Array Data Structure

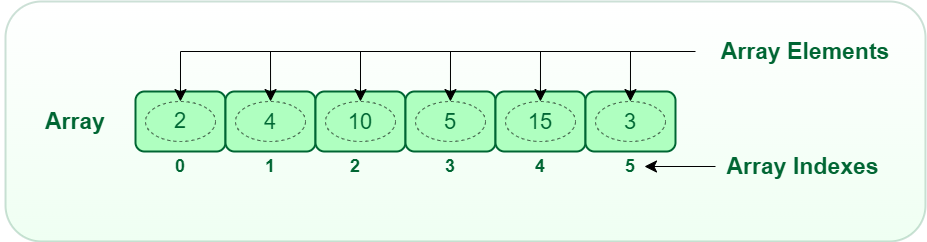
[**Data Structure and Algorithms Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=article_ad_mid&utm_campaign=negative_keyword_negation)

[**Practice Problems on Arrays**](https://practice.geeksforgeeks.org/explore/?category%5B%5D=Arrays&page=1&category%5B%5D=Arrays)

[**Recent articles on Arrays**](https://www.geeksforgeeks.org/category/c-arrays/)

**What is Array?**

*An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).*

[](https://media.geeksforgeeks.org/wp-content/uploads/20220721080308/array.png)

*Array Data Structure*

The above image can be looked as a top-level view of a staircase where you are at the base of the staircase. Each element can be uniquely identified by their index in the array (in a similar way as you could identify your friends by the step on which they were on in the above example).

**Topics :**

* [Introduction](https://www.geeksforgeeks.org/array-data-structure/?ref=ghm#introduction)
* [Implementation in different languages](https://www.geeksforgeeks.org/array-data-structure/?ref=ghm#difflang)
* [Basic Operations](https://www.geeksforgeeks.org/array-data-structure/?ref=ghm#basicop)
* [Standard problem on Array](https://www.geeksforgeeks.org/array-data-structure/?ref=ghm#standard)

**Array Introduction:**

1. [What is Array](https://www.geeksforgeeks.org/what-is-array/)
2. [Introduction to Arrays – Data Structure and Algorithm Tutorials](https://www.geeksforgeeks.org/introduction-to-arrays-data-structure-and-algorithm-tutorials/)
3. [Applications, Advantages and Disadvantages of Array](https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-array-data-structure/)

**Introduction of Array in Different language:**

1. [Arrays in C/C++](https://www.geeksforgeeks.org/arrays-in-c-language-set-1-introduction/)
2. [Arrays in Java](https://www.geeksforgeeks.org/arrays-in-java/)
3. [Arrays in Python](https://www.geeksforgeeks.org/array-python-set-1-introduction-functions/)
4. [Arrays in C#](https://www.geeksforgeeks.org/c-sharp-arrays/)
5. [Arrays in Javascript](https://www.geeksforgeeks.org/arrays-in-javascript/)

**Basic Operations:**

1. [Linear Search Algorithm](https://www.geeksforgeeks.org/linear-search/)
2. [Sort an Array](https://www.geeksforgeeks.org/c-program-to-sort-an-array-in-ascending-order/)
3. [Search, insert and delete in an unsorted array](https://www.geeksforgeeks.org/search-insert-and-delete-in-an-unsorted-array/)
4. [Search, insert and delete in a sorted array](https://www.geeksforgeeks.org/search-insert-and-delete-in-a-sorted-array/)
5. [Left Rotate an Array](https://www.geeksforgeeks.org/array-rotation/)
6. [Print array after it is right rotated K times](https://www.geeksforgeeks.org/print-array-after-it-is-right-rotated-k-times/)
7. [Generate all subarrays](https://www.geeksforgeeks.org/generating-subarrays-using-recursion/)

**Standard problem on Array:**

* **Easy**
  1. [Rearrange an array such that arr[i] = i](https://www.geeksforgeeks.org/rearrange-array-arri/)
  2. [Move all zeroes to end of array](https://www.geeksforgeeks.org/move-zeroes-end-array/)
  3. [Rearrange array such that even positioned are greater than odd](https://www.geeksforgeeks.org/rearrange-array-such-that-even-positioned-are-greater-than-odd/)
  4. [Rearrange an array in maximum minimum form using Two Pointer Technique/a>](https://www.geeksforgeeks.org/rearrange-array-maximum-minimum-form/)
  5. [Segregate even and odd numbers](https://www.geeksforgeeks.org/segregate-even-odd-numbers-set-3/)
  6. [Reversal algorithm for array rotation](https://www.geeksforgeeks.org/program-for-array-rotation-continued-reversal-algorithm/)
  7. [Print left rotation of array in O(n) time and O(1) space](https://www.geeksforgeeks.org/print-left-rotation-array/)
  8. [K’th Smallest/Largest Element in Unsorted Array](https://www.geeksforgeeks.org/kth-smallest-largest-element-in-unsorted-array/)
  9. [Find the largest three elements in an array](https://www.geeksforgeeks.org/find-the-largest-three-elements-in-an-array/)
  10. [Find Second largest element in an array](https://www.geeksforgeeks.org/find-second-largest-element-array/)
  11. [Sort an array in wave form](https://www.geeksforgeeks.org/sort-array-wave-form-2/)
  12. [Sort an array which contain 1 to n values](https://www.geeksforgeeks.org/sort-array-contain-1-n-values/)
  13. [Count the number of possible triangles](https://www.geeksforgeeks.org/find-number-of-triangles-possible/)
  14. [Print All Distinct Elements of a given integer array](https://www.geeksforgeeks.org/print-distinct-elements-given-integer-array/)
  15. [Find the element that appears once](https://www.geeksforgeeks.org/find-element-appears-array-every-element-appears-twice/)
  16. [Find sub-array with given sum](https://www.geeksforgeeks.org/find-subarray-with-given-sum/)
* **Medium**
  1. [Rearrange an array such that arr[i] = i](https://www.geeksforgeeks.org/rearrange-array-arri/)
  2. [Rearrange positive and negative numbers in O(n) time and O(1) extra space](https://www.geeksforgeeks.org/rearrange-positive-and-negative-numbers-publish/)
  3. [Reorder an array according to given indexes](https://www.geeksforgeeks.org/reorder-a-array-according-to-given-indexes/)
  4. [Search an element in a sorted and rotated array](https://www.geeksforgeeks.org/search-an-element-in-a-sorted-and-pivoted-array/)
  5. [Find the Rotation Count in Rotated Sorted array](https://www.geeksforgeeks.org/find-rotation-count-rotated-sorted-array/)
  6. [K-th Largest Sum Contiguous Subarray](https://www.geeksforgeeks.org/k-th-largest-sum-contiguous-subarray/)
  7. [Find the smallest missing number](https://www.geeksforgeeks.org/find-the-first-missing-number/)
  8. [Difference Array | Range update query in O(1)](https://www.geeksforgeeks.org/difference-array-range-update-query-o1/)
  9. [Maximum profit by buying and selling a share at most twice](https://www.geeksforgeeks.org/maximum-profit-by-buying-and-selling-a-share-at-most-twice/)
  10. [Smallest subarray with sum greater than a given value](https://www.geeksforgeeks.org/minimum-length-subarray-sum-greater-given-value/)
  11. [Inversion count in Array using Merge Sort](https://www.geeksforgeeks.org/inversion-count-in-array-using-merge-sort/)
  12. [Sort an array of 0s, 1s and 2s](https://www.geeksforgeeks.org/sort-an-array-of-0s-1s-and-2s/)
  13. [Merge two sorted arrays with O(1) extra space](https://www.geeksforgeeks.org/merge-two-sorted-arrays-o1-extra-space/)
  14. [Majority Element](https://www.geeksforgeeks.org/majority-element/)
  15. [Two Pointers Technique](https://www.geeksforgeeks.org/two-pointers-technique/)
  16. [Find a peak element](https://www.geeksforgeeks.org/find-a-peak-in-a-given-array/)
  17. [Find a triplet that sum to a given value](https://www.geeksforgeeks.org/find-a-triplet-that-sum-to-a-given-value/)
  18. [Minimum increment by k operations to make all elements equal](https://www.geeksforgeeks.org/minimum-increment-k-operations-make-elements-equal/)
  19. [Equilibrium index of an array](https://www.geeksforgeeks.org/equilibrium-index-of-an-array/)
* **Hard**
  1. [Find k numbers with most occurrences in the given array](https://www.geeksforgeeks.org/find-k-numbers-occurrences-given-array/)
  2. [MO’s Algorithm](https://www.geeksforgeeks.org/mos-algorithm-query-square-root-decomposition-set-1-introduction/)
  3. [Square Root (Sqrt) Decomposition Algorithm](https://www.geeksforgeeks.org/square-root-sqrt-decomposition-algorithm/)
  4. [Sparse Table](https://www.geeksforgeeks.org/sparse-table/)
  5. [Range sum query using Sparse Table](https://www.geeksforgeeks.org/range-sum-query-using-sparse-table/)
  6. [Range Minimum Query (Square Root Decomposition and Sparse Table)](https://www.geeksforgeeks.org/range-minimum-query-for-static-array/)
  7. [Range LCM Queries](https://www.geeksforgeeks.org/range-lcm-queries/)
  8. [Merge Sort Tree for Range Order Statistics](https://www.geeksforgeeks.org/merge-sort-tree-for-range-order-statistics/)
  9. [Minimum number of jumps to reach end](https://www.geeksforgeeks.org/minimum-number-of-jumps-to-reach-end-of-a-given-array/)
  10. [Space optimization using bit manipulations](https://www.geeksforgeeks.org/space-optimization-using-bit-manipulations/)
  11. [Sort a nearly sorted (or K sorted) array](https://www.geeksforgeeks.org/nearly-sorted-algorithm/)
  12. [Find maximum value of Sum( i\*arr[i]) with only rotations on given array allowed](https://www.geeksforgeeks.org/find-maximum-value-of-sum-iarri-with-only-rotations-on-given-array-allowed/)
  13. [Median in a stream of integers (running integers)](https://www.geeksforgeeks.org/median-of-stream-of-integers-running-integers/)
  14. [Construct an array from its pair-sum array](https://www.geeksforgeeks.org/construct-array-pair-sum-array/)
  15. [Maximum equlibrium sum in an array](https://www.geeksforgeeks.org/maximum-equlibrium-sum-array/)
  16. [Leaders in an array](https://www.geeksforgeeks.org/leaders-in-an-array/)
  17. [Smallest Difference Triplet from Three arrays](https://www.geeksforgeeks.org/smallest-difference-triplet-from-three-arrays/)
  18. [Find all triplets with zero sum](https://www.geeksforgeeks.org/find-triplets-array-whose-sum-equal-zero/)
  19. [Implement two stacks in an array](https://www.geeksforgeeks.org/implement-two-stacks-in-an-array/)
  20. [Minimum increment by k operations to make all elements equal](https://www.geeksforgeeks.org/minimum-increment-k-operations-make-elements-equal/)

**Easy questions:**

**1-Rearrange an array such that arr[i] = i**

Given an array of elements of length N, ranging from 0 to N – 1. All elements may not be present in the array. If the element is not present then there will be -1 present in the array. Rearrange the array such that A[i] = i and if i is not present, display -1 at that place.

**Examples:**

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

Input : arr = {19, 7, 0, 3, 18, 15, 12, 6, 1, 8,  
 11, 10, 9, 5, 13, 16, 2, 14, 17, 4}  
Output : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,   
 11, 12, 13, 14, 15, 16, 17, 18, 19]

# Python3 program for above approach

# Function to transform the array

**def** fixArray(ar, n):

    # Iterate over the array

**for** i **in** range(n):

**for** j **in** range(n):

            # Check is any ar[j]

            # exists such that

            # ar[j] is equal to i

**if** (ar[j] **==** i):

                ar[j], ar[i] **=** ar[i], ar[j]

    # Iterate over array

**for** i **in** range(n):

        # If not present

**if** (ar[i] !**=** i):

            ar[i] **= -**1

    # Print the output

    print("Array after Rearranging")

**for** i **in** range(n):

**print**(ar[i], end **=** " ")

# Driver Code

ar **=** [ **-**1, **-**1, 6, 1, 9, 3, 2, **-**1, 4, **-**1 ]

n **=** len(ar)

# Function Call

fixArray(ar, n);

# This code is contributed by rag2127

**Output**

Array after Rearranging  
-1 1 2 3 4 -1 6 -1 -1 9

**2-Move all zeroes to end of array**

Given an array of random numbers, Push all the zero’s of a given array to the end of the array. For example, if the given arrays is {1, 9, 8, 4, 0, 0, 2, 7, 0, 6, 0}, it should be changed to {1, 9, 8, 4, 2, 7, 6, 0, 0, 0, 0}. The order of all other elements should be same. Expected time complexity is O(n) and extra space is O(1).

**Example:**

Input : arr[] = {1, 2, 0, 4, 3, 0, 5, 0};  
Output : arr[] = {1, 2, 4, 3, 5, 0, 0, 0};

Input : arr[] = {1, 2, 0, 0, 0, 3, 6};  
Output : arr[] = {1, 2, 3, 6, 0, 0, 0};

# Python3 code to move all zeroes

# at the end of array

# Function which pushes all

# zeros to end of an array.

**def** pushZerosToEnd(arr, n):

    count **=** 0 # Count of non-zero elements

    # Traverse the array. If element

    # encountered is non-zero, then

    # replace the element at index

    # 'count' with this element

**for** i **in** range(n):

**if** arr[i] !**=** 0:

            # here count is incremented

            arr[count] **=** arr[i]

            count**+=**1

    # Now all non-zero elements have been

    # shifted to front and 'count' is set

    # as index of first 0. Make all

    # elements 0 from count to end.

**while** count < n:

        arr[count] **=** 0

        count **+=** 1

# Driver code

arr **=** [1, 9, 8, 4, 0, 0, 2, 7, 0, 6, 0, 9]

n **=** len(arr)

pushZerosToEnd(arr, n)

print("Array after pushing all zeros to end of array:")

print(arr)

# This code is contributed by "Abhishek Sharma 44"

**Output**

Array after pushing all zeros to end of array:  
1 9 8 4 2 7 6 9 0 0 0 0

**3-Rearrange array such that even positioned are greater than odd**

Given an array A of n elements, sort the array according to the following relations :

, if i is even.

, 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 the largest [n/2] elements to the even positions and the rest of the elements to the odd positions, our problem is solved. Because element at the odd position will always be less than the element at the even position as it is the maximum element and vice versa. Sort the array and assign the first [n/2] elements at even positions.*

Below is the implementation of the above approach:

# 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".

**Output**

1 5 2 3 2

**Rearrange an array in maximum minimum form using Two Pointer Technique**

Given a sorted array of positive integers, rearrange the array alternately i.e first element should be a maximum value, at second position minimum value, at third position second max, at fourth position second min, and so on.

**Examples:**

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

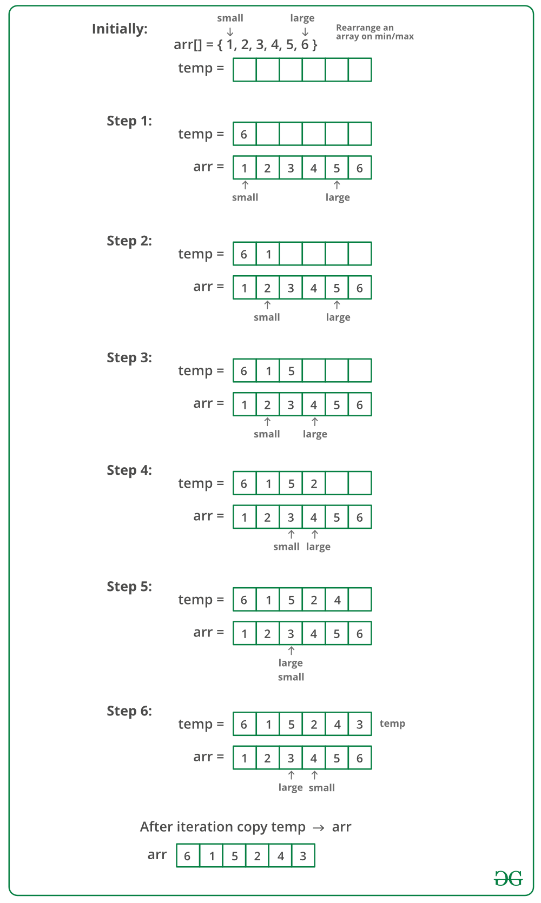
***Output****: arr[] = {7, 1, 6, 2, 5, 3, 4}*

***Input****: arr[] = {1, 2, 3, 4, 5, 6}*

***Output****: arr[] = {6, 1, 5, 2, 4, 3}*

**Rearrange an array in maximum minimum form using**[Two Pointers](https://www.geeksforgeeks.org/two-pointers-technique/)**:**

*The idea is to use an auxiliary array. We maintain two pointers one to the leftmost or smallest element and the other to the 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 the auxiliary array back to the original array.*



Below is the implementation of the above approach:

# 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 code

arr **=** [1, 2, 3, 4, 5, 6]

n **=** len(arr)

print("Original Array")

print(arr)

**print**("Modified Array")

**print**(rearrange(arr, n))

# This code is contributed by Pratik Chhajer

**Output**

Original Array  
1 2 3 4 5 6   
Modified Array  
6 1 5 2 4 3

**Segregate even and odd numbers | Set 3**

iven an array arr[] of integers, segregate even and odd numbers in the array. Such that all the even numbers should be present first, and then the odd numbers.

**Examples:**

***Input:****arr[] =**1 9 5 3 2 6 7 11*

***Output:****2 6 5 3 1 9 7 11*

***Input:****arr[] = 1 3 2 4 7 6 9 10*

***Output:****2 4 6 10 7 1 9 3*

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have discussed two different approaches in below posts:

1. [Segregate Even and Odd numbers](https://www.geeksforgeeks.org/segregate-even-and-odd-numbers/)
2. [Segregate even and odd numbers | Set 2](https://www.geeksforgeeks.org/segregate-even-odd-set-2/)

**Brute-Force Solution:**

As we need to maintain the order of elements then this can be done in the following steps :

1. Create a temporary array A of size n and an integer index which will keep the index of elements inserted .
2. Initialize index with zero and iterate over the original array and if even number is found then put that number at A[index] and then increment the value of index .
3. Again iterate over array and if an odd number is found then put it in A[index] and then increment the value of index.
4. Iterate over the temporary array A and copy its values in the original array.

# Python3 implementation of the above approach

**def** arrayEvenAndOdd(arr,  n):

    index **=** 0;

    a **=** [0 **for** i **in** range(n)]

**for** i **in** range(n):

**if** (arr[i] **%** 2 **==** 0):

            a[index] **=** arr[i]

            ind **+=** 1

**for** i **in** range(n):

**if** (arr[i] **%** 2 !**=** 0):

            a[index] **=** arr[i]

            ind **+=** 1

**for** i **in** range(n):

**print**(a[i], end **=** " ")

    print()

# Driver code

arr **=** [ 1, 3, 2, 4, 7, 6, 9, 10 ]

n **=** len(arr)

# Function call

arrayEvenAndOdd(arr, n)

# This code is contributed by rohitsingh07052

**Output**

2 4 6 10 1 3 7 9

**Reversal algorithm for Array rotation**

Given an [array](https://www.geeksforgeeks.org/introduction-to-arrays/) **arr[]** of size **N**, the task is to rotate the array by **d** position to the left.

