Problem 3420: Count Non-Decreasing Subarrays After K Operations

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

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 <= 10

5

1 <= nums[i] <= 10

9

1 <= k <= 10

9

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:
```

Python:

```
class Solution(object):
def countNonDecreasingSubarrays(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var countNonDecreasingSubarrays = function(nums, k) {

};
```

}

TypeScript:

```
function countNonDecreasingSubarrays(nums: number[], k: number): number {

};
```

C#:

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

}
}
```

C:

```
long long countNonDecreasingSubarrays(int* nums, int numsSize, int k) {

}
```

Go:

```
func countNonDecreasingSubarrays(nums []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun countNonDecreasingSubarrays(nums: IntArray, k: Int): Long {

}
}
```

Swift:

```
class Solution {
func countNonDecreasingSubarrays(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_non_decreasing_subarrays(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def count_non_decreasing_subarrays(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function countNonDecreasingSubarrays($nums, $k) {

}
}
```

Dart:

```
class Solution {
int countNonDecreasingSubarrays(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def countNonDecreasingSubarrays(nums: Array[Int], k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_non_decreasing_subarrays(nums :: [integer], k :: integer) ::
integer
def count_non_decreasing_subarrays(nums, k) do

end
end
```

Erlang:

```
-spec count_non_decreasing_subarrays(Nums :: [integer()], K :: integer()) ->
integer().
count_non_decreasing_subarrays(Nums, K) ->
.
```

Racket:

```
(define/contract (count-non-decreasing-subarrays nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Non-Decreasing Subarrays After K Operations
* Difficulty: Hard
* Tags: array, tree, stack, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
```

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

}
};
```

Java Solution:

```
/**
* Problem: Count Non-Decreasing Subarrays After K Operations
* Difficulty: Hard
* Tags: array, tree, stack, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Count Non-Decreasing Subarrays After K Operations
Difficulty: Hard
Tags: array, tree, stack, queue

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def countNonDecreasingSubarrays(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countNonDecreasingSubarrays(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Non-Decreasing Subarrays After K Operations
* Difficulty: Hard
* Tags: array, tree, stack, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var countNonDecreasingSubarrays = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Non-Decreasing Subarrays After K Operations
* Difficulty: Hard
* Tags: array, tree, stack, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
function countNonDecreasingSubarrays(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Non-Decreasing Subarrays After K Operations
* Difficulty: Hard
* Tags: array, tree, stack, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
}
```

C Solution:

```
/*
* Problem: Count Non-Decreasing Subarrays After K Operations
* Difficulty: Hard
* Tags: array, tree, stack, queue
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

long long countNonDecreasingSubarrays(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Count Non-Decreasing Subarrays After K Operations
// Difficulty: Hard
// Tags: array, tree, stack, queue
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func countNonDecreasingSubarrays(nums []int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countNonDecreasingSubarrays(nums: IntArray, k: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func countNonDecreasingSubarrays(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Non-Decreasing Subarrays After K Operations
// Difficulty: Hard
// Tags: array, tree, stack, queue
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn count_non_decreasing_subarrays(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def count_non_decreasing_subarrays(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function countNonDecreasingSubarrays($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int countNonDecreasingSubarrays(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def countNonDecreasingSubarrays(nums: Array[Int], k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_non_decreasing_subarrays(nums :: [integer], k :: integer) ::
integer
def count_non_decreasing_subarrays(nums, k) do

end
end
```

Erlang Solution:

```
-spec count_non_decreasing_subarrays(Nums :: [integer()], K :: integer()) ->
integer().
count_non_decreasing_subarrays(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (count-non-decreasing-subarrays nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```