Hashing Data Structure

[**Data Structure and Algorithms Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=gfg&utm_medium=header+link+click&utm_campaign=dsa+course+tracker&utm_term=dsa+course+promo&utm_content=hash-lp)

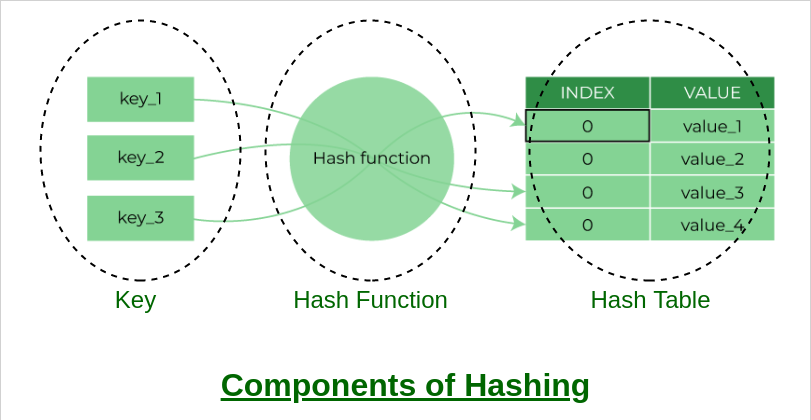
[**Practice Problems on Hashing**](https://practice.geeksforgeeks.org/explore/?category%5B%5D=Hash&page=1&category%5B%5D=Hash&utm_source=gfg&utm_medium=header+link+click&utm_campaign=practice+tracker&utm_term=practice+promo&utm_content=hash-lp)

[**Recent Articles on Hashing**](https://www.geeksforgeeks.org/category/Hash/?utm_source=gfg&utm_medium=header+link+click&utm_campaign=recent+article+tracker&utm_term=recent+article+tracker&utm_content=hash-lp)

[**What is Hashing?**](https://www.geeksforgeeks.org/introduction-to-hashing-data-structure-and-algorithm-tutorials/)

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.

Let a hash function H(x) maps the value **x** at the index **x%10** in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.



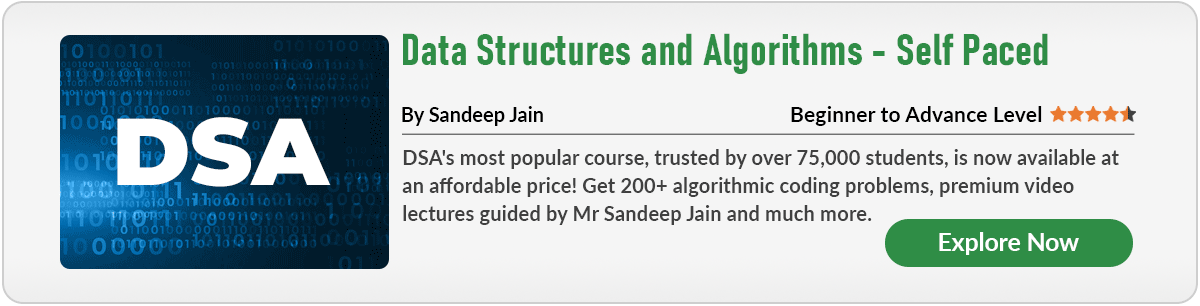
*Hashing Data Structure*

**Topics:**

* [Introduction](https://www.geeksforgeeks.org/hashing-data-structure/?ref=ghm#introduction)
* [Standard problems on Hashing](https://www.geeksforgeeks.org/hashing-data-structure/?ref=ghm#standard)

**Introduction:**

1. [Introduction to Hashing – Data Structure and Algorithm Tutorials](https://www.geeksforgeeks.org/introduction-to-hashing-data-structure-and-algorithm-tutorials/)
2. [What is Hashing?](https://www.geeksforgeeks.org/what-is-hashing/)
3. [Index Mapping (or Trivial Hashing)](https://www.geeksforgeeks.org/index-mapping-or-trivial-hashing-with-negatives-allowed/)
4. [Separate Chaining for Collision Handling](https://www.geeksforgeeks.org/hashing-set-2-separate-chaining/)
5. [Open Addressing for Collision Handling](https://www.geeksforgeeks.org/hashing-set-3-open-addressing/)
6. [Double Hashing](https://www.geeksforgeeks.org/double-hashing/)
7. [Load Factor and Rehashing](https://www.geeksforgeeks.org/load-factor-and-rehashing/)

[](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=page&utm_medium=page&utm_campaign=dsa-self-paced)

**Standard problem on Hashing:**

* **Easy:**
  1. [Find whether an array is subset of another array](https://www.geeksforgeeks.org/find-whether-an-array-is-subset-of-another-array-set-1/)
  2. [Union and Intersection of two linked lists](https://www.geeksforgeeks.org/union-intersection-two-linked-lists-set-3-hashing/)
  3. [Given an array A[] and a number x, check for pair in A[] with sum as x](https://www.geeksforgeeks.org/write-a-c-program-that-given-a-set-a-of-n-numbers-and-another-number-x-determines-whether-or-not-there-exist-two-elements-in-s-whose-sum-is-exactly-x/)
  4. [Maximum distance between two occurrences of same element in array](https://www.geeksforgeeks.org/maximum-distance-two-occurrences-element-array/)
  5. [Count maximum points on same line](https://www.geeksforgeeks.org/count-maximum-points-on-same-line/)
  6. [Most frequent element in an array](https://www.geeksforgeeks.org/frequent-element-array/)
  7. [Find the only repetitive element between 1 to n-1](https://www.geeksforgeeks.org/find-repetitive-element-1-n-1/)
  8. [How to check if two given sets are disjoint?](https://www.geeksforgeeks.org/check-two-given-sets-disjoint/)
  9. [Non-overlapping sum of two sets](https://www.geeksforgeeks.org/overlapping-sum-two-array/)
  10. [Check if two arrays are equal or not](https://www.geeksforgeeks.org/check-if-two-arrays-are-equal-or-not/)
  11. [Find missing elements of a range](https://www.geeksforgeeks.org/find-missing-elements-of-a-range/)
  12. [Minimum number of subsets with distinct elements](https://www.geeksforgeeks.org/minimum-number-subsets-distinct-elements/)
  13. [Remove minimum number of elements such that no common element exist in both array](https://www.geeksforgeeks.org/remove-minimum-number-elements-no-common-element-exist-array/)
  14. [Find pairs with given sum such that elements of pair are in different rows](https://www.geeksforgeeks.org/find-pairs-given-sum-elements-pair-different-rows/)
  15. [Count pairs with given sum](https://www.geeksforgeeks.org/count-pairs-with-given-sum/)
  16. [Count quadruples from four sorted arrays whose sum is equal to a given value x](https://www.geeksforgeeks.org/count-quadruples-four-sorted-arrays-whose-sum-equal-given-value-x/)
  17. [Sort elements by frequency](https://www.geeksforgeeks.org/sort-elements-frequency-set-4-efficient-approach-using-hash/)
  18. [Find all pairs (a, b) in an array such that a % b = k](https://www.geeksforgeeks.org/find-pairs-b-array-b-k/)
  19. [Group words with same set of characters](https://www.geeksforgeeks.org/print-words-together-set-characters/)
  20. [k-th distinct (or non-repeating) element in an array.](https://www.geeksforgeeks.org/k-th-distinct-or-non-repeating-element-in-an-array/)
* **Medium:**
  1. [Find Itinerary from a given list of tickets](https://www.geeksforgeeks.org/find-itinerary-from-a-given-list-of-tickets/)
  2. [Find number of Employees Under every Employee](https://www.geeksforgeeks.org/find-number-of-employees-under-every-manager/)
  3. [Longest subarray with sum divisible by k](https://www.geeksforgeeks.org/longest-subarray-sum-divisible-k/)
  4. [Find the largest subarray with 0 sum](https://www.geeksforgeeks.org/find-the-largest-subarray-with-0-sum/)
  5. [Longest Increasing consecutive subsequence](https://www.geeksforgeeks.org/longest-increasing-consecutive-subsequence/)
  6. [Count distinct elements in every window of size k](https://www.geeksforgeeks.org/count-distinct-elements-in-every-window-of-size-k/)
  7. [Design a data structure that supports insert, delete, search and getRandom in constant time](https://www.geeksforgeeks.org/design-a-data-structure-that-supports-insert-delete-search-and-getrandom-in-constant-time/)
  8. [Find subarray with given sum | Set 2 (Handles Negative Numbers)](https://www.geeksforgeeks.org/find-subarray-with-given-sum-in-array-of-integers/)
  9. [Implementing our Own Hash Table with Separate Chaining in Java](https://www.geeksforgeeks.org/implementing-our-own-hash-table-with-separate-chaining-in-java/)
  10. [Implementing own Hash Table with Open Addressing Linear Probing in C++](https://www.geeksforgeeks.org/implementing-hash-table-open-addressing-linear-probing-cpp/)
  11. [Minimum insertions to form a palindrome with permutations allowed](https://www.geeksforgeeks.org/minimum-insertions-to-form-a-palindrome-with-permutations-allowed/)
  12. [Maximum possible difference of two subsets of an array](https://www.geeksforgeeks.org/maximum-possible-difference-two-subsets-array/)
  13. [Sorting using trivial hash function](https://www.geeksforgeeks.org/sorting-using-trivial-hash-function/)
  14. [Smallest subarray with k distinct numbers](https://www.geeksforgeeks.org/smallest-subarray-k-distinct-numbers/)
* **Hard:**
  1. [Clone a Binary Tree with Random Pointers](https://www.geeksforgeeks.org/clone-binary-tree-random-pointers/)
  2. [Largest subarray with equal number of 0s and 1s](https://www.geeksforgeeks.org/largest-subarray-with-equal-number-of-0s-and-1s/)
  3. [All unique triplets that sum up to a given value](https://www.geeksforgeeks.org/unique-triplets-sum-given-value/)
  4. [Palindrome Substring Queries](https://www.geeksforgeeks.org/palindrome-substring-queries/)
  5. [Range Queries for Frequencies of array elements](https://www.geeksforgeeks.org/range-queries-for-frequencies-of-array-elements/)
  6. [Elements to be added so that all elements of a range are present in array](https://www.geeksforgeeks.org/elements-to-be-added-so-that-all-elements-of-a-range-are-present-in-array/)
  7. [Cuckoo Hashing – Worst case O(1) Lookup!](https://www.geeksforgeeks.org/cuckoo-hashing/)
  8. [Count subarrays having total distinct elements same as original array](https://www.geeksforgeeks.org/count-subarrays-total-distinct-elements-original-array/)
  9. [Maximum array from two given arrays keeping order same](https://www.geeksforgeeks.org/maximum-array-from-two-given-arrays-keeping-order-same/)
  10. [Find Sum of all unique sub-array sum for a given array.](https://www.geeksforgeeks.org/find-sum-unique-sub-array-sum-given-array/)
  11. [Recaman’s sequence](https://www.geeksforgeeks.org/recamans-sequence/)
  12. [Length of longest strict bitonic subsequence](https://www.geeksforgeeks.org/length-longest-strict-bitonic-subsequence/)
  13. [Find All Duplicate Subtrees](https://www.geeksforgeeks.org/find-duplicate-subtrees/)
  14. [Find if there is a rectangle in binary matrix with corners as 1](https://www.geeksforgeeks.org/find-rectangle-binary-matrix-corners-1/)

**Quick Links :**

* [‘Practice Problems’ on Hashing](https://practice.geeksforgeeks.org/topics/hashing/)
* [Top 20 Hashing Technique based Interview Questions](https://www.geeksforgeeks.org/top-20-hashing-technique-based-interview-questions/)
* [‘Quizzes’ on Hashing](https://www.geeksforgeeks.org/data-structure-gq/hash-gq/)
* [‘Videos’ on Hashing](https://www.youtube.com/playlist?list=PLqM7alHXFySGwXaessYMemAnITqlZdZVE)

**Easy Questions:**

**Find whether an array is subset of another array**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 23 Dec, 2022
* Read
* Discuss(250+)
* Courses
* Practice
* Video

Given two arrays: arr1[0..m-1] and arr2[0..n-1]. Find whether arr2[] is a **subset** of arr1[] or not. Both arrays are **not in sorted order**. It may be assumed that **elements** in both arrays are **distinct**.

**Examples:**

***Input****: arr1[] = {11, 1, 13, 21, 3, 7}, arr2[] = {11, 3, 7, 1}*

***Output****: arr2[] is a subset of arr1[]*

***Input****: arr1[] = {1, 2, 3, 4, 5, 6}, arr2[] = {1, 2, 4}*

***Output****: arr2[] is a subset of arr1[]*

***Input****: arr1[] = {10, 5, 2, 23, 19}, arr2[] = {19, 5, 3}*

***Output****: arr2[] is not a subset of arr1[]*

Recommended Problem

Array Subset of another array

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

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

+4 more

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

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

+1 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/array-subset-of-another-array2317/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 1.6L

**Naive Approach to Find whether an array is subset of another array**

Use**two loops**: The outer loop picks all the elements of arr2[] one by one. The inner loop linearly searches for the element picked by the outer loop. If all elements are found then return 1, else return 0.

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* PHP
* Javascript

# Python 3 program to find whether an array

# is subset of another array

# Return 1 if arr2[] is a subset of

# arr1[]

**def** isSubset(arr1, arr2, m, n):

    i **=** 0

    j **=** 0

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

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

**if**(arr2[i] **==** arr1[j]):

**break**

        # If the above inner loop was

        # not broken at all then arr2[i]

        # is not present in arr1[]

**if** (j **==** m):

**return** 0

    # If we reach here then all

    # elements of arr2[] are present

    # in arr1[]

**return** 1

# Driver code

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

    arr1 **=** [11, 1, 13, 21, 3, 7]

    arr2 **=** [11, 3, 7, 1]

    m **=** len(arr1)

    n **=** len(arr2)

**if**(isSubset(arr1, arr2, m, n)):

        print("arr2[] is subset of arr1[] ")

**else**:

**print**("arr2[] is not a subset of arr1[]")

# This code is contributed by ita\_c

**Output**

arr2[] is subset of arr1[]

**Time Complexity:** O(m\*n)

**Auxiliary Space:** O(1)

**Find whether an array is subset of another array using**[**Sorting**](https://www.geeksforgeeks.org/sorting-algorithms/)**and**[**Binary Search**](https://www.geeksforgeeks.org/binary-search/)

The idea is to sort the given array arr1[], and then for each element in arr2[] do a binary search for it in sorted arr1[]. If the element is **not found** then **return 0**. If all elements are present then **return 1**.

**Illustration:**

*Given array****arr1[] = { 11, 1, 13, 21, 3, 7 }****and****arr2[] = { 11, 3, 7, 1 }****.*

***Step 1:****We will****sort****the array arr1[], and have arr1[] = { 1, 3, 7, 11, 13, 21}.*

***Step 2:****We will look for each element in arr2[] in arr1[] using*[***binary search***](https://www.geeksforgeeks.org/binary-search/)*.*

* *arr2[] = {****11****, 3, 7, 1 }, 11 is present in arr1[] = { 1, 3, 7,****11****, 13, 21}*
* *arr2[] = { 11,****3****, 7, 1 }, 3 is present in arr1[] = { 1,****3****, 7, 11, 13, 21}*
* *arr2[] = { 11, 3,****7****, 1 }, 7 is present in arr1[] = { 1, 3,****7****, 11, 13, 21}*
* *arr2[] = { 11, 3, 7,****1****}, 1 is present in arr1[] = {****1****, 3, 7, 11, 13, 21}*

*As all the elements are found we can conclude arr2[] is the subset of arr1[].*

**Algorithm:**

The algorithm is pretty straightforward.

* Sort the first array arr1[].
* Look for the elements of arr2[] in sorted arr1[].
* If we encounter a particular value that is present in arr2[] but not in arr1[], the code will terminate, arr2[] can never be the subset of arr1[].
* Else arr2[] is the subset of arr1[].