**Examples:**

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

***Output:****3, 4, 5, 6, 7, 1, 2*

***Explanation:****If the array is rotated by 1 position to the left,*

*it becomes {2, 3, 4, 5, 6, 7, 1}.*

*When it is rotated further by 1 position,*

*it becomes: {3, 4, 5, 6, 7, 1, 2}*

***Input:****arr[] = {1, 6, 7, 8}, d = 3*

***Output:****8, 1, 6, 7*

**Approach:** We have already discussed several methods in [**this**](https://www.geeksforgeeks.org/array-rotation/) post. The ways discussed there are:

1. Using another temporary array.
2. Rotating one by one.
3. Using a juggling algorithm.

**Another Approach (The Reversal Algorithm):** Here we will be discussing another method which uses the concept of reversing a part of array. The intuition behind the idea is mentioned below:

**Intuition:**

*If we observe closely, we can see that a group of array elements is changing its position. For example see the following array:*

***arr[] = {1, 2, 3, 4, 5, 6, 7}****and****d = 2****. The rotated array is****{3, 4, 5, 6, 7, 1, 2}***

*The group having the first two elements is moving to the end of the array. This is like reversing the array.*

1. *But the issue is that if we only reverse the array, it becomes {7, 6, 5, 4, 3, 2, 1}.*
2. *After rotation the elements in the chunks having the first 5 elements****{7, 6, 5, 4, 3}****and the last 2 elements****{2, 1}****should be in the actual order as of the initial array [i.e.,****{3, 4, 5, 6, 7} and {1, 2}****]but here it gets reversed.*
3. *So if those blocks are reversed again we get the desired rotated array.*

*So the sequence of operations is:*

1. *Reverse the whole array*
2. *Then reverse the last ‘d’ elements and*
3. *Then reverse the first (N-d) elements.*

*As we are performing reverse operations it is also similar to the following sequence:*

1. *Reverse the first ‘d’ elements*
2. *Reverse last (N-d) elements*
3. *Reverse the whole array.*

**Algorithm:** The algorithm can be described with the help of the below pseudocode:

**Pseudocode:**

*Algorithm reverse(arr, start, end):*

*mid = (start + end)/2*

*loop from i = start to mid:*

*swap (arr[i], arr[end-(mid-i+1)])*

*Algorithm rotate(arr, d, N):*

*reverse(arr, 1, d) ;*

*reverse(arr, d + 1, N);*

*reverse(arr, 1, N);*

**Illustration:**

Follow the illustration below to for  better understanding of the algorithm and intuition:

*For example take the array****arr[] = {1, 2, 3, 4, 5, 6, 7}****and****d = 2****.*

Array

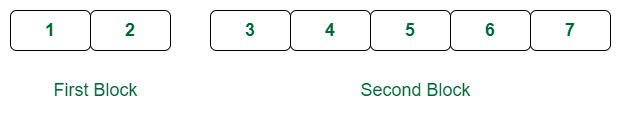
*Array*

*The rotated array will look like:*

Rotated Array

*Rotated Array*

***1st Step:****Consider the array as a combination of two blocks. One containing the first two elements and the other containing the remaining elements as shown above.*



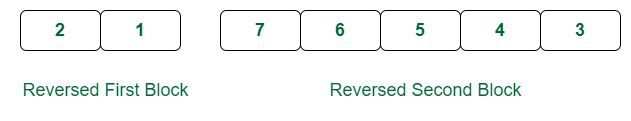
*Considered 2 blocks*

***2nd Step:****Now reverse the first****d****elements. It becomes as shown in the image*



*Reverse the first K elements*

***3rd Step:****Now reverse the last****(N-d)****elements. It become as it is shown in the below image:*



*Reverse the last (N-K) elements*

***4th Step:****Now the array is the exact reversed form of how it should be if left shifted****d****times. So reverse the whole array and you will get the required rotated array.*

The total array is reversed

*The total array is reversed*

*See that the array is now the same as the rotated array.*

Below is the implementation of the above approach:

# Python program for reversal algorithm of array rotation

# Function to reverse arr[] from index start to end

**def** reverseArray(arr, start, end):

**while** (start < end):

        temp **=** arr[start]

        arr[start] **=** arr[end]

        arr[end] **=** temp

        start **+=** 1

        end **=** end**-**1

# Function to left rotate arr[] of size n by d

**def** leftRotate(arr, d):

**if** d **==** 0:

**return**

    n **=** len(arr)

    # in case the rotating factor is

    # greater than array length

    d **=** d **%** n

    reverseArray(arr, 0, d**-**1)

    reverseArray(arr, d, n**-**1)

    reverseArray(arr, 0, n**-**1)

# Function to print an array

**def** printArray(arr):

**for** i **in** range(0, len(arr)):

        print (arr[i],end**=**' ')

# Driver function to test above functions

arr **=** [1, 2, 3, 4, 5, 6, 7]

n **=** len(arr)

d **=** 2

leftRotate(arr, d)  # Rotate array by 2

printArray(arr)

# This code is contributed by Devesh Agrawal

**Output**

3 4 5 6 7 1 2

**K’th Smallest/Largest Element in Unsorted Array**

Given an array and a number **K** where **K** is smaller than the size of the array. Find the K’th smallest element in the given array. Given that all array elements are distinct.

**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*

We have discussed a similar [problem to print k largest elements](https://www.geeksforgeeks.org/k-largestor-smallest-elements-in-an-array/).

Kth smallest element

**K’th smallest element in an unsorted array using sorting:**

*Sort the given array and return the element at index K-1 in the sorted array.*

Follow the given steps to solve the problem:

1. Sort the input array in the increasing order
2. Return the element at the K-1 index (0 – Based indexing) in the sorted array

Below is the Implementation of the above approach:

# Python3 program to find K'th smallest  
# element

# Function to return K'th smallest  
# element in a given array

def kthSmallest(arr, N, K):

# Sort the given array  
 arr.sort()

# Return k'th element in the  
 # sorted array  
 return arr[K-1]

# Driver code  
if \_\_name\_\_ == '\_\_main\_\_':  
 arr = [12, 3, 5, 7, 19]  
 N = len(arr)  
 K = 2

# Function call  
 print("K'th smallest element is",  
 kthSmallest(arr, N, K))

# This code is contributed by  
# Shrikant13

**Output**

K'th smallest element is 5

**Time Complexity:**O(N log N)

**Auxiliary Space:**O(1)

**K’th smallest element in an unsorted array using set data structure:**

*Set data structure can be used to find the kth smallest element as it stores the distinct elements in sorted order. Set can be used because it is mentioned in the question that all the elements in the array are distinct.*

Follow the given steps to solve the problem:

1. Insert all array elements into the set
2. Advance the iterator to the Kth element in the set
3. Return the value of the element at which the iterator is pointing

Below is the Implementation of the above approach:

# Python3 code for the above approach

if \_\_name\_\_ == '\_\_main\_\_':  
 arr = [12, 3, 5, 7, 19]  
 N = len(arr)  
 K = 4

s = set(arr)

for itr in s:  
 if K == 1:  
 print(itr) # itr is the Kth element in the set  
 break  
 K -= 1

# This code is contributed by Abhijeet Kumar(abhijeet19403)

**Output**

12

**Find the largest three distinct elements in an array**

Given an array with all distinct elements, find the largest three elements. Expected time complexity is O(n) and extra space is O(1).

**Examples :**

Input: arr[] = {10, 4, 3, 50, 23, 90}  
Output: 90, 50, 23

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Method 1:**

**Algorithm:**

1) Initialize the largest three elements as minus infinite.  
 first = second = third = -∞

2) Iterate through all elements of array.  
 a) Let current array element be x.  
 b) If (x > first)  
 {  
 // This order of assignment is important  
 third = second  
 second = first  
 first = x   
 }  
 c) Else if (x > second and x != first)  
 {  
 third = second  
 second = x   
 }  
 d) Else if (x > third and x != second)  
 {  
 third = x   
 }

3) Print first, second and third.

Below is the implementation of the above algorithm.

# Python3 code to find largest three

# elements in an array

**import** sys

# Function to print three largest

# elements

**def** print3largest(arr, arr\_size):

    # There should be atleast three

    # elements

**if** (arr\_size < 3):

**print**(" Invalid Input ")

**return**

    third **=** first **=** second **= -**sys.maxsize

**for** i **in** range(0, arr\_size):

        # If current element is greater

        # than first

**if** (arr[i] > first):

            third **=** second

            second **=** first

            first **=** arr[i]

        # If arr[i] is in between first

        # and second then update second

**elif** (arr[i] > second):

            third **=** second

            second **=** arr[i]

**elif** (arr[i] > third):

            third **=** arr[i]

**print**("Three largest elements are",

                  first, second, third)

# Driver program to test above function

arr **=** [12, 13, 1, 10, 34, 1]

n **=** len(arr)

print3largest(arr, n)

# This code is contributed by Smitha Dinesh Semwal

# and edited by Ayush Singla(@ayusin51).

**Output**

Three largest elements are 34 13 12

**Find Second largest element in an array**

Given an array of integers, our task is to write a program that efficiently finds the second-largest element present in the array.

**Example:**

**Input:** arr[] = {12, 35, 1, 10, 34, 1}  
**Output:** The second largest element is 34.  
**Explanation:** The largest element of the   
array is 35 and the second   
largest element is 34

**Input:** arr[] = {10, 5, 10}  
**Output:** The second largest element is 5.  
**Explanation:** The largest element of   
the array is 10 and the second   
largest element is 5

**Input:** arr[] = {10, 10, 10}  
**Output:** The second largest does not exist.  
**Explanation:** Largest element of the array   
is 10 there is no second largest element

Recommended Problem

Second Largest

**Naive approach:**

*The idea is to sort the array in descending order and then return the second element which is not equal to the largest element from the sorted array.*

Below is the implementation of the above idea:

# Python3 program to find second

# largest element in an array

# Function to print the

# second largest elements

**def** print2largest(arr,

                  arr\_size):

  # There should be

  # atleast two elements

**if** (arr\_size < 2):

**print**(" Invalid Input ")

**return**

  # Sort the array

  arr.sort

  # Start from second last

  # element as the largest

  # element is at last

**for** i **in** range(arr\_size**-**2,

**-**1, **-**1):

    # If the element is not

    # equal to largest element

**if** (arr[i] !**=** arr[arr\_size **-** 1]) :

      print("The second largest element is",

            arr[i])

**return**

**print**("There is no second largest element")

# Driver code

arr **=** [12, 35, 1, 10, 34, 1]

n **=** len(arr)

print2largest(arr, n)

# This code is contributed by divyeshrabadiya07

**Output**

The second largest element is 34

**Sort an array in wave form**

Given an unsorted array of integers, sort the array into a wave array. An array **arr[0..n-1]** is sorted in wave form if:

**arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= …..**

**Examples:**

***Input:****arr[] = {10, 5, 6, 3, 2, 20, 100, 80}*

***Output:****arr[] = {10, 5, 6, 2, 20, 3, 100, 80}*

***Explanation:***

*here you can see {10, 5, 6, 2, 20, 3, 100, 80} first element is larger than the second and the same thing is repeated again and again. large element – small element-large element -small element and so on .it can be small element-larger element – small element-large element -small element too. all you need to maintain is the up-down fashion which represents a wave. there can be multiple answers.*

***Input:****arr[] = {20, 10, 8, 6, 4, 2}*

***Output:****arr[] = {20, 8, 10, 4, 6, 2}*

Recommended Practice

[Wave Array](https://practice.geeksforgeeks.org/problems/wave-array-1587115621/1/)

[Try It!](https://practice.geeksforgeeks.org/problems/wave-array-1587115621/1/)

**What is a wave array?**

well, you have seen waves right? how do they look? if you will form a graph of them it would be some in some up-down fashion. that is what you have to do here, you are supposed to arrange numbers in such a way that if we will form a graph it will be in an up-down fashion rather than a straight line.

**Wave Array using sorting**

*A idea is to use sorting. First sort the input array, then swap all adjacent elements.*

Follow the steps mentioned below to implement the idea:

1. Sort the array.
2. Traverse the array from index**0** to **N-1,** and increase the value of the index by **2**.
3. While traversing the array swap **arr[i]** with **arr[i+1].**
4. Print the final array.

Below is the implementation of the above approach:

# 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],end**=**" ")

# This code is contributed by \_\_Devesh Agrawal\_\_

**Output**

2 1 10 5 49 23 90

**Time Complexity:**O(N\*log(N))

**Auxiliary Space:**O(1)

**Wave Array Optimized Approach**

*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 oddly positioned elements.*

Follow the steps mentioned below to implement the idea:

1. Traverse all even positioned elements of the input array, and do the following.
2. If the current element is smaller than the previous odd element, swap the previous and current.
3. If the current element is smaller than the next odd element, swap next and current.

Below is the implementation of the above approach:

# 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 **-** 1, 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], end**=**" ")

# This code is contributed by \_\_Devesh Agrawal\_\_

**Output**

90 10 49 1 5 2 23

**Count the number of possible triangles**

Given an unsorted array of positive integers, find the number of triangles that can be formed with three different array elements as three sides of triangles. For a triangle to be possible from 3 values, the sum of any of the two values (or sides) must be greater than the third value (or third side).

**Examples:**

***Input:****arr= {4, 6, 3, 7}*

***Output:****3*

***Explanation:****There are three triangles*

*possible {3, 4, 6}, {4, 6, 7} and {3, 6, 7}.*

*Note that {3, 4, 7} is not a possible triangle.*

***Input:****arr= {10, 21, 22, 100, 101, 200, 300}.*

***Output:****6*

***Explanation:****There can be 6 possible triangles:*

*{10, 21, 22}, {21, 100, 101}, {22, 100, 101},*

*{10, 100, 101}, {100, 101, 200} and {101, 200, 300}*

Recommended Problem

Count the number of possible triangles

**Naive Approach:** To solve the problem follow the below idea:

*The brute force method is to run three loops and keep track of the number of triangles possible so far. The three loops select three different values from an array. The innermost loop checks for the triangle property which specifies the sum of any two sides must be greater than the value of the third side).*

Follow the given steps to solve the problem:

1. Run three nested loops each loop starting from the index of the previous loop to the end of the array i.e run first loop from 0 to n, loop j from i to n, and k from j to n
2. Check if array[i] + array[j] > array[k], i.e. sum of two sides is greater than the third
3. Check condition 2 for all combinations of sides by interchanging i, j, k
4. If all three conditions match, then increase the count
5. Print the count

Below is the implementation of the above approach:

# Python3 code to count the number of

# possible triangles using brute

# force approach

# Function to count all possible

# triangles with arr[] elements

**def** findNumberOfTriangles(arr, n):

    # Count of triangles

    count **=** 0

    # The three loops select three

    # different values from array

**for** i **in** range(n):

**for** j **in** range(i **+** 1, n):

            # The innermost loop checks for

            # the triangle property

**for** k **in** range(j **+** 1, n):

                # Sum of two sides is greater

                # than the third

**if** (arr[i] **+** arr[j] > arr[k] **and**

                    arr[i] **+** arr[k] > arr[j] **and**

                        arr[k] **+** arr[j] > arr[i]):

                    count **+=** 1

**return** count

# Driver code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    arr **=** [10, 21, 22, 100, 101, 200, 300]

    size **=** len(arr)

    # Function call

**print**("Total number of triangles possible is",

          findNumberOfTriangles(arr, size))

# This code is contributed by shubhamsingh10

**Output**

Total number of triangles possible is 6

**Print All Distinct Elements of a given integer array**

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

Recommended Problem

Make a Distinct Digit Array

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

[MakeMyTrip](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=MakeMyTrip&sortBy=submissions)

+1 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/make-a-distinct-digit-array2007/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 7.3K

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.

**Implementation:**

# 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.

**Output**

6 10 5 4 9 120

**Find the element that appears once in an array where every other element appears twice**

Given an array of integers. All numbers occur twice except one number which occurs once. Find the number in O(n) time & constant extra space.

**Example :**

***Input:****arr[] = {2, 3, 5, 4, 5, 3, 4}*

***Output:****2*

**Approach (Brute-force):**One solution is to check every element if it appears once or not. Once an element with a single occurrence is found, return it.

Below is the implementation of the approach:

# Python code to find the array element that appears only once

# Function to find the array

# element that appears only once

**def** findSingle(A, ar\_size):

    # iterate over every element

**for** i **in** range(ar\_size):

        # Initialize count to 0

        count **=** 0

**for** j **in** range(ar\_size):

            # Count the frequency

            # of the element

**if**(A[i] **==** A[j]):

                count **+=** 1

        # If the frequency of

        # the element is one

**if**(count **==** 1):

**return** A[i]

    # If no element exist

    # at most once

**return -**1

ar **=** [2, 3, 5, 4, 5, 3, 4]

n **=** len(ar)

# Function call

print("Element occurring once is", findSingle(ar, n))

# This code is contributed by lokesh

**Output**

Element occurring once is 2

**Find Subarray with given sum | Set 1 (Non-negative Numbers)**

Given an array **arr[]** of non-negative integers and an integer **sum**, find a subarray that adds to a given **sum**.

**Note:** There may be more than one subarray with sum as the given sum, print first such subarray.

**Examples:**

***Input****: arr[] = {1, 4, 20, 3, 10, 5}, sum = 33*

***Output****: Sum found between indexes 2 and 4*

***Explanation:****Sum of elements between indices 2 and 4 is 20 + 3 + 10 = 33*

***Input****: arr[] = {1, 4, 0, 0, 3, 10, 5}, sum = 7*

***Output****: Sum found between indexes 1 and 4*

***Explanation:****Sum of elements between indices 1 and 4 is 4 + 0 + 0 + 3 = 7*

***Input****: arr[] = {1, 4}, sum = 0*

***Output****: No subarray found*

***Explanation:****There is no subarray with 0 sum*

Subarray with given sum

**Find subarray with given sum using Nested loop**

*The idea is to consider all subarrays one by one and check the sum of every subarray. Following program implements the given idea.*

*Run two loops: the outer loop picks a starting point****i****and the inner loop tries all subarrays starting from****i.***

Follow the steps given below to implement the approach:

1. Traverse the array from start to end.
2. From every index start another loop from**i**to the end of the array to get all subarrays starting from **i**, and keep a variable**currentSum** to calculate the **sum**of every subarray.
3. For every index in inner loop update **currentSum** **= currentSum + arr[j]**
4. If the **currentSum** is equal to the **given sum** then print the subarray.

 Below is the implementation of the above approach.

# A simple program to print subarray with sum as given sum

# Returns true if the there is a subarray of arr[] with sum equal to 'sum' otherwise returns false. Also, prints the result

**def** subArraySum(arr, n, sum):

    # Pick a starting point

**for** i **in** range(0,n):

        currentSum **=** arr[i]

**if**(currentSum **==** sum):

            print("Sum found at indexes",i)

**return**

**else**:

            # Try all subarrays starting with 'i'

**for** j **in** range(i**+**1,n):

                currentSum **+=** arr[i]

**if**(currentSum **==** sum):

                    print("Sum found between indexes",i,"and",j)

**return**

**print**("No Subarray Found")

# Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    arr **=** [15,2,4,8,9,5,10,23]

    n **=** len(arr)

    sum **=** 23

    subArraySum(arr, n, sum)

    # This code is contributed by ajaymakvana

**Output**

Sum found between indexes 1 and 4

**Medium questions:**

**Rearrange an array such that arr[i] = i**

Given an array of elements of length N, ranging from 0 to N – 1. All elements may not be present in the array. If the element is not present then there will be -1 present in the array. Rearrange the array such that A[i] = i and if i is not present, display -1 at that place.

**Examples:**

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

Input : arr = {19, 7, 0, 3, 18, 15, 12, 6, 1, 8,  
 11, 10, 9, 5, 13, 16, 2, 14, 17, 4}  
Output : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,   
 11, 12, 13, 14, 15, 16, 17, 18, 19]

Recommended Problem

Reorganize The Array

**Approach(Naive Approach):**

1. Nav­i­gate the numbers from 0 to n-1.
2. Now navigate through array.
3. If (i==a[j]) , then replace the element at i position with a[j] position.
4. If there is any element in which -1 is used instead of the number then it will be replaced automatically.
5. Now, iterate through the array and check if (a[i]!=i) , if it s true then replace a[i] with -1.

