j < $n; $j++)
        {
            // store quotient
            $q = (int)($mat[$i][$j] / $n);

            // final destination location
```

```
// (i_des, j_des) of the element
// mat[i][j] is being calculated
$i_des = $q;
$j_des = $mat[$i][$j] - ($n * $q);

// energy required for the movement
// of the element mat[i][j] is
// calculated and then accumulated
// in the 'tot_energy'
$tot_energy += abs($i_des - $i) +
               abs($j_des - $j);
}

}

// required total energy
return $tot_energy;
}

// Driver Code
$mat = array(array (4, 7, 0, 3),
             array (8, 5, 6, 1),
             array (9, 11, 10, 2),
             array (15, 13, 14, 12));
$n = 4;

echo "Total energy required = ",
calculateEnergy($mat, $n) , " units";

// This code is contributed by ajit
?>
```

Output :

Total energy required = 22 units

Time Complexity : $O(n^2)$

Improved By : shrikant13, jit_t

Source

<https://www.geeksforgeeks.org/minimum-cost-sort-matrix-numbers-0-n2-1/>

Chapter 202

Minimum difference between groups of size two

Minimum difference between groups of size two - GeeksforGeeks

Given an array of even number of elements, form groups of 2 using these array elements such that the difference between the group with highest sum and the one with lowest sum is minimum.

Note: An element can be a part of one group only and it has to be a part of at least 1 group.

Examples:

```
Input : arr[] = {2, 6, 4, 3}
Output : 1
Groups formed will be (2, 6) and (4, 3),
the difference between highest sum group
(2, 6) i.e 8 and lowest sum group (3, 4)
i.e 7 is 1.
```

```
Input : arr[] = {11, 4, 3, 5, 7, 1}
Output : 3
Groups formed will be (1, 11), (4, 5) and
(3, 7), the difference between highest
sum group (1, 11) i.e 12 and lowest sum
group (4, 5) i.e 9 is 3.
```

Simple Approach: A simple approach would be to try against all combinations of array elements and check against each set of combination difference between the group with the highest sum and the one with lowest sum. A total of $n*(n-1)/2$ such groups would be formed ($nC2$).

Time Complexity : $O(n^3)$ To generate groups n^2 iterations will be needed and to check against each group n iterations will be needed and hence n^3 iterations will be needed in worst case.

Efficient Approach: Efficient approach would be to use the greedy approach. Sort the whole array and generate groups by selecting one element from the start of the array and one from the end.

```
// CPP program to find minimum difference
// between groups of highest and lowest
// sums.
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

ll calculate(ll a[], ll n)
{
    // Sorting the whole array.
    sort(a, a + n);

    // Generating sum groups.
    vector<ll> s;
    for (int i = 0, j = n - 1; i < j; i++, j--)
        s.push_back(a[i] + a[j]);

    ll mini = *min_element(s.begin(), s.end());
    ll maxi = *max_element(s.begin(), s.end());

    return abs(maxi - mini);
}

int main()
{
    ll a[] = { 2, 6, 4, 3 };
    int n = sizeof(a) / (sizeof(a[0]));
    cout << calculate(a, n) << endl;
    return 0;
}
```

Output:

1

Time Complexity: $O(n * \log n)$

Asked in: Inmobi

Reference: <https://www.hackerearth.com/problem/algorithm/project-team/>

Source

<https://www.geeksforgeeks.org/minimum-difference-between-groups-of-size-two/>

Chapter 203

Minimum difference between max and min of all K-size subsets

Minimum difference between max and min of all K-size subsets - GeeksforGeeks

Given an array of integer values, we need to find the minimum difference between maximum and minimum of all possible K-length subsets.

Examples :

```
Input : arr[] = [3, 5, 100, 101, 102]
        K = 3
Output : 2
```

Explanation : Possible subsets of K-length with their differences are,

[3 5 100] max min diff is (100 - 3) = 97
[3 5 101] max min diff is (101 - 3) = 98
[3 5 102] max min diff is (102 - 3) = 99
[3 100 101] max min diff is (101 - 3) = 98
[3 100 102] max min diff is (102 - 3) = 99
[3 101 102] max min diff is (102 - 3) = 98
[5 100 101] max min diff is (101 - 5) = 96
[5 100 102] max min diff is (102 - 5) = 97
[5 101 102] max min diff is (102 - 5) = 97
[100 101 102] max min diff is (102 - 100) = 2
As the minimum difference is 2, it should be the answer for given array.

```
Input : arr[] = {5, 1, 10, 6}
```

```
k = 2
Output : 1
```

We get the above result considering subset
{5, 6}

We can solve this problem without iterating over all possible subsets by observing the fact that our result subset will always be consecutive, once we sort the given array. The reason is sorting brings value-wise close elements together.

We can prove above fact as follows – Suppose we chose number a1, a2, a3 ... aK which are in increasing order but not continuous, then our difference will be (aK – a1) but if we include the number which was not taken earlier (let aR) then our K length subset will be a2, a3, ... aR, ... aK. In this case, our difference will (aK – a2) which must be smaller than (aK – a1) because a2 > a1. So we can say that the subset which will contain our answer will always be consecutive in sorted array.

Stating above fact, for solving the problem first we sort the array then we will iterate over first (N – K) elements and each time we will take the difference between elements which are K distant apart and our final answer will be minimum of them.

C++

```
// C++ program to find minimum difference
// between max and min of all subset of K size
#include <bits/stdc++.h>

using namespace std;

// returns min difference between max
// and min of any K-size subset
int minDifferenceAmongMaxMin(int arr[], int N, int K)
{
    // sort the array so that close
    // elements come together.
    sort(arr, arr + N);

    // initialize result by a big integer number
    int res = INT_MAX;

    // loop over first (N - K) elements
    // of the array only
    for (int i = 0; i <= (N - K); i++)
    {
        // get difference between max and
        // min of current K-sized segment
        int curSeqDiff = arr[i + K - 1] - arr[i];
        res = min(res, curSeqDiff);
    }
}
```

```
    return res;
}

// Driver code
int main()
{
    int arr[] = {10, 20, 30, 100, 101, 102};
    int N = sizeof(arr) / sizeof(arr[0]);

    int K = 3;
    cout << minDifferenceAmongMaxMin(arr, N, K);
    return 0;
}
```

Java

```
// Java program to find minimum difference
// between max and min of all subset of
// K size
import java.util.Arrays;

class GFG
{

    // returns min difference between max
    // and min of any K-size subset
    static int minDifferenceAmongMaxMin(int arr[],
                                         int N, int K)
    {

        // sort the array so that close
        // elements come together.
        Arrays.sort(arr);

        // initialize result by
        // a big integer number
        int res = 2147483647;

        // loop over first (N - K) elements
        // of the array only
        for (int i = 0; i <= (N - K); i++)
        {

            // get difference between max and
            // min of current K-sized segment
            int curSeqDiff = arr[i + K - 1] - arr[i];
            res = Math.min(res, curSeqDiff);
        }
    }
}
```

```
        return res;
    }

// Driver code
public static void main(String[] args)
{
    int arr[] = {10, 20, 30, 100, 101, 102};
    int N = arr.length;

    int K = 3;
    System.out.print(
        minDifferenceAmongMaxMin(arr, N, K));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program to find minimum
# difference between max and min
# of all subset of K size

# Returns min difference between max
# and min of any K-size subset
def minDifferenceAmongMaxMin(arr, N, K):

    # sort the array so that close
    # elements come together.
    arr.sort()

    # initialize result by a
    # big integer number
    res = 2147483647

    # loop over first (N - K) elements
    # of the array only
    for i in range((N - K) + 1):

        # get difference between max and min
        # of current K-sized segment
        curSeqDiff = arr[i + K - 1] - arr[i]
        res = min(res, curSeqDiff)

    return res

# Driver Code
```

```
arr = [10, 20, 30, 100, 101, 102]
N = len(arr)
K = 3
print(minDifferenceAmongMaxMin(arr, N, K))

# This code is contributed by Anant Agarwal.
```

C#

```
// C# program to find minimum difference
// between max and min of all subset of
// K size
using System;

class GFG
{

    // returns min difference between max
    // and min of any K-size subset
    static int minDifferenceAmongMaxMin(int []arr,
                                         int N, int K)
    {

        // sort the array so that close
        // elements come together.
        Array.Sort(arr);

        // initialize result by
        // a big integer number
        int res = 2147483647;

        // loop over first (N - K) elements
        // of the array only
        for (int i = 0; i <= (N - K); i++)
        {

            // get difference between max and
            // min of current K-sized segment
            int curSeqDiff = arr[i + K - 1] - arr[i];
            res = Math.Min(res, curSeqDiff);
        }

        return res;
    }

    // Driver code
    public static void Main()
    {
```

```
int []arr= {10, 20, 30, 100, 101, 102};  
int N = arr.Length;  
  
int K = 3;  
Console.WriteLine(  
    minDifferenceAmongMaxMin(arr, N, K));  
}  
}  
  
// This code is contributed by nitin mittal
```

PHP

```
<?php  
// PHP program to find minimum difference  
// between max and min of all subset  
// of K size  
  
// returns min difference between max  
// and min of any K-size subset  
function minDifferenceAmongMaxMin($arr, $N,  
                                    $K)  
{  
    $INT_MAX = 2;  
  
    // sort the array so that close  
    // elements come together.  
    sort($arr); sort($arr, $N);  
  
    // initialize result by a  
    // big integer number  
    $res = $INT_MAX;  
  
    // loop over first (N - K) elements  
    // of the array only  
    for ($i = 0; $i <= ($N - $K); $i++)  
    {  
  
        // get difference between max and  
        // min of current K-sized segment  
        $curSeqDiff = $arr[$i + $K - 1] -  
                     $arr[$i];  
        $res = min($res, $curSeqDiff);  
    }  
  
    return $res;  
}
```

```
// Driver Code
$arr = array(10, 20, 30, 100, 101, 102);
$N = sizeof($arr);

$K = 3;
echo minDifferenceAmongMaxMin($arr, $N, $K);

// This code is contributed by Nitin Mittal.
?>
```

Output:

2

Time Complexity: $O(n \log n)$

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/minimum-difference-max-min-k-size-subsets/>

Chapter 204

Minimum number of distinct elements after removing m items

Minimum number of distinct elements after removing m items - GeeksforGeeks

Given an array of items, an i-th index element denotes the item id's and given a number m, the task is to remove m elements such that there should be minimum distinct id's left. Print the number of distinct id's.

Examples:

```
Input : arr[] = { 2, 2, 1, 3, 3, 3}
        m = 3
Output : 1
Remove 1 and both 2's. So, only 3 will be
left that's why distinct id is 1.

Input : arr[] = { 2, 4, 1, 5, 3, 5, 1, 3}
        m = 2
Output : 3
Remove 2 and 4 completely. So, remaining ids
are 1, 3 and 5 i.e. 3
```

Asked in : Morgan Stanley

- 1- Count the occurrence of elements and store in the hash.
- 2- Sort the hash.
- 3- Start removing elements from hash.
- 4- Return the number of values left in the hash.

C++

```
// C++ program for above implementation
#include <bits/stdc++.h>
using namespace std;

// Function to find distinct id's
int distinctIds(int arr[], int n, int mi)
{
    unordered_map<int, int> m;
    vector<pair<int, int>> v;
    int count = 0;

    // Store the occurrence of ids
    for (int i = 0; i < n; i++)
        m[arr[i]]++;

    // Store into the vector second as first and vice-versa
    for (auto it = m.begin(); it != m.end(); it++)
        v.push_back(make_pair(it->second, it->first));

    // Sort the vector
    sort(v.begin(), v.end());

    int size = v.size();

    // Start removing elements from the beginning
    for (int i = 0; i < size; i++) {

        // Remove if current value is less than
        // or equal to mi
        if (v[i].first <= mi) {
            mi -= v[i].first;
            count++;
        }

        // Return the remaining size
        else
            return size - count;
    }
    return size - count;
}

// Driver code
int main()
{
    int arr[] = { 2, 3, 1, 2, 3, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);

    int m = 3;
```

```
    cout << distinctIds(arr, n, m);
    return 0;
}
```

Java

```
//Java program for Minimum number of
//distinct elements after removing m items
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

public class DistinctIds
{
    // Function to find distinct id's
    static int distinctIds(int arr[], int n, int mi)
    {

        Map<Integer, Integer> m = new HashMap<Integer, Integer>();
        int count = 0;
        int size = 0;

        // Store the occurrence of ids
        for (int i = 0; i < n; i++)
        {

            // If the key is not add it to map
            if (m.containsKey(arr[i]) == false)
            {
                m.put(arr[i], 1);
                size++;
            }

            // If it is present then increase the value by 1
            else m.put(arr[i], m.get(arr[i]) + 1);
        }

        // Start removing elements from the beginning
        for (Entry<Integer, Integer> mp:m.entrySet())
        {
            // Remove if current value is less than
            // or equal to mi
            if (mp.getKey() <= mi)
            {
                mi -= mp.getKey();
                count++;
            }
        }
    }
}
```

```
// Return the remaining size
else return size - count;
}

return size - count;
}

//Driver method to test above function
public static void main(String[] args)
{
    // TODO Auto-generated method stub
    int arr[] = {2, 3, 1, 2, 3, 3};
    int m = 3;

    System.out.println(distinctIds(arr, arr.length, m));
}
}
//This code is contributed by Sumit Ghosh
```

Output:

1

Time Complexity : $O(n \log n)$

Source

<https://www.geeksforgeeks.org/minimum-number-of-distinct-elements-after-removing-m-items/>

Chapter 205

Minimum number of elements to add to make median equals x

Minimum number of elements to add to make median equals x - GeeksforGeeks

A median in an array with the length of n is an element which occupies position number $(n+1)/2$ after we sort the elements in the non-decreasing order (the array elements are numbered starting with 1). A median of an array (2, 6, 1, 2, 3) is the number 2, and a median of array (0, 96, 17, 23) — the number 17.

Examples :

Input : 3 10
 10 20 30
Output : 1
In the first sample we can add number 9
to array (10, 20, 30). The resulting array
(9, 10, 20, 30) will have a median in
position $(4+1)/2 = 2$, that is, 10

Input : 3 4
 1 2 3
Output : 4
In the second sample you should add numbers
4, 5, 5, 5. The resulting array has median
equal to 4.

First Approach:- The approach is to add one more number x to the array until the median of the array equals to x. Below is the implementation of the above approach:-

C++

```
// CPP program to find minimum number
// of elements needs to add to the
// array so that its median equals x.
#include <bits/stdc++.h>
using namespace std;

// Returns count of elements to be
// added to make median x. This function
// assumes that a[] has enough extra space.
int minNumber(int a[], int n, int x)
{
    // to sort the array in increasing order.
    sort(a, a + n);

    int k;
    for (k = 0; a[(n - 1) / 2] != x; k++) {
        a[n++] = x;
        sort(a, a + n);
    }
    return k;
}

// Driver code
main()
{
    int x = 10;
    int a[6] = { 10, 20, 30 };
    int n = 3;
    cout << minNumber(a, n, x) << endl;
    return 0;
}
```

PHP

```
<?php
// PHP program to find minimum
// number of elements needs to
// add to the array so that its
// median equals x.

// Returns count of elements
// to be added to make median
// x. This function assumes
// that a[] has enough extra space.
function minNumber($a, $n, $x)
{
    // to sort the array in
    // increasing order.
```

```
sort($a);

$k;
for ($k = 0;
     $a[($n - 1) / 2] != $x; $k++)
{
    $a[$n++] = $x;
    sort($a);
}
return $k;
}

// Driver code
$x = 10;
$a = array (10, 20, 30);
$n = 3;
echo minNumber($a, $n, $x), "\n";

// This code is contributed by ajit
?>
```

Output :

1

Time complexity : O($kn\log n$)

Second Approach:- Better approach is to count all the elements equal to x(that is e), greater than x(that is h) and smaller than x(that is l). And then –
if l is greater than h then, the ans will be $(l - h) + 1 - e$;
And if h is greater than l then, ans will be $(h - l - 1) + 1 - e$;

We can use [Hoare's partition scheme](#) to count smaller, equal and greater elements.

Below is the implementation of the above approach:

C++

```
// CPP program to find minimum number of
// elements to add so that its median
// equals x.
#include <bits/stdc++.h>
using namespace std;

int minNumber(int a[], int n, int x)
{
    int l = 0, h = 0, e = 0;
    for (int i = 0; i < n; i++) {
```

```
// no. of elements equals to x,
// that is, e.
if (a[i] == x)
    e++;

// no. of elements greater than x,
// that is, h.
else if (a[i] > x)
    h++;

// no. of elements smaller than x,
// that is, l.
else if (a[i] < x)
    l++;
}

int ans = 0;
if (l > h)
    ans = l - h;
else if (l < h)
    ans = h - l - 1;

// subtract the no. of elements
// that are equal to x.
return ans + 1 - e;
}

// Driver code
int main()
{
    int x = 10;
    int a[] = { 10, 20, 30 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << minNumber(a, n, x) << endl;
    return 0;
}
```

Java

```
// Java program to find minimum number
// of elements to add so that its
// median equals x.
import java.util.*;
import java.lang.*;

class GFG {

    public static int minNumber(int a[],
```

```
int n, int x)
{
    int l = 0, h = 0, e = 0;
    for (int i = 0; i < n; i++)
    {
        // no. of elements equals to
        // x, that is, e.
        if (a[i] == x)
            e++;

        // no. of elements greater
        // than x, that is, h.
        else if (a[i] > x)
            h++;

        // no. of elements smaller
        // than x, that is, l.
        else if (a[i] < x)
            l++;
    }

    int ans = 0;
    if (l > h)
        ans = l - h;
    else if (l < h)
        ans = h - l - 1;

    // subtract the no. of elements
    // that are equal to x.
    return ans + 1 - e;
}

// Driven Program
public static void main(String[] args)
{
    int x = 10;
    int a[] = { 10, 20, 30 };
    int n = a.length;
    System.out.println(
        minNumber(a, n, x));
}
}

// This code is contributed by
// Prasad Kshirsagar
```

Python3

```
# Python3 program to find minimum number
# of elements to add so that its median
# equals x.

def minNumber (a, n, x):
    l = 0
    h = 0
    e = 0
    for i in range(n):

        # no. of elements equals to x,
        # that is, e.
        if a[i] == x:
            e+=1

        # no. of elements greater than x,
        # that is, h.
        elif a[i] > x:
            h+=1

        # no. of elements smaller than x,
        # that is, l.
        elif a[i] < x:
            l+=1

    ans = 0;
    if l > h:
        ans = l - h
    elif l < h:
        ans = h - l - 1;

    # subtract the no. of elements
    # that are equal to x.
    return ans + 1 - e

# Driver code
x = 10
a = [10, 20, 30]
n = len(a)
print(minNumber(a, n, x))

# This code is contributed
# by "Abhishek Sharma 44"

C#
// C# program to find minimum
// number of elements to add
```

```
// so that its median equals x.
using System;

class GFG
{
public static int minNumber(int []a,
                           int n,
                           int x)
{
    int l = 0, h = 0, e = 0;
    for (int i = 0; i < n; i++)
    {

        // no. of elements
        // equals to x,
        // that is, e.
        if (a[i] == x)
            e++;

        // no. of elements
        // greater than x,
        // that is, h.
        else if (a[i] > x)
            h++;

        // no. of elements smaller
        // than x, that is, l.
        else if (a[i] < x)
            l++;
    }

    int ans = 0;
    if (l > h)
        ans = l - h;
    else if (l < h)
        ans = h - l - 1;

    // subtract the no.
    // of elements that
    // are equal to x.
    return ans + 1 - e;
}

// Driver Code
public static void Main()
{
    int x = 10;
    int []a = {10, 20, 30};
```

```
    int n = a.Length;
    Console.WriteLine(
        minNumber(a, n, x));
}
}

// This code is contributed
// by anuj_67.
```

PHP

```
<?php
// PHP program to find minimum
// number of elements to add so
// that its median equals x.

function minNumber($a, $n, $x)
{
    $l = 0; $h = 0; $e = 0;
    for ($i = 0; $i < $n; $i++)
    {

        // no. of elements equals
        // to x, that is, e.
        if ($a[$i] == $x)
            $e++;

        // no. of elements greater
        // than x, that is, h.
        else if ($a[$i] > $x)
            $h++;

        // no. of elements smaller
        // than x, that is, l.
        else if ($a[$i] < $x)
            $l++;
    }

    $ans = 0;
    if ($l > $h)
        $ans = $l - $h;
    else if ($l < $h)
        $ans = $h - $l - 1;

    // subtract the no. of elements
    // that are equal to x.
    return $ans + 1 - $e;
}
```

```
// Driver code
$x = 10;
$a = array (10, 20, 30);
$n = sizeof($a) ;
echo minNumber($a, $n, $x), "\n";

// This code is contributed by jit_t
?>
```

Output :

1

Time complexity : O(n)

Improved By : [Prasad_Kshirsagar](#), [jit_t](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/minimum-number-elements-add-make-median-equals-x/>

Chapter 206

Minimum number of subsets with distinct elements

Minimum number of subsets with distinct elements - GeeksforGeeks

You are given an array of n-element. You have to make subsets from the array such that no subset contain duplicate elements. Find out minimum number of subset possible.

Examples :

```
Input : arr[] = {1, 2, 3, 4}
Output :1
Explanation : A single subset can contains all
values and all values are distinct

Input : arr[] = {1, 2, 3, 3}
Output : 2
Explanation : We need to create two subsets
{1, 2, 3} and {3} [or {1, 3} and {2, 3}] such
that both subsets have distinct elements.
```

We basically need to find the most frequent element in the array. The result is equal to the frequency of the most frequent element.

A **simple solution** is to run two nested loops to count frequency of every element and return the frequency of the most frequent element. Time complexity of this solution is $O(n^2)$.

A **better solution** is to first sort the array and then start count number of repetitions of elements in an iterative manner as all repetition of any number lie beside the number itself. By this method you can find the maximum frequency or repetition by simply traversing the sorted array. This approach will cost $O(n\log n)$ time complexity

C++

```
// A sorting based solution to find the
// minimum number of subsets of a set
// such that every subset contains distinct
// elements.
#include <bits/stdc++.h>
using namespace std;

// Function to count subsets such that all
// subsets have distinct elements.
int subset(int ar[], int n)
{
    // Take input and initialize res = 0
    int res = 0;

    // Sort the array
    sort(ar, ar + n);

    // Traverse the input array and
    // find maximum frequency
    for (int i = 0; i < n; i++) {
        int count = 1;

        // For each number find its repetition / frequency
        for (; i < n - 1; i++) {
            if (ar[i] == ar[i + 1])
                count++;
            else
                break;
        }

        // Update res
        res = max(res, count);
    }
}

return res;
}

// Driver code
int main()
{
    int arr[] = { 5, 6, 9, 3, 4, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << subset(arr, n);
    return 0;
}
```

Java

```
// A sorting based solution to find the
// minimum number of subsets of a set
// such that every subset contains distinct
// elements.
import java.util.*;
import java.lang.*;

public class GfG{

    // Function to count subsets such that all
    // subsets have distinct elements.
    public static int subset(int ar[], int n)
    {
        // Take input and initialize res = 0
        int res = 0;

        // Sort the array
        Arrays.sort(ar);

        // Traverse the input array and
        // find maximum frequency
        for (int i = 0; i < n; i++) {
            int count = 1;

            // For each number find its repetition / frequency
            for (; i < n - 1; i++) {
                if (ar[i] == ar[i + 1])
                    count++;
                else
                    break;
            }

            // Update res
            res = Math.max(res, count);
        }

        return res;
    }

    // Driver function
    public static void main(String argc[])
    {
        int arr[] = { 5, 6, 9, 3, 4, 3, 4 };
        int n = 7;
        System.out.println(subset(arr, n));
    }
}
```

```
/* This code is contributed by Sagar Shukla */
```

Python3

```
# A sorting based solution to find the
# minimum number of subsets of a set
# such that every subset contains distinct
# elements.

# function to count subsets such that all
# subsets have distinct elements.
def subset(ar, n):

    # take input and initialize res = 0
    res = 0

    # sort the array
    ar.sort()

    # traverse the input array and
    # find maximum frequency
    for i in range(0, n) :
        count = 1

        # for each number find its repetition / frequency
        for i in range(n - 1):
            if ar[i] == ar[i + 1]:
                count+=1
            else:
                break

        # update res
        res = max(res, count)

    return res

# Driver code
ar = [ 5, 6, 9, 3, 4, 3, 4 ]
n = len(ar)
print(subset(ar, n))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
// A sorting based solution to find the
// minimum number of subsets of a set
// such that every subset contains distinct
// elements.
using System;

public class GfG {

    // Function to count subsets such that all
    // subsets have distinct elements.
    public static int subset(int []ar, int n)
    {
        // Take input and initialize res = 0
        int res = 0;

        // Sort the array
        Array.Sort(ar);

        // Traverse the input array and
        // find maximum frequency
        for (int i = 0; i < n; i++) {
            int count = 1;

            // For each number find its
            // repetition / frequency
            for ( ; i < n - 1; i++) {
                if (ar[i] == ar[i + 1])
                    count++;
                else
                    break;
            }

            // Update res
            res = Math.Max(res, count);
        }

        return res;
    }

    // Driver function
    public static void Main()
    {
        int []arr = { 5, 6, 9, 3, 4, 3, 4 };
        int n = 7;

        Console.WriteLine(subset(arr, n));
    }
}
```

```
}
```

```
/* This code is contributed by Vt_m */
```

Output :

2

An **efficient solution** is to use hashing. We count frequencies of all elements in a hash table. Finally we return the key with maximum value in hash table.

```
// A hashing based solution to find the
// minimum number of subsets of a set
// such that every subset contains distinct
// elements.
#include <bits/stdc++.h>
using namespace std;

// Function to count subsets such that all
// subsets have distinct elements.
int subset(int arr[], int n)
{
    // Traverse the input array and
    // store frequencies of elements
    unordered_map<int, int> mp;
    for (int i = 0; i < n; i++)
        mp[arr[i]]++;

    // Find the maximum value in map.
    int res = 0;
    for (auto x : mp)
        res = max(res, x.second);

    return res;
}

// Driver code
int main()
{
    int arr[] = { 5, 6, 9, 3, 4, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << subset(arr, n);
    return 0;
}
```

Output :

2

Source

<https://www.geeksforgeeks.org/minimum-number-subsets-distinct-elements/>

Chapter 207

Minimum number of subtract operation to make an array decreasing

Minimum number of subtract operation to make an array decreasing - GeeksforGeeks

You are given a sequence of numbers $\text{arr}[0], \text{arr}[1], \dots, \text{arr}[N - 1]$ and a positive integer K . In each operation, you may subtract K from any element of the array. You are required to find the minimum number of operations to make the given array decreasing.

An array $\text{arr} = [a_0, a_1, \dots, a_{N-1}]$ is called decreasing if $a_0 > a_1 > \dots > a_{N-1}$ for each $i: 0 \leq i < N - 1$.

Input : $N = 4, K = 5, \text{arr}[] = \{1, 1, 2, 3\}$

Output : 3

Explanation :

Since $\text{arr}[1] == \text{arr}[0]$ so no subtraction is required for $\text{arr}[1]$. For $\text{arr}[2]$, since $\text{arr}[2] > \text{arr}[1]$ ($2 > 1$) so we have to subtract $\text{arr}[2]$ by k and after the one subtraction value of $\text{arr}[2]$ is -3 which is less than the value of $\text{arr}[1]$, so number of subtraction required only 1 and now value of $\text{arr}[2]$ has been updated by -3 . Similarly for $\text{arr}[3]$, since $\text{arr}[3] > \text{arr}[2]$ ($3 > -3$) so for this we have to subtract $\text{arr}[3]$ by k two times to make the value of $\text{arr}[3]$ lesser than $\text{arr}[2]$, the number of subtraction required 2 and the updated value of $\text{arr}[3]$ is -7 . Now count total number of subtraction /operation required by adding number of operation on each step and that is $= 0+1+2 = 3$.

Input : $N = 5, K = 2, \text{arr}[] = \{5, 4, 3, 2, 1\}$

Output : 0

Approach :

1. Traverse each element of array from 1 to n-1.
2. Check if ($\text{arr}[i] > \text{arr}[i-1]$) then
 Find noOfSubtraction;
 noOfSubtraction =

 If ($(\text{arr}[i] - \text{arr}[i-1]) \% k == 0$)
 then noOfSubtraction++

 Modify arr[i];
 arr[i] =

Below is implementation of above approach :
CPP

```
// CPP program to make an array decreasing
#include <iostream>
using namespace std;

// Function to count minimum no of operation
int min_noOf_operation(int arr[], int n, int k)
{
    int noOfSubtraction;
    int res = 0;
    for (int i = 1; i < n; i++) {
        noOfSubtraction = 0;

        if (arr[i] > arr[i - 1]) {

            // Count how many times we have to subtract.
            noOfSubtraction = (arr[i] - arr[i - 1]) / k;

            // Check an additional subtraction is
            // required or not.
            if ((arr[i] - arr[i - 1]) % k != 0)
                noOfSubtraction++;

            // Modify the value of arr[i].
            arr[i] = arr[i] - k * noOfSubtraction;
        }

        // Count total no of operation/subtraction .
        res = res + noOfSubtraction;
    }

    return res;
}
```

```
// Driver Code
int main()
{
    int arr[] = { 1, 1, 2, 3 };
    int N = sizeof(arr) / sizeof(arr[0]);
    int k = 5;
    cout << min_noOf_operation(arr, N, k) << endl;
    return 0;
}
```

Java

```
// Java program to make an
// array decreasing
import java.util.*;
import java.lang.*;

public class GfG{

    // Function to count minimum no of operation
    public static int min_noOf_operation(int arr[],
                                         int n, int k)
    {
        int noOfSubtraction;
        int res = 0;

        for (int i = 1; i < n; i++) {
            noOfSubtraction = 0;

            if (arr[i] > arr[i - 1]) {

                // Count how many times
                // we have to subtract.
                noOfSubtraction = (arr[i] - arr[i - 1]) / k;

                // Check an additional subtraction
                // is required or not.
                if ((arr[i] - arr[i - 1]) % k != 0)
                    noOfSubtraction++;

                // Modify the value of arr[i]
                arr[i] = arr[i] - k * noOfSubtraction;
            }

            // Count total no of subtraction
            res = res + noOfSubtraction;
        }
    }
}
```

```
        return res;
    }

    // driver function
    public static void main(String argc[]){
        int arr = { 1, 1, 2, 3 };
        int N = 4;
        int k = 5;
        System.out.println(min_noOf_operation(arr,
                                              N, k));
    }

}

/* This code is contributed by Sagar Shukla */
```

Python3

```
# Python program to make an array decreasing

# Function to count minimum no of operation
def min_noOf_operation(arr, n, k):

    res = 0
    for i in range(1,n):
        noOfSubtraction = 0

        if (arr[i] > arr[i - 1]):

            # Count how many times we have to subtract.
            noOfSubtraction = (arr[i] - arr[i - 1]) / k;

            # Check an additional subtraction is
            # required or not.
            if ((arr[i] - arr[i - 1]) % k != 0):
                noOfSubtraction+=1

            # Modify the value of arr[i].
            arr[i] = arr[i] - k * noOfSubtraction

        # Count total no of operation/subtraction .
        res = res + noOfSubtraction

    return int(res)
```

```
# Driver Code
arr = [ 1, 1, 2, 3 ]
N = len(arr)
k = 5
print(min_noOf_operation(arr, N, k))

# This code is contributed by
# Smitha Dinesh Semwal

C#

// C# program to make an
// array decreasing
using System;

public class GfG{

    // Function to count minimum no of operation
    public static int min_noOf_operation(int []arr,
                                         int n, int k)
    {
        int noOfSubtraction;
        int res = 0;

        for (int i = 1; i < n; i++) {
            noOfSubtraction = 0;

            if (arr[i] > arr[i - 1]) {

                // Count how many times
                // we have to subtract.
                noOfSubtraction = (arr[i] - arr[i - 1]) / k;

                // Check an additional subtraction
                // is required or not.
                if ((arr[i] - arr[i - 1]) % k != 0)
                    noOfSubtraction++;

                // Modify the value of arr[i]
                arr[i] = arr[i] - k * noOfSubtraction;
            }

            // Count total no of subtraction
            res = res + noOfSubtraction;
        }

        return res;
    }
}
```

```
// driver function
public static void Main()
{
    int []arr = { 1, 1, 2, 3 };
    int N = 4;
    int k = 5;
    Console.WriteLine(min_noOf_operation(arr,
                                         N, k));
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to make an array decreasing

// Function to count minimum no of operation
function min_noOf_operation($arr, $n, $k)
{

    $noOfSubtraction;
    $res = 0;
    for($i = 1; $i < $n; $i++)
    {
        $noOfSubtraction = 0;

        if ($arr[$i] > $arr[$i - 1])
        {

            // Count how many times we
            // have to subtract.
            $noOfSubtraction = ($arr[$i] -
                                $arr[$i - 1]) / $k;

            // Check an additional subtraction
            // is required or not.
            if (($arr[$i] - $arr[$i - 1])
                % $k != 0)
                $noOfSubtraction++;

        // Modify the value of arr[i].
        $arr[$i] = $arr[$i] - $k *
                    $noOfSubtraction;
    }
}
```

```
// Count total no of
// operation/subtraction .
$res = $res + $noOfSubtraction;
}

return floor($res);
}

// Driver Code
$arr = array(1, 1, 2, 3);
$N = count($arr);
$k = 5;
echo min_noOf_operation($arr, $N, $k) ;

// This code is contributed by anuj_67.
?>
```

Output :

3

Time Complexity : O(N).

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-minimum-number-operation-make-array-decreasing/>

Chapter 208

Minimum number of swaps required to sort an array

Minimum number of swaps required to sort an array - GeeksforGeeks

Given an array of **n** distinct elements, find the minimum number of swaps required to sort the array.

Examples:

Input : {4, 3, 2, 1}

Output : 2

Explanation : Swap index 0 with 3 and 1 with 2 to form the sorted array {1, 2, 3, 4}.

Input : {1, 5, 4, 3, 2}

Output : 2

This can be easily done by visualizing the problem as a graph. We will have **n** nodes and an edge directed from node **i** to node **j** if the element at **i**'th index must be present at **j**'th index in the sorted array.

Graph for {4, 3, 2, 1}

The graph will now contain many non-intersecting cycles. Now a cycle with 2 nodes will only require 1 swap to reach the correct ordering, similarly a cycle with 3 nodes will only require 2 swap to do so.

Graph for {4, 5, 2, 1, 5}

Hence,

$$\text{ans} = \sum_{i=1}^k (\text{cycle_size} - 1)$$

where k is the number of cycles

Below is the C++ implementation of the idea.

C++

```
// C++ program to find minimum number of swaps
// required to sort an array
#include<bits/stdc++.h>

using namespace std;

// Function returns the minimum number of swaps
// required to sort the array
int minSwaps(int arr[], int n)
{
    // Create an array of pairs where first
    // element is array element and second element
    // is position of first element
    pair<int, int> arrPos[n];
    for (int i = 0; i < n; i++)
    {
        arrPos[i].first = arr[i];
        arrPos[i].second = i;
    }

    // Sort the array by array element values to
    // get right position of every element as second
    // element of pair.
    sort(arrPos, arrPos + n);

    // To keep track of visited elements. Initialize
    // all elements as not visited or false.
    vector<bool> vis(n, false);

    // Initialize result
    int ans = 0;

    // Traverse array elements
    for (int i = 0; i < n; i++)
    {
        // already swapped and corrected or
        // already present at correct pos
        if (vis[i] || arrPos[i].second == i)
            continue;

        int correct_pos = arrPos[i].second;
```

```
// find out the number of node in
// this cycle and add in ans
int cycle_size = 0;
int j = i;
while (!vis[j])
{
    vis[j] = 1;

    // move to next node
    j = arrPos[j].second;
    cycle_size++;
}

// Update answer by adding current cycle.
if(cycle_size > 0)
{
    ans += (cycle_size - 1);
}
}

// Return result
return ans;
}

// Driver program to test the above function
int main()
{
    int arr[] = {1, 5, 4, 3, 2};
    int n = (sizeof(arr) / sizeof(int));
    cout << minSwaps(arr, n);
    return 0;
}
```

Java

```
// Java program to find minimum number of swaps
// required to sort an array
import javafx.util.Pair;
import java.util.ArrayList;
import java.util.*;

class GfG
{
    // Function returns the minimum number of swaps
    // required to sort the array
    public static int minSwaps(int[] arr)
    {
        int n = arr.length;
```

```
// Create two arrays and use as pairs where first
// array is element and second array
// is position of first element
ArrayList <Pair <Integer, Integer> > arrpos =
    new ArrayList <Pair <Integer, Integer> > ();
for (int i = 0; i < n; i++)
    arrpos.add(new Pair <Integer, Integer> (arr[i], i));

// Sort the array by array element values to
// get right position of every element as the
// elements of second array.
arrpos.sort(new Comparator<Pair<Integer, Integer>>()
{
    @Override
    public int compare(Pair<Integer, Integer> o1,
                       Pair<Integer, Integer> o2)
    {
        if (o1.getKey() > o2.getKey())
            return -1;

        // We can change this to make it then look at the
        // words alphabetical order
        else if (o1.getKey().equals(o2.getKey()))
            return 0;

        else
            return 1;
    }
});

// To keep track of visited elements. Initialize
// all elements as not visited or false.
Boolean[] vis = new Boolean[n];
Arrays.fill(vis, false);

// Initialize result
int ans = 0;

// Traverse array elements
for (int i = 0; i < n; i++)
{
    // already swapped and corrected or
    // already present at correct pos
    if (vis[i] || arrpos.get(i).getValue() == i)
        continue;

    // find out the number of node in
```

```
// this cycle and add in ans
int cycle_size = 0;
int j = i;
while (!vis[j])
{
    vis[j] = true;

    // move to next node
    j = arrpos.get(j).getValue();
    cycle_size++;
}

// Update answer by adding current cycle.
if(cycle_size > 0)
{
    ans += (cycle_size - 1);
}
}

// Return result
return ans;
}

// Driver class
class MinSwaps
{
    // Driver program to test the above function
    public static void main(String[] args)
    {
        int []a = {1, 5, 4, 3, 2};
        GfG g = new GfG();
        System.out.println(g.minSwaps(a));
    }
}
// This code is contributed by Saksham Seth

[ /sourcecode]
```

Python3

```
# Python3 program to find minimum number
# of swaps required to sort an array

# Function returns the minimum
# number of swaps required to sort the array
def minSwaps(arr):
    n = len(arr)
```

```
# Create two arrays and use
# as pairs where first array
# is element and second array
# is position of first element
arrpos = [*enumerate(arr)]

# Sort the array by array element
# values to get right position of
# every element as the elements
# of second array.
arrpos.sort(key = lambda it:it[1])

# To keep track of visited elements.
# Initialize all elements as not
# visited or false.
vis = {k:False for k in range(n)}

# Initialize result
ans = 0
for i in range(n):

    # already swapped or
    # already present at
    # correct position
    if vis[i] or arrpos[i][0] == i:
        continue

    # find number of nodes
    # in this cycle and
    # add it to ans
    cycle_size = 0
    j = i
    while not vis[j]:

        # mark node as visited
        vis[j] = True

        # move to next node
        j = arrpos[j][0]
        cycle_size += 1

    # update answer by adding
    # current cycle
    if cycle_size > 0:
        ans += (cycle_size - 1)
# return answer
return ans
```

```
# Driver Code
arr = [1, 5, 4, 3, 2]
print(minSwaps(arr))

# This code is contributed
# by Dharan Aditya
```

Output:

2

Time Complexity: $O(n \log n)$

Auxiliary Space: $O(n)$

Related Article :

[Number of swaps to sort when only adjacent swapping allowed](#)

Reference:

<http://stackoverflow.com/questions/15152322/compute-the-minimal-number-of-swaps-to-order-a-sequence/15152602#15152602>

Improved By : [dharan1011](#), PREM UKKOJI

Source

<https://www.geeksforgeeks.org/minimum-number-swaps-required-sort-array/>

Chapter 209

Minimum partitions of maximum size 2 and sum limited by given value

Minimum partitions of maximum size 2 and sum limited by given value - GeeksforGeeks

Given an array arr[] of positive numbers, find minimum number of sets in array which satisfy following property,

1. A set can contain maximum two elements in it. The two elements need not to be contiguous.
2. Sum of elements of set should be less than or equal to given Key. It may be assumed that given key is greater than or equal to the largest array element.

Examples:

Input: arr[] = [10, 20, 12], key = 25
Output: 2
We break into two parts {10, 12} and {2}

Input : arr[] = [3, 5, 3, 4], key=5
Output : 4
Explanation: 4 sets (3), (5), (3), (4)

The idea is to first sort the array, then follow [two pointer approach](#). We begin two pointers from two corners of the sorted array. If their sum is smaller than or equal to given key, then we make set of them, else we consider the last element alone.

Below is the implementation of the above approach :
C++

```
// C++ program to count minimum number of partitions
// of size 2 and sum smaller than or equal to given
// key.
#include <algorithm>
#include <iostream>
using namespace std;

int minimumSets(int arr[], int n, int key)
{
    int i, j;

    // sort the array
    sort(arr, arr + n);

    // if sum of ith smaller and jth larger element is
    // less than key, then pack both numbers in a set
    // otherwise pack the jth larger number
    // alone in the set
    for (i = 0, j = n - 1; i <= j; ++i)
        if (arr[i] + arr[j] <= key)
            j--;
        else
            break;

    // After ending of loop i will contain minimum
    // number of sets
    return i;
}

int main()
{
    int arr[] = { 3, 5, 3, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 5;
    cout << minimumSets(arr, n, key);
    return 0;
}
```

Java

```
// Java program to count minimum number of partitions
// of size 2 and sum smaller than or equal to given
// key.

import java.util.Arrays;
class GFG {

    static int minimumSets(int arr[], int n, int key)
    {
```

```
int i, j;

// sort the array
Arrays.sort(arr);

// if sum of ith smaller and jth larger element is
// less than key, then pack both numbers in a set
// otherwise pack the jth larger number
// alone in the set
for (i = 0, j = n - 1; i <= j; ++i)
    if (arr[i] + arr[j] <= key)
        j--;

// After ending of loop i will contain minimum
// number of sets
return i;
}

public static void main (String[] args) {
int []arr = { 3, 5, 3, 4 };
int n = arr.length;
int key = 5;
System.out.println( minimumSets(arr, n, key));
}
}

// This code is contributed by chandan_jnu.
```

C#

```
// C# program to count minimum
// number of partitions of size
// 2 and sum smaller than or
// equal to given key.
using System;
class GFG
{

static int minimumSets(int []arr,
                      int n, int key)
{
    int i, j;

    // sort the array
    Array.Sort(arr);

    // if sum of ith smaller and
```

```
// jth larger element is less
// than key, then pack both
// numbers in a set otherwise
// pack the jth larger number
// alone in the set
for (i = 0, j = n - 1; i <= j; ++i)
    if (arr[i] + arr[j] <= key)
        j--;
    else
        i++;

// After ending of loop i
// will contain minimum
// number of sets
return i;
}

// Driver Code
public static void Main ()
{
    int []arr = { 3, 5, 3, 4 };
    int n = arr.Length;
    int key = 5;
    Console.WriteLine(minimumSets(arr, n, key));
}
}

// This code is contributed
// by chandan_jnu.
```

PHP

```
<?php
// PHP program to count minimum
// number of partitions of size
// 2 and sum smaller than or
// equal to given key.
function minimumSets($arr, $n, $key)
{
    $i; $j;

    // sort the array
    sort($arr);

    // if sum of ith smaller and
    // jth larger element is less
    // than key, then pack both
    // numbers in a set otherwise
    // pack the jth larger number
    // alone in the set
```

```
for ($i = 0, $j = $n - 1; $i <= $j; ++$i)
    if ($arr[$i] + $arr[$j] <= $key)
        $j--;
    return $i;
}

// Driver Code
$arr = array( 3, 5, 3, 4 );
$n = count($arr);
$key = 5;
echo minimumSets($arr, $n, $key);

// This code is contributed
// by chandan_jnu
?>
```

Output:

4

Time complexity: O(nlogn)

Improved By : [Chandan_Kumar](#)

Source

<https://www.geeksforgeeks.org/minimum-partitions-of-maximum-size-2-and-sum-limited-by-given-value/>

Chapter 210

Minimum product of k integers in an array of positive Integers

Minimum product of k integers in an array of positive Integers - GeeksforGeeks

Given an array of n positive integers. We are required to write a program to print the minimum product of k integers of the given array.

Examples:

```
Input : 198 76 544 123 154 675
        k = 2
Output : 9348
We get minimum product after multiplying
76 and 123.
```

```
Input : 11 8 5 7 5 100
        k = 4
Output : 1400
```

The idea is simple, we find the smallest k elements and print multiplication of them. In below implementation, we have used simple [Heap](#) based approach where we insert array elements into a min heap and then find product of top k elements.

C++

```
// CPP program to find minimum product of
// k elements in an array
#include <bits/stdc++.h>
using namespace std;

int minProduct(int arr[], int n, int k)
```

```
{  
    priority_queue<int, vector<int>, greater<int> > pq;  
    for (int i = 0; i < n; i++)  
        pq.push(arr[i]);  
  
    int count = 0, ans = 1;  
  
    // One by one extract items from max heap  
    while (pq.empty() == false && count < k) {  
        ans = ans * pq.top();  
        pq.pop();  
        count++;  
    }  
  
    return ans;  
}  
  
// Driver code  
int main()  
{  
    int arr[] = {198, 76, 544, 123, 154, 675};  
    int k = 2;  
    int n = sizeof(arr) / sizeof(arr[0]);  
    cout << "Minimum product is "  
        << minProduct(arr, n, k);  
    return 0;  
}
```

Python3

```
# Python3 program to find minimum  
# product of k elements in an array  
import math  
import heapq  
  
def minProduct(arr, n, k):  
  
    heapq.heapify(arr)  
    count = 0  
    ans = 1  
  
    # One by one extract  
    # items from min heap  
    while (arr) and count < k:  
        x = heapq.heappop(arr)  
        ans = ans * x  
        count = count + 1
```

```
return ans;

# Driver method
arr = [198, 76, 544, 123, 154, 675]
k = 2
n = len(arr)
print ("Minimum product is",
       minProduct(arr, n, k))
```

Output:

```
Minimum product is 9348
```

Time Complexity : $O(n * \log n)$

Note that the above problem can be solved in $O(n)$ time using methods discussed [here](#) and [here](#).

Source

<https://www.geeksforgeeks.org/minimum-product-k-integers-array-positive-integers/>

Chapter 211

Minimum product pair an array of positive Integers

Minimum product pair an array of positive Integers - GeeksforGeeks

Given an array of positive integers. We are required to write a program to print the minimum product of any two numbers of the given array.

Examples:

Input : 11 8 5 7 5 100
Output : 25
Explanation : The minimum product of any two numbers will be $5 * 5 = 25$.

Input : 198 76 544 123 154 675
Output : 7448
Explanation : The minimum product of any two numbers will be $76 * 123 = 7448$.

Simple Approach : A simple approach will be to run two nested loops to generate all possible pair of elements and keep track of the minimum product.

Time Complexity: $O(n * n)$

Auxiliary Space: $O(1)$

Better Approach: An efficient approach will be to first sort the given array and print the product of first two numbers, sorting will take $O(n \log n)$. Answer will be then $a[0] * a[1]$

Time Complexity: $O(n * \log(n))$

Auxiliary Space: $O(1)$

Best Approach: The idea is linearly traverse given array and keep track of minimum two elements. Finally return product of two minimum elements.

Below is the implementation of above approach.

C++

```
// C++ program to calculate minimum
// product of a pair
#include <bits/stdc++.h>
using namespace std;

// Function to calculate minimum product
// of pair
int printMinimumProduct(int arr[], int n)
{
    // Initialize first and second
    // minimums. It is assumed that the
    // array has at least two elements.
    int first_min = min(arr[0], arr[1]);
    int second_min = max(arr[0], arr[1]);

    // Traverse remaining array and keep
    // track of two minimum elements (Note
    // that the two minimum elements may
    // be same if minimum element appears
    // more than once)
    // more than once)
    for (int i=2; i<n; i++)
    {
        if (arr[i] < first_min)
        {
            second_min = first_min;
            first_min = arr[i];
        }
        else if (arr[i] < second_min)
            second_min = arr[i];
    }

    return first_min * second_min;
}

// Driver program to test above function
int main()
{
    int a[] = { 11, 8 , 5 , 7 , 5 , 100 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << printMinimumProduct(a,n);
    return 0;
}
```

Java

```
// Java program to calculate minimum
// product of a pair
import java.util.*;

class GFG {

    // Function to calculate minimum product
    // of pair
    static int printMinimumProduct(int arr[], int n)
    {
        // Initialize first and second
        // minimums. It is assumed that the
        // array has at least two elements.
        int first_min = Math.min(arr[0], arr[1]);
        int second_min = Math.max(arr[0], arr[1]);

        // Traverse remaining array and keep
        // track of two minimum elements (Note
        // that the two minimum elements may
        // be same if minimum element appears
        // more than once)
        // more than once)
        for (int i = 2; i < n; i++)
        {
            if (arr[i] < first_min)
            {
                second_min = first_min;
                first_min = arr[i];
            }
            else if (arr[i] < second_min)
                second_min = arr[i];
        }

        return first_min * second_min;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int a[] = { 11, 8, 5, 7, 5, 100 };
        int n = a.length;
        System.out.print(printMinimumProduct(a,n));

    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
# Python program to
# calculate minimum
# product of a pair

# Function to calculate
# minimum product
# of pair
def printMinimumProduct(arr,n):

    # Initialize first and second
    # minimums. It is assumed that the
    # array has at least two elements.
    first_min = min(arr[0], arr[1])
    second_min = max(arr[0], arr[1])

    # Traverse remaining array and keep
    # track of two minimum elements (Note
    # that the two minimum elements may
    # be same if minimum element appears
    # more than once)
    # more than once)
    for i in range(2,n):

        if (arr[i] < first_min):

            second_min = first_min
            first_min = arr[i]

        elif (arr[i] < second_min):
            second_min = arr[i]

    return first_min * second_min

# Driver code

a= [ 11, 8 , 5 , 7 , 5 , 100 ]
n = len(a)

print(printMinimumProduct(a,n))

# This code is contributed
# by Anant Agarwal.
```

C#

```
// C# program to calculate minimum
```

```
// product of a pair
using System;

class GFG {

    // Function to calculate minimum
    // product of pair
    static int printMinimumProduct(int []arr,
                                    int n)
    {

        // Initialize first and second
        // minimums. It is assumed that
        // the array has at least two
        // elements.
        int first_min = Math.Min(arr[0],
                                arr[1]);

        int second_min = Math.Max(arr[0],
                                arr[1]);

        // Traverse remaining array and
        // keep track of two minimum
        // elements (Note that the two
        // minimum elements may be same
        // if minimum element appears
        // more than once)
        for (int i = 2; i < n; i++)
        {
            if (arr[i] < first_min)
            {
                second_min = first_min;
                first_min = arr[i];
            }
            else if (arr[i] < second_min)
                second_min = arr[i];
        }

        return first_min * second_min;
    }

    /* Driver program to test above
    function */
    public static void Main()
    {
        int []a = { 11, 8 , 5 , 7 ,
                   5 , 100 };
        int n = a.Length;
```

```
        Console.WriteLine(
            printMinimumProduct(a, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to calculate minimum
// product of a pair

// Function to calculate minimum
// product of pair
function printMinimumProduct($arr, $n)
{

    // Initialize first and second
    // minimums. It is assumed that the
    // array has at least two elements.
    $first_min = min($arr[0], $arr[1]);
    $second_min = max($arr[0], $arr[1]);

    // Traverse remaining array and keep
    // track of two minimum elements (Note
    // that the two minimum elements may
    // be same if minimum element appears
    // more than once)
    // more than once)
    for ($i = 2; $i < $n; $i++)
    {
        if ($arr[$i] < $first_min)
        {
            $second_min = $first_min;
            $first_min = $arr[$i];
        }
        else if ($arr[$i] < $second_min)
            $second_min = $arr[$i];
    }

    return $first_min * $second_min;
}

// Driver Code
$a = array(11, 8, 5, 7, 5, 100);
$n = sizeof($a);
```

```
echo(printMinimumProduct($a, $n));  
// This code is contributed by Ajit.  
?>
```

Output:

25

Time Complexity: O(n)

Auxiliary Space: O(1)

Improved By : [vt_m](#), [jit_t](#)

Source

<https://www.geeksforgeeks.org/minimum-product-pair-an-array-of-positive-integers/>

Chapter 212

Minimum product subset of an array

Minimum product subset of an array - GeeksforGeeks

Given an array a, we have to find minimum product possible with the subset of elements present in the array. The minimum product can be single element also.

Examples:

```
Input : a[] = { -1, -1, -2, 4, 3 }
Output : -24
Explanation : Minimum product will be ( -2 * -1 * -1 * 4 * 3 ) = -24
```

```
Input : a[] = { -1, 0 }
Output : -1
Explanation : -1(single element) is minimum product possible
```

```
Input : a[] = { 0, 0, 0 }
Output : 0
```

A simple solution is to [generate all subsets](#), find product of every subset and return maximum product.

A better solution is to use the below facts.

1. If there are even number of negative numbers and no zeros, the result is the product of all except the largest valued negative number.
2. If there are an odd number of negative numbers and no zeros, the result is simply the product of all.

3. If there are zeros and positive, no negative, the result is 0. The exceptional case is when there is no negative number and all other elements positive then our result should be the first minimum positive number.

C++

```
// CPP program to find maximum product of
// a subset.
#include <bits/stdc++.h>
using namespace std;

int minProductSubset(int a[], int n)
{
    if (n == 1)
        return a[0];

    // Find count of negative numbers, count
    // of zeros, maximum valued negative number,
    // minimum valued positive number and product
    // of non-zero numbers
    int max_neg = INT_MIN;
    int min_pos = INT_MAX;
    int count_neg = 0, count_zero = 0;
    int prod = 1;
    for (int i = 0; i < n; i++) {

        // If number is 0, we don't
        // multiply it with product.
        if (a[i] == 0) {
            count_zero++;
            continue;
        }

        // Count negatives and keep
        // track of maximum valued negative.
        if (a[i] < 0) {
            count_neg++;
            max_neg = max(max_neg, a[i]);
        }

        // Track minimum positive
        // number of array
        if (a[i] > 0)
            min_pos = min(min_pos, a[i]);

        prod = prod * a[i];
    }
}
```

```
// If there are all zeros
// or no negative number present
if (count_zero == n ||
    (count_neg == 0 && count_zero > 0))
    return 0;

// If there are all positive
if (count_neg == 0)
    return min_pos;

// If there are even number of
// negative numbers and count_neg not 0
if (!(count_neg & 1) && count_neg != 0) {

    // Otherwise result is product of
    // all non-zeros divided by maximum
    // valued negative.
    prod = prod / max_neg;
}

return prod;
}

int main()
{
    int a[] = { -1, -1, -2, 4, 3 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << minProductSubset(a, n);
    return 0;
}
```

Java

```
// Java program to find maximum product of
// a subset.
class GFG {

    static int minProductSubset(int a[], int n)
    {
        if (n == 1)
            return a[0];

        // Find count of negative numbers,
        // count of zeros, maximum valued
        // negative number, minimum valued
        // positive number and product of
        // non-zero numbers
        int negmax = Integer.MIN_VALUE;
```

```
int posmin = Integer.MIN_VALUE;
int count_neg = 0, count_zero = 0;
int product = 1;

for (int i = 0; i < n; i++)
{
    // if number is zero, count it
    // but dont multiply
    if(a[i] == 0){
        count_zero++;
        continue;
    }

    // count the negative numbers
    // and find the max negative number
    if(a[i] < 0)
    {
        count_neg++;
        negmax = Math.max(negmax, a[i]);
    }

    // find the minimum positive number
    if(a[i] > 0 && a[i] < posmin)
        posmin = a[i];

    product *= a[i];
}

// if there are all zeroes
// or zero is present but no
// negative number is present
if (count_zero == n ||
    (count_neg == 0 && count_zero > 0))
    return 0;

// If there are all positive
if (count_neg == 0)
    return posmin;

// If there are even number except
// zero of negative numbers
if (count_neg % 2 == 0 && count_neg != 0)
{
    // Otherwise result is product of
    // all non-zeros divided by maximum
    // valued negative.
```

```
        product = product / negmax;
    }

    return product;
}

// main function
public static void main(String[] args)
{
    int a[] = { -1, -1, -2, 4, 3 };
    int n = 5;

    System.out.println(minProductSubset(a, n));
}
}

// This code is contributed by Arnab Kundu.
```

Python3

```
# Python3 program to find maximum
# product of a subset.

# def to find maximum
# product of a subset
def minProductSubset(a, n) :
    if (n == 1) :
        return a[0]

    # Find count of negative numbers,
    # count of zeros, maximum valued
    # negative number, minimum valued
    # positive number and product
    # of non-zero numbers
    max_neg = float('-inf')
    min_pos = float('inf')
    count_neg = 0
    count_zero = 0
    prod = 1
    for i in range(0,n) :

        # If number is 0, we don't
        # multiply it with product.
        if (a[i] == 0) :
            count_zero = count_zero + 1
            continue
```

```
# Count negatives and keep
# track of maximum valued
# negative.
if (a[i] < 0) :
    count_neg = count_neg + 1
    max_neg = max(max_neg, a[i])

# Track minimum positive
# number of array
if (a[i] > 0) :
    min_pos = min(min_pos, a[i])

prod = prod * a[i]

# If there are all zeros
# or no negative number
# present
if (count_zero == n or (count_neg == 0
                        and count_zero > 0)) :
    return 0;

# If there are all positive
if (count_neg == 0) :
    return min_pos

# If there are even number of
# negative numbers and count_neg
# not 0
if ((count_neg & 1) == 0 and
    count_neg != 0) :

    # Otherwise result is product of
    # all non-zeros divided by
    # maximum valued negative.
    prod = int(prod / max_neg)

return prod;

# Driver code
a = [ -1, -1, -2, 4, 3 ]
n = len(a)
print (minProductSubset(a, n))
# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
// C# program to find maximum product of
// a subset.
using System;

public class GFG {

    static int minProductSubset(int[] a, int n)
    {
        if (n == 1)
            return a[0];

        // Find count of negative numbers,
        // count of zeros, maximum valued
        // negative number, minimum valued
        // positive number and product of
        // non-zero numbers
        int negmax = int.MinValue;
        int posmin = int.MaxValue;
        int count_neg = 0, count_zero = 0;
        int product = 1;

        for (int i = 0; i < n; i++)
        {

            // if number is zero, count it
            // but dont multiply
            if (a[i] == 0) {
                count_zero++;
                continue;
            }

            // count the negetive numbers
            // and find the max negetive number
            if (a[i] < 0) {
                count_neg++;
                negmax = Math.Max(negmax, a[i]);
            }

            // find the minimum positive number
            if (a[i] > 0 && a[i] < posmin) {
                posmin = a[i];
            }

            product *= a[i];
        }

        // if there are all zeroes
        // or zero is present but no
```

```
// negetive number is present
if (count_zero == n || (count_neg == 0
                        && count_zero > 0))
    return 0;

// If there are all positive
if (count_neg == 0)
    return posmin;

// If there are even number except
// zero of negative numbers
if (count_neg % 2 == 0 && count_neg != 0)
{

    // Otherwise result is product of
    // all non-zeros divided by maximum
    // valued negative.
    product = product / negmax;
}

return product;
}

// main function
public static void Main()
{

    int[] a = new int[] { -1, -1, -2, 4, 3 };
    int n = 5;

    Console.WriteLine(minProductSubset(a, n));
}
}

// This code is contributed by Ajit.
```

PHP

```
<?php
// PHP program to find maximum
// product of a subset.

// Function to find maximum
// product of a subset
function minProductSubset($a, $n)
{

    if ($n == 1)
```

```
return $a[0];

// Find count of negative numbers,
// count of zeros, maximum valued
// negative number, minimum valued
// positive number and product
// of non-zero numbers
$max_neg = PHP_INT_MIN;
$min_pos = PHP_INT_MAX;
$count_neg = 0; $count_zero = 0;
$prod = 1;
for ($i = 0; $i < $n; $i++)
{
    // If number is 0, we don't
    // multiply it with product.
    if ($a[$i] == 0)
    {
        $count_zero++;
        continue;
    }

    // Count negatives and keep
    // track of maximum valued
    // negative.
    if ($a[$i] < 0)
    {
        $count_neg++;
        $max_neg = max($max_neg, $a[$i]);
    }

    // Track minimum positive
    // number of array
    if ($a[$i] > 0)
        $min_pos = min($min_pos, $a[$i]);

    $prod = $prod * $a[$i];
}

// If there are all zeros
// or no negative number
// present
if ($count_zero == $n ||
    ($count_neg == 0 &&
     $count_zero > 0))
    return 0;

// If there are all positive
```

```
if ($count_neg == 0)
    return $min_pos;

// If there are even number of
// negative numbers and count_neg
// not 0
if (!($count_neg & 1) &&
    $count_neg != 0)
{
    // Otherwise result is product of
    // all non-zeros divided by maximum
    // valued negative.
    $prod = $prod / $max_neg;
}

return $prod;
}

// Driver code
$a = array( -1, -1, -2, 4, 3 );
$n = sizeof($a);
echo(minProductSubset($a, $n));

// This code is contributed by Ajit.
?>
```

Output:

-24

Time Complexity : **O(n)**
Auxiliary Space : **O(1)**

Improved By : [jit_t](#), [andrew1234](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/minimum-product-subset-array/>

Chapter 213

Minimum sum of absolute difference of pairs of two arrays

Minimum sum of absolute difference of pairs of two arrays - GeeksforGeeks

Given two arrays $a[]$ and $b[]$ of equal length n . The task is to pair each element of array a to an element in array b , such that sum S of **absolute differences of all the pairs is minimum.**

Suppose, two elements $a[i]$ and $a[j]$ ($i \neq j$) of a are paired with elements $b[p]$ and $b[q]$ of b respectively,
then p should not be equal to q .

Examples:

```
Input : a[] = {3, 2, 1}
        b[] = {2, 1, 3}
Output : 0
```

```
Input : n = 4
        a[] = {4, 1, 8, 7}
        b[] = {2, 3, 6, 5}
Output : 6
```

The solution to the problem is a simple greedy approach. It consists of **two** steps.

Step 1 : Sort both the arrays in $O(n \log n)$ time.

Step 2 : Find absolute difference of each pair of **corresponding elements** (*elements at same index*) of both arrays and add the result to the sum S . The time complexity of this step is $O(n)$.

Hence, the overall time complexity of the program is $O(n \log n)$.

C++

```
// C++ program to find minimum sum of absolute
// differences of two arrays.
#include <bits/stdc++.h>
using namespace std;

// Returns minimum possible pairwise absolute
// difference of two arrays.
long long int findMinSum(int a[], int b[], int n)
{
    // Sort both arrays
    sort(a, a+n);
    sort(b, b+n);

    // Find sum of absolute differences
    long long int sum= 0 ;
    for (int i=0; i<n; i++)
        sum = sum + abs(a[i]-b[i]);

    return sum;
}

// Driver code
int main()
{
    // Both a[] and b[] must be of same size.
    long long int a[] = {4, 1, 8, 7};
    long long int b[] = {2, 3, 6, 5};
    int n = sizeof(a)/sizeof(a[0]);
    printf("%lld\n", findMinSum(a, b, n));
    return 0;
}
```

Java

```
// Java program to find minimum sum of
// absolute differences of two arrays.
import java.util.Arrays;

class MinSum
{
    // Returns minimum possible pairwise
    // absolute difference of two arrays.
    static long findMinSum(long a[], long b[], long n)
    {
        // Sort both arrays
        Arrays.sort(a);
        Arrays.sort(b);
```

```
// Find sum of absolute differences
long sum = 0 ;
for (int i = 0; i < n; i++)
    sum = sum + Math.abs(a[i] - b[i]);

return sum;
}

// Driver code
public static void main(String[] args)
{
    // Both a[] and b[] must be of same size.
    long a[] = {4, 1, 8, 7};
    long b[] = {2, 3, 6, 5};
    int n = a.length;
    System.out.println(findMinSum(a, b, n));
}
}

// This code is contributed by Raghav Sharma
```

Python3

```
# Python3 program to find minimum sum
# of absolute differences of two arrays.
def findMinSum(a, b, n):

    # Sort both arrays
    a.sort()
    b.sort()

    # Find sum of absolute differences
    sum = 0

    for i in range(n):
        sum = sum + abs(a[i] - b[i])

    return sum

# Driver program

# Both a[] and b[] must be of same size.
a = [4, 1, 8, 7]
b = [2, 3, 6, 5]
n = len(a)

print(findMinSum(a, b, n))
```

```
# This code is contributed by Anant Agarwal.
```

C#

```
// C# program to find minimum sum of
// absolute differences of two arrays.
using System;

class MinSum {

    // Returns minimum possible pairwise
    // absolute difference of two arrays.
    static long findMinSum(long []a, long []b,
                          long n)
    {

        // Sort both arrays
        Array.Sort(a);
        Array.Sort(b);

        // Find sum of absolute differences
        long sum = 0 ;
        for (int i = 0; i < n; i++)
            sum = sum + Math.Abs(a[i] - b[i]);

        return sum;
    }

    // Driver code
    public static void Main(String[] args)
    {
        // Both a[] and b[] must be of same size.
        long []a = {4, 1, 8, 7};
        long []b = {2, 3, 6, 5};
        int n = a.Length;
        Console.WriteLine(findMinSum(a, b, n));
    }
}

// This code is contributed by parashar...
```

PHP

```
<?php
// PHP program to find minimum sum
// of absolute differences of two
// arrays.
```

```
// Returns minimum possible pairwise
// absolute difference of two arrays.
function findMinSum($a, $b, $n)
{
    // Sort both arrays
    sort($a);
    sort($a, $n);
    sort($b);
    sort($b, $n);

    // Find sum of absolute
    // differences
    $sum= 0 ;
    for ($i = 0; $i < $n; $i++)
        $sum = $sum + abs($a[$i] -
                           $b[$i]);

    return $sum;
}

// Driver Code
// Both a[] and b[] must
// be of same size.
$a = array(4, 1, 8, 7);
$b = array(2, 3, 6, 5);
$n = sizeof($a);
echo(findMinSum($a, $b, $n));

// This code is contributed by nitin mittal.
?>
```

Output :

6

Improved By : [parashar](#), [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/minimum-sum-absolute-difference-pairs-two-arrays/>

Chapter 214

Minimum sum of two numbers formed from digits of an array

Minimum sum of two numbers formed from digits of an array - GeeksforGeeks

Given an array of digits (values are from 0 to 9), find the minimum possible sum of two numbers formed from digits of the array. All digits of given array must be used to form the two numbers.

Examples :

Input: [6, 8, 4, 5, 2, 3]

Output: 604

The minimum sum is formed by numbers
358 and 246

Input: [5, 3, 0, 7, 4]

Output: 82

The minimum sum is formed by numbers
35 and 047

A minimum number will be formed from set of digits when smallest digit appears at most significant position and next smallest digit appears at next most significant position and so on..

The idea is to sort the array in increasing order and build two numbers by alternating picking digits from the array. So first number is formed by digits present in odd positions in the array and second number is formed by digits from even positions in the array. Finally, we return the sum of first and second number.

Below is the implementation of above idea.

C/C++

```
// C++ program to find minimum sum of two numbers
// formed from digits of the array.
#include <bits/stdc++.h>
using namespace std;

// Function to find and return minimum sum of
// two numbers formed from digits of the array.
int solve(int arr[], int n)
{
    // sort the array
    sort(arr, arr + n);

    // let two numbers be a and b
    int a = 0, b = 0;
    for (int i = 0; i < n; i++)
    {
        // fill a and b with every alternate digit
        // of input array
        if (i & 1)
            a = a*10 + arr[i];
        else
            b = b*10 + arr[i];
    }

    // return the sum
    return a + b;
}

// Driver code
int main()
{
    int arr[] = {6, 8, 4, 5, 2, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Sum is " << solve(arr, n);
    return 0;
}
```

Java

```
// Java program to find minimum sum of two numbers
// formed from digits of the array.
import java.util.Arrays;

class GFG {

    // Function to find and return minimum sum of
    // two numbers formed from digits of the array.
    static int solve(int arr[], int n)
```

```
{  
  
    // sort the array  
    Arrays.sort(arr);  
  
    // let two numbers be a and b  
    int a = 0, b = 0;  
    for (int i = 0; i < n; i++)  
    {  
  
        // fill a and b with every alternate  
        // digit of input array  
        if (i % 2 != 0)  
            a = a * 10 + arr[i];  
        else  
            b = b * 10 + arr[i];  
    }  
  
    // return the sum  
    return a + b;  
}  
  
//driver code  
public static void main (String[] args)  
{  
    int arr[] = {6, 8, 4, 5, 2, 3};  
    int n = arr.length;  
  
    System.out.print("Sum is "  
                    + solve(arr, n));  
}  
}  
  
//This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program to find minimum sum of two  
# numbers formed from digits of the array.  
  
# Function to find and return minimum sum of  
# two numbers formed from digits of the array.  
def solve(arr, n):  
  
    # sort the array  
    arr.sort()
```

```
# let two numbers be a and b
a = 0; b = 0
for i in range(n):

    # Fill a and b with every alternate
    # digit of input array
    if (i % 2 != 0):
        a = a * 10 + arr[i]
    else:
        b = b * 10 + arr[i]

# return the sum
return a + b

# Driver code
arr = [6, 8, 4, 5, 2, 3]
n = len(arr)
print("Sum is ", solve(arr, n))

# This code is contributed by Anant Agarwal.
```

C#

```
// C# program to find minimum
// sum of two numbers formed
// from digits of the array.
using System;

class GFG
{
    // Function to find and return
    // minimum sum of two numbers
    // formed from digits of the array.
    static int solve(int []arr, int n)
    {
        // sort the array
        Array.Sort(arr);

        // let two numbers be a and b
        int a = 0, b = 0;
        for (int i = 0; i < n; i++)
        {
            // fill a and b with every alternate digit
            // of input array
            if (i % 2 != 0)
                a = a * 10 + arr[i];
            else
                b = b * 10 + arr[i];
        }
    }
}
```

```
}

// return the sum
return a + b;
}

// Driver code
public static void Main ()
{
    int []arr = {6, 8, 4, 5, 2, 3};
    int n = arr.Length;
    Console.WriteLine("Sum is " + solve(arr, n));
}
}

// This code is contributed by Anant Agarwal.
```

PHP

```
<?php
// PHP program to find minimum
// sum of two numbers formed
// from digits of the array.

// Function to find and return
// minimum sum of two numbers
// formed from digits of the array.
function solve($arr, $n)
{
    // sort the array
    sort($arr); sort($arr , $n);

    // let two numbers be a and b
    $a = 0; $b = 0;
    for ($i = 0; $i < $n; $i++)
    {
        // fill a and b with every
        // alternate digit of input array
        if ($i & 1)
            $a = $a * 10 + $arr[$i];
        else
            $b = $b * 10 + $arr[$i];
    }

    // return the sum
    return $a + $b;
}
```

```
// Driver code
$arr = array(6, 8, 4, 5, 2, 3);
$n = sizeof($arr);
echo "Sum is " , solve($arr, $n);

// This code is contributed by nitin mittal.
?>
```

Output :

Sum is 604

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/minimum-sum-two-numbers-formed-digits-array/>

Chapter 215

Minimum swap required to convert binary tree to binary search tree

Minimum swap required to convert binary tree to binary search tree - GeeksforGeeks

Given the array representation of Complete Binary Tree i.e, if index i is the parent, index $2*i + 1$ is the left child and index $2*i + 2$ is the right child. The task is to find the minimum number of swap required to convert it into Binary Search Tree.

Examples:

Input : arr[] = { 5, 6, 7, 8, 9, 10, 11 }

Output : 3

Binary tree of the given array:

Swap 1: Swap node 8 with node 5.

Swap 2: Swap node 9 with node 10.

Swap 3: Swap node 10 with node 7.

So, minimum 3 swaps are required.

Input : arr[] = { 1, 2, 3 }

Output : 1

Binary tree of the given array:

After swapping node 1 with node 2.

So, only 1 swap required.

The idea is to use the fact that inorder traversal of Binary Search Tree is in increasing order of their value.

So, find the inorder traversal of the Binary Tree and store it in the array and try to sort the array. The minimum number of swap required to get the array sorted will be the answer. Please refer below post to find minimum number of swaps required to get the array sorted.

[Minimum number of swaps required to sort an array](#)

Time Complexity: $O(n \log n)$.

Exercise: Can we extend this to normal binary tree, i.e., a binary tree represented using left and right pointers, and not necessarily complete?

Source

<https://www.geeksforgeeks.org/minimum-swap-required-convert-binary-tree-binary-search-tree/>

Chapter 216

Minimum swaps required to Sort Binary array

Minimum swaps required to Sort Binary array - GeeksforGeeks

Given a binary array, task is to sort this binary array.

Examples :

```
Input : [0, 0, 1, 0, 1, 0, 1, 1]
Output : 3
1st swap : [0, 0, 1, 0, 0, 1, 1, 1]
2nd swap : [0, 0, 0, 1, 0, 1, 1, 1]
3rd swap : [0, 0, 0, 0, 1, 1, 1, 1]

Input : Array = [0, 1, 0, 1, 0]
Output : 3
```

Approach :

This can be done by finding number of zeroes to the right side of every 1 and add them. In order to sort the array every one always has to perform a swap operation with every zero on its right side. So the total number of swap operations for a particular 1 in array is the number of zeroes on its right hand side. Find the number of zeroes on right side for every one i.e. the number of swaps and add them all to obtain the total number of swaps.

Implementation :

C++

```
// C++ code to find minimum number of
// swaps to sort a binary array
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to find minimum swaps to
// sort an array of 0s and 1s.
int findMinSwaps(int arr[], int n)
{
    // Array to store count of zeroes
    int noOfZeroes[n];
    memset(noOfZeroes, 0, sizeof(noOfZeroes));

    int i, count = 0;

    // Count number of zeroes
    // on right side of every one.
    noOfZeroes[n - 1] = 1 - arr[n - 1];
    for (i = n - 2; i >= 0; i--) {
        noOfZeroes[i] = noOfZeroes[i + 1];
        if (arr[i] == 0)
            noOfZeroes[i]++;
    }

    // Count total number of swaps by adding number
    // of zeroes on right side of every one.
    for (i = 0; i < n; i++) {
        if (arr[i] == 1)
            count += noOfZeroes[i];
    }

    return count;
}

// Driver code
int main()
{
    int arr[] = { 0, 0, 1, 0, 1, 0, 1, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findMinSwaps(arr, n);
    return 0;
}
```

Java

```
// Java code to find minimum number of
// swaps to sort a binary array
class gfg {

    static int findMinSwaps(int arr[], int n)
    {
```

```
// Array to store count of zeroes
int noOfZeroes[] = new int[n];
int i, count = 0;

// Count number of zeroes
// on right side of every one.
noOfZeroes[n - 1] = 1 - arr[n - 1];
for (i = n - 2; i >= 0; i--)
{
    noOfZeroes[i] = noOfZeroes[i + 1];
    if (arr[i] == 0)
        noOfZeroes[i]++;
}

// Count total number of swaps by adding number
// of zeroes on right side of every one.
for (i = 0; i < n; i++)
{
    if (arr[i] == 1)
        count += noOfZeroes[i];
}
return count;
}

// Driver Code
public static void main(String args[])
{
    int ar[] = { 0, 0, 1, 0, 1, 0, 1, 1 };
    System.out.println(findMinSwaps(ar, ar.length));
}
}

// This code is contributed by Niraj_Pandey.
```

Python3

```
# Python3 code to find minimum number of
# swaps to sort a binary array

# Function to find minimum swaps to
# sort an array of 0s and 1s.
def findMinSwaps(arr, n) :
    # Array to store count of zeroes
    noOfZeroes = [0] * n
    count = 0

    # Count number of zeroes
    # on right side of every one.
```

```
noOfZeroes[n - 1] = 1 - arr[n - 1]
for i in range(n-2, -1, -1) :
    noOfZeroes[i] = noOfZeroes[i + 1]
    if (arr[i] == 0) :
        noOfZeroes[i] = noOfZeroes[i] + 1

# Count total number of swaps by adding
# number of zeroes on right side of
# every one.
for i in range(0, n) :
    if (arr[i] == 1) :
        count = count + noOfZeroes[i]

return count

# Driver code
arr = [ 0, 0, 1, 0, 1, 0, 1, 1 ]
n = len(arr)
print (findMinSwaps(arr, n))

# This code is contributed by Manish Shaw
# (manishshaw1)
```

C#

```
// C# code to find minimum number of
// swaps to sort a binary array
using System;

class GFG {

    static int findMinSwaps(int []arr, int n)
    {

        // Array to store count of zeroes
        int []noOfZeroes = new int[n];
        int i, count = 0;

        // Count number of zeroes
        // on right side of every one.
        noOfZeroes[n - 1] = 1 - arr[n - 1];
        for (i = n - 2; i >= 0; i--)
        {
            noOfZeroes[i] = noOfZeroes[i + 1];
            if (arr[i] == 0)
                noOfZeroes[i]++;
        }
    }
}
```

```
// Count total number of swaps by
// adding number of zeroes on right
// side of every one.
for (i = 0; i < n; i++)
{
    if (arr[i] == 1)
        count += noOfZeroes[i];
}

return count;
}

// Driver Code
public static void Main()
{
    int []ar = { 0, 0, 1, 0, 1,
                0, 1, 1 };

    Console.WriteLine(
        findMinSwaps(ar, ar.Length));
}
}

// This code is contributed by vt_m.
```

Output :

3

Time Complexity : O(n)

Auxiliary Space : O(n)

Improved By : [Niraj_Pandey](#), [vt_m](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/minimum-swaps-required-sort-binary-array/>

Chapter 217

Minimum swaps so that binary search can be applied

Minimum swaps so that binary search can be applied - GeeksforGeeks

Given an unsorted array of length n and an integer k, find the minimum swaps to get the position of k before using the binary search. Here we can swap any two numbers as many times as we want. If we cannot get the position by swapping elements, print “-1”.

Examples:

```
Input : arr = {3, 10, 6, 7, 2, 5, 4}
        k = 4
Output : 2
Explanation :
Here, after swapping 3 with 7 and 2 with 5, the
final array will look like {7, 10, 6, 3, 5, 2, 4}.
Now, if we provide this array to binary search we
can get the position of 4.
```

```
Input : arr = {3, 10, 6, 7, 2, 5, 4}
        k = 10
Output : -1
Explanation :
Here, we can not get the position of 10 if we
provide this array to the binary search even
not with swapping some pairs.
```

Approach: Before discussing the approach, we have to assume that here we are not supposed to swap the pairs. We just need to count the minimum number of swaps so that if we provide the newly created array to the binary search, we can get the position of k. In order to do so, we need to pass the given array to binary search and focus on the following things:-

- Before going to the binary search we need to calculate the number of minimum elements i.e. **num_min** of k and number of maximum elements i.e. **num_max** of k. Here, num_min denotes the number of smaller available elements of k and num_max denotes the number of greater available elements that we can use for swapping
- The **actual position of k** in the given array.

Now following are the test cases that will occur during implementing the binary search:-

Case 1: If arr[mid] is greater than k, but position of k is greater than mid. Binary search would take us to (arr[0] to arr[mid-1]). But actually our element is in between (arr[mid+1] to arr[last element]). So, in order to go in the right direction, we need something smaller than k so that we can swap it with arr[mid] and we can go between arr[mid+1] to arr[last_element]. So, here we require one swap i.e. **need_minimum**.

Case 2: If arr[mid] is less than k but position of k is smaller than mid. Binary search would take us to (arr[mid+1] to arr[last_element]). But actually our element is in between (arr[0] to arr[mid-1]). So, in order to go in the right direction, we need something greater than k so that we can swap it with arr[mid] and we can go between arr[0] to arr[mid-1]. So, here we required one swap i.e. **need_maximum**.

Case 3:

If arr[mid] is greater than k and position of k is lesser than mid. Now, in this case binary search would work fine. But wait, here is the important thing on which we have to work on. As we know in this case binary search will work fine, arr[mid] is at right position so this will not be used in any swap so here we have to decrease the one of it's greater available element i.e. from num_max. Same goes with the case when arr[mid] is lesser than k and position of k is greater than mid. Here, we have to decrease one of it's smaller available element i.e. from num_min.

Case 4: If arr[mid] == k Or pos == mid then we can easily come out from the binary search.

So, till now we have calculated the **need_minimum** i.e. number of minimum elements required for swapping, **need_maximum** i.e. number of maximum elements required for swapping, **num_max** i.e. total number of greater elements from k that are still available for swapping and **num_min** i.e. total number of minimum elements from k that are available for swapping.

Now here we have to consider two cases:

Case 1: If need_minimum is greater than need_maximum. In this case, we have to swap all these needed maximum elements with the smaller of k. So we have to use the smaller elements from num_min. Now all the need_maximum swaps are done. Here the main thing is that when we swapped all these needed maximum elements with smaller elements, these smaller elements got their right positions. So, indirectly we have done some of the needed smaller elements swaps and that will be calculated as **need_minimum - need_maximum** and also available num_min will be **num_min - need_maximum**. Now, we have to calculate the remaining need_minimum swaps. We can calculate these swaps, if we have enough num_min i.e. num_min > need_minimum. For this case, swaps will be **need_maximum + need_minimum** otherwise it will be -1. The same concept goes with the case when we have need_minimum is smaller than need_maximum.

Below is the basic implementation of the above approach:-

```
// CPP program to find Minimum number
// of swaps to get the position of
// the element if we provide an
// unsorted array to binary search.
#include <bits/stdc++.h>
using namespace std;

// Function to find minimum swaps.
int findMinimumSwaps(int* arr, int n,
                      int k)
{
    int pos, num_min, num_max,
        need_minimum, need_maximum, swaps;
    num_min = num_max = need_minimum = 0;
    need_maximum = swaps = 0;

    // Here we are getting number of
    // smaller and greater elements of k.
    for (int i = 0; i < n; i++) {
        if (arr[i] < k)
            num_min++;

        else if (arr[i] > k)
            num_max++;
    }

    // Here we are calculating the actual
    // position of k in the array.
    for (int i = 0; i < n; i++) {
        if (arr[i] == k) {
            pos = i;
            break;
        }
    }

    int left, right, mid;
    left = 0;
    right = n - 1;

    // Implementing binary search as
    // per the above-discussed cases.
    while (left <= right) {
        mid = (left + right) / 2;

        // If we find the k.
        if (arr[mid] == k) {
```

```
        break;
    }

    else if (arr[mid] > k) {

        // If we need minimum
        // element swap.
        if (pos > mid)
            need_minimum++;

        else

            // Else the elemnt is
            // at the right position.
            num_min--;

        left = mid + 1;
    }

    else {
        if (pos < mid)

            // If we need maximum
            // element swap.
            need_maximum++;

        else

            // Else element is at
            // the right position
            num_max--;

        right = mid - 1;
    }
}

// Calculating the required swaps.
if (need_minimum > need_maximum) {
    swaps = swaps + need_maximum;
    num_min = num_min - need_maximum;
    need_minimum = need_minimum - need_maximum;
    need_maximum = 0;
}

else {
    swaps = swaps + need_minimum;
    num_max = num_max - need_minimum;
    need_maximum = need_maximum - need_minimum;
}
```

```
    need_minimum = 0;
}

// If it is impossible.
if (need_maximum > num_max || need_minimum > num_min)
    return -1;

else
    return (swaps + need_maximum + need_minimum);
}

// Driver function
int main()
{
    int arr[] = { 3, 10, 6, 7, 2, 5, 4 }, k = 4;
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findMinimumSwaps(arr, n, k);
}
```

Output:

2

Source

<https://www.geeksforgeeks.org/minimum-swaps-so-that-binary-search-can-be-applied/>

Chapter 218

Minimum swaps to make two arrays identical

Minimum swaps to make two arrays identical - GeeksforGeeks

Given two arrays which have same values but in different order, we need to make second array same as first array using minimum number of swaps.

Examples:

```
Input  : arrA[] = {3, 6, 4, 8},  
         arrB[] = {4, 6, 8, 3}  
Output : 2  
we can make arrB to same as arrA in 2 swaps  
which are shown below,  
swap 4 with 8,   arrB = {8, 6, 4, 3}  
swap 8 with 3,   arrB = {3, 6, 4, 8}
```

This problem can be solved by modifying the array B. We save the index of array A elements in array B i.e. if ith element of array A is at jth position in array B, then we will make $arrB[i] = j$

For above given example, modified array B will be, $arrB = \{3, 1, 0, 2\}$. This modified array represents distribution of array A element in array B and our goal is to sort this modified array in minimum number of swaps because after sorting only array B element will be aligned with array A elements.

Now count of [minimum swaps for sorting an array](#) can be found by visualizing the problem as a graph, this problem is already explained in [previous article](#).

So we count these swaps in modified array and that will be our final answer.

Please see below code for better understanding.

```
// C++ program to make an array same to another  
// using minimum number of swap
```

```
#include <bits/stdc++.h>
using namespace std;

// Function returns the minimum number of swaps
// required to sort the array
// This method is taken from below post
// https://www.geeksforgeeks.org/minimum-number-swaps-required-sort-array/
int minSwapsToSort(int arr[], int n)
{
    // Create an array of pairs where first
    // element is array element and second element
    // is position of first element
    pair<int, int> arrPos[n];
    for (int i = 0; i < n; i++)
    {
        arrPos[i].first = arr[i];
        arrPos[i].second = i;
    }

    // Sort the array by array element values to
    // get right position of every element as second
    // element of pair.
    sort(arrPos, arrPos + n);

    // To keep track of visited elements. Initialize
    // all elements as not visited or false.
    vector<bool> vis(n, false);

    // Initialize result
    int ans = 0;

    // Traverse array elements
    for (int i = 0; i < n; i++)
    {
        // already swapped and corrected or
        // already present at correct pos
        if (vis[i] || arrPos[i].second == i)
            continue;

        // find out the number of node in
        // this cycle and add in ans
        int cycle_size = 0;
        int j = i;
        while (!vis[j])
        {
            vis[j] = 1;

            // move to next node
```

```
j = arrPos[j].second;
cycle_size++;
}

// Update answer by adding current cycle.
ans += (cycle_size - 1);
}

// Return result
return ans;
}

// method returns minimum number of swap to make
// array B same as array A
int minSwapToMakeArraySame(int a[], int b[], int n)
{
    // map to store position of elements in array B
    // we basically store element to index mapping.
    map<int, int> mp;
    for (int i = 0; i < n; i++)
        mp[b[i]] = i;

    // now we're storing position of array A elements
    // in array B.
    for (int i = 0; i < n; i++)
        b[i] = mp[a[i]];

    /* We can uncomment this section to print modified
       b array
    for (int i = 0; i < N; i++)
        cout << b[i] << " ";
    cout << endl; */

    // returning minimum swap for sorting in modified
    // array B as final answer
    return minSwapsToSort(b, n);
}

// Driver code to test above methods
int main()
{
    int a[] = {3, 6, 4, 8};
    int b[] = {4, 6, 8, 3};

    int n = sizeof(a) / sizeof(int);
    cout << minSwapToMakeArraySame(a, b, n);
    return 0;
}
```

Output:

2

Source

<https://www.geeksforgeeks.org/minimum-swaps-to-make-two-array-identical/>

Chapter 219

Minimum swaps to reach permuted array with at most 2 positions left swaps allowed

Minimum swaps to reach permuted array with at most 2 positions left swaps allowed - GeeksforGeeks

Given a permuted array of length N of first N natural numbers, we need to tell the minimum number of swaps required in the sorted array of first N natural number to reach given permuted array where a number can be swapped with at most 2 positions left to it. If it is not possible to reach permuted array by above swap condition then print not possible.

Examples:

```
Input : arr = [1, 2, 5, 3, 4]
Output : 2
We can reach to above-permuted array
in total 2 swaps as shown below,
[1, 2, 3, 4, 5] -> [1, 2, 3, 5, 4] ->
[1, 2, 5, 3, 4]
```

```
Input : arr[] = [5, 1, 2, 3, 4]
Output : Not Possible
It is not possible to reach above array
just by swapping numbers 2 positions left
to it.
```

We can solve this problem using [inversions](#). As we can see that if a number is at a position which is more than 2 places away from its actual position then it is not possible to reach there just by swapping with elements at 2 left positions and if all element satisfy this property (there are ≤ 2 elements smaller than it on the right) then answer will simply be total

number of inversions in the array because that many swaps will be needed to transform the array into permuted array.

We can find the number of inversions in $N \log N$ time using merge sort technique explained [here](#) so total time complexity of solution will be $O(N \log N)$ only.

```
// C++ program to find minimum number of swaps
// to reach a permutation with at most 2 left
// swaps allowed for every element
#include <bits/stdc++.h>
using namespace std;

/* This function merges two sorted arrays and returns inversion
   count in the arrays.*/
int merge(int arr[], int temp[], int left, int mid, int right)
{
    int inv_count = 0;

    int i = left; /* i is index for left subarray*/
    int j = mid; /* j is index for right subarray*/
    int k = left; /* k is index for resultant merged subarray*/
    while ((i <= mid - 1) && (j <= right))
    {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
        {
            temp[k++] = arr[j++];
            inv_count = inv_count + (mid - i);
        }
    }

    /* Copy the remaining elements of left subarray
     (if there are any) to temp*/
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    /* Copy the remaining elements of right subarray
     (if there are any) to temp*/
    while (j <= right)
        temp[k++] = arr[j++];

    /*Copy back the merged elements to original array*/
    for (i = left; i <= right; i++)
        arr[i] = temp[i];

    return inv_count;
}
```

```
/* An auxiliary recursive function that sorts the
   input array and returns the number of inversions
   in the array. */
int _mergeSort(int arr[], int temp[], int left, int right)
{
    int mid, inv_count = 0;
    if (right > left)
    {
        /* Divide the array into two parts and
           call _mergeSortAndCountInv() for each
           of the parts */
        mid = (right + left)/2;

        /* Inversion count will be sum of inversions
           in left-part, right-part and number of inversions
           in merging */
        inv_count = _mergeSort(arr, temp, left, mid);
        inv_count += _mergeSort(arr, temp, mid+1, right);

        /*Merge the two parts*/
        inv_count += merge(arr, temp, left, mid+1, right);
    }
    return inv_count;
}

/* This function sorts the input array and returns the
   number of inversions in the array */
int mergeSort(int arr[], int array_size)
{
    int *temp = (int *)malloc(sizeof(int)*array_size);
    return _mergeSort(arr, temp, 0, array_size - 1);
}

// method returns minimum number of swaps to reach
// permuted array 'arr'
int minSwapToReachArr(int arr[], int N)
{
    // loop over all elements to check Invalid
    // permutation condition
    for (int i = 0; i < N; i++)
    {
        /* if an element is at distance more than 2
           from its actual position then it is not
           possible to reach permuted array just
           by swapping with 2 positions left elements
           so returning -1 */
        if ((arr[i] - 1) - i > 2)
```

```
        return -1;
    }

/* If permuted array is not Invalid, then number
   of Inversion in array will be our final answer */
int numOfInversion = mergeSort(arr, N);
return numOfInversion;
}

// Driver code to test above methods
int main()
{
    // change below example
    int arr[] = {1, 2, 5, 3, 4};
    int N = sizeof(arr) / sizeof(int);
    int res = minSwapToReachArr(arr, N);
    if (res == -1)
        cout << "Not Possible\n";
    else
        cout << res << endl;
    return 0;
}
```

Output:

2

Source

<https://www.geeksforgeeks.org/minimum-swaps-reach-permuted-array-2-positions-left-swaps-allowed/>

Chapter 220

Most frequent element in an array

Most frequent element in an array - GeeksforGeeks

Given an array, find the most frequent element in it. If there are multiple elements that appear maximum number of times, print any one of them.

Examples:

```
Input : arr[] = {1, 3, 2, 1, 4, 1}
Output : 1
1 appears three times in array which
is maximum frequency.
```

```
Input : arr[] = {10, 20, 10, 20, 30, 20, 20}
Output : 20
```

A **simple solution** is to run two loops. The outer loop picks all elements one by one. The inner loop finds frequency of the picked element and compares with the maximum so far. Time complexity of this solution is $O(n^2)$

A **better solution** is to do sorting. We first sort the array, then linearly traverse the array.

C++

```
// CPP program to find the most frequent element
// in an array.
#include <bits/stdc++.h>
using namespace std;

int mostFrequent(int arr[], int n)
{
```

```
// Sort the array
sort(arr, arr + n);

// find the max frequency using linear traversal
int max_count = 1, res = arr[0], curr_count = 1;
for (int i = 1; i < n; i++) {
    if (arr[i] == arr[i - 1])
        curr_count++;
    else {
        if (curr_count > max_count) {
            max_count = curr_count;
            res = arr[i - 1];
        }
        curr_count = 1;
    }
}

// If last element is most frequent
if (curr_count > max_count)
{
    max_count = curr_count;
    res = arr[n - 1];
}

return res;
}

// driver program
int main()
{
    int arr[] = { 1, 5, 2, 1, 3, 2, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << mostFrequent(arr, n);
    return 0;
}
```

Java

```
//Java program to find the most frequent element
//in an array
import java.util.*;

class GFG {

    static int mostFrequent(int arr[], int n)
    {

        // Sort the array
```

```
Arrays.sort(arr);

// find the max frequency using linear
// traversal
int max_count = 1, res = arr[0];
int curr_count = 1;

for (int i = 1; i < n; i++)
{
    if (arr[i] == arr[i - 1])
        curr_count++;
    else
    {
        if (curr_count > max_count)
        {
            max_count = curr_count;
            res = arr[i - 1];
        }
        curr_count = 1;
    }
}

// If last element is most frequent
if (curr_count > max_count)
{
    max_count = curr_count;
    res = arr[n - 1];
}

return res;
}

// Driver program
public static void main (String[] args) {

    int arr[] = {1, 5, 2, 1, 3, 2, 1};
    int n = arr.length;

    System.out.println(mostFrequent(arr,n));
}

// This code is contributed by Akash Singh.
```

Python3

```
# Python3 program to find the most
```

```
# frequent element in an array.

def mostFrequent(arr, n):

    # Sort the array
    arr.sort()

    # find the max frequency using
    # linear traversal
    max_count = 1; res = arr[0]; curr_count = 1

    for i in range(1, n):
        if (arr[i] == arr[i - 1]):
            curr_count += 1

        else :
            if (curr_count > max_count):
                max_count = curr_count
                res = arr[i - 1]

            curr_count = 1

    # If last element is most frequent
    if (curr_count > max_count):

        max_count = curr_count
        res = arr[n - 1]

    return res

# Driver Code
arr = [1, 5, 2, 1, 3, 2, 1]
n = len(arr)
print(mostFrequent(arr, n))

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
// C# program to find the most
// frequent element in an array
using System;

class GFG {

    static int mostFrequent(int []arr, int n)
    {
```

```
// Sort the array
Array.Sort(arr);

// find the max frequency using
// linear traversal
int max_count = 1, res = arr[0];
int curr_count = 1;

for (int i = 1; i < n; i++)
{
    if (arr[i] == arr[i - 1])
        curr_count++;
    else
    {
        if (curr_count > max_count)
        {
            max_count = curr_count;
            res = arr[i - 1];
        }
        curr_count = 1;
    }
}

// If last element is most frequent
if (curr_count > max_count)
{
    max_count = curr_count;
    res = arr[n - 1];
}

return res;
}

// Driver code
public static void Main ()
{

    int []arr = {1, 5, 2, 1, 3, 2, 1};
    int n = arr.Length;

    Console.WriteLine(mostFrequent(arr,n));
}

}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find the
// most frequent element
// in an array.

function mostFrequent( $arr, $n)
{

    // Sort the array
    sort($arr);
    sort($arr , $n);

    // find the max frequency
    // using linear traversal
    $max_count = 1;
    $res = $arr[0];
    $curr_count = 1;
    for ($i = 1; $i < $n; $i++)
    {
        if ($arr[$i] == $arr[$i - 1])
            $curr_count++;
        else
        {
            if ($curr_count > $max_count)
            {
                $max_count = $curr_count;
                $res = $arr[$i - 1];
            }
            $curr_count = 1;
        }
    }

    // If last element
    // is most frequent
    if ($curr_count > $max_count)
    {
        $max_count = $curr_count;
        $res = $arr[$n - 1];
    }

    return $res;
}

// Driver Code
{
    $arr = array(1, 5, 2, 1, 3, 2, 1);
    $n = sizeof($arr) / sizeof($arr[0]);
    echo mostFrequent($arr, $n);
```

```
    return 0;
}

// This code is contributed by nitin mittal
?>
```

Output :

1

Time Complexity : $O(n \log n)$
Auxiliary Space : $O(1)$

An **efficient solution** is to use hashing. We create a hash table and store elements and their frequency counts as key value pairs. Finally we traverse the hash table and print the key with maximum value.

C++

```
// CPP program to find the most frequent element
// in an array.
#include <bits/stdc++.h>
using namespace std;

int mostFrequent(int arr[], int n)
{
    // Insert all elements in hash.
    unordered_map<int, int> hash;
    for (int i = 0; i < n; i++)
        hash[arr[i]]++;

    // find the max frequency
    int max_count = 0, res = -1;
    for (auto i : hash) {
        if (max_count < i.second) {
            res = i.first;
            max_count = i.second;
        }
    }

    return res;
}

// driver program
int main()
{
    int arr[] = { 1, 5, 2, 1, 3, 2, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    cout << mostFrequent(arr, n);
    return 0;
}
```

Java

```
//Java program to find the most frequent element
//in an array
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

class GFG {

    static int mostFrequent(int arr[], int n)
    {

        // Insert all elements in hash
        Map<Integer, Integer> hp =
            new HashMap<Integer, Integer>();

        for(int i = 0; i < n; i++)
        {
            int key = arr[i];
            if(hp.containsKey(key))
            {
                int freq = hp.get(key);
                freq++;
                hp.put(key, freq);
            }
            else
            {
                hp.put(key, 1);
            }
        }

        // find max frequency.
        int max_count = 0, res = -1;

        for(Entry<Integer, Integer> val : hp.entrySet())
        {
            if (max_count < val.getValue())
            {
                res = val.getKey();
                max_count = val.getValue();
            }
        }
    }
}
```

```
        return res;
    }

// Driver code
public static void main (String[] args) {

    int arr[] = {1, 5, 2, 1, 3, 2, 1};
    int n = arr.length;

    System.out.println(mostFrequent(arr, n));
}
}

// This code is contributed by Akash Singh.
```

C#

```
// C# program to find the most
// frequent element in an array
using System;
using System.Collections.Generic;

class GFG
{
    static int mostFrequent(int []arr,
                           int n)
    {
        // Insert all elements in hash
        Dictionary<int, int> hp =
            new Dictionary<int, int>();

        for (int i = 0; i < n; i++)
        {
            int key = arr[i];
            if(hp.ContainsKey(key))
            {
                int freq = hp[key];
                freq++;
                hp[key] = freq;
            }
            else
                hp.Add(key, 1);
        }

        // find max frequency.
        int min_count = 0, res = -1;

        foreach (KeyValuePair<int,
```

```
        int> pair in hp)
{
    if (min_count < pair.Value)
    {
        res = pair.Key;
        min_count = pair.Value;
    }
}
return res;
}

// Driver code
static void Main ()
{
    int []arr = new int[]{1, 5, 2,
                         1, 3, 2, 1};
    int n = arr.Length;

    Console.WriteLine(mostFrequent(arr, n));
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

1

Time Complexity : O(n)
Auxiliary Space : O(n)

Improved By : [vt_m](#), [nitin mittal](#), [dipesh_jain](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/frequent-element-array/>

Chapter 221

Most frequent word in an array of strings

Most frequent word in an array of strings - GeeksforGeeks

Given an array of words find the most occurring word in it

Examples:

```
Input : arr[] = {"geeks", "for", "geeks", "a",
                 "portal", "to", "learn", "can",
                 "be", "computer", "science",
                 "zoom", "yup", "fire", "in",
                 "be", "data"}

Output : Geeks
"geeks" is the most frequent word as it
occurs 3 times
```

A **simple solution** is to run two loops and count occurrences of every word. Time complexity of this solution is $O(n * n * \text{MAX_WORD_LEN})$.

An **efficient solution** is to use [Trie data structure](#). The idea is simple first we will insert in trie. In trie, we keep counts of words ending at a node. We do preorder traversal and compare count present at each node and find the maximum occurring word

```
// CPP code to find most frequent word in
// an array of strings
#include <bits/stdc++.h>
using namespace std;

/*structing the trie*/
struct Trie {
```

```
string key;
int cnt;
unordered_map<char, Trie*> map;
};

/* Function to return a new Trie node */
Trie* getNewTrieNode()
{
    Trie* node = new Trie;
    node->cnt = 0;
    return node;
}

/* function to insert a string */
void insert(Trie*& root, string &str)
{
    // start from root node
    Trie* temp = root;

    for (int i=0; i<str.length(); i++) {

        char x = str[i];

        /*a new node if path doesn't exists*/
        if (temp->map.find(x) == temp->map.end())
            temp->map[x] = getNewTrieNode();

        // go to next node
        temp = temp->map[x];
    }

    // store key and its count in leaf nodes
    temp->key = str;
    temp->cnt += 1;
}

/* function for preorder traversal */
bool preorder(Trie* temp, int& maxcnt, string& key)
{
    if (temp == NULL)
        return false;

    for (auto it : temp->map) {

        /*leaf node will have non-zero count*/
        if (maxcnt < it.second->cnt) {
            key = it.second->key;
            maxcnt = it.second->cnt;
        }
    }
}
```

```
}

    // recurse for current node children
    preorder(it.second, maxcnt, key);
}
}

void mostFrequentWord(string arr[], int n)
{
    // Insert all words in a Trie
    Trie* root = getNewTrieNode();
    for (int i = 0; i < n; i++)
        insert(root, arr[i]);

    // Do preorder traversal to find the
    // most frequent word
    string key;
    int cnt = 0;
    preorder(root, cnt, key);

    cout << "The word that occurs most is : "
        << key << endl;
    cout << "No of times: " << cnt << endl;
}

// Driver code
int main()
{
    // given set of keys
    string arr[] = {"geeks", "for", "geeks", "a",
                    "portal", "to", "learn", "can", "be",
                    "computer", "science", "zoom", "yup",
                    "fire", "in", "be", "data", "geeks"};
    int n = sizeof(arr) / sizeof(arr[0]);

    mostFrequentWord(arr, n);

    return 0;
}
```

Output:

```
The word that occurs most is : geeks
No of times: 3
```

Time Complexity : $O(n * \text{MAX_WORD_LEN})$

Another efficient solution is to use hashing. Please refer [Find winner of an election where votes are represented as candidate names](#) for details.

Source

<https://www.geeksforgeeks.org/frequent-word-array-strings/>

Chapter 222

Multiset Equivalence Problem

Multiset Equivalence Problem - GeeksforGeeks

Unlike a set, a multiset may contain multiple occurrences of same number. The *multiset* equivalence problem states to check if two given multisets are equal or not. For example let **A** = {1, 2, 3} and **B** = {1, 1, 2, 3}. Here **A** is set but **B** is not (1 occurs twice in B), whereas **A** and **B** are both multisets. More formally, “Are the *sets* of pairs defined

as $\{ \{x, y\} \mid \text{Frequency}(x) >= \text{Frequency}(y) \}$ equal for the two given multisets?”

Given two multisets A and B, write a program to check if the two multisets are equal.

Note: Elements in the multisets can be of order 10^9

Examples:

```
Input : A = {1, 1, 3, 4},  
        B = {1, 1, 3, 4}  
Output : Yes
```

```
Input : A = {1, 3},  
        B = {1, 1}  
Output : No
```

Since the elements are as large as 10^9 we cannot use [direct index table](#).

One solution is to sort both multisets and compare them one by one.

```
// C++ program to check if two given multisets  
// are equivalent  
#include <bits/stdc++.h>  
using namespace std;
```

```

bool areSame(vector<int>& a, vector<int>& b)
{
    // sort the elements of both multisets
    sort(a.begin(), a.end());
    sort(b.begin(), b.end());

    // Return true if both multisets are same.
    return (a == b);
}

int main()
{
    vector<int> a({ 7, 7, 5 }), b({ 7, 5, 5 });
    if (areSame(a, b))
        cout << "Yes\n";
    else
        cout << "No\n";
    return 0;
}

```

Output:

No

A better solution is to use hashing. We create two empty hash tables (implemented using [unordered_map in C++](#)). We first insert all items of first multimap in first table and all items of second multiset in second table. Now we check if both hash tables contain same items and frequencies or not.

```

// C++ program to check if two given multisets
// are equivalent
#include <bits/stdc++.h>
using namespace std;

bool areSame(vector<int>& a, vector<int>& b)
{
    if (a.size() != b.size())
        return false;

    // Create two unordered maps m1 and m2
    // and insert values of both vectors.
    unordered_map<int, int> m1, m2;
    for (int i = 0; i < a.size(); i++) {
        m1[a[i]]++;
        m2[b[i]]++;
    }
}

```

```
// Now we check if both unordered_maps
// are same or not.
for (auto x : m1) {
    if (m2.find(x.first) == m2.end() || 
        m2[x.first] != x.second)
        return false;
}

return true;
}

// Driver code
int main()
{
    vector<int> a({ 7, 7, 5 }), b({ 7, 7, 5 });
    if (areSame(a, b))
        cout << "Yes\n";
    else
        cout << "No\n";
    return 0;
}
```

Output:

Yes

Time complexity : $O(n)$ under the assumption that `unordered_map` `find()` and `insert()` operations work in $O(1)$ time.

Source

<https://www.geeksforgeeks.org/multiset-equivalence-problem/>

Chapter 223

No of pairs ($a[j] \geq a[i]$) with k numbers in range ($a[i], a[j]$) that are divisible by x

No of pairs ($a[j] \geq a[i]$) with k numbers in range ($a[i], a[j]$) that are divisible by x - GeeksforGeeks

Given an array and two numbers x and k. Find the number of different ordered pairs of indexes (i, j) such that $a[j] \geq a[i]$ and there are exactly k integers num such that num is divisible by x and num is in range $a[i]-a[j]$.

Examples:

```
Input : arr[] = {1, 3, 5, 7}
        x = 2, k = 1
Output : 3
Explanation: The pairs (1, 3), (3, 5) and (5, 7)
have k (which is 1) integers i.e., 2, 4, 6
respectively for every pair in between them.
```

```
Input : arr[] = {5, 3, 1, 7}
        x = 2, k = 0
Output : 4
Explanation: The pairs with indexes (1, 1), (2, 2),
(3, 3), (4, 4) have k = 0 integers that are
divisible by 2 in between them.
```

A **naive approach** is to traverse through all pairs possible and count the number of pairs that have k integers in between them which are divisible by x.

Time complexity: $O(n^2)$

An **efficient approach** is to sort the array and use [binary search](#) to find out the right and left boundaries of numbers(use [lower_bound function](#) inbuilt function to do it) which satisfy the condition and which do not. We have to sort the array as it is given every pair should be $a[j] \geq a[i]$ irrespective of value of i and j. After sorting we traverse through n elements, and find the number with whose multiplication with x gives $a[i]-1$, so that we can find k number by adding k to d = $a[i]-1/x$. So we binary search for the value $(d+k)*x$ to get the multiple with which we can make a pair of $a[i]$ as it will have exactly k integers in between $a[i]$ and $a[j]$. In this way we get the left boundary for $a[j]$ using binary search in $O(\log n)$, and for all other pairs possible with $a[i]$, we need to find out the right-most boundary by searching the the number equal to or greater then $(d+k+1)*x$ where we will get k+1 multiples and we get the no of pairs as (right-left) boundary [index-wise].

```
// cpp program to calculate the number
// pairs satisfying th condition
#include <bits/stdc++.h>
using namespace std;

// function to calculate the number of pairs
int countPairs(int a[], int n, int x, int k)
{
    sort(a, a + n);

    // traverse through all elements
    int ans = 0;
    for (int i = 0; i < n; i++) {

        // current number's divisor
        int d = (a[i] - 1) / x;

        // use binary search to find the element
        // after k multiples of x
        int it1 = lower_bound(a, a + n,
                              max((d + k) * x, a[i])) - a;

        // use binary search to find the element
        // after k+1 multiples of x so that we get
        // the answer by subtracting
        int it2 = lower_bound(a, a + n,
                              max((d + k + 1) * x, a[i])) - a;

        // the difference of index will be the answer
        ans += it2 - it1;
    }
    return ans;
}

// driver code to check the above fucntion
int main()
```

```
{  
    int a[] = { 1, 3, 5, 7 };  
    int n = sizeof(a) / sizeof(a[0]);  
    int x = 2, k = 1;  
  
    // function call to get the number of pairs  
    cout << countPairs(a, n, x, k);  
    return 0;  
}
```

Output:

3

Time complexity: $O(n \log n)$

Source

<https://www.geeksforgeeks.org/no-pairs-aj-ai-k-numbers-range-ai-aj-divisible-x/>

Chapter 224

Noble integers in an array (count of greater elements is equal to value)

Noble integers in an array (count of greater elements is equal to value) - GeeksforGeeks

Given an array arr[], find a Noble integer in it. An integer x is said to be Noble in arr[] if the number of integers greater than x are equal to x. If there are many Noble integers, return any of them. If there is no, then return -1.

Examples :

```
Input  : [7, 3, 16, 10]
Output : 3
Number of integers greater than 3
is three.
```

```
Input  : [-1, -9, -2, -78, 0]
Output : 0
Number of integers greater than 0
is zero.
```

Method 1 (Brute Force)

Iterate through the array. For every element arr[i], find the number of elements greater than arr[i].

CPP

```
// C++ program to find Noble elements
// in an array.
#include <bits/stdc++.h>
```

```
using namespace std;

// Returns a Noble integer if present,
// else returns -1.
int nobleInteger(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        int count = 0;
        for (int j = 0; j < size; j++)
            if (arr[i] < arr[j])
                count++;

        // If count of greater elements
        // is equal to arr[i]
        if (count == arr[i])
            return arr[i];
    }

    return -1;
}

// Driver code
int main()
{
    int arr[] = {10, 3, 20, 40, 2};
    int size = sizeof(arr) / sizeof(arr[0]);
    int res = nobleInteger(arr, size);

    if (res != -1)
        cout<<"The noble integer is "<< res;
    else
        cout<<"No Noble Integer Found";
}

// This code is contributed by Smitha.
```

Java

```
// Java program to find Noble elements
// in an array.
import java.util.ArrayList;

class GFG {

    // Returns a Noble integer if present,
    // else returns -1.
    public static int nobleInteger(int arr[])
}
```

```
{  
    int size = arr.length;  
    for (int i = 0; i < size; i++)  
    {  
        int count = 0;  
        for (int j = 0; j < size; j++)  
            if (arr[i] < arr[j])  
                count++;  
  
        // If count of greater elements  
        // is equal to arr[i]  
        if (count == arr[i])  
            return arr[i];  
    }  
    return -1;  
}  
  
// Driver code  
public static void main(String args[])  
{  
    int [] arr = {10, 3, 20, 40, 2};  
    int res = nobleInteger(arr);  
    if (res != -1)  
        System.out.println("The noble "  
                           + "integer is "+ res);  
    else  
        System.out.println("No Noble "  
                           + "Integer Found");  
}  
}
```

Python3

```
# Python3 program to find Noble  
# elements in an array.  
  
# Returns a Noble integer if  
# present, else returns -1.  
def nobleInteger(arr, size):  
  
    for i in range(0, size):  
  
        count = 0  
        for j in range(0, size):  
            if (arr[i] < arr[j]):  
                count += 1  
        # If count of greater  
        # elements is equal
```

```
# to arr[i]
if (count == arr[i]):
    return arr[i]

return -1

# Driver code
arr = [10, 3, 20, 40, 2]
size = len(arr)
res = nobleInteger(arr,size)
if (res != -1):
    print("The noble integer is ",
          res)
else:
    print("No Noble Integer Found")

# This code is contributed by
# Smitha.
```

C#

```
// C# program to find Noble elements
// in an array.
using System;

class GFG {

    // Returns a Noble integer if present,
    // else returns -1.
    public static int nobleInteger(int [] arr)
    {
        int size = arr.Length;
        for (int i = 0; i < size; i++ )
        {
            int count = 0;
            for (int j = 0; j < size; j++)
                if (arr[i] < arr[j])
                    count++;

            // If count of greater elements
            // is equal to arr[i]
            if (count == arr[i])
                return arr[i];
        }

        return -1;
    }
}
```

```
// Driver code
public static void Main()
{
    int [] arr = {10, 3, 20, 40, 2};
    int res = nobleInteger(arr);
    if (res != -1)
        Console.WriteLine("The noble integer"
                           + " is " + res);
    else
        Console.WriteLine("No Noble Integer"
                           + " Found");
}
}

// This code is contributed by Smitha.
```

PHP

```
<?php
// PHP program to find Noble
// elements in an array.

// Returns a Noble integer
// if present, else returns -1.
function nobleInteger( $arr, $size)
{
    for ( $i = 0; $i < $size; $i++ )
    {
        $count = 0;
        for ( $j = 0; $j < $size; $j++)
            if ($arr[$i] < $arr[$j])
                $count++;

        // If count of greater elements
        // is equal to arr[i]
        if ($count == $arr[$i])
            return $arr[$i];
    }

    return -1;
}

// Driver code
$arr = array(10, 3, 20, 40, 2);
$size = count($arr);
$res = nobleInteger($arr, $size);

if ($res != -1)
```

```
echo "The noble integer is ", $res;
else
    echo "No Noble Integer Found";

// This code is contributed by anuj_67..
?>
```

Output :

```
The noble integer is 3
```

Method 2 (Use Sorting)

1. Sort the Array arr[] in ascending order. This step takes ($O(n \log n)$).
2. Iterate through the array. Compare the value of index i to the number of elements after index i. If arr[i] equals the number of elements after arr[i], it is a noble Integer. Condition to check: ($A[i] == length - i - 1$). This step takes $O(n)$.

Note: Array may have duplicate elements. So, we should skip the elements (adjacent elements in the sorted array) that are same.

Java

```
// Java program to find Noble elements
// in an array.
import java.util.Arrays;

public class Main
{
    // Returns a Noble integer if present,
    // else returns -1.
    public static int nobleInteger(int arr[])
    {
        Arrays.sort(arr);

        // Return a Noble element if present
        // before last.
        int n = arr.length;
        for (int i=0; i<n-1; i++)
        {
            if (arr[i] == arr[i+1])
                continue;

            // In case of duplicates, we
            // reach last occurrence here.
```

```
        if (arr[i] == n-i-1)
            return arr[i];
    }

    if (arr[n-1] == 0)
        return arr[n-1];

    return -1;
}

// Driver code
public static void main(String args[])
{
    int [] arr = {10, 3, 20, 40, 2};
    int res = nobleInteger(arr);
    if (res != -1)
        System.out.println("The noble integer is "+ res);
    else
        System.out.println("No Noble Integer Found");
}
}
```

C#

```
// C# program to find Noble elements
// in an array.
using System;

public class GFG {

    public static int nobleInteger(int[] arr)
    {
        Array.Sort(arr);

        // Return a Noble element if present
        // before last.
        int n = arr.Length;
        for (int i = 0; i < n-1; i++)
        {
            if (arr[i] == arr[i+1])
                continue;

            // In case of duplicates, we
            // reach last occurrence here.
            if (arr[i] == n-i-1)
                return arr[i];
        }
    }
}
```

```
        if (arr[n-1] == 0)
            return arr[n-1];

        return -1;
    }

    // Driver code
    static public void Main ()
    {
        int [] arr = {10, 3, 20, 40, 2};
        int res = nobleInteger(arr);
        if (res != -1)
            Console.WriteLine("The noble integer is "
                + res);
        else
            Console.WriteLine("No Noble Integer "
                + "Found");

    }
}

// This code is contributed by Shrikant13.
```

PHP

```
<?php
// PHP program to find Noble elements

// Returns a Noble integer if present,
// else returns -1.
function nobleInteger( $arr)
{
    sort($arr);

    // Return a Noble element if
    // present before last.
    $n = count($arr);
    for ( $i = 0; $i < $n - 1; $i++)
    {
        if ($arr[$i] == $arr[$i + 1])
            continue;

        // In case of duplicates, we
        // reach last occurrence here.
        if ($arr[$i] == $n - $i - 1)
            return $arr[$i];
    }
}
```

```
if ($arr[$n - 1] == 0)
    return $arr[$n - 1];

return -1;
}

// Driver code
$arr = array(10, 3, 20, 40, 2);
$res = nobleInteger($arr);
if ($res != -1)
    echo "The noble integer is ", $res;
else
    echo "No Noble Integer Found";

// This code is contributed by anuj_67.
?>
```

Output :

The noble integer is 3.

Improved By : [shrikanth13](#), [Smitha Dinesh Semwal](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/noble-integers-in-an-array-count-of-greater-elements-is-equal-to-value/>

Chapter 225

Number of elements greater than K in the range L to R using Fenwick Tree (Offline queries)

Number of elements greater than K in the range L to R using Fenwick Tree (Offline queries)
- GeeksforGeeks

Prerequisites: [Fenwick Tree \(Binary Indexed Tree\)](#)

Given an array of N numbers, and a number of queries where each query will contain three numbers(l, r and k). The task is to calculate the number of array elements which are greater than K in the subarray[L, R].

Examples:

```
Input: n=6
      q=2
      arr[ ] = { 7, 3, 9, 13, 5, 4 }
      Query1: l=1, r=4, k=6
      Query2: l=2, r=6, k=8
```

```
Output: 3
        2
```

For the first query, [7, 3, 9, 13] represents the subarray from index 1 till 4, in which there are 3 numbers which are greater than k=6 that are {7, 9, 13}.

For the second query, there are only two numbers in the query range which are greater than k.

Naive Approach is to find the answer for each query by simply traversing the array from index l till r and keep adding 1 to the count whenever the array element is greater than k.
Time Complexity: $O(n*q)$

A Better Approach is to use Merge Sort Tree. In this approach, build a Segment Tree with a vector at each node containing all the elements of the sub-range in a sorted order. Answer each query using the segment tree where Binary Search can be used to calculate how many numbers are present in each node whose sub-range lies within the query range which are greater than k.

Time complexity: $O(q * \log(n) * \log(n))$

An **Efficient Approach** is to solve the problem using offline queries and [Fenwick Trees](#). Below are the steps:

- First store all the array elements and the queries in the same array. For this, we can create a self-structure or class.
- Then sort the structural array in descending order (in case of collision the query will come first then the array element).
- Process the whole array of structure again, but before that create another BIT array (Binary Indexed Tree) whose $\text{query}(i)$ function will return the count of all the elements which are present in the array till i'th index.
- Initially, fill the whole array with 0.
- Create an answer array, in which the answers of each query are stored.
- Process the array of structure.
- If it is an array element, then update the BIT array with +1 from the index of that element.
- If it is a query, then subtract the $\text{query}(r) - \text{query}(l-1)$ and this will be the answer for that query which will be stored in answer array at the index corresponding to the query number.
- Finally output the answer array.

The key observation here is that since the array of the structure has been sorted in descending order. Whenever we encounter any query only the elements which are greater than 'k' comprises the count in the BIT array which is the answer that is needed.

Below is the explanation of structure used in the program:

Pos: stores the order of query. In case of array elements it is kept as 0.

L: stores the starting index of the query's subarray. In case of array elements it is also 0.

R: stores the ending index of the query's subarray. In case of array element it is used to store the position of element in the array.

Val: store 'k' of the query and all the array elements.

Below is the implementation of the above approach:

```
// C++ program to print the number of elements
// greater than k in a subarray of range L-R.
#include <bits/stdc++.h>
using namespace std;

// Structure which will store both
// array elements and queries.
struct node {
    int pos;
    int l;
    int r;
    int val;
};

// Boolean comparator that will be used
// for sorting the structural array.
bool comp(node a, node b)
{
    // If 2 values are equal the query will
    // occur first then array element
    if (a.val == b.val)
        return a.l > b.l;

    // Otherwise sorted in descending order.
    return a.val > b.val;
}

// Updates the node of BIT array by adding
// 1 to it and its ancestors.
void update(int* BIT, int n, int idx)
{
    while (idx <= n) {
        BIT[idx]++;
        idx += idx & (-idx);
    }
}

// Returns the count of numbers of elements
// present from starting till idx.
int query(int* BIT, int idx)
```

```
{  
    int ans = 0;  
    while (idx) {  
        ans += BIT[idx];  
  
        idx -= idx & (-idx);  
    }  
    return ans;  
}  
  
// Function to solve the queries offline  
void solveQuery(int arr[], int n, int QueryL[],  
                 int QueryR[], int QueryK[], int q)  
{  
    // create node to store the elements  
    // and the queries  
    node a[n + q + 1];  
    // 1-based indexing.  
  
    // traverse for all array numbers  
    for (int i = 1; i <= n; ++i) {  
        a[i].val = arr[i - 1];  
        a[i].pos = 0;  
        a[i].l = 0;  
        a[i].r = i;  
    }  
  
    // iterate for all queries  
    for (int i = n + 1; i <= n + q; ++i) {  
        a[i].pos = i - n;  
        a[i].val = QueryK[i - n - 1];  
        a[i].l = QueryL[i - n - 1];  
        a[i].r = QueryR[i - n - 1];  
    }  
  
    // In-built sort function used to  
    // sort node array using comp function.  
    sort(a + 1, a + n + q + 1, comp);  
  
    // Binary Indexed tree with  
    // initially 0 at all places.  
    int BIT[n + 1];  
  
    // initially 0  
    memset(BIT, 0, sizeof(BIT));  
  
    // For storing answers for each query( 1-based indexing ).  
    int ans[q + 1];
```

```
// traverse for numbers and query
for (int i = 1; i <= n + q; ++i) {
    if (a[i].pos != 0) {

        // call function to returns answer for each query
        int cnt = query(BIT, a[i].r) - query(BIT, a[i].l - 1);

        // This will ensure that answer of each query
        // are stored in order it was initially asked.
        ans[a[i].pos] = cnt;
    }
    else {
        // a[i].r contains the position of the
        // element in the original array.
        update(BIT, n, a[i].r);
    }
}
// Output the answer array
for (int i = 1; i <= q; ++i) {
    cout << ans[i] << endl;
}
}

// Driver Code
int main()
{
    int arr[] = { 7, 3, 9, 13, 5, 4 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // 1-based indexing
    int QueryL[] = { 1, 2 };
    int QueryR[] = { 4, 6 };

    // k for each query
    int QueryK[] = { 6, 8 };

    // number of queries
    int q = sizeof(QueryL) / sizeof(QueryL[0]);

    // Function call to get
    solveQuery(arr, n, QueryL, QueryR, QueryK, q);

    return 0;
}
```

Output:

3

2

Time Complexity: $O(N * \log N)$ where $N = (n+q)$

What is offline query?

In some questions, it is hard to answer queries in any random order. So instead of answering each query separately, store all the queries and then order them accordingly to calculate answer for them efficiently. Store all the answers and then output it in the order it was initially given.

This technique is called *Offline Query*.

Note: Instead of Fenwick Tree, segment tree can also be used where each node of the segment tree will store the number of elements inserted till that iteration. The update and query functions will change, rest of the implementation will remain same.

Necessary Condition For Offline Query: This technique can be used only when the answer of one query does not depend on the answers of previous queries since after sorting the order of queries may change.

Improved By : [dhruvgupta167](#)

Source

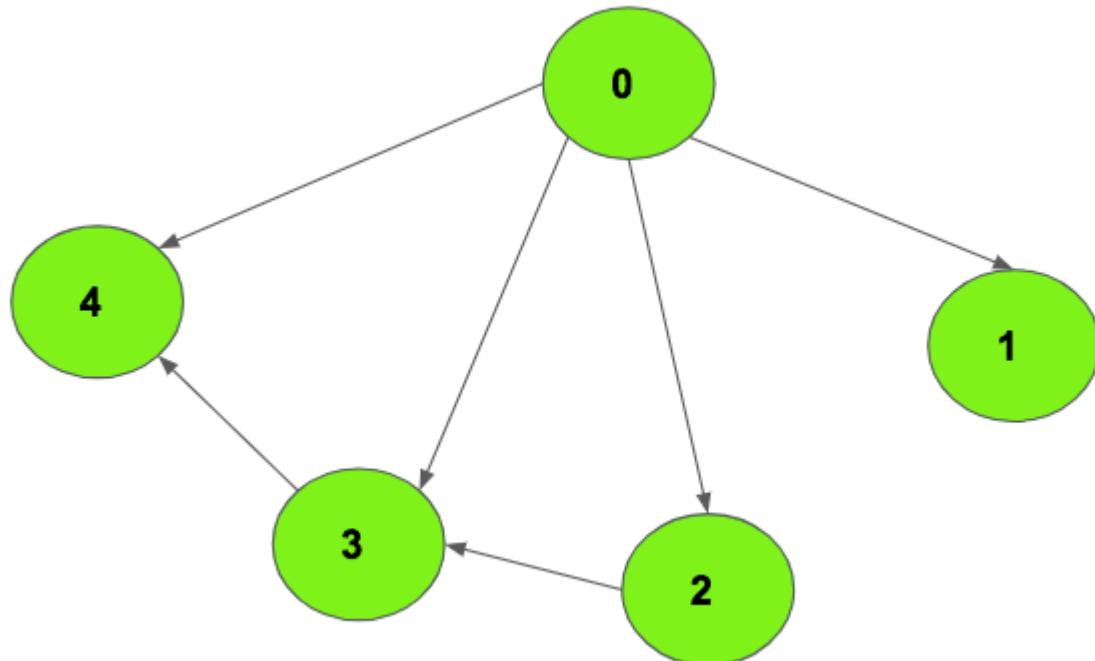
<https://www.geeksforgeeks.org/number-of-elements-greater-than-k-in-the-range-l-to-r-using-fenwick-tree-offline-qu>

Chapter 226

Number of paths from source to destination in a directed acyclic graph

Number of paths from source to destination in a directed acyclic graph - GeeksforGeeks

Given a Directed Acyclic Graph with n vertices and m edges. The task is to find the number of different paths that exist from a source vertex to destination vertex.



Examples:

Input: source = 0, destination = 4

Output: 3

Explanation:

0 -> 2 -> 3 -> 4

0 -> 3 -> 4

0 -> 4

Input: source = 0, destination = 1

Output: 1

Explanation: There exists only one path 0->1

Approach : Let $f(u)$ be the number of ways one can travel from node u to destination vertex. Hence, $f(\text{source})$ is required answer. As $f(\text{destination}) = 1$ here so there is just one path from destination to itself. One can observe, $f(u)$ depends on nothing other than the f values of all the nodes which are possible to travel from u . It makes sense because the number of different paths from u to the destination is the sum of all different paths from **v1, v2, v3...** **v-n** to destination vertex where v_1 to v_n are all the vertices that have a direct path from vertex u . This approach, however, is too slow to be useful. Each function call branches out into further calls, and that branches into further calls, until each and every path is explored once.

The problem with this approach is the calculation of $f(u)$ again and again each time the function is called with argument u . Since this problem exhibits both [overlapping subproblems](#) and [optimal substructure](#), [dynamic programming](#) is applicable here. In order to evaluate $f(u)$ for each u just once, evaluate $f(v)$ for all v that can be visited from u before evaluating $f(u)$. This condition is satisfied by reverse [topological sorted](#) order of the nodes of the graph.

Below is the implementation of the above approach:

```

// C++ program for Number of paths
// from one vertex to another vertex
// in a Directed Acyclic Graph
#include <bits/stdc++.h>
using namespace std;
#define MAXN 1000005

// to make graph
vector<int> v[MAXN];

// function to add edge in graph
void add_edge(int a, int b, int fre[])
{
    // there is path from a to b.
    v[a].push_back(b);
    fre[b]++;
}

// function to make topological sorting
vector<int> topological_sorting(int fre[], int n)

```

```
{  
    queue<int> q;  
  
    // insert all vertices which  
    // don't have any parent.  
    for (int i = 0; i < n; i++)  
        if (!fre[i])  
            q.push(i);  
  
    vector<int> l;  
  
    // using kahn's algorithm  
    // for topological sorting  
    while (!q.empty()) {  
        int u = q.front();  
        q.pop();  
  
        // insert front element of queue to vector  
        l.push_back(u);  
  
        // go through all it's childs  
        for (int i = 0; i < v[u].size(); i++) {  
            fre[v[u][i]]--;  
  
            // whenever the frequency is zero then add  
            // this vertex to queue.  
            if (!fre[v[u][i]])  
                q.push(v[u][i]);  
        }  
    }  
    return l;  
}  
  
// Function that returns the number of paths  
int numberofPaths(int source, int destination, int n, int fre[])  
{  
  
    // make topological sorting  
    vector<int> s = topological_sorting(fre, n);  
  
    // to store required answer.  
    int dp[n] = { 0 };  
  
    // answer from destination  
    // to destination is 1.  
    dp[destination] = 1;  
  
    // traverse in reverse order
```

```
for (int i = s.size() - 1; i >= 0; i--) {
    for (int j = 0; j < v[s[i]].size(); j++) {
        dp[s[i]] += dp[v[s[i]][j]];
    }
}

return dp;
}

// Driver code
int main()
{

    // here vertices are numbered from 0 to n-1.
    int n = 5;
    int source = 0, destination = 4;

    // to count number of vertex which don't
    // have any parents.
    int fre[n] = { 0 };

    // to add all edges of graph
    add_edge(0, 1, fre);
    add_edge(0, 2, fre);
    add_edge(0, 3, fre);
    add_edge(0, 4, fre);
    add_edge(2, 3, fre);
    add_edge(3, 4, fre);

    // Function that returns the number of paths
    cout << numberofPaths(source, destination, n, fre);
}
```

Output:

3

Source

<https://www.geeksforgeeks.org/number-of-paths-from-source-to-destination-in-a-directed-acyclic-graph/>

Chapter 227

Number of sextuplets (or six values) that satisfy an equation

Number of sextuplets (or six values) that satisfy an equation - GeeksforGeeks

Given an array of **n** elements. The task is to find number of sextuplets that satisfy the below equation such that a, b, c, d, e and f belong to the given array:

$$\frac{a * b + c - e}{d} = f$$

Examples:

```
Input : arr[] = { 1 }.
Output : 1
a = b = c = d = e = f = 1 satisfy
the equation.
```

```
Input : arr[] = { 2, 3 }
Output : 4
```

```
Input : arr[] = { 1, -1 }
Output : 24
```

First, reorder the equation, $a * b + c = (f + e) * d$.

Now, make two arrays, one for LHS (Left Hand Side) of the equation and one for the RHS (Right Hand Side) of the equation. Search each element of RHS's array in the LHS's array. Whenever you find a value of RHS in LHS, check how many times it is repeated in LHS and add that count to the total. Searching can be done using binary search, by sorting the LHS array.