Below is the code implementation of the above approach :

* C++
* C
* Java
* Python3
* C#
* PHP
* Javascript

# Python3 program to find whether an array

# is subset of another array

# Return 1 if arr2[] is a subset of arr1[]

**def** isSubset(arr1, arr2, m, n):

    i **=** 0

    quickSort(arr1, 0, m**-**1)

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

**if** (binarySearch(arr1, 0, m **-** 1, arr2[i]) **== -**1):

**return** 0

    # If we reach here then all elements

    # of arr2[] are present in arr1[]

**return** 1

# FOLLOWING FUNCTIONS ARE ONLY FOR

# SEARCHING AND SORTING PURPOSE

# Standard Binary Search function

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

**if**(high >**=** low):

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

        # Check if arr[mid] is the first

        # occurrence of x.

        # arr[mid] is first occurrence if x is

        # one of the following

        # is true:

        # (i) mid == 0 and arr[mid] == x

        # (ii) arr[mid-1] < x and arr[mid] == x

**if**((mid **==** 0 **or** x > arr[mid**-**1]) **and** (arr[mid] **==** x)):

**return** mid

**elif**(x > arr[mid]):

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

**else**:

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

**return -**1

**def** partition(A, si, ei):

    x **=** A[ei]

    i **=** (si **-** 1)

**for** j **in** range(si, ei):

**if**(A[j] <**=** x):

            i **+=** 1

            A[i], A[j] **=** A[j], A[i]

    A[i **+** 1], A[ei] **=** A[ei], A[i **+** 1]

**return** (i **+** 1)

# Implementation of Quick Sort

# A[] --> Array to be sorted

# si --> Starting index

# ei --> Ending index

**def** quickSort(A, si, ei):

    # Partitioning index

**if**(si < ei):

        pi **=** partition(A, si, ei)

        quickSort(A, si, pi **-** 1)

        quickSort(A, pi **+** 1, ei)

# Driver code

arr1 **=** [11, 1, 13, 21, 3, 7]

arr2 **=** [11, 3, 7, 1]

m **=** len(arr1)

n **=** len(arr2)

**if**(isSubset(arr1, arr2, m, n)):

    print("arr2[] is subset of arr1[] ")

**else**:

    print("arr2[] is not a subset of arr1[] ")

# This code is contributed by chandan\_jnu

**Output**

arr2[] is subset of arr1[]

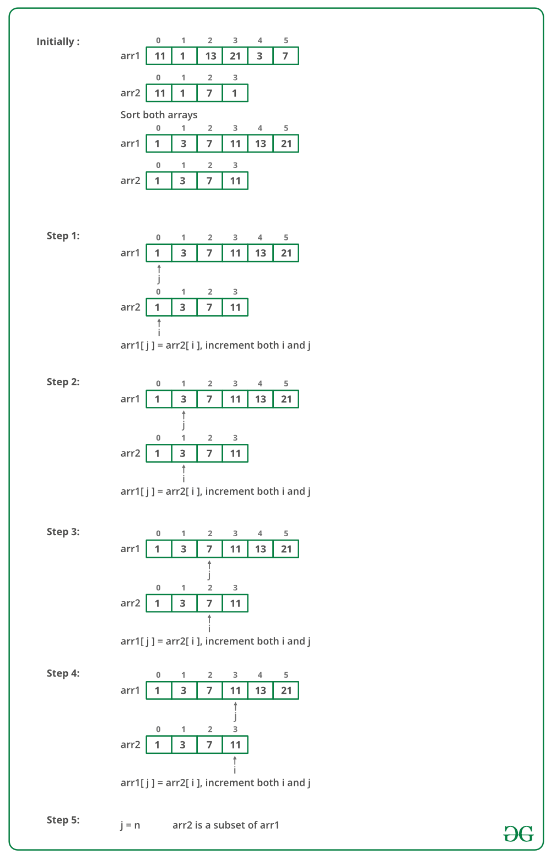
**Time Complexity:** O(mLog(m) + nlog(m)). O(mLog(m)) for sorting and O(nlog(m)) for binary searching each element of one array in another. In the above code, Quick Sort is used and the worst-case time complexity of Quick Sort is O(m2).

**Auxiliary Space:**O(n)

**Find whether an array is subset of another array using**[**Sorting**](https://www.geeksforgeeks.org/sorting-algorithms/)**and**[**Merging**](https://www.geeksforgeeks.org/merge-two-sorted-arrays/)

The idea is to sort the two arrays and then iterate on the second array looking for the same values on the first array using two pointers. Whenever we encounter the same values we will increment both the pointer and if we encounter any values less than that of the second array, we will increment the value of the pointer pointing to the first array. If the value is greater than that of the second array, we know the second array is not the subset of the first array.

**Illustration:**



**Algorithm:**

The initial step will be to sort the two arrays.

* Set two pointers j and i or arr1[] and arr2[] respectively.
* If arr1[j] < arr2[i], we will increase j by 1.
* If arr1[j] = arr2[i], we will increase j and i by 1.
* If arr1[j] > arr2[i], we will terminate as arr2[] is not the subset of arr1[].

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* PHP
* Javascript

# Python3 program to find whether an array

# is subset of another array

# Return 1 if arr2[] is a subset of arr1[] \*/

**def** isSubset(arr1, arr2, m, n):

    i **=** 0

    j **=** 0

**if** m < n:

**return** 0

    arr1.sort()

    arr2.sort()

**while** i < n **and** j < m:

**if** arr1[j] < arr2[i]:

            j **+=** 1

**elif** arr1[j] **==** arr2[i]:

            j **+=** 1

            i **+=** 1

**elif** arr1[j] > arr2[i]:

**return** 0

**return** False **if** i < n **else** True

# Driver code

arr1 **=** [11, 1, 13, 21, 3, 7]

arr2 **=** [11, 3, 7, 1]

m **=** len(arr1)

n **=** len(arr2)

**if** isSubset(arr1, arr2, m, n) **==** True:

**print**("arr2[] is subset of arr1[] ")

**else**:

    printf("arr2[] is not a subset of arr1[] ")

# This code is contributed by Shrikant13

**Output**

arr2[] is subset of arr1[]

**Time Complexity:** O(mLog(m) + nLog(n)) which is better than approach 2.

**Auxiliary Space:**O(1)

Thanks to **Parthsarthi** for suggesting this method.

**Find whether an array is a subset of another array using**[**Hashing**](https://www.geeksforgeeks.org/what-is-hashing-a-complete-tutorial/)

The idea is to insert all the elements of the first array in a HashSet, and then iterate on the second array and find if the element exists in the HashSet, if the HashSet doesn’t contain any particular value then the second array is not the subset of the first array.

**Illustration:**

*Given array****arr1[] = { 11, 1, 13, 21, 3, 7 }****and****arr2[] = { 11, 3, 7, 1 }****.*

***Step 1:****We will store**the array arr1[] elements in****HashSet***

***Step 2:****We will look for each element in arr2[] in arr1[] using*[***binary search***](https://www.geeksforgeeks.org/binary-search/)*.*

* *arr2[] = {****11****, 3, 7, 1 }, 11 is present in the HashSet = { 1, 3, 7,****11****, 13, 21}*
* *arr2[] = { 11,****3****, 7, 1 }, 3 is present in the HashSet = { 1,****3****, 7, 11, 13, 21}*
* *arr2[] = { 11, 3,****7****, 1 }, 7 is present in the HashSet = { 1, 3,****7****, 11, 13, 21}*
* *arr2[] = { 11, 3, 7,****1****}, 1 is present in the HashSet = {****1****, 3, 7, 11, 13, 21}*

*As all the elements are found we can conclude arr2[] is the subset of arr1[].*

**Algorithm:**

The algorithm is pretty straightforward.

* Store the first array arr1[] in a **HashSet**.
* Look for the elements of arr2[] in the **HashSet**.
* If we encounter a particular value that is present in arr2[] but not in the **HashSet**, the code will terminate, arr2[] can never be the subset of arr1[].
* Else arr2[] is the subset of arr1[].

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find whether an array

# is subset of another array

# Return true if arr2[] is a subset

# of arr1[]

**def** isSubset(arr1, m, arr2, n):

    # Using STL set for hashing

    hashset **=** set()

    # hset stores all the values of arr1

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

        hashset.add(arr1[i])

    # Loop to check if all elements

    # of arr2 also lies in arr1

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

**if** arr2[i] **in** hashset:

**continue**

**else**:

**return** False

**return** True

# Driver Code

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

    arr1 **=** [11, 1, 13, 21, 3, 7]

    arr2 **=** [11, 3, 7, 1]

    m **=** len(arr1)

    n **=** len(arr2)

**if** (isSubset(arr1, m, arr2, n)):

        print("arr2[] is subset of arr1[] ")

**else**:

        print("arr2[] is not a subset of arr1[] ")

# This code is contributed by akhilsaini

**Output**

arr2[] is subset of arr1[]

**Time Complexity:**O(m+n\*logm)

**Auxiliary Space:**O(m)

**Find whether an array is a subset of another array using**[**Set**](https://www.geeksforgeeks.org/set-in-cpp-stl/)

The idea is to insert all the elements of the first array and second array in the set, if the size of the set is equal to the size of arr1[] then the arr2[] is the subset of arr1[]. As no new elements are found in arr2[] hence is the subset.

**Illustration:**

*Given array****arr1[] = { 11, 1, 13, 21, 3, 7 }****and****arr2[] = { 11, 3, 7, 1 }****.*

***Step 1:****We will store**the array arr1[] and arr2[] elements in****Set***

* *The final****Set****= { 1, 3, 7, 11, 13, 21}*

***Step 2:****Size of arr1[] = 6 and size of the Set = 6*

* *Hence no new elements are found in arr2[]*

*As all the elements are found we can conclude arr2[] is the subset of arr1[].*

**Algorithm:**

The algorithm is pretty straightforward.

* Store the first array arr1[] in a **Set**.
* Store the first array arr1[] in the same**Set**.
* If the size of arr1[] = size of the Set, arr2[] is the subset of arr1[].
* Else arr2[] is not the subset of arr1[].
* C++
* Java
* Python3
* C#
* Javascript

# Python3 code

arr1 **=** [11, 1, 13, 21, 3, 7]

arr2 **=** [11, 3, 7, 1]

m **=** len(arr1)

n **=** len(arr2)

s **=** set()

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

    s.add(arr1[i])

p **=** len(s)

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

    s.add(arr2[i])

**if** (len(s) **==** p):

**print**("arr2[] is subset of arr1[] ")

**else**:

    print("arr2[] is not subset of arr1[] ")

    # This code is contributed by divyeshrabadiya07.

**Output**

arr2[] is subset of arr1[]

**Time Complexity:** O(m+n) because we are using unordered\_set and inserting in it, If we would be using an ordered set inserting would have taken log n increasing the TC to O(mlogm+nlogn), but order does not matter in this approach.

**Auxiliary Space:**O(n+m)

**Find whether an array is a subset of another array using the Frequency Table**

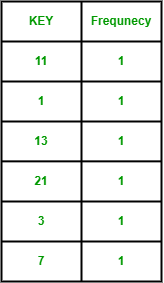
The idea is to store the frequency of the elements present in the first array, then look for the elements present in arr2[] in the frequency array. As no new elements are found in arr2[] hence is the subset.

**Illustration:**

*Given array****arr1[] = { 11, 1, 13, 21, 3, 7 }****and****arr2[] = { 11, 3, 7, 1 }****.*

***Step 1:****We will store**the array arr1[] elements frequency in the frequency array*

* *The frequency array**will look like this*



*frequency array*

***Step 2:****We will look for arr2[] elements in the frequency array.*

* *arr2[] = {****11****, 3, 7, 1 }, 11 is present in the frequency array*
* *arr2[] = { 11,****3****, 7, 1 }, 3 is present in the frequency array*
* *arr2[] = { 11, 3,****7****, 1 }, 7 is present in the frequency array*
* *arr2[] = { 11, 3, 7,****1****}, 1 is present in the frequency array*

*As all the elements are found we can conclude arr2[] is the subset of arr1[].*

**Algorithm:**

The algorithm is pretty straightforward.

* Store the frequency of the first array elements of arr1[] in the frequency array.
* Iterate on the arr2[] and look for its elements in the frequency array.
* If the value is found in the frequency array reduce the frequency value by one.
* If for any elements in arr2[] frequency is less than 1, we will conclude arr2[] is not the subset of arr1[],

Below is the implementation of the above approach:

* C++14
* Java
* Python3
* C#
* Javascript

# Python3 program to find whether an array

# is subset of another array

# Return true if arr2[] is a subset of arr1[]

**def** isSubset(arr1, m, arr2, n):

    # Create a Frequency Table using STL

    frequency **=** {}

    # Increase the frequency of each element

    # in the frequency table.

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

**if** arr1[i] **in** frequency:

            frequency[arr1[i]] **=** frequency[arr1[i]] **+** 1

**else**:

            frequency[arr1[i]] **=** 1

    # Decrease the frequency if the

    # element was found in the frequency

    # table with the frequency more than 0.

    # else return 0 and if loop is

    # completed return 1.

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

**if** (frequency[arr2[i]] > 0):

            frequency[arr2[i]] **-=** 1

**else**:

**return** False

**return** True

# Driver Code

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

    arr1 **=** [11, 1, 13, 21, 3, 7]

    arr2 **=** [11, 3, 7, 1]

    m **=** len(arr1)

    n **=** len(arr2)

**if** (isSubset(arr1, m, arr2, n)):

        print("arr2[] is subset of arr1[] ")

**else**:

**print**("arr2[] is not a subset of arr1[] ")

# This code is contributed by akhilsaini

**Output**

arr2[] is subset of arr1[]

**Time Complexity:**O(m+n) which is better than methods 1,2,3

**Auxiliary Space:**O(n)

**Note** that method 1, method 2, method 4, and method 5 don’t handle the cases when we have duplicates in arr2[]. For example, {1, 4, 4, 2} is not a subset of {1, 4, 2}, but these methods will print it as a subset.

*From <*[*https://www.geeksforgeeks.org/find-whether-an-array-is-subset-of-another-array-set-1/*](https://www.geeksforgeeks.org/find-whether-an-array-is-subset-of-another-array-set-1/)*>*

**Union and Intersection of two Linked List using Hashing**

* Difficulty Level : [Basic](https://www.geeksforgeeks.org/basic/)
* Last Updated : 15 Dec, 2022
* Read
* Discuss
* Courses
* Practice
* Video

Given two Linked Lists, create union and intersection lists that contain union and intersection of the elements present in the given lists. Order of elements in output lists doesn’t matter. **Examples:**

**Input:**  
 List1: 10 -> 15 -> 4 -> 20  
 List2: 8 -> 4 -> 2 -> 10  
**Output:**  
 Intersection List: 4 -> 10  
 Union List: 2 -> 8 -> 20 -> 4 -> 15 -> 10  
**Explanation:** In this two lists 4 and 10 nodes   
are common. The union lists contains   
all the nodes of both the lists.

**Input:**  
 List1: 1 -> 2 -> 3 -> 4  
 List2: 3 -> 4 -> 8 -> 10  
**Output:**  
 Intersection List: 3 -> 4  
 Union List: 1 -> 2 -> 3 -> 4 -> 8 -> 10  
**Explanation:** In this two lists 4 and 3 nodes   
are common. The union lists contains   
all the nodes of both the lists.

Recommended Problem

Intersection of Two Linked Lists

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

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

+3 more

[24\*7 Innovation Labs](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=24*7%20Innovation%20Labs&sortBy=submissions)

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

+8 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/intersection-of-two-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 44.2K

We have already discussed [Method-1](https://www.geeksforgeeks.org/union-and-intersection-of-two-linked-lists/) and [Method-2](https://www.geeksforgeeks.org/union-intersection-two-linked-lists-set-2-using-merge-sort/) of this question. In this post, its Method-3 (Using Hashing) is discussed with a Time Complexity of O(m+n) i.e. better than both methods discussed earlier.

**Implementation:**  
1- Start traversing both the lists.  
 a) Store the current element of both lists  
 with its occurrence in the map.  
2- For Union: Store all the elements of the map   
 in the resultant list.  
3- For Intersection: Store all the elements only   
 with an occurrence of 2 as 2 denotes that   
 they are present in both the lists.

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

Below is the C++ implementation of the above steps.

