Problem 3095: Shortest Subarray With OR at Least K I

Problem Information

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

Problem Description

You are given an array

nums

of

non-negative

integers and an integer

k

.

An array is called

special

if the bitwise

OR

of all of its elements is

at least

k

.

Return

the length of the

shortest

special

non-empty

subarray

of

nums

,

or return

-1

if no special subarray exists

.

Example 1:

Input:

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

Output:

1

Explanation:

The subarray

[3]

has

OR

value of

3

. Hence, we return

1

.

Note that

[2]

is also a special subarray.

Example 2:

Input:

nums = [2,1,8], k = 10

Output:

3

Explanation:

The subarray

[2,1,8]

has

OR

value of

11

. Hence, we return

3

.

Example 3:

Input:

nums = [1,2], k = 0

Output:

1

Explanation:

The subarray

[1]

has

OR

value of

1

. Hence, we return

1

.

Constraints:

1 <= nums.length <= 50

0 <= nums[i] <= 50

0 <= k < 64

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Shortest Subarray With OR at Least K I
* Difficulty: Easy
* 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 minimumSubarrayLength(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Subarray With OR at Least K I
* Difficulty: Easy
* 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 minimumSubarrayLength(int[] nums, int k) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Shortest Subarray With OR at Least K I
Difficulty: Easy
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 minimumSubarrayLength(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Subarray With OR at Least K I
* Difficulty: Easy
* 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 minimumSubarrayLength = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Subarray With OR at Least K I
* Difficulty: Easy
* 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 minimumSubarrayLength(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Shortest Subarray With OR at Least K I
* Difficulty: Easy
* 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 MinimumSubarrayLength(int[] nums, int k) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Shortest Subarray With OR at Least K I
* Difficulty: Easy
* 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 minimumSubarrayLength(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Shortest Subarray With OR at Least K I
// Difficulty: Easy
// 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 minimumSubarrayLength(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shortest Subarray With OR at Least K I
// Difficulty: Easy
// 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 minimum_subarray_length(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
```

```
function minimumSubarrayLength($nums, $k) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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

```
)
```