Below is the implementation for the above approach:

# Python3 program for above approach

# Function to transform the array

**def** fixArray(ar, n):

    # Iterate over the array

**for** i **in** range(n):

**for** j **in** range(n):

            # Check is any ar[j]

            # exists such that

            # ar[j] is equal to i

**if** (ar[j] **==** i):

                ar[j], ar[i] **=** ar[i], ar[j]

    # Iterate over array

**for** i **in** range(n):

        # If not present

**if** (ar[i] !**=** i):

            ar[i] **= -**1

    # Print the output

    print("Array after Rearranging")

**for** i **in** range(n):

**print**(ar[i], end **=** " ")

# Driver Code

ar **=** [ **-**1, **-**1, 6, 1, 9, 3, 2, **-**1, 4, **-**1 ]

n **=** len(ar)

# Function Call

fixArray(ar, n);

# This code is contributed by rag2127

**Output**

Array after Rearranging  
-1 1 2 3 4 -1 6 -1 -1 9

**Rearrange positive and negative numbers in O(n) time and O(1) extra space**

An array contains both positive and negative numbers in random order. Rearrange the array elements so that positive and negative numbers are placed alternatively. A number of positive and negative numbers need not be equal. If there are more positive numbers they appear at the end of the array. If there are more negative numbers, they too appear at the end of the array.

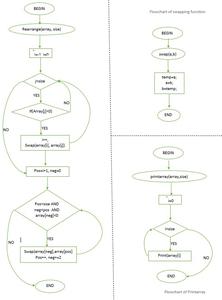
For example, if the input array is [-1, 2, -3, 4, 5, 6, -7, 8, 9], then the output should be [9, -7, 8, -3, 5, -1, 2, 4, 6]

**Note:** The partition process changes the relative order of elements. I.e. the order of the appearance of elements is not maintained with this approach. [See this](https://www.geeksforgeeks.org/rearrange-array-alternating-positive-negative-items-o1-extra-space/)for maintaining the order of appearance of elements in this problem.

The solution is to first separate positive and negative numbers using the partition process of QuickSort. In the partition process, consider 0 as the value of the pivot element so that all negative numbers are placed before positive numbers. Once negative and positive numbers are separated, we start from the first negative number and first positive number and swap every alternate negative number with the next positive number.

[Recommended: Please solve it on “***PRACTICE*** ” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/array-of-alternate-ve-and-ve-nos/0)

**Flowchart**



*Flowchart of below code*

#  Python program to put positive numbers at even indexes (0,  // 2, 4,..) and

#  negative numbers at odd indexes (1, 3, 5, ..)

# The main function that rearranges elements of given array.

# It puts  positive elements at even indexes (0, 2, ..) and

# negative numbers at odd indexes (1, 3, ..).

**def** rearrange(arr, n):

    # The following few lines are similar to partition process

    # of QuickSort.  The idea is to consider 0 as pivot and

    # divide the array around it.

    i **= -**1

**for** j **in** range(n):

**if** (arr[j] < 0):

            i **+=** 1

            # swapping of arr

            arr[i], arr[j] **=** arr[j], arr[i]

    # Now all positive numbers are at end and negative numbers

    # at the beginning of array. Initialize indexes for starting

    # point of positive and negative numbers to be swapped

    pos, neg **=** i**+**1, 0

    # Increment the negative index by 2 and positive index by 1,

    # i.e., swap every alternate negative number with next

    # positive number

**while** (pos < n **and** neg < pos **and** arr[neg] < 0):

        # swapping of arr

        arr[neg], arr[pos] **=** arr[pos], arr[neg]

        pos **+=** 1

        neg **+=** 2

# A utility function to print an array

**def** printArray(arr, n):

**for** i **in** range(n):

**print** (arr[i],end**=**" ")

# Driver program to test above functions

arr **=** [**-**1, 2, **-**3, 4, 5, 6, **-**7, 8, 9]

n **=** len(arr)

rearrange(arr, n)

printArray(arr, n)

# Contributed by Afzal

**Output:**

4 -3 5 -1 6 -7 2 8 9

**Reorder an array according to given indexes**

Given two integer arrays of same size, “arr[]” and “index[]”, reorder elements in “arr[]” according to given index array. It is not allowed to given array arr’s length.

**Example:**

***Input:****arr[]   = [10, 11, 12];*

*index[] = [1, 0, 2];*

***Output:****arr[]   = [11, 10, 12]*

*index[] = [0,  1,  2]*

***Input:****arr[]   = [50, 40, 70, 60, 90]*

*index[] = [3,  0,  4,  1,  2]*

***Output:****arr[]   = [40, 60, 90, 50, 70]*

*index[] = [0,  1,  2,  3,   4]*

Expected time complexity O(n) and auxiliary space O(1)

**We strongly recommend you to minimize your browser and try this yourself first.**

A **Simple Solution**is to use an auxiliary array temp[] of same size as given arrays. Traverse the given array and put all elements at their correct place in temp[] using index[]. Finally copy temp[] to arr[] and set all values of index[i] as i.

# Python3 program to sort

# an array according to given

# indexes

# Function to reorder

# elements of arr[] according

# to index[]

**def** reorder(arr,index, n):

    temp **=** [0] **\*** n;

    # arr[i] should be

        # present at index[i] index

**for** i **in** range(0,n):

        temp[index[i]] **=** arr[i]

    # Copy temp[] to arr[]

**for** i **in** range(0,n):

        arr[i] **=** temp[i]

        index[i] **=** i

# Driver program

arr **=** [50, 40, 70, 60, 90]

index **=** [3, 0, 4, 1, 2]

n **=** len(arr)

reorder(arr, index, n)

print("Reordered array is:")

**for** i **in** range(0,n):

**print**(arr[i],end **=** " ")

**print**("\nModified Index array is:")

**for** i **in** range(0,n):

    print(index[i],end **=** " ")

# This code is contributed by

# Smitha Dinesh Semwal

**Output:**

Reordered array is:   
40 60 90 50 70   
Modified Index array is:   
0 1 2 3 4

**Search an element in a sorted and rotated Array**

Given a sorted and rotated array **arr[]** of size **N** and a **key**, the task is to find the key in the array.

**Note:** Find the element in O(logN) time and assume that all the elements are distinct.

**Example:**

***Input  :****arr[] = {5, 6, 7, 8, 9, 10, 1, 2, 3}, key = 3*

***Output :****Found at index 8*

***Input  :****arr[] = {5, 6, 7, 8, 9, 10, 1, 2, 3}, key = 30*

***Output :****Not found*

***Input :****arr[] = {30, 40, 50, 10, 20}, key = 10*

***Output :****Found at index 3*

Recommended Practice

[Search in a Rotated Array](https://practice.geeksforgeeks.org/problems/search-in-a-rotated-array4618/1/)

[Try It!](https://practice.geeksforgeeks.org/problems/search-in-a-rotated-array4618/1/)

**Approach 1 (Finding Pivot where rotation has happened):** The primary idea to solve the problem is as follows.

*The idea is to find the pivot point, divide the array into two sub-arrays and perform a binary search.*

*The main idea for finding a pivot is –*

1. *For a sorted (in increasing order) and rotated array, the pivot element is the only element for which the next element to it is smaller than it.*
2. *Using*[*binary search*](https://www.geeksforgeeks.org/binary-search/)*based on the above idea, pivot can be found.*
3. *It can be observed that for a search space of indices in range****[l, r]****where the middle index is****mid****,*
4. *If rotation has happened in the left half, then obviously the element at****l****will be greater than the one at****mid****.*
5. *Otherwise the left half will be sorted but the element at****mid****will be greater than the one at****r****.*
6. *After the pivot is found divide the array into two sub-arrays.*
7. *Now the individual sub-arrays are sorted so the element can be searched using Binary Search.*

Follow the steps mentioned below to implement the idea:

1. Find out the pivot point using binary search. We will set the low pointer as the first array index and high with the last array index.
2. From the high and low we will calculate the mid value.
3. If the value at **mid-1** is greater than the one at **mid**, return that value as the pivot.
4. Else if the value at the **mid+1** is less than **mid**, return mid value as the pivot.
5. Otherwise, if the value at **low** position is greater than **mid** position, consider the left half. Otherwise, consider the right half.
6. Divide the array into two sub-arrays based on the pivot that was found.
7. Now call binary search for one of the two sub-arrays.
8. If theelement is greater than the 0th element then search in the left array
9. Else search in the right array.
10. Iftheelement is found in the selected sub-array then return the index
11. Elsereturn **-1**.

Follow the below illustration for a better understanding

**Illustration:**

*Consider****arr[] = {3, 4, 5, 1, 2}, key = 1***

***Pivot finding:***

***low = 0, high = 4:***

*=>  mid = 2*

*=>  arr[mid] = 5, arr[mid + 1] = 1*

*=> arr[mid] > arr[mid +1],*

*=> Therefore the pivot =****mid = 2***

*Array is divided into two parts****{3, 4, 5}, {1, 2}***

*Now  according to the conditions and the key, we need to find in the part {1, 2}*

***Key Finding:***

*We will apply Binary search on {1, 2}.*

***low = 3 , high = 4.***

*=>  mid = 3*

*=>  arr[mid] = 1 , key = 1, hence arr[mid] = key matches.*

*=>  The required index =****mid = 3***

*So the element is  found at index****3****.*

Below is the implementation of the above approach:

# Python Program to search an element

# in a sorted and pivoted array

# Searches an element key in a pivoted

# sorted array arrp[] of size n

**def** pivotedBinarySearch(arr, n, key):

    pivot **=** findPivot(arr, 0, n**-**1)

    # If we didn't find a pivot,

    # then array is not rotated at all

**if** pivot **== -**1:

**return** binarySearch(arr, 0, n**-**1, key)

    # If we found a pivot, then first

    # compare with pivot and then

    # search in two subarrays around pivot

**if** arr[pivot] **==** key:

**return** pivot

**if** arr[0] <**=** key:

**return** binarySearch(arr, 0, pivot**-**1, key)

**return** binarySearch(arr, pivot **+** 1, n**-**1, key)

# Function to get pivot. For array

# 3, 4, 5, 6, 1, 2 it returns 3

# (index of 6)

**def** findPivot(arr, low, high):

    # base cases

**if** high < low:

**return -**1

**if** high **==** low:

**return** low

    # low + (high - low)/2;

    mid **=** int((low **+** high)**/**2)

**if** mid < high **and** arr[mid] > arr[mid **+** 1]:

**return** mid

**if** mid > low **and** arr[mid] < arr[mid **-** 1]:

**return** (mid**-**1)

**if** arr[low] >**=** arr[mid]:

**return** findPivot(arr, low, mid**-**1)

**return** findPivot(arr, mid **+** 1, high)

# Standard Binary Search function

**def** binarySearch(arr, low, high, key):

**if** high < low:

**return -**1

    # low + (high - low)/2;

    mid **=** int((low **+** high)**/**2)

**if** key **==** arr[mid]:

**return** mid

**if** key > arr[mid]:

**return** binarySearch(arr, (mid **+** 1), high,

                            key)

**return** binarySearch(arr, low, (mid **-** 1), key)

# Driver program to check above functions

# Let us search 3 in below array

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    arr1 **=** [5, 6, 7, 8, 9, 10, 1, 2, 3]

    n **=** len(arr1)

    key **=** 3

    print("Index of the element is : ", \

          pivotedBinarySearch(arr1, n, key))

# This is contributed by Smitha Dinesh Semwal

**Output**

Index of the element is : 8

**Find the Rotation Count in Rotated Sorted array**

Given an array **arr[]** of size **N** having distinct numbers sorted in increasing order and the array has been right rotated (i.e, the last element will be cyclically shifted to the starting position of the array) **k** number of times, the task is to find the value of **k**.

**Examples:**

***Input:****arr[] = {15, 18, 2, 3, 6, 12}*

***Output:****2*

***Explanation:****Initial array must be {2, 3, 6, 12, 15, 18}.*

*We get the given array after rotating the initial array twice.*

***Input:****arr[] = {7, 9, 11, 12, 5}*

***Output:****4*

***Input:****arr[] = {7, 9, 11, 12, 15};*

***Output:****0*

**Approach 1 (Using**[**linear search**](https://www.geeksforgeeks.org/linear-search/)**):** This problem can be solved using linear search.

*If we take a closer look at examples, we can notice that the number of rotations is equal to the index of the minimum element. A simple linear solution is to find the minimum element and returns its index.*

**Illustration:**

*Consider the array****arr[]={15, 18, 2, 3, 6, 12};***

*Initially****minimum = 15****,****min\_index = 0***

***At i = 1:****min = 15, min\_index = 0*

***At i = 2****: min = min(2, 15) = 2, min\_index = 2*

***At i = 3:****min = 2, min\_index = 2*

***At i = 4:****min = 2, min\_index = 2*

***At i = 5:****min = 2, min\_index = 2*

*The array is rotated twice to the right*

Follow the steps mentioned below to implement the idea:

1. Initialize two variables to store the minimum value and the index of that value.
2. Traverse the array from start to the end:
3. Find the minimum value and index where the minimum value is stored.
4. Return the index of the minimum value.

Below is the code implementation of the above idea.

# Python3 program to find number

# of rotations in a sorted and

# rotated array.

# Returns count of rotations for

# an array which is first sorted

# in ascending order, then rotated

**def** countRotations(arr, n):

    # We basically find index

    # of minimum element

    min **=** arr[0]

    min\_index **=** 0

**for** i **in** range(0, n):

**if** (min > arr[i]):

            min **=** arr[i]

            min\_index **=** i

**return** min\_index;

# Driver code

arr **=** [15, 18, 2, 3, 6, 12]

n **=** len(arr)

print(countRotations(arr, n))

# This code is contributed by Smitha Dinesh Semwal

**Output**

2

Given an array **arr[]** of size **N** having distinct numbers sorted in increasing order and the array has been right rotated (i.e, the last element will be cyclically shifted to the starting position of the array) **k** number of times, the task is to find the value of **k**.

**Examples:**

***Input:****arr[] = {15, 18, 2, 3, 6, 12}*

***Output:****2*

***Explanation:****Initial array must be {2, 3, 6, 12, 15, 18}.*

*We get the given array after rotating the initial array twice.*

***Input:****arr[] = {7, 9, 11, 12, 5}*

***Output:****4*

***Input:****arr[] = {7, 9, 11, 12, 15};*

***Output:****0*

**Approach 1 (Using**[**linear search**](https://www.geeksforgeeks.org/linear-search/)**):** This problem can be solved using linear search.

*If we take a closer look at examples, we can notice that the number of rotations is equal to the index of the minimum element. A simple linear solution is to find the minimum element and returns its index.*

**Illustration:**

*Consider the array****arr[]={15, 18, 2, 3, 6, 12};***

*Initially****minimum = 15****,****min\_index = 0***

***At i = 1:****min = 15, min\_index = 0*

***At i = 2****: min = min(2, 15) = 2, min\_index = 2*

***At i = 3:****min = 2, min\_index = 2*

***At i = 4:****min = 2, min\_index = 2*

***At i = 5:****min = 2, min\_index = 2*

*The array is rotated twice to the right*

Follow the steps mentioned below to implement the idea:

1. Initialize two variables to store the minimum value and the index of that value.
2. Traverse the array from start to the end:
3. Find the minimum value and index where the minimum value is stored.
4. Return the index of the minimum value.

Below is the code implementation of the above idea.

# Python3 program to find number

# of rotations in a sorted and

# rotated array.

# Returns count of rotations for

# an array which is first sorted

# in ascending order, then rotated

**def** countRotations(arr, n):

    # We basically find index

    # of minimum element

    min **=** arr[0]

    min\_index **=** 0

**for** i **in** range(0, n):

**if** (min > arr[i]):

            min **=** arr[i]

            min\_index **=** i

**return** min\_index;

# Driver code

arr **=** [15, 18, 2, 3, 6, 12]

n **=** len(arr)

print(countRotations(arr, n))

# This code is contributed by Smitha Dinesh Semwal

**Output**

2

**K-th Largest Sum Contiguous Subarray**

Given an array of integers. Write a program to find the **K-th** largest sum of contiguous subarray within the array of numbers that has both negative and positive numbers.

**Examples:**

***Input:****a[] = {20, -5, -1}, K = 3*

***Output:****14*

***Explanation:****All sum of contiguous subarrays are (20, 15, 14, -5, -6, -1)*

*so the 3rd largest sum is 14.*

***Input:****a[] = {10, -10, 20, -40}, k = 6*

***Output:****-10*

***Explanation:****The 6th largest sum among*

*sum of all contiguous subarrays is -10.*

Recommended Problem

K-th Largest Sum Contiguous Subarray

**Brute force Approach:** Store all the contiguous sums in another array and sort it and print the Kth largest. But in the case of the number of elements being large, the array in which we store the contiguous sums will run out of memory as the number of contiguous subarrays will be large (quadratic order)

**Kth largest sum contiguous subarray using Min-Heap:**

*The key idea**is to store the pre-sum of the array in a sum[] array. One can find the sum of contiguous subarray from index i to j as sum[j] – sum[i-1]. Now generate all possible contiguous subarray sums and push them into the Min-Heap only if the size of Min-Heap is less than K or the current sum is greater than the root of the Min-Heap. In the end, the root of the Min-Heap is the required answer*

Follow the given steps to solve the problem using the above approach:

1. Create a [prefix sum](https://www.geeksforgeeks.org/prefix-sum-array-implementation-applications-competitive-programming/) array of the input array
2. Create a Min-Heap that stores the subarray sum
3. Iterate over the given array using the variable i such that 1 <= i <= N, here i denotes the starting point of the subarray
4. Create a nested loop inside this loop using a variable j such that i <= j <= N, here j denotes the ending point of the subarray
5. Calculate the sum of the current subarray represented by i and j, using the prefix sum array
6. If the size of the Min-Heap is less than K, then push this sum into the heap
7. Otherwise, if the current sum is greater than the root of the Min-Heap, then pop out the root and push the current sum into the Min-Heap
8. Now the root of the Min-Heap denotes the Kth largest sum, Return it

Below is the implementation of the above approach:

# Python program to find the K-th largest sum

# of subarray

**import** heapq

# function to calculate Kth largest element

# in contiguous subarray sum

**def** kthLargestSum(arr, N, K):

    # array to store prefix sums

    sum **=** []

    sum.append(0)

    sum.append(arr[0])

**for** i **in** range(2, N **+** 1):

        sum.append(sum[i **-** 1] **+** arr[i **-** 1])

    # priority\_queue of min heap

    Q **=** []

    heapq.heapify(Q)

    # loop to calculate the contiguous subarray

    # sum position-wise

**for** i **in** range(1, N **+** 1):

        # loop to traverse all positions that

        # form contiguous subarray

**for** j **in** range(i, N **+** 1):

            x **=** sum[j] **-** sum[i **-** 1]

            # if queue has less then k elements,

            # then simply push it

**if** len(Q) < K:

                heapq.heappush(Q, x)

**else**:

                # it the min heap has equal to

                # k elements then just check

                # if the largest kth element is

                # smaller than x then insert

                # else its of no use

**if** Q[0] < x:

                    heapq.heappop(Q)

                    heapq.heappush(Q, x)

    # the top element will be then kth

    # largest element

**return** Q[0]

# Driver's code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    a **=** [10, **-**10, 20, **-**40]

    N **=** len(a)

    K **=** 6

    # Function call

    print(kthLargestSum(a, N, K))

# This code is contributed by Kumar Suman

**Output**

-10

**Find the smallest missing number**

Given a **sorted**array of n distinct integers where each integer is in the range from 0 to m-1 and m > n. Find the smallest number that is missing from the array.

**Examples:**

**Input:** {0, 1, 2, 6, 9}, n = 5, m = 10   
**Output:** 3

**Input:** {4, 5, 10, 11}, n = 4, m = 12   
**Output:** 0

**Input:** {0, 1, 2, 3}, n = 4, m = 5   
**Output:** 4

**Input:** {0, 1, 2, 3, 4, 5, 6, 7, 10}, n = 9, m = 11   
**Output:** 8

Thanks to Ravichandra for suggesting following two methods.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Method 1 (Use**[**Binary Search**](https://www.geeksforgeeks.org/binary-search/)**)**

For i = 0 to m-1, do binary search for i in the array. If i is not present in the array then return i.

