Problem 2393: Count Strictly Increasing Subarrays

Problem Information

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

Problem Description

You are given an array

nums

consisting of

positive

integers.

Return

the number of

subarrays

of

nums

that are in

strictly increasing

order.

A

subarray

is a

contiguous

part of an array.

Example 1:

Input:

nums = [1,3,5,4,4,6]

Output:

10

Explanation:

The strictly increasing subarrays are the following: - Subarrays of length 1: [1], [3], [5], [4], [4], [6]. - Subarrays of length 2: [1,3], [3,5], [4,6]. - Subarrays of length 3: [1,3,5]. The total number of subarrays is 6 + 3 + 1 = 10.

Example 2:

Input:

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

Output:

15

Explanation:

Every subarray is strictly increasing. There are 15 possible subarrays that we can take.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long countSubarrays(int[] nums) {

}
}
```

Python3:

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

Python:

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

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long CountSubarrays(int[] nums) {

}
}
```

C:

```
long long countSubarrays(int* nums, int numsSize) {

}
```

Go:

```
func countSubarrays(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun countSubarrays(nums: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Strictly Increasing Subarrays
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Count Strictly Increasing Subarrays
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public long countSubarrays(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Strictly Increasing Subarrays
Difficulty: Medium
Tags: array, dp, math
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Strictly Increasing Subarrays
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Strictly Increasing Subarrays
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Count Strictly Increasing Subarrays
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public long CountSubarrays(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Strictly Increasing Subarrays
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

long long countSubarrays(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Strictly Increasing Subarrays
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func countSubarrays(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countSubarrays(nums: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Strictly Increasing Subarrays
// Difficulty: Medium
// Tags: array, dp, math
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn count_subarrays(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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