Question 1

2281. Sum of Total Strength of Wizards

Problem Statement : As the ruler of a kingdom, you have an army of wizards at your command.

You are given a 0-indexed integer array strength, where strength[i] denotes the strength of the ith wizard. For a contiguous group of wizards (i.e. the wizards' strengths form a subarray of strength), the total strength is defined as the product of the following two values:

The strength of the weakest wizard in the group.
The total of all the individual strengths of the wizards in the group.
Return the sum of the total strengths of all contiguous groups of wizards. Since the answer may be very large, return it modulo 109 + 7.

A subarray is a contiguous non-empty sequence of elements within an array.

Example 1:

Input: strength = [1,3,1,2]
Output: 44
Explanation: The following are all the contiguous groups of wizards:
- [1] from [1,3,1,2] has a total strength of min([1]) * sum([1]) = 1 * 1 = 1
- [3] from [1,3,1,2] has a total strength of min([3]) * sum([3]) = 3 * 3 = 9
- [1] from [1,3,1,2] has a total strength of min([1]) * sum([1]) = 1 * 1 = 1
- [2] from [1,3,1,2] has a total strength of min([2]) * sum([2]) = 2 * 2 = 4
- [1,3] from [1,3,1,2] has a total strength of min([1,3]) * sum([1,3]) = 1 * 4 = 4
- [3,1] from [1,3,1,2] has a total strength of min([3,1]) * sum([3,1]) = 1 * 4 = 4
- [1,2] from [1,3,1,2] has a total strength of min([1,2]) * sum([1,2]) = 1 * 3 = 3
- [1,3,1] from [1,3,1,2] has a total strength of min([1,3,1]) * sum([1,3,1]) = 1 * 5 = 5
- [3,1,2] from [1,3,1,2] has a total strength of min([3,1,2]) * sum([3,1,2]) = 1 * 6 = 6
- [1,3,1,2] from [1,3,1,2] has a total strength of min([1,3,1,2]) * sum([1,3,1,2]) = 1 * 7 = 7
The sum of all the total strengths is 1 + 9 + 1 + 4 + 4 + 4 + 3 + 5 + 6 + 7 = 44.
Example 2:

Input: strength = [5,4,6]
Output: 213
Explanation: The following are all the contiguous groups of wizards:
- [5] from [5,4,6] has a total strength of min([5]) * sum([5]) = 5 * 5 = 25
- [4] from [5,4,6] has a total strength of min([4]) * sum([4]) = 4 * 4 = 16
- [6] from [5,4,6] has a total strength of min([6]) * sum([6]) = 6 * 6 = 36
- [5,4] from [5,4,6] has a total strength of min([5,4]) * sum([5,4]) = 4 * 9 = 36
- [4,6] from [5,4,6] has a total strength of min([4,6]) * sum([4,6]) = 4 * 10 = 40
- [5,4,6] from [5,4,6] has a total strength of min([5,4,6]) * sum([5,4,6]) = 4 * 15 = 60

The sum of all the total strengths is 25 + 16 + 36 + 36 + 40 + 60 = 213.

Constraints:

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

Question 2

3412. Find Mirror Score of a String

Problem Statement: You are given a string s.

We define the mirror of a letter in the English alphabet as its corresponding letter when the alphabet is reversed. For example, the mirror of 'a' is 'z', and the mirror of 'y' is 'b'.

Initially, all characters in the string s are unmarked.

You start with a score of 0, and you perform the following process on the string s:

Iterate through the string from left to right.
At each index i, find the closest unmarked index j such that j < i and s[j] is the mirror of s[i].
Then, mark both indices i and j, and add the value i - j to the total score.
If no such index j exists for the index i, move on to the next index without making any changes.
Return the total score at the end of the process.

Example 1:

Input: s = "aczzx"

Output: 5

Explanation:

i = 0. There is no index j that satisfies the conditions, so we skip.
i = 1. There is no index j that satisfies the conditions, so we skip.
i = 2. The closest index j that satisfies the conditions is j = 0, so we mark both indices 0 and 2, and then add 2 - 0 = 2 to the score.
i = 3. There is no index j that satisfies the conditions, so we skip.
i = 4. The closest index j that satisfies the conditions is j = 1, so we mark both indices 1 and 4, and then add 4 - 1 = 3 to the score.
Example 2:

Input: s = "abcdef"

Output: 0

Explanation:

For each index i, there is no index j that satisfies the conditions.

Constraints:

1 <= s.length <= 10^5
s consists only of lowercase English letters.

Question 3

2866. Beautiful Towers II

Problem Statement: You are given a 0-indexed array maxHeights of n integers.

You are tasked with building n towers in the coordinate line. The ith tower is built at coordinate i and has a height of heights[i].

A configuration of towers is beautiful if the following conditions hold:

1 <= heights[i] <= maxHeights[i]
heights is a mountain array.
Array heights is a mountain if there exists an index i such that:

For all 0 < j <= i, heights[j - 1] <= heights[j]
For all i <= k < n - 1, heights[k + 1] <= heights[k]
Return the maximum possible sum of heights of a beautiful configuration of towers.

Example 1:

Input: maxHeights = [5,3,4,1,1]
Output: 13
Explanation: One beautiful configuration with a maximum sum is heights = [5,3,3,1,1]. This configuration is beautiful since:
- 1 <= heights[i] <= maxHeights[i]
- heights is a mountain of peak i = 0.

