Problem 3739: Count Subarrays With Majority Element II

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

target

.

Return the number of

subarrays

of

nums

in which

target

is the

majority element

.

The

majority element

of a subarray is the element that appears

strictly more than half

of the times in that subarray.

Example 1:

Input:

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

Output:

5

Explanation:

Valid subarrays with

target = 2

as the majority element:

nums[1..1] = [2]

nums[2..2] = [2]

nums[1..2] = [2,2]

nums[0..2] = [1,2,2]

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

So there are 5 such subarrays.

Example 2:

Input:

nums = [1,1,1,1], target = 1

Output:

10

Explanation:

All 10 subarrays have 1 as the majority element.

Example 3:

Input:

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

Output:

0

Explanation:

target = 4

does not appear in

nums

at all. Therefore, there cannot be any subarray where 4 is the majority element. Hence the answer is 0.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

1 <= target <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long countMajoritySubarrays(vector<int>& nums, int target) {

}
};
```

Java:

```
class Solution {
public long countMajoritySubarrays(int[] nums, int target) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countMajoritySubarrays(nums: number[], target: number): number {

};
```

C#:

```
public class Solution {
public long CountMajoritySubarrays(int[] nums, int target) {

}
}
```

C:

```
long long countMajoritySubarrays(int* nums, int numsSize, int target) {

}
```

Go:

```
func countMajoritySubarrays(nums []int, target int) int64 {
}
```

Kotlin:

```
class Solution {
fun countMajoritySubarrays(nums: IntArray, target: Int): Long {

}
}
```

Swift:

```
class Solution {
func countMajoritySubarrays(_ nums: [Int], _ target: Int) -> Int {

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def count_majority_subarrays(nums, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function countMajoritySubarrays($nums, $target) {

}
}
```

Dart:

```
class Solution {
int countMajoritySubarrays(List<int> nums, int target) {

}
}
```

Scala:

```
object Solution {
def countMajoritySubarrays(nums: Array[Int], target: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_majority_subarrays(nums :: [integer], target :: integer) ::
integer
def count_majority_subarrays(nums, target) do

end
end
```

Erlang:

```
-spec count_majority_subarrays(Nums :: [integer()], Target :: integer()) ->
integer().
count_majority_subarrays(Nums, Target) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Subarrays With Majority Element II
* Difficulty: Hard
* Tags: array, tree, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
long long countMajoritySubarrays(vector<int>& nums, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Count Subarrays With Majority Element II
* Difficulty: Hard
* Tags: array, tree, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public long countMajoritySubarrays(int[] nums, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Subarrays With Majority Element II
Difficulty: Hard
Tags: array, tree, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Subarrays With Majority Element II
* Difficulty: Hard
* Tags: array, tree, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Subarrays With Majority Element II
* Difficulty: Hard
* Tags: array, tree, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function countMajoritySubarrays(nums: number[], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Subarrays With Majority Element II
* Difficulty: Hard
* Tags: array, tree, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public long CountMajoritySubarrays(int[] nums, int target) {

}

}
```

C Solution:

```
/*
* Problem: Count Subarrays With Majority Element II
* Difficulty: Hard
* Tags: array, tree, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

long long countMajoritySubarrays(int* nums, int numsSize, int target) {

}
```

Go Solution:

```
// Problem: Count Subarrays With Majority Element II
// Difficulty: Hard
// Tags: array, tree, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func countMajoritySubarrays(nums []int, target int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countMajoritySubarrays(nums: IntArray, target: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func countMajoritySubarrays(_ nums: [Int], _ target: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Subarrays With Majority Element II
// Difficulty: Hard
// Tags: array, tree, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn count_majority_subarrays(nums: Vec<i32>, target: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def count_majority_subarrays(nums, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function countMajoritySubarrays($nums, $target) {
}
}
```

Dart Solution:

```
class Solution {
int countMajoritySubarrays(List<int> nums, int target) {

}
}
```

Scala Solution:

```
object Solution {
def countMajoritySubarrays(nums: Array[Int], target: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_majority_subarrays(nums :: [integer], target :: integer) ::
integer
def count_majority_subarrays(nums, target) do

end
end
```

Erlang Solution:

```
-spec count_majority_subarrays(Nums :: [integer()], Target :: integer()) ->
integer().
count_majority_subarrays(Nums, Target) ->
.
```

Racket Solution:

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