* CPP
* Java
* Python3

# Python code for finding union and intersection of linkedList

**class** linkedList:

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

        self.head **=** None

        self.tail **=** None

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

**if** self.head **is** None:

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

            self.tail **=** self.head

**else**:

            self.tail.next **=** Node(data)

            self.tail **=** self.tail.next

**class** Node:

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

        self.data **=** data

        self.next **=** None

# return the head of new list containing the intersection of 2 linkedList

**def** findIntersection(head1, head2):

    # creating a map

    hashmap **=** {}

    # traversing on first list

**while**(head1 !**=** None):

        data **=** head1.data

**if**(data **not in** hashmap.keys()):

            hashmap[data] **=** 1

        head1 **=** head1.next

    # making a new linkedList

    ans **=** linkedList()

**while**(head2 !**=** None):

        data **=** head2.data

**if**(data **in** hashmap.keys()):

            # adding data to new list

            ans.insert(data)

        head2 **=** head2.next

**return** ans.head

# return the head of new list containing the union of 2 linkedList

**def** union(head1, head2):

    # creating a map

    hashmap **=** {}

    # traversing on first list

**while**(head1 !**=** None):

        data **=** head1.data

**if**(data **not in** hashmap.keys()):

            hashmap[data] **=** 1

        head1 **=** head1.next

**while**(head2 !**=** None):

        data **=** head2.data

**if**(data **not in** hashmap.keys()):

            hashmap[data] **=** 1

        head2 **=** head2.next

    # making a new linkedList

    ans **=** linkedList()

    # traverse on hashmap

**for** key, value **in** hashmap.items():

        ans.insert(key)

**return** ans.head

**def** printList(head):

**while** head:

**print**(head.data, end**=**' ')

        head **=** head.next

    print()

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

    # first list

    ll1 **=** linkedList()

    ll1.insert(1)

    ll1.insert(2)

    ll1.insert(3)

    ll1.insert(4)

    ll1.insert(5)

    # second list

    ll2 **=** linkedList()

    ll2.insert(1)

    ll2.insert(3)

    ll2.insert(5)

    ll2.insert(6)

**print**("First list is ")

    printList(ll1.head)

    print("Second list is ")

    printList(ll2.head)

    print("Intersection list is")

    printList(findIntersection(ll1.head, ll2.head))

**print**("Union list is ")

    printList(union(ll1.head, ll2.head))

# This code is contributed by Arpit Jain

**Output:**

First list is   
5 4 3 2 1   
Second list is   
6 5 3 1   
Intersection list is   
3 5 1   
Union list is   
3 4 6 5 2 1

We can also handle the case of duplicates by maintaining separate Hash for both the lists. **Complexity Analysis:**

* **Time Complexity:** O(m+n). Here ‘m’ and ‘n’ are number of elements present in first and second lists respectively. **Reason:** For Union: Traverse both the lists, store the elements in Hash-map and update the respective count. For Intersection: Check if count of an element in hash-map is ‘2’.
* **Auxiliary Space:** O(m+n). Use of Hash-map data structure for storing values.

*From <*[*https://www.geeksforgeeks.org/union-and-intersection-of-two-linked-list-using-hashing/*](https://www.geeksforgeeks.org/union-and-intersection-of-two-linked-list-using-hashing/)*>*

**Check if a pair exists with given sum in given array**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 20 Dec, 2022
* Read
* Discuss(80+)
* Courses
* Practice
* Video

Given an array A[] of n numbers and another number x, the task is to check whether or not there exist two elements in A[] whose sum is exactly x.

**Examples:**

***Input:****arr[] = {0, -1, 2, -3, 1}, x= -2*

***Output:****Yes*

***Explanation:****If we calculate the sum of the output,1 + (-3) = -2*

***Input:****arr[] = {1, -2, 1, 0, 5}, x = 0*

***Output:****No*

Recommended Problem

Key Pair

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

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

+1 more

[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)

+11 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/key-pair5616/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 1.6L

**Naive Approach:** The basic approach to solve this problem is by nested traversal.

* Traverse the array using a loop
* For each element:
* Check if there exists another in the array with sum as x
* Return true if yes, else continue
* If no such pair is found, return false.

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript
* Go

# This python program tells if there exists a pair in array whose sum results in x.

# Function to find and print pair

**def** chkPair(A, size, x):

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

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

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

**return** 1

**return** 0

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

    A **=** [0, **-**1, 2, **-**3, 1]

    x **= -**2

    size **=** len(A)

**if** (chkPair(A, size, x)):

        print("Yes")

**else**:

**print**("No")

    # This code is contributed by rakeshsahni

**Output**

Yes

**Time Complexity:** O(N2), Finding pair for every element in the array of size N.

**Auxiliary Space:**O(1)

**Two Sum using Sorting and**[Two-Pointers technique](https://www.geeksforgeeks.org/two-pointers-technique/)**:**

*The idea is to use the two-pointer technique. But for using the two-pointer technique, the array must be sorted. Once the array is sorted the two pointers can be taken which mark the beginning and end of the array respectively. If the sum is****greater****than the sum of those two elements, shift the right pointer to decrease the value of the required sum and if the sum is****lesser****than the required value, shift the left pointer to increase the value of the required sum.*

**Illustration:**

*Let an array be {1, 4, 45, 6, 10, -8} and sum to find be 16*

*After sorting the array*

*A = {-8, 1, 4, 6, 10, 45}*

*Now, increment ‘l’ when the sum of the pair is less than the required sum and decrement ‘r’ when the sum of the pair is more than the required sum.*

*This is because when the sum is less than the required sum then to get the number which could increase the sum of pair, start moving from left to right(also sort the array) thus “l++” and vice versa.*

*Initialize l = 0, r = 5*

*A[l] + A[r] ( -8 + 45) > 16 => decrement r. Now r = 4*

*A[l] + A[r] ( -8 + 10) increment l. Now l = 1*

*A[l] + A[r] ( 1 + 10) increment l. Now l = 2*

*A[l] + A[r] ( 4 + 10) increment l. Now l = 3*

*A[l] + A[r] ( 6 + 10) == 16 => Found candidates (return 1)*

**Note:** If there is more than one pair having the given sum then this algorithm reports only one. Can be easily extended for this though.

Follow the steps below to solve the problem:

* hasArrayTwoCandidates (A[], ar\_size, sum)
* Sort the array in non-decreasing order.
* Initialize two index variables to find the candidate   
  elements in the sorted array.
* Initialize first to the leftmost index: l = 0
* Initialize second the rightmost index: r = ar\_size-1
* Loop while l < r.
* If (A[l] + A[r] == sum) then return 1
* Else if( A[l] + A[r] < sum ) then l++
* Else r–
* No candidates in the whole array – return 0

Below is the implementation of the above approach:

* C++
* C
* Java
* Python
* C#
* PHP
* Javascript

# Python program to check for the sum

# condition to be satisfied

**def** hasArrayTwoCandidates(A, arr\_size, sum):

    # sort the array

    quickSort(A, 0, arr\_size**-**1)

    l **=** 0

    r **=** arr\_size**-**1

    # traverse the array for the two elements

**while** l < r:

**if** (A[l] **+** A[r] **==** sum):

**return** 1

**elif** (A[l] **+** A[r] < sum):

            l **+=** 1

**else**:

            r **-=** 1

**return** 0

# Implementation of Quick Sort

# A[] --> Array to be sorted

# si  --> Starting index

# ei  --> Ending index

**def** quickSort(A, si, ei):

**if** si < ei:

        pi **=** partition(A, si, ei)

        quickSort(A, si, pi**-**1)

        quickSort(A, pi **+** 1, ei)

# Utility function for partitioning

# the array(used in quick sort)

**def** partition(A, si, ei):

    x **=** A[ei]

    i **=** (si**-**1)

**for** j **in** range(si, ei):

**if** A[j] <**=** x:

            i **+=** 1

            # This operation is used to swap

            # two variables is python

            A[i], A[j] **=** A[j], A[i]

        A[i **+** 1], A[ei] **=** A[ei], A[i **+** 1]

**return** i **+** 1

# Driver program to test the functions

A **=** [1, 4, 45, 6, 10, **-**8]

n **=** 16

**if** (hasArrayTwoCandidates(A, len(A), n)):

    print("Yes")

**else**:

    print("No")

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

**Output**

Yes

**Time Complexity:** O(NlogN), Time complexity for sorting the array

**Auxiliary Space:** O(1)

**Two Sum using**[Binary Search](https://www.geeksforgeeks.org/binary-search/)**:**

*Sort the array, then traverse the array elements and perform binary search for (target – a[i]) on the remaining part*

Follow the below steps to solve the problem:

* Sort the array in non-decreasing order.
* Traverse from **0** to **N-1**
* Initialize **searchKey** = **sum – A[i]**
* If(binarySearch(searchKey, A, i + 1, N) == **True**
* Return **True**
* Return **False**

Below is the implementation of the above approach:

* C++
* Java
* Python
* C#
* Javascript

# Python program to check for the sum

# condition to be satisfied

**def** binarySearch(A, low, high, searchKey):

    m **=** 0

**while** (low <**=** high):

        m **=** (high **+** low) **//** 2

        # Check if searchKey is present at mid

**if** (A[m] **==** searchKey):

**return** 1

        # If searchKey greater, ignore left half

**if** (A[m] < searchKey):

            low **=** m **+** 1

        # If searchKey is smaller, ignore right half

**else**:

            high **=** m **-** 1

    # if we reach here, then element was

    # not present

**return** 0

**def** checkTwoSum(A, arr\_size, sum):

    # sort the array

    A.sort()

    l **=** 0

    r **=** arr\_size**-**1

    #  Traversing all element in an array search for searchKey

    i **=** 0

**while** i < arr\_size**-**1:

        searchKey **=** sum**-**A[i]

        # calling binarySearch function

**if**(binarySearch(A, i**+**1, r, searchKey) **==** 1):

**return** 1

        i **=** i**+**1

**return** 0

# Driver program to test the functions

A **=** [1, 4, 45, 6, 10, **-**8]

n **=** 14

**if** (checkTwoSum(A, len(A), n)):

    print("Yes")

**else**:

**print**("No")

**Output**

Yes

Time Complexity: O(NlogN)

Auxiliary Space: O(1)

**Two Sum using**[Hashing](http://www.geeksforgeeks.org/hashing-data-structure/)**:**

This problem can be solved efficiently by using the technique of hashing. Use a **hash\_map** to check for the current array value **x(let)**, if there exists a value **target\_sum-x** which on adding to the former gives **target\_sum**. This can be done in constant time.

**Illustration:**

*arr[] = {0, -1, 2, -3, 1}*

*sum = -2*

*Now start traversing:*

*Step 1: For ‘0’ there is no valid number ‘-2’ so store ‘0’ in hash\_map.*

*Step 2: For ‘-1’ there is no valid number ‘-1’ so store ‘-1’ in hash\_map.*

*Step 3: For ‘2’ there is no valid number ‘-4’ so store ‘2’ in hash\_map.*

*Step 4: For ‘-3’ there is no valid number ‘1’ so store ‘-3’ in hash\_map.*

*Step 5: For ‘1’ there is a valid number ‘-3’ so answer is 1, -3*

*unordered\_set s*

*for(i=0 to end)*

*if(s.find(target\_sum – arr[i]) == s.end)*

*insert(arr[i] into s)*

*else*

*print arr[i], target-arr[i]*

Follow the steps below to solve the problem:

* Initialize an empty hash table s.
* Do the following for each element A[i] in A[]
* If s[x – A[i]] is set then print the pair (A[i], x – A[i])
* Insert A[i] into s.

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python program to find if there are

# two elements with given sum

# function to check for the given sum

# in the array

**def** printPairs(arr, arr\_size, sum):

    # Create an empty hash map

    # using an hashmap allows us to store the indices

    hashmap **=** {}

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

        temp **=** sum**-**arr[i]

**if** (temp **in** hashmap):

**print**('Yes')

**return**

        hashmap[arr[i]] **=** i

    print("No")

# driver code

A **=** [1, 4, 45, 6, 10, 8]

n **=** 16

printPairs(A, len(A), n)

# This code will also work in case the array has the same number twice

# and target is the sum of those numbers

# Eg: Array = [4,6,4] Target = 8

# This code is contributed by \_\_Achyut Upadhyay\_\_

**Output**

Yes

**Time Complexity:**O(N), As the whole array is needed to be traversed only once.

**Auxiliary Space:** O(N), A hash map has been used to store array elements.

**Note:** The solution will work even if the range of numbers includes negative numbers + if the pair is formed by numbers recurring twice in array eg: array = [3,4,3]; pair = (3,3); target sum = 6.

**Two Sum Using remainders of the elements less than x:**

*The idea is to count the elements with remainders when divided by x, i.e****0 to x-1****, each remainder separately. Suppose we have****x as 6****, then the numbers which are less than 6 and have remainders which add up to 6 gives sum as 6 when added. For example, we have elements, 2,4 in the array and 2%6 = 2 and 4%6 =4, and these remainders add up to give 6. Like that we have to check for pairs with remainders (1,5),(2,4),(3,3). if we have one or more elements with remainder 1 and one or more elements with remainder 5, then surely we get a sum as 6. Here we do not consider (0,6) as the elements for the resultant pair should be less than 6. when it comes to (3,3) we have to check if we have two elements with remainder 3, then we can say that “There exists a pair whose sum is x”.*

Follow the steps below to solve the problem:

* 1. Create an array with size x.
* 2. Initialize all rem elements to zero.
* 3. Traverse the given array
* Do the following if arr[i] is less than x:
* r=arr[i]%x which is done to get the remainder.
* rem[r]=rem[r]+1 i.e. increasing the count of elements that have remainder r when divided with x.
* 4. Now, traverse the rem array from 1 to x/2.
* If(rem[i]> 0 and rem[x-i]>0) then print “YES” and come out of the loop. This means that we have a pair that results in x upon doing.
* 5. Now when we reach at x/2 in the above loop
* If x is even, for getting a pair we should have two elements with remainder x/2.
* If rem[x/2]>1 then print “YES” else print “NO”
* If it is not satisfied that is x is odd, it will have a separate pair with x-x/2.
* If rem[x/2]>0 and rem[x-x/2]>0 , then print “Yes” else, print”No”;

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript

# Code in Python3 to tell if there

# exists a pair in array whose

# sum results in x.

# Function to print pairs

**def** printPairs(a, n, x):

    rem **=** []

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

        # Initializing the rem

        # values with 0's.

        rem.append(0)

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

**if** (a[i] < x):

            # Perform the remainder operation

            # only if the element is x, as

            # numbers greater than x can't

            # be used to get a sum x.Updating

            # the count of remainders.

            rem[a[i] **%** x] **+=** 1

    # Traversing the remainder list from

    # start to middle to find pairs

**for** i **in** range(1, x **//** 2):

**if** (rem[i] > 0 **and** rem[x **-** i] > 0):

            # The elements with remainders

            # i and x-i will result to a

            # sum of x. Once we get two

            # elements which add up to x,

            # we print x and break.

**print**("Yes")

**break**

    # Once we reach middle of

    # remainder array, we have to

    # do operations based on x.

**if** (i >**=** x **//** 2):

**if** (x **%** 2 **==** 0):

**if** (rem[x **//** 2] > 1):

                # If x is even and we have more

                # than 1 elements with remainder

                # x/2, then we will have two

                # distinct elements which add up

                # to x. if we dont have than 1

                # element, print "No".

                print("Yes")

**else**:

                print("No")

**else**:

            # When x is odd we continue

            # the same process which we

            # did in previous loop.

**if** (rem[x **//** 2] > 0 **and**

                    rem[x **-** x **//** 2] > 0):

**print**("Yes")

**else**:

                print("No")

# Driver Code

A **=** [1, 4, 45, 6, 10, 8]

n **=** 16

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

# Function calling

printPairs(A, arr\_size, n)

# This code is contributed by subhammahato348

**Output**

Yes

**Time Complexity:**O(N+X), Traversing over the array of size N and Checking for remainders til X

**Auxiliary Space:**O(X), Space for storing remainders

*From <*[*https://www.geeksforgeeks.org/given-an-array-a-and-a-number-x-check-for-pair-in-a-with-sum-as-x/*](https://www.geeksforgeeks.org/given-an-array-a-and-a-number-x-check-for-pair-in-a-with-sum-as-x/)*>*

**Maximum distance between two occurrences of same element in array**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 06 Jul, 2022
* Read
* Discuss
* Courses
* Practice
* Video

Given an array with repeated elements, the task is to find the maximum distance between two occurrences of an element.

**Examples:**

Input : arr[] = {3, 2, 1, 2, 1, 4, 5, 8, 6, 7, 4, 2}  
Output: 10  
// maximum distance for 2 is 11-1 = 10   
// maximum distance for 1 is 4-2 = 2   
// maximum distance for 4 is 10-5 = 5

Recommended Problem

Max distance between same elements

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

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

+1 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/max-distance-between-same-elements/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 36.5K

A **simple solution** for this problem is to, one by one, pick each element from the array and find its **first** and **last** occurrence in the array and take the difference between the first and last occurrence for maximum distance. The **time complexity** for this approach is**O(n2).**

An **efficient solution** to this problem is to use hashing. The idea is to traverse the input array and store the index of the first occurrence in a hash map. For every other occurrence, find the difference between the index and the first index stored in the hash map. If the difference is more than the result so far, then update the result.

Below are implementations of the idea. The implementation uses [unordered\_map in](https://www.geeksforgeeks.org/unordered_map-in-stl-and-its-applications/).

