Problem 2334: Subarray With Elements Greater Than Varying Threshold

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

threshold

.

Find any subarray of

nums

of length

k

such that

every

element in the subarray is

greater

than

threshold / k

.

Return

the

size

of

any

such subarray

. If there is no such subarray, return

-1

.

A

subarray

is a contiguous non-empty sequence of elements within an array.

Example 1:

Input:

nums = [1,3,4,3,1], threshold = 6

Output:

3

Explanation:

The subarray [3,4,3] has a size of 3, and every element is greater than 6 / 3 = 2. Note that this is the only valid subarray.

Example 2:

Input:

nums = [6,5,6,5,8], threshold = 7

Output:

1

Explanation:

The subarray [8] has a size of 1, and 8 > 7 / 1 = 7. So 1 is returned. Note that the subarray [6,5] has a size of 2, and every element is greater than 7 / 2 = 3.5. Similarly, the subarrays [6,5,6], [6,5,6,5], [6,5,6,5,8] also satisfy the given conditions. Therefore, 2, 3, 4, or 5 may also be returned.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i], threshold <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int validSubarraySize(vector<int>& nums, int threshold) {

}
};
```

Java:

```
class Solution {
public int validSubarraySize(int[] nums, int threshold) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function validSubarraySize(nums: number[], threshold: number): number {

};
```

C#:

```
public class Solution {
public int ValidSubarraySize(int[] nums, int threshold) {

}
}
```

C:

```
int validSubarraySize(int* nums, int numsSize, int threshold) {

}
```

Go:

```
func validSubarraySize(nums []int, threshold int) int {

}
```

Kotlin:

```
class Solution {
fun validSubarraySize(nums: IntArray, threshold: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} threshold
# @return {Integer}
def valid_subarray_size(nums, threshold)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $threshold
* @return Integer
*/
function validSubarraySize($nums, $threshold) {

}
}
```

Dart:

```
class Solution {
int validSubarraySize(List<int> nums, int threshold) {

}
}
```

Scala:

```
object Solution {
def validSubarraySize(nums: Array[Int], threshold: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec valid_subarray_size(nums :: [integer], threshold :: integer) :: integer
def valid_subarray_size(nums, threshold) do

end
end
```

Erlang:

```
-spec valid_subarray_size(Nums :: [integer()], Threshold :: integer()) ->
integer().
valid_subarray_size(Nums, Threshold) ->
.
```

Racket:

```
(define/contract (valid-subarray-size nums threshold)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Subarray With Elements Greater Than Varying Threshold
* Difficulty: Hard
* Tags: array, graph, stack
*
* 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 validSubarraySize(vector<int>& nums, int threshold) {

}
};
```

Java Solution:

```
/**
* Problem: Subarray With Elements Greater Than Varying Threshold
* Difficulty: Hard
* Tags: array, graph, stack
*
* 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 validSubarraySize(int[] nums, int threshold) {

}
}
```

Python3 Solution:

```
"""
Problem: Subarray With Elements Greater Than Varying Threshold
Difficulty: Hard
Tags: array, graph, stack

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 validSubarraySize(self, nums: List[int], threshold: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Subarray With Elements Greater Than Varying Threshold
* Difficulty: Hard
* Tags: array, graph, stack
*
* 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} threshold
* @return {number}
*/
var validSubarraySize = function(nums, threshold) {

};
```

TypeScript Solution:

```
/**
* Problem: Subarray With Elements Greater Than Varying Threshold
* Difficulty: Hard
* Tags: array, graph, stack
*
* 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 validSubarraySize(nums: number[], threshold: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Subarray With Elements Greater Than Varying Threshold
* Difficulty: Hard
* Tags: array, graph, stack
*
* 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 ValidSubarraySize(int[] nums, int threshold) {

}
}
```

C Solution:

```
/*
* Problem: Subarray With Elements Greater Than Varying Threshold
* Difficulty: Hard
* Tags: array, graph, stack
*
* 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 validSubarraySize(int* nums, int numsSize, int threshold) {

}
```

Go Solution:

```
// Problem: Subarray With Elements Greater Than Varying Threshold
// Difficulty: Hard
```

```
// Tags: array, graph, stack
//
// 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 validSubarraySize(nums []int, threshold int) int {

}
```

Kotlin Solution:

```
class Solution {
fun validSubarraySize(nums: IntArray, threshold: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Subarray With Elements Greater Than Varying Threshold
// Difficulty: Hard
// Tags: array, graph, stack
//
// 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 valid_subarray_size(nums: Vec<i32>, threshold: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} threshold
# @return {Integer}
def valid_subarray_size(nums, threshold)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $threshold
* @return Integer
*/
function validSubarraySize($nums, $threshold) {

}
}
```

Dart Solution:

```
class Solution {
int validSubarraySize(List<int> nums, int threshold) {

}
}
```

Scala Solution:

```
object Solution {
def validSubarraySize(nums: Array[Int], threshold: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec valid_subarray_size(nums :: [integer], threshold :: integer) :: integer
def valid_subarray_size(nums, threshold) do

end
end
```

Erlang Solution:

```
-spec valid_subarray_size(Nums :: [integer()], Threshold :: integer()) ->
integer().
valid_subarray_size(Nums, Threshold) ->
.
```

Racket Solution:

```
(define/contract (valid-subarray-size nums threshold)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```