

**Searching and SortingQUESTIONS *:-***

**Very Easy (Questions 1–5)**

**1.Searching a Number**

Given an integer k and array arr. Your task is to return the position of the first occurrence of k in the given array and if element k is not present in the array then return -1.

Note: 1-based indexing is followed here.

**Example1**:

**Input:** k = 16 , arr = [9, 7, 16, 16, 4]

**Output:** 3

**Explanation:** The value 16 is found in the given array at positions 3 and 4, with position 3 being the first occurrence.

**Example2:**

**Input:** k=98 , arr = [1, 22, 57, 47, 34, 18, 66]

**Output:** -1

**Example2:**

**Input:** k=9 , arr = [1, 22, 57, 47, 34, 9, 66]

**Output:** 6

**Explanation:** k = 98 isn't found in the given array.

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

**Expected Auxiliary Space:** O(1)

**Constraints:**

* 1 <= arr.size <= 106
* 1 <= arr[i] <= 109
* 1 <= k <= 106

**Reference:**: [https://www.geeksforgeeks.org/problems/searching-a-number0324/1](https://www.geeksforgeeks.org/problems/searching-a-number0324/1?page=1&category=Searching&difficulty=Basic&sortBy=submissions)

**2.Sorted array Search.**

Given an array, arr[] sorted in ascending order and an integer k. Return true if k is present in the array, otherwise, false.

**Example 1:**

**Input:** arr[] = [1,2,3,4,6], k=6

**Output:** true

**Explanation:** Since, 6 is present in the array at index4 (0-based indexing), Output is true.

**Example 2:**

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

**Output:** false

**Example 3:**

**Input:** arr[] = [1, 2, 4, 5, 6], k = 6

**Output:** true

**Exlpanation:** Since, 3 is not present in the array, output is false.

**Constraints:**

* + - 1 <= arr.size() <= 106
    - 1 <= k <= 106
    - 1 <= arr[i] <= 106

**Reference:** <https://www.geeksforgeeks.org/problems/who-will-win-1587115621/1>

**3. Find Target Indices After Sorting Array.**

You are given a 0-indexed integer array nums and a target element target.

A target index is an index i such that nums[i] == target.

Return a list of the target indices of nums after sorting nums in non-decreasing order. If there are no target indices, return an empty list. The returned list must be sorted in increasing order.

**Example 1:**

**Input:** nums = [1,2,5,2,3], target = 2

**Output:** [1,2]

**Explanation:** After sorting, nums is [1,2,2,3,5].

The indices where nums[i] == 2 are 1 and 2.

**Example 2:**

**Input:** nums = [1,2,5,2,3], target = 3

**Output:** [3]

**Explanation:** After sorting, nums is [1,2,2,3,5].

The index where nums[i] == 3 is 3.

**Example 3:**

**Input:** nums = [1,2,5,2,3], target = 5

**Output:** [4]

**Explanation:** After sorting, nums is [1,2,2,3,5].

The index where nums[i] == 5 is 4.

**Constraints:**

1 <= nums.length <= 100

1 <= nums[i], target <= 100

**Reference:-** [**https://leetcode.com/problems/find-target-indices-after-sorting-array/description/**](https://leetcode.com/problems/find-target-indices-after-sorting-array/description/)

4. **Search Insert Position.**

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with O(log n) runtime complexity.

**Example 1:**

Input: nums = [1,3,5,6], target = 5

**Output: 2**

Example 2:

**Input**: nums = [1,3,5,6], target = 2

**Output**: 1

**Example 3:**

Input: nums = [1,3,5,6], target = 7

**Output**: 4

**Constraints:**

1 <= nums.length <= 104

-104 <= nums[i] <= 104

nums contains distinct values sorted in ascending order.

-104 <= target <= 104

**Constraints:**

n == seats.length == students.length

1 <= n <= 100

1 <= seats[i], students[j] <= 100

**Reference:** [https://leetcode.com/problems/search-insert-position/description/](https://leetcode.com/problems/search-insert-position/description/%20%20%20%20%20%20%20)

**5.Relative Sort Array.**

Given two arrays arr1 and arr2, the elements of arr2 are distinct, and all elements in arr2 are also in arr1.

Sort the elements of arr1 such that the relative ordering of items in arr1 are the same as in arr2. Elements that do not appear in arr2 should be placed at the end of arr1 in ascending order.

