Problem 2970: Count the Number of Incremovable Subarrays I

Problem Information

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

Problem Description

You are given a

0-indexed

array of

positive

integers

nums

.

A subarray of

nums

is called

incremovable

if

nums

becomes

strictly increasing

on removing the subarray. For example, the subarray

[3, 4]

is an incremovable subarray of

[5, 3, 4, 6, 7]

because removing this subarray changes the array

[5, 3, 4, 6, 7]

to

[5, 6, 7]

which is strictly increasing.

Return

the total number of

incremovable

subarrays of

nums

.

Note

that an empty array is considered strictly increasing.

A

subarray

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

Example 1:

Input:

nums = [1,2,3,4]

Output:

10

Explanation:

The 10 incremovable subarrays are: [1], [2], [3], [4], [1,2], [2,3], [3,4], [1,2,3], [2,3,4], and [1,2,3,4], because on removing any one of these subarrays nums becomes strictly increasing. Note that you cannot select an empty subarray.

Example 2:

Input:

nums = [6,5,7,8]

Output:

7

Explanation:

The 7 incremovable subarrays are: [5], [6], [5,7], [6,5], [5,7,8], [6,5,7] and [6,5,7,8]. It can be shown that there are only 7 incremovable subarrays in nums.

Example 3:

Input:

nums = [8,7,6,6]

Output:

3

Explanation:

The 3 incremovable subarrays are: [8,7,6], [7,6,6], and [8,7,6,6]. Note that [8,7] is not an incremovable subarray because after removing [8,7] nums becomes [6,6], which is sorted in ascending order but not strictly increasing.

Constraints:

1 <= nums.length <= 50

1 <= nums[i] <= 50

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count the Number of Incremovable Subarrays I
* Difficulty: Easy
* Tags: array, sort, search
*
* 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 incremovableSubarrayCount(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count the Number of Incremovable Subarrays I
* Difficulty: Easy
* Tags: array, sort, search
*
* 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 incremovableSubarrayCount(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count the Number of Incremovable Subarrays I
Difficulty: Easy
Tags: array, sort, search

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 incremovableSubarrayCount(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count the Number of Incremovable Subarrays I
* Difficulty: Easy
* Tags: array, sort, search
*
* 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 incremovableSubarrayCount = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count the Number of Incremovable Subarrays I
* Difficulty: Easy
* Tags: array, sort, search
*
* 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 incremovableSubarrayCount(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count the Number of Incremovable Subarrays I
* Difficulty: Easy
* Tags: array, sort, search
*
* 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 IncremovableSubarrayCount(int[] nums) {

}
}
```

C Solution:

```c
/*
* Problem: Count the Number of Incremovable Subarrays I
* Difficulty: Easy
* Tags: array, sort, search
*
* 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 incremovableSubarrayCount(int* nums, int numsSize) {

}
```

Go Solution:

```go
// Problem: Count the Number of Incremovable Subarrays I
// Difficulty: Easy
// Tags: array, sort, search
//
// 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 incremovableSubarrayCount(nums []int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun incremovableSubarrayCount(nums: IntArray): Int {

}
}
```

Swift Solution:

```swift
class Solution {
func incremovableSubarrayCount(_ nums: [Int]) -> Int {
}
}
```

Rust Solution:

```
// Problem: Count the Number of Incremovable Subarrays I
// Difficulty: Easy
// Tags: array, sort, search
//
// 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 incremovable_subarray_count(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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