It can be shown that there exists no other beautiful configuration with a sum of heights greater than 13.
Example 2:

Input: maxHeights = [6,5,3,9,2,7]
Output: 22
Explanation: One beautiful configuration with a maximum sum is heights = [3,3,3,9,2,2]. This configuration is beautiful since:
- 1 <= heights[i] <= maxHeights[i]
- heights is a mountain of peak i = 3.
It can be shown that there exists no other beautiful configuration with a sum of heights greater than 22.
Example 3:

Input: maxHeights = [3,2,5,5,2,3]
Output: 18
Explanation: One beautiful configuration with a maximum sum is heights = [2,2,5,5,2,2]. This configuration is beautiful since:
- 1 <= heights[i] <= maxHeights[i]
- heights is a mountain of peak i = 2.
Note that, for this configuration, i = 3 can also be considered a peak.
It can be shown that there exists no other beautiful configuration with a sum of heights greater than 18.

Constraints:

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

Question 4

1574. Shortest Subarray to be Removed to Make Array Sorted

Problem Statement : Given an integer array arr, remove a subarray (can be empty) from arr such that the remaining elements in arr are non-decreasing.

Return the length of the shortest subarray to remove.

A subarray is a contiguous subsequence of the array.

Example 1:

Input: arr = [1,2,3,10,4,2,3,5]
Output: 3
Explanation: The shortest subarray we can remove is [10,4,2] of length 3. The remaining elements after that will be [1,2,3,3,5] which are sorted.
Another correct solution is to remove the subarray [3,10,4].
Example 2:

Input: arr = [5,4,3,2,1]
Output: 4
Explanation: Since the array is strictly decreasing, we can only keep a single element. Therefore we need to remove a subarray of length 4, either [5,4,3,2] or [4,3,2,1].
Example 3:

Input: arr = [1,2,3]
Output: 0
Explanation: The array is already non-decreasing. We do not need to remove any elements.

Constraints:

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

Question 5

3113. Find the Number of Subarrays Where Boundary Elements Are Maximum

Problem Statement : You are given an array of positive integers nums.

Return the number of subarrays of nums, where the first and the last elements of the subarray are equal to the largest element in the subarray.

Example 1:

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

Output: 6

Explanation:

There are 6 subarrays which have the first and the last elements equal to the largest element of the subarray:

subarray [1,4,3,3,2], with its largest element 1. The first element is 1 and the last element is also 1.
subarray [1,4,3,3,2], with its largest element 4. The first element is 4 and the last element is also 4.
subarray [1,4,3,3,2], with its largest element 3. The first element is 3 and the last element is also 3.
subarray [1,4,3,3,2], with its largest element 3. The first element is 3 and the last element is also 3.
subarray [1,4,3,3,2], with its largest element 2. The first element is 2 and the last element is also 2.
subarray [1,4,3,3,2], with its largest element 3. The first element is 3 and the last element is also 3.
Hence, we return 6.

Example 2:

Input: nums = [3,3,3]

Output: 6

Explanation:

There are 6 subarrays which have the first and the last elements equal to the largest element of the subarray:

subarray [3,3,3], with its largest element 3. The first element is 3 and the last element is also 3.
subarray [3,3,3], with its largest element 3. The first element is 3 and the last element is also 3.
subarray [3,3,3], with its largest element 3. The first element is 3 and the last element is also 3.
subarray [3,3,3], with its largest element 3. The first element is 3 and the last element is also 3.
subarray [3,3,3], with its largest element 3. The first element is 3 and the last element is also 3.
subarray [3,3,3], with its largest element 3. The first element is 3 and the last element is also 3.
Hence, we return 6.

Example 3:

Input: nums = [1]

Output: 1

Explanation:

There is a single subarray of nums which is [1], with its largest element 1. The first element is 1 and the last element is also 1.

Hence, we return 1.

Constraints:

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

Question 6

2751. Robot Collisions

There are n 1-indexed robots, each having a position on a line, health, and movement direction.

You are given 0-indexed integer arrays positions, healths, and a string directions (directions[i] is either 'L' for left or 'R' for right). All integers in positions are unique.

All robots start moving on the line simultaneously at the same speed in their given directions. If two robots ever share the same position while moving, they will collide.

If two robots collide, the robot with lower health is removed from the line, and the health of the other robot decreases by one. The surviving robot continues in the same direction it was going. If both robots have the same health, they are both removed from the line.

Your task is to determine the health of the robots that survive the collisions, in the same order that the robots were given, i.e. final health of robot 1 (if survived), final health of robot 2 (if survived), and so on. If there are no survivors, return an empty array.

Return an array containing the health of the remaining robots (in the order they were given in the input), after no further collisions can occur.

Note: The positions may be unsorted.

Example 1:

Input: positions = [5,4,3,2,1], healths = [2,17,9,15,10], directions = "RRRRR"
Output: [2,17,9,15,10]
Explanation: No collision occurs in this example, since all robots are moving in the same direction. So, the health of the robots in order from the first robot is returned, [2, 17, 9, 15, 10].
Example 2:

Input: positions = [3,5,2,6], healths = [10,10,15,12], directions = "RLRL"
Output: [14]
Explanation: There are 2 collisions in this example. Firstly, robot 1 and robot 2 will collide, and since both have the same health, they will be removed from the line. Next, robot 3 and robot 4 will collide and since robot 4's health is smaller, it gets removed, and robot 3's health becomes 15 - 1 = 14. Only robot 3 remains, so we return [14].
Example 3:

Input: positions = [1,2,5,6], healths = [10,10,11,11], directions = "RLRL"
Output: []
Explanation: Robot 1 and robot 2 will collide and since both have the same health, they are both removed. Robot 3 and 4 will collide and since both have the same health, they are both removed. So, we return an empty array, [].

Constraints:

1 <= positions.length == healths.length == directions.length == n <= 105
1 <= positions[i], healths[i] <= 109
directions[i] == 'L' or directions[i] == 'R'