Below is the C++ implementation of this approach:

```
// C++ program to count of 6 values from an array
// that satisfy an equation with 6 variables
#include<bits/stdc++.h>
using namespace std;

// Returns count of 6 values from arr[]
// that satisfy an equation with 6 variables
int findSextuplets(int arr[], int n)
{
    // Generating possible values of RHS of the equation
    int index = 0;
    int RHS[n*n*n + 1];
    for (int i = 0; i < n; i++)
        if (arr[i]) // Checking d should be non-zero.
            for (int j = 0; j < n; j++)
                for (int k = 0; k < n; k++)
                    RHS[index++] = arr[i] * (arr[j] + arr[k]);

    // Sort RHS[] so that we can do binary search in it.
    sort(RHS, RHS + n);

    // Generating all possible values of LHS of the equation
    // and finding the number of occurrences of the value in RHS.
    int result = 0;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            for (int k = 0; k < n; k++)
            {
                int val = arr[i] * arr[j] + arr[k];
                result += (upper_bound(RHS, RHS + index, val) -
                           lower_bound(RHS, RHS + index, val));
            }
        }
    }

    return result;
}

// Driven Program
int main()
{
    int arr[] = {2, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
```

```
cout << findSextuplets(arr, n) << endl;
return 0;
}
```

Output:

4

Time Complexity : $O(N^3 \log N)$

Source

<https://www.geeksforgeeks.org/number-sextuplets-six-values-satisfy-equation/>

Chapter 228

Number of swaps to sort when only adjacent swapping allowed

Number of swaps to sort when only adjacent swapping allowed - GeeksforGeeks

Given an array arr[] of non negative integers. We can perform a swap operation on any two adjacent elements in the array. Find the minimum number of swaps needed to sort the array in ascending order.

Examples :

```
Input : arr[] = {3, 2, 1}
Output : 3
We need to do following swaps
(3, 2), (3, 1) and (1, 2)
```

```
Input : arr[] = {1, 20, 6, 4, 5}
Output : 5
```

)

There is an interesting solution to this problem. It can be solved using the fact that number of swaps needed is equal to number of [inversions](#). So we basically need to [count inversions in array](#).

The fact can be established using below observations:

- 1) A sorted array has no inversions.
- 2) An adjacent swap can reduce one inversion. Doing x adjacent swaps can reduce x inversions in an array.

C++

```
// C++ program to count number of swaps required
```

```
// to sort an array when only swapping of adjacent
// elements is allowed.
#include <bits/stdc++.h>

/* This function merges two sorted arrays and returns inversion
   count in the arrays.*/
int merge(int arr[], int temp[], int left, int mid, int right)
{
    int inv_count = 0;

    int i = left; /* i is index for left subarray*/
    int j = mid; /* i is index for right subarray*/
    int k = left; /* i is index for resultant merged subarray*/
    while ((i <= mid - 1) && (j <= right))
    {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
        {
            temp[k++] = arr[j++];

            /* this is tricky -- see above explanation/
               diagram for merge()*/
            inv_count = inv_count + (mid - i);
        }
    }

    /* Copy the remaining elements of left subarray
       (if there are any) to temp*/
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    /* Copy the remaining elements of right subarray
       (if there are any) to temp*/
    while (j <= right)
        temp[k++] = arr[j++];

    /*Copy back the merged elements to original array*/
    for (i=left; i <= right; i++)
        arr[i] = temp[i];

    return inv_count;
}

/* An auxiliary recursive function that sorts the input
   array and returns the number of inversions in the
   array. */
int _mergeSort(int arr[], int temp[], int left, int right)
```

```
{  
    int mid, inv_count = 0;  
    if (right > left)  
    {  
        /* Divide the array into two parts and call  
           _mergeSortAndCountInv() for each of the parts */  
        mid = (right + left)/2;  
  
        /* Inversion count will be sum of inversions in  
           left-part, right-part and number of inversions  
           in merging */  
        inv_count = _mergeSort(arr, temp, left, mid);  
        inv_count += _mergeSort(arr, temp, mid+1, right);  
  
        /*Merge the two parts*/  
        inv_count += merge(arr, temp, left, mid+1, right);  
    }  
  
    return inv_count;  
}  
  
/* This function sorts the input array and returns the  
   number of inversions in the array */  
int countSwaps(int arr[], int n)  
{  
    int temp[n];  
    return _mergeSort(arr, temp, 0, n - 1);  
}  
  
/* Driver program to test above functions */  
int main(int argc, char** args)  
{  
    int arr[] = {1, 20, 6, 4, 5};  
    int n = sizeof(arr)/sizeof(arr[0]);  
    printf("Number of swaps is %d \n", countSwaps(arr, n));  
    return 0;  
}
```

Java

```
// java program to count number of  
// swaps required to sort an array  
// when only swapping of adjacent  
// elements is allowed.  
import java.io.*;  
  
class GFG {
```

```
// This function merges two sorted
// arrays and returns inversion
// count in the arrays.
static int merge(int arr[], int temp[],
                 int left, int mid, int right)
{
    int inv_count = 0;

    /* i is index for left subarray*/
    int i = left;

    /* i is index for right subarray*/
    int j = mid;

    /* i is index for resultant merged subarray*/
    int k = left;

    while ((i <= mid - 1) && (j <= right))
    {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
        {
            temp[k++] = arr[j++];

            /* this is tricky -- see above /
             explanation diagram for merge()*/
            inv_count = inv_count + (mid - i);
        }
    }

    /* Copy the remaining elements of left
     subarray (if there are any) to temp*/
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    /* Copy the remaining elements of right
     subarray (if there are any) to temp*/
    while (j <= right)
        temp[k++] = arr[j++];

    /*Copy back the merged elements
     to original array*/
    for (i=left; i <= right; i++)
        arr[i] = temp[i];

    return inv_count;
}
```

```
// An auxiliary recursive function that
// sorts the input array and returns
// the number of inversions in the array.
static int _mergeSort(int arr[], int temp[],
                      int left, int right)
{
    int mid, inv_count = 0;
    if (right > left)
    {
        // Divide the array into two parts and
        // call _mergeSortAndCountInv() for
        // each of the parts
        mid = (right + left)/2;

        /* Inversion count will be sum of
        inversions in left-part, right-part
        and number of inversions in merging */
        inv_count = _mergeSort(arr, temp,
                              left, mid);

        inv_count += _mergeSort(arr, temp,
                               mid+1, right);

        /*Merge the two parts*/
        inv_count += merge(arr, temp,
                           left, mid+1, right);
    }

    return inv_count;
}

// This function sorts the input
// array and returns the number
// of inversions in the array
static int countSwaps(int arr[], int n)
{
    int temp[] = new int[n];
    return _mergeSort(arr, temp, 0, n - 1);
}

// Driver Code
public static void main (String[] args)
{

    int arr[] = {1, 20, 6, 4, 5};
    int n = arr.length;
    System.out.println("Number of swaps is "
```

```
        + countSwaps(arr, n));
    }
}
```

```
// This code is contributed by vt_m
```

C#

```
// C# program to count number of
// swaps required to sort an array
// when only swapping of adjacent
// elements is allowed.
using System;

class GFG
{

    // This function merges two
    // sorted arrays and returns
    // inversion count in the arrays.
    static int merge(int []arr, int []temp,
                    int left, int mid,
                    int right)
    {
        int inv_count = 0;

        /* i is index for
        left subarray*/
        int i = left;

        /* i is index for
        right subarray*/
        int j = mid;

        /* i is index for resultant
        merged subarray*/
        int k = left;

        while ((i <= mid - 1) &&
               (j <= right))
        {
            if (arr[i] <= arr[j])
                temp[k++] = arr[i++];
            else
            {
                temp[k++] = arr[j++];

                /* this is tricky -- see above /

```

```
explanation diagram for merge()*/
inv_count = inv_count + (mid - i);
}

/*
Copy the remaining elements
of left subarray (if there are
any) to temp*/
while (i <= mid - 1)
    temp[k++] = arr[i++];

/*
Copy the remaining elements
of right subarray (if there are
any) to temp*/
while (j <= right)
    temp[k++] = arr[j++];

/*Copy back the merged
elements to original array*/
for (i=left; i <= right; i++)
    arr[i] = temp[i];

return inv_count;
}

// An auxiliary recursive function
// that sorts the input array and
// returns the number of inversions
// in the array.
static int _mergeSort(int []arr, int []temp,
                      int left, int right)
{
    int mid, inv_count = 0;
    if (right > left)
    {
        // Divide the array into two parts
        // and call _mergeSortAndCountInv()
        // for each of the parts
        mid = (right + left) / 2;

        /* Inversion count will be sum of
        inversions in left-part, right-part
        and number of inversions in merging */
        inv_count = _mergeSort(arr, temp,
                              left, mid);

        inv_count += _mergeSort(arr, temp,
                               mid + 1, right);
    }
}
```

```
/*Merge the two parts*/
inv_count += merge(arr, temp,
                    left, mid + 1, right);
}

return inv_count;
}

// This function sorts the input
// array and returns the number
// of inversions in the array
static int countSwaps(int []arr, int n)
{
    int []temp = new int[n];
    return _mergeSort(arr, temp, 0, n - 1);
}

// Driver Code
public static void Main ()
{

    int []arr = {1, 20, 6, 4, 5};
    int n = arr.Length;
    Console.Write("Number of swaps is " +
                  countSwaps(arr, n));
}
}

// This code is contributed by nitin mittal.
```

Output :

Number of swaps is 5

Time Complexity : $O(n \log n)$

Related Post :

[Minimum number of swaps required to sort an array](#)

References :

<http://stackoverflow.com/questions/20990127/sorting-a-sequence-by-swapping-adjacent-elements-using-minimum->

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/number-swaps-sort-adjacent-swapping-allowed/>

Chapter 229

Number of visible boxes after putting one inside another

Number of visible boxes after putting one inside another - GeeksforGeeks

Given N boxes and their size in an array. You are allowed to keep a box inside another box only if the box in which it is held is empty and the size of the box is at least twice as large as the size of the box. The task is to find minimum number of visible boxes.

Examples –

```
Input : arr[] = { 1, 3, 4, 5 }
Output : 3
Put box of size 1 in box of size 3.
```

```
Input : arr[] = { 4, 2, 1, 8 }
Output : 1
Put box of size 1 in box of size 2
and box of size 2 in box of size 4.
And put box of size 4 in box of size 8.
```

The idea is to sort the array. Now, make a queue and insert first element of sorted array. Now traverse the array from first element and insert each element in the queue, also check if front element of queue is less than or equal to half of current traversed element. So, the number of visible box will be number of element in queue after traversing the sorted array. Basically, we are trying to put a box of size in smallest box which is greater than or equal to 2^x .

For example, if $\text{arr}[] = \{ 2, 3, 4, 6 \}$, then we trying to put box of size 2 in box of size 4 instead of box of size 6 because if we put box of size 2 in box of size 6 then box of size 3 cannot be kept in any other box and we need to minimize the number of visible box.

C++

```
// CPP program to count number of visible boxes.
#include <bits/stdc++.h>
using namespace std;

// return the minimum number of visible boxes
int minimumBox(int arr[], int n)
{
    queue<int> q;

    // sorting the array
    sort(arr, arr + n);

    q.push(arr[0]);

    // traversing the array
    for (int i = 1; i < n; i++)  {

        int now = q.front();

        // checking if current element
        // is greater than or equal to
        // twice of front element
        if (arr[i] >= 2 * now)
            q.pop();

        // Pushing each element of array
        q.push(arr[i]);
    }

    return q.size();
}

// driver Program
int main()
{
    int arr[] = { 4, 1, 2, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << minimumBox(arr, n) << endl;
    return 0;
}
```

Java

```
// Java program to count number of visible
// boxes.

import java.util.LinkedList;
import java.util.Queue;
```

```
import java.util.Arrays;

public class GFG {

    // return the minimum number of visible
    // boxes
    static int minimumBox(int []arr, int n)
    {

        // New Queue of integers.
        Queue<Integer> q = new LinkedList<>();

        // sorting the array
        Arrays.sort(arr);

        q.add(arr[0]);

        // traversing the array
        for (int i = 1; i < n; i++)
        {
            int now = q.element();

            // checking if current element
            // is greater than or equal to
            // twice of front element
            if (arr[i] >= 2 * now)
                q.remove();

            // Pushing each element of array
            q.add(arr[i]);
        }

        return q.size();
    }

    // Driver code
    public static void main(String args[])
    {
        int [] arr = { 4, 1, 2, 8 };
        int n = arr.length;

        System.out.println(minimumBox(arr, n));
    }
}

// This code is contributed by Sam007.
```

C#

```
// C# program to count number of visible
// boxes.
using System;
using System.Collections.Generic;

class GFG {

    // return the minimum number of visible
    // boxes
    static int minimumBox(int []arr, int n)
    {

        // New Queue of integers.
        Queue<int> q = new Queue<int>();

        // sorting the array
        Array.Sort(arr);

        q.Enqueue(arr[0]);

        // traversing the array
        for (int i = 1; i < n; i++)
        {
            int now = q.Peek();

            // checking if current element
            // is greater than or equal to
            // twice of front element
            if (arr[i] >= 2 * now)
                q.Dequeue();

            // Pushing each element of array
            q.Enqueue(arr[i]);
        }

        return q.Count;
    }

    // Driver code
    public static void Main()
    {
        int [] arr = { 4, 1, 2, 8 };
        int n = arr.Length;

        Console.WriteLine(minimumBox(arr, n));
    }
}
```

// This code is contributed by Sam007.

Output –

1

Time Complexity: O(nlogn)

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/number-visible-boxes-putting-one-inside-another/>

Chapter 230

Odd-Even Sort / Brick Sort

Odd-Even Sort / Brick Sort - GeeksforGeeks

This is basically a variation of [bubble-sort](#). This algorithm is divided into two phases- Odd and Even Phase. The algorithm runs until the array elements are sorted and in each iteration two phases occurs- Odd and Even Phases.

In the odd phase, we perform a bubble sort on odd indexed elements and in the even phase, we perform a bubble sort on even indexed elements.

C++

```
// A C++ Program to implement Odd-Even / Brick Sort
#include<bits/stdc++.h>
using namespace std;

// A function to sort the algorithm using Odd Even sort
void oddEvenSort(int arr[], int n)
{
    bool isSorted = false; // Initially array is unsorted

    while (!isSorted)
    {
        isSorted = true;

        // Perform Bubble sort on odd indexed element
        for (int i=1; i<=n-2; i=i+2)
        {
            if (arr[i] > arr[i+1])
            {
                swap(arr[i], arr[i+1]);
                isSorted = false;
            }
        }
    }
}
```

```
// Perform Bubble sort on even indexed element
for (int i=0; i<=n-2; i=i+2)
{
    if (arr[i] > arr[i+1])
    {
        swap(arr[i], arr[i+1]);
        isSorted = false;
    }
}
}

// A utility function ot print an array of size n
void printArray(int arr[], int n)
{
    for (int i=0; i < n; i++)
        cout << arr[i] << " ";
    cout << "\n";
}

// Driver program to test above functions.
int main()
{
    int arr[] = {34, 2, 10, -9};
    int n = sizeof(arr)/sizeof(arr[0]);

    oddEvenSort(arr, n);
    printArray(arr, n);

    return (0);
}
```

Java

```
// Java Program to implement
// Odd-Even / Brick Sort
import java.io.*;

class GFG
{
    public static void oddEvenSort(int arr[], int n)
    {
        boolean isSorted = false; // Initially array is unsorted

        while (!isSorted)
```

```

{
    isSorted = true;
    int temp =0;

    // Perform Bubble sort on odd indexed element
    for (int i=1; i<=n-2; i=i+2)
    {
        if (arr[i] > arr[i+1])
        {
            temp = arr[i];
            arr[i] = arr[i+1];
            arr[i+1] = temp;
            isSorted = false;
        }
    }

    // Perform Bubble sort on even indexed element
    for (int i=0; i<=n-2; i=i+2)
    {
        if (arr[i] > arr[i+1])
        {
            temp = arr[i];
            arr[i] = arr[i+1];
            arr[i+1] = temp;
            isSorted = false;
        }
    }

    return;
}
public static void main (String[] args)
{
    int arr[] = {34, 2, 10, -9};
    int n = arr.length;

    oddEvenSort(arr, n);
    for (int i=0; i < n; i++)
        System.out.print(arr[i] + " ");

    System.out.println(" ");
}
// Code Contribute by Mohit Gupta_OMG <(0_o)>

```

Python3

```
# Python Program to implement
```

```
# Odd-Even / Brick Sort

def oddEvenSort(arr, n):
    # Initially array is unsorted
    isSorted = 0
    while isSorted == 0:
        isSorted = 1
        temp = 0
        for i in range(1, n-1, 2):
            if arr[i] > arr[i+1]:
                arr[i], arr[i+1] = arr[i+1], arr[i]
                isSorted = 0

        for i in range(0, n-1, 2):
            if arr[i] > arr[i+1]:
                arr[i], arr[i+1] = arr[i+1], arr[i]
                isSorted = 0

    return

arr = [34, 2, 10, -9]
n = len(arr)

oddEvenSort(arr, n);
for i in range(0, n):
    print(arr[i], end = ' ')
```

Code Contribute by Mohit Gupta_OMG <(0_o)>

C#

```
// C# Program to implement
// Odd-Even / Brick Sort
using System;

class GFG
{
    public static void oddEvenSort(int []arr, int n)
    {
        // Initially array is unsorted
        bool isSorted = false;

        while (!isSorted)
        {
            isSorted = true;
            int temp = 0;
```

```
// Perform Bubble sort on
// odd indexed element
for (int i = 1; i <= n - 2; i = i + 2)
{
    if (arr[i] > arr[i+1])
    {
        temp = arr[i];
        arr[i] = arr[i+1];
        arr[i+1] = temp;
        isSorted = false;
    }
}

// Perform Bubble sort on
// even indexed element
for (int i = 0; i <= n - 2; i = i + 2)
{
    if (arr[i] > arr[i+1])
    {
        temp = arr[i];
        arr[i] = arr[i+1];
        arr[i+1] = temp;
        isSorted = false;
    }
}
return;
}

// Driver code
public static void Main ()
{
    int []arr = {34, 2, 10, -9};
    int n = arr.Length;

    // Function calling
    oddEvenSort(arr, n);
    for (int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");

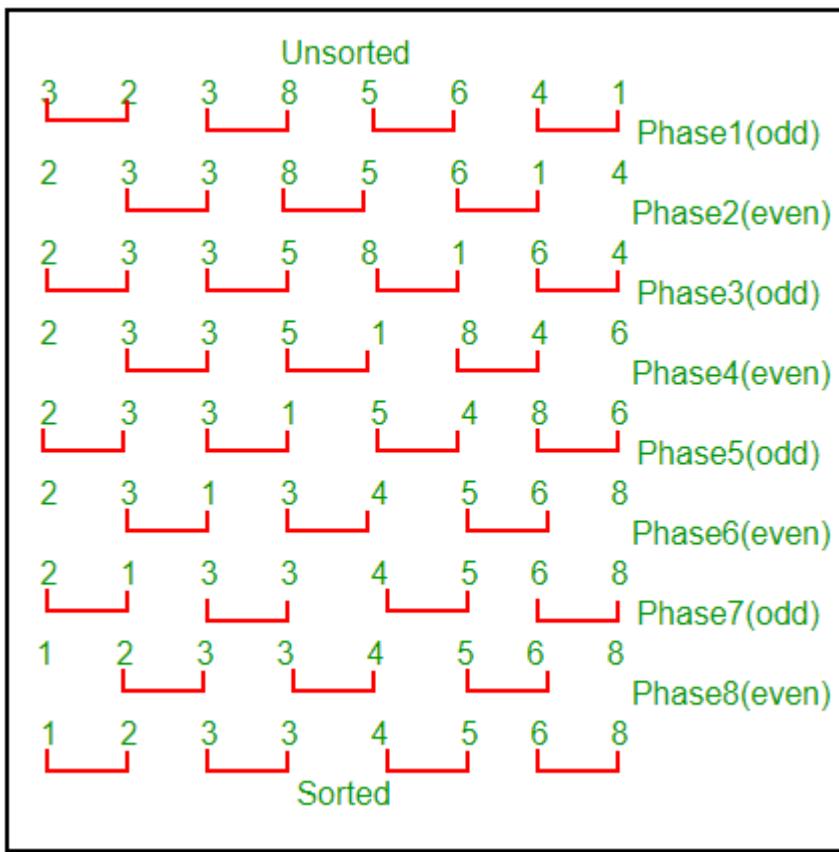
    Console.WriteLine(" ");
}
}

// This code is contributed by Sam007
```

Output :

-9 2 10 34

We demonstrate the above algorithm using the below illustration on the array = {3, 2, 3, 8, 5, 6, 4, 1}



Please refer [wiki](#) for proof of correctness.

Time Complexity : $O(N^2)$ where, N = Number of elements in the input array.

Auxiliary Space : $O(1)$. Just like bubble sort this is also an in-place algorithm.

Exercise

In our program in each iteration we first do bubble sort on odd indexed elements and then a bubble sort on the even indexed elements.

Will we get a sorted result if we first perform a bubble sort on even indexed element first and then on the odd indexed element ?

References

https://en.wikipedia.org/wiki/Odd%20even_sort

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/odd-even-sort-brick-sort/>

Chapter 231

PHP Sort array of strings in natural and standard orders

PHP Sort array of strings in natural and standard orders - GeeksforGeeks

You are given an array of strings. You have to sort the given array in standard way (case of alphabets matters) as well as natural way (alphabet case does not matter).

```
Input : arr[] = {"Geeks", "for", "geeks"}  
Output : Standard sorting: Geeks for geeks  
          Natural sorting: for Geeks geeks  
  
Input : arr[] = {"Code", "at", "geeks", "Practice"}  
Output : Standard sorting: Code Practice at geeks  
          Natural sorting: at Code geeks Practice
```

If you are trying to sort the array of string in a simple manner you can simple create a comparison function for character comparison and sort the given array of strings. But that will differentiate lower case and upper case alphabets. To solve this problem if you are opting to solve this in c/java you have to write your own comparision function which specially take care of cases of alphabets. But if we will opt [PHP](#) as our language then we can sort it directly with the help of `natcasesort()`.

`natcasesort()` : It sort strings regardless of their case. Means ‘a’ & ‘A’ are treated smaller than ‘b’ & ‘B’ in this sorting method.

```
// declare array  
$arr = array ("Hello", "to", "geeks", "for", "GEEks");  
  
// Standard sort  
$standard_result = sort($arr);
```

```
print_r($standart_result);

// natural sort
$natural_result = natcasesort($arr);
print_r($natural_result);

<?php
// PHP program to sort an array
// in standard and natural ways.

// function to print array
function printArray ($arr)
{
    foreach ($arr as $value) {
        echo "$value ";
    }
}

// declare array
$arr = array ("Hello", "to", "geeks", "for", "GEEks");

// Standard sort
$standard_result = $arr;
sort($standard_result);
echo "Array after Standard sorting: ";
printArray($standard_result);

// natural sort
$natural_result = $arr;
natcasesort($natural_result);
echo "\nArray after Natural sorting: ";
printArray($natural_result);
?>
```

Output:

```
Array after Standard sorting: GEEks Hello for geeks to
Array after Natural sorting: for geeks GEEks Hello to
```

Source

<https://www.geeksforgeeks.org/php-sort-array-strings-natural-standard-orders/>

Chapter 232

Pair formation such that maximum pair sum is minimized

Pair formation such that maximum pair sum is minimized - GeeksforGeeks

Given an array of size $2 * N$ integers. Divide the array into N pairs, such that the maximum pair sum is minimized. In other words, the optimal division of array into N pairs should result into a maximum pair sum which is minimum of other maximum pair sum of all possibilities.

Examples:

Input : $N = 2$
arr[] = { 5, 8, 3, 9 }
Output : (3, 9) (5, 8)

Explanation:

Possible pairs are :

1. (8, 9) (3, 5) Maximum Sum of a Pair = 17
2. (5, 9) (3, 8) Maximum Sum of a Pair = 14
3. (3, 9) (5, 8) Maximum Sum of a Pair = 13

Thus, in case 3, the maximum pair sum is minimum of all the other cases. Hence, the answer is (3, 9) (5, 8).

Input : $N = 2$
arr[] = { 9, 6, 5, 1 }
Output : (1, 9) (5, 6)

Approach: The idea is to first sort the given array and then iterate over the loop to form pairs (i, j) where i would start from 0 and j would start from end of array correspondingly. Increment i and Decrement j to form the next pair and so on.

Below is the implementation of above approach.

C++

```
// CPP Program to divide the array into
// N pairs such that maximum pair is minimized
#include <bits/stdc++.h>

using namespace std;

void findOptimalPairs(int arr[], int N)
{
    sort(arr, arr + N);

    // After Sorting Maintain two variables i and j
    // pointing to start and end of array Such that
    // smallest element of array pairs with largest
    // element
    for (int i = 0, j = N - 1; i <= j; i++, j--)
        cout << "(" << arr[i] << ", " << arr[j] << ")" << " ";
}

// Driver Code
int main()
{
    int arr[] = { 9, 6, 5, 1 };
    int N = sizeof(arr) / sizeof(arr[0]);

    findOptimalPairs(arr, N);
    return 0;
}
```

Java

```
// Java Program to divide the array into
// N pairs such that maximum pair is minimized
import java.io.*;
import java.util.Arrays;

class GFG {

    static void findOptimalPairs(int arr[], int N)
    {
        Arrays.sort(arr);

        // After Sorting Maintain two variables i and j
        // pointing to start and end of array Such that
        // smallest element of array pairs with largest
        // element
        for (int i = 0, j = N - 1; i <= j; i++, j--)
            System.out.print( "(" + arr[i] + ", " + arr[j] + ")" + " ");
    }
}
```

```
// Driver Code
public static void main (String[] args)
{
    int arr[] = {9, 6, 5, 1};
    int N = arr.length;

    findOptimalPairs(arr, N);
}
}

// This code is contributed by anuj_67.
```

Output:

(1, 9) (5, 6)

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/pair-formation-maximum-pair-sum-minimized/>

Chapter 233

Pairs such that one is a power multiple of other

Pairs such that one is a power multiple of other - GeeksforGeeks

You are given an array A[] of n-elements and a positive integer k. Now you have find the number of pairs Ai, Aj such that $A_i = A_j * (k^x)$ where x is an integer.

Note: (Ai, Aj) and (Aj, Ai) must be count once.

Examples :

Input : A[] = {3, 6, 4, 2}, k = 2

Output : 2

Explanation : We have only two pairs
(4, 2) and (3, 6)

Input : A[] = {2, 2, 2}, k = 2

Output : 3

Explanation : (2, 2), (2, 2), (2, 2)
that are (A1, A2), (A2, A3) and (A1, A3) are
total three pairs where $A_i = A_j * (k^0)$

To solve this problem, we first sort the given array and then for each element Ai, we find number of elements equal to value $A_i * k^x$ for different value of x till $A_i * k^x$ is less than or equal to largest of Ai.

Algorithm:

```
// sort the given array
sort(A, A+n);

// for each A[i] traverse rest array
```

```
for (int i=0; i<n; i++)
{
    for (int j=i+1; j<n; j++)
    {
        // count Aj such that Ai*k^x = Aj
        int x = 0;

        // increase x till Ai * k^x <=
        // largest element
        while ((A[i]*pow(k, x)) <= A[j])
        {
            if ((A[i]*pow(k, x)) == A[j])
            {
                ans++;
                break;
            }
            x++;
        }
    }
}
// return answer
return ans;
```

C++

```
// Program to find pairs count
#include <bits/stdc++.h>
using namespace std;

// function to count the required pairs
int countPairs(int A[], int n, int k) {
    int ans = 0;
    // sort the given array
    sort(A, A + n);

    // for each A[i] traverse rest array
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {

            // count Aj such that Ai*k^x = Aj
            int x = 0;

            // increase x till Ai * k^x <= largest element
            while ((A[i] * pow(k, x)) <= A[j]) {
                if ((A[i] * pow(k, x)) == A[j]) {
                    ans++;
                    break;
                }
            }
        }
    }
}
```

```
        x++;
    }
}
return ans;
}

// driver program
int main() {
    int A[] = {3, 8, 9, 12, 18, 4, 24, 2, 6};
    int n = sizeof(A) / sizeof(A[0]);
    int k = 3;
    cout << countPairs(A, n, k);
    return 0;
}
```

Java

```
// Java program to find pairs count
import java.io.*;
import java.util.*;

class GFG {

    // function to count the required pairs
    static int countPairs(int A[], int n, int k)
    {
        int ans = 0;

        // sort the given array
        Arrays.sort(A);

        // for each A[i] traverse rest array
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++)
            {

                // count Aj such that Ai*k^x = Aj
                int x = 0;

                // increase x till Ai * k^x <= largest element
                while ((A[i] * Math.pow(k, x)) <= A[j])
                {
                    if ((A[i] * Math.pow(k, x)) == A[j])
                    {
                        ans++;
                        break;
                    }
                }
            }
        }
    }
}
```

```
        x++;
    }
}
return ans;
}

// Driver program
public static void main (String[] args)
{
    int A[] = {3, 8, 9, 12, 18, 4, 24, 2, 6};
    int n = A.length;
    int k = 3;
    System.out.println (countPairs(A, n, k));

}
}

// This code is contributed by vt_m.
```

Python3

```
# Program to find pairs count
import math

# function to count the required pairs
def countPairs(A, n, k):
    ans = 0

    # sort the given array
    A.sort()

    # for each A[i] traverse rest array
    for i in range(0,n):

        for j in range(i + 1, n):

            # count Aj such that Ai*k^x = Aj
            x = 0

            # increase x till Ai * k^x <= largest element
            while ((A[i] * math.pow(k, x)) <= A[j]) :
                if ((A[i] * math.pow(k, x)) == A[j]) :
                    ans+=1
                    break
                x+=1
    return ans
```

```
# driver program
A = [3, 8, 9, 12, 18, 4, 24, 2, 6]
n = len(A)
k = 3
```

```
print(countPairs(A, n, k))
```

```
# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
// C# program to find pairs count
using System;

class GFG {

    // function to count the required pairs
    static int countPairs(int []A, int n, int k)
    {
        int ans = 0;

        // sort the given array
        Array.Sort(A);

        // for each A[i] traverse rest array
        for (int i = 0; i < n; i++)
        {
            for (int j = i + 1; j < n; j++)
            {

                // count Aj such that Ai*k^x = Aj
                int x = 0;

                // increase x till Ai * k^x <= largest element
                while ((A[i] * Math.Pow(k, x)) <= A[j])
                {
                    if ((A[i] * Math.Pow(k, x)) == A[j])
                    {
                        ans++;
                        break;
                    }
                    x++;
                }
            }
        }
        return ans;
    }
}
```

```
}

// Driver program
public static void Main ()
{
    int []A = {3, 8, 9, 12, 18, 4, 24, 2, 6};
    int n = A.Length;
    int k = 3;
    Console.WriteLine(countPairs(A, n, k));

}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP Program to find pairs count

// function to count
// the required pairs
function countPairs($A, $n, $k)
{
$ans = 0;

// sort the given array
sort($A);

// for each A[i]
// traverse rest array
for ($i = 0; $i < $n; $i++)
{
    for ($j = $i + 1; $j < $n; $j++)
    {

        // count Aj such that Ai*k^x = Aj
        $x = 0;

        // increase x till Ai *
        // k^x <= largest element
        while ((($A[$i] * pow($k, $x)) <= $A[$j])
        {
            if (($A[$i] * pow($k, $x)) == $A[$j])
            {
                $ans++;
                break;
            }
        }
    }
}
```

```
        $x++;
    }
}
}
return $ans;
}

// Driver Code

$A = array(3, 8, 9, 12, 18,
           4, 24, 2, 6);
$n = count($A);
$k = 3;
echo countPairs($A, $n, $k);

// This code is contributed by anuj_67.
?>
```

Output :

6

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/pairs-one-power-multiple/>

Chapter 234

Pairs with Difference less than K

Pairs with Difference less than K - GeeksforGeeks

Given an array of n integers, We need to find all pairs with difference less than k

Examples :

```
Input : a[] = {1, 10, 4, 2}
        K = 3
Output : 2
We can make only two pairs
with difference less than 3.
(1, 2) and (4, 2)
```

```
Input : a[] = {1, 8, 7}
        K = 7
Output : 2
Pairs with difference less than 7
are (1, 7) and (8, 7)
```

Method 1 (Simple) : Run two nested loops. The outer loop picks every element x one by one. The inner loop considers all elements after x and checks if difference is within limits or not.

C++

```
// CPP code to find count of Pairs with
// difference less than K.
#include <bits/stdc++.h>
using namespace std;
```

```
int countPairs(int a[], int n, int k)
{
    int res = 0;
    for (int i = 0; i < n; i++)
        for (int j=i+1; j<n; j++)
            if (abs(a[j] - a[i]) < k)
                res++;

    return res;
}

// Driver code
int main()
{
    int a[] = {1, 10, 4, 2};
    int k = 3;
    int n = sizeof(a) / sizeof(a[0]);
    cout << countPairs(a, n, k) << endl;
    return 0;
}
```

Java

```
// java code to find count of Pairs with
// difference less than K.
import java.io.*;

class GFG {
    static int countPairs(int a[], int n, int k)
    {
        int res = 0;
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                if (Math.abs(a[j] - a[i]) < k)
                    res++;

        return res;
    }

    // Driver code
    public static void main (String[] args)
    {
        int a[] = {1, 10, 4, 2};
        int k = 3;
        int n = a.length;
        System.out.println(countPairs(a, n, k));
    }
}
```

```
}
```

```
// This code is contributed by vt_m.
```

Python3

```
# Python3 code to find count of Pairs
# with difference less than K.

def countPairs(a, n, k):
    res = 0
    for i in range(n):
        for j in range(i + 1, n):
            if (abs(a[j] - a[i]) < k):
                res += 1

    return res

# Driver code
a = [1, 10, 4, 2]
k = 3
n = len(a)
print(countPairs(a, n, k), end = "")
```

```
# This code is contributed by Azkia Anam.
```

C#

```
// C# code to find count of Pairs
// with difference less than K.
using System;

class GFG {

    // Function to count pairs
    static int countPairs(int []a, int n,
                          int k)
    {
        int res = 0;
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                if (Math.Abs(a[j] - a[i]) < k)
                    res++;

        return res;
    }
}
```

```
// Driver code
public static void Main ()
{
    int []a = {1, 10, 4, 2};
    int k = 3;
    int n = a.Length;
    Console.WriteLine(countPairs(a, n, k));

}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP code to find count of Pairs
// with difference less than K.

function countPairs( $a, $n, $k)
{
    $res = 0;
    for($i = 0; $i < $n; $i++)
        for($j = $i + 1; $j < $n; $j++)
            if (abs($a[$j] - $a[$i]) < $k)
                $res++;

    return $res;
}

// Driver code
$a = array(1, 10, 4, 2);
$k = 3;
$n = count($a);
echo countPairs($a, $n, $k);

// This code is contributed by anuj_67.
?>
```

Output :

2

Time Complexity : $O(n^2)$
Auxiliary Space : $O(1)$

Method 2 (Sorting) : First we sort the array. Then we start from first element and keep considering pairs while difference is less than k. If we stop the loop when we find difference

more than or equal to k and move to next element.

C++

```
// CPP code to find count of Pairs with
// difference less than K.
#include <bits/stdc++.h>
using namespace std;

int countPairs(int a[], int n, int k)
{
    // to sort the array.
    sort(a, a + n);

    int res = 0;
    for (int i = 0; i < n; i++) {

        // Keep incrementing result while
        // subsequent elements are within
        // limits.
        int j = i+1;
        while (j < n && a[j] - a[i] < k) {
            res++;
            j++;
        }
    }
    return res;
}

// Driver code
int main()
{
    int a[] = {1, 10, 4, 2};
    int k = 3;
    int n = sizeof(a) / sizeof(a[0]);
    cout << countPairs(a, n, k) << endl;
    return 0;
}
```

Java

```
// Java code to find count of Pairs with
// difference less than K.
import java.io.*;
import java.util.Arrays;

class GFG
{
```

```
static int countPairs(int a[], int n, int k)
{
    // to sort the array.
    Arrays.sort(a);

    int res = 0;
    for (int i = 0; i < n; i++)
    {

        // Keep incrementing result while
        // subsequent elements are within
        // limits.
        int j = i + 1;
        while (j < n && a[j] - a[i] < k)
        {
            res++;
            j++;
        }
    }
    return res;
}

// Driver code
public static void main (String[] args)
{
    int a[] = {1, 10, 4, 2};
    int k = 3;
    int n = a.length;
    System.out.println(countPairs(a, n, k));
}
}

// This code is contributed by vt_m.
```

Python3

```
# Python code to find count of Pairs
# with difference less than K.

def countPairs(a, n, k):

    # to sort the array
    a.sort()
    res = 0
    for i in range(n):

        # Keep incrementing result while
        # subsequent elements are within limits.
```

```
j = i+1
while (j < n and a[j] - a[i] < k):
    res += 1
    j += 1
return res

# Driver code
a = [1, 10, 4, 2]
k = 3
n = len(a)
print(countPairs(a, n, k), end = "")

# This code is contributed by Azkia Anam.
```

C#

```
// C# code to find count of Pairs
// with difference less than K.
using System;

class GFG {

    // Function to count pairs
    static int countPairs(int []a, int n,
                          int k)
    {

        // to sort the array.
        Array.Sort(a);

        int res = 0;
        for (int i = 0; i < n; i++)
        {

            // Keep incrementing result while
            // subsequent elements are within
            // limits.
            int j = i + 1;
            while (j < n && a[j] - a[i] < k)
            {
                res++;
                j++;
            }
        }
        return res;
    }

    // Driver code
```

```
public static void Main ()
{
    int []a = {1, 10, 4, 2};
    int k = 3;
    int n = a.Length;
    Console.WriteLine(countPairs(a, n, k));
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP code to find count of
// Pairs with difference less than K.

function countPairs( $a, $n, $k)
{
    // to sort the array.
    sort($a);

    $res = 0;
    for ( $i = 0; $i < $n; $i++)
    {

        // Keep incrementing result
        // while subsequent elements
        // are within limits.
        $j = $i + 1;
        while ( $j < $n and
                $a[$j] - $a[$i] < $k)
        {
            $res++;
            $j++;
        }
    }
    return $res;
}

// Driver code
$a = array(1, 10, 4, 2);
$k = 3;
$n = count($a);
echo countPairs($a, $n, $k);

// This code is contributed by anuj_67.
?>
```

Output :

2

Time complexity : $O(\text{res})$ where **res** is number of pairs in output. Note that in worst case this also takes $O(n^2)$ time but works much better in general.

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/pairs-difference-less-k/>

Chapter 235

Pancake sorting

Pancake sorting - GeeksforGeeks

Given an unsorted array, sort the given array. You are allowed to do only following operation on array.

```
flip(arr, i): Reverse array from 0 to i
```

Unlike a traditional sorting algorithm, which attempts to sort with the fewest comparisons possible, the goal is to sort the sequence in as few reversals as possible.

The idea is to do something similar to [Selection Sort](#). We one by one place maximum element at the end and reduce the size of current array by one.

Following are the detailed steps. Let given array be arr[] and size of array be n.

- 1) Start from current size equal to n and reduce current size by one while it's greater than 1.
 1. Let the current size be curr_size. Do following for every curr_size
 - a) Find index of the maximum element in arr[0..curr_size-1]. Let the index be 'mi'
 - b) Call flip(arr, mi)
 - c) Call flip(arr, curr_size-1)

See following video for visualization of the above algorithm.

<http://www.youtube.com/embed/kk-DDgoXfk>

C

```
// C program to
// sort array using
// pancake sort
#include <stdlib.h>
#include <stdio.h>

/* Reverses arr[0..i] */
```

```
void flip(int arr[], int i)
{
    int temp, start = 0;
    while (start < i)
    {
        temp = arr[start];
        arr[start] = arr[i];
        arr[i] = temp;
        start++;
        i--;
    }
}

// Returns index of the
// maximum element in
// arr[0..n-1]
int findMax(int arr[], int n)
{
    int mi, i;
    for (mi = 0, i = 0; i < n; ++i)
        if (arr[i] > arr[mi])
            mi = i;
    return mi;
}

// The main function that
// sorts given array using
// flip operations
int pancakeSort(int *arr, int n)
{
    // Start from the complete
    // array and one by one
    // reduce current size
    // by one
    for (int curr_size = n; curr_size > 1; --curr_size)
    {
        // Find index of the
        // maximum element in
        // arr[0..curr_size-1]
        int mi = findMax(arr, curr_size);

        // Move the maximum
        // element to end of
        // current array if
        // it's not already
        // at the end
        if (mi != curr_size-1)
        {
```

```
// To move at the end,
// first move maximum
// number to beginning
flip(arr, mi);

// Now move the maximum
// number to end by
// reversing current array
flip(arr, curr_size-1);
}
}

// A utility function to print
// n array of size n
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
        printf("%d ", arr[i]);
}

// Driver program to test above function
int main()
{
    int arr[] = {23, 10, 20, 11, 12, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    pancakeSort(arr, n);

    puts("Sorted Array ");
    printArray(arr, n);

    return 0;
}
```

Java

```
// Java program to
// sort array using
// pancake sort
import java.io.*;

class PancakeSort {

    /* Reverses arr[0..i] */
    static void flip(int arr[], int i)
    {
        int temp, start = 0;
```

```
while (start < i)
{
    temp = arr[start];
    arr[start] = arr[i];
    arr[i] = temp;
    start++;
    i--;
}
}

// Returns index of the
// maximum element in
// arr[0..n-1]
static int findMax(int arr[], int n)
{
    int mi, i;
    for (mi = 0, i = 0; i < n; ++i)
        if (arr[i] > arr[mi])
            mi = i;
    return mi;
}

// The main function that
// sorts given array using
// flip operations
static int pancakeSort(int arr[], int n)
{
    // Start from the complete
    // array and one by one
    // reduce current size by one
    for (int curr_size = n; curr_size > 1; --curr_size)
    {
        // Find index of the
        // maximum element in
        // arr[0..curr_size-1]
        int mi = findMax(arr, curr_size);

        // Move the maximum element
        // to end of current array
        // if it's not already at
        // the end
        if (mi != curr_size-1)
        {
            // To move at the end,
            // first move maximum
            // number to beginning
            flip(arr, mi);
        }
    }
}
```

```
// Now move the maximum
// number to end by
// reversing current array
flip(arr, curr_size-1);
}
}
return 0;
}

/* Utility function to print array arr[] */
static void printArray(int arr[], int arr_size)
{
    for (int i = 0; i < arr_size; i++)
        System.out.print(arr[i] + " ");
    System.out.println("");
}

/* Driver function to check for above functions*/
public static void main (String[] args)
{
    int arr[] = {23, 10, 20, 11, 12, 6, 7};
    int n = arr.length;

    pancakeSort(arr, n);

    System.out.println("Sorted Array: ");
    printArray(arr, n);
}
}
/* This code is contributed by Devesh Agrawal*/
```

Python3

```
# Python3 program to
# sort array using
# pancake sort

# Reverses arr[0..i] */
def flip(arr, i):
    start = 0
    while start < i:
        temp = arr[start]
        arr[start] = arr[i]
        arr[i] = temp
        start += 1
        i -= 1

# Returns index of the maximum
```

```
# element in arr[0..n-1] */
def findMax(arr, n):
    mi = 0
    for i in range(0,n):
        if arr[i] > arr[mi]:
            mi = i
    return mi

# The main function that
# sorts given array
# using flip operations
def pancakeSort(arr, n):

    # Start from the complete
    # array and one by one
    # reduce current size
    # by one
    curr_size = n
    while curr_size > 1:
        # Find index of the maximum
        # element in
        # arr[0..curr_size-1]
        mi = findMax(arr, curr_size)

        # Move the maximum element
        # to end of current array
        # if it's not already at
        # the end
        if mi != curr_size-1:
            # To move at the end,
            # first move maximum
            # number to beginning
            flip(arr, mi)

            # Now move the maximum
            # number to end by
            # reversing current array
            flip(arr, curr_size-1)
        curr_size -= 1

    # A utility function to
    # print an array of size n
def printArray(arr, n):
    for i in range(0,n):
        print ("%d"% (arr[i]),end=" ")

# Driver program
arr = [23, 10, 20, 11, 12, 6, 7]
```

```
n = len(arr)
pancakeSort(arr, n);
print ("Sorted Array ")
printArray(arr,n)

# This code is contributed by shreyanshi_arun.
```

C#

```
// C# program to sort array using
// pancake sort
using System;

class GFG {

    // Reverses arr[0..i]
    static void flip(int []arr, int i)
    {
        int temp, start = 0;
        while (start < i)
        {
            temp = arr[start];
            arr[start] = arr[i];
            arr[i] = temp;
            start++;
            i--;
        }
    }

    // Returns index of the
    // maximum element in
    // arr[0..n-1]
    static int findMax(int []arr, int n)
    {
        int mi, i;
        for (mi = 0, i = 0; i < n; ++i)
            if (arr[i] > arr[mi])
                mi = i;

        return mi;
    }

    // The main function that
    // sorts given array using
    // flip operations
    static int pancakeSort(int []arr, int n)
    {
```

```
// Start from the complete
// array and one by one
// reduce current size by one
for (int curr_size = n; curr_size > 1;
     --curr_size)
{
    // Find index of the
    // maximum element in
    // arr[0..curr_size-1]
    int mi = findMax(arr, curr_size);

    // Move the maximum element
    // to end of current array
    // if it's not already at
    // the end
    if (mi != curr_size - 1)
    {
        // To move at the end,
        // first move maximum
        // number to beginning
        flip(arr, mi);

        // Now move the maximum
        // number to end by
        // reversing current array
        flip(arr, curr_size - 1);
    }
}

return 0;
}

// Utility function to print
// array arr[]
static void printArray(int []arr,
                      int arr_size)
{
    for (int i = 0; i < arr_size; i++)
        Console.Write(arr[i] + " ");

    Console.Write("");
}

// Driver function to check for
// above functions
public static void Main ()
{
```

```
int []arr = {23, 10, 20, 11, 12, 6, 7};  
int n = arr.Length;  
  
pancakeSort(arr, n);  
  
Console.WriteLine("Sorted Array: ");  
printArray(arr, n);  
}  
}  
  
// This code is contributed by nitin mittal.
```

Output:

```
Sorted Array  
6 7 10 11 12 20 23
```

Total $O(n)$ flip operations are performed in above code. The overall time complexity is $O(n^2)$.

References:

http://en.wikipedia.org/wiki/Pancake_sorting

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/pancake-sorting/>

Chapter 236

Pandigital Product

Pandigital Product - GeeksforGeeks

A [Pandigital Number](#) is number which makes the use of all digits 1 to 9 exactly once. We are given a number, we need to find if there are two numbers whose multiplication is given number and given three numbers together are pandigital.

Examples:

```
Input : 7254
Output : Yes
39 * 186 = 7254. We can notice that
the three numbers 39, 186 and 7254
together have all digits from 1 to 9.
```

```
Input : 6952
Output : Yes
```

The idea is to consider all pairs that multiply to given number. For every pair, create a string containing three numbers (given number and current pair). We sort the created string and check if sorted string is equal to “123456789”.

C++

```
// C++ code to check the number
// is Pandigital Product or not
#include <bits/stdc++.h>
using namespace std;

// To check the string formed
// from multiplicand, multiplier
// and product is pandigital
bool isPandigital(string str)
```

```

{
    if (str.length() != 9)
        return false;

    char ch[str.length()];
    strcpy(ch, str.c_str());
    sort(ch, ch + str.length());
    string s = ch;

    if(s.compare("123456789") == 0)
        return true;
    else
        return true;
}

// calculate the multiplicand,
// multiplier, and product
// eligible for pandigital
bool PandigitalProduct_1_9(int n)
{
    for (int i = 1; i * i <= n; i++)
        if (n % i == 0 && isPandigital(to_string(n) +
                                         to_string(i) +
                                         to_string(n / i)))
            return true;
    return false;
}

// Driver Code
int main()
{
    int n = 6952;
    if (PandigitalProduct_1_9(n) == true)
        cout << "yes";
    else
        cout << "no";
    return 0;
}

// This code is contributed by
// Manish Shaw(manishshaw1)

```

Java

```

// Java code to check the number
// is Pandigital Product or not
import java.io.*;
import java.util.*;

```

```
class GFG {

    // calculate the multiplicand, multiplier, and product
    // eligible for pandigital
    public static boolean PandigitalProduct_1_9(int n)
    {
        for (int i = 1; i*i <= n; i++)
            if (n % i == 0 && isPandigital("" + n + i + n / i))
                return true;
        return false;
    }

    // To check the string formed from multiplicand
    // multiplier and product is pandigital
    public static boolean isPandigital(String str)
    {
        if (str.length() != 9)
            return false;
        char ch[] = str.toCharArray();
        Arrays.sort(ch);
        return new String(ch).equals("123456789");
    }

    // Driver function
    public static void main(String[] args)
    {
        int n = 6952;
        if (PandigitalProduct_1_9(n) == true)
            System.out.println("yes");
        else
            System.out.println("no");
    }
}
```

C#

```
// C# code to check the number
// is Pandigital Product or not.
using System;

class GFG {

    // calculate the multiplicand,
    // multiplier, and product
    // eligible for pandigital
    public static bool PandigitalProduct_1_9(int n)
    {
        for (int i = 1; i*i <= n; i++)
```

```
        if (n % i == 0 && isPandigital("'" + n
                                  + i + n / i))
            return true;

        return false;
    }

    // To check the string formed from multiplicand
    // multiplier and product is pandigital
    public static bool isPandigital(String str)
    {
        if (str.Length != 9)
            return false;

        char []ch = str.ToCharArray();
        Array.Sort(ch);

        return new String(ch).Equals("123456789");
    }

    // Driver function
    public static void Main()
    {
        int n = 6952;

        if (PandigitalProduct_1_9(n) == true)
            Console.WriteLine("yes");
        else
            Console.WriteLine("no");
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// PHP code to check the number
// is Pandigital Product or not

// To check the string formed
// from multiplicand, multiplier
// and product is pandigital
function isPandigital($str)
{
    if (strlen($str) != 9)
        return false;
    $x = str_split($str);
```

```
sort($x);
$x = implode($x);
return strcmp($x, "123456789");
}

// calculate the multiplicand,
// multiplier, and product
// eligible for pandigital
function PandigitalProduct_1_9($n)
{
    for ($i = 1;
        $i * $i <= $n; $i++)
        if ($n % $i == 0 &&
            isPandigital(strval($n) .
                strval($i) .
                strval((int)($n / $i))))
            return true;
    return false;
}

// Driver Code
$n = 6050;
if (PandigitalProduct_1_9($n))
    echo "yes";
else
    echo "no";

// This code is contributed
// by mits
?>
```

Output:

yes

Improved By : [nitin mittal](#), [manishshaw1](#), [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/pandigital-product/>

Chapter 237

Permute two arrays such that sum of every pair is greater or equal to K

Permute two arrays such that sum of every pair is greater or equal to K - GeeksforGeeks

Given two arrays of equal size **n** and an integer **k**. The task is to permute both arrays such that sum of their corresponding element is greater than or equal to k i.e $a[i] + b[i] \geq k$. The task is print “Yes” if any such permutation exists, otherwise print “No”.

Examples :

```
Input : a[] = {2, 1, 3},  
        b[] = { 7, 8, 9 },  
        k = 10.  
Output : Yes  
Permutation a[] = { 1, 2, 3 } and b[] = { 9, 8, 7 }  
satisfied the condition a[i] + b[i] >= K.  
  
Input : a[] = {1, 2, 2, 1},  
        b[] = { 3, 3, 3, 4 },  
        k = 5.  
Output : No
```

The idea is to sort one array in ascending order and another array in descending order and if any index does not satisfy the condition $a[i] + b[i] \geq K$ then print “No”, else print “Yes”.

If the condition fails on sorted arrays, then there exists no permutation of arrays which can satisfy the inequality. **Proof,**

Assume **a_{sort}**[] be sorted a[] in ascending order and **b_{sort}**[] be sorted b[] in descending order. Let new permutation b[] is created by swapping any two indices i, j of b_{sort}[],

- **Case 1:** $i < j$ and element at $b[i]$ is now $b_{\text{sort}}[j]$.
Now, $a_{\text{sort}}[i] + b_{\text{sort}}[j] < K$, because $b_{\text{sort}}[i] > b_{\text{sort}}[j]$ as $b[]$ is sorted in decreasing order and we know $a_{\text{sort}}[i] + b_{\text{sort}}[i] < k$.
- **Case 2:** $i > j$ and element at $b[i]$ is now $b_{\text{sort}}[j]$.
Now, $a_{\text{sort}}[j] + b_{\text{sort}}[i] < k$, because $a_{\text{sort}}[i] > a_{\text{sort}}[j]$ as $a[]$ is sorted in increasing order and we know $a_{\text{sort}}[i] + b_{\text{sort}}[i] < k$.

Below is the implementation of this approach:

C++

```
// C++ program to check whether permutation of two
// arrays satisfy the condition a[i] + b[i] >= k.
#include<bits/stdc++.h>
using namespace std;

// Check whether any permutation exists which
// satisfy the condition.
bool isPossible(int a[], int b[], int n, int k)
{
    // Sort the array a[] in decreasing order.
    sort(a, a + n);

    // Sort the array b[] in increasing order.
    sort(b, b + n, greater<int>());

    // Checking condition on each index.
    for (int i = 0; i < n; i++)
        if (a[i] + b[i] < k)
            return false;

    return true;
}

// Driven Program
int main()
{
    int a[] = { 2, 1, 3 };
    int b[] = { 7, 8, 9 };
    int k = 10;
    int n = sizeof(a)/sizeof(a[0]);

    isPossible(a, b, n, k) ? cout << "Yes" :
                                cout << "No";
    return 0;
}
```

Java

```
// Java program to check whether
// permutation of two arrays satisfy
// the condition a[i] + b[i] >= k.
import java.util.*;

class GFG
{
    // Check wheather any permutation
    // exists which satisfy the condition.
    static boolean isPossible(Integer a[], int b[],
                           int n, int k)
    {
        // Sort the array a[] in decreasing order.
        Arrays.sort(a, Collections.reverseOrder());

        // Sort the array b[] in increasing order.
        Arrays.sort(b);

        // Checking condition on each index.
        for (int i = 0; i < n; i++)
            if (a[i] + b[i] < k)
                return false;

        return true;
    }

    // Driver code
    public static void main(String[] args) {
        Integer a[] = {2, 1, 3};
        int b[] = {7, 8, 9};
        int k = 10;
        int n = a.length;

        if (isPossible(a, b, n, k))
            System.out.print("Yes");
        else
            System.out.print("No");
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python program to check
# whether permutation of two
# arrays satisfy the condition
# a[i] + b[i] >= k.
```

```
# Check whether any
# permutation exists which
# satisfy the condition.
def isPossible(a,b,n,k):

    # Sort the array a[]
    # in decreasing order.
    a.sort(reverse=True)

    # Sort the array b[]
    # in increasing order.
    b.sort()

    # Checking condition
    # on each index.
    for i in range(n):
        if (a[i] + b[i] < k):
            return False

    return True

# Driver code

a = [ 2, 1, 3]
b = [7, 8, 9]
k = 10
n =len(a)

if(isPossible(a, b, n, k)):
    print("Yes")
else:
    print("No")

# This code is contributed
# by Anant Agarwal.
```

C#

```
// C# program to check whether
// permutation of two arrays satisfy
// the condition a[i] + b[i] >= k.
using System;

class GFG
{
    // Check wheather any permutation
```

```
// exists which satisfy the condition.
static bool isPossible(int []a, int []b,
                      int n, int k)
{
    // Sort the array a[]
    // in decreasing order.
    Array.Sort(a);

    // Sort the array b[]
    // in increasing order.
    Array.Reverse(b);

    // Checking condition on each index.
    for (int i = 0; i < n; i++)
        if (a[i] + b[i] < k)
            return false;

    return true;
}

// Driver code
public static void Main()
{
    int []a = {2, 1, 3};
    int []b = {7, 8, 9};
    int k = 10;
    int n = a.Length;

    if (isPossible(a, b, n, k))
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP program to check whether
// permutation of two arrays satisfy
// the condition a[i] + b[i] >= k.

// Check wheather any permutation
// exists which satisfy the condition.
function isPossible( $a, $b, $n, $k)
{
```

```
// Sort the array a[] in
// decreasing order.
sort($a);

// Sort the array b[] in
// increasing order.
rsort($b);

// Checking condition on each
// index.
for ( $i = 0; $i < $n; $i++ )
    if ($a[$i] + $b[$i] < $k)
        return false;

return true;
}

// Driven Program
$a = array( 2, 1, 3 );
$b = array( 7, 8, 9 );
$k = 10;
$n = count($a);

if(isPossible($a, $b, $n, $k))
    echo "Yes" ;
else
    echo "No";

// This code is contributed by
// anuj_67.
?>
```

Output:

Yes

Time Complexity: O(n log n).

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/permute-two-arrays-sum-every-pair-greater-equal-k/>

Chapter 238

Pigeonhole Sort

Pigeonhole Sort - GeeksforGeeks

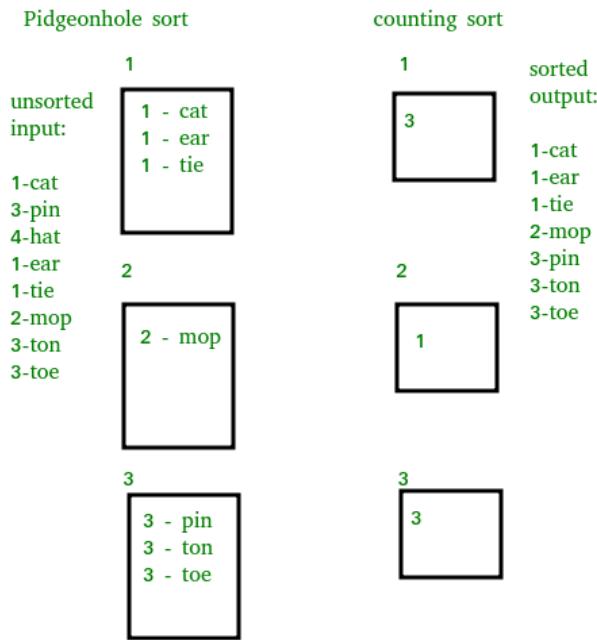
[Pigeonhole sorting](#) is a sorting algorithm that is suitable for sorting lists of elements where the number of elements and the number of possible key values are approximately the same. It requires $O(n + \text{Range})$ time where n is number of elements in input array and ‘Range’ is number of possible values in array.

Working of Algorithm :

1. Find minimum and maximum values in array. Let the minimum and maximum values be ‘min’ and ‘max’ respectively. Also find range as ‘max-min-1’.
2. Set up an array of initially empty “pigeonholes” the same size as of the range.
3. Visit each element of the array and then put each element in its pigeonhole. An element $\text{arr}[i]$ is put in hole at index $\text{arr}[i] - \text{min}$.
4. Start the loop all over the pigeonhole array in order and put the elements from non-empty holes back into the original array.

Comparison with Counting Sort :

It is similar to [counting sort](#), but differs in that it “moves items twice: once to the bucket array and again to the final destination “.



C++

```
/* C program to implement Pigeonhole Sort */
#include <bits/stdc++.h>
using namespace std;

/* Sorts the array using pigeonhole algorithm */
void pigeonholeSort(int arr[], int n)
{
    // Find minimum and maximum values in arr[]
    int min = arr[0], max = arr[0];
    for (int i = 1; i < n; i++)
    {
        if (arr[i] < min)
            min = arr[i];
        if (arr[i] > max)
            max = arr[i];
    }
    int range = max - min + 1; // Find range

    // Create an array of vectors. Size of array
    // range. Each vector represents a hole that
    // is going to contain matching elements.
    vector<int> holes[range];

    // Traverse through input array and put every
    // element in its respective hole
```

```
for (int i = 0; i < n; i++)
    holes[arr[i]-min].push_back(arr[i]);

// Traverse through all holes one by one. For
// every hole, take its elements and put in
// array.
int index = 0; // index in sorted array
for (int i = 0; i < range; i++)
{
    vector<int>::iterator it;
    for (it = holes[i].begin(); it != holes[i].end(); ++it)
        arr[index++] = *it;
}
}

// Driver program to test the above function
int main()
{
    int arr[] = {8, 3, 2, 7, 4, 6, 8};
    int n = sizeof(arr)/sizeof(arr[0]);

    pigeonholeSort(arr, n);

    printf("Sorted order is : ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

Java

```
/* Java program to implement Pigeonhole Sort */

import java.lang.*;
import java.util.*;

public class GFG
{
    public static void pigeonhole_sort(int arr[],
                                      int n)
    {
        int min = arr[0];
        int max = arr[0];
        int range, i, j, index;

        for(int a=0; a<n; a++)
        {
```

```
        if(arr[a] > max)
            max = arr[a];
        if(arr[a] < min)
            min = arr[a];
    }

range = max - min + 1;
int[] phole = new int[range];
Arrays.fill(phole, 0);

for(i = 0; i<n; i++)
    phole[arr[i] - min]++;

index = 0;

for(j = 0; j<range; j++)
    while(phole[j]-->0)
        arr[index++]=j+min;

}

public static void main(String[] args)
{
    GFG sort = new GFG();
    int[] arr = {8, 3, 2, 7, 4, 6, 8};

    System.out.print("Sorted order is : ");

    sort.pigeonhole_sort(arr,arr.length);

    for(int i=0 ; i<arr.length ; i++)
        System.out.print(arr[i] + " ");
}
}

// Code contributed by Mohit Gupta_OMG <(0_o)>
```

Python3

```
# Python program to implement Pigeonhole Sort */

# source code : "https://en.wikibooks.org/wiki/
#   Algorithm_Implementation/Sorting/Pigeonhole_sort"
def pigeonhole_sort(a):
    # size of range of values in the list
    # (ie, number of pigeonholes we need)
```

```
my_min = min(a)
my_max = max(a)
size = my_max - my_min + 1

# our list of pigeonholes
holes = [0] * size

# Populate the pigeonholes.
for x in a:
    assert type(x) is int, "integers only please"
    holes[x - my_min] += 1

# Put the elements back into the array in order.
i = 0
for count in range(size):
    while holes[count] > 0:
        holes[count] -= 1
        a[i] = count + my_min
        i += 1


a = [8, 3, 2, 7, 4, 6, 8]
print("Sorted order is : ", end = ' ')
pigeonhole_sort(a)

for i in range(0, len(a)):
    print(a[i], end = ' ')
```

C#

```
// C# program to implement
// Pigeonhole Sort
using System;

class GFG
{
    public static void pigeonhole_sort(int []arr,
                                       int n)
    {
        int min = arr[0];
        int max = arr[0];
        int range, i, j, index;

        for(int a = 0; a < n; a++)
        {
            if(arr[a] > max)
```

```
        max = arr[a];
        if(arr[a] < min)
            min = arr[a];
    }

range = max - min + 1;
int[] phole = new int[range];

for(i = 0; i < n; i++)
    phole[i] = 0;

for(i = 0; i < n; i++)
    phole[arr[i] - min]++;

index = 0;

for(j = 0; j < range; j++)
    while(phole[j] --> 0)
        arr[index++] = j + min;

}

// Driver Code
static void Main()
{
    int[] arr = {8, 3, 2, 7,
                 4, 6, 8};

    Console.WriteLine("Sorted order is : ");

    pigeonhole_sort(arr,arr.Length);

    for(int i = 0 ; i < arr.Length ; i++)
        Console.Write(arr[i] + " ");
}
}

// This code is contributed
// by Sam007
```

Output:

```
Sorted order is : 2 3 4 6 7 8 8
```

Pigeonhole sort has limited use as requirements are rarely met. For arrays where range is

much larger than n , bucket sort is a generalization that is more efficient in space and time.

References:

https://en.wikipedia.org/wiki/Pigeonhole_sort

This article is contributed by **Ayush Govil**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz

[Selection Sort](#), [Bubble Sort](#), [Insertion Sort](#), [Merge Sort](#), [Heap Sort](#), [QuickSort](#), [Radix Sort](#),
[Counting Sort](#), [Bucket Sort](#), [ShellSort](#), [Comb Sort](#),

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/pigeonhole-sort/>

Chapter 239

Position of an element after stable sort

Position of an element after stable sort - GeeksforGeeks

Given an array of integers which may contain duplicate elements, an element of this array is given to us, we need to tell the final position of this element in the array, if a stable sort algorithm is applied.

Examples :

```
Input : arr[] = [3, 4, 3, 5, 2, 3, 4, 3, 1, 5], index = 5
Output : 4
Element initial index - 5 (third 3)
After sorting array by stable sorting algorithm, we get
array as shown below
[1(8), 2(4), 3(0), 3(2), 3(5), 3(7), 4(1), 4(6), 5(3), 5(9)]
with their initial indices shown in parentheses next to them,
Element's index after sorting = 4
```

One easy way to solve this problem is to use any stable sorting algorithm like [Insertion Sort](#), [Merge Sort](#) etc and then get the new index of given element but we can solve this problem without sorting the array.

As position of an element in a sorted array is decided by only those elements which are smaller than given element. We count all array elements smaller than given element and for those elements which are equal to given element, elements occurring before given elements' index will be included in count of smaller elements this will insure the stability of the result's index.

Simple code to implement above approach is implemented below:

C/C++

```
// C++ program to get index of array element in
```

```
// sorted array
#include <bits/stdc++.h>
using namespace std;

// Method returns the position of arr[idx] after
// performing stable-sort on array
int getIndexInSortedArray(int arr[], int n, int idx)
{
    /* Count of elements smaller than current
       element plus the equal element occurring
       before given index*/
    int result = 0;
    for (int i = 0; i < n; i++) {
        // If element is smaller then increase
        // the smaller count
        if (arr[i] < arr[idx])
            result++;

        // If element is equal then increase count
        // only if it occurs before
        if (arr[i] == arr[idx] && i < idx)
            result++;
    }
    return result;
}

// Driver code to test above methods
int main()
{
    int arr[] = { 3, 4, 3, 5, 2, 3, 4, 3, 1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);

    int idxOfEle = 5;
    cout << getIndexInSortedArray(arr, n, idxOfEle);

    return 0;
}
```

Java

```
// Java program to get index of array
// element in sorted array

class ArrayIndex {

    // Method returns the position of
    // arr[idx] after performing stable-sort
    // on array
```

```
static int getIndexInSortedArray(int arr[],
                                int n, int idx)
{
    /* Count of elements smaller than
       current element plus the equal element
       occurring before given index*/
    int result = 0;
    for (int i = 0; i < n; i++) {

        // If element is smaller then
        // increase the smaller count
        if (arr[i] < arr[idx])
            result++;

        // If element is equal then increase
        // count only if it occurs before
        if (arr[i] == arr[idx] && i < idx)
            result++;
    }
    return result;
}

// Driver code to test above methods
public static void main(String[] args)
{
    int arr[] = { 3, 4, 3, 5, 2, 3, 4, 3, 1, 5 };
    int n = arr.length;

    int idxOfEle = 5;
    System.out.println(getIndexInSortedArray(arr,
                                             n, idxOfEle));
}
}

// This code is contributed by Raghav sharma
```

Python

```
# Python program to get index of array element in
# sorted array
# Method returns the position of arr[idx] after
# performing stable-sort on array

def getIndexInSortedArray(arr, n, idx):
    # Count of elements smaller than current
    # element plus the equal element occurring
    # before given index
    result = 0
```

```
for i in range(n):
    # If element is smaller then increase
    # the smaller count
    if (arr[i] < arr[idx]):
        result += 1

    # If element is equal then increase count
    # only if it occurs before
    if (arr[i] == arr[idx] and i < idx):
        result += 1
return result;

# Driver code to test above methods
arr = [3, 4, 3, 5, 2, 3, 4, 3, 1, 5]
n = len(arr)

idxOfEle = 5
print getIndexInSortedArray(arr, n, idxOfEle)

# Contributed by: Afzal Ansari
```

C#

```
// C# program to get index of array
// element in sorted array
using System;

class ArrayIndex {

    // Method returns the position of
    // arr[idx] after performing stable-sort
    // on array
    static int getIndexInSortedArray(int[] arr,
                                    int n, int idx)
    {
        /* Count of elements smaller than
        current element plus the equal element
        occurring before given index*/
        int result = 0;
        for (int i = 0; i < n; i++) {

            // If element is smaller then
            // increase the smaller count
            if (arr[i] < arr[idx])
                result++;

            // If element is equal then increase
            // count only if it occurs before
```

```
        if (arr[i] == arr[idx] && i < idx)
            result++;
    }
    return result;
}

// Driver code to test above methods
public static void Main()
{
    int[] arr = { 3, 4, 3, 5, 2, 3, 4, 3, 1, 5 };
    int n = arr.Length;

    int idxOfEle = 5;
    Console.WriteLine(getIndexInSortedArray(arr, n,
                                            idxOfEle));
}
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to get index of
// array element in sorted array

// Method returns the position of
// arr[idx] after performing
// stable-sort on array
function getIndexInSortedArray( $arr, $n, $idx)
{

    /* Count of elements smaller
       than current      element plus
       the equal element occurring
       before given index */
    $result = 0;
    for($i = 0; $i < $n; $i++)
    {

        // If element is smaller then
        // increase the smaller count
        if ($arr[$i] < $arr[$idx])
            $result++;

        // If element is equal then
        // increase count only if
        // it occurs before
```

```
        if ($arr[$i] == $arr[$idx] and
            $i < $idx)
            $result++;
    }
    return $result;
}

// Driver Code
$arr = array(3, 4, 3, 5, 2, 3, 4, 3, 1, 5);
$n = count($arr);

$idxOfEle = 5;
echo getIndexInSortedArray($arr, $n,
                           $idxOfEle);

// This code is contributed by anuj_67.
?>
```

Output:

4

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/position-element-stable-sort/>

Chapter 240

Possible to form a triangle from array values

Possible to form a triangle from array values - GeeksforGeeks

Given an array of integers, we need to find out whether it is possible to construct at least one non-degenerate triangle using array values as its sides. In other words, we need to find out 3 such array indices which can become sides of a non-degenerate triangle.

Examples :

Input : [4, 1, 2]

Output : No

No triangle is possible from given array values

Input : [5, 4, 3, 1, 2]

Output : Yes

Sides of possible triangle are 2 3 4

For a non-degenerate triangle, its sides should follow these constraints,

$A + B > C$ and

$B + C > A$ and

$C + A > B$

where A, B and C are length of sides of the triangle.

The task is to find any triplet from array that satisfies above condition.

A **Simple Solution** is to generate all triplets and for every triplet check if it forms a triangle or not by checking above three conditions.

An **Efficient Solution** is use sorting. First, we sort the array then we loop once and we will check three consecutive elements of this array if any triplet satisfies $\text{arr}[i] + \text{arr}[i+1] > \text{arr}[i+2]$, then we will output that triplet as our final result.

Why checking only 3 consecutive elements will work instead of trying all possible triplets of sorted array?

Let we are at index i and 3 line segments are $\text{arr}[i]$, $\text{arr}[i + 1]$ and $\text{arr}[i + 2]$ with relation $\text{arr}[i] < \text{arr}[i+1] < \text{arr}[i+2]$, If they can't form a non-degenerate triangle, Line segments of lengths $\text{arr}[i-1]$, $\text{arr}[i+1]$ and $\text{arr}[i+2]$ or $\text{arr}[i]$, $\text{arr}[i+1]$ and $\text{arr}[i+3]$ can't form a non-degenerate triangle also because sum of $\text{arr}[i-1]$ and $\text{arr}[i+1]$ will be even less than sum of $\text{arr}[i]$ and $\text{arr}[i+1]$ in first case and sum of $\text{arr}[i]$ and $\text{arr}[i+1]$ must be less than $\text{arr}[i+3]$ in second case, So we don't need to try all the combinations, we will try just 3 consecutive indices of array in sorted form.

The total complexity of below solution is $O(n \log n)$

C++

```

// C++ program to find if it
// is possible to form a
// triangle from array values
#include <bits/stdc++.h>
using namespace std;

// Method prints possible
// triangle when array values
// are taken as sides
bool isPossibleTriangle(int arr[],
                        int N)
{
    // If number of elements are
    // less than 3, then no
    // triangle is possible
    if (N < 3)
        return false;

    // first sort the array
    sort(arr, arr + N);

    // loop for all 3
    // consecutive triplets
    for (int i = 0; i < N - 2; i++)

        // If triplet satisfies
        // triangle condition, break
        if (arr[i] + arr[i + 1] > arr[i + 2])
            return true;
}

// Driver Code

```

```
int main()
{
    int arr[] = {5, 4, 3, 1, 2};
    int N = sizeof(arr) / sizeof(int);

    isPossibleTriangle(arr, N) ? cout << "Yes" :
                                cout << "No";
    return 0;
}
```

JAVA

```
// Java program to find if it is
// possible to form a triangle
// from array values
import java.io.*;
import java.util.Arrays;

class GFG
{

    // Method prints possible
    // triangle when array values
    // are taken as sides
    static boolean isPossibleTriangle(int []arr,
                                      int N)
    {

        // If number of elements are
        // less than 3, then no
        // triangle is possible
        if (N < 3)
            return false;

        // first sort the array
        Arrays.sort(arr);

        // loop for all 3
        // consecutive triplets
        for (int i = 0; i < N - 2; i++)

            // If triplet satisfies
            // triangle condition, break
            if (arr[i] + arr[i + 1] > arr[i + 2])
                return true;

        return false;
    }
}
```

```
// Driver Code
static public void main (String[] args)
{
    int []arr = {5, 4, 3, 1, 2};
    int N = arr.length;

    if(isPossibleTriangle(arr, N))
        System.out.println("Yes" );
    else
        System.out.println("No");
}
}

// This code is contributed by vt_m.
```

Python

```
# Python3 code to find if
# it is possible to form a
# triangle from array values

# Method prints possible
# triangle when array
# values are taken as sides
def isPossibleTriangle (arr , N):

    # If number of elements
    # are less than 3, then
    # no triangle is possible
    if N < 3:
        return False

    # first sort the array
    arr.sort()

    # loop for all 3
    # consecutive triplets
    for i in range(N - 2):

        # If triplet satisfies triangle
        # condition, break
        if arr[i] + arr[i + 1] > arr[i + 2]:
            return True

# Driver Code
arr = [5, 4, 3, 1, 2]
N = len(arr)
```

```
print("Yes" if isPossibleTriangle(arr, N) else "No")

# This code is contributed
# by "Sharad_Bhardwaj".

C#

// C# program to find if
// it is possible to form
// a triangle from array values
using System;

class GFG
{

    // Method prints possible
    // triangle when array values
    // are taken as sides
    static bool isPossibleTriangle(int []arr,
                                    int N)
    {
        // If number of elements
        // are less than 3, then
        // no triangle is possible
        if (N < 3)
            return false;

        // first sort the array
        Array.Sort(arr);

        // loop for all 3
        // consecutive triplets
        for (int i = 0; i < N - 2; i++)

            // If triplet satisfies triangle
            // condition, break
            if (arr[i] + arr[i + 1] > arr[i + 2])
                return true;

        return false;
    }

    // Driver Code
    static public void Main ()
    {
        int []arr = {5, 4, 3, 1, 2};
        int N = arr.Length;
```

```
        if(isPossibleTriangle(arr, N))
            Console.WriteLine("Yes" );
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find if
// it is possible to form
// a triangle from array values

// Method prints possible
// triangle when array values
// are taken as sides
function isPossibleTriangle( $arr, $N)
{
    // If number of elements are
    // less than 3, then no
    // triangle is possible
    if ($N < 3)
        return false;

    // first sort the array
    sort($arr);

    // loop for all 3
    // consecutive triplets
    for ( $i = 0; $i < $N - 2; $i++)

        // If triplet satisfies triangle
        // condition, break
        if ($arr[$i] + $arr[$i + 1] > $arr[$i + 2])
            return true;
}

// Driver Code
$arr = array(5, 4, 3, 1, 2);
$N = count($arr);

if(isPossibleTriangle($arr,$N))
echo "Yes" ;
else
```

```
echo "No";  
// This code is contributed by vt_m  
?>
```

Output :

Yes

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/possible-form-triangle-array-values/>

Chapter 241

Print All Distinct Elements of a given integer array

Print All Distinct Elements of a given integer array - GeeksforGeeks

Given an integer array, print all distinct elements in array. The given array may contain duplicates and the output should print every element only once. The given array is not sorted.

Examples:

Input: arr[] = {12, 10, 9, 45, 2, 10, 10, 45}
Output: 12, 10, 9, 45, 2

Input: arr[] = {1, 2, 3, 4, 5}
Output: 1, 2, 3, 4, 5

Input: arr[] = {1, 1, 1, 1, 1}
Output: 1

A **Simple Solution** is to use two nested loops. The outer loop picks an element one by one starting from the leftmost element. The inner loop checks if the element is present on left side of it. If present, then ignores the element, else prints the element. Following is the implementation of the simple algorithm.

C++

```
// C++ program to print all distinct elements in a given array
#include <iostream>
#include <algorithm>
using namespace std;
```

```
void printDistinct(int arr[], int n)
{
    // Pick all elements one by one
    for (int i=0; i<n; i++)
    {
        // Check if the picked element is already printed
        int j;
        for (j=0; j<i; j++)
            if (arr[i] == arr[j])
                break;

        // If not printed earlier, then print it
        if (i == j)
            cout << arr[i] << " ";
    }
}

// Driver program to test above function
int main()
{
    int arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    printDistinct(arr, n);
    return 0;
}
```

Java

```
// Java program to print all distinct
// elements in a given array
import java.io.*;

class GFG {

    static void printDistinct(int arr[], int n)
    {
        // Pick all elements one by one
        for (int i = 0; i < n; i++)
        {
            // Check if the picked element
            // is already printed
            int j;
            for (j = 0; j < i; j++)
                if (arr[i] == arr[j])
                    break;

            // If not printed earlier,
            // then print it
```

```
        if (i == j)
            System.out.print( arr[i] + " ");
    }
}

// Driver program
public static void main (String[] args)
{
    int arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10};
    int n = arr.length;
    printDistinct(arr, n);

}
}

// This code is contributed by vt_m
```

Python3

```
# python program to print all distinct
# elements in a given array

def printDistinct(arr, n):

    # Pick all elements one by one
    for i in range(0, n):

        # Check if the picked element
        # is already printed
        d = 0
        for j in range(0, i):
            if (arr[i] == arr[j]):
                d = 1
                break

        # If not printed earlier,
        # then print it
        if (d == 0):
            print(arr[i])

# Driver program to test above function
arr = [6, 10, 5, 4, 9, 120, 4, 6, 10]
n = len(arr)
printDistinct(arr, n)

# This code is contributed by Sam007.
```

C#

```
// C# program to print all distinct
// elements in a given array
using System;

class GFG {

    static void printDistinct(int []arr, int n)
    {

        // Pick all elements one by one
        for (int i = 0; i < n; i++)
        {

            // Check if the picked element
            // is already printed
            int j;
            for (j = 0; j < i; j++)
                if (arr[i] == arr[j])
                    break;

            // If not printed earlier,
            // then print it
            if (i == j)
                Console.Write(arr[i] + " ");
        }
    }

    // Driver program
    public static void Main ()
    {
        int []arr = {6, 10, 5, 4, 9, 120,
                    4, 6, 10};
        int n = arr.Length;

        printDistinct(arr, n);
    }
}

// This code is contributed by Sam007.
```

PHP

```
<?php
// PHP program to print all distinct
// elements in a given array

function printDistinct($arr, $n)
```

```
{  
    // Pick all elements one by one  
    for($i = 0; $i < $n; $i++)  
    {  
  
        // Check if the picked element  
        // is already printed  
        $j;  
        for($j = 0; $j < $i; $j++)  
            if ($arr[$i] == $arr[$j])  
                break;  
  
        // If not printed  
        // earlier, then print it  
        if ($i == $j)  
            echo $arr[$i] , " ";  
    }  
}  
  
// Driver Code  
$arr = array(6, 10, 5, 4, 9, 120, 4, 6, 10);  
$n = sizeof($arr);  
printDistinct($arr, $n);  
  
// This code is contributed by nitin mittal  
?>
```

Output:

```
6 10 5 4 9 120
```

Time Complexity of above solution is $O(n^2)$. We can **Use Sorting** to solve the problem in $O(n\log n)$ time. The idea is simple, first sort the array so that all occurrences of every element become consecutive. Once the occurrences become consecutive, we can traverse the sorted array and print distinct elements in $O(n)$ time. Following is the implementation of the idea.

C++

```
// C++ program to print all distinct elements in a given array  
#include <iostream>  
#include <algorithm>  
using namespace std;  
  
void printDistinct(int arr[], int n)  
{  
    // First sort the array so that all occurrences become consecutive  
    sort(arr, arr + n);
```

```
// Traverse the sorted array
for (int i=0; i<n; i++)
{
    // Move the index ahead while there are duplicates
    while (i < n-1 && arr[i] == arr[i+1])
        i++;

    // print last occurrence of the current element
    cout << arr[i] << " ";
}
}

// Driver program to test above function
int main()
{
    int arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    printDistinct(arr, n);
    return 0;
}
```

Java

```
// Java program to print all distinct
// elements in a given array
import java.io.*;
import java.util.*;

class GFG
{
    static void printDistinct(int arr[], int n)
    {
        // First sort the array so that
        // all occurrences become consecutive
        Arrays.sort(arr);

        // Traverse the sorted array
        for (int i = 0; i < n; i++)
        {
            // Move the index ahead while
            // there are duplicates
            while (i < n - 1 && arr[i] == arr[i + 1])
                i++;

            // print last occurrence of
            // the current element
            System.out.print(arr[i] + " ");
        }
    }
}
```

```
        }
    }

// Driver program
public static void main (String[] args)
{
    int arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10};
    int n = arr.length;
    printDistinct(arr, n);

}
}

// This code is contributed by vt_m
```

C#

```
// C# program to print all distinct
// elements in a given array
using System;

class GFG {

    static void printDistinct(int []arr, int n)
    {

        // First sort the array so that
        // all occurrences become consecutive
        Array.Sort(arr);

        // Traverse the sorted array
        for (int i = 0; i < n; i++)
        {

            // Move the index ahead while
            // there are duplicates
            while (i < n - 1 && arr[i] == arr[i + 1])
                i++;

            // print last occurrence of
            // the current element
            Console.Write(arr[i] + " ");
        }
    }

// Driver program
public static void Main ()
{
```

```
int []arr = {6, 10, 5, 4, 9, 120, 4, 6, 10};  
int n = arr.Length;  
  
    printDistinct(arr, n);  
}  
}  
  
// This code is contributed by Sam007.
```

PHP

```
<?php  
// PHP program to print all distinct  
// elements in a given array  
  
function printDistinct( $arr, $n)  
{  
  
    // First sort the array so  
    // that all occurrences  
    // become consecutive  
    sort($arr);  
  
    // Traverse the sorted array  
    for ($i = 0; $i < $n; $i++)  
    {  
  
        // Move the index ahead  
        // while there are duplicates  
        while ($i < $n - 1 and  
              $arr[$i] == $arr[$i + 1])  
            $i++;  
  
        // print last occurrence  
        // of the current element  
        echo $arr[$i] , " ";  
    }  
}  
  
// Driver Code  
$arr = array(6, 10, 5, 4, 9, 120, 4, 6, 10);  
$n = count($arr);  
printDistinct($arr, $n);  
  
// This code is contributed by anuj_67.  
?>
```

Output:

4 5 6 9 10 120

We can **Use Hashing** to solve this in $O(n)$ time on average. The idea is to traverse the given array from left to right and keep track of visited elements in a hash table. Following is Java implementation of the idea.

```
/* Java program to print all distinct elements of a given array */
import java.util.*;

class Main
{
    // This function prints all distinct elements
    static void printDistinct(int arr[])
    {
        // Creates an empty hashset
        HashSet<Integer> set = new HashSet<>();

        // Traverse the input array
        for (int i=0; i<arr.length; i++)
        {
            // If not present, then put it in hashtable and print it
            if (!set.contains(arr[i]))
            {
                set.add(arr[i]);
                System.out.print(arr[i] + " ");
            }
        }
    }

    // Driver method to test above method
    public static void main (String[] args)
    {
        int arr[] = {10, 5, 3, 4, 3, 5, 6};
        printDistinct(arr);
    }
}
```

Output:

10 5 3 4 6

One more advantage of hashing over sorting is, the elements are printed in same order as they are in input array.

Improved By : [Sam007](#), [nitin mittal](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/print-distinct-elements-given-integer-array/>

Chapter 242

Print Binary Tree levels in sorted order

Print Binary Tree levels in sorted order - GeeksforGeeks

Given a Binary tree, the task is to print its all level in sorted order

Examples:

Input : 7
 / \
 6 5
 / \ / \\\
 4 3 2 1
Output :
7
5 6
1 2 3 4

Input : 7
 / \
 16 1
 / \
 4 13
Output :
7
1 16
4 13

Here we can use two [Priority queue](#) for print in sorted order. We create an empty queue q and two priority queues, current_level and next_level. We use NULL as a separator between two levels. Whenever we encounter NULL in normal level order traversal, we swap current_level and next_level.

```
// CPP program to print levels in sorted order.
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

// A Binary Tree Node
struct Node {
    int data;
    struct Node *left, *right;
};

// Iterative method to find height of Binary Tree
void printLevelOrder(Node* root)
{
    // Base Case
    if (root == NULL)
        return;

    // Create an empty queue for level order traversal
    queue<Node*> q;

    // A priority queue (or min heap) of integers for
    // to store all elements of current level.
    priority_queue<int, vector<int>, greater<int> > current_level;

    // A priority queue (or min heap) of integers for
    // to store all elements of next level.
    priority_queue<int, vector<int>, greater<int> > next_level;

    // push the root for traverse all next level nodes
    q.push(root);

    // for go level by level
    q.push(NULL);

    // push the first node data in previous_level queue
    current_level.push(root->data);

    while (q.empty() == false) {

        // Get top of priority queue
        int data = current_level.top();

        // Get top of queue
        Node* node = q.front();

        // if node == NULL (Means this is boundary
```

```
// between two levels), swap current_level
// next_level priority queues.
if (node == NULL) {
    q.pop();

    // here queue is empty represent
    // no element in the actual
    // queue
    if (q.empty())
        break;

    q.push(NULL);
    cout << "\n";

    // swap next_level to current_level level
    // for print in sorted order
    current_level.swap(next_level);

    continue;
}

// print the current_level data
cout << data << " ";

q.pop();
current_level.pop();

/* Enqueue left child */
if (node->left != NULL) {
    q.push(node->left);

    // Enqueue left child in next_level queue
    next_level.push(node->left->data);
}

/*Enqueue right child */
if (node->right != NULL) {
    q.push(node->right);

    // Enqueue right child in next_level queue
    next_level.push(node->right->data);
}

}

}

// Utility function to create a new tree node
Node* newNode(int data)
{
```

```
Node* temp = new Node;
temp->data = data;
temp->left = temp->right = NULL;
return temp;
}

// Driver program to test above functions
int main()
{
    // Let us create binary tree shown in above diagram
    Node* root = newNode(7);
    root->left = newNode(6);
    root->right = newNode(5);
    root->left->left = newNode(4);
    root->left->right = newNode(3);
    root->right->left = newNode(2);
    root->right->right = newNode(1);

    /*      7
           /   \
          6     5
         / \   / \
        4  3   2  1 */

    cout << "Level Order traversal of binary tree is \n";
    printLevelOrder(root);
    return 0;
}
```

Output:

```
Level Order traversal of binary tree is
7
5 6
1 2 3 4
```

Source

<https://www.geeksforgeeks.org/print-binary-tree-levels-sorted-order/>

Chapter 243

Print Binary Tree levels in sorted order Set 2 (Using set)

Print Binary Tree levels in sorted order Set 2 (Using set) - GeeksforGeeks

Given a tree, print the level order traversal in sorted order.

Examples :

Input : 7
 / \
 6 5
 / \ / \
 4 3 2 1

Output :

7
5 6
1 2 3 4

Input : 7
 / \
 16 1
 / \
 4 13

Output :

7
1 16
4 13

We have discussed a priority queue based solution in below post.

[Print Binary Tree levels in sorted order Set 1 \(Using Priority Queue\)](#)

In this post, a [set](#) (which is implemented using balanced binary search tree) based solution is discussed.

Approach :

1. Start level order traversal of tree.
2. Store all the nodes in a set(or any other similar data structures).
3. Print elements of set.

C++

```
// CPP code to print level order
// traversal in sorted order
#include <bits/stdc++.h>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int dat = 0)
        : data(dat), left(nullptr),
          right(nullptr)
    {
    }
};

// Function to print sorted
// level order traversal
void sorted_level_order(Node* root)
{
    queue<Node*> q;
    set<int> s;

    q.push(root);
    q.push(nullptr);

    while (q.empty() == false) {
        Node* tmp = q.front();
        q.pop();

        if (tmp == nullptr) {
            if (s.empty() == true)
                break;
            for (set<int>::iterator it =
                  s.begin(); it != s.end(); ++it)
                cout << *it << " ";
            q.push(nullptr);
            s.clear();
        }
    }
}
```

```
        else {
            s.insert(tmp->data);

            if (tmp->left != nullptr)
                q.push(tmp->left);
            if (tmp->right != nullptr)
                q.push(tmp->right);
        }
    }

// Driver code
int main()
{
    Node* root = new Node(7);
    root->left = new Node(6);
    root->right = new Node(5);
    root->left->left = new Node(4);
    root->left->right = new Node(3);
    root->right->left = new Node(2);
    root->right->right = new Node(1);
    sorted_level_order(root);
    return 0;
}
```

Output:

7 5 6 1 2 3 4

Source

<https://www.geeksforgeeks.org/print-binary-tree-levels-sorted-order-2/>

Chapter 244

Print all the pairs that contains the positive and negative values of an element

Print all the pairs that contains the positive and negative values of an element - Geeks-forGeeks

Given an array of distinct integers, print all the pairs having a positive value and negative value of a number that exists in the array.

Note: Order of the pairs doesn't matter.

Examples:

Input: arr[] = { 1, -3, 2, 3, 6, -1 }
Output: -1 1 -3 3

Input: arr[] = { 4, 8, 9, -4, 1, -1, -8, -9 }
Output: -1 1 -4 4 -8 8 -9 9

A **naive approach** is to run two loops i.e. Consider each element of the array using the outer loop and search for its corresponding positive/negative value in the array using an inner loop. Similarly, find all the pairs. Time Complexity of this approach will be $O(n^2)$.

A **better approach** is to use **sorting** i.e. first sort the array and then for each negative element, do a binary search to find its counterpart (+ve number). If found, print that pair. If the current element is positive then break that loop as after that there will be all the positive numbers.

C++

```
// CPP program to find pairs of positive
```

```
// and negative values present in an array.
#include <bits/stdc++.h>
using namespace std;

void printPairs(int arr[], int n)
{
    bool pair_exists = false;
    // Sort the array
    sort(arr, arr + n);

    // Traverse the array
    for (int i = 0; i < n; i++) {

        // For every arr[i] < 0 element,
        // do a binary search for arr[i] > 0.
        if (arr[i] < 0) {

            // If found, print the pair.
            if (binary_search(arr, arr + n, -arr[i])) {
                cout << arr[i] << ", " << -arr[i] << endl;

                pair_exists = true;
            }
        }

        else
            break;
    }

    if (pair_exists == false)
        cout << "No such pair exists";
}

// Driver code
int main()
{
    int arr[] = { 4, 8, 9, -4, 1, -1, -8, -9 };
    int n = sizeof(arr) / sizeof(arr[0]);

    printPairs(arr, n);

    return 0;
}
```

Java

```
// Java program to find pairs
// of positive and negative
```

```
// values present in an array.
import java.util.*;
class GFG
{
static void printPairs(int arr[], int n)
{
    boolean pair_exists = false;

    // Sort the array
    Arrays.sort(arr);

    // Traverse the array
    for (int i = 0; i < n; i++)
    {

        // For every arr[i] < 0 element,
        // do a binary search for arr[i] > 0.
        if (arr[i] < 0)
        {

            // If found, print the pair.
            if (java.util.Arrays.binarySearch(arr, -arr[i])!=-1)
            {
                System.out.println(arr[i] + " , " + -arr[i] );
                pair_exists = true;
            }
        }

        else
            break;
    }

    if (pair_exists == false)
        System.out.println("No such pair exists");
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 4, 8, 9, -4, 1, -1, -8, -9 };
    int n =arr.length;

    printPairs(arr, n);
}
}

// This code is contributed
```

// by Arnab Kundu

Output:

```
-9, 9  
-8, 8  
-4, 4  
-1, 1
```

Time Complexity: O(nlogn)

An **efficient Approach** is to use **hashing**. Below are the required steps:

- Start traversing the array.
- Store all the psoitve values in an unordered_set.
- Check for each negative element, if their corresponding positive element exists in the set or not.
- If yes, print the pair
- Also, maintain a flag to check if no such pair exists.

C++

```
// CPP program to find pairs of positive  
// and negative values present in an array  
#include <bits/stdc++.h>  
using namespace std;  
  
// Function to print pairs of positive  
// and negative values present in the array  
void printPairs(int arr[], int n)  
{  
    unordered_set<int> pairs;  
    bool pair_exists = false;  
  
    // Store all the positive elements  
    // in the unordered_set  
    for (int i = 0; i < n; i++)  
        if (arr[i] > 0)  
            pairs.insert(arr[i]);  
  
    // Start traversing the array
```

```
for (int i = 0; i < n; i++) {  
  
    // Check if the positive value of current  
    // element exists in the set or not  
    if (arr[i] < 0)  
        if (pairs.find(-arr[i]) != pairs.end())  
  
            { // Print that pair  
                cout << arr[i] << ", " << -arr[i] << endl;  
  
                pair_exists = true;  
            }  
    }  
  
    if (pair_exists == false)  
        cout << "No such pair exists";  
}  
  
// Driver code  
int main()  
{  
    int arr[] = { 4, 8, 9, -4, 1, -1, -8, -9 };  
    int n = sizeof(arr) / sizeof(arr[0]);  
  
    printPairs(arr, n);  
    return 0;  
}
```

Output:

```
-4, 4  
-1, 1  
-8, 8  
-9, 9
```

Time Complexity: O(n)

Improved By : andrew1234

Source

<https://www.geeksforgeeks.org/print-all-the-pairs-that-contains-the-positive-and-negative-values-of-an-element/>

Chapter 245

Print all triplets with given sum

Print all triplets with given sum - GeeksforGeeks

Given an array of distinct elements. The task is to find triplets in array whose sum is equal to a given number.

Examples :

```
Input : arr[] = {0, -1, 2, -3, 1}
        sum = -2
Output : 0  -3  1
         -1  2  -3
```

```
Input : arr[] = {1, -2, 1, 0, 5}
        sum = 0
Output : 1 -2  1
```

Method 1 (Simple : O(n³))

The naive approach is that run three loops and check one by one that sum of three elements is given sum or not If sum of three elements is given sum, then print elements otherwise print not found.

C++

```
// A simple C++ program to find three elements
// whose sum is equal to given sum
#include <bits/stdc++.h>
using namespace std;

// Prints all triplets in arr[] with given sum
void findTriplets(int arr[], int n, int sum)
{
    for (int i = 0; i < n - 2; i++) {
```

```
for (int j = i + 1; j < n - 1; j++) {
    for (int k = j + 1; k < n; k++) {
        if (arr[i] + arr[j] + arr[k] == sum) {
            cout << arr[i] << " "
                << arr[j] << " "
                << arr[k] << endl;
        }
    }
}
}

// Driver code
int main()
{
    int arr[] = { 0, -1, 2, -3, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    findTriplets(arr, n, -2);
    return 0;
}
```

Java

```
// A simple Java program
// to find three elements
// whose sum is equal to
// given sum
import java.io.*;

class GFG
{

    // Prints all triplets in
    // arr[] with given sum
    static void findTriplets(int arr[],
                            int n, int sum)
    {
        for (int i = 0;
             i < n - 2; i++)
        {
            for (int j = i + 1;
                 j < n - 1; j++)
            {
                for (int k = j + 1;
                     k < n; k++)
                {
                    if (arr[i] + arr[j] + arr[k] == sum)
                    {
```

```
        System.out.println(arr[i]+ " "+
                           arr[j] +" "+
                           arr[k] );
    }
}
}
}

// Driver code
public static void main (String[] args)
{
    int arr[] = {0, -1, 2, -3, 1};
    int n = arr.length;
    findTriplets(arr, n, -2);
}
}

// This code is contributed by m_kit
```

Python 3

```
# A simple Python 3 program
# to find three elements
# whose sum is equal to
# given sum

# Prints all triplets in
# arr[] with given sum
def findTriplets(arr, n, sum):

    for i in range(0 , n - 2):
        for j in range(i + 1 , n - 1):
            for k in range(j + 1, n):
                if (arr[i] + arr[j] +
                    arr[k] == sum):
                    print(arr[i], " ",
                          arr[j] , " ",
                          arr[k] , sep = "")

# Driver code
arr = [ 0, -1, 2, -3, 1 ]
n = len(arr)
findTriplets(arr, n, -2)

# This code is contributed
# by Smitha
```

C#

```
// A simple C# program
// to find three elements
// whose sum is equal to
// given sum
using System;

class GFG
{

    // Prints all triplets in
    // arr[] with given sum
    static void findTriplets(int []arr,
                            int n, int sum)
    {
        for (int i = 0;
              i < n - 2; i++)
        {
            for (int j = i + 1;
                  j < n - 1; j++)
            {
                for (int k = j + 1;
                      k < n; k++)
                {
                    if (arr[i] + arr[j] + arr[k] == sum)
                    {
                        Console.WriteLine(arr[i]+ " "+
                                         arr[j] + " "+
                                         arr[k] );
                    }
                }
            }
        }
    }

    // Driver code
    static public void Main ()
    {
        int []arr = {0, -1, 2, -3, 1};
        int n = arr.Length;
        findTriplets(arr, n, -2);
    }
}

// This code is contributed by akt_mit
```

PHP

```
<?php
// A simple PHP program to
// find three elements whose
// sum is equal to given sum

// Prints all triplets in
// arr[] with given sum
function findTriplets($arr, $n, $sum)
{
    for ($i = 0; $i < $n - 2; $i++)
    {
        for ($j = $i + 1; $j < $n - 1; $j++)
        {
            for ($k = $j + 1; $k < $n; $k++)
            {
                if ($arr[$i] + $arr[$j] +
                    $arr[$k] == $sum)
                {
                    echo $arr[$i] , " ",
                        $arr[$j] , " ",
                        $arr[$k] , "\n";
                }
            }
        }
    }

// Driver code
$arr = array (0, -1, 2, -3, 1);
$n = sizeof($arr);
findTriplets($arr, $n, -2);

// This code is contributed by aj_36
?>
```

Output :

```
0 -3 1
-1 2 -3
```

Time Complexity : $O(n^3)$
Auxiliary Space : $O(1)$

Method 2 (Hashing : $O(n^2)$)

We iterate through every element. For every element $\text{arr}[i]$, we find a pair with sum “ $-\text{arr}[i]$ ”. This problem reduces to pairs sum and can be solved in $O(n)$ time using hashing.

```
Run a loop from i=0 to n-2
    Create an empty hash table
    Run inner loop from j=i+1 to n-1
        If -(arr[i] + arr[j]) is present in hash table
            print arr[i], arr[j] and -(arr[i]+arr[j])
        Else
            Insert arr[j] in hash table.
```

C++

```
// C++ program to find triplets in a given
// array whose sum is equal to given sum.
#include <bits/stdc++.h>
using namespace std;

// function to print triplets with given sum
void findTriplets(int arr[], int n, int sum)
{
    for (int i = 0; i < n - 1; i++) {
        // Find all pairs with sum equals to
        // "sum-arr[i]"
        unordered_set<int> s;
        for (int j = i + 1; j < n; j++) {
            int x = sum - (arr[i] + arr[j]);
            if (s.find(x) != s.end())
                printf("%d %d %d\n", x, arr[i], arr[j]);
            else
                s.insert(arr[j]);
        }
    }
}

// Driver code
int main()
{
    int arr[] = { 0, -1, 2, -3, 1 };
    int sum = -2;
    int n = sizeof(arr) / sizeof(arr[0]);
    findTriplets(arr, n, sum);
    return 0;
}
```

Output:

```
-3 0 1
2 -1 -3
```

Time Complexity : $O(n^2)$

Auxiliary Space : $O(n)$

Method 3 (Sorting : $O(n^2)$)

The above method requires extra space. We can solve in $O(1)$ extra space. The idea is based on method 2 of [this](#) post.

1. Sort all element of array
2. Run loop from $i=0$ to $n-2$.
 Initialize two index variables $l=i+1$ and $r=n-1$
3. while ($l < r$)
 Check sum of $arr[i]$, $arr[l]$, $arr[r]$ is
 given sum or not if sum is 'sum', then print
 the triplet and do $l++$ and $r--$.
4. If sum is less than given sum then $l++$
5. If sum is greater than given sum then $r--$
6. If not exist in array then print not found.

C++

```
// C++ program to find triplets in a given
// array whose sum is given sum.
#include <bits/stdc++.h>
using namespace std;

// function to print triplets with given sum
void findTriplets(int arr[], int n, int sum)
{
    // sort array elements
    sort(arr, arr + n);

    for (int i = 0; i < n - 1; i++) {
        // initialize left and right
        int l = i + 1;
        int r = n - 1;
        int x = arr[i];
        while (l < r) {
            if (x + arr[l] + arr[r] == sum) {
                // print elements if it's sum is given sum.
                printf("%d %d %d\n", x, arr[l], arr[r]);
                l++;
                r--;
            }
        }

        // If sum of three elements is less
        // than 'sum' then increment in left
        else if (x + arr[l] + arr[r] < sum)
            l++;
    }
}
```

```
        l++;

        // if sum is greater than given sum, then
        // decrement in right side
        else
            r--;
    }
}

// Driver code
int main()
{
    int arr[] = { 0, -1, 2, -3, 1 };
    int sum = -2;
    int n = sizeof(arr) / sizeof(arr[0]);
    findTriplets(arr, n, sum);
    return 0;
}
```

Java

```
// Java program to find triplets
// in a given array whose sum
// is given sum.
import java.io.*;
import java.util.*;

class GFG
{

    // function to print
    // triplets with given sum
    static void findTriplets(int[] arr,
                            int n, int sum)
    {
        // sort array elements
        Arrays.sort(arr);

        for (int i = 0;
             i < n - 1; i++)
        {
            // initialize left and right
            int l = i + 1;
            int r = n - 1;
            int x = arr[i];
            while (l < r)
            {
```

```
if (x + arr[l] + arr[r] == sum)
{
    // print elements if it's
    // sum is given sum.
    System.out.println(x + " " + arr[l] +
                       " " + arr[r]);
    l++;
    r--;
}

// If sum of three elements
// is less than 'sum' then
// increment in left
else if (x + arr[l] +
          arr[r] < sum)
    l++;

// if sum is greater than
// given sum, then decrement
// in right side
else
    r--;
}
}

// Driver code
public static void main(String args[])
{
    int[] arr = new int[]{ 0, -1, 2, -3, 1 };
    int sum = -2;
    int n = arr.length;
    findTriplets(arr, n, sum);
}
}

// This code is contributed
// by Akanksha Rai(Addy_akku)
```

C#

```
// C# program to find triplets
// in a given array whose sum
// is given sum.
using System;

class GFG
{
```

```
// function to print
// triplets with given sum
static void findTriplets(int[] arr,
                        int n, int sum)
{
    // sort array elements
    Array.Sort(arr);

    for (int i = 0; i < n - 1; i++)
    {
        // initialize left and right
        int l = i + 1;
        int r = n - 1;
        int x = arr[i];
        while (l < r)
        {
            if (x + arr[l] + arr[r] == sum)
            {
                // print elements if it's
                // sum is given sum.
                Console.WriteLine(x + " " + arr[l] +
                                  " " + arr[r]);
                l++;
                r--;
            }

            // If sum of three elements
            // is less than 'sum' then
            // increment in left
            else if (x + arr[l] +
                      arr[r] < sum)
                l++;

            // if sum is greater than
            // given sum, then decrement
            // in right side
            else
                r--;
        }
    }
}

// Driver code
static int Main()
{
    int[] arr = new int[]{ 0, -1, 2, -3, 1 };
    int sum = -2;
```

```
int n = arr.Length;
findTriplets(arr, n, sum);
return 0;
}
}

// This code is contributed by rahul
```

PHP

```
<?php
// PHP program to find triplets
// in a given array whose sum
// is given sum.

// function to print triplets
// with given sum
function findTriplets($arr, $n, $sum)
{
    // sort array elements
    sort($arr);

    for ($i = 0; $i < $n - 1; $i++)
    {
        // initialize left and right
        $l = $i + 1;
        $r = $n - 1;
        $x = $arr[$i];
        while ($l < $r)
        {
            if ($x + $arr[$l] +
                $arr[$r] == $sum)
            {
                // print elements if it's
                // sum is given sum.
                echo $x, " ", $arr[$l],
                      " ", $arr[$r], "\n";
                $l++;
                $r--;
            }

            // If sum of three elements
            // is less than 'sum' then
            // increment in left
            else if ($x + $arr[$l] +
                     $arr[$r] < $sum)
                $l++;
        }
    }
}
```

```
// if sum is greater
// than given sum, then
// decrement in right side
else
    $r--;
}
}
}

// Driver code
$arr = array(0, -1, 2, -3, 1);
$sum = -2;
$n = sizeof($arr);
findTriplets($arr, $n, $sum);

// This code is contributed by ajit
?>
```

Output:

```
-3 -1 2
-3 0 1
```

Time Complexity : $O(n^2)$
Auxiliary Space : $O(1)$

Improved By : [jit_t](#), [Smitha Dinesh Semwal](#), [mithunkumarmnnit321](#), [Abby_akku](#)

Source

<https://www.geeksforgeeks.org/print-all-triplets-with-given-sum/>

Chapter 246

Print array of strings in sorted order without copying one string into another

Print array of strings in sorted order without copying one string into another - GeeksforGeeks

Given an array of n strings. The task is to print the strings in sorted order. The approach should be such that no string should be copied to another string during sorting process.

Examples:

Input : {"geeks", "for", "geeks", "quiz"}
Output : for geeks geeks quiz

Input : {"ball", "pen", "apple", "kite"}
Output : apple ball kite pen

Approach: It has the following steps:

1. Maintain another array **indexed_arr** which stores/maintains the index of each string.
2. We can apply any sorting technique to this **indexed_arr**.

An Illustration:

```
--> str[] = {"world", "hello"}  
--> corresponding index array will be  
     indexed_arr = {0, 1}  
--> Now, how the strings are compared and
```

```
accordingly values in indexed_arr are changed.  
--> Comparison process:  
    if (str[index[0]].compare(str[index[1]]) > 0  
        temp = index[0]  
        index[0] = index[1]  
        index[1] = temp  
  
    // after sorting values of  
    // indexed_arr = {1, 0}  
    --> for i=0 to 1  
        print str[index[i]]
```

This is how the strings are compared and their corresponding indexes in the indexed_arr are being manipulated/swapped so that after the sorting process is completed, the order of indexes in the indexed_arr gives us the sorted order of the strings.

```
// C++ implementation to print array of strings in sorted  
// order without copying one string into another  
#include <bits/stdc++.h>  
  
using namespace std;  
  
// function to print strings in sorted order  
void printInSortedOrder(string arr[], int n)  
{  
    int index[n];  
    int i, j, min;  
  
    // Initially the index of the strings  
    // are assigned to the 'index[]'  
    for (i=0; i<n; i++)  
        index[i] = i;  
  
    // selection sort technique is applied  
    for (i=0; i<n-1; i++)  
    {  
        min = i;  
        for (j=i+1; j<n; j++)  
        {  
            // with the help of 'index[]'  
            // strings are being compared  
            if (arr[index[min]].compare(arr[index[j]]) > 0)  
                min = j;  
        }  
  
        // index of the smallest string is placed
```

```
// at the ith index of 'index[]'  
if (min != i)  
{  
    int temp = index[min];  
    index[min] = index[i];  
    index[i] = temp;  
}  
}  
  
// printing strings in sorted order  
for (i=0; i<n; i++)  
    cout << arr[index[i]] << " ";  
}  
  
// Driver program to test above  
int main()  
{  
    string arr[] = {"geeks", "quiz", "geeks", "for"};  
    int n = 4;  
    printInSortedOrder(arr, n);  
    return 0;  
}
```

Output:

```
for geeks geeks quiz
```

Time Complexity: $O(n^2)$

The approach can have its usage when we have to minimize the number of **disc writes** as in the case of array of structures. The structure values are compared but their values are not being swapped, instead their index is maintained in another array, which is manipulated so as to keep the indexes in an order which represents the sorted array of structures.

Exercise: Apply this approach with the help of other sorting techniques like merge sort, insertion sort, etc.

Source

<https://www.geeksforgeeks.org/print-array-strings-sorted-order-without-copying-one-string-another/>

Chapter 247

Print n smallest elements from given array in their original order

Print n smallest elements from given array in their original order - GeeksforGeeks

We are given an array of m-elements, we need to find n smallest elements from the array but they must be in the same order as they are in given array.

Examples:

```
Input : arr[] = {4, 2, 6, 1, 5},  
        n = 3  
Output : 4 2 1  
Explanation :  
1, 2 and 4 are 3 smallest numbers and  
4 2 1 is their order in given array.
```

```
Input : arr[] = {4, 12, 16, 21, 25},  
        n = 3  
Output : 4 12 16  
Explanation :  
4, 12 and 16 are 3 smallest numbers and  
4 12 16 is their order in given array.
```

Make a copy of original array and then sort copy array. After sorting the copy array, save all n smallest numbers. Further for each element in original array, check whether it is in n-smallest number or not if it present in n-smallest array then print it otherwise move forward.

Make copy_arr[]
sort(copy_arr)
For all elements in arr[] -

- Find arr[i] in n-smallest element of copy_arr
- If found then print the element

Below is CPP implementation of above approach :

```
// CPP for printing smallest n number in order
#include <algorithm>
#include <iostream>
using namespace std;

// Function to print smallest n numbers
void printSmall(int arr[], int asize, int n)
{
    // Make copy of array
    vector<int> copy_arr(arr, arr + asize);

    // Sort copy array
    sort(copy_arr.begin(), copy_arr.begin() + asize);

    // For each arr[i] find whether
    // it is a part of n-smallest
    // with binary search
    for (int i = 0; i < asize; ++i)
        if (binary_search(copy_arr.begin(),
                           copy_arr.begin() + n, arr[i]))
            cout << arr[i] << " ";
}

// Driver program
int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int asize = sizeof(arr) / sizeof(arr[0]);
    int n = 5;
    printSmall(arr, asize, n);
    return 0;
}
```

Output :

1 3 4 2 0

For making a copy of array we need space complexity of $O(n)$ and then for sorting we will need complexity of order $O(n \log n)$. Further for each element in $\text{arr}[]$ we are performing searching in $\text{copy_arr}[]$, which will result $O(n)$ for linear search but we can improve it by applying binary search and hence our overall time complexity will be **$O(n \log n)$** .

Source

<https://www.geeksforgeeks.org/find-n-smallest-element-given-array-order-array/>

Chapter 248

Print number in ascending order which contains 1, 2 and 3 in their digits.

Print number in ascending order which contains 1, 2 and 3 in their digits. - GeeksforGeeks

Given an array of numbers, the task is to print those numbers in ascending order separated by commas which have 1, 2 and 3 in their digits. If no number containing digits 1, 2, and 3 present then print -1.

Examples:

Input : numbers[] = {123, 1232, 456, 234, 32145}
Output : 123, 1232, 32145

Input : numbers[] = {9821, 627183, 12, 1234}
Output : 1234, 627183

Input : numbers[] = {12, 232, 456, 234}
Output : -1

Asked in : Goldman Sachs

Approach: First finding all the number in from of array which contains **1, 2 & 3** then sort the number according to 1, 2 and 3 and then print it.

CPP

```
// CPP program to print all number containing  
// 1, 2 and 3 in any order.  
#include<bits/stdc++.h>  
using namespace std;
```

```
// convert the number to string and find
// if it contains 1, 2 & 3.
bool findContainsOneTwoThree(int number)
{
    string str = to_string(number);
    int countOnes = 0, countTwo = 0, countThree = 0;
    for(int i = 0; i < str.length(); i++) {
        if(str[i] == '1') countOnes++;
        else if(str[i] == '2') countTwo++;
        else if(str[i] == '3') countThree++;
    }
    return (countOnes && countTwo && countThree);
}
// prints all the number containing 1, 2, 3
string printNumbers(int numbers[], int n)
{
    vector<int> oneTwoThree;
    for (int i = 0; i < n; i++)
    {
        // check if the number contains 1,
        // 2 & 3 in any order
        if (findContainsOneTwoThree(numbers[i]))
            oneTwoThree.push_back(numbers[i]);
    }

    // sort all the numbers
    sort(oneTwoThree.begin(), oneTwoThree.end());

    string result = "";
    for(auto number: oneTwoThree)
    {
        int value = number;
        if (result.length() > 0)
            result += ", ";

        result += to_string(value);
    }

    return (result.length() > 0) ? result : "-1";
}

// Driver Code
int main() {
    int numbers[] = { 123, 1232, 456, 234, 32145 };
```

Chapter 248. Print number in ascending order which contains 1, 2 and 3 in their digits.

```
int n = sizeof(numbers)/sizeof(numbers[0]);  
  
string result = printNumbers(numbers, n);  
cout << result;  
return 0;  
}  
// This code is contributed  
// by Sirjan13
```

Java

```
// Java program to print all number containing  
// 1, 2 and 3 in any order.  
import java.io.FileNotFoundException;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Iterator;  
  
class GFG {  
  
    // prints all the number containing 1, 2, 3  
    // in any order  
    private static String printNumbers(int[] numbers)  
    {  
  
        ArrayList<Integer> array = new ArrayList<>();  
        for (int number : numbers) {  
  
            // check if the number contains 1,  
            // 2 & 3 in any order  
            if (findContainsOneTwoThree(number))  
                array.add(number);  
        }  
  
        // sort all the numbers  
        Collections.sort(array);  
  
        StringBuffer strbuf = new StringBuffer();  
        Iterator it = array.iterator();  
        while (it.hasNext()) {  
  
            int value = (int)it.next();  
            if (strbuf.length() > 0)  
                strbuf.append(", ");  
  
            strbuf.append(Integer.toString(value));  
        }  
    }
```

Chapter 248. Print number in ascending order which contains 1, 2 and 3 in their digits.

```
        return (strbuf.length() > 0) ?
                strbuf.toString() : "-1";
}

// convert the number to string and find
// if it contains 1, 2 & 3.
private static boolean findContainsOneTwoThree(
        int number)
{

    String str = Integer.toString(number);
    return (str.contains("1") && str.contains("2") &&
            str.contains("3"));
}

public static void main(String[] args)
{
    int[] numbers = { 123, 1232, 456, 234, 32145 };
    System.out.println(printNumbers(numbers));
}
}
```

Python

```
# Python program for printing
# all numbers containing 1,2 and 3

def printNumbers(numbers):

    # convert all numbers
    # to strings
    numbers = map(str, numbers)
    result = []
    for num in numbers:

        # check if each number
        # in the list has 1,2 and 3
        if ('1' in num and
            '2' in num and
            '3' in num):
            result.append(num)

    # if there are no
    # valid numbers
    if not result:
        result = ['-1']

    return sorted(result);
```

Chapter 248. Print number in ascending order which contains 1, 2 and 3 in their digits.

```
# Driver Code
numbers = [123, 1232, 456,
           234, 32145]
result = printNumbers(numbers)
print ', '.join(num for num in result)

# This code is contributed
# by IshitaTripathi
```

Output:

123, 1232, 32145

Time Complexity: Time complexity of the above approach is **O(n)**.

Improved By : [IshitaTripathi](#), [sirjan13](#)

Source

<https://www.geeksforgeeks.org/print-number-ascending-order-contains-1-2-3-digits/>

Chapter 249

Print sorted distinct elements of array

Print sorted distinct elements of array - GeeksforGeeks

Given an array that might contain duplicates, print all distinct elements in sorted order.

Examples:

```
Input  : 1, 3, 2, 2, 1
Output : 1 2 3
```

```
Input  : 1, 1, 1, 2, 2, 3
Output : 1 2 3
```

Simple Solution is to sort the array first, then traverse the array and print only first occurrences of elements.

Another Approach is to use [set in C++ STL](#).
C++

```
// CPP program to print sorted distinct
// elements.
#include <bits/stdc++.h>
using namespace std;

void printRepeating(int arr[], int size)
{
    // Create a set using array elements
    set<int> s(arr, arr + size);

    // Print contents of the set.
```

```
    for (auto x : s)
        cout << x << " ";
}

// Driver code
int main()
{
    int arr[] = { 1, 3, 2, 2, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printRepeating(arr, n);
    return 0;
}
```

Java

```
// Java program to print sorted distinct
// elements.
import java.io.*;
import java.util.*;

public class GFG {

    static void printRepeating(Integer []arr, int size)
    {
        // Create a set using array elements
        SortedSet<Integer> s = new TreeSet<>();
        Collections.addAll(s, arr);

        // Print contents of the set.
        System.out.print(s);
    }

    // Driver code
    public static void main(String args[])
    {
        Integer []arr = {1, 3, 2, 2, 1};
        int n = arr.length;
        printRepeating(arr, n);
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

C#

```
// C# program to print sorted distinct
```

```
// elements.
using System;
using System.Collections.Generic;
using System.Linq;

class GFG {

    static void printRepeating(int []arr, int size)
    {
        // Create a set using array elements
        SortedSet<int> s = new SortedSet<int>(arr);

        // Print contents of the set.
        foreach (var n in s)
        {
            Console.Write(n + " ");
        }
    }

    // Driver code
    public static void Main()
    {
        int []arr = {1, 3, 2, 2, 1};
        int n = arr.Length;
        printRepeating(arr, n);
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

Output:

1 2 3

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/print-sorted-distinct-elements-array-c/>

Chapter 250

Print triplets with sum less than or equal to k

Print triplets with sum less than or equal to k - GeeksforGeeks

Given an array of distinct integers and a sum value. Print all triplets with sum smaller than given sum value. Expected Time Complexity is $O(n^2)$.

Examples:

```
Input : arr[] = {-2, 0, 1, 3}
        sum = 2.
Output : (-2, 0, 1)
        (-2, 0, 3)
Explanation : The two triplets have sum less than
or equal to 2.
```

```
Input : arr[] = {5, 1, 3, 4, 7}
        sum = 12.
Output : (1, 3, 4)
        (1, 3, 5)
        (1, 3, 7)
        (1, 4, 5)
```

A **Simple Solution** is to run three loops to consider all triplets one by one. For every triplet, compare the sums and print current triplet if its sum is smaller than given sum.

C++

```
// A Simple C++ program to count triplets with sum
// smaller than a given value
#include<bits/stdc++.h>
using namespace std;
```

```
int printTriplets(int arr[], int n, int sum)
{
    // Fix the first element as A[i]
    for (int i = 0; i < n-2; i++)
    {
        // Fix the second element as A[j]
        for (int j = i+1; j < n-1; j++)
        {
            // Now look for the third number
            for (int k = j+1; k < n; k++)
                if (arr[i] + arr[j] + arr[k] < sum)
                    cout << arr[i] << ", " << arr[j]
                        << ", " << arr[k] << endl;
        }
    }
}

// Driver program
int main()
{
    int arr[] = {5, 1, 3, 4, 7};
    int n = sizeof arr / sizeof arr[0];
    int sum = 12;
    printTriplets(arr, n, sum);
    return 0;
}
```

Java

```
// A Simple Java program to
// count triplets with sum
// smaller than a given value
import java.io.*;

class GFG
{
static int printTriplets(int arr[],
                        int n, int sum)
{
    // Fix the first
    // element as A[i]
    for (int i = 0; i < n - 2; i++)
    {

        // Fix the second
        // element as A[j]
        for (int j = i + 1;
```

```
j < n - 1; j++)
{
    // Now look for
    // the third number
    for (int k = j + 1; k < n; k++)
        if (arr[i] + arr[j] + arr[k] < sum)
            System.out.println(arr[i] + ", " +
                               arr[j] + ", " +
                               arr[k]);
}
}
return 0;
}

// Driver Code
public static void main (String[] args)
{
    int arr[] = {5, 1, 3, 4, 7};
    int n = arr.length;
    int sum = 12;
    printTriplets(arr, n, sum);
}
}

// This code is contributed
// by anuj_67.
```

Output:

```
5, 1, 3
5, 1, 4
1, 3, 4
1, 3, 7
```

Time complexity of above solution is $O(n^3)$.

An **Efficient Solution** can print triplets in $O(n^2)$ by sorting the array first, and then using method 1 of [this](#) post in a loop.

- 1) Sort the input array in increasing order.
- 2) Initialize result as 0.
- 3) Run a loop from $i = 0$ to $n-2$. An iteration of this loop finds all triplets with $arr[i]$ as first element.
 - a) Initialize other two elements as corner elements

```
of subarray
arr[i+1..n-1], i.e., j = i+1 and k = n-1
b) Move j and k toward each other until they meet,
i.e., while (j = sum), then do k--
// Else for current i and j, there are (k-j) possible
// third elements that satisfy the constraint.
(ii) Else print elements from j to k
```

Below is the implementation of above idea.

C++

```
// C++ program to print triplets with sum smaller
// than a given value
#include <bits/stdc++.h>
using namespace std;

int printTriplets(int arr[], int n, int sum)
{
    // Sort input array
    sort(arr, arr + n);

    // Every iteration of loop counts triplet with
    // first element as arr[i].
    for (int i = 0; i < n - 2; i++) {

        // Initialize other two elements as corner
        // elements of subarray arr[j+1..k]
        int j = i + 1, k = n - 1;

        // Use Meet in the Middle concept
        while (j < k) {

            // If sum of current triplet is more or equal,
            // move right corner to look for smaller values
            if (arr[i] + arr[j] + arr[k] >= sum)
                k--;

            // Else move left corner
            else {

                // This is important. For current i and j,
                // there are total k-j third elements.
                for (int x = j + 1; x <= k; x++)
                    cout << arr[i] << ", " << arr[j]
                    << ", " << arr[x] << endl;
                j++;
            }
        }
    }
}
```

```
        }
    }
}

// Driver program
int main()
{
    int arr[] = { 5, 1, 3, 4, 7 };
    int n = sizeof arr / sizeof arr[0];
    int sum = 12;
    printTriplets(arr, n, sum);
    return 0;
}
```

Java

```
// Java program to print
// triplets with sum smaller
// than a given value
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{
static void printTriplets(int arr[],
                         int n, int sum)
{
    // Sort input array
    Arrays.sort(arr);

    // Every iteration of loop
    // counts triplet with
    // first element as arr[i].
    for (int i = 0; i < n - 2; i++)
    {

        // Initialize other two elements
        // as corner elements of subarray
        // arr[j+1..k]
        int j = i + 1, k = n - 1;

        // Use Meet in the
        // Middle concept
        while (j < k)
        {

            // If sum of current triplet
```

```
// is more or equal, move right
// corner to look for smaller values
if (arr[i] + arr[j] + arr[k] >= sum)
    k--;

// Else move left corner
else
{

    // This is important. For
    // current i and j, there
    // are total k-j third elements.
    for (int x = j + 1; x <= k; x++)
        System.out.println(arr[i] + ", " +
                           arr[j] + ", " +
                           arr[x]);

    j++;
}
}

// Driver Code
public static void main(String args[])
{
    int arr[] = { 5, 1, 3, 4, 7 };
    int n = arr.length;
    int sum = 12;
    printTriplets(arr, n, sum);
}
}

// This code is contributed
// by Subhadeep
```

Output:

```
1, 3, 4
1, 3, 5
1, 3, 7
1, 4, 5
```

Improved By : [vt_m](#), [tufan_gupta2000](#)

Source

<https://www.geeksforgeeks.org/print-triplets-with-sum-less-than-or-equal-to-k/>

Chapter 251

Printing frequency of each character just after its consecutive occurrences

Printing frequency of each character just after its consecutive occurrences - GeeksforGeeks

Given a string in such a way that every character occurs in a repeated manner. Your task is to print the string by inserting the frequency of each unique character after it and also eliminating all repeated characters.

Examples:

Input : GeeeEEKKKss
Output : G1e3E2K3s2

Input : cccc0ddEEE
Output : c4o1d2E3

One approach to solve the above problem is to start a loop till the end of the string and for every iteration, increment a count till the character at i^{th} position matches the following character.

Below is the implementation to the above given problem.

C++

```
// CPP program to print run
// length encoding of a string
#include <iostream>
using namespace std;

void printRLE(string s)
```

```
{  
    for (int i = 0; s[i] != '\0'; i++)  
    {  
  
        // Counting occurrences of s[i]  
        int count = 1;  
        while (s[i] == s[i + 1])  
        {  
            i++;  
            count++;  
        }  
        cout << s[i] << count << " ";  
    }  
  
    cout << endl;  
}  
  
// Driver code  
int main()  
{  
    printRLE("GeeeEEKKKss");  
    printRLE("cccc0ddEEE");  
    return 0;  
}
```

Java

```
// Java program to print run  
// length encoding of a string  
class GFG  
{  
  
    static void printRLE(String s)  
{  
        for (int i = 0;  
             i < s.length() - 1; i++)  
        {  
  
            // Counting occurrences of s[i]  
            int count = 1;  
            while (s.charAt(i) == s.charAt(i + 1))  
            {  
                i++;  
                count++;  
                if(i + 1 == s.length())  
                    break;  
            }  
            System.out.print(s.charAt(i) + "" +
```

```
        count + " ");
    }

    System.out.println();
}

// Driver code
public static void main(String args[])
{
    printRLE("GeeeEEKKKss");
    printRLE("cccc0ddEEE");
}
}

// This code is contributed
// by Arnab Kundu
```

Python 3

```
# Python 3 program to print run
# length encoding of a string

def printRLE(s) :

    i = 0
    while( i < len(s) - 1) :

        # Counting occurrences of s[i]
        count = 1

        while s[i] == s[i+1] :

            i += 1
            count += 1

            if i + 1 == len(s):
                break

        print(str(s[i]) + str(count),
              end = " ")
        i += 1

    print()

# Driver Code
if __name__ == "__main__":
    # function calling
```

Chapter 251. Printing frequency of each character just after its consecutive occurrences

```
printRLE("GeeeEEKKKss")
printRLE("cccc0ddEEE")

# This code is contributed by ANKITRAI1
```

Output:

```
G1 e3 E2 K3 s2
c4 O1 d2 E3
```

Improved By : [andrew1234](#), [ANKITRAI1](#)

Source

<https://www.geeksforgeeks.org/printing-frequency-of-each-character-just-after-its-consecutive-occurrences/>

Chapter 252

Program for sorting variables of any data type

Program for sorting variables of any data type - GeeksforGeeks

Write a program for sorting variables of any datatype without the use of std::sort .

Examples:

Input : 2000, 456, -10, 0
Output : -10 0 456 2000

Input : "We do nothing"
"Hi I have something"
"Hello Join something!"
"(Why to do work)"
Output :(Why to do work)
Hello Join something!
Hi I have something
We do nothing

The examples above show, we can have any data type elements present as an input and output will be in a sorted form of the input data.

The idea here to solve this problem is to make a [template](#).

Method 1 (Writing our own sort) In below code, we have implemented [Bubble Sort](#)to sort the array.

```
// CPP program to sort array of any data types.  
#include <bits/stdc++.h>  
using namespace std;
```

```
// Template formed so that sorting of any
// type variable is possible
template <class T>
void sortArray(T a[], int n)
{
    // boolean variable to check that
    // whether it is sorted or not
    bool b = true;
    while (b) {
        b = false;
        for (size_t i=0; i<n-1; i++) {

            // swapping the variable
            // for sorting order
            if (a[i] > a[i + 1]) {
                T temp = a[i];
                a[i] = a[i + 1];
                a[i + 1] = temp;
                b = true;
            }
        }
    }
}

// Template formed so that sorting of any
// type variable is possible
template <class T>
void printArray(T a[], int n)
{
    for (size_t i = 0; i < n; ++i)
        cout << a[i] << "    ";
    cout << endl;
}

// Driver code
int main()
{
    int n = 4;
    int intArr[n] = { 2000, 456, -10, 0 };
    sortArray(intArr, n);
    printArray(intArr, n);

    string strArr[n] = { "We do nothing",
                        "Hi I have something",
                        "Hello Join something!",
                        "(Why to do work)" };
    sortArray(strArr, n);
    printArray(strArr, n);
```

```
float floatArr[n] = { 23.4, 11.4, -9.7, 11.17 };
sortArray(floatArr, n);
printArray(floatArr, n);

return 0;
}
```

Output:

```
-10 0 456 2000
(Why to do work) Hello Join something! Hi I have something We do nothing
-9.7 11.17 11.4 23.4
```

Method 2 (Using Library Function)

We can use `std::sort` in C++ to sort array of any data type.

```
// CPP program to sort array of any data types.
#include <bits/stdc++.h>
using namespace std;

// Template formed so that sorting of any
// type variable is possible
template <class T>
void printArray(T a[], int n)
{
    for (size_t i = 0; i < n; ++i)
        cout << a[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int n = 4;
    int intArr[n] = { 2000, 456, -10, 0 };
    sort(intArr, intArr + n);
    printArray(intArr, n);

    string strArr[n] = { "We do nothing",
                        "Hi I have something",
                        "Hello Join something!",
                        "(Why to do work)" };
    sort(strArr, strArr + n);
    printArray(strArr, n);
}
```

```
float floatArr[n] = { 23.4, 11.4, -9.7, 11.17 };
sort(floatArr, floatArr+n);
printArray(floatArr, n);

return 0;
}
```

Output:

```
-10    0    456    2000
(Why to do work)  Hello Join something!  Hi I have something  We do nothing
-9.7    11.17    11.4    23.4
```

Source

<https://www.geeksforgeeks.org/program-sorting-variables-data-type/>

Chapter 253

Program to check if an array is sorted or not (Iterative and Recursive)

Program to check if an array is sorted or not (Iterative and Recursive) - GeeksforGeeks

Given an array of size **n**, write a program to check if it is sorted in ascending order or not. Equal values are allowed in array and two consecutive equal values are considered sorted.

Examples:

Input : 20 21 45 89 89 90
Output : Yes

Input : 20 20 45 89 89 90
Output : Yes

Input : 20 20 78 98 99 97
Output : No

Recursive approach:

The basic idea for recursive approach:

- 1: If size of array is zero or one, return true.
 - 2: Check last two elements of array, if they are sorted, perform a recursive call with $n-1$ else, return false.
- If all the elements will be found sorted, n will eventually fall to one, satisfying Step 1.

Below is the implementation using recursion:

Source

<https://www.geeksforgeeks.org/program-check-array-sorted-not-iterative-recursive/>

C++

```
// Recursive approach to check if an
// Array is sorted or not
#include<bits/stdc++.h>
using namespace std;

// Function that returns 0 if a pair
// is found unsorted
int arraySortedOrNot(int arr[], int n)
{
    // Array has one or no element or the
    // rest are already checked and approved.
    if (n == 1 || n == 0)
        return 1;

    // Unsorted pair found (Equal values allowed)
    if (arr[n-1] < arr[n-2])
        return 0;

    // Last pair was sorted
    // Keep on checking
    return arraySortedOrNot(arr, n-1);
}

// Driver code
int main()
{
    int arr[] = {20, 23, 23, 45, 78, 88};
    int n = sizeof(arr)/sizeof(arr[0]);
    if (arraySortedOrNot(arr, n))
        cout << "Yes\n";
    else
        cout << "No\n";
}
```

Java

```
// Recursive approach to check if an
// Array is sorted or not
```

```
class CkeckSorted
{
    // Function that returns 0 if a pair
    // is found unsorted
    static int arraySortedOrNot(int arr[], int n)
    {
        // Array has one or no element or the
        // rest are already checked and approved.
        if (n == 1 || n == 0)
            return 1;

        // Unsorted pair found (Equal values allowed)
        if (arr[n-1] < arr[n-2])
            return 0;

        // Last pair was sorted
        // Keep on checking
        return arraySortedOrNot(arr, n-1);
    }

    // main function
    public static void main (String[] args)
    {
        int arr[] = {20, 23, 23, 45, 78, 88};
        int n = arr.length;
        if (arraySortedOrNot(arr, n)!=0)
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

Python3

```
# Recursive approach to check if an
# Array is sorted or not

# Function that returns 0 if a pair
# is found unsorted
def arraySortedOrNot(arr):

    # Calculating length
    n = len(arr)

    # Array has one or no element or the
    # rest are already checked and approved.
    if n == 1 or n == 0:
        return True
```

```
# Recursion applied till last element
return arr[0]<=arr[1] and arraySortedOrNot(arr[1:])

arr = [20, 23, 23, 45, 78, 88]

# Displaying result
if arraySortedOrNot(arr): print("Yes")
else: print("No")
```

Output:

Yes

Time Complexity : O(n)

Auxiliary Space : O(n) for Recursion Call Stack.

