Problem 862: Shortest Subarray with Sum at Least K

Problem Information

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

Problem Description

Given an integer array

nums

and an integer

k

, return

the length of the shortest non-empty

subarray

of

nums

with a sum of at least

k

. If there is no such

subarray

, return

-1

.

A

subarray

is a

contiguous

part of an array.

Example 1:

Input:

nums = [1], k = 1

Output:

1

Example 2:

Input:

nums = [1,2], k = 4

Output:

-1

Example 3:

Input:

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

Output:

3

Constraints:

1 <= nums.length <= 10

5

-10

5

<= nums[i] <= 10

5

1 <= k <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (shortest-subarray nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Subarray with Sum at Least K
* Difficulty: Hard
* Tags: array, search, queue, heap
*
* 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 shortestSubarray(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Subarray with Sum at Least K
```

```
* Difficulty: Hard
* Tags: array, search, queue, heap
*
* 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 shortestSubarray(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Subarray with Sum at Least K
Difficulty: Hard
Tags: array, search, queue, heap

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Subarray with Sum at Least K
* Difficulty: Hard
* Tags: array, search, queue, heap
*
* 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 shortestSubarray = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Subarray with Sum at Least K
* Difficulty: Hard
* Tags: array, search, queue, heap
*
* 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 shortestSubarray(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Shortest Subarray with Sum at Least K
* Difficulty: Hard
* Tags: array, search, queue, heap
```

```
*
* 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 ShortestSubarray(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Subarray with Sum at Least K
* Difficulty: Hard
* Tags: array, search, queue, heap
*
* 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 shortestSubarray(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Shortest Subarray with Sum at Least K
// Difficulty: Hard
// Tags: array, search, queue, heap
//
// 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 shortestSubarray(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shortest Subarray with Sum at Least K
// Difficulty: Hard
// Tags: array, search, queue, heap
//
// 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 shortest_subarray(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (shortest-subarray nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```