* C++
* Java
* Python3
* C#
* Javascript

# Python program to find maximum distance between two

# same occurrences of a number.

# Function to find maximum distance between equal elements

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

    # Used to store element to first index mapping

    mp **=** {}

    # Traverse elements and find maximum distance between

    # same occurrences with the help of map.

    maxDict **=** 0

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

        # If this is first occurrence of element, insert its

        # index in map

**if** arr[i] **not in** mp.keys():

            mp[arr[i]] **=** i

        # Else update max distance

**else**:

            maxDict **=** max(maxDict, i**-**mp[arr[i]])

**return** maxDict

# Driver Program

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

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

    n **=** len(arr)

    print (maxDistance(arr, n))

# Contributed By: Harshit Sidhwa

**Output**

10

**Time complexity : O(n)**under the assumption that unordered\_map’s search and insert operations take O(1) time.

**Auxiliary Space : O(n).**

*From <*[*https://www.geeksforgeeks.org/maximum-distance-two-occurrences-element-array/*](https://www.geeksforgeeks.org/maximum-distance-two-occurrences-element-array/)*>*

**Count maximum points on same line**

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 24 Aug, 2022
* Read
* Discuss(50+)
* Courses
* Practice
* Video

Given N point on a 2D plane as pair of (x, y) co-ordinates, we need to find maximum number of point which lie on the same line.

**Examples:**

Input : points[] = {-1, 1}, {0, 0}, {1, 1},   
 {2, 2}, {3, 3}, {3, 4}   
Output : 4  
Then maximum number of point which lie on same  
line are 4, those point are {0, 0}, {1, 1}, {2, 2},  
{3, 3}

Recommended Practice

[Hit most Balloons](https://practice.geeksforgeeks.org/problems/4e75764f8f1638eb4f1c5478ca1986043e15e39a/1/)

[Try It!](https://practice.geeksforgeeks.org/problems/4e75764f8f1638eb4f1c5478ca1986043e15e39a/1/)

We can solve above problem by following approach – For each point p, calculate its slope with other points and use a map to record how many points have same slope, by which we can find out how many points are on same line with p as their one point. For each point keep doing the same thing and update the maximum number of point count found so far.

Some things to note in implementation are:

1. if two point are (x1, y1) and (x2, y2) then their slope will be (y2 – y1) / (x2 – x1) which can be a double value and can cause precision problems. To get rid of the precision problems, we treat slope as pair ((y2 – y1), (x2 – x1)) instead of ratio and reduce pair by their gcd before inserting into map. In below code points which are vertical or repeated are treated separately.
2. If we use [unordered\_map in c++](https://www.geeksforgeeks.org/unordered_map-in-stl-and-its-applications/) or [HashMap in Java](https://www.geeksforgeeks.org/hashmap-treemap-java/) for storing the slope pair, then total time complexity of solution will be O(n^2) and space complexity will be O(n).

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# python3 program to find maximum number of 2D points that lie on the same line.

**from** collections **import** defaultdict

**from** math **import** gcd

**from** typing **import** DefaultDict, List, Tuple

IntPair **=** Tuple[int, int]

**def** normalized\_slope(a: IntPair, b: IntPair) **-**> IntPair:

    """

    Returns normalized (rise, run) tuple. We won't return the actual rise/run

    result in order to avoid floating point math, which leads to faulty

    comparisons.

    See

<https://en.wikipedia.org/wiki/Floating-point_arithmetic#Accuracy_problems>

    """

    run **=** b[0] **-** a[0]

    # normalize undefined slopes to (1, 0)

**if** run **==** 0:

**return** (1, 0)

    # normalize to left-to-right

**if** run < 0:

        a, b **=** b, a

        run **=** b[0] **-** a[0]

    rise **=** b[1] **-** a[1]

    # Normalize by greatest common divisor.

    # math.gcd only works on positive numbers.

    gcd\_ **=** gcd(abs(rise), run)

**return** (

        rise **//** gcd\_,

        run **//** gcd\_,

    )

**def** maximum\_points\_on\_same\_line(points: List[List[int]]) **-**> int:

    # You need at least 3 points to potentially have non-collinear points.

    # For [0, 2] points, all points are on the same line.

**if** len(points) < 3:

**return** len(points)

    # Note that every line we find will have at least 2 points.

    # There will be at least one line because len(points) >= 3.

    # Therefore, it's safe to initialize to 0.

    max\_val **=** 0

**for** a\_index **in** range(0, len(points) **-** 1):

        # All lines in this iteration go through point a.

        # Note that lines a-b and a-c cannot be parallel.

        # Therefore, if lines a-b and a-c have the same slope, they're the same

        # line.

        a **=** tuple(points[a\_index])

        # Fresh lines already have a, so default=1

        slope\_counts: DefaultDict[IntPair, int] **=** defaultdict(**lambda**: 1)

**for** b\_index **in** range(a\_index **+** 1, len(points)):

            b **=** tuple(points[b\_index])

            slope\_counts[normalized\_slope(a, b)] **+=** 1

        max\_val **=** max(

            max\_val,

            max(slope\_counts.values()),

        )

**return** max\_val

print(maximum\_points\_on\_same\_line([

    [**-**1, 1],

    [0, 0],

    [1, 1],

    [2, 2],

    [3, 3],

    [3, 4],

]))

# This code is contributed by Jose Alvarado Torre

**Output**

4

**Time Complexity:** **O(n2logn),** where n denoting length of string.

**Auxiliary Space: O(n).**

*From <*[*https://www.geeksforgeeks.org/count-maximum-points-on-same-line/*](https://www.geeksforgeeks.org/count-maximum-points-on-same-line/)*>*

**Most frequent element in an array**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 19 Oct, 2022
* Read
* Discuss
* Courses
* Practice
* Video

Given an array, find the most frequent element in it. If there are multiple elements that appear a maximum number of times, print any one of them.

**Examples:**

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

***Output :****1*

***Explanation:****1 appears three times in array which is maximum frequency.*

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

***Output :****20*

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

A **simple solution**is to run two loops. The outer loop picks all elements one by one. The inner loop finds the frequency of the picked element and compares it with the maximum so far.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find the most

# frequent element in an array.

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

  maxcount **=** 0;

  element\_having\_max\_freq **=** 0;

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

    count **=** 0

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

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

        count **+=** 1

**if**(count > maxcount):

      maxcount **=** count

      element\_having\_max\_freq **=** arr[i]

**return** element\_having\_max\_freq;

# Driver Code

arr **=** [40,50,30,40,50,30,30]

n **=** len(arr)

print(mostFrequent(arr, n))

# This code is contributed by Arpit Jain

**Output**

30

**The time complexity**of this solution is **O(n2)** since 2 loops are running from i=0 to i=n we can improve its time complexity by taking a visited  array and skipping numbers for which we already calculated the frequency.

**Auxiliary space**: O(1) as it is using constant space for variables

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

A **better solution** is to do the sorting. We first sort the array, then linearly traverse the array.

**Implementation:**

* C++
* C
* Java
* Python3
* C#
* PHP
* Javascript

# Python3 program to find the most

# frequent element in an array.

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

    # Sort the array

    arr.sort()

    # find the max frequency using

    # linear traversal

    max\_count **=** 1

    res **=** arr[0]

    curr\_count **=** 1

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

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

            curr\_count **+=** 1

**else**:

            curr\_count **=** 1

         # If last element is most frequent

**if** (curr\_count > max\_count):

            max\_count **=** curr\_count

            res **=** arr[i **-** 1]

**return** res

# Driver Code

arr **=** [40,50,30,40,50,30,30]

n **=** len(arr)

print(mostFrequent(arr, n))

# This code is contributed by Smitha Dinesh Semwal.

**Output**

30

**Time Complexity: O(nlog(n))**

**Auxiliary Space: O(1)**

An **efficient solution** is to use hashing. We create a hash table and store elements and their frequency counts as key-value pairs. Finally, we traverse the hash table and print the key with the maximum value.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find the most

# frequent element in an array.

**import** math as mt

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

    # Insert all elements in Hash.

    Hash **=** dict()

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

**if** arr[i] **in** Hash.keys():

            Hash[arr[i]] **+=** 1

**else**:

            Hash[arr[i]] **=** 1

    # find the max frequency

    max\_count **=** 0

    res **= -**1

**for** i **in** Hash:

**if** (max\_count < Hash[i]):

            res **=** i

            max\_count **=** Hash[i]

**return** res

# Driver Code

arr **=** [ 40,50,30,40,50,30,30]

n **=** len(arr)

**print**(mostFrequent(arr, n))

# This code is contributed

# by Mohit kumar 29

**Output**

30

**Time Complexity:** **O(n)**

**Auxiliary Space: O(n)**

An efficient solution to this problem can be to solve this problem by Moore’s voting Algorithm.

**NOTE:**THE ABOVE VOTING ALGORITHM ONLY WORKS WHEN THE MAXIMUM OCCURRING ELEMENT IS MORE THAN (SIZEOFARRAY/2) TIMES;

In this method, we will find the maximum occurred integer by counting the votes a number has.

* C++
* Java
* Python3
* C#
* Javascript

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

    # Using moore's voting algorithm

    res **=** 0

    count **=** 1

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

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

            count **+=** 1

**else**:

            count **-=** 1

**if** (count **==** 0):

            res **=** i

            count **=** 1

**return** arr[res]

# Driver code

arr **=** [ 40, 50, 30, 40, 50, 30, 30 ]

n **=** len(arr)

freq **=**  maxFreq(arr, n)

count **=** 0

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

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

            count **+=** 1

**print**("Element ", maxFreq(arr , n),

      " occurs ", count, " times")

# This code is contributed by shivanisinghss2110

**Output**

Element 30 occurs 3 times

**Time Complexity:** **O(n)**

**Auxiliary Space: O(1)**

*From <*[*https://www.geeksforgeeks.org/frequent-element-array/*](https://www.geeksforgeeks.org/frequent-element-array/)*>*

**Find the only repetitive element between 1 to N-1**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 09 Dec, 2022
* Read
* Discuss(20+)
* Courses
* Practice
* Video

Given an array of size **N** filled with numbers from 1 to N-1 in random order. The array has only one repetitive element. The task is to find the repetitive element.

**Examples:**

***Input:****a[] = {1, 3, 2, 3, 4}*

***Output:****3*

***Explanation:****The number 3 is the only repeating element.*

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

***Output:****1*

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

*Use two nested loops. The outer loop traverses through all elements and the inner loop check if the element picked by the outer loop appears anywhere else.*

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find the only

# repeating element in an array where

# elements are from 1 to N-1.

**def** findRepeating(arr, N):

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

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

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

**return** arr[i]

# Driver's Code

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

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

  N **=** len(arr)

  # Function call

**print**(findRepeating(arr, N))

# This code is contributed by Arpit Jain

**Output**

8

**Time Complexity:** O(N2)

**Auxiliary Space:** O(1)

**Find the only repetitive element using sorting:**

*Sort the given input array. Traverse the array and if value of the ith element is not equal to i+1, then the current element is repetitive as value of elements is between 1 and N-1 and every element appears only once except one element.*

Follow the below steps to solve the problem:

* [Sort the given array](https://www.geeksforgeeks.org/sorting-algorithms/).
* Traverse the array and  compare the array elements with its index
* if **arr[i] != i+1**, it means that **arr[i]** is repetitive, So Just return **arr[i]**.
* Otherwise, the array does not contain duplicates from 1 to n-1, In this case, return -1

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find the only

# repeating element in an array where

# elements are from 1 to N-1.

**def** findRepeating(arr, N):

    arr.sort()

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

**if**(arr[i] !**=** i**+**1):

**return** arr[i]

# Driver's Code

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

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

  N **=** len(arr)

  # Function call

  print(findRepeating(arr, N))

# This code is contributed by Arpit Jain

**Output**

8

**Time complexity:** O(N \* log N)

**Auxiliary Space:** O(1)

**Find the only repetitive element using the hash set:**

*Use a hash table to store elements visited. If an already visited element appears again, return it.*

Follow the below steps to solve the problem:

* Create a hash set to store the visited elements
* Traverse the array
* If the given element is already present in the hash set then, return this element
* else insert this element into the hash set
* Return -1, if no repeating is found

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find the only

# repeating element in an array

# where elements are from 1 to n-1.

**def** findRepeating(arr, N):

    s **=** set()

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

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

**return** arr[i]

        s.add(arr[i])

    # If input is correct, we should

    # never reach here

**return -**1

# Driver code

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

  arr **=** [9, 8, 2, 6, 1, 8, 5, 3]

  N **=** len(arr)

  # Function call

**print**(findRepeating(arr, N))

# This code is contributed

# by Shrikant13

**Output**

8

**Time Complexity:** O(N)

**Auxiliary Space:** O(N)

**Find the only repetitive element using the Sum of first N elements:**

*We know*[*sum of first n-1 natural numbers*](https://www.geeksforgeeks.org/program-find-sum-first-n-natural-numbers/)*is (N – 1)\*N/2. We compute sum of array elements and subtract natural number sum from it to find the only missing element.*

Follow the below steps to solve the problem:

* Calculate the sum of array elements and the sum of first (N-1) natural numbers
* Return (array sum) – ((N-1) natural numbers sum)

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find the only

# repeating element in an array where

# elements are from 1 to N-1.

**def** findRepeating(arr, N):

    # Find array sum and subtract sum

    # first n-1 natural numbers from it

    # to find the result.

**return** sum(arr) **-** (((N **-** 1) **\*** N) **//** 2)

# Driver's Code

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

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

  N **=** len(arr)

  # Function call

  print(findRepeating(arr, N))

# This code is contributed

# by mohit kumar

**Output**

8

**Time Complexity:** O(N)

**Auxiliary Space:** O(1)

**Note:** This approach Causes overflow for large arrays.

**Find the only repetitive element using XOR:**

*The idea is based on the fact that x ^ x = 0 and if x ^ y = z then x ^ z = y*

Follow the below steps to solve the problem:

* Compute XOR of elements from 1 to n-1.
* Compute XOR of array elements.
* XOR of the above two would be our result.

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* PHP
* Javascript

# Python3 program to find the only

# repeating element in an array where

# elements are from 1 to N-1.

**def** findRepeating(arr, N):

    # res is going to store value of

    # 1 ^ 2 ^ 3 .. ^ (N-1) ^ arr[0] ^

    # arr[1] ^ .... arr[n-1]

    res **=** 0

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

        res **=** res ^ (i**+**1) ^ arr[i]

    res **=** res ^ arr[N**-**1]

**return** res

# Driver code

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

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

  N **=** len(arr)

  # Function call

  print(findRepeating(arr, N))

# This code is contributed by Smitha Dinesh Semwal.

**Output**

8

**Time Complexity:** O(N)

**Auxiliary Space:**O(1)

**Find the only repetitive element using indexing:**

*As there are only positive numbers, so visit the index equal to the current element and make it negative. If an index value is already negative, then it means that current element is repeated*

Follow the below steps to solve the problem:

* Iterate through the array.
* For every index visit **arr[arr[i]]**, if it is positive change the sign of elements at that index, else print the element.

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* PHP
* Javascript

# Python3 program to find the only

# repeating element in an array

# where elements are from 1 to N-1.

# Function to find repeated element

**def** findRepeating(arr, N):

    missingElement **=** 0

    # indexing based

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

        element **=** arr[abs(arr[i])]

**if**(element < 0):

            missingElement **=** arr[i]

**break**

        arr[abs(arr[i])] **= -**arr[abs(arr[i])]

**return** abs(missingElement)

# Driver code

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

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

  N **=** len(arr)

  # Function call

  print(findRepeating(arr, N))

# This code is contributed by Smitha Dinesh Semwal.

**Output**

8

**Time Complexity:** O(N)

**Auxiliary Space:** O(1)

**Find the only repetitive element using Linked-List cycle method:**

*Use two pointers the fast and the slow. The fast one goes forward two steps each time, while the slow one goes only step each time. They must meet the same item when slow==fast.*

*In fact, they meet in a circle, the duplicate number must be the entry point of the circle when visiting the array from array[0].*

*Next we just need to find the entry point. We use a point(we can use the fast one before) to visit from the beginning with one step each time, do the same job to slow. When fast==slow, they meet at the entry point of the circle.*

Follow the below steps to solve the problem:

* Declare two integer pointers as slow and fast
* Move the slow pointer one time and fast pointer two times, until slow is not equal to fast
* Once they are equal then again start the fast pointer from the start of the array
* Move both the pointers, one step at a time until both of them are equal
* Return slow or fast pointer as the answer

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

**class** GFG :

    @staticmethod

**def**  findDuplicate( nums) :

        slow **=** nums[0]

        fast **=** nums[0]

**while** True :

            slow **=** nums[slow]

            fast **=** nums[nums[fast]]

**if**((slow !**=** fast) **==** False) :

**break**

        fast **=** nums[0]

**while** (slow !**=** fast) :

            slow **=** nums[slow]

            fast **=** nums[fast]

**return** slow

    @staticmethod

**def** main( args) :

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

        # Function call

        ans **=** GFG.findDuplicate(arr)

        print(ans)

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

    GFG.main([])

    # This code is contributed by aadityaburujwale.

**Output**

8

**Time Complexity:** O(N)

**Auxiliary Space:** O(1)

*From <*[*https://www.geeksforgeeks.org/find-repetitive-element-1-n-1/*](https://www.geeksforgeeks.org/find-repetitive-element-1-n-1/)*>*

**How to check if two given sets are disjoint?**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 23 Dec, 2022
* Read
* Discuss(20+)
* Courses
* Practice
* Video

Given two sets represented by two arrays, how to check if the given two sets are disjoint or not? It may be assumed that the given arrays have no duplicates.