Iterative approach:

The idea is pretty much the same. The benefit of iterative approach is it avoids the usage of recursion stack space and recursion overhead.

Below is the implementation using iteration:

C++

```
// C++ program to check if an
// Array is sorted or not
#include<bits/stdc++.h>
using namespace std;

// Function that returns true if array is
// sorted in non-decreasing order.
bool arraySortedOrNot(int arr[], int n)
{
    // Array has one or no element
    if (n == 0 || n == 1)
        return true;

    for (int i = 1; i < n; i++)

        // Unsorted pair found
        if (arr[i-1] > arr[i])
            return false;

    // No unsorted pair found
    return true;
}
```

```
// Driver code
int main()
{
    int arr[] = {20, 23, 23, 45, 78, 88};
    int n = sizeof(arr)/sizeof(arr[0]);
    if (arraySortedOrNot(arr, n))
        cout << "Yes\n";
    else
        cout << "No\n";
}
```

Java

```
// Recursive approach to check if an
// Array is sorted or not
class GFG {

    // Function that returns true if array is
    // sorted in non-decreasing order.
    static boolean arraySortedOrNot(int arr[], int n)
    {

        // Array has one or no element
        if (n == 0 || n == 1)
            return true;

        for (int i = 1; i < n; i++)

            // Unsorted pair found
            if (arr[i-1] > arr[i])
                return false;

        // No unsorted pair found
        return true;
    }

    //driver code
    public static void main (String[] args)
    {

        int arr[] = {20, 23, 23, 45, 78, 88};
        int n = arr.length;

        if (arraySortedOrNot(arr, n))
            System.out.print("Yes\n");
        else
            System.out.print("No\n");
    }
}
```

```
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program to check if an
# Array is sorted or not

# Function that returns true if array is
# sorted in non-decreasing order.
def arraySortedOrNot(arr, n):

    # Array has one or no element
    if (n == 0 or n == 1):
        return True

    for i in range(1, n):

        # Unsorted pair found
        if (arr[i-1] > arr[i]):
            return False

    # No unsorted pair found
    return True

# Driver code
arr = [20, 23, 23, 45, 78, 88]
n = len(arr)
if (arraySortedOrNot(arr, n)):
    print("Yes")
else:
    print("No")

# This code is contributed by Anant Agarwal.
```

Output:

Yes

Time Complexity : O(n)
Auxiliary Space : O(1)

Chapter 254

Program to print an array in Pendulum Arrangement

Program to print an array in Pendulum Arrangement - GeeksforGeeks

Write a program to input a list of integers in an array and arrange them in a way similar to the to-and-fro movement of a Pendulum.

- The minimum element out of the list of integers, must come in center position of array.
- The number in the ascending order next to the minimum, goes to the right, the next higher number goes to the left of minimum number and it continues.
- As higher numbers are reached, one goes to one side in a to-and-fro manner similar to that of a Pendulum.

Examples:

Input : 1 3 2 5 4
Output : 5 3 1 2 4

Explanation:

The minimum element is 1, so it is moved to the middle.
The next higher element 2 is moved to the right of the middle element while the next higher element 3 is moved to the left of the middle element and this process is continued.

Input : 11 12 31 14 5
Output : 31 12 5 11 14

The idea is to sort the array first. Once the array is sorted, use an auxiliary array to store elements one by one.

C++

```
// C++ program for pendulum arrangement of numbers
#include <bits/stdc++.h>
using namespace std;

// Prints pendulam arrangement of arr[]
void pendulumArrangement(int arr[], int n)
{
    // sorting the elements
    sort(arr, arr+n);

    // Auxiliary array to store output
    int op[n];

    // calculating the middle index
    int mid = (n-1)/2;

    // storing the minimum element in the middle
    // i is index for output array and j is for
    // input array.
    int j = 1, i = 1;
    op[mid] = arr[0];
    for (i = 1; i <= mid; i++)
    {
        op[mid+i] = arr[j++];
        op[mid-i] = arr[j++];
    }

    // adjustment for when no. of elements is even
    if (n%2 == 0)
        op[mid+i] = arr[j];

    // Printing the pendulum arrangement
    cout << "Pendulum arrangement:" << endl;
    for (i = 0 ; i < n; i++)
        cout << op[i] << " ";

    cout << endl;
}

// Driver function
int main()
{
    //input Array
    int arr[] = {14, 6, 19, 21, 12};

    // calculating the length of array A
    int n = sizeof(arr)/sizeof(arr[0]);
}
```

```
// calling pendulum function
pendulumArrangement(arr, n);

return 0;
}

Java

// Java program for pendulum arrangement of numbers

import java.util.Arrays;

class Test
{
    // Prints pendulum arrangement of arr[]
    static void pendulumArrangement(int arr[], int n)
    {
        // sorting the elements
        Arrays.sort(arr);

        // Auxiliary array to store output
        int op[] = new int[n];

        // calculating the middle index
        int mid = (n-1)/2;

        // storing the minimum element in the middle
        // i is index for output array and j is for
        // input array.
        int j = 1, i = 1;
        op[mid] = arr[0];
        for (i = 1; i <= mid; i++)
        {
            op[mid+i] = arr[j++];
            op[mid-i] = arr[j++];
        }

        // adjustment for when no. of elements is even
        if (n%2 == 0)
            op[mid+i] = arr[j];

        // Printing the pendulum arrangement
        System.out.println("Pendulum arrangement:");
        for (i = 0 ; i < n; i++)
            System.out.print(op[i] + " ");
    }
}
```

```
        System.out.println();
    }

// Driver method
public static void main(String[] args)
{
    //input Array
    int arr[] = {14, 6, 19, 21, 12};

    // calling pendulum function
    pendulumArrangement(arr, arr.length);
}
}
```

Python3

```
# Python 3 program for pendulum
# arrangement of numbers

# Prints pendulam arrangement of arr[]
def pendulumArrangement(arr, n):

    # sorting the elements
    arr.sort()

    # Auxiliary array to store output
    op = [0] * n

    # calculating the middle index
    mid = int((n-1)/2)

    # storing the minimum
    # element in the middle
    # i is index for output
    # array and j is for
    # input array.
    j = 1
    i = 1
    op[mid] = arr[0]
    for i in range(1,mid+1):

        op[mid+i] = arr[j]
        j+=1
        op[mid-i] = arr[j]
        j+=1

    # adjustment for when no.
```

```
# of elements is even
if (int(n/2) == 0):
    op[mid+i] = arr[j]

# Printing the pendulum arrangement
print("Pendulum arrangement:")
for i in range(0,n):
    print(op[i],end=" ")

# Driver function
# input Array
arr = [14, 6, 19, 21, 12]

# calculating the length of array A
n = len(arr)

# calling pendulum function
pendulumArrangement(arr, n)

# This code is contributed by
# Smitha Dinesh Semwal

C#
// C# program for pendulum
// arrangement of numbers
using System;

class Test {

    // Prints pendulum arrangement of arr[]
    static void pendulumArrangement(int []arr,
                                      int n)
    {
        // sorting the elements
        Array.Sort(arr);

        // Auxiliary array to store output
        int []op = new int[n];

        // calculating the middle index
        int mid = (n - 1) / 2;

        // storing the minimum element in
        // the middle i is index for output
        // array and j is for input array.
```

```
int j = 1, i = 1;
op[mid] = arr[0];
for (i = 1; i <= mid; i++)
{
    op[mid + i] = arr[j++];
    op[mid - i] = arr[j++];
}

// adjustment for when no.
// of elements is even
if (n % 2 == 0)
    op[mid + i] = arr[j];

// Printing the pendulum arrangement
Console.WriteLine("Pendulum arrangement:");
for (i = 0 ; i < n; i++)
    Console.Write(op[i] + " ");

Console.WriteLine();
}

// Driver code
public static void Main()
{

    //input Array
    int []arr = {14, 6, 19, 21, 12};

    // calling pendulum function
    pendulumArrangement(arr, arr.Length);
}
}

// This code is contributed by Nitin Mittal.
```

PHP

```
<?php
// PHP program for pendulum
// arrangement of numbers

// Prints pendulam arrangement of arr[]
function pendulumArrangement($arr, $n)
{

    // sorting the elements
    sort($arr, $n);
```

```
sort($arr);

// Auxiliary array to
// store output
$op[$n] = NULL;

// calculating the
// middle index
$mid = floor(($n - 1) / 2);

// storing the minimum
// element in the middle
// i is index for output
// array and j is for
// input array.
$j = 1;
$i = 1;
$op[$mid] = $arr[0];
for ($i = 1; $i <= $mid; $i++)
{
    $op[$mid + $i] = $arr[$j++];
    $op[$mid - $i] = $arr[$j++];
}

// adjustment for when no.
// of elements is even
if ($n % 2 == 0)
    $op[$mid + $i] = $arr[$j];

// Printing the pendulum
// arrangement
echo "Pendulum arrangement:" ;
for ($i = 0 ; $i < $n; $i++)
    echo $op[$i], " ";

echo "\n";
}

// Driver Code
//input Array
$arr = array(14, 6, 19, 21, 12);

// calculating the length
// of array A
$n = sizeof($arr);

// calling pendulum function
```

```
pendulumArrangement($arr, $n);  
// This code is contributed by nitin mittal.  
?>
```

Output:

```
Pendulum arrangement:  
21 14 6 12 19
```

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/program-print-array-pendulum-arrangement/>

Chapter 255

Program to sort string in descending order

Program to sort string in descending order - GeeksforGeeks

Given a string, sort it in descending order.

Examples:

Input : alkasingh
Output : snlkihgaa

Input : nupursingh
Output : uusrpnnihg

Input : geeksforgeeks
Output : ssrokkggfeeee

A **simple solution** is to use library sort function `std::sort()`

C++

```
// CPP program to sort a string in descending
// order using library function
#include <bits/stdc++.h>
using namespace std;

void descOrder(string s)
{
    sort(s.begin(), s.end(), greater<char>());
}
```

```
int main()
{
    string s = "geeksforgeeks";
    descOrder(s); // function call
    return 0;
}
```

Python

```
# Python program to sort
# a string in descending
# order using library function

def descOrder(s):
    s.sort(reverse = True)
    str1 = ''.join(s)
    print(str1)

def main():
    s = list('geeksforgeeks')

    # function call
    descOrder(s)

if __name__ == "__main__":
    main()

# This code is contributed by
# prabhat kumar singh
```

Output:

```
ssrokkggfeeee
```

The time complexity is : $O(n \log n)$

An **efficient approach** will be to observe first that there can be a total of 26 unique characters only. So, we can store the count of occurrences of all the characters from 'a' to 'z' in a hashed array. The first index of the hashed array will represent character 'a', second will represent 'b' and so on. Finally, we will simply traverse the hashed array and print the characters from 'z' to 'a' the number of times they occurred in input string.

Below is the implementation of above idea:

C++

```
// C++ program to sort a string of characters
```

```
// in descending order
#include <bits/stdc++.h>
using namespace std;

const int MAX_CHAR = 26;

// function to print string in sorted order
void sortString(string& str)
{
    // Hash array to keep count of characters.
    // Initially count of all characters is
    // initialized to zero.
    int charCount[MAX_CHAR] = { 0 };

    // Traverse string and increment
    // count of characters
    for (int i = 0; i < str.length(); i++)

        // 'a'-'a' will be 0, 'b'-'a' will be 1,
        // so for location of character in count
        // array we will do str[i]-'a'.
        charCount[str[i] - 'a']++;

    // Traverse the hash array and print
    // characters
    for (int i = MAX_CHAR - 1; i >= 0; i--)
        for (int j = 0; j < charCount[i]; j++)
            cout << (char)('a' + i);
}

// Driver program to test above function
int main()
{
    string s = "alkasingh";
    sortString(s);
    return 0;
}
```

Output:

```
snlkihgaa
```

Time Complexity: $O(n)$, where n is the length of input string.
Auxiliary Space: $O(1)$.

Improved By : [prabhat kumar singh](#)

Source

<https://www.geeksforgeeks.org/program-sort-string-descending-order/>

Chapter 256

Python Code for time Complexity plot of Heap Sort

Python Code for time Complexity plot of Heap Sort - GeeksforGeeks

Prerequisite : [HeapSort](#)

Heap sort is a comparison based sorting technique based on [Binary Heap](#) data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

We implement Heap Sort here, call it for different sized random lists, measure time taken for different sizes and generate a plot of input size vs time taken.

```
# Python Code for Implementation and running time Algorithm
# Complexity plot of Heap Sort
# by Ashok Kajal
# This python code intends to implement Heap Sort Algorithm
# Plots its time Complexity on list of different sizes

# -----Important Note -----
# numpy, time and matplotlib.pyplot are required to run this code
import time
from numpy.random import seed
from numpy.random import randint
import matplotlib.pyplot as plt

# find left child of node i
def left(i):
    return 2 * i + 1

# find right child of node i
def right(i):
```

```
return 2 * i + 2

# calculate and return array size
def heapSize(A):
    return len(A)-1

# This function takes an array and Heapsifies
# the at node i
def MaxHeapify(A, i):
    # print("in heapify", i)
    l = left(i)
    r = right(i)

    # heapSize = len(A)
    # print("left", l, "Rightt", r, "Size", heapSize)
    if l<= heapSize(A) and A[l] > A[i] :
        largest = l
    else:
        largest = i
    if r<= heapSize(A) and A[r] > A[largest]:
        largest = r
    if largest != i:
        # print("Largest", largest)
        A[i], A[largest]= A[largest], A[i]
        # print("List", A)
        MaxHeapify(A, largest)

# this function makes a heapified array
def BuildMaxHeap(A):
    for i in range(int(heapSize(A)/2)-1, -1, -1):
        MaxHeapify(A, i)

# Sorting is done using heap of array
def HeapSort(A):
    BuildMaxHeap(A)
    B = list()
    heapSize1 = heapSize(A)
    for i in range(heapSize(A), 0, -1):
        A[0], A[i]= A[i], A[0]
        B.append(A[heapSize1])
        A = A[:-1]
        heapSize1 = heapSize1-1
        MaxHeapify(A, 0)

# randomly generates list of different
# sizes and call HeapSort function
```

```
elements = list()
times = list()
for i in range(1, 10):

    # generate some integers
    a = randint(0, 1000 * i, 1000 * i)
    # print(i)
    start = time.clock()
    HeapSort(a)
    end = time.clock()

    # print("Sorted list is ", a)
    print(len(a), "Elements Sorted by HeapSort in ", end-start)
    elements.append(len(a))
    times.append(end-start)

plt.xlabel('List Length')
plt.ylabel('Time Complexity')
plt.plot(elements, times, label ='Heap Sort')
plt.grid()
plt.legend()
plt.show()
# This code is contributed by Ashok Kajal
```

Output :

```
Input : Unsorted Lists of Different sizes are Generated Randomly
Output :
1000 Elements Sorted by HeapSort in  0.023797415087301488
2000 Elements Sorted by HeapSort in  0.053856713614550245
3000 Elements Sorted by HeapSort in  0.08474737185133563
4000 Elements Sorted by HeapSort in  0.13578669978414837
5000 Elements Sorted by HeapSort in  0.1658182863213824
6000 Elements Sorted by HeapSort in  0.1875901601906662
7000 Elements Sorted by HeapSort in  0.21982946862249264
8000 Elements Sorted by HeapSort in  0.2724293921580738
9000 Elements Sorted by HeapSort in  0.30996323029421546
```

Complexity PLot for Heap Sort is Given Below

Source

<https://www.geeksforgeeks.org/python-code-for-time-complexity-plot-of-heap-sort/>

Chapter 257

Python Program for Binary Insertion Sort

Python Program for Binary Insertion Sort - GeeksforGeeks

We can use binary search to reduce the number of comparisons in [normal insertion sort](#).
Binary Insertion Sort find use binary search to find the proper location to insert the selected item at each iteration.
In normal insertion, sort it takes $O(i)$ (at ith iteration) in worst case. we can reduce it to $O(\log i)$ by using [binary search](#).

Source

<https://www.geeksforgeeks.org/python-program-for-binary-insertion-sort/>

Python

```
# Python Program implementation  
# of binary insertion sort  
  
def binary_search(arr, val, start, end):  
    # we need to distinguish whether we should insert  
    # before or after the left boundary.  
    # imagine [0] is the last step of the binary search  
    # and we need to decide where to insert -1  
    if start == end:  
        if arr[start] > val:  
            return start  
        else:  
            return start+1
```

```
# this occurs if we are moving beyond left\'s boundary
# meaning the left boundary is the least position to
# find a number greater than val
if start > end:
    return start

mid = (start+end)/2
if arr[mid] < val:
    return binary_search(arr, val, mid+1, end)
elif arr[mid] > val:
    return binary_search(arr, val, start, mid-1)
else:
    return mid

def insertion_sort(arr):
    for i in xrange(1, len(arr)):
        val = arr[i]
        j = binary_search(arr, val, 0, i-1)
        arr = arr[:j] + [val] + arr[j:i] + arr[i+1:]
    return arr

print("Sorted array:")
print insertion_sort([37, 23, 0, 17, 12, 72, 31,
                     46, 100, 88, 54])

# Code contributed by Mohit Gupta_OMG
```

Please refer complete article on [Binary Insertion Sort](#) for more details!

Chapter 258

Python Program for Bubble Sort

Python Program for Bubble Sort - GeeksforGeeks

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Source

<https://www.geeksforgeeks.org/python-program-for-bubble-sort/>

Python

```
# Python program for implementation of Bubble Sort

def bubbleSort(arr):
    n = len(arr)

    # Traverse through all array elements
    for i in range(n):

        # Last i elements are already in place
        for j in range(0, n-i-1):

            # traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

```
# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print ("Sorted array is:")
for i in range(len(arr)):
    print ("%d" %arr[i]),
```

Please refer complete article on [Bubble Sort](#) for more details!

Chapter 259

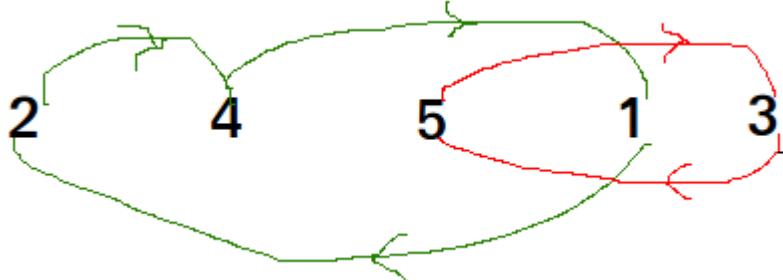
Python Program for Cycle Sort

Python Program for Cycle Sort - GeeksforGeeks

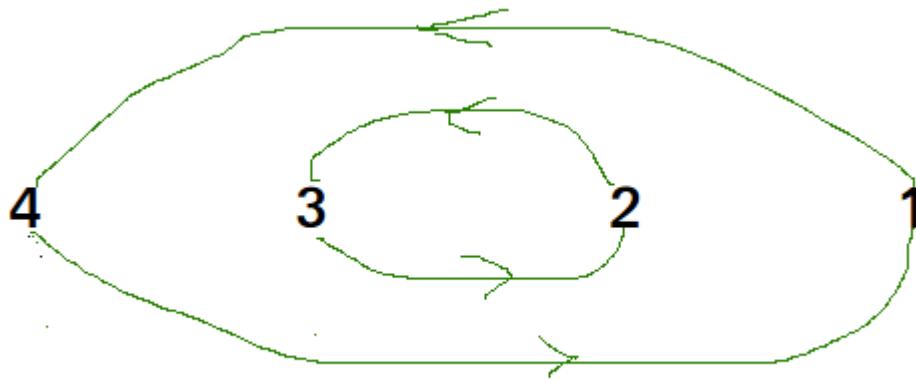
Cycle sort is an in-place sorting Algorithm, [unstable sorting algorithm](#), a comparison sort that is theoretically optimal in terms of the total number of writes to the original array.

- It is optimal in terms of number of memory writes. It [minimizes the number of memory writes](#) to sort (Each value is either written zero times, if it's already in its correct position, or written one time to its correct position.)
- It is based on the idea that array to be sorted can be divided into cycles. Cycles can be visualized as a graph. We have n nodes and an edge directed from node i to node j if the element at i-th index must be present at j-th index in the sorted array.

Cycle in arr[] = {4, 5, 2, 1, 5}



Cycle in arr[] = {4, 3, 2, 1}



We one by one consider all cycles. We first consider the cycle that includes first element. We find correct position of first element, place it at its correct position, say j. We consider old value of arr[j] and find its correct position, we keep doing this till all elements of current cycle are placed at correct position, i.e., we don't come back to cycle starting point.

Source

<https://www.geeksforgeeks.org/python-program-for-cycle-sort/>

Python3

```
# Python program to implement cycle sort

def cycleSort(array):
    writes = 0

    # Loop through the array to find cycles to rotate.
    for cycleStart in range(0, len(array) - 1):
        item = array[cycleStart]

        # Find where to put the item.
        pos = cycleStart
        for i in range(cycleStart + 1, len(array)):
            if array[i] < item:
                pos += 1

        # If the item is already there, this is not a cycle.
        if pos == cycleStart:
            continue

        # Otherwise, swap the item to its correct position.
        array[pos], array[cycleStart] = array[cycleStart], array[pos]
        writes += 1

    return writes
```

```
# Otherwise, put the item there or right after any duplicates.
while item == array[pos]:
    pos += 1
array[pos], item = item, array[pos]
writes += 1

# Rotate the rest of the cycle.
while pos != cycleStart:

    # Find where to put the item.
    pos = cycleStart
    for i in range(cycleStart + 1, len(array)):
        if array[i] < item:
            pos += 1

    # Put the item there or right after any duplicates.
    while item == array[pos]:
        pos += 1
    array[pos], item = item, array[pos]
    writes += 1

return writes

# driver code
arr = [1, 8, 3, 9, 10, 10, 2, 4 ]
n = len(arr)
cycleSort(arr)

print("After sort : ")
for i in range(0, n) :
    print(arr[i], end = ' ')

# Code Contributed by Mohit Gupta_OMG <(0_o)>
```

Output:

```
After sort :
1 2 3 4 8 9 10 10
```

Please refer complete article on [Cycle Sort](#) for more details!

Chapter 260

Python Program for Iterative Merge Sort

Python Program for Iterative Merge Sort - GeeksforGeeks

Following is a typical recursive implementation of [Merge Sort](#) that uses last element as pivot.

Source

<https://www.geeksforgeeks.org/python-program-for-iterative-merge-sort/>

Python

```
# Recursive Python Program for merge sort

def merge(left, right):
    if not len(left) or not len(right):
        return left or right

    result = []
    i, j = 0, 0
    while (len(result) < len(left) + len(right)):
        if left[i] < right[j]:
            result.append(left[i])
            i+= 1
        else:
            result.append(right[j])
            j+= 1
    if i == len(left) or j == len(right):
        result.extend(left[i:] or right[j:])
        break
```

```
return result

def mergesort(list):
    if len(list) < 2:
        return list

    middle = len(list)/2
    left = mergesort(list[:middle])
    right = mergesort(list[middle:])

    return merge(left, right)

seq = [12, 11, 13, 5, 6, 7]
print("Given array is")
print(seq);
print("\n")
print("Sorted array is")
print(mergesort(seq))

# Code Contributed by Mohit Gupta_OMG
```

Please refer complete article on [Iterative Merge Sort](#) for more details!

Chapter 261

Python Program for QuickSort

Python Program for QuickSort - GeeksforGeeks

Like [Merge Sort](#), QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Pseudo Code for recursive QuickSort function :

```
/* low --> Starting index, high --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1); // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

Source

<https://www.geeksforgeeks.org/python-program-for-quicksort/>

Python

```
# Python program for implementation of Quicksort Sort

# This function takes last element as pivot, places
# the pivot element at its correct position in sorted
# array, and places all smaller (smaller than pivot)
# to left of pivot and all greater elements to right
# of pivot
def partition(arr,low,high):
    i = ( low-1 )          # index of smaller element
    pivot = arr[high]        # pivot

    for j in range(low , high):

        # If current element is smaller than or
        # equal to pivot
        if arr[j] <= pivot:

            # increment index of smaller element
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[high] = arr[high],arr[i+1]
    return ( i+1 )

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low --> Starting index,
# high --> Ending index

# Function to do Quick sort
def quickSort(arr,low,high):
    if low < high:

        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr,low,high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)
```

```
# Driver code to test above
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quickSort(arr,0,n-1)
print ("Sorted array is:")
for i in range(n):
    print ("%d" %arr[i]),

# This code is contributed by Mohit Kumra
```

Please refer complete article on [QuickSort](#) for more details!

Chapter 262

Python Program for Recursive Insertion Sort

Python Program for Recursive Insertion Sort - GeeksforGeeks

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Below is an iterative algorithm for insertion sort

Algorithm

```
// Sort an arr[] of size n
insertionSort(arr, n)
    Loop from i = 1 to n-1.
        a) Pick element arr[i] and insert
            it into sorted sequence arr[0..i-1]
```

Source

<https://www.geeksforgeeks.org/python-program-for-recursive-insertion-sort/>

Python

```
# Recursive Python program for insertion sort
# Recursive function to sort an array using insertion sort

def insertionSortRecursive(arr,n):
    # base case
    if n<=1:
        return

    # Sort first n-1 elements
```

```
insertionSortRecursive(arr,n-1)
    '''Insert last element at its correct position
     in sorted array.'''
last = arr[n-1]
j = n-2

    # Move elements of arr[0..i-1], that are
    # greater than key, to one position ahead
    # of their current position
while (j>=0 and arr[j]>last):
    arr[j+1] = arr[j]
    j = j-1

arr[j+1]=last

# A utility function to print an array of size n
def printArray(arr,n):
    for i in range(n):
        print arr[i],

# Driver program to test insertion sort
arr = [12,11,13,5,6]
n = len(arr)
insertionSortRecursive(arr, n)
printArray(arr, n)

# Contributed by Harsh Valecha
```

Please refer complete article on [Recursive Insertion Sort](#) for more details!

Chapter 263

Python Program for Stooge Sort

Python Program for Stooge Sort - GeeksforGeeks

The Stooge sort is a recursive sorting algorithm. It is defined as below (for ascending order sorting).

Step 1 : If value at index 0 is greater than
value at last index, swap them.
Step 2: Recursively,
a) Stooge sort the initial 2/3rd of the array.
b) Stooge sort the last 2/3rd of the array.
c) Stooge sort the initial 2/3rd again to confirm.

Source

<https://www.geeksforgeeks.org/python-program-for-stooge-sort/>

Python3

```
# Python program to implement stooge sort

def stoogesort(arr, l, h):
    if l >= h:
        return

    # If first element is smaller
    # than last,swap them
    if arr[l]>arr[h]:
        t = arr[l]
        arr[l] = arr[h]
        arr[h] = t

    stoogesort(arr, l+1, h)
    stoogesort(arr, l, h-1)
    stoogesort(arr, l+1, h)
```

```
arr[h] = t

# If there are more than 2 elements in
# the array
if h-l+1 > 2:
    t = (int)((h-l+1)/3)

    # Recursively sort first 2/3 elements
    stoogesort(arr, l, (h-t))

    # Recursively sort last 2/3 elements
    stoogesort(arr, l+t, (h))

    # Recursively sort first 2/3 elements
    # again to confirm
    stoogesort(arr, l, (h-t))

# deriver
arr = [2, 4, 5, 3, 1]
n = len(arr)

stoogesort(arr, 0, n-1)

for i in range(0, n):
    print(arr[i], end = ' ')
```

Code Contributed by Mohit Gupta_OMG <(0_o)>

Output:

1 2 3 4 5

Please refer complete article on [Stooge Sort](#) for more details!

Chapter 264

Python list sort()

Python list sort() - GeeksforGeeks

The sort function can be used to sort a list in ascending, descending or user defined order.
To sort the list in ascending order.

List_name.sort()
This will sort the given list in ascending order.

This function can be used to sort list of integers, floating point number, string and others.

```
numbers = [1, 3, 4, 2]

# Sorting list of Integers in ascending
numbers.sort()

print(numbers)
```

Output:

```
[1, 2, 3, 4]
```

To sort the list in descending order.

```
list_name.sort(reverse=True)
This will sort the given list in descending order.

numbers = [1, 3, 4, 2]

# Sorting list of Integers in descending
numbers.sort(reverse = True)

print(numbers)
```

Output:

[4, 3, 2, 1]

Sorting user using user defined order.

```
list_name.sort(key=..., reverse=...) – it sorts according to user's choice

# Python program to demonstrate sorting by user's
# choice

# function to return the second element of the
# two elements passed as the parameter
def sortSecond(val):
    return val[1]

# list1 to demonstrate the use of sorting
# using using second key
list1 = [(1, 2), (3, 3), (1, 1)]

# sorts the array in ascending according to
# second element
list1.sort(key = sortSecond)
print(list1)

# sorts the array in descending according to
# second element
list1.sort(key = sortSecond, reverse = True)
print(list1)
```

Output:

[(1, 1), (1, 2), (3, 3)]
[(3, 3), (1, 2), (1, 1)]

- Sorting lists of different data types.
- Sort a list according to the second element in sublist.

Source

<https://www.geeksforgeeks.org/python-list-sort/>

Chapter 265

Python Sort Tuples in Increasing Order by any key

Python Sort Tuples in Increasing Order by any key - GeeksforGeeks

Given a tuple, sort the list of tuples in increasing order by any key in tuple.

Examples:

Input : tuple = [(2, 5), (1, 2), (4, 4), (2, 3)]
m = 0

Output : [(1, 2), (2, 3), (2, 5), (4, 4)]

Explanation: Sorted using the 0th index key.

Input : [(23, 45, 20), (25, 44, 39), (89, 40, 23)]
m = 2

Output : Sorted: [(23, 45, 20), (89, 40, 23), (25, 44, 39)]

Explanation: Sorted using the 2nd index key

Given tuples, we need to sort them according to any given key. This can be done using `sorted()` function where we sort them using `key=last` and store last as the key index according to which we have to sort the given tuples.

Below is the Python implementation of the above approach:

```
# Python code to sort a list of tuples  
# according to given key.  
  
# get the last key.  
def last(n):  
    return n[m]
```

```
# function to sort the tuple
def sort(tuples):

    # We pass used defined function last
    # as a parameter.
    return sorted(tuples, key = last)

# driver code
a = [(23, 45, 20), (25, 44, 39), (89, 40, 23)]
m = 2
print("Sorted:"),
print(sort(a))
```

Output:

```
Sorted: [(23, 45, 20), (89, 40, 23), (25, 44, 39)]
```

Source

<https://www.geeksforgeeks.org/python-sort-tuples-increasing-order-key/>

Chapter 266

Python Sort a List according to the Length of the Elements

Python Sort a List according to the Length of the Elements - GeeksforGeeks

In this program, we need to accept a list and sort it based on the length of the elements present within.

Examples:

```
Input : list = ["rohan", "amy", "sapna", "muhammad",
                "akash", "raunak", "chinmoy"]
Output : ['amy', 'rohan', 'sapna', 'akash', 'raunak',
          'chinmoy', 'muhammad']

Input : list = [["ram", "mohan", "aman"], ["gaurav"],
                ["amy", "sima", "ankita", "rinku"]]
Output : [['gaurav'], ['ram', 'mohan', 'aman'],
          ['amy', 'sima', 'ankita', 'rinku']]
```

Note: The first example comprises of Strings whose length can be calculated. The second example comprises of sublists, which is also arranged according to their length.

There are many ways of performing this. Anyone can use their own algorithmic technique, but Python provides us with various built-in functions to perform these. The built-in functions include **sort()** and **sorted()** along with the key parameter. We can perform these in two ways. One way is to sort the list by creating a new list and another way is to sort within the given list, saving space.

The syntax for sorting by creating a new list is:

```
sorted_list = sorted(unsorted_list, key=len)

# Python code to sort a list by creating
# another list Use of sorted()
def Sorting(lst):
    lst2 = sorted(lst, key=len)
    return lst2

# Driver code
lst = ["rohan", "amy", "sapna", "muhammad",
       "aakash", "raunak", "chinmoy"]
print(Sorting(lst))
```

The syntax for sorting without creating a new list is:

```
unsorted_list.sort(key=len)

# Python code to sort a list without
# creating another list Use of sort()
def Sorting(lst):
    lst.sort(key=len)
    return lst

# Driver code
lst = ["rohan", "amy", "sapna", "muhammad",
       "aakash", "raunak", "chinmoy"]
print(Sorting(lst))
```

Output:

```
['amy', 'rohan', 'sapna', 'aakash', 'raunak', 'chinmoy', 'muhammad']
```

Working:

These key function of Python's while sorting implemented is known as the **decorate-sort-undecorate design pattern**. It follows the following steps:

1. Each element of the list is temporarily replaced with a “decorated” version that includes the result of the key function applied to the element.
2. The list is sorted based upon the natural order of the keys.
3. The decorated elements are replaced by the original elements.

Reference: [stackoverflow](#)

Source

<https://www.geeksforgeeks.org/python-sort-list-according-length-elements/>

Chapter 267

Python Sort a list according to the second element in sublist

Python Sort a list according to the second element in sublist - GeeksforGeeks

In this article, we will learn how to sort any list, according to the second element of the sublist present within the main list. We will see two methods of doing this. We will learn three methods of performing this sort. One by the use of Bubble Sort, second by using the `sort()` method and last but not the least by the use of `sorted()` method. In this program, we have sorted the list in ascending order.

Examples:

```
Input : [[['rishav', 10], ['akash', 5], ['ram', 20], ['gaurav', 15]]
Output : [[['akash', 5], ['rishav', 10], ['gaurav', 15], ['ram', 20]]
```

```
Input : [['452', 10], ['256', 5], ['100', 20], ['135', 15]]
Output : [['256', 5], ['452', 10], ['135', 15], ['100', 20]]
```

Method 1: Using the Bubble Sort technique

Here we have used the technique of [Bubble Sort](#) to perform the sorting. We have tried to access the second element of the sublists using the nested loops. This performs the in-place method of sorting. The time complexity is similar to the Bubble Sort i.e., $O(n^2)$

```
# Python code to sort the lists using the second element of sublists
# Inplace way to sort, use of third variable
def Sort(sub_li):
    l = len(sub_li)
    for i in range(0, l):
        for j in range(0, l-i-1):
            if (sub_li[j][1] > sub_li[j + 1][1]):
                tempo = sub_li[j]
```

```
        sub_li[j]= sub_li[j + 1]
        sub_li[j + 1]= tempo
    return sub_li

# Driver Code
sub_li =[['rishav', 10], ['akash', 5], ['ram', 20], ['gaurav', 15]]
print(Sort(sub_li))
```

Output:

```
[['akash', 5], ['rishav', 10], ['gaurav', 15], ['ram', 20]]
```

Method 2: Sorting by the use of sort() method

While sorting via this method the actual content of the tuple is changed, and just like the previous method, in-place method of sort is performed.

```
# Python code to sort the tuples using second element
# of sublist Inplace way to sort using sort()
def Sort(sub_li):

    # reverse = None (Sorts in Ascending order)
    # key is set to sort using second element of
    # sublist lambda has been used
    sub_li.sort(key = lambda x: x[1])
    return sub_li

# Driver Code
sub_li =[['rishav', 10], ['akash', 5], ['ram', 20], ['gaurav', 15]]
print(Sort(sub_li))
```

Output:

```
[['akash', 5], ['rishav', 10], ['gaurav', 15], ['ram', 20]]
```

Method 3: Sorting by the use of sorted() method

Sorted() sorts a list and always returns a list with the elements in a sorted manner, without modifying the original sequence. It takes three parameters from which two are optional, here we tried to use all of the three:

1. Iterable : sequence (list, tuple, string) or collection (dictionary, set, frozenset) or any other iterator that needs to be sorted.
2. Key(optional) : A function that would serve as a key or a basis of sort comparison.

3. Reverse(optional) : To sort this in ascending order we could have just ignored the third parameter, which we did in this program. If set true, then the iterable would be sorted in reverse (descending) order, by default it is set as false.

```
# Python code to sort the tuples using second element
# of sublist Function to sort using sorted()
def Sort(sub_li):

    # reverse = None (Sorts in Ascending order)
    # key is set to sort using second element of
    # sublist lambda has been used
    return(sorted(sub_li, key = lambda x: x[1]))

# Driver Code
sub_li =[['rishav', 10], ['akash', 5], ['ram', 20], ['gaurav', 15]]
print(Sort(sub_li))
```

Output:

```
[['akash', 5], ['rishav', 10], ['gaurav', 15], ['ram', 20]]
```

Source

<https://www.geeksforgeeks.org/python-sort-list-according-second-element-sublist/>

Chapter 268

Python Sort an array according to absolute difference

Python Sort an array according to absolute difference - GeeksforGeeks

Given an array of **N** distinct elements and a number **val**, rearrange the array elements according to the absolute difference with **val**, i. e., element having minimum difference comes first and so on. Also the order of array elements should be maintained in case two or more elements have equal differences.

Examples:

Input: val = 6, a = [7, 12, 2, 4, 8, 0]

Output: a = [7 4 8 2 12 0]

Explanation:

Consider the absolute difference of each array element and ‘val’

7 – 6 = 1

12 – 6 = 6

2 – 6 = 4 (abs)

4 – 6 = 2 (abs)

8 – 6 = 2

0 – 6 = 6 (abs)

So, according to the obtained differences, sort the array differences in increasing order,

1, 2, 2, 4, 6, 6 is obtained, and their corresponding array elements are 7, 4, 8, 2, 12, 0.

Input: val = -2, a = [5, 2, 0, -4]

Output: a = [0 -4 2 5]

Approach: Normally a dictionary in Python can hold only one value corresponding to a key. But in this problem we may have to store multiple values for a particular key. Clearly, simple dictionary cannot be used, we need something like a [multimap in C++](#). So, here

Default dictionary is used, which works as Multimap in Python. Following are the steps to solve the problem.

- Store values in dictionary, where key is absolute difference between ‘val’ and array elements (i.e. $\text{abs}(\text{val}-\text{a}[i])$) and value is array elements (i.e. $\text{a}[i]$)
- In multimap, the values will be already in sorted order according to key because it implements self-balancing-binary-search-tree internally.
- Update the values of original array by storing the dictionary elements one by one.

Below is the implementation of the above approach.

```
# Python program to sort the array
# according to a value val

# Using Default dictionary (works as Multimap in Python)
from collections import defaultdict
def rearrange(a, n, val):
    # initialize Default dictionary
    dict = defaultdict(list)

    # Store values in dictionary (i.e. multimap) where,
    # key: absolute difference between 'val'
    # and array elements (i.e.  $\text{abs}(\text{val}-\text{a}[i])$ ) and
    # value: array elements (i.e.  $\text{a}[i]$ )
    for i in range(0, n):
        dict[abs(val-a[i])].append(a[i])

    index = 0

    # Update the values of original array
    # by storing the dictionary elements one by one
    for i in dict:

        pos = 0
        while (pos<len(dict[i])):
            a[index]= dict[i][pos]
            index+= 1
            pos+= 1

    # Driver code
    a =[7, 12, 2, 4, 8, 0]
    val = 6

    # len(a) gives the length of the list
    rearrange(a, len(a), val)
```

```
# print the modified final list
for i in a:
    print(i, end = ' ')
```

Output:

```
7 4 8 2 12 0
```

Time Complexity: $O(N * \log N)$

Auxiliary Space: $O(N)$

Alternative Approach: Use a 2-D matrix of size $n*2$ to store the absolute difference and index at $(i, 0)$ and $(i, 1)$. Sort the 2-D matrix. According to the inbuilt matrix property, the matrix gets sorted in terms of first element, and if the first element is same, it gets sorted according to the second element. Get the index and print the array element accordingly.

Below is the implementation of above approach.

```
def printsorted(a, n, val):

    # declare a 2-D matrix
    b = [[0 for x in range(2)] for y in range(n)]

    for i in range(0, n):
        b[i][0] = abs(a[i]-val)
        b[i][1] = i

    b.sort()

    for i in range(0, n):
        print a[b[i][1]],

a = [7, 12, 2, 4, 8, 0]
n = len(a)
val = 6
printsorted(a, n, val)
```

Output:

```
7 4 8 2 12 0
```

Time Complexity: $O(N * \log N)$

Auxiliary Space: $O(N)$

Source

<https://www.geeksforgeeks.org/python-sort-an-array-according-to-absolute-difference/>

Chapter 269

Python Sort words of sentence in ascending order

Python Sort words of sentence in ascending order - GeeksforGeeks

Given a sentence, sort it alphabetically in ascending order.

Examples:

Input : to learn programming refer geeksforgeeks
Output : geeksforgeeks learn programming refer to

Input : geeks for geeks
Output : for geeks geeks

We will use built in library function to sort the words of the sentence in ascending order.

Prerequisites:

1. [split\(\)](#)
2. [sort\(\) in Python](#)
3. [join\(\)](#)

- Split the Sentence in words.
- Sort the words alphabetically
- Join the sorted words alphabetically to form a new Sentence.

Below is the implementation of above idea.

```
# Function to sort the words
# in ascending order
def sortedSentence(Sentence):
```

```
# Spliting the Sentence into words
words = Sentence.split(" ")

# Sorting the words
words.sort()

# Making new Sentence by
# joining the sorted words
newSentence = " ".join(words)

# Return newSentence
return newSentence

# Driver's Code

Sentence = "to learn programming refer geeksforgeeks"
# Print the sortedSentence
print(sortedSentence(Sentence))

Sentence = "geeks for geeks"
# Print the sortedSentence
print(sortedSentence(Sentence))
```

Output:

```
geeksforgeeks learn programming refer to
for geeks geeks
```

Source

<https://www.geeksforgeeks.org/python-sort-words-sentence-ascending-order/>

Chapter 270

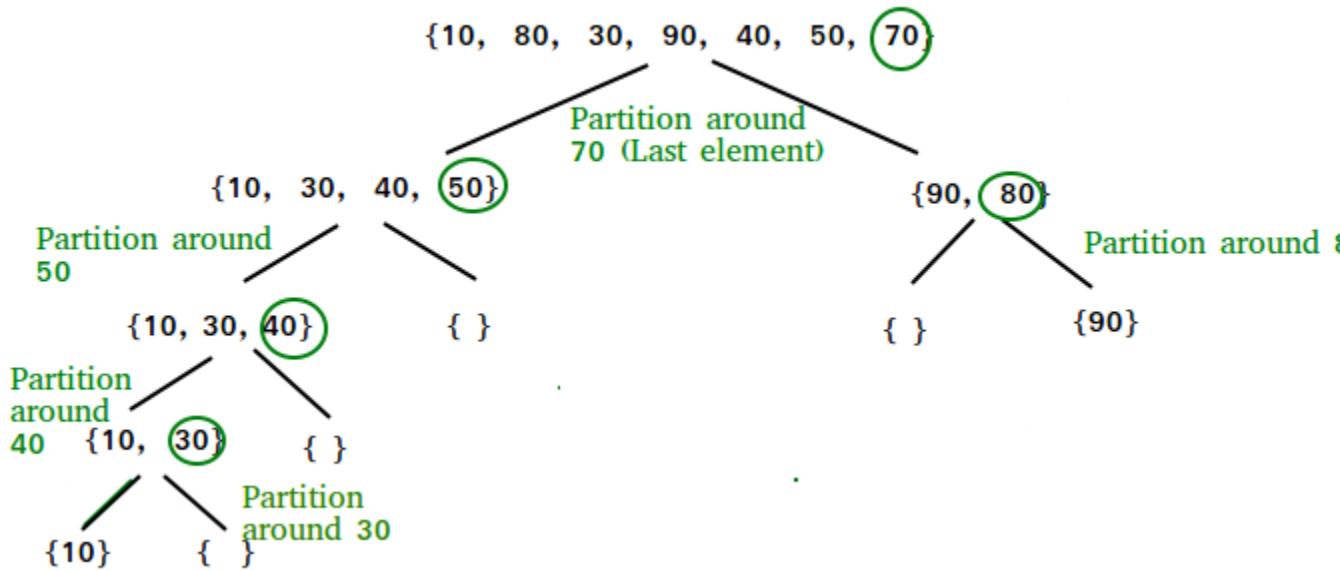
Quick Sort vs Merge Sort

Quick Sort vs Merge Sort - GeeksforGeeks

Prerequisite : [Merge Sort](#) and [Quick Sort](#)

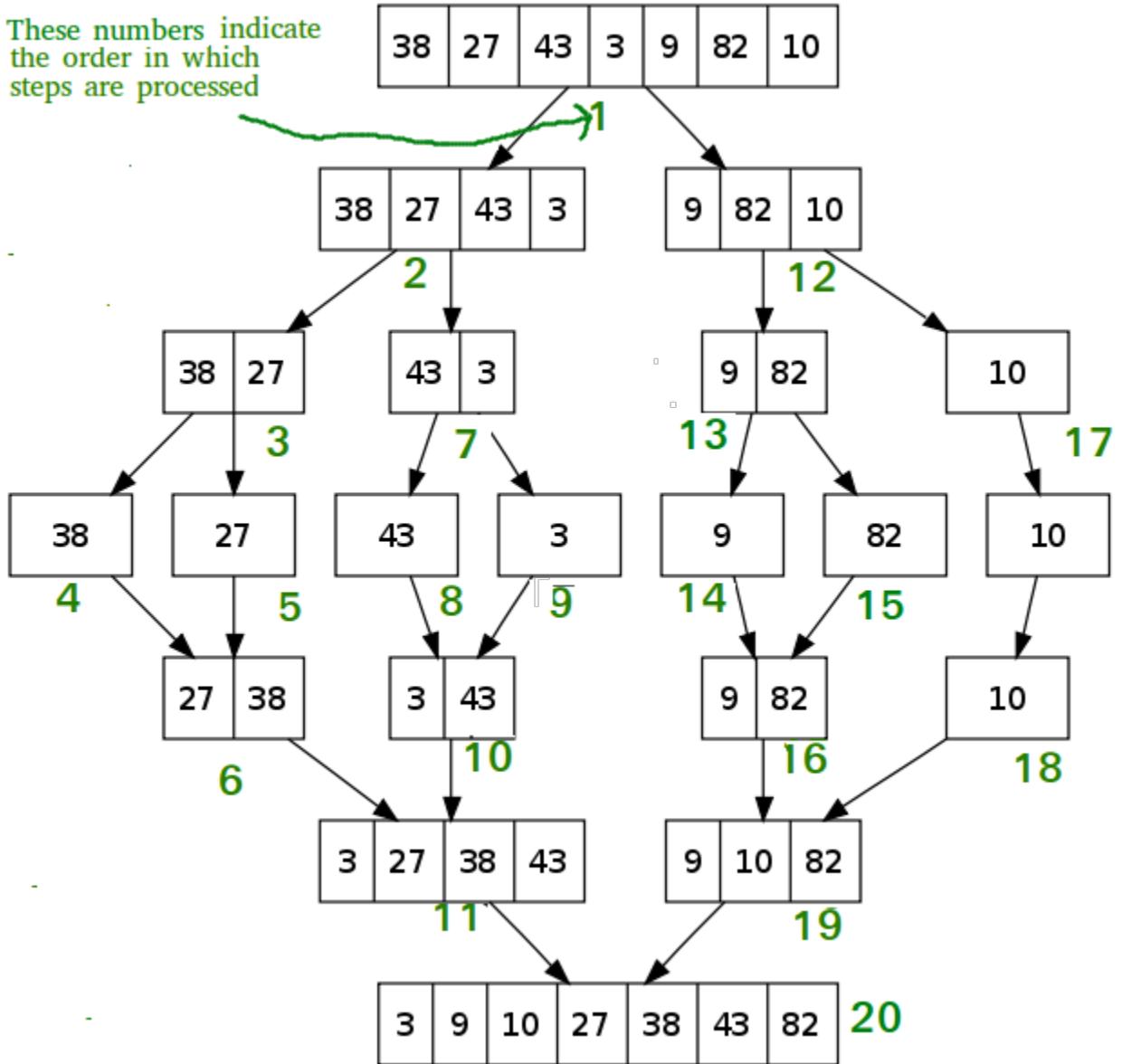
Quick sort is an internal algorithm which is based on divide and conquer strategy. In this:

- The array of elements is divided into parts repeatedly until it is not possible to divide it further.
- It is also known as “**partition exchange sort**”.
- It uses a key element (pivot) for partitioning the elements.
- One left partition contains all those elements that are smaller than the pivot and one right partition contains all those elements which are greater than the key element.



Merge sort is an external algorithm and based on divide and conquer strategy. In this:

- The elements are split into two sub-arrays ($n/2$) again and again until only one element is left.
- Merge sort uses additional storage for sorting the auxiliary array.
- Merge sort uses three arrays where two are used for storing each half, and the third external one is used to store the final sorted list by merging other two and each array is then sorted recursively.
- At last, the all sub arrays are merged to make it ' n ' element size of the array.



1. Partition of elements in the array :

In the merge sort, the array is parted into just 2 halves (i.e. $n/2$).

whereas

In case of quick sort, the array is parted into any ratio. There is no compulsion of dividing the array of elements into equal parts in quick sort.

2. Worst case complexity :

The worst case complexity of quick sort is $O(n^2)$ as there is need of lot of comparisons in the worst condition.

whereas

In merge sort, worst case and average case has same complexities $O(n \log n)$.

3. Usage with datasets :

Merge sort can work well on any type of data sets irrespective of its size (either large or small).

whereas

The quick sort cannot work well with large datasets.

4. Additional storage space requirement :

Merge sort is not in place because it requires additional memory space to store the auxiliary arrays.

whereas

The quick sort is in place as it doesn't require any additional storage.

5. Efficiency :

Merge sort is more efficient and works faster than quick sort in case of larger array size or datasets.

whereas

Quick sort is more efficient and works faster than merge sort in case of smaller array size or datasets.

6. Sorting method :

The quick sort is internal sorting method where the data is sorted in main memory.

whereas

The merge sort is external sorting method in which the data that is to be sorted cannot be accommodated in the memory and needed auxiliary memory for sorting.

7. Stability :

Merge sort is stable as two elements with equal value appear in the same order in sorted output as they were in the input unsorted array.

whereas

Quick sort is unstable in this scenario. But it can be made stable using some changes in code.

8. Preferred for :

Quick sort is preferred for arrays.

whereas

Merge sort is preferred for linked lists.

9. Locality of reference :

Quicksort exhibits good cache locality and this makes quicksort faster than merge sort (in many cases like in virtual memory environment).

Basis for comparison	Quick Sort	Merge Sort
The partition of elements in the array [the-partition-of-elements-in-the-array]	The splitting of a array of elements is in any ratio, not necessarily divided into half.	The splitting of a array of elements is in any ratio, not necessarily divided into half.

Basis for comparison	Quick Sort	Merge Sort
Worst case complexity [worst-case-complexity]	$O(n^2)$	$O(n \log n)$
Works well on [works-well-on]	It works well on smaller array	It operates fine on any size of array
Speed of execution [speed-of-execution]	It works faster than other sorting algorithms for small data set like Selection sort etc	It has a consistent speed on any size of data
Additional storage space requirement [additional-storage-space-requirement]	Less(In-place)	More(not In-place)
Efficiency [efficiency]	Inefficient for larger arrays	More efficient
Sorting method [sorting-method]	Internal	External
Stability [stability]	Not Stable	Stable
Preferred for [preferred-for]	for Arrays	for Linked Lists
Locality of reference [locality-of-reference]	good	poor

Source

<https://www.geeksforgeeks.org/quick-sort-vs-merge-sort/>

Chapter 271

QuickSort

QuickSort - GeeksforGeeks

Like [Merge Sort](#), QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

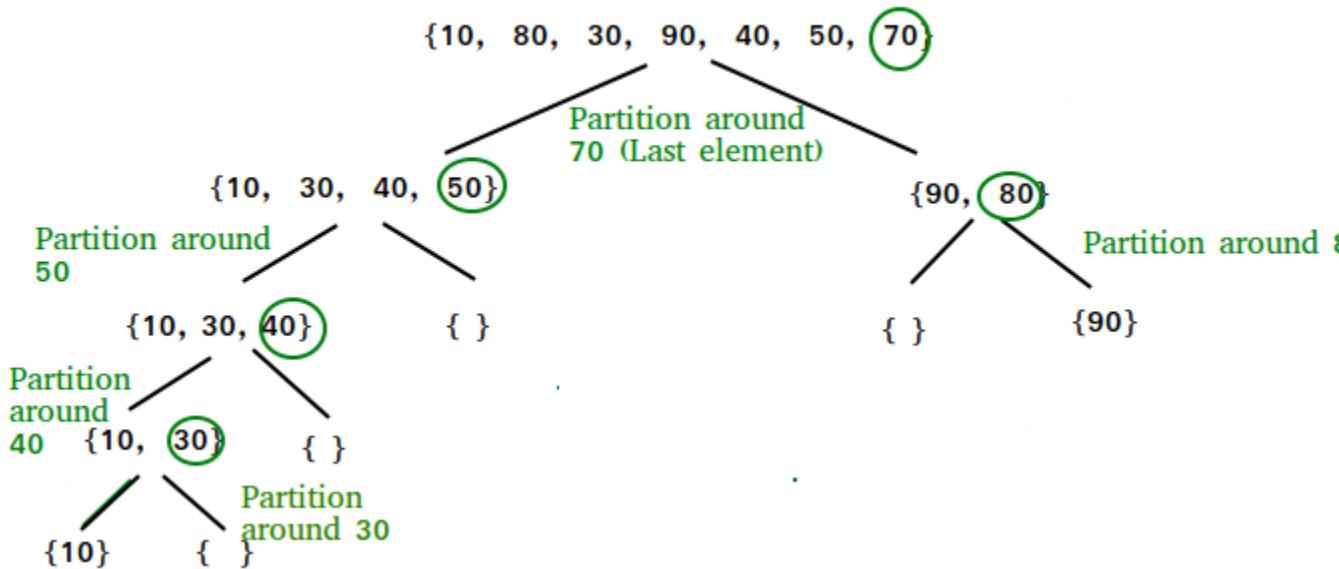
1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Pseudo Code for recursive QuickSort function :

```
/* low --> Starting index, high --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1); // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```



Partition Algorithm

There can be many ways to do partition, following pseudo code adopts the method given in CLRS book. The logic is simple, we start from the leftmost element and keep track of index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

```
/* low --> Starting index, high --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1); // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

Pseudo code for partition()

```
/* This function takes last element as pivot, places
   the pivot element at its correct position in sorted
   array, and places all smaller (smaller than pivot)
```

```

        to left of pivot and all greater elements to right
        of pivot */
partition (arr[], low, high)
{
    // pivot (Element to be placed at right position)
    pivot = arr[high];

    i = (low - 1) // Index of smaller element

    for (j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++; // increment index of smaller element
            swap arr[i] and arr[j]
        }
    }
    swap arr[i + 1] and arr[high])
    return (i + 1)
}

```

Illustration of partition() :

```

arr[] = {10, 80, 30, 90, 40, 50, 70}
Indexes: 0   1   2   3   4   5   6

low = 0, high = 6, pivot = arr[h] = 70
Initialize index of smaller element, i = -1

Traverse elements from j = low to high-1
j = 0 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
i = 0
arr[] = {10, 80, 30, 90, 40, 50, 70} // No change as i and j
                                         // are same

j = 1 : Since arr[j] > pivot, do nothing
// No change in i and arr[]

j = 2 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
i = 1
arr[] = {10, 30, 80, 90, 40, 50, 70} // We swap 80 and 30

j = 3 : Since arr[j] > pivot, do nothing
// No change in i and arr[]

j = 4 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])

```

```
i = 2
arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 and 40 Swapped
j = 5 : Since arr[j] <= pivot, do i++ and swap arr[i] with arr[j]
i = 3
arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 and 50 Swapped
```

We come out of loop because j is now equal to high-1.
Finally we place pivot at correct position by swapping
arr[i+1] and arr[high] (or pivot)
arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 and 70 Swapped

Now 70 is at its correct place. All elements smaller than 70 are before it and all elements greater than 70 are after it.

Implementation:

Following are C++, Java and Python implementations of QuickSort.

C/C++

```
/* C implementation QuickSort */
#include<stdio.h>

// A utility function to swap two elements
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function takes last element as pivot, places
   the pivot element at its correct position in sorted
   array, and places all smaller (smaller than pivot)
   to left of pivot and all greater elements to right
   of pivot */
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];      // pivot
    int i = (low - 1); // Index of smaller element

    for (int j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;      // increment index of smaller element
```

```

        swap(&arr[i], &arr[j]);
    }
}
swap(&arr[i + 1], &arr[high]);
return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    quickSort(arr, 0, n-1);
    printf("Sorted array: n");
    printArray(arr, n);
    return 0;
}

```

Java

```

// Java program for implementation of QuickSort
class QuickSort
{
    /* This function takes last element as pivot,
       places the pivot element at its correct
       position in sorted array, and places all
       smaller (smaller than pivot) to left of
       pivot and all greater elements to right
       of pivot */
    int partition(int arr[], int low, int high)
    {
        int pivot = arr[high];
        int i = (low-1); // index of smaller element
        for (int j=low; j<high; j++)
        {
            // If current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;
                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }

    /* The main function that implements QuickSort()
       arr[] --> Array to be sorted,
       low --> Starting index,
       high --> Ending index */
    void sort(int arr[], int low, int high)
    {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi] is
               now at right place */
            int pi = partition(arr, low, high);

```

```

        // Recursively sort elements before
        // partition and after partition
        sort(arr, low, pi-1);
        sort(arr, pi+1, high);
    }
}

/* A utility function to print array of size n */
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i]+" ");
    System.out.println();
}

// Driver program
public static void main(String args[])
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = arr.length;

    QuickSort ob = new QuickSort();
    ob.sort(arr, 0, n-1);

    System.out.println("sorted array");
    printArray(arr);
}
}
/*This code is contributed by Rajat Mishra */

```

Python

```

# Python program for implementation of Quicksort Sort

# This function takes last element as pivot, places
# the pivot element at its correct position in sorted
# array, and places all smaller (smaller than pivot)
# to left of pivot and all greater elements to right
# of pivot
def partition(arr,low,high):
    i = ( low-1 )          # index of smaller element
    pivot = arr[high]       # pivot

    for j in range(low , high):

        # If current element is smaller than or

```

```

# equal to pivot
if arr[j] <= pivot:

    # increment index of smaller element
    i = i+1
    arr[i],arr[j] = arr[j],arr[i]

arr[i+1],arr[high] = arr[high],arr[i+1]
return ( i+1 )

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low --> Starting index,
# high --> Ending index

# Function to do Quick sort
def quickSort(arr,low,high):
    if low < high:

        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr,low,high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)

# Driver code to test above
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quickSort(arr,0,n-1)
print ("Sorted array is:")
for i in range(n):
    print ("%d" %arr[i]),

# This code is contributed by Mohit Kumra

```

C#

```

// C# program for implementation of QuickSort
using System;

class GFG {

    /* This function takes last element as pivot,
    places the pivot element at its correct
    position in sorted array, and places all

```

```

smaller (smaller than pivot) to left of
pivot and all greater elements to right
of pivot */
static int partition(int []arr, int low,
                     int high)
{
    int pivot = arr[high];

    // index of smaller element
    int i = (low - 1);
    for (int j = low; j < high; j++)
    {
        // If current element is smaller
        // than or equal to pivot
        if (arr[j] <= pivot)
        {
            i++;

            // swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    // swap arr[i+1] and arr[high] (or pivot)
    int temp1 = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp1;

    return i+1;
}

/* The main function that implements QuickSort()
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
static void quickSort(int []arr, int low, int high)
{
    if (low < high)
    {

        /* pi is partitioning index, arr[pi] is
        now at right place */
        int pi = partition(arr, low, high);

        // Recursively sort elements before

```

```

        // partition and after partition
        quickSort(arr, low, pi-1);
        quickSort(arr, pi+1, high);
    }
}

// A utility function to print array of size n
static void printArray(int []arr, int n)
{
    for (int i = 0; i < n; ++i)
        Console.Write(arr[i] + " ");

    Console.WriteLine();
}

// Driver program
public static void Main()
{
    int []arr = {10, 7, 8, 9, 1, 5};
    int n = arr.Length;
    quickSort(arr, 0, n-1);
    Console.WriteLine("sorted array ");
    printArray(arr, n);
}
}

// This code is contributed by Sam007.

```

Output:

```

Sorted array:
1 5 7 8 9 10

```

Analysis of QuickSort

Time taken by QuickSort in general can be written as following.

$$T(n) = T(k) + T(n-k-1) + (n)$$

The first two terms are for two recursive calls, the last term is for the partition process. k is the number of elements which are smaller than pivot.

The time taken by QuickSort depends upon the input array and partition strategy. Following are three cases.

Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.

$T(n) = T(0) + T(n-1) + (n)$
 which is equivalent to
 $T(n) = T(n-1) + (n)$

The solution of above recurrence is $\Theta(n^2)$.

Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.

$$T(n) = 2T(n/2) + (n)$$

The solution of above recurrence is $\Theta(n\log n)$. It can be solved using case 2 of [Master Theorem](#).

Average Case:

To do average case analysis, we need to consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy.

We can get an idea of average case by considering the case when partition puts $O(n/9)$ elements in one set and $O(9n/10)$ elements in other set. Following is recurrence for this case.

$$T(n) = T(n/9) + T(9n/10) + (n)$$

Solution of above recurrence is also $O(n\log n)$

Although the worst case time complexity of QuickSort is $O(n^2)$ which is more than many other sorting algorithms like [Merge Sort](#) and [Heap Sort](#), QuickSort is faster in practice, because its inner loop can be efficiently implemented on most architectures, and in most real-world data. QuickSort can be implemented in different ways by changing the choice of pivot, so that the worst case rarely occurs for a given type of data. However, merge sort is generally considered better when data is huge and stored in external storage.

Is QuickSort stable?

The default implementation is not stable. However any sorting algorithm can be made stable by considering indexes as comparison parameter.

What is 3-Way QuickSort?

In simple QuickSort algorithm, we select an element as pivot, partition the array around pivot and recur for subarrays on left and right of pivot.

Consider an array which has many redundant elements. For example, $\{1, 4, 2, 4, 2, 4, 1, 2, 4, 1, 2, 2, 2, 4, 1, 4, 4, 4\}$. If 4 is picked as pivot in Simple QuickSort, we fix only one 4 and recursively process remaining occurrences. In 3 Way QuickSort, an array $arr[l..r]$ is divided in 3 parts:

- a) $arr[l..i]$ elements less than pivot.
- b) $arr[i+1..j-1]$ elements equal to pivot.

c) arr[j..r] elements greater than pivot.

See [this](#) for implementation.

How to implement QuickSort for Linked Lists?

[QuickSort on Singly Linked List](#)

[QuickSort on Doubly Linked List](#)

Can we implement QuickSort Iteratively?

Yes, please refer [Iterative Quick Sort](#).

Why Quick Sort is preferred over MergeSort for sorting Arrays

Quick Sort in its general form is an in-place sort (i.e. it doesn't require any extra storage) whereas merge sort requires $O(N)$ extra storage, N denoting the array size which may be quite expensive. Allocating and de-allocating the extra space used for merge sort increases the running time of the algorithm. Comparing average complexity we find that both type of sorts have $O(N \log N)$ average complexity but the constants differ. For arrays, merge sort loses due to the use of extra $O(N)$ storage space.

Most practical implementations of Quick Sort use randomized version. The randomized version has expected time complexity of $O(n \log n)$. The worst case is possible in randomized version also, but worst case doesn't occur for a particular pattern (like sorted array) and randomized Quick Sort works well in practice.

Quick Sort is also a cache friendly sorting algorithm as it has good locality of reference when used for arrays.

Quick Sort is also tail recursive, therefore tail call optimizations is done.

Why MergeSort is preferred over QuickSort for Linked Lists?

In case of linked lists the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike array, in linked list, we can insert items in the middle in $O(1)$ extra space and $O(1)$ time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.

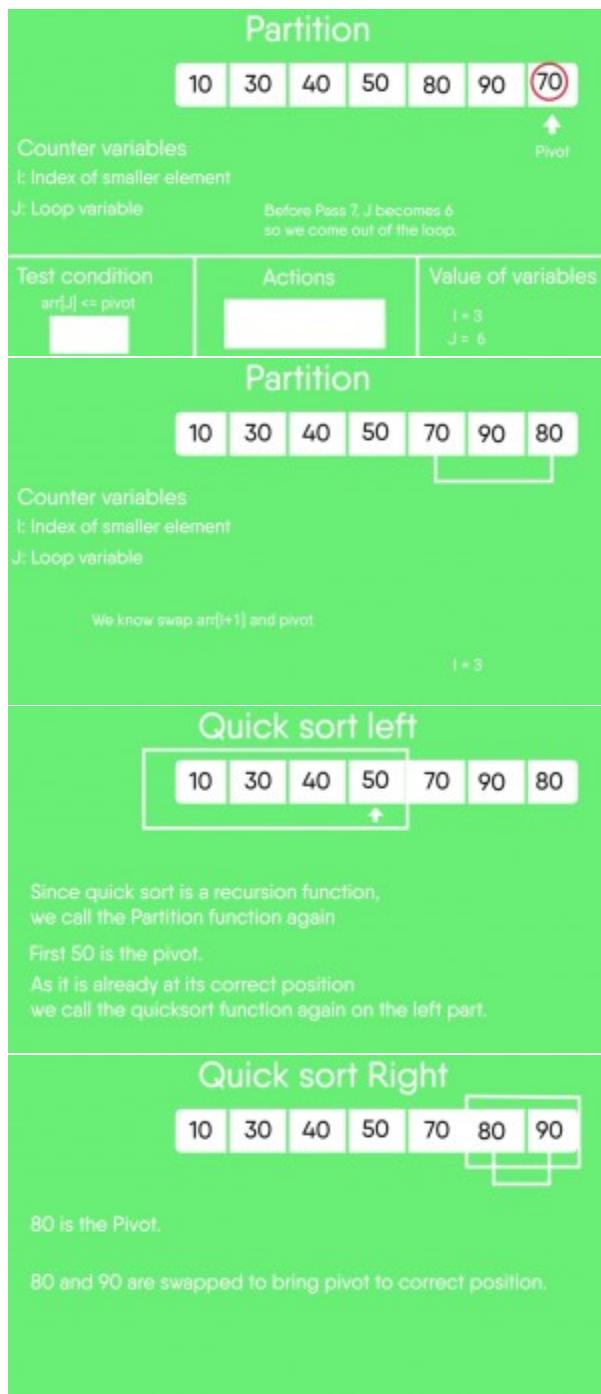
In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of $A[0]$ be x then to access $A[i]$, we can directly access the memory at $(x + i * 4)$. Unlike arrays, we can not do random access in linked list. Quick Sort requires a lot of this kind of access. In linked list to access i 'th index, we have to travel each and every node from the head to i 'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

How to optimize QuickSort so that it takes $O(\log n)$ extra space in worst case?

Please see [QuickSort Tail Call Optimization \(Reducing worst case space to Log n \)](#)

Snapshots:

Partition		
Test condition arr[j] <= pivot	Actions	Value of variables I = -1 J = 0
		Pivot
Counter variables I: Index of smaller element J: Loop variable		
		We start the loop with initial values.
Partition		
Test condition arr[j] <= pivot	Actions	Value of variables I = -1 J = 0
		Pivot
Counter variables I: Index of smaller element J: Loop variable		
		Pass 2
Test condition arr[j] <= pivot		
80 < 70 False	No action	I = 0 J = 1
Partition		
		Pivot
Counter variables I: Index of smaller element J: Loop variable		
Test condition arr[j] <= pivot		
30 < 70 True	I++ Swap(arr[I], arr[J])	I = 1 J = 2
Partition		
		Pivot
Counter variables I: Index of smaller element J: Loop variable		
		Pass 5
Test condition arr[j] <= pivot		
40 < 70 True	I++ Swap(arr[I], arr[J])	I = 2 J = 4



- Quiz on QuickSort
- Recent Articles on QuickSort

- Coding practice for sorting.

References:

<http://en.wikipedia.org/wiki/Quicksort>

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket Sort, ShellSort, Comb Sort, Pigeonhole Sort

Improved By : Palak Jain 5

Source

<https://www.geeksforgeeks.org/quick-sort/>

Chapter 272

QuickSort Tail Call Optimization (Reducing worst case space to Log n)

QuickSort Tail Call Optimization (Reducing worst case space to Log n) - GeeksforGeeks

In [QuickSort](#), partition function is in-place, but we need extra space for recursive function calls. A simple implementation of QuickSort makes two calls to itself and in worst case requires $O(n)$ space on function call stack.

The worst case happens when the selected pivot always divides the array such that one part has 0 elements and other part has $n-1$ elements. For example, in below code, if we choose last element as pivot, we get worst case for sorted arrays (See [this](#) for visualization)

```
/* A Simple implementation of QuickSort that makes two
   two recursive calls. */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
// See below link for complete running code
// http://geeksquiz.com/quick-sort/
```

Can we reduce the auxiliary space for function call stack?

We can limit the auxiliary space to $O(\log n)$. The idea is based on [tail call elimination](#). As seen in the [previous post](#), we can convert the code so that it makes one recursive call. For example, in the below code, we have converted the above code to use a while loop and have reduced the number of recursive calls.

```
/* QuickSort after tail call elimination using while loop */
void quickSort(int arr[], int low, int high)
{
    while (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);

        low = pi+1;
    }
}
// See below link for complete running code
// https://ide.geeksforgeeks.org/qrlM31
```

Although we have reduced number of recursive calls, the above code can still use $O(n)$ auxiliary space in worst case. In worst case, it is possible that array is divided in a way that the first part always has $n-1$ elements. For example, this may happen when last element is chosen as pivot and array is sorted in decreasing order.

We can optimize the above code to make a recursive call only for the smaller part after partition. Below is implementation of this idea.

Further Optimization :

```
/* This QuickSort requires O(Log n) auxiliary space in
   worst case. */
void quickSort(int arr[], int low, int high)
{
    while (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        // If left part is smaller, then recur for left
        // part and handle right part iteratively
        if (pi - low < high - pi)
```

```
{  
    quickSort(arr, low, pi - 1);  
    low = pi + 1;  
}  
  
// Else recur for right part  
else  
{  
    quickSort(arr, pi + 1, high);  
    high = pi - 1;  
}  
}  
}  
// See below link for complete running code  
// https://ide.geeksforgeeks.org/LHxwPk
```

In the above code, if left part becomes smaller, then we make recursive call for left part. Else for the right part. In worst case (for space), when both parts are of equal sizes in all recursive calls, we use $O(\log n)$ extra space.

Reference:

<http://www.cs.nthu.edu.tw/~wkhon/algo08-tutorials/tutorial2b.pdf>

This article is contributed by **Dheeraj Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/quicksort-tail-call-optimization-reducing-worst-case-space-log-n/>

Chapter 273

QuickSort on Doubly Linked List

QuickSort on Doubly Linked List - GeeksforGeeks

Following is a typical recursive implementation of [QuickSort](#) for arrays. The implementation uses last element as pivot.

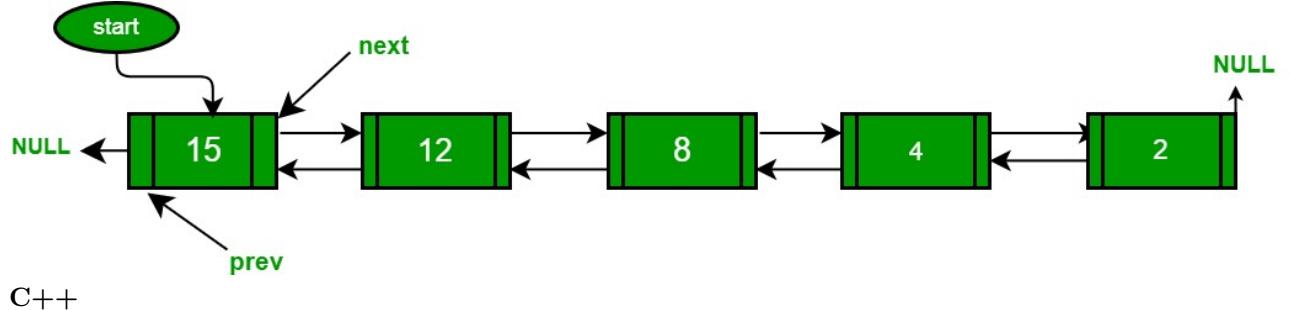
```
/* A typical recursive implementation of Quicksort for array*/  
  
/* This function takes last element as pivot, places the pivot element at its  
correct position in sorted array, and places all smaller (smaller than  
pivot) to left of pivot and all greater elements to right of pivot */  
int partition (int arr[], int l, int h)  
{  
    int x = arr[h];  
    int i = (l - 1);  
  
    for (int j = l; j <= h- 1; j++)  
    {  
        if (arr[j] <= x)  
        {  
            i++;  
            swap (&arr[i], &arr[j]);  
        }  
    }  
    swap (&arr[i + 1], &arr[h]);  
    return (i + 1);  
}  
  
/* A[] --> Array to be sorted, l --> Starting index, h --> Ending index */  
void quickSort(int A[], int l, int h)  
{
```

```

if (l < h)
{
    int p = partition(A, l, h); /* Partitioning index */
    quickSort(A, l, p - 1);
    quickSort(A, p + 1, h);
}
    
```

Can we use same algorithm for Linked List?

Following is C++ implementation for doubly linked list. The idea is simple, we first find out pointer to last node. Once we have pointer to last node, we can recursively sort the linked list using pointers to first and last nodes of linked list, similar to the above recursive function where we pass indexes of first and last array elements. The partition function for linked list is also similar to partition for arrays. Instead of returning index of the pivot element, it returns pointer to the pivot element. In the following implementation, quickSort() is just a wrapper function, the main recursive function is _quickSort() which is similar to quickSort() for array implementation.



C++

```

// A C++ program to sort a linked list using Quicksort
#include <iostream>
#include <stdio.h>
using namespace std;

/* a node of the doubly linked list */
struct Node
{
    int data;
    struct Node *next;
    struct Node *prev;
};

/* A utility function to swap two elements */
void swap ( int* a, int* b )
{   int t = *a;      *a = *b;      *b = t;  }

// A utility function to find last node of linked list
struct Node *lastNode(Node *root)
    
```

```

{
    while (root && root->next)
        root = root->next;
    return root;
}

/* Considers last element as pivot, places the pivot element at its
   correct position in sorted array, and places all smaller (smaller than
   pivot) to left of pivot and all greater elements to right of pivot */
Node* partition(Node *l, Node *h)
{
    // set pivot as h element
    int x = h->data;

    // similar to i = l-1 for array implementation
    Node *i = l->prev;

    // Similar to "for (int j = l; j <= h- 1; j++)"
    for (Node *j = l; j != h; j = j->next)
    {
        if (j->data <= x)
        {
            // Similar to i++ for array
            i = (i == NULL)? l : i->next;

            swap(&(i->data), &(j->data));
        }
    }
    i = (i == NULL)? l : i->next; // Similar to i++
    swap(&(i->data), &(h->data));
    return i;
}

/* A recursive implementation of quicksort for linked list */
void _quickSort(struct Node* l, struct Node *h)
{
    if (h != NULL && l != h && l != h->next)
    {
        struct Node *p = partition(l, h);
        _quickSort(l, p->prev);
        _quickSort(p->next, h);
    }
}

// The main function to sort a linked list. It mainly calls _quickSort()
void quickSort(struct Node *head)
{
    // Find last node
}

```

```

struct Node *h = lastNode(head);

// Call the recursive QuickSort
_quickSort(head, h);
}

// A utility function to print contents of arr
void printList(struct Node *head)
{
    while (head)
    {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

/* Function to insert a node at the begining of the Doubly Linked List */
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = new Node;      /* allocate node */
    new_node->data = new_data;

    /* since we are adding at the begining, prev is always NULL */
    new_node->prev = NULL;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* change prev of head node to new node */
    if ((*head_ref) != NULL)  (*head_ref)->prev = new_node ;

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Driver program to test above function */
int main()
{
    struct Node *a = NULL;
    push(&a, 5);
    push(&a, 20);
    push(&a, 4);
    push(&a, 3);
    push(&a, 30);

    cout << "Linked List before sorting \n";
    printList(a);
}

```

```
    quickSort(a);

    cout << "Linked List after sorting \n";
    printList(a);

    return 0;
}
```

Java

```
// A Java program to sort a linked list using Quicksort
class QuickSort_using_Doubly_LinkedList{
    Node head;

    /* a node of the doubly linked list */
    static class Node{
        private int data;
        private Node next;
        private Node prev;

        Node(int d){
            data = d;
            next = null;
            prev = null;
        }
    }

    // A utility function to find last node of linked list
    Node lastNode(Node node){
        while(node.next!=null)
            node = node.next;
        return node;
    }

    /* Considers last element as pivot, places the pivot element at its
       correct position in sorted array, and places all smaller (smaller than
       pivot) to left of pivot and all greater elements to right of pivot */
    Node partition(Node l,Node h)
    {
        // set pivot as h element
        int x = h.data;

        // similar to i = l-1 for array implementation
        Node i = l.prev;

        // Similar to "for (int j = l; j <= h- 1; j++)"

```

```
for(Node j=l; j!=h; j=j.next)
{
    if(j.data <= x)
    {
        // Similar to i++ for array
        i = (i==null) ? l : i.next;
        int temp = i.data;
        i.data = j.data;
        j.data = temp;
    }
    i = (i==null) ? l : i.next; // Similar to i++
    int temp = i.data;
    i.data = h.data;
    h.data = temp;
    return i;
}

/* A recursive implementation of quicksort for linked list */
void _quickSort(Node l,Node h)
{
    if(h!=null && l!=h && l!=h.next){
        Node temp = partition(l,h);
        _quickSort(l,temp.prev);
        _quickSort(temp.next,h);
    }
}

// The main function to sort a linked list. It mainly calls _quickSort()
public void quickSort(Node node)
{
    // Find last node
    Node head = lastNode(node);

    // Call the recursive QuickSort
    _quickSort(node,head);
}

// A utility function to print contents of arr
public void printList(Node head)
{
    while(head!=null){
        System.out.print(head.data+" ");
        head = head.next;
    }
}

/* Function to insert a node at the beginning of the Doubly Linked List */
```

```
void push(int new_Data)
{
    Node new_Node = new Node(new_Data);      /* allocate node */

    // if head is null, head = new_Node
    if(head==null){
        head = new_Node;
        return;
    }

    /* link the old list off the new node */
    new_Node.next = head;

    /* change prev of head node to new node */
    head.prev = new_Node;

    /* since we are adding at the begining, prev is always NULL */
    new_Node.prev = null;

    /* move the head to point to the new node */
    head = new_Node;
}

/* Driver program to test above function */
public static void main(String[] args){
    QuickSort_using_Doubly_LinkedList list = new QuickSort_using_Doubly_LinkedList();

    list.push(5);
    list.push(20);
    list.push(4);
    list.push(3);
    list.push(30);

    System.out.println("Linked List before sorting ");
    list.printList(list.head);
    System.out.println("\nLinked List after sorting");
    list.quickSort(list.head);
    list.printList(list.head);

}

// This code has been contributed by Amit Khandelwal
```

Output :

Linked List before sorting

30 3 4 20 5

Linked List after sorting

3 4 5 20 30

Time Complexity: Time complexity of the above implementation is same as time complexity of QuickSort() for arrays. It takes $O(n^2)$ time in worst case and $O(n \log n)$ in average and best cases. The worst case occurs when the linked list is already sorted.

Can we implement random quick sort for linked list?

Quicksort can be implemented for Linked List only when we can pick a fixed point as pivot (like last element in above implementation). Random QuickSort cannot be efficiently implemented for Linked Lists by picking random pivot.

Exercise:

The above implementation is for doubly linked list. Modify it for singly linked list. Note that we don't have prev pointer in singly linked list.

Refer [QuickSort on Singly Linked List](#) for solution.

Source

<https://www.geeksforgeeks.org/quicksort-for-linked-list/>

Chapter 274

QuickSort on Singly Linked List

QuickSort on Singly Linked List - GeeksforGeeks

QuickSort on Doubly Linked List is discussed [here](#). QuickSort on Singly linked list was given as an exercise. Following is C++ implementation for same. The important things about implementation are, it changes pointers rather swapping data and time complexity is same as the implementation for Doubly Linked List.

In **partition()**, we consider last element as pivot. We traverse through the current list and if a node has value greater than pivot, we move it after tail. If the node has smaller value, we keep it at its current position.

In **QuickSortRecur()**, we first call **partition()** which places pivot at correct position and returns pivot. After pivot is placed at correct position, we find tail node of left side (list before pivot) and recur for left list. Finally, we recur for right list.

```
// C++ program for Quick Sort on Singly Linled List
#include <iostream>
#include <cstdio>
using namespace std;

/* a node of the singly linked list */
struct Node
{
    int data;
    struct Node *next;
};

/* A utility function to insert a node at the beginning of linked list */
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node = new Node;

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}
```

```
new_node->data = new_data;

/* link the old list off the new node */
new_node->next = (*head_ref);

/* move the head to point to the new node */
(*head_ref) = new_node;
}

/* A utility function to print linked list */
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

// Returns the last node of the list
struct Node *getTail(struct Node *cur)
{
    while (cur != NULL && cur->next != NULL)
        cur = cur->next;
    return cur;
}

// Partitions the list taking the last element as the pivot
struct Node *partition(struct Node *head, struct Node *end,
                      struct Node **newHead, struct Node **newEnd)
{
    struct Node *pivot = end;
    struct Node *prev = NULL, *cur = head, *tail = pivot;

    // During partition, both the head and end of the list might change
    // which is updated in the newHead and newEnd variables
    while (cur != pivot)
    {
        if (cur->data < pivot->data)
        {
            // First node that has a value less than the pivot - becomes
            // the new head
            if ((*newHead) == NULL)
                (*newHead) = cur;

            prev = cur;
            cur = cur->next;
        }
    }
}
```

```

    }

    else // If cur node is greater than pivot
    {
        // Move cur node to next of tail, and change tail
        if (prev)
            prev->next = cur->next;
        struct Node *tmp = cur->next;
        cur->next = NULL;
        tail->next = cur;
        tail = cur;
        cur = tmp;
    }
}

// If the pivot data is the smallest element in the current list,
// pivot becomes the head
if ((*newHead) == NULL)
    (*newHead) = pivot;

// Update newEnd to the current last node
(*newEnd) = tail;

// Return the pivot node
return pivot;
}

//here the sorting happens exclusive of the end node
struct Node *quickSortRecur(struct Node *head, struct Node *end)
{
    // base condition
    if (!head || head == end)
        return head;

    Node *newHead = NULL, *newEnd = NULL;

    // Partition the list, newHead and newEnd will be updated
    // by the partition function
    struct Node *pivot = partition(head, end, &newHead, &newEnd);

    // If pivot is the smallest element - no need to recur for
    // the left part.
    if (newHead != pivot)
    {
        // Set the node before the pivot node as NULL
        struct Node *tmp = newHead;
        while (tmp->next != pivot)
            tmp = tmp->next;
    }
}

```

```
tmp->next = NULL;

// Recur for the list before pivot
newHead = quickSortRecur(newHead, tmp);

// Change next of last node of the left half to pivot
tmp = getTail(newHead);
tmp->next = pivot;
}

// Recur for the list after the pivot element
pivot->next = quickSortRecur(pivot->next, newEnd);

return newHead;
}

// The main function for quick sort. This is a wrapper over recursive
// function quickSortRecur()
void quickSort(struct Node **headRef)
{
    (*headRef) = quickSortRecur(*headRef, getTail(*headRef));
    return;
}

// Driver program to test above functions
int main()
{
    struct Node *a = NULL;
    push(&a, 5);
    push(&a, 20);
    push(&a, 4);
    push(&a, 3);
    push(&a, 30);

    cout << "Linked List before sorting \n";
    printList(a);

    quickSort(&a);

    cout << "Linked List after sorting \n";
    printList(a);

    return 0;
}
```

Output:

```
Linked List before sorting
```

```
30 3 4 20 5  
Linked List after sorting  
3 4 5 20 30
```

This article is contributed by **Balasubramanian.N**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/quicksort-on-singly-linked-list/>

Chapter 275

QuickSort using Random Pivoting

QuickSort using Random Pivoting - GeeksforGeeks

In this article we will discuss how to implement [QuickSort](#) using random pivoting. In QuickSort we first partition the array in place such that all elements to the left of the pivot element are smaller, while all elements to the right of the pivot are greater than the pivot. Then we recursively call the same procedure for left and right subarrays.

Unlike [merge sort](#) we don't need to merge the two sorted arrays. Thus Quicksort requires lesser auxillary space than Merge Sort, which is why it is often preferred to Merge Sort. Using a randomly generated pivot we can further improve the time complexity of QuickSort.

We have discussed at two popular methods for partitioning the arrays [Hoare's vs Lomuto partition scheme](#)

It is advised that the reader has read that article or knows how to implement the QuickSort using either of the two partition schemes.

Algorithm for random pivoting using Lomuto Partitioning

```
partition(arr[], lo, hi)
    pivot = arr[hi]
    i = lo      // place for swapping
    for j := lo to hi - 1 do
        if arr[j] <= pivot then
            swap arr[i] with arr[j]
            i = i + 1
    swap arr[i] with arr[hi]
    return i

partition_r(arr[], lo, hi)
```

```
r = Random Number from lo to hi
Swap arr[r] and arr[hi]
return partition(arr, lo, hi)

quicksort(arr[], lo, hi)
if lo < hi
    p = partition_r(arr, lo, hi)
    quicksort(arr, p-1, hi)
    quicksort(arr, p+1, hi)
```

Algorithm for random pivoting using Hoare Partitioning

```
partition(arr[], lo, hi)
pivot = arr[lo]
i = lo - 1 // Initialize left index
j = hi + 1 // Initialize right index

// Find a value in left side greater
// than pivot
do
    i = i + 1
while arr[i] == pivot

if i >= j then
    return j

swap arr[i] with arr[j]

partition_r(arr[], lo, hi)
r = Random number from lo to hi
Swap arr[r] and arr[lo]
return partition(arr, lo, hi)

quicksort(arr[], lo, hi)
if lo < hi
    p = partition_r(arr, lo, hi)
    quicksort(arr, p, hi)
    quicksort(arr, p+1, hi)
```

Below is the CPP implementation of the Algorithms

Lomuto (C++)

```
/* C++ implementation QuickSort using Lomuto's partition
Scheme.*/
#include <cstdlib>
#include <iostream>
```

```
using namespace std;

/* This function takes last element as pivot, places
   the pivot element at its correct position in sorted
   array, and places all smaller (smaller than pivot)
   to left of pivot and all greater elements to right
   of pivot */
int partition(int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Index of smaller element

    for (int j = low; j <= high - 1; j++) {

        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot) {

            i++; // increment index of smaller element
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

// Generates Random Pivot, swaps pivot with
// end element and calls the partition function
int partition_r(int arr[], int low, int high)
{
    // Generate a random number in between
    // low .. high
    srand(time(NULL));
    int random = low + rand() % (high - low);

    // Swap A[random] with A[high]
    swap(arr[random], arr[high]);

    return partition(arr, low, high);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high) {
```

```
/* pi is partitioning index, arr[p] is now
at right place */
int pi = partition_r(arr, low, high);

// Separately sort elements before
// partition and after partition
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = { 10, 7, 8, 9, 1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Hoare (C++)

```
/* C++ implementation of QuickSort using Hoare's
partition scheme. */
#include <cstdlib>
#include <iostream>
using namespace std;

/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition(int arr[], int low, int high)
{
    int pivot = arr[low];
```

```
int i = low - 1, j = high + 1;

while (true) {

    // Find leftmost element greater than
    // or equal to pivot
    do {
        i++;
    } while (arr[i] < pivot);

    // Find rightmost element smaller than
    // or equal to pivot
    do {
        j--;
    } while (arr[j] > pivot);

    // If two pointers met.
    if (i >= j)
        return j;

    swap(arr[i], arr[j]);
}

// Generates Random Pivot, swaps pivot with
// end element and calls the partition function
// In Hoare partition the low element is selected
// as first pivot
int partition_r(int arr[], int low, int high)
{
    // Generate a random number in between
    // low .. high
    srand(time(NULL));
    int random = low + rand() % (high - low);

    // Swap A[random] with A[high]
    swap(arr[random], arr[high]);

    return partition(arr, low, high);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high) {
```

```
/* pi is partitioning index, arr[p] is now
   at right place */
int pi = partition_r(arr, low, high);

// Separately sort elements before
// partition and after partition
quickSort(arr, low, pi);
quickSort(arr, pi + 1, high);
}

/* Function to print an array */
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = { 10, 7, 8, 9, 1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Output:

Sorted array: 1 5 7 8 9 10

[Analysis of Randomized Quick Sort](#)

Notes

- Using random pivoting we improve the expected or average time complexity to $O(N \log N)$. The Worst Case complexity is still $O(N^2)$.

Source

<https://www.geeksforgeeks.org/quicksort-using-random-pivoting/>

Chapter 276

Quickselect Algorithm

Quickselect Algorithm - GeeksforGeeks

Quickselect is a selection algorithm to find the k-th smallest element in an unordered list. It is related to the quick sort sorting algorithm.

Examples:

Input: arr[] = {7, 10, 4, 3, 20, 15}
k = 3
Output: 7

Input: arr[] = {7, 10, 4, 3, 20, 15}
k = 4
Output: 10

The algorithm is similar to QuickSort. The difference is, instead of recurring for both sides (after finding pivot), it recurs only for the part that contains the k-th smallest element. The logic is simple, if index of partitioned element is more than k, then we recur for left part. If index is same as k, we have found the k-th smallest element and we return. If index is less than k, then we recur for right part. This reduces the expected complexity from O(n log n) to O(n), with a worst case of O(n^2).

```
function quickSelect(list, left, right, k)

    if left = right
        return list[left]

    Select a pivotIndex between left and right

    pivotIndex := partition(list, left, right,
```

```

        pivotIndex)
if k = pivotIndex
    return list[k]
else if k < pivotIndex
    right := pivotIndex - 1
else
    left := pivotIndex + 1

// CPP program for implementation of QuickSelect
#include <bits/stdc++.h>
using namespace std;

// Standard partition process of QuickSort().
// It considers the last element as pivot
// and moves all smaller element to left of
// it and greater elements to right
int partition(int arr[], int l, int r)
{
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++) {
        if (arr[j] <= x) {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}

// This function returns k'th smallest
// element in arr[l..r] using QuickSort
// based method. ASSUMPTION: ALL ELEMENTS
// IN ARR[] ARE DISTINCT
int kthSmallest(int arr[], int l, int r, int k)
{
    // If k is smaller than number of
    // elements in array
    if (k > 0 && k <= r - l + 1) {

        // Partition the array around last
        // element and get position of pivot
        // element in sorted array
        int index = partition(arr, l, r);

        // If position is same as k
        if (index - l == k - 1)
            return arr[index];
    }
}

```

```
// If position is more, recur
// for left subarray
if (index - 1 > k - 1)
    return kthSmallest(arr, l, index - 1, k);

// Else recur for right subarray
return kthSmallest(arr, index + 1, r,
                    k - index + 1 - 1);
}

// If k is more than number of
// elements in array
return INT_MAX;
}

// Driver program to test above methods
int main()
{
    int arr[] = { 10, 4, 5, 8, 6, 11, 26 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    cout << "K-th smallest element is "
        << kthSmallest(arr, 0, n - 1, k);
    return 0;
}
```

Output:

```
K-th smallest element is 6
```

Important Points:

1. Like quicksort, it is fast in practice, but has poor worst-case performance. It is used in
2. The partition process is same as QuickSort, only recursive code differs.
3. There exists an algorithm that finds **k-th smallest element in O(n)** in worst case, but QuickSelect performs better on average.

Related C++ function : [std::nth_element in C++](#)

Source

<https://www.geeksforgeeks.org/quickselect-algorithm/>

Chapter 277

Radix Sort

Radix Sort - GeeksforGeeks

The lower bound for Comparison based sorting algorithm (Merge Sort, Heap Sort, Quick-Sort .. etc) is $\Omega(n\log n)$, i.e., they cannot do better than $n\log n$.

Counting sort is a linear time sorting algorithm that sorts in $O(n+k)$ time when elements are in range from 1 to k .

What if the elements are in range from 1 to n^2 ?

We can't use counting sort because counting sort will take $O(n^2)$ which is worse than comparison based sorting algorithms. Can we sort such an array in linear time?

Radix Sort is the answer. The idea of Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort.

The Radix Sort Algorithm

- 1) Do following for each digit i where i varies from least significant digit to the most significant digit.
 - a) Sort input array using counting sort (or any stable sort) according to the i 'th digit.

Example:

Original, unsorted list:

170, 45, 75, 90, 802, 24, 2, 66

Sorting by least significant digit (1s place) gives: [*Notice that we keep 802 before 2, because 802 occurred before 2 in the original list, and similarly for pairs 170 & 90 and 45 & 75.]

170, 90, 802, 2, 24, 45, 75, 66

Sorting by next digit (10s place) gives: [*Notice that 802 again comes before 2 as 802 comes before 2 in the previous list.]

802, 2, 24, 45, 66, 170, 75, 90

Sorting by most significant digit (100s place) gives:

2, 24, 45, 66, 75, 90, 170, 802

What is the running time of Radix Sort?

Let there be d digits in input integers. Radix Sort takes $O(d*(n+b))$ time where b is the base for representing numbers, for example, for decimal system, b is 10. What is the value of d ? If k is the maximum possible value, then d would be $O(\log_b(k))$. So overall time complexity is $O((n+b) * \log_b(k))$. Which looks more than the time complexity of comparison based sorting algorithms for a large k . Let us first limit k . Let $k \leq n^c$ where c is a constant. In that case, the complexity becomes $O(n\log_b(n))$. But it still doesn't beat comparison based sorting algorithms.

What if we make value of b larger?. What should be the value of b to make the time complexity linear? If we set b as n , we get the time complexity as $O(n)$. In other words, we can sort an array of integers with range from 1 to n^c if the numbers are represented in base n (or every digit takes $\log_2(n)$ bits).

Is Radix Sort preferable to Comparison based sorting algorithms like Quick-Sort?

If we have $\log_2 n$ bits for every digit, the running time of Radix appears to be better than Quick Sort for a wide range of input numbers. The constant factors hidden in asymptotic notation are higher for Radix Sort and Quick-Sort uses hardware caches more effectively. Also, Radix sort uses counting sort as a subroutine and counting sort takes extra space to sort numbers.

Implementation of Radix Sort

Following is a simple C++ implementation of Radix Sort. For simplicity, the value of d is assumed to be 10. We recommend you to see [Counting Sort](#) for details of countSort() function in below code.

C/C++

```
// C++ implementation of Radix Sort
#include<iostream>
using namespace std;

// A utility function to get maximum value in arr[]
int getMax(int arr[], int n)
{
    int mx = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > mx)
            mx = arr[i];
    return mx;
}

// A function to do counting sort of arr[] according to
// the digit represented by exp.
void countSort(int arr[], int n, int exp)
{
    int output[n]; // output array
    int i, count[10] = {0};
```

```

// Store count of occurrences in count[]
for (i = 0; i < n; i++)
    count[ (arr[i]/exp)%10 ]++;

// Change count[i] so that count[i] now contains actual
// position of this digit in output[]
for (i = 1; i < 10; i++)
    count[i] += count[i - 1];

// Build the output array
for (i = n - 1; i >= 0; i--)
{
    output[count[ (arr[i]/exp)%10 ] - 1] = arr[i];
    count[ (arr[i]/exp)%10 ]--;
}

// Copy the output array to arr[], so that arr[] now
// contains sorted numbers according to current digit
for (i = 0; i < n; i++)
    arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using
// Radix Sort
void radixsort(int arr[], int n)
{
    // Find the maximum number to know number of digits
    int m = getMax(arr, n);

    // Do counting sort for every digit. Note that instead
    // of passing digit number, exp is passed. exp is  $10^i$ 
    // where i is current digit number
    for (int exp = 1; m/exp > 0; exp *= 10)
        countSort(arr, n, exp);
}

// A utility function to print an array
void print(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver program to test above functions
int main()
{
    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    int n = sizeof(arr)/sizeof(arr[0]);
}

```

```
    radixsort(arr, n);
    print(arr, n);
    return 0;
}
```

Java

```
// Radix sort Java implementation
import java.io.*;
import java.util.*;

class Radix {

    // A utility function to get maximum value in arr[]
    static int getMax(int arr[], int n)
    {
        int mx = arr[0];
        for (int i = 1; i < n; i++)
            if (arr[i] > mx)
                mx = arr[i];
        return mx;
    }

    // A function to do counting sort of arr[] according to
    // the digit represented by exp.
    static void countSort(int arr[], int n, int exp)
    {
        int output[] = new int[n]; // output array
        int i;
        int count[] = new int[10];
        Arrays.fill(count, 0);

        // Store count of occurrences in count[]
        for (i = 0; i < n; i++)
            count[(arr[i]/exp)%10]++;
        
        // Change count[i] so that count[i] now contains
        // actual position of this digit in output[]
        for (i = 1; i < 10; i++)
            count[i] += count[i - 1];

        // Build the output array
        for (i = n - 1; i >= 0; i--)
        {
            output[count[(arr[i]/exp)%10] - 1] = arr[i];
            count[(arr[i]/exp)%10]--;
        }
    }
}
```

```

// Copy the output array to arr[], so that arr[] now
// contains sorted numbers according to current digit
for (i = 0; i < n; i++)
    arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using
// Radix Sort
static void radixsort(int arr[], int n)
{
    // Find the maximum number to know number of digits
    int m = getMax(arr, n);

    // Do counting sort for every digit. Note that instead
    // of passing digit number, exp is passed. exp is 10^i
    // where i is current digit number
    for (int exp = 1; m/exp > 0; exp *= 10)
        countSort(arr, n, exp);
}

// A utility function to print an array
static void print(int arr[], int n)
{
    for (int i=0; i<n; i++)
        System.out.print(arr[i]+" ");
}

/*Driver function to check for above function*/
public static void main (String[] args)
{
    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    int n = arr.length;
    radixsort(arr, n);
    print(arr, n);
}
/*
 * This code is contributed by Devesh Agrawal */

```

Python

```

# Python program for implementation of Radix Sort

# A function to do counting sort of arr[] according to
# the digit represented by exp.
def countingSort(arr, exp1):

    n = len(arr)

```

```
# The output array elements that will have sorted arr
output = [0] * (n)

# initialize count array as 0
count = [0] * (10)

# Store count of occurrences in count[]
for i in range(0, n):
    index = (arr[i]/exp1)
    count[ (index)%10 ] += 1

# Change count[i] so that count[i] now contains actual
# position of this digit in output array
for i in range(1,10):
    count[i] += count[i-1]

# Build the output array
i = n-1
while i>=0:
    index = (arr[i]/exp1)
    output[ count[ (index)%10 ] - 1 ] = arr[i]
    count[ (index)%10 ] -= 1
    i -= 1

# Copying the output array to arr[],
# so that arr now contains sorted numbers
i = 0
for i in range(0,len(arr)):
    arr[i] = output[i]

# Method to do Radix Sort
def radixSort(arr):

    # Find the maximum number to know number of digits
    max1 = max(arr)

    # Do counting sort for every digit. Note that instead
    # of passing digit number, exp is passed. exp is 10^i
    # where i is current digit number
    exp = 1
    while max1/exp > 0:
        countingSort(arr,exp)
        exp *= 10

    # Driver code to test above
    arr = [ 170, 45, 75, 90, 802, 24, 2, 66]
    radixSort(arr)
```

```
for i in range(len(arr)):  
    print(arr[i]),  
  
# This code is contributed by Mohit Kumra
```

Output:

```
2 24 45 66 75 90 170 802
```

Snapshots:

Consider this input array

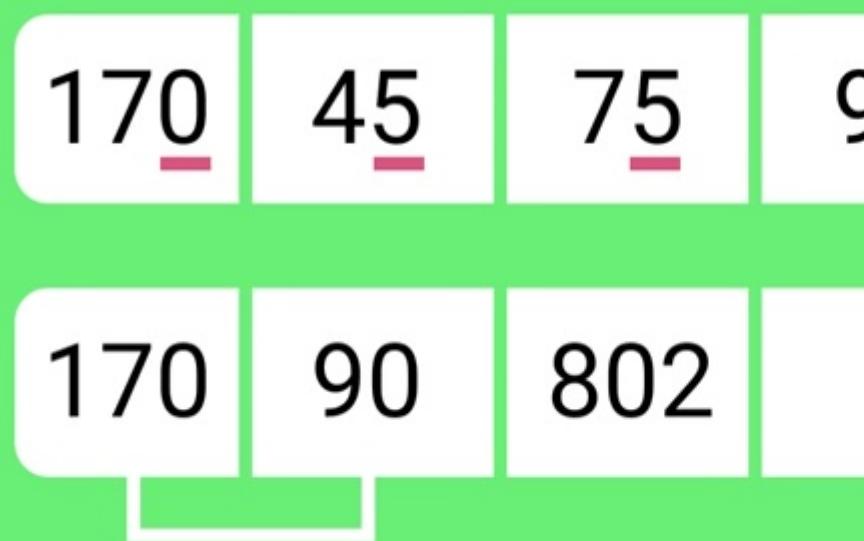


First consider the one's place

Consider this input array

170	45	75	9
170	90	802	

Consider this input array



Observe that 170 has come before 90
because it appeared before 90 in the array.

Consider this input array

170	45	75	9
170	90	802	-
802	2	24	4

Now consider the 100's place

Array is No

2

24

45

6

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

- Selection Sort
- Bubble Sort
- Insertion Sort
- Merge Sort
- Heap Sort
- QuickSort
- Counting Sort
- Bucket Sort
- ShellSort

References:

http://en.wikipedia.org/wiki/Radix_sort
<http://alg12.wikischolars.columbia.edu/file/view/RADIX.pdf>
MIT Video Lecture
Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

Source

<https://www.geeksforgeeks.org/radix-sort/>

Chapter 278

Rank of all elements in an array

Rank of all elements in an array - GeeksforGeeks

Given an array of N integers with duplicates allowed. All elements are ranked from 1 to N if they are distinct. If there are say x repeated elements of a particular value then each element should be assigned a rank equal to the arithmetic mean of x consecutive ranks.

Examples:

Input : 20 30 10

Output : 2.0 3.0 1.0

Input : 10 12 15 12 10 25 12

Output : 1.5, 4.0, 6.0, 4.0, 1.5, 7.0, 4.0

10 is the smallest and there are two 10s so
take the average of two consecutive ranks

1 and 2 i.e. 1.5 . Next smallest element is 12.

Since, two elements are already ranked, the
next rank that can be given is 3. However, there
are three 12's so the rank of 2 is $(3+4+5) / 3 = 4$.
Next smallest element is 15. There is only one 15
so 15 gets a rank of 6 since 5 elements are ranked.
Next element is 25 and it gets a rank of 7.

Input : 1, 2, 5, 2, 1, 60, 3

Output : 1.5, 3.5, 6.0, 3.5, 1.5, 7.0, 5.0

Explanation for first input :

Method I (Simple).

Consider that there are no repeated elements. In such a case the rank of each element is simply $1 + \text{the count of smaller elements in the array}$. Now if the array were to contain repeated elements then modify the ranks by considering the no of equal elements too. If

there are exactly r elements which are less than e and s elements which are equal to e , then e gets the rank given by

$$(r + r+1 + r+2 \dots r+s-1)/s$$

[Separating all r 's and applying
natural number sum formula]
 $= (r*s + s*(s-1)/2)/s$
 $= r + 0.5*(s-1)$

Algorithm

```
function rankify(A)
    N = length of A
    R is the array for storing ranks
    for i in 0..N-1
        r = 1, s = 1
        for j in 0..N-1
            if j != i and A[j] < A[i]
                r += 1
            if j != i and A[j] = A[i]
                s += 1
        // Assign Rank to A[i]
        R[i] = r + 0.5*(s-1)
    return R
```

Implementation of the method is given below

C++

```
// CPP Code to find rank of elements
#include <bits/stdc++.h>
using namespace std;

// Function to find rank
void rankify(int* A , int n) {

    // Rank Vector
    float R[n] = {0};

    // Sweep through all elements in A for each
    // element count the number of less than and
    // equal elements separately in r and s.
    for (int i = 0; i < n; i++) {
        int r = 1, s = 1;
```

```
for (int j = 0; j < n; j++) {  
    if (j != i && A[j] < A[i])  
        r += 1;  
  
    if (j != i && A[j] == A[i])  
        s += 1;  
}  
  
// Use formula to obtain rank  
R[i] = r + (float)(s - 1) / (float) 2;  
  
}  
  
for (int i = 0; i < n; i++)  
    cout << R[i] << ' ';  
  
}  
  
// Driver Code  
int main() {  
    int A[] = {1, 2, 5, 2, 1, 25, 2};  
    int n = sizeof(A) / sizeof(A[0]);  
  
    for (int i = 0; i < n; i++)  
        cout << A[i] << ' ';  
    cout << '\n';  
  
    rankify(A, n);  
  
    return 0;  
}  
  
// This code is contributed by Gitanjali.
```

Java

```
// Java Code to find rank of elements  
public class GfG {  
  
    // Function to print m Maximum elements  
    public static void rankify(int A[], int n)  
    {  
        // Rank Vector  
        float R[] = new float[n];  
  
        // Sweep through all elements in A  
        // for each element count the number
```

```
// of less than and equal elements
// separately in r and s
for (int i = 0; i < n; i++) {
    int r = 1, s = 1;

    for (int j = 0; j < n; j++)
    {
        if (j != i && A[j] < A[i])
            r += 1;

        if (j != i && A[j] == A[i])
            s += 1;
    }

    // Use formula to obtain rank
    R[i] = r + (float)(s - 1) / (float) 2;
}

for (int i = 0; i < n; i++)
    System.out.print(R[i] + " ");

}

// Driver code
public static void main(String args[])
{
    int A[] = {1, 2, 5, 2, 1, 25, 2};
    int n = A.length;

    for (int i = 0; i < n; i++)
        System.out.print(A[i] + " ");
    System.out.println();
    rankify(A, n);
}
}

// This code is contributed by Swetank Modi
```

Python3

```
# Python Code to find
# rank of elements
def rankify(A):

    # Rank Vector
    R = [0 for x in range(len(A))]
```

```
# Sweep through all elements
# in A for each element count
# the number of less than and
# equal elements separately
# in r and s.
for i in range(len(A)):
    (r, s) = (1, 1)
    for j in range(len(A)):
        if j != i and A[j] < A[i]:
            r += 1
        if j != i and A[j] == A[i]:
            s += 1

    # Use formula to obtain rank
    R[i] = r + (s - 1) / 2

# Return Rank Vector
return R

if __name__ == "__main__":
    A = [1, 2, 5, 2, 1, 25, 2]
    print(A)
    print(rankify(A))
```

C#

```
// C# Code to find rank of elements
using System;

public class GfG {

    // Function to print m Maximum
    // elements
    public static void rankify(int []A, int n)
    {
        // Rank Vector
        float []R = new float[n];

        // Sweep through all elements
        // in A for each element count
        // the number of less than and
        // equal elements separately in
        // r and s
        for (int i = 0; i < n; i++) {
            int r = 1, s = 1;

            for (int j = 0; j < n; j++)
            {
```

```
if (j != i && A[j] < A[i])
    r += 1;

if (j != i && A[j] == A[i])
    s += 1;
}

// Use formula to obtain rank
R[i] = r + (float)(s - 1) / (float) 2;

}

for (int i = 0; i < n; i++)
    Console.WriteLine(R[i] + " ");

}

// Driver code
public static void Main()
{
    int []A = {1, 2, 5, 2, 1, 25, 2};
    int n = A.Length;

    for (int i = 0; i < n; i++)
        Console.WriteLine(A[i] + " ");
    Console.WriteLine();
    rankify(A, n);
}
}

// This code is contributed by vt_m.
```

Output

```
[1, 2, 5, 2, 1, 25, 2]
[1.5, 4.0, 6.0, 4.0, 1.5, 7.0, 4.0]
```

Time Complexity is $O(N^2)$, while space complexity is $O(1)$ (excluding space required to hold ranks)

Method II (Efficient)

In this method, create another array (T) of tuples. The first element of the tuple stores the value while the second element refers to the index of the value in the array. Then, sort T in ascending order using the first value of each tuple. Once sorted it is guaranteed that equal elements become adjacent. Then simply walk down T, find the no of adjacent elements and set ranks for each of these elements. Use the second member of each tuple to determine the indices of the values.

Algorithm

```
function rankify_improved(A)
    N = Length of A
    T = Array of tuples (i,j),
        where i = A[i] and j = i
    R = Array for storing ranks
    Sort T in ascending order
    according to i

    for j in 0...N-1
        k = j

        // Find adjacent elements
        while A[k] == A[k+1]
            k += 1

        // No of adjacent elements
        n = k - j + 1

        // Modify rank for each
        // adjacent element
        for j in 0..n-1

            // Get the index of the
            // jth adjacent element
            index = T[i+j][1]
            R[index] = r + (n-1)*0.5

        // Skip n ranks
        r += n

        // Skip n indices
        j += n

    return R
```

The Python implementation of the method is given below
Python3

```
# Python code to find
# rank of elements
def rankify_improved(A):

    # create rank vector
    R = [0 for i in range(len(A))]
```

```
# Create an auxiliary array of tuples
# Each tuple stores the data as well
# as its index in A
T = [(A[i], i) for i in range(len(A))]

# T[] [0] is the data and T[] [1] is
# the index of data in A

# Sort T according to first element
T.sort(key=lambda x: x[0])

(rank, n, i) = (1, 1, 0)

while i < len(A):
    j = i

    # Get no of elements with equal rank
    while j < len(A) - 1 and T[j][0] == T[j + 1][0]:
        j += 1
    n = j - i + 1

    for j in range(n):

        # For each equal element use formula
        # obtain index of T[i+j][0] in A
        idx = T[i+j][1]
        R[idx] = rank + (n - 1) * 0.5

        # Increment rank and i
        rank += n
        i += n

return R

if __name__ == "__main__":
    A = [1, 2, 5, 2, 1, 25, 2]
    print(A)
    print(rankify_improved(A))
```

Output

```
[1, 2, 5, 2, 1, 25, 2]
[1.5, 4.0, 6.0, 4.0, 1.5, 7.0, 4.0]
```

Time Complexity depends on the sorting procedure which is typically $O(N \log N)$. The auxiliary space is $O(N)$, since we need to store the indices as well as values.

Source

<https://www.geeksforgeeks.org/rank-elements-array/>

Chapter 279

Rearrange an array in maximum minimum form Set 1

Rearrange an array in maximum minimum form Set 1 - GeeksforGeeks

Given a sorted array of positive integers, rearrange the array alternately i.e first element should be maximum value, second minimum value, third second max, fourth second min and so on.

Examples:

Input : arr[] = {1, 2, 3, 4, 5, 6, 7}
Output : arr[] = {7, 1, 6, 2, 5, 3, 4}

Input : arr[] = {1, 2, 3, 4, 5, 6}
Output : arr[] = {6, 1, 5, 2, 4, 3}

Expected time complexity is O(n).

The idea is use an auxiliary array. We maintain two pointers one to leftmost or smallest element and other to rightmost or largest element. We move both pointers toward each other and alternatively copy elements at these pointers to an auxiliary array. Finally we copy auxiliary array back to original array.

Below are programs based on above facts.

C++

```
// C++ program to rearrange an array in minimum  
// maximum form  
#include <bits/stdc++.h>  
using namespace std;
```

```
// Prints max at first position, min at second position
// second max at third position, second min at fourth
// position and so on.
void rearrange(int arr[], int n)
{
    // Auxiliary array to hold modified array
    int temp[n];

    // Indexes of smallest and largest elements
    // from remaining array.
    int small=0, large=n-1;

    // To indicate whether we need to copy remaining
    // largest or remaining smallest at next position
    int flag = true;

    // Store result in temp[]
    for (int i=0; i<n; i++)
    {
        if (flag)
            temp[i] = arr[large--];
        else
            temp[i] = arr[small++];

        flag = !flag;
    }

    // Copy temp[] to arr[]
    for (int i=0; i<n; i++)
        arr[i] = temp[i];
}

// Driver program to test above function
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Original Arrayn";
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";

    rearrange(arr, n);

    cout << "nModified Arrayn";
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

}

Java

```
// Java program to rearrange an array in minimum
// maximum form

import java.util.Arrays;

public class GFG
{
    // Prints max at first position, min at second position
    // second max at third position, second min at fourth
    // position and so on.
    static void rearrange(int[] arr, int n)
    {
        // Auxiliary array to hold modified array
        int temp[] = new int[n];

        // Indexes of smallest and largest elements
        // from remaining array.
        int small=0, large=n-1;

        // To indicate whether we need to copy rmaining
        // largest or remaining smallest at next position
        boolean flag = true;

        // Store result in temp[]
        for (int i=0; i<n; i++)
        {
            if (flag)
                temp[i] = arr[large--];
            else
                temp[i] = arr[small++];

            flag = !flag;
        }

        // Copy temp[] to arr[]
        arr = temp.clone();
    }

    // Driver method to test the above function
    public static void main(String[] args)
    {
        int arr[] = new int[]{1, 2, 3, 4, 5, 6, 7, 8, 9};

        System.out.println("Original Array ");
    }
}
```

```
System.out.println(Arrays.toString(arr));  
  
rearrange(arr,arr.length);  
  
System.out.println("Modified Array ");  
System.out.println(Arrays.toString(arr));  
  
}  
}  
  
}
```

Python

```
# Python program to rearrange an array in minimum  
# maximum form  
  
# Prints max at first position, min at second position  
# second max at third position, second min at fourth  
# position and so on.  
def rearrange(arr, n):  
    # Auxiliary array to hold modified array  
    temp = n*[None]  
  
    # Indexes of smallest and largest elements  
    # from remaining array.  
    small,large =0,n-1  
  
    # To indicate whether we need to copy remaining  
    # largest or remaining smallest at next position  
    flag = True  
  
    # Store result in temp[]  
    for i in range(n):  
        if flag is True:  
            temp[i] = arr[large]  
            large -= 1  
        else:  
            temp[i] = arr[small]  
            small += 1  
  
    flag = bool(1-flag)  
  
    # Copy temp[] to arr[]  
    for i in range(n):  
        arr[i] = temp[i]  
    return arr  
  
# Driver program to test above function  
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
n = len(arr)
print("Original Array")
print(arr)
print("Modified Array")
print(rearrange(arr, n))

# This code is contributed by Pratik Chhajer
```

C#

```
// C# program to rearrange
// an array in minimum
// maximum form
using System;

class GFG
{

    // Prints max at first position,
    // min at second position second
    // max at third position, second
    // min at fourth position and so on.
    static void rearrage(int[] arr,
                         int n)
    {
        // Auxiliary array to
        // hold modified array
        int []temp = new int[n];

        // Indexes of smallest
        // and largest elements
        // from remaining array.
        int small = 0, large = n - 1;

        // To indicate whether we
        // need to copy remaining
        // largest or remaining
        // smallest at next position
        bool flag = true;

        // Store result in temp[]
        for (int i = 0; i < n; i++)
        {
            if (flag)
                temp[i] = arr[large--];
            else
                temp[i] = arr[small++];
        }
    }
}
```

```
        flag = !flag;
    }

    // Copy temp[] to arr[]
    for (int i = 0; i < n; i++)
        arr[i] = temp[i];
}

// Driver Code
static void Main()
{
    int[] arr = {1, 2, 3, 4,
                 5, 6, 7, 8, 9};

    Console.WriteLine("Original Array");
    for(int i = 0; i < arr.Length; i++)
        Console.Write(arr[i] + " ");

    rearrage(arr, arr.Length);

    Console.WriteLine("\nModified Array");
    for(int i = 0; i < arr.Length; i++)
        Console.Write(arr[i] + " ");
}
}

// This code is contributed
// by Sam007
```

Output:

```
Original Array
1 2 3 4 5 6 7 8 9
Modified Array
9 1 8 2 7 3 6 4 5
```

Time Complexity : O(n)

Auxiliary Space : O(n)

Exercise : How to solve this problem if extra space is not allowed?

[**Rearrange an array in maximum minimum form Set 2 \(O\(1\) extra space\)**](#)

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/rearrange-array-maximum-minimum-form/>

Chapter 280

Rearrange an array in maximum minimum form Set 2 (O(1) extra space)

Rearrange an array in maximum minimum form Set 2 (O(1) extra space) - GeeksforGeeks

Given a sorted array of positive integers, rearrange the array alternately i.e first element should be the maximum value, second minimum value, third-second max, fourth-second min and so on.

Examples:

Input : arr[] = {1, 2, 3, 4, 5, 6, 7}
Output : arr[] = {7, 1, 6, 2, 5, 3, 4}

Input : arr[] = {1, 2, 3, 4, 5, 6}
Output : arr[] = {6, 1, 5, 2, 4, 3}

We have discussed a solution in below post:

Rearrange an array in maximum minimum form Set 1 : The solution discussed here requires extra space, how to solve this problem with O(1) extra space.

In this post a solution that requires O(n) time and O(1) extra space is discussed. The idea is to use multiplication and modular trick to store two elements at an index.

```
even index : remaining maximum element.  
odd index  : remaining minimum element.  
  
max_index : Index of remaining maximum element
```

```
(Moves from right to left)
min_index : Index of remaining minimum element
            (Moves from left to right)

Initialize: max_index = 'n-1'
            min_index = 0
            max_element = arr[max_index] + 1

For i = 0 to n-1
    If 'i' is even
        arr[i] += arr[max_index] % max_element * max_element
        max_index--
    ELSE // if 'i' is odd
        arr[i] += arr[min_index] % max_element * max_element
        min_index++
```

How does expression “`arr[i] += arr[max_index] % max_element * max_element`” work ?
The purpose of this expression is to store two elements at index `arr[i]`. `arr[max_index]` is stored as multiplier and “`arr[i]`” is stored as remainder. For example in {1 2 3 4 5 6 7 8 9}, `max_element` is 10 and we store 91 at index 0. With 91, we can get original element as $91 \% 10$ and new element as $91 / 10$.

Below implementation of above idea

C++

```
// C++ program to rearrange an array in minimum
// maximum form
#include <bits/stdc++.h>
using namespace std;

// Prints max at first position, min at second position
// second max at third position, second min at fourth
// position and so on.
void rearrange(int arr[], int n)
{
    // initialize index of first minimum and first
    // maximum element
    int max_idx = n - 1, min_idx = 0;

    // store maximum element of array
    int max_elem = arr[n - 1] + 1;

    // traverse array elements
    for (int i = 0; i < n; i++) {
        // at even index : we have to put maximum element
        if (i % 2 == 0) {
            arr[i] += (arr[max_idx] % max_elem) * max_elem;
            max_idx--;
        }
    }
}
```

```
}

// at odd index : we have to put minimum element
else {
    arr[i] += (arr[min_idx] % max_elem) * max_elem;
    min_idx++;
}
}

// array elements back to it's original form
for (int i = 0; i < n; i++)
    arr[i] = arr[i] / max_elem;
}

// Driver program to test above function
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original Array\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    rearrange(arr, n);

    cout << "\nModified Array\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Java

```
// Java program to rearrange an
// array in minimum maximum form

public class Main {

    // Prints max at first position, min at second
    // position second max at third position, second
    // min at fourth position and so on.
    public static void rearrange(int arr[], int n)
    {
        // initialize index of first minimum and first
        // maximum element
        int max_idx = n - 1, min_idx = 0;
```

```
// store maximum element of array
int max_elem = arr[n - 1] + 1;

// traverse array elements
for (int i = 0; i < n; i++) {
    // at even index : we have to put
    // maximum element
    if (i % 2 == 0) {
        arr[i] += (arr[max_idx] % max_elem) * max_elem;
        max_idx--;
    }

    // at odd index : we have to put minimum element
    else {
        arr[i] += (arr[min_idx] % max_elem) * max_elem;
        min_idx++;
    }
}

// array elements back to it's original form
for (int i = 0; i < n; i++)
    arr[i] = arr[i] / max_elem;
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int n = arr.length;

    System.out.println("Original Array");
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");

    rearrange(arr, n);

    System.out.print("\nModified Array\n");
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}

// This code is contributed by Swetank Modi
```

Python3

```
# Python3 program to rearrange an
# array in minimum maximum form
```

```
# Prints max at first position, min at second position
# second max at third position, second min at fourth
# position and so on.
def rearrange(arr, n):

    # Initialize index of first minimum
    # and first maximum element
    max_idx = n - 1
    min_idx = 0

    # Store maximum element of array
    max_elem = arr[n-1] + 1

    # Traverse array elements
    for i in range(0, n) :

        # At even index : we have to put maximum element
        if i % 2 == 0 :
            arr[i] += (arr[max_idx] % max_elem ) * max_elem
            max_idx -= 1

        # At odd index : we have to put minimum element
        else :
            arr[i] += (arr[min_idx] % max_elem ) * max_elem
            min_idx += 1

    # array elements back to it's original form
    for i in range(0, n) :
        arr[i] = arr[i] / max_elem

# Driver Code
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
n = len(arr)

print ("Original Array")

for i in range(0, n):
    print (arr[i], end = " ")

rearrange(arr, n)

print ("\nModified Array")
for i in range(0, n):
    print (int(arr[i]), end = " ")

# This code is contributed by Shreyanshi Arun.
```

C#

```
// C# program to rearrange an
// array in minimum maximum form
using System;

class main {

    // Prints max at first position, min at second
    // position, second max at third position, second
    // min at fourth position and so on.
    public static void rearrange(int[] arr, int n)
    {
        // initialize index of first minimum
        // and first maximum element
        int max_idx = n - 1, min_idx = 0;

        // store maximum element of array
        int max_elem = arr[n - 1] + 1;

        // traverse array elements
        for (int i = 0; i < n; i++) {

            // at even index : we have to put
            // maximum element
            if (i % 2 == 0) {
                arr[i] += (arr[max_idx] % max_elem) * max_elem;
                max_idx--;
            }

            // at odd index : we have to
            // put minimum element
            else {
                arr[i] += (arr[min_idx] % max_elem) * max_elem;
                min_idx++;
            }
        }

        // array elements back to it's original form
        for (int i = 0; i < n; i++)
            arr[i] = arr[i] / max_elem;
    }

    // Driver code
    public static void Main()
    {
        int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        int n = arr.Length;
```

```
Console.WriteLine("Original Array");
for (int i = 0; i < n; i++)
    Console.Write(arr[i] + " ");
Console.WriteLine();

rearrange(arr, n);

Console.WriteLine("Modified Array");
for (int i = 0; i < n; i++)
    Console.Write(arr[i] + " ");
}

}

// This code is contributed by vt_m.
```

Output :

```
Original Array
1 2 3 4 5 6 7 8 9
Modified Array
9 1 8 2 7 3 6 4 5
```

Thanks Saurabh Srivastava and Gaurav Ahirwar for suggesting this approach.

Another Approach: A simpler approach will be to observe indexing positioning of maximum elements and minimum elements. The even index stores maximum elements and the odd index stores the minimum elements. With every increasing index, the maximum element decreases by one and the minimum element increases by one. A simple traversal can be done and arr[] can be filled in again.

Below is the implementation of the above approach:

C++

```
// C++ program to rearrange an array in minimum
// maximum form
#include <bits/stdc++.h>
using namespace std;

// Prints max at first position, min at second position
// second max at third position, second min at fourth
// position and so on.
void rearrange(int arr[], int n)
{
    // initialize index of first minimum and first
    // maximum element
    int max_ele = arr[n - 1];
    int min_ele = arr[0];
    // traverse array elements
```

```
for (int i = 0; i < n; i++) {
    // at even index : we have to put maximum element
    if (i % 2 == 0) {
        arr[i] = max_ele;
        max_ele -= 1;
    }

    // at odd index : we have to put minimum element
    else {
        arr[i] = min_ele;
        min_ele += 1;
    }
}

// Driver program to test above function
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original Array\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    rearrange(arr, n);

    cout << "\nModified Array\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Java

```
// Java program to rearrange an
// array in minimum maximum form

public class Main {

    // Prints max at first position, min at second
    // position second max at third position, second
    // min at fourth position and so on.
    public static void rearrange(int arr[], int n)
    {
        // initialize index of first minimum and first
        // maximum element
        int max_ele = arr[n - 1];
```

```
int min_ele = arr[0];
// traverse array elements
for (int i = 0; i < n; i++) {
    // at even index : we have to put maximum element
    if (i % 2 == 0) {
        arr[i] = max_ele;
        max_ele -= 1;
    }

    // at odd index : we have to put minimum element
    else {
        arr[i] = min_ele;
        min_ele += 1;
    }
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int n = arr.length;

    System.out.println("Original Array");
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");

    rearrange(arr, n);

    System.out.print("\nModified Array\n");
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}
```

C#

```
// C# program to rearrange
// an array in minimum
// maximum form
using System;

class GFG
{
    // Prints max at first
    // position, min at second
    // position second max at
    // third position, second
```

```
// min at fourth position
// and so on.
public static void rearrange(int []arr,
                             int n)
{
    // initialize index of
    // first minimum and
    // first maximum element
    int max_ele = arr[n - 1];
    int min_ele = arr[0];

    // traverse array elements
    for (int i = 0; i < n; i++)
    {
        // at even index : we have
        // to put maximum element
        if (i % 2 == 0)
        {
            arr[i] = max_ele;
            max_ele -= 1;
        }

        // at odd index : we have
        // to put minimum element
        else
        {
            arr[i] = min_ele;
            min_ele += 1;
        }
    }
}

// Driver code
static public void Main ()
{
    int []arr = {1, 2, 3, 4,
                5, 6, 7, 8, 9};
    int n = arr.Length;

    Console.WriteLine("Original Array");
    for (int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");

    rearrange(arr, n);

    Console.WriteLine("\nModified Array\n");
    for (int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
}
```

```
    }  
}  
  
// This code is contributed by ajit
```

Output :

```
Original Array  
1 2 3 4 5 6 7 8 9  
Modified Array  
9 1 8 2 7 3 6 4 5
```

Thanks **Apollo Doley** for suggesting this approach.

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/rearrange-array-maximum-minimum-form-set-2-o1-extra-space/>

Chapter 281

Rearrange an array in order – smallest, largest, 2nd smallest, 2nd largest, ..

Rearrange an array in order - smallest, largest, 2nd smallest, 2nd largest, .. - GeeksforGeeks

Given an array of integers, task is to print the array in the order – smallest number, Largest number, 2nd smallest number, 2nd largest number, 3rd smallest number, 3rd largest number and so on.....

Examples:

Input : arr[] = [5, 8, 1, 4, 2, 9, 3, 7, 6]
Output :arr[] = {1, 9, 2, 8, 3, 7, 4, 6, 5}

Input : arr[] = [1, 2, 3, 4]
Output :arr[] = {1, 4, 2, 3}

A **simple solution** is to first find the smallest element, swap it with first element. Then find largest element, swap it with second element and so on. Time complexity of this solution is $O(n^2)$.

An **efficient solution** is to use [sorting](#).

1. Sort the elements of array.
2. Take two variables say i and j and point them to the first and last index of the array respectively.
3. Now run a loop and store the elements in the array one by one by incrementing i and decrementing j.

Let's take an array with input 5, 8, 1, 4, 2, 9, 3, 7, 6 and sort them so the array become 1, 2, 3, 4, 5, 6, 7, 8, 9. Now take two variables say i and j and point them to the first and last

index of the array respectively, run a loop and store value into new array by incrementing i and decrementing j. We get final result as 1 9 2 8 3 7 4 6 5.

C++

```
// C++ program to print the array in given order
#include <bits/stdc++.h>
using namespace std;

// Function which arrange the array.
void rearrangeArray(int arr[], int n)
{
    // Sorting the array elements
    sort(arr, arr + n);

    int tempArr[n]; // To store modified array

    // Adding numbers from sorted array to
    // new array accordingly
    int ArrIndex = 0;

    // Traverse from begin and end simultaneously
    for (int i = 0, j = n-1; i <= n / 2 || j > n / 2; i++, j--) {
        tempArr[ArrIndex] = arr[i];
        ArrIndex++;
        tempArr[ArrIndex] = arr[j];
        ArrIndex++;
    }

    // Modifying original array
    for (int i = 0; i < n; i++)
        arr[i] = tempArr[i];
}

// Driver Code
int main()
{
    int arr[] = { 5, 8, 1, 4, 2, 9, 3, 7, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    rearrangeArray(arr, n);

    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

return 0;
}
```

Java

```
// Java program to print the array in given order
import java.util.Arrays;

public class GFG {

    // Function which arrange the array.
    static void rearrangeArray(int arr[], int n)
    {
        // Sorting the array elements
        Arrays.sort(arr);

        int[] tempArr = new int[n]; // To store modified array

        // Adding numbers from sorted array to
        // new array accordingly
        int ArrIndex = 0;

        // Traverse from begin and end simultaneously
        for (int i = 0, j = n-1; i <= n / 2 || j > n / 2;
             i++, j--) {
            if(ArrIndex < n)
            {
                tempArr[ArrIndex] = arr[i];
                ArrIndex++;
            }

            if(ArrIndex < n)
            {
                tempArr[ArrIndex] = arr[j];
                ArrIndex++;
            }
        }

        // Modifying original array
        for (int i = 0; i < n; i++)
            arr[i] = tempArr[i];
    }

    // Driver Code
    public static void main(String args[])
    {
        int arr[] = { 5, 8, 1, 4, 2, 9, 3, 7, 6 };
        int n = arr.length;
        rearrangeArray(arr, n);

        for (int i = 0; i < n; i++)
            System.out.print(arr[i]+" ");
    }
}
```

```
}
```

// This code is contributed by Sumit Ghosh

Python3

```
# Python 3 program to print
# the array in given order

# Function which arrange the
# array.
def rearrangeArray(arr, n) :

    # Sorting the array elements
    arr.sort()

    # To store modified array
    tempArr = [0] * (n + 1)

    # Adding numbers from sorted
    # array to new array accordingly
    ArrIndex = 0

    # Traverse from begin and end
    # simultaneously
    i = 0
    j = n-1

    while(i <= n // 2 or j > n // 2) :

        tempArr[ArrIndex] = arr[i]
        ArrIndex = ArrIndex + 1
        tempArr[ArrIndex] = arr[j]
        ArrIndex = ArrIndex + 1
        i = i + 1
        j = j - 1

    # Modifying original array
    for i in range(0, n) :
        arr[i] = tempArr[i]

# Driver Code
arr = [ 5, 8, 1, 4, 2, 9, 3, 7, 6 ]
n = len(arr)
rearrangeArray(arr, n)

for i in range(0, n) :
    print( arr[i], end = " ")
```

```
# This code is contributed by Nikita Tiwari.
```

C#

```
// C# program to print the
// array in given order
using System;

public class GFG {

    // Function which arrange the array.
    static void rearrangeArray(int []arr, int n)
    {
        // Sorting the array elements
        Array.Sort(arr);

        // To store modified array
        int []tempArr = new int[n];

        // Adding numbers from sorted array
        // to new array accordingly
        int ArrIndex = 0;

        // Traverse from begin and end simultaneously
        for (int i = 0, j = n-1; i <= n / 2
             || j > n / 2; i++, j--) {

            if(ArrIndex < n)
            {
                tempArr[ArrIndex] = arr[i];
                ArrIndex++;
            }

            if(ArrIndex < n)
            {
                tempArr[ArrIndex] = arr[j];
                ArrIndex++;
            }
        }

        // Modifying original array
        for (int i = 0; i < n; i++)
            arr[i] = tempArr[i];
    }

    // Driver Code
    public static void Main(String []args)
```

```
{  
    int []arr = {5, 8, 1, 4, 2, 9, 3, 7, 6};  
    int n = arr.Length;  
    rearrangeArray(arr, n);  
  
    for (int i = 0; i < n; i++)  
        Console.Write(arr[i] + " ");  
}  
}  
  
// This code is contributed by Nitin Mittal.
```

PHP

```
<?php  
// PHP program to print  
// the array in given order  
  
// Function which  
// arrange the array.  
function rearrangeArray($arr, $n)  
{  
    // Sorting the  
    // array elements  
    sort($arr);  
  
    // To store modified array  
    $tempArr = array($n);  
  
    // Adding numbers from  
    // sorted array to new  
    // array accordingly  
    $ArrIndex = 0;  
  
    // Traverse from begin  
    // and end simultaneously  
    for ($i = 0, $j = $n - 1;  
         $i <= $n / 2 || $j > $n / 2;  
         $i++, $j--)  
    {  
        $tempArr[$ArrIndex] = $arr[$i];  
        $ArrIndex++;  
        $tempArr[$ArrIndex] = $arr[$j];  
        $ArrIndex++;  
    }  
  
    // Modifying original array  
    for ($i = 0; $i < $n; $i++)
```

```
$arr[$i] = $tempArr[$i];  
  
for ($i = 0; $i < $n; $i++)  
    echo $arr[$i] . " ";  
}  
  
// Driver Code  
$arr = array(5, 8, 1, 4, 2,  
            9, 3, 7, 6 );  
$n = count($arr) ;  
rearrangeArray($arr, $n);  
  
// This code is contributed  
// by Sam007  
?>
```

Output:

1 9 2 8 3 7 4 6 5

Time Complexity : $O(n \log n)$
Auxiliary Space : $O(n)$

Improved By : nitin mittal, Sam007

Source

<https://www.geeksforgeeks.org/rearrange-array-order-smallest-largest-2nd-smallest-2nd-largest/>

Chapter 282

Rearrange an array to minimize sum of product of consecutive pair elements

Rearrange an array to minimize sum of product of consecutive pair elements - GeeksforGeeks

We are given an array of even size, we have to sort the array in such a way that the sum of product of alternate elements is minimum also we have to find that minimum sum.

Examples:

```
Input : arr[] = {9, 2, 8, 4, 5, 7, 6, 0}
Output : Minimum sum of the product of
         consecutive pair elements: 74
         Sorted arr[] for minimum sum:
         {9, 0, 8, 2, 7, 4, 6, 5}
Explanation : We get 74 using below
calculation in rearranged array.
9*0 + 8*2 + 7*4 + 6*5 = 74
```

```
Input : arr[] = {1, 2, 1, 4, 0, 5, 6, 0}
Output : Minimum sum of the product of
         consecutive pair elements: 6
         Sorted arr[] for minimum sum:
         {6, 0, 5, 0, 4, 1, 2, 1}
Explanation : We get 6 using below:
6*0 + 5*0 + 4*1 + 2*1 = 6
```

This problem is a variation of [Minimize the sum of product of two arrays with permutations allowed.](#)

For rearranging the array in such a way that we should get the sum of the product of consecutive element pairs is minimum we should have all even index element in decreasing and odd index element in increasing order with all $n/2$ maximum elements as even indexed and next $n/2$ elements as odd indexed or vice-versa.

Now, for that our idea is simple, we should sort the main array and further create two auxiliary arrays evenArr[] and oddArr[] respectively. We traverse input array and put $n/2$ maximum elements in evenArr[] and next $n/2$ elements in oddArr[]. Then we sort evenArr[] in descending and oddArr[] in ascending order. Finally, copy evenArr[] and oddArr[] element by element to get the required result and should calculate the minimum required sum.