**Example 1:**

**Input:** arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]

**Output:** [2,2,2,1,4,3,3,9,6,7,19]

**Example 2:**

**Input:** arr1 = [28,6,22,8,44,17], arr2 = [22,28,8,6]

**Output:** [22,28,8,6,17,44]

**Constraints:-**

* 1 <= arr1.length, arr2.length <= 1000
* 0 <= arr1[i], arr2[i] <= 1000
* All the elements of arr2 are distinct.
* Each arr2[i] is in arr1.

Reference:-[**https://leetcode.com/problems/relative-sort-array/description/**](https://leetcode.com/problems/relative-sort-array/description/)

**Easy (Questions 6–10)**

**6. Minimum Number of Moves to Seat Everyone**

There are n available seats and n students standing in a room. You are given an array seats of length n, where seats[i] is the position of the ith seat. You are also given the array students of length n, where students[j] is the position of the jth student.

You may perform the following move any number of times:

Increase or decrease the position of the ith student by 1 (i.e., moving the ith student from position x to x + 1 or x - 1)

Return the minimum number of moves required to move each student to a seat such that no two students are in the same seat.

Note that there may be multiple seats or students in the same position at the beginning.

**Example 1:**

**Input:** seats = [3,1,5], students = [2,7,4]

**Output:** 4

**Explanation:** The students are moved as follows:

- The first student is moved from position 2 to position 1 using 1 move.

- The second student is moved from position 7 to position 5 using 2 moves.

- The third student is moved from position 4 to position 3 using 1 move.

In total, 1 + 2 + 1 = 4 moves were used.

**Example 2:**

**Input:** seats = [4,1,5,9], students = [1,3,2,6]

**Output:** 7

**Explanation:** The students are moved as follows:

- The first student is not moved.

- The second student is moved from position 3 to position 4 using 1 move.

- The third student is moved from position 2 to position 5 using 3 moves.

- The fourth student is moved from position 6 to position 9 using 3 moves.

In total, 0 + 1 + 3 + 3 = 7 moves were used.

Ø **Reference:** https://leetcode.com/problems/minimum-number-of-moves-to-seat-everyone/description/

**7.Squares of a Sorted Array**

Given an integer array nums sorted in non-decreasing order, return an array of the squares of each number sorted in non-decreasing order.

**Example 1:**

**Input:** nums = [-4,-1,0,3,10]

**Output:** [0,1,9,16,100]

**Explanation:** After squaring, the array becomes [16,1,0,9,100].

After sorting, it becomes [0,1,9,16,100].

**Example 2:**

**Input:** nums = [-7,-3,2,3,11]

**Output:** [4,9,9,49,121]

**Constraints:**

* 1 <= nums.length <= 104
* -104 <= nums[i] <= 104
* nums is sorted in non-decreasing order.

Reference:-<https://leetcode.com/problems/squares-of-a-sorted-array/description/>

**8. Common in 3 Sorted Arrays.**

You are given three arrays sorted in increasing order. Find the elements that are common in all three arrays.

If there are no such elements return an empty array. In this case, the output will be -1.

Note: can you handle the duplicates without using any additional Data Structure?

**Example1:**

**Input:** arr1 = [1, 5, 10, 20, 40, 80] , arr2 = [6, 7, 20, 80, 100] , arr3 = [3, 4, 15, 20, 30, 70, 80, 120]

**Output:** [20, 80]

**Explanation:** 20 and 80 are the only common elements in arr, brr and crr.

**Example 2:**

**Input:** arr1 = [1, 1, 1, 2, 2, 2], B = [1, 1, 2, 2, 2], arr3 = [1, 1, 1, 1, 2, 2, 2, 2]

**Output:** [1, 2]

**Explanation:** We do not need to consider duplicates

**Example3:**

**Input:** arr1 = [1, 2, 3, 4, 5] , arr2 = [6, 7] , arr3 = [8,9,10]

**Output:** [-1]

**Explanation:** There are no common elements in arr, brr and crr.

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

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

**Constraints:**

1 <= arr1.size(), arr2.size(), arr3.size() <= 105

-105 <= arr1i , arr2i , 1arr3i <= 105

**Reference:** <https://www.geeksforgeeks.org/problems/common-elements1132/1>

**9. Sort Even and Odd Indices Independently.**

You are given a 0-indexed integer array nums. Rearrange the values of nums according to the following rules:

Sort the values at odd indices of nums in non-increasing order.