Input: set1[] = {12, 34, 11, 9, 3}  
 set2[] = {2, 1, 3, 5}  
Output: Not Disjoint  
3 is common in two sets.

Input: set1[] = {12, 34, 11, 9, 3}  
 set2[] = {7, 2, 1, 5}  
Output: Yes, Disjoint  
There is no common element in two sets.

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

There are plenty of methods to solve this problem, it’s a good test to check how many solutions you can guess.

**Method 1 (Simple):**Iterate through every element of the first set and search it in another set, if any element is found, return false. If no element is found, return true. The time complexity of this method is O(mn).

Following is the implementation of the above idea.

* C++
* Java
* Python
* C#
* PHP
* Javascript

# A Simple python 3 program to check

# if two sets are disjoint

# Returns true if set1[] and set2[] are disjoint, else false

**def** areDisjoint(set1, set2, m, n):

    # Take every element of set1[] and search it in set2

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

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

**if** (set1[i] **==** set2[j]):

**return** False

    # If no element of set1 is present in set2

**return** True

# Driver program

set1 **=** [12, 34, 11, 9, 3]

set2 **=** [7, 2, 1, 5]

m **=** len(set1)

n **=** len(set2)

**print**("yes") **if** areDisjoint(set1, set2, m, n) **else**(" No")

# This code ia contributed by Smitha Dinesh Semwal

**Output**

Yes

**Time Complexity:** O(m\*n)

**Auxiliary Space:**O(1),As constant extra space is used.

**Method 2 (Use Sorting and Merging)** :

1. Sort first and second sets.
2. Use merge like the process to compare elements.

Following is the implementation of the above idea.

* C++
* Java
* Python3
* C#
* Javascript

# A Simple Python 3 program to check

# if two sets are disjoint

# Returns true if set1[] and set2[]

# are disjoint, else false

**def** areDisjoint(set1, set2, m, n):

    # Sort the given two sets

    set1.sort()

    set2.sort()

    # Check for same elements

    # using merge like process

    i **=** 0; j **=** 0

**while** (i < m **and** j < n):

**if** (set1[i] < set2[j]):

            i **+=** 1

**elif** (set2[j] < set1[i]):

            j **+=** 1

**else**: # if set1[i] == set2[j]

**return** False

**return** True

# Driver Code

set1 **=** [12, 34, 11, 9, 3]

set2 **=** [7, 2, 1, 5]

m **=** len(set1)

n **=** len(set2)

print("Yes") **if** areDisjoint(set1, set2, m, n) **else** print("No")

# This code is contributed by Smitha Dinesh Semwal

**Output**

Yes

**Time Complexity:** O(m\*log m + n\*log n), The above solution first sorts both sets and then takes O(m+n) time to find the intersection. If we are given that the input sets are sorted, then this method is best among all.

**Auxiliary Space:**O(1), As constant extra space is used.

**Method 3 (Use Sorting and Binary Search):**

This is similar to method 1. Instead of a linear search, we use [Binary Search](https://www.geeksforgeeks.org/binary-search/).

1. Sort first set.
2. Iterate through every element of the second set, and use binary search to search every element in the first set. If an element is found return it.

The time complexity of this method is O(mLogm + nLogm)

**Method 4 (Use Binary Search Tree):**

1. Create a self-balancing binary search tree ([Red Black](https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/), [AVL](https://www.geeksforgeeks.org/avl-tree-set-1-insertion/), [Splay](https://www.geeksforgeeks.org/splay-tree-set-1-insert/), etc) of all elements in the first set.
2. Iterate through all elements of the second set and search every element in the above constructed Binary Search Tree. If the element is found, return false.
3. If all elements are absent, return true.

The time complexity of this method is O(m\*log m + n\*log m).

**Method 5 (Use Hashing):**

1. Create an empty hash table.
2. Iterate through the first set and store every element in the hash table.
3. Iterate through the second set and check if any element is present in the hash table. If present, then returns false, else ignore the element.
4. If all elements of the second set are not present in the hash table, return true.

The following is the implementation of this method.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to

# check if two sets are

# distinct or not

# This function prints

# all distinct elements

**def** areDisjoint(set1, set2,

                n1, n2):

  # Creates an empty hashset

  myset **=** set([])

  # Traverse the first set

  # and store its elements in hash

**for** i **in** range (n1):

    myset.add(set1[i])

  # Traverse the second set

  # and check if any element of it

  # is already in hash or not.

**for** i **in** range (n2):

**if** (set2[i] **in** myset):

**return** False

**return** True

# Driver method to test above method

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

  set1 **=** [10, 5, 3, 4, 6]

  set2 **=** [8, 7, 9, 3]

  n1 **=** len(set1)

  n2 **=** len(set2)

**if** (areDisjoint(set1, set2,

                  n1, n2)):

    print ("Yes")

**else**:

    print("No")

# This code is contributed by Chitranayal

**Output**

No

**Time Complexity:** O(m+n) under the assumption that hash set operations like add() and contains() work in O(1) time.

**Auxiliary Space:**O(n), The extra space is used to store the elements in the set.

*From <*[*https://www.geeksforgeeks.org/check-two-given-sets-disjoint/*](https://www.geeksforgeeks.org/check-two-given-sets-disjoint/)*>*

**Non-overlapping sum of two sets**

* Difficulty Level : [Basic](https://www.geeksforgeeks.org/basic/)
* Last Updated : 26 Dec, 2022
* Read
* Discuss
* Courses
* Practice
* Video

Given two arrays A[] and B[] of size n. It is given that both array individually contains distinct elements. We need to find the sum of all elements that are not common.

**Examples:**

Input : A[] = {1, 5, 3, 8}  
 B[] = {5, 4, 6, 7}  
Output : 29  
1 + 3 + 4 + 6 + 7 + 8 = 29

Input : A[] = {1, 5, 3, 8}  
 B[] = {5, 1, 8, 3}  
Output : 0  
All elements are common.

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

**Brute Force Method:**One simple approach is that for each element in A[] check whether it is present in B[], if it is present in then add it to the result. Similarly, traverse B[] and for every element that is not present in B, add it to result.

**Time Complexity:** O(n2).

**Auxiliary Space:**O(1), As constant extra space is used.

**Hashing concept:** Create an empty hash and insert elements of both arrays into it. Now traverse hash table and add all those elements whose count is 1. (As per the question, both arrays individually have distinct elements)

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find Non-overlapping sum

**from** collections **import** defaultdict

# Function for calculating

# Non-overlapping sum of two array

**def** findSum(A, B, n):

    # Insert elements of both arrays

    Hash **=** defaultdict(**lambda**:0)

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

        Hash[A[i]] **+=** 1

        Hash[B[i]] **+=** 1

    # calculate non-overlapped sum

    Sum **=** 0

**for** x **in** Hash:

**if** Hash[x] **==** 1:

            Sum **+=** x

**return** Sum

# Driver code

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

    A **=** [5, 4, 9, 2, 3]

    B **=** [2, 8, 7, 6, 3]

    # size of array

    n **=** len(A)

    # Function call

    print(findSum(A, B, n))

# This code is contributed

# by Rituraj Jain

**Output**

39

**Time Complexity:**O(n), since inserting in an unordered map is amortized constant.

**Auxiliary Space:** O(n).

**Another method:** Using set data structure

* Insert elements of Array A in the **set**data structure and add into **sum**
* Check if B’s elements are there in set if exist then remove current element from set, otherwise add current element to sum
* Finally, return sum

*From <*[*https://www.geeksforgeeks.org/overlapping-sum-two-array/*](https://www.geeksforgeeks.org/overlapping-sum-two-array/)*>*

**Check if two arrays are equal or not**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 26 Dec, 2022
* Read
* Discuss(50+)
* Courses
* Practice
* Video

Given two arrays, **arr1** and **arr2**of equal length **N**, the task is to find if the given arrays are equal or not.

Two arrays are said to be equal if:

* both of them contain the same set of elements,
* arrangements (or permutations) of elements might/might not be same.
* If there are repetitions, then counts of repeated elements must also be the same for two arrays to be equal.

**Examples:**

***Input:****arr1[] = {1, 2, 5, 4, 0}, arr2[] = {2, 4, 5, 0, 1}*

***Output:****Yes*

***Input:****arr1[] = {1, 2, 5, 4, 0, 2, 1}, arr2[] = {2, 4, 5, 0, 1, 1, 2}*

***Output:****Yes*

***Input:****arr1[] = {1, 7, 1}, arr2[] = {7, 7, 1}*

***Output:****No*

Recommended Problem

Check if two arrays are equal or not

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

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

+3 more

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

[Solve Problem](https://practice.geeksforgeeks.org/problems/check-if-two-arrays-are-equal-or-not3847/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 1.9L

**Check if two arrays are equal or not using Sorting**

Follow the steps below to solve the problem using this approach:

* Sort both the arrays
* Then linearly compare elements of both the arrays
* If all are equal then return true, else return false

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* PHP
* Javascript

# Python3 program to find given

# two array are equal or not

# Returns true if arr1[0..n-1] and

# arr2[0..m-1] contain same elements.

**def** areEqual(arr1, arr2, N, M):

    # If lengths of array are not

    # equal means array are not equal

**if** (N !**=** M):

**return** False

    # Sort both arrays

    arr1.sort()

    arr2.sort()

    # Linearly compare elements

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

**if** (arr1[i] !**=** arr2[i]):

**return** False

    # If all elements were same.

**return** True

# Driver Code

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

    arr1 **=** [3, 5, 2, 5, 2]

    arr2 **=** [2, 3, 5, 5, 2]

    n **=** len(arr1)

    m **=** len(arr2)

**if** (areEqual(arr1, arr2, n, m)):

        print("Yes")

**else**:

        print("No")

**Output**

Yes

**Time Complexity:** O(N\*log(N))

**Auxiliary Space:**O(1)

**Check if two arrays are equal or not using Hashing**

*Store count of all elements of arr1[] in a hash table. Then traverse arr2[] and check if the count of every element in arr2[] matches with the count of elements of arr1[].*

Follow the steps mentioned below to implement the approach:

* First check if length of arr1 is not equal to the length of arr2 then return false
* Then traverse over first array and store the count of every element in the hash map
* Then traverse over second array and decrease the count of its elements in the hash map. If that element is not present or the count of that element is   
  zero in the hash map, then return false, else decrease the count of that element
* Return true at the end, as both the arrays are equal by now

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program for the above approach

# Returns true if arr1[0..N-1] and

# arr2[0..M-1] contain same elements.

**def** is\_arr\_equal(arr1, arr2):

        # Check if the length of arrays are

    # equal or not: A Easy Logic Check

**if** len(arr1) !**=** len(arr2):

**return** False

    # Create a dict named count to

    # store counts of each element

    count **=** {}

    # Store the elements of arr1

    # and their counts in the dictionary

**for** i **in** arr1:

**if** i **in** count:

                # Element already in dict, simply increment its count

            count[i] **+=** 1

**else**:

                # Element found for first time, initialize it with value 1.

            count[i] **=** 1

    # Traverse through arr2 and compare

    # the elements and its count with

    # the elements of arr1

**for** i **in** arr2:

        # Return false if the element

        # is not in count or if any element

        # appears more no. of times than in arr1

**if** i **not in** count **or** count[i] **==** 0:

**return** False

**else**:

                # If element is found, decrement

                # its value in the dictionary

            count[i] **-=** 1

    # Return true if both arr1 and

    # arr2 are equal

**return** True

# Driver Code

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

    arr1 **=** [3, 5, 2, 5, 2]

    arr2 **=** [2, 3, 5, 5, 2]

**if** is\_arr\_equal(arr1, arr2):

**print**("Yes")

**else**:

**print**("No")

**Output**

Yes

**Time Complexity:**O(N)

**Auxiliary Space:**O(N)

**Another Method: Using Map**

* Initialise a map say **unmap**
* Insert all elements of array **A** into map
* Remove all elements of array **B** from map
* Check if the size of **unmap**becomes **zero**
* If zero, return **true**
* Otherwise, return **false**

*From <*[*https://www.geeksforgeeks.org/check-if-two-arrays-are-equal-or-not/*](https://www.geeksforgeeks.org/check-if-two-arrays-are-equal-or-not/)*>*

**Find missing elements of a range**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 01 Jul, 2022
* Read
* Discuss(20+)
* Courses
* Practice
* Video

Given an array, arr[0..n-1] of distinct elements and a range [low, high], find all numbers that are in a range, but not the array. The missing elements should be printed in sorted order.

**Examples:**

**Input:** arr[] = {10, 12, 11, 15},   
 low = 10, high = 15  
**Output:** 13, 14

**Input:** arr[] = {1, 14, 11, 51, 15},   
 low = 50, high = 55  
**Output:** 50, 52, 53, 54 55

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

There can be two approaches to solve the problem.

**Use Sorting:** Sort the array, then do a binary search for ‘low’. Once the location of low is found, start traversing the array from that location and keep printing all missing numbers.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python library for binary search

**from** bisect **import** bisect\_left

# A sorting based C++ program to find missing

# elements from an array

# Print all elements of range [low, high] that

# are not present in arr[0..n-1]

**def** printMissing(arr, n, low, high):

    # Sort the array

    arr.sort()

    # Do binary search for 'low' in sorted

    # array and find index of first element

    # which either equal to or greater than

    # low.

    ptr **=** bisect\_left(arr, low)

    index **=** ptr

    # Start from the found index and linearly

    # search every range element x after this

    # index in arr[]

    i **=** index

    x **=** low

**while** (i < n **and** x <**=** high):

    # If x doesn't match with current element

    # print it

**if**(arr[i] !**=** x):

            print(x, end **=**" ")

    # If x matches, move to next element in arr[]

**else**:

            i **=** i **+** 1

    # Move to next element in range [low, high]

        x **=** x **+** 1

    # Print range elements that are greater than the

    # last element of sorted array.

**while** (x <**=** high):

**print**(x, end **=**" ")

        x **=** x **+** 1

# Driver code

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

n **=** len(arr)

low **=** 1

high **=** 10

printMissing(arr, n, low, high);

# This code is contributed by YatinGupta

**Output**

2 6 7 8 9 10

***Time Complexity:*** O(n log n + k) where k is the number of missing elements

***Auxiliary Space:***O(n) or O(1) depending on the type of the array.

**Using Arrays:** Create a boolean array, where each index will represent whether the (i+low)th element is present in the array or not. Mark all those elements which are in the given range and are present in the array. Once all array items present in the given range have been marked true in the array, we traverse through the Boolean array and print all elements whose value is false.

**Implementation:**

* C++14
* Java
* Python3
* C#
* Javascript

# An array-based Python3 program to

# find missing elements from an array

# Print all elements of range

# [low, high] that are not

# present in arr[0..n-1]

**def** printMissing(arr, n, low, high):

    # Create boolean list of size

    # high-low+1, each index i

    # representing whether (i+low)th

    # element found or not.

    points\_of\_range **=** [False] **\*** (high**-**low**+**1)

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

        # if ith element of arr is in range

        # low to high then mark corresponding

        # index as true in array

**if** ( low <**=** arr[i] **and** arr[i] <**=** high ) :

            points\_of\_range[arr[i]**-**low] **=** True

    # Traverse through the range

    # and print all elements  whose value

    # is false

**for** x **in** range(high**-**low**+**1) :

**if** (points\_of\_range[x]**==**False) :

            print(low**+**x, end **=** " ")

# Driver Code

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

n **=** len(arr)

low, high **=** 1, 10

printMissing(arr, n, low, high)

# This code is contributed

# by Shubh Bansal

**Output**

2 6 7 8 9 10

***Time Complexity:* O(n + (high-low+1))**

***Auxiliary Space:*O(n)**

**Use Hashing:** Create a hash table and insert all array items into the hash table. Once all items are in hash table, traverse through the range and print all missing elements.

