Problem 3171: Find Subarray With Bitwise OR Closest to K

Problem Information

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

Problem Description

You are given an array

nums

and an integer

k

. You need to find a

subarray

of

nums

such that the

absolute difference

between

k

and the bitwise

OR

of the subarray elements is as

small

as possible. In other words, select a subarray

nums[l..r]

such that

|k - (nums[l] OR nums[l + 1] ... OR nums[r])|

is minimum.

Return the

minimum

possible value of the absolute difference.

A

subarray

is a contiguous

non-empty

sequence of elements within an array.

Example 1:

Input:

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

Output:

0

Explanation:

The subarray

nums[0..1]

has

OR

value 3, which gives the minimum absolute difference

|3 - 3| = 0

.

Example 2:

Input:

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

Output:

1

Explanation:

The subarray

nums[1..1]

has

OR

value 3, which gives the minimum absolute difference

|3 - 2| = 1

.

Example 3:

Input:

nums = [1], k = 10

Output:

9

Explanation:

There is a single subarray with

OR

value 1, which gives the minimum absolute difference

|10 - 1| = 9

.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

1 <= k <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find Subarray With Bitwise OR Closest to K
* Difficulty: Hard
```

```
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Find Subarray With Bitwise OR Closest to K
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Find Subarray With Bitwise OR Closest to K
Difficulty: Hard
Tags: array, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
```

```
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Subarray With Bitwise OR Closest to K
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Find Subarray With Bitwise OR Closest to K
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

C# Solution:

```
/*
* Problem: Find Subarray With Bitwise OR Closest to K
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
}
```

C Solution:

```
/*
* Problem: Find Subarray With Bitwise OR Closest to K
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

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

}
```

Go Solution:

```
// Problem: Find Subarray With Bitwise OR Closest to K
// Difficulty: Hard
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Subarray With Bitwise OR Closest to K
// Difficulty: Hard
// Tags: array, tree, search
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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