For example, if nums = [4,1,2,3] before this step, it becomes [4,3,2,1] after. The values at odd indices 1 and 3 are sorted in non-increasing order.

Sort the values at even indices of nums in non-decreasing order.

For example, if nums = [4,1,2,3] before this step, it becomes [2,1,4,3] after. The values at even indices 0 and 2 are sorted in non-decreasing order.

Return the array formed after rearranging the values of nums.

**Example 1:**

**Input:** nums = [4,1,2,3]

**Output:** [2,3,4,1]

**Explanation:**

First, we sort the values present at odd indices (1 and 3) in non-increasing order. So, nums changes from [4,1,2,3] to [4,3,2,1].

Next, we sort the values present at even indices (0 and 2) in non-decreasing order.So, nums changes from [4,1,2,3] to [2,3,4,1].

Thus, the array formed after rearranging the values is [2,3,4,1].

**Example 2:**

**Input:** nums = [2,1]

**Output:** [2,1]

**Explanation:**

Since there is exactly one odd index and one even index, no rearrangement of values takes place.The resultant array formed is [2,1], which is the same as the initial array.

**Constraints:**

* 1 <= nums.length <= 100
* 1 <= nums[i] <= 100

**Reference:-**

[**https://leetcode.com/problems/sort-even-and-odd-indices-independently/description/**](https://leetcode.com/problems/sort-even-and-odd-indices-independently/description/)

**10. Left most and Right most index.**

Given a sorted array with possibly duplicate elements. The task is to find indexes of first and last occurrences of an element X in the given array.

Note: If the element is not present in the array return {-1,-1} as pair.

**Example1:**

**Input:** N = 9

v[] = {1, 3, 5, 5, 5, 5, 67, 123, 125}

X = 5

**Output:**2 5

**Explanation:**

Index of first occurrence of 5 is 2

and index of last occurrence of 5 is 5.

**Example2:**

**Input:**

N = 9

v[] = {1, 3, 5, 5, 5, 5, 7, 123, 125}

X = 7

**Output:**

6 6

**Expected Time Complexity:** O(Log(N))

**Expected Auxiliary Space:** O(1)

Constraints:

1 ≤ N ≤ 105

1 ≤ v[i], X ≤ 1018

**Reference:-** [**https://www.geeksforgeeks.org/problems/find-first-and-last-occurrence-of-x0849/1**](https://www.geeksforgeeks.org/problems/find-first-and-last-occurrence-of-x0849/1)

**Medium (Questions 11–15)**

1. **Search in 2D Matrix.**

You are given an m x n integer matrix matrix with the following two properties:

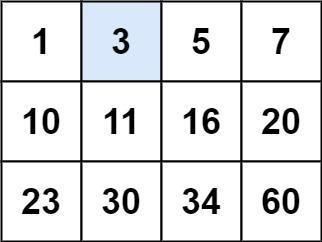
Each row is sorted in non-decreasing order.

The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true if target is in matrix or false otherwise.

You must write a solution in O(log(m \* n)) time complexity.

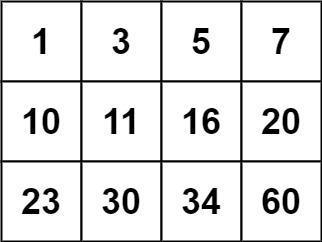
**Example 1:**



**Input:** matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

**Output:** true

**Example2:**



**Input:** matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13

**Output:** false

**Constraints:**

m == matrix.length

n == matrix[i].length

1 <= m, n <= 100

-104 <= matrix[i][j], target <= 104

Ø **Reference:** [https://leetcode.com/problems/search-a-2d-matrix/description/](https://leetcode.com/problems/search-a-2d-matrix/description/?envType=study-plan-v2&envId=top-interview-150)

1. **Find First and Last Position of Element in Sorted Array.**

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.

If target is not found in the array, return [-1, -1].

You must write an algorithm with O(log n) runtime complexity.

**Example 1:**

**Input:** nums = [5,7,7,8,8,10], target = 8

**Output:** [3,4]

**Example 2:**

**Input:** nums = [5,7,7,8,8,10], target = 6

**Output:** [-1,-1]

**Example 3:**

**Input:** nums = [], target = 0

**Output:** [-1,-1]

**Constraints:**

0 <= nums.length <= 105

-109 <= nums[i] <= 109

nums is a non-decreasing array.

