Arrays Assignment

Instructions: Follow the same pattern as before —
1. Dry run each input on paper,
2. Write and test your code,
3. Submit both your dry run and final working code.

1. Count Equal and Divisible Pairs in an Array

Given a 0-indexed integer array `nums` of length `n` and an integer `k`, return the number of pairs `(i, j)` where `0 <= i < j < n`, such that `nums[i] == nums[j]` and `(i * j)` is divisible by `k`.

Example 1:

Input:
`nums = [3,1,2,2,2,1,3], k = 2`
Output:
`4`

Explanation:
There are 4 pairs that meet all the requirements:

- `nums[0] == nums[6]`, and `0 * 6 == 0`, which is divisible by 2.
- `nums[2] == nums[3]`, and `2 * 3 == 6`, which is divisible by 2.
- `nums[2] == nums[4]`, and `2 * 4 == 8`, which is divisible by 2.
- `nums[3] == nums[4]`, and `3 * 4 == 12`, which is divisible by 2.

Example 2:

Input:
`nums = [1,2,3,4], k = 1`
Output:
`0`

Explanation:
Since no value in `nums` is repeated, there are no pairs `(i,j)` that meet all the requirements.

Constraints:

- `1 <= nums.length <= 100`
- `1 <= nums[i], k <= 100`

2. Find the Highest Altitude

There is a biker going on a road trip. The road trip consists of `n + 1` points at different altitudes. The biker starts his trip on point `0` with altitude equal `0`.

You are given an integer array `gain` of length `n` where `gain[i]` is the net gain in altitude between points `i` and `i + 1` for all `(0 <= i < n)`. Return the highest altitude of a point.

Example 1:

Input:
`gain = [-5,1,5,0,-7]`
Output:
`1`

Explanation:
The altitudes are `[0,-5,-4,1,1,-6]`. The highest is `1`.

Example 2:

Input:
`gain = [-4,-3,-2,-1,4,3,2]`
Output:
`0`

Explanation:
The altitudes are `[0,-4,-7,-9,-10,-6,-3,-1]`. The highest is `0`.

Constraints:

- `n == gain.length`
- `1 <= n <= 100`
- `-100 <= gain[i] <= 100`

3. Matrix Diagonal Sum

Given a square matrix `mat`, return the sum of the matrix diagonals.

Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

Example 1:

Input:

```
mat = [[1,2,3],
       [4,5,6],
       [7,8,9]]
```

Output:

```
25
```

Explanation:
Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25
Notice that element `mat[1][1] = 5` is counted only once.

Example 2:

Input:

```
mat = [[1,1,1,1],
       [1,1,1,1],
       [1,1,1,1],
       [1,1,1,1]]
```

Output:

```
8
```

Example 3:

Input:

```
mat = [[5]]
```

Output:

```
5
```

Constraints:

- `n == mat.length == mat[i].length`
- `1 <= n <= 100`

- `1 <= mat[i][j] <= 100`

4. Find the Difference of Two Arrays

Given two 0-indexed integer arrays `nums1` and `nums2`, return a list `answer` of size `2` where:

- `answer[0]` is a list of all distinct integers in `nums1` which are not present in `nums2`.
- `answer[1]` is a list of all distinct integers in `nums2` which are not present in `nums1`.

Note that the integers in the lists may be returned in any order.

Example 1:

Input:
`nums1 = [1,2,3]`, `nums2 = [2,4,6]`
Output:
`[[1,3],[4,6]]`

Explanation:
For `nums1`, `nums1[1] = 2` is present at index `0` of `nums2`, whereas `nums1[0] = 1` and `nums1[2] = 3` are not present in `nums2`. Therefore, `answer[0] = [1,3]`.
For `nums2`, `nums2[0] = 2` is present at index `1` of `nums1`, whereas `nums2[1] = 4` and `nums2[2] = 6` are not present in `nums1`. Therefore, `answer[1] = [4,6]`.