Time Complexity: O(m log n)

**Method 2 (**[**Linear Search**](https://www.geeksforgeeks.org/linear-search/)**)**

If arr[0] is not 0, return 0. Otherwise traverse the input array starting from index 0, and for each pair of elements a[i] and a[i+1], find the difference between them. if the difference is greater than 1 then a[i]+1 is the missing number.

Time Complexity: O(n)

**Method 3 (Use Modified Binary Search)**

Thanks to yasein and **Jams** for suggesting this method.

In the standard Binary Search process, the element to be searched is compared with the middle element and on the basis of comparison result, we decide whether to search is over or to go to left half or right half.

In this method, we modify the standard Binary Search algorithm to compare the middle element with its index and make decision on the basis of this comparison.

1. If the first element is not same as its index then return first index
2. Else get the middle index say mid
3. If arr[mid] greater than mid then the required element lies in left half.
4. Else the required element lies in right half.

# Python3 program to find the smallest

# elements missing in a sorted array.

**def** findFirstMissing(array, start, end):

**if** (start > end):

**return** end **+** 1

**if** (start !**=** array[start]):

**return** start;

    mid **=** int((start **+** end) **/** 2)

    # Left half has all elements

    # from 0 to mid

**if** (array[mid] **==** mid):

**return** findFirstMissing(array,

                          mid**+**1, end)

**return** findFirstMissing(array,

                          start, mid)

# driver program to test above function

arr **=** [0, 1, 2, 3, 4, 5, 6, 7, 10]

n **=** len(arr)

**print**("Smallest missing element is",

      findFirstMissing(arr, 0, n**-**1))

# This code is contributed by Smitha Dinesh Semwal

**Output**

Smallest missing element is 8

**Difference Array | Range update query in O(1)**

Consider an array A[] of integers and following two types of queries.

1. update(l, r, x) : Adds x to all values from A[l] to A[r] (both inclusive).
2. printArray() : Prints the current modified array.

**Examples :**

Input : A [] { 10, 5, 20, 40 }  
 update(0, 1, 10)  
 printArray()  
 update(1, 3, 20)  
 update(2, 2, 30)  
 printArray()  
Output : 20 15 20 40  
 20 35 70 60  
Explanation : The query update(0, 1, 10)   
adds 10 to A[0] and A[1]. After update,  
A[] becomes {20, 15, 20, 40}   
Query update(1, 3, 20) adds 20 to A[1],  
A[2] and A[3]. After update, A[] becomes  
{20, 35, 40, 60}.  
Query update(2, 2, 30) adds 30 to A[2].   
After update, A[] becomes {20, 35, 70, 60}.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

A **simple solution** is to do following :

1. update(l, r, x) : Run a loop from l to r and add x to all elements from A[l] to A[r]
2. printArray() : Simply print A[].

Time complexities of both of the above operations is O(n)

An **efficient solution** is to use difference array.

**Difference array** D[i] of a given array A[i] is defined as D[i] = A[i]-A[i-1] (for 0 < i < N ) and D[0] = A[0] considering 0 based indexing. Difference array can be used to perform range update queries “l r x” where l is left index, r is right index and x is value to be added and after all queries you can return original array from it. Where update range operations can be performed in O(1) complexity.

1. update(l, r, x) : Add x to D[l] and subtract it from D[r+1], i.e., we do D[l] += x, D[r+1] -= x
2. printArray() : Do A[0] = D[0] and print it. For rest of the elements, do A[i] = A[i-1] + D[i] and print them.

Time complexity of update here is improved to **O(1)**. Note that printArray() still takes O(n) time.

# Python3 code to demonstrate Difference Array

# Creates a diff array D[] for A[] and returns

# it after filling initial values.

**def** initializeDiffArray( A):

    n **=** len(A)

    # We use one extra space because

    # update(l, r, x) updates D[r+1]

    D **=** [0 **for** i **in** range(0 , n **+** 1)]

    D[0] **=** A[0]; D[n] **=** 0

**for** i **in** range(1, n ):

        D[i] **=** A[i] **-** A[i **-** 1]

**return** D

# Does range update

**def** update(D, l, r, x):

    D[l] **+=** x

    D[r **+** 1] **-=** x

# Prints updated Array

**def** printArray(A, D):

**for** i **in** range(0 , len(A)):

**if** (i **==** 0):

            A[i] **=** D[i]

        # Note that A[0] or D[0] decides

        # values of rest of the elements.

**else**:

            A[i] **=** D[i] **+** A[i **-** 1]

**print**(A[i], end **=** " ")

**print** ("")

# Driver Code

A **=** [ 10, 5, 20, 40 ]

# Create and fill difference Array

D **=** initializeDiffArray(A)

# After below update(l, r, x), the

# elements should become 20, 15, 20, 40

update(D, 0, 1, 10)

printArray(A, D)

# After below updates, the

# array should become 30, 35, 70, 60

update(D, 1, 3, 20)

update(D, 2, 2, 30)

printArray(A, D)

# This code is contributed by Gitanjali.

Output:

20 15 20 40   
20 35 70 60

**Maximum profit by buying and selling a share at most twice**

In daily share trading, a buyer buys shares in the morning and sells them on the same day. If the trader is allowed to make at most 2 transactions in a day, the second transaction can only start after the first one is complete (Buy->sell->Buy->sell). Given stock prices throughout the day, find out the maximum profit that a share trader could have made.

**Examples:**

***Input:****price[] = {10, 22, 5, 75, 65, 80}*

***Output:****87*

*Trader earns 87 as sum of 12, 75*

*Buy at 10, sell at 22,*

*Buy at 5 and sell at 80*

***Input:****price[] = {2, 30, 15, 10, 8, 25, 80}*

***Output:****100*

*Trader earns 100 as sum of 28 and 72*

*Buy at price 2, sell at 30, buy at 8 and sell at 80*

***Input:****price[] = {100, 30, 15, 10, 8, 25, 80};*

***Output:****72*

*Buy at price 8 and sell at 80.*

***Input:****price[] = {90, 80, 70, 60, 50}\*

***Output:****0*

*Not possible to earn.*

Recommended Problem

Buy and Sell a Share at most twice

**Naive approach:**A Simple Solution is to consider every index ‘i’ and do the following

*Max profit with at most two transactions =*

*MAX {max profit with one transaction and subarray price[0..i] +*

*max profit with one transaction and subarray price[i+1..n-1] }*

*i varies from 0 to n-1.*

The maximum possible using one transaction can be calculated using the following O(n) algorithm

[The maximum difference between two elements such that](https://www.geeksforgeeks.org/maximum-difference-between-two-elements/)the [larger element appears after the smaller number](https://www.geeksforgeeks.org/maximum-difference-between-two-elements/)

**Time Complexity:**O(n2).

**Efficient Solution**. The idea is to store the maximum possible profit of every subarray and solve the problem in the following two phases.

**1)** Create a table profit[0..n-1] and initialize all values in it 0.

**2)** Traverse price[] from right to left and update profit[i] such that profit[i] stores maximum profit achievable from one transaction in subarray price[i..n-1]

**3)**Traverse price[] from left to right and update profit[i] such that profit[i] stores maximum profit such that profit[i] contains maximum achievable profit from two transactions in subarray price[0..i].

**4)**Return profit[n-1]

To do step 2, we need to keep track of the maximum price from right to left side, and to do step 3, we need to keep track of the minimum price from left to right. Why we traverse in reverse directions? The idea is to save space, in the third step, we use the same array for both purposes, maximum with 1 transaction and maximum with 2 transactions. After iteration i, the array profit[0..i] contains the maximum profit with 2 transactions, and profit[i+1..n-1] contains profit with two transactions.

Below are the implementations of the above approach.

# Returns maximum profit with

# two transactions on a given

# list of stock prices price[0..n-1]

**def** maxProfit(price, n):

    # Create profit array and initialize it as 0

    profit **=** [0]**\***n

    # Get the maximum profit

    # with only one transaction

    # allowed. After this loop,

    # profit[i] contains maximum

    # profit from price[i..n-1]

    # using at most one trans.

    max\_price **=** price[n**-**1]

**for** i **in** range(n**-**2, 0, **-**1):

**if** price[i] > max\_price:

            max\_price **=** price[i]

        # we can get profit[i] by

        # taking maximum of:

        # a) previous maximum,

        # i.e., profit[i+1]

        # b) profit by buying at

        # price[i] and selling at

        #    max\_price

        profit[i] **=** max(profit[i**+**1], max\_price **-** price[i])

    # Get the maximum profit

    # with two transactions allowed

    # After this loop, profit[n-1]

    # contains the result

    min\_price **=** price[0]

**for** i **in** range(1, n):

**if** price[i] < min\_price:

            min\_price **=** price[i]

        # Maximum profit is maximum of:

        # a) previous maximum,

        # i.e., profit[i-1]

        # b) (Buy, Sell) at

        # (min\_price, A[i]) and add

        #  profit of other trans.

        # stored in profit[i]

        profit[i] **=** max(profit[i**-**1], profit[i]**+**(price[i]**-**min\_price))

    result **=** profit[n**-**1]

**return** result

# Driver function

price **=** [2, 30, 15, 10, 8, 25, 80]

print ("Maximum profit is", maxProfit(price, len(price)))

# This code is contributed by \_\_Devesh Agrawal\_\_

**Output**

Maximum Profit = 100

**Smallest subarray with sum greater than a given value**

Given an array **arr[]**of integers and a number **x**, the task is to find the smallest subarray with a sum greater than the given value.

**Examples:**

arr[] = {1, 4, 45, 6, 0, 19}  
 x = 51  
Output: 3  
Minimum length subarray is {4, 45, 6}

arr[] = {1, 10, 5, 2, 7}  
 x = 9  
Output: 1  
Minimum length subarray is {10}

arr[] = {1, 11, 100, 1, 0, 200, 3, 2, 1, 250}  
 x = 280  
Output: 4  
Minimum length subarray is {100, 1, 0, 200}

arr[] = {1, 2, 4}  
 x = 8  
Output : Not Possible  
Whole array sum is smaller than 8.

Recommended Problem

Smallest subarray with sum greater than x

[Arrays](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Arrays&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Accolite](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Accolite&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

+3 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/smallest-subarray-with-sum-greater-than-x5651/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 83.9K

**Naive approach:**A simple solution is to use two nested loops. The outer loop picks a starting element, the inner loop considers all elements (on right side of current start) as ending element. Whenever sum of elements between current start and end becomes more than the given number, update the result if current length is smaller than the smallest length so far.

Below is the implementation of the above approach:

# Python3 program to find Smallest

# subarray with sum greater

# than a given value

# Returns length of smallest subarray

# with sum greater than x. If there

# is no subarray with given sum,

# then returns n+1

**def** smallestSubWithSum(arr, n, x):

    # Initialize length of smallest

    # subarray as n+1

    min\_len **=** n **+** 1

    # Pick every element as starting point

**for** start **in** range(0,n):

        # Initialize sum starting

        # with current start

        curr\_sum **=** arr[start]

        # If first element itself is greater

**if** (curr\_sum > x):

**return** 1

        # Try different ending points

        # for curremt start

**for** end **in** range(start**+**1,n):

            # add last element to current sum

            curr\_sum **+=** arr[end]

            # If sum becomes more than x

            # and length of this subarray

            # is smaller than current smallest

            # length, update the smallest

            # length (or result)

**if** curr\_sum > x **and** (end **-** start **+** 1) < min\_len:

                min\_len **=** (end **-** start **+** 1)

**return** min\_len;

# Driver program to test above function \*/

arr1 **=** [1, 4, 45, 6, 10, 19]

x **=** 51

n1 **=** len(arr1)

res1 **=** smallestSubWithSum(arr1, n1, x);

**if** res1 **==** n1**+**1:

**print**("Not possible")

**else**:

**print**(res1)

arr2 **=** [1, 10, 5, 2, 7]

n2 **=** len(arr2)

x **=** 9

res2 **=** smallestSubWithSum(arr2, n2, x);

**if** res2 **==** n2**+**1:

    print("Not possible")

**else**:

**print**(res2)

arr3 **=** [1, 11, 100, 1, 0, 200, 3, 2, 1, 250]

n3 **=** len(arr3)

x **=** 280

res3 **=** smallestSubWithSum(arr3, n3, x)

**if** res3 **==** n3**+**1:

    print("Not possible")

**else**:

    print(res3)

# This code is contributed by Smitha Dinesh Semwal

**Output**

3  
1  
4

**nversion count in Array using Merge Sort**

**Inversion Count**for an array indicates – how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0, but if the array is sorted in reverse order, the inversion count is the maximum.

Given an array **a[]**. The task is to find the inversion count of **a[]**. Where two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.

**Examples:**

***Input:****arr[] = {8, 4, 2, 1}*

***Output:****6*

***Explanation:****Given array has six inversions: (8, 4), (4, 2), (8, 2), (8, 1), (4, 1), (2, 1).*

***Input:****arr[] = {1, 20, 6, 4, 5}*

***Output:****5*

***Explanation:****Given array has five inversions: (20, 6), (20, 4), (20, 5), (6, 4), (6, 5).*

Recommended Problem

Count Inversions

**Naive Approach:**

*Traverse through the array, and for every index, find the number of smaller elements on its right side of the array. This can be done using a nested loop. Sum up the counts for all indices in the array and print the sum.*

Follow the below steps to Implement the idea:

1. Traverse through the array from start to end
2. For every element, find the count of elements smaller than the current number up to that index using another loop.
3. Sum up the count of inversion for every index.
4. Print the count of inversions.

Below is the Implementation of the above approach:

# 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

**Output**

Number of inversions are 5

**Sort an array of 0s, 1s and 2s | Dutch National Flag problem**

Given an array **A[]** consisting of only **0s**, **1s,** and **2s**. The task is to write a function that sorts the given array. The functions should put all 0s first, then all 1s and all 2s in last.

This problem is also the same as the famous **“Dutch National Flag problem”**. The problem was proposed by Edsger Dijkstra. The problem is as follows:

*Given N balls of colour red, white or blue arranged in a line in random order. You have to arrange all the balls such that the balls with the same colours are adjacent with the order of the balls, with the order of the colours being red, white and blue (i.e., all red coloured balls come first then the white coloured balls and then the blue coloured balls).*

**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, 0, 1, 1, 1, 1, 1, 2, 2}*

Recommended Problem

Sort an array of 0s, 1s and 2s

**Sort an array of 0s, 1s, and 2s using the Pointer Approach:**

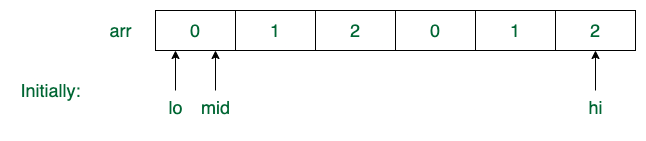
This approach is based on the following idea:

1. *The problem is similar to*[*“Segregate 0s and 1s in an array”*](https://www.geeksforgeeks.org/segregate-0s-and-1s-in-an-array-by-traversing-array-once/)*.*
2. *The problem was posed with three colors, here `0′, `1′ and `2′. The array is divided into four sections:*
3. *arr[1] to arr[low – 1]*
4. *arr[low] to arr[mid – 1]*
5. *arr[mid] to arr[high – 1]*
6. *arr[high] to arr[n]*
7. *If the ith element is 0 then swap the element to the low range.*
8. *Similarly, if the element is 1 then keep it as it is.*
9. *If the element is 2 then swap it with an element in high range.*

**Illustration:**

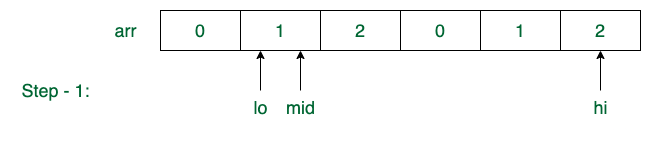
***arr[] = {0, 1, 2, 0, 1, 2}***

***lo****= 0,****mid****= 0,****hi****= 5*



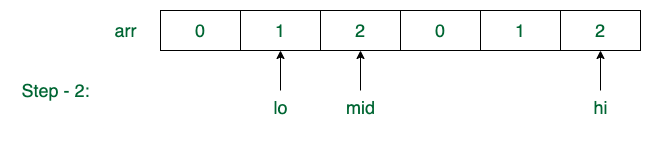
***Step – 1:****arr[mid] == 0*

1. *swap(arr[lo], arr[mid])*
2. *lo = lo + 1 = 1*
3. *mid = mid + 1 = 1*
4. *arr[] = {0, 1, 2, 0, 1, 2}*



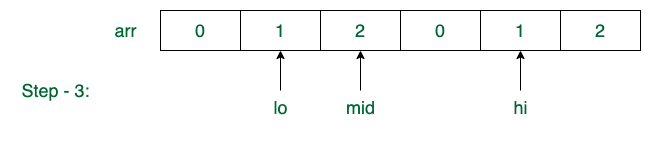
***Step – 2:****arr[mid] == 1*

1. *mid = mid + 1 = 2*
2. *arr[] = {0, 1, 2, 0, 1, 2}*



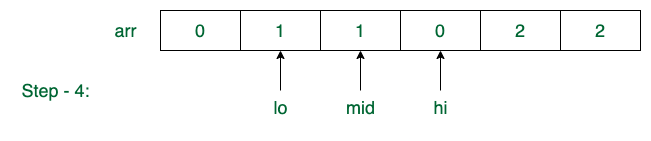
***Step – 3:****arr[mid] == 2*

1. *swap(arr[mid], arr[hi])*
2. *hi = hi – 1 = 4*
3. *arr[] = {0, 1, 2, 0, 1, 2}*



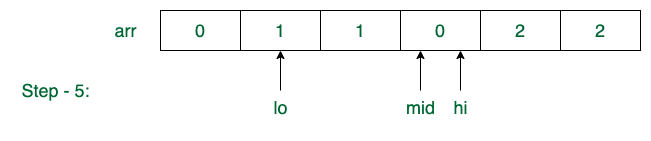
***Step – 4:****arr[mid] == 2*

1. *swap(arr[mid], arr[hi])*
2. *hi = hi – 1 = 3*
3. *arr[] = {0, 1, 1, 0, 2, 2}*



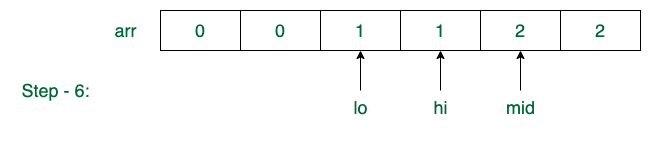
***Step – 5:****arr[mid] == 1*

1. *mid = mid + 1 = 3*
2. *arr[] = {0, 1, 1, 0, 2, 2}*



***Step – 6:****arr[mid] == 0*

1. *swap(arr[lo], arr[mid])*
2. *lo = lo + 1 = 2*
3. *mid = mid + 1 = 4*
4. *arr[] = {0, 0, 1, 1, 2, 2}*



*Hence,****arr[] = {0, 0, 1, 1, 2, 2}***

Follow the steps below to solve the given problem:

1. Keep three indices low = 1, mid = 1, and high = N and there are four ranges, 1 to low (the range containing 0), low to mid (the range containing 1), mid to high (the range containing unknown elements) and high to N (the range containing 2).
2. Traverse the array from start to end and mid is less than high. (Loop counter is i)
3. If the element is 0 then swap the element with the element at index low and update low = low + 1 and mid = mid + 1
4. If the element is 1 then update mid = mid + 1
5. If the element is 2 then swap the element with the element at index high and update high = high – 1 and update i = i – 1. As the swapped element is not processed
6. Print the array.