-109 <= target <= 109

Ø **Reference:** <https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/description/>

13. **Find Minimum in Rotated Sorted Array.**

Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

* [4,5,6,7,0,1,2] if it was rotated 4 times.
* [0,1,2,4,5,6,7] if it was rotated 7 times.

Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of unique elements, return the minimum element of this array.

You must write an algorithm that runs in O(log n) time.

**Example 1:**

**Input:** nums = [3,4,5,1,2]

**Output:** 1

**Explanation:** The original array was [1,2,3,4,5] rotated 3 times.

**Example 2:**

**Input:** nums = [4,5,6,7,0,1,2]

**Output:** 0

**Explanation:** The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.

**Example 3:**

**Input:** nums = [11,13,15,17]

**Output:** 11

**Explanation:** The original array was [11,13,15,17] and it was rotated 4 times.

**Constraints:**

* n == nums.length
* 1 <= n <= 5000
* -5000 <= nums[i] <= 5000
* All the integers of nums are unique.
* nums is sorted and rotated between 1 and n times.

Refrence:-

<https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/description/>

1. **Smallest Positive Missing Number.**

You are given an integer array arr[]. Your task is to find the smallest positive number missing from the array.

Note: Positive number starts from 1. The array can have negative integers too.

**Examples1:**

**Input:** arr[] = [2, -3, 4, 1, 1, 7]

**Output:** 3

**Explanation**: Smallest positive missing number is 3.

**Examples2:**

**Input:** arr[] = [5, 3, 2, 5, 1]

**Output:** 4

**Explanation:** Smallest positive missing number is 4.

**Examples3:**

**Input:** arr[] = [-8, 0, -1, -4, -3]

**Output:** 1

**Explanation:** Smallest positive missing number is 1.

**Constraints:**

1 <= arr.size() <= 105

-106 <= arr[i] <= 106

**Reference:**<https://www.geeksforgeeks.org/problems/smallest-positive-missing-number-1587115621/1>

1. **Pair Sum Closet to 0.**

Given an integer array of N elements. You need to find the maximum sum of two elements such that sum is closest to zero.

**Example 1:**

**Input:**

N = 3

arr[] = {-8 -66 -60}

**Output:** -68

**Explanation:** Sum of two elements closest to

zero is -68 using numbers -60 and -8.

**Example 2:**

**Input:**

N = 6

arr[] = {-21 -67 -37 -18 4 -65}

**Output**: -14

**Explanation:** Sum of two elements closest to

zero is -14 using numbers -18 and 4.

**Note :** In Case if we have two of more ways to form sum of two elements closest to zero return the maximum sum.

**Your Task:**

You don't need to read input or print anything. You just need to complete the function closestToZero() which takes an array arr[] and its size n as inputs and returns the maximum sum closest to zero that can be formed by summing any two elements in the array.

**Expected Time Complexity:** O(N\*logN).

**Expected Auxiliary Space**: O(1).

**Constraints:**

* 2 ≤ N ≤ 5 \* 105
* -106 ≤ arr[i] ≤ 106

**Reference:**<https://www.geeksforgeeks.org/problems/two-numbers-with-sum-closest-to-zero1737/1>

**Hard (Questions 16–20)**

1. **Sort Items by Groups Respecting Dependencies**

There are n items each belonging to zero or one of m groups where group[i] is the group that the i-th item belongs to and it's equal to -1 if the i-th item belongs to no group. The items and the groups are zero indexed. A group can have no item belonging to it.

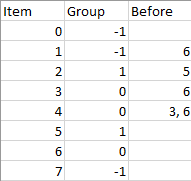
Return a sorted list of the items such that:

The items that belong to the same group are next to each other in the sorted list.

There are some relations between these items where beforeItems[i] is a list containing all the items that should come before the i-th item in the sorted array (to the left of the i-th item).

Return any solution if there is more than one solution and return an empty list if there is no solution.