* C++
* Java
* Python3
* C#
* Javascript

# A hashing based Python3 program to

# find missing elements from an array

# Print all elements of range

# [low, high] that are not

# present in arr[0..n-1]

**def** printMissing(arr, n, low, high):

    # Insert all elements of

    # arr[] in set

    s **=** set(arr)

    # Traverse through the range

    # and print all missing elements

**for** x **in** range(low, high **+** 1):

**if** x **not in** s:

            print(x, end **=** ' ')

# Driver Code

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

n **=** len(arr)

low, high **=** 1, 10

printMissing(arr, n, low, high)

# This code is contributed

# by SamyuktaSHegde

**Output**

2 6 7 8 9 10

***me Complexity:*O(n + (high-low+1))**

***Auxiliary Space:* O(n)**

**Which approach is better?**

The time complexity of the first approach is O(nLogn + k) where k is the number of missing elements (Note that k may be more than n Log n if the array is small and the range is big)

The time complexity of the second and third solutions is O(n + (high-low+1)).

If the given array has almost all elements of the range, i.e., n is close to the value of (high-low+1), then the second and third approaches are definitely better as there is no Log n factor. But if n is much smaller than the range, then the first approach is better as it doesn’t require extra space for hashing. We can also modify the first approach to print adjacent missing elements as range to save time. For example, if 50, 51, 52, 53, 54, 59 are missing, we can print them as 50-54, 59 in the first method. And if printing this way is allowed, the first approach takes only O(n Log n) time. Out of the Second and Third Solutions, the second solution is better because the worst-case time complexity of the second solution is better than the third.

*From <*[*https://www.geeksforgeeks.org/find-missing-elements-of-a-range/*](https://www.geeksforgeeks.org/find-missing-elements-of-a-range/)*>*

**Minimum number of subsets with distinct elements**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 28 Jul, 2022
* Read
* Discuss
* Courses
* Practice
* Video

You are given an array of n-element. You have to make subsets from the array such that no subset contain duplicate elements. Find out minimum number of subset possible.

**Examples :**

Input : arr[] = {1, 2, 3, 4}  
Output :1  
Explanation : A single subset can contains all   
values and all values are distinct

Input : arr[] = {1, 2, 3, 3}  
Output : 2  
Explanation : We need to create two subsets  
{1, 2, 3} and {3} [or {1, 3} and {2, 3}] such  
that both subsets have distinct elements.

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

We basically need to find the most frequent element in the array. The result is equal to the frequency of the most frequent element.

A **simple solution**is to run two nested loops to count frequency of every element and return the frequency of the most frequent element. Time complexity of this solution is O(n2).

A **better solution**is to first sort the array and then start count number of repetitions of elements in an iterative manner as all repetition of any number lie beside the number itself. By this method you can find the maximum frequency or repetition by simply traversing the sorted array. This approach will cost O(nlogn) time complexity

**Implementation:**

* C++
* Java
* Python3
* C#
* PHP
* Javascript

# A sorting based solution to find the

# minimum number of subsets of a set

# such that every subset contains distinct

# elements.

# function to count subsets such that all

# subsets have distinct elements.

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

    # take input and initialize res = 0

    res **=** 0

    # sort the array

    ar.sort()

    # traverse the input array and

    # find maximum frequency

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

        count **=** 1

        # for each number find its repetition / frequency

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

**if** ar[i] **==** ar[i **+** 1]:

                count**+=**1

**else**:

**break**

        # update res

        res **=** max(res, count)

**return** res

# Driver code

ar **=** [ 5, 6, 9, 3, 4, 3, 4 ]

n **=** len(ar)

print(subset(ar, n))

# This code is contributed by

# Smitha Dinesh Semwal

**Output**

2

**Time Complexity: O(n2)**

**Auxiliary Space: O(1)**

An **efficient solution** is to use hashing. We count frequencies of all elements in a hash table. Finally we return the key with maximum value in hash table.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# A hashing based solution to find the

# minimum number of subsets of a set such

# that every subset contains distinct

# elements.

# Function to count subsets such that

# all subsets have distinct elements.

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

    # Traverse the input array and

    # store frequencies of elements

    mp **=** {i:0 **for** i **in** range(10)}

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

        mp[arr[i]] **+=** 1

    # Find the maximum value in map.

    res **=** 0

**for** key, value **in** mp.items():

        res **=** max(res, value)

**return** res

# Driver code

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

    arr **=** [5, 6, 9, 3, 4, 3, 4]

    n **=** len(arr)

    print(subset(arr, n))

# This code is contributed by

# Surendra\_Gangwar

**Output**

2

**Time Complexity: O(n)**

**Auxiliary Space: O(n)**

*From <*[*https://www.geeksforgeeks.org/minimum-number-subsets-distinct-elements/*](https://www.geeksforgeeks.org/minimum-number-subsets-distinct-elements/)*>*

**Remove minimum number of elements such that no common element exist in both array**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 19 Jul, 2022
* Read
* Discuss
* Courses
* Practice
* Video

Given two arrays A[] and B[] consisting of n and m elements respectively. Find the minimum number of elements to remove from each array such that no common element exist in both.

**Examples:**

Input : A[] = { 1, 2, 3, 4}  
 B[] = { 2, 3, 4, 5, 8 }  
Output : 3  
We need to remove 2, 3 and 4 from any array.

Input : A[] = { 4, 2, 4, 4}  
 B[] = { 4, 3 }  
Output : 1  
We need to remove 4 from B[]

Input : A[] = { 1, 2, 3, 4 }  
 B[] = { 5, 6, 7 }  
Output : 0  
There is no common element in both.

Recommended Problem

Remove minimum number of elements

[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)

[Solve Problem](https://practice.geeksforgeeks.org/problems/remove-minimum-number-of-elements4032/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 2K

Count occurrence of each number in both arrays. If there is a number in both array remove number from array in which it appears less number of times add it to the result.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find minimum

# element to remove so no common

# element exist in both array

# To find no elements to remove

# so no common element exist

**def** minRemove(a, b, n, m):

    # To store count of array element

    countA **=** dict()

    countB **=** dict()

    # Count elements of a

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

        countA[a[i]] **=** countA.get(a[i], 0) **+** 1

    # Count elements of b

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

        countB[b[i]] **=** countB.get(b[i], 0) **+** 1

    # Traverse through all common

    # element, and pick minimum

    # occurrence from two arrays

    res **=** 0

**for** x **in** countA:

**if** x **in** countB.keys():

            res **+=** min(countA[x],countB[x])

    # To return count of

    # minimum elements

**return** res

# Driver Code

a **=** [ 1, 2, 3, 4 ]

b **=** [2, 3, 4, 5, 8 ]

n **=** len(a)

m **=** len(b)

print(minRemove(a, b, n, m))

# This code is contributed

# by mohit kumar

**Output**

3

**Time Complexity: O(n+m)**

**Auxiliary Space: O(n+m)**

*From <*[*https://www.geeksforgeeks.org/remove-minimum-number-elements-no-common-element-exist-array/*](https://www.geeksforgeeks.org/remove-minimum-number-elements-no-common-element-exist-array/)*>*

**Remove minimum number of elements such that no common element exist in both array**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 19 Jul, 2022
* Read
* Discuss
* Courses
* Practice
* Video

Given two arrays A[] and B[] consisting of n and m elements respectively. Find the minimum number of elements to remove from each array such that no common element exist in both.

**Examples:**

Input : A[] = { 1, 2, 3, 4}  
 B[] = { 2, 3, 4, 5, 8 }  
Output : 3  
We need to remove 2, 3 and 4 from any array.

Input : A[] = { 4, 2, 4, 4}  
 B[] = { 4, 3 }  
Output : 1  
We need to remove 4 from B[]

Input : A[] = { 1, 2, 3, 4 }  
 B[] = { 5, 6, 7 }  
Output : 0  
There is no common element in both.

Recommended Problem

Remove minimum number of elements

[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)

[Solve Problem](https://practice.geeksforgeeks.org/problems/remove-minimum-number-of-elements4032/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 2K

Count occurrence of each number in both arrays. If there is a number in both array remove number from array in which it appears less number of times add it to the result.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find minimum

# element to remove so no common

# element exist in both array

# To find no elements to remove

# so no common element exist

**def** minRemove(a, b, n, m):

    # To store count of array element

    countA **=** dict()

    countB **=** dict()

    # Count elements of a

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

        countA[a[i]] **=** countA.get(a[i], 0) **+** 1

    # Count elements of b

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

        countB[b[i]] **=** countB.get(b[i], 0) **+** 1

    # Traverse through all common

    # element, and pick minimum

    # occurrence from two arrays

    res **=** 0

**for** x **in** countA:

**if** x **in** countB.keys():

            res **+=** min(countA[x],countB[x])

    # To return count of

    # minimum elements

**return** res

# Driver Code

a **=** [ 1, 2, 3, 4 ]

b **=** [2, 3, 4, 5, 8 ]

n **=** len(a)

m **=** len(b)

print(minRemove(a, b, n, m))

# This code is contributed

# by mohit kumar

**Output**

3

**Time Complexity: O(n+m)**

**Auxiliary Space: O(n+m)**

*From <*[*https://www.geeksforgeeks.org/remove-minimum-number-elements-no-common-element-exist-array/*](https://www.geeksforgeeks.org/remove-minimum-number-elements-no-common-element-exist-array/)*>*

**Time Complexity:**O(m2\*n2), where m and n are the numbers of rows and columns of the given matrix respectively.

**Auxiliary Space:**O(1)

**Method 2 (Use Sorting)**

* Sort all the rows in ascending order. The **time complexity** for this preprocessing will be O(n2 logn).
* Now we will select each row one by one and find pair elements in the remaining rows after the current row.
* Take two iterators, **left** and **right**. **left** iterator points left corner of the current i’th row and **right** iterator points right corner of the next j’th row in which we are going to find a pair of elements.
* If **mat[i][left] + mat[j][right] < sum** then **left++** i.e; move in i’th row towards the right corner, otherwise **right++** i.e; move in j’th row towards the left corner

**Implementation:**

* C++
* Java
* Python 3
* C#
* Javascript

# Python 3 program to find a pair with

# given sum such that every element of

# pair is in different rows.

MAX **=** 100

# Function to find pair for given

# sum in matrix mat[][] --> given matrix

# n --> order of matrix

# sum --> given sum for which we

# need to find pair

**def** pairSum(mat, n, sum):

    # First sort all the rows

    # in ascending order

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

        mat[i].sort()

    # Select i'th row and find pair for

    # element in i'th row in j'th row

    # whose summation is equal to given sum

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

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

            left **=** 0

            right **=** n **-** 1

**while** (left < n **and** right >**=** 0):

**if** ((mat[i][left] **+** mat[j][right]) **==** sum):

                    print( "(", mat[i][left],

                           ", ", mat[j][right], "), ",

                                            end **=** " ")

                    left **+=** 1

                    right **-=** 1

**else**:

**if** ((mat[i][left] **+**

                         mat[j][right]) < sum):

                        left **+=** 1

**else**:

                        right **-=** 1

# Driver Code

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

    n **=** 4

    sum **=** 11

    mat **=** [[1, 3, 2, 4],

           [5, 8, 7, 6],

           [9, 10, 13, 11],

           [12, 0, 14, 15]]

    pairSum(mat, n, sum)

# This code is contributed

# by ChitraNayal

**Output**

(3, 8), (4, 7), (1, 10), (2, 9), (11, 0),

**Time complexity :** O(n2logn + n^3)

**Auxiliary space :**O(1)

**Method 3 (**[**Hashing**](https://www.geeksforgeeks.org/hashing/)**)**

1. Create an empty hash table and store all elements of the matrix in the hash as keys and their locations as values.
2. Traverse the matrix again to check for every element whether its pair exists in the hash table or not. If it exists, then compare row numbers. If row numbers are not the same, then print the pair.

**Implementation:**

* CPP
* Java
* Python3
* C#
* Javascript

# Python3 program to find pairs with given sum such

# the two elements of pairs are from different rows

MAX **=** 100

# Function to find pair for given sum in matrix

  # mat[][] --> given matrix

  # n --> order of matrix

  # sum --> given sum for which we need to find pair

**def** pairSum(mat, n, sum):

    # Create a hash and store all elements of matrix

    # as keys, and row and column numbers as values

    hm **=** {}

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

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

            hm[mat[i][j]] **=** [i, j]

    # Traverse the matrix again to check for every

    # element whether its pair exists or not.

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

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

            # Look for remaining sum in hash

            remSum **=** sum **-** mat[i][j]

            # If an element with value equal to remaining sum exists

**if** remSum **in** hm:

                # Find row and column numbers of element with

                # value equal to remaining sum.

                p**=**hm[remSum]

                row **=** p[0]

                col **=** p[1]

                # If row number of pair is not same as current

                # row, then print it as part of result.

                # Second condition is there to make sure that a

                # pair is printed only once.

**if** (row !**=** i **and** row > i):

                    print("(" , mat[i][j] , "," , mat[row][col] , "), ", end**=**"")

# Driver code

n **=** 4

sum **=** 11

mat **=** [[1, 3, 2, 4],

                   [5, 8, 7, 6],

                   [9, 10, 13, 11],

                   [12, 0, 14, 15]]

pairSum(mat, n, sum)

# This code is contributed by patel2127

**Output**

(8, 3), (7, 4), (9, 2), (10, 1), (0, 11),

One important thing is, when we traverse a matrix, a pair may be printed twice. To make sure that a pair is printed only once, we check if the row number of other elements picked from the hash table is more than the row number of the current element.

**Time Complexity:** O(n2) under the assumption that hash table inserts and search operations take O(1) time.

**Auxiliary Space**:O(n2) because using HashMap

*From <*[*https://www.geeksforgeeks.org/find-pairs-given-sum-elements-pair-different-rows/*](https://www.geeksforgeeks.org/find-pairs-given-sum-elements-pair-different-rows/)*>*

**Count pairs with given sum**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 29 Dec, 2022
* Read
* Discuss(150)
* Courses
* Practice
* Video

Given an array of **N** integers, and a number **sum,**the task is tofind the **number of pairs** of integers in the array whose sum is equal to sum.

**Examples:**

***Input:****arr[] = {1, 5, 7, -1}, sum = 6*

***Output:****2*

***Explanation:****Pairs with sum 6 are (1, 5) and (7, -1).*

***Input:****arr[] = {1, 5, 7, -1, 5}, sum = 6*

***Output:****3*

***Explanation:****Pairs with sum 6 are (1, 5), (7, -1) & (1, 5).*

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

***Output:****6*

***Explanation:****Pairs with sum 2 are (1, 1), (1, 1), (1, 1), (1, 1), (1, 1).*

***Input:****arr[] = {10, 12, 10, 15, -1, 7, 6, 5, 4, 2, 1, 1, 1}, sum = 11*

***Output:****9*

***Explanation:****Pairs with sum 11 are (10, 1), (10, 1), (10, 1), (12, -1), (10, 1), (10, 1), (10, 1), (7, 4), (6, 5).*

Recommended Practice

[Count pairs with given sum](https://practice.geeksforgeeks.org/problems/count-pairs-with-given-sum5022/1/)

[Try It!](https://practice.geeksforgeeks.org/problems/count-pairs-with-given-sum5022/1/)

**Naive Approach:**

* *A****simple solution****is to traverse each element and check if there’s another number in the array which can be added to it to give****sum****.*
* *This can be achieved by*[*nested loops*](https://www.geeksforgeeks.org/nested-loops-in-c-with-examples-2/)*.*

**Illustration:**

***Given arr[] = {1, 5, 7, -1}, sum = 6***

*count = 0*

* *First Iteration : For index = 0  
  {1,****5****, 7, -1}, pair = (1, 5), count = 1*
* *Second Iteration : For index = 1  
  {1, 5, 7, -1}, count = 1*
* *Third Iteration : For index = 2  
  {1, 5, 7,****-1****}, count = 2*

*Hence****output****is 2*

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

Follow the steps below to solve the given problem:

* Initialize the **count** variable with **0**which stores the result.
* Iterate **arr**and if the sum of **ith** and **jth [i + 1…..n – 1]** element is equal to **sum** i.e. **arr[i] + arr[j] == sum**, then increment the count variable.
* Return the **count**.

Below is the implementation of the above approach.