```
// Program to sort an array such that
// sum of product of alternate element
// is minimum.
#include <bits/stdc++.h>
using namespace std;

int minSum(int arr[], int n)
{
    // create evenArr[] and oddArr[]
    vector<int> evenArr;
    vector<int> oddArr;

    // sort main array in ascending order
    sort(arr, arr+n);

    // Put elements in oddArr[] and evenArr[]
    // as per desired value.
    for (int i = 0; i < n; i++)
    {
        if (i < n/2)
            oddArr.push_back(arr[i]);
        else
            evenArr.push_back(arr[i]);
    }

    // sort evenArr[] in descending order
    sort(evenArr.begin(), evenArr.end(), greater<int>());

    // merge both sub-array and
    // calculate minimum sum of
    // product of alternate elements
    int i = 0, sum = 0;
    for (int j=0; j<evenArr.size(); j++)
    {
        arr[i++] = evenArr[j];
        arr[i++] = oddArr[j];
        sum += evenArr[j] * oddArr[j];
    }
}
```

```
    return sum;
}

// Driver Program
int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Minimum required sum = " << minSum(arr, n);
    cout << "\nSorted array in required format : ";
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Output:

```
Minimum required sum = 60
Sorted array in required format : 9 0 8 1 7 2 6 3 5 4
```

Source

<https://www.geeksforgeeks.org/rearrange-array-minimize-sum-product-consecutive-pair-elements/>

Chapter 283

Rearrange array such that even positioned are greater than odd

Rearrange array such that even positioned are greater than odd - GeeksforGeeks

Given an array A of n elements, sort the array according to the following relations :

$A[i] \geq A[i+1]$, if i is even.

$A[i] \leq A[i+1]$, if i is odd.

Print the resultant array.

Examples :

Input : A[] = {1, 2, 2, 1}

Output : 1 2 1 2

Explanation :

For 1st element, 1 < 1, i = 2 is even.

3rd element, 1 < 1, i = 4 is even.

Input : A[] = {1, 3, 2}

Output : 1 3 2

Explanation :

Here, the array is also sorted as per the conditions.

1 < 1 and 2 < 3.

Observe that array consists of $[n/2]$ even positioned elements. If we assign largest $[n/2]$ elements to the even positions and rest of the elements to the odd positions, our problem is solved. Because element at odd position will always be less than the element at even position as it is maximum element and vice versa. Sort the array and assign the first $[n/2]$ elements at even positions.

Below is the implementation of above approach:

C++

```
// C++ program to rearrange the elements
// in array such that even positioned are
// greater than odd positioned elements
#include<bits/stdc++.h>
using namespace std;

void assign(int a[], int n)
{
    // Sort the array
    sort(a, a + n);

    int ans[n];
    int p = 0, q = n - 1;
    for (int i = 0; i < n; i++)
    {
        // Assign even indexes with maximum elements
        if ((i + 1) % 2 == 0)
            ans[i] = a[q--];

        // Assign odd indexes with remaining elements
        else
            ans[i] = a[p++];
    }

    // Print result
    for (int i = 0; i < n; i++)
        cout << ans[i] << " ";
}

// Driver Code
int main()
{
    int A[] = { 1, 3, 2, 2, 5 };
    int n = sizeof(A) / sizeof(A[0]);
    assign(A, n);
    return 0;
}
```

Java

```
// Java program to rearrange the elements
// in array such that even positioned are
// greater than odd positioned elements
import java.io.*;
```

```
import java.util.*;

class GFG {

    static void assign(int a[], int n) {

        // Sort the array
        Arrays.sort(a);

        int ans[] = new int[n];
        int p = 0, q = n - 1;
        for (int i = 0; i < n; i++) {

            // Assign even indexes with maximum elements
            if ((i + 1) % 2 == 0)
                ans[i] = a[q--];

            // Assign odd indexes with remaining elements
            else
                ans[i] = a[p++];
        }

        // Print result
        for (int i = 0; i < n; i++)
            System.out.print(ans[i] + " ");
    }

    // Driver code
    public static void main(String args[]) {
        int A[] = {1, 3, 2, 2, 5};
        int n = A.length;
        assign(A, n);
    }
}

// This code is contributed by Nikita Tiwari.
```

Python3

```
# Python3 code to rearrange the
# elements in array such that
# even positioned are greater
# than odd positioned elements

def assign(a, n):

    # Sort the array
    a.sort()
```

```
ans = [0] * n
p = 0
q = n - 1
for i in range(n):

    # Assign even indexes with
    # maximum elements
    if (i + 1) % 2 == 0:
        ans[i] = a[q]
        q = q - 1

    # Assign odd indexes with
    # remaining elements
    else:
        ans[i] = a[p]
        p = p + 1

# Print result
for i in range(n):
    print(ans[i], end = " ")

# Driver Code
A = [ 1, 3, 2, 2, 5 ]
n = len(A)
assign(A, n)

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# program to rearrange the elements
// in array such that even positioned are
// greater than odd positioned elements
using System;

class GFG {

    static void assign(int[] a, int n)
    {
        // Sort the array
        Array.Sort(a);

        int[] ans = new int[n];
        int p = 0, q = n - 1;
        for (int i = 0; i < n; i++) {

            // Assign even indexes with maximum elements
```

```
        if ((i + 1) % 2 == 0)
            ans[i] = a[q--];

        // Assign odd indexes with remaining elements
        else
            ans[i] = a[p++];
    }

    // Print result
    for (int i = 0; i < n; i++)
        Console.Write(ans[i] + " ");
}

// Driver code
public static void Main()
{
    int[] A = { 1, 3, 2, 2, 5 };
    int n = A.Length;
    assign(A, n);
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to rearrange
// the elements in array such
// that even positioned are
// greater than odd positioned
// elements

function assign($a, $n)
{

    // Sort the array
    sort($a);

    $p = 0; $q = $n - 1;
    for ($i = 0; $i < $n; $i++)
    {

        // Assign even indexes
        // with maximum elements
        if (($i + 1) % 2 == 0)
            $ans[$i] = $a[$q--];
    }
}
```

```
// Assign odd indexes
// with remaining elements
else
    $ans[$i] = $a[$p++];
}

// Print result
for ($i = 0; $i < $n; $i++)
    echo($ans[$i] . " ");
}

// Driver Code
$A = array( 1, 3, 2, 2, 5 );
$n = sizeof($A);
assign($A, $n);

// This code is contributed by Ajit.
?>
```

Output:

1 5 2 3 2

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/rearrange-array-such-that-even-positioned-are-greater-than-odd/>

Chapter 284

Rearrange positive and negative numbers using inbuilt sort function

Rearrange positive and negative numbers using inbuilt sort function - GeeksforGeeks

Given an array of positive and negative numbers, arrange them such that all negative integers appear before all the positive integers in the array without using any additional data structure like hash table, arrays, etc. The order of appearance should be maintained.

Examples:

```
Input : arr[] = [12, 11, -13, -5, 6, -7, 5, -3, -6]
Output : arr[] = [-13, -5, -7, -3, -6, 12, 11, 6, 5]
```

```
Input : arr[] = [-12, 11, 0, -5, 6, -7, 5, -3, -6]
Output : arr[] = [-12, -5, -7, -3, -6, 11, 0, 6, 5]
```

Previous Approaches : Some of the approaches have already been discussed [here](#). They were implemented at best.

Another Approach : There is another method to do so. In c++ STL, There is an inbuilt function [`std::sort\(\)`](#). We can modify the `comp()` function to obtain the desired result. As we have to place negative numbers first and then positive numbers. We also have to keep zero's(if present) between positive and negative numbers.

The `comp()` function in this code rearranges the given array in required order. Here in **bool comp(int a, int b)**, if integer 'a' is of j-th index and integer 'b' is of i-th index elements in the `arr[]`, then $j > i$. `comp()` function will be called in this way. If the `comp()` return true then swap will be done.

```
// CPP program to rearrange positive and negative
// integers keeping order of elements.
#include <bits/stdc++.h>

using namespace std;

bool comp(int a, int b)
{
    //swap not needed
    if((a>0 && b>0) || (a<0 && b<0) || (a>0 && b<0) )
    return false;

    //swap needed
    if(a<0 && b>0)
    return true;

    //swap not needed
    if((a==0 && b<0) || (a>0 && b==0))
    return false;

    //swap needed
    if((a==0 && b>0) || (a<0 && b==0) )
    return true;
}

void rearrange(int arr[], int n)
{
    sort(arr, arr + n, comp);
}

// Driver code
int main()
{
    int arr[] = { -12, 11, -13, -5, 6, -7, 5, -3, -6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    rearrange(arr, n);
    for (int i = 0; i < n; i++)
        cout << " " << arr[i];

    return 0;
}
```

Output:

```
-12 -13 -5 -7 -3 -6 11 6 5
```

Time complexity is same as sorting i.e. $O(n \log n)$. As we are using standard sort function. But it is really faster, because inbuilt sort function uses [introsort](#).

Improved By : [Siva Saindra Reddy K](#)

Source

<https://www.geeksforgeeks.org/rearrange-positive-negative-numbers-using-inbuilt-sort-function/>

Chapter 285

Rearrange positive and negative numbers with constant extra space

Rearrange positive and negative numbers with constant extra space - GeeksforGeeks

Given an array of positive and negative numbers, arrange them such that all negative integers appear before all the positive integers in the array **without using any additional data structure** like hash table, arrays, etc. The order of appearance should be maintained.

Examples:

Input: [12 11 -13 -5 6 -7 5 -3 -6]

Output: [-13 -5 -7 -3 -6 12 11 6 5]

A simple solution is to use another array. We copy all elements of original array to new array. We then traverse the new array and copy all negative and positive elements back in original array one by one. This approach is discussed [here](#). The problem with this approach is that it uses auxiliary array and we're not allowed to use any data structure to solve this problem.

One approach that does not use any data structure is to use use partition process of Quick-Sort. The idea is to consider 0 as pivot and divide the array around it. The problem with this approach is that it changes relative order of elements. The similar partition process is discussed [here](#).

Let's now discuss few methods which do not use any other data structure and also preserves relative order of elements.

Approach 1: Modified Insertion Sort

We can modify [insertion sort](#) to solve this problem.

Algorithm –

Loop from $i = 1$ to $n - 1$.

- a) If the current element is positive, do nothing.
- b) If the current element $arr[i]$ is negative, we insert it into sequence $arr[0..i-1]$ such that all positive elements in $arr[0..i-1]$ are shifted one position to their right and $arr[i]$ is inserted at index of first positive element.

Below is the implementation –

C++

```
// C++ program to Rearrange positive and negative
// numbers in a array
#include <stdio.h>

// A utility function to print an array of size n
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Function to Rearrange positive and negative
// numbers in a array
void RearrangePosNeg(int arr[], int n)
{
    int key, j;
    for(int i = 1; i < n; i++)
    {
        key = arr[i];

        // if current element is positive
        // do nothing
        if (key > 0)
            continue;

        /* if current element is negative,
        shift positive elements of arr[0..i-1],
        to one position to their right */
        j = i - 1;
        while (j >= 0 && arr[j] > 0)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
    }
}
```

```
// Put negative element at its right position
arr[j + 1] = key;
}

/* Driver program to test above functions */
int main()
{
    int arr[] = { -12, 11, -13, -5, 6, -7, 5, -3, -6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    RearrangePosNeg(arr, n);
    printArray(arr, n);

    return 0;
}
```

Java

```
// Java program to Rearrange positive
// and negative numbers in a array
import java.io.*;

class GFG
{
    // A utility function to print
    // an array of size n
    static void printArray(int arr[], int n)
    {
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Function to Rearrange positive and negative
    // numbers in a array
    static void RearrangePosNeg(int arr[], int n)
    {
        int key, j;
        for(int i = 1; i < n; i++)
        {
            key = arr[i];

            // if current element is positive
            // do nothing
            if (key > 0)
                continue;

            j = 0;
```

```
/* if current element is negative,
shift positive elements of arr[0..i-1],
to one position to their right */
j = i - 1;
while (j >= 0 && arr[j] > 0)
{
    arr[j + 1] = arr[j];
    j = j - 1;
}

// Put negative element at its right position
arr[j + 1] = key;
}

// Driver program
public static void main (String[] args)
{
    int arr[] = { -12, 11, -13, -5, 6, -7, 5, -3, -6 };
    int n = arr.length;
    RearrangePosNeg(arr, n);
    printArray(arr, n);

}
}

// This code is contributed by vt_m.
```

C#

```
// C# program to Rearrange positive
// and negative numbers in a array
using System;

class GFG {

    // A utility function to print
    // an array of size n
    static void printArray(int[] arr, int n)
    {
        for (int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");
        Console.WriteLine();
    }

    // Function to Rearrange positive and negative
    // numbers in a array
    static void RearrangePosNeg(int[] arr, int n)
```

```

{
    int key, j;
    for (int i = 1; i < n; i++) {
        key = arr[i];

        // if current element is positive
        // do nothing
        if (key > 0)
            continue;

        /* if current element is negative,
        shift positive elements of arr[0..i-1],
        to one position to their right */
        j = i - 1;
        while (j >= 0 && arr[j] > 0) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }

        // Put negative element at its right position
        arr[j + 1] = key;
    }
}

// Driver program
public static void Main()
{
    int[] arr = { -12, 11, -13, -5, 6,
                  -7, 5, -3, -6 };
    int n = arr.Length;
    RearrangePosNeg(arr, n);
    printArray(arr, n);
}
}

// This code is contributed by vt_m.

```

Output:

-12 -13 -5 -7 -3 -6 11 6 5

Time complexity of above solution is $O(n^2)$ and auxiliary space is $O(1)$. We have maintained the order of appearance and have not used any other data structure.

Approach 2: Optimized Merge Sort

Merge method of standard merge sort algorithm can be modified to solve this problem. While merging two sorted halves say left and right, we need to merge in such a way that

negative part of left and right sub-array is copied first followed by positive part of left and right sub-array.

Below is the implementation of the idea –

C++

```
// C++ program to Rearrange positive and negative
// numbers in a array
#include <iostream>
using namespace std;

/* Function to print an array */
void printArray(int A[], int size)
{
    for(int i = 0; i < size; i++)
        cout << A[i] << " ";
    cout << endl;
}

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray

    /* Note the order of appearance of elements should
     * be maintained - we copy elements of left subarray
     * first followed by that of right subarray

    // copy negative elements of left subarray
    while (i < n1 && L[i] < 0)
        arr[k++] = L[i++];
```

```
// copy negative elements of right subarray
while (j < n2 && R[j] < 0)
    arr[k++] = R[j++];

// copy positive elements of left subarray
while (i < n1)
    arr[k++] = L[i++];

// copy positive elements of right subarray
while (j < n2)
    arr[k++] = R[j++];
}

// Function to Rearrange positive and negative
// numbers in a array
void RearrangePosNeg(int arr[], int l, int r)
{
    if(l < r)
    {
        // Same as (l + r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - 1) / 2;

        // Sort first and second halves
        RearrangePosNeg(arr, l, m);
        RearrangePosNeg(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

/* Driver program to test above functions */
int main()
{
    int arr[] = { -12, 11, -13, -5, 6, -7, 5, -3, -6 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    RearrangePosNeg(arr, 0, arr_size - 1);

    printArray(arr, arr_size);

    return 0;
}

Java

// Java program to Rearrange positive
```

```
// and negative numbers in a array
import java.io.*;

class GFG
{
    /* Function to print an array */
    static void printArray(int A[], int size)
    {
        for(int i = 0; i < size; i++)
            System.out.print(A[i] + " ");
        System.out.println();
    }

    // Merges two subarrays of arr[].
    // First subarray is arr[l..m]
    // Second subarray is arr[m+1..r]
    static void merge(int arr[], int l, int m, int r)
    {
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;

        /* Create temp arrays */
        int L[] = new int[n1];
        int R[] = new int[n2];

        /* Copy data to temp arrays L[] and R[] */
        for (i = 0; i < n1; i++)
            L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
            R[j] = arr[m + 1 + j];

        /* Merge the temp arrays back into arr[l..r]*/
        // Initial index of first subarray
        i = 0;

        // Initial index of second subarray
        j = 0;

        // Initial index of merged subarray
        k = l;

        // Note the order of appearance of elements should
        // be maintained - we copy elements of left subarray
        // first followed by that of right subarray

        // copy negative elements of left subarray
        while (i < n1 && L[i] < 0)
```

```
arr[k++] = L[i++];

// copy negative elements of right subarray
while (j < n2 && R[j] < 0)
    arr[k++] = R[j++];

// copy positive elements of left subarray
while (i < n1)
    arr[k++] = L[i++];

// copy positive elements of right subarray
while (j < n2)
    arr[k++] = R[j++];
}

// Function to Rearrange positive and negative
// numbers in a array
static void RearrangePosNeg(int arr[], int l, int r)
{
    if(l < r)
    {
        // Same as (l + r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - 1) / 2;

        // Sort first and second halves
        RearrangePosNeg(arr, l, m);
        RearrangePosNeg(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

// Driver program
public static void main (String[] args)
{
    int arr[] = { -12, 11, -13, -5, 6, -7, 5, -3, -6 };
    int arr_size = arr.length;
    RearrangePosNeg(arr, 0, arr_size - 1);
    printArray(arr, arr_size);

}

// This code is contributed by vt_m.
```

C#

```
// C# program to Rearrange positive
// and negative numbers in a array
using System;

class GFG {

    /* Function to print an array */
    static void printArray(int[] A, int size)
    {
        for (int i = 0; i < size; i++)
            Console.Write(A[i] + " ");
        Console.WriteLine();
    }

    // Merges two subarrays of arr[].
    // First subarray is arr[l..m]
    // Second subarray is arr[m+1..r]
    static void merge(int[] arr, int l, int m, int r)
    {
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;

        /* create temp arrays */
        int[] L = new int[n1];
        int[] R = new int[n2];

        /* Copy data to temp arrays L[] and R[] */
        for (i = 0; i < n1; i++)
            L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
            R[j] = arr[m + 1 + j];

        /* Merge the temp arrays back into arr[l..r]*/
        // Initial index of first subarray
        i = 0;

        // Initial index of second subarray
        j = 0;

        // Initial index of merged subarray
        k = l;

        // Note the order of appearance of elements should
        // be maintained - we copy elements of left subarray
        // first followed by that of right subarray

        // copy negative elements of left subarray
```

```
while (i < n1 && L[i] < 0)
    arr[k++] = L[i++];

// copy negative elements of right subarray
while (j < n2 && R[j] < 0)
    arr[k++] = R[j++];

// copy positive elements of left subarray
while (i < n1)
    arr[k++] = L[i++];

// copy positive elements of right subarray
while (j < n2)
    arr[k++] = R[j++];
}

// Function to Rearrange positive and negative
// numbers in a array
static void RearrangePosNeg(int[] arr, int l, int r)
{
    if (l < r) {

        // Same as (l + r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - 1) / 2;

        // Sort first and second halves
        RearrangePosNeg(arr, l, m);
        RearrangePosNeg(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

// Driver program
public static void Main()
{
    int[] arr = { -12, 11, -13, -5, 6, -7, 5, -3, -6 };
    int arr_size = arr.Length;
    RearrangePosNeg(arr, 0, arr_size - 1);
    printArray(arr, arr_size);
}
}

// This code is contributed by vt_m.
```

Output:

-12 -13 -5 -7 -3 -6 11 6 5

Time complexity of above solution is $O(n \log n)$. The problem with this approach is we are using auxiliary array for merging but we're not allowed to use any data structure to solve this problem. We can do merging in-place without using any data-structure. The idea is taken from [here](#).

Let L_n and L_p denotes the negative part and positive part of left sub-array respectively. Similarly, R_n and R_p denotes the negative and positive part of right sub-array respectively. Below are the steps to convert $[L_n \ L_p \ R_n \ R_p]$ to $[L_n \ R_n \ L_p \ R_p]$ without using extra space.

1. Reverse L_p and R_n . We get $[L_p] \rightarrow [L_p']$ and $[R_n] \rightarrow [R_n']$
 $[L_n \ L_p \ R_n \ R_p] \rightarrow [L_n \ L_p' \ R_n' \ R_p]$
2. Reverse $[L_p' \ R_n']$. We get $[R_n \ L_p]$.
 $[L_n \ L_p' \ R_n' \ R_p] \rightarrow [L_n \ R_n \ L_p \ R_p]$

Below is C++ implementation of above idea –

```
// C++ program to Rearrange positive and negative
// numbers in a array
#include <bits/stdc++.h>
using namespace std;

/* Function to print an array */
void printArray(int A[], int size)
{
    for (int i = 0; i < size; i++)
        cout << A[i] << " ";
    cout << endl;
}

/* Function to reverse an array. An array can be
reversed in O(n) time and O(1) space. */
void reverse(int arr[], int l, int r)
{
    if (l < r)
    {
        swap(arr[l], arr[r]);
        reverse(arr, ++l, --r);
    }
}

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
```

```
void merge(int arr[], int l, int m, int r)
{
    int i = l; // Initial index of 1st subarray
    int j = m + 1; // Initial index of 2nd

    while (i <= m && arr[i] < 0)
        i++;

    // arr[i..m] is positive

    while (j <= r && arr[j] < 0)
        j++;

    // arr[j..r] is positive

    // reverse positive part of left sub-array (arr[i..m])
    reverse(arr, i, m);

    // reverse negative part of right sub-array (arr[m+1..j-1])
    reverse(arr, m + 1, j - 1);

    // reverse arr[i..j-1]
    reverse(arr, i, j - 1);
}

// Function to Rearrange positive and negative
// numbers in a array
void RearrangePosNeg(int arr[], int l, int r)
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - 1) / 2;

        // Sort first and second halves
        RearrangePosNeg(arr, l, m);
        RearrangePosNeg(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {-12, 11, -13, -5, 6, -7, 5, -3, -6};
    int arr_size = sizeof(arr) / sizeof(arr[0]);
```

```
RearrangePosNeg(arr, 0, arr_size - 1);

printArray(arr, arr_size);

return 0;
}
```

Output:

```
-12 -13 -5 -7 -3 -6 11 6 5
```

Time complexity of above solution is $O(n \log n)$, $O(\log n)$ space for recursive calls, and no additional data structure.

Source

<https://www.geeksforgeeks.org/rearrange-positive-and-negative-numbers/>

Chapter 286

Recursive Bubble Sort

Recursive Bubble Sort - GeeksforGeeks

Background :

[Bubble Sort](#) is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example:

First Pass:

(5 1 4 2 8) -> (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps since $5 > 1$.

(1 5 4 2 8) -> (1 4 5 2 8), Swap since $5 > 4$

(1 4 5 2 8) -> (1 4 2 5 8), Swap since $5 > 2$

(1 4 2 5 8) -> (1 4 2 5 8), Now, since these elements are already in order ($8 > 5$), algorithm does not swap them.

Second Pass:

(1 4 2 5 8) -> (1 4 2 5 8)

(1 4 2 5 8) -> (1 2 4 5 8), Swap since $4 > 2$

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

Following is iterative Bubble sort algorithm :

```
// Iterative Bubble Sort
bubbleSort(arr[], n)
```

```
{  
    for (i = 0; i < n-1; i++)  
  
        // Last i elements are already in place  
        for (j = 0; j < arr[j+1])  
            swap(arr[j], arr[j+1]);  
}
```

See [Bubble Sort](#) for more details.

How to implement it recursively?

Recursive Bubble Sort has no performance/implementation advantages, but can be a good question to check one's understanding of Bubble Sort and recursion.

If we take a closer look at Bubble Sort algorithm, we can notice that in first pass, we move largest element to end (Assuming sorting in increasing order). In second pass, we move second largest element to second last position and so on.

Recursion Idea.

1. Base Case: If array size is 1, return.
2. Do One Pass of normal Bubble Sort. This pass fixes last element of current subarray.
3. Recur for all elements except last of current subarray.

Below is implementation of above idea.

C/C++

```
// C/C++ program for recursive implementation  
// of Bubble sort  
#include <bits/stdc++.h>  
using namespace std;  
  
// A function to implement bubble sort  
void bubbleSort(int arr[], int n)  
{  
    // Base case  
    if (n == 1)  
        return;  
  
    // One pass of bubble sort. After  
    // this pass, the largest element  
    // is moved (or bubbled) to end.  
    for (int i=0; i<n-1; i++)  
        if (arr[i] > arr[i+1])  
            swap(arr[i], arr[i+1]);
```

```
// Largest element is fixed,
// recur for remaining array
bubbleSort(arr, n-1);
}

/* Function to print an array */
void printArray(int arr[], int n)
{
    for (int i=0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array : \n");
    printArray(arr, n);
    return 0;
}
```

Java

```
// Java program for recursive implementation
// of Bubble sort

import java.util.Arrays;

public class GFG
{
    // A function to implement bubble sort
    static void bubbleSort(int arr[], int n)
    {
        // Base case
        if (n == 1)
            return;

        // One pass of bubble sort. After
        // this pass, the largest element
        // is moved (or bubbled) to end.
        for (int i=0; i<n-1; i++)
            if (arr[i] > arr[i+1])
            {
                // swap arr[i], arr[i+1]
```

```
        int temp = arr[i];
        arr[i] = arr[i+1];
        arr[i+1] = temp;
    }

    // Largest element is fixed,
    // recur for remaining array
    bubbleSort(arr, n-1);
}

// Driver Method
public static void main(String[] args)
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};

    bubbleSort(arr, arr.length);

    System.out.println("Sorted array : ");
    System.out.println(Arrays.toString(arr));
}
}
```

Python3

```
# Python Program for implementation of
# Recursive Bubble sort

def bubble_sort(listt):
    for i, num in enumerate(listt):
        try:
            if listt[i+1] < num:
                listt[i] = listt[i+1]
                listt[i+1] = num
                bubble_sort(listt)
        except IndexError:
            pass
    return listt

listt = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(listt)

print("Sorted array:");
for i in range(0, len(listt)):
    print(listt[i], end=' ')

# Code contributed by Mohit Gupta_OMG
```

C#

```
// C# program for recursive
// implementation of Bubble sort
using System;

class GFG
{

    // A function to implement
    // bubble sort
    static void bubbleSort(int []arr,
                           int n)
    {
        // Base case
        if (n == 1)
            return;

        // One pass of bubble
        // sort. After this pass,
        // the largest element
        // is moved (or bubbled)
        // to end.
        for (int i = 0; i < n - 1; i++)
            if (arr[i] > arr[i + 1])
            {
                // swap arr[i], arr[i+1]
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
            }

        // Largest element is fixed,
        // recur for remaining array
        bubbleSort(arr, n - 1);
    }

    // Driver code
    static void Main()
    {
        int []arr = {64, 34, 25,
                    12, 22, 11, 90};

        bubbleSort(arr, arr.Length);

        Console.WriteLine("Sorted array : ");
        for(int i = 0; i < arr.Length; i++)
            Console.Write(arr[i] + " ");
    }
}
```

```
}
```

```
}
```

```
// This code is contributed
```

```
// by Sam007
```

Output :

Sorted array :

```
11 12 22 25 34 64 90
```

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/recursive-bubble-sort/>

Chapter 287

Recursive Insertion Sort

Recursive Insertion Sort - GeeksforGeeks

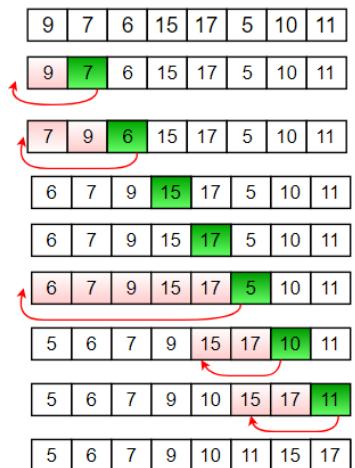
Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Below is an iterative algorithm for insertion sort

Algorithm

```
// Sort an arr[] of size n
insertionSort(arr, n)
    Loop from i = 1 to n-1.
        a) Pick element arr[i] and insert
            it into sorted sequence arr[0..i-1]
```

Example:



Refer [Insertion Sort](#) for more details.

How to implement it recursively?

Recursive Insertion Sort has no performance/implementation advantages, but can be a good question to check one's understanding of Insertion Sort and recursion.

If we take a closer look at Insertion Sort algorithm, we keep processed elements sorted and insert new elements one by one in the inserted array.

Recursion Idea.

1. Base Case: If array size is 1 or smaller, return.
2. Recursively sort first n-1 elements.
3. Insert last element at its correct position in sorted array.

Below is implementation of above idea.

C/C++

```
// Recursive C++ program for insertion sort
#include <iostream>
using namespace std;

// Recursive function to sort an array using
// insertion sort
void insertionSortRecursive(int arr[], int n)
{
    // Base case
    if (n <= 1)
        return;

    // Sort first n-1 elements
    insertionSortRecursive( arr, n-1 );

    // Insert last element at its correct position
    // in sorted array.
    int last = arr[n-1];
    int j = n-2;

    /* Move elements of arr[0..i-1], that are
       greater than key, to one position ahead
       of their current position */
    while (j >= 0 && arr[j] > last)
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = last;
```

```
}

// A utility function to print an array of size n
void printArray(int arr[], int n)
{
    for (int i=0; i < n; i++)
        cout << arr[i] << " ";
}

/* Driver program to test insertion sort */
int main()
{
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);

    insertionSortRecursive(arr, n);
    printArray(arr, n);

    return 0;
}
```

Java

```
// Recursive Java program for insertion sort

import java.util.Arrays;

public class GFG
{
    // Recursive function to sort an array using
    // insertion sort
    static void insertionSortRecursive(int arr[], int n)
    {
        // Base case
        if (n <= 1)
            return;

        // Sort first n-1 elements
        insertionSortRecursive( arr, n-1 );

        // Insert last element at its correct position
        // in sorted array.
        int last = arr[n-1];
        int j = n-2;

        /* Move elements of arr[0..i-1], that are
           greater than key, to one position ahead
           of their current position */
    }
}
```

```
        while (j >= 0 && arr[j] > last)
        {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = last;
    }

    // Driver Method
    public static void main(String[] args)
    {
        int arr[] = {12, 11, 13, 5, 6};

        insertionSortRecursive(arr, arr.length);

        System.out.println(Arrays.toString(arr));
    }
}
```

Python

```
# Recursive Python program for insertion sort
# Recursive function to sort an array using insertion sort

def insertionSortRecursive(arr,n):
    # base case
    if n<=1:
        return

    # Sort first n-1 elements
    insertionSortRecursive(arr,n-1)
    '''Insert last element at its correct position
       in sorted array.'''
    last = arr[n-1]
    j = n-2

    # Move elements of arr[0..i-1], that are
    # greater than key, to one position ahead
    # of their current position
    while (j>=0 and arr[j]>last):
        arr[j+1] = arr[j]
        j = j-1

    arr[j+1]=last

# A utility function to print an array of size n
def printArray(arr,n):
    for i in range(n):
```

```
print arr[i],  
  
# Driver program to test insertion sort  
arr = [12,11,13,5,6]  
n = len(arr)  
insertionSortRecursive(arr, n)  
printArray(arr, n)  
  
# Contributed by Harsh Valecha  
  
C#  
  
// Recursive C# program  
// for insertion sort  
using System;  
  
class GFG  
{  
  
    // Recursive function to sort  
    // an array using insertion sort  
    static void insertionSortRecursive(int []arr,  
                                         int n)  
    {  
        // Base case  
        if (n <= 1)  
            return;  
  
        // Sort first n-1 elements  
        insertionSortRecursive(arr, n - 1);  
  
        // Insert last element at  
        // its correct position  
        // in sorted array.  
        int last = arr[n - 1];  
        int j = n - 2;  
  
        /* Move elements of arr[0..i-1],  
        that are greater than key, to  
        one position ahead of their  
        current position */  
        while (j >= 0 && arr[j] > last)  
        {  
            arr[j + 1] = arr[j];  
            j--;  
        }  
        arr[j + 1] = last;  
    }  
}
```

```
//Driver Code
static void Main()
{
    int []arr = {12, 11, 13, 5, 6};

    insertionSortRecursive(arr, arr.Length);

    for(int i = 0; i < arr.Length; i++)
        Console.Write(arr[i] + " ");
}
}

// This code is contributed by Sam007
```

PHP

```
<?php
// Recursive PHP program for insertion sort

// Recursive function to sort an
// array using insertion sort
function insertionSortRecursive(&$arr, $n)
{

    // Base case
    if ($n <= 1)
        return;

    // Sort first n-1 elements
    insertionSortRecursive($arr, $n - 1);

    // Insert last element at its correct
    // position in sorted array.
    $last = $arr[$n - 1];
    $j = $n - 2;

    // Move elements of arr[0..i-1], that are
    // greater than key, to one position ahead
    // of their current position
    while ($j >= 0 && $arr[$j] > $last)
    {
        $arr[$j + 1] = $arr[$j];
        $j--;
    }
    $arr[$j + 1] = $last;
}
```

```
// A utility function to
// print an array of size n
function printArray(&$arr, $n)
{
    for ($i = 0; $i < $n; $i++)
        echo $arr[$i]. " ";
}

// Driver Code
$arr = array(12, 11, 13, 5, 6);
$n = sizeof($arr);

insertionSortRecursive($arr, $n);
printArray($arr, $n);

// This code is contributed by ChitraNayal.
?>
```

Output :

5 6 11 12 13

Improved By : [Sam007](#), [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/recursive-insertion-sort/>

Chapter 288

Recursive Selection Sort

Recursive Selection Sort - GeeksforGeeks

The [Selection Sort](#) algorithm sorts maintains two parts.

1. First part that is already sorted
2. Second part that is yet to be sorted.

The algorithm works by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the end of sorted part.

```
arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]
// and place it at beginning
11 25 12 22 64

// Find the minimum element in arr[1...4]
// and place it at beginning of arr[1...4]
11 12 25 22 64

// Find the minimum element in arr[2...4]
// and place it at beginning of arr[2...4]
11 12 22 25 64

// Find the minimum element in arr[3...4]
// and place it at beginning of arr[3...4]
11 12 22 25 64
```

We have already discussed about [Iterative Selection Sort](#). In this article recursive approach is discussed. The idea of recursive solution is to one by one increment sorted part and recursively call for remaining (yet to be sorted) part.

C++

```
// Recursive C++ program to sort an array
// using selection sort
#include <iostream>
using namespace std;

// Return minimum index
int minIndex(int a[], int i, int j)
{
    if (i == j)
        return i;

    // Find minimum of remaining elements
    int k = minIndex(a, i + 1, j);

    // Return minimum of current and remaining.
    return (a[i] < a[k])? i : k;
}

// Recursive selection sort. n is size of a[] and index
// is index of starting element.
void recurSelectionSort(int a[], int n, int index = 0)
{
    // Return when starting and size are same
    if (index == n)
        return;

    // calling minimum index function for minimum index
    int k = minIndex(a, index, n-1);

    // Swapping when index nd minimum index are not same
    if (k != index)
        swap(a[k], a[index]);

    // Recursively calling selection sort function
    recurSelectionSort(a, n, index + 1);
}

// Driver code
int main()
{
    int arr[] = {3, 1, 5, 2, 7, 0};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Calling function
    recurSelectionSort(arr, n);
```

```
//printing sorted array
for (int i = 0; i<n ; i++)
    cout << arr[i] << " ";
cout << endl;
return 0;
}
```

Java

```
// Recursive Java program to sort an array
// using selection sort

class Test
{
    // Return minimum index
    static int minIndex(int a[], int i, int j)
    {
        if (i == j)
            return i;

        // Find minimum of remaining elements
        int k = minIndex(a, i + 1, j);

        // Return minimum of current and remaining.
        return (a[i] < a[k])? i : k;
    }

    // Recursive selection sort. n is size of a[] and index
    // is index of starting element.
    static void recurSelectionSort(int a[], int n, int index)
    {

        // Return when starting and size are same
        if (index == n)
            return;

        // calling minimum index function for minimum index
        int k = minIndex(a, index, n-1);

        // Swapping when index nd minimum index are not same
        if (k != index){
            // swap
            int temp = a[k];
            a[k] = a[index];
            a[index] = temp;
        }
        // Recursively calling selection sort function
        recurSelectionSort(a, n, index + 1);
    }
}
```

```
}

// Driver method
public static void main(String args[])
{
    int arr[] = {3, 1, 5, 2, 7, 0};

    // Calling function
    recurSelectionSort(arr, arr.length, 0);

    //printing sorted array
    for (int i = 0; i< arr.length; i++)
        System.out.print(arr[i] + " ");
}
}
```

Python3

```
# Recursive Python3 code to sort
# an array using selection sort

# Return minimum index
def minIndex( a , i , j ):
    if i == j:
        return i

    # Find minimum of remaining elements
    k = minIndex(a, i + 1, j)

    # Return minimum of current
    # and remaining.
    return (i if a[i] < a[k] else k)

# Recursive selection sort. n is
# size of a[] and index is index of
# starting element.
def recurSelectionSort(a, n, index = 0):

    # Return when starting and
    # size are same
    if index == n:
        return -1

    # calling minimum index function
    # for minimum index
    k = minIndex(a, index, n-1)
```

```
# Swapping when index and minimum
# index are not same
if k != index:
    a[k], a[index] = a[index], a[k]

# Recursively calling selection
# sort function
recurSelectionSort(a, n, index + 1)

# Driver code
arr = [3, 1, 5, 2, 7, 0]
n = len(arr)

# Calling function
recurSelectionSort(arr, n)

# printing sorted array
for i in arr:
    print(i, end = ' ')

# This code is contributed by "Sharad_Bhardwaj".
```

Output:

0 1 2 3 5 7

Source

<https://www.geeksforgeeks.org/recursive-selection-sort/>

Chapter 289

Recursive selection sort for singly linked list Swapping node links

Recursive selection sort for singly linked list Swapping node links - GeeksforGeeks

Given a singly linked list containing n nodes. The problem is to sort the list using recursive selection sort technique. The approach should be such that it involves swapping node links instead of swapping nodes data.

Examples:

```
Input : 10 -> 12 -> 8 -> 4 -> 6
Output : 4 -> 6 -> 8 -> 10 -> 12
```

In Selection Sort, we first find minimum element, swap it with the beginning node and recur for remaining list. Below is recursive implementation of these steps for linked list.

```
recurSelectionSort(head)
    if head->next == NULL
        return head
    Initialize min = head
    Initialize beforeMin = NULL
    Initialize ptr = head

    while ptr->next != NULL
        if min->data > ptr->next->data
            min = ptr->next
            beforeMin = ptr
```

```
ptr = ptr->next

if min != head
    swapNodes(&head, head, min, beforeMin)

head->next = recurSelectionSort(head->next)
return head

swapNodes(head_ref, currX, currY, prevY)
    head_ref = currY
    prevY->next = currX

    Initialize temp = currY->next
    currY->next = currX->next
    currX->next = temp
```

The `swapNodes(head_ref, currX, currY, prevY)` is based on the approach discussed [here](#) but it is modified accordingly for the implementation of this post.

```
// C++ implementation of recursive selection sort
// for singly linked list | Swapping node links
#include <bits/stdc++.h>
using namespace std;

// A Linked list node
struct Node {
    int data;
    struct Node* next;
};

// function to swap nodes 'currX' and 'currY' in a
// linked list without swapping data
void swapNodes(struct Node** head_ref, struct Node* currX,
               struct Node* currY, struct Node* prevY)
{
    // make 'currY' as new head
    *head_ref = currY;

    // adjust links
    prevY->next = currX;

    // Swap next pointers
    struct Node* temp = currY->next;
    currY->next = currX->next;
    currX->next = temp;
}

// function to sort the linked list using
```

```
// recursive selection sort technique
struct Node* recurSelectionSort(struct Node* head)
{
    // if there is only a single node
    if (head->next == NULL)
        return head;

    // 'min' - pointer to store the node having
    // minimum data value
    struct Node* min = head;

    // 'beforeMin' - pointer to store node previous
    // to 'min' node
    struct Node* beforeMin = NULL;
    struct Node* ptr;

    // traverse the list till the last node
    for (ptr = head; ptr->next != NULL; ptr = ptr->next) {

        // if true, then update 'min' and 'beforeMin'
        if (ptr->next->data < min->data) {
            min = ptr->next;
            beforeMin = ptr;
        }
    }

    // if 'min' and 'head' are not same,
    // swap the head node with the 'min' node
    if (min != head)
        swapNodes(&head, head, min, beforeMin);

    // recursively sort the remaining list
    head->next = recurSelectionSort(head->next);

    return head;
}

// function to sort the given linked list
void sort(struct Node** head_ref)
{
    // if list is empty
    if ((*head_ref) == NULL)
        return;

    // sort the list using recursive selection
    // sort technique
    *head_ref = recurSelectionSort(*head_ref);
}
```

```
// function to insert a node at the
// beginning of the linked list
void push(struct Node** head_ref, int new_data)
{
    // allocate node
    struct Node* new_node =
        (struct Node*)malloc(sizeof(struct Node));

    // put in the data
    new_node->data = new_data;

    // link the old list to the new node
    new_node->next = (*head_ref);

    // move the head to point to the new node
    (*head_ref) = new_node;
}

// function to print the linked list
void printList(struct Node* head)
{
    while (head != NULL) {
        cout << head->data << " ";
        head = head->next;
    }
}

// Driver program to test above
int main()
{
    struct Node* head = NULL;

    // create linked list 10->12->8->4->6
    push(&head, 6);
    push(&head, 4);
    push(&head, 8);
    push(&head, 12);
    push(&head, 10);

    cout << "Linked list before sorting:n";
    printList(head);

    // sort the linked list
    sort(&head);

    cout << "\nLinked list after sorting:n";
    printList(head);
}
```

```
    return 0;  
}
```

Output:

```
Linked list before sorting:  
10 12 8 4 6  
Linked list after sorting:  
4 6 8 10 12
```

Time Complexity: $O(n^2)$

Source

<https://www.geeksforgeeks.org/recursive-selection-sort-singly-linked-list-swapping-node-links/>

Chapter 290

Replacing an element makes array elements consecutive

Replacing an element makes array elements consecutive - GeeksforGeeks

Given an array of positive distinct integers. We need to find the only element whose replacement with any other value makes array elements distinct consecutive. If it is not possible to make array elements consecutive, return -1.

Examples :

Input : arr[] = {45, 42, 46, 48, 47}

Output : 42

Explanation: We can replace 42 with either 44 or 48 to make array consecutive.

Input : arr[] = {5, 6, 7, 9, 10}

Output : 5 [OR 10]

Explanation: We can either replace 5 with 8 or 10 with 8 to make array elements consecutive.

Input : arr[] = {5, 6, 7, 9, 8}

Output : Array elements are already consecutive

A **Naive Approach** is to check each element of arr[], after replacing of which makes consecutive or not. Time complexity for this approach $O(n^2)$

A **Better Approach** is based on an important observation that either the smallest or the largest element would be answer if answer exists. If answer exists, then there are two cases.

- 1) Series of consecutive elements starts with minimum element of array then continues by adding 1 to previous.

- 2) Series of consecutive elements start with maximum element of array, then continues by subtracting 1 from previous.

We make above two series and for every series, we search series elements in array. If for both series, number of mismatches are more than 1, then answer does not exist. If any series is found with one mismatch, then we have answer.

C++

```
// CPP program to find an element replacement
// of which makes the array elements consecutive.
#include <bits/stdc++.h>
using namespace std;

int findElement(int arr[], int n)
{
    sort(arr, arr+n);

    // Making a series starting from first element
    // and adding 1 to every element.
    int mismatch_count1 = 0, res;
    int next_element = arr[n-1] - n + 1;
    for (int i=0; i<n-1; i++) {
        if (binary_search(arr, arr+n, next_element) == 0)
        {
            res = arr[0];
            mismatch_count1++;
        }
        next_element++;
    }

    // If only one mismatch is found.
    if (mismatch_count1 == 1)
        return res;

    // If no mismatch found, elements are
    // already consecutive.
    if (mismatch_count1 == 0)
        return 0;

    // Making a series starting from last element
    // and subtracting 1 to every element.
    int mismatch_count2 = 0;
    next_element = arr[0] + n - 1;
    for (int i=n-1; i>=1; i--) {
        if (binary_search(arr, arr+n, next_element) == 0)
        {
            res = arr[n-1];
            mismatch_count2++;
        }
        next_element--;
    }
}
```

```
        }
        next_element--;
    }

    // If only one mismatch is found.
    if (mismatch_count2 == 1)
        return res;

    return -1;
}

// Driver code
int main()
{
    int arr[] = {7, 5, 12, 8} ;
    int n = sizeof(arr)/sizeof(arr[0]);
    int res = findElement(arr,n);
    if (res == -1)
        cout << "Answer does not exist";
    else if (res == 0)
        cout << "Elements are already consecutive";
    else
        cout << res;
    return 0;
}
```

Java

```
// Java program to find an element
// replacement of which makes
// the array elements consecutive.
import java.io.*;
import java.util.Arrays;

class GFG
{
    static int findElement(int []arr,
                          int n)
    {
        Arrays.sort(arr);

        // Making a series starting
        // from first element and
        // adding 1 to every element.
        int mismatch_count1 = 0,
            res = 0;
        int next_element = arr[n - 1] -
                          n + 1;
```

```
for (int i = 0; i < n - 1; i++)
{
    if (Arrays.binarySearch(arr,
                           next_element) < 0)
    {
        res = arr[0];
        mismatch_count1++;
    }
    next_element++;
}

// If only one mismatch is found.
if (mismatch_count1 == 1)
    return res;

// If no mismatch found, elements
// are already consecutive.
if (mismatch_count1 == 0)
    return 0;

// Making a series starting
// from last element and
// subtracting 1 to every element.
int mismatch_count2 = 0;
next_element = arr[0] + n - 1;

for (int i = n - 1; i >= 1; i--)
{
    if (Arrays.binarySearch(arr,
                           next_element) < 0)
    {
        res = arr[n - 1];
        mismatch_count2++;
    }
    next_element--;
}

// If only one mismatch is found.
if (mismatch_count2 == 1)
    return res;

return -1;
}

// Driver code
public static void main(String args[])
{
```

```
int []arr = new int[]{7, 5, 12, 8} ;
int n = arr.length;
int res = findElement(arr,n);
if (res == -1)
    System.out.print("Answer does not exist");
else if (res == 0)
    System.out.print("Elements are " +
                      "already consecutive");
else
    System.out.print(res);
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
// C# program to find an element
// replacement of which makes
// the array elements consecutive.
using System;
using System.Linq;
using System.Collections.Generic;

class GFG
{
    static int findElement(int []arr,
                          int n)
    {
        Array.Sort(arr);

        // Making a series starting
        // from first element and
        // adding 1 to every element.
        int mismatch_count1 = 0, res = 0;
        int next_element = arr[n - 1] - n + 1;

        for (int i = 0; i < n - 1; i++)
        {
            if (Array.BinarySearch(arr,
                                  next_element) < 0)
            {
                res = arr[0];
                mismatch_count1++;
            }
            next_element++;
        }
    }
}
```

```
// If only one mismatch is found.
if (mismatch_count1 == 1)
    return res;

// If no mismatch found, elements
// are already consecutive.
if (mismatch_count1 == 0)
    return 0;

// Making a series starting
// from last element and
// subtracting 1 to every element.
int mismatch_count2 = 0;
next_element = arr[0] + n - 1;

for (int i = n - 1; i >= 1; i--)
{
    if (Array.BinarySearch(arr,
                           next_element) < 0)
    {
        res = arr[n - 1];
        mismatch_count2++;
    }
    next_element--;
}

// If only one mismatch is found.
if (mismatch_count2 == 1)
    return res;

return -1;
}

// Driver code
static void Main()
{
    int []arr = new int[]{7, 5, 12, 8} ;
    int n = arr.Length;
    int res = findElement(arr,n);
    if (res == -1)
        Console.WriteLine("Answer does not exist");
    else if (res == 0)
        Console.WriteLine("Elements are " +
                         "already consecutive");
    else
        Console.WriteLine(res);
}
```

}

```
// This code is contributed by  
// Manish Shaw(manishshaw1)
```

Output :

12

Time Complexity : $O(n \log n)$

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/replacing-an-element-makes-array-elements-consecutive/>

Chapter 291

Ropes left after every removal of smallest

Ropes left after every removal of smallest - GeeksforGeeks

Given an array of an integer of size N. Array contains N ropes of length Ropes[i]. You have to perform a cut operation on ropes such that all of them are reduced by the length of the smallest rope. Display the number of ropes left after every cut. Perform operations till length of each rope becomes zero.

Note: IF no ropes left after a single operation, in this case, we print 0.

Examples:

Input : Ropes[] = { 5, 1, 1, 2, 3, 5 }

Output : 4 3 2

Explanation : In first operation the minimum ropes is 1 so we reduce length 1 from all of them after reducing we left with 4 ropes and we do same for rest.

Input : Ropes[] = { 5, 1, 6, 9, 8, 11, 2, 2, 6, 5 }

Output : 9 7 5 3 2 1

Simple solution is to we traverse a loop from [0...n-1], In each iterations first we find min length rope. After that we reduce all ropes length by it and then count how many ropes are left whose length is greater the zero. this process is done until all ropes length is greater than zero. This solution work in $O(n^2)$ time.

Efficient solution works in $O(n \log(n))$. First we have to sort all Ropes in increasing order of there lenght. after that we have follow the step.

```
//initial cutting lenght "min rope"
CuttingLength = Ropes[0]
Now Traverse a loop from left to right [1...n]
.During traverse we check that
```

```
is current ropes length is greater than zero or not
IF ( Ropes[i] - CuttingLength > 0 )
.... IF Yes then all ropes to it's right side also greater than 0
.... Print number of ropes remains (n - i)
.... update Cutting Length by current rope length
..... CuttingLength = Ropes[i]
Do the same process for the rest.
```

Below is the implementation of above idea.

C++

```
// C++ program to print how many
// Ropes are Left After Every Cut
#include <bits/stdc++.h>
using namespace std;

// Function print how many Ropes are
// Left After Every Cutting operation
void cuttringRopes(int Ropes[], int n)
{
    // sort all Ropes in increase
    // of there length
    sort(Ropes, Ropes + n);

    int singleOperation = 0;

    // min length rope
    int cuttingLenght = Ropes[0];

    // now traverse through the given
    // Ropes in increase order of length
    for (int i = 1; i < n; i++)
    {
        // After cutting if current rope length
        // is greater than '0' that mean all
        // ropes to it's right side are also
        // greater than 0
        if (Ropes[i] - cuttingLenght > 0)
        {
            // print number of ropes remains
            cout << (n - i) << " ";

            // now current rope become
            // min length rope
            cuttingLenght = Ropes[i];
            singleOperation++;
        }
    }
}
```

```
    if (singleOperation == 0)
        cout << "0 ";
}
int main()
{
    int Ropes[] = { 5, 1, 1, 2, 3, 5 };
    int n = sizeof(Ropes) / sizeof(Ropes[0]);
    cuttringRopes(Ropes, n);
    return 0;
}
```

Java

```
// Java program to print how many
// Ropes are Left After Every Cut
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG {

    // function print how many Ropes are Left After
    // Every Cutting operation
    public static void cuttringRopes(int Ropes[], int n)
    {
        // sort all Ropes in increasing
        // order of their length
        Arrays.sort(Ropes);

        int singleOperation = 0;

        // min length rope
        int cuttingLenght = Ropes[0];

        // now traverse through the given Ropes in
        // increase order of length
        for (int i = 1; i < n; i++)
        {
            // After cutting if current rope length
            // is greater than '0' that mean all
            // ropes to it's right side are also
            // greater than 0
            if (Ropes[i] - cuttingLenght > 0)
            {
                System.out.print(n - i + " ");
            }
            // now current rope become
            // min length rope
        }
    }
}
```

```
        cuttingLenght = Ropes[i];

        singleOperation++;
    }
}

// after first operation all ropes
// length become zero
if (singleOperation == 0)
    System.out.print("0");
}

public static void main(String[] arg)
{
    int[] Ropes = { 5, 1, 1, 2, 3, 5 };
    int n = Ropes.length;
    cuttringRopes(Ropes, n);
}
}
```

Python3

```
# Python 3 program to
# print how many
# Ropes are Left After
# Every Cut

# Function print how many Ropes are
# Left After Every Cutting operation
def cuttringRopes(Ropes, n) :

    # sort all Ropes in increase
    # of there length
    Ropes.sort()

    singleOperation = 0

    # min length rope
    cuttingLenght = Ropes[0]

    # now traverse through the given
    # Ropes in increase order of length
    for i in range(1,n) :

        # After cutting if current rope length
        # is greater than '0' that mean all
        # ropes to it's right side are also
        # greater than 0
```

```
if (Ropes[i] - cuttingLength > 0) :  
  
    # print number of ropes remains  
    print((n - i) ,end= " ")  
  
    # now current rope become  
    # min length rope  
    cuttingLength = Ropes[i]  
    singleOperation = singleOperation + 1  
  
if (singleOperation == 0) :  
    print("0 ",end="")  
  
Ropes = [ 5, 1, 1, 2, 3, 5 ]  
n = len(Ropes)  
cuttringRopes(Ropes, n)  
  
# This code is contributed by Nikita Tiwari.
```

C#

```
// C# program to print how many  
// Ropes are Left After Every Cut  
using System;  
  
class GFG {  
  
    // function print how many Ropes are Left After  
    // Every Cutting operation  
    public static void cuttringRopes(int []Ropes, int n)  
    {  
        // sort all Ropes in increasing  
        // order of their length  
        Array.Sort(Ropes);  
  
        int singleOperation = 0;  
  
        // min length rope  
        int cuttingLength = Ropes[0];  
  
        // now traverse through the given Ropes in  
        // increase order of length  
        for (int i = 1; i < n; i++)  
        {
```

```
// After cutting if current rope length
// is greater than '0' that mean all
// ropes to it's right side are also
// greater than 0
if (Ropes[i] - cuttingLength > 0)
{
    Console.Write(n - i + " ");

    // now current rope become
    // min length rope
    cuttingLength = Ropes[i];

    singleOperation++;
}
}

// after first operation all ropes
// length become zero
if (singleOperation == 0)
    Console.WriteLine("0");
}

// Driver code
public static void Main()
{
    int[] Ropes = { 5, 1, 1, 2, 3, 5 };
    int n = Ropes.Length;
    cuttringRopes(Ropes, n);
}
}

// This code is contributed by vt_m.
```

```
php
<?php
// PHP program to print how many
// Ropes are Left After Every Cut

// Function print how many Ropes are
// Left After Every Cutting operation
function cuttringRopes($Ropes, $n)
{

    // sort all Ropes in increase
    // of there length
    sort($Ropes);
```

```
$singleOperation = 0;

// min length rope
$cuttingLength = $Ropes[0];

// now traverse through the given
// Ropes in increase order of length
for ($i = 1; $i < $n; $i++)
{
    // After cutting if current rope length
    // is greater than '0' that mean all
    // ropes to it's right side are also
    // greater than 0
    if ($Ropes[$i] - $cuttingLength > 0)
    {
        // print number of ropes remains
        echo ($n - $i). " ";

        // now current rope become
        // min length rope
        $cuttingLength = $Ropes[$i];
        $singleOperation++;
    }
}
if ($singleOperation == 0)
    echo "0 ";
}

// Driver Code
$Ropes = array(5, 1, 1, 2, 3, 5);
$n = count($Ropes);
cuttingRopes($Ropes, $n);

// This code is contributed by Sam007
?>
```

4 3 2

Time Complexity : O(n long (n))
Space complexity : O(1)

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/ropes-left-every-cut/>

Chapter 292

Row wise sorting in 2D array

Row wise sorting in 2D array - GeeksforGeeks

Given a 2D array, sort each row of this array and print the result.

Examples:

Input :
77 11 22 3
11 89 1 12
32 11 56 7
11 22 44 33
Output :
3 11 22 77
1 11 12 89
7 11 32 56
11 22 33 44

Input :
8 6 4 5
3 5 2 1
9 7 4 2
7 8 9 5
Output :
4 5 6 8
1 2 3 5
2 4 7 9
5 7 8 9

Method 1 (Using Bubble Sort)

Start iterating through each row of given 2D array, and sort elements of each row using efficient sorting algorithm.

Java

```
// Java code to sort 2D matrix row-wise
import java.io.*;

public class Sort2DMatrix {

    static int sortRowWise(int m[][])
    {
        // loop for rows of matrix
        for (int i = 0; i < m.length; i++) {

            // loop for column of matrix
            for (int j = 0; j < m[i].length; j++) {

                // loop for comparison and swapping
                for (int k = 0; k < m[i].length - j; k++) {
                    if (m[i][k] > m[i][k + 1]) {

                        // swapping of elements
                        int t = m[i][k];
                        m[i][k] = m[i][k + 1];
                        m[i][k + 1] = t;
                    }
                }
            }
        }

        // printing the sorted matrix
        for (int i = 0; i < m.length; i++) {
            for (int j = 0; j < m[i].length; j++)
                System.out.print(m[i][j] + " ");
            System.out.println();
        }
    }

    return 0;
}

// driver code
public static void main(String args[])
{
    int m[][] = { { 9, 8, 7, 1 },
                  { 7, 3, 0, 2 },
                  { 9, 5, 3, 2 },
                  { 6, 3, 1, 2 } };
    sortRowWise(m);
}
}
```

Output

```
1 7 8 9  
0 2 3 7  
2 3 5 9  
1 2 3 6
```

Method 2 (Using Library Function)

The idea is to use [Arrays.sort\(\)](#) for every row of matrix.

Java

```
// Java code to sort 2D matrix row-wise  
import java.io.*;  
import java.util.Arrays;  
  
public class Sort2DMatrix {  
  
    static int sortRowWise(int m[][])  
    {  
        // One by one sort individual rows.  
        for (int i = 0; i < m.length; i++)  
            Arrays.sort(m[i]);  
  
        // printing the sorted matrix  
        for (int i = 0; i < m.length; i++) {  
            for (int j = 0; j < m[i].length; j++)  
                System.out.print(m[i][j] + " ");  
            System.out.println();  
        }  
  
        return 0;  
    }  
  
    // driver code  
    public static void main(String args[])  
    {  
        int m[][] = { { 9, 8, 7, 1 },  
                     { 7, 3, 0, 2 },  
                     { 9, 5, 3, 2 },  
                     { 6, 3, 1, 2 } };  
  
        sortRowWise(m);  
    }  
}
```

Output

```
1 7 8 9  
0 2 3 7  
2 3 5 9  
1 2 3 6
```

Source

<https://www.geeksforgeeks.org/row-wise-sorting-2d-array/>

Chapter 293

Segregate 0s and 1s in an array

Segregate 0s and 1s in an array - GeeksforGeeks

You are given an array of 0s and 1s in random order. Segregate 0s on left side and 1s on right side of the array. Traverse array only once.

```
Input array = [0, 1, 0, 1, 0, 0, 1, 1, 1, 0]
Output array = [0, 0, 0, 0, 0, 1, 1, 1, 1]
```

Method 1 (Count 0s or 1s)

Thanks to Naveen for suggesting this method.

- 1) Count the number of 0s. Let count be C.
- 2) Once we have count, we can put C 0s at the beginning and 1s at the remaining $n - C$ positions in array.

Time Complexity : $O(n)$

C++

```
// C++ code to Segregate 0s and 1s in an array
#include <bits/stdc++.h>
using namespace std;

// Function to segregate 0s and 1s
void segregate0and1(int arr[], int n)
{
    int count = 0; // Counts the no of zeros in arr

    for (int i = 0; i < n; i++) {
        if (arr[i] == 0)
            count++;
    }
}
```

```
// Loop fills the arr with 0 until count
for (int i = 0; i < count; i++)
    arr[i] = 0;

// Loop fills remaining arr space with 1
for (int i = count; i < n; i++)
    arr[i] = 1;
}

// Function to print segregated array
void print(int arr[], int n)
{
    cout << "Array after segregation is ";

    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver function
int main()
{
    int arr[] = { 0, 1, 0, 1, 1, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);

    segregate0and1(arr, n);
    print(arr, n);

    return 0;
}

// This code is contributed by Sahil_Bansall
```

Java

```
// Java code to Segregate 0s and 1s in an array
class GFG {

    // function to segregate 0s and 1s
    static void segregate0and1(int arr[], int n)
    {
        int count = 0; // counts the no of zeros in arr

        for (int i = 0; i < n; i++) {
            if (arr[i] == 0)
                count++;
        }

        // loop fills the arr with 0 until count
```

```
for (int i = 0; i < count; i++)
    arr[i] = 0;

// loop fills remaining arr space with 1
for (int i = count; i < n; i++)
    arr[i] = 1;
}

// function to print segregated array
static void print(int arr[], int n)
{
    System.out.print("Array after segregation is ");
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}

// driver function
public static void main(String[] args)
{
    int arr[] = new int[]{ 0, 1, 0, 1, 1, 1 };
    int n = arr.length;

    segregate0and1(arr, n);
    print(arr, n);

}
}

// This code is contributed by Kamal Rawal
```

Python3

```
# Python 3 code to Segregate
# 0s and 1s in an array

# Function to segregate 0s and 1s
def segregate0and1(arr, n) :

    # Counts the no of zeros in arr
    count = 0

    for i in range(0, n) :
        if (arr[i] == 0) :
            count = count + 1

    # Loop fills the arr with 0 until count
    for i in range(0, count) :
        arr[i] = 0
```

```
# Loop fills remaining arr space with 1
for i in range(count, n) :
    arr[i] = 1

# Function to print segregated array
def print_arr(arr , n) :
    print( "Array after segregation is ",end = "")

    for i in range(0, n) :
        print(arr[i] , end = " ")

# Driver function
arr = [ 0, 1, 0, 1, 1, 1 ]
n = len(arr)

segregate0and1(arr, n)
print_arr(arr, n)

# This code is contributed by Nikita Tiwari.
```

C#

```
// C# code to Segregate 0s and 1s in an array
using System;

class GFG {

    // function to segregate 0s and 1s
    static void segregate0and1(int []arr, int n)
    {
        // counts the no of zeros in arr
        int count = 0;

        for (int i = 0; i < n; i++) {
            if (arr[i] == 0)
                count++;
        }

        // loop fills the arr with 0 until count
        for (int i = 0; i < count; i++)
            arr[i] = 0;

        // loop fills remaining arr space with 1
```

```
        for (int i = count; i < n; i++)
            arr[i] = 1;
    }

    // function to print segregated array
    static void print(int []arr, int n)
    {
        Console.WriteLine("Array after segregation is ");
        for (int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");
    }

    // driver function
    public static void Main()
    {
        int []arr = new int[]{ 0, 1, 0, 1, 1, 1 };
        int n = arr.Length;

        segregate0and1(arr, n);
        print(arr, n);

    }
}

//This code is contributed by vt_m.
```

PHP

```
<?php
// PHP code to Segregate
// 0s and 1s in an array

// Function to segregate
// 0s and 1s
function segregate0and1(&$arr, $n)
{
    $count = 0; // Counts the no
                // of zeros in arr

    for ($i = 0; $i < $n; $i++)
    {
        if ($arr[$i] == 0)
            $count++;
    }

    // Loop fills the arr
    // with 0 until count
    for ($i = 0; $i < $count; $i++)
```

```
$arr[$i] = 0;

// Loop fills remaining
// arr space with 1
for ($i = $count; $i < $n; $i++)
    $arr[$i] = 1;
}

// Function to print
// segregated array
function toprint(&$arr , $n)
{
    echo ("Array after segregation is ");

    for ($i = 0; $i < $n; $i++)
        echo ( $arr[$i] . " ");
}

// Driver Code
$arr = array(0, 1, 0, 1, 1, 1 );
$n = sizeof($arr);

segregate0and1($arr, $n);
toprint($arr, $n);

// This code is contributed
// by Shivi_Aggarwal
?>
```

Output :

```
Array after segregation is 0 0 1 1 1 1
```

The method 1 traverses the array two times. Method 2 does the same in a single pass.

Method 2 (Use two indexes to traverse)

Maintain two indexes. Initialize first index *left* as 0 and second index *right* as n-1.

Do following while *left* < *right*

- a) Keep incrementing index *left* while there are 0s at it
- b) Keep decrementing index *right* while there are 1s at it
- c) If *left* < *right* then exchange arr[left] and arr[right]

Implementation:

C/C++

```
// C program to sort a binary array in one pass
```

```
#include<stdio.h>

/*Function to put all 0s on left and all 1s on right*/
void segregate0and1(int arr[], int size)
{
    /* Initialize left and right indexes */
    int left = 0, right = size-1;

    while (left < right)
    {
        /* Increment left index while we see 0 at left */
        while (arr[left] == 0 && left < right)
            left++;

        /* Decrement right index while we see 1 at right */
        while (arr[right] == 1 && left < right)
            right--;

        /* If left is smaller than right then there is a 1 at left
           and a 0 at right. Exchange arr[left] and arr[right]*/
        if (left < right)
        {
            arr[left] = 0;
            arr[right] = 1;
            left++;
            right--;
        }
    }
}

/* driver program to test */
int main()
{
    int arr[] = {0, 1, 0, 1, 1, 1};
    int i, arr_size = sizeof(arr)/sizeof(arr[0]);

    segregate0and1(arr, arr_size);

    printf("Array after segregation ");
    for (i = 0; i < 6; i++)
        printf("%d ", arr[i]);

    getchar();
    return 0;
}
```

Java

```
class Segregate
{
    /*Function to put all 0s on left and all 1s on right*/
    void segregate0and1(int arr[], int size)
    {
        /* Initialize left and right indexes */
        int left = 0, right = size - 1;

        while (left < right)
        {
            /* Increment left index while we see 0 at left */
            while (arr[left] == 0 && left < right)
                left++;

            /* Decrement right index while we see 1 at right */
            while (arr[right] == 1 && left < right)
                right--;

            /* If left is smaller than right then there is a 1 at left
               and a 0 at right. Exchange arr[left] and arr[right]*/
            if (left < right)
            {
                arr[left] = 0;
                arr[right] = 1;
                left++;
                right--;
            }
        }
    }

    /* Driver Program to test above functions */
    public static void main(String[] args)
    {
        Segregate seg = new Segregate();
        int arr[] = new int[]{0, 1, 0, 1, 1, 1};
        int i, arr_size = arr.length;

        seg.segregate0and1(arr, arr_size);

        System.out.print("Array after segregation is ");
        for (i = 0; i < 6; i++)
            System.out.print(arr[i] + " ");
    }
}
```

Python

```
# Python program to sort a binary array in one pass
```

```
# Function to put all 0s on left and all 1s on right
def segregate0and1(arr, size):
    # Initialize left and right indexes
    left, right = 0, size-1

    while left < right:
        # Increment left index while we see 0 at left
        while arr[left] == 0 and left < right:
            left += 1

        # Decrement right index while we see 1 at right
        while arr[right] == 1 and left < right:
            right -= 1

        # If left is smaller than right then there is a 1 at left
        # and a 0 at right. Exchange arr[left] and arr[right]
        if left < right:
            arr[left] = 0
            arr[right] = 1
            left += 1
            right -= 1

    return arr

# driver program to test
arr = [0, 1, 0, 1, 1, 1]
arr_size = len(arr)
print("Array after segregation")
print(segregate0and1(arr, arr_size))

# This code is contributed by Pratik Chhajer
```

C#

```
// C# program to sort a binary array in one pass
using System;

class Segregate
{
    /*Function to put all 0s on
     left and all 1s on right*/
    void segregate0and1(int []arr, int size)
    {
        /* Initialize left and right indexes */
        int left = 0, right = size - 1;

        while (left < right)
```

```
{  
    /* Increment left index while  
       we see 0 at left */  
    while (arr[left] == 0 && left < right)  
        left++;  
  
    /* Decrement right index while  
       we see 1 at right */  
    while (arr[right] == 1 && left < right)  
        right--;  
  
    /* If left is smaller than right then  
       there is a 1 at left and a 0 at right.  
       Exchange arr[left] and arr[right]*/  
    if (left < right)  
    {  
        arr[left] = 0;  
        arr[right] = 1;  
        left++;  
        right--;  
    }  
}  
  
/* Driver Program to test above functions */  
public static void Main()  
{  
    Segregate seg = new Segregate();  
    int []arr = new int[]{0, 1, 0, 1, 1, 1};  
    int i, arr_size = arr.Length;  
  
    seg.segregate0and1(arr, arr_size);  
  
    Console.WriteLine("Array after segregation is ");  
    for (i = 0; i < 6; i++)  
        Console.Write(arr[i] + " ");  
}  
}  
  
//This code is contributed by vt_m.
```

PHP

```
<?php  
// PHP program to sort a  
// binary array in one pass  
  
// Function to put all 0s on
```

```
// left and all 1s on right
function segregate0and1(&$arr, $size)
{
    // Initialize left and
    // right indexes
    $left = 0;
    $right = $size - 1;

    while ($left < $right)
    {
        // Increment left index
        // while we see 0 at left
        while ($arr[$left] == 0 &&
               $left < $right)
            $left++;

        // Decrement right index
        // while we see 1 at right
        while ($arr[$right] == 1 &&
               $left < $right)
            $right--;

        // If left is smaller than right
        // then there is a 1 at left
        // and a 0 at right. Exchange
        // arr[left] and arr[right]
        if ($left < $right)
        {
            $arr[$left] = 0;
            $arr[$right] = 1;
            $left++;
            $right--;
        }
    }
}

// Driver code
$arr = array(0, 1, 0, 1, 1, 1);
$arr_size = sizeof($arr);

segregate0and1($arr, $arr_size);

printf("Array after segregation is ");
for ($i = 0; $i < 6; $i++)
    echo ($arr[$i]. " ");

// This code is contributed
// by Shivi_Agarwal
```

?>

Output:

```
Array after segregation is 0 0 1 1 1 1
```

Time Complexity: O(n)

Another approach :

1. Take two pointer type0(for element 0) starting from beginning (index = 0) and type1(for element 1) starting from end (index = array.length-1).
Initialize type0 = 0 and type1 = array.length-1
2. It is intended to Put 1 to the right side of the array. Once it is done, then 0 will definitely towards left side of array.

C++

```
// C++ program to sort a
// binary array in one pass
#include <bits/stdc++.h>
using namespace std;

/*Function to put all 0s on
left and all 1s on right*/
void segregate0and1(int arr[],
                     int size)
{
    int type0 = 0;
    int type1 = size - 1;

    while(type0 < type1)
    {
        if(arr[type0] == 1)
        {
            swap(arr[type0],
                  arr[type1]);
            type1--;
        }
        else
            type0++;
    }
}

// Driver Code
int main()
{
    int arr[] = {0, 1, 0, 1, 1, 1};
```

```
int i, arr_size = sizeof(arr) /  
                           sizeof(arr[0]);  
  
segregate0and1(arr, arr_size);  
  
cout << "Array after segregation is ";  
for (i = 0; i < arr_size; i++)  
    cout << arr[i] << " ";  
  
return 0;  
}
```

Java

```
// Java code to segregate 0 and 1  
import java.util.*;  
  
class GFG{  
/**  
Method for segregation 0 and 1 given input array  
*/  
static void segregate0and1(int arr[]) {  
    int type0 = 0;  
    int type1 = arr.length - 1;  
  
    while (type0 < type1) {  
        if (arr[type0] == 1) {  
            // swap  
            arr[type1] = arr[type1]+ arr[type0];  
            arr[type0] = arr[type1]-arr[type0];  
            arr[type1] = arr[type1]-arr[type0];  
            type1--;  
        } else {  
            type0++;  
        }  
    }  
}  
  
// Driver program  
public static void main(String[] args) {  
  
    int[] array = {0, 1, 0, 1, 1, 1};  
  
    segregate0and1(array);  
  
    for(int a : array){  
        System.out.print(a+" ");  
    }  
}
```

```
        }
    }
}
```

Python 3

```
# Python program to sort a
# binary array in one pass

# Function to put all 0s on
# left and all 1s on right
def segregate0and1(arr, size):

    type0 = 0
    type1 = size - 1

    while(type0 < type1):
        if(arr[type0] == 1):
            (arr[type0],
             arr[type1]) = (arr[type1],
                            arr[type0])
            type1 -= 1
        else:
            type0 += 1

    # Driver Code
    arr = [0, 1, 0, 1, 1, 1]
    arr_size = len(arr)
    segregate0and1(arr, arr_size)
    print("Array after segregation is",
          end = " ")
    for i in range(0, arr_size):
        print(arr[i], end = " ")

# This code is contributed
# by Shivi_Agarwal
```

PHP

```
<?php
// PHP program to sort a
// binary array in one pass

// Function to put all 0s on
// left and all 1s on right
function segregate0and1(&$arr , $size)
{
```

```
$type0 = 0;
$type1 = $size - 1;

while($type0 < $type1)
{
    if($arr[$type0] == 1)
    {
        $temp = $arr[$type0];
        $arr[$type0] = $arr[$type1];
        $arr[$type1] = $temp;
        $type1--;
    }
    else
        $type0++;
}
}

// Driver Code
$arr = array(0, 1, 0, 1, 1, 1);
$arr_size = sizeof($arr);

segregate0and1($arr, $arr_size);

echo ("Array after segregation is ");
for ($i = 0; $i < $arr_size; $i++)
    echo ($arr[$i] . " ");

// This code is contributed
// by Shivi_Agarwal
?>
```

Output:

```
Array after segregation is 0 0 1 1 1 1
```

Time complexity: O(n)

// Thanks [san4net](#) for suggesting this method.

Improved By : [nik1996](#), [Shivi_Agarwal](#)

Source

<https://www.geeksforgeeks.org/segregate-0s-and-1s-in-an-array-by-traversing-array-once/>

Chapter 294

Segregate even and odd numbers Set 3

Segregate even and odd numbers Set 3 - GeeksforGeeks

Given an array of integers, segregate even and odd numbers in the array. All the even numbers should be present first, and then the odd numbers.

Examples:

Input : 1 9 5 3 2 6 7 11
Output : 2 6 5 3 1 9 7 11

Input : 1 3 2 4 7 6 9 10
Output : 2 4 6 10 7 1 9 3

We have discussed two different approaches in below posts:

1. [Segregate Even and Odd numbers](#)
2. [Segregate even and odd numbers Set 2](#)

The idea discussed in this post is based on [Lomuto's Partition Scheme](#)

1. Maintain a pointer to the position before first odd element in the array.
2. Traverse the array and if even number is encountered then swap it with the first odd element.
3. Continue the traversal.

C++

```
// CPP code to segregate even odd
// numbers in an array
#include <bits/stdc++.h>
using namespace std;

// Function to segregate even odd numbers
void arrayEvenAndOdd(int arr[], int n)
{

    int i = -1, j = 0;
    int t;
    while (j != n) {
        if (arr[j] % 2 == 0) {
            i++;
            // Swapping even and odd numbers
            swap(arr[i], arr[j]);
        }
        j++;
    }

    // Printing segregated array
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver code
int main()
{
    int arr[] = { 1, 3, 2, 4, 7, 6, 9, 10 };
    int n = sizeof(arr) / sizeof(int);
    arrayEvenAndOdd(arr, n);
    return 0;
}
```

Java

```
// java code to segregate even odd
// numbers in an array
public class GFG {

    // Function to segregate even
    // odd numbers
    static void arrayEvenAndOdd(
        int arr[], int n)
    {

        int i = -1, j = 0;
```

```
while (j != n) {
    if (arr[j] % 2 == 0)
    {
        i++;
        // Swapping even and
        // odd numbers
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
    j++;
}

// Printing segregated array
for (int k = 0; k < n; k++)
    System.out.print(arr[k] + " ");
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 1, 3, 2, 4, 7,
                 6, 9, 10 };
    int n = arr.length;
    arrayEvenAndOdd(arr, n);
}
}

// This code is contributed by Sam007
```

C#

```
// C# code to segregate even odd
// numbers in an array
using System;

class GFG {

    // Function to segregate even
    // odd numbers
    static void arrayEvenAndOdd(
                int []arr, int n)
    {

        int i = -1, j = 0;
        while (j != n) {
            if (arr[j] % 2 == 0)
```

```
{  
    i++;  
  
    // Swapping even and  
    // odd numbers  
    int temp = arr[i];  
    arr[i] = arr[j];  
    arr[j] = temp;  
}  
j++;  
}  
  
// Printing segregated array  
for (int k = 0; k < n; k++)  
    Console.WriteLine(arr[k] + " ");  
}  
  
// Driver code  
static void Main()  
{  
    int []arr = { 1, 3, 2, 4, 7,  
                 6, 9, 10 };  
    int n = arr.Length;  
    arrayEvenAndOdd(arr, n);  
}  
}  
  
// This code is contributed by Sam007
```

PHP

```
<?php  
// PHP code to segregate even odd  
// numbers in an array  
  
// Function to segregate  
// even odd numbers  
function arrayEvenAndOdd($arr, $n)  
{  
    $i = -1;  
    $j = 0;  
    $t;  
    while ($j != $n)  
    {  
        if ($arr[$j] % 2 == 0)  
        {  
            $i++;
```

```
// Swapping even and
// odd numbers
$x = $arr[$i];
$arr[$i] = $arr[$j];
$arr[$j] = $x;
}
$j++;
}

// Printing segregated
// array
for ($i = 0; $i < $n; $i++)
    echo $arr[$i] . " ";
}

// Driver code
$arr = array(1, 3, 2, 4, 7, 6, 9, 10);
$n = sizeof($arr);
arrayEvenAndOdd($arr, $n);

// This code is contributed by Anuj_67
?>
```

Output:

2 4 6 10 7 1 9 3

Time Complexity : O(n)
Auxiliary Space : O(1)

Improved By : Sam007, vt_m

Source

<https://www.geeksforgeeks.org/segregate-even-odd-numbers-set-3/>

Chapter 295

Selection Sort

Selection Sort - GeeksforGeeks

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

- 1) The subarray which is already sorted.
- 2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Following example explains the above steps:

```
arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]
// and place it at beginning
11 25 12 22 64

// Find the minimum element in arr[1...4]
// and place it at beginning of arr[1...4]
11 12 25 22 64

// Find the minimum element in arr[2...4]
// and place it at beginning of arr[2...4]
11 12 22 25 64

// Find the minimum element in arr[3...4]
// and place it at beginning of arr[3...4]
11 12 22 25 64
```

C/C++

```
// C program for implementation of selection sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Python

```
# Python program for implementation of Selection
# Sort
import sys
A = [64, 25, 12, 22, 11]

# Traverse through all array elements
for i in range(len(A)):

    # Find the minimum element in remaining
    # unsorted array
    min_idx = i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx = j

    # Swap the found minimum element with
    # the first element
    A[i], A[min_idx] = A[min_idx], A[i]

# Driver code to test above
print ("Sorted array")
for i in range(len(A)):
    print("%d" %A[i]),
```

Java

```
// Java program for implementation of Selection Sort
class SelectionSort
{
    void sort(int arr[])
    {
        int n = arr.length;

        // One by one move boundary of unsorted subarray
        for (int i = 0; i < n-1; i++)
        {
            // Find the minimum element in unsorted array
            int min_idx = i;
            for (int j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min_idx = j;

            // Swap the found minimum element with the first
            // element
            int temp = arr[min_idx];
```

```
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

// Prints the array
void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i]+" ");
    System.out.println();
}

// Driver code to test above
public static void main(String args[])
{
    SelectionSort ob = new SelectionSort();
    int arr[] = {64,25,12,22,11};
    ob.sort(arr);
    System.out.println("Sorted array");
    ob.printArray(arr);
}
}
/* This code is contributed by Rajat Mishra*/
```

C#

```
// C# program for implementation
// of Selection Sort
using System;

class GFG
{
    static void sort(int []arr)
    {
        int n = arr.Length;

        // One by one move boundary of unsorted subarray
        for (int i = 0; i < n - 1; i++)
        {
            // Find the minimum element in unsorted array
            int min_idx = i;
            for (int j = i + 1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min_idx = j;

            // Swap the found minimum element with the first
            int temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }
    }
}
```

```
// element
    int temp = arr[min_idx];
    arr[min_idx] = arr[i];
    arr[i] = temp;
}
}

// Prints the array
static void printArray(int []arr)
{
    int n = arr.Length;
    for (int i=0; i<n; ++i)
        Console.Write(arr[i]+ " ");
    Console.WriteLine();
}

// Driver code
public static void Main()
{
    int []arr = {64,25,12,22,11};
    sort(arr);
    Console.WriteLine("Sorted array");
    printArray(arr);
}

}
// This code is contributed by Sam007
```

Output:

```
Sorted array:
11 12 22 25 64
```

Time Complexity: $O(n^2)$ as there are two nested loops.

Auxiliary Space: $O(1)$

The good thing about selection sort is it never makes more than $O(n)$ swaps and can be useful when memory write is a costly operation.

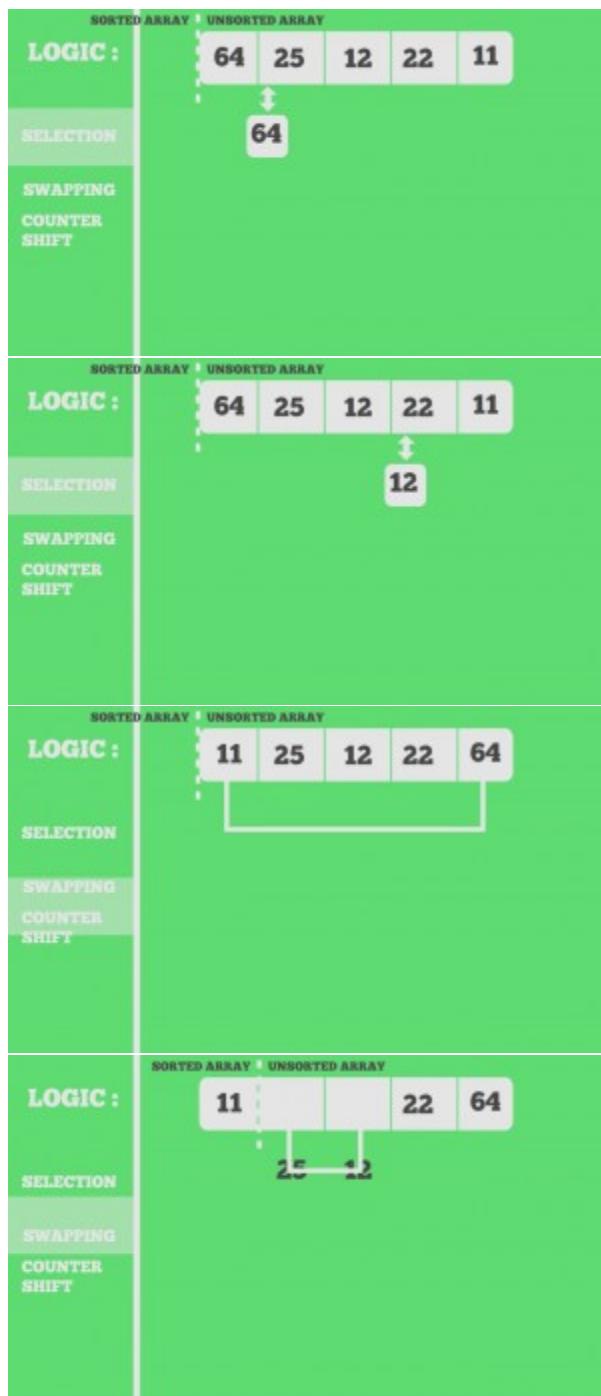
Exercise :

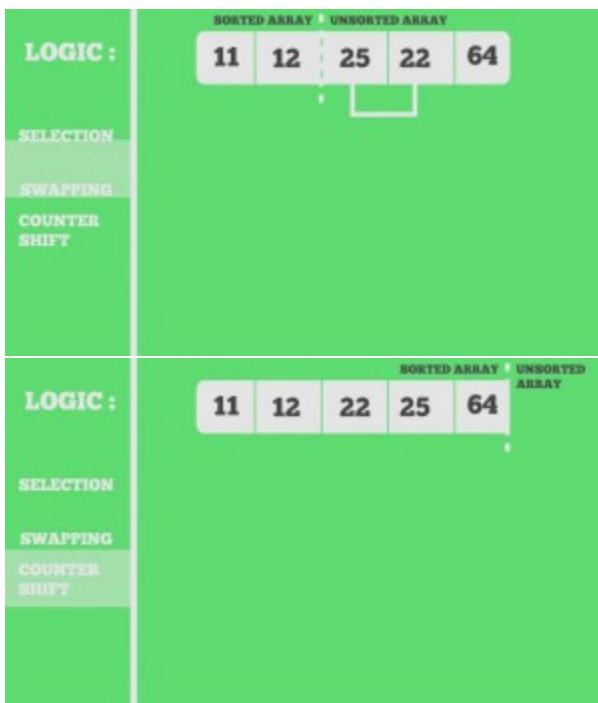
[Sort an array of strings using Selection Sort](#)

Stability : The default implementation is not stable. However it can be made stable. Please see [stable selection sort](#) for details.

In Place : Yes, it does not require extra space.

Snapshots:





Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

[Bubble Sort](#)

[Insertion Sort](#)

[Merge Sort](#)

[Heap Sort](#)

[QuickSort](#)

[Radix Sort](#)

[Counting Sort](#)

[Bucket Sort](#)

[ShellSort](#)

Source

<https://www.geeksforgeeks.org/selection-sort/>

Chapter 296

Serial Sort v/s Parallel Sort in Java

Serial Sort v/s Parallel Sort in Java - GeeksforGeeks

We often need to sort array while programming. For this, we use inbuilt method provided by Java in Arrays class i.e sort(). sort() method uses merge sort or Tim Sort to sort the array elements. In both the cases sort() method sequentially sort the elements of an array.

In Java 8, there is a new API introduced for sorting which is Parallel Sorting.

Parallel Sort uses [Fork/Join](#) framework introduced in Java 7 to assign the sorting tasks to multiple threads available in the thread pool. Fork/Join implements a [work stealing algorithm](#) where in a idle thread can steal tasks queued up in another thread.

For Example, the following code is a program that sorts a randomized array of doubles using [Arrays.sort\(\)](#) and [Arrays.parallelSort\(\)](#). This program simply measures the performance difference between this two approaches. :

```
// Java program to demonstrate time taken by sort()
// and parallelSort() methods.
import java.util.Arrays;

public class ParallelSortTest
{
    private static final int BASE_ARRAY_SIZE = 10000;

    // A utility function to generate and return an
    // an array of given size filled with randomly
    // generated elements.
    public static double[] generateArray(int size)
    {
        if (size <= 0 || size > Integer.MAX_VALUE)
            return null;
```

```
double[] result = new double[size];
for (int i = 0; i < size; i++)
    result[i] = Math.random();

return result;
}

// Driver code to compare two sortings
public static void main(String[] args)
{
    for (int i = 1; i < 10000; i *= 10)
    {
        int size = BASE_ARRAY_SIZE * i;
        double[] arr1 = generateArray(size);

        // Creating a copy of arr1 so that we can
        // use same content for both sortings.
        double[] arr2 = Arrays.copyOf(arr1, arr1.length);
        System.out.println("Array Size: " + size);

        // Sorting arr1[] using serial sort
        long startTime = System.currentTimeMillis();
        Arrays.sort(arr1);
        long endTime = System.currentTimeMillis();
        System.out.println("Time take in serial: " +
                           (endTime - startTime) + "ms.");

        // Sorting arr2[] using serial sort
        startTime = System.currentTimeMillis();
        Arrays.parallelSort(arr2);
        endTime = System.currentTimeMillis();
        System.out.println("Time take in parallel: " +
                           (endTime - startTime) + "ms.");
        System.out.println();
    }
}
}
```

Environment :

2.6 GHz Intel Core i7
java version "1.8.0_25"

Note : Required Time may vary due to random values in the array.

The key differences between both the algorithm are as follow :

1) **Arrays.sort()** : is a sequential sorting.

- The API uses single thread for the operation.
- It takes bit longer time to perform the operation.

2. **Arrays.ParallelSort()** : is a parallel sorting.

- The API uses multiple threads for the operation.
- It's faster when there are a lot of elements whereas slower for lesser elements.

Analysis :

The results show that parallel sorting on a multicore machine can achieve performance improvements at 1 million or more elements. While below this threshold it may actually be slower than sequential sorting. This result meets the expectation, and the suitable size here may be 1 million. Your mileage may vary, it depends on your environment.

Explanation :

Now, let's take a look at the code to figure out how this parallel sorting works.

```
public static void parallelSort(double[] a) {
    int n = a.length, p, g;
    if (n <= MIN_ARRAY_SORT_GRAN ||
        (p = ForkJoinPool.getCommonPoolParallelism()) == 1)
        DualPivotQuicksort.sort(a, 0, n - 1, null, 0, 0);
    else
        new ArraysParallelSortHelpers.FJDouble.Sorter
            (null, a, new double[n], 0, n, 0,
             ((g = n / (p << 2)) <= MIN_ARRAY_SORT_GRAN) ?
             MIN_ARRAY_SORT_GRAN : g).invoke();
}
```

As we can see, there is a minimum granularity (`java.util.Arrays.MIN_ARRAY_SORT_GRAN = 8192 [0x2000]`), and if the length of the array is less than the minimum granularity, it is sorted using the `DualPivotQuicksort.sort` directly instead of the sorting task partition. Typically, using smaller sizes results in memory contention across tasks that makes parallel speedups unlikely.

Another notable judgement is `ForkJoinPool.getCommonPoolParallelism()` which returns the targeted parallelism level of the common pool (by default, equal to the number of available processors `Runtime.getRuntime().availableProcessors()`). And if your machine has only 1 worker thread, it will not use parallel task either.

When the array length reaches a minimum granularity and you got more than 1 worker thread, the array is sorted using the **parallel sort method**. And the ForkJoin common pool is used to execute parallel tasks here.

Reference :

<http://download.java.net/lambda/b84/docs/api/java/util/Arrays.html#parallelSort%28int>

Source

<https://www.geeksforgeeks.org/serial-sort-vs-parallel-sort-java/>

Chapter 297

ShellSort

ShellSort - GeeksforGeeks

ShellSort is mainly a variation of [Insertion Sort](#). In insertion sort, we move elements only one position ahead. When an element has to be moved far ahead, many movements are involved. The idea of shellSort is to allow exchange of far items. In shellSort, we make the array h-sorted for a large value of h. We keep reducing the value of h until it becomes 1. An array is said to be h-sorted if all sublists of every h'th element is sorted.

Following is the implementation of ShellSort.

C++

```
// C++ implementation of Shell Sort
#include <iostream>
using namespace std;

/* function to sort arr using shellSort */
int shellSort(int arr[], int n)
{
    // Start with a big gap, then reduce the gap
    for (int gap = n/2; gap > 0; gap /= 2)
    {
        // Do a gapped insertion sort for this gap size.
        // The first gap elements a[0..gap-1] are already in gapped order
        // keep adding one more element until the entire array is
        // gap sorted
        for (int i = gap; i < n; i += 1)
        {
            // add a[i] to the elements that have been gap sorted
            // save a[i] in temp and make a hole at position i
            int temp = arr[i];

            // shift earlier gap-sorted elements up until the correct
            // position for a[i] is found
```

```

        // location for a[i] is found
        int j;
        for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
            arr[j] = arr[j - gap];

        // put temp (the original a[i]) in its correct location
        arr[j] = temp;
    }
}
return 0;
}

void printArray(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}

int main()
{
    int arr[] = {12, 34, 54, 2, 3}, i;
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Array before sorting: \n";
    printArray(arr, n);

    shellSort(arr, n);

    cout << "\nArray after sorting: \n";
    printArray(arr, n);

    return 0;
}

```

Java

```

// Java implementation of ShellSort
class ShellSort
{
    /* An utility function to print array of size n*/
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
}

```

```

/* function to sort arr using shellSort */
int sort(int arr[])
{
    int n = arr.length;

    // Start with a big gap, then reduce the gap
    for (int gap = n/2; gap > 0; gap /= 2)
    {
        // Do a gapped insertion sort for this gap size.
        // The first gap elements a[0..gap-1] are already
        // in gapped order keep adding one more element
        // until the entire array is gap sorted
        for (int i = gap; i < n; i += 1)
        {
            // add a[i] to the elements that have been gap
            // sorted save a[i] in temp and make a hole at
            // position i
            int temp = arr[i];

            // shift earlier gap-sorted elements up until
            // the correct location for a[i] is found
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];

            // put temp (the original a[i]) in its correct
            // location
            arr[j] = temp;
        }
    }
    return 0;
}

// Driver method
public static void main(String args[])
{
    int arr[] = {12, 34, 54, 2, 3};
    System.out.println("Array before sorting");
    printArray(arr);

    ShellSort ob = new ShellSort();
    ob.sort(arr);

    System.out.println("Array after sorting");
    printArray(arr);
}
}

/*This code is contributed by Rajat Mishra */

```

Python

```
# Python program for implementation of Shell Sort

def shellSort(arr):

    # Start with a big gap, then reduce the gap
    n = len(arr)
    gap = n/2

    # Do a gapped insertion sort for this gap size.
    # The first gap elements a[0..gap-1] are already in gapped
    # order keep adding one more element until the entire array
    # is gap sorted
    while gap > 0:

        for i in range(gap,n):

            # add a[i] to the elements that have been gap sorted
            # save a[i] in temp and make a hole at position i
            temp = arr[i]

            # shift earlier gap-sorted elements up until the correct
            # location for a[i] is found
            j = i
            while j >= gap and arr[j-gap] >temp:
                arr[j] = arr[j-gap]
                j -= gap

            # put temp (the original a[i]) in its correct location
            arr[j] = temp
        gap /= 2

# Driver code to test above
arr = [ 12, 34, 54, 2, 3]

n = len(arr)
print ("Array before sorting:")
for i in range(n):
    print(arr[i]),

shellSort(arr)

print ("\nArray after sorting:")
for i in range(n):
    print(arr[i]),
```

```
# This code is contributed by Mohit Kumra
```

C#

```
// C# implementation of ShellSort
using System;

class ShellSort
{
    /* An utility function to
       print array of size n*/
    static void printArray(int []arr)
    {
        int n = arr.Length;
        for (int i=0; i<n; ++i)
            Console.Write(arr[i] + " ");
        Console.WriteLine();
    }

    /* function to sort arr using shellSort */
    int sort(int []arr)
    {
        int n = arr.Length;

        // Start with a big gap,
        // then reduce the gap
        for (int gap = n/2; gap > 0; gap /= 2)
        {
            // Do a gapped insertion sort for this gap size.
            // The first gap elements a[0..gap-1] are already
            // in gapped order keep adding one more element
            // until the entire array is gap sorted
            for (int i = gap; i < n; i += 1)
            {
                // add a[i] to the elements that have
                // been gap sorted save a[i] in temp and
                // make a hole at position i
                int temp = arr[i];

                // shift earlier gap-sorted elements up until
                // the correct location for a[i] is found
                int j;
                for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                    arr[j] = arr[j - gap];

                // put temp (the original a[i])
                // in its correct location
                arr[j] = temp;
            }
        }
    }
}
```

```
        }
    }
    return 0;
}

// Driver method
public static void Main()
{
    int []arr = {12, 34, 54, 2, 3};
    Console.WriteLine("Array before sorting :\n");
    printArray(arr);

    ShellSort ob = new ShellSort();
    ob.sort(arr);

    Console.WriteLine("Array after sorting :\n");
    printArray(arr);
}
}

// This code is contributed by nitin mittal.
```

Output:

```
Array before sorting:
12 34 54 2 3
Array after sorting:
2 3 12 34 54
```

Time Complexity: Time complexity of above implementation of shellsort is $O(n^2)$. In the above implementation gap is reduce by half in every iteration. There are many other ways to reduce gap which lead to better time complexity. See [this](#)for more details.

References:

<https://www.youtube.com/watch?v=pGhazjsFW28>
<http://en.wikipedia.org/wiki/Shellsort>

Snapshots:

12

3

Start with gap = $n/2$ (2 in this case)

One by one select elements
place them at their appropriate position

12

3

Elements left of 54 are already sorted.

One by one select elements
place them at their appropriate positions.

12

3

Compare 2 with arr[3-2] = 34
arr[gap+1 = 3].

12

Compare 2 with arr[3-2] = 34
arr[gap+1 = 3].

3

T

Now gap reduces to $1(n/4)$.

Select all elements starting from them with elements within the

2

Now gap reduces to 0

Sorting stops and array is sorted

- Selection Sort
- Bubble Sort
- Insertion Sort
- Merge Sort
- Heap Sort
- QuickSort
- Radix Sort
- Counting Sort
- Bucket Sort

Source

<https://www.geeksforgeeks.org/shellsort/>

Chapter 298

Shortest Un-ordered Subarray

Shortest Un-ordered Subarray - GeeksforGeeks

An array is given of n length, and problem is that we have to find the length of shortest unordered {neither increasing nor decreasing} sub array in given array.

Examples:

```
Input : n = 5
        7 9 10 8 11
Output : 3
Explanation : 9 10 8 unordered sub array.
```

```
Input : n = 5
        1 2 3 4 5
Output : 0
Explanation : Array is in increasing order.
```

The idea is based on the fact that size of shortest subarray would be either 0 or 3. We have to check array element is either increasing or decreasing, if all array elements are in increasing or decreasing, then length of shortest sub array is 0, And if either the array element is not follow the increasing or decreasing then it shortest length is 3.

C++

```
// CPP program to find shortest subarray which is
// unsorted.
#include <bits/stdc++.h>
using namespace std;

// bool function for checking an array elements
// are in increasing.
bool increasing(int a[], int n)
```

```
{  
    for (int i = 0; i < n - 1; i++)  
        if (a[i] >= a[i + 1])  
            return false;  
    return true;  
}  
  
// bool function for checking an array  
// elements are in decreasing.  
bool decreasing(int a[], int n)  
{  
    for (int i = 0; i < n - 1; i++)  
        if (a[i] < a[i + 1])  
            return false;  
    return true;  
}  
  
int shortestUnsorted(int a[], int n)  
{  
    // increasing and decreasing are two functions.  
    // if function return true value then print  
    // 0 otherwise 3.  
    if (increasing(a, n) == true ||  
        decreasing(a, n) == true)  
        return 0;  
    else  
        return 3;  
}  
  
// Driver code  
int main()  
{  
    int ar[] = { 7, 9, 10, 8, 11 };  
    int n = sizeof(ar) / sizeof(ar[0]);  
    cout << shortestUnsorted(ar, n);  
    return 0;  
}
```

Java

```
// JAVA program to find shortest subarray which is  
// unsorted.  
import java.util.*;  
import java.io.*;  
  
class GFG {  
  
    // boolean function to check array elements
```

```
// are in increasing order or not
public static boolean increasing(int a[],int n)
{
    for (int i = 0; i < n - 1; i++)
        if (a[i] >= a[i + 1])
            return false;

    return true;
}

// boolean function to check array elements
// are in decreasing order or not
public static boolean decreasing(int arr[],int n)
{
    for (int i = 0; i < n - 1; i++)
        if (arr[i] < arr[i + 1])
            return false;

    return true;
}

public static int shortestUnsorted(int a[],int n)
{
    // increasing and decreasing are two functions.
    // if function return true value then print
    // 0 otherwise 3.
    if (increasing(a, n) == true ||
        decreasing(a, n) == true)
        return 0;
    else
        return 3;
}

// driver program
public static void main (String[] args) {

    int ar[] = new int[]{7, 9, 10, 8, 11};
    int n = ar.length;

    System.out.println(shortestUnsorted(ar,n));
}
}

// This code is contributed by Akash Singh.
```

Python3

```
# Python3 program to find shortest
# subarray which is unsorted

# Bool function for checking an array
# elements are in increasing
def increasing(a, n):

    for i in range(0, n - 1):
        if (a[i] >= a[i + 1]):
            return False

    return True

# Bool function for checking an array
# elements are in decreasing
def decreasing(a, n):

    for i in range(0, n - 1):
        if (a[i] < a[i + 1]):
            return False

    return True

def shortestUnsorted(a, n):

    # increasing and decreasing are two functions.
    # if function return True value then print
    # 0 otherwise 3.
    if (increasing(a, n) == True or
        decreasing(a, n) == True):
        return 0
    else:
        return 3

# Driver code
ar = [7, 9, 10, 8, 11]
n = len(ar)
print(shortestUnsorted(ar, n))

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
// Program to find the shortest
// subarray which is unsorted.
using System;

class GFG {
```

```
// boolean function to check
// array elements are in the
// increasing order or not
public static bool increasing(int[] a, int n)
{
    for (int i = 0; i < n - 1; i++)
        if (a[i] >= a[i + 1])
            return false;

    return true;
}

// boolean function to check
// array elements are in the
// decreasing order or not
public static bool decreasing(int[] arr, int n)
{
    for (int i = 0; i < n - 1; i++)
        if (arr[i] < arr[i + 1])
            return false;

    return true;
}

public static int shortestUnsorted(int[] a, int n)
{

    // increasing and decreasing are
    // two functions. function return
    // true value then print 0 else 3
    if (increasing(a, n) == true ||
        decreasing(a, n) == true)
        return 0;
    else
        return 3;
}

// Driver program
public static void Main()
{

    int[] ar = new int[] { 7, 9, 10, 8, 11 };
    int n = ar.Length;
    Console.WriteLine(shortestUnsorted(ar, n));
}
}
```

```
// This code is contributed by vt_m.
```

PHP

```
<?php
// php program to find shortest
// subarray which is unsorted.

// bool function for checking an
// array elements are in increasing.
function increasing($a, $n)
{
    for ( $i = 0; $i < $n - 1; $i++)
        if ($a[$i] >= $a[$i + 1])
            return false;

    return true;
}

// bool function for checking an
// array elements are in decreasing.
function decreasing($a, $n)
{
    for ($i = 0; $i < $n - 1; $i++)
        if ($a[$i] < $a[$i + 1])
            return false;
    return true;
}

function shortestUnsorted($a, $n)
{

    // increasing and decreasing are
    // two functions. if function
    // return true value then print
    // 0 otherwise 3.
    if (increasing($a, $n) == true ||
        decreasing($a, $n) == true)
        return 0;
    else
        return 3;
}

// Driver code
$ar = array( 7, 9, 10, 8, 11 );
$n = sizeof($ar);
echo shortestUnsorted($ar, $n);
```

```
// This code is contributed by  
// nitin mittal.  
?>
```

Output :

3

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/shortest-un-ordered-subarray/>

Chapter 299

Sleep Sort – The King of Laziness / Sorting while Sleeping

Sleep Sort – The King of Laziness / Sorting while Sleeping - GeeksforGeeks

In this algorithm we create different threads for each of the elements in the input array and then each thread sleeps for an amount of time which is proportional to the value of corresponding array element.

Hence, the thread having the least amount of sleeping time wakes up first and the number gets printed and then the second least element and so on. The largest element wakes up after a long time and then the element gets printed at the last. Thus the output is a sorted one.

All this multithreading process happens in background and at the core of the OS. We do not get to know anything about what's happening in the background, hence this is a "mysterious" sorting algorithm.

Example : Let's assume (for convenience) we have a computer that's so slow it takes 3 seconds to work through each element:

INPUT: 8 2 9

```
3s: sleep 8
6s: sleep 2
8s: "2" (2 wakes up so print it)
9s: sleep 9
11s: "8" (8 wakes up so print it)
18s: "9" (9 wakes up so print it)
```

OUTPUT: 2 8 9

Implementation

To implement sleep sort, we need multithreading functions, such as `_beginthread()` and `WaitForMultipleObjects()`. Hence we need to include `windows.h` to use these functions. This won't compile on [Online IDE](#). We must run it in your PC (Note this code is for WINDOWS and not for LINUX).

To perform a sleep sort we need to create threads for each of the value in the input array. We do this using the function `_beginthread()`.

In each of the threads we assign two instructions:

1) Sleep : Sleep this thread till `arr[i]` milliseconds (where `arr[i]` is the array element which this thread is associated to). We do this using `Sleep()` function. The `Sleep(n)` function suspends the activity associated with this thread till 'n' milliseconds. Hence if we write `Sleep(1000)`, then it means that the thread will sleep for 1 second (1000 milliseconds = 1 second)

2) Print : When the thread 'wakes' up after the sleep then print the array element – `arr[i]` which this thread is associated to.

After creating the threads, we process these threads. We do this using `WaitForMultipleObjects()`.

```
// C implementation of Sleep Sort
#include <stdio.h>
#include <windows.h>
#include <process.h>

// This is the instruction set of a thread
// So in these threads, we "sleep" for a particular
// amount of time and then when it wakes up
// the number is printed out
void routine(void *a)
{
    int n = *(int *) a; // typecasting from void to int

    // Sleeping time is proportional to the number
    // More precisely this thread sleep for 'n' milliseconds
    Sleep(n);

    // After the sleep, print the number
    printf("%d ", n);
}

/* A function that performs sleep sort
_beginthread() is a C run-time library call that creates a new
'thread' for all the integers in the array and returns that
thread.

Each of the 'thread' sleeps for a time proportional to that
```

integer and print it after waking.

We pass three parameters to `_beginthread` :-

- 1) `start_address` --> start address of the routine/function which creates a new thread
- 2) `stack_size` --> Stack Size of the new thread (which is 0)
- 3) `arglist` --> Address of the argument to be passed

The return value of `_beginthread()` function is a handle to the thread which is created. So we must accept it using the datatype- 'HANDLE' which is included in windows.h header

'HANDLE' datatype is used to represent an event/thread/process etc
So 'HANDLE' datatype is used to define a thread

We store the threads in an array - `threads[]` which is declared using 'HANDLE' datatype.

`WaitForMultipleObjects()` is a function that processes the threads and has four arguments-

- 1) `no_of_threads` --> Number of threads to be processed
- 2) `array_of_threads` --> This is the array of threads which should be processed. This array must be of the type 'HANDLE'
- 3) TRUE or FALSE --> We pass TRUE if we want all the threads in the array to be processed
- 4) `time_limit` --> The threads will be processed until this time limit is crossed. So if we pass a 0 then no threads will be processed, otherwise if we pass an INFINITE, then the program will stop only when all the threads are processed. We can put a cap on the execution time of the program by passing the desired time limit */

```
void sleepSort(int arr[], int n)
{
    // An array of threads, one for each of the elements
    // in the input array
    HANDLE threads[n];

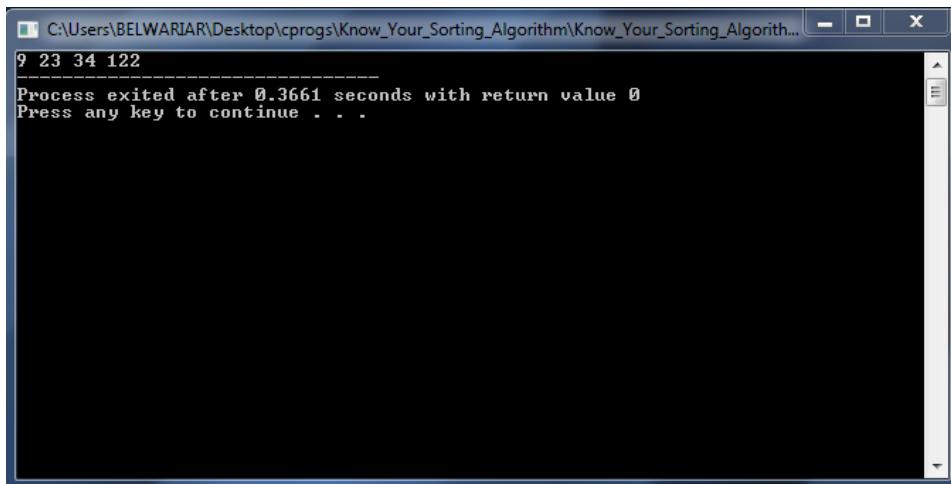
    // Create the threads for each of the input array elements
    for (int i = 0; i < n; i++)
        threads[i] = (HANDLE)_beginthread(&routine, 0, &arr[i]);
```

```
    // Process these threads
    WaitForMultipleObjects(n, threads, TRUE, INFINITE);
    return;
```

```
}
```

```
// Driver program to test above functions
int main()
```

```
{  
    // Doesn't work for negative numbers  
    int arr[] = {34, 23, 122, 9};  
    int n = sizeof(arr) / sizeof(arr[0]);  
  
    sleepSort (arr, n);  
  
    return(0);  
}
```



Limitations

- 1) This algorithm won't work for negative numbers as a thread cannot sleep for a negative amount of time.
- 2) Since this algorithm depends on the input elements, so a huge number in the input array causes this algorithm to slow down drastically (as the thread associated with that number has to sleep for a long time). So even if the input array element contains only 2 elements, like- {1, 100000000}, then also we have to wait for a much longer duration to sort.
- 3) This algorithm doesn't produce a correct sorted output every time. This generally happens when there is a very small number to the left of a very large number in the input array. For example – {34, 23, 1, 12253, 9}.

The output after sleep sorting is {9, 1, 23, 34, 1223}

A wrong output also occurs when the input array is reverse sorted initially, like- {10, 9, 8, 7, 6, 5}.

The reason for such an unexpected output is because some time is taken between scanning through each element as well as some other OS operations (like inserting each threads in a priority queue for scheduling). We cannot simply ignore the time taken by all these things.

We describe this using the below example-

Let's assume (for convenience) we have a computer that's so slow it takes 3 seconds to work through each element:
INPUT: 10 9 8 7 6 5

```
3s: sleep 10
6s: sleep 9
9s: sleep 8
12s: sleep 7
13s: "10" (10 wakes up so print it)
15s: sleep 6
15s: "9" (9 wakes up so print it)
17s: "8" (8 wakes up so print it)
18s: sleep 5
19s: "7" (7 wakes up so print it)
21s: "6" (6 wakes up so print it)
23s: "5" (5 wakes up so print it)
```

OUTPUT: 10 9 8 7 6 5

The above output is just an example.

Obviously, modern-day computers computer are not so slow (to take 3 seconds to scan through each element).

In reality running sleep sort on a modern computer on the above array gives the output – {9, 5, 7, 10, 8, 6}

How to fix this ?

- 1) We can fix this by repeatedly sleep sorting on the new output until the output becomes sorted. Every time it will sort the elements more accurately.
- 2) The wrong output as discussed earlier happens due to the time taken by other OS works and scanning through each element.

In our program we have used the function **Sleep(arr[i])**, which means that each thread associated with the array elements sleep for 'arr[i]' milliseconds. Since milliseconds is a very small quantity and other OS tasks can take more time than 'arr[i]' milliseconds which ultimately can cause an error in sleep sorting. Increasing the sleeping time by even 10 times can give a sorted output as the OS tasks will finish all its task in between this much sleep, hence not producing any errors.

Had we used **Sleep(10*arr[i])** instead of just **Sleep(arr[i])** then we will certainly get a more precise output than the latter one. For example the input array – {10, 9, 8, 7, 6, 5} will give the correct sorted output – {5, 6, 7, 8, 9, 10} if we use **Sleep(10*arr[i])** instead of just **Sleep(arr[i])** .

However it is still possible that **Sleep(10*arr[i])** will give wrong results for some test cases. To make it more precise increase the sleep time more , say something like – **Sleep(20*arr[i])**.

Hence the bottom line is that more the sleep time, more accurate the results are. (Sounds interesting, eh?) . But again that would increase the runtime of this algorithm.

Exercise to the readers-

1) The above algorithm tries to sort it in ascending order. Can you sort an input array in descending order using sleep sort. Think upon it.

2) Is it a comparison based sorting algorithm ? How many comparisons this algorithm makes ?

[Answer : No, it makes zero comparisons]

3) Can we do sleeping sort without using windows.h header and without using Sleep() function?

[One idea can be to create a priority queue where the elements are arranged according to the time left before waking up and getting printed. The element at the front of the priority queue will be the first one to get waked up. However the implementation doesn't looks easy. Think on it.]

Time Complexity

Although there are many conflicting opinions about the time complexity of sleep sort, but we can approximate the time complexity using the below reasoning-

Since Sleep() function and creating multiple threads is done internally by the OS using a priority queue (used for scheduling purposes). Hence inserting all the array elements in the priority queue takes $O(N \log N)$ time. Also the output is obtained only when all the threads are processed, i.e- when all the elements ‘wakes’ up. Since it takes $O(\text{arr}[i])$ time to wake the ith array element’s thread. So it will take a maximum of $O(\max(\text{input}))$ for the largest element of the array to wake up. Thus the overall time complexity can be assumed as $O(N \log N + \max(\text{input}))$,

where, N = number of elements in the input array, and input = input array elements

Auxiliary Space

All the things are done by the internal priority queue of the OS. Hence auxiliary space can be ignored.

Conclusion

Sleep Sort is related to Operating System more than any other sorting algorithm. This sorting algorithm is a perfect demonstration of multi-threading and scheduling done by OS.

The phrase “Sorting while Sleeping” itself sounds very unique. Overall it is a fun, lazy, weird algorithm. But as rightly said by someone- “If it works then it is not lazy”.

Source

<https://www.geeksforgeeks.org/sleep-sort-king-laziness-sorting-sleeping/>

Chapter 300

Smallest Difference Triplet from Three arrays

Smallest Difference Triplet from Three arrays - GeeksforGeeks

Three arrays of same size are given. Find a triplet such that maximum – minimum in that triplet is minimum of all the triplets. A triplet should be selected in a way such that it should have one number from each of the three given arrays.

If there are 2 or more smallest difference triplets, then the one with the smallest sum of its elements should be displayed.

Examples :

```
Input : arr1[] = {5, 2, 8}
        arr2[] = {10, 7, 12}
        arr3[] = {9, 14, 6}
Output : 8, 7, 6
```

```
Input : arr1[] = {15, 12, 18, 9}
        arr2[] = {10, 17, 13, 8}
        arr3[] = {14, 16, 11, 5}
Output : 11, 10, 9
```

Note:The elements of the triplet are displayed in non-decreasing order.

Simple Solution : Consider each an every triplet and find the required smallest difference triplet out of them. Complexity of $O(n^3)$.

Efficient Solution:

1. Sort the 3 arrays in non-decreasing order.
2. Start three pointers from left most elements of three arrays.

3. Now find min and max and calculate max-min from these three elements.
4. Now increment pointer of minimum element's array.
5. Repeat steps 2, 3, 4, for the new set of pointers until any one pointer reaches to its end.

C++

```
// C++ implementation of smallest difference triplet
#include <bits/stdc++.h>
using namespace std;

// function to find maximum number
int maximum(int a, int b, int c)
{
    return max(max(a, b), c);
}

// function to find minimum number
int minimum(int a, int b, int c)
{
    return min(min(a, b), c);
}

// Finds and prints the smallest Difference Triplet
void smallestDifferenceTriplet(int arr1[], int arr2[],
                               int arr3[], int n)
{
    // sorting all the three arrays
    sort(arr1, arr1+n);
    sort(arr2, arr2+n);
    sort(arr3, arr3+n);

    // To store resultant three numbers
    int res_min, res_max, res_mid;

    // pointers to arr1, arr2, arr3
    // respectively
    int i = 0, j = 0, k = 0;

    // Loop until one array reaches to its end
    // Find the smallest difference.
    int diff = INT_MAX;
    while (i < n && j < n && k < n)
    {
        int sum = arr1[i] + arr2[j] + arr3[k];
```

```
// maximum number
int max = maximum(arr1[i], arr2[j], arr3[k]);

// Find minimum and increment its index.
int min = minimum(arr1[i], arr2[j], arr3[k]);
if (min == arr1[i])
    i++;
else if (min == arr2[j])
    j++;
else
    k++;

// comparing new difference with the
// previous one and updating accordingly
if (diff > (max-min))
{
    diff = max - min;
    res_max = max;
    res_mid = sum - (max + min);
    res_min = min;
}
}

// Print result
cout << res_max << ", " << res_mid << ", " << res_min;
}

// Driver program to test above
int main()
{
    int arr1[] = {5, 2, 8};
    int arr2[] = {10, 7, 12};
    int arr3[] = {9, 14, 6};
    int n = sizeof(arr1) / sizeof(arr1[0]);
    smallestDifferenceTriplet(arr1, arr2, arr3, n);
    return 0;
}
```

Java

```
// Java implementation of smallest difference
// triplet
import java.util.Arrays;

class GFG {

    // function to find maximum number
    static int maximum(int a, int b, int c)
```

```
{  
    return Math.max(Math.max(a, b), c);  
}  
  
// function to find minimum number  
static int minimum(int a, int b, int c)  
{  
    return Math.min(Math.min(a, b), c);  
}  
  
// Finds and prints the smallest Difference  
// Triplet  
static void smallestDifferenceTriplet(int arr1[],  
                                      int arr2[], int arr3[], int n)  
{  
  
    // sorting all the three arrays  
    Arrays.sort(arr1);  
    Arrays.sort(arr2);  
    Arrays.sort(arr3);  
  
    // To store resultant three numbers  
    int res_min=0, res_max=0, res_mid=0;  
  
    // pointers to arr1, arr2, arr3  
    // respectively  
    int i = 0, j = 0, k = 0;  
  
    // Loop until one array reaches to its end  
    // Find the smallest difference.  
    int diff = 2147483647;  
  
    while (i < n && j < n && k < n)  
    {  
        int sum = arr1[i] + arr2[j] + arr3[k];  
  
        // maximum number  
        int max = maximum(arr1[i], arr2[j], arr3[k]);  
  
        // Find minimum and increment its index.  
        int min = minimum(arr1[i], arr2[j], arr3[k]);  
        if (min == arr1[i])  
            i++;  
        else if (min == arr2[j])  
            j++;  
        else  
            k++;  
    }  
}
```

```
// comparing new difference with the
// previous one and updating accordingly
if (diff > (max - min))
{
    diff = max - min;
    res_max = max;
    res_mid = sum - (max + min);
    res_min = min;
}
}

// Print result
System.out.print(res_max + ", " + res_mid
                  + ", " + res_min);
}

//driver code
public static void main (String[] args)
{
    int arr1[] = {5, 2, 8};
    int arr2[] = {10, 7, 12};
    int arr3[] = {9, 14, 6};

    int n = arr1.length;

    smallestDifferenceTriplet(arr1, arr2, arr3, n);
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 implementation of smallest
# difference triplet

# Function to find maximum number
def maximum(a, b, c):
    return max(max(a, b), c)

# Function to find minimum number
def minimum(a, b, c):
    return min(min(a, b), c)

# Finds and prints the smallest
# Difference Triplet
def smallestDifferenceTriplet(arr1, arr2, arr3, n):
```

```
# sorting all the three arrays
arr1.sort()
arr2.sort()
arr3.sort()

# To store resultant three numbers
res_min = 0; res_max = 0; res_mid = 0

# pointers to arr1, arr2,
# arr3 respectively
i = 0; j = 0; k = 0

# Loop until one array reaches to its end
# Find the smallest difference.
diff = 2147483647
while (i < n and j < n and k < n):

    sum = arr1[i] + arr2[j] + arr3[k]

    # maximum number
    max = maximum(arr1[i], arr2[j], arr3[k])

    # Find minimum and increment its index.
    min = minimum(arr1[i], arr2[j], arr3[k])
    if (min == arr1[i]):
        i += 1
    elif (min == arr2[j]):
        j += 1
    else:
        k += 1

    # Comparing new difference with the
    # previous one and updating accordingly
    if (diff > (max - min)):

        diff = max - min
        res_max = max
        res_mid = sum - (max + min)
        res_min = min

    # Print result
    print(res_max, ",", res_mid, ",", res_min)

# Driver code
arr1 = [5, 2, 8]
arr2 = [10, 7, 12]
arr3 = [9, 14, 6]
```

```
n = len(arr1)
smallestDifferenceTriplet(arr1, arr2, arr3, n)
```

```
# This code is contributed by Anant Agarwal.
```

C#

```
// C# implementation of smallest
// difference triplet
using System;

class GFG
{

    // function to find
    // maximum number
    static int maximum(int a, int b, int c)
    {
        return Math.Max(Math.Max(a, b), c);
    }

    // function to find
    // minimum number
    static int minimum(int a, int b, int c)
    {
        return Math.Min(Math.Min(a, b), c);
    }

    // Finds and prints the
    // smallest Difference Triplet
    static void smallestDifferenceTriplet(int []arr1,
                                           int []arr2,
                                           int []arr3,
                                           int n)
    {

        // sorting all the
        // three arrays
        Array.Sort(arr1);
        Array.Sort(arr2);
        Array.Sort(arr3);

        // To store resultant
        // three numbers
        int res_min = 0, res_max = 0, res_mid = 0;

        // pointers to arr1, arr2,
        // arr3 respectively
```

```
int i = 0, j = 0, k = 0;

// Loop until one array
// reaches to its end
// Find the smallest difference.
int diff = 2147483647;

while (i < n && j < n && k < n)
{
    int sum = arr1[i] +
              arr2[j] + arr3[k];

    // maximum number
    int max = maximum(arr1[i],
                       arr2[j], arr3[k]);

    // Find minimum and
    // increment its index.
    int min = minimum(arr1[i],
                       arr2[j], arr3[k]);
    if (min == arr1[i])
        i++;
    else if (min == arr2[j])
        j++;
    else
        k++;

    // comparing new difference
    // with the previous one and
    // updating accordingly
    if (diff > (max - min))
    {
        diff = max - min;
        res_max = max;
        res_mid = sum - (max + min);
        res_min = min;
    }
}

// Print result
Console.WriteLine(res_max + ", " +
                  res_mid + ", " +
                  res_min);

// Driver code
static public void Main ()
{
```

```
int []arr1 = {5, 2, 8};
int []arr2 = {10, 7, 12};
int []arr3 = {9, 14, 6};

int n = arr1.Length;

smallestDifferenceTriplet(arr1, arr2,
                           arr3, n);
}

}

// This code is contributed by ajit.
```

PHP

```
<?php
// PHP implementation of
// smallest difference triplet

// function to find
// maximum number
function maximum($a, $b, $c)
{
    return max(max($a, $b), $c);
}

// function to find
// minimum number
function minimum($a, $b, $c)
{
    return min(min($a, $b), $c);
}

// Finds and prints the
// smallest Difference Triplet
function smallestDifferenceTriplet($arr1, $arr2,
                                   $arr3, $n)
{
    // sorting all the
    // three arrays
    sort($arr1);
    sort($arr2);
    sort($arr3);

    // To store resultant
    // three numbers
    $res_min; $res_max; $res_mid;
```

```
// pointers to arr1, arr2,
// arr3 respectively
$i = 0; $j = 0; $k = 0;

// Loop until one array reaches
// to its end
// Find the smallest difference.
$diff = PHP_INT_MAX;
while ($i < $n && $j < $n && $k < $n)
{
    $sum = $arr1[$i] +
           $arr2[$j] +
           $arr3[$k];

    // maximum number
    $max = maximum($arr1[$i],
                   $arr2[$j],
                   $arr3[$k]);

    // Find minimum and
    // increment its index.
    $min = minimum($arr1[$i],
                   $arr2[$j],
                   $arr3[$k]);
    if ($min == $arr1[$i])
        $i++;
    else if ($min == $arr2[$j])
        $j++;
    else
        $k++;

    // comparing new difference
    // with the previous one
    // and updating accordingly
    if ($diff > ($max - $min))
    {
        $diff = $max - $min;
        $res_max = $max;
        $res_mid = $sum - ($max + $min);
        $res_min = $min;
    }
}

// Print result
echo $res_max , " , " ,
      $res_mid , " , " ,
      $res_min;
```

}

```
// Driver Code
$arr1 = array(5, 2, 8);
$arr2 = array(10, 7, 12);
$arr3 = array(9, 14, 6);

$n = sizeof($arr1);
smallestDifferenceTriplet($arr1, $arr2,
                           $arr3, $n);

// This code is contributed by ajit
?>
```

Output :

7, 6, 5

Time Complexity : $O(n \log n)$

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/smallest-difference-triplet-from-three-arrays/>

Chapter 301

Smallest Difference pair of values between two unsorted Arrays

Smallest Difference pair of values between two unsorted Arrays - GeeksforGeeks

Given two arrays of integers, compute the pair of values (one value in each array) with the smallest (non-negative) difference. Return the difference.

Examples :

```
Input : A[] = {1, 3, 15, 11, 2}  
        B[] = {23, 127, 235, 19, 8}  
Output : 3  
That is, the pair (11, 8)
```

```
Input : A[] = {10, 5, 40}  
        B[] = {50, 90, 80}  
Output : 10  
That is, the pair (40, 50)
```

A **simple solution** is to Brute Force using two loops with **Time Complexity $O(n^2)$** .

A **better solution** is to sort the arrays. Once the arrays are sorted, we can find the minimum difference by iterating through the arrays using the approach discussed in below post.

[Find the closest pair from two sorted arrays](#)

Consider the following two arrays:

A: {1, 2, 11, 15}
B: {4, 12, 19, 23, 127, 235}

1. Suppose a pointer a points to the beginning of A and a pointer b points to the beginning of B. The current difference between a and b is 3. Store this as the min.
2. How can we (potentially) make this difference smaller? Well, the value at b is bigger than the value at a, so moving b will only make the difference larger. Therefore, we want to move a.
3. Now a points to 2 and b (still) points to 4. This difference is 2, so we should update min. Move a, since it is smaller.
4. Now a points to 11 and b points to 4. Move b.
5. Now a points to 11 and b points to 12. Update min to 1. Move b. And so on.

Below is the implementation of the idea.

C++

```
// C++ Code to find Smallest
// Difference between two Arrays
#include <bits/stdc++.h>
using namespace std;

// function to calculate Small
// result between two arrays
int findSmallestDifference(int A[], int B[],
                           int m, int n)
{
    // Sort both arrays using
    // sort function
    sort(A, A + m);
    sort(B, B + n);

    int a = 0, b = 0;

    // Initialize result as max value
    int result = INT_MAX;

    // Scan Both Arrays upto
    // sizeof of the Arrays
    while (a < m && b < n)
    {
        if (abs(A[a] - B[b]) < result)
            result = abs(A[a] - B[b]);

        // Move Smaller Value
        if (A[a] < B[b])
            a++;
        else
            b++;
    }
}
```

```
}

// return final sma result
return result;
}

// Driver Code
int main()
{
    // Input given array A
    int A[] = {1, 2, 11, 5};

    // Input given array B
    int B[] = {4, 12, 19, 23, 127, 235};

    // Calculate size of Both arrays
    int m = sizeof(A) / sizeof(A[0]);
    int n = sizeof(B) / sizeof(B[0]);

    // Call function to print
    // smallest result
    cout << findSmallestDifference(A, B, m, n);

    return 0;
}
```

Java

```
// Java Code to find Smallest
// Difference between two Arrays
import java.util.*;

class GFG
{

    // function to calculate Small
    // result between two arrays
    static int findSmallestDifference(int A[], int B[],
                                      int m, int n)
    {
        // Sort both arrays
        // using sort function
        Arrays.sort(A);
        Arrays.sort(B);

        int a = 0, b = 0;
```

```
// Initialize result as max value
int result = Integer.MAX_VALUE;

// Scan Both Arrays upto
// sizeof of the Arrays
while (a < m && b < n)
{
    if (Math.abs(A[a] - B[b]) < result)
        result = Math.abs(A[a] - B[b]);

    // Move Smaller Value
    if (A[a] < B[b])
        a++;

    else
        b++;
}

// return final sma result
return result;
}

// Driver Code
public static void main(String[] args)
{
    // Input given array A
    int A[] = {1, 2, 11, 5};

    // Input given array B
    int B[] = {4, 12, 19, 23, 127, 235};

    // Calculate size of Both arrays
    int m = A.length;
    int n = B.length;

    // Call function to
    // print smallest result
    System.out.println(findSmallestDifference
        (A, B, m, n));
}

// This code is contributed
// by Arnav Kr. Mandal.
```

Python3

```
# Python 3 Code to find
# Smallest Difference between
# two Arrays
import sys

# function to calculate
# Small result between
# two arrays
def findSmallestDifference(A, B, m, n):

    # Sort both arrays
    # using sort function
    A.sort()
    B.sort()

    a = 0
    b = 0

    # Initialize result as max value
    result = sys.maxsize

    # Scan Both Arrays upto
    # sizeof of the Arrays
    while (a < m and b < n):

        if (abs(A[a] - B[b]) < result):
            result = abs(A[a] - B[b])

        # Move Smaller Value
        if (A[a] < B[b]):
            a += 1

        else:
            b += 1
    # return final sma result
    return result

# Driver Code

# Input given array A
A = [1, 2, 11, 5]

# Input given array B
B = [4, 12, 19, 23, 127, 235]

# Calculate size of Both arrays
m = len(A)
n = len(B)
```

```
# Call function to
# print smallest result
print(findSmallestDifference(A, B, m, n))

# This code is contributed by
# Smitha Dinesh Semwal

C#

// C# Code to find Smallest
// Difference between two Arrays
using System;

class GFG
{

    // function to calculate Small
    // result between two arrays
    static int findSmallestDifference(int []A, int []B,
                                      int m, int n)
    {

        // Sort both arrays using
        // sort function
        Array.Sort(A);
        Array.Sort(B);

        int a = 0, b = 0;

        // Initialize result as max value
        int result = int.MaxValue;

        // Scan Both Arrays upto
        // sizeof of the Arrays
        while (a < m && b < n)
        {
            if (Math.Abs(A[a] - B[b]) < result)
                result = Math.Abs(A[a] - B[b]);

            // Move Smaller Value
            if (A[a] < B[b])
                a++;

            else
                b++;
        }
    }
}
```

```
// return final sma result
return result;
}

// Driver Code
public static void Main()
{
    // Input given array A
    int []A = {1, 2, 11, 5};

    // Input given array B
    int []B = {4, 12, 19, 23, 127, 235};

    // Calculate size of Both arrays
    int m = A.Length;
    int n = B.Length;

    // Call function to
    // print smallest result
    Console.WriteLine(findSmallestDifference
        (A, B, m, n));
}

// This code is contributed
// by nitin mittal.
```

PHP

```
<?php
// PHP Code to find Smallest
// Difference between two Arrays

// function to calculate Small
// result between two arrays
function findSmallestDifference($A, $B,
                                $m, $n)
{
    // Sort both arrays
    // using sort function
    sort($A);
    sort($A, $m);
    sort($B);
    sort($B, $n);
```

```
$a = 0; $b = 0;
$INT_MAX = 1;

// Initialize result
// as max value
$result = $INT_MAX;

// Scan Both Arrays upto
// sizeof of the Arrays
while ($a < $m && $b < $n)
{
    if (abs($A[$a] - $B[$b]) < $result)
        $result = abs($A[$a] - $B[$b]);

    // Move Smaller Value
    if ($A[$a] < $B[$b])
        $a++;

    else
        $b++;
}

// return final sma result
return $result;
}

// Driver Code
{
    // Input given array A
    $A = array(1, 2, 11, 5);

    // Input given array B
    $B = array(4, 12, 19, 23, 127, 235);

    // Calculate size of Both arrays
    $m = sizeof($A) / sizeof($A[0]);
    $n = sizeof($B) / sizeof($B[0]);

    // Call function to print
    // smallest result
    echo findSmallestDifference($A, $B, $m, $n);

    return 0;
}

// This code is contributed by nitin mittal.
```

?>

Output :

1

This algorithm takes $O(m \log m + n \log n)$ time to sort and $O(m + n)$ time to find the minimum difference. Therefore, the overall runtime is $O(m \log m + n \log n)$.

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/smallest-difference-pair-values-two-unsorted-arrays/>

Chapter 302

Smallest element in an array that is repeated exactly ‘k’ times.

Smallest element in an array that is repeated exactly 'k' times. - GeeksforGeeks

Given an array of size n, the goal is to find out the smallest number that is repeated exactly 'k' times where k > 0?

Assume that array has only positive integers and $1 \leq \text{arr}[i] < 1000$ for each $i = 0$ to $n - 1$.

Examples:

```
Input : arr[] = {2 2 1 3 1}
        k = 2
Output: 1
Explanation:
Here in array,
2 is repeated 2 times
1 is repeated 2 times
3 is repeated 1 time
Hence 2 and 1 both are repeated 'k' times
i.e 2 and min(2, 1) is 1
```

```
Input : arr[] = {3 5 3 2}
        k = 1
Output : 2
Explanation:
Both 2 and 5 are repeating 1 time but
min(5, 2) is 2
```

Simple Approach: A simple approach is to use two nested loops. The outer loop picks an

element one by one starting from the leftmost element. The inner loop checks if the same element is present on right side of it. If present increase the count and make the number negative which we got at the right side to prevent it from counting again.

C++

```
// C++ program to find smallest number
// in array that is repeated exactly
// 'k' times.
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1000;

int findDuplicate(int arr[], int n, int k)
{
    // Since arr[] has numbers in range from
    // 1 to MAX
    int res = MAX + 1;
    for (int i = 0; i < n; i++) {
        if (arr[i] > 0) {

            // set count to 1 as number is present
            // once
            int count = 1;
            for (int j = i + 1; j < n; j++)
                if (arr[i] == arr[j])
                    count += 1;

            // If frequency of number is equal to 'k'
            if (count == k)
                res = min(res, arr[i]);
        }
    }
    return res;
}

// Driver code
int main()
{
    int arr[] = { 2, 2, 1, 3, 1 };
    int k = 2;
    int n = sizeof(arr) / (sizeof(arr[0]));
    cout << findDuplicate(arr, n, k);
    return 0;
}
```

Java

```
// Java program to find smallest number
// in array that is repeated exactly
// 'k' times.
public class GFG {
    static final int MAX = 1000;

    // finds the smallest number in arr[]
    // that is repeated k times
    static int findDuplicate(int arr[], int n, int k)
    {
        // Since arr[] has numbers in range from
        // 1 to MAX
        int res = MAX + 1;
        for (int i = 0; i < n; i++) {
            if (arr[i] > 0) {

                // set count to 1 as number is
                // present once
                int count = 1;
                for (int j = i + 1; j < n; j++)
                    if (arr[i] == arr[j])
                        count += 1;

                // If frequency of number is equal
                // to 'k'
                if (count == k)
                    res = Math.min(res, arr[i]);
            }
        }
        return res;
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 2, 2, 1, 3, 1 };
        int k = 2;
        int n = arr.length;
        System.out.println(findDuplicate(arr, n, k));
    }
}
// This article is contributed by Sumit Ghosh
```

Python3

```
# Python 3 program to find smallest
# number in array that is repeated
# exactly 'k' times.
```

```
MAX = 1000

def findDuplicate(arr, n, k):

    # Since arr[] has numbers in
    # range from 1 to MAX
    res = MAX + 1

    for i in range(0, n):
        if (arr[i] > 0):

            # set count to 1 as number
            # is present once
            count = 1
            for j in range(i + 1, n):
                if (arr[i] == arr[j]):
                    count += 1

            # If frequency of number is equal to 'k'
            if (count == k):
                res = min(res, arr[i])

    return res

# Driver code
arr = [2, 2, 1, 3, 1]
k = 2
n = len(arr)
print(findDuplicate(arr, n, k))

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
// C# program to find smallest number
// in array that is repeated exactly
// 'k' times.
using System;

public class GFG {

    static int MAX = 1000;

    // finds the smallest number in arr[]
    // that is repeated k times
    static int findDuplicate(int[] arr,
                            int n, int k)
{
```

```
// Since arr[] has numbers in range
// from 1 to MAX
int res = MAX + 1;

for (int i = 0; i < n; i++)
{
    if (arr[i] > 0)
    {

        // set count to 1 as number
        // is present once
        int count = 1;
        for (int j = i + 1; j < n; j++)
            if (arr[i] == arr[j])
                count += 1;

        // If frequency of number is
        // equal to 'k'
        if (count == k)
            res = Math.Min(res, arr[i]);
    }
}

return res;
}

// Driver code
public static void Main()
{
    int[] arr = { 2, 2, 1, 3, 1 };
    int k = 2;
    int n = arr.Length;

    Console.WriteLine(
        findDuplicate(arr, n, k));
}
}

// This article is contributed by vt_m.
```

Output:

Time Complexity : $O(n^2)$

Auxiliary Space : $O(1)$

This solution doesn't require array elements to be in limited range.

Better Solution : Sort the input array and find the first element with exactly k count of appearances.

C++

```
// C++ program to find smallest number
// in array that is repeated exactly
// 'k' times.
#include <bits/stdc++.h>
using namespace std;

int findDuplicate(int arr[], int n, int k)
{
    // Sort the array
    sort(arr, arr + n);

    // Find the first element with exactly
    // k occurrences.
    int i = 0;
    while (i < n) {
        int j, count = 1;
        for (j = i + 1; j < n && arr[j] == arr[i]; j++)
            count++;

        if (count == k)
            return arr[i];

        i = j;
    }

    return -1;
}

// Driver code
int main()
{
    int arr[] = { 2, 2, 1, 3, 1 };
    int k = 2;
    int n = sizeof(arr) / (sizeof(arr[0]));
    cout << findDuplicate(arr, n, k);
    return 0;
}
```

Java

```
// Java program to find smallest number
```

```
// in array that is repeated exactly
// 'k' times.
import java.util.Arrays;
public class GFG {

    // finds the smallest number in arr[]
    // that is repeated k times
    static int findDuplicate(int arr[], int n, int k)
    {
        // Sort the array
        Arrays.sort(arr);

        // Find the first element with exactly
        // k occurrences.
        int i = 0;
        while (i < n) {
            int j, count = 1;
            for (j = i + 1; j < n && arr[j] == arr[i]; j++)
                count++;

            if (count == k)
                return arr[i];

            i = j;
        }

        return -1;
    }

    // Driver code
    public static void main(String[] args)
    {
        int arr[] = { 2, 2, 1, 3, 1 };
        int k = 2;
        int n = arr.length;
        System.out.println(findDuplicate(arr, n, k));
    }
}
// This article is contributed by Sumit Ghosh
```

Python

```
# Python program to find smallest number
# in array that is repeated exactly
# 'k' times.

def findDuplicate(arr, n, k):
```

```
# Sort the array
arr.sort()

# Find the first element with exactly
# k occurrences.
i = 0
while (i < n):
    j, count = i + 1, 1
    while (j < n and arr[j] == arr[i]):
        count += 1
        j += 1

    if (count == k):
        return arr[i]

    i = j

return -1

# Driver code
arr = [ 2, 2, 1, 3, 1 ];
k = 2
n = len(arr)
print findDuplicate(arr, n, k)

# This code is contributed by Sachin Bisht
```

C#

```
// C# program to find smallest number
// in array that is repeated exactly
// 'k' times.
using System;

public class GFG {

    // finds the smallest number in arr[]
    // that is repeated k times
    static int findDuplicate(int[] arr,
                            int n, int k)
    {

        // Sort the array
        Array.Sort(arr);

        // Find the first element with
        // exactly k occurrences.
        int i = 0;
```

```
while (i < n) {
    int j, count = 1;
    for (j = i + 1; j < n &&
         arr[j] == arr[i]; j++)
        count++;
    if (count == k)
        return arr[i];
    i = j;
}
return -1;
}

// Driver code
public static void Main()
{
    int[] arr = { 2, 2, 1, 3, 1 };
    int k = 2;
    int n = arr.Length;

    Console.WriteLine(
        findDuplicate(arr, n, k));
}
}

// This article is contributed by vt_m.
```

Output:

1

Time Complexity : $O(n \log n)$
Auxiliary Space : $O(1)$

Efficient Approach : Efficient approach is based on the fact that array has numbers in small range (1 to 1000). We solve this problem by using a frequency array of size max and store the frequency of every number in that array.

C++

```
// C++ program to find smallest number
// in array that is repeated exactly
// 'k' times.
#include <bits/stdc++.h>
using namespace std;
```

```
const int MAX = 1000;

int findDuplicate(int arr[], int n, int k)
{
    // Computing frequencies of all elements
    int freq[MAX];
    memset(freq, 0, sizeof(freq));
    for (int i = 0; i < n; i++) {
        if (arr[i] < 1 && arr[i] > MAX) {
            cout << "Out of range";
            return -1;
        }
        freq[arr[i]] += 1;
    }

    // Finding the smallest element with
    // frequency as k
    for (int i = 0; i < MAX; i++) {

        // If frequency of any of the number
        // is equal to k starting from 0
        // then return the number
        if (freq[i] == k)
            return i;
    }

    return -1;
}

// Driver code
int main()
{
    int arr[] = { 2, 2, 1, 3, 1 };
    int k = 2;
    int n = sizeof(arr) / (sizeof(arr[0]));
    cout << findDuplicate(arr, n, k);
    return 0;
}
```

Java

```
// Java program to find smallest number
// in array that is repeated exactly
// 'k' times.
public class GFG {

    static final int MAX = 1000;
```

```
// finds the smallest number in arr[]
// that is repeated k times
static int findDuplicate(int arr[], int n, int k)
{
    // Computing frequencies of all elements
    int[] freq = new int[MAX];

    for (int i = 0; i < n; i++) {
        if (arr[i] < 1 && arr[i] > MAX) {
            System.out.println("Out of range");
            return -1;
        }
        freq[arr[i]] += 1;
    }

    // Finding the smallest element with
    // frequency as k
    for (int i = 0; i < MAX; i++) {

        // If frequency of any of the number
        // is equal to k starting from 0
        // then return the number
        if (freq[i] == k)
            return i;
    }

    return -1;
}

// Driver code
public static void main(String[] args)
{
    int arr[] = { 2, 2, 1, 3, 1 };
    int k = 2;
    int n = arr.length;
    System.out.println(findDuplicate(arr, n, k));
}
}

// This article is contributed by Sumit Ghosh
```

Python

```
# Python program to find smallest number
# in array that is repeated exactly
# 'k' times.