**Example 1:**



**Input:** n = 8, m = 2, group = [-1,-1,1,0,0,1,0,-1], beforeItems = [[],[6],[5],[6],[3,6],[],[],[]]

**Output:** [6,3,4,1,5,2,0,7]

**Example 2:**

**Input:** n = 8, m = 2, group = [-1,-1,1,0,0,1,0,-1], beforeItems = [[],[6],[5],[6],[3],[],[4],[]]

**Output:** []

**Explanation:** This is the same as example 1 except that 4 needs to be before 6 in the sorted list.

**Constraints:**

* 1 <= m <= n <= 3 \* 104
* group.length == beforeItems.length == n
* -1 <= group[i] <= m - 1
* 0 <= beforeItems[i].length <= n - 1
* 0 <= beforeItems[i][j] <= n - 1
* i != beforeItems[i][j]
* beforeItems[i] does not contain duplicates elements.

Reference:-

<https://leetcode.com/problems/sort-items-by-groups-respecting-dependencies/description/>

1. **Find the Kth Smallest Sum of a Matrix With Sorted Rows.**

You are given an m x n matrix mat that has its rows sorted in non-decreasing order and an integer k.

You are allowed to choose exactly one element from each row to form an array.

Return the kth smallest array sum among all possible arrays.

**Example 1:**

**Input:** mat = [[1,3,11],[2,4,6]], k = 5

**Output:** 7

**Explanation**: Choosing one element from each row, the first k smallest sum are:

[1,2], [1,4], [3,2], [3,4], [1,6]. Where the 5th sum is 7.

**Example 2:**

**Input:** mat = [[1,3,11],[2,4,6]], k = 9

**Output:** 17

**Example 3:**

**Input:** mat = [[1,10,10],[1,4,5],[2,3,6]], k = 7

**Output:** 9

**Explanation:** Choosing one element from each row, the first k smallest sum are:

[1,1,2], [1,1,3], [1,4,2], [1,4,3], [1,1,6], [1,5,2], [1,5,3]. Where the 7th sum is 9.

**Constraints:**

* m == mat.length
* n == mat.length[i]
* 1 <= m, n <= 40
* 1 <= mat[i][j] <= 5000
* 1 <= k <= min(200, nm)
* mat[i] is a non-decreasing array.

Reference:-

<https://leetcode.com/problems/find-the-kth-smallest-sum-of-a-matrix-with-sorted-rows/description/>

1. **Merge k Sorted Lists.**

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

**Example 1:**

**Input:** lists = [[1,4,5],[1,3,4],[2,6]]

**Output:** [1,1,2,3,4,4,5,6]

**Explanation:** The linked-lists are:

[

1->4->5,

1->3->4,

2->6

]

merging them into one sorted list:

1->1->2->3->4->4->5->6

**Example 2:**

**Input:** lists = []

**Output:** []

**Example 3:**

**Input:** lists = [[]]

**Output:** []

**Constraints:**

* k == lists.length
* 0 <= k <= 104
* 0 <= lists[i].length <= 500
* -104 <= lists[i][j] <= 104
* lists[i] is sorted in ascending order.
* The sum of lists[i].length will not exceed 104.

Reference:-<https://leetcode.com/problems/merge-k-sorted-lists/description/>

19. **Max Chunks To Make Sorted II**

You are given an integer array arr.

We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.

Return the largest number of chunks we can make to sort the array.

**Example 1:**

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

**Output:** 1

**Explanation:**

Splitting into two or more chunks will not return the required result.

For example, splitting into [5, 4], [3, 2, 1] will result in [4, 5, 1, 2, 3], which isn't sorted.

**Example 2:**

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

**Output:** 4

**Explanation:**

We can split into two chunks, such as [2, 1], [3, 4, 4].

However, splitting into [2, 1], [3], [4], [4] is the highest number of chunks possible.

**Constraints:**

* 1 <= arr.length <= 2000
* 0 <= arr[i] <= 108

Reference:-<https://leetcode.com/problems/max-chunks-to-make-sorted-ii/description/>

**20. Max Chunks To Make Sorted II**

You are given an integer array arr.

We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.

Return the largest number of chunks we can make to sort the array.

**Example 1:**

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

**Output:** 1

**Explanation:**

Splitting into two or more chunks will not return the required result.

For example, splitting into [5, 4], [3, 2, 1] will result in [4, 5, 1, 2, 3], which isn't sorted.

**Example 2:**

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

**Output:** 4

**Explanation:**

We can split into two chunks, such as [2, 1], [3, 4, 4].

However, splitting into [2, 1], [3], [4], [4] is the highest number of chunks possible.

