Problem 2444: Count Subarrays With Fixed Bounds

Problem Information

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

Problem Description

You are given an integer array

nums

and two integers

minK

and

maxK

.

A

fixed-bound subarray

of

nums

is a subarray that satisfies the following conditions:

The

minimum

value in the subarray is equal to

minK

.

The

maximum

value in the subarray is equal to

maxK

.

Return

the

number

of fixed-bound subarrays

.

A

subarray

is a

contiguous

part of an array.

Example 1:

Input:

nums = [1,3,5,2,7,5], minK = 1, maxK = 5

Output:

2

Explanation:

The fixed-bound subarrays are [1,3,5] and [1,3,5,2].

Example 2:

Input:

nums = [1,1,1,1], minK = 1, maxK = 1

Output:

10

Explanation:

Every subarray of nums is a fixed-bound subarray. There are 10 possible subarrays.

Constraints:

2 <= nums.length <= 10

5

1 <= nums[i], minK, maxK <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} minK
* @param {number} maxK
* @return {number}
*/
```

```
var countSubarrays = function(nums, minK, maxK) {

};
```

TypeScript:

```
function countSubarrays(nums: number[], minK: number, maxK: number): number {

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

```
func countSubarrays(nums []int, minK int, maxK int) int64 {

}
```

Kotlin:

```
class Solution {
fun countSubarrays(nums: IntArray, minK: Int, maxK: Int): Long {

}
}
```

Swift:

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

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int countSubarrays(List<int> nums, int minK, int maxK) {
}
}
```

Scala:

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

}
}
```

Elixir:

```elixir
defmodule Solution do
@spec count_subarrays(nums :: [integer], min_k :: integer, max_k :: integer)
:: integer
def count_subarrays(nums, min_k, max_k) do

end
end
```

Erlang:

```erlang
-spec count_subarrays(Nums :: [integer()], MinK :: integer(), MaxK ::
integer()) -> integer().
count_subarrays(Nums, MinK, MaxK) ->
.
```

Racket:

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

Solutions

C++ Solution:

```cpp
/*
* Problem: Count Subarrays With Fixed Bounds
* Difficulty: Hard
* Tags: array, queue
*
* 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 countSubarrays(vector<int>& nums, int minK, int maxK) {

}
};
```

Java Solution:

```
/**
* Problem: Count Subarrays With Fixed Bounds
* Difficulty: Hard
* Tags: array, queue
*
* 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 countSubarrays(int[] nums, int minK, int maxK) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Subarrays With Fixed Bounds
Difficulty: Hard
Tags: array, queue

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Subarrays With Fixed Bounds
* Difficulty: Hard
* Tags: array, queue
*
* 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} minK
* @param {number} maxK
* @return {number}
*/
var countSubarrays = function(nums, minK, maxK) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Subarrays With Fixed Bounds
* Difficulty: Hard
* Tags: array, queue
*
* 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 countSubarrays(nums: number[], minK: number, maxK: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Subarrays With Fixed Bounds
* Difficulty: Hard
* Tags: array, queue
*
* 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 CountSubarrays(int[] nums, int minK, int maxK) {

}
}
```

C Solution:

```
/*
* Problem: Count Subarrays With Fixed Bounds
* Difficulty: Hard
* Tags: array, queue
*
* 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 countSubarrays(int* nums, int numsSize, int minK, int maxK) {

}
```

Go Solution:

```
// Problem: Count Subarrays With Fixed Bounds
// Difficulty: Hard
// Tags: array, queue
//
// 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 countSubarrays(nums []int, minK int, maxK int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countSubarrays(nums: IntArray, minK: Int, maxK: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Subarrays With Fixed Bounds
// Difficulty: Hard
// Tags: array, queue
//
// 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 count_subarrays(nums: Vec<i32>, min_k: i32, max_k: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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