Problem 3113: Find the Number of Subarrays Where Boundary Elements Are Maximum

Problem Information

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

Problem Description

You are given an array of

positive

integers

nums

.

Return the number of

subarrays

of

nums

, where the

first

and the

last

elements of the subarray are

equal

to the

largest

element in the subarray.

Example 1:

Input:

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

Output:

6

Explanation:

There are 6 subarrays which have the first and the last elements equal to the largest element of the subarray:

subarray

[

1

,4,3,3,2]

, with its largest element 1. The first element is 1 and the last element is also 1.

subarray

[1,

4

,3,3,2]

, with its largest element 4. The first element is 4 and the last element is also 4.

subarray

[1,4,

3

,3,2]

, with its largest element 3. The first element is 3 and the last element is also 3.

subarray

[1,4,3,

3

,2]

, with its largest element 3. The first element is 3 and the last element is also 3.

subarray

[1,4,3,3,

2

]

, with its largest element 2. The first element is 2 and the last element is also 2.

subarray

[1,4,

3,3

,2]

, with its largest element 3. The first element is 3 and the last element is also 3.

Hence, we return 6.

Example 2:

Input:

nums = [3,3,3]

Output:

6

Explanation:

There are 6 subarrays which have the first and the last elements equal to the largest element of the subarray:

subarray

[

3

,3,3]

, with its largest element 3. The first element is 3 and the last element is also 3.

subarray

[3,

3

,3]

, with its largest element 3. The first element is 3 and the last element is also 3.

subarray

[3,3,

3

]

, with its largest element 3. The first element is 3 and the last element is also 3.

subarray

[

3,3

,3]

, with its largest element 3. The first element is 3 and the last element is also 3.

subarray

[3,

3,3

]

, with its largest element 3. The first element is 3 and the last element is also 3.

subarray

[

3,3,3

]

, with its largest element 3. The first element is 3 and the last element is also 3.

Hence, we return 6.

Example 3:

Input:

nums = [1]

Output:

1

Explanation:

There is a single subarray of

nums

which is

[

1

]

, with its largest element 1. The first element is 1 and the last element is also 1.

Hence, we return 1.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find the Number of Subarrays Where Boundary Elements Are Maximum
* Difficulty: Hard
* Tags: array, search, 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:
long long numberOfSubarrays(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Number of Subarrays Where Boundary Elements Are Maximum
* Difficulty: Hard
* Tags: array, search, 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 long numberOfSubarrays(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Number of Subarrays Where Boundary Elements Are Maximum
Difficulty: Hard
Tags: array, search, 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 numberOfSubarrays(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Number of Subarrays Where Boundary Elements Are Maximum
* Difficulty: Hard
* Tags: array, search, 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 numberOfSubarrays = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Number of Subarrays Where Boundary Elements Are Maximum
* Difficulty: Hard
* Tags: array, search, 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 numberOfSubarrays(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Find the Number of Subarrays Where Boundary Elements Are Maximum
* Difficulty: Hard
* Tags: array, search, 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 long NumberOfSubarrays(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find the Number of Subarrays Where Boundary Elements Are Maximum
* Difficulty: Hard
* Tags: array, search, 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
*/

long long numberOfSubarrays(int* nums, int numsSize) {
}
```

Go Solution:

```
// Problem: Find the Number of Subarrays Where Boundary Elements Are Maximum
// Difficulty: Hard
// Tags: array, search, 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 numberOfSubarrays(nums []int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Number of Subarrays Where Boundary Elements Are Maximum
// Difficulty: Hard
// Tags: array, search, 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 number_of_subarrays(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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