**Constraints:**

* 1 <= arr.length <= 2000
* 0 <= arr[i] <= 108

**Very Hard (Questions 21–25)**

**21.Find Minimum in Rotated Sorted Array II.**

Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,4,4,5,6,7] might become:

[4,5,6,7,0,1,4] if it was rotated 4 times.

[0,1,4,4,5,6,7] if it was rotated 7 times.

Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums that may contain duplicates, return the minimum element of this array.

You must decrease the overall operation steps as much as possible.

**Example 1:**

**Input:** nums = [1,3,5]

**Output:** 1

**Example 2:**

**Input:** nums = [2,2,2,0,1]

**Output:** 0

**Constraints:**

* n == nums.length
* 1 <= n <= 5000
* -5000 <= nums[i] <= 5000
* nums is sorted and rotated between 1 and n times.

**Reference:-**

[**https://leetcode.com/problems/find-minimum-in-rotated-sorted-array-ii/description/**](https://leetcode.com/problems/find-minimum-in-rotated-sorted-array-ii/description/)

**22.Median of Two Sorted Arrays.**

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

**Example 1:**

**Input:** nums1 = [1,3], nums2 = [2]

**Output:** 2.00000

**Explanation:** merged array = [1,2,3] and median is 2.

**Example 2:**

**Input:** nums1 = [1,2], nums2 = [3,4]

**Output:** 2.50000

**Explanation:** merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

**Constraints:**

* nums1.length == m
* nums2.length == n
* 0 <= m <= 1000
* 0 <= n <= 1000
* 1 <= m + n <= 2000
* -106 <= nums1[i], nums2[i] <= 106

**Rererence:-** [**https://leetcode.com/problems/median-of-two-sorted-arrays/description/**](https://leetcode.com/problems/median-of-two-sorted-arrays/description/)

**23. Create Sorted Array through Instructions.**

Given an integer array instructions, you are asked to create a sorted array from the elements in instructions. You start with an empty container nums. For each element from left to right in instructions, insert it into nums. The cost of each insertion is the minimum of the following:

The number of elements currently in nums that are strictly less than instructions[i].

The number of elements currently in nums that are strictly greater than instructions[i].

For example, if inserting element 3 into nums = [1,2,3,5], the cost of insertion is min(2, 1) (elements 1 and 2 are less than 3, element 5 is greater than 3) and nums will become [1,2,3,3,5].

Return the total cost to insert all elements from instructions into nums. Since the answer may be large, return it modulo 109 + 7

**Example 1:**

**Input:** instructions = [1,5,6,2]

**Output:** 1

**Explanation:** Begin with nums = [].

Insert 1 with cost min(0, 0) = 0, now nums = [1].

Insert 5 with cost min(1, 0) = 0, now nums = [1,5].

Insert 6 with cost min(2, 0) = 0, now nums = [1,5,6].

Insert 2 with cost min(1, 2) = 1, now nums = [1,2,5,6].

The total cost is 0 + 0 + 0 + 1 = 1.

**Example 2:**

**Input:** instructions = [1,2,3,6,5,4]

**Output: 3**

**Explanation:** Begin with nums = [].

Insert 1 with cost min(0, 0) = 0, now nums = [1].

Insert 2 with cost min(1, 0) = 0, now nums = [1,2].

Insert 3 with cost min(2, 0) = 0, now nums = [1,2,3].

Insert 6 with cost min(3, 0) = 0, now nums = [1,2,3,6].

Insert 5 with cost min(3, 1) = 1, now nums = [1,2,3,5,6].

Insert 4 with cost min(3, 2) = 2, now nums = [1,2,3,4,5,6].

The total cost is 0 + 0 + 0 + 0 + 1 + 2 = 3.

**Example 3:**

**Input:** instructions = [1,3,3,3,2,4,2,1,2]

**Output:** 4

**Explanation:** Begin with nums = [].

Insert 1 with cost min(0, 0) = 0, now nums = [1].

Insert 3 with cost min(1, 0) = 0, now nums = [1,3].

Insert 3 with cost min(1, 0) = 0, now nums = [1,3,3].

Insert 3 with cost min(1, 0) = 0, now nums = [1,3,3,3].

Insert 2 with cost min(1, 3) = 1, now nums = [1,2,3,3,3].

Insert 4 with cost min(5, 0) = 0, now nums = [1,2,3,3,3,4].

​​​​​​​Insert 2 with cost min(1, 4) = 1, now nums = [1,2,2,3,3,3,4].

