Problem 2962: Count Subarrays Where Max Element Appears at Least K Times

Problem Information

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

Problem Description

You are given an integer array

nums

and a

positive

integer

k

.

Return

the number of subarrays where the

maximum

element of

nums

appears

at least

k

times in that subarray.

A

subarray

is a contiguous sequence of elements within an array.

Example 1:

Input:

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

Output:

6

Explanation:

The subarrays that contain the element 3 at least 2 times are: [1,3,2,3], [1,3,2,3,3], [3,2,3], [3,2,3,3], [2,3,3] and [3,3].

Example 2:

Input:

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

Output:

0

Explanation:

No subarray contains the element 4 at least 3 times.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

6

1 <= k <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
long long countSubarrays(int* nums, int numsSize, int k) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Subarrays Where Max Element Appears at Least K Times
* 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:
long long countSubarrays(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Count Subarrays Where Max Element Appears at Least K Times
* 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 long countSubarrays(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Subarrays Where Max Element Appears at Least K Times
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 countSubarrays(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Subarrays Where Max Element Appears at Least K Times
* 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} k
* @return {number}
*/
var countSubarrays = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Subarrays Where Max Element Appears at Least K Times
* 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 countSubarrays(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Subarrays Where Max Element Appears at Least K Times
* 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 long CountSubarrays(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Count Subarrays Where Max Element Appears at Least K Times
* 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
*/

long long countSubarrays(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Count Subarrays Where Max Element Appears at Least K Times
// 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 countSubarrays(nums []int, k int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Subarrays Where Max Element Appears at Least K Times
// 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 count_subarrays(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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