Problem 2488: Count Subarrays With Median K

Problem Information

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

Problem Description

You are given an array

nums

of size

n

consisting of

distinct

integers from

1

to

n

and a positive integer

k

.

Return

the number of non-empty subarrays in

nums

that have a

median

equal to

k

.

Note

:

The median of an array is the

middle

element after sorting the array in

ascending

order. If the array is of even length, the median is the

left

middle element.

For example, the median of

[2,3,1,4]

is

2

, and the median of

[8,4,3,5,1]

is

4

.

A subarray is a contiguous part of an array.

Example 1:

Input:

nums = [3,2,1,4,5], k = 4

Output:

3

Explanation:

The subarrays that have a median equal to 4 are: [4], [4,5] and [1,4,5].

Example 2:

Input:

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

Output:

1

Explanation:

[3] is the only subarray that has a median equal to 3.

Constraints:

n == nums.length

1 <= n <= 10

5

1 <= nums[i], k <= n

The integers in

nums

are distinct.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

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): Int = {

}
}
```

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 With Median K
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Count Subarrays With Median K
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

}
}
```

Python3 Solution:

```
"""
Problem: Count Subarrays With Median K
Difficulty: Hard
Tags: array, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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 With Median K
* Difficulty: Hard
* Tags: array, hash, sort
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Subarrays With Median K
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

C# Solution:

```
/*
* Problem: Count Subarrays With Median K
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

}
}
```

C Solution:

```
/*
* Problem: Count Subarrays With Median K
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
```

Go Solution:

```
// Problem: Count Subarrays With Median K
// Difficulty: Hard
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
```

Kotlin Solution:

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

```
}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Subarrays With Median K
// Difficulty: Hard
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

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): Int = {

}
}
```

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?)
)
```