Example 2:

Input:
`nums1 = [1,2,3,3]`, `nums2 = [1,1,2,2]`
Output:
`[[3],[]]`

Explanation:
For `nums1`, `nums1[2]` and `nums1[3]` are not present in `nums2`. Since `nums1[2] == nums1[3]`, their value is only included once and `answer[0] = [3]`.
Every integer in `nums2` is present in `nums1`. Therefore, `answer[1] = []`.

Constraints:

- `1 <= nums1.length, nums2.length <= 1000`
- `-1000 <= nums1[i], nums2[i] <= 1000`

5. Minimum Operations to Make the Array Increasing

You are given an integer array `nums` (0-indexed). In one operation, you can choose an element of the array and increment it by 1.

For example, if `nums = [1,2,3]`, you can choose to increment `nums[1]` to make `nums = [1,3,3]`.
Return the minimum number of operations needed to make `nums` strictly increasing.

An array `nums` is strictly increasing if `nums[i] < nums[i+1]` for all `0 <= i < nums.length - 1`.
An array of length `1` is trivially strictly increasing.

Example 1:

Input:
`nums = [1,1,1]`
Output:
`3`
Explanation:
You can do the following operations:

1. Increment `nums[2]`, so `nums` becomes `[1,1,2]`.
2. Increment `nums[1]`, so `nums` becomes `[1,2,2]`.
3. Increment `nums[2]`, so `nums` becomes `[1,2,3]`.

Example 2:

Input:
`nums = [1,5,2,4,1]`
Output:
`14`

Example 3:

Input:
`nums = [8]`
Output:
`0`

Constraints:

- `1 <= nums.length <= 5000`

- `1 <= nums[i] <=` 10^4

6. Lucky Numbers in a Matrix

Given an `m x n` matrix of distinct numbers, return all **lucky numbers** in the matrix in **any order**.

A **lucky number** is an element of the matrix such that it is the **minimum element in its row** and **maximum in its column**.

Example 1:

Input:
`matrix = [[3,7,8],[9,11,13],[15,16,17]]`
Output:
`[15]`
Explanation:
15 is the only lucky number since it is the minimum in its row and the maximum in its column.

Example 2:

Input:
`matrix = [[1,10,4,2],[9,3,8,7],[15,16,17,12]]`
Output:
`[12]`
Explanation:
12 is the only lucky number since it is the minimum in its row and the maximum in its column.

Example 3:

Input:
`matrix = [[7,8],[1,2]]`
Output:
`[7]`
Explanation:
7 is the only lucky number since it is the minimum in its row and the maximum in its column.

Constraints:

- `m == matrix.length`
- `n == matrix[i].length`
- `1 <= n, m <= 50`
- `1 <= matrix[i][j] <=` 10^5
- All elements in the matrix are **distinct**

7. Partition Array According to Given Pivot

You are given a 0-indexed integer array `nums` and an integer `pivot`. Rearrange `nums` such that the following conditions are satisfied:

- Every element **less than pivot** appears before every element **greater than pivot**.
- Every element **equal to pivot** appears **in between** the elements less than and greater than pivot.
- The **relative order** of the elements less than pivot and the elements greater than pivot is **maintained**.

More formally, consider every `pi`, `pj` where `pi` is the new position of the `i`th element and `pj` is the new position of the `j`th element. If `i < j` and both elements are smaller (or larger) than pivot, then `pi < pj`.

Return `nums` after the rearrangement.

Example 1:

Input:
`nums = [9,12,5,10,14,3,10]`, `pivot = 10`
Output:
`[9,5,3,10,10,12,14]`
Explanation:
The elements 9, 5, and 3 are less than the pivot so they are on the left side of the array.
The elements 12 and 14 are greater than the pivot so they are on the right side of the array.
The relative ordering of the elements less than and greater than pivot is also maintained. [9, 5, 3] and [12, 14] are the respective orderings.

Example 2:

Input:
`nums = [-3,4,3,2]`, `pivot = 2`
Output:
`[-3,2,4,3]`
Explanation:
The element -3 is less than the pivot so it is on the left side of the array.
The elements 4 and 3 are greater than the pivot so they are on the right side of the array.
The relative ordering of the elements less than and greater than pivot is also maintained. [-3] and [4, 3] are the respective orderings.