1. C++
2. C
3. Java
4. Python3
5. C#
6. PHP
7. Javascript

# 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

    # Iterate till all the elements

    # are sorted

**while** mid <**=** hi:

        # If the element is 0

**if** a[mid] **==** 0:

            a[lo], a[mid] **=** a[mid], a[lo]

            lo **=** lo **+** 1

            mid **=** mid **+** 1

        # If the element is 1

**elif** a[mid] **==** 1:

            mid **=** mid **+** 1

        # If the element is 2

**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, end**=**' ')

# Driver Program

arr **=** [0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1]

arr\_size **=** len(arr)

arr **=** sort012(arr, arr\_size)

printArray(arr)

# Contributed by Harshit Agrawal

**Output**

0 0 0 0 0 1 1 1 1 1 2 2

**Merge two sorted arrays with O(1) extra space**

We are given two sorted arrays. 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

**Examples:**

***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}*

Recommended Problem

Merge Without Extra Space

**Note:**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.  Though further optimizations are possible

**Efficient Approach:** To solve the problem follow the below idea:

*The idea is to begin from the last element of ar2[] and search for it in ar1[]. If there is a greater element in ar1[], then we move the last element of ar1[] to ar2[]. To keep ar1[] and ar2[] sorted, we need to place the last element of ar2[] at the correct place in ar1[]. We can use the*[*Insertion Sort*](https://www.geeksforgeeks.org/insertion-sort/)*for this*

Follow the below steps to solve the problem:

1. Iterate through every element of ar2[] starting from the last element
2. Do the following for every element ar2[i]
3. Store last element of ar1[]: last = ar1[m-1]
4. Loop from the second last element of ar1[] while element ar1[j] is greater than ar2[i].
5. ar1[j+1] = ar1[j] Move element one position ahead, then j–
6. If last element of ar1[] is greater than ar2[i], then ar1[j+1] = ar2[i] and ar2[i] = last
7. Print the arrays

**Note:**In the above loop, elements in ar1[] and ar2[] are always kept sorted.

Below is the implementation of the above approach:

# 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** (last > ar2[i]):

            ar1[j**+**1] **=** ar2[i]

            ar2[i] **=** last

# Driver code

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

**Majority Element**

Find the majority element in the array. A ***majority element*** in an array A[] of size n is an element that appears more than n/2 times (and hence there is at most one such element).

**Examples :**

***Input :****{3, 3, 4, 2, 4, 4, 2, 4, 4}*

***Output :****4*

***Explanation:****The frequency of 4 is 5 which is greater than the half of the size of the array size.*

***Input :****{3, 3, 4, 2, 4, 4, 2, 4}*

***Output :****No Majority Element*

***Explanation:****There is no element whose frequency is greater than the half of the size of the array size.*

Recommended Problem

Majority Element

**Naive Approach:**

*The basic solution is to have two loops and keep track of the****maximum****count for all different elements. If the maximum count becomes greater than****n/2****then break the loops and return the element having the maximum count. If the maximum count doesn’t become more than n/2 then the majority element****doesn’t****exist.*

**Illustration:**

***arr[] = {3, 4, 3, 2, 4, 4, 4, 4}, n = 8***

*For i = 0:*

1. *count = 0*
2. *Loop over the array, whenever an element is equal to arr[i] (is****3****), increment count*
3. *count of arr[i] is****2,****which is less than****n/2,****hence it can’t be****majority element.***

*For i = 1:*

1. *count = 0*
2. *Loop over the array, whenever an element is equal to arr[i] (is****4****), increment count*
3. *count of arr[i] is****5,****which is****greater****than****n/2****(i.e 4)****,****hence it will be****majority element.***

*Hence,****4****is the****majority element****.*

Follow the steps below to solve the given problem:

1. Create a variable to store the max count, *count = 0*
2. Traverse through the array from start to end.
3. For every element in the array run another loop to find the count of similar elements in the given array.
4. If the count is greater than the max count update the max count and store the index in another variable.
5. If the maximum count is greater than half the size of the array, print the element. Else print there is no majority element.

Below is the implementation of the above idea:

# Python3 program to find Majority

# element in an array

# Function to find Majority

# element in an array

**def** findMajority(arr, n):

    maxCount **=** 0

    index **= -**1  # sentinels

**for** i **in** range(n):

        count **=** 0

**for** j **in** range(n):

**if**(arr[i] **==** arr[j]):

                count **+=** 1

        # update maxCount if count of

        # current element is greater

**if**(count > maxCount):

            maxCount **=** count

            index **=** i

    # if maxCount is greater than n/2

    # return the corresponding element

**if** (maxCount > n**//**2):

        print(arr[index])

**else**:

        print("No Majority Element")

# Driver code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    arr **=** [1, 1, 2, 1, 3, 5, 1]

    n **=** len(arr)

    # Function calling

    findMajority(arr, n)

# This code is contributed

# by ChitraNayal

**Output**

1

**Time Complexity:** O(n\*n), A nested loop is needed where both the loops traverse the array from start to end.

**Auxiliary Space:** O(1), No extra space is required.

**Majority Element using**[Binary Search Tree](https://www.geeksforgeeks.org/binary-search-tree-data-structure/)

*Insert elements in****BST****one by one and if an element is already present then increment the count of the node. At any stage, if the count of a node becomes more than****n/2****then return.*

**Illustration:**

Follow the steps below to solve the given problem:

1. Create a binary search tree, if the same element is entered in the binary search tree the frequency of the node is increased.
2. traverse the array and insert the element in the binary search tree.
3. If the maximum frequency of any node is greater than half the size of the array, then perform an inorder traversal and find the node with a frequency greater than half
4. Else print No majority Element.

Below is the implementation of the above idea:

1. C++14
2. Java
3. Python3
4. C#
5. Javascript

# Python3 program to demonstrate insert operation in binary

# search tree.

# class for creating node

**class** Node():

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** None

        self.right **=** None

        self.count **=** 1  # count of number of times data is inserted in tree

# class for binary search tree

# it initialises tree with None root

# insert function inserts node as per BST rule

# and also checks for majority element

# if no majority element is found yet, it returns None

**class** BST():

**def** \_\_init\_\_(self):

        self.root **=** None

**def** insert(self, data, n):

        out **=** None

**if** (self.root **==** None):

            self.root **=** Node(data)

**else**:

            out **=** self.insertNode(self.root, data, n)

**return** out

**def** insertNode(self, currentNode, data, n):

**if** (currentNode.data **==** data):

            currentNode.count **+=** 1

**if** (currentNode.count > n**//**2):

**return** currentNode.data

**else**:

**return** None

**elif** (currentNode.data < data):

**if** (currentNode.right):

                self.insertNode(currentNode.right, data, n)

**else**:

                currentNode.right **=** Node(data)

**elif** (currentNode.data > data):

**if** (currentNode.left):

                self.insertNode(currentNode.left, data, n)

**else**:

                currentNode.left **=** Node(data)

# Driver code

# declaring an array

arr **=** [3, 2, 3]

n **=** len(arr)

# declaring None tree

tree **=** BST()

flag **=** 0

**for** i **in** range(n):

    out **=** tree.insert(arr[i], n)

**if** (out !**=** None):

        print(arr[i])

        flag **=** 1

**break**

**if** (flag **==** 0):

**print**("No Majority Element")

**Output**

3

**Time Complexity:** If a [Binary Search Tree](https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/) is used then time complexity will be O(n²). If a [self-balancing-binary-search](http://en.wikipedia.org/wiki/Self-balancing_binary_search_tree) tree is used then it will be O(nlogn)

**Auxiliary Space:**O(n), As extra space is needed to store the array in the tree.

**Majority Element Using**[Moore’s Voting Algorithm](https://www.geeksforgeeks.org/boyer-moore-majority-voting-algorithm/)**:**

*This is a two-step process:*

1. *The first step gives the element that may be the majority element in the array. If there is a majority element in an array, then this step will definitely return majority element, otherwise, it will return candidate for majority element.*
2. *Check if the element obtained from the above step is the majority element. This step is necessary as there might be no majority element.*

**Illustration:**

***arr[] = {3, 4, 3, 2, 4, 4, 4, 4}, n = 8***

*maj\_index = 0, count = 1*

*At****i = 1****: arr[maj\_index] != arr[i]*

1. *count = count – 1 = 1 – 1 = 0*
2. *now count == 0 then:*
3. *maj\_index = i = 1*
4. *count = count + 1 = 0 + 1 = 1*

*At****i = 2****: arr[maj\_index] != arr[i]*

1. *count = count – 1 = 1 – 1 = 0*
2. *now count == 0 then:*
3. *maj\_index = i = 2*
4. *count = count + 1 = 0 + 1 = 1*

*At****i = 3****: arr[maj\_index] != arr[i]*

1. *count = count – 1 = 1 – 1 = 0*
2. *now count == 0 then:*
3. *maj\_index = i = 3*
4. *count = count + 1 = 0 + 1 = 1*

*At****i = 4****: arr[maj\_index] != arr[i]*

1. *count = count – 1 = 1 – 1 = 0*
2. *now count == 0 then:*
3. *maj\_index = i = 4*
4. *count = count + 1 = 0 + 1 = 1*

*At****i = 5****: arr[maj\_index] == arr[i]*

1. *count = count + 1 = 1 + 1 = 2*

*At****i = 6****: arr[maj\_index] == arr[i]*

1. *count = count + 1 = 2 + 1 = 3*

*At****i = 7****: arr[maj\_index] == arr[i]*

1. *count = count + 1 = 3 + 1 = 4*

*Therefore, the****arr[maj\_index]****may be the possible candidate for majority element.*

*Now, Again traverse the array and check whether****arr[maj\_index]****is the majority element or not.*

***arr[maj\_index] is 4***

***4****occurs****5 times****in the array therefore 4 is our****majority element.***

Follow the steps below to solve the given problem:

1. Loop through each element and maintains a count of the majority element, and a majority index, *maj\_index*
2. If the next element is the same then increment the count if the next element is not the same then decrement the count.
3. if the count reaches 0 then change the maj\_index to the current element and set the count again to 1.
4. Now again traverse through the array and find the count of the majority element found.
5. If the count is greater than half the size of the array, print the element
6. Else print that there is no majority element

Below is the implementation of the above idea:

# Program for finding out majority element in an array

# Function to find the candidate for Majority

**def** findCandidate(A):

    maj\_index **=** 0

    count **=** 1

**for** i **in** range(len(A)):

**if** A[maj\_index] **==** A[i]:

            count **+=** 1

**else**:

            count **-=** 1

**if** count **==** 0:

            maj\_index **=** i

            count **=** 1

**return** A[maj\_index]

# Function to check if the candidate occurs more than n/2 times

**def** isMajority(A, cand):

    count **=** 0

**for** i **in** range(len(A)):

**if** A[i] **==** cand:

            count **+=** 1

**if** count > len(A)**/**2:

**return** True

**else**:

**return** False

# Function to print Majority Element

**def** printMajority(A):

    # Find the candidate for Majority

    cand **=** findCandidate(A)

    # Print the candidate if it is Majority

**if** isMajority(A, cand) **==** True:

**print**(cand)

**else**:

        print("No Majority Element")

# Driver code

A **=** [1, 3, 3, 1, 2]

# Function call

printMajority(A)

**Output**

No Majority Element

**Two Pointers Technique**

Two pointers is really an easy and effective technique that is typically used for searching pairs in a sorted array.

Given a sorted array A (sorted in ascending order), having N integers, find if there exists any pair of elements (A[i], A[j]) such that their sum is equal to X.

**Illustration :**

A[] = {10, 20, 35, 50, 75, 80}  
X = =70  
i = 0  
j = 5

A[i] + A[j] = 10 + 80 = 90  
Since A[i] + A[j] > X, j--  
i = 0  
j = 4

A[i] + A[j] = 10 + 75 = 85  
Since A[i] + A[j] > X, j--  
i = 0  
j = 3

A[i] + A[j] = 10 + 50 = 60  
Since A[i] + A[j] < X, i++  
i = 1  
j = 3  
m  
A[i] + A[j] = 20 + 50 = 70  
Thus this signifies that Pair is Found.

Let us do discuss the [working of two pointer algorithm](https://www.geeksforgeeks.org/two-pointers-technique/) in brief which is as follows. The algorithm basically uses the fact that the input array is sorted. We start the sum of extreme values (smallest and largest) and conditionally move both pointers. We move left pointer ‘i’ when the sum of A[i] and A[j] is less than X. We do not miss any pair because the sum is already smaller than X. Same logic applies for right pointer j.

**Methods:**

Here we will be proposing a two-pointer algorithm by starting off with the naïve approach only in order to showcase the execution of operations going on in both methods and secondary to justify how two-pointer algorithm optimizes code via time complexities across all dynamic programming languages such as C+, Java, Python, and even JavaScript

1. Naïve Approach using loops
2. Optimal approach using two pointer algorithm

**Implementation:**

**Method 1:**Naïve Approach

**Examples**

# Python Program Illustrating Naive Approach to

# Find if There is a Pair in A[0..N-1] with Given Sum

# Method

**def** isPairSum(A, N, X):

**for** i **in** range(N):

**for** j **in** range(N):

            # as equal i and j means same element

**if**(i **==** j):

**continue**

            # pair exists

**if** (A[i] **+** A[j] **==** X):

**return** True

            # as the array is sorted

**if** (A[i] **+** A[j] > X):

**break**

    # No pair found with given sum

**return** 0

# Driver code

arr **=** [2, 3, 5, 8, 9, 10, 11]

val **=** 17

print(isPairSum(arr, len(arr), val))

# This code is contributed by maheshwaripiyush9

**Output**

1

**Find a peak element which is not smaller than its neighbours**

Given an array **arr[]**of integers. Find a peak element i.e. an element that is **not smaller**than its neighbors.

**Note:**For corner elements, we need to consider only one neighbor.

**Example:**

***Input:****array[]= {5, 10, 20, 15}*

***Output:****20*

***Explanation:****The element 20 has neighbors 10 and 15, both of them are less than 20.*

***Input:****array[] = {10, 20, 15, 2, 23, 90, 67}*

***Output:****20 or 90*

***Explanation:****The element 20 has neighbors 10 and 15, both of them are less than 20, similarly 90 has neighbors 23 and 67.*

The following corner cases give a better idea about the problem.

1. If the input array is sorted in a strictly increasing order, the last element is always a peak element. For example, 50 is peak element in {10, 20, 30, 40, 50}.
2. If the input array is sorted in a strictly decreasing order, the first element is always a peak element. 100 is the peak element in {100, 80, 60, 50, 20}.
3. If all elements of the input array are the same, every element is a peak element.

It is clear from the above examples that there is always a peak element in the input array.

Recommended Problem

Bitonic Point

**Naive Approach:** Below is the idea to solve the problem

*The array can be traversed and the element whose neighbors are less than that element can be returned.*

Follow the below steps to Implement the idea:

* If the first element is greater than the second or the last element is greater than the second last, print the respective element and terminate the program.
* Else traverse the array from the second index to the second last index i.e. **1**to **N – 1**
* If for an element array[i] is greater than both its neighbors, i.e.,

and

, then print that element and terminate.

Below is the implementation of above idea.

# A Python3 program to find a peak element

# Find the peak element in the array

**def** findPeak(arr, n) :

    # first or last element is peak element

**if** (n **==** 1) :

**return** 0

**if** (arr[0] >**=** arr[1]) :

**return** 0

**if** (arr[n **-** 1] >**=** arr[n **-** 2]) :

**return** n **-** 1

    # check for every other element

**for** i **in** range(1, n **-** 1) :

        # check if the neighbors are smaller

**if** (arr[i] >**=** arr[i **-** 1] **and** arr[i] >**=** arr[i **+** 1]) :

**return** i

# Driver code.

arr **=** [ 1, 3, 20, 4, 1, 0 ]

n **=** len(arr)

print("Index of a peak point is", findPeak(arr, n))

# This code is contributed by divyeshrabadiya07

**Output**

Index of a peak point is 2

**Find a triplet that sum to a given value**

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.

**Examples:**

***Input:****array = {12, 3, 4, 1, 6, 9}, sum = 24;*

***Output:****12, 3, 9*

***Explanation:****There is a triplet (12, 3 and 9) present*

*in the array whose sum is 24.*

***Input:****array = {1, 2, 3, 4, 5}, sum = 9*

***Output:****5, 3, 1*

***Explanation:****There is a triplet (5, 3 and 1) present*

*in the array whose sum is 9.*

**Method 1:** This is the naive approach towards solving the above problem.

* **Approach:** 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.
* **Algorithm:**

1. Given an array of length *n* and a sum *s*
2. Create three nested loop first loop runs from start to end (loop counter i), second loop runs from i+1 to end (loop counter j) and third loop runs from j+1 to end (loop counter k)
3. The counter of these loops represents the index of 3 elements of the triplets.
4. Find the sum of ith, jth and kth element. If the sum is equal to given sum. Print the triplet and break.
5. If there is no triplet, then print that no triplet exist.

* **Implementation:**

# 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

**Output**

Triplet is 4, 10, 8

**Minimum increment by k operations to make all elements equal**

You are given an array of n-elements, you have to find the number of operations needed to make all elements of array equal. Where a single operation can increment an element by k. If it is not possible to make all elements equal print -1.

**Example :**

**Input :** arr[] = {4, 7, 19, 16}, k = 3  
**Output :** 10

**Input :** arr[] = {4, 4, 4, 4}, k = 3  
**Output :** 0

**Input :** arr[] = {4, 2, 6, 8}, k = 3  
**Output :** -1

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

To solve this question we require to check whether all elements can became equal or not and that too only by incrementing k from elements value. For this we have to check that the difference of any two elements should always be divisible by k. If it is so, then all elements can become equal otherwise they can not became equal in any case by incrementing k from them. Also, the number of operations required can be calculated by finding value of (max – Ai)/k for all elements. where max is maximum element of array.

**Algorithm :**

// iterate for all elements  
for (int i=0; i<n; i++)  
{  
 // check if element can make equal to max  
 // or not if not then return -1  
 if ((max - arr[i]) % k != 0 )  
 return -1;

// else update res for required operations  
 else  
 res += (max - arr[i]) / k ;  
}

return res;

**Implementation:**

# Python3 Program to make all array equal

# function for calculating min operations

**def** minOps(arr, n, k):

    # max elements of array

    max1 **=** max(arr)

    res **=** 0

    # iterate for all elements

**for** i **in** range(0, n):

        # check if element can make equal to

        # max or not if not then return -1

**if** ((max1 **-** arr[i]) **%** k !**=** 0):

**return -**1

        # else update res for

        # required operations

**else**:

            res **+=** (max1 **-** arr[i]) **/** k

    # return result

**return** int(res)

# driver program

arr **=** [21, 33, 9, 45, 63]

n **=** len(arr)

k **=** 6

print(minOps(arr, n, k))

# This code is contributed by

# Smitha Dinesh Semwal

**Output**

24

**Equilibrium index of an array**

Given a sequence **arr[]** of size **n**, Write a function ***int equilibrium(int[] arr, int n)*** that returns an equilibrium index (if any) or -1 if no equilibrium index exists.

*The****equilibrium index of an array****is an index such that the sum of elements at lower indexes is equal to the sum of elements at higher indexes.*

**Examples:**

***Input****: A[] = {-7, 1, 5, 2, -4, 3, 0}*

***Output****: 3*

*3 is an equilibrium index, because:*

*A[0] + A[1] + A[2] = A[4] + A[5] + A[6]*

***Input****: A[] = {1, 2, 3}*

***Output****: -1*

Recommended Problem

Equilibrium Point

**Naive approach:** To solve the problem follow the below idea:

*Use two loops. The Outer loop iterates through all the element and inner loop finds out whether the current index picked by the outer loop is equilibrium index or not*

Below is the implementation of the above approach:

# Python3 program to find equilibrium

# index of an array

# function to find the equilibrium index

**def** equilibrium(arr):

    leftsum **=** 0

    rightsum **=** 0

    n **=** len(arr)

    # Check for indexes one by one

    # until an equilibrium index is found

**for** i **in** range(n):

        leftsum **=** 0

        rightsum **=** 0

        # get left sum

**for** j **in** range(i):

            leftsum **+=** arr[j]

        # get right sum

**for** j **in** range(i **+** 1, n):

            rightsum **+=** arr[j]

        # if leftsum and rightsum are same,

        # then we are done

**if** leftsum **==** rightsum:

**return** i

    # return -1 if no equilibrium index is found

**return -**1

# Driver code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    arr **=** [**-**7, 1, 5, 2, **-**4, 3, 0]

    # Function call

    print(equilibrium(arr))

# This code is contributed by Abhishek Sharama

**Output**

3

***Hard Questions:***

**Find K most occurring elements in the given Array**

Given an array of **N** numbers and a positive integer **K**. The problem is to find **K** numbers with the most occurrences, i.e., the top **K** numbers having the maximum frequency. If two numbers have the same frequency then the number with a larger value should be given preference. The numbers should be displayed in decreasing order of their frequencies. It is assumed that the array consists of at least K numbers.