MAX = 1000
```

```
def findDuplicate(arr, n, k):

    # Computing frequencies of all elements
    freq = [0 for i in range(MAX)]

    for i in range(n):
        if (arr[i] < 1 and arr[i] > MAX):
            print "Out of range"
            return -1
        freq[arr[i]] += 1

    # Finding the smallest element with
    # frequency as k
    for i in range(MAX):

        # If frequency of any of the number
        # is equal to k starting from 0
        # then return the number
        if (freq[i] == k):
            return i

    return -1

# Driver code
arr = [ 2, 2, 1, 3, 1 ]
k = 2
n = len(arr)
print findDuplicate(arr, n, k)

# This code is contributed by Sachin Bisht
```

C#

```
// C# program to find smallest number
// in array that is repeated exactly
// 'k' times.
using System;

public class GFG {

    static int MAX = 1000;

    // finds the smallest number in arr[]
    // that is repeated k times
    static int findDuplicate(int[] arr,
                            int n, int k)
{
```

```
// Computing frequencies of all
// elements
int[] freq = new int[MAX];

for (int i = 0; i < n; i++)
{
    if (arr[i] < 1 && arr[i] > MAX)
    {
        Console.WriteLine("Out of range");
        return -1;
    }

    freq[arr[i]] += 1;
}

// Finding the smallest element with
// frequency as k
for (int i = 0; i < MAX; i++) {

    // If frequency of any of the
    // number is equal to k starting
    // from 0 then return the number
    if (freq[i] == k)
        return i;
}

return -1;
}

// Driver code
public static void Main()
{
    int[] arr = { 2, 2, 1, 3, 1 };
    int k = 2;
    int n = arr.Length;

    Console.WriteLine(
        findDuplicate(arr, n, k));
}
}

// This article is contributed by vt_m.
```

Output:

Time Complexity: $O(\text{MAX} + n)$

Auxiliary Space : $O(\text{MAX})$

Can we solve it in $O(n)$ time if range is not limited?

Please see [Smallest element repeated exactly 'k' times \(not limited to small range\)](#)

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/smallest-element-array-repeated-exactly-k-times/>

Chapter 303

Smallest greater elements in whole array

Smallest greater elements in whole array - GeeksforGeeks

An array is given of n length, and we need to calculate the next greater element for each element in given array. If next greater element is not available in given array then we need to fill ‘_’ at that index place.

Examples :

Input : 6 3 9 8 10 2 1 15 7
Output : 7 6 10 9 15 3 2 _ 8
Here every element of array has next greater element but at index 7,
15 is the greatest element of given array
and no other element is greater from 15
so at the index of 15 we fill with '_' .

Input : 13 6 7 12
Output : _ 7 12 13
Here, at index 0, 13 is the greatest value in given array and no other array element is greater from 13 so at index 0 we fill '_' .

Asked in : Zoho

A **simple solution** is to use two loops nested. The outer loop picks all elements one by one and inner loop finds the next greater element by linearly searching from beginning to end.
C++

```
// Simple CPP program to find smallest
// greater element in whole array for
// every element.
#include <bits/stdc++.h>
using namespace std;

void smallestGreater(int arr[], int n)
{
    for (int i = 0; i < n; i++) {

        // Find the closest greater element
        // for arr[j] in the entire array.
        int diff = INT_MAX, closest = -1;
        for (int j = 0; j < n; j++) {
            if (arr[i] < arr[j] &&
                arr[j] - arr[i] < diff)
            {
                diff = arr[j] - arr[i];
                closest = j;
            }
        }

        // Check if arr[i] is largest
        (closest == -1)? cout << "_" :
            cout << arr[closest] << " ";
    }
}

// Driver code
int main()
{
    int ar[] = { 6, 3, 9, 8, 10, 2, 1, 15, 7 };
    int n = sizeof(ar) / sizeof(ar[0]);
    smallestGreater(ar, n);
    return 0;
}
```

Java

```
// Simple Java program to find
// smallest greater element in
// whole array for every element.
import java.io.*;

class GFG
{
static void smallestGreater(int arr[],
                           int n)
```

```
{  
    for (int i = 0; i < n; i++)  
    {  
  
        // Find the closest greater  
        // element for arr[j] in  
        // the entire array.  
        int diff = Integer.MAX_VALUE;  
        int closest = -1;  
        for (int j = 0; j < n; j++)  
        {  
            if (arr[i] < arr[j] &&  
                arr[j] - arr[i] < diff)  
            {  
                diff = arr[j] - arr[i];  
                closest = j;  
            }  
        }  
  
        // Check if arr[i] is largest  
        if(closest == -1)  
            System.out.print( "_ " );  
        else  
            System.out.print(arr[closest] + " ");  
    }  
}  
  
// Driver code  
public static void main (String[] args)  
{  
    int ar[] = {6, 3, 9, 8, 10,  
               2, 1, 15, 7};  
    int n = ar.length;  
    smallestGreater(ar, n);  
}  
}  
  
// This code is contributed by anuj_67.
```

Python3

```
# Simple Python program to find smallest  
# greater element in whole array for  
# every element.  
def smallestGreater(arr, n) :  
    for i in range(0, n) :  
  
        # Find the closest greater element
```

```
# for arr[j] in the entire array.
diff = 1000;
closest = -1;
for j in range(0, n) :
    if ( arr[i] < arr[j] and
        arr[j] - arr[i] < diff) :
        diff = arr[j] - arr[i];
        closest = j;

# Check if arr[i] is largest
if (closest == -1) :
    print ("_ ", end = "");
else :
    print ("{} ".format(arr[closest]),
          end = "");

# Driver code
ar = [6, 3, 9, 8, 10, 2, 1, 15, 7];
n = len(ar) ;
smallestGreater(ar, n);

# This code is contributed by Manish Shaw
# (manishshaw1)
```

C#

```
// Simple C# program to find
// smallest greater element in
// whole array for every element.
using System;

class GFG
{
static void smallestGreater(int []arr,
                           int n)
{
    for (int i = 0; i < n; i++)
    {

        // Find the closest greater
        // element for arr[j] in
        // the entire array.
        int diff = int.MaxValue;
        int closest = -1;
        for (int j = 0; j < n; j++)
        {
            if (arr[i] < arr[j] &&
                arr[j] - arr[i] < diff)
```

```
{  
    diff = arr[j] - arr[i];  
    closest = j;  
}  
}  
  
// Check if arr[i] is largest  
if(closest == -1)  
    Console.Write( "_ " );  
else  
    Console.Write(arr[closest] + " ");  
}  
}  
  
// Driver code  
public static void Main()  
{  
    int []ar = {6, 3, 9, 8, 10,  
               2, 1, 15, 7};  
    int n = ar.Length;  
    smallestGreater(ar, n);  
}  
}  
  
// This code is contributed by anuj_67.
```

PHP

```
<?php  
// Simple PHP program to find smallest  
// greater element in whole array for  
// every element.  
  
function smallestGreater($arr, $n)  
{  
    for ( $i = 0; $i < $n; $i++) {  
  
        // Find the closest greater element  
        // for arr[j] in the entire array.  
        $diff = PHP_INT_MAX; $closest = -1;  
        for ( $j = 0; $j < $n; $j++) {  
            if ( $arr[$i] < $arr[$j] &&  
                $arr[$j] - $arr[$i] < $diff)  
            {  
                $diff = $arr[$j] - $arr[$i];  
                $closest = $j;  
            }  
        }  
    }  
}
```

```
// Check if arr[i] is largest
if ($closest == -1)
    echo "_ ";
else
    echo $arr[$closest] , " ";
}

// Driver code
$ar = array (6, 3, 9, 8, 10, 2, 1, 15, 7);
$n = sizeof($ar) ;
smallestGreater($ar, $n);

// This code is contributed by ajit
?>
```

Output:

```
7 6 10 9 15 3 2 _ 8
```

Time Complexity : O(n^*n)
Auxiliary Space: O(1)

An **efficient solution** is to one by one insert elements in a set (A self balancing binary search tree). After inserting into set, we search elements. After we find iterator of the searched element, we move iterator to next (note that set stores elements in sorted order) to find element which is just greater.

C++

```
// Efficient CPP program to find smallest
// greater element in whole array for
// every element.
#include <bits/stdc++.h>
using namespace std;

void smallestGreater(int arr[], int n)
{
    set<int> s;
    for (int i = 0; i < n; i++)
        s.insert(arr[i]);

    for (int i = 0; i < n; i++)
    {
        auto it = s.find(arr[i]);
        it++;
    }
}
```

```
if (it != s.end())
    cout << *it << " ";
else
    cout << "_ ";
}

// Driver code
int main()
{
    int ar[] = { 6, 3, 9, 8, 10, 2, 1, 15, 7 };
    int n = sizeof(ar) / sizeof(ar[0]);
    smallestGreater(ar, n);
    return 0;
}
```

Output :

```
7 6 10 9 15 3 2 _ 8
```

Time Complexity : $O(n \log n)$. Note that self balancing search tree (implemented by set in C++) insert operations take $O(\log n)$ time to insert and find.

Auxiliary Space: $O(n)$

We can also use [sorting](#) followed by [binary searches](#) to solve the above problem in same time and same auxiliary space.

Improved By : [jit_t](#), [vt_m](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/smallest-greater-elements-in-whole-array/>

Chapter 304

Smallest subset with sum greater than all other elements

Smallest subset with sum greater than all other elements - GeeksforGeeks

Given an array of non-negative integers. Our task is to find minimum number of elements such that their sum should be greater than the sum of rest of the elements of the array.

Examples :

```
Input : arr[] = {3, 1, 7, 1}
Output : 1
Smallest subset is {7}. Sum of
this subset is greater than all
other elements {3, 1, 1}
```

```
Input : arr[] = {2, 1, 2}
Output : 2
In this example one element is not
enough. We can pick elements with
values 1, 2 or 2, 2. In any case,
the minimum count is 2.
```

The **Brute force** approach is to find sum of all the possible subsets and then compare sum with sum of remaining elements.

The **Efficient Approach** is to take the largest elements. We sort values in descending order, then take elements from the largest, until we get strictly more than half of total sum of the given array.

CPP

```
// CPP program to find minimum number of
```

```
// elements such that their sum is greater
// than sum of remaining elements of the array.
#include <bits/stdc++.h>
#include <string.h>
using namespace std;

// function to find minimum elements needed.
int minElements(int arr[], int n)
{
    // calculating HALF of array sum
    int halfSum = 0;
    for (int i = 0; i < n; i++)
        halfSum = halfSum + arr[i];
    halfSum = halfSum / 2;

    // sort the array in descending order.
    sort(arr, arr + n, greater<int>());

    int res = 0, curr_sum = 0;
    for (int i = 0; i < n; i++) {

        curr_sum += arr[i];
        res++;

        // current sum greater than sum
        if (curr_sum > halfSum)
            return res;
    }
    return res;
}

// Driver function
int main()
{
    int arr[] = {3, 1, 7, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << minElements(arr, n) << endl;
    return 0;
}
```

Java

```
// Java code to find minimum number of elements
// such that their sum is greater than sum of
// remaining elements of the array.
import java.io.*;
import java.util.*;
```

```
class GFG {

    // Function to find minimum elements needed
    static int minElements(int arr[], int n)
    {
        // Calculating HALF of array sum
        int halfSum = 0;
        for (int i = 0; i < n; i++)
            halfSum = halfSum + arr[i];
        halfSum = halfSum / 2;

        // Sort the array in ascending order and
        // start traversing array from the ascending
        // sort in descending order.
        Arrays.sort(arr);

        int res = 0, curr_sum = 0;
        for (int i = n-1; i >= 0; i--) {

            curr_sum += arr[i];
            res++;

            // Current sum greater than sum
            if (curr_sum > halfSum)
                return res;
        }
        return res;
    }

    // Driver Code
    public static void main (String[] args) {
        int arr[] = {3, 1, 7, 1};
        int n = arr.length;
        System.out.println(minElements(arr, n));
    }
}
```

// This code is contributed by Gitanjali

Python3

```
# Pyhton3 code to find minimum number of
# elements such that their sum is greater
# than sum of remaining elements of the array.

# function to find minimum elements needed.
def minElements(arr , n):
```

```
# calculating HALF of array sum
halfSum = 0
for i in range(n):
    halfSum = halfSum + arr[i]

halfSum = int(halfSum / 2)

# sort the array in descending order.
arr.sort(reverse = True)

res = 0
curr_sum = 0
for i in range(n):

    curr_sum += arr[i]
    res += 1

    # current sum greater than sum
    if curr_sum > halfSum:
        return res

return res

# driver code
arr = [3, 1, 7, 1]
n = len(arr)
print(minElements(arr, n))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# code to find minimum number of elements
// such that their sum is greater than sum of
// remaining elements of the array.
using System;

class GFG {

    // Function to find minimum elements needed
    static int minElements(int []arr, int n)
    {

        // Calculating HALF of array sum
        int halfSum = 0;

        for (int i = 0; i < n; i++)

```

```
halfSum = halfSum + arr[i];  
  
halfSum = halfSum / 2;  
  
// Sort the array in ascending order and  
// start traversing array from the ascending  
// sort in descending order.  
Array.Sort(arr);  
  
int res = 0, curr_sum = 0;  
for (int i = n-1; i >= 0; i--) {  
  
    curr_sum += arr[i];  
    res++;  
  
    // Current sum greater than sum  
    if (curr_sum > halfSum)  
        return res;  
}  
  
return res;  
}  
  
// Driver Code  
public static void Main ()  
{  
    int []arr = {3, 1, 7, 1};  
    int n = arr.Length;  
  
    Console.WriteLine(minElements(arr, n));  
}  
}  
  
// This code is contributed by vt_m.
```

Output:

1

Time Complexity : $O(n \log n)$

Source

<https://www.geeksforgeeks.org/smallest-subset-sum-greater-elements/>

Chapter 305

Sort 1 to N by swapping adjacent elements

Sort 1 to N by swapping adjacent elements - GeeksforGeeks

Given an array A of size N consisting of elements 1 to N. A boolean array B consisting of N-1 elements indicates that if B[i] is 1, then A[i] can be swapped with A[i+1]. Find out if A can be sorted by swapping elements.

Examples:

Input : A[] = {1, 2, 5, 3, 4, 6}
B[] = {0, 1, 1, 1, 0}
Output : A can be sorted
We can swap a[3] with a[4] and then a[4] with a[5].

Input : A[] = {2, 3, 1, 4, 5, 6}
B[] = {0, 1, 1, 1, 1}
Output : A can not be sorted
We can not sort A by swapping elements.

Here we can swap only A[i] with A[i+1]. So to find whether array can be sorted or not. Using boolean array B we can sort array for a continuous sequence of 1 for B. At last we can check, if A is sorted or not.

C++

```
// CPP program to test whether array
// can be sorted by swapping adjacent
// elements using boolean array
#include <bits/stdc++.h>
using namespace std;
```

```
// Return true if array can be
// sorted otherwise false
bool sortedAfterSwap(int A[], bool B[], int n)
{
    int i, j;

    // Check bool array B and sorts
    // elements for continuos sequence of 1
    for (i = 0; i < n - 1; i++) {
        if (B[i]) {
            j = i;
            while (B[j])
                j++;

            // Sort array A from i to j
            sort(A + i, A + 1 + j);
            i = j;
        }
    }

    // Check if array is sorted or not
    for (i = 0; i < n; i++) {
        if (A[i] != i + 1)
            return false;
    }

    return true;
}

// Driver program to test sortedAfterSwap()
int main()
{
    int A[] = { 1, 2, 5, 3, 4, 6 };
    bool B[] = { 0, 1, 1, 1, 0 };
    int n = sizeof(A) / sizeof(A[0]);

    if (sortedAfterSwap(A, B, n))
        cout << "A can be sorted\n";
    else
        cout << "A can not be sorted\n";

    return 0;
}
```

Java

```
import java.util.Arrays;
```

```
// Java program to test whether array
// can be sorted by swapping adjacent
// elements using boolean array

class GFG {

    // Return true if array can be
    // sorted otherwise false
    static boolean sortedAfterSwap(int A[],
                                  boolean B[], int n)
    {
        int i, j;

        // Check bool array B and sorts
        // elements for continuos sequnce of 1
        for (i = 0; i < n - 1; i++) {
            if (B[i]) {
                j = i;
                while (B[j]) {
                    j++;
                }
                // Sort array A from i to j
                Arrays.sort(A, i, 1 + j);
                i = j;
            }
        }

        // Check if array is sorted or not
        for (i = 0; i < n; i++) {
            if (A[i] != i + 1) {
                return false;
            }
        }

        return true;
    }

    // Driver program to test sortedAfterSwap()
    public static void main(String[] args)
    {
        int A[] = { 1, 2, 5, 3, 4, 6 };
        boolean B[] = { false, true, true, true, false };
        int n = A.length;

        if (sortedAfterSwap(A, B, n)) {
            System.out.println("A can be sorted");
        }
    }
}
```

```
        else {
            System.out.println("A can not be sorted");
        }
    }
}
```

Python3

```
# Python 3 program to test whether array
# can be sorted by swapping adjacent
# elements using boolean array

# Return true if array can be
# sorted otherwise false
def sortedAfterSwap(A, B, n) :

    # Check bool array B and sorts
    # elements for continuos sequnce of 1
    for i in range(0, n - 1) :
        if (B[i]== 1) :
            j = i
            while (B[j]== 1) :
                j = j + 1

            # Sort array A from i to j
            A = A[0:i] + sorted(A[i:j + 1]) + A[j + 1:]
            i = j

    # Check if array is sorted or not
    for i in range(0, n) :
        if (A[i] != i + 1) :
            return False

    return True

# Driver program to test sortedAfterSwap()
A = [ 1, 2, 5, 3, 4, 6 ]
B = [ 0, 1, 1, 1, 0 ]
n = len(A)

if (sortedAfterSwap(A, B, n)) :
    print("A can be sorted")
else :
    print("A can not be sorted")
```

```
# This code is contributed
# by Nikita Tiwari.
```

C#

```
// C# program to test whether array
// can be sorted by swapping adjacent
// elements using boolean array
using System;
class GFG {

    // Return true if array can be
    // sorted otherwise false
    static bool sortedAfterSwap(int[] A,
                                bool[] B,
                                int n)
    {
        int i, j;

        // Check bool array B and sorts
        // elements for continuos sequnce of 1
        for (i = 0; i < n - 1; i++) {
            if (B[i]) {
                j = i;
                while (B[j]) {
                    j++;
                }
                // Sort array A from i to j
                Array.Sort(A, i, 1 + j);
                i = j;
            }
        }

        // Check if array is sorted or not
        for (i = 0; i < n; i++) {
            if (A[i] != i + 1) {
                return false;
            }
        }

        return true;
    }

    // Driver Code
    public static void Main()
    {
```

```
int[] A = { 1, 2, 5, 3, 4, 6 };
bool[] B = { false, true, true, true, false };
int n = A.Length;

if (sortedAfterSwap(A, B, n)) {
    Console.WriteLine("A can be sorted");
}

else {
    Console.WriteLine("A can not be sorted");
}
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP program to test whether array
// can be sorted by swapping adjacent
// elements using boolean array

// Return true if array can be
// sorted otherwise false
function sortedAfterSwap($A, $B, $n)
{

    // Check bool array B and sorts
    // elements for continuos sequnce of 1
    for ($i = 0; $i < $n - 1; $i++)
    {
        if ($B[$i])
        {
            $j = $i;
            while ($B[$j])
                $j++;

            // Sort array A from i to j
            sort($A);
            $i = $j;
        }
    }

    // Check if array is sorted or not
    for ($i = 0; $i < $n; $i++)
    {
        if ($A[$i] != $i + 1)
```

```
        return false;
    }

    return true;
}

// Driver Code
$A = array(1, 2, 5, 3, 4, 6);
$B = array(0, 1, 1, 1, 0);
$n = count($A);

if (sortedAfterSwap($A, $B, $n))
    echo "A can be sorted\n";
else
    echo "A can not be sorted\n";

// This code is contributed by Sam007
?>
```

Output:

```
A can be sorted
```

Alternative Approach

Here we discuss a very intuitive approach which too gives the answer in $O(n)$ time for all cases. The idea here is that whenever the binary array has 1, we check if that index in array A has $i+1$ or not. If it does not contain $i+1$, we simply swap $a[i]$ with $a[i+1]$.

The reason for this is that the array should have $i+1$ stored at index i . And if at all the array is sortable, then the only operation allowed is swapping. Hence, if the required condition is not satisfied, we simply swap. If the array is sortable, swapping will take us one step closer to the correct answer. And as expected, if the array is not sortable, then swapping would lead to just another unsorted version of the same array.

```
// CPP program to test whether array
// can be sorted by swapping adjacent
// elements using boolean array
#include <bits/stdc++.h>
using namespace std;

// Return true if array can be
// sorted otherwise false
bool sortedAfterSwap(int A[], bool B[], int n)
{
    for (int i = 0; i < n - 1; i++) {
        if (B[i]) {
            if (A[i] != i + 1)
```

```
        swap(A[i], A[i + 1]);
    }
}

// Check if array is sorted or not
for (int i = 0; i < n; i++) {
    if (A[i] != i + 1)
        return false;
}

return true;
}

// Driver program to test sortedAfterSwap()
int main()
{
    int A[] = { 1, 2, 5, 3, 4, 6 };
    bool B[] = { 0, 1, 1, 1, 0 };
    int n = sizeof(A) / sizeof(A[0]);

    if (sortedAfterSwap(A, B, n))
        cout << "A can be sorted\n";
    else
        cout << "A can not be sorted\n";

    return 0;
}
```

Output:

A can be sorted

Improved By : [Sam007, 02DCE](#)

Source

<https://www.geeksforgeeks.org/sort-1-n-swapping-adjacent-elements/>

Chapter 306

Sort 3 Integers without using if condition or using only max() function

Sort 3 Integers without using if condition or using only max() function - GeeksforGeeks

Given three integers, print them in sorted order without using if condition.

Examples :

Input : a = 3, b = 2, c = 9
Output : 2 3 9

Input : a = 4, b = 1, c = 9
Output : 1 4 9

Approach :

1. Find the maximum of a, b, c using max() function.
3. Multiply all integers by -1. Again find Minimum of $-a$, $-b$, $-c$ using max() function.
4. Add the Max and Min from above steps and subtract the sum from $(a+b+c)$. It gives us middle element.

It works for negative numbers also.

C++

```
// C++ program to print three numbers
// in sorted order using max function
#include<bits/stdc++.h>
using namespace std;
```

```
void printSorted(int a, int b, int c)
{
    // Find maximum element
    int get_max = max(a, max(b, c));

    // Find minimum element
    int get_min = -max(-a, max(-b, -c));

    int get_mid = (a + b + c)
                  - (get_max + get_min);

    cout << get_min << " " << get_mid
        << " " << get_max;
}

// Driver code
int main()
{
    int a = 4, b = 1, c = 9;
    printSorted(a, b, c);
    return 0;
}
```

Java

```
// Java program to print three numbers
// in sorted order using max function
class GFG
{

    static void printSorted(int a, int b, int c)
    {
        // Find maximum element
        int get_max = Math.max(a, Math.max(b, c));

        // Find minimum element
        int get_min = -Math.max(-a, Math.max(-b, -c));

        int get_mid = (a + b + c)
                      - (get_max + get_min);

        System.out.print(get_min + " " + get_mid
                        + " " + get_max);
    }

    // Driver code
    public static void main(String[] args)
    {
```

```
int a = 4, b = 1, c = 9;  
  
    printSorted(a, b, c);  
}  
}  
  
// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program to print three numbers  
# in sorted order using max function  
  
def printSorted(a, b, c):  
  
    # Find maximum element  
    get_max = max(a, max(b, c))  
  
    # Find minimum element  
    get_min = -max(-a, max(-b, -c))  
  
    get_mid = (a + b + c) - (get_max + get_min)  
  
    print(get_min, " ", get_mid, " ", get_max)  
  
# Driver Code  
a, b, c = 4, 1, 9  
printSorted(a, b, c)  
  
# This code is contributed by Anant Agarwal.
```

C#

```
// C# program to print three numbers  
// in sorted order using max function  
using System;  
  
class GFG {  
  
    static void printSorted(int a, int b, int c)  
    {  
        // Find maximum element  
        int get_max = Math.Max(a, Math.Max(b, c));  
  
        // Find minimum element  
        int get_min = -Math.Max(-a, Math.Max(-b, -c));  
    }  
}
```

```
int get_mid = (a + b + c) -  
              (get_max + get_min);  
  
Console.WriteLine(get_min + " " + get_mid  
                  + " " + get_max);  
}  
  
// Driver code  
public static void Main()  
{  
    int a = 4, b = 1, c = 9;  
  
    printSorted(a, b, c);  
}  
}  
  
// This code is contributed by nitin mittal.
```

PHP

```
<?php  
// PHP program to print three numbers  
// in sorted order using max function  
  
function printSorted($a, $b, $c)  
{  
  
    // Find maximum element  
    $get_max = max($a, max($b, $c));  
  
    // Find minimum element  
    $get_min = -max(-$a, max(-$b, -$c));  
  
    $get_mid = ($a + $b + $c) -  
              ($get_max + $get_min);  
  
    echo $get_min , " " , $get_mid, " " , $get_max;  
}  
  
// Driver Code  
$a = 4;  
$b = 1;  
$c = 9;  
printSorted($a, $b,$c);  
  
// This code is contributed by nitin mittal.  
?>
```

Chapter 306. Sort 3 Integers without using if condition or using only max() function

Output:

1 4 9

Improved By : nitin mittal

Source

<https://www.geeksforgeeks.org/sort-3-integers-without-using-condition-using-max-function/>

Chapter 307

Sort 3 numbers

Sort 3 numbers - GeeksforGeeks

Given three numbers, how to sort them?

Examples:

Input : arr[] = {3, 2, 1}
Output : arr[] = {1, 2, 3}

Input : arr[] = {6, 5, 0}
Output :arr[] = {0, 5, 6}

One simple solution is to use **sort** function.

C++

```
// C++ program to sort an array of size 3
#include <algorithm>
#include <iostream>
using namespace std;

int main()
{
    int a[] = {10, 12, 5};

    sort(a, a + 3);

    for (int i = 0; i < 3; i++)
        cout << a[i] << " ";

    return 0;
}
```

Java

```
// Java program to sort
// an array of size 3
import java.io.*;
import java.util.*;

class GFG
{
    public static void main (String[] args)
    {
        int a[] = {10, 12, 5};

        Arrays.sort(a);

        for (int i = 0; i < 3; i++)
            System.out.print( a[i] + " ");
    }
}

// This code is contributed
// by inder_verma.
```

C#

```
// C# program to sort
// an array of size 3
using System;
class GFG
{
    public static void Main ()
    {
        int []a = {10, 12, 5};

        Array.Sort(a);

        for (int i = 0; i < 3; i++)
            Console.Write( a[i] + " ");
    }
}

// This code is contributed
// by chandan_jnu.
```

PHP

```
<?php
```

```
// PHP program to sort
// an array of size 3
$a = array(10, 12, 5);

sort($a);

for ($i = 0; $i < 3; $i++)
    echo $a[$i] , " ";

// This code is contributed
// by chandan_jnu.
?>
```

Output:

```
5 10 12
```

How to write our own sort function that does minimum comparison and does not use extra variables?

The idea is to use [insertion sort](#) as insertion sort works best for small arrays.
C++

```
// C++ program to sort an array of size 3
#include <algorithm>
#include <iostream>
using namespace std;

int sort3(int arr[])
{
    // Insert arr[1]
    if (arr[1] < arr[0])
        swap(arr[0], arr[1]);

    // Insert arr[2]
    if (arr[2] < arr[1])
    {
        swap(arr[1], arr[2]);
        if (arr[1] < arr[0])
            swap(arr[1], arr[0]);
    }
}

int main()
{
    int a[] = {10, 12, 5};
```

```
sort3(a);

for (int i = 0; i < 3; i++)
    cout << a[i] << " ";

return 0;
}
```

Java

```
// Java program to sort
// an array of size 3
import java.io.*;
import java.util.*;

class GFG
{
static void sort3(int arr[],
                  int temp[])
{
    // Insert arr[1]
    if (arr[1] < arr[0])
    {
        temp[0] = arr[0];
        arr[0] = arr[1];
        arr[1] = temp[0];
    }

    // Insert arr[2]
    if (arr[2] < arr[1])
    {
        temp[0] = arr[1];
        arr[1] = arr[2];
        arr[2] = temp[0];

        if (arr[1] < arr[0])
        {
            temp[0] = arr[0];
            arr[0] = arr[1];
            arr[1] = temp[0];
        }
    }
}

// Driver Code
public static void main(String args[])
{
```

```
int a[] = new int[]{10, 12, 5};
int temp1[] = new int[10];
sort3(a, temp1);

for (int i = 0; i < 3; i++)
    System.out.print( a[i] + " ");
}

// This code is contributed
// by Akanksha Rai(Abby_akku)
```

C#

```
// C# program to sort
// an array of size 3
using System;

class GFG {

static void sort3(int []arr, int []temp)
{

    // Insert arr[1]
    if (arr[1] < arr[0])
    {
        temp[0] = arr[0];
        arr[0] = arr[1];
        arr[1] = temp[0];
    }

    // Insert arr[2]
    if (arr[2] < arr[1])
    {
        temp[0] = arr[1];
        arr[1] = arr[2];
        arr[2] = temp[0];

        if (arr[1] < arr[0])
        {
            temp[0] = arr[0];
            arr[0] = arr[1];
            arr[1] = temp[0];
        }
    }
}

// Driver Code
```

```
public static void Main(String []args)
{
    int []a= new int[]{10, 12, 5};
    int []temp1 = new int[10];
    sort3(a, temp1);

    for (int i = 0; i < 3; i++)
        Console.Write( a[i] + " ");
}
}

// This code is contributed
// by Akanksha Rai( Abby_akku)
```

Output:

5 10 12

Improved By : [inderDuMCA](#), [Chandan_Kumar](#), [Abby_akku](#)

Source

<https://www.geeksforgeeks.org/sort-3-numbers/>

Chapter 308

Sort a Matrix in all way increasing order

Sort a Matrix in all way increasing order - GeeksforGeeks

Given a square matrix of order $N \times N$ having distinct elements, the task is to sort given matrix in such a way that its rows, columns and both diagonals (diagonal and anti-diagonal) are in increasing order.

Examples:

```
Input : arr[3][3] = {1, 4, 2,
                     3, 5, 6,
                     9, 7, 8}
Output :{1, 2, 3,
        4, 5, 6,
        7, 8, 9}
```

```
Input : arr[2][2] = {0, 4,
                     5, 2}
Output :{0, 2,
        4, 5}
```

Sorting any matrix in a way that its rows, columns and main diagonal are in increasing order is easy. If we consider matrix elements in sequence according to row-major order and sort the sequence, we get the desired result.

```
Example: arr[2][2] : {1, 2
                      3, 4}
Rows in increasing order: {1,2} and {3,4}
```

```
Columns in increasing order: {1,3} and {2,4}
Diagonal in increasing order: {1,4}
Anti-diagonal in increasing order: {2,3}
```

```
// C++ program to sort matrix in all-way
#include<bits/stdc++.h>
using namespace std;
#define N 3

// Sorts a matrix in increasing order
void sortAllWay(int arr[][])
{
    // Consider matrix elements (in row major
    // order) and sort the sequence.
    int *ptr = (int *)arr;
    sort(ptr, ptr+N*N);
}

// driver program
int main()
{
    int arr[N][N] = {1, 0, 3,
                     2, 5, 6,
                     9, 4, 8};
    sortAllWay(arr);

    // print resultant matrix
    for (int i=0; i<N; i++)
    {
        for (int j=0; j<N; j++)
            cout << arr[i][j] << " ";
        cout <<"\n";
    }

    return 0;
}
```

Output:

```
0 1 2
3 4 5
6 8 9
```

Time Complexity : O(N*N log N)
Auxiliary Space : (N*N)

Source

<https://www.geeksforgeeks.org/sort-matrix-way-increasing-order/>

Chapter 309

Sort a Rotated Sorted Array

Sort a Rotated Sorted Array - GeeksforGeeks

You are given a [rotated sorted array](#) and your aim is to restore its original sort in place.

Expected to use $O(1)$ extra space and $O(n)$ time complexity.

Examples:

Input : [3, 4, 1, 2]
Output : [1, 2, 3, 4]

Input : [2, 3, 4, 1]
Output : [1, 2, 3, 4]

We find the point of rotation. Then we [rotate array using reversal algorithm](#).

1. First find the split point where the sorting breaks.
2. Then call the reverse function in three steps.
 - From zero index to split index.
 - From split index to end index.
 - From zero index to end index.

```
// C++ implementation for restoring original
// sort in rotated sorted array
#include <bits/stdc++.h>
using namespace std;

// Function to restore the Original Sort
void restoreSortedArray(int arr[], int n)
{
    for (int i = 0; i < n; i++) {
```

```
if (arr[i] > arr[i + 1]) {  
  
    // In reverse(), the first parameter  
    // is iterator to beginning element  
    // and second parameter is iterator  
    // to last element plus one.  
    reverse(arr, arr+i+1);  
    reverse(arr + i + 1, arr + n);  
    reverse(arr, arr + n);  
}  
}  
}  
  
// Function to print the Array  
void printArray(int arr[], int size)  
{  
    for (int i = 0; i < size; i++)  
        cout << arr[i] << " ";  
}  
  
// Driver function  
int main()  
{  
    int arr[] = { 3, 4, 5, 1, 2 };  
    int n = sizeof(arr) / sizeof(arr[0]);  
    restoreSortedArray(arr, n);  
    printArray(arr, n);  
    return 0;  
}
```

Output:

1 2 3 4 5

We can [binary search](#) to find the rotation point as discussed [here](#).

Efficient code approach using binary search:

1. First find the index of minimum element (split index) in the array using binary search
2. Then call the reverse function in three steps.
 - From zero index to split index.
 - From split index to end index.
 - From zero index to end index.

```
// C++ implementation for restoring original
// sort in rotated sorted array using binary search
#include <bits/stdc++.h>

using namespace std;

// Function to find start index of array
int findStartIndexOfArray(int arr[], int low, int high)
{
    if (low > high)
    {
        return -1;
    }

    if (low == high)
    {
        return low;
    }

    int mid = low + (high - low) / 2;
    if (arr[mid] > arr[mid + 1])
        return mid + 1;

    if (arr[mid - 1] > arr[mid])
        return mid;

    if (arr[low] > arr[mid])
        return findStartIndexOfArray(arr, low, mid - 1);
    else
        return findStartIndexOfArray(arr, mid + 1, high);
}

// Function to restore the Original Sort
void restoreSortedArray(int arr[], int n)
{
    // array is already sorted
    if (arr[0] < arr[n - 1])
        return;

    int start = findStartIndexOfArray(arr, 0, n - 1);
    // In reverse(), the first parameter
    // is iterator to beginning element
    // and second parameter is iterator
    // to last element plus one.
    reverse(arr, arr + start);
    reverse(arr + start, arr + n);
    reverse(arr, arr + n);
}
```

```
}

// Function to print the Array
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
}

// Driver function
int main()
{
    int arr[] = { 1, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    restoreSortedArray(arr, n);
    printArray(arr, n);
    return 0;
}
```

Output:

1 2 3 4 5

Improved By : Neha Jain

Source

<https://www.geeksforgeeks.org/sort-rotated-sorted-array/>

Chapter 310

Sort a binary array using one traversal

Sort a binary array using one traversal - GeeksforGeeks

Given a binary array, sort it using one traversal and no extra space.

Examples :

Input : 1 0 0 1 0 1 0 1 1 1 1 1 0 0 1 1 0 1 0 0
Output : 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1

Input : 1 0 1 0 1 0 1 0
Output : 0 0 0 0 1 1 1 1

This concept is related to [partition of quick sort](#). In quick sort' partition, after one scan, the left of array in smallest and right of array is largest of selected pivot element. So this concept related to quick sort.but it works in $O(n)$ time only.

let's understand this concept with this code

CPP

```
// CPP program to sort a binary array
#include <iostream>
using namespace std;

void sortBinaryArray(int a[], int n)
{
    int j = -1;
    for (int i = 0; i < n; i++) {

        // if number is smaller than 1
```

```
// then swap it with j-th number
if (a[i] < 1) {
    j++;
    swap(a[i], a[j]);
}
}

// Driver code
int main()
{
    int a[] = { 1, 0, 0, 1, 0, 1, 0, 1, 1,
               1, 1, 0, 0, 1, 1, 0, 1, 0, 0 };
    int n = sizeof(a) / sizeof(a[0]);
    sortBinaryArray(a, n);
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";

    return 0;
}
```

Java

```
// JAVA Code for Sort a binary
// array using one traversal
import java.util.*;

class GFG {

    static void sortBinaryArray(int a[], int n)
    {
        int j = -1;
        for (int i = 0; i < n; i++) {

            // if number is smaller than 1
            // then swap it with j-th number
            if (a[i] < 1) {
                j++;
                int temp = a[j];
                a[j] = a[i];
                a[i] = temp;
            }
        }
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
```

```
int a[] = { 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
            1, 1, 0, 0, 1, 1, 0, 1, 0, 0 };

int n = a.length;

sortBinaryArray(a, n);

for (int i = 0; i < n; i++)
    System.out.print(a[i] + " ");
}
```

Python3

```
# A Python program to sort a
# binary array
def sortBinaryArray(a, n):
    j = -1
    for i in range(n):

        # if number is smaller
        # than 1 then swap it
        # with j-th number
        if a[i] < 1:
            j = j + 1

        # swap
        a[i], a[j] = a[j], a[i]

# Driver program
a = [1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
      1, 1, 0, 0, 1, 1, 0, 1, 0, 0]
n = len(a)

sortBinaryArray(a, n)

for i in range(n):
    print(a[i], end = " ")

# This code is contributed by Shrikant13.
```

C#

```
// C# Code for Sort a binary
// array using one traversal
```

```
using System;

class GFG {

    static void sortBinaryArray(int[] a,
                                int n)
    {
        int j = -1;
        for (int i = 0; i < n; i++)
        {

            // if number is smaller than
            // 1 then swap it with j-th
            // number
            if (a[i] < 1) {
                j++;
                int temp = a[j];
                a[j] = a[i];
                a[i] = temp;
            }
        }
    }

    /* Driver program to test above
    function */
    public static void Main()
    {

        int[] a = { 1, 0, 0, 1, 0, 1, 0,
                   1, 1, 1, 1, 1, 0,
                   0, 1, 1, 0, 1, 0, 0 };

        int n = a.Length;

        sortBinaryArray(a, n);

        for (int i = 0; i < n; i++)
            Console.Write(a[i] + " ");
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP Code for Sort a binary
// array using one traversal
```

```
function sortBinaryArray($a, $n)
{
    $j = -1;
    for ($i = 0; $i < $n; $i++)
    {
        // if number is smaller than
        // 1 then swap it with j-th
        // number
        if ($a[$i] < 1)
        {
            $j++;
            $temp = $a[$j];
            $a[$j] = $a[$i];
            $a[$i] = $temp;
        }
    }
    for ($i = 0; $i < $n; $i++)
        echo $a[$i] . " ";
}

// Driver Code
$a = array(1, 0, 0, 1, 0, 1, 0,
           1, 1, 1, 1, 1, 0,
           0, 1, 1, 0, 1, 0, 0);

$n = count($a);
sortBinaryArray($a, $n);

// This code is contributed by Sam007
?>
```

Output :

```
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
```

Improved By : [vt_m](#), [shrikanth13](#), [Sam007](#)

Source

<https://www.geeksforgeeks.org/sort-binary-array-using-one-traversal/>

Chapter 311

Sort a linked list of 0s, 1s and 2s

Sort a linked list of 0s, 1s and 2s - GeeksforGeeks

Given a linked list of 0s, 1s and 2s, sort it.

Source: [Microsoft Interview Set 1](#)

Following steps can be used to sort the given linked list.

- 1) Traverse the list and count the number of 0s, 1s and 2s. Let the counts be n1, n2 and n3 respectively.
- 2) Traverse the list again, fill the first n1 nodes with 0, then n2 nodes with 1 and finally n3 nodes with 2.

C/C++

```
// C Program to sort a linked list 0s, 1s or 2s
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

// Function to sort a linked list of 0s, 1s and 2s
void sortList(struct Node *head)
{
    int count[3] = {0, 0, 0}; // Initialize count of '0', '1' and '2' as 0
    struct Node *ptr = head;

    /* count total number of '0', '1' and '2'
     * count[0] will store total number of '0's
     * count[1] will store total number of '1's
     * count[2] will store total number of '2's
    */
    for(ptr = head; ptr != NULL; ptr = ptr->next)
    {
        if(ptr->data == 0)
            count[0]++;
        else if(ptr->data == 1)
            count[1]++;
        else
            count[2]++;
    }

    ptr = head;
    for(i = 0; i < count[0]; i++)
        ptr->data = 0;
    for(i = 0; i < count[1]; i++)
        ptr->data = 1;
    for(i = 0; i < count[2]; i++)
        ptr->data = 2;
}
```

```
* count[2] will store total number of '2's */
while (ptr != NULL)
{
    count[ptr->data] += 1;
    ptr = ptr->next;
}

int i = 0;
ptr = head;

/* Let say count[0] = n1, count[1] = n2 and count[2] = n3
 * now start traversing list from head node,
 * 1) fill the list with 0, till n1 > 0
 * 2) fill the list with 1, till n2 > 0
 * 3) fill the list with 2, till n3 > 0 */
while (ptr != NULL)
{
    if (count[i] == 0)
        ++i;
    else
    {
        ptr->data = i;
        --count[i];
        ptr = ptr->next;
    }
}
}

/* Function to push a node */
void push (struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Function to print linked list */
void printList(struct Node *node)
{
```

```
while (node != NULL)
{
    printf("%d ", node->data);
    node = node->next;
}
printf("\n");

/* Drier program to test above function*/
int main(void)
{
    struct Node *head = NULL;
    push(&head, 0);
    push(&head, 1);
    push(&head, 0);
    push(&head, 2);
    push(&head, 1);
    push(&head, 1);
    push(&head, 2);
    push(&head, 1);
    push(&head, 2);

    printf("Linked List Before Sortingn");
    printList(head);

    sortList(head);

    printf("Linked List After Sortingn");
    printList(head);

    return 0;
}
```

Java

```
// Java program to sort a linked list of 0, 1 and 2
class LinkedList
{
    Node head; // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }
```

```
void sortList()
{
    // initialise count of 0 1 and 2 as 0
    int count[] = {0, 0, 0};

    Node ptr = head;

    /* count total number of '0', '1' and '2'
     * count[0] will store total number of '0's
     * count[1] will store total number of '1's
     * count[2] will store total number of '2's */
    while (ptr != null)
    {
        count[ptr.data]++;
        ptr = ptr.next;
    }

    int i = 0;
    ptr = head;

    /* Let say count[0] = n1, count[1] = n2 and count[2] = n3
     * now start traversing list from head node,
     * 1) fill the list with 0, till n1 > 0
     * 2) fill the list with 1, till n2 > 0
     * 3) fill the list with 2, till n3 > 0 */
    while (ptr != null)
    {
        if (count[i] == 0)
            i++;
        else
        {
            ptr.data= i;
            --count[i];
            ptr = ptr.next;
        }
    }
}

/* Utility functions */

/* Inserts a new Node at front of the list. */
public void push(int new_data)
{
    /* 1 & 2: Allocate the Node &
       Put in the data*/
    Node new_node = new Node(new_data);
```

```
/* 3. Make next of new Node as head */
new_node.next = head;

/* 4. Move the head to point to new Node */
head = new_node;
}

/* Function to print linked list */
void printList()
{
    Node temp = head;
    while (temp != null)
    {
        System.out.print(temp.data+" ");
        temp = temp.next;
    }
    System.out.println();
}

/* Drier program to test above functions */
public static void main(String args[])
{
    LinkedList llist = new LinkedList();

    /* Constructed Linked List is 1->2->3->4->5->6->7->
       8->8->9->null */
    llist.push(0);
    llist.push(1);
    llist.push(0);
    llist.push(2);
    llist.push(1);
    llist.push(1);
    llist.push(2);
    llist.push(1);
    llist.push(2);

    System.out.println("Linked List before sorting");
    llist.printList();

    llist.sortList();

    System.out.println("Linked List after sorting");
    llist.printList();
}
}

/* This code is contributed by Rajat Mishra */
```

Python

```
# Python program to sort a linked list of 0, 1 and 2
class LinkedList(object):
    def __init__(self):

        # head of list
        self.head = None

    # Linked list Node
    class Node(object):
        def __init__(self, d):
            self.data = d
            self.next = None

    def sortList(self):

        # initialise count of 0 1 and 2 as 0
        count = [0, 0, 0]

        ptr = self.head

        # count total number of '0', '1' and '2'
        # * count[0] will store total number of '0's
        # * count[1] will store total number of '1's
        # * count[2] will store total number of '2's
        while ptr != None:
            count[ptr.data]+=1
            ptr = ptr.next

        i = 0
        ptr = self.head

        # Let say count[0] = n1, count[1] = n2 and count[2] = n3
        # * now start traversing list from head node,
        # * 1) fill the list with 0, till n1 > 0
        # * 2) fill the list with 1, till n2 > 0
        # * 3) fill the list with 2, till n3 > 0
        while ptr != None:
            if count[i] == 0:
                i+=1
            else:
                ptr.data = i
                count[i]-=1
                ptr = ptr.next

    # Utility functions
    # Inserts a new Node at front of the list.
    def push(self, new_data):
```

```
# 1 & 2: Allocate the Node &
# Put in the data
new_node = self.Node(new_data)

# 3. Make next of new Node as head
new_node.next = self.head

# 4. Move the head to point to new Node
self.head = new_node

# Function to print linked list
def printList(self):
    temp = self.head
    while temp != None:
        print str(temp.data),
        temp = temp.next
    print ''

# Drier program to test above functions
llist = LinkedList()
llist.push(0)
llist.push(1)
llist.push(0)
llist.push(2)
llist.push(1)
llist.push(1)
llist.push(2)
llist.push(1)
llist.push(2)

print "Linked List before sorting"
llist.printList()

llist.sortList()

print "Linked List after sorting"
llist.printList()

# This code is contributed by BHAVYA JAIN
```

Output:

```
Linked List Before Sorting
2 1 2 1 1 2 0 1 0
Linked List After Sorting
0 0 1 1 1 2 2 2
```

Time Complexity: $O(n)$ where n is number of nodes in linked list.

Auxiliary Space: $O(1)$

Sort a linked list of 0s, 1s and 2s by changing links

This article is compiled by **Narendra Kangalkar**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

<https://www.geeksforgeeks.org/sort-a-linked-list-of-0s-1s-or-2s/>

Chapter 312

Sort a linked list of 0s, 1s and 2s by changing links

Sort a linked list of 0s, 1s and 2s by changing links - GeeksforGeeks

Given a linked list of 0s, 1s and 2s, sort it.

Examples:

Input : 2->1->2->1->1->2->0->1->0
Output : 0->0->1->1->1->1->2->2->2

Input : 2->1->0
Output : 0->1->2

We have discussed a solution in below post that works by changing data of nodes.

[Sort a linked list of 0s, 1s and 2s](#)

The above solution does not work when these values have associated data with them. For example, these three represent three colors and different types of objects associated with the colors and we want to sort objects (connected with a linked list) based on colors.

In this post, a new solution is discussed that works by changing links.

Iterate through the linked list. Maintain 3 pointers named zero, one and two to point to current ending nodes of linked lists containing 0, 1, and 2 respectively. For every traversed node, we attach it to the end of its corresponding list. Finally we link all three lists. To avoid many null checks, we use three dummy pointers zeroD, oneD and twoD that work as dummy headers of three lists.

```
// CPP Program to sort a linked list 0s, 1s
// or 2s by changing links
#include <stdio.h>
```

```
/* Link list node */
struct Node {
    int data;
    struct Node* next;
};

Node* newNode(int data);

// Sort a linked list of 0s, 1s and 2s
// by changing pointers.
Node* sortList(Node* head)
{
    if (!head || !(head->next))
        return head;

    // Create three dummy nodes to point to
    // beginning of three linked lists. These
    // dummy nodes are created to avoid many
    // null checks.
    Node* zeroD = newNode(0);
    Node* oneD = newNode(0);
    Node* twoD = newNode(0);

    // Initialize current pointers for three
    // lists and whole list.
    Node* zero = zeroD, *one = oneD, *two = twoD;

    // Traverse list
    Node* curr = head;
    while (curr) {
        if (curr->data == 0) {
            zero->next = curr;
            zero = zero->next;
            curr = curr->next;
        } else if (curr->data == 1) {
            one->next = curr;
            one = one->next;
            curr = curr->next;
        } else {
            two->next = curr;
            two = two->next;
            curr = curr->next;
        }
    }

    // Attach three lists
    zero->next = (oneD->next) ? (oneD->next) : (twoD->next);
}
```

```
one->next = twoD->next;
two->next = NULL;

// Updated head
head = zeroD->next;

// Delete dummy nodes
delete zeroD;
delete oneD;
delete twoD;

return head;
}

// function to create and return a node
Node* newNode(int data)
{
    // allocating space
    Node* newNode = new Node;

    // inserting the required data
    newNode->data = data;
    newNode->next = NULL;
}

/* Function to print linked list */
void printList(struct Node* node)
{
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}

/* Drier program to test above function*/
int main(void)
{
    // Creating the list 1->2->4->5
    Node* head = newNode(1);
    head->next = newNode(2);
    head->next->next = newNode(0);
    head->next->next->next = newNode(1);

    printf("Linked List Before Sorting\n");
    printList(head);

    head = sortList(head);
```

```
printf("Linked List After Sorting\n");
printList(head);

return 0;
}
```

Output :

```
Linked List Before Sorting
1 2 0 1
Linked List After Sorting
0 1 1 2
```

Thanks to Musarrat_123 for suggesting above solution in a comment [here](#).

Time Complexity: $O(n)$ where n is number of nodes in linked list.

Auxiliary Space: $O(1)$

Source

<https://www.geeksforgeeks.org/sort-linked-list-0s-1s-2s-changing-links/>

Chapter 313

Sort a linked list that is sorted alternating ascending and descending orders?

Sort a linked list that is sorted alternating ascending and descending orders? - Geeks-forGeeks

Given a Linked List. The Linked List is in alternating ascending and descending orders. Sort the list efficiently.

Example:

Input List: 10->40->53->30->67->12->89->NULL
Output List: 10->12->30->43->53->67->89->NULL

A **Simple Solution** is to use [Merge Sort for linked List](#). This solution takes $O(n \log n)$ time.

An **Efficient Solution** works in $O(n)$ time. Below are all steps.

1. Separate two lists.
2. Reverse the one with descending order
3. Merge both lists.

Below are C++ and Java implementations of above algorithm.

C++

```
// C++ program to sort a linked list that is alternatively
// sorted in increasing and decreasing order
#include<bits/stdc++.h>
using namespace std;

// Linked list node
```

Chapter 313. Sort a linked list that is sorted alternating ascending and descending orders?

```
struct Node
{
    int data;
    struct Node *next;
};

Node *mergelist(Node *head1, Node *head2);
void splitList(Node *head, Node **Ahead, Node **Dhead);
void reverselist(Node *&head);

// This is the main function that sorts the
// linked list
void sort(Node **head)
{
    // Split the list into lists
    Node *Ahead, *Dhead;
    splitList(*head, &Ahead, &Dhead);

    // Reverse the descending linked list
    reverselist(Dhead);

    // Merge the two linked lists
    *head = mergelist(Ahead, Dhead);
}

// A utility function to create a new node
Node* newNode(int key)
{
    Node *temp = new Node;
    temp->data = key;
    temp->next = NULL;
    return temp;
}

// A utility function to reverse a linked list
void reverselist(Node *&head)
{
    Node* prev = NULL, *curr = head, *next;
    while (curr)
    {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
}
```

```
// A utility function to print a linked list
void printlist(Node *head)
{
    while (head != NULL)
    {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

// A utility function to merge two sorted linked lists
Node *mergelist(Node *head1, Node *head2)
{
    // Base cases
    if (!head1) return head2;
    if (!head2) return head1;

    Node *temp = NULL;
    if (head1->data < head2->data)
    {
        temp = head1;
        head1->next = mergelist(head1->next, head2);
    }
    else
    {
        temp = head2;
        head2->next = mergelist(head1, head2->next);
    }
    return temp;
}

// This function alternatively splits a linked list with head
// as head into two:
// For example, 10->20->30->15->40->7 is splitted into 10->30->40
// and 20->15->7
// "Ahead" is reference to head of ascending linked list
// "Dhead" is reference to head of descending linked list
void splitList(Node *head, Node **Ahead, Node **Dhead)
{
    // Create two dummy nodes to initialize heads of two linked list
    *Ahead = newNode(0);
    *Dhead = newNode(0);

    Node *ascn = *Ahead;
    Node *dscn = *Dhead;
    Node *curr = head;
```

Chapter 313. Sort a linked list that is sorted alternating ascending and descending orders?

```
// Link alternate nodes
while (curr)
{
    // Link alternate nodes of ascending linked list
    ascn->next = curr;
    ascn = ascn->next;
    curr = curr->next;

    // Link alternate nodes of descending linked list
    if (curr)
    {
        dscn->next = curr;
        dscn = dscn->next;
        curr = curr->next;
    }
}

ascn->next = NULL;
dscn->next = NULL;
*Ahead = (*Ahead)->next;
*Dhead = (*Dhead)->next;
}

// Driver program to test above function
int main()
{
    Node *head = newNode(10);
    head->next = newNode(40);
    head->next->next = newNode(53);
    head->next->next->next = newNode(30);
    head->next->next->next->next = newNode(67);
    head->next->next->next->next->next = newNode(12);
    head->next->next->next->next->next->next = newNode(89);

    cout << "Given Linked List is " << endl;
    printlist(head);

    sort(&head);

    cout << "Sorted Linked List is " << endl;
    printlist(head);

    return 0;
}
```

Java

```
// Java program to sort a linked list that is alternatively
```

```
// sorted in increasing and decreasing order
class LinkedList
{
    Node head; // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) { data = d; next = null; }
    }

    Node newNode(int key)
    {
        return new Node(key);
    }

    /* This is the main function that sorts
       the linked list.*/
    void sort()
    {
        /* Create 2 dummy nodes and initialise as
           heads of linked lists */
        Node Ahead = new Node(0), Dhead = new Node(0);

        // Split the list into lists
        splitList(Ahead, Dhead);

        Ahead = Ahead.next;
        Dhead = Dhead.next;

        // reverse the descending list
        Dhead = reverseList(Dhead);

        // merge the 2 linked lists
        head = mergeList(Ahead,Dhead);
    }

    /* Function to reverse the linked list */
    Node reverseList(Node Dhead)
    {
        Node current = Dhead;
        Node prev = null;
        Node next;
        while (current != null)
        {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
        return prev;
    }
}
```

Chapter 313. Sort a linked list that is sorted alternating ascending and descending orders?

```
        current.next = prev;
        prev = current;
        current = next;
    }
    Dhead = prev;
    return Dhead;
}

/* Function to print linked list */
void printList()
{
    Node temp = head;
    while (temp != null)
    {
        System.out.print(temp.data+" ");
        temp = temp.next;
    }
    System.out.println();
}

// A utility function to merge two sorted linked lists
Node mergeList(Node head1, Node head2)
{
    // Base cases
    if (head1 == null) return head2;
    if (head2 == null) return head1;

    Node temp = null;
    if (head1.data < head2.data)
    {
        temp = head1;
        head1.next = mergeList(head1.next, head2);
    }
    else
    {
        temp = head2;
        head2.next = mergeList(head1, head2.next);
    }
    return temp;
}

// This function alternatively splits a linked list with head
// as head into two:
// For example, 10->20->30->15->40->7 is splitted into 10->30->40
// and 20->15->7
// "Ahead" is reference to head of ascending linked list
// "Dhead" is reference to head of descending linked list
void splitList(Node Ahead, Node Dhead)
```

```
{  
    Node ascn = Ahead;  
    Node dscn = Dhead;  
    Node curr = head;  
  
    // Link alternate nodes  
  
    while (curr != null)  
    {  
        // Link alternate nodes in ascending order  
        ascn.next = curr;  
        ascn = ascn.next;  
        curr = curr.next;  
  
        if (curr != null)  
        {  
            dscn.next = curr;  
            dscn = dscn.next;  
            curr = curr.next;  
        }  
    }  
  
    ascn.next = null;  
    dscn.next = null;  
}  
  
/* Driver program to test above functions */  
public static void main(String args[])  
{  
    LinkedList llist = new LinkedList();  
    llist.head = llist.newNode(10);  
    llist.head.next = llist.newNode(40);  
    llist.head.next.next = llist.newNode(53);  
    llist.head.next.next.next = llist.newNode(30);  
    llist.head.next.next.next.next = llist.newNode(67);  
    llist.head.next.next.next.next.next = llist.newNode(12);  
    llist.head.next.next.next.next.next.next = llist.newNode(89);  
  
    System.out.println("Given linked list");  
    llist.printList();  
  
    llist.sort();  
  
    System.out.println("Sorted linked list");  
    llist.printList();  
}  
  
} /* This code is contributed by Rajat Mishra */
```

Python

```
# Python program to sort a linked list that is alternatively
# sorted in increasing and decreasing order
class LinkedList(object):
    def __init__(self):
        self.head = None

    # Linked list Node
    class Node(object):
        def __init__(self, d):
            self.data = d
            self.next = None

    def newNode(self, key):
        return self.Node(key)

    # This is the main function that sorts
    # the linked list.
    def sort(self):
        # Create 2 dummy nodes and initialise as
        # heads of linked lists
        Ahead = self.Node(0)
        Dhead = self.Node(0)
        # Split the list into lists
        self.splitList(Ahead, Dhead)
        Ahead = Ahead.next
        Dhead = Dhead.next
        # reverse the descending list
        Dhead = self.reverseList(Dhead)
        # merge the 2 linked lists
        self.head = self.mergeList(Ahead, Dhead)

    # Function to reverse the linked list
    def reverseList(self, Dhead):
        current = Dhead
        prev = None
        while current != None:
            self._next = current.next
            current.next = prev
            prev = current
            current = self._next
        Dhead = prev
        return Dhead

    # Function to print linked list
    def printList(self):
        temp = self.head
```

Chapter 313. Sort a linked list that is sorted alternating ascending and descending orders?

```
        while temp != None:
            print temp.data,
            temp = temp.next
        print ''

# A utility function to merge two sorted linked lists
def mergeList(self, head1, head2):
    # Base cases
    if head1 == None:
        return head2
    if head2 == None:
        return head1
    temp = None
    if head1.data < head2.data:
        temp = head1
        head1.next = self.mergeList(head1.next, head2)
    else:
        temp = head2
        head2.next = self.mergeList(head1, head2.next)
    return temp

# This function alternatively splits a linked list with head
# as head into two:
# For example, 10->20->30->15->40->7 is splitted into 10->30->40
# and 20->15->7
# "Ahead" is reference to head of ascending linked list
# "Dhead" is reference to head of descending linked list
def splitList(self, Ahead, Dhead):
    ascn = Ahead
    dscn = Dhead
    curr = self.head
    # Link alternate nodes
    while curr != None:
        # Link alternate nodes in ascending order
        ascn.next = curr
        ascn = ascn.next
        curr = curr.next
        if curr != None:
            dscn.next = curr
            dscn = dscn.next
            curr = curr.next
    ascn.next = None
    dscn.next = None

# Driver program
llist = LinkedList()
llist.head = llist.newNode(10)
llist.head.next = llist.newNode(40)
```

Chapter 313. Sort a linked list that is sorted alternating ascending and descending orders?

```
llist.head.next.next = llist.newNode(53)
llist.head.next.next.next = llist.newNode(30)
llist.head.next.next.next.next = llist.newNode(67)
llist.head.next.next.next.next.next = llist.newNode(12)
llist.head.next.next.next.next.next.next = llist.newNode(89)

print 'Given linked list'
llist.printList()

llist.sort()

print 'Sorted linked list'
llist.printList()

# This code is contributed by BHAVYA JAIN
```

Output:

```
Given Linked List is
10 40 53 30 67 12 89
Sorted Linked List is
10 12 30 40 53 67 89
```

Thanks to Gaurav Ahirwar for suggesting this method.

Source

<https://www.geeksforgeeks.org/how-to-sort-a-linked-list-that-is-sorted-alternating-ascending-and-descending-order/>

Chapter 314

Sort a nearly sorted (or K sorted) array

Sort a nearly sorted (or K sorted) array - GeeksforGeeks

Given an array of n elements, where each element is at most k away from its target position, devise an algorithm that sorts in $O(n \log k)$ time. For example, let us consider k is 2, an element at index 7 in the sorted array, can be at indexes 5, 6, 7, 8, 9 in the given array.

```
Input : arr[] = {6, 5, 3, 2, 8, 10, 9}
        k = 3
Output : arr[] = {2, 3, 5, 6, 8, 9, 10}

Input : arr[] = {10, 9, 8, 7, 4, 70, 60, 50}
        k = 4
Output : arr[] = {4, 7, 8, 9, 10, 50, 60, 70}
```

We can **use Insertion Sort** to sort the elements efficiently. Following is the C code for standard Insertion Sort.

```
/* Function to sort an array using insertion sort*/
void insertionSort(int A[], int size)
{
    int i, key, j;
    for (i = 1; i < size; i++)
    {
        key = A[i];
        j = i-1;

        /* Move elements of A[0..i-1], that are greater than key, to one
           position ahead of their current position.
```

```

    This loop will run at most k times */
while (j >= 0 && A[j] > key)
{
    A[j+1] = A[j];
    j = j-1;
}
A[j+1] = key;
}
}

```

The inner loop will run at most k times. To move every element to its correct place, at most k elements need to be moved. So overall *complexity will be O(nk)*

We can sort such arrays **more efficiently with the help of Heap data structure**. Following is the detailed process that uses Heap.

- 1) Create a Min Heap of size k+1 with first k+1 elements. This will take O(k) time (See [this GFact](#))
- 2) One by one remove min element from heap, put it in result array, and add a new element to heap from remaining elements.

Removing an element and adding a new element to min heap will take Logk time. So overall complexity will be $O(k) + O((n-k)*\log k)$

C++

```

// A STL based C++ program to sort a nearly sorted array.
#include <bits/stdc++.h>
using namespace std;

// Given an array of size n, where every element
// is k away from its target position, sorts the
// array in O(nLogk) time.
int sortK(int arr[], int n, int k)
{
    // Insert first k+1 items in a priority queue (or min heap)
    // (A O(k) operation). We assume, k < n.
    priority_queue<int, vector<int>, greater<int> > pq(arr, arr + k + 1);

    // i is index for remaining elements in arr[] and index
    // is target index of for current minimum element in
    // Min Heapm 'hp'.
    int index = 0;
    for (int i = k + 1; i < n; i++) {
        arr[index++] = pq.top();
        pq.pop();
        pq.push(arr[i]);
    }

    while (pq.empty() == false) {

```

```
        arr[index++] = pq.top();
        pq.pop();
    }
}

// A utility function to print array elements
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver program to test above functions
int main()
{
    int k = 3;
    int arr[] = { 2, 6, 3, 12, 56, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    sortK(arr, n, k);

    cout << "Following is sorted arrayn";
    printArray(arr, n);

    return 0;
}
```

C

```
#include<iostream>
using namespace std;

// Prototype of a utility function to swap two integers
void swap(int *x, int *y);

// A class for Min Heap
class MinHeap
{
    int *harr; // pointer to array of elements in heap
    int heap_size; // size of min heap
public:
    // Constructor
    MinHeap(int a[], int size);

    // to heapify a subtree with root at given index
    void MinHeapify(int );

    // to get index of left child of node at index i
```

```
int left(int i) { return (2*i + 1); }

// to get index of right child of node at index i
int right(int i) { return (2*i + 2); }

// to remove min (or root), add a new value x, and return old root
int replaceMin(int x);

// to extract the root which is the minimum element
int extractMin();

};

// Given an array of size n, where every element is k away from its target
// position, sorts the array in O(nLogk) time.
int sortK(int arr[], int n, int k)
{
    // Create a Min Heap of first (k+1) elements from
    // input array
    int *harr = new int[k+1];
    for (int i = 0; i<=k && i<n; i++) // i < n condition is needed when k > n
        harr[i] = arr[i];
    MinHeap hp(harr, k+1);

    // i is index for remaining elements in arr[] and ti
    // is target index of for cuurent minimum element in
    // Min Heapm 'hp'.
    for(int i = k+1, ti = 0; ti < n; i++, ti++)
    {
        // If there are remaining elements, then place
        // root of heap at target index and add arr[i]
        // to Min Heap
        if (i < n)
            arr[ti] = hp.replaceMin(arr[i]);

        // Otherwise place root at its target index and
        // reduce heap size
        else
            arr[ti] = hp.extractMin();
    }
}

// FOLLOWING ARE IMPLEMENTATIONS OF STANDARD MIN HEAP METHODS FROM CORMEN BOOK
// Constructor: Builds a heap from a given array a[] of given size
MinHeap::MinHeap(int a[], int size)
{
    heap_size = size;
    harr = a; // store address of array
    int i = (heap_size - 1)/2;
```

```
while (i >= 0)
{
    MinHeapify(i);
    i--;
}
}

// Method to remove minimum element (or root) from min heap
int MinHeap::extractMin()
{
    int root = harr[0];
    if (heap_size > 1)
    {
        harr[0] = harr[heap_size-1];
        heap_size--;
        MinHeapify(0);
    }
    return root;
}

// Method to change root with given value x, and return the old root
int MinHeap::replaceMin(int x)
{
    int root = harr[0];
    harr[0] = x;
    if (root < x)
        MinHeapify(0);
    return root;
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && harr[l] < harr[i])
        smallest = l;
    if (r < heap_size && harr[r] < harr[smallest])
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}
```

```
// A utility function to swap two elements
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// A utility function to print array elements
void printArray(int arr[], int size)
{
    for (int i=0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver program to test above functions
int main()
{
    int k = 3;
    int arr[] = {2, 6, 3, 12, 56, 8};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortK(arr, n, k);

    cout << "Following is sorted array";
    printArray (arr, n);

    return 0;
}
```

Output:

```
Following is sorted array
2 3 6 8 12 56
```

The Min Heap based method takes $O(n\log k)$ time and uses $O(k)$ auxiliary space.

We can also **use a Balanced Binary Search Tree** instead of Heap to store $K+1$ elements. The **insert** and **delete** operations on Balanced BST also take $O(\log k)$ time. So Balanced BST based method will also take $O(n\log k)$ time, but the Heap based method seems to be more efficient as the minimum element will always be at root. Also, Heap doesn't need extra space for left and right pointers.

Source

<https://www.geeksforgeeks.org/nearly-sorted-algorithm/>

Chapter 315

Sort a nearly sorted array using STL

Sort a nearly sorted array using STL - GeeksforGeeks

Given an array of n elements, where each element is at most k away from its target position, devise an algorithm that sorts in $O(n \log k)$ time. For example, let us consider k is 2, an element at index 7 in the sorted array, can be at indexes 5, 6, 7, 8, 9 in the given array. It may be assumed that $k < n$.

```
Input : arr[] = {6, 5, 3, 2, 8, 10, 9}
        k = 3
Output : arr[] = {2, 3, 5, 6, 8, 9, 10}

Input : arr[] = {10, 9, 8, 7, 4, 70, 60, 50}
        k = 4
Output : arr[] = {4, 7, 8, 9, 10, 50, 60, 70}
```

A **simple solution** is to sort the array using any standard sorting algorithm. The time complexity of this solution is $O(n \log n)$

A better solution is to use priority queue (or heap data structure).

- 1) Build a priority queue pq of first $(k+1)$ elements.
- 2) Initialize index = 0 (For result array).
- 3) Do following for elements from $k+1$ to $n-1$.
 - a) Pop an item from pq and put it at index, increment index.
 - b) Push $arr[i]$ to pq.
- 4) While pq is not empty
 Pop an item from pq and put it at index, increment index.

We have discussed a simple implementation in [Sort a nearly sorted \(or K sorted\) array](#). In this post, an STL based implementation is done.

```
// A STL based C++ program to sort a nearly sorted array.
#include <bits/stdc++.h>
using namespace std;

// Given an array of size n, where every element
// is k away from its target position, sorts the
// array in O(nLogk) time.
int sortK(int arr[], int n, int k)
{
    // Insert first k+1 items in a priority queue (or min heap)
    //((A O(k) operation)
    priority_queue<int, vector<int>, greater<int> > pq(arr, arr + k + 1);

    // i is index for remaining elements in arr[] and index
    // is target index of for current minimum element in
    // Min Heap 'hp'.
    int index = 0;
    for (int i = k + 1; i < n; i++) {
        arr[index++] = pq.top();
        pq.pop();
        pq.push(arr[i]);
    }

    while (pq.empty() == false) {
        arr[index++] = pq.top();
        pq.pop();
    }
}

// A utility function to print array elements
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver program to test above functions
int main()
{
    int k = 3;
    int arr[] = { 2, 6, 3, 12, 56, 8 };
    int n = sizeof(arr) / sizeof(arr[0]);
    sortK(arr, n, k);

    cout << "Following is sorted arrayn";
    printArray(arr, n);
}
```

```
    return 0;  
}
```

Output:

Following is sorted arrayn2 3 6 8 12 56

Time Complexity : $O(n \log k)$
Auxiliary Space : $O(k)$

Source

<https://www.geeksforgeeks.org/sort-a-nearly-sorted-array-using-stl/>

Chapter 316

Sort a stack using a temporary stack

Sort a stack using a temporary stack - GeeksforGeeks

Given a stack of integers, sort it in ascending order using another temporary stack.

Examples:

Input : [34, 3, 31, 98, 92, 23]
Output : [3, 23, 31, 34, 92, 98]

Input : [3, 5, 1, 4, 2, 8]
Output : [1, 2, 3, 4, 5, 8]

We follow this algorithm.

1. Create a temporary stack say **tmpStack**.
2. While input stack is NOT empty do this:
 - Pop an element from input stack call it **temp**
 - while temporary stack is NOT empty and top of temporary stack is greater than temp,
pop from temporary stack and push it to the input stack
 - push **temp** in temporary stack
3. The sorted numbers are in tmpStack

Here is a dry run of above pseudo code.

```
input: [34, 3, 31, 98, 92, 23]
```

```
Element taken out: 23
input: [34, 3, 31, 98, 92]
tmpStack: [23]
```

```
Element taken out: 92
input: [34, 3, 31, 98]
tmpStack: [23, 92]
```

```
Element taken out: 98
input: [34, 3, 31]
tmpStack: [23, 92, 98]
```

```
Element taken out: 31
input: [34, 3, 98, 92]
tmpStack: [23, 31]
```

```
Element taken out: 92
input: [34, 3, 98]
tmpStack: [23, 31, 92]
```

```
Element taken out: 98
input: [34, 3]
tmpStack: [23, 31, 92, 98]
```

```
Element taken out: 3
input: [34, 98, 92, 31, 23]
tmpStack: [3]
```

```
Element taken out: 23
input: [34, 98, 92, 31]
tmpStack: [3, 23]
```

```
Element taken out: 31
input: [34, 98, 92]
tmpStack: [3, 23, 31]
```

```
Element taken out: 92
input: [34, 98]
tmpStack: [3, 23, 31, 92]
```

```
Element taken out: 98
input: [34]
tmpStack: [3, 23, 31, 92, 98]
```

```
Element taken out: 34
```

```
input: [98, 92]
tmpStack: [3, 23, 31, 34]

Element taken out: 92
input: [98]
tmpStack: [3, 23, 31, 34, 92]

Element taken out: 98
input: []
tmpStack: [3, 23, 31, 34, 92, 98]

final sorted list: [3, 23, 31, 34, 92, 98]
```

C++

```
// C++ program to sort a stack using an
// auxiliary stack.
#include <bits/stdc++.h>
using namespace std;

// This function return the sorted stack
stack<int> sortStack(stack<int> &input)
{
    stack<int> tmpStack;

    while (!input.empty())
    {
        // pop out the first element
        int tmp = input.top();
        input.pop();

        // while temporary stack is not empty and top
        // of stack is greater than temp
        while (!tmpStack.empty() && tmpStack.top() > tmp)
        {
            // pop from temporary stack and push
            // it to the input stack
            input.push(tmpStack.top());
            tmpStack.pop();
        }

        // push temp in temporary stack
        tmpStack.push(tmp);
    }

    return tmpStack;
}
```

```
// main function
int main()
{
    stack<int> input;
    input.push(34);
    input.push(3);
    input.push(31);
    input.push(98);
    input.push(92);
    input.push(23);

    // This is the temporary stack
    stack<int> tmpStack = sortStack(input);
    cout << "Sorted numbers are:\n";

    while (!tmpStack.empty())
    {
        cout << tmpStack.top() << " ";
        tmpStack.pop();
    }
}
```

Java

```
// Java program to sort a stack using
// a auxiliary stack.
import java.util.*;

class SortStack
{
    // This function return the sorted stack
    public static Stack<Integer> sortstack(Stack<Integer>
                                              input)
    {
        Stack<Integer> tmpStack = new Stack<Integer>();
        while(!input.isEmpty())
        {
            // pop out the first element
            int tmp = input.pop();

            // while temporary stack is not empty and
            // top of stack is greater than temp
            while(!tmpStack.isEmpty() && tmpStack.peek()
                  > tmp)
            {
                // pop from temporary stack and
                // push it to the input stack

```

```
        input.push(tmpStack.pop());
    }

    // push temp in temporary of stack
    tmpStack.push(tmp);
}
return tmpStack;
}

// Driver Code
public static void main(String args[])
{
    Stack<Integer> input = new Stack<Integer>();
    input.add(34);
    input.add(3);
    input.add(31);
    input.add(98);
    input.add(92);
    input.add(23);

    // This is the temporary stack
    Stack<Integer> tmpStack=sortstack(input);
    System.out.println("Sorted numbers are:");

    while (!tmpStack.empty())
    {
        System.out.print(tmpStack.pop()+" ");
    }
}
// This code is contributed by Danish Kaleem
```

Python3

```
# Python program to sort a
# stack using auxiliary stack.

# This function return the sorted stack
def sortStack ( stack ):
    tmpStack = createStack()
    while(isEmpty(stack) == False):

        # pop out the first element
        tmp = top(stack)
        pop(stack)

        # while temporary stack is not
        # empty and top of stack is
```

```
# greater than temp
while(isEmpty(tmpStack) == False and
      int(top(tmpStack)) > int(tmp)):

    # pop from temporary stack and
    # push it to the input stack
    push(stack,top(tmpStack))
    pop(tmpStack)

    # push temp in temporary stack
    push(tmpStack,tmp)

return tmpStack

# Below is a complete running
# program for testing above
# function.

# Function to create a stack.
# It initializes size of stack
# as 0
def createStack():
    stack = []
    return stack

# Function to check if
# the stack is empty
def isEmpty( stack ):
    return len(stack) == 0

# Function to push an
# item to stack
def push( stack, item ):
    stack.append( item )

# Function to get top
# item of stack
def top( stack ):
    p = len(stack)
    return stack[p-1]

# Function to pop an
# item from stack
def pop( stack ):

    # If stack is empty
    # then error
    if(isEmpty( stack )):
```

```
print("Stack Underflow ")
exit(1)

return stack.pop()

# Function to print the stack
def prints(stack):
    for i in range(len(stack)-1, -1, -1):
        print(stack[i], end = ' ')
    print()

# Driver Code
stack = createStack()
push( stack, str(34) )
push( stack, str(3) )
push( stack, str(31) )
push( stack, str(98) )
push( stack, str(92) )
push( stack, str(23) )

print("Sorted numbers are: ")
sortedst = sortStack ( stack )
prints(sortedst)

# This code is contributed by
# Prasad Kshirsagar
```

Output:

```
Sorted numbers are:
98 92 34 31 23 3
```

Microsoft

Improved By : [programmer2k17](#), [Prasad_Kshirsagar](#)

Source

<https://www.geeksforgeeks.org/sort-stack-using-temporary-stack/>

Chapter 317

Sort all even numbers in ascending order and then sort all odd numbers in descending order

Sort all even numbers in ascending order and then sort all odd numbers in descending order
- GeeksforGeeks

Given an array of integers (both odd and even), sort them in such a way that the first part of the array contains odd numbers sorted in descending order, rest portion contains even numbers sorted in ascending order.

Examples:

Input : arr[] = {1, 2, 3, 5, 4, 7, 10}
Output : arr[] = {7, 5, 3, 1, 2, 4, 10}

Input : arr[] = {0, 4, 5, 3, 7, 2, 1}
Output : arr[] = {7, 5, 3, 1, 0, 2, 4}

Asked in : Microsoft

Method 1 (Using Partition)

1. Partition the input array such that all odd elements are moved to left and all even elements on right. This step takes $O(n)$.
2. Once the array is partitioned, sort left and right parts individually. This step takes $O(n \log n)$.

Below is implementation of above idea.

C++

```
// C++ program sort array in even and odd manner.  
// The odd numbers are to be sorted in descending  
// order and the even numbers in ascending order  
#include<bits/stdc++.h>  
using namespace std;  
  
// To do two way sort. First sort even numbers in  
// ascending order, then odd numbers in descending  
// order.  
void twoWaySort(int arr[], int n)  
{  
    // Current indexes from left and right  
    int l = 0, r = n-1;  
  
    // Count of odd numbers  
    int k = 0;  
  
    while (l < r)  
    {  
        // Find first odd number from left side.  
        while (arr[l]%2 != 0)  
        {  
            l++;  
            k++;  
        }  
  
        // Find first even number from right side.  
        while (arr[r]%2 == 0 && l < r)  
            r--;  
  
        // Swap odd number present on left and even  
        // number right.  
        if (l < r)  
            swap(arr[l], arr[r]);  
    }  
  
    // Sort odd number in descending order  
    sort(arr, arr+k, greater<int>());  
  
    // Sort even number in ascending order  
    sort(arr+k, arr+n);  
}  
  
// Driver code  
int main()
```

```
{  
    int arr[] = {1, 3, 2, 7, 5, 4};  
    int n = sizeof(arr)/sizeof(int);  
    twoWaySort(arr, n);  
    for (int i=0; i<n; i++)  
        cout << arr[i] << " ";  
    return 0;  
}
```

Java

```
// Java program sort array in even and odd manner.  
// The odd numbers are to be sorted in descending  
// order and the even numbers in ascending order  
  
import java.util.Arrays;  
import java.util.Collections;  
  
public class GFG  
{  
    // To do two way sort. First sort even numbers in  
    // ascending order, then odd numbers in descending  
    // order.  
    static void twoWaySort(Integer arr[], int n)  
    {  
        // Current indexes from left and right  
        int l = 0, r = n-1;  
  
        // Count of odd numbers  
        int k = 0;  
  
        while (l < r)  
        {  
            // Find first odd number from left side.  
            while (arr[l]%2 != 0)  
            {  
                l++;  
                k++;  
            }  
  
            // Find first even number from right side.  
            while (arr[r]%2 == 0 && l<r)  
                r--;  
  
            // Swap odd number present on left and even  
            // number right.  
            if (l < r)  
            {
```

```
// swap arr[l] arr[r]
int temp = arr[l];
arr[l] = arr[r];
arr[r] = temp;

}

// Sort odd number in descending order
Arrays.sort(arr, 0, k,Collections.reverseOrder());

// Sort even number in ascending order
Arrays.sort(arr, k, n);
}

// Driver Method
public static void main(String[] args)
{
    Integer arr[] = {1, 3, 2, 7, 5, 4};

    twoWaySort(arr, arr.length);

    System.out.println(Arrays.toString(arr));
}
}
```

Python

```
# Python program to sort array
# in even and odd manner
# The odd numbers are to be
# sorted in descending order
# and the even numbers in
# ascending order

# To do two way sort. First
# sort even numbers in ascending
# order, then odd numbers in
# descending order.
def two_way_sort(arr, arr_len):

    # Current indexes l->left
    # and r->right
    l, r = 0, arr_len - 1

    # Count of number of
    # odd numbers, used in
```

```
# slicing the array later.
k = 0

# Run till left(l) < right(r)
while(l < r):

    # While left(l) is odd, if yes
    # increment the left(l) plus
    # odd count(k) if not break the
    # while for even number found
    # here to be swaped
    while(arr[l] % 2 != 0):
        l += 1
        k += 1

    # While right(r) is even,
    # if yes decrement right(r)
    # if not break the while for
    # odd number found here to
    # be swaped
    while(arr[r] % 2 == 0 and l < r):
        r -= 1

    # Swap the left(l) and right(r),
    # which is even and odd numbers
    # encountered in above loops
    if(l < r):
        arr[l], arr[r] = arr[r], arr[l]

    # Slice the number on the
    # basis of odd count(k)
    odd = arr[:k]
    even = arr[k:]

    # Sort the odd and
    # even array accordingly
    odd.sort(reverse = True)
    even.sort()

    # Extend the odd array with
    # even values and return it.
    odd.extend(even)

return odd

# Driver code
arr_len = 6
arr = [1, 3, 2, 7, 5, 4]
```

```
result = two_way_sort(arr, arr_len)
for i in result:
    print(str(i) + " ")

# This code is contributed
# by JaySiyaRam
```

Output:

```
7 5 3 1 2 4
```

Time complexity: $O(n \log n)$
space complexity: $O(1)$

Method 2 (Using negative multiplication) :

1. Make all odd numbers negative.
2. Sort all numbers.
3. Revert the changes made in step 1 to get original elements back.

C++

```
// C++ program sort array in even and odd manner.
// The odd numbers are to be sorted in descending
// order and the even numbers in ascending order
#include<bits/stdc++.h>
using namespace std;

// To do two way sort. First sort even numbers in
// ascending order, then odd numbers in descending
// order.
void twoWaySort(int arr[], int n)
{
    // Make all odd numbers negative
    for (int i=0 ; i<n ; i++)
        if (arr[i] & 1) // Check for odd
            arr[i] *= -1;

    // Sort all numbers
    sort(arr, arr+n);

    // Retaining original array
    for (int i=0 ; i<n ; i++)
        if (arr[i] & 1)
            arr[i] *= -1;
```

```
}
```

```
// Driver code
int main()
{
    int arr[] = {1, 3, 2, 7, 5, 4};
    int n = sizeof(arr)/sizeof(int);
    twoWaySort(arr, n);
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Java

```
// Java program sort array in even and odd manner.
// The odd numbers are to be sorted in descending
// order and the even numbers in ascending order

import java.util.Arrays;

public class GFG
{
    // To do two way sort. First sort even numbers in
    // ascending order, then odd numbers in descending
    // order.
    static void twoWaySort(int arr[], int n)
    {
        // Make all odd numbers negative
        for (int i=0 ; i<n ; i++)
            if (((arr[i] & 1) != 0) // Check for odd
                arr[i] *= -1;

        // Sort all numbers
        Arrays.sort(arr);

        // Retaining original array
        for (int i=0 ; i<n ; i++)
            if (((arr[i] & 1) != 0)
                arr[i] *= -1;
    }

    // Driver Method
    public static void main(String[] args)
    {
        int arr[] = {1, 3, 2, 7, 5, 4};

        twoWaySort(arr, arr.length);
    }
}
```

```
        System.out.println(Arrays.toString(arr));
    }
}
```

Python3

```
# Python 3 program to sort array in
# even and odd manner. The odd
# numbers are to be sorted in
# descending order and the even
# numbers in ascending order

# To do two way sort. First sort
# even numbers in ascending order,
# then odd numbers in descending order.
def twoWaySort(arr, n):

    # Make all odd numbers negative
    for i in range(0, n):

        # Check for odd
        if (arr[i] & 1):
            arr[i] *= -1

    # Sort all numbers
    arr.sort()

    # Retaining original array
    for i in range(0, n):
        if (arr[i] & 1):
            arr[i] *= -1

# Driver code
arr = [1, 3, 2, 7, 5, 4]
n = len(arr)
twoWaySort(arr, n);
for i in range(0, n):
    print(arr[i], end = " ")

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// Java program sort array in even and
// odd manner. The odd numbers are to
// be sorted in descending order and
```

```
// the even numbers in ascending order
using System;

public class GFG {

    // To do two way sort. First sort
    // even numbers in ascending order,
    // then odd numbers in descending
    // order.
    static void twoWaySort(int []arr, int n)
    {

        // Make all odd numbers negative
        for (int i = 0; i < n; i++)

            // Check for odd
            if ((arr[i] & 1) != 0)
                arr[i] *= -1;

        // Sort all numbers
        Array.Sort(arr);

        // Retaining original array
        for (int i = 0; i < n; i++)
            if ((arr[i] & 1) != 0)
                arr[i] *= -1;
    }

    // Driver Method
    public static void Main()
    {
        int []arr = {1, 3, 2, 7, 5, 4};

        twoWaySort(arr, arr.Length);

        for(int i = 0; i < arr.Length; i++)
            Console.Write(arr[i] + " ");
    }
}

// This code is contributed by Smitha
```

Output:

7 5 3 1 2 4

Time complexity: $O(n \log n)$
space complexity: $O(1)$

Chapter 317. Sort all even numbers in ascending order and then sort all odd numbers in descending order

This method may not work when input array contains negative numbers. However, there is a way to handle this. We count the positive odd integers in the input array then sort again. Readers may refer [this](#) for implementation.

Thanks to **Amandeep Singh** for suggesting this solution.

Improved By : [Smitha Dinesh Semwal, jaysiyaram](#)

Source

<https://www.geeksforgeeks.org/sort-even-numbers-ascending-order-sort-odd-numbers-descending-order/>

Chapter 318

Sort an almost sorted array where only two elements are swapped

Sort an almost sorted array where only two elements are swapped - GeeksforGeeks

Given an almost sorted array where only two elements are swapped, how to sort the array efficiently?

Examples :

```
Input: arr[] = {10, 20, 60, 40, 50, 30}  
// 30 and 60 are swapped  
Output: arr[] = {10, 20, 30, 40, 50, 60}
```

```
Input: arr[] = {10, 20, 40, 30, 50, 60}  
// 30 and 40 are swapped  
Output: arr[] = {10, 20, 30, 40, 50, 60}
```

```
Input: arr[] = {1, 5, 3}  
// 3 and 5 are swapped  
Output: arr[] = {1, 3, 5}
```

Expected time complexity is $O(n)$ and only one swap operation to fix the array.

The idea is to traverse from rightmost side and find the first out of order element (element which is smaller than previous element). Once first element is found, find the other out of order element by traversing the array toward left side.

Below is implementation of above idea.

C++

```
// C++ program to sort using one swap
#include<iostream>
#include<algorithm>
using namespace std;

// This function sorts an array that can be sorted
// by single swap
void sortByOneSwap(int arr[], int n)
{
    // Travers the given array from rightmost side
    for (int i = n-1; i > 0; i--)
    {
        // Check if arr[i] is not in order
        if (arr[i] < arr[i-1])
        {
            // Find the other element to be
            // swapped with arr[i]
            int j = i-1;
            while (j>=0 && arr[i] < arr[j])
                j--;

            // Swap the pair
            swap(arr[i], arr[j+1]);
            break;
        }
    }
}

// A utility function ot print an array of size n
void printArray(int arr[], int n)
{
    int i;
    for (i=0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

/* Driver program to test insertion sort */
int main()
{
    int arr[] = {10, 30, 20, 40, 50, 60, 70};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Given array is \n";
    printArray(arr, n);

    sortByOneSwap(arr, n);
}
```

```
    cout << "Sorted array is \n";
    printArray(arr, n);

    return 0;
}
```

Java

```
// Java program to
// sort using one swap
import java.io.*;

class GFG
{
// This function sorts an array
// that can be sorted by single swap
static void sortByOneSwap(int arr[],
                           int n)
{
    // Traverses the given array
    // from rightmost side
    for (int i = n - 1; i > 0; i--)
    {
        // Check if arr[i]
        // is not in order
        if (arr[i] < arr[i - 1])
        {
            // Find the other element
            // to be swapped with arr[i]
            int j = i - 1;
            while (j >= 0 && arr[i] < arr[j])
                j--;

            // Swap the pair
            int temp = arr[i];
            arr[i] = arr[j + 1];
            arr[j + 1] = temp;

            break;
        }
    }

    // A utility function to
    // print an array of size n
    static void printArray(int arr[], int n)
    {
        int i;
```

```
for (i = 0; i < n; i++)
    System.out.print(arr[i] + " ");
System.out.println();
}

// Driver Code
public static void main(String[] args)
{
int arr[] = {10, 30, 20,
             40, 50, 60, 70};
int n = arr.length;

System.out.println("Given array is ");
printArray(arr, n);

sortByOneSwap(arr, n);

System.out.println("Sorted array is ");
printArray(arr, n);
}
}

// This code is contributed by anuj_67.
```

Output :

```
Given array is
10 30 20 40 50 60 70
Sorted array is
10 20 30 40 50 60 70
```

The above program works in $O(n)$ time and swaps only one element.

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/sort-an-almost-sorted-array-where-only-two-elements-are-swapped/>

Chapter 319

Sort an array according to absolute difference with a given value “using constant extra space”

Sort an array according to absolute difference with a given value "using constant extra space"
- GeeksforGeeks

Given an array of n distinct elements and a number x, arrange array elements according to the absolute difference with x, i. e., element having minimum difference comes first and so on, using constant extra space.

Note : If two or more elements are at equal distance arrange them in same sequence as in the given array.

Examples:

```
Input  : arr[] = {10, 5, 3, 9, 2}
         x = 7
Output : arr[] = {5, 9, 10, 3, 2}
Explanation :
7 - 10 = 3(abs)
7 - 5 = 2
7 - 3 = 4
7 - 9 = 2(abs)
7 - 2 = 5
So according to the difference with X,
elements are arranged as 5, 9, 10, 3, 2.
```

```
Input  : arr[] = {1, 2, 3, 4, 5}
```

```
x = 6
Output : arr[] = {5, 4, 3, 2, 1}
```

The above problem has already been explained in a previous post [here](#). It takes $O(n \log n)$ time and $O(n)$ extra space. The below solution though has a relatively bad time complexity i.e $O(n^2)$ but it does the work without using any additional space or memory.

The solution is based on [Insertion Sort](#). For every i ($1 \leq i < n$) we compare the absolute value of the difference of $arr[i]$ with the given number x (Let this be 'diff'). We then compare this difference with the difference of $abs(arr[j]-x)$ where $0 \leq j < i$ (Let this be abdiff). If diff is greater than abdiff, we shift the values in the array to accommodate $arr[i]$ in its correct position.

```
// C++ program to sort an array based on absolute
// difference with a given value x.
#include <bits/stdc++.h>
using namespace std;

void arrange(int arr[], int n, int x)
{
    // Below lines are similar to insertion sort
    for (int i = 1; i < n; i++) {
        int diff = abs(arr[i] - x);

        // Insert arr[i] at correct place
        int j = i - 1;
        if (abs(arr[j] - x) > diff) {
            int temp = arr[i];
            while (abs(arr[j] - x) > diff && j >= 0) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = temp;
        }
    }

    // Function to print the array
    void print(int arr[], int n)
    {
        for (int i = 0; i < n; i++)
            cout << arr[i] << " ";
    }

    // Main Function
    int main()
    {
        int arr[] = { 10, 5, 3, 9, 2 };
        arrange(arr, 5, 6);
        print(arr, 5);
    }
}
```

```
int n = sizeof(arr) / sizeof(arr[0]);
int x = 7;

arrange(arr, n, x);
print(arr, n);

return 0;
}
```

Output:

5 9 10 3 2

Time Complexity : $O(n^2)$ where n is the size of the array.
Auxiliary Space : $O(1)$

Source

<https://www.geeksforgeeks.org/sort-array-according-absolute-difference-given-value-using-constant-extra-space/>

Chapter 320

Sort an array according to absolute difference with given value

Sort an array according to absolute difference with given value - GeeksforGeeks

Given an array of n distinct elements and a number x, arrange array elements according to the absolute difference with x, i. e., element having minimum difference comes first and so on.

Note : If two or more elements are at equal distance arrange them in same sequence as in the given array.

Examples :

Input : arr[] : x = 7, arr[] = {10, 5, 3, 9, 2}

Output : arr[] = {5, 9, 10, 3, 2}

Explanation:

7 - 10 = 3(abs)

7 - 5 = 2

7 - 3 = 4

7 - 9 = 2(abs)

7 - 2 = 5

So according to the difference with X,
elements are arranged as 5, 9, 10, 3, 2.

Input : x = 6, arr[] = {1, 2, 3, 4, 5}

Output : arr[] = {5, 4, 3, 2, 1}

Input : x = 5, arr[] = {2, 6, 8, 3}

Output : arr[] = {6, 3, 2, 8}

The idea is to use a self balancing binary search tree. We traverse input array and for every element, we find its difference with x and store the difference as key and element as value in self balancing binary search tree. Finally we traverse the tree and print its inorder traversal which is required output.

C++ Implementation :

In C++, self-balancing-binary-search-tree is implemented by [set](#), [map](#) and [multimap](#). We can't use set here as we have key value pairs (not only keys). We also can't directly use map also as a single key can belong to multiple values and map allows a single value for a key. So we use multimap which stores key value pairs and can have multiple values for a key.

1. Store the values in the multimap with the difference with X as key.
2. In multimap, the values will be already in sorted order according to key i.e. difference with X because it implements self-balancing-binary-search-tree internally.
3. Update all the values of array with the values of map so that array has the required output.

```
// C++ program to sort an array according absolute
// difference with x.
#include<bits/stdc++.h>
using namespace std;

// Function to sort an array according absolute
// difference with x.
void rearrange(int arr[], int n, int x)
{
    multimap<int, int> m;
    multimap<int ,int >:: iterator it;
    // Store values in a map with the difference
    // with X as key
    for (int i = 0 ; i < n; i++)
        m.insert(make_pair(abs(x-arr[i]),arr[i]));

    // Update the values of array
    int i = 0;
    for (it = m.begin(); it != m.end(); it++)
        arr[i++] = (*it).second ;
}

// Function to print the array
void printArray(int arr[] , int n)
{
    for (int i = 0 ; i < n; i++)
        cout << arr[i] << " ";
}

// Driver code
```

```
int main()
{
    int arr[] = {10, 5, 3, 9, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 7;
    rearrange(arr, n, x);
    printArray(arr, n);
    return 0;
}
```

Output:

5 9 10 3 2

Time Complexity : $O(n \log n)$
Auxiliary Space : $O(n)$

Improved By : [bhaveshsingh](#)

Source

<https://www.geeksforgeeks.org/sort-an-array-according-to-absolute-difference-with-given-value/>

Chapter 321

Sort an array according to count of set bits

Sort an array according to count of set bits - GeeksforGeeks

Given an array of positive integers, sort the array in decreasing order of count of set bits in binary representations of array elements.

For integers having same number of set bits in their binary representation, sort according to their position in the original array i.e., a stable sort. For example, if input array is {3, 5}, then output array should also be {3, 5}. Note that both 3 and 5 have same number set bits.

Examples:

Input: arr[] = {5, 2, 3, 9, 4, 6, 7, 15, 32};
Output: 15 7 5 3 9 6 2 4 32

Explanation:

The integers in their binary representation are:

15 -1111
7 -0111
5 -0101
3 -0011
9 -1001
6 -0110
2 -0010
4 -0100
32 -10000

hence the non-increasing sorted order is:
{15}, {7}, {5, 3, 9, 6}, {2, 4, 32}

Input: arr[] = {1, 2, 3, 4, 5, 6};
Output: 3 5 6 1 2 4

Explanation:

```
3 - 0011
5 - 0101
6 - 0110
1 - 0001
2 - 0010
4 - 0100
hence the non-increasing sorted order is
{3, 5, 6}, {1, 2, 4}
```

Method 1: Simple

1. Create an auxiliary array and store the set-bit counts of all integers in the aux array
2. Simultaneously sort both arrays according to the non-increasing order of auxiliary array. (Note that we need to use a stable sort algorithm)

Before sort:

```
int arr[] = {1, 2, 3, 4, 5, 6};
int aux[] = {1, 1, 2, 1, 2, 2}
```

After sort:

```
arr = {3, 5, 6, 1, 2, 4}
aux = {2, 2, 2, 1, 1, 1}
```

Implementation:

C++

```
// C++ program to implement simple approach to sort
// an array according to count of set bits.
#include <iostream>

using namespace std;

// a utility function that returns total set bits
// count in an integer
int countBits(int a)
{
    int count = 0;
    while (a)
    {
        if (a & 1)
            count+= 1;
        a = a>>1;
    }
    return count;
}
```

```
// Function to simultaneously sort both arrays
// using insertion sort
// ( http://quiz.geeksforgeeks.org/insertion-sort/ )
void insertionSort(int arr[], int aux[], int n)
{
    for (int i = 1; i < n; i++)
    {
        // use 2 keys because we need to sort both
        // arrays simultaneously
        int key1 = aux[i];
        int key2 = arr[i];
        int j = i-1;

        /* Move elements of arr[0..i-1] and aux[0..i-1],
           such that elements of aux[0..i-1] are
           greater than key1, to one position ahead
           of their current position */
        while (j >= 0 && aux[j] < key1)
        {
            aux[j+1] = aux[j];
            arr[j+1] = arr[j];
            j = j-1;
        }
        aux[j+1] = key1;
        arr[j+1] = key2;
    }
}

// Function to sort according to bit count using
// an auxiliary array
void sortBySetBitCount(int arr[], int n)
{
    // Create an array and store count of
    // set bits in it.
    int aux[n];
    for (int i=0; i<n; i++)
        aux[i] = countBits(arr[i]);

    // Sort arr[] according to values in aux[]
    insertionSort(arr, aux, n);
}

// Utility function to print an array
void printArr(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}
```

```
// Driver Code
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortBySetBitCount(arr, n);
    printArr(arr, n);
    return 0;
}
```

Java

```
// Java program to implement
// simple approach to sort
// an array according to
// count of set bits.
import java.io.*;

class GFG
{

    // a utility function that
    // returns total set bits
    // count in an integer
    static int countBits(int a)
    {
        int count = 0;
        while (a > 0)
        {
            if ((a & 1) > 0)
                count+= 1;
            a = a >> 1;
        }
        return count;
    }

    // Function to simultaneously
    // sort both arrays using
    // insertion sort
    // (http://quiz.geeksforgeeks.org/insertion-sort/)
    static void insertionSort(int arr[],
                           int aux[], int n)
    {
        for (int i = 1; i < n; i++)
        {
            // use 2 keys because we
            // need to sort both
```

```
// arrays simultaneously
int key1 = aux[i];
int key2 = arr[i];
int j = i - 1;

/* Move elements of arr[0..i-1]
and aux[0..i-1], such that
elements of aux[0..i-1] are
greater than key1, to one
position ahead of their current
position */
while (j >= 0 && aux[j] < key1)
{
    aux[j + 1] = aux[j];
    arr[j + 1] = arr[j];
    j = j - 1;
}
aux[j + 1] = key1;
arr[j + 1] = key2;
}

// Function to sort according
// to bit count using an
// auxiliary array
static void sortBySetBitCount(int arr[],
                             int n)
{
    // Create an array and
    // store count of
    // set bits in it.
    int aux[] = new int[n];
    for (int i = 0; i < n; i++)
        aux[i] = countBits(arr[i]);

    // Sort arr[] according
    // to values in aux[]
    insertionSort(arr, aux, n);
}

// Utility function
// to print an array
static void printArr(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}
```

```
// Driver Code
public static void main (String[] args)
{
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = arr.length;
    sortBySetBitCount(arr, n);
    printArr(arr, n);
}
}

// This code is contributed by anuj_67.
```

Output:

```
3 5 6 1 2 4
```

Auxiliary Space: O(n)
Time complexity: O(n^2)

Note: Time complexity can be improved to O(nLogn) by using a stable O(nlogn) sorting algorithm.

Method 2 : Using std::sort()

Using custom comparator of std::sort to sort the array according to set-bit count

C++

```
// C++ program to sort an array according to
// count of set bits using std::sort()
#include <bits/stdc++.h>

using namespace std;

// a utility function that returns total set bits
// count in an integer
int countBits(int a)
{
    int count = 0;
    while (a)
    {
        if (a & 1)
            count+= 1;
        a = a>>1;
    }
    return count;
}
```

```
// custom comparator of std::sort
int cmp(int a,int b)
{
    int count1 = countBits(a);
    int count2 = countBits(b);

    // this takes care of the stability of
    // sorting algorithm too
    if (count1 <= count2)
        return false;
    return true;
}

// Function to sort according to bit count using
// std::sort
void sortBySetBitCount(int arr[], int n)
{
    stable_sort(arr, arr+n, cmp);
}

// Utility function to print an array
void printArr(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}

// Driver Code
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortBySetBitCount(arr, n);
    printArr(arr, n);
    return 0;
}
```

Output:

3 5 6 1 2 4

Auxiliary Space : O(1)
Time complexity : O(n log n)

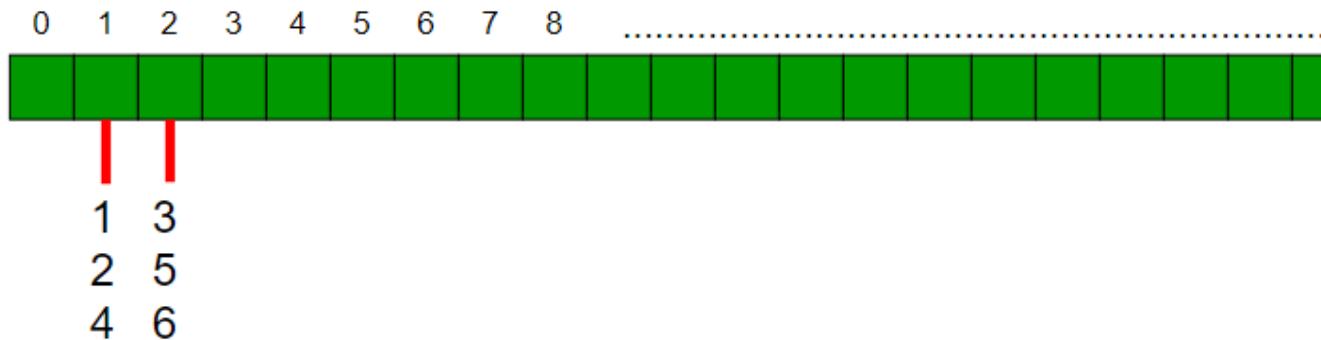
Method 3 : [Counting Sort](#) based

This problem can be solved in $O(n)$ time. The idea is similar to counting sort.

Note: There can be minimum 1 set-bit and only a maximum of 31set-bits in any integer.

Steps (assuming that an integer takes 32 bits):

1. Create a vector “count” of size 32. Each cell of count i.e., $\text{count}[i]$ is another vector that stores all the elements whose set-bit-count is i
2. Traverse the array and do following for each element:
 - (a) Count the number set-bits of this element. Let it be ‘setbitcount’
 - (b) $\text{count}[\text{setbitcount}].\text{push_back}(\text{element})$
3. Traverse ‘count’ in reverse fashion(as we need to sort in non-increasing order) and modify the array.



C++

```
// C++ program to sort an array according to
// count of set bits using std::sort()
#include <bits/stdc++.h>
using namespace std;

// a utility function that returns total set bits
// count in an integer
int countBits(int a)
{
    int count = 0;
    while (a)
    {
        if (a & 1)
            count+= 1;
        a = a>>1;
    }
    return count;
```

```
}

// Function to sort according to bit count
// This function assumes that there are 32
// bits in an integer.
void sortBySetBitCount(int arr[], int n)
{
    vector<vector<int>> count(32);
    int setbitcount = 0;
    for (int i=0; i<n; i++)
    {
        setbitcount = countBits(arr[i]);
        count[setbitcount].push_back(arr[i]);
    }

    int j = 0; // Used as an index in final sorted array

    // Traverse through all bit counts (Note that we
    // sort array in decreasing order)
    for (int i=31; i>=0; i--)
    {
        vector<int> v1 = count[i];
        for (int i=0; i<v1.size(); i++)
            arr[j++] = v1[i];
    }
}

// Utility function to print an array
void printArr(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}

// Driver Code
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortBySetBitCount(arr, n);
    printArr(arr, n);
    return 0;
}
```

Output:

3 5 6 1 2 4

Improved By : [vt_m](#), [prasad gujar](#)

Source

<https://www.geeksforgeeks.org/sort-array-according-count-set-bits/>

Chapter 322

Sort an array according to the order defined by another array

Sort an array according to the order defined by another array - GeeksforGeeks

Given two arrays A1[] and A2[], sort A1 in such a way that the relative order among the elements will be same as those are in A2. For the elements not present in A2, append them at last in sorted order.

```
Input: A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8}
       A2[] = {2, 1, 8, 3}
Output: A1[] = {2, 2, 1, 1, 8, 8, 3, 5, 6, 7, 9}
```

The code should handle all cases like number of elements in A2[] may be more or less compared to A1[]. A2[] may have some elements which may not be there in A1[] and vice versa is also possible.

Source: [Amazon Interview Set 110 \(On-Campus\)](#)

Method 1 (Using Sorting and Binary Search)

Let size of A1[] be m and size of A2[] be n.

- 1) Create a temporary array temp of size m and copy contents of A1[] to it.
- 2) Create another array visited[] and initialize all entries in it as false. visited[] is used to mark those elements in temp[] which are copied to A1[].
- 3) Sort temp[]
- 4) Initialize the output index ind as 0.
- 5) Do following for every element of A2[i] in A2[]
 -a) Binary search for all occurrences of A2[i] in temp[], if present then copy all occurrences to A1[ind] and increment ind. Also mark the copied elements visited[]
- 6) Copy all unvisited elements from temp[] to A1[].

Time complexity: The steps 1 and 2 require O(m) time. Step 3 requires O(mLogm) time. Step 5 requires O(nLogm) time. Therefore overall time complexity is O(mLogm + nLogm).

Thanks to [vivek](#)for suggesting this method. Following is the implementation of above algorithm.

C++

```
// A C++ program to sort an array according to the order defined
// by another array
#include <iostream>
#include <algorithm>

using namespace std;

/* A Binary Search based function to find index of FIRST occurrence
   of x in arr[]. If x is not present, then it returns -1 */
int first(int arr[], int low, int high, int x, int n)
{
    if (high >= low)
    {
        int mid = low + (high-low)/2; /* (low + high)/2; */
        if ((mid == 0 || x > arr[mid-1]) && arr[mid] == x)
            return mid;
        if (x > arr[mid])
            return first(arr, (mid + 1), high, x, n);
        return first(arr, low, (mid - 1), x, n);
    }
    return -1;
}

// Sort A1[0..m-1] according to the order defined by A2[0..n-1].
void sortAccording(int A1[], int A2[], int m, int n)
{
    // The temp array is used to store a copy of A1[] and visited[]
    // is used mark the visited elements in temp[].
    int temp[m], visited[m];
    for (int i=0; i<m; i++)
    {
        temp[i] = A1[i];
        visited[i] = 0;
    }

    // Sort elements in temp
    sort(temp, temp + m);

    int ind = 0; // for index of output which is sorted A1[]

    // Consider all elements of A2[], find them in temp[]
    // and copy to A1[] in order.
    for (int i=0; i<n; i++)
    {
```

```
// Find index of the first occurrence of A2[i] in temp
int f = first(temp, 0, m-1, A2[i], m);

// If not present, no need to proceed
if (f == -1) continue;

// Copy all occurrences of A2[i] to A1[]
for (int j = f; (j<m && temp[j]==A2[i]); j++)
{
    A1[ind++] = temp[j];
    visited[j] = 1;
}
}

// Now copy all items of temp[] which are not present in A2[]
for (int i=0; i<m; i++)
    if (visited[i] == 0)
        A1[ind++] = temp[i];
}

// Utility function to print an array
void printArray(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver program to test above function.
int main()
{
    int A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8};
    int A2[] = {2, 1, 8, 3};
    int m = sizeof(A1)/sizeof(A1[0]);
    int n = sizeof(A2)/sizeof(A2[0]);
    cout << "Sorted array is \n";
    sortAccording(A1, A2, m, n);
    printArray(A1, m);
    return 0;
}
```

Java

```
// A JAVA program to sort an array according
// to the order defined by another array
import java.io.*;
import java.util.Arrays;
```

```
class GFG {

    /* A Binary Search based function to find
    index of FIRST occurrence of x in arr[].
    If x is not present, then it returns -1 */
    static int first(int arr[], int low, int high,
                     int x, int n)
    {
        if (high >= low)
        {
            /* (low + high)/2; */
            int mid = low + (high-low)/2;

            if ((mid == 0 || x > arr[mid-1]) &&
                arr[mid] == x)
                return mid;
            if (x > arr[mid])
                return first(arr, (mid + 1), high,
                            x, n);
            return first(arr, low, (mid -1), x, n);
        }
        return -1;
    }

    // Sort A1[0..m-1] according to the order
    // defined by A2[0..n-1].
    static void sortAccording(int A1[], int A2[], int m,
                             int n)
    {
        // The temp array is used to store a copy
        // of A1[] and visited[] is used to mark the
        // visited elements in temp[].
        int temp[] = new int[m], visited[] = new int[m];
        for (int i = 0; i < m; i++)
        {
            temp[i] = A1[i];
            visited[i] = 0;
        }

        // Sort elements in temp
        Arrays.sort(temp);

        // for index of output which is sorted A1[]
        int ind = 0;

        // Consider all elements of A2[], find them
        // in temp[] and copy to A1[] in order.
        for (int i = 0; i < n; i++)
    }
```

```
{  
    // Find index of the first occurrence  
    // of A2[i] in temp  
    int f = first(temp, 0, m-1, A2[i], m);  
  
    // If not present, no need to proceed  
    if (f == -1) continue;  
  
    // Copy all occurrences of A2[i] to A1[]  
    for (int j = f; (j < m && temp[j] == A2[i]);  
         j++)  
    {  
        A1[ind++] = temp[j];  
        visited[j] = 1;  
    }  
}  
  
// Now copy all items of temp[] which are  
// not present in A2[]  
for (int i = 0; i < m; i++)  
    if (visited[i] == 0)  
        A1[ind++] = temp[i];  
}  
  
// Utility function to print an array  
static void printArray(int arr[], int n)  
{  
    for (int i = 0; i < n; i++)  
        System.out.print(arr[i] + " ");  
    System.out.println();  
}  
  
// Driver program to test above function.  
public static void main(String args[])  
{  
    int A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8};  
    int A2[] = {2, 1, 8, 3};  
    int m = A1.length;  
    int n = A2.length;  
    System.out.println("Sorted array is ");  
    sortAccording(A1, A2, m, n);  
    printArray(A1, m);  
}  
}
```

/*This code is contributed by Nikita Tiwari.*/

Python3

```
"""A Python 3 program to sort an array
according to the order defined by
another array"""

"""A Binary Search based function to find
index of FIRST occurrence of x in arr[].
If x is not present, then it returns -1 """

def first(arr, low, high, x, n) :
    if (high >= low) :
        mid = low + (high - low) // 2; # (low + high)/2;
        if ((mid == 0 or x > arr[mid-1]) and arr[mid] == x) :
            return mid
        if (x > arr[mid]) :
            return first(arr, (mid + 1), high, x, n)
        return first(arr, low, (mid - 1), x, n)

    return -1

# Sort A1[0..m-1] according to the order
# defined by A2[0..n-1].
def sortAccording(A1, A2, m, n) :

    """The temp array is used to store a copy
    of A1[] and visited[] is used mark the
    visited elements in temp[]."""
    temp = [0] * m
    visited = [0] * m

    for i in range(0, m) :
        temp[i] = A1[i]
        visited[i] = 0

    # Sort elements in temp
    temp.sort()

    # for index of output which is sorted A1[]
    ind = 0

    """Consider all elements of A2[], find
    them in temp[] and copy to A1[] in order."""
    for i in range(0,n) :

        # Find index of the first occurrence
        # of A2[i] in temp
        f = first(temp, 0, m-1, A2[i], m)
```

```
# If not present, no need to proceed
if (f == -1) :
    continue

# Copy all occurrences of A2[i] to A1[]
j = f
while (j<m and temp[j]==A2[i]) :
    A1[ind] = temp[j];
    ind=ind+1
    visited[j] = 1
    j = j + 1

# Now copy all items of temp[] which are
# not present in A2[]
for i in range(0, m) :
    if (visited[i] == 0) :
        A1[ind] = temp[i]
        ind = ind + 1

# Utility function to print an array
def printArray(arr, n) :
    for i in range(0, n) :
        print(arr[i], end = " ")
    print("")

# Driver program to test above function.
A1 = [2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8]
A2 = [2, 1, 8, 3]
m = len(A1)
n = len(A2)
print("Sorted array is ")
sortAccording(A1, A2, m, n)
printArray(A1, m)

# This code is contributed by Nikita Tiwari.
```

C#

```
// A C# program to sort an array according
// to the order defined by another array
using System;

class GFG {

    /* A Binary Search based function to find
```

```
index of FIRST occurrence of x in arr[].  
If x is not present, then it returns -1 */  
static int first(int []arr, int low,  
                  int high, int x, int n)  
{  
    if (high >= low)  
    {  
        /* (low + high)/2; */  
        int mid = low + (high - low) / 2;  
  
        if ((mid == 0 || x > arr[mid-1]) &&  
            arr[mid] == x)  
            return mid;  
        if (x > arr[mid])  
            return first(arr, (mid + 1), high,  
                         x, n);  
        return first(arr, low, (mid - 1), x, n);  
    }  
    return -1;  
}  
  
// Sort A1[0..m-1] according to the order  
// defined by A2[0..n-1].  
static void sortAccording(int []A1, int []A2,  
                        int m, int n)  
{  
  
    // The temp array is used to store a copy  
    // of A1[] and visited[] is used to mark  
    // the visited elements in temp[].  
    int []temp = new int[m];  
    int []visited = new int[m];  
  
    for (int i = 0; i < m; i++)  
    {  
        temp[i] = A1[i];  
        visited[i] = 0;  
    }  
  
    // Sort elements in temp  
    Array.Sort(temp);  
  
    // for index of output which is  
    // sorted A1[]  
    int ind = 0;  
  
    // Consider all elements of A2[], find  
    // them in temp[] and copy to A1[] in
```

```
// order.
for (int i = 0; i < n; i++)
{
    // Find index of the first occurrence
    // of A2[i] in temp
    int f = first(temp, 0, m-1, A2[i], m);

    // If not present, no need to proceed
    if (f == -1) continue;

    // Copy all occurrences of A2[i] to A1[]
    for (int j = f; (j < m &&
                      temp[j] == A2[i]); j++)
    {
        A1[ind++] = temp[j];
        visited[j] = 1;
    }
}

// Now copy all items of temp[] which are
// not present in A2[]
for (int i = 0; i < m; i++)
    if (visited[i] == 0)
        A1[ind++] = temp[i];
}

// Utility function to print an array
static void printArray(int []arr, int n)
{
    for (int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
    Console.WriteLine();
}

// Driver program to test above function.
public static void Main()
{
    int []A1 = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8};
    int []A2 = {2, 1, 8, 3};
    int m = A1.Length;
    int n = A2.Length;
    Console.WriteLine("Sorted array is ");
    sortAccording(A1, A2, m, n);
    printArray(A1, m);
}
```

```
// This code is contributed by nitin mittal.
```

Output:

```
Sorted array is
2 2 1 1 8 8 3 5 6 7 9
```

Method 2 (Using Self-Balancing Binary Search Tree)

We can also use a self balancing BST like [AVL Tree](#), [Red Black Tree](#), etc. Following are detailed steps.

- 1) Create a self balancing BST of all elements in A1[]. In every node of BST, also keep track of count of occurrences of the key and a bool field visited which is initialized as false for all nodes.
- 2) Initialize the output index ind as 0.
- 3) Do following for every element of A2[i] in A2[]
.....a) Search for A2[i] in the BST, if present then copy all occurrences to A1[ind] and increment ind. Also mark the copied elements visited in the BST node.
- 4) Do an inorder traversal of BST and copy all unvisited keys to A1[].

Time Complexity of this method is same as the previous method. Note that in a self balancing Binary Search Tree, all operations require logm time.

Method 3 (Use Hashing)

1. Loop through A1[], store the count of every number in a HashMap (key: number, value: count of number) .
2. Loop through A2[], check if it is present in HashMap, if so, put in output array that many times and remove the number from HashMap.
3. Sort the rest of the numbers present in HashMap and put in output array.

Thanks to [Anurag Sigh](#) for suggesting this method.

The steps 1 and 2 on average take $O(m+n)$ time under the assumption that we have a good hashing function that takes $O(1)$ time for insertion and search on average. The third step takes $O(pLogp)$ time where p is the number of elements remained after considering elements of A2[].

Method 4 (By Writing a Customized Compare Method)

We can also customize compare method of a sorting algorithm to solve the above problem. For example [qsort\(\) in C allows us to pass our own customized compare method](#).

1. If num1 and num2 both are in A2 then number with lower index in A2 will be treated smaller than other.
2. If only one of num1 or num2 present in A2, then that number will be treated smaller than the other which doesn't present in A2.
3. If both are not in A2, then natural ordering will be taken.

Time complexity of this method is $O(mnLogm)$ if we use a $O(nLogn)$ time complexity sorting algorithm. We can improve time complexity to $O(mLogm)$ by using a Hashing instead of doing linear search.