Constraints:

- `1 <= nums.length <=` 10^5
- `-`10^6 `<= nums[i] <=` 10^6
- `pivot` equals to an element of `nums`

8. Minimum Operations to Make Binary Array Elements Equal to One I

You are given a **binary array** `nums`.

You can do the following operation on the array any number of times (possibly zero):

Choose any 3 consecutive elements from the array and **flip all of them**.
Flipping an element means changing its value from `0` to `1`, and from `1` to `0`.

Return the **minimum number of operations** required to make **all elements in `nums` equal to 1**.
If it is **impossible**, return `-1`.

Example 1:

Input:
`nums = [0,1,1,1,0,0]`
Output:
`3`

Explanation:
We can do the following operations:

1. Choose elements at indices `0,1,2` → `nums = [1,0,0,1,0,0]`
2. Choose elements at indices `1,2,3` → `nums = [1,1,1,0,0,0]`
3. Choose elements at indices `3,4,5` → `nums = [1,1,1,1,1,1]`

Example 2:

Input:
`nums = [0,1,1,1]`
Output:
`-1`

Explanation:
It is **impossible** to make all elements equal to 1 using only the allowed operation.

Constraints:

- `3 <= nums.length <= 10`^{5}
- `0 <= nums[i] <= 1`

9. Watering Plants

You want to water n plants in your garden with a watering can. The plants are arranged in a row and are labeled from 0 to n - 1 from left to right where the ith plant is located at x = i. There is a river at x = -1 that you can refill your watering can at.

Each plant needs a specific amount of water. You will water the plants in the following way:

- Water the plants in order from left to right.
- After watering the current plant, if you do not have enough water to completely water the next plant, return to the river to fully refill the watering can.
- You cannot refill the watering can early.
- You are initially at the river (i.e., x = -1). It takes one step to move one unit on the x-axis.

Given a 0-indexed integer array plants of n integers, where plants[i] is the amount of water the ith plant needs, and an integer capacity representing the watering can capacity, return the number of steps needed to water all the plants.

Example 1:
Input: plants = [2,2,3,3], capacity = 5
Output: 14
Explanation: Start at the river with a full watering can:

- Walk to plant 0 (1 step) and water it. Watering can has 3 units of water.
- Walk to plant 1 (1 step) and water it. Watering can has 1 unit of water.
- Since you cannot completely water plant 2, walk back to the river to refill (2 steps).
- Walk to plant 2 (3 steps) and water it. Watering can has 2 units of water.
- Since you cannot completely water plant 3, walk back to the river to refill (3 steps).
- Walk to plant 3 (4 steps) and water it.
  Steps needed = 1 + 1 + 2 + 3 + 3 + 4 = 14.

Example 2:
Input: plants = [1,1,1,4,2,3], capacity = 4

Output: 30
Explanation: Start at the river with a full watering can:

- Water plants 0, 1, and 2 (3 steps). Return to river (3 steps).
- Water plant 3 (4 steps). Return to river (4 steps).
- Water plant 4 (5 steps). Return to river (5 steps).
- Water plant 5 (6 steps).
  Steps needed = 3 + 3 + 4 + 4 + 5 + 5 + 6 = 30.

Example 3:
Input: plants = [7,7,7,7,7,7,7], capacity = 8
Output: 49
Explanation: You have to refill before watering each plant.
Steps needed = 1 + 1 + 2 + 2 + 3 + 3 + 4 + 4 + 5 + 5 + 6 + 6 + 7 = 49.

Constraints:

- n == plants.length
- 1 <= n <= 1000
- 1 <= plants[i] <= 10^6
- max(plants[i]) <= capacity <= 10^9

10. Find the Score of All Prefixes of an Array

We define the conversion array conver of an array arr as follows:

conver[i] = arr[i] + max(arr[0..i]) where max(arr[0..i]) is the maximum value of arr[j] over 0 <= j <= i.
We also define the score of an array arr as the sum of the values of the conversion array of arr.