**Examples:**

***Input:****arr[] = {3, 1, 4, 4, 5, 2, 6, 1}, K = 2*

***Output:****4 1*

***Explanation:***

*Frequency of 4 = 2, Frequency of 1 = 2*

*These two have the maximum frequency and 4 is larger than 1.*

***Input:****arr[] = {7, 10, 11, 5, 2, 5, 5, 7, 11, 8, 9}, K = 4*

***Output:****5 11 7 10*

***Explanation:***

*Frequency of 5 = 3, Frequency of 11 = 2, Frequency of 7 = 2, Frequency of 10 = 1*

*These four have the maximum frequency and 5 is largest among rest.*

Recommended Problem

Top K Frequent Elements in Array - |

**Find K most occurring elements in the given Array using Map**

To solve the problem using this approach follow the below idea:

*create a Map to store the element-frequency pair. Map is used to perform insertion and updation in constant time. Then sort the element-frequency pair in decreasing order of frequency. This gives the information about each element and the number of times they are present in the array. To get K elements of the array, print the first K elements of the sorted array.*

Follow the given steps to solve the problem:

* Create a map mp, to store key-value pair, i.e. element-frequency pair.
* Traverse the array from start to end.
* For every element in the array update mp*[array[i]]++*
* Store the element-frequency pair in a vector and sort the vector in decreasing order of frequency.
* Print the first k elements of the sorted array.

Below is the Implementation of the above approach:

# Python3 implementation to find k numbers

# with most occurrences in the given array

# Function to print the k numbers with

# most occurrences

**def** pr\_N\_mostFrequentNumber(arr, N, K):

    mp **=** {}

**for** i **in** range(N):

**if** arr[i] **in** mp:

            mp[arr[i]] **+=** 1

**else**:

            mp[arr[i]] **=** 1

    a **=** [0] **\*** (len(mp))

    j **=** 0

**for** i **in** mp:

        a[j] **=** [i, mp[i]]

        j **+=** 1

    a **=** sorted(a, key**=lambda** x: x[0],

               reverse**=**True)

    a **=** sorted(a, key**=lambda** x: x[1],

               reverse**=**True)

    # Display the top k numbers

    print(K, "numbers with most occurrences are:")

**for** i **in** range(K):

**print**(a[i][0], end**=**" ")

# Driver code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    arr **=** [3, 1, 4, 4, 5, 2, 6, 1]

    N **=** 8

    K **=** 2

    # Function call

    pr\_N\_mostFrequentNumber(arr, N, K)

# This code is contributed by

# Shubham Singh(SHUBHAMSINGH10)

**Output**

2 numbers with most occurrences are:  
4 1

**MO’s Algorithm (Query Square Root Decomposition) | Set 1 (Introduction)**

Let us consider the following problem to understand MO’s Algorithm.

We are given an array and a set of query ranges, we are required to find the sum of every query range.

Example:

Input: arr[] = {1, 1, 2, 1, 3, 4, 5, 2, 8};  
 query[] = [0, 4], [1, 3] [2, 4]  
Output: Sum of arr[] elements in range [0, 4] is 8  
 Sum of arr[] elements in range [1, 3] is 4   
 Sum of arr[] elements in range [2, 4] is 6

Recommended Problem

Interesting Queries

A **Naive Solution** is to run a loop from L to R and calculate the sum of elements in given range for every query [L, R]

# Python program to compute sum of ranges for different range queries.

# Function that accepts array and list of queries and print sum of each query

**def** printQuerySum(arr,Q):

**for** q **in** Q: # Traverse through each query

        L,R **=** q # Extract left and right indices

        s **=** 0

**for** i **in** range(L,R**+**1): # Compute sum of current query range

            s **+=** arr[i]

**print**("Sum of",q,"is",s) # Print sum of current query range

# Driver script

arr **=** [1, 1, 2, 1, 3, 4, 5, 2, 8]

Q **=** [[0, 4], [1, 3], [2, 4]]

printQuerySum(arr,Q)

#This code is contributed by Shivam Singh

**Output:**

Sum of [0, 4] is 8  
Sum of [1, 3] is 4  
Sum of [2, 4] is 6

The time complexity of above solution is O(mn).

The idea of **MO’s algorithm** is to pre-process all queries so that result of one query can be used in next query. Below are steps.

Let **a[0…n-1]** be input array and **q[0..m-1]** be array of queries.

1. Sort all queries in a way that queries with L values from **0** to **√n – 1** are put together, then all queries from **√n** to **2\*√n – 1**, and so on. All queries within a block are sorted in increasing order of R values.
2. Process all queries one by one in a way that every query uses sum computed in the previous query.

* Let ‘sum’ be sum of previous query.
* Remove extra elements of previous query. For example if previous query is [0, 8] and current query is [3, 9], then we subtract a[0],a[1] and a[2] from sum
* Add new elements of current query. In the same example as above, we add a[9] to sum.

The great thing about this algorithm is, in step 2, index variable for R change at most **O(n \* √n)** times throughout the run and same for L changes its value at most **O(m \* √n)** times (See below, after the code, for details). All these bounds are possible only because the queries are sorted first in blocks of **√n** size.

The preprocessing part takes O(m Log m) time.

Processing all queries takes **O(n \* √n)** + **O(m \* √n)** = **O((m+n) \* √n)**time.

Below is the implementation of the above idea.

# Python program to compute sum of ranges for different range queries

**import** math

# Function that accepts array and list of queries and print sum of each query

**def** queryResults(arr,Q):

    #Q.sort(): # Sort by L

    #sort all queries so that all queries in the increasing order of R values .

    Q.sort(key**=lambda** x: x[1])

    # Initialize current L, current R and current sum

    currL,currR,currSum **=** 0,0,0

    # Traverse through all queries

**for** i **in** range(len(Q)):

        L,R **=** Q[i] # L and R values of current range

        # Remove extra elements from previous range

        # if previous range is [0, 3] and current

        # range is [2, 5], then a[0] and a[1] are subtracted

**while** currL<L:

            currSum**-=**arr[currL]

            currL**+=**1

        # Add elements of current range

**while** currL>L:

            currSum**+=**arr[currL**-**1]

            currL**-=**1

**while** currR<**=**R:

            currSum**+=**arr[currR]

            currR**+=**1

        # Remove elements of previous range

        # when previous range is [0, 10] and current range

        # is [3, 8], then a[9] and a[10] are subtracted

**while** currR>R**+**1:

            currSum**-=**arr[currR**-**1]

            currR**-=**1

        # Print the sum of current range

**print**("Sum of",Q[i],"is",currSum)

arr **=** [1, 1, 2, 1, 3, 4, 5, 2, 8]

Q **=** [[0, 4], [1, 3], [2, 4]]

queryResults(arr,Q)

#This code is contributed by Shivam Singh

**Output:**

Sum of [1, 3] is 4  
Sum of [0, 4] is 8  
Sum of [2, 4] is 6

**Square Root (Sqrt) Decomposition Algorithm**

Sqrt (or Square Root) Decomposition Technique is one of the most [common query optimization technique used by competitive programmers](https://www.geeksforgeeks.org/range-minimum-query-for-static-array/). This technique helps us to reduce Time Complexity by a factor of **sqrt(n)**.

*The key concept of this technique is to decompose given array into small chunks specifically of size sqrt(n).*

Let’s say we have an array of n elements and we decompose this array into small chunks of size sqrt(n). We will be having exactly sqrt(n) such chunks provided that n is a perfect square. Therefore, now our array on n elements is decomposed into sqrt(n) blocks, where each block contains sqrt(n) elements (assuming size of array is perfect square).

Let’s consider these chunks or blocks as an individual array each of which contains sqrt(n) elements and you have computed your desired answer(according to your problem) individually for all the chunks. Now, you need to answer certain queries asking you the answer for the elements in range l to r(l and r are starting and ending indices of the array) in the original n sized array.

The **naive approach** is simply to iterate over each element in range l to r and calculate its corresponding answer. Therefore, the Time Complexity per query will be O(n).

**Sqrt Decomposition Trick :** As we have already precomputed the answer for all individual chunks and now we need to answer the queries in range l to r. Now we can simply combine the answers of the chunks that lie in between the range l to r in the original array. So, if we see carefully here we are jumping sqrt(n) steps at a time instead of jumping 1 step at a time as done in naive approach. Let’s just analyze its Time Complexity and implementation considering the below problem :-

**Problem :**  
Given an array of n elements. We need to answer q   
queries telling the sum of elements in range l to   
r in the array. Also the array is not static i.e   
the values are changed via some point update query.

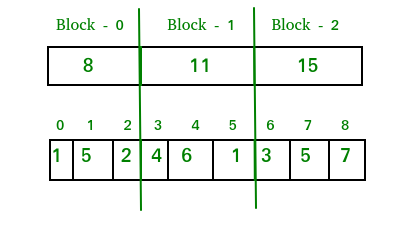
Range Sum Queries are of form : **Q l r** ,   
where l is the starting index r is the ending   
index

Point update Query is of form : **U idx val**,   
where idx is the index to update val is the   
updated value

Let us consider that we have an array of 9 elements.

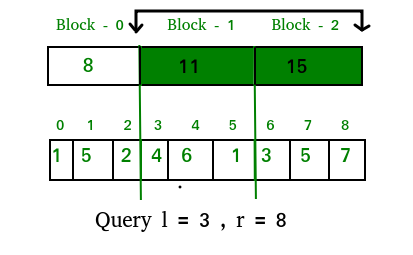
**A[] = {1, 5, 2, 4, 6, 1, 3, 5, 7}**

Let’s decompose this array into sqrt(9) blocks, where each block will contain the sum of elements lying in it. Therefore now our decomposed array would look like this :



Till now we have constructed the decomposed array of sqrt(9) blocks and now we need to print the sum of elements in a given range. So first let’s see two basic types of overlap that a range query can have on our array :-

**Range Query of type 1 (Given Range is on Block Boundaries) :**

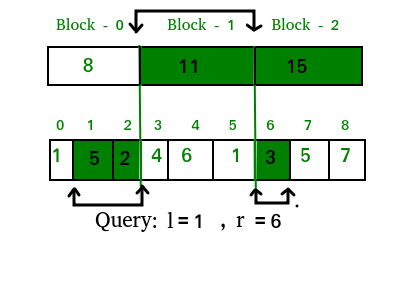


In this type the query, the range may totally cover the continuous sqrt blocks. So we can easily answer the sum of values in this range as the sum of completely overlapped blocks.

*So answer for above query in the described image will be :* ans = 11 + 15 = 26

**Time Complexity:** In the worst case our range can be 0 to n-1(where n is the size of array and assuming n to be a perfect square). In this case all the blocks are completely overlapped by our query range. Therefore,to answer this query we need to iterate over all the decomposed blocks for the array and we know that the number of blocks = sqrt(n). Hence, the complexity for this type of query will be **O(sqrt(n))** in worst case.

**Range Query of type 2 (Given Range is NOT on boundaries)**



We can deal these type of queries by summing the data from the completely overlapped decomposed blocks lying in the query range and then summing elements one by one from the original array whose corresponding block is not completely overlapped by the query range.

*So answer for above query in the described image will be :* ans = 5 + 2 + 11 + 3 = 21

**Time Complexity:** Let’s consider a query [l = 1 and r = n-2] (n is the size of the array and has a 0 based indexing). Therefore, for this query exactly ( sqrt(n) – 2 ) blocks will be completely overlapped where as the first and last block will be partially overlapped with just one element left outside the overlapping range. So, the completely overlapped blocks can be summed up in ( sqrt(n) – 2 ) ~ sqrt(n) iterations, whereas first and last block are needed to be traversed one by one separately. But as we know that the number of elements in each block is at max sqrt(n), to sum up last block individually we need to make,

(sqrt(n)-1) ~ sqrt(n) iterations and same for the last block.

So, the overall Complexity = O(sqrt(n)) + O(sqrt(n)) + O(sqrt(n)) = O(3\*sqrt(N)) = **O(sqrt(n))**

**Update Queries(Point update) :**

In this query we simply find the block in which the given index lies, then subtract its previous value and add the new updated value as per the point update query.

**Time Complexity :** O(1)

**Implementation :**

The implementation of the above trick is given below

# Python 3 program to demonstrate working of Square Root

# Decomposition.

**from** math **import** sqrt

MAXN **=** 10000

SQRSIZE **=** 100

arr **=** [0]**\***(MAXN)         # original array

block **=** [0]**\***(SQRSIZE)     # decomposed array

blk\_sz **=** 0                 # block size

# Time Complexity : O(1)

**def** update(idx, val):

    blockNumber **=** idx **//** blk\_sz

    block[blockNumber] **+=** val **-** arr[idx]

    arr[idx] **=** val

# Time Complexity : O(sqrt(n))

**def** query(l, r):

    sum **=** 0

**while** (l < r **and** l **%** blk\_sz !**=** 0 **and** l !**=** 0):

        # traversing first block in range

        sum **+=** arr[l]

        l **+=** 1

**while** (l **+** blk\_sz **-** 1 <**=** r):

        # traversing completely overlapped blocks in range

        sum **+=** block[l**//**blk\_sz]

        l **+=** blk\_sz

**while** (l <**=** r):

        # traversing last block in range

        sum **+=** arr[l]

        l **+=** 1

**return** sum

# Fills values in input[]

**def** preprocess(input, n):

    # initiating block pointer

    blk\_idx **= -**1

    # calculating size of block

**global** blk\_sz

    blk\_sz **=** int(sqrt(n))

    # building the decomposed array

**for** i **in** range(n):

        arr[i] **=** input[i];

**if** (i **%** blk\_sz **==** 0):

            # entering next block

            # incrementing block pointer

            blk\_idx **+=** 1;

        block[blk\_idx] **+=** arr[i]

# Driver code

# We have used separate array for input because

# the purpose of this code is to explain SQRT

# decomposition in competitive programming where

# we have multiple inputs.

input**=** [1, 5, 2, 4, 6, 1, 3, 5, 7, 10]

n **=** len(input)

preprocess(input, n)

**print**("query(3,8) : ",query(3, 8))

**print**("query(1,6) : ",query(1, 6))

update(8, 0)

print("query(8,8) : ",query(8, 8))

# This code is contributed by Sanjit\_Prasad

**Output**

query(3,8) : 26  
query(1,6) : 21  
query(8,8) : 0

**Sparse Table**

We have briefly discussed sparse table in [Range Minimum Query (Square Root Decomposition and Sparse Table)](https://www.geeksforgeeks.org/range-minimum-query-for-static-array/)

Sparse table concept is used for fast queries on a set of static data (elements do not change). It does preprocessing so that the queries can be answered efficiently.

**Example Problem 1 : Range Minimum Query**

We have an array arr[0 . . . n-1]. We need to efficiently find the minimum value from index L (query start) to R (query end) where 0 <= **L** <= **R** <= **n-1**. Consider a situation when there are many range queries.

**Example:**

Input: arr[] = {7, 2, 3, 0, 5, 10, 3, 12, 18};  
 query[] = [0, 4], [4, 7], [7, 8]

Output: Minimum of [0, 4] is 0  
 Minimum of [4, 7] is 3  
 Minimum of [7, 8] is 12

The idea is to precompute minimum of all subarrays of size **2j**where j varies from 0 to **Log n**. We make a table lookup[i][j] such that lookup[i][j] contains minimum of range starting from i and of size 2j. For example lookup[0][3] contains minimum of range [0, 7] (starting with 0 and of size 23)

**How to fill this lookup or sparse table?**

The idea is simple, fill in a bottom-up manner using previously computed values. We compute ranges with current power of 2 using values of lower power of two. For example, to find a minimum of range [0, 7] (Range size is a power of 3), we can use the minimum of following two.

a) Minimum of range [0, 3] (Range size is a power of 2)

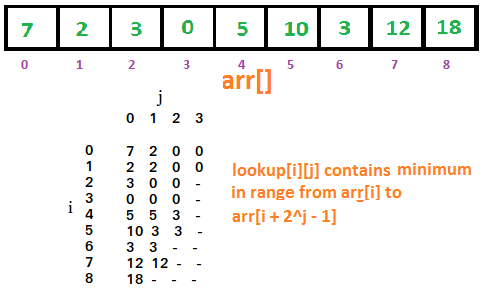
b) Minimum of range [4, 7] (Range size is a power of 2)

Based on above example, below is formula,

// Minimum of single element subarrays is same  
// as the only element.  
lookup[i][0] = arr[i]

// If lookup[0][2] <= lookup[4][2],   
// then lookup[0][3] = lookup[0][2]  
**If** lookup[i][j-1] <= lookup[i+2j-1][j-1]  
 lookup[i][j] = lookup[i][j-1]

// If lookup[0][2] > lookup[4][2],   
// then lookup[0][3] = lookup[4][2]  
**Else**   
 lookup[i][j] = lookup[i+2j-1][j-1]



For any arbitrary range [l, R], we need to use ranges which are in powers of 2. The idea is to use the closest power of 2. We always need to do at most one comparison (compare the minimum of two ranges which are powers of 2). One range starts with L and ends with “L + closest-power-of-2”. The other range ends at R and starts with “R – same-closest-power-of-2 + 1”. For example, if the given range is (2, 10), we compare the minimum of two ranges (2, 9) and (3, 10).

Based on above example, below is formula,

// For (2, 10), j = floor(Log2(10-2+1)) = 3  
j = floor(Log(R-L+1))

// If lookup[2][3] <= lookup[3][3],   
// then RMQ(2, 10) = lookup[2][3]  
**If** lookup[L][j] <= lookup[R-(int)pow(2, j)+1][j]  
 RMQ(L, R) = lookup[L][j]

// If lookup[2][3] > arr[lookup[3][3],   
// then RMQ(2, 10) = lookup[3][3]  
**Else**   
 RMQ(L, R) = lookup[R-(int)pow(2, j)+1][j]

Since we do only one comparison, the time complexity of query is O(1).

Below is the implementation of the above idea.

# Python3 program to do range minimum

# query using sparse table

**import** math

# Fills lookup array lookup[][] in

# bottom up manner.

**def** buildSparseTable(arr, n):

    # Initialize M for the intervals

    # with length 1

**for** i **in** range(0, n):

        lookup[i][0] **=** arr[i]

    j **=** 1

    # Compute values from smaller to

    # bigger intervals

**while** (1 << j) <**=** n:

        # Compute minimum value for all

        # intervals with size 2^j

        i **=** 0

**while** (i **+** (1 << j) **-** 1) < n:

            # For arr[2][10], we compare arr[lookup[0][7]]

            # and arr[lookup[3][10]]

**if** (lookup[i][j **-** 1] <

                lookup[i **+** (1 << (j **-** 1))][j **-** 1]):

                lookup[i][j] **=** lookup[i][j **-** 1]

**else**:

                lookup[i][j] **=** \

                        lookup[i **+** (1 << (j **-** 1))][j **-** 1]

            i **+=** 1

        j **+=** 1

# Returns minimum of arr[L..R]

**def** query(L, R):

    # Find highest power of 2 that is smaller

    # than or equal to count of elements in

    # given range. For [2, 10], j = 3

    j **=** int(math.log2(R **-** L **+** 1))

    # Compute minimum of last 2^j elements

    # with first 2^j elements in range.

    # For [2, 10], we compare arr[lookup[0][3]]

    # and arr[lookup[3][3]],

**if** lookup[L][j] <**=** lookup[R **-** (1 << j) **+** 1][j]:

**return** lookup[L][j]

**else**:

**return** lookup[R **-** (1 << j) **+** 1][j]

# Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    a **=** [7, 2, 3, 0, 5, 10, 3, 12, 18]

    n **=** len(a)

    MAX **=** 500

    # lookup[i][j] is going to store minimum

    # value in arr[i..j]. Ideally lookup table

    # size should not be fixed and should be

    # determined using n Log n. It is kept

    # constant to keep code simple.

    lookup **=** [[0 **for** i **in** range(MAX)]

**for** j **in** range(MAX)]

    buildSparseTable(a, n)

**print**(query(0, 4))

    print(query(4, 7))

    print(query(7, 8))

# This code is contributed by Rituraj Jain

**Output**

0  
3  
12

**Range sum query using Sparse Table**

We have an array arr[]. We need to find the sum of all the elements in the range L and R where 0 <= L <= R <= n-1. Consider a situation when there are many range queries.

