Problem 3420: Count Non-Decreasing Subarrays After K Operations

Problem Information

Difficulty: Hard
Acceptance Rate: 22.56%
Paid Only: No
Tags: Array, Stack, Segment Tree, Queue, Sliding Window, Monotonic Stack, Monotonic Queue

Problem Description

You are given an array `nums` of `n` integers and an integer `k`.

For each subarray of `nums`, you can apply **up to** `k` operations on it. In each operation, you increment any element of the subarray by 1.

Note that each subarray is considered independently, meaning changes made to one subarray do not persist to another.

Return the number of subarrays that you can make **non-decreasing** ■■■■■after performing at most `k` operations.

An array is said to be **non-decreasing** if each element is greater than or equal to its previous element, if it exists.

Example 1:

Input: nums = [6,3,1,2,4,4], k = 7

Output: 17

Explanation:

Out of all 21 possible subarrays of `nums`, only the subarrays `[6, 3, 1]`, `[6, 3, 1, 2]`, `[6, 3, 1, 2, 4]` and `[6, 3, 1, 2, 4, 4]` cannot be made non- decreasing after applying up to k = 7 operations. Thus, the number of non- decreasing subarrays is `21 - 4 = 17`.

Example 2:

Input: nums = [6,3,1,3,6], k = 4

Output: 12

Explanation:

The subarray `[3, 1, 3, 6]` along with all subarrays of `nums` with three or fewer elements, except `[6, 3, 1]`, can be made non-decreasing after `k` operations. There are 5 subarrays of a single element, 4 subarrays of two elements, and 2 subarrays of three elements except `[6, 3, 1]`, so there are `1 + 5 + 4 + 2 = 12` subarrays that can be made non-decreasing.

Constraints:

* `1 <= nums.length <= 105` * `1 <= nums[i] <= 109` * `1 <= k <= 109`

Code Snippets

C++:

```
class Solution {
public:
long long countNonDecreasingSubarrays(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public long countNonDecreasingSubarrays(int[] nums, int k) {

}
}
```

Python3:

```
class Solution:
def countNonDecreasingSubarrays(self, nums: List[int], k: int) -> int:
```