Given a 0-indexed integer array nums of length n, return an array ans of length n where ans[i] is the score of the prefix nums[0..i].

Example 1:

Input: nums = [2,3,7,5,10]
Output: [4,10,24,36,56]
Explanation:
For the prefix [2], the conversion array is [4] hence the score is 4
For the prefix [2, 3], the conversion array is [4, 6] hence the score is 10
For the prefix [2, 3, 7], the conversion array is [4, 6, 14] hence the score is 24
For the prefix [2, 3, 7, 5], the conversion array is [4, 6, 14, 12] hence the score is 36
For the prefix [2, 3, 7, 5, 10], the conversion array is [4, 6, 14, 12, 20] hence the score is 56

Example 2:

Input: nums = [1,1,2,4,8,16]
Output: [2,4,8,16,32,64]
Explanation:
For the prefix [1], the conversion array is [2] hence the score is 2
For the prefix [1, 1], the conversion array is [2, 2] hence the score is 4
For the prefix [1, 1, 2], the conversion array is [2, 2, 4] hence the score is 8
For the prefix [1, 1, 2, 4], the conversion array is [2, 2, 4, 8] hence the score is 16
For the prefix [1, 1, 2, 4, 8], the conversion array is [2, 2, 4, 8, 16] hence the score is 32
For the prefix [1, 1, 2, 4, 8, 16], the conversion array is [2, 2, 4, 8, 16, 32] hence the score is 64

Constraints:

1 <= nums.length <= 10^5
1 <= nums[i] <= 10^9

11. Split the Array

You are given an integer array nums of even length. You have to split the array into two parts nums1 and nums2 such that:

- nums1.length == nums2.length == nums.length / 2.
- nums1 should contain distinct elements.
- nums2 should also contain distinct elements.

Return true if it is possible to split the array, and false otherwise.

Example 1:

Input: nums = [1,1,2,2,3,4]
Output: true
Explanation: One of the possible ways to split nums is nums1 = [1,2,3] and nums2 = [1,2,4].

Example 2:

Input: nums = [1,1,1,1]
Output: false
Explanation: The only possible way to split nums is nums1 = [1,1] and nums2 = [1,1]. Both nums1 and nums2 do not contain distinct elements. Therefore, we return false.

Constraints:

1 <= nums.length <= 100
nums.length % 2 == 0
1 <= nums[i] <= 100

12. Find the Middle Index in Array

Given a 0-indexed integer array nums, find the leftmost middleIndex (i.e., the smallest amongst all the possible ones).

A middleIndex is an index where nums[0] + nums[1] + ... + nums[middleIndex-1] == nums[middleIndex+1] + nums[middleIndex+2] + ... + nums[nums.length-1].

If middleIndex == 0, the left side sum is considered to be 0. Similarly, if middleIndex == nums.length - 1, the right side sum is considered to be 0.

Return the leftmost middleIndex that satisfies the condition, or -1 if there is no such index.

Example 1:

Input: nums = [2,3,-1,8,4]
Output: 3
Explanation: The sum of the numbers before index 3 is: 2 + 3 + -1 = 4
The sum of the numbers after index 3 is: 4 = 4

Example 2:

Input: nums = [1,-1,4]
Output: 2
Explanation: The sum of the numbers before index 2 is: 1 + -1 = 0
The sum of the numbers after index 2 is: 0

Example 3:

Input: nums = [2,5]
Output: -1
Explanation: There is no valid middleIndex.

Constraints:

1 <= nums.length <= 100
-1000 <= nums[i] <= 1000

13. Contiguous Array

Given a binary array nums, return the maximum length of a contiguous subarray with an equal number of 0 and 1.

Example 1:

Input: nums = [0,1]
Output: 2
Explanation: [0, 1] is the longest contiguous subarray with an equal number of 0 and 1.

Example 2:

Input: nums = [0,1,0]
Output: 2
Explanation: [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.