* C++
* C
* Java
* Python3
* C#
* PHP
* Javascript

# Python3 implementation of simple method

# to find count of pairs with given sum.

# Returns number of pairs in arr[0..n-1]

# with sum equal to 'sum'

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

    count **=** 0  # Initialize result

    # Consider all possible pairs

    # and check their sums

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

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

**if** arr[i] **+** arr[j] **==** sum:

                count **+=** 1

**return** count

# Driver function

arr **=** [1, 5, 7, **-**1, 5]

n **=** len(arr)

sum **=** 6

print("Count of pairs is",

      getPairsCount(arr, n, sum))

# This code is contributed by Smitha Dinesh Semwal

**Output**

Count of pairs is 3

**Time Complexity:** O(n2), traversing the array for each element

**Auxiliary Space:**O(1)

**Count pairs with given sum using Binary Search**

This approach is based on the following idea:

* *If the array is sorted then for each array element****arr[i]****, find the number of pairs by finding all the values (****sum – arr[i]****) which are situated after****ith****index.*
* *This can be achieved using*[***Binary Search***](https://www.geeksforgeeks.org/binary-search/)*.*

**Illustration:**

***Given arr[] = {1, 5, 7, -1}, sum = 6***

*Array after sorting: arr[] = {-1, 1, 5, 7}*

*count = 0*

*At index = 0: val = sum – arr[0] = 6 – (-1) = 7*

*count = count + upperBound(1, 3, 7) – lowerBound(1, 3, 7)*

*count = 1*

*At index = 1: val = sum – arr[1] = 6 – 1 = 5*

*count = count + upperBound(2, 3, 5) – lowerBound(2, 3, 5)*

*count = 2*

*At index = 2: val = sum – arr[2] = 6 – 5 = 1*

*count = count + upperBound(3, 3, 1) – lowerBound(3, 3, 1)*

*count = 2*

*Number of pairs = 2*

Follow the steps below to solve the given problem:

* Sort the array **arr[]** in increasing order.
* Loop from **i = 0 to N-1**.
* Find the index of the first element having value same or just greater than **(sum – arr[i])** using [lower bound](https://www.geeksforgeeks.org/lower_bound-in-cpp/).
* Find the index of the first element having value just greater than **(sum – arr[i])** using [upper bound](https://www.geeksforgeeks.org/upper_bound-in-cpp/).
* The gap between these two indices is the number of elements with value same as (**sum – arr[i]**).
* Add this with the final **count** of pairs.
* Return the final count after the iteration is over.

Below is the implementation of the above approach.

* C++
* Java
* Python3

# Python code to implement the approach

**import** bisect

# Function to find the count of pairs

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

    arr.sort()

    x, c **=** 0, 0

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

        x **=** k**-**arr[i]

        # Lower bound from i+1

        y **=** bisect.bisect\_left(arr, x, i**+**1, n)

        # Upper bound from i+1

        z **=** bisect.bisect(arr, x, i**+**1, n)

        c **=** c**+**z**-**y

**return** c

# Driver function

arr **=** [1, 5, 7, **-**1, 5]

n **=** len(arr)

k **=** 6

# Function call

print("Count of pairs is", getPairsCount(arr, n, k))

# This code is contributed by Pushpesh Raj

**Output**

Count of pairs is 3

**Time Complexity:** O(n \* log(n) ), applying binary search on each element

**Auxiliary Space:** O(1)

**Count pairs with given sum using Hashing**

This approach is based on the following idea:

* *Check the frequency of****sum – arr[i]****in the****arr***
* *This can be achieved using*[***Hashing***](https://www.geeksforgeeks.org/what-is-hashing-a-complete-tutorial/)*.*

**Illustration:**

***Given arr[] = {1, 5, 7, -1}, sum = 6***

* *Store the frequency of every element:****freq[arr[i]] = freq[arr[i]] + 1****freq[1] : 1  
  freq[5] : 1  
  freq[7] : 1  
  freq[-1] : 1*
* *Initialise a variable****count****with 0 to find the required count of pairs*
* *At index = 0: freq[sum – arr[0]] = freq[6 – 1] = freq[5] = 1  
  count = 1*
* *At index = 1: freq[sum – arr[1]] = freq[6 – 5] = freq[1] = 1  
  count = 2*
* *At index = 2: freq[sum – arr[2]] = freq[6 – 7] = freq[-1] = 1  
  count = 3*
* *At index = 3: freq[sum – arr[3]] = freq[6 – (-1)] = freq[7] = 1  
  count = 4*
* *The above also contains repeated pairs from front and last, i.e. pair (a, b) and (b, a) are considered as different pairs till now.  
  Therefore, we will reduce the count by half to determine the count of unique pairs.****count = count / 2 = 2***
* *Therefore, required Number of pairs with given sum = 2*

Follow the steps below to solve the given problem:

* Create a map to store the frequency of each number in the array. (Single traversal is required)
* In the next traversal, for every element check if it can be combined with any other element (other than itself!) to give the desired sum. Increment the counter accordingly.
* After completion of the second traversal, we’d have twice the required value stored in counter because every pair is counted two times. Hence divide the count by 2 and return.

Below is the implementation of the above idea :

* C++
* Java
* Python3
* C#
* Javascript

# Python 3 implementation of simple method

# to find count of pairs with given sum.

**import** sys

# Returns number of pairs in arr[0..n-1]

# with sum equal to 'sum'

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

    m **=** [0] **\*** 1000

    # Store counts of all elements in map m

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

        m[arr[i]] **+=** 1

    twice\_count **=** 0

    # Iterate through each element and increment

    # the count (Notice that every pair is

    # counted twice)

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

        twice\_count **+=** m[sum **-** arr[i]]

        # if (arr[i], arr[i]) pair satisfies the

        # condition, then we need to ensure that

        # the count is  decreased by one such

        # that the (arr[i], arr[i]) pair is not

        # considered

**if** (sum **-** arr[i] **==** arr[i]):

            twice\_count **-=** 1

    # return the half of twice\_count

**return** int(twice\_count **/** 2)

# Driver function

arr **=** [1, 5, 7, **-**1, 5]

n **=** len(arr)

sum **=** 6

**print**("Count of pairs is", getPairsCount(arr,

                                         n, sum))

# This code is contributed by

# Smitha Dinesh Semwal

**Output**

Count of pairs is 3

**Time Complexity:** O(n), to iterate over the array

**Auxiliary Space:**O(n), to make a map of size n

**Count pairs with given sum using Hashing in Single loop**

This approach is based on the following idea:

* *The idea is to solve in****single****loop.*
* *Check the frequency of****sum – arr[i]****in the****arr***
* *This can be achieved using*[***Hashing***](https://www.geeksforgeeks.org/what-is-hashing-a-complete-tutorial/)*.*

**Illustration:**

***Given arr[] = {1, 5, 7, -1}, sum = 6***

*count = 0*

*At index = 0: freq[sum – arr[0]] = freq[6 – 1] = freq[5] = 0*

*count = 0*

*freq[arr[0]] = freq[1] = 1*

*At index = 1: freq[sum – arr[1]] = freq[6 – 5] = freq[1] = 1*

*count = 1*

*freq[arr[1]] = freq[5] = 1*

*At index = 2: freq[sum – arr[2]] = freq[6 – 7] = freq[-1] = 0*

*count = 1*

*freq[arr[2]] = freq[7] = 1*

*At index = 3: freq[sum – arr[3]] = freq[6 – (-1)] = freq[7] = 1*

*count = 2*

*freq[arr[3]] = freq[-1] = 1*

*count = 2*

*Number of pairs  = 2*

Follow the steps below to solve the given problem:

* Create a map to store the frequency of each number in the array.
* Check if (**sum – arr[i]**) is present in the map, if present then increment the **count** variable by its frequency.
* After traversal is over, return the **count**.

Below is the implementation of the above idea :

* C++
* Java
* Python3
* C#
* Javascript

# Python implementation of simple method to find count of

# pairs with given sum.

# Returns number of pairs in arr[0..n-1] with sum equal to 'sum'

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

    unordered\_map **=** {}

    count **=** 0

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

**if** sum **-** arr[i] **in** unordered\_map:

            count **+=** unordered\_map[sum **-** arr[i]]

**if** arr[i] **in** unordered\_map:

            unordered\_map[arr[i]] **+=** 1

**else**:

            unordered\_map[arr[i]] **=** 1

**return** count

# Driver code

arr **=** [1, 5, 7, **-**1, 5]

n **=** len(arr)

sum **=** 6

print('Count of pairs is', getPairsCount(arr, n, sum))

# This code is contributed by Manish Thapa

**Output**

Count of pairs is 3

**Time Complexity:**O(n), to iterate over the array

**Auxiliary Space:** O(n), to make a map of size n

*From <*[*https://www.geeksforgeeks.org/count-pairs-with-given-sum/*](https://www.geeksforgeeks.org/count-pairs-with-given-sum/)*>*

**Count quadruples from four sorted arrays whose sum is equal to a given value x**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 27 Sep, 2022
* Read
* Discuss
* Courses
* Practice
* Video

Given four sorted arrays each of size **n** of distinct elements. Given a value **x**. The problem is to count all **quadruples**(group of four numbers) from all the four arrays whose sum is equal to **x**.

**Note:** The quadruple has an element from each of the four arrays.

**Examples:**

Input : arr1 = {1, 4, 5, 6},  
 arr2 = {2, 3, 7, 8},  
 arr3 = {1, 4, 6, 10},  
 arr4 = {2, 4, 7, 8}   
 n = 4, x = 30  
Output : 4  
The quadruples are:  
**(4, 8, 10, 8), (5, 7, 10, 8),**  
**(5, 8, 10, 7), (6, 7, 10, 7)**

Input : For the same above given fours arrays  
 x = 25  
Output : 14

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

**Method 1 (Naive Approach):**

Using four nested loops generate all quadruples and check whether elements in the quadruple sum up to **x** or not.

* C++
* Java
* Python3
* C#
* PHP
* Javascript

# A Python3 implementation to count

# quadruples from four sorted arrays

# whose sum is equal to a given value x

# function to count all quadruples

# from four sorted arrays whose sum

# is equal to a given value x

**def** countquadruples(arr1, arr2,

                     arr3, arr4, n, x):

    count **=** 0

    # generate all possible

    # quadruples from the four

    # sorted arrays

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

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

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

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

                    # check whether elements of

                    # quadruple sum up to x or not

**if** (arr1[i] **+** arr2[j] **+**

                        arr3[k] **+** arr4[l] **==** x):

                        count **+=** 1

    # required count of quadruples

**return** count

# Driver Code

arr1 **=** [1, 4, 5, 6]

arr2 **=** [2, 3, 7, 8]

arr3 **=** [1, 4, 6, 10]

arr4 **=** [2, 4, 7, 8 ]

n **=** len(arr1)

x **=** 30

print("Count = ", countquadruples(arr1, arr2,

                                   arr3, arr4, n, x))

# This code is contributed

# by Shrikant13

**Output:**

Count = 4

**Time Complexity:** O(n4)

**Auxiliary Space:** O(1)

**Method 2 (Binary Search):** Generate all triplets from the 1st three arrays. For each triplet so generated, find the sum of elements in the triplet. Let it be **T**. Now, search the value **(x – T)** in the 4th array. If the value found in the 4th array, then increment **count**. This process is repeated for all the triplets generated from the 1st three arrays.

* C++
* Java
* Python3
* C#
* PHP
* Javascript

# Python implementation to count quadruples from

# four sorted arrays whose sum is equal to a

# given value x

# find the 'value' in the given array 'arr[]'

# binary search technique is applied

**def** isPresent(arr,low,high,value):

**while**(low<**=**high):

        mid**=**(low**+**high)**//**2

        # 'value' found

**if**(arr[mid]**==**value):

**return** True

**elif**(arr[mid]>value):

            high**=**mid**-**1

**else**:

            low**=**mid**+**1

    # 'value' not found

**return** False

# function to count all quadruples from four

# sorted arrays whose sum is equal to a given value x

**def** countQuadruples(arr1,arr2,arr3,arr4,n,x):

    count**=**0

    #generate all triplets from the 1st three arrays

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

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

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

                # calculate the sum of elements in

                # the triplet so generated

                T**=**arr1[i]**+**arr2[j]**+**arr3[k]

                # check if 'x-T' is present in 4th

                # array or not

**if**(isPresent(arr4,0,n**-**1,x**-**T)):

                    # increment count

                    count**=**count**+**1

    # required count of quadruples

**return** count

# Driver program to test above

# four sorted arrays each of size 'n'

arr1**=**[1, 4, 5, 6]

arr2**=**[2, 3, 7, 8]

arr3**=**[1, 4, 6, 10]

arr4**=**[2, 4, 7, 8]

n**=**len(arr1)

x**=**30

print("Count = {}".format(countQuadruples(arr1,arr2,arr3,arr4,n,x)))

# This code is contributed by Pushpesh Raj.

**Output:**

Count = 4

**Time Complexity:** O(n3logn)

**Auxiliary Space:** O(1)

**Method 3 (Use of two pointers):** Generate all pairs from the 1st two arrays. For each pair so generated, find the sum of elements in the pair. Let it be **p\_sum**. For each **p\_sum**, [count pairs from the 3rd and 4th sorted array with sum equal to **(x – p\_sum)**](https://www.geeksforgeeks.org/count-pairs-two-sorted-arrays-whose-sum-equal-given-value-x/). Accumulate these count in the **total\_count** of quadruples.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 implementation to

# count quadruples from four

# sorted arrays whose sum is

# equal to a given value x

# count pairs from the two

# sorted array whose sum

# is equal to the given 'value'

**def** countPairs(arr1, arr2,

               n, value):

     count **=** 0

     l **=** 0

     r **=** n **-** 1

     # traverse 'arr1[]' from

     # left to right

     # traverse 'arr2[]' from

     # right to left

**while** (l < n **and** r >**=** 0):

          sum **=** arr1[l] **+** arr2[r]

          # if the 'sum' is equal

          # to 'value', then

          # increment 'l', decrement

          # 'r' and increment 'count'

**if** (sum **==** value):

               l **+=** 1

               r **-=** 1

               count **+=** 1

               # if the 'sum' is greater

               # than 'value', then decrement r

**elif** (sum > value):

               r **-=** 1

          # else increment l

**else**:

               l **+=** 1

     # required count of pairs

     # print(count)

**return** count

# function to count all quadruples

# from four sorted arrays whose sum

# is equal to a given value x

**def** countQuadruples(arr1, arr2,

                    arr3, arr4,

                    n, x):

     count **=** 0

     # generate all pairs from

     # arr1[] and arr2[]

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

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

               # calculate the sum of

               # elements in the pair

               # so generated

               p\_sum **=** arr1[i] **+** arr2[j]

               # count pairs in the 3rd

               # and 4th array having

               # value 'x-p\_sum' and then

               # accumulate it to 'count

               count **+=** int(countPairs(arr3, arr4,

                                       n, x **-** p\_sum))

     # required count of quadruples

**return** count

# Driver code

arr1 **=** [1, 4, 5, 6]

arr2 **=** [2, 3, 7, 8]

arr3 **=** [1, 4, 6, 10]

arr4 **=** [2, 4, 7, 8]

n **=** len(arr1)

x **=** 30

print("Count = ", countQuadruples(arr1, arr2,

                                  arr3, arr4,

                                  n, x))

# This code is contributed by Stream\_Cipher

**Output:**

Count = 4

**Time Complexity:** O(n3)

**Auxiliary Space:** O(1)

**Method 4 Efficient Approach(Hashing):** Create a hash table where **(key, value)** tuples are represented as **(sum, frequency)** tuples. Here the sum are obtained from the pairs of 1st and 2nd array and their frequency count is maintained in the hash table. Hash table is implemented using [unordered\_map in C++](https://www.geeksforgeeks.org/unordered_map-in-stl-and-its-applications/). Now, generate all pairs from the 3rd and 4th array. For each pair so generated, find the sum of elements in the pair. Let it be **p\_sum**. For each **p\_sum**, check whether **(x – p\_sum)** exists in the hash table or not. If it exists, then add the frequency of **(x – p\_sum)** to the **count** of quadruples.