**Examples:**

Input : 3 7 2 5 8 9  
 query(0, 5)  
 query(3, 5)  
 query(2, 4)  
Output : 34  
 22  
 15

Note : array is 0 based indexed  
 and queries too.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

Since there are no updates/modifications, we use the [Sparse table](https://www.geeksforgeeks.org/sparse-table/) to answer queries efficiently. In a sparse table, we break queries in powers of 2.

Suppose we are asked to compute sum of   
elements from arr[i] to arr[i+12].   
We do the following:

// Use sum of 8 (or 23) elements   
table[i][3] = sum(arr[i], arr[i + 1], ...  
 arr[i + 7]).

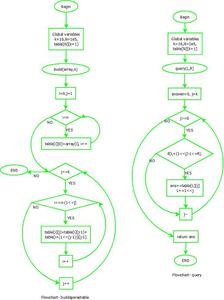
// Use sum of 4 elements  
table[i+8][2] = sum(arr[i+8], arr[i+9], ..  
 arr[i+11]).

// Use sum of single element  
table[i + 12][0] = sum(arr[i + 12]).

Our result is sum of above values.

Notice that it took only 4 actions to compute the result over a subarray of size 13.

**Flowchart:**



*Flowchart*

# Python3 program to find the sum in a given

# range in an array using sparse table.

# Because 2^17 is larger than 10^5

k **=** 16

# Maximum value of array

n **=** 100000

# k + 1 because we need to access

# table[r][k]

table **=** [[0 **for** j **in** range(k**+**1)] **for** i **in** range(n)]

# it builds sparse table

**def** buildSparseTable(arr, n):

**global** table, k

**for** i **in** range(n):

        table[i][0] **=** arr[i]

**for** j **in** range(1,k**+**1):

**for** i **in** range(0,n**-**(1<<j)**+**1):

            table[i][j] **=** table[i][j**-**1] **+** \

                          table[i **+** (1 << (j **-** 1))][j **-** 1]

# Returns the sum of the elements in the range

# L and R.

**def** query(L, R):

**global** table, k

    # boundaries of next query, 0 - indexed

    answer **=** 0

**for** j **in** range(k,**-**1,**-**1):

**if** (L **+** (1 << j) **-** 1 <**=** R):

            answer **=** answer **+** table[L][j]

            # instead of having L ', we

            # increment L directly

            L**+=**1<<j

**return** answer

# Driver program

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    arr **=** [3, 7, 2, 5, 8, 9]

    n **=** len(arr)

    buildSparseTable(arr, n)

    print(query(0,5))

**print**(query(3,5))

**print**(query(2,4))

# This code is contributed by

# chaudhary\_19 (Mayank Chaudhary)

**Output:**

34  
22  
15

**Range Minimum Query (Square Root Decomposition and Sparse Table)**

We have an array arr[0 . . . n-1]. We should be able to efficiently find the minimum value from index L (query start) to R (query end) where 0 <= **L** <= **R** <= **n-1**. Consider a situation when there are many range queries.

**Example:**

Input: arr[] = {7, 2, 3, 0, 5, 10, 3, 12, 18};  
 query[] = [0, 4], [4, 7], [7, 8]

Output: Minimum of [0, 4] is 0  
 Minimum of [4, 7] is 3  
 Minimum of [7, 8] is 12

A **simple solution** is to run a loop from **L** to **R** and find the minimum element in the given range. This solution takes **O(n)**time to query in the worst case.

Another approach is to use [**Segment tree**](https://www.geeksforgeeks.org/segment-tree-set-1-range-minimum-query/). With segment tree, preprocessing time is O(n) and time to for range minimum query is **O(Logn)**. The extra space required is O(n) to store the segment tree. Segment tree allows updates also in**O(Log n)** time.

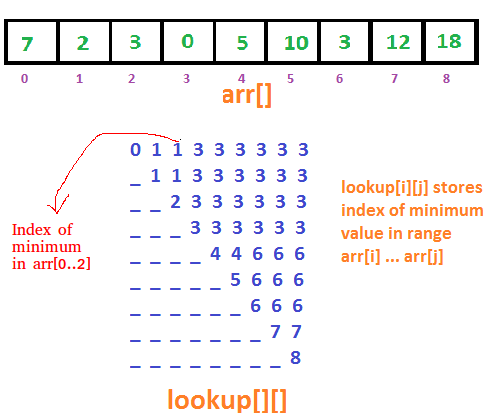
**Can we do better if we know that the array is static?**

How to optimize query time when there are no update operations and there are many range minimum queries?

Below are different methods.

**Method 1 (Simple Solution)**

A Simple Solution is to create a 2D array lookup[][] where an entry lookup[i][j] stores the minimum value in range arr[i..j]. The minimum of a given range can now be calculated in O(1) time.



# Python3 program to do range

# minimum query in O(1) time with

# O(n\*n) extra space and O(n\*n)

# preprocessing time.

MAX **=** 500

# lookup[i][j] is going to store

# index of minimum value in

# arr[i..j]

lookup **=** [[0 **for** j **in** range(MAX)]

**for** i **in** range(MAX)]

# Structure to represent

# a query range

**class** Query:

**def** \_\_init\_\_(self, L, R):

        self.L **=** L

        self.R **=** R

# Fills lookup array lookup[n][n]

# for all possible values

# of query ranges

**def** preprocess(arr, n):

    # Initialize lookup[][] for the

    # intervals with length 1

**for** i **in** range(n):

        lookup[i][i] **=** i;

    # Fill rest of the entries in

    # bottom up manner

**for** i **in** range(n):

**for** j **in** range(i **+** 1, n):

            # To find minimum of [0,4],

            # we compare minimum

            # of arr[lookup[0][3]] with arr[4].

**if** (arr[lookup[i][j **-** 1]] < arr[j]):

                lookup[i][j] **=** lookup[i][j **-** 1];

**else**:

                lookup[i][j] **=** j;

# Prints minimum of given m

# query ranges in arr[0..n-1]

**def** RMQ(arr, n, q, m):

    # Fill lookup table for

    # all possible input queries

    preprocess(arr, n);

    # One by one compute sum of

    # all queries

**for** i **in** range(m):

        # Left and right boundaries

        # of current range

        L **=** q[i].L

        R **=** q[i].R;

        # Print sum of current query range

**print**("Minimum of [" **+** str(L) **+** ", " **+**

               str(R) **+** "] is " **+**

               str(arr[lookup[L][R]]))

# Driver code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    a **=** [7, 2, 3, 0, 5,

         10, 3, 12, 18]

    n **=** len(a)

    q **=** [Query(0, 4),

         Query(4, 7),

         Query(7, 8)]

    m **=** len(q)

    RMQ(a, n, q, m);

# This code is contributed by Rutvik\_56

**Output:**

Minimum of [0, 4] is 0  
Minimum of [4, 7] is 3  
Minimum of [7, 8] is 12

**Range LCM Queries**

Given an array arr[] of integers of size N and an array of Q queries, query[], where each query is of type [L, R] denoting the range from index L to index R, the task is to find the LCM of all the numbers of the range for all the queries.

**Examples:**

***Input:****arr[] = {5, 7, 5, 2, 10, 12 ,11, 17, 14, 1, 44}*

*query[] = {{2, 5}, {5, 10}, {0, 10}}*

***Output:****60,15708, 78540*

***Explanation:****In the first query LCM(5, 2, 10, 12) = 60*

*In the second query LCM(12, 11, 17, 14, 1, 44) = 15708*

*In the last query LCM(5, 7, 5, 2, 10, 12, 11, 17, 14, 1, 44) = 78540*

***Input:****arr[] = {2, 4, 8, 16}, query[] = {{2, 3}, {0, 1}}*

***Output:****16, 4*

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Naive Approach:** The approach is based on the following mathematical idea:

*Mathematically,  LCM(l, r) = LCM(arr[l],  arr[l+1] , . . . ,arr[r-1], arr[r]) and*

*LCM(a, b) = (a\*b) / GCD(a,b)*

So traverse the array for every query and calculate the answer by using the above formula for LCM.

**Time Complexity:** O(N \* Q)

**Auxiliary Space:** O(1)

**RangeLCM Queries using**[Segment tree](https://www.geeksforgeeks.org/introduction-to-segment-trees/)**:**

As the number of queries can be large, the naive solution would be impractical. This time can be reduced

*There is no update operation in this problem. So we can initially build a segment tree and use that to answer the queries in logarithmic time.*

*Each node in the tree should store the LCM value for that particular segment and we can use the same formula as above to combine the segments.*

Follow the steps mentioned below to implement the idea:

* Build a segment tree from the given array.
* Traverse through the queries. For each query:
* Find that particular range in the segment tree.
* Use the above mentioned formula to combine the segments and calculate the LCM for that range.
* Print the answer for that segment.

Below is the implementation of the above approach.

# LCM of given range queries using Segment Tree

MAX **=** 1000

# allocate space for tree

tree **=** [0] **\*** (4 **\*** MAX)

# declaring the array globally

arr **=** [0] **\*** MAX

# Function to return gcd of a and b

**def** gcd(a: int, b: int):

**if** a **==** 0:

**return** b

**return** gcd(b **%** a, a)

# utility function to find lcm

**def** lcm(a: int, b: int):

**return** (a **\*** b) **//** gcd(a, b)

# Function to build the segment tree

# Node starts beginning index of current subtree.

# start and end are indexes in arr[] which is global

**def** build(node: int, start: int, end: int):

    # If there is only one element

    # in current subarray

**if** start **==** end:

        tree[node] **=** arr[start]

**return**

    mid **=** (start **+** end) **//** 2

    # build left and right segments

    build(2 **\*** node, start, mid)

    build(2 **\*** node **+** 1, mid **+** 1, end)

    # build the parent

    left\_lcm **=** tree[2 **\*** node]

    right\_lcm **=** tree[2 **\*** node **+** 1]

    tree[node] **=** lcm(left\_lcm, right\_lcm)

# Function to make queries for array range )l, r).

# Node is index of root of current segment in segment

# tree (Note that indexes in segment tree begin with 1

# for simplicity).

# start and end are indexes of subarray covered by root

# of current segment.

**def** query(node: int, start: int,

          end: int, l: int, r: int):

    # Completely outside the segment,

    # returning 1 will not affect the lcm;

**if** end < l **or** start > r:

**return** 1

    # completely inside the segment

**if** l <**=** start **and** r >**=** end:

**return** tree[node]

    # partially inside

    mid **=** (start **+** end) **//** 2

    left\_lcm **=** query(2 **\*** node, start, mid, l, r)

    right\_lcm **=** query(2 **\*** node **+** 1,

                      mid **+** 1, end, l, r)

**return** lcm(left\_lcm, right\_lcm)

# Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    # initialize the array

    arr[0] **=** 5

    arr[1] **=** 7

    arr[2] **=** 5

    arr[3] **=** 2

    arr[4] **=** 10

    arr[5] **=** 12

    arr[6] **=** 11

    arr[7] **=** 17

    arr[8] **=** 14

    arr[9] **=** 1

    arr[10] **=** 44

    # build the segment tree

    build(1, 0, 10)

    # Now we can answer each query efficiently

    # Print LCM of (2, 5)

**print**(query(1, 0, 10, 2, 5))

    # Print LCM of (5, 10)

**print**(query(1, 0, 10, 5, 10))

    # Print LCM of (0, 10)

**print**(query(1, 0, 10, 0, 10))

# This code is contributed by

# sanjeev2552

**Output**

60  
15708  
78540

**Merge Sort Tree for Range Order Statistics**

Given an array of n numbers, the task is to answer the following queries:

kthSmallest(start, end, k) : Find the Kth smallest   
 number in the range from array  
 index 'start' to 'end'.

**Examples:**

**Input :** arr[] = {3, 2, 5, 1, 8, 9|  
 Query 1: start = 2, end = 5, k = 2  
 Query 2: start = 1, end = 6, k = 4  
**Output :** 2  
 5  
**Explanation:**  
[2, 5, 1, 8] represents the range from 2 to   
5 and 2 is the 2nd smallest number   
in the range[3, 2, 5, 1, 8, 9] represents   
the range from 1 to 6 and 5 is the 4th  
smallest number in the range

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

The key idea is to build a [Segment Tree](https://www.geeksforgeeks.org/segment-tree-set-1-sum-of-given-range/) with a vector at every node and the vector contains all the elements of the sub-range in a sorted order. And if we observe this segment tree structure this is somewhat similar to the tree formed during the [merge sort algorithm](https://www.geeksforgeeks.org/merge-sort/)(that is why it is called merge sort tree) We use same implementation as discussed in [Merge Sort Tree (Smaller or equal elements in given row range)](https://www.geeksforgeeks.org/merge-sort-tree-smaller-or-equal-elements-in-given-row-range/) Firstly, we maintain a vector of pairs where each pair {value, index} is such that first element of pair represents the element of the input array and the second element of the pair represents the index at which it occurs.

Now we sort this vector of pairs on the basis of the first element of each pair. After this we build a Merge Sort Tree where each node has a vector of indices in the sorted range. When we have to answer a query we find if the Kth smallest number lies in the left sub-tree or in the right sub-tree.

The idea is to use two binary searches and find the number of elements in the left sub-tree such  that the indices lie within the given query range. Let the number of such indices be M. If M>=K, it means we will be able to find the Kth smallest Number in the left sub-tree thus we call on the left sub-tree. Else the Kth smallest number lies in the right sub-tree but this time we don’t have to look for the K th smallest number as we already have first M smallest numbers of the range in the left sub-tree thus we should look for the remaining part ie the (K-M)th number in the right sub-tree. This is the Index of Kth smallest number the value at this index is the required number.

**Implementation:**

**Minimum number of jumps to reach end**

Given an array **arr[]**where each element represents the max number of steps that can be made forward from that index. The task is to find the minimum number of jumps to reach the end of the array starting from index **0**. If the end isn’t reachable, return **-1**.

**Examples:**

***Input:****arr[] = {1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9}*

***Output:****3 (1-> 3 -> 9 -> 9)*

***Explanation:****Jump from 1st element to 2nd element as there is only 1 step.*

*Now there are three options 5, 8 or 9. I*

*f 8 or 9 is chosen then the end node 9 can be reached. So 3 jumps are made.*

***Input:****arr[] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}*

***Output:****10*

***Explanation:****In every step a jump is needed so the count of jumps is 10.*

Recommended Practice

[Jump Game](https://practice.geeksforgeeks.org/problems/jump-game/1/)

[Try It!](https://practice.geeksforgeeks.org/problems/jump-game/1/)

**Minimum number of jumps to reach the end using**[Recursion](https://www.geeksforgeeks.org/introduction-to-recursion-data-structure-and-algorithm-tutorials/)**:**

*Start from the first element and recursively call for all the elements reachable from the first element. The minimum number of jumps to reach end from first can be calculated using the minimum value from the recursive calls.*

***minJumps(start, end) = Min ( minJumps(k, end) )****for all k reachable from start.*

Follow the steps mentioned below to implement the idea:

* Create a recursive function.
* In each recursive call get all the reachable nodes from that index.
* For each of the index call the recursive function.
* Find the minimum number of jumps to reach the end from current index.
* Return the minimum number of jumps from the recursive call.

Below is the Implementation of the above approach:

# Python3 program to find Minimum

# number of jumps to reach end

# Returns minimum number of jumps

# to reach arr[h] from arr[l]

**def** minJumps(arr, l, h):

    # Base case: when source and

    # destination are same

**if** (h **==** l):

**return** 0

    # when nothing is reachable

    # from the given source

**if** (arr[l] **==** 0):

**return** float('inf')

    # Traverse through all the points

    # reachable from arr[l]. Recursively

    # get the minimum number of jumps

    # needed to reach arr[h] from

    # these reachable points.

    min **=** float('inf')

**for** i **in** range(l **+** 1, h **+** 1):

**if** (i < l **+** arr[l] **+** 1):

            jumps **=** minJumps(arr, i, h)

**if** (jumps !**=** float('inf') **and**

                    jumps **+** 1 < min):

                min **=** jumps **+** 1

**return** min

# Driver program to test above function

arr **=** [1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9]

n **=** len(arr)

print('Minimum number of jumps to reach',

      'end is', minJumps(arr, 0, n**-**1))

# This code is contributed by Soumen Ghosh

**Output**

Minimum number of jumps to reach the end is 3

**Space optimization using bit manipulations**

There are many situations where we use integer values as index in array to see presence or absence, we can use bit manipulations to optimize space in such problems.

Let us consider below problem as an example.

Given two numbers say a and b, mark the multiples of 2 and 5 between a and b using less than O(|b – a|) space and output each of the multiples.

**Note :** We have to **mark** the multiples i.e save (key, value) pairs in memory such that each key either have value as 1 or 0 representing as multiple of 2 or 5 or not respectively.

**Examples :**

**Input :** 2 10  
**Output :** 2 4 5 6 8 10

**Input:** 60 95  
**Output:** 60 62 64 65 66 68 70 72 74 75 76 78   
 80 82 84 85 86 88 90 92 94 95

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Approach 1 (Simple):**

Hash the indices in an array from a to b and mark each of the indices as 1 or 0.

**Space complexity : O(max(a, b))**

C:\Users\qj771f\AppData\Local\Temp\msohtmlclip1\02\clip_image022.png

**Approach 2 (Better than simple):**

Save memory, by translating a to 0th index and b to (b-a)th index.

**Space complexity : O(|b-a|).**

C:\Users\qj771f\AppData\Local\Temp\msohtmlclip1\02\clip_image023.png

Simply hash |b – a| positions of an array as 0 and 1.

**Implementation:**

**Python3**

# Python3 program to mark numbers

# as multiple of 2 or 5

**import** math

# Driver code

a **=** 2

b **=** 10

size **=** abs(b **-** a) **+** 1

array **=** [0] **\*** size

# Iterate through a to b,

# If it is a multiple of 2

# or 5 Mark index in array as 1

**for** i **in** range(a, b **+** 1):

**if** (i **%** 2 **==** 0 **or** i **%** 5 **==** 0):

            array[i **-** a] **=** 1

print("MULTIPLES of 2 and 5:")

**for** i **in** range(a, b **+** 1):

**if** (array[i **-** a] **==** 1):

            print(i, end**=**" ")

**Sort a nearly sorted (or K sorted) array**

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.

**Examples:**

***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}*

Recommended Problem

Nearly sorted

**Sort a nearly sorted (or K sorted) array using**[insertion sort](https://www.geeksforgeeks.org/insertion-sort/)**:**

To solve the problem follow the below idea:

*We can use insertion sort to sort the array efficiently as index of every element can be changed by atmost K places, which will reduce the time complexity of this algorithm from O(N2) to O(NK).*

Follow the below steps to solve the problem:

* Iterate from arr[1] to arr[N] over the array.
* Compare the current element (key) to its predecessor.
* If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

Below is the implementation of the above approach:

# Function to sort an array using

# insertion sort

**def** insertionSort(A, size):

    i, key, j **=** 0, 0, 0

**for** i **in** range(size):

        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 **and** A[j] > key:

            A[j **+** 1] **=** A[j]

            j **=** j **-** 1

        A[j **+** 1] **=** key

**Output**

2 3 5 6 8 9 10

**Find maximum value of Sum( i\*arr[i]) with only rotations on given array allowed**

Given an array **arr[]** of size **N**, the task is to find the maximum possible sum of **i\*arr[i]** when the array can be rotated any number of times.

