Problem 1343: Number of Sub-arrays of Size K and Average Greater than or Equal to Threshold

Problem Information

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

Problem Description

Given an array of integers

arr

and two integers

k

and

threshold

, return

the number of sub-arrays of size

k

and average greater than or equal to

threshold

.

Example 1:

Input:

arr = [2,2,2,2,5,5,5,8], k = 3, threshold = 4

Output:

3

Explanation:

Sub-arrays [2,5,5],[5,5,5] and [5,5,8] have averages 4, 5 and 6 respectively. All other sub-arrays of size 3 have averages less than 4 (the threshold).

Example 2:

Input:

arr = [11,13,17,23,29,31,7,5,2,3], k = 3, threshold = 5

Output:

6

Explanation:

The first 6 sub-arrays of size 3 have averages greater than 5. Note that averages are not integers.

Constraints:

1 <= arr.length <= 10

5

1 <= arr[i] <= 10

4

1 <= k <= arr.length

0 <= threshold <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int numOfSubarrays(vector<int>& arr, int k, int threshold) {

}
};
```

Java:

```
class Solution {
public int numOfSubarrays(int[] arr, int k, int threshold) {

}
}
```

Python3:

```
class Solution:
def numOfSubarrays(self, arr: List[int], k: int, threshold: int) -> int:
```

Python:

```
class Solution(object):
def numOfSubarrays(self, arr, k, threshold):
"""
:type arr: List[int]
:type k: int
:type threshold: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} arr
* @param {number} k
* @param {number} threshold
* @return {number}
*/
var numOfSubarrays = function(arr, k, threshold) {

};
```

TypeScript:

```
function numOfSubarrays(arr: number[], k: number, threshold: number): number
{

};
```

C#:

```
public class Solution {
public int NumOfSubarrays(int[] arr, int k, int threshold) {

}
}
```

C:

```
int numOfSubarrays(int* arr, int arrSize, int k, int threshold) {

}
```

Go:

```
func numOfSubarrays(arr []int, k int, threshold int) int {

}
```

Kotlin:

```
class Solution {
fun numOfSubarrays(arr: IntArray, k: Int, threshold: Int): Int {

}
```

```
}
```

Swift:

```
class Solution {
func numOfSubarrays(_ arr: [Int], _ k: Int, _ threshold: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_of_subarrays(arr: Vec<i32>, k: i32, threshold: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer} k
# @param {Integer} threshold
# @return {Integer}
def num_of_subarrays(arr, k, threshold)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $k
* @param Integer $threshold
* @return Integer
*/
function numOfSubarrays($arr, $k, $threshold) {

}
}
```

Dart:

```
class Solution {
int numOfSubarrays(List<int> arr, int k, int threshold) {

}
}
```

Scala:

```
object Solution {
def numOfSubarrays(arr: Array[Int], k: Int, threshold: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_of_subarrays(arr :: [integer], k :: integer, threshold :: integer)
:: integer
def num_of_subarrays(arr, k, threshold) do

end
end
```

Erlang:

```
-spec num_of_subarrays(Arr :: [integer()], K :: integer(), Threshold ::
integer()) -> integer().
num_of_subarrays(Arr, K, Threshold) ->
.
```

Racket:

```
(define/contract (num-of-subarrays arr k threshold)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Sub-arrays of Size K and Average Greater than or Equal
to Threshold
* 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 numOfSubarrays(vector<int>& arr, int k, int threshold) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Sub-arrays of Size K and Average Greater than or Equal
to Threshold
* 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 numOfSubarrays(int[] arr, int k, int threshold) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Sub-arrays of Size K and Average Greater than or Equal to
```

```
Threshold
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 numOfSubarrays(self, arr: List[int], k: int, threshold: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numOfSubarrays(self, arr, k, threshold):
"""
:type arr: List[int]
:type k: int
:type threshold: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Sub-arrays of Size K and Average Greater than or Equal
to Threshold
* 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[]} arr
* @param {number} k
* @param {number} threshold
```

```
* @return {number}
*/
var numOfSubarrays = function(arr, k, threshold) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Sub-arrays of Size K and Average Greater than or Equal
to Threshold
* 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 numOfSubarrays(arr: number[], k: number, threshold: number): number
{

};
```

C# Solution:

```
/*
* Problem: Number of Sub-arrays of Size K and Average Greater than or Equal
to Threshold
* 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 NumOfSubarrays(int[] arr, int k, int threshold) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Number of Sub-arrays of Size K and Average Greater than or Equal
to Threshold
* 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 numOfSubarrays(int* arr, int arrSize, int k, int threshold) {

}
```

Go Solution:

```
// Problem: Number of Sub-arrays of Size K and Average Greater than or Equal
to Threshold
// 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 numOfSubarrays(arr []int, k int, threshold int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numOfSubarrays(arr: IntArray, k: Int, threshold: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numOfSubarrays(_ arr: [Int], _ k: Int, _ threshold: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Sub-arrays of Size K and Average Greater than or Equal
to Threshold
// 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_of_subarrays(arr: Vec<i32>, k: i32, threshold: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} k
# @param {Integer} threshold
# @return {Integer}
def num_of_subarrays(arr, k, threshold)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $k
```

```
* @param Integer $threshold
* @return Integer
*/
function numOfSubarrays($arr, $k, $threshold) {

}
}
```

Dart Solution:

```
class Solution {
int numOfSubarrays(List<int> arr, int k, int threshold) {

}
}
```

Scala Solution:

```
object Solution {
def numOfSubarrays(arr: Array[Int], k: Int, threshold: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_of_subarrays(arr :: [integer], k :: integer, threshold :: integer)
:: integer
def num_of_subarrays(arr, k, threshold) do

end
end
```

Erlang Solution:

```
-spec num_of_subarrays(Arr :: [integer()], K :: integer(), Threshold ::
integer()) -> integer().
num_of_subarrays(Arr, K, Threshold) ->
.
```

Racket Solution:

```
(define/contract (num-of-subarrays arr k threshold)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```