Problem 795: Number of Subarrays with Bounded Maximum

Problem Information

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

Problem Description

Given an integer array

nums

and two integers

left

and

right

, return

the number of contiguous non-empty

subarrays

such that the value of the maximum array element in that subarray is in the range

[left, right]

.

The test cases are generated so that the answer will fit in a

32-bit

integer.

Example 1:

Input:

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

Output:

3

Explanation:

There are three subarrays that meet the requirements: [2], [2, 1], [3].

Example 2:

Input:

nums = [2,9,2,5,6], left = 2, right = 8

Output:

7

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

0 <= left <= right <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int numSubarrayBoundedMax(vector<int>& nums, int left, int right) {

}
};
```

Java:

```
class Solution {
public int numSubarrayBoundedMax(int[] nums, int left, int right) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} left
* @param {number} right
* @return {number}
*/
var numSubarrayBoundedMax = function(nums, left, right) {

};
```

TypeScript:

```
function numSubarrayBoundedMax(nums: number[], left: number, right: number):
number {

};
```

C#:

```
public class Solution {
public int NumSubarrayBoundedMax(int[] nums, int left, int right) {

}
}
```

C:

```
int numSubarrayBoundedMax(int* nums, int numsSize, int left, int right) {

}
```

Go:

```
func numSubarrayBoundedMax(nums []int, left int, right int) int {

}
```

Kotlin:

```
class Solution {
fun numSubarrayBoundedMax(nums: IntArray, left: Int, right: Int): Int {

}
}
```

Swift:

```
class Solution {
func numSubarrayBoundedMax(_ nums: [Int], _ left: Int, _ right: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_subarray_bounded_max(nums: Vec<i32>, left: i32, right: i32) -> i32
{

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} left
# @param {Integer} right
# @return {Integer}
def num_subarray_bounded_max(nums, left, right)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $left
* @param Integer $right
* @return Integer
*/
function numSubarrayBoundedMax($nums, $left, $right) {

}
}
```

Dart:

```
class Solution {
int numSubarrayBoundedMax(List<int> nums, int left, int right) {

}
}
```

Scala:

```
object Solution {
def numSubarrayBoundedMax(nums: Array[Int], left: Int, right: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_subarray_bounded_max(nums :: [integer], left :: integer, right ::
integer) :: integer
def num_subarray_bounded_max(nums, left, right) do

end
end
```

Erlang:

```
-spec num_subarray_bounded_max(Nums :: [integer()], Left :: integer(), Right
:: integer()) -> integer().
num_subarray_bounded_max(Nums, Left, Right) ->
.
```

Racket:

```
(define/contract (num-subarray-bounded-max nums left right)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Subarrays with Bounded Maximum
* Difficulty: Medium
* Tags: array
*
* 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 numSubarrayBoundedMax(vector<int>& nums, int left, int right) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Subarrays with Bounded Maximum
* Difficulty: Medium
* Tags: array
*
* 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 numSubarrayBoundedMax(int[] nums, int left, int right) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Subarrays with Bounded Maximum
```

```
Difficulty: Medium
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Subarrays with Bounded Maximum
* Difficulty: Medium
* Tags: array
*
* 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
* @param {number} left
* @param {number} right
* @return {number}
*/
var numSubarrayBoundedMax = function(nums, left, right) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Subarrays with Bounded Maximum
* Difficulty: Medium
* Tags: array
*
* 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 numSubarrayBoundedMax(nums: number[], left: number, right: number):
number {

};
```

C# Solution:

```
/*
* Problem: Number of Subarrays with Bounded Maximum
* Difficulty: Medium
* Tags: array
*
* 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 NumSubarrayBoundedMax(int[] nums, int left, int right) {

}
}
```

C Solution:

```
/*
* Problem: Number of Subarrays with Bounded Maximum
* Difficulty: Medium
* Tags: array
*
* 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 numSubarrayBoundedMax(int* nums, int numsSize, int left, int right) {

}
```

Go Solution:

```
// Problem: Number of Subarrays with Bounded Maximum
// Difficulty: Medium
// Tags: array
//
// 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 numSubarrayBoundedMax(nums []int, left int, right int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numSubarrayBoundedMax(nums: IntArray, left: Int, right: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numSubarrayBoundedMax(_ nums: [Int], _ left: Int, _ right: Int) -> Int {
}
}
```

Rust Solution:

```
// Problem: Number of Subarrays with Bounded Maximum
// Difficulty: Medium
// Tags: array
//
// 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 num_subarray_bounded_max(nums: Vec<i32>, left: i32, right: i32) -> i32
{

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} left
# @param {Integer} right
# @return {Integer}
def num_subarray_bounded_max(nums, left, right)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $left
* @param Integer $right
* @return Integer
*/
function numSubarrayBoundedMax($nums, $left, $right) {

}
}
```

Dart Solution:

```
class Solution {
int numSubarrayBoundedMax(List<int> nums, int left, int right) {

}
}
```

Scala Solution:

```
object Solution {
def numSubarrayBoundedMax(nums: Array[Int], left: Int, right: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_subarray_bounded_max(nums :: [integer], left :: integer, right ::
integer) :: integer
def num_subarray_bounded_max(nums, left, right) do

end
end
```

Erlang Solution:

```
-spec num_subarray_bounded_max(Nums :: [integer()], Left :: integer(), Right
:: integer()) -> integer().
num_subarray_bounded_max(Nums, Left, Right) ->
.
```

Racket Solution:

```
(define/contract (num-subarray-bounded-max nums left right)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```