**Examples :**

***Input:****arr[] = {1, 20, 2, 10}*

***Output:****72.We can get 72 by rotating array twice.*

*{2, 10, 1, 20}*

*20\*3 + 1\*2 + 10\*1 + 2\*0 = 72*

***Input:****arr[] = {10, 1, 2, 3, 4, 5, 6, 7, 8, 9}*

***Output:****330*

*We can get 330 by rotating array 9 times.*

*{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};*

*0\*1 + 1\*2 + 2\*3 … 9\*10 = 330*

**Naive Approach:** The basic idea of this approach is

*Find all rotations one by one, check the sum of every rotation and return the maximum sum.*

**Time Complexity:** O(N2)

**Auxiliary Space:** O(1)

**Efficient Approach:** The idea is as follows:

*Let****Rj****be value of****i\*arr[i]****with****j****rotations.*

* *The idea is to calculate the next rotation value from the previous rotation, i.e., calculate****Rj****from****Rj-1****.*
* *We can calculate the initial value of the result as R0, then keep calculating the next rotation values.*

**How to efficiently calculate Rj from Rj-1?**

*This can be done in****O(1)****time. Below are the details.*

*Let us calculate initial value of****i\*arr[i]****with no rotation*

***R0 = 0\*arr[0] + 1\*arr[1] +…+ (n-1)\*arr[n-1]***

*After 1 rotation****arr[n-1]****, becomes first element of array,*

* *arr[0] becomes second element, arr[1] becomes third element and so on.*
* *R1 = 0\*arr[n-1] + 1\*arr[0] +…+ (n-1)\*arr[n-2]*
* *R1 – R0 = arr[0] + arr[1] + … + arr[n-2] – (n-1)\*arr[n-1]*

*After 2 rotations****arr[n-2]****, becomes first element of array,*

* *arr[n-1] becomes second element, arr[0] becomes third element and so on.*
* *R2 = 0\*arr[n-2] + 1\*arr[n-1] +…+ (n-1)\*arr[n-3]*
* *R2 – R1 = arr[0] + arr[1] + … + arr[n-3] – (n-1)\*arr[n-2] + arr[n-1]*

*If we take a closer look at above values, we can observe below pattern*

***Rj – Rj-1 = arrSum – n \* arr[n-j]****,*

*Where****arrSum****is sum of all array elements, i.e.,****arrSum = ∑ arr[i]****, 0 ≤ i ≤ N-1*

Follow the below illustration for a better understanding.

**Illustration:**

*Given arr[]={10, 1, 2, 3, 4, 5, 6, 7, 8, 9},*

***arrSum = 55****, currVal = summation of (i\*arr[i]) =****285***

*In each iteration the currVal is****currVal = currVal + arrSum-n\*arr[n-j]****,*

***1st rotation:****currVal = 285 + 55 –  (10 \*  9) =  250*

***2nd rotation:****currVal = 285 + 55 – (10 \* 8) = 260*

***3rd rotation:****currVal = 285 + 55 – (10 \* 7) = 270*

*.*

*.*

*.*

***Last rotation:****currVal = 285 + 55 – (10 \* 1) = 330*

*Previous currVal was 285, now it becomes 330.*

*It’s the maximum value we can find hence return****330****.*

Follow the steps mentioned below to implement the above approach:

* Compute the sum of all array elements. Let this sum be ‘**arrSum**‘.
* Compute **R0** for the given array. Let this value be **currVal**.
* Loop from **j = 1 to N-1** to calculate the value for each rotation:
* Update the **currVal** using the formula mentioned above.
* Update the maximum sum accordingly in each step.
* Return the maximum value as the required answer.

Below is the implementation of the above idea.

'''Python program to find maximum value of Sum(i\*arr[i])'''

# returns max possible value of Sum(i\*arr[i])

**def** maxSum(arr):

    # stores sum of arr[i]

    arrSum **=** 0

    # stores sum of i\*arr[i]

    currVal **=** 0

    n **=** len(arr)

**for** i **in** range(0, n):

        arrSum **=** arrSum **+** arr[i]

        currVal **=** currVal **+** (i**\***arr[i])

    # initialize result

    maxVal **=** currVal

    # try all rotations one by one and find the maximum

    # rotation sum

**for** j **in** range(1, n):

        currVal **=** currVal **+** arrSum**-**n**\***arr[n**-**j]

**if** currVal > maxVal:

            maxVal **=** currVal

    # return result

**return** maxVal

# test maxsum(arr) function

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    arr **=** [10, 1, 2, 3, 4, 5, 6, 7, 8, 9]

    print "Max sum is: ", maxSum(arr)

**Output**

Max sum is 330

**Median in a stream of integers (running integers)**

Given that integers are read from a data stream. Find median of elements read so for in an efficient way. For simplicity assume, there are no duplicates. For example, let us consider the stream 5, 15, 1, 3 …

After reading 1st element of stream - 5 -> median - 5  
After reading 2nd element of stream - 5, 15 -> median - 10  
After reading 3rd element of stream - 5, 15, 1 -> median - 5  
After reading 4th element of stream - 5, 15, 1, 3 -> median - 4, so on...

Making it clear, when the input size is odd, we take the middle element of sorted data. If the input size is even, we pick the average of the middle two elements in the sorted stream.

Note that output is the *effective median* of integers read from the stream so far. Such an algorithm is called an online algorithm. Any algorithm that can guarantee the output of *i*-elements after processing *i*-th element, is said to be ***online algorithm***. Let us discuss three solutions to the above problem.

Recommended Problem

Find median in a stream

**Method 1:** Insertion Sort

If we can sort the data as it appears, we can easily locate the median element. *Insertion Sort* is one such online algorithm that sorts the data appeared so far. At any instance of sorting, say after sorting *i*-th element, the first *i* elements of the array are sorted. The insertion sort doesn’t depend on future data to sort data input till that point. In other words, insertion sort considers data sorted so far while inserting the next element. This is the key part of insertion sort that makes it an online algorithm.

However, insertion sort takes O(n2) time to sort *n* elements. Perhaps we can use *binary search* on *insertion sort* to find the location of the next element in O(log n) time. Yet, we can’t do data movement in O(log n) time. No matter how efficient the implementation is, it takes polynomial time in case of insertion sort.

Interested readers can try the implementation of Method 1.

# Function to find position to insert current element of

# stream using binary search

**def** binarySearch(arr, item, low, high):

**if** (low >**=** high):

**return** (low **+** 1) **if** (item > arr[low]) **else** low

    mid **=** (low **+** high) **//** 2

**if** (item **==** arr[mid]):

**return** mid **+** 1

**if** (item > arr[mid]):

**return** binarySearch(arr, item, mid **+** 1, high)

**return** binarySearch(arr, item, low, mid **-** 1)

# Function to print median of stream of integers

**def** printMedian(arr, n):

    i, j, pos, num **=** 0, 0, 0, 0

    count **=** 1

    print(f"Median after reading 1 element is {arr[0]}")

**for** i **in** range(1, n):

        median **=** 0

        j **=** i **-** 1

        num **=** arr[i]

        # find position to insert current element in sorted

        # part of array

        pos **=** binarySearch(arr, num, 0, j)

        # move elements to right to create space to insert

        # the current element

**while** (j >**=** pos):

            arr[j **+** 1] **=** arr[j]

            j **-=** 1

        arr[j **+** 1] **=** num

        # increment count of sorted elements in array

        count **+=** 1

        # if odd number of integers are read from stream

        # then middle element in sorted order is median

        # else average of middle elements is median

**if** (count **%** 2 !**=** 0):

            median **=** arr[count **//** 2]

**else**:

            median **=** (arr[(count **//** 2) **-** 1] **+** arr[count **//** 2]) **//** 2

**print**(f"Median after reading {i + 1} elements is {median} ")

# Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    arr **=** [5, 15, 1, 3, 2, 8, 7, 9, 10, 6, 11, 4]

    n **=** len(arr)

    printMedian(arr, n)

# This code is contributed by rakeshsahni

**Output**

Median after reading 1 element is 5  
Median after reading 2 elements is 10  
Median after reading 3 elements is 5  
Median after reading 4 elements is 4  
Median after reading 5 elements is 3  
Median after reading 6 elements is 4  
Median after reading 7 elements is 5  
Median after reading 8 elements is 6  
Median after reading 9 elements is 7  
Median after reading 10 elements is 6  
Median after reading 11 elements is 7  
Median after reading 12 elements is 6

**Construct an array from its pair-sum array**

Given a pair-sum array and size of the original array (n), construct the original array.

A pair-sum array for an array is the array that contains sum of all pairs in ordered form. For example pair-sum array for arr[] = {6, 8, 3, 4} is {14, 9, 10, 11, 12, 7}.

In general, pair-sum array for arr[0..n-1] is {arr[0]+arr[1], arr[0]+arr[2], ……., arr[1]+arr[2], arr[1]+arr[3], ……., arr[2]+arr[3], arr[2]+arr[4], …., arr[n-2]+arr[n-1}.

“Given a pair-sum array, construct the original array.”

**We strongly recommend to minimize your browser and try this yourself.**

Let the given array be “pair[]” and let there be n elements in original array. If we take a look at few examples, we can observe that arr[0] is half of pair[0] + pair[1] – pair[n-1]. Note that the value of pair[0] + pair[1] – pair[n-1] is (arr[0] + arr[1]) + (arr[0] + arr[2]) – (arr[1] + arr[2]).

Once we have evaluated arr[0], we can evaluate other elements by subtracting arr[0]. For example arr[1] can be evaluated by subtracting arr[0] from pair[0], arr[2] can be evaluated by subtracting arr[0] from pair[1].

Following is the implementation of the above idea.

# Fills element in arr[] from its

# pair sum array pair[].

# n is size of arr[]

**def** constructArr(arr,pair,n):

    arr [0] **=** (pair[0]**+**pair[1]**-**pair[n**-**1])**//**2

**for** i **in** range(1,n):

        arr[i] **=** pair[i**-**1]**-**arr[0]

# Driver code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    pair **=** [15, 13, 11, 10, 12, 10, 9, 8, 7, 5]

    n **=**5

    arr **=** [0]**\***n

    constructArr(arr,pair,n)

**for** i **in** range(n):

        print(arr[i],end **=**" ")

# This code is contributed by

# Shrikant13

**Output:**

8 7 5 3 2

**Maximum equilibrium sum in an array**

Given an array arr[]. Find the maximum value of prefix sum which is also suffix sum for index i in arr[].

**Examples :**

Input : arr[] = {-1, 2, 3, 0, 3, 2, -1}  
Output : 4  
Prefix sum of arr[0..3] =   
 Suffix sum of arr[3..6]

Input : arr[] = {-2, 5, 3, 1, 2, 6, -4, 2}  
Output : 7  
Prefix sum of arr[0..3] =   
 Suffix sum of arr[3..7]

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

A **Simple Solution** is to one by one check the given condition (prefix sum equal to suffix sum) for every element and returns the element that satisfies the given condition with maximum value.

# Python 3 program to find maximum

# equilibrium sum.

**import** sys

# Function to find maximum equilibrium sum.

**def** findMaxSum(arr, n):

    res **= -**sys.maxsize **-** 1

**for** i **in** range(n):

        prefix\_sum **=** arr[i]

**for** j **in** range(i):

            prefix\_sum **+=** arr[j]

        suffix\_sum **=** arr[i]

        j **=** n **-** 1

**while**(j > i):

            suffix\_sum **+=** arr[j]

            j **-=** 1

**if** (prefix\_sum **==** suffix\_sum):

            res **=** max(res, prefix\_sum)

**return** res

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    arr **=** [**-**2, 5, 3, 1, 2, 6, **-**4, 2]

    n **=** len(arr)

**print**(findMaxSum(arr, n))

# This code is contributed by

# Surendra\_Gangwar

**Output :**

7

**Leaders in an array**

Write a program to print all the LEADERS in the array. An element is a leader if it is greater than all the elements to its right side. And the rightmost element is always a leader.

**For example:**

***Input:****arr[] = {16, 17, 4, 3, 5, 2},*

***Output:****17, 5, 2*

***Input:****arr[] = {1, 2, 3, 4, 5, 2},*

***Output:****5, 2*

Recommended Problem

Leaders in an array

**Naive Approach**: The problem can be solved based on the idea mentioned below:

*Use two loops. The outer loop runs from 0 to size – 1 and one by one pick all elements from left to right. The inner loop compares the picked element to all the elements on its right side. If the picked element is greater than all the elements to its right side, then the picked element is the leader.*

Follow the below steps to implement the idea:

* We run a loop from the first index to the 2nd last index.
* And for each index, we run another loop from the next index to the last index.
* If all the values to the right of that index are smaller than the index, we simply add the value in our answer data structure.

Below is the implementation of the above approach.

# Python Function to print leaders in array

**def** printLeaders(arr,size):

**for** i **in** range(0, size):

**for** j **in** range(i**+**1, size):

**if** arr[i]<**=**arr[j]:

**break**

**if** j **==** size**-**1: # If loop didn't break

**print** (arr[i],end**=**' ')

# Driver function

arr**=**[16, 17, 4, 3, 5, 2]

printLeaders(arr, len(arr))

# This code is contributed by \_Devesh Agrawal\_\_

**Output**

17 5 2

**Smallest Difference Triplet from Three arrays**

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 : 7, 6, 5

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.

Recommended Problem

Happiest Triplet

**Simple Solution :**Consider each an every triplet and find the required smallest difference triplet out of them. Complexity of O(n3).

**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.

**Implementatipon:**

# 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

**else if** (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.

**Output**

7, 6, 5

**Find all triplets with zero sum**

Given an array of distinct elements. The task is to find triplets in the array whose sum is zero.

**Examples :**

***Input:****arr[] = {0, -1, 2, -3, 1}*

***Output:****(0 -1 1), (2 -3 1)*

***Explanation:****The triplets with zero sum are 0 + -1 + 1 = 0 and 2 + -3 + 1 = 0*

***Input:****arr[] = {1, -2, 1, 0, 5}*

***Output:****1 -2  1*

***Explanation:****The triplets with zero sum is 1 + -2 + 1 = 0*

Recommended Problem

Find triplets with zero sum

**Naive approach:**Below is the idea to solve the problem

*Run three loops and check one by one whether the sum of the three elements is zero or not. If the sum of three elements is zero then****print elements****otherwise print****not found****.*

Follow the below steps to Implement the Idea:

* Run three nested loops with loop counter **i, j, k**
* The first loops will run from **0** to **n-3**and second loop from **i+1** to **n-2**and the third loop from **j+1** to b. The loop counter represents the three elements of the triplet.
* Check if the sum of elements at i’th, j’th, k’th is equal to zero or not. If yes print the sum else continue.

# 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 **=** False

**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

**Output**

0 -1 1  
2 -3 1

**Implement two Stacks in an Array**

Create a data structure **twoStacks**that represent two stacks. Implementation of **twoStacks**should use only one array, i.e., both stacks should use the same array for storing elements.

*Following functions must be supported by twoStacks.*

* *push1(int x) –> pushes x to first stack*
* *push2(int x) –> pushes x to second stack*
* *pop1() –> pops an element from first stack and return the popped element*
* *pop2() –> pops an element from second stack and return the popped element*

*Implementation of twoStack should be space efficient.*

Recommended Problem

**Implement two stacks in an array by Dividing the space into two halves:**

*The idea to implement two stacks is to divide the array into two halves and assign two halves to two stacks, i.e., use arr[0] to arr[n/2] for stack1, and arr[(n/2) + 1] to arr[n-1] for stack2 where arr[] is the array to be used to implement two stacks and size of array be n.*

Follow the steps below to solve the problem:

* To implement **push1():**
* First, check whether the **top1** is greater than 0
* If it is then add an element at the top1 index and decrement top1 by 1
* Else return Stack Overflow
* To implement **push2():**
* First, check whether **top2** is less than n – 1
* If it is then add an element at the top2 index and increment the top2 by 1
* Else return Stack Overflow
* To implement **pop1():**
* First, check whether the **top1** is less than or equal to n / 2
* If it is then increment the top1 by 1 and return that element.
* Else return Stack Underflow
* To implement **pop2():**
* First, check whether the **top2** is greater than or equal to (n + 1) / 2
* If it is then decrement the top2 by 1 and return that element.
* Else return Stack Underflow

Below is the implementation of the above approach.

# Python Script to Implement two stacks in a list

**import** math

**class** twoStacks:

**def** \_\_init\_\_(self, n):     # constructor

        self.size **=** n

        self.arr **=** [None] **\*** n

        self.top1 **=** math.floor(n**/**2) **+** 1

        self.top2 **=** math.floor(n**/**2)

    # Method to push an element x to stack1

**def** push1(self, x):

        # There is at least one empty space for new element

**if** self.top1 > 0:

            self.top1 **=** self.top1 **-** 1

            self.arr[self.top1] **=** x

**else**:

**print**("Stack Overflow by element : ", x)

    # Method to push an element x to stack2

**def** push2(self, x):

        # There is at least one empty space for new element

**if** self.top2 < self.size **-** 1:

            self.top2 **=** self.top2 **+** 1

            self.arr[self.top2] **=** x

**else**:

**print**("Stack Overflow by element : ", x)

    # Method to pop an element from first stack

**def** pop1(self):

**if** self.top1 <**=** self.size**/**2:

            x **=** self.arr[self.top1]

            self.top1 **=** self.top1 **+** 1

**return** x

**else**:

**print**("Stack Underflow")

            exit(1)

    # Method to pop an element from second stack

**def** pop2(self):

**if** self.top2 >**=** math.floor(self.size**/**2) **+** 1:

            x **=** self.arr[self.top2]

            self.top2 **=** self.top2 **-** 1

**return** x

**else**:

            print("Stack Underflow")

            exit(1)

# Driver program to test twoStacks class

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    ts **=** twoStacks(5)

    ts.push1(5)

    ts.push2(10)

    ts.push2(15)

    ts.push1(11)

    ts.push2(7)

    print("Popped element from stack1 is : " **+** str(ts.pop1()))

    ts.push2(40)

**print**("Popped element from stack2 is : " **+** str(ts.pop2()))

# This code is contributed by Gautam goel

**Output**

Stack Overflow By element : 7  
Popped element from stack1 is : 11  
Stack Overflow By element : 40  
Popped element from stack2 is : 15

**Minimum increment by k operations to make all elements equal**

You are given an array of n-elements, you have to find the number of operations needed to make all elements of array equal. Where a single operation can increment an element by k. If it is not possible to make all elements equal print -1.

**Example :**

**Input :** arr[] = {4, 7, 19, 16}, k = 3  
**Output :** 10

**Input :** arr[] = {4, 4, 4, 4}, k = 3  
**Output :** 0

**Input :** arr[] = {4, 2, 6, 8}, k = 3  
**Output :** -1

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

To solve this question we require to check whether all elements can became equal or not and that too only by incrementing k from elements value. For this we have to check that the difference of any two elements should always be divisible by k. If it is so, then all elements can become equal otherwise they can not became equal in any case by incrementing k from them. Also, the number of operations required can be calculated by finding value of (max – Ai)/k for all elements. where max is maximum element of array.

**Algorithm :**

// iterate for all elements  
for (int i=0; i<n; i++)  
{  
 // check if element can make equal to max  
 // or not if not then return -1  
 if ((max - arr[i]) % k != 0 )  
 return -1;

// else update res for required operations  
 else  
 res += (max - arr[i]) / k ;  
}

return res;

**Implementation:**

# Python3 Program to make all array equal

# function for calculating min operations

**def** minOps(arr, n, k):

    # max elements of array

    max1 **=** max(arr)

    res **=** 0

    # iterate for all elements

**for** i **in** range(0, n):

        # check if element can make equal to

        # max or not if not then return -1

**if** ((max1 **-** arr[i]) **%** k !**=** 0):

**return -**1

        # else update res for

        # required operations

**else**:

            res **+=** (max1 **-** arr[i]) **/** k

    # return result

**return** int(res)

# driver program

arr **=** [21, 33, 9, 45, 63]

n **=** len(arr)

k **=** 6

print(minOps(arr, n, k))

# This code is contributed by

# Smitha Dinesh Semwal

**Output**

24