* C++
* Java
* Python3
* C#
* Javascript

# Python implementation to count quadruples from

# four sorted arrays whose sum is equal to a

# given value x

# function to count all quadruples from four sorted

# arrays whose sum is equal to a given value x

**def** countQuadruples(arr1, arr2, arr3, arr4, n, x):

    count **=** 0

    # unordered\_map 'um' implemented as hash table

    # for <sum, frequency> tuples

    m **=** {}

    # count frequency of each sum obtained from the

    # pairs of arr1[] and arr2[] and store them in 'um'

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

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

**if** (arr1[i] **+** arr2[j]) **in** m:

                m[arr1[i] **+** arr2[j]] **+=** 1

**else**:

                m[arr1[i] **+** arr2[j]] **=** 1

    # generate pair from arr3[] and arr4[]

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

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

            # calculate the sum of elements in

            # the pair so generated

            p\_sum **=** arr3[k] **+** arr4[l]

            # if 'x-p\_sum' is present in 'um' then

            # add frequency of 'x-p\_sum' to 'count'

**if** (x **-** p\_sum) **in** m:

                count **+=** m[x **-** p\_sum]

    # required count of quadruples

**return** count

# Driver program to test above

# four sorted arrays each of size 'n'

arr1 **=** [1, 4, 5, 6]

arr2 **=** [2, 3, 7, 8 ]

arr3 **=** [1, 4, 6, 10]

arr4 **=** [2, 4, 7, 8 ]

n **=** len(arr1)

x **=** 30

print("Count =", countQuadruples(arr1, arr2, arr3, arr4, n, x))

# This code is contributed by avanitrachhadiya2155

**Output:**

Count = 4

**Time Complexity:** O(n2)

**Auxiliary Space:** O(n2)

*From <*[*https://www.geeksforgeeks.org/count-quadruples-four-sorted-arrays-whose-sum-equal-given-value-x/*](https://www.geeksforgeeks.org/count-quadruples-four-sorted-arrays-whose-sum-equal-given-value-x/)*>*

**Sort elements by frequency | Set 4 (Efficient approach using hash)**

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 19 Jul, 2022
* Read
* Discuss(40)
* Courses
* Practice
* Video

Print the elements of an array in the decreasing frequency if 2 numbers have the same frequency then print the one which came first.

**Examples:**

Input : arr[] = {2, 5, 2, 8, 5, 6, 8, 8}  
Output : arr[] = {8, 8, 8, 2, 2, 5, 5, 6}

Input : arr[] = {2, 5, 2, 6, -1, 9999999, 5, 8, 8, 8}  
Output : arr[] = {8, 8, 8, 2, 2, 5, 5, 6, -1, 9999999}

[Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/sorting-elements-of-an-array-by-frequency/0)

We have discussed different approaches in below posts :

[Sort elements by frequency | Set 1](https://www.geeksforgeeks.org/sort-elements-by-frequency/)

[Sort elements by frequency | Set 2](https://www.geeksforgeeks.org/sort-elements-by-frequency-set-2/)

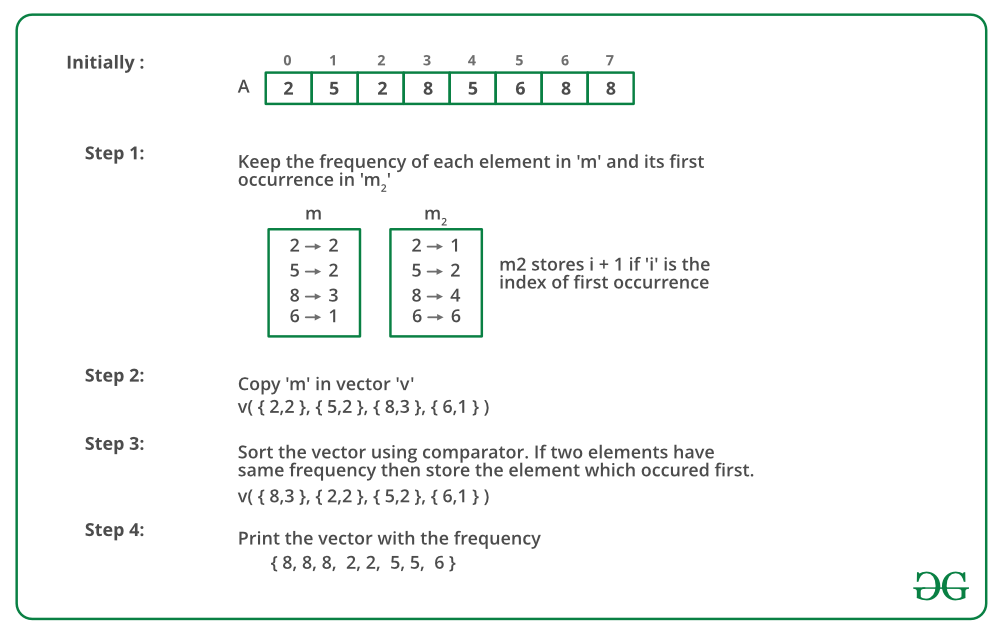
Sorting Array Elements By Frequency | Set 3 (Using STL)

All of the above approaches work in O(n Log n) time where n is total number of elements. In this post, a new approach is discussed that works in **O(n + m Log m)**time where n is total number of elements and m is total number of distinct elements.

The idea is to use [hashing](https://www.geeksforgeeks.org/hashing-data-structure/).

1. We insert all elements and their counts into a hash. This step takes O(n) time where n is number of elements.
2. We copy the contents of hash to an array (or vector) and sort them by counts. This step takes O(m Log m) time where m is total number of distinct elements.
3. For maintaining the order of elements if the frequency is the same, we use another hash which has the key as elements of the array and value as the index. If the frequency is the same for two elements then sort elements according to the index.

The below image is a dry run of the above approach:



We do not need to declare another map m2, as it does not provide the proper expected result for the problem.

instead, we need to just check for the first values of the pairs sent as parameters in the sortByVal function.

Below is the implementation of the above approach:

* C++
* Python3
* Javascript

# Used for sorting by frequency. And if frequency is same,

# then by appearance

**from** functools **import** cmp\_to\_key

**def** sortByVal(a,b):

    # If frequency is same then sort by index

**if** (a[1] **==** b[1]):

**return** a[0] **-** b[0]

**return** b[1] **-** a[1]

# function to sort elements by frequency

**def** sortByFreq(a, n):

    res **=** []

    m **=** {}

    v **=** []

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

        # Map m is used to keep track of count

        # of elements in array

**if**(a[i] **in** m):

            m[a[i]] **=** m[a[i]]**+**1

**else**:

            m[a[i]] **=** 1

**for** key,value **in** m.items():

        v.append([key,value])

    # Sort the element of array by frequency

    v.sort(key **=** cmp\_to\_key(sortByVal))

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

**while**(v[i][1]):

            res.append(v[i][0])

            v[i][1] **-=** 1

**return** res

# Driver program

a **=** [ 2, 5, 2, 6, **-**1, 9999999, 5, 8, 8, 8 ]

n **=** len(a)

res **=** []

res **=** sortByFreq(a, n)

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

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

# This code is contributed by shinjanpatra

**Output**

8 8 8 2 2 5 5 -1 6 9999999

**Time Complexity :** **O(n) + O(m Log m)**where n is total number of elements and m is total number of distinct elements

**Auxiliary Space:** **O(n)**

*From <*[*https://www.geeksforgeeks.org/sort-elements-frequency-set-4-efficient-approach-using-hash/*](https://www.geeksforgeeks.org/sort-elements-frequency-set-4-efficient-approach-using-hash/)*>*

**Find all pairs (a, b) in an array such that a % b = k**

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 13 Jul, 2022
* Read
* Discuss
* Courses
* Practice
* Video

Given an array with distinct elements, the task is to find the pairs in the array such that a % b = k, where k is a given integer.

**Examples :**

Input : arr[] = {2, 3, 5, 4, 7}   
 k = 3  
Output : (7, 4), (3, 4), (3, 5), (3, 7)  
7 % 4 = 3  
3 % 4 = 3  
3 % 5 = 3  
3 % 7 = 3

Recommended Problem

Mr Modulo and Pairs

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

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

+2 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/mr-modulo-and-pairs5610/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 6.4K

A **Naive Solution** is to make all pairs one by one and check their modulo is equal to k or not. If equals to k, then print that pair.

**Implementation:**

* C++
* Java
* Python3
* C#
* PHP
* Javascript

# Python3 implementation to find such pairs

# Function to find pair such that (a % b = k)

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

    isPairFound **=** True

    # Consider each and every pair

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

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

            # Print if their modulo equals to k

**if** (i !**=** j **and** arr[i] **%** arr[j] **==** k):

                print("(", arr[i], ", ", arr[j], ")",

                                 sep **=** "", end = " ")

                isPairFound **=** True

**return** isPairFound

# Driver Code

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

n **=** len(arr)

k **=** 3

**if** (printPairs(arr, n, k) **==** False):

**print**("No such pair exists")

# This article is contributed by Smitha Dinesh Semwal.

**Output**

(3, 5) (3, 4) (3, 7) (7, 4)

**Time Complexity :** **O(n2)**

**Auxiliary Space: O(1)**

An **Efficient solution** is based on below observations :

1. If k itself is present in arr[], then k forms a pair with all elements arr[i] where k < arr[i]. For all such arr[i], we have k % arr[i] = k.
2. For all elements greater than or equal to k, we use the following fact.

If arr[i] % arr[j] = k,   
 ==> arr[i] = x \* arr[j] + k  
 ==> (arr[i] - k) = x \* arr[j]  
 We find all divisors of (arr[i] - k)  
 and see if they are present in arr[].

To quickly check if an element is present in the array, we use hashing.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find all pairs

# such that a % b = k.

# Utility function to find the divisors

# of n and store in vector v[]

**import** math as mt

**def** findDivisors(n):

    v **=** []

    # Vector is used to store the divisors

**for** i **in** range(1, mt.floor(n**\*\***(.5)) **+** 1):

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

            # If n is a square number, push

            # only one occurrence

**if** (n **/** i **==** i):

                v.append(i)

**else**:

                v.append(i)

                v.append(n **//** i)

**return** v

# Function to find pairs such that (a%b = k)

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

    # Store all the elements in the map

    # to use map as hash for finding elements

    # in O(1) time.

    occ **=** dict()

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

        occ[arr[i]] **=** True

    isPairFound **=** False

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

        # Print all the pairs with (a, b) as

        # (k, numbers greater than k) as

        # k % (num (> k)) = k i.e. 2%4 = 2

**if** (occ[k] **and** k < arr[i]):

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

            isPairFound **=** True

        # Now check for the current element as 'a'

        # how many b exists such that a%b = k

**if** (arr[i] >**=** k):

            # find all the divisors of (arr[i]-k)

            v **=** findDivisors(arr[i] **-** k)

            # Check for each divisor i.e. arr[i] % b = k

            # or not, if yes then print that pair.

**for** j **in** range(len(v)):

**if** (arr[i] **%** v[j] **==** k **and**

                    arr[i] !**=** v[j] **and**

                    occ[v[j]]):

**print**("(", arr[i], ",", v[j],

                                       ")", end **=** " ")

                    isPairFound **=** True

**return** isPairFound

# Driver Code

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

n **=** len(arr)

k **=** 2

**if** (printPairs(arr, n, k) **==** False):

    print("No such pair exists")

# This code is contributed by mohit kumar

**Output**

(2, 3) (2, 5) (5, 3) (2, 4)

**Time Complexity:** **O(n\* sqrt(max))** where max is the maximum element in the array.

**Auxiliary Space: O(n)**

*From <*[*https://www.geeksforgeeks.org/find-pairs-b-array-b-k/*](https://www.geeksforgeeks.org/find-pairs-b-array-b-k/)*>*

**Group words with same set of characters**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 24 Aug, 2022
* Read
* Discuss(20+)
* Courses
* Practice
* Video

Given a list of words with lower cases. Implement a function to find all Words that have the same unique character set .

**Example:**

Input: words[] = { "may", "student", "students", "dog",  
 "studentssess", "god", "cat", "act",  
 "tab", "bat", "flow", "wolf", "lambs",  
 "amy", "yam", "balms", "looped",   
 "poodle"};  
Output :   
looped, poodle,   
lambs, balms,   
flow, wolf,   
tab, bat,   
may, amy, yam,   
student, students, studentssess,   
dog, god,   
cat, act,

All words with same set of characters are printed   
together in a line.

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

The idea is to use hashing. We generate a key for all words. The key contains all unique character (Size of key is at most 26 for lower case alphabets). We store indexes of words as values for a key. Once we have filled all keys and values in hash table, we can print the result by traversing the table.

Below is the implementation of above idea.

* C++
* Java
* Python3
* C#

# Python program to print all words that

# have the same unique character set

# Function to group all strings with same characters

**from** collections **import** Counter

**def** groupStrings(input):

    # traverse all strings one by one

    # dict is an empty dictionary

    dict**=**{}

**for** word **in** input:

        # sort the current string and take it's

        # sorted value as key

        # sorted return list of sorted characters

        # we need to join them to get key as string

        # Counter() method returns dictionary with frequency of

        # each character as value

        wordDict**=**Counter(word)

        # now get list of keys

        key **=** wordDict.keys()

        # now sort these keys

        key **=** sorted(key)

        # join these characters to produce key string

        key **=** ''.join(key)

        # now check if this key already exist in

        # dictionary or not

        # if exist then simply append current word

        # in mapped list on key

        # otherwise first assign empty list to key and

        # then append current word in it

**if** key **in** dict.keys():

            dict[key].append(word)

**else**:

            dict[key]**=**[]

            dict[key].append(word)

        # now traverse complete dictionary and print

        # list of mapped strings in each key separated by ,

**for** (key,value) **in** dict.items():

        print (','.join(dict[key]))

# Driver program

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

    input**=**['may','student','students','dog','studentssess','god','cat','act','tab','bat','flow','wolf','lambs','amy','yam','balms','looped','poodle']

    groupStrings(input)

**Output**

looped, poodle,   
student, students, studentssess,   
may, amy, yam,   
dog, god,   
cat, act,   
tab, bat,   
lambs, balms,   
flow, wolf,

**Time complexity:** **O(n\*k)**where n is number of words in dictionary and k is maximum length of a word.

**Auxiliary Space:** **O(n\*k),** where n is number of words in dictionary and k is maximum length of a word.

*From <*[*https://www.geeksforgeeks.org/print-words-together-set-characters/*](https://www.geeksforgeeks.org/print-words-together-set-characters/)*>*

**k-th distinct (or non-repeating) element among unique elements in an array.**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 19 Oct, 2022
* Read
* Discuss
* Courses
* Practice
* Video

Given an integer array, print **k-th** distinct element in an array. The given array may contain duplicates and the output should print **k-th** element among all unique elements. If**k** is more than number of distinct elements, print **-1**.

**Examples :**

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

First non-repeating element is 3  
Second non-repeating element is 4

**Input :** arr[] = {1, 2, 50, 10, 20, 2},   
 k = 3  
**Output :** 10

**Input :** {2, 2, 2, 2},   
 k = 2  
**Output :** -1

Recommended Problem

K-th distinct element

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

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

[Solve Problem](https://practice.geeksforgeeks.org/problems/k-th-distinct-element4510/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 3.1K

A **simple solution** is to use two nested loops where outer loop picks elements from left to right, and inner loop checks if the picked element is present somewhere else. If not present, then increment count of distinct elements. If count becomes **k**, return current element.

**Implementation:**

* C++
* Java
* Python3
* C#
* PHP
* Javascript

# Python3 program to print k-th distinct

# element in a given array

# Returns k-th distinct

# element in arr.

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

    dist\_count **=** 0

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

        # Check if current element is

        # present somewhere else.

        j **=** 0

**while** j < n:

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

**break**

            j **+=** 1

        # If element is unique

**if** (j **==** n):

            dist\_count **+=** 1

**if** (dist\_count **==** k):

**return** arr[i]

**return -**1

# Driver Code

ar **=** [1, 2, 1, 3, 4, 2]

n **=** len(ar)

k **=** 2

**print**(printKDistinct(ar, n, k))

# This code is contributed by Mohit Kumar

**Output**

4

***Time Complexity:*** **O(n^2)**

***Auxiliary Space:*O(1)**

An **efficient solution** is to use Hashing to solve this in **O(n)** time on average.

1. create an empty hash table.
2. Traverse input array from left to right and store elements and their counts in the hash table.
3. Traverse input array again from left to right. Keep counting elements with count as 1.
4. If count becomes k, return current element.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to print k-th

# distinct element in a given array

**def** printKDistinct(arr, size, KthIndex):

    dict **=** {}

    vect **=** []

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

**if**(arr[i] **in** dict):

            dict[arr[i]] **=** dict[arr[i]] **+** 1

**else**:

            dict[arr[i]] **=** 1

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

**if**(dict[arr[i]] > 1):

**continue**

**else**:

            KthIndex **=** KthIndex **-** 1

**if**(KthIndex **==** 0):

**return** arr[i]

**return -**1

# Driver Code

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

size **=** len(arr)

**print**(printKDistinct(arr, size, 2))

# This code is contributed

# by Akhand Pratap Singh

**Output**

4

***Time Complexity:*O(n)**

***Auxiliary Space:* O(n)**