Following is C implementation of this method.

```
// A C++ program to sort an array according to the order defined
// by another array
#include <stdio.h>
#include <stdlib.h>

// A2 is made global here so that it can be accessed by compareByA2()
// The syntax of qsort() allows only two parameters to compareByA2()
int A2[5];
int size = 5; // size of A2[]

int search(int key)
{
    int i=0, idx = 0;
    for (i=0; i<size; i++)
        if (A2[i] == key)
            return i;
    return -1;
}

// A custom compare method to compare elements of A1[] according
// to the order defined by A2[].
int compareByA2(const void * a, const void * b)
{
    int idx1 = search(*(int*)a);
    int idx2 = search(*(int*)b);
    if (idx1 != -1 && idx2 != -1)
        return idx1 - idx2;
    else if(idx1 != -1)
        return -1;
    else if(idx2 != -1)
        return 1;
    else
        return ( *(int*)a - *(int*)b );
}

// This method mainly uses qsort to sort A1[] according to A2[]
void sortA1ByA2(int A1[], int size1)
{
    qsort(A1, size1, sizeof (int), compareByA2);
}

// Driver program to test above function
int main(int argc, char *argv[])
{
    int A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8, 7, 5, 6, 9, 7, 5};

    //A2[] = {2, 1, 8, 3, 4};
    A2[0] = 2;
```

```
A2[1] = 1;
A2[2] = 8;
A2[3] = 3;
A2[4] = 4;
int size1 = sizeof(A1)/sizeof(A1[0]);

sortA1ByA2(A1, size1);

printf("Sorted Array is ");
int i;
for (i=0; i<size1; i++)
    printf("%d ", A1[i]);
return 0;
}
```

Output:

```
Sorted Array is 2 2 1 1 8 8 3 5 5 5 6 6 7 7 7 9 9
```

This method is based on comments by readers (Xinuo Chen, Pranay Doshi and javakurious) and compiled by Anurag Singh.

This article is compiled by **Piyush**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [nitin mittal](#), [Suryaveer Singh](#)

Source

<https://www.geeksforgeeks.org/sort-array-according-order-defined-another-array/>

Chapter 323

Sort an array after applying the given equation

Sort an array after applying the given equation - GeeksforGeeks

We have an integer array that is sorted in ascending order. We also have 3 integers A, B and C. We need to apply $A*x*x + B*x + C$ for each element x in the array and sort the modified array.

```
Input : arr[] = {-1, 0, 1, 2, 3, 4}
        A = -1, B = 2, C = -1
Output : {-9, -4, -4, -1, -1, 0}
Input array is {-1, 0, 1, 2, 3, 4}. After
applying the equation A*x*x + B*x + C on
every element x we get, {-4,-1, 0, -1, -4, -9}
After sorting, we get {-9, -4, -4, -1, -1, 0}
```

Asked in : Adobe

Method 1 (Simple) :

- 1- Apply the given equation on all the elements. $O(n)$
- 2- Sort the modified array. $O(n \log n)$

Time complexity of $O(n \log n)$

Method 2(Efficient): Parabolic Property

The equation given is parabolic. So the result of applying it to a sorted array will result in an array that will have a maximum/minimum with the sub-arrays to its left and right sorted.

In the above example, maximum is 0 and the sub array to its left $\{-4, -1\}$ is sorted in ascending order and the sub-array to its right $\{-1, -4, -9\}$ is sorted in descending order. All we need to do is merge these sorted arrays which is linear in time.

So the algorithm is:

1. Apply equation on each element.
2. Find maximum/minimum.
3. Merge sub-arrays.

Note : The below code assumes that the modified array is first increasing then decreasing.

C++

```
// C program to sort an array after applying equation
// A*x*x + B*x + C
#include<bits/stdc++.h>
using namespace std;

// Function to sort an array after applying given
// equation.
void sortArray(int arr[], int n, int A, int B, int C)
{
    // Apply equation on all elements
    for (int i = 0; i < n; i++)
        arr[i] = A*arr[i]*arr[i] + B*arr[i] + C;

    // Find maximum element in resultant array
    int index, maximum = INT_MIN;
    for (int i = 0; i < n; i++)
    {
        if (maximum < arr[i])
        {
            index = i;
            maximum = arr[i];
        }
    }

    // Use maximum element as a break point
    // and merge both subarrays usin simple
    // merge function of merge sort
    int i = 0, j = n-1;
    int new_arr[n], k = 0;
    while (i < index && j > index)
    {
        if (arr[i] < arr[j])
            new_arr[k++] = arr[i++];
        else
            new_arr[k++] = arr[j--];
    }
}
```

```
// Merge remaining elements
while (i < index)
    new_arr[k++] = arr[i++];
while (j > index)
    new_arr[k++] = arr[j--];

new_arr[n-1] = maximum;

// Modify original array
for (int i = 0; i < n ; i++)
    arr[i] = new_arr[i];
}

// Driver code
int main()
{
    int arr[] = {-21 ,-15, 12, 13, 14 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int A = -6, B =-7, C = 2;

    sortArray(arr, n, A, B, C);

    cout << "Array after sorting is : n";
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Java

```
// Java program to sort an array after applying equation
// A*x*x + B*x + C

class Main
{
    // Function to sort an array after applying given
    // equation.
    static void sortArray(int arr[], int n, int A, int B, int C)
    {
        // Apply equation on all elements
        for (int i = 0; i < n; i++)
            arr[i] = A*arr[i]*arr[i] + B*arr[i] + C;

        // Find maximum element in resultant array
        int index=-1;
        int maximum = -999999;
```

```
for (int i = 0; i< n; i++)
{
    if (maximum < arr[i])
    {
        index = i;
        maximum = arr[i];
    }
}

// Use maximum element as a break point
// and merge both subarrays usin simple
// merge function of merge sort
int i = 0, j = n-1;
int[] new_arr = new int[n];
int k = 0;
while (i < index && j > index)
{
    if (arr[i] < arr[j])
        new_arr[k++] = arr[i++];
    else
        new_arr[k++] = arr[j--];
}

// Merge remaining elements
while (i < index)
    new_arr[k++] = arr[i++];
while (j > index)
    new_arr[k++] = arr[j--];

new_arr[n-1] = maximum;

// Modify original array
for (int p = 0; p < n ; p++)
    arr[p] = new_arr[p];
}

// main function
public static void main (String[] args)
{
    int arr[] = {-21 ,-15, 12, 13, 14 };
    int n = arr.length;
    int A = -6, B =-7, C = 2;

    sortArray(arr, n, A, B, C);

    System.out.println("Array after sorting is : ");
    for (int i=0; i<n; i++)
        System.out.print(arr[i]+" ");
}
```

```
    }  
}  
  
/* This code is contributed by Harsh Agarwal */
```

Output:

```
Array after sorting is :  
-2497 -1272 -1243 -1103 -946
```

Time Complexity : $O(n)$
Auxiliary Space : $O(n)$

Reference:

<http://stackoverflow.com/questions/4551599/sorting-result-array>

Source

<https://www.geeksforgeeks.org/sort-array-applying-given-equation/>

Chapter 324

Sort an array containing two types of elements

Sort an array containing two types of elements - GeeksforGeeks

We are given an array of 0s and 1s in random order. Segregate 0s on left side and 1s on right side of the array. Traverse array only once.

Examples:

```
Input : arr[] = [0, 1, 0, 1, 0, 0, 1, 1, 1, 0]
Output : arr[] = [0, 0, 0, 0, 1, 1, 1, 1]
```

```
Input : arr[] = [1, 1, 1, 0, 1, 0, 0, 1, 1, 1]
Output : arr[] = [0, 0, 0, 1, 1, 1, 1, 1]
```

We have already discussed a solution [Segregate 0s and 1s in an array](#)

In this post, a new solution is discussed.

Step 1 : Here we can take two pointers type0 (for element 0) starting from beginning (index = 0) and type1 (for element 1) starting from end index.

Step 2: We intend to put 1 to the right side of the array. Once we have done this then 0 will definitely towards left side of array to achieve this we do following.

We compare elements at index type0

1) if this is 1 then this should be moved to right side so we need to swap this with index type1 once swapped we are sure that element at index type1 is '1' so we need to decrement index type1

2) else it will be 0 then we need to simple increment index type0

C++

```
// CPP program to sort an array with two types
// of values in one traversal.
#include <bits/stdc++.h>
using namespace std;

/* Method for segregation 0 and 1 given
   input array */
void segregate0and1(int arr[], int n)
{
    int type0 = 0;
    int type1 = n - 1;

    while (type0 < type1) {
        if (arr[type0] == 1) {
            swap(arr[type0], arr[type1]);
            type1--;
        }
        else {
            type0++;
        }
    }
}

// Driver program
int main()
{
    int arr[] = { 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1 };
    int n = sizeof(arr)/sizeof(arr[0]);
    segregate0and1(arr, n);
    for (int a : arr)
        cout << a << " ";
}
```

Java

```
// Java program to sort an array with two types
// of values in one traversal.
public class GFG {
    /* Method for segregation 0 and 1
       given input array */
    static void segregate0and1(int arr[], int n) {
        int type0 = 0;
        int type1 = n - 1;

        while (type0 < type1) {
            if (arr[type0] == 1) {

                // swap type0 and type1
                arr[type0] = arr[type0] + arr[type1];
                arr[type1] = arr[type0] - arr[type1];
            }
            type1--;
        }
    }
}
```

```
        arr[type1] = arr[type0]-arr[type1];
        arr[type0] = arr[type0]-arr[type1];
        type1--;
    } else {
        type0++;
    }
}

// Driver program
public static void main(String[] args) {
    int arr[] = { 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1 };

    segregate0and1(arr, arr.length);
    for (int a : arr)
        System.out.print(a+" ");
}
}
```

Python3

```
# Python3 program to sort an array with
# two types of values in one traversal.

# Method for segregation 0 and
# 1 given input array
def segregate0and1(arr, n):

    type0 = 0; type1 = n - 1

    while (type0 < type1):
        if (arr[type0] == 1):
            arr[type0], arr[type1] = arr[type1], arr[type0]
            type1 -= 1

        else:
            type0 += 1

# Driver Code
arr = [1, 1, 1, 0, 1, 0, 0, 1, 1, 1]
n = len(arr)
segregate0and1(arr, n)
for i in range(0, n):
    print(arr[i], end = " ")

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// C# program to sort an array with two types
// of values in one traversal.
using System;

class GFG {

    static void segregate0and1(int []arr, int n)
    {
        int type0 = 0;
        int type1 = n - 1;

        while (type0 < type1)
        {

            if (arr[type0] == 1)
            {

                // swap type0 and type1
                arr[type0] = arr[type0] + arr[type1];
                arr[type1] = arr[type0]-arr[type1];
                arr[type0] = arr[type0]-arr[type1];
                type1--;
            }
            else {
                type0++;
            }
        }
    }

    // Driver program
    public static void Main()
    {

        int []arr = { 1, 1, 1, 0, 1, 0, 0,
                     1, 1, 1, 1 };

        segregate0and1(arr, arr.Length);

        for (int i = 0; i < arr.Length; i++)
            Console.Write(arr[i] + " ");
    }
}

// This code is contributed by vt_m.
```

Output:

0 0 0 1 1 1 1 1 1 1

Improved By : [san4net](#)

Source

<https://www.geeksforgeeks.org/sort-array-containing-two-types-elements/>

Chapter 325

Sort an array in wave form

Sort an array in wave form - GeeksforGeeks

Given an unsorted array of integers, sort the array into a wave like array. An array ‘arr[0..n-1]’ is sorted in wave form if $\text{arr}[0] \geq \text{arr}[1] \leq \text{arr}[2] \geq \text{arr}[3] \leq \text{arr}[4] \geq \dots$

Examples:

Input: arr[] = {10, 5, 6, 3, 2, 20, 100, 80}
Output: arr[] = {10, 5, 6, 2, 20, 3, 100, 80} OR
{20, 5, 10, 2, 80, 6, 100, 3} OR
any other array that is in wave form

Input: arr[] = {20, 10, 8, 6, 4, 2}
Output: arr[] = {20, 8, 10, 4, 6, 2} OR
{10, 8, 20, 2, 6, 4} OR
any other array that is in wave form

Input: arr[] = {2, 4, 6, 8, 10, 20}
Output: arr[] = {4, 2, 8, 6, 20, 10} OR
any other array that is in wave form

Input: arr[] = {3, 6, 5, 10, 7, 20}
Output: arr[] = {6, 3, 10, 5, 20, 7} OR
any other array that is in wave form

A **Simple Solution** is to use sorting. First sort the input array, then swap all adjacent elements.

For example, let the input array be {3, 6, 5, 10, 7, 20}. After sorting, we get {3, 5, 6, 7, 10, 20}. After swapping adjacent elements, we get {5, 3, 7, 6, 20, 10}.

Below are implementations of this simple approach.

C++

```
// A C++ program to sort an array in wave form using
// a sorting function
#include<iostream>
#include<algorithm>
using namespace std;

// A utility method to swap two numbers.
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// This function sorts arr[0..n-1] in wave form, i.e.,
// arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= arr[5]..
void sortInWave(int arr[], int n)
{
    // Sort the input array
    sort(arr, arr+n);

    // Swap adjacent elements
    for (int i=0; i<n-1; i += 2)
        swap(&arr[i], &arr[i+1]);
}

// Driver program to test above function
int main()
{
    int arr[] = {10, 90, 49, 2, 1, 5, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortInWave(arr, n);
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Java

```
// Java implementation of naive method for sorting
// an array in wave form.
import java.util.*;

class SortWave
{
    // A utility method to swap two numbers.
    void swap(int arr[], int a, int b)
    {
```

```
int temp = arr[a];
arr[a] = arr[b];
arr[b] = temp;
}

// This function sorts arr[0..n-1] in wave form, i.e.,
// arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4]..
void sortInWave(int arr[], int n)
{
    // Sort the input array
    Arrays.sort(arr);

    // Swap adjacent elements
    for (int i=0; i<n-1; i += 2)
        swap(arr, i, i+1);
}

// Driver method
public static void main(String args[])
{
    SortWave ob = new SortWave();
    int arr[] = {10, 90, 49, 2, 1, 5, 23};
    int n = arr.length;
    ob.sortInWave(arr, n);
    for (int i : arr)
        System.out.print(i + " ");
}
/*This code is contributed by Rajat Mishra*/
```

Python

```
# Python function to sort the array arr[0..n-1] in wave form,
# i.e., arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= arr[5]
def sortInWave(arr, n):

    #sort the array
    arr.sort()

    # Swap adjacent elements
    for i in range(0,n-1,2):
        arr[i], arr[i+1] = arr[i+1], arr[i]

    # Driver program
    arr = [10, 90, 49, 2, 1, 5, 23]
    sortInWave(arr, len(arr))
    for i in range(0,len(arr)):
        print arr[i],
```

```
# This code is contributed by __Devesh Agrawal__
```

C#

```
// C# implementation of naive method
// for sorting an array in wave form.
using System;

class SortWave {

    // A utility method to swap two numbers.
    void swap(int[] arr, int a, int b)
    {
        int temp = arr[a];
        arr[a] = arr[b];
        arr[b] = temp;
    }

    // This function sorts arr[0..n-1] in wave form, i.e.,
    // arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4]...
    void sortInWave(int[] arr, int n)
    {
        // Sort the input array
        Array.Sort(arr);

        // Swap adjacent elements
        for (int i = 0; i < n - 1; i += 2)
            swap(arr, i, i + 1);
    }

    // Driver method
    public static void Main()
    {
        SortWave ob = new SortWave();
        int[] arr = { 10, 90, 49, 2, 1, 5, 23 };
        int n = arr.Length;

        ob.sortInWave(arr, n);
        for (int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");
    }
}

// This code is contributed by vt_m.
```

Output:

2 1 10 5 49 23 90

The time complexity of the above solution is $O(n\log n)$ if a $O(n\log n)$ sorting algorithm like [Merge Sort](#), [Heap Sort](#), .. etc is used.

This can be done in **$O(n)$ time by doing a single traversal** of given array. The idea is based on the fact that if we make sure that all even positioned (at index 0, 2, 4, ..) elements are greater than their adjacent odd elements, we don't need to worry about odd positioned element. Following are simple steps.

1) Traverse all even positioned elements of input array, and do following.

....a) If current element is smaller than previous odd element, swap previous and current.

....b) If current element is smaller than next odd element, swap next and current.

Below are implementations of above simple algorithm.

C++

```
// A O(n) program to sort an input array in wave form
#include<iostream>
using namespace std;

// A utility method to swap two numbers.
void swap(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

// This function sorts arr[0..n-1] in wave form, i.e., arr[0] >=
// arr[1] <= arr[2] >= arr[3] <= arr[4] >= arr[5] ....
void sortInWave(int arr[], int n)
{
    // Traverse all even elements
    for (int i = 0; i < n; i+=2)
    {
        // If current even element is smaller than previous
        if (i>0 && arr[i-1] > arr[i] )
            swap(&arr[i], &arr[i-1]);

        // If current even element is smaller than next
        if (i<n-1 && arr[i] < arr[i+1] )
            swap(&arr[i], &arr[i + 1]);
    }
}

// Driver program to test above function
int main()
{
```

```
int arr[] = {10, 90, 49, 2, 1, 5, 23};  
int n = sizeof(arr)/sizeof(arr[0]);  
sortInWave(arr, n);  
for (int i=0; i<n; i++)  
    cout << arr[i] << " ";  
return 0;  
}
```

Java

```
// A O(n) Java program to sort an input array in wave form  
class SortWave  
{  
    // A utility method to swap two numbers.  
    void swap(int arr[], int a, int b)  
    {  
        int temp = arr[a];  
        arr[a] = arr[b];  
        arr[b] = temp;  
    }  
  
    // This function sorts arr[0..n-1] in wave form, i.e.,  
    // arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4]....  
    void sortInWave(int arr[], int n)  
    {  
        // Traverse all even elements  
        for (int i = 0; i < n; i+=2)  
        {  
            // If current even element is smaller  
            // than previous  
            if (i>0 && arr[i-1] > arr[i] )  
                swap(arr, i-1, i);  
  
            // If current even element is smaller  
            // than next  
            if (i<n-1 && arr[i] < arr[i+1] )  
                swap(arr, i, i + 1);  
        }  
    }  
  
    // Driver program to test above function  
    public static void main(String args[])  
    {  
        SortWave ob = new SortWave();  
        int arr[] = {10, 90, 49, 2, 1, 5, 23};  
        int n = arr.length;  
        ob.sortInWave(arr, n);  
        for (int i : arr)
```

```
        System.out.print(i+" ");
    }
}
/*This code is contributed by Rajat Mishra*/
```

Python

```
# Python function to sort the array arr[0..n-1] in wave form,
# i.e., arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= arr[5]
def sortInWave(arr, n):

    # Traverse all even elements
    for i in range(0, n, 2):

        # If current even element is smaller than previous
        if (i> 0 and arr[i] < arr[i-1]):
            arr[i],arr[i-1] = arr[i-1],arr[i]

        # If current even element is smaller than next
        if (i < n-1 and arr[i] < arr[i+1]):
            arr[i],arr[i+1] = arr[i+1],arr[i]

    # Driver program
    arr = [10, 90, 49, 2, 1, 5, 23]
    sortInWave(arr, len(arr))
    for i in range(0,len(arr)):
        print arr[i],

# This code is contributed by _Devesh Agrawal_
```

C#

```
// A O(n) C# program to sort an
// input array in wave form
using System;

class SortWave {

    // A utility method to swap two numbers.
    void swap(int[] arr, int a, int b)
    {
        int temp = arr[a];
        arr[a] = arr[b];
        arr[b] = temp;
    }

    // This function sorts arr[0..n-1] in wave form, i.e.,
}
```

```
// arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4]....  
void sortInWave(int[] arr, int n)  
{  
    // Traverse all even elements  
    for (int i = 0; i < n; i += 2) {  
  
        // If current even element is smaller  
        // than previous  
        if (i > 0 && arr[i - 1] > arr[i])  
            swap(arr, i - 1, i);  
  
        // If current even element is smaller  
        // than next  
        if (i < n - 1 && arr[i] < arr[i + 1])  
            swap(arr, i, i + 1);  
    }  
}  
  
// Driver program to test above function  
public static void Main()  
{  
    SortWave ob = new SortWave();  
    int[] arr = { 10, 90, 49, 2, 1, 5, 23 };  
    int n = arr.Length;  
  
    ob.sortInWave(arr, n);  
    for (int i = 0; i < n; i++)  
        Console.Write(arr[i] + " ");  
}  
}  
  
// This code is contributed by vt_m.
```

Output:

90 10 49 1 5 2 23

This article is contributed by **Shivam**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

<https://www.geeksforgeeks.org/sort-array-wave-form-2/>

Chapter 326

Sort an array of 0s, 1s and 2s

Sort an array of 0s, 1s and 2s - GeeksforGeeks

Given an array A[] consisting 0s, 1s and 2s, write a function that sorts A[]. The functions should put all 0s first, then all 1s and all 2s in last.

Examples:

Input : {0, 1, 2, 0, 1, 2}
Output : {0, 0, 1, 1, 2, 2}

Input : {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1}
Output : {0, 0, 0, 0, 1, 1, 1, 1, 2, 2}

A simple solution is discussed in below post.

[Sort an array of 0s, 1s and 2s \(Simple Counting\)](#)

The problem is similar to our old post [Segregate 0s and 1s in an array](#), and both of these problems are variation of famous [Dutch national flag problem](#).

The problem was posed with three colours, here '0 , '1 and '2 . The array is divided into four sections:

1. a[1..Lo-1] zeroes (red)
2. a[Lo..Mid-1] ones (white)
3. a[Mid..Hi] unknown
4. a[Hi+1..N] twos (blue)

The unknown region is shrunk while maintaining these conditions

1. Lo := 1; Mid := 1; Hi := N;

2. while Mid <= Hi do

- (a) Invariant: a[1..Lo-1]=0 and a[Lo..Mid-1]=1 and a[Hi+1..N]=2; a[Mid..Hi] are unknown.
- (b) case a[Mid] in
 - 0: swap a[Lo] and a[Mid]; Lo++; Mid++
 - 1: Mid++
 - 2: swap a[Mid] and a[Hi]; Hi--

— Dutch National Flag Algorithm, or 3-way Partitioning —

[YOU NEED A JavaScript ENABLED BOWSER!!!]

Part way through the process, some red, white and blue elements are known and are in the “right” place. The section of unknown elements, a[Mid..Hi], is shrunk by examining a[Mid]:

Examine a[Mid]. There are three possibilities:

a[Mid] is (0) red, (1) white or (2) blue.

Case (0) a[Mid] is red, swap a[Lo] and a[Mid]; Lo++; Mid++

Case (1) a[Mid] is white, Mid++

Case (2) a[Mid] is blue, swap a[Mid] and a[Hi]; Hi--

Continue until Mid>Hi.

Below is C implementation of above algorithm.

C

```
// C program to sort an array with 0,1 and 2
// in a single pass
#include<stdio.h>

/* Function to swap *a and *b */
void swap(int *a, int *b);
```

```
// Sort the input array, the array is assumed to
// have values in {0, 1, 2}
void sort012(int a[], int arr_size)
{
    int lo = 0;
    int hi = arr_size - 1;
    int mid = 0;

    while (mid <= hi)
    {
        switch (a[mid])
        {
            case 0:
                swap(&a[lo++], &a[mid++]);
                break;
            case 1:
                mid++;
                break;
            case 2:
                swap(&a[mid], &a[hi--]);
                break;
        }
    }
}

/* UTILITY FUNCTIONS */
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

/* Utility function to print array arr[] */
void printArray(int arr[], int arr_size)
{
    int i;
    for (i = 0; i < arr_size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

/* driver program to test */
int main()
{
    int arr[] = {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1};
    int arr_size = sizeof(arr)/sizeof(arr[0]);
```

```
int i;

sort012(arr, arr_size);

printf("array after segregation ");
printArray(arr, arr_size);

getchar();
return 0;
}
```

Java

```
// Java program to sort an array of 0, 1 and 2
import java.io.*;

class countzot {

    // Sort the input array, the array is assumed to
    // have values in {0, 1, 2}
    static void sort012(int a[], int arr_size)
    {
        int lo = 0;
        int hi = arr_size - 1;
        int mid = 0,temp=0;
        while (mid <= hi)
        {
            switch (a[mid])
            {
                case 0:
                {
                    temp = a[lo];
                    a[lo] = a[mid];
                    a[mid] = temp;
                    lo++;
                    mid++;
                    break;
                }
                case 1:
                    mid++;
                    break;
                case 2:
                {
                    temp = a[mid];
                    a[mid] = a[hi];
                    a[hi] = temp;
                    hi--;
                    break;
                }
            }
        }
    }
}
```

```
        }
    }
}

/* Utility function to print array arr[] */
static void printArray(int arr[], int arr_size)
{
    int i;
    for (i = 0; i < arr_size; i++)
        System.out.print(arr[i]+" ");
    System.out.println("");
}

/*Driver function to check for above functions*/
public static void main (String[] args)
{
    int arr[] = {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1};
    int arr_size = arr.length;
    sort012(arr, arr_size);
    System.out.println("Array after segregation ");
    printArray(arr, arr_size);
}
}

/*This code is contributed by Devesh Agrawal*/
```

Python

```
# Python program to sort an array with 0,1 and 2 in a single pass

# Function to sort array
def sort012( a, arr_size):
    lo = 0
    hi = arr_size - 1
    mid = 0
    while mid <= hi:
        if a[mid] == 0:
            a[lo],a[mid] = a[mid],a[lo]
            lo = lo + 1
            mid = mid + 1
        elif a[mid] == 1:
            mid = mid + 1
        else:
            a[mid],a[hi] = a[hi],a[mid]
            hi = hi - 1
    return a

# Function to print array
```

```
def printArray( a):
    for k in a:
        print k,
    print

# Driver Program
arr = [0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1]
arr_size = len(arr)
arr = sort012( arr, arr_size)
print "Array aftter segregation :",
printArray(arr)

# Contributed by Harshit Agrawal
```

C#

```
// C# program to sort an
// array of 0, 1 and 2
using System;

class GFG
{
    // Sort the input array, the array is assumed to
    // have values in {0, 1, 2}
    static void sort012(int []a, int arr_size)
    {
        int lo = 0;
        int hi = arr_size - 1;
        int mid = 0,temp=0;

        while (mid <= hi)
        {
            switch (a[mid])
            {
                case 0:
                {
                    temp = a[lo];
                    a[lo] = a[mid];
                    a[mid] = temp;
                    lo++;
                    mid++;
                    break;
                }
                case 1:
                    mid++;
                    break;
                case 2:

```

```
{  
    temp = a[mid];  
    a[mid] = a[hi];  
    a[hi] = temp;  
    hi--;  
    break;  
}  
}  
}  
  
/* Utility function to print array arr[] */  
static void printArray(int []arr, int arr_size)  
{  
    int i;  
  
    for (i = 0; i < arr_size; i++)  
        Console.Write(arr[i] + " ");  
    Console.WriteLine("");  
}  
  
/*Driver function to check for above functions*/  
public static void Main ()  
{  
    int []arr = {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1};  
    int arr_size = arr.Length;  
    sort012(arr, arr_size);  
  
    Console.Write("Array after segregation ");  
  
    printArray(arr, arr_size);  
}  
}  
  
//This code is contributed by Sam007
```

Output:

```
array after segregation 0 0 0 0 1 1 1 1 1 2 2
```

Time Complexity: O(n)

The above code performs unnecessary swaps for inputs like 0 0 0 0 1 1 1 2 2 2 2 : lo=4 and mid=7 and hi=11. In present code: first 7 exchanged with 11 and hi become 10 and mid is still pointing to 7. again the same operation is done till mid <= hi. But it is really not required. We can change the swap function to do a check that the values being swapped are same or not, if not same, then only swap values. Thanks to Ankur Roy for suggesting this optimization. Source: <http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Sort/Flag/>

Source

<https://www.geeksforgeeks.org/sort-an-array-of-0s-1s-and-2s/>

Chapter 327

Sort an array of 0s, 1s and 2s (Simple Counting)

Sort an array of 0s, 1s and 2s (Simple Counting) - GeeksforGeeks

Given an array A[] consisting 0s, 1s and 2s, write a function that sorts A[]. The functions should put all 0s first, then all 1s and all 2s in last.

Examples:

Input : {0, 1, 2, 0, 1, 2}
Output : {0, 0, 1, 1, 2, 2}

Input : {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1}
Output : {0, 0, 0, 0, 1, 1, 1, 1, 2, 2}

Count the number of 0's, 1's and 2's. After Counting, put all 0's first, then 1's and lastly 2's in the array. We traverse the array two times. Time complexity will be O(n).

CPP

```
// Simple C++ program to sort an array of 0s
// 1s and 2s.
#include <iostream>
using namespace std;

void sort012(int* arr, int n)
{
    // Variables to maintain the count of 0's,
    // 1's and 2's in the array
    int count0 = 0, count1 = 0, count2 = 0;
    for (int i = 0; i < n; i++) {
```

```
        if (arr[i] == 0)
            count0++;
        if (arr[i] == 1)
            count1++;
        if (arr[i] == 2)
            count2++;
    }

    // Putting the 0's in the array in starting.
    for (int i = 0; i < count0; i++)
        arr[i] = 0;

    // Putting the 1's in the array after the 0's.
    for (int i = count0; i < (count0 + count1); i++)
        arr[i] = 1;

    // Putting the 2's in the array after the 1's
    for (int i = (count0 + count1); i < n; i++)
        arr[i] = 2;

    return;
}

// Prints the array
void printArray(int* arr, int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int arr[] = { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    sort012(arr, n);
    printArray(arr, n);
    return 0;
}
```

Java

```
// Simple Java program
// to sort an array of 0s
// 1s and 2s.
import java.util.*;
import java.lang.*;
```

```
public class GfG{  
  
    public static void sort012(int arr[], int n)  
    {  
        // Variables to maintain  
        // the count of 0's,  
        // 1's and 2's in the array  
        int count0 = 0, count1 = 0;  
        int count2 = 0;  
        for (int i = 0; i < n; i++) {  
            if (arr[i] == 0)  
                count0++;  
            if (arr[i] == 1)  
                count1++;  
            if (arr[i] == 2)  
                count2++;  
        }  
  
        // Putting the 0's in the  
        // array in starting.  
        for (int i = 0; i < count0; i++)  
            arr[i] = 0;  
  
        // Putting the 1's in the  
        // array after the 0's.  
        for (int i = count0; i <  
             (count0 + count1); i++)  
            arr[i] = 1;  
  
        // Putting the 2's in the  
        // array after the 1's  
        for (int i = (count0 + count1);  
             i < n; i++)  
            arr[i] = 2;  
  
        printArray(arr, n);  
    }  
  
    // Prints the array  
    public static void printArray(int arr[], int n)  
    {  
        for (int i = 0; i < n; i++)  
            System.out.print(arr[i] + " ");  
        System.out.println();  
    }  
  
    // Driver function
```

```
public static void main(String argc[])
{
    int arr[] = { 0, 1, 1, 0, 1, 2, 1,
                  2, 0, 0, 0, 1 };
    int n = 12;
    sort012(arr, n);
}
}

// This code is contributed by Sagar Shukla
```

Python3

```
# Python C++ program to sort an array of 0s
# 1s and 2s.
import math

def sort012(arr, n):

    # Variables to maintain the count of 0's,
    # 1's and 2's in the array
    count0 = 0
    count1 = 0
    count2 = 0
    for i in range(0,n):
        if (arr[i] == 0):
            count0=count0+1
        if (arr[i] == 1):
            count1=count1+1
        if (arr[i] == 2):
            count2=count2+1

    # Putting the 0's in the array in starting.
    for i in range(0,count0):
        arr[i] = 0

    # Putting the 1's in the array after the 0's.
    for i in range( count0, (count0 + count1)) :
        arr[i] = 1

    # Putting the 2's in the array after the 1's
    for i in range((count0 + count1),n) :
        arr[i] = 2

return
```

```
# Prints the array
def printArray( arr, n):

    for i in range(0,n):
        print( arr[i] , end=" ")
    print()

# Driver code
arr = [ 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 ]
n = len(arr)
sort012(arr, n)
printArray(arr, n)

# This code is contributed by Gitanjali.
```

C#

```
// Simple C# program
// to sort an array of 0s
// 1s and 2s.
using System;

public class GfG{

    public static void sort012(int []arr, int n)
    {
        // Variables to maintain
        // the count of 0's,
        // 1's and 2's in the array
        int count0 = 0, count1 = 0;
        int count2 = 0;
        for (int i = 0; i < n; i++) {
            if (arr[i] == 0)
                count0++;
            if (arr[i] == 1)
                count1++;
            if (arr[i] == 2)
                count2++;
        }

        // Putting the 0's in the
        // array in starting.
        for (int i = 0; i < count0; i++)
            arr[i] = 0;

        // Putting the 1's in the
```

```
// array after the 0's.
for (int i = count0; i <
     (count0 + count1); i++)
    arr[i] = 1;

// Putting the 2's in the
// array after the 1's
for (int i = (count0 + count1);
     i < n; i++)
    arr[i] = 2;

printArray(arr, n);
}

// Prints the array
public static void printArray(int []arr, int n)
{
    for (int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
    Console.WriteLine();
}

// Driver function
public static void Main()
{
    int []arr = { 0, 1, 1, 0, 1, 2, 1,
                 2, 0, 0, 0, 1 };
    int n = 12;
    sort012(arr, n);
}
}

// This code is contributed by vt_m
```

Output:

```
0 0 0 0 0 1 1 1 1 2 2
```

Problems with above solution.

- 1) It requires two traversals of array.
- 2) This solution may not work if values are a part of structure. For example, consider a situation where 0 represents Computer Science Stream, 1 represents Electronics and 2 represents Mechanical. We have a list of student objects (or structures) and we want to sort them. We cannot use above sort as we simply put 0s, 1s and 2s one by one.

Optimal Solution that handles above issues :

[Sort an array of 0s, 1s and 2s \(Dutch National Flag Algorithm\)](#)

Source

<https://www.geeksforgeeks.org/sort-array-0s-1s-2s-simple-counting/>

Chapter 328

Sort an array of large numbers

Sort an array of large numbers - GeeksforGeeks

Given an array of numbers where every number is represented as string. The numbers may be very large (may not fit in long long int), the task is to sort these numbers.

Examples:

```
Input : arr[] = {"5", "1237637463746732323", "12" };
Output : arr[] = {"5", "12", "1237637463746732323"};
```

```
Input : arr[] = {"50", "12", "12", "1"};
Output : arr[] = {"1", "12", "12", "50"};
```

The idea is to use [sort\(\)](#) function in C++ or [Arrays.sort](#) in Java.

C++

```
// C++ program to sort large numbers represented
// as strings.
#include<bits/stdc++.h>
using namespace std;

// Returns true if str1 is smaller than str2.
bool compareNumbers(string str1, string str2)
{
    // Calculate lengths of both string
    int n1 = str1.length(), n2 = str2.length();

    if (n1 < n2)
        return true;
    if (n2 < n1)
        return false;
```

```
// If lengths are same
for (int i=0; i<n1; i++)
{
    if (str1[i] < str2[i])
        return true;
    if (str1[i] > str2[i])
        return false;
}

return false;
}

// Function for sort an array of large numbers
// represented as strings
void sortLargeNumbers(string arr[], int n)
{
    sort(arr, arr+n, compareNumbers);
}

// Driver code
int main()
{
    string arr[] = {"5", "1237637463746732323",
                    "97987", "12" };
    int n = sizeof(arr)/sizeof(arr[0]);

    sortLargeNumbers(arr, n);

    for (int i=0; i<n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Java

```
// Java program to sort large numbers represented
// as strings.
import java.io.*;
import java.util.*;

class main
{
    // Function for sort an array of large numbers
    // represented as strings
    static void sortLargeNumbers(String arr[])
    {
```

```
// Refer below post for understanding below expression:  
// https://www.geeksforgeeks.org/lambda-expressions-java-8/  
Arrays.sort(arr, (left, right) ->  
{  
    /* If length of left != right, then return  
       the diff of the length else use compareTo  
       function to compare values.*/  
    if (left.length() != right.length())  
        return left.length() - right.length();  
    return left.compareTo(right);  
});  
}  
  
// Driver code  
public static void main(String args[])  
{  
    String arr[] = {"5", "1237637463746732323",  
                    "97987", "12" };  
    sortLargeNumbers(arr);  
    for (String s : arr)  
        System.out.print(s + " ");  
}  
}
```

Output:

```
5 12 97987 1237637463746732323
```

Time complexity : $O(k * n \log n)$ where k is length of the longest number. Here assumption is that the sort() function uses a $O(n \log n)$ sorting algorithm.

Similar Post :

[Sorting Big Integers](#)

Source

<https://www.geeksforgeeks.org/sort-array-large-numbers/>

Chapter 329

Sort an array of strings according to string lengths

Sort an array of strings according to string lengths - GeeksforGeeks

We are given an array of strings, we need to sort the array in increasing order of string lengths.

Examples:

```
Input : {"GeeksforGeeeks", "I", "from", "am"}  
Output : I am from GeeksforGeeks
```

```
Input : {"You", "are", "beautiful", "looking"}  
Output : You are looking beautiful
```

A **simple solution** is to write our own sort function that compares string lengths to decide which string should come first. Below is the C++ implementation that uses [Insertion Sort](#) to sort the array.

```
// C++ program to sort an Array of  
// Strings according to their lengths  
#include<iostream>  
using namespace std;  
  
// Function to print the sorted array of string  
void printArraystring(string,int);  
  
// Function to Sort the array of string  
// according to lengths. This function  
// implements Insertion Sort.
```

```
void sort(string s[], int n)
{
    for (int i=1 ;i<n; i++)
    {
        string temp = s[i];

        // Insert s[j] at its correct position
        int j = i - 1;
        while (j >= 0 && temp.length() < s[j].length())
        {
            s[j+1] = s[j];
            j--;
        }
        s[j+1] = temp;
    }
}

// Function to print the sorted array of string
void printArraystring(string str[], int n)
{
    for (int i=0; i<n; i++)
        cout << str[i] << " ";
}

// Driver function
int main()
{
    string arr[] = {"GeeksforGeeks", "I", "from", "am"};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Function to perform sorting
    sort(arr, n);

    // Calling the function to print result
    printArraystring(arr, n);

    return 0;
}
```

I am from GeeksforGeeks

A **better solution** is to use sort function provided by programming languages like C++, Java. These functions also allow us to write our own custom comparator. Below is C++ implementation that uses [C++ STL Sort function](#).

```
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to check the small string
bool compare(string &s1, string &s2)
{
    return s1.size() < s2.size();
}

// Function to print the sorted array of string
void printArraystring(string str[], int n)
{
    for (int i=0; i<n; i++)
        cout << str[i] << " ";
}

// Driver function
int main()
{
    string arr[] = {"GeeksforGeeks", "I", "from", "am"};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Function to perform sorting
    sort(arr, arr+n, compare);

    // Calling the function to print result
    printArraystring(arr, n);

    return 0;
}
```

I am from GeeksforGeeks

Source

<https://www.geeksforgeeks.org/sort-array-strings-according-string-lengths/>

Chapter 330

Sort an array using socket programming in C

Sort an array using socket programming in C - GeeksforGeeks

Given an array of unsorted positive integer, sort the given array using the [Socket programming](#).

Examples:

```
Input : 4 5 6 1 8 2 7 9 3 0
Output :0 1 2 3 4 5 6 7 8 9
```

```
Input : 9 8 1 4 0
Output : 0 1 4 8 9
```

Compile these files using **gcc** command (gcc client.c -o client and gcc server.c -o server). Run the program using **./server** and **./client** (Please note : First you should run server program which will be waiting for client's response and then client code).

In this program, client will take the input and send it to server and the server will sort the array using the bubble sort.

```
// Client code in C to sort the array
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

// Driver code
int main(int argc, char* argv[])
{
```

```
{  
    int sock;  
    struct sockaddr_in server;  
    int server_reply[10];  
    int number[10] = { 5, 4, 3, 8, 9, 1, 2, 0, 6 }, i, temp;  
  
    // Create socket  
    sock = socket(AF_INET, SOCK_STREAM, 0);  
    if (sock == -1) {  
        printf("Could not create socket");  
    }  
    puts("Socket created");  
  
    server.sin_addr.s_addr = inet_addr("127.0.0.1");  
    server.sin_family = AF_INET;  
    server.sin_port = htons(8880);  
  
    // Connect to remote server  
    if (connect(sock, (struct sockaddr*)&server, sizeof(server)) < 0) {  
        perror("connect failed. Error");  
        return 1;  
    }  
  
    puts("Connected\n");  
  
    if (send(sock, &number, 10 * sizeof(int), 0) < 0) {  
        puts("Send failed");  
        return 1;  
    }  
  
    // Receive a reply from the server  
    if (recv(sock, &server_reply, 10 * sizeof(int), 0) < 0) {  
        puts("recv failed");  
        return 0;  
    }  
  
    puts("Server reply :\n");  
    for (i = 0; i < 10; i++) {  
        printf("%d\n", server_reply[i]);  
    }  
  
    // close the socket  
    close(sock);  
    return 0;  
}
```

Note : Save above file as client.c

```
// Server code in C to sort the array
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

void bubble_sort(int[], int);

// Driver code
int main(int argc, char* argv[])
{
    int socket_desc, client_sock, c, read_size;
    struct sockaddr_in server, client;
    int message[10], i;

    // Create socket
    socket_desc = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_desc == -1) {
        printf("Could not create socket");
    }
    puts("Socket created");

    // Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(8880);

    // Bind the socket
    if (bind(socket_desc, (struct sockaddr*)&server, sizeof(server)) < 0) {

        // print the error message
        perror("bind failed. Error");
        return 1;
    }
    puts("bind done");

    // Listen to the socket
    listen(socket_desc, 3);

    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);

    // accept connection from an incoming client
    client_sock = accept(socket_desc, (struct sockaddr*)&client, (socklen_t*)&c);

    if (client_sock < 0) {
        perror("accept failed");
    }
}
```

```
        return 1;
    }

    puts("Connection accepted");

    // Receive a message from client
    while ((read_size = recv(client_sock, &message, 10 * sizeof(int), 0)) > 0) {

        bubble_sort(message, 10);

        write(client_sock, &message, 10 * sizeof(int));
    }

    if (read_size == 0) {
        puts("Client disconnected");
    }
    else if (read_size == -1) {
        perror("recv failed");
    }

    return 0;
}

// Function to sort the array
void bubble_sort(int list[], int n)
{
    int c, d, t;

    for (c = 0; c < (n - 1); c++) {
        for (d = 0; d < n - c - 1; d++) {
            if (list[d] > list[d + 1]) {

                /* Swapping */
                t = list[d];
                list[d] = list[d + 1];
                list[d + 1] = t;
            }
        }
    }
}
```

Note : Save above file as server.c

Output:

```
0 1 2 3 4 5 6 7 8 9
```

Source

<https://www.geeksforgeeks.org/sort-array-using-socket-programming/>

Chapter 331

Sort an array when two halves are sorted

Sort an array when two halves are sorted - GeeksforGeeks

Given an integer array of which both first half and second half are sorted. Task is to merge two sorted halves of array into single sorted array.

Examples:

Input : A[] = { 2, 3, 8, -1, 7, 10 }
Output : -1, 2, 3, 7, 8, 10

Input : A[] = {-4, 6, 9, -1, 3 }
Output : -4, -1, 3, 6, 9

A **Simple Solution** is to sort the array.

Below is the implementation of above approach :

C++

```
// C++ program to Merge two sorted halves of
// array Into Single Sorted Array
#include <bits/stdc++.h>
using namespace std;

void mergeTwoHalf(int A[], int n)
{
    // Sort the given array using sort STL
    sort(A, A + n);
}

// Driver program to test above function
```

```
int main()
{
    int A[] = { 2, 3, 8, -1, 7, 10 };
    int n = sizeof(A) / sizeof(A[0]);
    mergeTwoHalf(A, n);

    // Print sorted Array
    for (int i = 0; i < n; i++)
        cout << A[i] << " ";
    return 0;
}
```

Java

```
// Java program to Merge two sorted halves of
// array Into Single Sorted Array
import java.io.*;
import java.util.*;

class GFG {

    static void mergeTwoHalf(int[] A, int n)
    {
        // Sort the given array using sort STL
        Arrays.sort(A);
    }

    // Driver program to test above function
    static public void main(String[] args)
    {
        int[] A = { 2, 3, 8, -1, 7, 10 };
        int n = A.length;
        mergeTwoHalf(A, n);

        // Print sorted Array
        for (int i = 0; i < n; i++)
            System.out.print(A[i] + " ");
    }
}

// This code is contributed by vt_m .
```

C#

```
// C# program to Merge two sorted halves of
// array Into Single Sorted Array
using System;
```

```
class GFG {

    static void mergeTwoHalf(int[] A, int n)
    {
        // Sort the given array using sort STL
        Array.Sort(A);
    }

    // Driver program to test above function
    static public void Main()
    {
        int[] A = {2, 3, 8, -1, 7, 10};
        int n = A.Length;
        mergeTwoHalf(A, n);

        // Print sorted Array
        for (int i = 0; i < n; i++)
            Console.Write(A[i] + " ");
    }
}

// This code is contributed by vt_m .
```

Output:

```
-1 2 3 7 8 10
```

Time Complexity O(nlogn) Sort Given array using quick sort or merge sort

An **efficient solution** is to use an auxiliary array one half. Now whole process is same as the Merge Function of [Merge sort](#).

Below is the implementation of above approach :

C++

```
// C++ program to Merge Two Sorted Halves Of
// Array Into Single Sorted Array
#include <bits/stdc++.h>
using namespace std;

// Merge two sorted halves of Array into single
// sorted array
void mergeTwoHalf(int A[], int n)
{
    int half_i = 0; // starting index of second half

    // Temp Array store sorted resultant array
```

```
int temp[n];

// First Find the point where array is divide
// into two half
for (int i = 0; i < n - 1; i++) {
    if (A[i] > A[i + 1]) {
        half_i = i + 1;
        break;
    }
}

// If Given array is all-ready sorted
if (half_i == 0)
    return;

// Merge two sorted arrays in single sorted array
int i = 0, j = half_i, k = 0;
while (i < half_i && j < n) {
    if (A[i] < A[j])
        temp[k++] = A[i++];
    else
        temp[k++] = A[j++];
}

// Copy the remaining elements of A[i to half_! ]
while (i < half_i)
    temp[k++] = A[i++];

// Copy the remaining elements of A[ half_! to n ]
while (j < n)
    temp[k++] = A[j++];

for (int i = 0; i < n; i++)
    A[i] = temp[i];
}

// Driver program to test above function
int main()
{
    int A[] = { 2, 3, 8, -1, 7, 10 };
    int n = sizeof(A) / sizeof(A[0]);
    mergeTwoHalf(A, n);

    // Print sorted Array
    for (int i = 0; i < n; i++)
        cout << A[i] << " ";
    return 0;
}
```

Java

```
// java program to Merge Two Sorted Halves Of
// Array Into Single Sorted Array
import java.io.*;

class GFG {

    // Merge two sorted halves of Array
    // into single sorted array
    static void mergeTwoHalf(int[] A, int n)
    {
        int half_i = 0; // starting index of second half
        int i;

        // Temp Array store sorted resultant array
        int[] temp = new int[n];

        // First Find the point where array is divide
        // into two half
        for (i = 0; i < n - 1; i++) {
            if (A[i] > A[i + 1]) {
                half_i = i + 1;
                break;
            }
        }

        // If Given array is all-ready sorted
        if (half_i == 0)
            return;

        // Merge two sorted arrays in single sorted array
        i = 0;
        int j = half_i;
        int k = 0;
        while (i < half_i && j < n) {
            if (A[i] < A[j])
                temp[k++] = A[i++];
            else
                temp[k++] = A[j++];
        }

        // Copy the remaining elements of A[i to half_! ]
        while (i < half_i)
            temp[k++] = A[i++];

        // Copy the remaining elements of A[ half_! to n ]
        while (j < n)
            temp[k++] = A[j++];
    }
}
```

```
temp[k++] = A[j++];

for (i = 0; i < n; i++)
    A[i] = temp[i];
}

// Driver program to test above function
static public void main(String[] args)
{
    int[] A = {2, 3, 8, -1, 7, 10};
    int n = A.length;
    mergeTwoHalf(A, n);

    // Print sorted Array
    for (int i = 0; i < n; i++)
        System.out.print(A[i] + " ");
}
}

// This code is contributed by vt_m .
```

C#

```
// C# program to Merge Two Sorted Halves Of
// Array Into Single Sorted Array
using System;

class GFG {

    // Merge two sorted halves of Array
    // into single sorted array
    static void mergeTwoHalf(int[] A, int n)
    {
        int half_i = 0; // starting index of second half
        int i;

        // Temp Array store sorted resultant array
        int[] temp = new int[n];

        // First Find the point where array is divide
        // into two half
        for (i = 0; i < n - 1; i++) {
            if (A[i] > A[i + 1]) {
                half_i = i + 1;
                break;
            }
        }
    }
}
```

```
// If Given array is all-ready sorted
if (half_i == 0)
    return;

// Merge two sorted arrays in single sorted array
i = 0;
int j = half_i;
int k = 0;
while (i < half_i && j < n) {
    if (A[i] < A[j])
        temp[k++] = A[i++];
    else
        temp[k++] = A[j++];
}

// Copy the remaining elements of A[i to half_! ]
while (i < half_i)
    temp[k++] = A[i++];

// Copy the remaining elements of A[ half_! to n ]
while (j < n)
    temp[k++] = A[j++];

for (i = 0; i < n; i++)
    A[i] = temp[i];
}

// Driver program to test above function
static public void Main()
{
    int[] A = { 2, 3, 8, -1, 7, 10 };
    int n = A.Length;
    mergeTwoHalf(A, n);

    // Print sorted Array
    for (int i = 0; i < n; i++)
        Console.Write(A[i] + " ");
}
}

// This code is contributed by vt_m .
```

Output:

-1 2 3 7 8 10

Time Complexity : O(n)

Reference : <https://www.careercup.com/question?id=8412257>

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/sort-array-two-halves-sorted/>

Chapter 332

Sort an array which contain 1 to n values

Sort an array which contain 1 to n values - GeeksforGeeks

You have given an array which contain 1 to n element, your task is to sort this array in an efficient way and without replace with 1 to n numbers.

Examples :

```
Input : arr[] = {10, 7, 9, 2, 8,
                 3, 5, 4, 6, 1};
Output : 1 2 3 4 5 6 7 8 9 10
```

Native approach :

Sort this array with the use of any type of sorting method. it takes $O(n \log n)$ minimum time.

Efficient approach :

Replace every element with its position. it takes $O(n)$ efficient time and give you the sorted array. Let's understand this approach with the code below.

C++

```
// Efficient C++ program to sort an array of
// numbers in range from 1 to n.
#include <iostream>

using namespace std;

// function for sort array
void sortit(int arr[], int n)
{
```

```
for (int i = 0; i < n; i++) {  
    arr[i]=i+1;  
}  
}  
  
// Driver code  
int main()  
{  
    int arr[] = { 10, 7, 9, 2, 8, 3, 5, 4, 6, 1 };  
    int n = sizeof(arr) / sizeof(arr[0]);  
  
    // for sort an array  
    sortit(arr, n);  
  
    // for print all the element in sorted way  
    for (int i = 0; i < n; i++)  
        cout << arr[i] << " ";  
}
```

Java

```
// Efficient Java program to sort an  
// array of numbers in range from 1  
// to n.  
import java.io.*;  
import java.util.*;  
  
public class GFG {  
  
    // function for sort array  
    static void sortit(int []arr, int n)  
    {  
        for (int i = 0; i < n; i++)  
        {  
            arr[i]=i+1;  
  
        }  
    }  
  
    // Driver code  
    public static void main(String args[])  
    {  
        int []arr = {10, 7, 9, 2, 8,  
                    3, 5, 4, 6, 1};  
        int n = arr.length;
```

```
// for sort an array
sortit(arr, n);

// for print all the
// element in sorted way
for (int i = 0; i < n; i++)
    System.out.print(arr[i] + " ");
}

}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

C#

```
// Efficient C# program to sort an array of
// numbers in range from 1 to n.
using System;
using System.Collections.Generic;

class GFG {

    // function for sort array
    static void sortit(int []arr, int n)
    {
        for (int i = 0; i < n; i++)
        {

            arr[i]=i+1;
        }
    }

    // Driver code
    public static void Main()
    {
        int []arr = {10, 7, 9, 2, 8,
                    3, 5, 4, 6, 1};
        int n = arr.Length;

        // for sort an array
        sortit(arr, n);

        // for print all the
        // element in sorted way
        for (int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");
    }
}
```

```
}
```

```
// This code is contributed by
// Manish Shaw (manishshaw1)
```

PHP

```
<?php
// Efficient PHP program to sort an
// array of numbers in range from 1 to n.

// function for sort array
function sortit(&$arr, $n)
{
    for ($i = 0; $i < $n; $i++)
    {
        $arr[$i]=$i+1;
    }
}

// Driver code
$arr = array(10, 7, 9, 2, 8,
            3, 5, 4, 6, 1);
$n = count($arr);

// for sort an array
sortit($arr, $n);

// for print all the
// element in sorted way
for ($i = 0; $i < $n; $i++)
    echo $arr[$i]." ";

//This code is contributed by Manish Shaw
//(manishshaw1)
?>
```

Output :

```
1 2 3 4 5 6 7 8 9 10
```

Improved By : [manishshaw1](#), [_Keep_Silence_](#)

Source

<https://www.geeksforgeeks.org/sort-array-contains-1-n-values/>

Chapter 333

Sort an array with swapping only with a special element is allowed

Sort an array with swapping only with a special element is allowed - GeeksforGeeks

Given an array of length $n + 1$, containing elements 1 through n and a space, Requires the use of a given swap (index i, index j) function to sort the array, You can only swap the gap and a number, in the end, put the gap at the end.

There will be a number 999 in array as gap or space.

Examples:

```
Input : arr = {1, 5, 4, 999, 3, 2}
Output : arr = {1, 2, 3, 4, 5, 999}
We need to sort only by moving
```

```
Input : arr = {1, 5, 4, 3, 2, 8, 7, 999, 6}
Output : arr = {1, 2, 3, 4, 5, 6, 7, 8, 999}
```

We follow a recursive approach to solve this problem. As we can only swap numbers with the space. First of all we find the index of space. If the index is start of array, then move this space to the second last index by swapping with each number in its right.

If space is neither start of array nor last element of array and element before it greater than the element next to space then do following.

Step 1: Swap space and element next to space

In case of {3, 999, 2} make it {3, 2, 999}

Step 2 : Swap space and greater element

eg-convert {3, 2, 999} to {999, 2, 3}

Otherwise, elements next to index are sorted and swap it with the previous element. Again call the sort function with size of array decreased by 1 and index of space – 1 as we will get one sorted element each time.

```
// CPP program to sort an array by moving one
// space around.
#include <bits/stdc++.h>
using namespace std;

// n is total number of elements.
// index is index of 999 or space.
// k is number of elements yet to be sorted.
void sortRec(int arr[], int index, int k, int n)
{
    // print the sorted array when loop reaches
    // the base case
    if (k == 0) {
        for (int i = 1; i < n; i++)
            cout << arr[i] << " ";
        cout << 999;
        return;
    }

    // else if k>0 and space is at 0th index swap
    // each number with space and store index at
    // second last index
    else if (k > 0 && index == 0) {
        index = n - 2;
        for (int i = 1; i <= index; i++) {
            arr[i - 1] = arr[i];
        }
        arr[index] = 999;
    }

    // if space is neither start of array nor last
    // element of array and element before it greater
    // than/ the element next to space
    if (index - 1 >= 0 && index + 1 < n &&
        arr[index - 1] > arr[index + 1]) {

        // first swap space and element next to space
        // in case of {3, 999, 2} make it {3, 2, 999}
        swap(arr[index], arr[index + 1]);

        // then swap space and greater element
        // convert {3, 2, 999} to {999, 2, 3}
        swap(arr[index - 1], arr[index + 1]);
    }
}
```

```
    else
        swap(arr[index], arr[index - 1]);

    sortRec(arr, index - 1, k - 1, n);
}

// Wrapper over sortRec.
void sort(int arr[], int n)
{
    // Find index of space (or 999)
    int index = -1;
    for (int i = 0; i < n; i++) {
        if (arr[i] == 999) {
            index = i;
            break;
        }
    }

    // Invalid input
    if (index == -1)
        return;

    sortRec(arr, index, n, n);
}

// driver program
int main()
{
    int arr[] = { 3, 2, 999, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    sort(arr, n);
    return 0;
}
```

Output:

1 2 3 999

Time Complexity- $O(n^2)$

Source

<https://www.geeksforgeeks.org/sort-array-swapping-special-element-allowed/>

Chapter 334

Sort array after converting elements to their squares

Sort array after converting elements to their squares - GeeksforGeeks

Given a array of both positive and negative integers ‘arr[]’ which are sorted. Task is to sort square of the numbers of the Array.

Examples:

Input : arr[] = {-6, -3, -1, 2, 4, 5}
Output : 1, 4, 9, 16, 25, 36

Input : arr[] = {-5, -4, -2, 0, 1}
Output : 0, 1, 4, 16, 25

Simple solution is to first convert each array elements into its square and than apply any “O(nlogn)” sorting algorithm to sort the array elements.

Below is the implementation of above idea

C++

```
// C++ program to Sort square of the numbers
// of the array
#include<bits/stdc++.h>
using namespace std;

// Function to sort an square array
void sortSquares(int arr[], int n)
{
    // First convert each array elements
    // into its square
    for (int i = 0 ; i < n ; i++)
        arr[i] = arr[i] * arr[i];
```

```
arr[i] = arr[i] * arr[i];

// Sort an array using "sort STL function "
sort(arr, arr+n);
}

// Driver program to test above function
int main()
{
    int arr[] = { -6 , -3 , -1 , 2 , 4 , 5 };
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Before sort " << endl;
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    sortSquares(arr, n);

    cout << "\nAfter Sort " << endl;
    for (int i = 0 ; i < n ; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Java

```
// Java program to Sort square of the numbers
// of the array
import java.util.*;
import java.io.*;

class GFG
{
    // Function to sort an square array
    public static void sortSquares(int arr[])
    {
        int n = arr.length;

        // First convert each array elements
        // into its square
        for (int i = 0 ; i < n ; i++)
            arr[i] = arr[i] * arr[i];

        // Sort an array using "inbuild sort function"
        // in Arrays class.
        Arrays.sort(arr);
    }
}
```

```
// Driver program to test above function
public static void main (String[] args)
{
    int arr[] = { -6 , -3 , -1 , 2 , 4 , 5 };
    int n = arr.length;

    System.out.println("Before sort ");
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");

    sortSquares(arr);
    System.out.println("");
    System.out.println("After Sort ");
    for (int i = 0 ; i < n ; i++)
        System.out.print(arr[i] + " ");

}
```

Python3

```
# Python program to Sort square
# of the numbers of the array

# Function to sort an square array
def sortSquare(arr, n):

    # First convert each array
    # elements into its square
    for i in range(n):
        arr[i]= arr[i] * arr[i]
    arr.sort()

# Driver code
arr = [-6, -3, -1, 2, 4, 5]
n = len(arr)

print("Before sort")
for i in range(n):
    print(arr[i], end= " ")

print("\n")

sortSquare(arr, n)

print("After sort")
for i in range(n):
    print(arr[i], end = " ")
```

```
# This code is contributed by
# Shrikant13

C#

// C# program to Sort square
// of the numbers of the array
using System;

class GFG
{

    // Function to sort
    // an square array
    public static void sortSquares(int []arr)
    {
        int n = arr.Length;

        // First convert each array
        // elements into its square
        for (int i = 0 ; i < n ; i++)
            arr[i] = arr[i] * arr[i];

        // Sort an array using
        // "inbuild sort function"
        // in Arrays class.
        Array.Sort(arr);
    }

    // Driver Code
    public static void Main ()
    {
        int []arr = {-6, -3, -1,
                    2, 4, 5 };
        int n = arr.Length;

        Console.WriteLine("Before sort ");
        for (int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");

        sortSquares(arr);
        Console.WriteLine("");
        Console.WriteLine("After Sort ");

        for (int i = 0 ; i < n ; i++)
            Console.Write(arr[i] + " ");
    }
}
```

```
    }
}

// This code is contributed by anuj_67.
```

Output:

```
Before sort
-6 -3 -1 2 4 5
After Sort
1 4 9 16 25 36
```

Time complexity: $O(n \log n)$

Efficient solution is based on the fact that given array is already sorted. We do following two steps.

1. Divide the array into two part “Negative and positive “.
2. Use [merge function](#)to merge two sorted arrays into a single sorted array.

Below is the implementation of above idea.

C++

```
// C++ program to Sort square of the numbers of the array
#include<bits/stdc++.h>
using namespace std;

// function to sort array after doing squares of elements
void sortSquares(int arr[], int n)
{
    // first dived array into part negative and positive
    int K = 0;
    for (K = 0 ; K < n; K++)
        if (arr[K] >= 0 )
            break;

    // Now do the same process that we learn
    // in merge sort to merge to two sorted array
    // here both two half are sorted and we traverse
    // first half in reverse meaner because
    // first half contain negative element
    int i = K-1; // Initial index of first half
    int j = K; // Initial index of second half
    int ind = 0; // Initial index of temp array
```

```
// store sorted array
int temp[n];
while (i >= 0 && j < n)
{
    if (arr[i] * arr[i] < arr[j] * arr[j])
    {
        temp[ind] = arr[i] * arr[i];
        i--;
    }
    else
    {
        temp[ind] = arr[j] * arr[j];
        j++;
    }
    ind++;
}

/* Copy the remaining elements of first half */
while (i >= 0)
{
    temp[ind] = arr[i] * arr[i];
    i--;
    ind++;
}

/* Copy the remaining elements of second half */
while (j < n)
{
    temp[ind] = arr[j] * arr[j];
    j++;
    ind++;
}

// copy 'temp' array into original array
for (int i = 0 ; i < n; i++)
    arr[i] = temp[i];
}

// Driver program to test above function
int main()
{
    int arr[] = { -6 , -3 , -1 , 2 , 4 , 5 };
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Before sort " << endl;
    for (int i = 0; i < n; i++)
        cout << arr[i] << " " ;
```

```
sortSquares(arr, n);

cout << "\nAfter Sort " << endl;
for (int i = 0 ; i < n ; i++)
    cout << arr[i] << " " ;

return 0;
}
```

Java

```
// Java program to Sort square of the numbers
// of the array
import java.util.*;
import java.io.*;

class GFG
{
    // Function to sort an square array
    public static void sortSquares(int arr[])
    {
        int n = arr.length;
        // first dived array into part negative and positive
        int k;
        for(k = 0; k < n; k++)
        {
            if(arr[k] >= 0)
                break;
        }

        // Now do the same process that we learn
        // in merge sort to merge to two sorted array
        // here both two half are sorted and we traverse
        // first half in reverse meaner because
        // first half contain negative element
        int i = k-1; // Initial index of first half
        int j = k; // Initial index of second half
        int ind = 0; // Initial index of temp array

        int[] temp = new int[n];
        while(i >= 0 && j < n)
        {
            if(arr[i] * arr[i] < arr[j] * arr[j])
            {
                temp[ind] = arr[i] * arr[i];
                i--;
            }
            else{
```

```
temp[ind] = arr[j] * arr[j];
j++;

}
ind++;
}

while(i >= 0)
{
    temp[ind++] = arr[i] * arr[i];
    i--;
}
while(j < n)
{
    temp[ind++] = arr[j] * arr[j];
    j++;
}

// copy 'temp' array into original array
for (int x = 0 ; x < n; x++)
    arr[x] = temp[x];
}

// Driver program to test above function
public static void main (String[] args)
{
    int arr[] = { -6 , -3 , -1 , 2 , 4 , 5 };
    int n = arr.length;

    System.out.println("Before sort ");
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");

    sortSquares(arr);
    System.out.println("");
    System.out.println("After Sort ");
    for (int i = 0 ; i < n ; i++)
        System.out.print(arr[i] + " ");

    }
}
```

```
Before sort
-6 -3 -1 2 4 5
After Sort
1 4 9 16 25 36
```

Time complexity: $O(n)$

space complexity: $O(n)$

Improved By : [shrikanth13](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/sort-array-converting-elements-squares/>

Chapter 335

Sort elements by frequency Set 1

Sort elements by frequency Set 1 - GeeksforGeeks

Print the elements of an array in the decreasing frequency if 2 numbers have same frequency then print the one which came first.

Examples:

Input: arr[] = {2, 5, 2, 8, 5, 6, 8, 8}
Output: arr[] = {8, 8, 8, 2, 2, 5, 5, 6}

Input: arr[] = {2, 5, 2, 6, -1, 9999999, 5, 8, 8, 8}
Output: arr[] = {8, 8, 8, 2, 2, 5, 5, 6, -1, 9999999}

METHOD 1 (Use Sorting)

- 1) Use a sorting algorithm to sort the elements $O(n\log n)$
- 2) Scan the sorted array and construct a 2D array of element and count $O(n)$.
- 3) Sort the 2D array according to count $O(n\log n)$.

Example:

Input 2 5 2 8 5 6 8 8

After sorting we get
2 2 5 5 6 8 8 8

Now construct the 2D array as

```
2, 2  
5, 2  
6, 1  
8, 3
```

Sort by count

```
8, 3  
2, 2  
5, 2  
6, 1
```

How to maintain order of elements if frequency is same?

The above approach doesn't make sure order of elements if frequency is same. To handle this, we should use indexes in step 3, if two counts are same then we should first process(or print) the element with lower index. In step 1, we should store the indexes instead of elements.

Input 5 2 2 8 5 6 8 8

After sorting we get

```
Element 2 2 5 5 6 8 8 8  
Index   1 2 0 4 5 3 6 7
```

Now construct the 2D array as

Index, Count

```
1,      2  
0,      2  
5,      1  
3,      3
```

Sort by count (consider indexes in case of tie)

```
3, 3  
0, 2  
1, 2  
5, 1
```

Print the elements using indexes in the above 2D array.

Below is C++ implementation of above approach.

```
// Sort elements by frequency. If two elements have same  
// count, then put the elements that appears first  
#include<bits/stdc++.h>  
using namespace std;
```

```
// Used for sorting
struct ele
{
    int count, index, val;
};

// Used for sorting by value
bool mycomp(struct ele a, struct ele b) {
    return (a.val < b.val);
}

// Used for sorting by frequency. And if frequency is same,
// then by appearance
bool mycomp2(struct ele a, struct ele b) {
    if (a.count != b.count) return (a.count < b.count);
    else return a.index > b.index;
}

void sortByFrequency(int arr[], int n)
{
    struct ele element[n];
    for (int i = 0; i < n; i++)
    {
        element[i].index = i; /* Fill Indexes */
        element[i].count = 0; /* Initialize counts as 0 */
        element[i].val = arr[i]; /* Fill values in structure
                                   elements */
    }

    /* Sort the structure elements according to value,
       we used stable sort so relative order is maintained. */
    stable_sort(element, element+n, mycomp);

    /* initialize count of first element as 1 */
    element[0].count = 1;

    /* Count occurrences of remaining elements */
    for (int i = 1; i < n; i++)
    {
        if (element[i].val == element[i-1].val)
        {
            element[i].count += element[i-1].count+1;

            /* Set count of previous element as -1 , we are
               doing this because we'll again sort on the
               basis of counts (if counts are equal than on
               the basis of index)*/
            element[i-1].count = -1;
        }
    }
}
```

```
/* Retain the first index (Remember first index
   is always present in the first duplicate we
   used stable sort. */
element[i].index = element[i-1].index;
}

/* Else If previous element is not equal to current
   so set the count to 1 */
else element[i].count = 1;
}

/* Now we have counts and first index for each element so now
   sort on the basis of count and in case of tie use index
   to sort.*/
stable_sort(element, element+n, mycomp2);
for (int i = n-1, index=0; i >= 0; i--)
    if (element[i].count != -1)
        for (int j=0; j<element[i].count; j++)
            arr[index++] = element[i].val;
}

// Driver program
int main()
{
    int arr[] = {2, 5, 2, 6, -1, 9999999, 5, 8, 8, 8};
    int n = sizeof(arr)/sizeof(arr[0]);

    sortByFrequency(arr, n);

    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Output:

```
8 8 8 2 2 5 5 6 -1 9999999
```

Thanks to Gaurav Ahirwar for providing above implementation.

METHOD 2(Use BST and Sorting)

1. Insert elements in BST one by one and if an element is already present then increment the count of the node. Node of the Binary Search Tree (used in this approach) will be as follows.

```
struct tree
```

```
{  
    int element;  
    int first_index /*To handle ties in counts*/;  
    int count;  
}BST;
```

2.Store the first indexes and corresponding counts of BST in a 2D array.

3 Sort the 2D array according to counts (and use indexes in case of tie).

Time Complexity: $O(n\log n)$ if a [Self Balancing Binary Search Tree](#) is used. This is implemented in [Set 2](#).

METHOD 3(Use Hashing and Sorting)

Using a hashing mechanism, we can store the elements (also first index) and their counts in a hash. Finally, sort the hash elements according to their counts.

Set 2:

[Sort elements by frequency Set 2](#)

Improved By : [praveen jain](#)

Source

<https://www.geeksforgeeks.org/sort-elements-by-frequency/>

Chapter 336

Sort elements by frequency Set 2

Sort elements by frequency Set 2 - GeeksforGeeks

Given an array of integers, sort the array according to frequency of elements. For example, if the input array is {2, 3, 2, 4, 5, 12, 2, 3, 3, 3, 12}, then modify the array to {3, 3, 3, 3, 2, 2, 2, 12, 12, 4, 5}.

In the [previous post](#), we have discussed all methods for sorting according to frequency. In this post, method 2 is discussed in detail and C++ implementation for the same is provided.

Following is detailed algorithm.

- 1) Create a BST and while creating BST maintain the count i,e frequency of each coming element in same BST. This step may take $O(n\log n)$ time if a self balancing BST is used.
- 2) Do Inorder traversal of BST and store every element and count of each element in an auxiliary array. Let us call the auxiliary array as ‘count[]’. Note that every element of this array is element and frequency pair. This step takes $O(n)$ time.
- 3) Sort ‘count[]’ according to frequency of the elements. This step takes $O(n \log n)$ time if a $O(n\log n)$ sorting algorithm is used.
- 4) Traverse through the sorted array ‘count[]’. For each element x, print it ‘freq’ times where ‘freq’ is frequency of x. This step takes $O(n)$ time.

Overall time complexity of the algorithm can be minimum $O(n\log n)$ if we use a $O(n\log n)$ sorting algorithm and use a self balancing BST with $O(\log n)$ insert operation.

Following is C++ implementation of the above algorithm.

```
// Implementation of above algorithm in C++.  
#include <iostream>  
#include <stdlib.h>  
using namespace std;  
  
/* A BST node has data, freq, left and right pointers */  
struct BSTNode
```

```
{  
    struct BSTNode *left;  
    int data;  
    int freq;  
    struct BSTNode *right;  
};  
  
// A structure to store data and its frequency  
struct dataFreq  
{  
    int data;  
    int freq;  
};  
  
/* Function for qsort() implementation. Compare frequencies to  
   sort the array according to decreasing order of frequency */  
int compare(const void *a, const void *b)  
{  
    return ( (*(const dataFreq*)b).freq - (*(const dataFreq*)a).freq );  
}  
  
/* Helper function that allocates a new node with the given data,  
   frequency as 1 and NULL left and right pointers.*/  
BSTNode* newNode(int data)  
{  
    struct BSTNode* node = new BSTNode;  
    node->data = data;  
    node->left = NULL;  
    node->right = NULL;  
    node->freq = 1;  
    return (node);  
}  
  
// A utility function to insert a given key to BST. If element  
// is already present, then increases frequency  
BSTNode *insert(BSTNode *root, int data)  
{  
    if (root == NULL)  
        return newNode(data);  
    if (data == root->data) // If already present  
        root->freq += 1;  
    else if (data < root->data)  
        root->left = insert(root->left, data);  
    else  
        root->right = insert(root->right, data);  
    return root;  
}
```

```

// Function to copy elements and their frequencies to count[] .
void store(BSTNode *root, dataFreq count[], int *index)
{
    // Base Case
    if (root == NULL) return;

    // Recur for left subtree
    store(root->left, count, index);

    // Store item from root and increment index
    count[(*index)].freq = root->freq;
    count[(*index)].data = root->data;
    (*index)++;

    // Recur for right subtree
    store(root->right, count, index);
}

// The main function that takes an input array as an argument
// and sorts the array items according to frequency
void sortByFrequency(int arr[], int n)
{
    // Create an empty BST and insert all array items in BST
    struct BSTNode *root = NULL;
    for (int i = 0; i < n; ++i)
        root = insert(root, arr[i]);

    // Create an auxiliary array 'count[]' to store data and
    // frequency pairs. The maximum size of this array would
    // be n when all elements are different
    dataFreq count[n];
    int index = 0;
    store(root, count, &index);

    // Sort the count[] array according to frequency (or count)
    qsort(count, index, sizeof(count[0]), compare);

    // Finally, traverse the sorted count[] array and copy the
    // i'th item 'freq' times to original array 'arr[]'
    int j = 0;
    for (int i = 0; i < index; i++)
    {
        for (int freq = count[i].freq; freq > 0; freq--)
            arr[j++] = count[i].data;
    }
}

// A utility function to print an array of size n

```

```
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {2, 3, 2, 4, 5, 12, 2, 3, 3, 3, 12};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortByFrequency(arr, n);
    printArray(arr, n);
    return 0;
}
```

Output:

```
3 3 3 3 2 2 2 12 12 5 4
```

Exercise:

The above implementation doesn't guarantee original order of elements with same frequency (for example, 4 comes before 5 in input, but 4 comes after 5 in output). Extend the implementation to maintain original order. For example, if two elements have same frequency then print the one which came 1st in input array.

This article is compiled by **Chandra Prakash**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/sort-elements-by-frequency-set-2/>

Chapter 337

Sort elements by frequency Set 4 (Efficient approach using hash)

Sort elements by frequency Set 4 (Efficient approach using hash) - GeeksforGeeks

Print the elements of an array in the decreasing frequency if 2 numbers have same frequency then print the one which came first.

Examples:

```
Input : arr[] = {2, 5, 2, 8, 5, 6, 8, 8}
Output : arr[] = {8, 8, 8, 2, 2, 5, 5, 6}
```

```
Input : arr[] = {2, 5, 2, 6, -1, 9999999, 5, 8, 8, 8}
Output : arr[] = {8, 8, 8, 2, 2, 5, 5, 6, -1, 9999999}
```

We have discussed different approaches in below posts :

[Sort elements by frequency Set 1](#)

[Sort elements by frequency Set 2](#)

[Sorting Array Elements By Frequency Set 3 \(Using STL\)](#)

All of the above approaches work in $O(n \log n)$ time where n is total number of elements. In this post, a new approach is discussed that works in $O(n + m \log m)$ time where n is total number of elements and m is total number of distinct elements.

The idea is to use [hashing](#).

1. We insert all elements and their counts into a hash. This step takes $O(n)$ time where n is number of elements.

2. We copy contents of hash to an array (or vector) and sort them by counts. This step takes $O(m \log m)$ time where m is total number of distinct elements.
3. For maintaining the order of elements if the frequency is same, we use another hash which has the key as elements of the array and value as the index. If the frequency is same for two elements then sort elements according to the index.

The C++ implementation is given below.

```
// Sort elements by frequency. If two elements have same
// count, then put the elements that appears first
#include <bits/stdc++.h>
using namespace std;

// Map m2 keeps track of indexes of elements in array
unordered_map<int, int> m2;

// Used for sorting by frequency. And if frequency is same,
// then by appearance
bool sortByVal(const pair<int, int>& a, const pair<int, int>& b)
{
    // If frequency is same then sort by index
    if (a.second == b.second)
        return m2[a.first] < m2[b.first];

    return a.second > b.second;
}

// function to sort elements by frequency
void sortByFreq(int a[], int n)
{
    unordered_map<int, int> m;
    vector<pair<int, int> > v;

    for (int i = 0; i < n; ++i) {

        // Map m is used to keep track of count
        // of elements in array
        m[a[i]]++;

        // Update the value of map m2 only once
        if (m2[a[i]] == 0)
            m2[a[i]] = i + 1;
    }

    // Copy map to vector
    copy(m.begin(), m.end(), back_inserter(v));
}
```

```
// Sort the element of array by frequency
sort(v.begin(), v.end(), sortByVal);

for (int i = 0; i < v.size(); ++i)
    for (int j = 0; j < v[i].second; ++j)
        cout << v[i].first << " ";
}

// Driver program
int main()
{
    int a[] = { 2, 5, 2, 6, -1, 9999999, 5, 8, 8, 8 };
    int n = sizeof(a) / sizeof(a[0]);

    sortByFreq(a, n);

    return 0;
}
```

Output:

```
8 8 8 2 2 5 5 6 -1 9999999
```

Time Complexity : $O(n) + O(m \log m)$ where n is total number of elements and m is total number of distinct elements

Source

<https://www.geeksforgeeks.org/sort-elements-frequency-set-4-efficient-approach-using-hash/>

Chapter 338

Sort elements on the basis of number of factors

Sort elements on the basis of number of factors - GeeksforGeeks

Given an array of positive integers. Sort the given array in decreasing order of number of factors of each element, i.e., element having the highest number of factors should be the first to be displayed and the number having least number of factors should be the last one. Two elements with equal number of factors should be in the same order as in the original array.

Examples:

```
Input : {5, 11, 10, 20, 9, 16, 23}
Output : 20 16 10 9 5 11 23
Number of distinct factors:
For 20 = 6, 16 = 5, 10 = 4, 9 = 3
and for 5, 11, 23 = 2 (same number of factors
therefore sorted in increasing order of index)
```

```
Input : {104, 210, 315, 166, 441, 180}
Output : 180 210 315 441 104 166
```

The following steps sort numbers in decreasing order of count of factors.

1. Count distinct number of factors of each element. Refer [this](#).
2. You can use a structure for each element to store its original index and count of factors. Create an array of such structures to store such information for all the elements.
3. Sort this array of structures on the basis of the problem statement using any sorting algorithm.

4. Traverse this array of structures from the beginning and get the number from the original array with the help of the index stored in the structure of each element of the sorted array of structures.

C++

```
// C++ implementation to sort numbers on
// the basis of factors
#include <bits/stdc++.h>

using namespace std;

// structure of each element having its index
// in the input array and number of factors
struct element
{
    int index, no_of_fact;
};

// function to count factors for
// a given number n
int countFactors(int n)
{
    int count = 0;
    int sq = sqrt(n);

    // if the number is a perfect square
    if (sq * sq == n)
        count++;

    // count all other factors
    for (int i=1; i<sqrt(n); i++)
    {
        // if i is a factor then n/i will be
        // another factor. So increment by 2
        if (n % i == 0)
            count += 2;
    }

    return count;
}

// comparsion function for the elements
// of the structure
bool compare(struct element e1, struct element e2)
{
    // if two elements have the same number
    // of factors then sort them in increasing
```

```
// order of their index in the input array
if (e1.no_of_fact == e2.no_of_fact)
    return e1.index < e2.index;

// sort in decreasing order of number of factors
return e1.no_of_fact > e2.no_of_fact;
}

// function to print numbers after sorting them in
// decreasing order of number of factors
void printOnBasisOfFactors(int arr[], int n)
{
    struct element num[n];

    // for each element of input array create a
    // structure element to store its index and
    // factors count
    for (int i=0; i<n; i++)
    {
        num[i].index = i;
        num[i].no_of_fact = countFactors(arr[i]);
    }

    // sort the array of structures as defined
    sort(num, num+n, compare);

    // access index from the structure element and correponding
    // to that index access the element from arr
    for (int i=0; i<n; i++)
        cout << arr[num[i].index] << " ";
}

// Driver program to test above
int main()
{
    int arr[] = {5, 11, 10, 20, 9, 16, 23};
    int n = sizeof(arr) / sizeof(arr[0]);
    printOnBasisOfFactors(arr, n);
    return 0;
}
```

Java

```
// Java implementation to sort numbers on
// the basis of factors

import java.util.Arrays;
import java.util.Comparator;
```

```
class Element
{
    //each element having its index
    // in the input array and number of factors
    int index, no_of_fact;

    public Element(int i, int countFactors)
    {
        index = i;
        no_of_fact = countFactors;
    }

    // method to count factors for
    // a given number n
    static int countFactors(int n)
    {
        int count = 0;
        int sq = (int)Math.sqrt(n);

        // if the number is a perfect square
        if (sq * sq == n)
            count++;

        // count all other factors
        for (int i=1; i<Math.sqrt(n); i++)
        {
            // if i is a factor then n/i will be
            // another factor. So increment by 2
            if (n % i == 0)
                count += 2;
        }

        return count;
    }

    // function to print numbers after sorting them in
    // decreasing order of number of factors
    static void printOnBasisOfFactors(int arr[], int n)
    {
        Element num[] = new Element[n];

        // for each element of input array create a
        // structure element to store its index and
        // factors count
        for (int i=0; i<n; i++)
        {
            num[i] = new Element(i, countFactors(arr[i]));
        }
    }
}
```

```
}

// sort the array of structures as defined
Arrays.sort(num,new Comparator<Element>() {

    @Override
    // compare method for the elements
    // of the structure
    public int compare(Element e1, Element e2) {
        // if two elements have the same number
        // of factors then sort them in increasing
        // order of their index in the input array
        if (e1.no_of_fact == e2.no_of_fact)
            return e1.index < e2.index ? -1 : 1;

        // sort in decreasing order of number of factors
        return e1.no_of_fact > e2.no_of_fact ? -1 : 1;
    }

});

// access index from the structure element and correponding
// to that index access the element from arr
for (int i=0; i<n; i++)
    System.out.print(arr[num[i].index]+" ");

// Driver program to test above
public static void main(String[] args)
{
    int arr[] = {5, 11, 10, 20, 9, 16, 23};
    printOnBasisOfFactors(arr, arr.length);

}
}

// This code is contributed by Gaurav Miglani
```

Output:

20 16 10 9 5 11 23

Time Complexity: $O(n \sqrt{n})$

Source

<https://www.geeksforgeeks.org/sort-elements-basis-number-factors/>

Chapter 339

Sort even-placed elements in increasing and odd-placed in decreasing order

Sort even-placed elements in increasing and odd-placed in decreasing order - GeeksforGeeks

We are given an array of n distinct numbers, the task is to sort all even-placed numbers in increasing and odd-place numbers in decreasing order. The modified array should contain all sorted even-placed numbers followed by reverse sorted odd-placed numbers.

Note that the first element is considered as even because of its index 0.

Examples:

```
Input: arr[] = {0, 1, 2, 3, 4, 5, 6, 7}
Output: arr[] = {0, 2, 4, 6, 7, 5, 3, 1}
Even-place elements : 0, 2, 4, 6
Odd-place elements : 1, 3, 5, 7
Even-place elements in increasing order :
0, 2, 4, 6
Odd-Place elements in decreasing order :
7, 5, 3, 1
```

```
Input: arr[] = {3, 1, 2, 4, 5, 9, 13, 14, 12}
Output: {2, 3, 5, 12, 13, 14, 9, 4, 1}
Even-place elements : 3, 2, 5, 13, 12
Odd-place elements : 1, 4, 9, 14
Even-place elements in increasing order :
2, 3, 5, 12, 13
Odd-Place elements in decreasing order :
14, 9, 4, 1
```

The idea is simple, we create two auxiliary arrays evenArr[] and oddArr[] respectively. We traverse input array and put all even-placed elements in evenArr[] and odd placed elements in oddArr[]. Then we sort evenArr[] in ascending and oddArr[] in descending order. Finally copy evenArr[] and oddArr[] to get the required result.

```
// Program to separately sort even-placed and odd
// placed numbers and place them together in sorted
// array.
#include <bits/stdc++.h>
using namespace std;

void bitonicGenerator(int arr[], int n)
{
    // create evenArr[] and oddArr[]
    vector<int> evenArr;
    vector<int> oddArr;

    // Put elements in oddArr[] and evenArr[] as
    // per their position
    for (int i = 0; i < n; i++)
    {
        if (!(i % 2))
            evenArr.push_back(arr[i]);
        else
            oddArr.push_back(arr[i]);
    }

    // sort evenArr[] in ascending order
    // sort oddArr[] in descending order
    sort(evenArr.begin(), evenArr.end());
    sort(oddArr.begin(), oddArr.end(), greater<int>());

    int i = 0;
    for (int j=0; j<evenArr.size(); j++)
        arr[i++] = evenArr[j];
    for (int j=0; j<oddArr.size(); j++)
        arr[i++] = oddArr[j];
}

//Driver Program
int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int n = sizeof(arr)/sizeof(arr[0]);
    bitonicGenerator(arr, n);
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

}

Output:

1 2 3 6 8 9 7 5 4 0

Time Complexity : O(n Log n)

Space Complexity : O(n)

The above problem can also be solve without use of Auxiliary space. The idea is to swaps first half odd index positions with second half even index positions and then sort the first half array in increasing order and second half array in decreasing order. Thanks to **SWARUPANANDA DHUA** for suggesting this.

```
// Program to sort even-placed elements in increasing and
// odd-placed in decreasing order with constant space complexity

#include <bits/stdc++.h>
using namespace std;

void bitonicGenerator(int arr[], int n)
{
    // first odd index
    int i = 1;

    // last index
    int j = n-1;

    // if last index is odd
    if(j%2 != 0)
        // decrement j to even index
        j--;

    // swapping till half of array
    while(i < j)
    {
        swap(arr[i], arr[j]);
        i += 2;
        j -= 2;
    }

    // Sort first half in increasing
    sort(arr, arr + (n+1)/2);

    // Sort second half in decreasing
    sort(arr + (n+1)/2, arr + n, greater<int>());
}
```

```
}  
  
//Driver Program  
int main()  
{  
    int arr[] = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};  
    int n = sizeof(arr)/sizeof(arr[0]);  
    bitonicGenerator(arr, n);  
    for (int i=0; i<n; i++)  
        cout << arr[i] << " ";  
    return 0;  
}  
// This code is contributed by SWARUPANANDA DHUA
```

Output:

1 2 3 6 8 9 7 5 4 0

Time Complexity : $O(n \log n)$

Space Complexity : $O(1)$

Source

<https://www.geeksforgeeks.org/sort-even-placed-elements-increasing-odd-placed-decreasing-order/>

Chapter 340

Sort first half in ascending and second half in descending order

1

Sort first half in ascending and second half in descending order 1 - GeeksforGeeks

Given an array of integers, sort the first half of the array in ascending order and second half in descending order.

Examples:

Input : arr[] = {5, 2, 4, 7, 9, 3, 1, 6, 8}
Output : arr[] = {1, 2, 3, 4, 9, 8, 7, 6, 5}

Input : arr[] = {1, 2, 3, 4, 5, 6}
Output : arr[] = {1, 2, 3, 6, 5, 4}

Algorithm :

1. Sort the given array.
2. Run a loop upto half the length of the array and print the elements of the sorted array.
3. Run a loop from last index of the array to the middle of the array and print the elements in reverse order.

Below is the implementation for the same.

C++

```
// C++ program to print first half in
// ascending order and the second half
// in descending order
#include <bits/stdc++.h>
using namespace std;
```

```
// function to print half of the array in
// ascending order and the other half in
// descending order
void printOrder(int arr[], int n)
{
    // sorting the array
    sort(arr, arr + n);

    // printing first half in ascending
    // order
    for (int i = 0; i < n / 2; i++)
        cout << arr[i] << " ";

    // printing second half in descending
    // order
    for (int j = n - 1; j >= n / 2; j--)
        cout << arr[j] << " ";
}

// driver code
int main()
{
    int arr[] = { 5, 4, 6, 2, 1, 3, 8, 9, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printOrder(arr, n);

    return 0;
}
```

Java

```
// Java program to print first half in
// ascending order and the second half
// in descending order
import java.util.*;

class GFG
{
    // function to print half of the array in
    // ascending order and the other half in
    // descending order
    static void printOrder(int[] arr, int n)
    {
        // sorting the array
        Arrays.sort(arr);

        // printing first half in ascending
```

```
// order
for (int i = 0; i < n / 2; i++)
    System.out.print(arr[i]+" ");

// printing second half in descending
// order
for (int j = n - 1; j >= n / 2; j--)
    System.out.print(arr[j]+" ");

}

// Driver code
public static void main(String[] args)
{
    int[] arr = { 5, 4, 6, 2, 1, 3, 8, 9, 7 };
    int n = arr.length;
    printOrder(arr, n);

}
/*
 * This code is contributed by Mr. Somesh Awasthi */

```

Python

```
# Python program to print first half in
# ascending order and the second half
# in descending order

# function to print half of the array in
# ascending order and the other half in
# descending order
def printOrder(arr,n) :

    # sorting the array
    arr.sort()

    # printing first half in ascending
    # order
    i = 0
    while (i< n/ 2 ) :
        print arr[i],
        i = i + 1

    # printing second half in descending
    # order
    j = n - 1
    while j >= n / 2 :
```

```
print arr[j],  
j = j - 1  
  
# Driver code  
arr = [5, 4, 6, 2, 1, 3, 8, 9, 7]  
n = len(arr)  
printOrder(arr, n)  
  
# This code is contributed by Nikita Tiwari.
```

C#

```
// C# program to print first half in  
// ascending order and the second half  
// in descending order  
using System;  
  
class GFG {  
  
    // function to print half of the array in  
    // ascending order and the other half in  
    // descending order  
    static void printOrder(int[] arr, int n)  
    {  
  
        // sorting the array  
        Array.Sort(arr);  
  
        // printing first half in ascending  
        // order  
        for (int i = 0; i < n / 2; i++)  
            Console.Write(arr[i] + " ");  
  
        // printing second half in descending  
        // order  
        for (int j = n - 1; j >= n / 2; j--)  
            Console.Write(arr[j] + " ");  
    }  
  
    // Driver code  
    public static void Main()  
    {  
        int[] arr = { 5, 4, 6, 2, 1, 3, 8, 9, 7 };  
        int n = arr.Length;  
  
        printOrder(arr, n);  
    }  
}
```

```
// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to print first half in
// ascending order and the second half in
// in descending order

// function to print half of the array in
// ascending order and the other half in
// descending order
function printOrder($arr, $n)
{

    // sorting the array
    sort($arr);

    // printing first half
    // in ascending order
    for ($i = 0; $i < intval($n / 2); $i++)
        echo $arr[$i] . " ";

    // printing second half
    // in descending order
    for ($j = $n - 1; $j >= intval($n / 2); $j--)
        echo $arr[$j] . " ";
}

// Driver Code
$arr = array(5, 4, 6, 2, 1,
            3, 8, 9, 7);
$n = count($arr);
printOrder($arr, $n);

// This code is contributed by Sam007
?>
```

Output:

```
1 2 3 4 9 8 7 6 5
```

Alternate Solution :

Here, The elements in the 1st half of array will remain in 1st half but in ascending order

among the elements in the 1st half and those in 2nd half of array will remain in 2nd half but in descending order among the elements in the 2nd half.

1. Sort the 1st half of the array in ascending order just because only the elements in the 1st half of the input array needs to be sorted in ascending order (In this way the original elements in the 1st half of the array will remain in the 1st half but in ascending order).
2. Sort the 2nd half of the array in descending order just because only the elements in the 2nd half of the input array needs to be sorted in descending order (In this way the original elements in the 2nd half of the array will remain in the 1st half but in descending order).

```
// C++ program to print first half in
// ascending order and the second half
// in descending order
#include <bits/stdc++.h>
using namespace std;

// function to print half of the array in
// ascending order and the other half in
// descending order
void printOrder(int arr[], int n)
{
    // sorting the array
    sort(arr, arr + n);

    // printing first half in ascending
    // order
    for (int i = 0; i < n / 2; i++)
        cout << arr[i] << " ";

    // printing second half in descending
    // order
    for (int j = n - 1; j >= n / 2; j--)
        cout << arr[j] << " ";
}

// driver code
int main()
{
    int arr[] = { 5, 4, 6, 2, 1, 3, 8, -1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printOrder(arr, n);

    return 0;
}
```

Output:

-1 1 2 3 8 6 5 4

Sort first half in ascending and second half in descending order Set 2

Improved By : [vt_m](#), [Sam007](#), [techmanager2](#)

Source

<https://www.geeksforgeeks.org/sort-first-half-in-ascending-and-second-half-in-descending-order/>

Chapter 341

Sort first half in ascending and second half in descending order Set 2

Sort first half in ascending and second half in descending order Set 2 - GeeksforGeeks

Given an array of integers, sort the first half of the array in ascending order and second half in descending order.

Examples:

Input : arr[] = {10, 20, 30, 40}
Output : arr[] = {10, 20, 40, 30}

Input : arr[] = {5, 4, 6, 2, 1, 3, 8, 9, 7 }
Output : arr[] = {2, 4, 5, 6, 9, 8, 7, 3, 1 }

We have discussed a solution that only prints the required order in [Sort first half in ascending and second half in descending order Set 1](#)

Simple Approach

The idea is simple, we sort the first half in increasing order and second half in decreasing using library function. Most of the languages like Java, C++ provide provision to sort a subarray in a specified order. In this post, a different solution is discussed that modifies the original array.

Java

```
// Java program to sort first half in increasing
// order and second half in decreasing
import java.util.*;

public class SortExample {
```

```
static void mySort(Integer[] arr)
{
    int n = arr.length;

    // Sort subarray from index 1 to 4, i.e.,
    // only sort subarray {7, 6, 45, 21} and
    // keep other elements as it is.
    Arrays.sort(arr, 0, n / 2);
    Arrays.sort(arr, n / 2, n, Collections.reverseOrder());
}

public static void main(String[] args)
{
    // Our arr contains 8 elements
    Integer[] arr = { 5, 4, 6, 2, 1, 3, 8, 9, 7 };
    mySort(arr);
    System.out.printf("Modified arr[] : %s",
                      Arrays.toString(arr));
}
}
```

C++

```
// C++ program to sort first half in increasing
// order and second half in decreasing
#include<bits/stdc++.h>
using namespace std;

void mySort(int arr[], int n)
{
    sort(arr, arr+n/2);
    sort(arr+n/2, arr+n, greater<int>());
}

int main()
{
    int arr[] = { 5, 4, 6, 2, 1, 3, 8, 9, 7 };
    int n = sizeof(arr)/sizeof(arr[0]);
    mySort(arr, n);
    cout << "Modified Array : \n";
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Output:

Modified arr[] : [2, 4, 5, 6, 9, 8, 7, 3, 1]

Alternate Solution

- 1) Sort the whole array in ascending order.
- 2) Reverse the second half after sorting.

Java

```
// Java program to sort first half in increasing
// order and second half in decreasing
import java.util.*;

public class SortExample {
    static void mySort(Integer[] arr)
    {
        int n = arr.length;

        // Sort the whole array
        Arrays.sort(arr, 0, n/2);
        Arrays.sort(arr, n/2, n);

        // Reverse the second half
        int low = n/2, high = n-1;
        while (low < high)
        {
            Integer temp = arr[low];
            arr[low] = arr[high];
            arr[high] = temp;
            low++; high--;
        }
    }

    public static void main(String[] args)
    {
        // Our arr contains 8 elements
        Integer[] arr = { 5, 4, 6, 2, 1, 3, 8, 9, 7 };
        mySort(arr);
        System.out.printf("Modified arr[] : %s",
                           Arrays.toString(arr));
    }
}
```

C++

```
// C++ program to sort first half in increasing
// order and second half in decreasing
#include<bits/stdc++.h>
using namespace std;
```

```
void mySort(int arr[], int n)
{
    // Sort the first half
    sort(arr, arr+n/2);
    sort(arr+n/2, arr+n);

    reverse(arr+n/2, arr+n);
}

int main()
{
    int arr[] = { 5, 4, 6, 2, 1, 3, 8, 9, 7 };
    int n = sizeof(arr)/sizeof(arr[0]);
    mySort(arr, n);
    cout << "Modified Array : \n";
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Output:

Modified arr[] : [2, 4, 5, 6, 9, 8, 7, 3, 1]

Time Complexity of both approaches is $O(n \log n)$

Improved By : [Harsh Bajpai](#)

Source

<https://www.geeksforgeeks.org/sort-first-half-in-ascending-and-second-half-in-descending-order-set-2/>

Chapter 342

Sort first k values in ascending order and remaining n-k values in descending order

Sort first k values in ascending order and remaining n-k values in descending order - Geeks-forGeeks

Given an array of size n, arrange the first k elements of the array in ascending order and the remaining n-k elements in descending order.

Examples:

Input: arr[] = {5, 4, 6, 2, 1, 3, 8, 9, -1}, k = 4
Output: 2 4 5 6 9 8 3 1 -1

Input: arr[] = {5, 4, 6}, k = 2
Output: 4 5 6

Algorithm:

1. Store the first k elements in an array and sort that in ascending order.
2. Store the remaining n-k elements in an array and sort that in decending order.
3. Merge the two arrays by adding the elements from second array in reverse order.

C++

```
// C++ program to sort first k elements
// in increasing order and remaining
// n-k elements in decreasing
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to sort the array
void printOrder(int arr[], int n, int k)
{
    int len1 = k, len2 = n - k;
    int arr1[k], arr2[n - k];

    // Store the k elements in an array
    for (int i = 0; i < k; i++)
        arr1[i] = arr[i];

    // Store the remaining n-k elements in an array
    for (int i = k; i < n; i++)
        arr2[i - k] = arr[i];

    // sorting the array from 0 to k-1 places
    sort(arr1, arr1 + len1);

    // sorting the array from k to n places
    sort(arr2, arr2 + len2);

    // storing the values in the final array arr
    for (int i = 0; i < n; i++) {
        if (i < k)
            arr[i] = arr1[i];

        else {
            arr[i] = arr2[len2 - 1];
            len2--;
        }
    }
    // printing the array
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver code
int main()
{
    int arr[] = { 5, 4, 6, 2, 1, 3, 8, 9, -1 };
    int k = 4;

    int n = sizeof(arr) / sizeof(arr[0]);

    printOrder(arr, n, k);

    return 0;
}
```

}

Java

```
// Java program to sort first k elements
// in increasing order and remaining
// n-k elements in decreasing
import java.util.*;

class GFG {
    // function to print half of the array in
    // ascending order and the other half in
    // descending order
    static void printOrder(int[] arr, int n, int k)
    {
        int len1 = k, len2 = n - k;
        int[] arr1 = new int[k];
        int[] arr2 = new int[n - k];

        // Store the k elements in an array
        for (int i = 0; i < k; i++)
            arr1[i] = arr[i];

        // Store the remaining n-k elements in an array
        for (int i = k; i < n; i++)
            arr2[i - k] = arr[i];

        // sorting the array from 0 to k-1 places
        Arrays.sort(arr1, 0, k);

        // sorting the array from k to n places
        Arrays.sort(arr2, k, n-k);

        // storing the values in the final array arr
        for (int i = 0; i < n; i++) {
            if (i < k)
                arr[i] = arr1[i];

            else {
                arr[i] = arr2[len2 - 1];
                len2--;
            }
        }
        // printing the array
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

```
// Driver code
public static void main(String[] args)
{
    int arr[] = { 5, 4, 6, 2, 1, 3, 8, 9, -1 };
    int k = 4;
    int n = arr.length;
    printOrder(arr, n, k);
}
}
```

C#

```
// Java program to sort first k elements
// in increasing order and remaining
// n-k elements in decreasing
using System;

class GFG
{
// function to print half of the array in
// ascending order and the other half in
// descending order
static void printOrder(int[] arr,
                      int n, int k)
{
    int len2 = n - k;
    int[] arr1 = new int[k];
    int[] arr2 = new int[n - k];

    // Store the k elements in an array
    for (int i = 0; i < k; i++)
        arr1[i] = arr[i];

    // Store the remaining n-k
    // elements in an array
    for (int i = k; i < n; i++)
        arr2[i - k] = arr[i];

    // sorting the array from
    // 0 to k-1 places
    Array.Sort(arr1, 0, k);

    // sorting the array from k to n places
    Array.Sort(arr2, 0, n-k);

    // storing the values in
    // the final array arr
}
```

```
for (int i = 0; i < n; i++)
{
    if (i < k)
        arr[i] = arr1[i];

    else {
        arr[i] = arr2[len2 - 1];
        len2--;
    }
}

// printing the array
for (int i = 0; i < n; i++)
{
    Console.Write(arr[i] + " ");
}
}

// Driver code
public static void Main()
{
    int []arr = { 5, 4, 6, 2, 1,
                  3, 8, 9, -1 };
    int k = 4;
    int n = arr.Length;
    printOrder(arr, n, k);
}
}

// This code is contributed by Subhadeep
```

Output:

```
2 4 5 6 9 8 3 1 -1
```

Efficient Approach: The idea is simple, sort the first k elements in increasing order and remaining n-k elements in decreasing using library function.

C++

```
// C++ program to sort first k elements
// in increasing order and remaining
// n-k elements in decreasing
#include <bits/stdc++.h>
using namespace std;

// function to sort the array
```

```
void printOrder(int arr[], int n, int k)
{
    // Sort first k elements in ascending order
    sort(arr, arr + k);

    // Sort remaining n-k elements in descending order
    sort(arr + k, arr + n, greater<int>());
}

// Driver code
int main()
{
    int arr[] = { 5, 4, 6, 2, 1, 3, 8, 9, -1 };
    int k = 4;
    int n = sizeof(arr) / sizeof(arr[0]);

    printOrder(arr, n, k);

    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Java

```
// Java program to sort first k elements
// in increasing order and remaining
// n-k elements in decreasing
import java.util.*;

public class SortExample {
    static void printOrder(Integer[] arr, int k)
    {
        int n = arr.length;

        // Sort first k elements in ascending order
        Arrays.sort(arr, 0, k);

        // Sort remaining n-k elements in descending order
        Arrays.sort(arr, k, n, Collections.reverseOrder());
    }

    public static void main(String[] args)
    {
        // Our arr contains 8 elements
        Integer[] arr = { 5, 4, 6, 2, 1, 3, 8, 9, -1 };
        int k = 4;
```

```
    printOrder(arr, k);
    System.out.printf("%s", Arrays.toString(arr));
}
}
```

Output:

2 4 5 6 9 8 3 1 -1

Improved By : [tufan_gupta2000](#)

Source

<https://www.geeksforgeeks.org/sort-first-k-values-in-ascending-order-and-remaining-n-k-values-in-descending-order/>

Chapter 343

Sort n numbers in range from 0 to $n^2 - 1$ in linear time

Sort n numbers in range from 0 to $n^2 - 1$ in linear time - GeeksforGeeks

Given an array of numbers of size n. It is also given that the array elements are in range from 0 to $n^2 - 1$. Sort the given array in linear time.

Examples:

Since there are 5 elements, the elements can be from 0 to 24.

Input: arr[] = {0, 23, 14, 12, 9}

Output: arr[] = {0, 9, 12, 14, 23}

Since there are 3 elements, the elements can be from 0 to 8.

Input: arr[] = {7, 0, 2}

Output: arr[] = {0, 2, 7}

Solution: If we use [Counting Sort](#), it would take $O(n^2)$ time as the given range is of size n^2 . Using any comparison based sorting like[Merge Sort](#), [Heap Sort](#), .. etc would take $O(n\log n)$ time.

Now question arises how to do this in $O(n)$? Firstly, is it possible? Can we use data given in question? n numbers in range from 0 to $n^2 - 1$?

The idea is to use [Radix Sort](#). Following is standard Radix Sort algorithm.

- 1) Do following for each digit i where i varies from least significant digit to the most significant digit.
 - a) Sort input array using counting sort (or any stable sort) according to the i'th digit

Let there be d digits in input integers. Radix Sort takes $O(d*(n+b))$ time where b is the base for representing numbers, for example, for decimal system, b is 10. Since n^2-1 is the

maximum possible value, the value of d would be $O(\log_b(n))$. So overall time complexity is $O((n+b)*O(\log_b(n)))$. Which looks more than the time complexity of comparison based sorting algorithms for a large k. The idea is to change base b. If we set b as n, the value of $O(\log_b(n))$ becomes $O(1)$ and overall time complexity becomes $O(n)$.

```
arr[] = {0, 10, 13, 12, 7}
```

Let us consider the elements in base 5. For example 13 in base 5 is 23, and 7 in base 5 is 12.

```
arr[] = {00(0), 20(10), 23(13), 22(12), 12(7)}
```

After first iteration (Sorting according to the last digit in base 5), we get.

```
arr[] = {00(0), 20(10), 12(7), 22(12), 23(13)}
```

After second iteration, we get

```
arr[] = {00(0), 12(7), 20(10), 22(12), 23(13)}
```

Following is the implementation to sort an array of size n where elements are in range from 0 to $n^2 - 1$.

C++

```
#include<iostream>
using namespace std;

// A function to do counting sort of arr[] according to
// the digit represented by exp.
int countSort(int arr[], int n, int exp)
{

    int output[n]; // output array
    int i, count[n] ;
    for (int i=0; i < n; i++)
        count[i] = 0;

    // Store count of occurrences in count[]
    for (i = 0; i < n; i++)
        count[ (arr[i]/exp)%n ]++;

    // Change count[i] so that count[i] now contains actual
    // position of this digit in output[]
    for (i = 1; i < n; i++)
        count[i] += count[i - 1];

    // Build the output array
    for (i = n - 1; i >= 0; i--)
```

```
{  
    output[count[ (arr[i]/exp)%n] - 1] = arr[i];  
    count[(arr[i]/exp)%n]--;  
}  
  
// Copy the output array to arr[], so that arr[] now  
// contains sorted numbers according to current digit  
for (i = 0; i < n; i++)  
    arr[i] = output[i];  
}  
  
// The main function to that sorts arr[] of size n using Radix Sort  
void sort(int arr[], int n)  
{  
    // Do counting sort for first digit in base n. Note that  
    // instead of passing digit number, exp (n^0 = 1) is passed.  
    countSort(arr, n, 1);  
  
    // Do counting sort for second digit in base n. Note that  
    // instead of passing digit number, exp (n^1 = n) is passed.  
    countSort(arr, n, n);  
}  
  
// A utility function to print an array  
void printArr(int arr[], int n)  
{  
    for (int i = 0; i < n; i++)  
        cout << arr[i] << " ";  
}  
  
// Driver program to test above functions  
int main()  
{  
    // Since array size is 7, elements should be from 0 to 48  
    int arr[] = {40, 12, 45, 32, 33, 1, 22};  
    int n = sizeof(arr)/sizeof(arr[0]);  
    cout << "Given array is n";  
    printArr(arr, n);  
  
    sort(arr, n);  
  
    cout << "nSorted array is n";  
    printArr(arr, n);  
    return 0;  
}
```

Java

```
// Java program to sort an array of size n where elements are
// in range from 0 to  $n^2 - 1$ .
class Sort1ToN2
{
    // A function to do counting sort of arr[] according to
    // the digit represented by exp.
    void countSort(int arr[], int n, int exp)
    {
        int output[] = new int[n]; // output array
        int i, count[] = new int[n] ;
        for (i=0; i < n; i++)
            count[i] = 0;

        // Store count of occurrences in count[]
        for (i = 0; i < n; i++)
            count[ (arr[i]/exp)%n ]++;

        // Change count[i] so that count[i] now contains actual
        // position of this digit in output[]
        for (i = 1; i < n; i++)
            count[i] += count[i - 1];

        // Build the output array
        for (i = n - 1; i >= 0; i--)
        {
            output[count[ (arr[i]/exp)%n ] - 1] = arr[i];
            count[(arr[i]/exp)%n]--;
        }

        // Copy the output array to arr[], so that arr[] now
        // contains sorted numbers according to current digit
        for (i = 0; i < n; i++)
            arr[i] = output[i];
    }

    // The main function to that sorts arr[] of size n using Radix Sort
    void sort(int arr[], int n)
    {
        // Do counting sort for first digit in base n. Note that
        // instead of passing digit number, exp ( $n^0 = 1$ ) is passed.
        countSort(arr, n, 1);

        // Do counting sort for second digit in base n. Note that
        // instead of passing digit number, exp ( $n^1 = n$ ) is passed.
        countSort(arr, n, n);
    }
}
```

```
// A utility function to print an array
void printArr(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}

// Driver program to test above functions
public static void main(String args[])
{
    Sort1ToN2 ob = new Sort1ToN2();

    // Since array size is 7, elements should be from 0 to 48
    int arr[] = {40, 12, 45, 32, 33, 1, 22};
    int n = arr.length;
    System.out.println("Given array");
    ob.printArr(arr, n);

    ob.sort(arr, n);

    System.out.println("Sorted array");
    ob.printArr(arr, n);
}
}

/*This code is contributed by Rajat Mishra */
```

C#

```
// C# program to sort an array of
// size n where elements are
// in range from 0 to  $n^2 - 1$ .
using System;

class GFG {

    // A function to do counting
    // sort of arr[] according to
    // the digit represented by exp.
    static void countSort(int[] arr,
                          int n,
                          int exp)
    {

        // output array
        int[] output = new int[n];
        int[] count = new int[n];
        int i;
        for (i = 0; i < n; i++)
```

```
count[i] = 0;

// Store count of
// occurrences in count[]
for (i = 0; i < n; i++)
    count[(arr[i] / exp) % n]++;

// Change count[i] so that
// count[i] now contains actual
// position of this digit in output[]
for (i = 1; i < n; i++)
    count[i] += count[i - 1];

// Build the output array
for (i = n - 1; i >= 0; i--)
{
    output[count[(arr[i] /
                  exp) % n] - 1] = arr[i];
    count[(arr[i] / exp) % n]--;
}

// Copy the output array to
// arr[], so that arr[] now
// contains sorted numbers
// according to current digit
for (i = 0; i < n; i++)
    arr[i] = output[i];
}

// The main function to that
// sorts arr[] of size n
// using Radix Sort
static void sort(int[] arr, int n)
{
    // Do counting sort for first
    // digit in base n. Note that
    // instead of passing digit number,
    // exp ( $n^0 = 1$ ) is passed.
    countSort(arr, n, 1);

    // Do counting sort for second
    // digit in base n. Note that
    // instead of passing digit number,
    // exp ( $n^1 = n$ ) is passed.
    countSort(arr, n, n);
}
```

```
// A utility function
// to print an array
static void printArr(int[] arr, int n)
{
    for (int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
}

// Driver Code
static public void Main ()
{
    // Since array size is 7,
    // elements should be
    // from 0 to 48
    int[] arr = {40, 12, 45, 32, 33, 1, 22};
    int n = arr.Length;
    Console.WriteLine("Given array");
    printArr(arr, n);

    sort(arr, n);

    Console.WriteLine("\nSorted array");
    printArr(arr, n);
}
}

// This code is contributed by Ajit.
```

Output:

```
Given array is
40 12 45 32 33 1 22
Sorted array is
1 12 22 32 33 40 45
```

How to sort if range is from 1 to n^2 ?

If range is from 1 to n^2 , the above process can not be directly applied, it must be changed. Consider $n = 100$ and range from 1 to 10000. Since the base is 100, a digit must be from 0 to 99 and there should be 2 digits in the numbers. But the number 10000 has more than 2 digits. So to sort numbers in a range from 1 to n^2 , we can use following process.

- 1) Subtract all numbers by 1.
- 2) Since the range is now 0 to n^2 , do counting sort twice as done in the above implementation.
- 3) After the elements are sorted, add 1 to all numbers to obtain the original numbers.

How to sort if range is from 0 to $n^3 - 1$?

Since there can be 3 digits in base n, we need to call counting sort 3 times.

This article is contributed by **Bateesh**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [Rahul-Chauhan](#), [jit_t](#)

Source

<https://www.geeksforgeeks.org/sort-n-numbers-range-0-n2-1-linear-time/>

Chapter 344

Sort numbers stored on different machines

Sort numbers stored on different machines - GeeksforGeeks

Given N machines. Each machine contains some numbers in sorted form. But the amount of numbers, each machine has is not fixed. Output the numbers from all the machine in sorted non-decreasing form.

Example:

```
Machine M1 contains 3 numbers: {30, 40, 50}
Machine M2 contains 2 numbers: {35, 45}
Machine M3 contains 5 numbers: {10, 60, 70, 80, 100}

Output: {10, 30, 35, 40, 45, 50, 60, 70, 80, 100}
```

Representation of stream of numbers on each machine is considered as linked list. A Min Heap can be used to print all numbers in sorted order.

Following is the detailed process

1. Store the head pointers of the linked lists in a minHeap of size N where N is number of machines.
2. Extract the minimum item from the minHeap. Update the minHeap by replacing the head of the minHeap with the next number from the linked list or by replacing the head of the minHeap with the last number in the minHeap followed by decreasing the size of heap by 1.
3. Repeat the above step 2 until heap is not empty.

Below is C++ implementation of the above approach.

```
// A program to take numbers from different machines and print them in sorted order
```

```
#include <stdio.h>

// A Linked List node
struct ListNode
{
    int data;
    struct ListNode* next;
};

// A Min Heap Node
struct MinHeapNode
{
    ListNode* head;
};

// A Min Heao (Collection of Min Heap nodes)
struct MinHeap
{
    int count;
    int capacity;
    MinHeapNode* array;
};

// A function to create a Min Heap of given capacity
MinHeap* createMinHeap( int capacity )
{
    MinHeap* minHeap = new MinHeap;
    minHeap->capacity = capacity;
    minHeap->count = 0;
    minHeap->array = new MinHeapNode [minHeap->capacity];
    return minHeap;
}

/* A utility function to insert a new node at the begining
   of linked list */
void push (ListNode** head_ref, int new_data)
{
    /* allocate node */
    ListNode* new_node = new ListNode;

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}
```

```
}  
  
// A utility function to swap two min heap nodes. This function  
// is needed in minHeapify  
void swap( MinHeapNode* a, MinHeapNode* b )  
{  
    MinHeapNode temp = *a;  
    *a = *b;  
    *b = temp;  
}  
  
// The standard minHeapify function.  
void minHeapify( MinHeap* minHeap, int idx )  
{  
    int left, right, smallest;  
    left = 2 * idx + 1;  
    right = 2 * idx + 2;  
    smallest = idx;  
  
    if ( left < minHeap->count &&  
        minHeap->array[left].head->data <  
        minHeap->array[smallest].head->data  
    )  
        smallest = left;  
  
    if ( right < minHeap->count &&  
        minHeap->array[right].head->data <  
        minHeap->array[smallest].head->data  
    )  
        smallest = right;  
  
    if( smallest != idx )  
    {  
        swap( &minHeap->array[smallest], &minHeap->array[idx] );  
        minHeapify( minHeap, smallest );  
    }  
}  
  
// A utility function to check whether a Min Heap is empty or not  
int isEmpty( MinHeap* minHeap )  
{  
    return (minHeap->count == 0);  
}  
  
// A standard function to build a heap  
void buildMinHeap( MinHeap* minHeap )  
{  
    int i, n;
```

```
n = minHeap->count - 1;
for( i = (n - 1) / 2; i >= 0; --i )
    minHeapify( minHeap, i );
}

// This function inserts array elements to heap and then calls
// buildHeap for heap property among nodes
void populateMinHeap( MinHeap* minHeap, ListNode* *array, int n )
{
    for( int i = 0; i < n; ++i )
        minHeap->array[ minHeap->count++ ].head = array[i];

    buildMinHeap( minHeap );
}

// Return minimum element from all linked lists
ListNode* extractMin( MinHeap* minHeap )
{
    if( isEmpty( minHeap ) )
        return NULL;

    // The root of heap will have minimum value
    MinHeapNode temp = minHeap->array[0];

    // Replace root either with next node of the same list.
    if( temp.head->next )
        minHeap->array[0].head = temp.head->next;
    else // If list empty, then reduce heap size
    {
        minHeap->array[0] = minHeap->array[ minHeap->count - 1 ];
        --minHeap->count;
    }

    minHeapify( minHeap, 0 );
    return temp.head;
}

// The main function that takes an array of lists from N machines
// and generates the sorted output
void externalSort( ListNode *array[], int N )
{
    // Create a min heap of size equal to number of machines
    MinHeap* minHeap = createMinHeap( N );

    // populate first item from all machines
    populateMinHeap( minHeap, array, N );

    while ( !isEmpty( minHeap ) )
```

```
{  
    ListNode* temp = extractMin( minHeap );  
    printf( "%d ",temp->data );  
}  
}  
  
// Driver program to test above functions  
int main()  
{  
    int N = 3; // Number of machines  
  
    // an array of pointers storing the head nodes of the linked lists  
    ListNode *array[N];  
  
    // Create a Linked List 30->40->50 for first machine  
    array[0] = NULL;  
    push (&array[0], 50);  
    push (&array[0], 40);  
    push (&array[0], 30);  
  
    // Create a Linked List 35->45 for second machine  
    array[1] = NULL;  
    push (&array[1], 45);  
    push (&array[1], 35);  
  
    // Create Linked List 10->60->70->80 for third machine  
    array[2] = NULL;  
    push (&array[2], 100);  
    push (&array[2], 80);  
    push (&array[2], 70);  
    push (&array[2], 60);  
    push (&array[2], 10);  
  
    // Sort all elements  
    externalSort( array, N );  
  
    return 0;  
}
```

Output:

10 30 35 40 45 50 60 70 80 100

Source

<https://www.geeksforgeeks.org/sort-numbers-stored-on-different-machines/>

Chapter 345

Sort on the basis of number of factors using STL

Sort on the basis of number of factors using STL - GeeksforGeeks

Given an array of positive integers. Sort the given array in decreasing order of number of factors of each element, i.e., element having the highest number of factors should be the first to be displayed and the number having least number of factors should be the last one. Two elements with equal number of factors should be in the same order as in the original array.

Examples:

```
Input : {5, 11, 10, 20, 9, 16, 23}
Output : 20 16 10 9 5 11 23
Number of distinct factors:
For 20 = 6, 16 = 5, 10 = 4, 9 = 3
and for 5, 11, 23 = 2 (same number of factors
therefore sorted in increasing order of index)
```

```
Input : {104, 210, 315, 166, 441, 180}
Output : 180 210 315 441 104 166
```

We have already discussed a [structure based solution to sort according to number of factors](#).

The following steps sort numbers in decreasing order of count of factors.

1. Count distinct number of factors of each element. Refer [this](#).
2. Create a `vector` of `pairs` which stores elements and their factor counts.
3. Sort this array on the basis of the problem statement using any sorting algorithm.

```
// Sort an array of numbers according
// to number of factors.
#include <bits/stdc++.h>
using namespace std;

// Function that helps to sort elements
// in descending order
bool compare(const pair<int, int> &a,
             const pair<int, int> &b) {
    return (a.first > b.first);
}

// Prints array elements sorted in descending
// order of number of factors.
void printSorted(int arr[], int n) {

    vector<pair<int, int>> v;

    for (int i = 0; i < n; i++) {

        // Count factors of arr[i].
        int count = 0;
        for (int j = 1; j * j <= arr[i]; j++) {

            // To check Given Number is Exactly
            // divisible
            if (arr[i] % j == 0) {
                count++;
            }

            // To check Given number is perfect
            // square
            if (arr[i] / j != j)
                count++;
        }
    }

    // Insert factor count and array element
    v.push_back(make_pair(count, arr[i]));
}

// Sort the vector
sort(v.begin(), v.end(), compare);

// Print the vector
for (int i = 0; i < v.size(); i++)
    cout << v[i].second << " ";
}
```

```
// Driver's Function
int main() {
    int arr[] = {5, 11, 10, 20, 9, 16, 23};
    int n = sizeof(arr) / sizeof(arr[0]);

    printSorted(arr, n);

    return 0;
}
```

Output:

20 16 10 9 5 11 23

Time Complexity: $O(n * \sqrt{n})$
Auxiliary Complexity: $O(n)$

Source

<https://www.geeksforgeeks.org/sort-basis-number-factors-using-stl/>

Chapter 346

Sort string of characters

Sort string of characters - GeeksforGeeks

Given a string of lowercase characters from ‘a’ – ‘z’. We need to write a program to print the characters of this string in sorted order.

Examples:

Input : bbccdefbbaa
Output : aabbccdef

Input : geeksforgeeks
Output : eeeefggkkorss

A **simple approach** will be to use sorting algorithms like [quick sort](#) or [merge sort](#) and sort the input string and print it.

C++

```
// C++ program to sort a string of characters
#include<bits/stdc++.h>
using namespace std;

// function to print string in sorted order
void sortString(string &str)
{
    sort(str.begin(), str.end());
    cout << str;
}

// Driver program to test above function
int main()
{
```

```
    string s = "geeksforgeeks";
    sortString(s);
    return 0;
}
```

Output:

```
eeeeefggkkorss
```

Time Complexity: $O(n \log n)$, where n is the length of string.

An **efficient approach** will be to observe first that there can be a total of 26 unique characters only. So, we can store the count of occurrences of all the characters from ‘a’ to ‘z’ in a hashed array. The first index of the hashed array will represent character ‘a’, second will represent ‘b’ and so on. Finally, we will simply traverse the hashed array and print the characters from ‘a’ to ‘z’ the number of times they occurred in input string.

Below is the implementation of above idea:

C++

```
// C++ program to sort a string of characters
#include<bits/stdc++.h>
using namespace std;

const int MAX_CHAR = 26;

// function to print string in sorted order
void sortString(string &str)
{
    // Hash array to keep count of characters.
    // Initially count of all characters is
    // initialized to zero.
    int charCount[MAX_CHAR] = {0};

    // Traverse string and increment
    // count of characters
    for (int i=0; i<str.length(); i++)

        // 'a'-'a' will be 0, 'b'-'a' will be 1,
        // so for location of character in count
        // array we will do str[i]-'a'.
        charCount[str[i]-'a']++;

    // Traverse the hash array and print
    // characters
    for (int i=0;i<MAX_CHAR;i++)
        for (int j=0;j<charCount[i];j++)
```

```
        cout << (char)('a'+i);
    }

// Driver program to test above function
int main()
{
    string s = "geeksforgeeks";
    sortString(s);
    return 0;
}
```

Java

```
// Java program to sort
// a string of characters
import java.util.Arrays;
import java.util.Collections;

class GFG
{
    // Method to sort a
    // string alphabetically
    public static String sortString(String inputString)
    {
        // convert input
        // string to char array
        char tempArray[] =
            inputString.toCharArray();

        // sort tempArray
        Arrays.sort(tempArray);

        // return new sorted string
        return new String(tempArray);
    }

    // Driver Code
    public static void main(String[] args)
    {
        String inputString = "geeksforgeeks";

        System.out.println(sortString(inputString));
    }
}

// This code is contributed
// by prabhat kumar singh
```

Output:

eeeeefggkkorss

Time Complexity: $O(n)$, where n is the length of input string.

Auxiliary Space: $O(1)$.

Improved By : [prabhat kumar singh](#)

Source

<https://www.geeksforgeeks.org/sort-string-characters/>

Chapter 347

Sort string of characters using Stack

Sort string of characters using Stack - GeeksforGeeks

Given a string of characters. The task is to write a program to print the characters of this string in sorted order using stack.

Examples:

Input: str = "geeksforgeeks"
Output: eeeeefggkkorss

Input: str = "hello395world216"
Output: 123569dehllloorw

Approach:

- Initialize two stacks, one **stack** and other **tempstack**.
- Insert the first character of the string in the **stack**.
-
- if the i^{th} character is greater than or equal to the top element of the stack, then push the element.
- if the i^{th} character is not greater, then push all the elements of the **stack** into **tempstack**, and then push the character into the **stack**. After this, push all the greater elements of **tempstack** to **stack**.

Print the all elements of the **stack** in reverse order when the iteration is completed.

Below is the implementation of the above approach:

```
# Python program to sort
# string of characters using stack

# function to print the characters
# in sorted order
def printSorted(s, l):

    # primary stack
    stack = []

    # secondary stack
    tempstack = []

    # append first character
    stack.append(s[0])

    # iterate for all character in string
    for i in range(1, l):

        # i-th character ASCII
        a = ord(s[i])

        # stack's top element ASCII
        b = ord(stack[-1])

        # if greater or equal to top element
        # then push to stack
        if((a-b)>= 1 or (a == b)):
            stack.append(s[i])

        # if smaller, then push all element
        # to the temporary stack
        elif((b-a)>= 1):

            # push all greater elements
            while((b-a)>= 1):

                # push operation
                tempstack.append(stack.pop())

            # push till the stack is not-empty
            if(len(stack)>0):
                b = ord(stack[-1])
            else:
                break

        # push the i-th character
```

```
stack.append(s[i])

# push the tempstack back to stack
while(len(tempstack)>0):
    stack.append(tempstack.pop())

# print the stack in reverse order
print(''.join(stack))

# Driver Code
s = "geeksforgeeks"
l = len(s)
printSorted(s, l)
```

Output:

eeeeefggkkorss

An efficient approach using hashing has been implemented in [this post](#)

Source

<https://www.geeksforgeeks.org/sort-string-of-characters-using-stack/>

Chapter 348

Sort the biotonic doubly linked list

Sort the biotonic doubly linked list - GeeksforGeeks

Sort the given biotonic doubly linked list. A biotonic doubly linked list is a doubly linked list which is first increasing and then decreasing. A strictly increasing or a strictly decreasing list is also a biotonic doubly linked list.

Examples:

```
Input : DLL: 2<->5<->7<->12<->10<->6<->4<->1
Output : 1<->2<->4<->5<->6<->7<->10<->12
```

```
Input : DLL: 20<->17<->14<->8<->3
Output : 3<->8<->14<->17<->20
```

Approach: Find the first node in the list which is smaller than its previous node. Let it be **current**. If no such node is present then list is already sorted. Else split the list into two lists, **first** starting from **head** node till the **current**'s previous node and **second** starting from **current** node till the end of the list. Reverse the **second** doubly linked list. Refer [this](#) post. Now merge the **first** and **second** sorted doubly linked list. Refer merge procedure of [this](#) post. The final merged list is the required sorted doubly linked list.