Example 3:

Input: nums = [0,1,1,1,1,1,0,0,0]
Output: 6
Explanation: [1,1,1,0,0,0] is the longest contiguous subarray with equal number of 0 and 1.

Constraints:

1 <= nums.length <= 10^5
nums[i] is either 0 or 1.

14. Removing Minimum and Maximum From Array

You are given a 0-indexed array of distinct integers nums.

There is an element in nums that has the lowest value and an element that has the highest value. We call them the minimum and maximum respectively. Your goal is to remove both these elements from the array.

A deletion is defined as either removing an element from the front of the array or removing an element from the back of the array.

Return the minimum number of deletions it would take to remove both the minimum and maximum element from the array.

Example 1:

Input: nums = [2,10,7,5,4,1,8,6]
Output: 5
Explanation:
The minimum element in the array is nums[5], which is 1.
The maximum element in the array is nums[1], which is 10.
We can remove both the minimum and maximum by removing 2 elements from the front and 3 elements from the back.
This results in 2 + 3 = 5 deletions, which is the minimum number possible.

Example 2:

Input: nums = [0,-4,19,1,8,-2,-3,5]
Output: 3
Explanation:
The minimum element in the array is nums[1], which is -4.
The maximum element in the array is nums[2], which is 19.

We can remove both the minimum and maximum by removing 3 elements from the front.
This results in only 3 deletions, which is the minimum number possible.

Example 3:

Input: nums = [101]
Output: 1
Explanation:
There is only one element in the array, which makes it both the minimum and maximum element.
We can remove it with 1 deletion.

Constraints:

1 <= nums.length <= 10^5
-10^5 <= nums[i] <= 10^5
The integers in nums are distinct.

15. Corporate Flight Bookings

There are n flights that are labeled from 1 to n.

You are given an array of flight bookings bookings, where bookings[i] = [firsti, lasti, seatsi] represents a booking for flights firsti through lasti (inclusive) with seatsi seats reserved for each flight in the range.

Return an array answer of length n, where answer[i] is the total number of seats reserved for flight i.

Example 1:

Input: bookings = [[1,2,10],[2,3,20],[2,5,25]], n = 5
Output: [10,55,45,25,25]
Explanation:
Flight labels: 1 2 3 4 5
Booking 1 reserved: 10 10
Booking 2 reserved: 20 20
Booking 3 reserved: 25 25 25 25
Total seats: 10 55 45 25 25
Hence, answer = [10,55,45,25,25]

Example 2:

Input: bookings = [[1,2,10],[2,2,15]], n = 2
Output: [10,25]
Explanation:
Flight labels: 1 2
Booking 1 reserved: 10 10
Booking 2 reserved: 15
Total seats: 10 25
Hence, answer = [10,25]

Constraints:

1 <= n <= 2 * 10^4
1 <= bookings.length <= 2 * 10^4
bookings[i].length == 3
1 <= firsti <= lasti <= n
1 <= seatsi <= 10^4

16. Maximum Matching of Players With Trainers

You are given a 0-indexed integer array players, where players[i] represents the ability of the ith player. You are also given a 0-indexed integer array trainers, where trainers[j] represents the training capacity of the jth trainer.

The ith player can match with the jth trainer if the player's ability is less than or equal to the trainer's training capacity. Additionally, the ith player can be matched with at most one trainer, and the jth trainer can be matched with at most one player.

Return the maximum number of matchings between players and trainers that satisfy these conditions.

Example 1:

Input: players = [4,7,9], trainers = [8,2,5,8]
Output: 2
Explanation:
One of the ways we can form two matchings is as follows:

- players[0] can be matched with trainers[0] since 4 <= 8.
- players[1] can be matched with trainers[3] since 7 <= 8.
  It can be proven that 2 is the maximum number of matchings that can be formed.

Example 2:

Input: players = [1,1,1], trainers = [10]
Output: 1
Explanation:
The trainer can be matched with any of the 3 players.
Each player can only be matched with one trainer, so the maximum answer is 1.