​​​​​​​Insert 1 with cost min(0, 6) = 0, now nums = [1,1,2,2,3,3,3,4].

​​​​​​​Insert 2 with cost min(2, 4) = 2, now nums = [1,1,2,2,2,3,3,3,4].

The total cost is 0 + 0 + 0 + 0 + 1 + 0 + 1 + 0 + 2 = 4.

**Constraints:**

* 1 <= instructions.length <= 105
* 1 <= instructions[i] <= 105

Reference:- <https://leetcode.com/problems/create-sorted-array-through-instructions/description/>

**24. Kth Smallest Product of Two Sorted Arrays.**

Given two sorted 0-indexed integer arrays nums1 and nums2 as well as an integer k, return the kth (1-based) smallest product of nums1[i] \* nums2[j] where 0 <= i < nums1.length and 0 <= j < nums2.length.

**Example 1:**

**Input:** nums1 = [2,5], nums2 = [3,4], k = 2

**Output:** 8

**Explanation:** The 2 smallest products are:

- nums1[0] \* nums2[0] = 2 \* 3 = 6

- nums1[0] \* nums2[1] = 2 \* 4 = 8

The 2nd smallest product is 8.

**Example 2:**

**Input:** nums1 = [-4,-2,0,3], nums2 = [2,4], k = 6

**Output:** 0

**Explanation:** The 6 smallest products are:

- nums1[0] \* nums2[1] = (-4) \* 4 = -16

- nums1[0] \* nums2[0] = (-4) \* 2 = -8

- nums1[1] \* nums2[1] = (-2) \* 4 = -8

- nums1[1] \* nums2[0] = (-2) \* 2 = -4

- nums1[2] \* nums2[0] = 0 \* 2 = 0

- nums1[2] \* nums2[1] = 0 \* 4 = 0

The 6th smallest product is 0.

**Example 3:**

**Input:** nums1 = [-2,-1,0,1,2], nums2 = [-3,-1,2,4,5], k = 3

**Output:** -6

**Explanation:** The 3 smallest products are:

- nums1[0] \* nums2[4] = (-2) \* 5 = -10

- nums1[0] \* nums2[3] = (-2) \* 4 = -8

- nums1[4] \* nums2[0] = 2 \* (-3) = -6

The 3rd smallest product is -6.

**Constraints:**

* 1 <= nums1.length, nums2.length <= 5 \* 104
* -105 <= nums1[i], nums2[j] <= 105
* 1 <= k <= nums1.length \* nums2.length
* nums1 and nums2 are sorted.

Reference:-<https://leetcode.com/problems/kth-smallest-product-of-two-sorted-arrays/description/>

**25. Sorted GCD Pair Queries.**

You are given an integer array nums of length n and an integer array queries.

Let gcdPairs denote an array obtained by calculating the GCD of all possible pairs (nums[i], nums[j]), where 0 <= i < j < n, and then sorting these values in ascending order.

For each query queries[i], you need to find the element at index queries[i] in gcdPairs.

Return an integer array answer, where answer[i] is the value at gcdPairs[queries[i]] for each query.

The term gcd(a, b) denotes the greatest common divisor of a and b.

**Example 1:**

**Input:** nums = [2,3,4], queries = [0,2,2]

**Output:** [1,2,2]

**Explanation:**

gcdPairs = [gcd(nums[0], nums[1]), gcd(nums[0], nums[2]), gcd(nums[1], nums[2])] = [1, 2, 1].

After sorting in ascending order, gcdPairs = [1, 1, 2].

So, the answer is [gcdPairs[queries[0]], gcdPairs[queries[1]], gcdPairs[queries[2]]] = [1, 2, 2].

**Example 2:**

**Input:** nums = [4,4,2,1], queries = [5,3,1,0]

**Output:** [4,2,1,1]

**Explanation:**

gcdPairs sorted in ascending order is [1, 1, 1, 2, 2, 4].

**Example 3:**

**Input:** nums = [2,2], queries = [0,0]

**Output:** [2,2]

**Explanation:**

gcdPairs = [2].

**Constraints:**

* 2 <= n == nums.length <= 105
* 1 <= nums[i] <= 5 \* 104
* 1 <= queries.length <= 105
* 0 <= queries[i] < n \* (n - 1) / 2

Reference:- <https://leetcode.com/problems/sorted-gcd-pair-queries/description/>