```
// C++ implementation to sort the biotonic doubly linked list
#include <bits/stdc++.h>

using namespace std;

// a node of the doubly linked list
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
}
```

```
    struct Node* prev;
};

// Function to reverse a Doubly Linked List
void reverse(struct Node** head_ref)
{
    struct Node* temp = NULL;
    struct Node* current = *head_ref;

    // swap next and prev for all nodes
    // of doubly linked list
    while (current != NULL) {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;
    }

    // Before changing head, check for the cases
    // like empty list and list with only one node
    if (temp != NULL)
        *head_ref = temp->prev;
}

// Function to merge two sorted doubly linked lists
struct Node* merge(struct Node* first, struct Node* second)
{
    // If first linked list is empty
    if (!first)
        return second;

    // If second linked list is empty
    if (!second)
        return first;

    // Pick the smaller value
    if (first->data < second->data) {
        first->next = merge(first->next, second);
        first->next->prev = first;
        first->prev = NULL;
        return first;
    } else {
        second->next = merge(first, second->next);
        second->next->prev = second;
        second->prev = NULL;
        return second;
    }
}
```

```
// function to sort a biotonic doubly linked list
struct Node* sort(struct Node* head)
{
    // if list is empty or if it contains a single
    // node only
    if (head == NULL || head->next == NULL)
        return head;

    struct Node* current = head->next;

    while (current != NULL) {

        // if true, then 'current' is the first node
        // which is smaller than its previous node
        if (current->data < current->prev->data)
            break;

        // move to the next node
        current = current->next;
    }

    // if true, then list is already sorted
    if (current == NULL)
        return head;

    // spilt into two lists, one starting with 'head'
    // and other starting with 'current'
    current->prev->next = NULL;
    current->prev = NULL;

    // reverse the list starting with 'current'
    reverse(&current);

    // merge the two lists and return the
    // final merged doubly linked list
    return merge(head, current);
}

// Function to insert a node at the beginning
// of the Doubly Linked List
void push(struct Node** head_ref, int new_data)
{
    // allocate node
    struct Node* new_node =
        (struct Node*)malloc(sizeof(struct Node));

    // put in the data
```

```
new_node->data = new_data;

// since we are adding at the begining,
// prev is always NULL
new_node->prev = NULL;

// link the old list off the new node
new_node->next = (*head_ref);

// change prev of head node to new node
if ((*head_ref) != NULL)
    (*head_ref)->prev = new_node;

// move the head to point to the new node
(*head_ref) = new_node;
}

// Function to print nodes in a given doubly
// linked list
void printList(struct Node* head)
{
    // if list is empty
    if (head == NULL)
        cout << "Doubly Linked list empty";

    while (head != NULL) {
        cout << head->data << " ";
        head = head->next;
    }
}

// Driver program to test above
int main()
{
    struct Node* head = NULL;

    // Create the doubly linked list:
    // 2<->5<->7<->12<->10<->6<->4<->1
    push(&head, 1);
    push(&head, 4);
    push(&head, 6);
    push(&head, 10);
    push(&head, 12);
    push(&head, 7);
    push(&head, 5);
    push(&head, 2);

    cout << "Original Doubly linked list:n";
```

```
printList(head);

// sort the biotonic DLL
head = sort(head);

cout << "\nDoubly linked list after sorting:";
printList(head);

return 0;
}
```

Output:

```
Original Doubly linked list:
2 5 7 12 10 6 4 1
Doubly linked list after sorting:
1 2 4 5 6 7 10 12
```

Time Complexity: O(n)

Source

<https://www.geeksforgeeks.org/sort-biotonic-doubly-linked-list/>

Chapter 349

Sort the given matrix

Sort the given matrix - GeeksforGeeks

Given a $n \times n$ matrix. The problem is to sort the given matrix in strict order. Here strict order means that matrix is sorted in a way such that all elements in a row are sorted in increasing order and for row ' i ', where $1 \leq i \leq n-1$, first element of row ' i ' is greater than or equal to the last element of row ' $i-1$ '.

Examples:

```
Input : mat[][] = { {5, 4, 7},  
                   {1, 3, 8},  
                   {2, 9, 6} }  
Output : 1 2 3  
        4 5 6  
        7 8 9
```

Approach: Create a **temp[]** array of size n^2 . Starting with the first row one by one copy the elements of the given matrix into temp[]. Sort temp[]. Now one by one copy the elements of temp[] back to the given matrix.

C++

```
// C++ implementation to sort the given matrix  
#include <bits/stdc++.h>  
using namespace std;  
  
#define SIZE 10  
  
// function to sort the given matrix  
void sortMat(int mat[SIZE][SIZE], int n)  
{  
    // temporary matrix of size  $n^2$ 
```

```
int temp[n * n];
int k = 0;

// copy the elements of matrix one by one
// into temp[]
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        temp[k++] = mat[i][j];

// sort temp[]
sort(temp, temp + k);

// copy the elements of temp[] one by one
// in mat[][] []
k = 0;
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        mat[i][j] = temp[k++];
}

// function to print the given matrix
void printMat(int mat[SIZE][SIZE], int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << mat[i][j] << " ";
        cout << endl;
    }
}

// Driver program to test above
int main()
{
    int mat[SIZE][SIZE] = { { 5, 4, 7 },
                           { 1, 3, 8 },
                           { 2, 9, 6 } };
    int n = 3;

    cout << "Original Matrix:\n";
    printMat(mat, n);

    sortMat(mat, n);

    cout << "\nMatrix After Sorting:\n";
    printMat(mat, n);

    return 0;
}
```

Java

```
// Java implementation to
// sort the given matrix
import java.io.*;
import java.util.*;

class GFG {

    static int SIZE = 10;

    // function to sort the given matrix
    static void sortMat(int mat[][] , int n)
    {
        // temporary matrix of size n^2
        int temp[] = new int[n * n];
        int k = 0;

        // copy the elements of matrix
        // one by one into temp[]
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                temp[k++] = mat[i][j];

        // sort temp[]
        Arrays.sort(temp);

        // copy the elements of temp[]
        // one by one in mat[] []
        k = 0;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                mat[i][j] = temp[k++];
    }

    // function to print the given matrix
    static void printMat(int mat[][] , int n)
    {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                System.out.print( mat[i][j] + " ");
            System.out.println();
        }
    }

    // Driver program to test above
    public static void main(String args[])
    {
```

```
int mat[][] = { { 5, 4, 7 },
                { 1, 3, 8 },
                { 2, 9, 6 } };
int n = 3;

System.out.println("Original Matrix:");
printMat(mat, n);

sortMat(mat, n);

System.out.println("Matrix After Sorting:");
printMat(mat, n);

}

}

// This code is contributed by Nikita Tiwari.
```

Python3

```
# Python3 implementation to sort
# the given matrix

SIZE = 10

# Function to sort the given matrix
def sortMat(mat, n) :

    # Temporary matrix of size n^2
    temp = [0] * (n * n)
    k = 0

    # Copy the elements of matrix
    # one by one into temp[]
    for i in range(0, n) :

        for j in range(0, n) :

            temp[k] = mat[i][j]
            k += 1

    # sort temp[]
    temp.sort()

    # copy the elements of temp[]
    # one by one in mat[] []
    k = 0
```

```
for i in range(0, n) :  
  
    for j in range(0, n) :  
        mat[i][j] = temp[k]  
        k += 1  
  
# Function to print the given matrix  
def printMat(mat, n) :  
  
    for i in range(0, n) :  
  
        for j in range( 0, n ) :  
  
            print(mat[i][j] , end = " ")  
  
        print()  
  
# Driver program to test above  
mat = [ [ 5, 4, 7 ],  
        [ 1, 3, 8 ],  
        [ 2, 9, 6 ] ]  
n = 3  
  
print( "Original Matrix:")  
printMat(mat, n)  
  
sortMat(mat, n)  
  
print("\nMatrix After Sorting:")  
printMat(mat, n)  
  
# This code is contributed by Nikita Tiwari.
```

C#

```
// C# implementation to  
// sort the given matrix  
using System;  
  
class GFG {  
    static int SIZE = 10;  
  
    // function to sort the given matrix  
    static void sortMat(int[, ] mat, int n)  
    {
```

```
// temporary matrix of size n^2
int[] temp = new int[n * n];
int k = 0;

// copy the elements of matrix
// one by one into temp[]
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        temp[k++] = mat[i, j];

// sort temp[]
Array.Sort(temp);

// copy the elements of temp[]
// one by one in mat[][] []
k = 0;
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        mat[i, j] = temp[k++];
}

// function to print the given matrix
static void printMat(int[, ] mat, int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            Console.Write(mat[i, j] + " ");
        Console.WriteLine();
    }
}

// Driver code
public static void Main()
{
    int[, ] mat = { { 5, 4, 7 },
                    { 1, 3, 8 },
                    { 2, 9, 6 } };
    int n = 3;

    Console.WriteLine("Original Matrix:");
    printMat(mat, n);

    sortMat(mat, n);

    Console.WriteLine("Matrix After Sorting:");
    printMat(mat, n);
}
}
```

```
// This code is contributed by Sam007
```

Output :

Original Matrix:

```
5 4 7  
1 3 8  
2 9 6
```

Matrix After Sorting:

```
1 2 3  
4 5 6  
7 8 9
```

Time Complexity: $O(n^2 \log_2 n)$.

Auxiliary Space: $O(n^2)$.

Source

<https://www.geeksforgeeks.org/sort-given-matrix/>

Chapter 350

Sort the given string using character search

Sort the given string using character search - GeeksforGeeks

Given a string **str** of size **n**. The problem is to sort the given string without using any sorting techniques (like [bubble](#), [selection](#), etc). The string contains only lowercase characters.

Examples:

Input : geeksforgeeks
Output : eeeefggkkorss

Input : coding
Output : cdgino

Algorithm:

```
sortString(str, n)
    Initialize new_str = ""

    for i = 'a' to 'z'
        for j = 0 to n-1
            if str[j] == i, then
                new_str += str[j]

    return new_str
```

C++

```
// C++ implementation to sort the given string without
// using any sorting technique
#include <bits/stdc++.h>
using namespace std;

// function to sort the given string without
// using any sorting technique
string sortString(string str, int n) {

    // to store the final sorted string
    string new_str = "";

    // for each character 'i'
    for (int i = 'a'; i <= 'z'; i++)

        // if character 'i' is present at a particular
        // index then add character 'i' to 'new_str'
        for (int j = 0; j < n; j++)
            if (str[j] == i)
                new_str += str[j];

    // required final sorted string
    return new_str;
}

// Driver program to test above
int main() {
    string str = "geeksforgeeks";
    int n = str.size();
    cout << sortString(str, n);
    return 0;
}
```

Java

```
// Java implementation to sort the given
// string without using any sorting technique
class GFG {

    // function to sort the given string
    // without using any sorting technique
    static String sortString(String str, int n)
    {

        // to store the final sorted string
        String new_str = "";

        // for each character 'i'
```

```
for (int i = 'a'; i <= 'z'; i++)  
  
    // if character 'i' is present at a  
    // particular index then add character  
    // 'i' to 'new_str'  
    for (int j = 0; j < n; j++)  
        if (str.charAt(j) == i)  
            new_str += str.charAt(j);  
  
    // required final sorted string  
    return new_str;  
}  
  
// Driver code  
public static void main(String[] args)  
{  
    String str = "geeksforgeeks";  
    int n = str.length();  
  
    System.out.print(sortString(str, n));  
}  
}  
  
// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 implementation to sort  
# the given string without using  
# any sorting technique  
  
# Function to sort the given string  
# without using any sorting technique  
def sortString(str, n):  
  
    # To store the final sorted string  
    new_str = ""  
  
    # for each character 'i'  
    for i in range(ord('a'), ord('z') + 1):  
  
        # if character 'i' is present at a particular  
        # index then add character 'i' to 'new_str'  
        for j in range(n):  
            if (str[j] == chr(i)):  
                new_str += str[j]  
  
    # required final sorted string
```

```
return new_str

# Driver Code
str = "geeksforgeeks"
n = len(str)
print(sortString(str, n))

# This code is contributed by Anant Agarwal.
```

C#

```
// C# implementation to sort the given
// string without using any sorting technique
using System;

class GFG {

    // function to sort the given string
    // without using any sorting technique
    static String sortString(String str, int n)
    {
        // to store the final sorted string
        String new_str = "";

        // for each character 'i'
        for (int i = 'a'; i <= 'z'; i++)

            // if character 'i' is present at a
            // particular index then add character
            // 'i' to 'new_str'
            for (int j = 0; j < n; j++)
                if (str[j] == i)
                    new_str += str[j];

        // required final sorted string
        return new_str;
    }

    // Driver code
    public static void Main()
    {
        String str = "geeksforgeeks";
        int n = str.Length;

        Console.WriteLine(sortString(str, n));
    }
}
```

```
// This code is contributed by Sam007
```

Output :

```
eeeeefggkkorss
```

Source

<https://www.geeksforgeeks.org/sort-given-string-using-character-search/>

Chapter 351

Sort the linked list in the order of elements appearing in the array

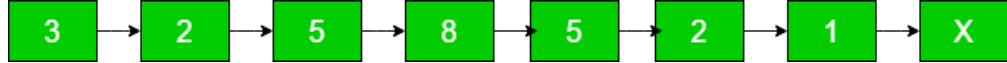
Sort the linked list in the order of elements appearing in the array - GeeksforGeeks

Given an array of size N and a Linked List where elements will be from the array but can also be duplicated, sort the linked list in the order, elements are appearing in the array. It may be assumed that the array covers all elements of the linked list.

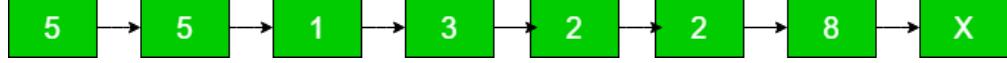
arr[] =

5	1	3	2	8
---	---	---	---	---

list =



Sorted list =



Asked in [Amazon](#)

First, make a hash table that stores the frequencies of elements in linked list. Then, simply traverse list and for each element of arr[i] check the frequency in the has table and modify the data of list by arr[i] element upto its frequency and at last Print the list.

```
// Efficient CPP program to sort given list in order
// elements are appearing in an array
#include <bits/stdc++.h>
using namespace std;
```

```
// Linked list node
struct Node {
    int data;
    struct Node* next;
};

// function prototype for printing the list
void printList(struct Node*);

// Function to insert a node at the
// beginning of the linked list
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = new Node;
    new_node -> data = new_data;
    new_node -> next = *head_ref;
    *head_ref = new_node;
}

// function to print the linked list
void printList(struct Node* head)
{
    while (head != NULL) {
        cout << head -> data << " -> ";
        head = head -> next;
    }
}

// Function that sort list in order of apperaing
// elements in an array
void sortlist(int arr[], int N, struct Node* head)
{
    // Store frequencies of elements in a
    // hash table.
    unordered_map<int, int> hash;
    struct Node* temp = head;
    while (temp) {
        hash[temp -> data]++;
        temp = temp -> next;
    }

    temp = head;

    // One by one put elements in lis according
    // to their appearance in array
    for (int i = 0; i < N; i++) {

        // Update 'frequency' nodes with value
```

```
// equal to arr[i]
int frequency = hash[arr[i]];
while (frequency--) {

    // Modify list data as element
    // appear in an array
    temp -> data = arr[i];
    temp = temp -> next;
}
}

// Driver Code
int main()
{
    struct Node* head = NULL;
    int arr[] = { 5, 1, 3, 2, 8 };
    int N = sizeof(arr) / sizeof(arr[0]);

    // creating the linked list
    push(&head, 3);
    push(&head, 2);
    push(&head, 5);
    push(&head, 8);
    push(&head, 5);
    push(&head, 2);
    push(&head, 1);

    // Function call to sort the list in order
    // elements are apperaing in an array
    sortlist(arr, N, head);

    // print the modified linked list
    cout << "Sorted List:" << endl;
    printList(head);
    return 0;
}
```

Output :

```
Sort list:
5 -> 5 -> 1 -> 3 -> 2 -> 2 -> 8
```

Source

<https://www.geeksforgeeks.org/sort-linked-list-order-elements-appearing-array/>

Chapter 352

Sort the matrix row-wise and column-wise

Sort the matrix row-wise and column-wise - GeeksforGeeks

Given a n x n matrix. The problem is to sort the matrix row-wise and column wise.

Examples:

```
Input : mat[][] = { {4, 1, 3},  
                   {9, 6, 8},  
                   {5, 2, 7} }
```

```
Output : 1 3 4  
         2 5 7  
         6 8 9
```

```
Input : mat[][] = { {12, 7, 1, 8},  
                   {20, 9, 11, 2},  
                   {15, 4, 5, 13},  
                   {3, 18, 10, 6} }
```

```
Output : 1 5 8 12  
         2 6 10 15  
         3 7 11 18  
         4 9 13 20
```

Approach: Following are the steps:

1. Sort each row of the matrix.
2. Get transpose of the matrix.
3. Again sort each row of the matrix.

4. Again get transpose of the matrix.

Algorithm for sorting each row of matrix using C++ STL sort():

```
for (int i = 0 ; i < n; i++)
    sort(mat[i] , mat[i] + n);
```

Algorithm for getting transpose of the matrix:

```
for (int i = 0; i < n; i++) {
    for (int j = i + 1; i < n; i++) {
        int temp = mat[i][j];
        mat[i][j] = mat[j][i];
        mat[j][i] = temp;
    }
}
```

C++

```
// C++ implementation to sort the matrix row-wise
// and column-wise
#include <bits/stdc++.h>

using namespace std;

#define MAX_SIZE 10

// function to sort each row of the matrix
void sortByRow(int mat[MAX_SIZE][MAX_SIZE], int n)
{
    for (int i = 0; i < n; i++)

        // sorting row number 'i'
        sort(mat[i] , mat[i] + n);
}

// function to find transpose of the matrix
void transpose(int mat[MAX_SIZE][MAX_SIZE], int n)
{
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)

            // swapping element at index (i, j)
            // by element at index (j, i)
```

```
        swap(mat[i][j], mat[j][i]);
    }

// function to sort the matrix row-wise
// and column-wise
void sortMatRowAndColWise(int mat[MAX_SIZE][MAX_SIZE],
                           int n)
{
    // sort rows of mat[] []
    sortByRow(mat, n);

    // get transpose of mat[] []
    transpose(mat, n);

    // again sort rows of mat[] []
    sortByRow(mat, n);

    // again get transpose of mat[] []
    transpose(mat, n);
}

// function to print the matrix
void printMat(int mat[MAX_SIZE][MAX_SIZE], int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << mat[i][j] << " ";
        cout << endl;
    }
}

// Driver program to test above
int main()
{
    int mat[MAX_SIZE][MAX_SIZE] = { { 4, 1, 3 },
                                    { 9, 6, 8 },
                                    { 5, 2, 7 } };
    int n = 3;

    cout << "Original Matrix:\n";
    printMat(mat, n);

    sortMatRowAndColWise(mat, n);

    cout << "\nMatrix After Sorting:\n";
    printMat(mat, n);

    return 0;
}
```

}

Java

```
// Java implementation to sort the
// matrix row-wise and column-wise
import java.util.Arrays;

class GFG
{
    static final int MAX_SIZE=10;

    // function to sort each row of the matrix
    static void sortByRow(int mat[][] , int n)
    {
        for (int i = 0; i < n; i++)

            // sorting row number 'i'
            Arrays.sort(mat[i]);
    }

    // function to find transpose of the matrix
    static void transpose(int mat[][] , int n)
    {
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
            {
                // swapping element at index (i, j)
                // by element at index (j, i)
                int temp=mat[i][j];
                mat[i][j]=mat[j][i];
                mat[j][i]=temp;
            }
    }

    // function to sort the matrix row-wise
    // and column-wise
    static void sortMatRowAndColWise(int mat[][] ,int n)
    {
        // sort rows of mat[][]
        sortByRow(mat, n);

        // get transpose of mat[][]
        transpose(mat, n);

        // again sort rows of mat[][]
        sortByRow(mat, n);
    }
}
```

```
// again get transpose of mat[] []
transpose(mat, n);
}

// function to print the matrix
static void printMat(int mat[][], int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            System.out.print(mat[i][j] + " ");
        System.out.println();
    }
}

// Driver code
public static void main (String[] args)
{
    int mat[][] = { { 4, 1, 3 },
                    { 9, 6, 8 },
                    { 5, 2, 7 } };
    int n = 3;

    System.out.print("Original Matrix:\n");
    printMat(mat, n);

    sortMatRowAndColWise(mat, n);

    System.out.print("\nMatrix After Sorting:\n");
    printMat(mat, n);
}
}

// This code is contributed by Anant Agarwal.
```

Python 3

```
# Python 3 implementation to
# sort the matrix row-wise
# and column-wise
MAX_SIZE = 10

# function to sort each
# row of the matrix
def sortByRow(mat, n):
    for i in range (n):

        # sorting row number 'i'
        for j in range(n-1):
```

```
if mat[i][j] > mat[i][j + 1]:  
    temp = mat[i][j]  
    mat[i][j] = mat[i][j + 1]  
    mat[i][j + 1] = temp  
  
# function to find  
# transpose of the matrix  
def transpose(mat, n):  
    for i in range(n):  
        for j in range(i + 1, n):  
  
            # swapping element at  
            # index (i, j) by element  
            # at index (j, i)  
            t = mat[i][j]  
            mat[i][j] = mat[j][i]  
            mat[j][i] = t  
  
# function to sort  
# the matrix row-wise  
# and column-wise  
def sortMatRowAndColWise(mat, n):  
  
    # sort rows of mat[] []  
    sortByRow(mat, n)  
  
    # get transpose of mat[] []  
    transpose(mat, n)  
  
    # again sort rows of mat[] []  
    sortByRow(mat, n)  
  
    # again get transpose of mat[] []  
    transpose(mat, n)  
  
# function to print the matrix  
def printMat(mat, n):  
    for i in range(n):  
        for j in range(n):  
            print(str(mat[i][j]), end = " ")  
        print();  
  
# Driver Code  
mat = [[ 4, 1, 3 ],  
       [ 9, 6, 8 ],  
       [ 5, 2, 7 ]]  
n = 3
```

```
print("Original Matrix:")
printMat(mat, n)

sortMatRowAndColWise(mat, n)

print("\nMatrix After Sorting:")
printMat(mat, n)

# This code is contributed
# by ChitraNayal

C#
// C# implementation to sort the
// matrix row-wise and column-wise
using System;

class GFG
{
    // function to sort each
    // row of the matrix
    static void sortByRow(int [,]mat,
                          int n)
    {

        // sorting row number 'i'
        for (int i = 0; i < n ; i++)
        {
            for(int j = 0;
                j < n - 1; j++)
            {
                if(mat[i, j] > mat[i, j + 1])
                {

                    var temp = mat[i, j];
                    mat[i, j] = mat[i, j + 1];
                    mat[i, j + 1] = temp;

                }
            }
        }
    }

    // function to find transpose
    // of the matrix
    static void transpose(int [,]mat,
                         int n)
{
```

```
for (int i = 0; i < n; i++)
    for (int j = i + 1;
        j < n; j++)
    {

        // swapping element at
        // index (i, j) by
        // element at index (j, i)
        var temp = mat[i, j];
        mat[i, j] = mat[j, i];
        mat[j, i] = temp;
    }
}

// function to sort
// the matrix row-wise
// and column-wise
static void sortMatRowAndColWise(int [,]mat,
                                  int n)
{
    // sort rows of mat[,]
    sortByRow(mat, n);

    // get transpose of mat[,]
    transpose(mat, n);

    // again sort rows of mat[,]
    sortByRow(mat, n);

    // again get transpose of mat[,]
    transpose(mat, n);
}

// function to print the matrix
static void printMat(int [,]mat, int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            Console.Write(mat[i, j] + " ");
        Console.WriteLine("\n");
    }
}

// Driver code
public static void Main ()
{
    int [,]mat = {{4, 1, 3},
```

```
        {9, 6, 8},
        {5, 2, 7}};
int n = 3;

Console.WriteLine("Original Matrix:\n");
printMat(mat, n);

sortMatRowAndColWise(mat, n);

Console.WriteLine("\nMatrix After Sorting:\n");
printMat(mat, n);
}

}

// This code is contributed
// by ChitraNayal
```

PHP

```
<?php
// PHP implementation to sort
// the matrix row-wise and
// column-wise
$MAX_SIZE = 10;

// function to sort each
// row of the matrix
function sortByRow(&$mat, $n)
{
    for ($i = 0; $i < $n; $i++)

        // sorting row number 'i'
        sort($mat[$i]);
}

// function to find
// transpose of the matrix
function transpose(&$mat, $n)
{
    for ($i = 0; $i < $n; $i++)
    {
        for ($j = $i + 1;
             $j < $n; $j++)
        {
            // swapping element at index (i, j)
            // by element at index (j, i)
            $t = $mat[$i][$j];
            $mat[$i][$j] = $mat[$j][$i];
```

```
$mat[$j][$i] = $t;
}
}

// function to sort
// the matrix row-wise
// and column-wise
function sortMatRowAndColWise(&$mat, $n)
{
    // sort rows of mat[] []
    sortByRow($mat, $n);

    // get transpose of mat[] []
    transpose($mat, $n);

    // again sort rows of mat[] []
    sortByRow($mat, $n);

    // again get transpose of mat[] []
    transpose($mat, $n);
}

// function to print the matrix
function printMat(&$mat, $n)
{
    for ($i = 0; $i < $n; $i++)
    {
        for ($j = 0; $j < $n; $j++)
            echo $mat[$i][$j] . " ";
        echo "\n";
    }
}

// Driver Code
$mat = array(array( 4, 1, 3 ),
             array( 9, 6, 8 ),
             array( 5, 2, 7 ));
$n = 3;

echo "Original Matrix:\n";
printMat($mat, $n);

sortMatRowAndColWise($mat, $n);

echo "\nMatrix After Sorting:\n";
printMat($mat, $n);
```

```
// This code is contributed  
// by ChitraNayal  
?>
```

Output:

Original Matrix:

```
4 1 3  
9 6 8  
5 2 7
```

Matrix After Sorting:

```
1 3 4  
2 5 7  
6 8 9
```

Time Complexity: $O(n^2 \log_2 n)$.

Auxiliary Space: $O(1)$.

Improved By : [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/sort-matrix-row-wise-column-wise/>

Chapter 353

Sort the words in lexicographical order in Python

Sort the words in lexicographical order in Python - GeeksforGeeks

Given a strings, we need to sort the words in lexicographical order (dictionary order).

Examples :

Input : "hello python program how are you"

Output : are
hello
how
program
python
you

Input : "Coders loves the algorithms"

Output : Coders
algorithms
loves
the

Note: The words which have first letter is capital letter they will print according alphabetical manner.

Approach :

Approach used in this program is very simple. Split the strings using split() function. After that sort the words in lexicographical order using sort(). Iterate the words through loop and print each word, which are already sorted.

```
# Python program to sort the words in lexicographical
```

```
# order

def sortLexo(my_string):

    # Split the my_string till where space is found.
    words = my_string.split()

    # sort() will sort the strings.
    words.sort()

    # Iterate i through 'words' to print the words
    # in alphabetical manner.
    for i in words:
        print( i )

# Driver code
if __name__ == '__main__':

    my_string = "hello this is example how to sort " \
                "the word in alphabetical manner"
    # Calling function
    sortLexo(my_string)
```

Output :

```
alphabetical
example
hello
how
in
is
manner
sort
the
this
to
word
```

Source

<https://www.geeksforgeeks.org/sort-words-lexicographical-order-python/>

Chapter 354

Sorting 2D Vector in C++ Set 2 (In descending order by row and column)

Sorting 2D Vector in C++ Set 2 (In descending order by row and column) - GeeksforGeeks

We have discussed some of the cases of sorting 2D vector in below set 1.

[Sorting 2D Vector in C++ Set 1 \(By row and column\)](#)

More cases are discussed in this article

Case 3 : To sort a particular row of 2D vector in descending order

This type of sorting arranges a selected row of 2D vector in descending order . This is achieved by using “sort()” and passing iterators of 1D vector as its arguments.

```
// C++ code to demonstrate sorting of a
// row of 2D vector in descending order
#include<iostream>
#include<vector> // for 2D vector
#include<algorithm> // for sort()
using namespace std;

int main()
{
    // Initializing 2D vector "vect" with
    // values
    vector< vector<int> > vect{{3, 5, 1},
                           {4, 8, 6},
                           {7, 2, 9}};
    // Number of rows;
    int m = vect.size();
```

```
// Number of columns (Assuming all rows
// are of same size). We can have different
// sizes though (like Java).
int n = vect[0].size();

// Displaying the 2D vector before sorting
cout << "The Matrix before sorting 1st row is:\n";
for (int i=0; i<m; i++)
{
    for (int j=0; j<n ;j++)
        cout << vect[i][j] << " ";
    cout << endl;
}

// Use of "sort()" for sorting first row
sort(vect[0].rbegin(), vect[0].rend());

// Displaying the 2D vector after sorting
cout << "The Matrix after sorting 1st row is:\n";
for (int i=0; i<m; i++)
{
    for (int j=0; j<n ;j++)
        cout << vect[i][j] << " ";
    cout << endl;
}

return 0;
}
```

Output:

The Matrix before sorting 1st row is:

3 5 1
4 8 6
7 2 9

The Matrix after sorting 1st row is:

5 3 1
4 8 6
7 2 9

Case 4 : To sort the entire 2D vector on basis of a particular column in descending order.

In this type of sorting 2D vector is entirely sorted on basis of a chosen column in descending order. For example if the chosen column is second, the row with greatest value in second column becomes first row, second greatest value in second column becomes second row, and so on.

```
{3, 5, 1},  
{4, 8, 6},  
{7, 2, 9};
```

After sorting this matrix by second column, we get

```
{4, 8, 6} // Row with greatest value in second column  
{3, 5, 1} // Row with second greatest value in second column  
{7, 2, 9}
```

This is achieved by passing a third argument in “sort()” as a call to user defined explicit function.

```
// C++ code to demonstrate sorting of a  
// 2D vector on basis of a column in  
// descending order  
#include<iostream>  
#include<vector> // for 2D vector  
#include<algorithm> // for sort()  
using namespace std;  
  
// Driver function to sort the 2D vector  
// on basis of a particular column in  
// descending order  
bool sortcol( const vector<int>& v1,  
              const vector<int>& v2 ) {  
    return v1[1] > v2[1];  
}  
  
int main()  
{  
    // Initializing 2D vector "vect" with  
    // values  
    vector< vector<int> > vect{{3, 5, 1},  
                             {4, 8, 6},  
                             {7, 2, 9}};  
  
    // Number of rows;  
    int m = vect.size();  
  
    // Number of columns (Assuming all rows  
    // are of same size). We can have different  
    // sizes though (like Java).  
    int n = vect[0].size();  
  
    // Displaying the 2D vector before sorting  
    cout << "The Matrix before sorting is:\n";  
    for (int i=0; i<m; i++)  
    {
```

```
for (int j=0; j<n ;j++)
    cout << vect[i][j] << " ";
cout << endl;
}

// Use of "sort()" for sorting on basis
// of 2nd column in descending order
sort(vect.begin(), vect.end(), sortcol);

// Displaying the 2D vector after sorting
cout << "The Matrix after sorting is:\n";
for (int i=0; i<m; i++)
{
    for (int j=0; j<n ;j++)
        cout << vect[i][j] << " ";
    cout << endl;
}
return 0;
}
```

Output:

The Matrix before sorting is:

3 5 1
4 8 6
7 2 9

The Matrix after sorting is:

4 8 6
3 5 1
7 2 9

Source

<https://www.geeksforgeeks.org/sorting-2d-vector-in-c-set-2-in-descending-order-by-row-and-column/>

Chapter 355

Sorting 2D Vector in C++ Set 3 (By number of columns)

Sorting 2D Vector in C++ Set 3 (By number of columns) - GeeksforGeeks

We have discussed some of the cases of sorting 2D vector in below set 1 and set 2.

[Sorting 2D Vector in C++ Set 1 \(By row and column\)](#)

[Sorting 2D Vector in C++ Set 2 \(In descending order by row and column\)](#)

More cases are discussed in this article

As mentioned in one of the article published of this set, A 2D Vector can also have rows with different number of columns. This property is unlike the 2D Array in which all rows have same number of columns.

```
// C++ code to demonstrate 2D Vector
// with different no. of columns
#include<iostream>
#include<vector> // for 2D vector
using namespace std;
int main()
{
    // Initializing 2D vector "vect" with
    // values
    vector< vector<int> > vect{{1, 2},
                           {3, 4, 5},
                           {6}};

    // Displaying the 2D vector
    for (int i=0; i<vect.size(); i++)
    {
        //loop till the size of particular
        //row
```

```
        for (int j=0; j<vect[i].size() ;j++)
            cout << vect[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

Output:

```
1 2
3 4 5
6
```

Case 5 : Sorting the 2D Vector on basis of no. of columns in row in ascending order.

In this type of sorting, 2D vector is sorted on basis of a no. of column in ascending order. This is achieved by passing a third argument in “sort()” as a call to user defined explicit function.

```
// C++ code to demonstrate sorting of
// 2D vector on basis of no. of columns
// in ascending order
#include<iostream>
#include<vector> // for 2D vector
#include<algorithm> // for sort()
using namespace std;

// Driver function to sort the 2D vector
// on basis of a no. of columns in
// ascending order
bool sizecom(const vector<int>& v1, const vector<int>& v2)
{
    return v1.size() < v2.size();
}

int main()
{
    // Initializing 2D vector "vect" with
    // values
    vector< vector<int> > vect{{1, 2},
                                {3, 4, 5},
                                {6}};
    // Displaying the 2D vector before sorting
```

```
cout << "The Matrix before sorting is:\n";
for (int i=0; i<vect.size(); i++)
{
    //loop till the size of particular
    //row
    for (int j=0; j<vect[i].size() ;j++)
        cout << vect[i][j] << " ";
    cout << endl;
}

//Use of "sort()" for sorting on
//basis of no. of columns in
//ascending order.
sort(vect.begin(), vect.end(), sizecom);

// Displaying the 2D vector after sorting
cout << "The Matrix after sorting is:\n";
for (int i=0; i<vect.size(); i++)
{
    //loop till the size of particular
    //row
    for (int j=0; j<vect[i].size() ;j++)
        cout << vect[i][j] << " ";
    cout << endl;
}

return 0;
}
```

Output:

```
The Matrix before sorting is:
1 2
3 4 5
6
The Matrix after sorting is:
6
1 2
3 4 5
```

Case 6 : Sorting the 2D Vector on basis of no. of columns in row in descending order.

In this type of sorting, 2D vector is sorted on basis of a no. of column in descending order. This is achieved by passing a third argument in “sort()” as a call to user defined explicit function.

```
// C++ code to demonstrate sorting of
// 2D vector on basis of no. of columns
// in descending order
#include<iostream>
#include<vector> // for 2D vector
#include<algorithm> // for sort()
using namespace std;

// Driver function to sort the 2D vector
// on basis of a no. of columns in
// descending order
bool sizecom(const vector<int>& v1, const vector<int>& v2)
{
    return v1.size() > v2.size();
}

int main()
{
    // Initializing 2D vector "vect" with
    // values
    vector< vector<int> > vect{{1, 2},
                                {3, 4, 5},
                                {6}};

    // Displaying the 2D vector before sorting
    cout << "The Matrix before sorting is:\n";
    for (int i=0; i<vect.size(); i++)
    {
        //loop till the size of particular
        //row
        for (int j=0; j<vect[i].size() ;j++)
            cout << vect[i][j] << " ";
        cout << endl;
    }

    //Use of "sort()" for sorting on
    //basis of no. of columns in
    //descending order.
    sort(vect.begin(), vect.end(), sizecom);

    // Displaying the 2D vector after sorting
    cout << "The Matrix after sorting is:\n";
    for (int i=0; i<vect.size(); i++)
    {
        //loop till the size of particular
        //row
        for (int j=0; j<vect[i].size() ;j++)
            cout << vect[i][j] << " ";
```

```
    cout << endl;
}

return 0;
}
```

Output:

The Matrix before sorting is:

```
1 2
3 4 5
6
The Matrix after sorting is:
3 4 5
1 2
6
```

Source

<https://www.geeksforgeeks.org/sorting-2d-vector-c-set-3-number-columns/>

Chapter 356

Sorting Array Elements By Frequency Set 3 (Using STL)

Sorting Array Elements By Frequency Set 3 (Using STL) - GeeksforGeeks

Given an array of integers, sort the array according to frequency of elements. If frequencies of two elements are same, print them in increasing order.

Examples:

```
Input : arr[] = {2, 3, 2, 4, 5, 12, 2, 3, 3, 3, 12}
Output : 3 3 3 3 2 2 2 12 12 4 5
Explanation :
No. Freq
2 : 3
3 : 4
4 : 1
5 : 1
12 : 2
```

We have discussed different approaches in below posts :

[Sort elements by frequency Set 1](#)
[Sort elements by frequency Set 2](#)

We can solve this problem using [map](#) and [pairs](#). Initially we create a map such that `map[element] = freq`. Once we are done building the map, we create an array of pairs. A pair which stores elements and their corresponding frequency will be used for the purpose of sorting. We write a custom compare function which compares two pairs firstly on the basis of freq and if there is a tie on the basis of values.

Below is its c++ implementation :

```
// C++ program to sort elements by frequency using
```

```
// STL
#include <bits/stdc++.h>
using namespace std;

// function to compare two pairs for inbuilt sort
bool compare(pair<int,int> &p1,
              pair<int, int> &p2)
{
    // If frequencies are same, compare
    // values
    if (p1.second == p2.second)
        return p1.first < p2.first;
    return p1.second > p2.second;
}

// function to print elements sorted by freq
void printSorted(int arr[], int n)
{
    // Store items and their frequencies
    map<int, int> m;
    for (int i = 0; i < n; i++)
        m[arr[i]]++;

    // no of distinct values in the array
    // is equal to size of map.
    int s = m.size();

    // an array of pairs
    pair<int, int> p[s];

    // Fill (val, freq) pairs in an array
    // of pairs.
    int i = 0;
    for (auto it = m.begin(); it != m.end(); ++it)
        p[i++] = make_pair(it->first, it->second);

    // sort the array of pairs using above
    // compare function.
    sort(p, p + s, compare);

    cout << "Elements sorted by frequency are: ";
    for (int i = 0; i < s; i++)
    {
        int freq = p[i].second;
        while (freq--)
            cout << p[i].first << " ";
    }
}
```

```
// driver program
int main()
{
    int arr[] = {2, 3, 2, 4, 5, 12, 2, 3,
                 3, 3, 12};
    int n = sizeof(arr)/ sizeof(arr[0]);
    printSorted(arr, n);
    return 0;
}
```

Output:

```
Elements sorted by frequency are:
3 3 3 2 2 2 12 12 4 5
```

Time Complexity : $O(n \ Log \ n)$

Source

<https://www.geeksforgeeks.org/sorting-array-elements-frequency-set-3-using-stl/>

Chapter 357

Sorting Big Integers

Sorting Big Integers - GeeksforGeeks

Given a array of **n** positive integers where each integer can have digits upto 10^6 , print the array elements in ascending order.

Input: arr[] = {54, 724523015759812365462, 870112101220845, 8723}

Output: 54 8723 870112101220845 724523015759812365462

Explanation:

All elements of array are sorted in non-descending(i.e., ascending) order of their integer value

Input: arr[] = {3643641264874311, 451234654453211101231,
4510122010112121012121}

Output: 3641264874311 451234654453211101231 4510122010112121012121

A **naive approach** is to use arbitrary precision data type such as **int** in python or **BigInteger** class in Java. But that approach will not be fruitful because internal conversion of string to int and then perform sorting will leads to slow down the calculations of addition and multiplications in binary number system.

Efficient Solution : As size of integer is very large even it can't be fit in **long long** data type of C/C++, so we just need to input all numbers as strings and sort them using a comparison function. Following are the key points compare function:-

1. If lengths of two strings are different, then we need to compare lengths to decide sorting order.
2. If Lengths are same then we just need to compare both the strings in lexicographically order.

Assumption : There are no leading zeros.

C++

```
// Below is C++ code to sort the Big integers in
// ascending order
#include<bits/stdc++.h>
using namespace std;

// comp function to perform sorting
bool comp(const string &left, const string &right)
{
    // if length of both string are equals then sort
    // them in lexicographically order
    if (left.size() == right.size())
        return left < right;

    // Otherwise sort them according to the length
    // of string in ascending order
    else
        return left.size() < right.size();
}

// Function to sort arr[] elements according
// to integer value
void SortingBigIntegers(string arr[], int n)
{
    // Copy the arr[] elements to sortArr[]
    vector<string> sortArr(arr, arr + n);

    // Inbuilt sort function using function as comp
    sort(sortArr.begin(), sortArr.end(), comp);

    // Print the final sorted array
    for (auto &ele : sortArr)
        cout << ele << " ";
}

// Driver code of above implementation
int main()
{
    string arr[] = {"54", "724523015759812365462",
                    "870112101220845", "8723"};
    int n = sizeof(arr) / sizeof(arr[0]);

    SortingBigIntegers(arr, n);

    return 0;
}
```

Python

```
# Below is Python code to sort the Big integers
# in ascending order
def SortingBigIntegers(arr, n):

    # Direct sorting using lamda operator
    # and length comparison
    arr.sort(key = lambda x: (len(x), x))

# Driver code of above implementation
arr = ["54", "724523015759812365462",
       "870112101220845", "8723"]
n = len(arr)

SortingBigIntegers(arr, n)

# Print the final sorted list using
# join method
print " ".join(arr)
```

Output: 54 8723 870112101220845 724523015759812365462

Time complexity: $O(\text{sum} * \log(n))$ where sum is the total sum of all string length and n is size of array

Auxiliary space: $O(n)$

Similar Post :

[Sort an array of large numbers](#)

Source

<https://www.geeksforgeeks.org/sorting-big-integers/>

Chapter 358

Sorting Strings using Bubble Sort

Sorting Strings using Bubble Sort - GeeksforGeeks

Given an array of strings arr[]. Sort given strings using Bubble Sort and display the sorted array.

In [Bubble Sort](#), the two successive strings arr[i] and arr[i+1] are exchanged whenever arr[i] > arr[i+1]. The larger values sink to the bottom and hence called sinking sort. At the end of each pass, smaller values gradually “bubble” their way upward to the top and hence called bubble sort.

After all the passes, we get all the strings in sorted order. Complexity of the above algorithm will be O(N²).

Let us look at the code snippet:

```
#include<bits/stdc++.h>
using namespace std;
#define MAX 100

void sortStrings(char arr[] [MAX], int n)
{
    char temp[MAX];

    // Sorting strings using bubble sort
    for (int j=0; j<n-1; j++)
    {
        for (int i=j+1; i<n; i++)
        {
            if (strcmp(arr[j], arr[i]) > 0)
            {
                strcpy(temp, arr[j]);
                arr[j] = arr[i];
                arr[i] = temp;
            }
        }
    }
}
```

```
        strcpy(arr[j], arr[i]);
        strcpy(arr[i], temp);
    }
}
}

int main()
{
    char arr[] [MAX] = {"GeeksforGeeks", "Quiz", "Practice", "Gblogs", "Coding"};
    int n = sizeof(arr)/sizeof(arr[0]);

    sortStrings(arr, n);

    printf("Strings in sorted order are : ");
    for (int i=0; i<n; i++)
        printf("\n String %d is %s", i+1, arr[i]);
    return 0;
}
```

Output:

```
Strings in sorted order are :
String 1 is Coding
String 2 is Gblogs
String 3 is GeeksforGeeks
String 4 is Practice
String 5 is Quiz
```

Source

<https://www.geeksforgeeks.org/sorting-strings-using-bubble-sort-2/>

Chapter 359

Sorting Terminology

Sorting Terminology - GeeksforGeeks

What is in-place sorting?

An in-place sorting algorithm uses constant extra space for producing the output (modifies the given array only). It sorts the list only by modifying the order of the elements within the list.

For example, Insertion Sort and Selection Sorts are in-place sorting algorithms as they do not use any additional space for sorting the list and a typical implementation of Merge Sort is not in-place, also the implementation for counting sort is not in-place sorting algorithm.

What are Internal and External Sortings?

When all data that needs to be sorted cannot be placed in-memory at a time, the sorting is called [external sorting](#). External Sorting is used for massive amount of data. Merge Sort and its variations are typically used for external sorting. Some external storage like hard-disk, CD, etc is used for external storage.

When all data is placed in-memory, then sorting is called internal sorting.

What is stable sorting?

See [Stable Sorting Algorithms](#)

Improved By : [AshishkrGoyal](#)

Source

<https://www.geeksforgeeks.org/sorting-terminology/>

Chapter 360

Sorting Vector of Pairs in C++ Set 1 (Sort by first and second)

Sorting Vector of Pairs in C++ Set 1 (Sort by first and second) - GeeksforGeeks

What is Vector of Pairs?

A [pair](#) is a container which stores two values mapped to each other, and a [vector](#) containing multiple number of such pairs is called a vector of pairs.

```
// C++ program to demonstrate vector of pairs
#include<bits/stdc++.h>
using namespace std;

int main()
{
    //declaring vector of pairs
    vector< pair <int,int> > vect;

    // initialising 1st and 2nd element of
    // pairs with array values
    int arr[] = {10, 20, 5, 40 };
    int arr1[] = {30, 60, 20, 50};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Entering values in vector of pairs
    for (int i=0; i<n; i++)
        vect.push_back( make_pair(arr[i],arr1[i]) );

    // Printing the vector
    for (int i=0; i<n; i++)
    {
        // "first" and "second" are used to access
```

```
// 1st and 2nd element of pair respectively
cout << vect[i].first << " "
    << vect[i].second << endl;
}

return 0;
}
```

Output:

```
10 30
20 60
5 20
40 50
```

Case 1 : Sorting the vector elements on the basis of first element of pairs in ascending order.

This type of sorting can be achieved using simple “ sort() ” function. By default the sort function sorts the vector elements on basis of first element of pairs.

```
// C++ program to demonstrate sorting in
// vector of pair according to 1st element
// of pair
#include<bits/stdc++.h>
using namespace std;

int main()
{
    // Declaring vector of pairs
    vector< pair <int,int> > vect;

    // Initializing 1st and 2nd element of
    // pairs with array values
    int arr[] = {10, 20, 5, 40 };
    int arr1[] = {30, 60, 20, 50};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Entering values in vector of pairs
    for (int i=0; i<n; i++)
        vect.push_back( make_pair(arr[i],arr1[i]) );

    // Printing the original vector(before sort())
    cout << "The vector before sort operation is:\n" ;
    for (int i=0; i<n; i++)
    {
        // "first" and "second" are used to access
```

```
// 1st and 2nd element of pair respectively
cout << vect[i].first << " "
    << vect[i].second << endl;

}

// Using simple sort() function to sort
sort(vect.begin(), vect.end());

// Printing the sorted vector(after using sort())
cout << "The vector after sort operation is:\n" ;
for (int i=0; i<n; i++)
{
    // "first" and "second" are used to access
    // 1st and 2nd element of pair respectively
    cout << vect[i].first << " "
        << vect[i].second << endl;
}

return 0;
}
```

Output:

The vector before applying sort operation is:

```
10 30
20 60
5 20
40 50
```

The vector after applying sort operation is:

```
5 20
10 30
20 60
40 50
```

Case 2 : Sorting the vector elements on the basis of second element of pairs in ascending order.

There are instances when we require to sort the elements of vector on the basis of second elements of pair. For that, we modify the sort() function and we pass a third argument, a call to an user defined explicit function in the sort() function.

```
// C++ program to demonstrate sorting in vector
// of pair according to 2nd element of pair
#include<bits/stdc++.h>
using namespace std;
```

```
// Driver function to sort the vector elements
// by second element of pairs
bool sortbysec(const pair<int,int> &a,
               const pair<int,int> &b)
{
    return (a.second < b.second);
}

int main()
{
    // declaring vector of pairs
    vector< pair <int, int> > vect;

    // Initialising 1st and 2nd element of pairs
    // with array values
    int arr[] = {10, 20, 5, 40 };
    int arr1[] = {30, 60, 20, 50};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Entering values in vector of pairs
    for (int i=0; i<n; i++)
        vect.push_back( make_pair(arr[i],arr1[i]) );

    // Printing the original vector(before sort())
    cout << "The vector before sort operation is:\n" ;
    for (int i=0; i<n; i++)
    {
        // "first" and "second" are used to access
        // 1st and 2nd element of pair respectively
        cout << vect[i].first << " "
            << vect[i].second << endl;
    }

    // Using sort() function to sort by 2nd element
    // of pair
    sort(vect.begin(), vect.end(), sortbysec);

    // Printing the sorted vector(after using sort())
    cout << "The vector after sort operation is:\n" ;
    for (int i=0; i<n; i++)
    {
        // "first" and "second" are used to access
        // 1st and 2nd element of pair respectively
        cout << vect[i].first << " "
            << vect[i].second << endl;
    }
    return 0;
}
```

}

Output:

The vector before applying sort operation is:

10 30
20 60
5 20
40 50

The vector after applying sort operation is:

5 20
10 30
40 50
20 60

Source

<https://www.geeksforgeeks.org/sorting-vector-of-pairs-in-c-set-1-sort-by-first-and-second/>

Chapter 361

Sorting Vector of Pairs in C++ Set 2 (Sort in descending order by first and second)

Sorting Vector of Pairs in C++ Set 2 (Sort in descending order by first and second) - GeeksforGeeks

We have discussed some of the cases of sorting vector of pairs in below set 1.

[Sorting Vector of Pairs in C++ Set 1 \(Sort by first and second\)](#)

More cases are discussed in this article

Sometimes we require to sort the vector in reverse order. In those instances, rather than first sorting the vector and later using “reverse” function increases time complexity of code. Therefore, to avoid this we sort the vector in descending order directly.

Case 3 : Sorting the vector elements on the basis of first element of pairs in descending order.

For these instances, we modify the sort() function and we pass a third argument, a call to an user defined explicit function in the sort() function.

```
// C++ program to demonstrate sorting in vector of
// pair according to 1st element of pair in
// descending order
#include<bits/stdc++.h>
using namespace std;

// Driver function to sort the vector elements by
// first element of pair in descending order
bool sortinrev(const pair<int,int> &a,
               const pair<int,int> &b)
```

```
{  
    return (a.first > b.first);  
}  
  
int main()  
{  
    // declaring vector of pairs  
    vector< pair <int,int> > vect;  
  
    // initializing 1st and 2nd element of  
    // pairs with array values  
    int arr[] = {5, 20, 10, 40 };  
    int arr1[] = {30, 60, 20, 50};  
    int n = sizeof(arr)/sizeof(arr[0]);  
  
    // Entering values in vector of pairs  
    for (int i=0; i<n; i++)  
        vect.push_back( make_pair(arr[i],arr1[i]) );  
  
    // Printing the original vector(before sort())  
    cout << "The vector before applying sort is:\n" ;  
    for (int i=0; i<n; i++)  
    {  
        // "first" and "second" are used to access  
        // 1st and 2nd element of pair respectively  
        cout << vect[i].first << " "  
            << vect[i].second << endl;  
    }  
  
    // using modified sort() function to sort  
    sort(vect.begin(), vect.end(), sortinrev);  
  
    // Printing the sorted vector(after using sort())  
    cout << "The vector after applying sort is:\n" ;  
    for (int i=0; i<n; i++)  
    {  
        // "first" and "second" are used to access  
        // 1st and 2nd element of pair respectively  
        cout << vect[i].first << " "  
            << vect[i].second << endl;  
    }  
    return 0;  
}
```

Output:

The vector before applying sort is:

```
5 30
20 60
10 20
40 50
```

The vector after applying sort is:

```
40 50
20 60
10 20
5 30
```

Case 4 : Sorting the vector elements on the basis of second element of pairs in descending order.

These instances can also be handled by modifying “sort()” function and again passing a call to user defined function.

```
// C++ program to demonstrate sorting/in vector of
// pair according to 2nd element of pair in
// descending order
#include<bits/stdc++.h>
using namespace std;

// Driver function to sort the vector elements by
// second element of pair in descending order
bool sortbysecdesc(const pair<int,int> &a,
                    const pair<int,int> &b)
{
    return a.second>b.second;
}

int main()
{
    // Declaring vector of pairs
    vector< pair <int,int> > vect;

    // Initializing 1st and 2nd element of
    // pairs with array values
    int arr[] = {5, 20, 10, 40 };
    int arr1[] = {30, 60, 20, 50};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Entering values in vector of pairs
    for (int i=0; i<n; i++)
        vect.push_back( make_pair(arr[i],arr1[i]) );

    // Printing the original vector(before sort())
}
```

```
cout << "The vector before sort operation is:\n" ;
for (int i=0; i<n; i++)
{
    // "first" and "second" are used to access
    // 1st and 2nd element of pair respectively
    cout << vect[i].first << " "
        << vect[i].second << endl;
}

// using modified sort() function to sort
sort(vect.begin(), vect.end(), sortbysecdesc);

// Printing the sorted vector(after using sort())
cout << "The vector after applying sort operation is:\n" ;
for (int i=0; i<n; i++)
{
    // "first" and "second" are used to access
    // 1st and 2nd element of pair respectively
    cout << vect[i].first << " "
        << vect[i].second << endl;
}
return 0;
}
```

Output:

```
The vector before sort operation is:
5 30
20 60
10 20
40 50
The vector after applying sort operation is:
20 60
40 50
5 30
10 20
```

Source

<https://www.geeksforgeeks.org/sorting-vector-of-pairs-in-c-set-2-sort-in-descending-order-by-first-and-second/>

Chapter 362

Sorting a Queue without extra space

Sorting a Queue without extra space - GeeksforGeeks

Given a queue with random elements, we need to sort it. We are not allowed to use extra space. The operations allowed on queue are :

1. enqueue() : Adds an item to rear of queue. In [C++ STL queue](#), this function is called push().
2. dequeue() : Removes an item from front of queue. In [C++ STL queue](#), this function is called pop().
3. isEmpty() : Checks if a queue is empty. In [C++ STL queue](#), this function is called empty().

Examples :

Input : A queue with elements
11 5 4 21

Output : Modified queue with
following elements
4 5 11 21

Input : A queue with elements
3 2 1 2

Output : Modified queue with
following elements
1 2 2 3

If we are allowed extra space, then we can simply move all items of queue to an array, then sort the array and finally move array elements back to queue.

How to do without extra space?

The idea: on every pass on the queue, we seek for the next minimum index. To do this we dequeue and enqueue elements until we find the next minimum. In this operation the queue is not changed at all. After we have found the minimum index, we dequeue and enqueue elements from the queue except for the minimum index, after we finish the traversal in the queue we insert the minimum to the rear of the queue. We keep on this until all minimums are pushed all way long to the front and the queue becomes sorted.

On every next seeking for the minimum, we exclude seeking on the minimums that have already sorted.

We repeat this method n times.

At first we seek for the maximum, because on every pass we need find the next minimum, so we need to compare it with the largest element in the queue.

Illustration:

Input :

```
-----  
11 5 4 21    min index = 2  
-----  
  
-----  
11 5 21 4    after inserting 4  
-----  
  
-----  
11 5 21 4    min index = 1  
-----  
  
-----  
11 21 4 5    after inserting 5  
-----  
  
-----  
11 21 4 5    min index = 0  
-----  
  
-----  
21 4 5 11    after inserting 11  
-----  
  
-----  
21 4 5 11    min index = 0  
-----  
  
-----  
4 5 11 21    after inserting 21  
-----  
Output : 4 5 11 21
```

C++

```
// C++ program to implement sorting a
// queue data structure
#include <bits/stdc++.h>
using namespace std;

// Queue elements after sortedIndex are
// already sorted. This function returns
// index of minimum element from front to
// sortedIndex
int minIndex(queue<int> &q, int sortedIndex)
{
    int min_index = -1;
    int min_val = INT_MAX;
    int n = q.size();
    for (int i=0; i<n; i++)
    {
        int curr = q.front();
        q.pop(); // This is dequeue() in C++ STL

        // we add the condition i <= sortedIndex
        // because we don't want to traverse
        // on the sorted part of the queue,
        // which is the right part.
        if (curr <= min_val && i <= sortedIndex)
        {
            min_index = i;
            min_val = curr;
        }
        q.push(curr); // This is enqueue() in
                      // C++ STL
    }
    return min_index;
}

// Moves given minimum element to rear of
// queue
void insertMinToRear(queue<int> &q, int min_index)
{
    int min_val;
    int n = q.size();
    for (int i = 0; i < n; i++)
    {
        int curr = q.front();
        q.pop();
        if (i != min_index)
            q.push(curr);
```

```
        else
            min_val = curr;
    }
    q.push(min_val);
}

void sortQueue(queue<int> &q)
{
    for (int i = 1; i <= q.size(); i++)
    {
        int min_index = minIndex(q, q.size() - i);
        insertMinToRear(q, min_index);
    }
}

// driver code
int main()
{
    queue<int> q;
    q.push(30);
    q.push(11);
    q.push(15);
    q.push(4);

    // Sort queue
    sortQueue(q);

    // Print sorted queue
    while (q.empty() == false)
    {
        cout << q.front() << " ";
        q.pop();
    }
    cout << endl;
    return 0;
}
```

Java

```
// Java program to implement sorting a
// queue data structure
import java.util.LinkedList;
import java.util.Queue;
class GFG
{
    // Queue elements after sortIndex are
    // already sorted. This function returns
    // index of minimum element from front to
```

```
// sortIndex
public static int minIndex(Queue<Integer> list,
                           int sortIndex)
{
    int min_index = -1;
    int min_value = Integer.MAX_VALUE;
    int s = list.size();
    for (int i = 0; i < s; i++)
    {
        int current = list.peek();

        // This is dequeue() in Java STL
        list.poll();

        // we add the condition i <= sortIndex
        // because we don't want to traverse
        // on the sorted part of the queue,
        // which is the right part.
        if (current <= min_value && i <= sortIndex)
        {
            min_index = i;
            min_value = current;
        }
        list.add(current);
    }
    return min_index;
}

// Moves given minimum element
// to rear of queue
public static void insertMinToRear(Queue<Integer> list,
                                    int min_index)
{
    int min_value = 0;
    int s = list.size();
    for (int i = 0; i < s; i++)
    {
        int current = list.peek();
        list.poll();
        if (i != min_index)
            list.add(current);
        else
            min_value = current;
    }
    list.add(min_value);
}

public static void sortQueue(Queue<Integer> list)
```

```
{  
    for(int i = 1; i <= list.size(); i++)  
    {  
        int min_index = minIndex(list, list.size() - i);  
        insertMinToRear(list, min_index);  
    }  
}  
  
//Driver function  
public static void main (String[] args)  
{  
    Queue<Integer> list = new LinkedList<Integer>();  
    list.add(30);  
    list.add(11);  
    list.add(15);  
    list.add(4);  
  
    //Sort Queue  
    sortQueue(list);  
  
    //print sorted Queue  
    while(list.isEmpty()== false)  
    {  
        System.out.print(list.peek() + " ");  
        list.poll();  
    }  
}  
}  
  
// This code is contributed by akash1295
```

C#

```
// C# program to implement  
// sorting a queue data structure  
using System;  
using System.Collections.Generic;  
  
class GFG  
{  
    // Queue elements after sorted  
    // Index are already sorted.  
    // This function returns index  
    // of minimum element from front  
    // to sortedIndex  
    static int minIndex(ref Queue<int> q,  
                        int sortedIndex)  
    {
```

```

int min_index = -1;
int min_val = int.MaxValue;
int n = q.Count;
for (int i = 0; i < n; i++)
{
    int curr = q.Peek();
    q.Dequeue(); // This is dequeue()
                 // in C++ STL

    // we add the condition
    // i <= sortedIndex because
    // we don't want to traverse
    // on the sorted part of the
    // queue, which is the right part.
    if (curr <= min_val &&
        i <= sortedIndex)
    {
        min_index = i;
        min_val = curr;
    }
    q.Enqueue(curr); // This is enqueue()
                     // in C++ STL
}
return min_index;
}

// Moves given minimum
// element to rear of queue
static void insertMinToRear(ref Queue<int> q,
                             int min_index)
{
    int min_val = 0;
    int n = q.Count;
    for (int i = 0; i < n; i++)
    {
        int curr = q.Peek();
        q.Dequeue();
        if (i != min_index)
            q.Enqueue(curr);
        else
            min_val = curr;
    }
    q.Enqueue(min_val);
}

static void sortQueue(ref Queue<int> q)
{
    for (int i = 1; i <= q.Count; i++)

```

```
{  
    int min_index = minIndex(ref q,  
                             q.Count - i);  
    insertMinToRear(ref q,  
                    min_index);  
}  
}  
  
// Driver Code  
static void Main()  
{  
    Queue<int> q = new Queue<int>();  
    q.Enqueue(30);  
    q.Enqueue(11);  
    q.Enqueue(15);  
    q.Enqueue(4);  
  
    // Sort queue  
    sortQueue(ref q);  
  
    // Print sorted queue  
    while (q.Count != 0)  
    {  
        Console.Write(q.Peek() + " ");  
        q.Dequeue();  
    }  
    Console.WriteLine();  
}  
}  
// This code is contributed by  
// Manish Shaw(manishshaw1)
```

Output:

4 11 15 30

Time complexity of this algorithm is $O(n^2)$.
Extra space needed is $O(1)$.

Improved By : [akash1295](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/sorting-queue-without-extra-space/>

Chapter 363

Sorting all array elements except one

Sorting all array elements except one - GeeksforGeeks

Given an array A of positive integers, sort the array in ascending order such that element at index K in unsorted array stays unmoved and all other elements are sorted.

Examples:

```
Input : arr[] = {10, 4, 11, 7, 6, 20}
        k = 2;
Output : arr[] = {4, 6, 11, 7, 10, 20}

Input : arr[] = {30, 20, 10}
        k = 0
Output : arr[] = {30, 10, 20}
```

A **simple solution** is to copy all elements except k-th of given array to another array. Then sort the other array using a sorting algorithm. Finally again copy the sorted array to original array. While copying, skip k-th element.

Below is an **efficient solution**.

1. Swap k-th element with last element.
2. Sort all elements except last.
3. For every element from (k+1)-th to last, move them one position ahead.
4. Copy k-th element back to position k.

C++

```
// CPP program to sort all elements except
// element at index k.
#include <bits/stdc++.h>
using namespace std;

int sortExceptK(int arr[], int k, int n)
{
    // Move k-th element to end
    swap(arr[k], arr[n-1]);

    // Sort all elements except last
    sort(arr, arr + n - 1);

    // Store last element (originally k-th)
    int last = arr[n-1];

    // Move all elements from k-th to one
    // position ahead.
    for (int i=n-1; i>k; i--)
        arr[i] = arr[i-1];

    // Restore k-th element
    arr[k] = last;
}

// Driver code
int main()
{
    int a[] = {10, 4, 11, 7, 6, 20 };
    int k = 2;
    int n = sizeof(a) / sizeof(a[0]);
    sortExceptK(a, k, n);
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
```

Java

```
// Java program to sort all elements except
// element at index k.
import java.util.Arrays;

class GFG {

    static int sortExceptK(int arr[], int k, int n)
    {

        // Move k-th element to end
```

```
int temp = arr[k];
arr[k] = arr[n-1];
arr[n-1] = temp;

// Sort all elements except last
Arrays.sort(arr, 0, n-1);

// Store last element (originally k-th)
int last = arr[n-1];

// Move all elements from k-th to one
// position ahead.
for (int i = n-1; i > k; i--)
    arr[i] = arr[i-1];

// Restore k-th element
arr[k] = last;
return 0;
}

//Driver code
public static void main (String[] args)
{
    int a[] = {10, 4, 11, 7, 6, 20 };
    int k = 2;
    int n = a.length;

    sortExceptK(a, k, n);

    for (int i = 0; i < n; i++)
        System.out.print(a[i] + " ");
}
}

//This code is contributed by Anant Agarwal.
```

C#

```
// C# program to sort all elements except
// element at index k.
using System;

public class GFG {

    static int sortExceptK(int[] arr, int k, int n)
    {
        // Move k-th element to end
        int temp = arr[k];
```

```
arr[k] = arr[n - 1];
arr[n - 1] = temp;

// Sort all elements except last
Array.Sort(arr, 0, n - 1);

// Store last element (originally k-th)
int last = arr[n - 1];

// Move all elements from k-th to one
// position ahead.
for (int i = n - 1; i > k; i--)
    arr[i] = arr[i - 1];

// Restore k-th element
arr[k] = last;

return 0;
}

// Driver code
public static void Main()
{
    int[] a = { 10, 4, 11, 7, 6, 20 };
    int k = 2;
    int n = a.Length;

    sortExceptK(a, k, n);

    for (int i = 0; i < n; i++)
        Console.Write(a[i] + " ");
}
}

// This article is contributed by shiv_bhakt
```

PHP

```
<?php
// PHP program to sort all
// elements except element
// at index k.
function sortExceptK(&$arr, $k, $n)
{
    // Move k-th element to end
    $t = $arr[$k];
    $arr[$k] = $arr[$n - 1];
    $arr[$n - 1] = $t;
```

```
// Sort all elements
// except last
$t = $arr[count($arr) - 1];
$arr = array_slice($arr, 0, -1);
sort($arr);
array_push($arr, $t);

// Store last element
// (originally k-th)
$last = $arr[$n - 1];

// Move all elements from
// k-th to one position ahead.
for ($i = $n - 1; $i > $k; $i--)
    $arr[$i] = $arr[$i - 1];

// Restore k-th element
$arr[$k] = $last;
}

// Driver code
$a = array(10, 4, 11,
           7, 6, 20 );
$k = 2;
$n = count($a);
sortExceptK($a, $k, $n);

for ($i = 0; $i < $n; $i++)
    echo ($a[$i]. " ");

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output:

4 6 11 7 10 20

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/sorting-array-elements-except-one/>

Chapter 364

Sorting an array according to another array using pair in STL

Sorting an array according to another array using pair in STL - GeeksforGeeks

We are given two arrays. We need to sort one array according to another.

Examples:

Input : 2 1 5 4 9 3 6 7 10 8
A B C D E F G H I J

Output : 1 2 3 4 5 6 7 8 9 10
B A F D C G H J E I

Here we are sorting second array
(a character array) according to
the first array (an integer array).

We have discussed different ways in below post.

[Sort an array according to the order defined by another array](#)

In this post we are focusing on using the `pair` container present in STL of C++.

To achieve our task we are going to make pairs of respective elements from both the arrays. Then simply use the `sort` function. The important thing to note is, the first element in the pairs should be from the array according to which the sorting is to be performed.

```
// Sort an array according to
// other using pair in STL.
#include <bits/stdc++.h>
using namespace std;

// Function to sort character array b[]
```

```
// according to the order defined by a[]
void pairsort(int a[], char b[], int n)
{
    pair<int, char> pait[n];

    // Storing the respective array
    // elements in pairs.
    for (int i = 0; i < n; i++)
    {
        pait[i].first = a[i];
        pait[i].second = b[i];
    }

    // Sorting the pair array.
    sort(pait, pait + n);

    // Modifying original arrays
    for (int i = 0; i < n; i++)
    {
        a[i] = pait[i].first;
        b[i] = pait[i].second;
    }
}

// Driver function
int main()
{
    int a[] = {2, 1, 5, 4, 9, 3, 6, 7, 10, 8};
    char b[] = {'A', 'B', 'C', 'D', 'E', 'F',
               'G', 'H', 'I', 'J'};

    int n = sizeof(a) / sizeof(a[0]);

    // Function calling
    pairsort(a, b, n);

    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << endl;

    for (int i = 0; i < n; i++)
        cout << b[i] << " ";

    return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10
B A F D C G H J E I
```

Improved By : [cs_abhi](#)

Source

<https://www.geeksforgeeks.org/sorting-array-according-another-array-using-pair-stl/>

Chapter 365

Sorting array except elements in a subarray

Sorting array except elements in a subarray - GeeksforGeeks

Given an array A of positive integers, sort the array in ascending order such that element in given subarray (start and end indexes are input) in unsorted array stay unmoved and all other elements are sorted.

Examples :

```
Input : arr[] = {10, 4, 11, 7, 6, 20}
        l = 1, u = 3
Output : arr[] = {6, 4, 11, 7, 10, 20}
We sort elements except arr[1..3] which
is {11, 7, 6}.
```

```
Input : arr[] = {5, 4, 3, 12, 14, 9};
        l = 1, u = 2;
Output : arr[] = {5, 4, 3, 9, 12, 14 }
We sort elements except arr[1..2] which
is {4, 3}.
```

Approach : Copy all elements except the given limit of given array to another array. Then sort the other array using a sorting algorithm. Finally again copy the sorted array to original array. While copying, skip given subarray.

C++

```
// CPP program to sort all elements except
// given subarray.
#include <bits/stdc++.h>
```

```
using namespace std;

// Sort whole array a[] except elements in
// range a[l..r]
void sortExceptUandL(int a[], int l, int u, int n)
{
    // Copy all those element that need
    // to be sorted to an auxiliary
    // array b[]
    int b[n - (u - l + 1)];
    for (int i = 0; i < l; i++)
        b[i] = a[i];
    for (int i = u+1; i < n; i++)
        b[l + (i - (u+1))] = a[i];

    // sort the array b
    sort(b, b + n - (u - l + 1));

    // Copy sorted elements back to a[]
    for (int i = 0; i < l; i++)
        a[i] = b[i];
    for (int i = u+1; i < n; i++)
        a[i] = b[l + (i - (u+1))];
}

// Driver code
int main()
{
    int a[] = { 5, 4, 3, 12, 14, 9 };
    int n = sizeof(a) / sizeof(a[0]);
    int l = 2, u = 4;
    sortExceptUandL(a, l, u, n);
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
}
```

Java

```
// Java program to sort all elements except
// given subarray.
import java.util.Arrays;
import java.io.*;

public class GFG {

    // Sort whole array a[] except elements in
    // range a[l..r]
    public static void sortExceptUandL(int a[],
```

```
        int l, int u, int n)
{

    // Copy all those element that need
    // to be sorted to an auxiliary
    // array b[]
    int b[] = new int[n - (u - l + 1)];

    for (int i = 0; i < l; i++)
        b[i] = a[i];
    for (int i = u+1; i < n; i++)
        b[l + (i - (u+1))] = a[i];

    // sort the array b
    Arrays.sort(b);

    // Copy sorted elements back to a[]
    for (int i = 0; i < l; i++)
        a[i] = b[i];
    for (int i = u+1; i < n; i++)
        a[i] = b[l + (i - (u+1))];

}

// Driver code
public static void main(String args[])
{
    int a[] = { 5, 4, 3, 12, 14, 9 };
    int n = a.length;
    int l = 2, u = 4;

    sortExceptUandL(a, l, u, n);

    for (int i = 0; i < n; i++)
        System.out.print(a[i] + " ");
}
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

Python3

```
# Python3 program to sort all elements
# except given subarray.

# Sort whole array a[] except elements in
# range a[l..r]
def sortExceptUandL(a, l, u, n) :
```

```
# Copy all those element that need
# to be sorted to an auxiliary
# array b[]
b = [0] * (n - (u - l + 1))

for i in range(0, l) :
    b[i] = a[i]

for i in range(u+1, n) :
    b[l + (i - (u+1))] = a[i]

# sort the array b
b.sort()

# Copy sorted elements back to a[]
for i in range(0, l) :
    a[i] = b[i]

for i in range(u+1, n) :
    a[i] = b[l + (i - (u+1))]

# Driver code
a = [ 5, 4, 3, 12, 14, 9 ]
n = len(a)
l = 2
u = 4
sortExceptUandL(a, l, u, n)

for i in range(0, n) :
    print ("{} ".format(a[i]), end="")

# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
// C# program to sort all elements except
// given subarray.
using System;
using System.Collections.Generic;

class GFG {

    // Sort whole array a[] except elements in
    // range a[l..r]
    static void sortExceptUandL(int []a, int l,
                                int u, int n)
    {
```

```
// Copy all those element that need
// to be sorted to an auxiliary
// array b[]
int[] b = new int[n - (u-l+1)];

for (int i = 0; i < l; i++)
    b[i] = a[i];

for (int i = u+1; i < n; i++)
    b[l + (i - (u+1))] = a[i];

// sort the array b
Array.Sort<int>(b, 0, n - (u - l + 1));

// Copy sorted elements back to a[]
for (int i = 0; i < l; i++)
    a[i] = b[i];

for (int i = u+1; i < n; i++)
    a[i] = b[l + (i - (u+1))];
}

// Driver code
public static void Main()
{
    int []a = { 5, 4, 3, 12, 14, 9 };
    int n = a.Length;
    int l = 2, u = 4;

    sortExceptUandL(a, l, u, n);

    for (int i = 0; i < n; i++)
        Console.Write(a[i] + " ");
}
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

PHP

```
<?php
// PHP program to sort all
// elements except given subarray.

// Sort whole array a[] except
// elements in range a[l..r]
```

```
function sortExceptUandL($a, $l,
                         $u, $n)
{
    // Copy all those element
    // that need to be sorted
    // to an auxiliary array b[]
    $b = array();
    for ($i = 0; $i < $l; $i++)
        $b[$i] = $a[$i];
    for ($i = $u + 1; $i < $n; $i++)
        $b[$l + ($i - ($u + 1))] = $a[$i];

    // sort the array b
    sort($b);

    // Copy sorted elements
    // back to a[]
    for ($i = 0; $i < $l; $i++)
        $a[$i] = $b[$i];
    for ($i = $u + 1; $i < $n; $i++)
        $a[$i] = $b[$l + ($i - ($u + 1))];
}

// Driver code
$a = array(4, 5, 3, 12, 14, 9);
$n = count($a);
$l = 2;
$u = 4;
sortExceptUandL($a, $l, $u, $n);
for ($i = 0; $i < $n; $i++)
    echo ($a[$i]. " ");

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output :

4 5 3 12 14 9

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/sorting-array-except-elements-subarray/>

Chapter 366

Sorting array of strings (or words) using Trie

Sorting array of strings (or words) using Trie - GeeksforGeeks

Given an array of strings, print them in alphabetical (dictionary) order. If there are duplicates in input array, we need to print them only once.

Examples:

Input : "abc", "xy", "bcd"
Output : abc bcd xy

Input : "geeks", "for", "geeks", "a", "portal",
"to", "learn", "can", "be", "computer",
"science", "zoom", "yup", "fire", "in", "data"
Output : a be can computer data fire for geeks
in learn portal science to yup zoom

Trie is an efficient data structure used for storing data like strings. To print the string in alphabetical order we have to first insert in the trie and then perform preorder traversal to print in alphabetical order.

```
// C++ program to sort an array of strings
// using Trie
#include <bits/stdc++.h>
using namespace std;

const int MAX_CHAR = 26;

struct Trie {
```

```
// index is set when node is a leaf
// node, otherwise -1;
int index;

Trie* child[MAX_CHAR];

/*to make new trie*/
Trie()
{
    for (int i = 0; i < MAX_CHAR; i++)
        child[i] = NULL;
    index = -1;
}
};

/* function to insert in trie */
void insert(Trie* root, string str, int index)
{
    Trie* node = root;

    for (int i = 0; i < str.size(); i++) {

        /* taking ascii value to find index of
           child node */
        char ind = str[i] - 'a';

        /* making new path if not already */
        if (!node->child[ind])
            node->child[ind] = new Trie();

        // go to next node
        node = node->child[ind];
    }

    // Mark leaf (end of word) and store
    // index of word in arr[]
    node->index = index;
}

/* function for preorder traversal */
bool preorder(Trie* node, string arr[])
{
    if (node == NULL)
        return false;

    for (int i = 0; i < MAX_CHAR; i++) {
        if (node->child[i] != NULL) {
```

```
/* if leaf node then print key*/
if (node->child[i]->index != -1)
    cout << arr[node->child[i]->index] << endl;

    preorder(node->child[i], arr);
}
}
}

void printSorted(string arr[], int n)
{
    Trie* root = new Trie();

    // insert all keys of dictionary into trie
    for (int i = 0; i < n; i++)
        insert(root, arr[i], i);

    // print keys in lexicographic order
    preorder(root, arr);
}

// Driver code
int main()
{
    string arr[] = { "abc", "xy", "bcd" };
    int n = sizeof(arr) / sizeof(arr[0]);
    printSorted(arr, n);
    return 0;
}
```

Output:

abc bcd xy

Source

<https://www.geeksforgeeks.org/sorting-array-strings-words-using-trie/>

Chapter 367

Sorting array of strings (or words) using Trie Set-2 (Handling Duplicates)

Sorting array of strings (or words) using Trie Set-2 (Handling Duplicates) - GeeksforGeeks

Given an array of strings, print them in alphabetical (dictionary) order. If there are duplicates in input array, we need to print all the occurrences.

Examples:

```
Input : arr[] = { "abc", "xyz", "abcd", "bcd", "abc" }  
Output : abc abc abcd bcd xyz
```

```
Input : arr[] = { "geeks", "for", "geeks", "a", "portal",  
                 "to", "learn" }  
Output : a for geeks geeks learn portal to
```

Prerequisite: [Trie \(Insert and Search\).](#)

Approach: In the [previous post](#) array of strings is being sorted, printing only single occurrence of duplicate strings. In this post all occurrences of duplicate strings are printed in lexicographic order. To print the strings in alphabetical order we have to first insert them in the trie and then perform preorder traversal to print in alphabetical order. The nodes of trie contain an **index[]** array which stores the index position of all the strings of **arr[]** ending at that node. Except for trie's leaf node all the other nodes have size 0 for the **index[]** array.

Below is the implementation of the above approach.

C++

```
// C++ implementation to sort an array
```

```
// of strings using Trie
#include <bits/stdc++.h>
using namespace std;

const int MAX_CHAR = 26;

struct Trie {

    // 'index' vectors size is greater than
    // 0 when node/ is a leaf node, otherwise
    // size is 0;
    vector<int> index;

    Trie* child[MAX_CHAR];

    /*to make new trie*/
    Trie()
    {
        // initializing fields
        for (int i = 0; i < MAX_CHAR; i++)
            child[i] = NULL;
    }
};

// function to insert a string in trie
void insert(Trie* root, string str, int index)
{
    Trie* node = root;

    for (int i = 0; i < str.size(); i++) {

        // taking ascii value to find index of
        // child node
        char ind = str[i] - 'a';

        // making a new path if not already
        if (!node->child[ind])
            node->child[ind] = new Trie();

        // go to next node
        node = node->child[ind];
    }

    // Mark leaf (end of string) and store
    // index of 'str' in index[]
    (node->index).push_back(index);
}
```

```
// function for preorder traversal of trie
void preorder(Trie* node, string arr[])
{
    // if node is empty
    if (node == NULL)
        return;

    for (int i = 0; i < MAX_CHAR; i++) {
        if (node->child[i] != NULL) {

            // if leaf node then print all the strings
            // for (node->child[i]->index).size() > 0)
            for (int j = 0; j < (node->child[i]->index).size(); j++)
                cout << arr[node->child[i]->index[j]] << " ";

            preorder(node->child[i], arr);
        }
    }
}

// function to sort an array
// of strings using Trie
void printSorted(string arr[], int n)
{
    Trie* root = new Trie();

    // insert all strings of dictionary into trie
    for (int i = 0; i < n; i++)
        insert(root, arr[i], i);

    // print strings in lexicographic order
    preorder(root, arr);
}

// Driver program to test above
int main()
{
    string arr[] = { "abc", "xyz", "abcd", "bcd", "abc" };
    int n = sizeof(arr) / sizeof(arr[0]);
    printSorted(arr, n);
    return 0;
}
```

Java

```
// Java implementation
// to sort an array of
// strings using Trie
```

```
import java.util.*;

class Trie {

    private Node rootNode;

    /*to make new trie*/
    Trie()
    {
        rootNode = null;
    }

    // function to insert
    // a string in trie
    void insert(String key, int index)
    {
        // making a new path
        // if not already
        if (rootNode == null)
        {
            rootNode = new Node();
        }

        Node currentNode = rootNode;

        for (int i = 0;i < key.length();i++)
        {
            char keyChar = key.charAt(i);

            if (currentNode.getChild(keyChar) == null)
            {
                currentNode.addChild(keyChar);
            }

            // go to next node
            currentNode = currentNode.getChild(keyChar);
        }

        // Mark leaf (end of string)
        // and store index of 'str'
        // in index[]
        currentNode.addIndex(index);
    }

    void traversePreorder(String[] array)
    {
        traversePreorder(rootNode, array);
    }
}
```

```
// function for preorder
// traversal of trie
private void traversePreorder(Node node,
                               String[] array)
{
    if (node == null)
    {
        return;
    }

    if (node.getIndices().size() > 0)
    {
        for (int index : node.getIndices())
        {
            System.out.print(array[index] + " ");
        }
    }

    for (char index = 'a';index <= 'z';index++)
    {
        traversePreorder(node.getChild(index), array);
    }
}

private static class Node {

    private Node[] children;
    private List<Integer> indices;

    Node()
    {
        children = new Node[26];
        indices = new ArrayList<>(0);
    }

    Node getChild(char index)
    {
        if (index < 'a' || index > 'z')
        {
            return null;
        }

        return children[index - 'a'];
    }

    void addChild(char index)
    {

```

```
        if (index < 'a' || index > 'z')
        {
            return;
        }

        Node node = new Node();
        children[index - 'a'] = node;
    }

    List<Integer> getIndices()
    {
        return indices;
    }

    void addIndex(int index)
    {
        indices.add(index);
    }
}

class SortStrings {

    // Driver program
    public static void main(String[] args)
    {
        String[] array = { "abc", "xyz",
                           "abcd", "bcd", "abc" };
        printInSortedOrder(array);
    }

    // function to sort an array
    // of strings using Trie
    private static void printInSortedOrder(String[] array)
    {
        Trie trie = new Trie();

        for (int i = 0;i < array.length;i++)
        {
            trie.insert(array[i], i);
        }

        trie.traversePreorder(array);
    }
}

// Contributed by Harikrishnan Rajan
```

Chapter 367. Sorting array of strings (or words) using Trie Set-2 (Handling Duplicates)

Output:

abc abc abcd bcd xyz

Source

<https://www.geeksforgeeks.org/sorting-array-strings-words-using-trie-set-2-handling-duplicates/>

Chapter 368

Sorting array using Stacks

Sorting array using Stacks - GeeksforGeeks

Given an array of elements, task is to sort these elements using stack.

Prerequisites : [Stacks](#)

Examples :

```
Input : 8 5 7 1 9 12 10
Output : 1 5 7 8 9 10 12
Explanation :
Output is sorted element set
```

```
Input : 7 4 10 20 2 5 9 1
Output : 1 2 4 5 7 9 10 20
```

We basically use [Sort a stack using a temporary stack](#). Then we put sorted stack elements back to array.

C++

```
// C++ program to sort an array using stack
#include <bits/stdc++.h>
using namespace std;

// This function return the sorted stack
stack<int> sortStack(stack<int> input)
{
    stack<int> tmpStack;

    while (!input.empty())
    {
        // pop out the first element
```

```
int tmp = input.top();
input.pop();

// while temporary stack is not empty
// and top of stack is smaller than temp
while (!tmpStack.empty() &&
       tmpStack.top() < tmp)
{
    // pop from temporary stack and
    // push it to the input stack
    input.push(tmpStack.top());
    tmpStack.pop();
}

// push temp in temporary stack
tmpStack.push(tmp);
}

return tmpStack;
}

void sortArrayUsingStacks(int arr[], int n)
{
    // Push array elements to stack
    stack<int> input;
    for (int i=0; i<n; i++)
        input.push(arr[i]);

    // Sort the temporary stack
    stack<int> tmpStack = sortStack(input);

    // Put stack elements in arrp[]
    for (int i=0; i<n; i++)
    {
        arr[i] = tmpStack.top();
        tmpStack.pop();
    }
}

// main function
int main()
{
    int arr[] = {10, 5, 15, 45};
    int n = sizeof(arr)/sizeof(arr[0]);

    sortArrayUsingStacks(arr, n);

    for (int i=0; i<n; i++)
```

```
    cout << arr[i] << " ";
```

```
    return 0;
}
```

Java

```
// Java program to sort an
// array using stack
import java.io.*;
import java.util.*;

class GFG
{
    // This function return
    // the sorted stack
    static Stack<Integer> sortStack(Stack<Integer> input)
    {
        Stack<Integer> tmpStack =
            new Stack<Integer>();

        while (!input.empty())
        {
            // pop out the
            // first element
            int tmp = input.peek();
            input.pop();

            // while temporary stack is
            // not empty and top of stack
            // is smaller than temp
            while (!tmpStack.empty() &&
                   tmpStack.peek() < tmp)
            {
                // pop from temporary
                // stack and push it
                // to the input stack
                input.push(tmpStack.peek());
                tmpStack.pop();
            }

            // push temp in
            // temporary of stack
            tmpStack.push(tmp);
        }

        return tmpStack;
    }
}
```

```
static void sortArrayUsingStacks(int []arr,
                                int n)
{
    // push array elements
    // to stack
    Stack<Integer> input =
        new Stack<Integer>();
    for (int i = 0; i < n; i++)
        input.push(arr[i]);

    // Sort the temporary stack
    Stack<Integer> tmpStack =
        sortStack(input);

    // Put stack elements
    // in arrp[]
    for (int i = 0; i < n; i++)
    {
        arr[i] = tmpStack.peek();
        tmpStack.pop();
    }
}

// Driver Code
public static void main(String args[])
{
    int []arr = {10, 5, 15, 45};
    int n = arr.length;

    sortArrayUsingStacks(arr, n);

    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

C#

```
// C# program to sort an
// array using stack
using System;
using System.Collections.Generic;

class GFG
```

```
{  
    // This function return  
    // the sorted stack  
    static Stack<int> sortStack(Stack<int> input)  
    {  
        Stack<int> tmpStack = new Stack<int>();  
  
        while (input.Count != 0)  
        {  
            // pop out the  
            // first element  
            int tmp = input.Peek();  
            input.Pop();  
  
            // while temporary stack is  
            // not empty and top of stack  
            // is smaller than temp  
            while (tmpStack.Count != 0 &&  
                  tmpStack.Peek() < tmp)  
            {  
                // pop from temporary  
                // stack and push it  
                // to the input stack  
                input.Push(tmpStack.Peek());  
                tmpStack.Pop();  
            }  
  
            // push temp in  
            // temporary stack  
            tmpStack.Push(tmp);  
        }  
  
        return tmpStack;  
    }  
  
    static void sortArrayUsingStacks(int []arr,  
                                    int n)  
    {  
        // Push array elements  
        // to stack  
        Stack<int> input = new Stack<int>();  
        for (int i = 0; i < n; i++)  
            input.Push(arr[i]);  
  
        // Sort the temporary stack  
        Stack<int> tmpStack = sortStack(input);  
  
        // Put stack elements in arrp[]
```

```
for (int i = 0; i < n; i++)
{
    arr[i] = tmpStack.Peek();
    tmpStack.Pop();
}
}

// Driver Code
static void Main()
{
    int []arr = new int[] {10, 5,
                          15, 45};
    int n = arr.Length;

    sortArrayUsingStacks(arr, n);

    for (int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

5 10 15 45

Time Complexity : O(n^*n)

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/sorting-array-using-stacks/>

Chapter 369

Sorting in Java

Sorting in Java - GeeksforGeeks

There are two in-built methods to sort in Java.

1. [Arrays.Sort\(\)](#) works for arrays which can be of primitive data type also.

```
// A sample Java program to demonstrate working of
// Arrays.sort().
// It by default sorts in ascending order.
import java.util.Arrays;

public class GFG {
    public static void main(String[] args)
    {
        int[] arr = { 13, 7, 6, 45, 21, 9, 101, 102 };

        Arrays.sort(arr);

        System.out.printf("Modified arr[] : %s",
                           Arrays.toString(arr));
    }
}
```

Output:

```
Modified arr[] : [6, 7, 9, 13, 21, 45, 101, 102]
```

2. [Collections.sort\(\)](#) works for objects Collections like ArrayList and LinkedList.

```
// Java program to demonstrate working of Collections.sort()
```

```
import java.util.*;

public class GFG {
    public static void main(String[] args)
    {
        // Create a list of strings
        ArrayList<String> al = new ArrayList<String>();
        al.add("Geeks For Geeks");
        al.add("Friends");
        al.add("Dear");
        al.add("Is");
        al.add("Superb");

        /* Collections.sort method is sorting the
        elements of ArrayList in ascending order. */
        Collections.sort(al);

        // Let us print the sorted list
        System.out.println("List after the use of"
                           + " Collection.sort() :\n" + al);
    }
}
```

- **Which sorting algorithm does Java use in sort()?**

Java's Arrays.sort method uses Quicksort for arrays of primitives and merge sort for arrays of objects.

- **Which order of sorting is done by default?**

It by default sorts in ascending order.

- **How to sort array or list in descending order?**

It can be done with the help of Collections.reverseOrder().

Example:

1. For Arrays.sort()

```
// A sample Java program to sort an array
// in descending order using Arrays.sort().
import java.util.Arrays;
import java.util.Collections;

public class GFG {
    public static void main(String[] args)
    {
        // Note that we have Integer here instead of
        // int[] as Collections.reverseOrder doesn't
        // work for primitive types.
        Integer[] arr = { 13, 7, 6, 45, 21, 9, 2, 100 };
```

```
// Sorts arr[] in descending order
Arrays.sort(arr, Collections.reverseOrder());

System.out.printf("Modified arr[] : %s",
    Arrays.toString(arr));
}
}
```

Output:

```
Modified arr[] : [100, 45, 21, 13, 9, 7, 6, 2]
```

2. For Collections.sort()

```
// Java program to demonstrate working of Collections.sort()
// to descending order.
import java.util.*;

public class GFG {
    public static void main(String[] args)
    {
        // Create a list of strings
        ArrayList<String> al = new ArrayList<String>();
        al.add("Geeks For Geeks");
        al.add("Friends");
        al.add("Dear");
        al.add("Is");
        al.add("Superb");

        /* Collections.sort method is sorting the
        elements of ArrayList in ascending order. */
        Collections.sort(al, Collections.reverseOrder());

        // Let us print the sorted list
        System.out.println("List after the use of"
            + " Collection.sort() :\n" + al);
    }
}
```

Output:

```
List after the use of Collection.sort() :
[Superb, Is, Geeks For Geeks, Friends, Dear]
```

- **How to sort only a subarray?**

Example:

```
// A sample Java program to sort a subarray
```

```
// using Arrays.sort().
import java.util.Arrays;

public class GFG {
    public static void main(String[] args)
    {
        // Our arr contains 8 elements
        int[] arr = { 13, 7, 6, 45, 21, 9, 2, 100 };

        // Sort subarray from index 1 to 4, i.e.,
        // only sort subarray {7, 6, 45, 21} and
        // keep other elements as it is.
        Arrays.sort(arr, 1, 5);

        System.out.printf("Modified arr[] : %s",
                           Arrays.toString(arr));
    }
}
```

Output:

```
Modified arr[] : [13, 6, 7, 21, 45, 9, 2, 100]
```

- **How to write my own sorting function in Java?<**
Please see Java programs for [Quick Sort](#), [Merge Sort](#), [Insertion Sort](#), [Selection Sort](#), [Heap Sort](#), [Bubble Sort](#)

Source

<https://www.geeksforgeeks.org/sorting-in-java/>

Chapter 370

Sorting of Vector of Tuple in C++ (Ascending Order)

Sorting of Vector of Tuple in C++ (Ascending Order) - GeeksforGeeks

What is Vector of Tuple?

A [tuple](#) is an object that can hold a number of elements and a [vector](#) containing multiple number of such tuple is called a vector of tuple. The elements can be of different data types. The elements of tuples are initialized as arguments in order in which they will be accessed.

```
// C++ program to demonstrate vector of tuple
#include <bits/stdc++.h>
using namespace std;
int main()
{
    vector<tuple<int, int, int> > v;
    v.push_back(make_tuple(10, 20, 30));
    v.push_back(make_tuple(15, 5, 25));
    v.push_back(make_tuple(3, 2, 1));

    // Printing vector tuples
    for (int i = 0; i < v.size(); i++)
        cout << get<0>(v[i]) << " "
            << get<1>(v[i]) << " "
            << get<2>(v[i]) << "\n";

    return 0;
}
```

Output:

```
10 20 30
15 5 25
3 2 1
```

Different ways to sort vector of tuples

Case 1 : Sorting the vector elements on the basis of first element of tuples in ascending order.

This type of sorting can be achieved using simple “ `sort()` ” function. By default the sort function sorts the vector elements on basis of first element of tuples.

```
// C++ program to demonstrate sorting in
// vector of tuple according to 1st element
// of tuple
#include <bits/stdc++.h>
using namespace std;
int main()
{
    vector<tuple<int, int, int> > v;
    v.push_back(make_tuple(10, 20, 30));
    v.push_back(make_tuple(15, 5, 25));
    v.push_back(make_tuple(3, 2, 1));

    // Using sort() function to sort by 1st
    // element of tuple
    sort(v.begin(), v.end());
    cout << "Sorted Vector of Tuple on basis"
         " of first element of tuple:\n";
    for (int i = 0; i < v.size(); i++)
        cout << get<0>(v[i]) << " "
             << get<1>(v[i]) << " "
             << get<2>(v[i]) << "\n";

    return 0;
}
```

Output:

```
Sorted Vector of Tuple on basis of first element of tuple:
3 2 1
10 20 30
15 5 25
```

Case 2 : Sorting the vector elements on the basis of second element of tuples in ascending order.

There are instances when we require to sort the elements of vector on the basis of second elements of tuples. For that, we modify the sort() function and we pass a third argument, a call to an user defined explicit function in the sort() function.

```
// C++ program to demonstrate sorting in vector
// of tuple according to 2nd element of tuples
#include <bits/stdc++.h>
using namespace std;

// Comparison function to sort the vector elements
// by second element of tuples
bool sortbysec(const tuple<int, int, int>& a,
               const tuple<int, int, int>& b)
{
    return (get<1>(a) < get<1>(b));
}

int main()
{
    vector<tuple<int, int, int> > v;
    v.push_back(make_tuple(10, 20, 30));
    v.push_back(make_tuple(15, 5, 25));
    v.push_back(make_tuple(3, 2, 1));

    // Using sort() function to sort by 2nd element
    // of tuple
    sort(v.begin(), v.end(), sortbysec);
    cout << "Sorted Vector of Tuple on basis"
         " of Second element of tuple:\n";

    for (int i = 0; i < v.size(); i++)
        cout << get<0>(v[i]) << " "
             << get<1>(v[i]) << " "
             << get<2>(v[i]) << "\n";
    return 0;
}
```

Output:

```
Sorted Vector of Tuple on basis of Second element of tuple:
3 2 1
15 5 25
10 20 30
```

Case 3 : Sorting the vector elements on the basis of third element of tuples in ascending order.

There are instances when we require to sort the elements of vector on the basis of third elements of tuples. For that, we modify the sort() function and we pass a third argument, a call to an user defined explicit function in the sort() function.

```
// C++ program to demonstrate sorting in vector
// of tuple according to 3rd element of tuple
#include <bits/stdc++.h>
using namespace std;
// Driver function to sort the vector elements
// by third element of tuple
bool sortbyth(const tuple<int, int, int>& a,
              const tuple<int, int, int>& b)
{
    return (get<2>(a) < get<2>(b));
}

int main()
{
    vector<tuple<int, int, int> > v;
    v.push_back(make_tuple(10, 20, 30));
    v.push_back(make_tuple(15, 5, 25));
    v.push_back(make_tuple(3, 2, 1));

    // Using sort() function to sort by 3rd element
    // of tuple
    sort(v.begin(), v.end(), sortbyth);
    cout << "Sorted Vector of Tuple on basis"
         " of Third element of tuple:\n";
    for (int i = 0; i < v.size(); i++)
        cout << get<0>(v[i]) << " "
             << get<1>(v[i]) << " "
             << get<2>(v[i]) << "\n";

    return 0;
}
```

Output:

```
Sorted Vector of Tuple on basis of Third element of tuple:
3 2 1
15 5 25
10 20 30
```

Improved By : [suri_kumkaran](#)

Source

<https://www.geeksforgeeks.org/sorting-vector-tuple-c-ascending-order/>

Chapter 371

Sorting rows of matrix in ascending order followed by columns in descending order

Sorting rows of matrix in ascending order followed by columns in descending order - Geeks-forGeeks

Given a matrix, sort the rows of matrix in ascending order followed by sorting the columns in descending order.

Examples :

```
Input : a[3][3] = {{1, 2, 3},  
                   {4, 5, 6},  
                   {7, 8, 9}};
```

```
Output : 7 8 9  
        4 5 6  
        1 2 3
```

```
Input : a[3][3] = {{3, 2, 1},  
                   {9, 8, 7},  
                   {6, 5, 4}};
```

```
Output : 7 8 9  
        4 5 6  
        1 2 3
```

- 1) Traverse all rows one by one and sort rows in ascending order using simple array sort.
- 2) Convert matrix to its [transpose](#)
- 3) Again sort all rows, but this time in ascending order.
- 4) Again convert matrix to its [transpose](#)

C++

```
// C++ implementation to sort the rows
// of matrix in ascending order followed by
// sorting the columns in descending order
#include <bits/stdc++.h>
using namespace std;

#define MAX_SIZE 10

// function to sort each row of the matrix
// according to the order specified by
// ascending.
void sortByRow(int mat[] [MAX_SIZE], int n,
               bool ascending)
{
    for (int i = 0; i < n; i++)
    {
        if (ascending)
            sort(mat[i], mat[i] + n);
        else
            sort(mat[i], mat[i] + n, greater<int>());
    }
}

// function to find transpose of the matrix
void transpose(int mat[] [MAX_SIZE], int n)
{
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)

            // swapping element at index (i, j)
            // by element at index (j, i)
            swap(mat[i][j], mat[j][i]);
}

// function to sort the matrix row-wise
// and column-wise
void sortMatRowAndColWise(int mat[] [MAX_SIZE],
                           int n)
{
    // sort rows of mat[] []
    sortByRow(mat, n, true);

    // get transpose of mat[] []
    transpose(mat, n);

    // again sort rows of mat[] [] in descending
    // order.
    sortByRow(mat, n, false);
}
```

```
// again get transpose of mat[] []
transpose(mat, n);
}

// function to print the matrix
void printMat(int mat[][] [MAX_SIZE], int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << mat[i][j] << " ";
        cout << endl;
    }
}

// Driver program to test above
int main()
{
    int n = 3;

    int mat[n][MAX_SIZE] = {{3, 2, 1},
                           {9, 8, 7},
                           {6, 5, 4}};

    cout << "Original Matrix:\n";
    printMat(mat, n);

    sortMatRowAndColWise(mat, n);

    cout << "\nMatrix After Sorting:\n";
    printMat(mat, n);

    return 0;
}
```

Java

```
// Java implementation to sort the rows
// of matrix in ascending order followed by
// sorting the columns in descending order
import java.util.Arrays;
import java.util.Collections;

class GFG
{
    static int MAX_SIZE=10;

    // function to sort each row of the matrix
```

```
// according to the order specified by
// ascending.
static void sortByRow(Integer mat[][] , int n,
                      boolean ascending)
{
    for (int i = 0; i < n; i++)
    {
        if (ascending)
            Arrays.sort(mat[i]);
        else
            Arrays.sort(mat[i], Collections.reverseOrder());
    }
}

// function to find transpose of the matrix
static void transpose(Integer mat[][] , int n)
{
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
        {
            // swapping element at index (i, j)
            // by element at index (j, i)
            int temp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = temp;
        }
}

// function to sort the matrix row-wise
// and column-wise
static void sortMatRowAndColWise(Integer mat[][] ,
                                   int n)
{
    // sort rows of mat[][]
    sortByRow(mat, n, true);

    // get transpose of mat[][]
    transpose(mat, n);

    // again sort rows of mat[][]
    // in descending
    // order.
    sortByRow(mat, n, false);

    // again get transpose of mat[][]
    transpose(mat, n);
}

// function to print the matrix
```

```
static void printMat(Integer mat[][] , int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            System.out.print(mat[i][j] + " ");
        System.out.println();
    }
}

// Driver code
public static void main (String[] args)
{
    int n = 3;

    Integer mat[][] = {{3, 2, 1},
                       {9, 8, 7},
                       {6, 5, 4}};

    System.out.print("Original Matrix:\n");
    printMat(mat, n);

    sortMatRowAndColWise(mat, n);

    System.out.print("\nMatrix After Sorting:\n");
    printMat(mat, n);
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python implementation to sort the rows
# of matrix in ascending order followed by
# sorting the columns in descending order

MAX_SIZE=10

# function to sort each row of the matrix
# according to the order specified by
# ascending.
def sortByRow(mat, n, ascending):

    for i in range(n):
        if (ascending):
            mat[i].sort()
        else:
```

```
mat[i].sort(reverse=True)

# function to find
# transpose of the matrix
def transpose(mat, n):

    for i in range(n):
        for j in range(i + 1, n):

            # swapping element at index (i, j)
            # by element at index (j, i)
            temp = mat[i][j]
            mat[i][j] = mat[j][i]
            mat[j][i] = temp

# function to sort
# the matrix row-wise
# and column-wise
def sortMatRowAndColWise(mat, n):

    # sort rows of mat[] []
    sortByRow(mat, n, True)

    # get transpose of mat[] []
    transpose(mat, n)

    # again sort rows of
    # mat[] [] in descending
    # order.
    sortByRow(mat, n, False)

    # again get transpose of mat[] []
    transpose(mat, n)

# function to print the matrix
def printMat(mat, n):

    for i in range(n):
        for j in range(n):
            print(mat[i][j], " ", end="")
        print()

#Driver code
n = 3

mat = [[3, 2, 1],
       [9, 8, 7],
       [6, 5, 4]]
```

```
print("Original Matrix:")
printMat(mat, n)

sortMatRowAndColWise(mat, n)

print("Matrix After Sorting:")
printMat(mat, n)

# This code is contributed
# by Anant Agarwal.
```

Output :

```
Original Matrix:
3 2 1
9 8 7
6 5 4

Matrix After Sorting:
7 8 9
4 5 6
1 2 3
```

Source

<https://www.geeksforgeeks.org/sorting-rows-matrix-ascending-order-followed-sorting-columns-descending-order/>

Chapter 372

Sorting using trivial hash function

Sorting using trivial hash function - GeeksforGeeks

Sorting using [trivial hash](#) function.

Examples:

```
Input : 9 4 3 5 8  
Output : 3 4 5 8 9
```

We have read about various sorting algorithms such as [heap sort](#), [bubble sort](#), [merge sort](#) and others.

Here we will see how can we sort N elements using hash array. But this algorithm has a limitation. We can sort only those N elements, where the value of elements is not large (typically not above 10^6).

Explanation of sorting using hash:

Step 1: create a hash array of size(max_element), since that is the maximum we will need

Step 2: traverse through all the elements and keep a count of number of occurrence of a particular element.

Step 3: after keep a count of occurrence of all elements in the hash table, simply iterate from 0 to max_element in the hash array

Step 4: while iterating in the hash array, if we find the value stored at any hash position is more than 0, which indicated that the element is present at least once in the original list of elements.

Step 5: Hash[i] has the count of the number of times a element is present in the list, so when its >0, we print those number of times the element.

If you want to store the elements, use another array to store them in a sorted way.

If we want to sort it in a descending order, we simply traverse from max to 0, and repeat the same procedure.

Below is the c++ implementation of the above approach:

```
// C++ program to sort an array using hash
// function
#include <bits/stdc++.h>
using namespace std;

void sortUsingHash(int a[], int n)
{
    // find the maximum element
    int max = *std::max_element(a, a + n);

    // create a hash function upto the max size
    int hash[max + 1] = { 0 };

    // traverse through all the elements and
    // keep a count
    for (int i = 0; i < n; i++)
        hash[a[i]] += 1;

    // Traverse upto all elements and check if
    // it is present or not. If it is present,
    // then print the element the number of times
    // it's present. Once we have printed n times,
    // that means we have printed n elements
    // so break out of the loop
    for (int i = 0; i <= max; i++) {

        // if present
        if (hash[i]) {

            // print the element that number of
            // times it's present
            for (int j = 0; j < hash[i]; j++) {
                cout << i << " ";
            }
        }
    }
}

// driver program
int main()
{
    int a[] = { 9, 4, 3, 2, 5, 2, 1, 0, 4,
               3, 5, 10, 15, 12, 18, 20, 19 };
    int n = sizeof(a) / sizeof(a[0]);

    sortUsingHash(a, n);
}
```

```
    return 0;
}
```

Output:

```
0 1 2 2 3 3 4 4 5 5 9 10 12 15 18 19 20
```

How to handle negative numbers?

In case the array has **negative numbers** and **positive** numbers, we keep two hash arrays to keep a track of positive and negative elements.

Explanation of sorting using hashing if the array has negative and positive numbers:

Step 1: Create two hash arrays, one for positive and the other for negative

Step 2: the positive hash array will have a size of max and the negative array will have a size of min

Step 3: traverse from min to 0 in the negative hash array, and print the elements in the same way we did for positives.

Step 4: Traverse from 0 to max for positive elements and print them in the same manner as explained above.

Below is the c++ implementation of the above approach:

```
// C++ program to sort an array using hash
// function with negative values allowed.
#include <bits/stdc++.h>
using namespace std;

void sortUsingHash(int a[], int n)
{
    // find the maximum element
    int max = *std::max_element(a, a + n);
    int min = abs(*std::min_element(a, a + n));

    // create a hash function upto the max size
    int hashpos[max + 1] = { 0 };
    int hashneg[min + 1] = { 0 };

    // traverse through all the elements and
    // keep a count
    for (int i = 0; i < n; i++) {
        if (a[i] >= 0)
            hashpos[a[i]] += 1;
        else
            hashneg[abs(a[i])] += 1;
    }

    // Traverse up to all negative elements and
```

```
// check if it is present or not. If it is
// present, then print the element the number
// of times it's present. Once we have printed
// n times, that means we have printed n elements
// so break out of the loop
for (int i = min; i > 0; i--) {
    if (hashneg[i]) {

        // print the element that number of times
        // it's present. Print the negative element
        for (int j = 0; j < hashneg[i]; j++) {
            cout << (-1) * i << " ";
        }
    }
}

// Traverse upto all elements and check if it is
// present or not. If it is present, then print
// the element the number of times it's present
// once we have printed n times, that means we
// have printed n elements, so break out of the
// loop
for (int i = 0; i <= max; i++) {

    // if present
    if (hashpos[i]) {

        // print the element that number of times
        // it's present
        for (int j = 0; j < hashpos[i]; j++) {
            cout << i << " ";
        }
    }
}

// driver program to test the above function
int main()
{
    int a[] = { -1, -2, -3, -4, -5, -6, 8, 7,
               5, 4, 3, 2, 1, 0 };
    int n = sizeof(a) / sizeof(a[0]);
    sortUsingHash(a, n);
    return 0;
}
```

Output:

-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 7 8

Complexity:

This sort function can have complexity $O(\max_element)$. So performance depends on that set of data provided.

Limitations:

1. Can only sort array elements of limited range (typically from -10^6 to $+10^6$)
2. Auxiliary space in worst cases is $O(\max_element) + O(\min_element)$

Source

<https://www.geeksforgeeks.org/sorting-using-trivial-hash-function/>

Chapter 373

Sorting without comparison of elements

Sorting without comparison of elements - GeeksforGeeks

Given an array with integer elements in small range, sort the array. We need to write a non-comparison based sorting algorithm with following assumptions about input.

1. All the entries in the array are integers.
2. The difference between the maximum value and the minimum value in the array is less than or equal to 10^6 .

```
Input : arr[] = {10, 30, 20, 4}
Output : 4 10 20 30
```

```
Input : arr[] = {10, 30, 1, 20, 4}
Output : 1 4 10 20 30
```

We are not allowed to use comparison based sorting algorithms like [QuickSort](#), [MergeSort](#), etc.

Since elements are small, we use array elements as index. We store element counts in a count array. Once we have count array, we traverse the count array and print every present element its count times.

```
// CPP program to sort an array without comparison
// operator.
#include <bits/stdc++.h>
using namespace std;

int sortArr(int arr[], int n, int min, int max)
```

```
{  
    // Count of elements in given range  
    int m = max - min + 1;  
  
    // Count frequencies of all elements  
    vector<int> c(m, 0);  
    for (int i=0; i<n; i++)  
        c[arr[i] - min]++;  
  
    // Traverse through range. For every  
    // element, print it its count times.  
    for (int i=0; i<=m; i++)  
        for (int j=0; j < c[i]; j++)  
            cout << (i + min) << " ";  
}  
  
int main()  
{  
    int arr[] = {10, 10, 1, 4, 4, 100, 0};  
    int min = 0, max = 100;  
    int n = sizeof(arr)/sizeof(arr[0]);  
    sortArr(arr, n, min, max);  
    return 0;  
}
```

Output:

0 1 4 4 10 10 100

What is time complexity?

Time complexity of above algorithm is $O(n + (\max - \min))$

Is above algorithm stable?

The above implementation is not stable as we do not care about order of same elements while sorting.

How to make above algorithm stable?

The stable version of above algorithm is called [Counting Sort](#). In counting sort, we store sums of all smaller or equal values in $c[i]$ so that $c[i]$ store actual position of i in sorted array. After filling $c[]$, we traverse input array again, place every element at its position and decrement count.

What are non-comparison based standard algorithms?

[Counting Sort](#), [Radix Sort](#) and [Bucket Sort](#).

Source

<https://www.geeksforgeeks.org/sorting-without-comparison-of-elements/>

Chapter 374

Sorting Natural Language Programming

Sorting Natural Language Programming - GeeksforGeeks

The only compound data types native to Osmosian Plain English are records and doubly-linked lists. When we need to sort a list, we use the simple recursive merge sort that I'll describe below. But first we need something to sort. Let's sort fruits, and let's begin with a type definition:

A fruit is a thing with a name.

When the word “thing” appears in a type definition, our compiler works a little magic behind the scenes to make things (pun intended) more powerful and flexible. The above definition, for example, actually causes the following data types to be defined:

fruit

@fruit record

fruit record

secret prefix	name		
@next fruit	@prev fruit	@first byte	@last byte

fruits

first	last
@first fruit	@last fruit

So a Fruit is really nothing but a pointer containing the address of a Fruit Record.

And each 16-byte Fruit Record has a hidden 8-byte prefix with two pointers for linking these records into lists, together with the fruit's name, which is a string. Plain English strings are stored in the Heap and can be any length. So the Name in the Fruit Record is actually just two pointers to the first and last bytes of the string on the Heap, respectively. *String* memory is managed automatically, but *thing* memory is managed by the programmer.

The third type generated by the compiler serves as the anchor for lists of Fruit Records. Such lists are simply (and intuitively) called Fruits, the plural of Fruit.

Now let's add some fruits to a list, in random order, and sort them. Here are the top level sentences in our test program:

To run:
Start up.
Create some fruits.
Write the fruits on the console.
Skip a line on the console.
Sort the fruits.
Write the fruits on the console.
Destroy the fruits.
Wait for the escape key.
Shut down.

And here are the routines that will be called to "Create some fruits":

```
To create some fruits:  
Add "banana" to the fruits.  
Add "apple" to the fruits.  
Add "orange" to the fruits.  
Add "bacon" to the fruits.  
Add "pineapple" to the fruits.  
Add "pomegranate" to the fruits.  
Add "tomato" to the fruits.  
Add "grape" to the fruits.  
Add "fig" to the fruits.  
Add "date" to the fruits.
```

```
To add a name to some fruits:  
Allocate memory for a fruit.  
Put the name into the fruit's name.  
Append the fruit to the fruits.
```

Now we're ready to sort. This is the sort routine:

```
To sort some fruits:  
If the fruits' first is the fruits' last, exit.  
Split the fruits into some left fruits and some right fruits.  
Sort the left fruits.  
Sort the right fruits.  
Loop.  
Put the left fruits' first into a left fruit.  
Put the right fruits' first into a right fruit.  
If the left fruit is nil, append the right fruits to the fruits; exit.  
If the right fruit is nil, append the left fruits to the fruits; exit.  
If the left fruit's name is greater than the right fruit's name,  
    move the right fruit from the right fruits to the fruits; repeat.  
Move the left fruit from the left fruits to the fruits.  
Repeat.
```

When we run this program, the output on the console looks like this:

```
banana
apple
orange
bacon
pineapple
pomegranate
tomato
grape
fig
date

apple
bacon
banana
date
fig
grape
orange
pineapple
pomegranate
tomato
```

But is it fast? Let's see using this modified test program:

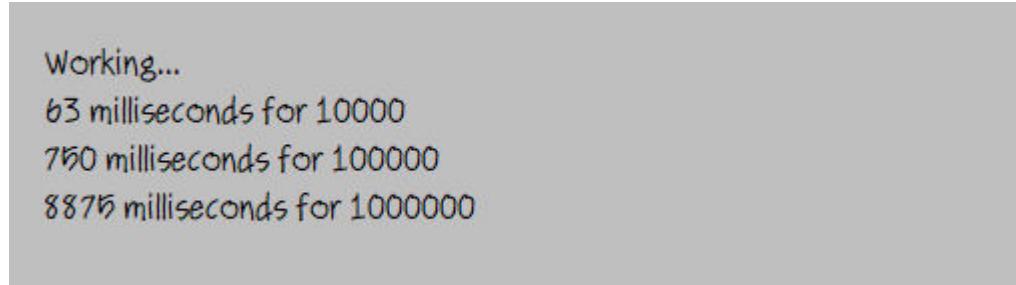
```
To run:
Start up.
Write "Working..." on the console.
Put 10000 into a count.
Create some fruits using "apple" and the count.
Start a timer. Sort the fruits. Stop the timer.
Write the timer then " milliseconds for " then the count on the console.
Destroy the fruits.
Put 100000 into the count.
Create the fruits using "apple" and the count.
Start the timer. Sort the fruits. Stop the timer.
```

Write the timer then " milliseconds for " then the count on the console.
Destroy the fruits.
Put 1000000 into the count.
Create the fruits using "apple" and the count.
Start the timer. Sort the fruits. Stop the timer.
Write the timer then " milliseconds for " then the count on the console.
Destroy the fruits.
Wait for the escape key.
Shut down.

The fruits this time, at the start, look like this:

```
apple 0010000  
apple 0009999  
apple 0009998  
...
```

And the output on the console looks like this:



```
Working...
63 milliseconds for 10000
750 milliseconds for 100000
8875 milliseconds for 1000000
```

Not quite linear, I admit. But not exponentially bad, either. Ten times as many records take *roughly* ten times as long to sort. There's a technical way of saying this using big "O's" and little "n's" and "logs" and stuff, but Plain English programmers don't generally think that way. And it's stable, as a Plain English programmer would expect — records with duplicate sort values retain their original sequence.

Good, simple, useful stuff.

Source

<https://www.geeksforgeeks.org/natural-language-programming-sorting/>

Chapter 375

Stability in sorting algorithms

Stability in sorting algorithms - GeeksforGeeks

What is it?

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted.

Formally stability may be defined as,

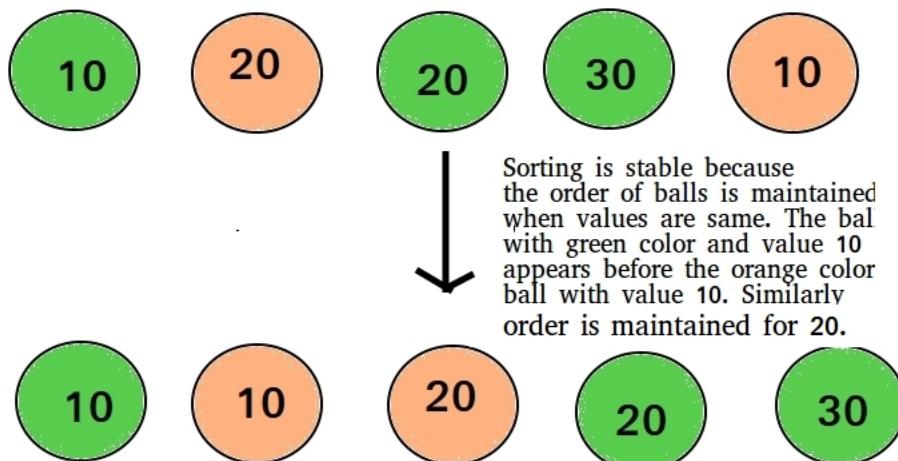
Let A be an array, and let \prec be a strict weak ordering on the elements of A .

A sorting algorithm is stable if

$i \prec j$ and $A[\pi(i)] = A[i]$ implies $\pi(\pi(i)) \prec \pi(j)$
 $A[\pi(i)] = \pi(i)$

where π is the sorting permutation (sorting moves $A[i]$ to position $\pi(i)$)

Informally, stability means that equivalent elements retain their relative positions, after sorting.



Do we care for simple arrays like array of integers?

When equal elements are indistinguishable, such as with integers, or more generally, any data where the entire element is the key, stability is not an issue. Stability is also not an issue if all keys are different.

An example where it is useful

Consider the following dataset of Student Names and their respective class sections.

(*Eric*, *A*)
 (*Eric*, *B*)
 (*Ken*, *A*)
 (*Eric*, *B*)
 (*Ken*, *A*)

If we sort this data according to name only, then it is highly unlikely that the resulting dataset will be grouped according to sections as well.

(*Eric*, *A*)
 (*Eric*, *B*)
 (*Dave*, *A*)
 (*Eric*, *B*)
 (*Ken*, *A*)

So we might have to sort again to obtain list of students section wise too. But in doing so, if the sorting algorithm is not stable, we might get a result like this-

(*Eric*, *A*)
 (*Eric*, *B*)
 (*Dave*, *A*)
 (*Ken*, *A*)
 (*Eric*, *B*)
 (*Aliya*, *B*)

The dataset is now sorted according to sections, but not according to names.

In the name-sorted dataset, the tuple (*Eric*, *A*) was before (*Eric*, *B*), but since the sorting algorithm is not stable, the relative order is lost.

If on the other hand we used a stable sorting algorithm, the result would be-

(Alice, A)
 (Bob, B)
 $(\text{Charlie}, \text{C})$
 (Alice, B)
 (Bob, B)

Here the relative order between different tuples is maintained. It may be the case that the relative order is maintained in an Unstable Sort but that is highly unlikely.

Which sorting algorithms are stable?

Some Sorting Algorithms are stable by nature, such as [Bubble Sort](#), [Insertion Sort](#), [Merge Sort](#), [Count Sort](#) etc.

Comparison based stable sorts such as Merge Sort and Insertion Sort, maintain stability by ensuring that-

Element $A[i]$ comes before $A[j]$ if and only if $A[i] < A[j]$, here i, j are indices and $i < j$.

Since $i < j$, the relative order is preserved $A[i] < A[j] \Rightarrow A[i] \leq A[j]$ i.e. $A[i]$ comes before $A[j]$.

Other non-comparison based sorts such as [Counting Sort](#) maintain stability by ensuring that the Sorted Array is filled in a reverse order so that elements with equivalent keys have the same relative position.

Some sorts such as [Radix Sort](#) depend on another sort, with the only requirement that the other sort should be stable.

Which sorting algorithms are unstable?

[Quick Sort](#), [Heap Sort](#) etc., can be made stable by also taking the position of the elements into consideration. This change may be done in a way which does not compromise a lot on

the performance and takes some extra space, possibly $\Theta(n^2)$.

Can we make any sorting algorithm stable?

Any given sorting algo which is not stable can be modified to be stable. There can be sorting algo specific ways to make it stable, but in general, any comparison based sorting algorithm which is not stable by nature can be modified to be stable by changing the key comparison operation so that the comparison of two keys considers position as a factor for objects with equal keys.

References:

- <http://www.math.uic.edu/~leon/cs-mcs401-s08/handouts/stability.pdf>
- http://en.wikipedia.org/wiki/Sorting_algorithm#Stability

Source

<https://www.geeksforgeeks.org/stability-in-sorting-algorithms/>

Chapter 376

Stable Selection Sort

Stable Selection Sort - GeeksforGeeks

A sorting algorithm is said to be **stable** if two objects with equal or same keys appear in the same order in sorted output as they appear in the input array to be sorted.

Any comparison based sorting algorithm which is not stable by nature can be modified to be stable by changing the key comparison operation so that the comparison of two keys considers position as a factor for objects with equal key or by tweaking it in a way such that its meaning doesn't change and it becomes stable as well.

Example :

Note: Subscripts are only used for understanding the concept.

Input : 4A 5 3 2 4B 1
Output : 1 2 3 4B 4A 5

Stable Selection Sort would have produced
Output : 1 2 3 4A 4B 5

Selection sort works by finding the minimum element and then inserting it in its correct position by swapping with the element which is in the position of this minimum element. This is what makes it unstable.

Swapping might impact in pushing a key(let's say A) to a position greater than the key(let's say B) which are equal keys. which makes them out of desired order.

In the above example 4A was pushed after 4B and after complete sorting this 4A remains after this 4B. Hence resulting in instability.

Selection sort can be made Stable if instead of swapping, the minimum element is placed in its position without swapping i.e. by placing the number in its position by pushing every element one step forward.

In simple terms use a technique like [insertion sort](#) which means inserting element in its correct place.

EXPLANATION WITH EXAMPLE:

Example: 4A 5 3 2 4B 1

First minimum element is 1, now instead of swapping. Insert 1 in its correct place and pushing every element one step forward i.e forward pushing.

1 4A 5 3 2 4B

Next minimum is 2 :

1 2 4A 5 3 4B

Next minimum is 3 :

1 2 3 4A 5 3 2 4B

Repeat the steps until array is sorted.

1 2 3 4A 4B 5

C++

```
// C++ program for modifying Selection Sort
// so that it becomes stable.
#include <iostream>
using namespace std;

void stableSelectionSort(int a[], int n)
{
    // Iterate through array elements
    for (int i = 0; i < n - 1; i++)
    {

        // Loop invariant : Elements till a[i - 1]
        // are already sorted.

        // Find minimum element from
        // arr[i] to arr[n - 1].
        int min = i;
        for (int j = i + 1; j < n; j++)
            if (a[min] > a[j])
                min = j;

        // Move minimum element at current i.
        int key = a[min];
        while (min > i)
        {
            a[min] = a[min - 1];
            min--;
        }
    }
}
```

```
        }
        a[i] = key;
    }
}

void printArray(int a[], int n)
{
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int a[] = { 4, 5, 3, 2, 4, 1 };
    int n = sizeof(a) / sizeof(a[0]);
    stableSelectionSort(a, n);
    printArray(a, n);
    return 0;
}
```

Java

```
// Java program for modifying Selection Sort
// so that it becomes stable.
class GFG
{
    static void stableSelectionSort(int[] a, int n)
    {
        // Iterate through array elements
        for (int i = 0; i < n - 1; i++)
        {

            // Loop invariant : Elements till
            // a[i - 1] are already sorted.

            // Find minimum element from
            // arr[i] to arr[n - 1].
            int min = i;
            for (int j = i + 1; j < n; j++)
                if (a[min] > a[j])
                    min = j;

            // Move minimum element at current i.
            int key = a[min];
            while (min > i)
            {
```

```
a[min] = a[min - 1];
min--;
}

a[i] = key;
}
}

static void printArray(int[] a, int n)
{
    for (int i = 0; i < n; i++)
        System.out.print(a[i]+ " ");

    System.out.println();
}

// Driver code
public static void main (String[] args)
{
    int[] a = { 4, 5, 3, 2, 4, 1 };
    int n = a.length;
    stableSelectionSort(a, n);
    printArray(a, n);
}
}

// This code is contributed by Mr. Somesh Awasthi
```

C#

```
// C# program for modifying Selection Sort
// so that it becomes stable.
using System;

class GFG
{
    static void stableSelectionSort(int[] a, int n)
    {
        // Iterate through array elements
        for (int i = 0; i < n - 1; i++)
        {

            // Loop invariant : Elements till
            // a[i - 1] are already sorted.

            // Find minimum element from
            // arr[i] to arr[n - 1].
            int min = i;
```

```
for (int j = i + 1; j < n; j++)
    if (a[min] > a[j])
        min = j;

    // Move minimum element at current i.
    int key = a[min];
    while (min > i)
    {
        a[min] = a[min - 1];
        min--;
    }

    a[i] = key;
}

static void printArray(int[] a, int n)
{
    for (int i = 0; i < n; i++)
        Console.WriteLine(a[i] + " ");

    Console.WriteLine();
}

// Driver code
public static void Main ()
{
    int[] a = { 4, 5, 3, 2, 4, 1 };
    int n = a.Length;
    stableSelectionSort(a, n);
    printArray(a, n);
}
}

// This code is contributed by vt_m.
```

Output:

1 2 3 4 4 5

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/stable-selection-sort/>

Chapter 377

Stable sort for descending order

Stable sort for descending order - GeeksforGeeks

Given an array of n integers, we have to reverse sort the array elements such the equal keys are stable after sorting.

Examples:

Input : arr[] = {4, 2, 3, 2, 4}
Output : 4, 4, 3, 2, 2

Prerequisite : [Stability in sorting algorithms](#)

Method 1 (Writing our own sorting function : Bubble Sort)

We know sorting algorithms like [Bubble Sort](#), [Insertion Sort](#), [Merge Sort](#), [Count Sort](#) are stable. We implement here Bubble Sort.

Explanation

First Pass

(4, 2, 3, 2, 4) -> (2, 4, 3, 4, 2) Here algorithm compares last two element and swaps since $2 < 4$.

(2, 4, 3, 4, 2) -> (2, 4, 4, 3, 2) swap since $3 < 4$

(2, 4, 4, 3, 2) -> (2, 4, 4, 3, 2)

(2, 4, 4, 3, 2) -> (4, 2, 4, 3, 2) swap since $2 < 4$.

Second Pass:

(4, 2, 4, 3, 2) -> (4, 2, 4, 3, 2)

(4, 2, 4, 3, 2) -> (4, 2, 4, 3, 2)

(4, 2, 4, 3, 2) -> (4, 4, 2, 3, 2) swap since $2 < 4$.

Third Pass:

(4, 4, 2, 3, 2) -> (4, 4, 2, 3, 2)

(4, 4, 2, 3, 2) -> (4, 4, 3, 2, 2) swap since $2 < 3$

Now, the array is in sorted order and same elements are in same order as they were in the original array.

C++

```
// Bubble sort implementation to sort
// elements in descending order.
#include <iostream>
#include <vector>
using namespace std;

void print(vector<int> a, int n)
{
    for (int i = 0; i <= n; i++)
        cout << a[i] << " ";
    cout << endl;
}

// Sorts a[] in descending order using
// bubble sort.
void sort(vector<int> a, int n)
{
    for (int i = n; i >= 0; i--)
        for (int j = n; j > n - i; j--)
            if (a[j] > a[j - 1])
                swap(a[j], a[j-1]);
    print(a, n);
}

// Driver code
int main()
{
    int n = 7;
    vector<int> a;
    a.push_back(2);
    a.push_back(4);
    a.push_back(3);
    a.push_back(2);
    a.push_back(4);
    a.push_back(5);
    a.push_back(3);
    sort(a, n - 1);
    return 0;
}
```

Java

```
// Bubble sort implementation
// to sort elements in
// descending order.
```

```
import java.io.*;
import java.util.*;

class GFG
{
    static void print(ArrayList<Integer> a,
                      int n)
    {
        for (int i = 0; i <= n; i++)
            System.out.print(a.get(i) + " ");
        System.out.println();
    }

    // Sorts a[] in descending
    // order using bubble sort.
    static void sort(ArrayList<Integer> a,
                     int n)
    {
        for (int i = n;
             i >= 0; i--)
            for (int j = n;
                 j > n - i; j--)
                if (a.get(j) > a.get(j - 1))
                {
                    int tempswap = a.get(j);
                    a.remove(j);
                    a.add(j, a.get(j - 1));
                    a.remove(j - 1);
                    a.add(j - 1, tempswap);
                }
        print(a, n);
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 6;
        ArrayList<Integer> a = new ArrayList<Integer>();
        a.add(2);
        a.add(4);
        a.add(3);
        a.add(2);
        a.add(4);
        a.add(5);
        a.add(3);
        sort(a, n);
    }
}
```

```
// This code is contributed by  
// Manish Shaw(manishshaw1)
```

Python3

```
# Bubble sort implementation to sort  
# elements in descending order.  
  
def print1(a, n):  
  
    for i in range(0,n+1):  
        print(a[i],end=" ")  
    print("")  
  
# Sorts a[] in descending order using  
# bubble sort.  
def sort(a, n):  
  
    for i in range(n,0,-1):  
        for j in range(n, n - i,-1):  
            if (a[j] > a[j - 1]):  
                a[j], a[j-1]=a[j-1], a[j]  
    print1(a,n)  
  
# Driver code  
n = 7  
a = [2,4,3,2,4,5,3]  
  
sort(a, n-1)  
  
# This code is contributed  
# by Smitha Dinesh Semwal
```

C#

```
// Bubble sort implementation  
// to sort elements in  
// descending order.  
using System;  
using System.Collections.Generic;  
  
class GFG  
{  
static void print(List<int> a,
```

```
        int n)
{
    for (int i = 0; i <= n; i++)
        Console.Write(a[i] + " ");
    Console.WriteLine();
}

// Sorts a[] in descending
// order using bubble sort.
static void sort(List<int> a,
                 int n)
{
    for (int i = n;
         i >= 0; i--)
        for (int j = n;
             j > n - i; j--)
            if (a[j] > a[j - 1])
            {
                int tempswap = a[j];
                a[j] = a[j - 1];
                a[j - 1] = tempswap;
            }
    print(a, n);
}

// Driver code
static void Main()
{
    int n = 6;
    List<int> a = new List<int>();
    a.Add(2);
    a.Add(4);
    a.Add(3);
    a.Add(2);
    a.Add(4);
    a.Add(5);
    a.Add(3);
    sort(a, n);
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

PHP

```
<?php
// Bubble sort implementation
```

```
// to sort elements in
// descending order.

function swap(&$x, &$y)
{
    $x ^= $y ^= $x ^= $y;
}

function print1($a, $n)
{
    for ($i = 0; $i <= $n; $i++)
        echo ($a[$i] . " ");
    echo ("\n");
}

// Sorts a[] in descending
// order using bubble sort.
function sort1($a, $n)
{
    for ($i = $n;
         $i >= 0; $i--)
    {
        for ($j = $n;
             $j > $n - $i; $j--)
        {
            if ($a[$j] > $a[$j - 1])
                swap($a[$j],
                      $a[$j - 1]);
        }
    }
    print1($a, $n);
}

// Driver code
$n = 6;
$a = array();
array_push($a, 2);
array_push($a, 4);
array_push($a, 3);
array_push($a, 2);
array_push($a, 4);
array_push($a, 5);
array_push($a, 3);
sort1($a, $n);

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output:

```
5 4 4 3 3 2 2
```

Method 2 (Using library function)

We can use `stable_sort` to sort elements in stable manner.

C++

```
// C++ program to demonstrate descending order
// stable sort using greater<>().
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int n = sizeof(arr) / sizeof(arr[0]);

    stable_sort(arr, arr + n, greater<int>());

    cout << "Array after sorting : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
```

Output:

```
Array after sorting :
9 8 7 6 5 4 3 2 1 0
```

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/stable-sort-descending-order/>

Chapter 378

Stooge Sort

Stooge Sort - GeeksforGeeks

The Stooge sort is a recursive sorting algorithm. It is defined as below (for ascending order sorting).

Step 1 : If value at index 0 is greater than
value at last index, swap them.
Step 2: Recursively,
a) Stooge sort the initial 2/3rd of the array.
b) Stooge sort the last 2/3rd of the array.
c) Stooge sort the initial 2/3rd again to confirm.

NOTE: Always take the ceil of $((2/3)^N)$ for selecting elements.

Illustration:

Input : 2 4 5 3 1

Output : 1 2 3 4 5

Explanation:

Initially, swap 2 and 1 following above step 1.

1 4 5 3 2

Now, recursively sort initial 2/3rd of the elements.

1 4 5 3 2

1 3 4 5 2

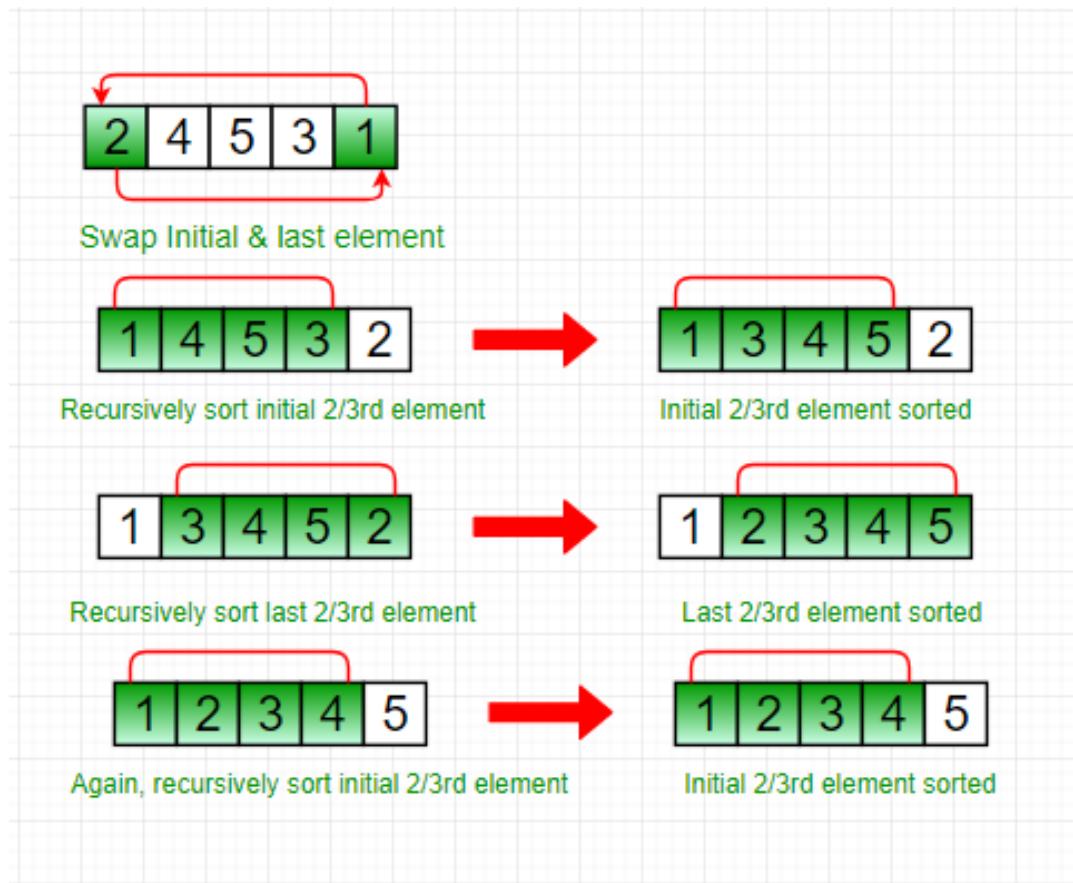
Then, recursively sort last 2/3rd of the elements.

1 3 4 5 2

1 2 3 4 5

Again, sort the initial 2/3rd of the elements to confirm final data is sorted.

