Problem 1063: Number of Valid Subarrays

Problem Information

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

Problem Description

Given an integer array

nums

, return

the number of non-empty

subarrays

with the leftmost element of the subarray not larger than other elements in the subarray

.

A

subarray

is a

contiguous

part of an array.

Example 1:

Input:

nums = [1,4,2,5,3]

Output:

11

Explanation:

There are 11 valid subarrays: [1],[4],[2],[5],[3],[1,4],[2,5],[1,4,2],[2,5,3],[1,4,2,5],[1,4,2,5,3].

Example 2:

Input:

nums = [3,2,1]

Output:

3

Explanation:

The 3 valid subarrays are: [3],[2],[1].

Example 3:

Input:

nums = [2,2,2]

Output:

6

Explanation:

There are 6 valid subarrays: [2],[2],[2],[2,2],[2,2],[2,2,2].

Constraints:

1 <= nums.length <= 5 * 10

4

0 <= nums[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int validSubarrays(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int validSubarrays(int[] nums) {

}
}
```

Python3:

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

Python:

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

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function validSubarrays(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int ValidSubarrays(int[] nums) {

}
}
```

C:

```
int validSubarrays(int* nums, int numsSize) {

}
```

Go:

```
func validSubarrays(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun validSubarrays(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func validSubarrays(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn valid_subarrays(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def valid_subarrays(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function validSubarrays($nums) {

}
}
```

Dart:

```
class Solution {
int validSubarrays(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def validSubarrays(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec valid_subarrays(nums :: [integer]) :: integer
def valid_subarrays(nums) do

end
end
```

Erlang:

```
-spec valid_subarrays(Nums :: [integer()]) -> integer().
valid_subarrays(Nums) ->
.
```

Racket:

```
(define/contract (valid-subarrays nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Valid Subarrays
* Difficulty: Hard
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int validSubarrays(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Valid Subarrays
* Difficulty: Hard
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int validSubarrays(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Valid Subarrays
Difficulty: Hard
Tags: array, stack
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Valid Subarrays
* Difficulty: Hard
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Number of Valid Subarrays
* Difficulty: Hard
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function validSubarrays(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Valid Subarrays
* Difficulty: Hard
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int ValidSubarrays(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Number of Valid Subarrays
* Difficulty: Hard
* Tags: array, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

int validSubarrays(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Number of Valid Subarrays
// Difficulty: Hard
// Tags: array, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func validSubarrays(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun validSubarrays(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func validSubarrays(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Valid Subarrays
// Difficulty: Hard
// Tags: array, stack
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn valid_subarrays(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def valid_subarrays(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function validSubarrays($nums) {

}
}
```

Dart Solution:

```
class Solution {
int validSubarrays(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def validSubarrays(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec valid_subarrays(nums :: [integer]) :: integer
def valid_subarrays(nums) do

end
end
```

Erlang Solution:

```
-spec valid_subarrays(Nums :: [integer()]) -> integer().
valid_subarrays(Nums) ->
.
```

Racket Solution:

```
(define/contract (valid-subarrays nums)
(-> (listof exact-integer?) exact-integer?)
)
```