Constraints:

1 <= players.length, trainers.length <= 10^5
1 <= players[i], trainers[j] <= 10^9

17. Reverse Pairs

You are given an integer array nums. Return the number of reverse pairs in the array.

A reverse pair is a pair (i, j) where:

- 0 <= i < j < nums.length, and
- nums[i] > 2 * nums[j].

Example 1:

Input: nums = [1,3,2,3,1]
Output: 2
Explanation: The reverse pairs are:

- (1, 4) --> nums[1] = 3, nums[4] = 1, 3 > 2 * 1
- (3, 4) --> nums[3] = 3, nums[4] = 1, 3 > 2 * 1

Example 2:

Input: nums = [2,4,3,5,1]
Output: 3
Explanation: The reverse pairs are:

- (1, 4) --> nums[1] = 4, nums[4] = 1, 4 > 2 * 1
- (2, 4) --> nums[2] = 3, nums[4] = 1, 3 > 2 * 1
- (3, 4) --> nums[3] = 5, nums[4] = 1, 5 > 2 * 1

Constraints:

- 1 <= nums.length <= 5 * 10^4
- -2^{31} <= nums[i] <= 2^{31} - 1

18. Count Inversions in an Array

Problem Statement

Given an array of **N** integers, count the number of inversion pairs in the array.

An inversion is defined as a pair `(i, j)` such that:

- `0 <= i < j < N`, and
- `arr[i] > arr[j]`

Return the total number of inversions in the array.

Input Format

- First line: Integer `N` — size of the array
- Second line: `N` space-separated integers — elements of the array

Output Format

- A single integer representing the count of inversion pairs.

Constraints

- `1 <= N <= 10^5`
- `-10^9 <= arr[i] <= 10^9`

Example 1:

Input:
5
1 2 3 4 5

Output:
0

Explanation:
The array is sorted, so no pairs `(i,j)` exist with `arr[i] > arr[j]`.

Example 2:

Input:
5
5 4 3 2 1

Output:
10

Explanation:
The array is reverse sorted. Every pair is an inversion. Total inversions = 10.

Example 3:

Input:
5
5 3 2 1 4

Output:
7

Explanation:
Inversion pairs are (5,3), (5,2), (5,1), (5,4), (3,2), (3,1), (2,1) totaling 7.

19. Find First and Last Position of Element in Sorted Array

Problem Statement

You are given an array of integers `nums` **sorted in non-decreasing order**, and an integer `target`.

Find the **starting and ending position** of the 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.

Input Format

- An array `nums` of integers sorted in non-decreasing order.
- An integer `target`.

Output Format

- A list with two integers representing the first and last index of the target in the array.

Constraints

- `0 <= nums.length <= 10^5`
- `-10^9 <= nums[i] <= 10^9`
- `nums` is a non-decreasing array.
- `-10^9 <= target <= 10^9`

Example 1:

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

Output:
[3,4]

Explanation:
Target `8` appears at indices `3` and `4`.

Example 2:

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

Output:
[-1,-1]

Explanation:
Target `6` is not present in the array.

Example 3:

Input:
nums = [], target = 0

Output:
[-1,-1]

Explanation:
Empty array, so target is not found.

20. Merge Intervals

Problem Statement

Given an array of intervals where `intervals[i] = [starti, endi]`, merge all **overlapping** intervals, and return an array of the **non-overlapping** intervals that cover all the intervals in the input.

Input Format

- An array `intervals` where each element is a list of two integers `[starti, endi]`.

Output Format

- A list of merged non-overlapping intervals.

Constraints

- `1 <= intervals.length <= 10^4`
- `intervals[i].length == 2`
- `0 <= starti <= endi <= 10^4`

Example 1:

Input:
intervals = [[1,3],[2,6],[8,10],[15,18]]

Output:
[[1,6],[8,10],[15,18]]

Explanation:
Since intervals [1,3] and [2,6] overlap, they are merged into [1,6].

Example 2:

Input:
intervals = [[1,4],[4,5]]

Output:
[[1,5]]

Explanation:
Intervals [1,4] and [4,5] are considered overlapping since the end of one interval touches the start of the next.