1 2 3 4 5



C++

```
// C++ code to implement stooge sort
#include <iostream>
using namespace std;

// Function to implement stooge sort
void stoogesort(int arr[], int l, int h)
{
    if (l >= h)
        return;

    // If first element is smaller than last,
    // swap them
    if (arr[l] > arr[h])
        swap(arr[l], arr[h]);

    // If there are more than 2 elements in
    // the array
    if (h - l + 1 > 2) {

```

```
int t = (h - l + 1) / 3;

// Recursively sort first 2/3 elements
stoogesort(arr, l, h - t);

// Recursively sort last 2/3 elements
stoogesort(arr, l + t, h);

// Recursively sort first 2/3 elements
// again to confirm
stoogesort(arr, l, h - t);
}

}

// Driver Code
int main()
{
    int arr[] = { 2, 4, 5, 3, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // Calling Stooge Sort function to sort
    // the array
    stoogesort(arr, 0, n - 1);

    // Display the sorted array
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Java

```
// Java program to implement stooge sort
import java.io.*;

public class stooge {
    // Function to implement stooge sort
    static void stoogesort(int arr[], int l, int h)
    {
        if (l >= h)
            return;

        // If first element is smaller
        // than last, swap them
        if (arr[l] > arr[h]) {
            int t = arr[l];
            arr[l] = arr[h];
            arr[h] = t;
        }

        stoogesort(arr, l, h - 1);
        stoogesort(arr, l + 1, h);
    }
}
```

```
        arr[h] = t;
    }

    // If there are more than 2 elements in
    // the array
    if (h - l + 1 > 2) {
        int t = (h - l + 1) / 3;

        // Recursively sort first 2/3 elements
        stoogesort(arr, l, h - t);

        // Recursively sort last 2/3 elements
        stoogesort(arr, l + t, h);

        // Recursively sort first 2/3 elements
        // again to confirm
        stoogesort(arr, l, h - t);
    }
}

// Driver Code
public static void main(String args[])
{
    int arr[] = { 2, 4, 5, 3, 1 };
    int n = arr.length;

    stoogesort(arr, 0, n - 1);

    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}
}

// Code Contributed by Mohit Gupta_OMG <(0_o)>
```

Python3

```
# Python program to implement stooge sort

def stoogesort(arr, l, h):
    if l >= h:
        return

    # If first element is smaller
    # than last, swap them
    if arr[l]>arr[h]:
        t = arr[l]
        arr[l] = arr[h]
        arr[h] = t

    stoogesort(arr, l+1, h)
```

```
# If there are more than 2 elements in
# the array
if h-l + 1 > 2:
    t = (int)((h-l + 1)/3)

    # Recursively sort first 2 / 3 elements
    stoogesort(arr, l, (h-t))

    # Recursively sort last 2 / 3 elements
    stoogesort(arr, l + t, (h))

    # Recursively sort first 2 / 3 elements
    # again to confirm
    stoogesort(arr, l, (h-t))

# deriver
arr = [2, 4, 5, 3, 1]
n = len(arr)

stoogesort(arr, 0, n-1)

for i in range(0, n):
    print(arr[i], end = ' ')
```

Code Contributed by Mohit Gupta_OMG <(0_o)>

C#

```
// C# program to implement stooge sort
using System;

class GFG {

    // Function to implement stooge sort
    static void stoogesort(int[] arr,
                          int l, int h)
    {
        if (l >= h)
            return;

        // If first element is smaller
        // than last, swap them
        if (arr[l] > arr[h]) {
            int t = arr[l];
            arr[l] = arr[h];
            arr[h] = t;
        }

        stoogesort(arr, l, h-1);
        stoogesort(arr, l+1, h);
```

```
}

// If there are more than 2
// elements in the array
if (h - l + 1 > 2) {
    int t = (h - l + 1) / 3;

    // Recursively sort first
    // 2/3 elements
    stoogesort(arr, l, h - t);

    // Recursively sort last
    // 2/3 elements
    stoogesort(arr, l + t, h);

    // Recursively sort first
    // 2/3 elements again to
    // confirm
    stoogesort(arr, l, h - t);
}

}

// Driver Code
public static void Main()
{
    int[] arr = { 2, 4, 5, 3, 1 };
    int n = arr.Length;

    // Calling Stooge Sort function
    // to sort the array
    stoogesort(arr, 0, n - 1);

    // Display the sorted array
    for (int i = 0; i < n; i++)
        Console.Write(arr[i] + " ");
}

}

// This code is contributed by Sam007.
```

Output:

1 2 3 4 5

The running time complexity of stooge sort can be written as,

$$T(n) = 3T(3n/2) + ?(1)$$

Solution of above recurrence is $O(n^{(\log 3 / \log 1.5)}) = O(n^{2.709})$, hence it is slower than even bubble sort(n^2).

Reference:

[Wikipedia](#)

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/stooge-sort/>

Chapter 379

Strand Sort

Strand Sort - GeeksforGeeks

Given a list of items, sort them in increasing order.

Examples:

```
Input : ip[] = {10, 5, 30, 40, 2, 4, 9}
Output : op[] = {2, 4, 5, 9, 10, 30, 40}
```

```
Input : ip[] = {1, 10, 7}
Output : op[] = {1, 7, 10}
```

Strand Sort is a sorting algorithm that works in $O(n)$ time if list is already sorted and works in $O(n^2)$ in worst case.

Below are simple steps used in the algorithm.

1. Let ip[] be input list and op[] be output list.
2. Create an empty sublist and move first item of ip[] to it.
3. Traverse remaining items of ip. For every item x, check if x is greater than last inserted item to sublist. If yes, remove x from ip and add at the end of sublist. If no, ignore x (Keep it in ip)
4. Merge sublist into op (output list)
5. Recur for remaining items in ip and current items in op.

Illustration :

```
Let ip[] = {10, 5, 30, 40, 2, 4, 9}
Initialize : op[] = {}
              sublist[] = {}

Move first item of ip to sublist.
sublist[] = {10}

Traverse remaining items of ip and
if current element is greater than
last item of sublist, move this item
from ip to sublist. Now
sublist[] = {10, 30, 40}
ip[] = {5, 2, 4, 9}

Merge sublist into op.
op = {10, 30, 40}

Next recursive call:
Move first item of ip to sublist.
sublist[] = {5}

Traverse remaining items of ip and move
elements greater than last inserted.
ip[] = {2, 4}
sublist[] = {5, 9}

Merge sublist into op.
op = {5, 9, 10, 30, 40}

Last Recursive Call:
{2, 4} are first moved to sublist and
then merged into op.
op = {2, 4, 5, 9, 10, 30, 40}
```

Below is the implementation of above algorithm. The C++ implementation uses [list in C++ STL](#).

```
// CPP program to implement Strand Sort
#include <bits/stdc++.h>
using namespace std;

// A recursive function to implement Strand
// sort.
// ip is input list of items (unsorted).
// op is output list of items (sorted)
void strandSort(list<int> &ip, list<int> &op)
{
```

```
// Base case : input is empty
if (ip.empty())
    return;

// Create a sorted sublist with
// first item of input list as
// first item of the sublist
list<int> sublist;
sublist.push_back(ip.front());
ip.pop_front();

// Traverse remaining items of ip list
for (auto it = ip.begin(); it != ip.end(); ) {

    // If current item of input list
    // is greater than last added item
    // to sublist, move current item
    // to sublist as sorted order is
    // maintained.
    if (*it > sublist.back()) {
        sublist.push_back(*it);

        // erase() on list removes an
        // item and returns iterator to
        // next of removed item.
        it = ip.erase(it);
    }

    // Otherwise ignore current element
    else
        it++;
}

// Merge current sublist into output
op.merge(sublist);

// Recur for remaining items in
// input and current items in op.
strandSort(ip, op);
}

// Driver code
int main(void)
{
    list<int> ip{10, 5, 30, 40, 2, 4, 9};

    // To store sorted output list
    list<int> op;
```

```
// Sorting the list
strandSort(ip, op);

// Printing the sorted list
for (auto x : op)
    cout << x << " ";
return 0;
}
```

Output:

2 4 5 9 10 30 40

More Sorting Algorithms :

- Selection Sort
- Bubble Sort
- Recursive Bubble Sort
- Insertion Sort
- Recursive Insertion Sort
- Merge Sort
- Iterative Merge Sort
- Quick Sort
- Iterative Quick Sort
- Heap Sort
- Counting Sort
- Radix Sort
- Bucket Sort
- ShellSort
- TimSort
- Comb Sort
- Pigeonhole Sort
- Cycle Sort

- Bitonic Sort
- Cocktail Sort
- Pancake sorting
- Binary Insertion Sort
- BogoSort or Permutation Sort
- Gnome Sort
- Sleep Sort – The King of Laziness / Sorting while Sleeping
- Structure Sorting (By Multiple Rules) in C++
- Stooge Sort
- Tag Sort (To get both sorted and original)
- Tree Sort
- Cartesian Tree Sorting
- Odd-Even Sort / Brick Sort
- QuickSort on Singly Linked List
- QuickSort on Doubly Linked List
- 3-Way QuickSort (Dutch National Flag)
- Merge Sort for Linked Lists
- Merge Sort for Doubly Linked List
- 3-way Merge Sort

[Practice Problems on Sorting](#)

Source

<https://www.geeksforgeeks.org/strand-sort/>

Chapter 380

Structure Sorting (By Multiple Rules) in C++

Structure Sorting (By Multiple Rules) in C++ - GeeksforGeeks

Prerequisite : [Structures in C](#)

Name and marks in different subjects (physics, chemistry and maths) are given for all students. The task is to compute total marks and ranks of all students. And finally display all students sorted by rank.

Rank of student is computed using below rules.

1. If total marks are different, then students with higher marks gets better rank.
2. If total marks are same, then students with higher marks in Maths gets better rank.
3. If total marks are same and marks in Maths are also same, then students with better marks in Physics gets better rank.
4. If all marks (total, Maths, Physics and Chemistry) are same, then any student can be assigned better rank.

We use below structure to store details of students.

```
struct Student
{
    string name; // Given
    int math; // Marks in math (Given)
    int phy; // Marks in Physics (Given)
    int che; // Marks in Chemistry (Given)
    int total; // Total marks (To be filled)
    int rank; // Rank of student (To be filled)
};
```

We use `std::sort()` for **Structure Sorting**. In Structure sorting, all the respective properties possessed by the structure object are sorted on the basis of one (or more) property of the object.

In this example, marks of students in different subjects are provided by user. These marks in individual subjects are added to calculate the total marks of the student, which is then used to sort different students on the basis of their ranks (as explained above).

```
// C++ program to demonstrate structure sorting in C++
#include <bits/stdc++.h>
using namespace std;

struct Student
{
    string name; // Given
    int math; // Marks in math (Given)
    int phy; // Marks in Physics (Given)
    int che; // Marks in Chemistry (Given)
    int total; // Total marks (To be filled)
    int rank; // Rank of student (To be filled)
};

// Function for comparing two students according
// to given rules
bool compareTwoStudents(Student a, Student b)
{
    // If total marks are not same then
    // returns true for higher total
    if (a.total != b.total)
        return a.total > b.total;

    // If marks in Maths are not same then
    // returns true for higher marks
    if (a.math != b.math)
        return a.math > b.math;

    return (a.phy > b.phy);
}

// Fills total marks and ranks of all Students
void computeRanks(Student a[], int n)
{
    // To calculate total marks for all Students
    for (int i=0; i<n; i++)
        a[i].total = a[i].math + a[i].phy + a[i].che;

    // Sort structure array using user defined
    // function compareTwoStudents()
    sort(a, a+n, compareTwoStudents);
}
```

```
// Assigning ranks after sorting
for (int i=0; i<n; i++)
    a[i].rank = i+1;
}

// Driver code
int main()
{
    int n = 5;

    // array of structure objects
    Student a[n];

    // Details of Student 1
    a[0].name = "Bryan" ;
    a[0].math = 80 ;
    a[0].phy = 95 ;
    a[0].che = 85 ;

    // Details of Student 2
    a[1].name= "Kevin" ;
    a[1].math= 95 ;
    a[1].phy= 85 ;
    a[1].che= 99 ;

    // Details of Student 3
    a[2].name = "Nick" ;
    a[2].math = 95 ;
    a[2].phy = 85 ;
    a[2].che = 80 ;

    // Details of Student 4
    a[3].name = "AJ" ;
    a[3].math = 80 ;
    a[3].phy = 70 ;
    a[3].che = 90 ;

    // Details of Student 5
    a[4].name = "Howie" ;
    a[4].math = 80 ;
    a[4].phy = 80 ;
    a[4].che = 80 ;

    computeRanks(a, n);

    //Column names for displaying data
    cout << "Rank" <<"t" << "Name" << "t";
```

```
cout << "Maths" <<"t" <<"Physics" <<"t"
      << "Chemistry";
cout << "t" << "Totaln";

// Display details of Students
for (int i=0; i<n; i++)
{
    cout << a[i].rank << "t";
    cout << a[i].name << "t";
    cout << a[i].math << "t"
        << a[i].phy << "t"
        << a[i].che << "tt";
    cout << a[i].total <<"t";
    cout <<"n";
}

return 0;
}
```

Output:

Rank	Name	Maths	Physics	Chemistry	Total
1	Kevin	95	85	99	279
2	Nick	95	85	80	260
3	Bryan	80	95	85	260
4	Howie	80	80	80	240
5	AJ	80	70	90	240

Related Articles:

[sort\(\) in C++ STL](#)

[Comparator function of qsort\(\) in C](#)

[C qsort\(\) vs C++ sort\(\)](#)

[Sort an array according to count of set bits](#)

Source

<https://www.geeksforgeeks.org/structure-sorting-in-c/>

Chapter 381

Substring Sort

Substring Sort - GeeksforGeeks

Given n strings, we need to sort those strings such that every string is a substring of all strings **after** it. If not possible to sort, then print the same.

Examples:

Input : {"d", "zddsaaz", "ds", "ddsaa", "dds"}

Output :

d
ds
dds
ddsaa
zddsaaz

Input : {"geeks", "ee", "geeksforgeeks", "forgeeks", "ee"}

Output :

ee
ee
geeks
forgeeks
geeksforgeeks

Observation 1

If A substring of B

Then length of A <= length of B

Observation 2

If (A substring of B) and (B substring of C)

Then A substring of C

Solution

Based on the two above observations, the solutions is as follows

1. Sort all the string from the shorter to the longer
2. Validate that each string is a substring of the following string

If we validate that each string is a substring of the following string then based on observation 2, each string is a substring of all the strings come after it.

```
// Java code to sort substrings
import java.util.Arrays;
import java.util.Comparator;

public class Demo {
    public static void substringSort(String[] arr, int n)
    {
        // sort the given array from shorter string to longer
        Arrays.sort(arr, new Comparator<String>() {
            public int compare(String s1, String s2)
            {
                return Integer.compare(s1.length(), s2.length());
            }
        });

        // validate that each string is a substring of
        // the following one'
        for (int i = 0; i < n - 1; i++) {
            if (!arr[i + 1].contains(arr[i])) {

                // the array cann't be sorted
                System.out.println("Cannot be sorted");
                return;
            }
        }

        // The array is valid and sorted
        // print the strings in order
        for (int i = 0; i < n - 1; i++) {
            System.out.println(arr[i]);
        }
    }

    public static void main(String[] args)
    {
        // Test 1
        String[] arr1 = { "d", "zddsaaz", "ds", "ddsaa", "dds" };
        substringSort(arr1, arr1.length);

        // Test 2
        String[] arr2 = { "for", "rof" };
    }
}
```

```
        substringSort(arr2, arr2.length);
    }
}
```

Output:

```
d
ds
dds
ddsaa
Cannot be sorted
```

Complexity

Time Complexity: $O(n \log n)$, where n is the number of strings.

Alternative approach

For better time complexity, we can use [counting sort](#) only if the maximum length of the strings is specified.

Suppose that ‘maxLen’ is the maximum length of the input strings. In this case, the solution is as follows:

1. Create array of length maxLen
2. Sort the input strings such that the string with length 1 is in the first place in the array
3. If there is two or more string has the same length they must be equal otherwise the strings cannot be sorted
4. Validate that each string is a substring of the next longer string

```
// Alternative code to sort substrings
import java.util.Arrays;

public class Demo {

    public static void substringSort(String[] arr, int n, int maxLen)
    {

        int count[] = new int[maxLen];
        String[] sortedArr = new String[maxLen];

        Arrays.fill(count, 0);
        Arrays.fill(sortedArr, "");

        // sort the input array
        for (int i = 0; i < n; i++) {
```

```
String s = arr[i];
int len = s.length();

if (count[len - 1] == 0) {
    sortedArr[len - 1] = s;
    count[len - 1] = 1;
}
else if (sortedArr[len - 1].equals(s)) {

    // repeated length should be the same string
    count[len - 1]++;
}
else {

    // two different strings with the same
    // length input array cannot be sorted
    System.out.println("Cannot be sorted");
    return;
}
}

// validate that each string is a substring
// of the following one
int index = 0;

// get first element
while (count[index] == 0)
    index++;

int prev = index;
String prevString = sortedArr[prev];

index++;

for (; index < maxLen; index++) {

    if (count[index] != 0) {
        String current = sortedArr[index];
        if (current.contains(prevString)) {
            prev = index;
            prevString = current;
        }
        else {
            System.out.println("Cannot be sorted");
            return;
        }
    }
}
```

```
}

// The array is valid and sorted
// print the strings in order
for (int i = 0; i < maxLen; i++) {
    String s = sortedArr[i];
    for (int j = 0; j < count[i]; j++) {
        System.out.println(s);
    }
}

public static void main(String[] args)
{
    int maxLen = 100;

    // Test 1
    String[] arr1 = { "d", "zddsaaz", "ds", "ddsaa",
                      "dds", "dds" };
    substringSort(arr1, arr1.length, maxLen);

    // Test 2
    String[] arr2 = { "for", "rof" };
    substringSort(arr2, arr2.length, maxLen);
}
```

Output:

```
d
ds
dds
dds
ddsaa
zddsaaz
Cannot be sorted
```

Source

<https://www.geeksforgeeks.org/substring-sort/>

Chapter 382

Sum of Areas of Rectangles possible for an array

Sum of Areas of Rectangles possible for an array - GeeksforGeeks

Given an array, task is to compute the sum of all possible maximum area rectangle which can be formed from the array elements. Also, you can reduce the elements of the array by at most 1.

Examples :

```
Input :a = {10, 10, 10, 10, 11,
           10, 11, 10}
Output : 210
Explanation :
We can form two rectangles one square (10 * 10)
and one (11 * 10). Hence, total area = 100 + 110 = 210.
```

```
Input : a = { 3, 4, 5, 6 }
Output : 15
Explanation :
We can reduce 4 to 3 and 6 to 5 so that we got
rectangle of (3 * 5). Hence area = 15.
```

```
Input : a = { 3, 2, 5, 2 }
Output : 0
```

Naive Approach : Check for all possible four elements of the array and then whichever can form a rectangle. In these rectangles, separate all those rectangles which are of maximum area formed by these elements. After getting the rectangles and their areas, sum them all to get our desired solution.

Efficient Approach : To get the maximum area rectangle, first sort the elements of the array in non-increasing array. After sorting, start the procedure to select the elements of the array. Here, selection of two elements of array (as length of rectangle) is possible if elements of array are equal ($a[i] == a[i+1]$) or if length of smaller element $a[i+1]$ is one less than $a[i]$ (*in this case we have our length $a[i+1]$ because $a[i]$ is decreased by 1*). One flag variable is maintained to check that *whether we get length and breadth both*. After getting length, set the flag variable, now calculate the breadth in the same way as we have done for length and sum the area of rectangle. After getting length and breadth both, again set the flag variable false so that we will now search for a new rectangle. This process is repeated and lastly, final sum of area is returned.

C++

```
// CPP code to find sum of all
// area rectangle possible
#include <bits/stdc++.h>
using namespace std;

// Function to find
// area of rectangles
int MaxTotalRectangleArea(int a[],
                           int n)
{
    // sorting the array in
    // descending order
    sort(a, a + n, greater<int>());

    // store the final sum of
    // all the rectangles area
    // possible
    int sum = 0;
    bool flag = false;

    // temporary variable to store
    // the length of rectangle
    int len;

    for (int i = 0; i < n; i++)
    {
        // Selecting the length of
        // rectangle so that difference
        // between any two number is 1
        // only. Here length is selected
        // so flag is set
        if ((a[i] == a[i + 1] || a[i] -
             a[i + 1] == 1) && (!flag))
        {
            // flag is set means
            // that we have found
            // a pair of numbers
            // which can form a
            // rectangle with
            // length as a[i]
            // and breadth as
            // a[i + 1]
            // so calculate
            // their product
            // and add it to
            // sum
            sum += a[i] * a[i + 1];
            flag = true;
        }
    }
}
```

```
// we have got length of
// rectangle
flag = true;

// length is set to
// a[i+1] so that if
// a[i] a[i+1] is less
// than by 1 then also
// we have the correct
// choice for length
len = a[i + 1];

// incrementing the counter
// one time more as we have
// considered a[i+1] element
// also so.
i++;
}

// Selecting the width of rectangle
// so that difference between any
// two number is 1 only. Here width
// is selected so now flag is again
// unset for next rectangle
else if ((a[i] == a[i + 1] ||
         a[i] - a[i + 1] == 1) && (flag))
{
    // area is calculated for
    // rectangle
    sum = sum + a[i + 1] * len;

    // flag is set false
    // for another rectangle
    // which we can get from
    // elements in array
    flag = false;

    // incrementing the counter
    // one time more as we have
    // considered a[i+1] element
    // also so.
    i++;
}
}

return sum;
}
```

```
// Driver code
int main()
{
    int a[] = { 10, 10, 10, 10,
                11, 10, 11, 10,
                9, 9, 8, 8 };
    int n = sizeof(a) / sizeof(a[0]);

    cout << MaxTotalRectangleArea(a, n);

    return 0;
}
```

Java

```
// Java code to find sum of
// all area rectangle possible
import java.io.*;
import java.util.Arrays;

class GFG
{
    // Function to find
    // area of rectangles
    static int MaxTotalRectangleArea(int []a,
                                    int n)
    {

        // sorting the array in
        // descending order
        Arrays.sort(a);

        // store the final sum of
        // all the rectangles area
        // possible
        int sum = 0;
        boolean flag = false;

        // temporary variable to
        // store the length of rectangle
        int len = 0;

        for (int i = 0; i < n; i++)
        {

            // Selecting the length of
            // rectangle so that difference
            // between any two number is 1
```

```
// only. Here length is selected
// so flag is set
if ((a[i] == a[i + 1] ||
     a[i] - a[i + 1] == 1) &&
    !flag)
{
    // flag is set means
    // we have got length of
    // rectangle
    flag = true;

    // length is set to
    // a[i+1] so that if
    // a[i] a[i+1] is less
    // than by 1 then also
    // we have the correct
    // choice for length
    len = a[i + 1];

    // incrementing the counter
    // one time more as we have
    // considered a[i+1] element
    // also so.
    i++;
}

// Selecting the width of rectangle
// so that difference between any
// two number is 1 only. Here width
// is selected so now flag is again
// unset for next rectangle
else if ((a[i] == a[i + 1] ||
          a[i] - a[i + 1] == 1) &&
          (flag))
{
    // area is calculated for
    // rectangle
    sum = sum + a[i + 1] * len;

    // flag is set false
    // for another rectangle
    // which we can get from
    // elements in array
    flag = false;

    // incrementing the counter
    // one time more as we have
    // considered a[i+1] element
```

```
// also so.
    i++;
}
}

return sum;
}

// Driver code
public static void main (String args[])
{
int []a = { 10, 10, 10, 10,
           11, 10, 11, 10,
           9, 9, 8, 8 };
int n = a.length;

System.out.print(MaxTotalRectangleArea(a, n));
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Python3

```
# Python3 code to find sum
# of all area rectangle
# possible

# Function to find
# area of rectangles
def MaxTotalRectangleArea(a, n) :

    # sorting the array in
    # descending order
    a.sort(reverse = True)

    # store the final sum of
    # all the rectangles area
    # possible
    sum = 0
    flag = False

    # temporary variable to store
    # the length of rectangle
    len = 0
    i = 0

    while (i < n-1) :
```

```
if(i != 0) :
    i = i + 1

# Selecting the length of
# rectangle so that difference
# between any two number is 1
# only. Here length is selected
# so flag is set
if ((a[i] == a[i + 1] or
     a[i] - a[i + 1] == 1)
     and flag == False) :

    # flag is set means
    # we have got length of
    # rectangle
    flag = True

    # length is set to
    # a[i+1] so that if
    # a[i+1] is less than a[i]
    # by 1 then also we have
    # the correct choice for length
    len = a[i + 1]

    # incrementing the counter
    # one time more as we have
    # considered a[i+1] element
    # also so.
    i = i + 1

# Selecting the width of rectangle
# so that difference between any
# two number is 1 only. Here width
# is selected so now flag is again
# unset for next rectangle
elif ((a[i] == a[i + 1] or
      a[i] - a[i + 1] == 1)
      and flag == True) :

    # area is calculated for
    # rectangle
    sum = sum + a[i + 1] * len

    # flag is set false
    # for another rectangle
    # which we can get from
    # elements in array
    flag = False
```

```
# incrementing the counter
# one time more as we have
# considered a[i+1] element
# also so.
i = i + 1

return sum

# Driver code
a = [ 10, 10, 10, 10, 11, 10,
      11, 10, 9, 9, 8, 8 ]
n = len(a)

print (MaxTotalRectangleArea(a, n))

# This code is contributed by
# Manish Shaw (manishshaw1)

C#
// C# code to find sum of all area rectangle
// possible
using System;

class GFG {

    // Function to find
    // area of rectangles
    static int MaxTotalRectangleArea(int []a,
                                      int n)
    {

        // sorting the array in descending
        // order
        Array.Sort(a);

        // store the final sum of all the
        // rectangles area possible
        int sum = 0;
        bool flag = false;

        // temporary variable to store the
        // length of rectangle
        int len = 0;

        for (int i = 0; i < n; i++)
        {
```

```
// Selecting the length of
// rectangle so that difference
// between any two number is 1
// only. Here length is selected
// so flag is set
if ((a[i] == a[i + 1] || a[i] -
    a[i + 1] == 1) && (!flag))
{
    // flag is set means
    // we have got length of
    // rectangle
    flag = true;

    // length is set to
    // a[i+1] so that if
    // a[i] a[i+1] is less
    // than by 1 then also
    // we have the correct
    // choice for length
    len = a[i + 1];

    // incrementing the counter
    // one time more as we have
    // considered a[i+1] element
    // also so.
    i++;
}

// Selecting the width of rectangle
// so that difference between any
// two number is 1 only. Here width
// is selected so now flag is again
// unset for next rectangle
else if ((a[i] == a[i + 1] ||
          a[i] - a[i + 1] == 1) && (flag))
{
    // area is calculated for
    // rectangle
    sum = sum + a[i + 1] * len;

    // flag is set false
    // for another rectangle
    // which we can get from
    // elements in array
    flag = false;

    // incrementing the counter
```

```
// one time more as we have
// considered a[i+1] element
// also so.
i++;
}
}

return sum;
}

// Driver code
static public void Main ()
{
    int []a = { 10, 10, 10, 10,
                11, 10, 11, 10,
                9, 9, 8, 8 };
    int n = a.Length;

    Console.WriteLine(
        MaxTotalRectangleArea(a, n));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP code to find sum
// of all area rectangle
// possible

// Function to find
// area of rectangles
function MaxTotalRectangleArea( $a, $n)
{
    // sorting the array in
    // descending order
    rsort($a);

    // store the final sum of
    // all the rectangles area
    // possible
    $sum = 0;
    $flag = false;

    // temporary variable to store
    // the length of rectangle
```

```
$len;

for ( $i = 0; $i < $n; $i++)
{
    // Selecting the length of
    // rectangle so that difference
    // between any two number is 1
    // only. Here length is selected
    // so flag is set
    if (($a[$i] == $a[$i + 1] or $a[$i] -
        $a[$i + 1] == 1) and (!$flag))
    {
        // flag is set means
        // we have got length of
        // rectangle
        $flag = true;

        // length is set to
        // a[i+1] so that if
        // a[i+1] is less than a[i]
        // by 1 then also we have
        // the correct choice for length
        $len = $a[$i + 1];

        // incrementing the counter
        // one time more as we have
        // considered a[i+1] element
        // also so.
        $i++;
    }

    // Selecting the width of rectangle
    // so that difference between any
    // two number is 1 only. Here width
    // is selected so now flag is again
    // unset for next rectangle
    else if (($a[$i] == $a[$i + 1] or
        $a[$i] - $a[$i + 1] == 1) and
        ($flag))
    {
        // area is calculated for
        // rectangle
        $sum = $sum + $a[$i + 1] * $len;

        // flag is set false
        // for another rectangle
        // which we can get from
```

```
// elements in array
$flag = false;

// incrementing the counter
// one time more as we have
// considered a[i+1] element
// also so.
$i++;
}

}

return $sum;
}

// Driver code
$a = array( 10, 10, 10, 10,
           11, 10, 11, 10,
           9, 9, 8, 8 );
$n = count($a);

echo MaxTotalRectangleArea($a, $n);

//This code is contributed by anuj_67.
?>
```

Output :

282

Time Complexity : **O(nlog(n))**
Auxiliary Space : **O(1)**

Improved By : [vt_m](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/sum-area-rectangles-possible-array/>

Chapter 383

Sum of Manhattan distances between all pairs of points

Sum of Manhattan distances between all pairs of points - GeeksforGeeks

Given n integer coordinates. The task is to find sum of manhattan distance between all pairs of coordinates.

Manhattan Distance between two points (x_1, y_1) and (x_2, y_2) is:

$$|x_1 - x_2| + |y_1 - y_2|$$

Examples :

```
Input : n = 4
        point1 = { -1, 5 }
        point2 = { 1, 6 }
        point3 = { 3, 5 }
        point4 = { 2, 3 }

Output : 22
```

Distance of { 1, 6 }, { 3, 5 }, { 2, 3 } from
{ -1, 5 } are 3, 4, 5 respectively.
Therefore, sum = 3 + 4 + 5 = 12

Distance of { 3, 5 }, { 2, 3 } from { 1, 6 }
are 3, 4 respectively.
Therefore, sum = 12 + 3 + 4 = 19

Distance of { 2, 3 } from { 3, 5 } is 3.
Therefore, sum = 19 + 3 = 22.

Method 1: (Brute Force)

The idea is to run two nested loop i.e for each each point, find manhattan distance for all other points.

```
for (i = 1; i < n; i++)
    for (j = i + 1; j < n; j++)
        sum += ((xi - xj) + (yi - yj))
```

Below is the implementation of this approach:

C++

```
// CPP Program to find sum of Manhattan distance
// between all the pairs of given points
#include <bits/stdc++.h>
using namespace std;

// Return the sum of distance between all
// the pair of points.
int distancesum(int x[], int y[], int n)
{
    int sum = 0;

    // for each point, finding distance to
    // rest of the point
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            sum += (abs(x[i] - x[j]) +
                    abs(y[i] - y[j]));
    return sum;
}

// Driven Program
int main()
{
    int x[] = { -1, 1, 3, 2 };
    int y[] = { 5, 6, 5, 3 };
    int n = sizeof(x) / sizeof(x[0]);
    cout << distancesum(x, y, n) << endl;
    return 0;
}
```

Java

```
// Java Program to find sum of Manhattan distance
// between all the pairs of given points

import java.io.*;

class GFG {
```

```
// Return the sum of distance between all
// the pair of points.
static int distancesum(int x[], int y[], int n)
{
    int sum = 0;

    // for each point, finding distance to
    // rest of the point
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            sum += (Math.abs(x[i] - x[j]) +
                    Math.abs(y[i] - y[j]));

    return sum;
}

// Driven Program
public static void main(String[] args)
{
    int x[] = { -1, 1, 3, 2 };
    int y[] = { 5, 6, 5, 3 };
    int n = x.length;

    System.out.println(distancesum(x, y, n));
}
}

// This code is contributed by vt_m.
```

Python3

```
# Python3 code to find sum of
# Manhattan distance between all
# the pairs of given points

# Return the sum of distance
# between all the pair of points.
def distancesum (x, y, n):
    sum = 0

    # for each point, finding distance
    # to rest of the point
    for i in range(n):
        for j in range(i+1,n):
            sum += (abs(x[i] - x[j]) +
                    abs(y[i] - y[j]))

    return sum
```

```
# Driven Code
x = [ -1, 1, 3, 2 ]
y = [ 5, 6, 5, 3 ]
n = len(x)
print(distancesum(x, y, n))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# Program to find sum of Manhattan distance
// between all the pairs of given points
using System;

class GFG {

    // Return the sum of distance between all
    // the pair of points.
    static int distancesum(int []x, int []y, int n)
    {
        int sum = 0;

        // for each point, finding distance to
        // rest of the point
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                sum += (Math.Abs(x[i] - x[j]) +
                        Math.Abs(y[i] - y[j]));

        return sum;
    }

    // Driven Program
    public static void Main()
    {
        int []x = { -1, 1, 3, 2 };
        int []y = { 5, 6, 5, 3 };
        int n = x.Length;

        Console.WriteLine(distancesum(x, y, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
```

```
// PHP Program to find sum
// of Manhattan distance
// between all the pairs
// of given points

// Return the sum of distance
// between all the pair of points.
function distancesum( $x, $y, $n)
{
    $sum = 0;

    // for each point, finding
    // distance to rest of
    // the point
    for($i = 0; $i < $n; $i++)
        for ($j = $i + 1; $j < $n; $j++)
            $sum += (abs($x[$i] - $x[$j]) +
                      abs($y[$i] - $y[$j]));

    return $sum;
}

// Driver Code
$x = array(-1, 1, 3, 2);
$y = array(5, 6, 5, 3);
$n = count($x);
echo distancesum($x, $y, $n);

// This code is contributed by anuj_67.
?>
```

Output:

22

Time Complexity: $O(n^2)$

Method 2: (Efficient Approach)

The idea is to use Greedy Approach. First observe, the manhattan formula can be decomposed into two independent sums, one for the difference between x coordinates and the second between y coordinates. If we know how to compute one of them we can use the same method to compute the other. So now we will stick to compute the sum of x coordinates distance.

Let's assume that we know all distances from a point x_i to all values of x 's smaller than x_i . Let's consider other points, the first one not smaller than x_i , and call it x_j . How to compute the distances from x_j to all smaller points ? We can use the corresponding distances from from x_i . Notice that each distance from x_j to some x_k , where $x_k < x_j$ equals the distance from x_i to x_k plus the distance between x_j and x_i . If there are A points smaller than x_j

and \mathbf{S} is the sum of distances from \mathbf{x}_i to smaller points, then the sum of distances from \mathbf{x}_j to smaller points equals $\mathbf{S} + (\mathbf{x}_j - \mathbf{x}_i) * \mathbf{A}$.

If we sort all points in non-decreasing order, we can easily compute the desired sum of distances along one axis between each pair of coordinates in $O(N)$ time, processing points from left to right and using the above method.

Also, we don't have to concern if two points are equal coordinates, after sorting points in non-decreasing order, we say that a point \mathbf{x}_i is smaller \mathbf{x}_j if and only if it appears earlier in the sorted array.

Below is the implementation of this approach:

C++

```
// CPP Program to find sum of Manhattan
// distances between all the pairs of
// given points
#include <bits/stdc++.h>
using namespace std;

// Return the sum of distance of one axis.
int distancesum(int arr[], int n)
{
    // sorting the array.
    sort(arr, arr + n);

    // for each point, finding the distance.
    int res = 0, sum = 0;
    for (int i = 0; i < n; i++) {
        res += (arr[i] * i - sum);
        sum += arr[i];
    }

    return res;
}

int totaldistancesum(int x[], int y[], int n)
{
    return distancesum(x, n) + distancesum(y, n);
}

// Driven Program
int main()
{
    int x[] = { -1, 1, 3, 2 };
    int y[] = { 5, 6, 5, 3 };
    int n = sizeof(x) / sizeof(x[0]);
    cout << totaldistancesum(x, y, n) << endl;
    return 0;
}
```

Java

```
// Java Program to find sum of Manhattan
// distances between all the pairs of
// given points

import java.io.*;
import java.util.*;

class GFG {

    // Return the sum of distance of one axis.
    static int distancesum(int arr[], int n)
    {

        // sorting the array.
        Arrays.sort(arr);

        // for each point, finding the distance.
        int res = 0, sum = 0;
        for (int i = 0; i < n; i++) {
            res += (arr[i] * i - sum);
            sum += arr[i];
        }

        return res;
    }

    static int totaldistancesum(int x[],
                                int y[], int n)
    {
        return distancesum(x, n) +
               distancesum(y, n);
    }

    // Driven Program
    public static void main(String[] args)
    {

        int x[] = { -1, 1, 3, 2 };
        int y[] = { 5, 6, 5, 3 };
        int n = x.length;
        System.out.println(totaldistancesum(x,
                                             y, n));
    }
}

// This code is contributed by vt_m.
```

Python3

```
# Python3 code to find sum of Manhattan
# distances between all the pairs of
# given points

# Return the sum of distance of one axis.
def distancesum (arr, n):

    # sorting the array.
    arr.sort()

    # for each point, finding
    # the distance.
    res = 0
    sum = 0
    for i in range(n):
        res += (arr[i] * i - sum)
        sum += arr[i]

    return res

def totaldistancesum( x , y , n ):
    return distancesum(x, n) + distancesum(y, n)

# Driven Code
x = [ -1, 1, 3, 2 ]
y = [ 5, 6, 5, 3 ]
n = len(x)
print(totaldistancesum(x, y, n) )

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# Program to find sum of Manhattan
// distances between all the pairs of
// given points

using System;

class GFG {

    // Return the sum of distance of one axis.
    static int distancesum(int []arr, int n)
    {
```

```
// sorting the array.  
Array.Sort(arr);  
  
// for each point, finding the distance.  
int res = 0, sum = 0;  
for (int i = 0; i < n; i++) {  
    res += (arr[i] * i - sum);  
    sum += arr[i];  
}  
  
return res;  
}  
  
static int totaldistancesum(int []x,  
                           int []y, int n)  
{  
    return distancesum(x, n) +  
           distancesum(y, n);  
}  
  
// Driven Program  
public static void Main()  
{  
  
    int []x = { -1, 1, 3, 2 };  
    int []y = { 5, 6, 5, 3 };  
    int n = x.Length;  
    Console.WriteLine(totaldistancesum(x,  
                                       y, n));  
}  
}  
  
// This code is contributed by vt_m.
```

PHP

```
<?php  
// PHP Program to find sum of  
// Manhattan distances between  
// all the pairs of given points  
  
// Return the sum of  
// distance of one axis.  
function distancesum($arr, $n)  
{  
    // sorting the array.  
    sort($arr);
```

```
// for each point,
// finding the distance.
$res = 0; $sum = 0;
for ($i = 0; $i < $n; $i++)
{
    $res += ($arr[$i] * $i - $sum);
    $sum += $arr[$i];
}

return $res;
}

function totaldistancesum($x, $y, $n)
{
    return distancesum($x, $n) +
           distancesum($y, $n);
}

// Driver Code
$x = array(-1, 1, 3, 2);
$y = array(5, 6, 5, 3);
$n = sizeof($x);
echo totaldistancesum($x, $y, $n), "\n";

// This code is contributed by m_kit
?>
```

Output :

22

Time Complexity : O(nlogn)

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/sum-manhattan-distances-pairs-points/>

Chapter 384

Sum of all elements between k1'th and k2'th smallest elements

Sum of all elements between k1'th and k2'th smallest elements - GeeksforGeeks

Given an array of integers and two numbers k1 and k2. Find sum of all elements between given two k1'th and k2'th smallest elements of array. It may be assumed that ($1 \leq k1 < k2 \leq n$) and all elements of array are distinct.

Examples :

```
Input : arr[] = {20, 8, 22, 4, 12, 10, 14}, k1 = 3, k2 = 6
Output : 26
         3rd smallest element is 10. 6th smallest element
         is 20. Sum of all element between k1 & k2 is
         12 + 14 = 26
```

```
Input : arr[] = {10, 2, 50, 12, 48, 13}, k1 = 2, k2 = 6
Output : 73
```

Method 1 (Sorting)

First sort the given array using a $O(n \log n)$ sorting algorithm like [Merge Sort](#), [Heap Sort](#), etc and return the sum of all element between index k1 and k2 in the sorted array.

Below is the implementation of the above idea :

C++

```
// C++ program to find sum of all element between
// to K1'th and k2'th smallest elements in array
#include <bits/stdc++.h>
```

```
using namespace std;

// Returns sum between two kth smallest element of array
int sumBetweenTwoKth(int arr[], int n, int k1, int k2)
{
    // Sort the given array
    sort(arr, arr + n);

    /* Below code is equivalent to
       int result = 0;
       for (int i=k1; i<k2-1; i++)
           result += arr[i];
       return accumulate(arr + k1, arr + k2 - 1, 0);
    }

// Driver program
int main()
{
    int arr[] = { 20, 8, 22, 4, 12, 10, 14 };
    int k1 = 3, k2 = 6;
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << sumBetweenTwoKth(arr, n, k1, k2);
    return 0;
}
```

Java

```
// Java program to find sum of all element
// between to K1'th and k2'th smallest
// elements in array
import java.util.Arrays;

class GFG {

    // Returns sum between two kth smallest
    // element of array
    static int sumBetweenTwoKth(int arr[],
                                int k1, int k2)
    {
        // Sort the given array
        Arrays.sort(arr);

        // Below code is equivalent to
        int result = 0;

        for (int i = k1; i < k2 - 1; i++)
            result += arr[i];
```

```
        return result;
    }

// Driver code
public static void main(String[] args)
{
    int arr[] = { 20, 8, 22, 4, 12, 10, 14 };
    int k1 = 3, k2 = 6;
    int n = arr.length;

    System.out.print(sumBetweenTwoKth(arr,
                                      k1, k2));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python program to find sum of
# all element between to K1'th and
# k2'th smallest elements in array

# Returns sum between two kth
# smallest element of array
def sumBetweenTwoKth(arr, n, k1, k2):

    # Sort the given array
    arr.sort()

    result = 0
    for i in range(k1, k2-1):
        result += arr[i]
    return result

# Driver code
arr = [ 20, 8, 22, 4, 12, 10, 14 ]
k1 = 3; k2 = 6
n = len(arr)
print(sumBetweenTwoKth(arr, n, k1, k2))
```

```
# This code is contributed by Anant Agarwal.
```

C#

```
// C# program to find sum of all element
```

```
// between to K1'th and k2'th smallest
// elements in array
using System;

class GFG {

    // Returns sum between two kth smallest
    // element of array
    static int sumBetweenTwoKth(int[] arr, int n,
                                int k1, int k2)
    {
        // Sort the given array
        Array.Sort(arr);

        // Below code is equivalent to
        int result = 0;

        for (int i = k1; i < k2 - 1; i++)
            result += arr[i];

        return result;
    }

    // Driver code
    public static void Main()
    {
        int[] arr = {20, 8, 22, 4, 12, 10, 14};
        int k1 = 3, k2 = 6;
        int n = arr.Length;

        Console.WriteLine(sumBetweenTwoKth(arr, n, k1, k2));
    }
}

// This code is contributed by nitin mittal.
```

Output:

26

Time Complexity: $O(n \log n)$

Method 2 (Using Min Heap)

We can optimize above solution be using a min heap.

- 1) Create a min heap of all array elements. (This step takes $O(n)$ time)
- 2) Do extract minimum k_1 times (This step takes $O(K_1 \log n)$ time)

3) Do extract minimum $k_2 - k_1 - 1$ times and sum all extracted elements. (This step takes $O((K_2 - k_1) * \log n)$ time)

Overall time complexity of this method is $O(n + k_2 \log n)$ which is better than sorting based method.

Improved By : [nitin mittal](#), [AbhishekVats3](#)

Source

<https://www.geeksforgeeks.org/sum-elements-k1th-k2th-smallest-elements/>

Chapter 385

Sum of minimum absolute difference of each array element

Sum of minimum absolute difference of each array element - GeeksforGeeks

Given an array of **n** distinct integers. The problem is to find the sum of minimum absolute difference of each array element. For an element **x** present at index **i** in the array its minimum absolute difference is calculated as:

Min absolute difference (x) = $\min(\text{abs}(x - \text{arr}[j]))$, where $1 \leq j \leq n$ and $j \neq i$ and **abs** is the absolute value.

Input Constraint: $2 \leq n$

Examples:

```
Input : arr = {4, 1, 5}
Output : 5
Sum of absolute differences is |4-5| + |1-4| + |5-4|
```

```
Input : arr = {5, 10, 1, 4, 8, 7}
Output : 9
```

```
Input : {12, 10, 15, 22, 21, 20, 1, 8, 9}
Output : 18
```

Naive approach: Using two loops. Pick an element of the array using outer loop and calculate its absolute difference with rest of the array elements using inner loop. Find the minimum absolute value and add it to the sum. Time Complexity $O(n^2)$.

Efficient Approach: The following steps are:

1. Sort the array of size **n**.

2. For the **1st** element of array its min absolute difference is calculated using the 2nd array element.
3. For the **last** array element its min absolute difference is calculated using the 2nd last array element.
4. For the rest of the array elements, $1 \leq i \leq n-2$, minimum absolute difference for an element at index **i** is calculated as: **minAbsDiff** = $\min(\text{abs}(\text{arr}[i] - \text{arr}[i-1]), \text{abs}(\text{arr}[i] - \text{arr}[i+1]))$.

C++

```
// C++ implementation to find the sum of minimum
// absolute difference of each array element
#include <bits/stdc++.h>
using namespace std;

// function to find the sum of
// minimum absolute difference
int sumOfMinAbsDifferences(int arr[], int n)
{
    // sort the given array
    sort(arr, arr+n);

    // initialize sum
    int sum = 0;

    // min absolute difference for
    // the 1st array element
    sum += abs(arr[0] - arr[1]);

    // min absolute difference for
    // the last array element
    sum += abs(arr[n-1] - arr[n-2]);

    // find min absolute difference for rest of the
    // array elements and add them to sum
    for (int i=1; i<n-1; i++)
        sum += min(abs(arr[i] - arr[i-1]), abs(arr[i] - arr[i+1]));

    // required sum
    return sum;
}

// Driver program to test above
int main()
{
    int arr[] = {5, 10, 1, 4, 8, 7};
```

```
int n = sizeof(arr) / sizeof(arr[0]);
cout << "Sum = "
     << sumOfMinAbsDifferences(arr, n);
}
```

Java

```
// java implementation to find the sum
// of minimum absolute difference of
// each array element
import java.*;
import java.util.Arrays;

public class GFG {

    // function to find the sum of
    // minimum absolute difference
    static int sumOfMinAbsDifferences(
        int arr[], int n)
    {

        // sort the given array
        Arrays.sort(arr);

        // initialize sum
        int sum = 0;

        // min absolute difference for
        // the 1st array element
        sum += Math.abs(arr[0] - arr[1]);

        // min absolute difference for
        // the last array element
        sum += Math.abs(arr[n-1] - arr[n-2]);

        // find min absolute difference for
        // rest of the array elements and
        // add them to sum
        for (int i = 1; i < n - 1; i++)
            sum +=
                Math.min(Math.abs(arr[i] - arr[i-1]),
                        Math.abs(arr[i] - arr[i+1]));

        // required sum
        return sum;
    }

    // Driver code
}
```

```
public static void main(String args[])
{
    int arr[] = {5, 10, 1, 4, 8, 7};
    int n = arr.length;

    System.out.println( "Sum = "
        + sumOfMinAbsDifferences(arr, n));
}
}

// This code is contributed by Sam007.
```

Python3

```
# Python implementation to find the
# sum of minimum absolute difference
# of each array element

# function to find the sum of
# minimum absolute difference

def sumOfMinAbsDifferences(arr,n):
    # sort the given array
    arr.sort()
    # initialize sum
    sum = 0

    # min absolute difference for
    # the 1st array element
    sum += abs(arr[0] - arr[1]);

    # min absolute difference for
    # the last array element
    sum += abs(arr[n - 1] - arr[n - 2]);

    # find min absolute difference for
    # rest of the array elements and
    # add them to sum
    for i in range(1, n - 1):
        sum += min(abs(arr[i] - arr[i - 1]),
                   abs(arr[i] - arr[i + 1]))

    # required sum
    return sum;

# Driver code
arr = [5, 10, 1, 4, 8, 7]
```

```
n = len(arr)
print( "Sum = ", sumOfMinAbsDifferences(arr, n))
```

```
#This code is contributed by Sam007
```

C#

```
// C# implementation to find the sum
// of minimum absolute difference of
// each array element
using System;

public class GFG {

    // function to find the sum of
    // minimum absolute difference
    static int sumOfMinAbsDifferences(
        int []arr ,int n)
    {

        // sort the given array
        Array.Sort(arr);

        // initialize sum
        int sum = 0;

        // min absolute difference for
        // the 1st array element
        sum += Math.Abs(arr[0] - arr[1]);

        // min absolute difference for
        // the last array element
        sum += Math.Abs(arr[n-1] - arr[n-2]);

        // find min absolute difference for
        // rest of the array elements and
        // add them to sum
        for (int i = 1; i < n - 1; i++)
            sum +=
                Math.Min(Math.Abs(arr[i] - arr[i-1]),
                        Math.Abs(arr[i] - arr[i+1]));

        // required sum
        return sum;
    }

    // Driver code
}
```

```
public static void Main ()
{
    int []arr = {5, 10, 1, 4, 8, 7};
    int n = arr.Length;

    Console.WriteLine( "Sum = "
        + sumOfMinAbsDifferences(arr, n));
}

// This code is contributed by Sam007.
```

PHP

```
<?php
// PHP implementation to find
// the sum of minimum absolute
// difference of each array element

// function to find the sum of
// minimum absolute difference
function sumOfMinAbsDifferences($arr, $n)
{

    // sort the given array
    sort($arr);
    sort($arr,$n);

    // initialize sum
    $sum = 0;

    // min absolute difference for
    // the 1st array element
    $sum += abs($arr[0] - $arr[1]);

    // min absolute difference for
    // the last array element
    $sum += abs($arr[$n - 1] - $arr[$n - 2]);

    // find min absolute difference
    // for rest of the array elements
    // and add them to sum
    for ($i = 1; $i < $n - 1; $i++)
        $sum += min(abs($arr[$i] - $arr[$i - 1]),
                   abs($arr[$i] - $arr[$i + 1]));

    // required sum
    return $sum;
}
```

```
}

// Driver Code
$arr = array(5, 10, 1, 4, 8, 7);
$n = sizeof($arr);
echo "Sum = ", sumOfMinAbsDifferences($arr, $n);

// This code is contributed by nitin mittal.
?>
```

Output:

Sum = 9

Time Complexity: O(nlogn)

Improved By : [Sam007](#), nitin mittal

Source

<https://www.geeksforgeeks.org/sum-minimum-absolute-difference-array-element/>

Chapter 386

Tag Sort (To get both sorted and original)

Tag Sort (To get both sorted and original) - GeeksforGeeks

This is not a new sorting algorithm, but an idea when we need to avoid swapping of large objects or need to access elements of a large array in both original and sorted orders.

A common sorting task is to sort elements of an array using a sorting algorithm like [Quick Sort](#), [Bubble Sort](#).. etc, but there may be times when we need to keep the actual array in tact and use a “tagged” array to store the correct positioning of the array when it is sorted. When we want to access elements in sorted way, we can use this “tagged” array.

Why to use Tag Sort?

When we are operating on large array of objects, it might be too costly to swap these large object. After all its about the disk swaps and we want to minimize it!

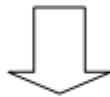
- Tag Sort allows sorting an integer array after tagging it with original object.
- In turn, we only swap the tag array integers instead of large array of objects.
- The actual elements are not being changed during the sort process. The positions in the tag array are being changed so they will hold the correct ordering of the elements when they are sorted.

In this example, the original elements in arr[] are not changed, but the original elements in tag[] are manipulated. The tag[] array will now hold the correct subscript ordering of the elements in arr[] so the array can be sorted into descending order when the tag[] array is called upon.

Array before Tag Sort

```
arr[6] = { 23.4, 64.4, 245.5, 52.51, 98.35, 345.51 };
```

```
tag[6] = { 0, 1, 2, 3, 4, 5 };
```



Array after the tag sort (for descending order)

```
arr[6] = { 23.4, 64.4, 245.5, 52.51, 98.35, 345.51 };
```

```
tag[6] = { 5, 3, 1, 4, 2, 0 };
```

Another Example, Suppose we have following Person object which inherently takes large chunk of memory(in GB or hundreds of MB).

```
class Person
{
    private int id;
    private float salary;
    private Object someBigObject = new Object();
    public Person(int id, float salary)
    {
    }
    public float getSalary()
    {
    }
    public String toString()
    {
    }
}
```

Hence, it might not be practical to move around these objects as disk seeks for swaps can eat up a lot of time. To avoid this, we start by creating a tag array.

- Every Person object is tagged to one element in the tag array and Instead of swapping the person object for sorting based on salary , we swap the tag[] integers.
- While printing the sorted array we take a cue from the tag array to print the sorted Persons array.
- Eventually, we'll escape swapping large Persons object.

Below is the implementation of above idea.

```
// Java Program to illustrate Tag Sort. This code
// uses Bubble Sort to modify tag array according
// to salaries. We can use other optimized sorting
// techniques also.
class Person
{
    private int id;
    private float salary;
    private Object someBigObject = new Object();

    public Person(int id, float salary)
    {
        this.id = id;
        this.salary = salary;
    }

    public float getSalary()
    {
        return salary;
    }

    @Override
    public String toString()
    {
        return "Person{" +
            "id=" + id +
            ", salary=" + salary +
            ", someBigObject=" + someBigObject +
            '}';
    }
}

public class Main
{
    public static void main(String[] args)
    {
        // Creating objects and their original
        // order (in tag array)
        int n = 5;
        Person persons[] = new Person[n];
        persons[0] = new Person(0, 233.5f);
        persons[1] = new Person(1, 23f);
        persons[2] = new Person(2, 13.98f);
        persons[3] = new Person(3, 143.2f);
        persons[4] = new Person(4, 3f);
        int tag[] = new int[n];
        for (int i = 0; i < n; i++)
            tag[i] = i;
```

```
// Every Person object is tagged to
// an element in the tag array.
System.out.println("Given Person and Tag ");
for (int i = 0; i < n; i++)
    System.out.println(persons[i] +
                       " : Tag: " + tag[i]);

// Modifying tag array so that we can access
// persons in sorted order.
tagSort(persons, tag);

System.out.println("New Tag Array after "+
                   "getting sorted as per Person[] ");
for (int i=0; i<n; i++)
    System.out.println(tag[i]);

// Accessing persons in sorted (by salary)
// way using modified tag array.
for (int i = 0; i < n; i++)
    System.out.println(persons[tag[i]]);
}

// Modifying tag array so that we can access
// persons in sorted order of salary.
public static void tagSort(Person persons[],
                           int tag[])
{
    int n = persons.length;
    for (int i=0; i<n; i++)
    {
        for (int j=i+1; j<n; j++)
        {
            if (persons[tag[i]].getSalary() >
                persons[tag[j]].getSalary())
            {
                // Note we are not sorting the
                // actual Persons array, but only
                // the tag array
                int temp = tag[i];
                tag[i] = tag[j];
                tag[j] = temp;
            }
        }
    }
}
```

Output:

```
Given Person and Tag
Person{id=0, salary=233.5, someBigObject=java.lang.Object@15db9742} : Tag: 0
Person{id=1, salary=23.0, someBigObject=java.lang.Object@6d06d69c} : Tag: 1
Person{id=2, salary=13.98, someBigObject=java.lang.Object@7852e922} : Tag: 2
Person{id=3, salary=143.2, someBigObject=java.lang.Object@4e25154f} : Tag: 3
Person{id=4, salary=3.0, someBigObject=java.lang.Object@70dea4e} : Tag: 4
New Tag Array after getting sorted as per Person[]
4
2
1
3
0
Person{id=4, salary=3.0, someBigObject=java.lang.Object@70dea4e}
Person{id=2, salary=13.98, someBigObject=java.lang.Object@7852e922}
Person{id=1, salary=23.0, someBigObject=java.lang.Object@6d06d69c}
Person{id=3, salary=143.2, someBigObject=java.lang.Object@4e25154f}
Person{id=0, salary=233.5, someBigObject=java.lang.Object@15db9742}
```

Source

<https://www.geeksforgeeks.org/tag-sort/>

Chapter 387

TimSort

TimSort - GeeksforGeeks

TimSort is a sorting algorithm based on [Insertion Sort](#) and [Merge Sort](#).

1. A stable sorting algorithm works in $O(n \log n)$ time
2. Used in Java's `Arrays.sort()` as well as Python's `sorted()` and `sort()`.
3. First sort small pieces using Insertion Sort, then merges the pieces using merge of merge sort.

We divides the Array into blocks known as **Run**. We sort those runs using insertion sort one by one and then merge those runs using combine function used in merge sort. If size of Array is less than run, then Array get sorted just by using Insertion Sort. The size of run may vary from 32 to 64 depending upon size of array. Note that merge function performs well when sizes subarrays are powers of 2. The idea is based on the fact that insertion sort performs well for small arrays.

Details of below implementation :

- We consider size of run as 32.
- We one by one sort pieces of size equal to run
- After sorting individual pieces, we merge them one by one. We double the size of merged subarrays after every iteration.

```
// C++ program to perform TimSort.  
#include<bits/stdc++.h>  
using namespace std;  
const int RUN = 32;  
  
// this function sorts array from left index to
```

```

// to right index which is of size atmost RUN
void insertionSort(int arr[], int left, int right)
{
    for (int i = left + 1; i <= right; i++)
    {
        int temp = arr[i];
        int j = i - 1;
        while (arr[j] > temp && j >= left)
        {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = temp;
    }
}

// merge function merges the sorted runs
void merge(int arr[], int l, int m, int r)
{
    // original array is broken in two parts
    // left and right array
    int len1 = m - l + 1, len2 = r - m;
    int left[len1], right[len2];
    for (int i = 0; i < len1; i++)
        left[i] = arr[l + i];
    for (int i = 0; i < len2; i++)
        right[i] = arr[m + 1 + i];

    int i = 0;
    int j = 0;
    int k = l;

    // after comparing, we merge those two array
    // in larger sub array
    while (i < len1 && j < len2)
    {
        if (left[i] <= right[j])
        {
            arr[k] = left[i];
            i++;
        }
        else
        {
            arr[k] = right[j];
            j++;
        }
        k++;
    }
}

```

```

// copy remaining elements of left, if any
while (i < len1)
{
    arr[k] = left[i];
    k++;
    i++;
}

// copy remaining element of right, if any
while (j < len2)
{
    arr[k] = right[j];
    k++;
    j++;
}
}

// iterative Timsort function to sort the
// array[0...n-1] (similar to merge sort)
void timSort(int arr[], int n)
{
    // Sort individual subarrays of size RUN
    for (int i = 0; i < n; i+=RUN)
        insertionSort(arr, i, min((i+31), (n-1)));

    // start merging from size RUN (or 32). It will merge
    // to form size 64, then 128, 256 and so on ....
    for (int size = RUN; size < n; size = 2*size)
    {
        // pick starting point of left sub array. We
        // are going to merge arr[left..left+size-1]
        // and arr[left+size, left+2*size-1]
        // After every merge, we increase left by 2*size
        for (int left = 0; left < n; left += 2*size)
        {
            // find ending point of left sub array
            // mid+1 is starting point of right sub array
            int mid = left + size - 1;
            int right = min((left + 2*size - 1), (n-1));

            // merge sub array arr[left.....mid] &
            // arr[mid+1.....right]
            merge(arr, left, mid, right);
        }
    }
}
}

```

```
// utility function to print the Array
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above function
int main()
{
    int arr[] = {5, 21, 7, 23, 19};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Given Array is\n");
    printArray(arr, n);

    timSort(arr, n);

    printf("After Sorting Array is\n");
    printArray(arr, n);
    return 0;
}
```

Output:

```
Given Array is
5 21 7 23 19
After Sorting Array is
5 7 19 21 23
```

References :

<https://svn.python.org/projects/python/trunk/Objects/listsort.txt>
https://en.wikipedia.org/wiki/Timsort#Minimum_size_.28minrun.29

Source

<https://www.geeksforgeeks.org/timsort/>

Chapter 388

Time Complexities of all Sorting Algorithms

Time Complexities of all Sorting Algorithms - GeeksforGeeks

Following is a quick revision sheet that you may refer at last minute

Algorithm

Time Complexity

Best

Average

Worst

Selection Sort

$\Omega(n^2)$

(n^2)

$O(n^2)$

Bubble Sort

$\Omega(n)$

(n^2)

$O(n^2)$

Insertion Sort

$\Omega(n)$

(n^2)

$O(n^2)$

[Heap Sort](#)

$\Omega(n \log(n))$

$(n \log(n))$

$O(n \log(n))$

[Quick Sort](#)

$\Omega(n \log(n))$

$(n \log(n))$

$O(n^2)$

[Merge Sort](#)

$\Omega(n \log(n))$

$(n \log(n))$

$O(n \log(n))$

[Bucket Sort](#)

$\Omega(n+k)$

$(n+k)$

$O(n^2)$

[Radix Sort](#)

$\Omega(nk)$

(nk)

$O(nk)$

Also see:

- [Searching and Sorting articles](#)
- [Previous year GATE Questions on Sorting](#)

Source

<https://www.geeksforgeeks.org/time-complexities-of-all-sorting-algorithms/>

Chapter 389

Time complexity of insertion sort when there are O(n) inversions?

Time complexity of insertion sort when there are O(n) inversions? - GeeksforGeeks

What is an inversion?

Given an array arr[], a pair arr[i] and arr[j] forms an inversion if arr[i] < arr[j] and i > j. For example, the array {1, 3, 2, 5} has one inversion (3, 2) and array {5, 4, 3} has inversions (5, 4), (5, 3) and (4, 3). We have discussed a [merge sort based algorithm to count inversions](#)

What is the time complexity of Insertion Sort when there are O(n) inversions?

Consider the following function of insertion sort.

```
/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i-1;

        /* Move elements of arr[0..i-1], that are
           greater than key, to one position ahead
           of their current position */
        while (j >= 0 && arr[j] > key)
        {
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j+1] = key;
```

```
    }  
}
```

If we take a closer look at the insertion sort code, we can notice that every iteration of while loop reduces one inversion. The while loop executes only if $i > j$ and $\text{arr}[i] < \text{arr}[j]$. Therefore total number of while loop iterations (For all values of i) is same as number of inversions. Therefore overall time complexity of the insertion sort is $O(n + f(n))$ where $f(n)$ is inversion count. If the inversion count is $O(n)$, then the time complexity of insertion sort is $O(n)$. In worst case, there can be $n*(n-1)/2$ inversions. The worst case occurs when the array is sorted in reverse order. So the worst case time complexity of insertion sort is $O(n^2)$.

Source

<https://www.geeksforgeeks.org/time-complexity-insertion-sort-inversions/>

Chapter 390

Tree Sort

Tree Sort - GeeksforGeeks

Tree sort is a sorting algorithm that is based on [Binary Search Tree](#) data structure. It first creates a binary search tree from the elements of the input list or array and then performs an in-order traversal on the created binary search tree to get the elements in sorted order.

Algorithm:

- Step 1: Take the elements input in an array.
- Step 2: Create a Binary search tree by inserting data items from the array into the binary search tree.
- Step 3: Perform in-order traversal on the tree to get the elements in sorted order.

C++

```
// C++ program to implement Tree Sort
#include<bits/stdc++.h>

using namespace std;

struct Node
{
    int key;
    struct Node *left, *right;
};

// A utility function to create a new BST Node
struct Node *newNode(int item)
{
    struct Node *temp = new Node;
    temp->key = item;
```

```

        temp->left = temp->right = NULL;
        return temp;
    }

// Stores inoder traversal of the BST
// in arr[]
void storeSorted(Node *root, int arr[], int &i)
{
    if (root != NULL)
    {
        storeSorted(root->left, arr, i);
        arr[i++] = root->key;
        storeSorted(root->right, arr, i);
    }
}

/* A utility function to insert a new
   Node with given key in BST */
Node* insert(Node* node, int key)
{
    /* If the tree is empty, return a new Node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) Node pointer */
    return node;
}

// This function sorts arr[0..n-1] using Tree Sort
void treeSort(int arr[], int n)
{
    struct Node *root = NULL;

    // Construct the BST
    root = insert(root, arr[0]);
    for (int i=1; i<n; i++)
        insert(root, arr[i]);

    // Store inoder traversal of the BST
    // in arr[]
    int i = 0;
    storeSorted(root, arr, i);
}

```

```
// Driver Program to test above functions
int main()
{
    //create input array
    int arr[] = {5, 4, 7, 2, 11};
    int n = sizeof(arr)/sizeof(arr[0]);

    treeSort(arr, n);

    for (int i=0; i<n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Java

```
// Java program to
// implement Tree Sort
class GFG
{

    // Class containing left and
    // right child of current
    // node and key value
    class Node
    {
        int key;
        Node left, right;

        public Node(int item)
        {
            key = item;
            left = right = null;
        }
    }

    // Root of BST
    Node root;

    // Constructor
    GFG()
    {
        root = null;
    }

    // This method mainly
```

```
// calls insertRec()
void insert(int key)
{
    root = insertRec(root, key);
}

/* A recursive function to
insert a new key in BST */
Node insertRec(Node root, int key)
{

    /* If the tree is empty,
    return a new node */
    if (root == null)
    {
        root = new Node(key);
        return root;
    }

    /* Otherwise, recur
    down the tree */
    if (key < root.key)
        root.left = insertRec(root.left, key);
    else if (key > root.key)
        root.right = insertRec(root.right, key);

    /* return the root */
    return root;
}

// A function to do
// inorder traversal of BST
void inorderRec(Node root)
{
    if (root != null)
    {
        inorderRec(root.left);
        System.out.print(root.key + " ");
        inorderRec(root.right);
    }
}
void treeins(int arr[])
{
    for(int i = 0; i < arr.length; i++)
    {
        insert(arr[i]);
    }
}
```

```
}

// Driver Code
public static void main(String[] args)
{
    GFG tree = new GFG();
    int arr[] = {5, 4, 7, 2, 11};
    tree.treeins(arr);
    tree.inorderRec(tree.root);
}
}

// This code is contributed
// by Vibin M
```

Output:

2 4 5 7 11

Average Case Time Complexity : $O(n \log n)$ Adding one item to a Binary Search tree on average takes $O(\log n)$ time. Therefore, adding n items will take $O(n \log n)$ time

Worst Case Time Complexity : $O(n^2)$. The worst case time complexity of Tree Sort can be improved by using a self-balancing binary search tree like Red Black Tree, AVL Tree. Using self-balancing binary tree Tree Sort will take $O(n \log n)$ time to sort the array in worst case.

Auxiliary Space : $O(n)$

References:

https://en.wikipedia.org/wiki/Tree_sort

Improved By : [vibi](#), [vsushko](#)

Source

<https://www.geeksforgeeks.org/tree-sort/>

Chapter 391

Triplets in array with absolute difference less than k

Triplets in array with absolute difference less than k - GeeksforGeeks

Given an array $A[]$ of n elements and an integer k . The task is to find the number of triplet (x, y, z) , where $0 \leq x, y, z < n$ and x, y, z are the index in the array $A[]$ such that:

$A[x] - A[y] \leq k$
 $A[y] - A[z] \leq k$
 $A[z] - A[x] \leq k$

Examples:

Input : $A[] = \{ 1, 1, 2, 2, 3 \}$, $k = 1$
Output : 5
(0, 1, 2), (0, 1, 3), (0, 2, 3), (1, 2, 3),
(2, 3, 4) are the triplet whose element will
satisfy the above three condition.

Input : $A[] = \{ 1, 2, 3 \}$, $k = 1$
Output : 1

A **simple solution** is to run three nested loops and count triplets with given constraints.

An **efficient solution** is based on the fact rearranging the elements in the array does not affect the answer because basically, we want index x, y, z to be in increasing order, not $A[x], A[y], A[z]$ to be sorted. Suppose, $A[x]$ is at index y , $A[y]$ at z , $A[z]$ at x , still we can pick the triplet (x, y, z) because the condition of difference between any two elements to be less k still stands.

Now, to calculate the number of triplet indices (x, y, z) which satisfies the above condition, we will sort the given array. Now, for each element $A[i]$, $i \geq 2$, we will find the lower bound index of $A[i] - k$, say lb . Now, observe all the element between index lb and i are less than

$A[i]$ and the difference between any two elements will be less than or equal to k . So, element at index i can be treated as index z and we can choose any two element from lb to $i - 1$. So, this will increase the count of the triplet by $^{i-lb}C^2$.

Below is the implementation of this approach:

C++

```
// CPP program to count triplets with difference less
// than k.
#include <bits/stdc++.h>
using namespace std;

// Return the lower bound i.e smallest index of
// element having value greater or equal to value
int binary_lower(int value, int arr[], int n)
{
    int start = 0;
    int end = n - 1;
    int ans = -1;
    int mid;

    while (start <= end) {
        mid = (start + end) / 2;
        if (arr[mid] >= value) {
            end = mid - 1;
            ans = mid;
        }
        else {
            start = mid + 1;
        }
    }
    return ans;
}

// Return the number of triplet indices satisfies
// the three constraints
int countTriplet(int arr[], int n, int k)
{
    int count = 0;

    // sort the array
    sort(arr, arr + n);

    // for each element from index 2 to n - 1.
    for (int i = 2; i < n; i++) {

        // finding the lower bound of arr[i] - k.
        int cur = binary_lower(arr[i] - k, arr, n);
```

```
// If there are at least two elements between
// lower bound and current element.
if (cur <= i - 2) {

    // increment the count by lb - i C 2.
    count += ((i - cur) * (i - cur - 1)) / 2;
}
}

return count;
}
int main()
{
    int arr[] = { 1, 1, 2, 2, 3 };
    int k = 1;
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << countTriplet(arr, n, k) << endl;
    return 0;
}
```

Java

```
// Java program to count triplets
// with difference less than k.
import java.io.*;
import java.util.*;

class GFG
{

    // Return the lower bound i.e
    // smallest index of element
    // having value greater or
    // equal to value
    static int binary_lower(int value,
                           int arr[],
                           int n)
    {
        int start = 0;
        int end = n - 1;
        int ans = -1;
        int mid;

        while (start <= end)
        {
            mid = (start + end) / 2;
            if (arr[mid] >= value)
```

```
{  
    end = mid - 1;  
    ans = mid;  
}  
else  
{  
    start = mid + 1;  
}  
}  
return ans;  
}  
  
// Return the number of  
// triplet indices satisfies  
// the three constraints  
static int countTriplet(int arr[],  
                        int n, int k)  
{  
    int count = 0;  
  
    // sort the array  
    Arrays.sort(arr);  
  
    // for each element from  
    // index 2 to n - 1.  
    for (int i = 2; i < n; i++)  
    {  
  
        // finding the lower  
        // bound of arr[i] - k.  
        int cur = binary_lower(arr[i] - k,  
                               arr, n);  
  
        // If there are at least two  
        // elements between lower  
        // bound and current element.  
        if (cur <= i - 2)  
        {  
  
            // increment the count  
            // by lb - i C 2.  
            count += ((i - cur) *  
                      (i - cur - 1)) / 2;  
        }  
    }  
    return count;  
}
```

```
// Driver Code
public static void main (String[] args)
{
    int arr[] = {1, 1, 2, 2, 3};
    int k = 1;
    int n = arr.length;

    System.out.println(countTriplet(arr, n, k));
}
}

// This code is contributed by anuj_67.
```

C#

```
// C# program to count triplets
// with difference less than k.
using System;

class GFG
{

    // Return the lower bound i.e
    // smallest index of element
    // having value greater or
    // equal to value
    static int binary_lower(int value,
                           int []arr,
                           int n)
    {
        int start = 0;
        int end = n - 1;
        int ans = -1;
        int mid;

        while (start <= end)
        {
            mid = (start + end) / 2;
            if (arr[mid] >= value)
            {
                end = mid - 1;
                ans = mid;
            }
            else
            {
                start = mid + 1;
            }
        }
    }
}
```

```
        return ans;
    }

// Return the number of
// triplet indices satisfies
// the three constraints
static int countTriplet(int []arr,
                        int n, int k)
{
    int count = 0;

    // sort the array
    Array.Sort(arr);

    // for each element from
    // index 2 to n - 1.
    for (int i = 2; i < n; i++)
    {

        // finding the lower
        // bound of arr[i] - k.
        int cur = binary_lower(arr[i] - k,
                               arr, n);

        // If there are at least two
        // elements between lower
        // bound and current element.
        if (cur <= i - 2)
        {

            // increment the count
            // by lb - i C 2.
            count += ((i - cur) *
                      (i - cur - 1)) / 2;
        }
    }
    return count;
}

// Driver Code
public static void Main ()
{
    int []arr = {1, 1, 2, 2, 3};
    int k = 1;
    int n = arr.Length;

    Console.WriteLine(countTriplet(arr, n, k));
}
```

```
}
```

```
// This code is contributed by anuj_67.
```

Output:

5

Complexity: O(nlogn)

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/triplets-array-absolute-difference-less-k/>

Chapter 392

Union and Intersection of two Linked Lists

Union and Intersection of two Linked Lists - GeeksforGeeks

Given two Linked Lists, create union and intersection lists that contain union and intersection of the elements present in the given lists. Order of elements in output lists doesn't matter.

Example:

Input:

List1: 10->15->4->20
List2: 8->4->2->10

Output:

Intersection List: 4->10
Union List: 2->8->20->4->15->10

Method 1 (Simple)

Following are simple algorithms to get union and intersection lists respectively.

Intersection (list1, list2)

Initialize result list as NULL. Traverse list1 and look for its each element in list2, if the element is present in list2, then add the element to result.

Union (list1, list2):

Initialize result list as NULL. Traverse list1 and add all of its elements to the result. Traverse list2. If an element of list2 is already present in result then do not insert it to result, otherwise insert.

This method assumes that there are no duplicates in the given lists.

Thanks to Shekhu for suggesting this method. Following are C and Java implementations of this method.

C/C++

```
// C/C++ program to find union and intersection of two unsorted
// linked lists
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

/* A utility function to insert a node at the beginning of
   a linked list*/
void push(struct Node** head_ref, int new_data);

/* A utility function to check if given data is present in a list */
bool isPresent(struct Node *head, int data);

/* Function to get union of two linked lists head1 and head2 */
struct Node *getUnion(struct Node *head1, struct Node *head2)
{
    struct Node *result = NULL;
    struct Node *t1 = head1, *t2 = head2;

    // Insert all elements of list1 to the result list
    while (t1 != NULL)
    {
        push(&result, t1->data);
        t1 = t1->next;
    }

    // Insert those elements of list2 which are not
    // present in result list
    while (t2 != NULL)
    {
        if (!isPresent(result, t2->data))
            push(&result, t2->data);
        t2 = t2->next;
    }

    return result;
}

/* Function to get intersection of two linked lists
   head1 and head2 */
struct Node *getIntersection(struct Node *head1,
                           struct Node *head2)
```

```
{  
    struct Node *result = NULL;  
    struct Node *t1 = head1;  
  
    // Traverse list1 and search each element of it in  
    // list2. If the element is present in list 2, then  
    // insert the element to result  
    while (t1 != NULL)  
    {  
        if (isPresent(head2, t1->data))  
            push (&result, t1->data);  
        t1 = t1->next;  
    }  
  
    return result;  
}  
  
/* A utility function to insert a node at the begining of a linked list*/  
void push (struct Node** head_ref, int new_data)  
{  
    /* allocate node */  
    struct Node* new_node =  
        (struct Node*) malloc(sizeof(struct Node));  
  
    /* put in the data */  
    new_node->data = new_data;  
  
    /* link the old list off the new node */  
    new_node->next = (*head_ref);  
  
    /* move the head to point to the new node */  
    (*head_ref) = new_node;  
}  
  
/* A utility function to print a linked list*/  
void printList (struct Node *node)  
{  
    while (node != NULL)  
    {  
        printf ("%d ", node->data);  
        node = node->next;  
    }  
}  
  
/* A utility function that returns true if data is  
   present in linked list else return false */  
bool isPresent (struct Node *head, int data)  
{
```

```
struct Node *t = head;
while (t != NULL)
{
    if (t->data == data)
        return 1;
    t = t->next;
}
return 0;
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
    struct Node* intersecn = NULL;
    struct Node* unin = NULL;

    /*create a linked lits 10->15->5->20 */
    push (&head1, 20);
    push (&head1, 4);
    push (&head1, 15);
    push (&head1, 10);

    /*create a linked lits 8->4->2->10 */
    push (&head2, 10);
    push (&head2, 2);
    push (&head2, 4);
    push (&head2, 8);

    intersecn = getIntersection (head1, head2);
    unin = getUnion (head1, head2);

    printf ("\n First list is \n");
    printList (head1);

    printf ("\n Second list is \n");
    printList (head2);

    printf ("\n Intersection list is \n");
    printList (intersecn);

    printf ("\n Union list is \n");
    printList (unin);

    return 0;
}
```

Java

```
// Java program to find union and intersection of two unsorted
// linked lists
class LinkedList
{
    Node head; // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    /* Function to get Union of 2 Linked Lists */
    void getUnion(Node head1, Node head2)
    {
        Node t1 = head1, t2 = head2;

        //insert all elements of list1 in the result
        while (t1 != null)
        {
            push(t1.data);
            t1 = t1.next;
        }

        // insert those elements of list2 that are not present
        while (t2 != null)
        {
            if (!isPresent(head, t2.data))
                push(t2.data);
            t2 = t2.next;
        }
    }

    void getIntersection(Node head1, Node head2)
    {
        Node result = null;
        Node t1 = head1;

        // Traverse list1 and search each element of it in list2.
        // If the element is present in list 2, then insert the
```

```
// element to result
while (t1 != null)
{
    if (isPresent(head2, t1.data))
        push(t1.data);
    t1 = t1.next;
}
}

/* Utility function to print list */
void printList()
{
    Node temp = head;
    while(temp != null)
    {
        System.out.print(temp.data+" ");
        temp = temp.next;
    }
    System.out.println();
}

/* Inserts a node at start of linked list */
void push(int new_data)
{
    /* 1 & 2: Allocate the Node &
       Put in the data*/
    Node new_node = new Node(new_data);

    /* 3. Make next of new Node as head */
    new_node.next = head;

    /* 4. Move the head to point to new Node */
    head = new_node;
}

/* A utility function that returns true if data is present
   in linked list else return false */
boolean isPresent (Node head, int data)
{
    Node t = head;
    while (t != null)
    {
        if (t.data == data)
            return true;
        t = t.next;
    }
}
```

```
        return false;
    }

/* Drier program to test above functions */
public static void main(String args[])
{
    LinkedList llist1 = new LinkedList();
    LinkedList llist2 = new LinkedList();
    LinkedList unin = new LinkedList();
    LinkedList intersecn = new LinkedList();

    /*create a linked lists 10->15->5->20 */
    llist1.push(20);
    llist1.push(4);
    llist1.push(15);
    llist1.push(10);

    /*create a linked lists 8->4->2->10 */
    llist2.push(10);
    llist2.push(2);
    llist2.push(4);
    llist2.push(8);

    intersecn.getIntersection(llist1.head, llist2.head);
    unin.getUnion(llist1.head, llist2.head);

    System.out.println("First List is");
    llist1.printList();

    System.out.println("Second List is");
    llist2.printList();

    System.out.println("Intersection List is");
    intersecn.printList();

    System.out.println("Union List is");
    unin.printList();
}

} /* This code is contributed by Rajat Mishra */
```

Output:

```
First list is
10 15 4 20
Second list is
8 4 2 10
```

```
Intersection list is
4 10
Union list is
2 8 20 4 15 10
```

Time Complexity: $O(mn)$ for both union and intersection operations. Here m is the number of elements in first list and n is the number of elements in second list.

Method 2 (Use Merge Sort)

In this method, algorithms for Union and Intersection are very similar. First we sort the given lists, then we traverse the sorted lists to get union and intersection.

Following are the steps to be followed to get union and intersection lists.

- 1) Sort the first Linked List using merge sort. This step takes $O(m\log m)$ time. Refer [this post](#) for details of this step.
- 2) Sort the second Linked List using merge sort. This step takes $O(n\log n)$ time. Refer [this post](#) for details of this step.
- 3) Linearly scan both sorted lists to get the union and intersection. This step takes $O(m + n)$ time. This step can be implemented using the same algorithm as sorted arrays algorithm discussed [here](#).

Time complexity of this method is $O(m\log m + n\log n)$ which is better than method 1's time complexity.

Method 3 (Use Hashing)

Union (list1, list2)

Initialize the result list as NULL and create an empty hash table. Traverse both lists one by one, for each element being visited, look the element in hash table. If the element is not present, then insert the element to result list. If the element is present, then ignore it.

Intersection (list1, list2)

Initialize the result list as NULL and create an empty hash table. Traverse list1. For each element being visited in list1, insert the element in hash table. Traverse list2, for each element being visited in list2, look the element in hash table. If the element is present, then insert the element to result list. If the element is not present, then ignore it.

Both of the above methods assume that there are no duplicates.

```
// Java code for Union and Intersection of two
// Linked Lists
import java.util.HashMap;
import java.util.HashSet;

class LinkedList {
    Node head; // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
    }
```

```
Node(int d)
{
    data = d;
    next = null;
}
}

/* Utility function to print list */
void printList()
{
    Node temp = head;
    while(temp != null)
    {
        System.out.print(temp.data+" ");
        temp = temp.next;
    }
    System.out.println();
}

/* Inserts a node at start of linked list */
void push(int new_data)
{
    /* 1 & 2: Allocate the Node &
       Put in the data*/
    Node new_node = new Node(new_data);

    /* 3. Make next of new Node as head */
    new_node.next = head;

    /* 4. Move the head to point to new Node */
    head = new_node;
}

public void append(int new_data)
{
    if(this.head == null)
    {
        Node n = new Node(new_data);
        this.head = n;
        return;
    }
    Node n1 = this.head;
    Node n2 = new Node(new_data);
    while(n1.next != null)
    {
        n1 = n1.next;
    }
}
```

```
n1.next = n2;
n2.next = null;
}

/* A utility function that returns true if data is
present in linked list else return false */
boolean isPresent (Node head, int data)
{
    Node t = head;
    while (t != null)
    {
        if (t.data == data)
            return true;
        t = t.next;
    }
    return false;
}

LinkedList getIntersection(Node head1, Node head2)
{
    HashSet<Integer> hset = new HashSet<>();
    Node n1 = head1;
    Node n2 = head2;
    LinkedList result = new LinkedList();

    // loop stores all the elements of list1 in hset
    while(n1 != null)
    {
        if(hset.contains(n1.data))
        {
            hset.add(n1.data);
        }
        else
        {
            hset.add(n1.data);
        }
        n1 = n1.next;
    }

    //For every element of list2 present in hset
    //loop inserts the element into the result
    while(n2 != null)
    {
        if(hset.contains(n2.data))
        {
            result.push(n2.data);
        }
        n2 = n2.next;
    }
}
```

```
        }
        return result;
    }

LinkedList getUnion(Node head1, Node head2)
{
    // HashMap that will store the
    // elements of the lists with their counts
    HashMap<Integer, Integer> hmap = new HashMap<>();
    Node n1 = head1;
    Node n2 = head2;
    LinkedList result = new LinkedList();

    // loop inserts the elements and the count of
    // that element of list1 into the hmap
    while(n1 != null)
    {
        if(hmap.containsKey(n1.data))
        {
            int val = hmap.get(n1.data);
            hmap.put(n1.data, val + 1);
        }
        else
        {
            hmap.put(n1.data, 1);
        }
        n1 = n1.next;
    }

    // loop further adds the elements of list2 with
    // their counts into the hmap
    while(n2 != null)
    {
        if(hmap.containsKey(n2.data))
        {
            int val = hmap.get(n2.data);
            hmap.put(n2.data, val + 1);
        }
        else
        {
            hmap.put(n2.data, 1);
        }
        n2 = n2.next;
    }

    // Eventually add all the elements
    // into the result that are present in the hmap
    for(int a:hmap.keySet())
```

```
{  
    result.append(a);  
}  
return result;  
}  
  
/* Driver program to test above functions */  
public static void main(String args[])  
{  
    LinkedList llist1 = new LinkedList();  
    LinkedList llist2 = new LinkedList();  
    LinkedList union = new LinkedList();  
    LinkedList intersection = new LinkedList();  
  
    /*create a linked list 10->15->4->20 */  
    llist1.push(20);  
    llist1.push(4);  
    llist1.push(15);  
    llist1.push(10);  
  
    /*create a linked list 8->4->2->10 */  
    llist2.push(10);  
    llist2.push(2);  
    llist2.push(4);  
    llist2.push(8);  
  
    intersection = intersection.getIntersection(llist1.head,  
                                              llist2.head);  
    union=union.getUnion(llist1.head, llist2.head);  
  
    System.out.println("First List is");  
    llist1.printList();  
  
    System.out.println("Second List is");  
    llist2.printList();  
  
    System.out.println("Intersection List is");  
    intersection.printList();  
  
    System.out.println("Union List is");  
    union.printList();  
}  
}  
// This code is contributed by Kamal Rawal
```

Output:

```
First List is  
10 15 4 20  
Second List is  
8 4 2 10  
Intersection List is  
10 4  
Union List is  
2 4 20 8 10 15
```

Time complexity of this method depends on the hashing technique used and the distribution of elements in input lists. In practical, this approach may turn out to be better than above 2 methods.

Source

<https://www.geeksforgeeks.org/union-and-intersection-of-two-linked-lists/>

Chapter 393

Union and Intersection of two linked lists Set-2 (Using Merge Sort)

Union and Intersection of two linked lists Set-2 (Using Merge Sort) - GeeksforGeeks

Given two Linked Lists, create union and intersection lists that contain union and intersection of the elements present in the given lists. Order of elements in output lists doesn't matter.

Examples:

Input:

```
List1: 10 -> 15 -> 4 -> 20  
List2: 8 -> 4 -> 2 -> 10
```

Output:

```
Intersection List: 4 -> 10  
Union List: 2 -> 8 -> 20 -> 4 -> 15 -> 10
```

There were three methods discussed in this [post](#) with an implementation of Method 1. In this post, we will see an implementation of Method 2 i.e. Using [Merge sort](#).

Implementation:

Following are the steps to be followed to get union and intersection lists.

- 1) Sort both Linked Lists using merge sort.
This step takes $O(m \log m)$ time.
- 2) Linearly scan both sorted lists to get the union and intersection.
This step takes $O(m + n)$ time.

Just like Method 1, This method also assumes that there are distinct elements in the lists.

```
// C++ program to find union and intersection of
// two unsorted linked lists in O(n Log n) time.
#include<iostream>
using namespace std;

/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

/* A utility function to insert a node at the begining
   of a linked list*/
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* UTILITY FUNCTIONS */
/* Split the nodes of the given list into front and back halves,
   and return the two lists using the reference parameters.
   If the length is odd, the extra node should go in the front list.
   Uses the fast/slow pointer strategy. */
void FrontBackSplit(struct Node* source, struct Node** frontRef,
                    struct Node** backRef)
{
    struct Node* fast;
    struct Node* slow;
    if (source==NULL || source->next==NULL)
    {
        /* length < 2 cases */
        *frontRef = source;
        *backRef = NULL;
    }
}
```

```
else
{
    slow = source;
    fast = source->next;

    /* Advance 'fast' two nodes, and advance 'slow' one node */
    while (fast != NULL)
    {
        fast = fast->next;
        if (fast != NULL)
        {
            slow = slow->next;
            fast = fast->next;
        }
    }

    /* 'slow' is before the midpoint in the list,
       so split it in two at that point. */
    *frontRef = source;
    *backRef = slow->next;
    slow->next = NULL;
}

/*
 * See https://www.geeksforgeeks.org/?p=3622 for details
 * of this function */
struct Node* SortedMerge(struct Node* a, struct Node* b)
{
    struct Node* result = NULL;

    /* Base cases */
    if (a == NULL)
        return(b);
    else if (b==NULL)
        return(a);

    /* Pick either a or b, and recur */
    if (a->data <= b->data)
    {
        result = a;
        result->next = SortedMerge(a->next, b);
    }
    else
    {
        result = b;
        result->next = SortedMerge(a, b->next);
    }
    return(result);
}
```

```
}

/* sorts the linked list by changing next pointers
 * (not data) */
void mergeSort(struct Node** headRef)
{
    struct Node *head = *headRef;
    struct Node *a, *b;

    /* Base case -- length 0 or 1 */
    if ((head == NULL) || (head->next == NULL))
        return;

    /* Split head into 'a' and 'b' sublists */
    FrontBackSplit(head, &a, &b);

    /* Recursively sort the sublists */
    mergeSort(&a);
    mergeSort(&b);

    /* answer = merge the two sorted lists together */
    *headRef = SortedMerge(a, b);
}

/* Function to get union of two linked lists head1 and head2 */
struct Node *getUnion(struct Node *head1, struct Node *head2)
{
    struct Node *result = NULL;
    struct Node *t1 = head1, *t2 = head2;

    // Traverse both lists and store the element in
    // the resultant list
    while (t1 != NULL && t2 != NULL)
    {
        // Move to the next of first list
        // if its element is smaller
        if (t1->data < t2->data)
        {
            push(&result, t1->data);
            t1 = t1->next;
        }

        // Else move to the next of second list
        else if (t1->data > t2->data)
        {
            push(&result, t2->data);
            t2 = t2->next;
        }
    }
}
```

```
}

// If same then move to the next node
// in both lists
else
{
    push(&result, t2->data);
    t1 = t1->next;
    t2 = t2->next;
}
}

/* Print remaining elements of the lists */
while (t1 != NULL)
{
    push(&result, t1->data);
    t1 = t1->next;
}
while (t2 != NULL)
{
    push(&result, t2->data);
    t2 = t2->next;
}

return result;
}

/* Function to get intersection of two linked lists
head1 and head2 */
struct Node *getIntersection(struct Node *head1,
                             struct Node *head2)
{
    struct Node *result = NULL;
    struct Node *t1 = head1, *t2 = head2;

    // Traverse both lists and store the same element
    // in the resultant list
    while (t1 != NULL && t2 != NULL)
    {
        // Move to the next of first list if smaller
        if (t1->data < t2->data)
            t1 = t1->next;

        // Move to the next of second list if it is smaller
        else if (t1->data > t2->data)
            t2 = t2->next;

        // If both are same
    }
}
```

```
    else
    {
        // Store current element in the list
        push(&result, t2->data);

        // Move to the next node of both lists
        t1 = t1->next;
        t2 = t2->next;
    }
}

//return the resultant list
return result;
}

/* A utility function to print a linked list*/
void printList (struct Node *node)
{
    while (node != NULL)
    {
        printf ("%d ", node->data);
        node = node->next;
    }
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
    struct Node* intersection_list = NULL;
    struct Node* union_list = NULL;

    /*create a linked lits 11->10->15->4->20 */
    push(&head1, 20);
    push(&head1, 4);
    push(&head1, 15);
    push(&head1, 10);
    push(&head1, 11);

    /*create a linked lits 8->4->2->10 */
    push(&head2, 10);
    push(&head2, 2);
    push(&head2, 4);
    push(&head2, 8);
```

```
/* Sort the above created Linked List */
mergeSort(&head1);
mergeSort(&head2);

intersection_list = getIntersection(head1, head2);
union_list = getUnion(head1, head2);

printf("First list is \n");
printList(head1);

printf("\nSecond list is \n");
printList(head2);

printf("\nIntersection list is \n");
printList(intersection_list);

printf("\nUnion list is \n");
printList(union_list);

return 0;
}
```

Output:

```
First list is
4 10 11 15 20
Second list is
2 4 8 10
Intersection list is
10 4
Union list is
20 15 11 10 8 4 2
```

Time complexity of this method is $O(m \log m + n \log n)$.

In the next post, Method-3 will be discussed i.e. using hashing.

Source

<https://www.geeksforgeeks.org/union-intersection-two-linked-lists-set-2-using-merge-sort/>

Chapter 394

Union and Intersection of two linked lists Set-3 (Hashing)

Union and Intersection of two linked lists Set-3 (Hashing) - GeeksforGeeks

Given two Linked Lists, create union and intersection lists that contain union and intersection of the elements present in the given lists. Order of elements in output lists doesn't matter.

Examples:

Input:

List1: 10 → 15 → 4 → 20
List2: 8 → 4 → 2 → 10

Output:

Intersection List: 4 → 10
Union List: 2 → 8 → 20 → 4 → 15 → 10

We have already discussed [Method-1](#) and [Method-2](#) of this question.

In this post, its Method-3 (Using Hashing) is discussed with a Time Complexity of $O(m+n)$ i.e. better than both methods discussed earlier.

Implementation:

- 1- Start traversing both the lists.
 - a) Store the current element of both lists with its occurrence in the map.
- 2- For Union: Store all the elements of the map in the resultant list.
- 3- For Intersection: Store all the elements only with an occurrence of 2 as 2 denotes that they are present in both the lists.

Below is C++ implementation of above steps.

```
// C++ program to find union and intersection of
// two unsorted linked lists in O(m+n) time.
#include<bits/stdc++.h>
using namespace std;

/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

/* A utility function to insert a node at the
   begining of a linked list*/
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Utility function to store the elements of both list */
void storeEle(struct Node* head1, struct Node *head2,
              unordered_map<int, int> &eleOcc)
{
    struct Node* ptr1 = head1;
    struct Node* ptr2 = head2;

    // Traverse both lists
    while (ptr1 != NULL || ptr2 != NULL)
    {
        // store element in the map
        if (ptr1!=NULL)
        {
            eleOcc[ptr1->data]++;
            ptr1=ptr1->next;
        }
    }
}
```

```
// store element in the map
if (ptr2 != NULL)
{
    eleOcc[ptr2->data]++;
    ptr2=ptr2->next;
}
}

/* Function to get union of two linked lists head1
   and head2 */
struct Node *getUnion(unordered_map<int, int> eleOcc)
{
    struct Node *result = NULL;

    // Push all the elements into the resultant list
    for (auto it=eleOcc.begin(); it!=eleOcc.end(); it++)
        push(&result, it->first);

    return result;
}

/* Function to get intersection of two linked lists
   head1 and head2 */
struct Node *getIntersection(unordered_map<int, int> eleOcc)
{
    struct Node *result = NULL;

    // Push a node with an element having occurrence
    // of 2 as that means the current element is present
    // in both the lists
    for (auto it=eleOcc.begin(); it!=eleOcc.end(); it++)
        if (it->second == 2)
            push(&result, it->first);

    // return resultant list
    return result;
}

/* A utility function to print a linked list*/
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf ("%d ", node->data);
        node = node->next;
    }
}
```

```
}

// Prints union and intersection of lists with head1
// and head2.
void printUnionIntersection(Node *head1, Node *head2)
{
    // Store all the elements of both lists in the map
    unordered_map<int, int> eleOcc;
    storeEle(head1, head2, eleOcc);

    Node *intersection_list = getIntersection(eleOcc);
    Node *union_list = getUnion(eleOcc);

    printf("\nIntersection list is \n");
    printList(intersection_list);

    printf("\nUnion list is \n");
    printList(union_list);
}

/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;

    /* create a linked lits 11->10->15->4->20 */
    push(&head1, 1);
    push(&head1, 2);
    push(&head1, 3);
    push(&head1, 4);
    push(&head1, 5);

    /* create a linked lits 8->4->2->10 */
    push(&head2, 1);
    push(&head2, 3);
    push(&head2, 5);
    push(&head2, 6);

    printf("First list is \n");
    printList(head1);

    printf("\nSecond list is \n");
    printList(head2);

    printUnionIntersection(head1, head2);
```

```
    return 0;  
}
```

Output:

```
First list is  
5 4 3 2 1  
Second list is  
6 5 3 1  
Intersection list is  
3 5 1  
Union list is  
3 4 6 5 2 1
```

We can also handle the case of duplicates by maintaining separate Hash for both the lists.

Time Complexity : $O(m + n)$

Auxiliary Space : $O(m + n)$

Source

<https://www.geeksforgeeks.org/union-intersection-two-linked-lists-set-3-hashing/>

Chapter 395

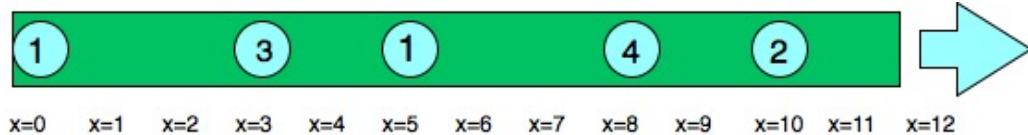
Water drop problem

Water drop problem - GeeksforGeeks

Consider a pipe of length L. The pipe has N water droplets at N different positions within it. Each water droplet is moving towards the end of the pipe($x=L$) at different rates. When a water droplet mixes with another water droplet, it assumes the speed of the water droplet it is mixing with. Determine the no of droplets that come out of the end of the pipe.

Refer to the figure below:

Numbers on circles indicates speed of water droplets



Examples:

Input: `length = 12, position = [10, 8, 0, 5, 3],
speed = [2, 4, 1, 1, 3]`

Output: 3

Explanation:

Droplets starting at $x=10$ and $x=8$ become a droplet, meeting each other at $x=12$ at time =1 sec.

The droplet starting at 0 doesn't mix with any other droplet, so it is a drop by itself.

Droplets starting at $x=5$ and $x=3$ become a single drop, mixing with each other at $x=6$ at time = 1 sec.

Note that no other droplets meet these drops before the end of the pipe, so the answer is 3.

Refer to the figure below

Numbers on circles indicates speed of water droplets.

Approach:

This problem uses greedy technique.

A drop will mix with another drop if two conditions are met:

1. If the drop is faster than the drop it is mixing with
2. If the position of the faster drop is behind the slower drop.

We use an array of **pairs** to store the position and the time that ith drop would take to reach the end of the pipe. Then we **sort** the array according to the position of the drops. Now we have a fair idea of which drops lie behind which drops and their respective time taken to reach the end. More time means less speed and less time means more speed. Now all the drops before a slower drop will mix with it. And all the drops after the slower drop will mix with the next slower drop and so on.

For example if the times to reach the end are- 12, 3, 7, 8, 1 (sorted according to positions)

0th drop is slowest, it won't mix with the next drop

1st drop is faster than the 2nd drop so they will mix and 2nd drop is faster than 3rd drop so all three will mix together. They cannot mix with the 4th drop because that is faster.

So we use a **stack** to maintain the local maxima of the times.

No of local maximal + residue(drops after last local maxima) = total no of drops

```
#include <bits/stdc++.h>
using namespace std;
int drops(int length, int position[], int speed[], int n)
{
    // stores position and time taken by a single
    // drop to reach the end as a pair
    vector<pair<int, double>> m(n);

    int i;
    for (i = 0; i < n; i++) {

        // calculates distance needs to be
        // covered by the ith drop
        int p = length - position[i];

        // inserts initial position of the
        // ith drop to the pair
        m[i].first = position[i];

        // inserts time taken by ith drop to reach
        // the end to the pair
        m[i].second = p * 1.0 / speed[i];
    }

    // sorts the pair according to increasing
    // order of their positions
    sort(m.begin(), m.end());
    int k = 0; // counter for no of final drops

    // stack to maintain the next slower drop
    // which might coalesce with the current drop
    stack<double> s;
```

```
// we traverse the array demo right to left
// to determine the slower drop
for (i = n - 1; i >= 0; i--)
{
    if (s.empty()) {
        s.push(m[i].second);
    }

    // checks for next slower drop
    if (m[i].second > s.top())
    {
        s.pop();
        k++;
        s.push(m[i].second);
    }
}

// calculating residual drops in the pipe
if (!s.empty())
{
    s.pop();
    k++;
}
return k;
}

// driver function
int main()
{
    int length = 12; // length of pipe
    int position[] = { 10, 8, 0, 5, 3 }; // position of droplets
    int speed[] = { 2, 4, 1, 1, 3 }; // speed of each droplets
    int n = sizeof(speed)/sizeof(speed[0]);
    cout << drops(length, position, speed, n);
    return 0;
}
```

Output:

3

Source

<https://www.geeksforgeeks.org/water-drop-problem/>

Chapter 396

Ways to sort list of dictionaries by values in Python – Using itemgetter

[Ways to sort list of dictionaries by values in Python – Using itemgetter - GeeksforGeeks](#)

Previous article of this segment dealt with sorting list of dictionaries by values using lambda function.

[Ways to sort list of dictionaries by values in Python – Using lambda function](#)

This article aims at performing this functionality using itemgetter and showing visible differences.

Itemgetter can be used instead of lambda function to achieve the similar functionality. Outputs in same way as sorted() and lambda, but has different internal implementation. It takes keys of dictionaries and converts them into tuples. It reduces overhead and is **faster** and more efficient. The “**operator**” module has to be imported for its working. The code is explained below

```
# Python code demonstrate the working of sorted()
# and itemgetter

# importing "operator" for implementing itemgetter
from operator import itemgetter

# Initializing list of dictionaries
lis = [{ "name" : "Nandini", "age" : 20},
       { "name" : "Manjeet", "age" : 20 },
       { "name" : "Nikhil" , "age" : 19 }]

# using sorted and itemgetter to print list sorted by age
```

```
print "The list printed sorting by age: "
print sorted(lis, key=itemgetter('age'))

print ("\r")

# using sorted and itemgetter to print list sorted by both age and name
# notice that "Manjeet" now comes before "Nandini"
print "The list printed sorting by age and name: "
print sorted(lis, key=itemgetter('age', 'name'))

print ("\r")

# using sorted and itemgetter to print list sorted by age in descending order
print "The list printed sorting by age in descending order: "
print sorted(lis, key=itemgetter('age'), reverse = True)
```

Output:

```
The list printed sorting by age:
[{'age': 19, 'name': 'Nikhil'}, {'age': 20, 'name': 'Nandini'}, {'age': 20, 'name': 'Manjeet'}]

The list printed sorting by age and name:
[{'age': 19, 'name': 'Nikhil'}, {'age': 20, 'name': 'Manjeet'}, {'age': 20, 'name': 'Nandini'}]

The list printed sorting by age in descending order:
[{'age': 20, 'name': 'Nandini'}, {'age': 20, 'name': 'Manjeet'}, {'age': 19, 'name': 'Nikhil'}]
```

- **Performance :** itemgetter performs better than lambda functions in the context of time.
- **Concise :** itemgetter looks more concise when accessing multiple values than lambda functions.

```
itemgetter(1,3,4,5) ---> Looks more concise
key(s[1], s[2], s[3], s[4]) ---> Looks less concise
```

Source

<https://www.geeksforgeeks.org/ways-sort-list-dictionaries-values-python-using-itemgetter/>

Chapter 397

When does the worst case of Quicksort occur?

When does the worst case of Quicksort occur? - GeeksforGeeks

The answer depends on strategy for choosing pivot. In early versions of Quick Sort where leftmost (or rightmost) element is chosen as pivot, the worst occurs in following cases.

- 1) Array is already sorted in same order.
- 2) Array is already sorted in reverse order.
- 3) All elements are same (special case of case 1 and 2)

Since these cases are very common use cases, the problem was easily solved by choosing either a random index for the pivot, choosing the middle index of the partition or (especially for longer partitions) choosing the median of the first, middle and last element of the partition for the pivot. With these modifications, the worst case of Quick sort has less chances to occur, but worst case can still occur if the input array is such that the maximum (or minimum) element is always chosen as pivot.

References:

<http://en.wikipedia.org/wiki/Quicksort>

Source

<https://www.geeksforgeeks.org/when-does-the-worst-case-of-quicksort-occur/>

Chapter 398

Where is Heap Sort used practically?

Where is Heap Sort used practically? - GeeksforGeeks

Although [QuickSort](#) works better in practice, the advantage of [HeapSort](#) worst case upper bound of $O(n\log n)$.

[MergeSort](#) also has upper bound as $O(n\log n)$ and works better in practice when compared to HeapSort. But MergeSort requires $O(n)$ extra space

HeapSort is not used much in practice, but can be useful in real time (or time bound where QuickSort doesn't fit) embedded systems where less space is available (MergeSort doesn't fit)

Source

<https://www.geeksforgeeks.org/where-is-heap-sort-used-practically/>

Chapter 399

Which sorting algorithm makes minimum number of memory writes?

Which sorting algorithm makes minimum number of memory writes? - GeeksforGeeks

Minimizing the number of writes is useful when making writes to some huge data set is very expensive, such as with [EEPROMs](#) or [Flash memory](#), where each write reduces the lifespan of the memory.

Among the sorting algorithms that we generally study in our data structure and algorithm courses, [Selection Sort](#) makes least number of writes (it makes $O(n)$ swaps). But, [Cycle Sort](#) almost always makes less number of writes compared to Selection Sort. In Cycle Sort, each value is either written zero times, if it's already in its correct position, or written one time to its correct position. This matches the minimal number of overwrites required for a completed in-place sort.

Sources:

http://en.wikipedia.org/wiki/Cycle_sort

http://en.wikipedia.org/wiki/Selection_sort

Source

<https://www.geeksforgeeks.org/which-sorting-algorithm-makes-minimum-number-of-writes/>

Chapter 400

Why Quick Sort preferred for Arrays and Merge Sort for Linked Lists?

Why Quick Sort preferred for Arrays and Merge Sort for Linked Lists? - GeeksforGeeks

Why is **Quick Sort** preferred for arrays?

Below are recursive and iterative implementations of Quick Sort and Merge Sort for arrays.

[Recursive Quick Sort for array.](#)

[Iterative Quick Sort for arrays.](#)

[Recursive Merge Sort for arrays](#)

[Iterative Merge Sort for arrays](#)

- Quick Sort in its general form is an in-place sort (i.e. it doesn't require any extra storage) whereas merge sort requires $O(N)$ extra storage, N denoting the array size which may be quite expensive. Allocating and de-allocating the extra space used for merge sort increases the running time of the algorithm.
- Comparing average complexity we find that both type of sorts have $O(N \log N)$ average complexity but the constants differ. For arrays, merge sort loses due to the use of extra $O(N)$ storage space.
- Most practical implementations of Quick Sort use randomized version. The randomized version has expected time complexity of $O(n \log n)$. The worst case is possible in randomized version also, but worst case doesn't occur for a particular pattern (like sorted array) and randomized Quick Sort works well in practice.
- Quick Sort is also a cache friendly sorting algorithm as it has good [locality of reference](#) when used for arrays.
- Quick Sort is also [tail recursive](#), therefore tail call optimizations is done.

Why is Merge Sort preferred for Linked Lists?

Below are implementations of Quicksort and Mergesort for singly and doubly linked lists.

[Quick Sort for Doubly Linked List](#)

[Quick Sort for Singly Linked List](#)

[Merge Sort for Singly Linked List](#)

[Merge Sort for Doubly Linked List](#)

- In case of [linked lists](#) the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory.
- Unlike array, in [linked list](#), we can insert items in the middle in $O(1)$ extra space and $O(1)$ time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.
- In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of $A[0]$ be x then to access $A[i]$, we can directly access the memory at $(x + i*4)$. Unlike arrays, we can not do random access in linked list.
- Quick Sort requires a lot of this kind of access. In linked list to access i 'th index, we have to travel each and every node from the head to i 'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

Related Articles:

- [Why quicksort is better than mergesort ?](#)
- [Know Your Sorting Algorithm Set 1 \(Sorting Weapons used by Programming Languages\)](#)
- [Iterative Merge Sort](#)
- [Iterative Quick Sort](#)

Thanks to [Sayan Mukhopadhyay](#) for providing initial draft for above article. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

<https://www.geeksforgeeks.org/why-quick-sort-preferred-for-arrays-and-merge-sort-for-linked-lists/>

Chapter 401

Why is it faster to process sorted array than an unsorted array ?

Why is it faster to process sorted array than an unsorted array ? - GeeksforGeeks

Here is a C++ code that illustrates that sorting the data miraculously makes the code faster than the unsorted version. Let's try out a sample C++ program to understand the problem statement better.

```
// CPP program to demonstrate processing
// time of sorted and unsorted array
#include <iostream>
#include <algorithm>
#include <ctime>
using namespace std;

const int N = 100001;

int main()
{
    int arr[N];

    // Assign random values to array
    for (int i=0; i<N; i++)
        arr[i] = rand()%N;

    // for loop for unsorted array
    int count = 0;
    double start = clock();
    for (int i=0; i<N; i++)
```

```
if (arr[i] < N/2)
    count++;

double end = clock();
cout << "Time for unsorted array :: "
    << ((end - start)/CLOCKS_PER_SEC)
    << endl;
sort(arr, arr+N);

// for loop for sorted array
count = 0;
start = clock();

for (int i=0; i<N; i++)
    if (arr[i] < N/2)
        count++;

end = clock();
cout << "Time for sorted array :: "
    << ((end - start)/CLOCKS_PER_SEC)
    << endl;

return 0;
}
```

Output :

Execution 1:

```
Time for unsorted array :: 0.00108
Time for sorted array :: 0.00053
```

Execution 2:

```
Time for unsorted array :: 0.001101
Time for sorted array :: 0.000593
```

Execution 3:

```
Time for unsorted array :: 0.0011
Time for sorted array :: 0.000418
```

Observe that time taken for processing a sorted array is less as compared to unsorted array.
The reason for this optimisation for sorted array is **branch prediction**.

What is branch prediction ?

In computer architecture, branch prediction means determining whether a conditional branch(jump) in the instruction flow of a program is likely to be taken or not.

All the pipelined processors do branch prediction in some form, because they must guess the address of the next instruction to fetch before the current instruction has been executed.

How branch prediction in applicable on above case ?

The **if** condition checks that **arr[i] < 5000**, but if you observe in case of sorted array, after passing the number 5000 the condition is always false, and before that it is always true, compiler optimises the code here and skips the if condition which is referred as branch prediction.

Case 1 : Sorted array

```
T = if condition true
F = if condition false
arr[] = {0,1,2,3,4,5,6, .... , 4999,5000,5001, ... , 100000}
       {T,T,T,T,T,T, .... , T,     F,     F,     ... ,     F }
```

We can observe that it is very easy to predict the branch in sorted array, as the sequence is **TTTTTTTTTTTT.....FFFFFFFFFFFF**

Case 2 : Unsorted array

```
T = if condition true
F = if condition false
arr[] = {5,0,5000,10000,17,13, ... , 3,21000,10}
       {T,T,F,     F,     T,     T, ... , T, F,     T}
```

It is very difficult to predict that **if** statement will be false or true, hence branch prediction don't play any significant role here.

Branch prediction works on the pattern the algorithm is following or basically the history, how it got executed in previous steps. If the guess is correct, then CPU continue executing and if it goes wrong, then CPU need to flush the pipeline and roll back to the branch and restart from beginning.

In case compiler is not able to utilise branch prediction as a tool for improving performance, programmer can implement his own hacks to improve performance.

References :

- [Branch_prediction](#)
- [StackOverflow](#)
- [Pipelining in computing](#)

Source

<https://www.geeksforgeeks.org/faster-process-sorted-array-unsorted-array/>

Chapter 402

Why quicksort is better than mergesort ?

Why quicksort is better than mergesort ? - GeeksforGeeks

This a common question asked in DS interviews that despite of better worst case performance of merge sort, [quicksort](#) is considered better than [mergesort](#). There are certain reasons due to which quicksort is better especially in case of arrays:

1. **Auxiliary Space :** Mergesort uses extra space, quicksort requires little space and exhibits good cache locality. Quick sort is an in-place sorting algorithm. In-place sorting means no additional storage space is needed to perform sorting. Merge sort requires a temporary array to merge the sorted arrays and hence it is not in-place giving Quick sort the advantage of space.
2. **Worst Cases :** The worst case of quicksort $O(n^2)$ can be avoided by using randomized quicksort. It can be easily avoided with high probability by choosing the right pivot. Obtaining an average case behavior by choosing right pivot element makes it improvise the performance and becoming as efficient as Merge sort.
3. **Locality of reference :** Quicksort in particular exhibits good cache locality and this makes it faster than merge sort in many cases like in virtual memory environment.
4. **Merge sort is better for large data structures:** Mergesort is a stable sort, unlike quicksort and heapsort, and can be easily adapted to operate on linked lists and very large lists stored on slow-to-access media such as disk storage or network attached storage. Refer [this](#) for details

The `std::sort()` function which is present in C++ STL is a hybrid sorting algorithm provides average and worst case time complexity of $O(n\log n)$. The sorting algorithm which it uses is called Introsort.

Introsort is combination of both quicksort and heapsort, It begins with quicksort and switch to heapsort if recursion depth exceeds a level based on the number of elements being sorted.

Related Article: [Why Quick Sort preferred for Arrays and Merge Sort for Linked Lists?](https://www.geeksforgeeks.org/quicksort-better-mergesort/)

Source

<https://www.geeksforgeeks.org/quicksort-better-mergesort/>

Chapter 403

k largest(or smallest) elements in an array added Min Heap method

k largest(or smallest) elements in an array added Min Heap method - GeeksforGeeks

Question: Write an efficient program for printing k largest elements in an array. Elements in array can be in any order.

For example, if given array is [1, 23, 12, 9, 30, 2, 50] and you are asked for the largest 3 elements i.e., k = 3 then your program should print 50, 30 and 23.

Method 1 (Use Bubble k times)

Thanks to Shailendra for suggesting this approach.

- 1) Modify [Bubble Sort](#) to run the outer loop at most k times.
- 2) Print the last k elements of the array obtained in step 1.

Time Complexity: O(nk)

Like Bubble sort, other sorting algorithms like [Selection Sort](#) can also be modified to get the k largest elements.

Method 2 (Use temporary array)

K largest elements from arr[0..n-1]

- 1) Store the first k elements in a temporary array temp[0..k-1].
- 2) Find the smallest element in temp[], let the smallest element be *min*.
- 3) For each element *x* in arr[k] to arr[n-1]
If *x* is greater than the min then remove *min* from temp[] and insert *x*.
- 4) Print final k elements of temp[]

Time Complexity: O((n-k)*k). If we want the output sorted then O((n-k)*k + klogk)

Thanks to nesamani1822 for suggesting this method.

Method 3(Use Sorting)

1) Sort the elements in descending order in O(nLogn)

2) Print the first k numbers of the sorted array O(k).

Following is the implementation of above.

C++

```
// C++ code for k largest elements in an array
#include<bits/stdc++.h>
using namespace std;

void kLargest(int arr[], int n, int k)
{
    // Sort the given array arr in reverse
    // order.
    sort(arr, arr+n, greater<int>());

    // Print the first kth largest elements
    for (int i = 0; i < k; i++)
        cout << arr[i] << " ";
}

// driver program
int main()
{
    int arr[] = {1, 23, 12, 9, 30, 2, 50};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    kLargest(arr, n, k);
}

// This article is contributed by Chhavi
```

Java

```
// Java code for k largest elements in an array
import java.util.Arrays;
import java.util.Collections;

class GFG
{
    public static void kLargest(Integer [] arr, int k)
    {
        // Sort the given array arr in reverse order
        // This method doesn't work with primitive data
        // types. So, instead of int, Integer type
        // array will be used
        Arrays.sort(arr, Collections.reverseOrder());
```

```
// Print the first kth largest elements
for (int i = 0; i < k; i++)
    System.out.print(arr[i] + " ");
}

public static void main(String[] args)
{
    Integer arr[] = new Integer[]{1, 23, 12, 9,
                                  30, 2, 50};
    int k = 3;
    kLargest(arr,k);
}
// This code is contributed by Kamal Rawal
```

Python

```
''' Python3 code for k largest elements in an array'''

def kLargest(arr, k):
    # Sort the given array arr in reverse
    # order.
    arr.sort(reverse=True)
    #Print the first kth largest elements
    for i in range(k):
        print (arr[i],end=" ")

# Driver program
arr = [1, 23, 12, 9, 30, 2, 50]
#n = len(arr)
k = 3
kLargest(arr, k)

# This code is contributed by shreyanshi_arun.
```

Output:

```
50 30 23
```

Time complexity: $O(n \log n)$

Method 4 (Use Max Heap)

- 1) Build a Max Heap tree in $O(n)$
- 2) Use Extract Max k times to get k maximum elements from the Max Heap $O(k \log n)$

Time complexity: $O(n + k \log n)$

Method 5(Use Order Statistics)

- 1) Use order statistic algorithm to find the k th largest element. Please [see the topic selection in worst-case linear time \$O\(n\)\$](#)
- 2) Use [QuickSort](#) Partition algorithm to partition around the k th largest number $O(n)$.
- 3) Sort the $k-1$ elements (elements greater than the k th largest element) $O(k \log k)$. This step is needed only if sorted output is required.

Time complexity: $O(n)$ if we don't need the sorted output, otherwise $O(n+k \log k)$

Thanks to Shilpi for suggesting the first two approaches.

Method 6 (Use Min Heap)

This method is mainly an optimization of method 1. Instead of using `temp[]` array, use Min Heap.

- 1) Build a Min Heap MH of the first k elements ($\text{arr}[0]$ to $\text{arr}[k-1]$) of the given array. $O(k)$
- 2) For each element, after the k th element ($\text{arr}[k]$ to $\text{arr}[n-1]$), compare it with root of MH.
.....a) If the element is greater than the root then make it root and call [heapify](#)for MH
.....b) Else ignore it.
// The step 2 is $O((n-k)*\log k)$
- 3) Finally, MH has k largest elements and root of the MH is the k th largest element.

Time Complexity: $O(k + (n-k)\log k)$ without sorted output. If sorted output is needed then $O(k + (n-k)\log k + k \log k)$

All of the above methods can also be used to find the k th largest (or smallest) element.

Please write comments if you find any of the above explanations/algorithms incorrect, or find better ways to solve the same problem.

References:

http://en.wikipedia.org/wiki/Selection_algorithm

Source

<https://www.geeksforgeeks.org/k-largest-or-smallest-elements-in-an-array/>

Chapter 404

k smallest elements in same order using O(1) extra space

k smallest elements in same order using O(1) extra space - GeeksforGeeks

You are given an array of n-elements you have to find k smallest elements from the array but they must be in the same order as they are in given array and we are allowed to use only O(1) extra space.

Examples:

```
Input : arr[] = {4, 2, 6, 1, 5},  
        k = 3  
Output : 4 2 1  
Explanation : 1, 2 and 4 are three smallest  
numbers and 4 2 1 is their order in given array
```

```
Input : arr[] = {4, 12, 16, 21, 25},  
        k = 3  
Output : 4 12 16  
Explanation : 4, 12 and 16 are 3 smallest numbers  
and 4 12 16 is their order in given array
```

We have discussed [efficient solution to find n smallest elements](#) of above problem with using extra space of O(n). To solve it without using any extra space we will use concept of insertion sort.

The idea is to move k minimum elements to beginning in same order. To do this, we start from (k+1)-th element and move till end. For every array element, we replace the largest element of first k elements with the current element if current element is smaller than the largest. To keep the order, we use [insertion sort](#) idea.

C++

```
// CPP for printing smallest k numbers in order
#include <algorithm>
#include <iostream>
using namespace std;

// Function to print smallest k numbers
// in arr[0..n-1]
void printSmall(int arr[], int n, int k)
{
    // For each arr[i] find whether
    // it is a part of n-smallest
    // with insertion sort concept
    for (int i = k; i < n; ++i)
    {
        // find largest from first k-elements
        int max_var = arr[k-1];
        int pos = k-1;
        for (int j=k-2; j>=0; j--)
        {
            if (arr[j] > max_var)
            {
                max_var = arr[j];
                pos = j;
            }
        }

        // if largest is greater than arr[i]
        // shift all element one place left
        if (max_var > arr[i])
        {
            int j = pos;
            while (j < k-1)
            {
                arr[j] = arr[j+1];
                j++;
            }
            // make arr[k-1] = arr[i]
            arr[k-1] = arr[i];
        }
    }
    // print result
    for (int i=0; i<k; i++)
        cout << arr[i] << " ";
}

// Driver program
int main()
```

```
{  
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };  
    int n = sizeof(arr) / sizeof(arr[0]);  
    int k = 5;  
    printSmall(arr, n, k);  
    return 0;  
}
```

Java

```
// Java for printing smallest k numbers in order  
import java.util.*;  
import java.lang.*;  
  
public class GfG {  
    // Function to print smallest k numbers  
    // in arr[0..n-1]  
    public static void printSmall(int arr[], int n, int k)  
    {  
        // For each arr[i] find whether  
        // it is a part of n-smallest  
        // with insertion sort concept  
        for (int i = k; i < n; ++i) {  
            // Find largest from top n-element  
            int max_var = arr[k - 1];  
            int pos = k - 1;  
            for (int j = k - 2; j >= 0; j--) {  
                if (arr[j] > max_var) {  
                    max_var = arr[j];  
                    pos = j;  
                }  
            }  
  
            // If largest is greater than arr[i]  
            // shift all element one place left  
            if (max_var > arr[i]) {  
                int j = pos;  
                while (j < k - 1) {  
                    arr[j] = arr[j + 1];  
                    j++;  
                }  
                // make arr[k-1] = arr[i]  
                arr[k - 1] = arr[i];  
            }  
        }  
        // print result  
        for (int i = 0; i < k; i++)  
            System.out.print(arr[i] + " ");  
    }  
}
```

```
}

// Driver function
public static void main(String argc[])
{
    int[] arr = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int n = 10;
    int k = 5;
    printSmall(arr, n, k);
}

/* This code is contributed by Sagar Shukla */
```

Python3

```
# Python 3 for printing smallest
# k numbers in order

# Function to print smallest k
# numbers in arr[0..n-1]
def printSmall(arr, n, k):

    # For each arr[i] find whether
    # it is a part of n-smallest
    # with insertion sort concept
    for i in range(k, n):

        # find largest from first k-elements
        max_var = arr[k - 1]
        pos = k - 1
        for j in range(k - 2, -1, -1):

            if (arr[j] > max_var):

                max_var = arr[j]
                pos = j

    # if largest is greater than arr[i]
    # shift all element one place left
    if (max_var > arr[i]):

        j = pos
        while (j < k - 1):

            arr[j] = arr[j + 1]
```

```
j += 1

# make arr[k-1] = arr[i]
arr[k - 1] = arr[i]

# print result
for i in range(0, k):
    print(arr[i], end = " ")

# Driver program
arr = [1, 5, 8, 9, 6, 7, 3, 4, 2, 0]
n = len(arr)
k = 5
printSmall(arr, n, k)

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
// C# for printing smallest k numbers in order
using System;

public class GfG {

    // Function to print smallest k numbers
    // in arr[0..n-1]
    public static void printSmall(int []arr,
                                  int n, int k)
    {

        // For each arr[i] find whether
        // it is a part of n-smallest
        // with insertion sort concept
        for (int i = k; i < n; ++i) {

            // Find largest from top n-element
            int max_var = arr[k - 1];
            int pos = k - 1;
            for (int j = k - 2; j >= 0; j--)
            {
                if (arr[j] > max_var)
                {
                    max_var = arr[j];
                    pos = j;
                }
            }
            arr[pos] = arr[i];
            arr[i] = max_var;
        }
    }
}
```

```
        }
    }

    // If largest is greater than arr[i]
    // shift all element one place left
    if (max_var > arr[i]) {
        int j = pos;
        while (j < k - 1) {
            arr[j] = arr[j + 1];
            j++;
        }

        // make arr[k-1] = arr[i]
        arr[k - 1] = arr[i];
    }
}

// print result
for (int i = 0; i < k; i++)
    Console.Write(arr[i] + " ");
}

// Driver function
public static void Main()
{
    int[] arr = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int n = 10;
    int k = 5;

    printSmall(arr, n, k);
}
}

/* This code is contributed by Vt_m */
```

PHP

```
<?php
// PHP for printing smallest
// k numbers in order

// Function to print smallest k
// numbers in arr[0..n-1]
function printSmall($arr, $n, $k)
{

    // For each arr[i] find whether
```

```
// it is a part of n-smallest
// with insertion sort concept
for ($i = $k; $i < $n; ++$i)
{
    // find largest from
    // first k-elements
    $max_var = $arr[$k - 1];
    $pos = $k - 1;
    for ($j = $k - 2; $j >= 0; $j--)
    {
        if ($arr[$j] > $max_var)
        {
            $max_var = $arr[$j];
            $pos = $j;
        }
    }

    // if largest is greater than arr[i]
    // shift all element one place left
    if ($max_var > $arr[$i])
    {
        $j = $pos;
        while ($j < $k - 1)
        {
            $arr[$j] = $arr[$j + 1];
            $j++;
        }

        // make arr[k - 1] = arr[i]
        $arr[$k - 1] = $arr[$i];
    }
}

// print result
for ($i = 0; $i < $k; $i++)
echo $arr[$i] , " ";

}

// Driver Code
$arr = array(1, 5, 8, 9, 6, 7, 3, 4, 2, 0);
$n = count($arr);
$k = 5;
printSmall($arr, $n, $k);

// This code is contributed by Vt_m
?>
```

Output :

1 3 4 2 0

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/k-smallest-elements-order-using-o1-extra-space/>

Chapter 405

sort() in Python

sort() in Python - GeeksforGeeks

Like [C++ sort\(\)](#), [Java sort\(\)](#) and other languages, python also provides built in function to sort.

The sort function can be used to sort the list in both ascending and descending order.

To sort the list in ascending order.

Syntax

```
List_name.sort()  
This will sort the given list in ascending order.
```

This function can be used to sort list of integers, floating point number, string and others.

```
# List of Integers  
numbers = [1, 3, 4, 2]  
  
# Sorting list of Integers  
numbers.sort()  
  
print(numbers)  
  
# List of Floating point numbers  
decimalnumber = [2.01, 2.00, 3.67, 3.28, 1.68]  
  
# Sorting list of Floating point numbers  
decimalnumber.sort()  
  
print(decimalnumber)
```

```
# List of strings
words = ["Geeks", "For", "Geeks"]

# Sorting list of strings
words.sort()

print(words)
```

Output:

```
[1, 2, 3, 4]
[1.68, 2.0, 2.01, 3.28, 3.67]
['For', 'Geeks', 'Geeks']
```

To sort the list in descending order.

Syntax

```
list_name.sort(reverse=True)
This will sort the given list in descending order.
```

```
# List of Integers
numbers = [1, 3, 4, 2]

# Sorting list of Integers
numbers.sort(reverse=True)

print(numbers)

# List of Floating point numbers
decimalnumber = [2.01, 2.00, 3.67, 3.28, 1.68]

# Sorting list of Floating point numbers
decimalnumber.sort(reverse=True)

print(decimalnumber)

# List of strings
words = ["Geeks", "For", "Geeks"]

# Sorting list of strings
words.sort(reverse=True)

print(words)
```

Output:

```
[4, 3, 2, 1]
[3.67, 3.28, 2.01, 2.0, 1.68]
['Geeks', 'Geeks', 'For']
```

Syntax :

```
list_name.sort() – it sorts in ascending order
list_name.sort(reverse=True) – it sorts in descending order
list_name.sort(key=..., reverse=...) – it sorts according to user's choice
```

Parameters:

By default, `sort()` doesn't require any extra parameters. However, it has two optional parameters:

reverse – If true, the list is sorted in descending order
key – function that serves as a key for the sort comparison

Return value:

It returns a sorted list according to the passed parameter.

```
# Python program to demonstrate sorting by user's
# choice

# function to return the second element of the
# two elements passed as the paramater
def sortSecond(val):
    return val[1]

# list1 to demonstrate the use of sorting
# using using second key
list1 = [(1,2),(3,3),(1,1)]

# sorts the array in ascending according to
# second element
list1.sort(key=sortSecond)
print(list1)

# sorts the array in descending according to
# second element
list1.sort(key=sortSecond,reverse=True)
print(list1)
```

Output:

```
[(1, 1), (1, 2), (3, 3)]
[(3, 3), (1, 2), (1, 1)]
```

Thanks to [striver](#) for inputs on this topic.

Source

<https://www.geeksforgeeks.org/sort-in-python/>

Chapter 406

stable_sort() in C++ STL

stable_sort() in C++ STL - GeeksforGeeks

Like [std::sort\(\)](#), stable_sort also sorts an array. The syntax is also same.

```
// C++ program to demonstrate default behaviour of
// sort() in STL.
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int n = sizeof(arr) / sizeof(arr[0]);

    stable_sort(arr, arr + n);

    cout << "\nArray after sorting using "
        "default sort is : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
```

Output :

```
Array after sorting using default sort is :
0 1 2 3 4 5 6 7 8 9
```

So by default, sort() sorts an array in ascending order.

How to sort in descending order?

Like sort(), stable_sort() takes a third parameter that is used to specify the order in which elements are to be sorted. We can pass “greater()” function to sort in increasing order. This function does comparison in a way that puts greater element before.

```
// C++ program to demonstrate descending order
// stable sort using greater<>().
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int n = sizeof(arr) / sizeof(arr[0]);

    stable_sort(arr, arr + n, greater<int>());

    cout << "Array after sorting : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
```

Output:

```
Array after sorting :
9 8 7 6 5 4 3 2 1 0
```

When to prefer stable_sort over sort()?

Sometime we want to make sure that order of equal elements is same in sorted array as it was in original array. This can be useful if these values have associated other fields. For example, consider sorting students by marks, if two students have same marks, we may want to keep them in same order as they appear input. Please refer [stability in sorting algorithms](#) for details.

Consider following example. We have time intervals sorted by ending time and we want to sort by start time. Also, if two start times are same, we want to keep them sorted by end time. This is not guaranteed by sort().

```
// A C++ program to demonstrate STL stable_sort()
// to sort intervals according to start time.
// Given intervals are sorted according to
// ending time
#include <bits/stdc++.h>
using namespace std;
```

```
// An interval has start time and end time
struct Interval {
    int start, end;
};

// Compares two intervals according to starting
// times.
bool compareInterval(Interval i1, Interval i2)
{
    return (i1.start < i2.start);
}

int main()
{
    // Given intervals are sorted according to end time
    Interval arr[] = { {1, 3}, {2, 4}, {1, 7}, {2, 19} };
    int n = sizeof(arr) / sizeof(arr[0]);

    // sort the intervals in increasing order of
    // start time such that the start time intervals
    // appear in same order as in input.
    stable_sort(arr, arr + n, compareInterval);

    cout << "Intervals sorted by start time : \n";
    for (int i = 0; i < n; i++)
        cout << "[" << arr[i].start << ", " << arr[i].end
            << "] ";
}

return 0;
}
```

Output:

```
Intervals sorted by start time :
[1, 3] [1, 7] [2, 4] [2, 19]
```

Implementation

`sort()` function usually uses [Introsort](#). Therefore, `sort()` may preserve the physical order of semantically equivalent values but can't be guaranteed.

`stable_sort()` function usually uses [mergesort](#). Therefore, `stable_sort()` preserve the physical order of semantically equivalent values and its guaranteed.

Time Complexity

For `sort()` it is $O(n * \log(n))$

For `stable_sort()` it is $O(n * \log^2(n))$ if additional memory linearly proportional to length is not available. If its available then $O(n * \log(n))$.

Source

https://www.geeksforgeeks.org/stable_sort-c-stl/

Chapter 407

std::sort() in C++ STL

std::sort() in C++ STL - GeeksforGeeks

We have discussed [qsort\(\)](#) in C. C++ STL provides a similar function sort that sorts a vector or array (items with random access). Below is a simple program to show working of sort().

```
// C++ program to demonstrate default behaviour of
// sort() in STL.
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
    int n = sizeof(arr)/sizeof(arr[0]);

    sort(arr, arr+n);

    cout << "\nArray after sorting using "
        "default sort is : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
```

Output :

```
Array after sorting using default sort is :
0 1 2 3 4 5 6 7 8 9
```

So by default, sort() sorts an array in ascending order.

How to sort in descending order?

sort() takes a third parameter that is used to specify the order in which elements are to be sorted. We can pass “greater()” function to sort in descending order. This function does comparison in a way that puts greater element before.

```
// C++ program to demonstrate descending order sort using
// greater<>().
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
    int n = sizeof(arr)/sizeof(arr[0]);

    sort(arr, arr+n, greater<int>());

    cout << "Array after sorting : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
```

Output:

```
Array after sorting :
9 8 7 6 5 4 3 2 1 0
```

How to sort in particular order?

We can also write our own comparator function and pass it as a third parameter.

```
// A C++ program to demonstrate STL sort() using
// our own comparator
#include<bits/stdc++.h>
using namespace std;

// An interval has start time and end time
struct Interval
{
    int start, end;
};

// Compares two intervals according to starting times.
bool compareInterval(Interval i1, Interval i2)
```

```
{  
    return (i1.start < i2.start);  
}  
  
int main()  
{  
    Interval arr[] = { {6,8}, {1,9}, {2,4}, {4,7} };  
    int n = sizeof(arr)/sizeof(arr[0]);  
  
    // sort the intervals in increasing order of  
    // start time  
    sort(arr, arr+n, compareInterval);  
  
    cout << "Intervals sorted by start time : \n";  
    for (int i=0; i<n; i++)  
        cout << "[" << arr[i].start << "," << arr[i].end  
           << "] ";  
  
    return 0;  
}
```

Output:

```
Intervals sorted by start time :  
[1,9] [2,4] [4,7] [6,8]
```

This article is contributed by **Shubham Agrawal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [rahuku](#)

Source

<https://www.geeksforgeeks.org/sort-c-stl/>