Problem 2401: Longest Nice Subarray

Problem Information

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

Problem Description

You are given an array

nums

consisting of

positive

integers.

We call a subarray of

nums

nice

if the bitwise

AND

of every pair of elements that are in

different

positions in the subarray is equal to

0

.

Return

the length of the

longest

nice subarray

.

A

subarray

is a

contiguous

part of an array.

Note

that subarrays of length

1

are always considered nice.

Example 1:

Input:

nums = [1,3,8,48,10]

Output:

3

Explanation:

The longest nice subarray is [3,8,48]. This subarray satisfies the conditions: - 3 AND 8 = 0. - 3 AND 48 = 0. - 8 AND 48 = 0. It can be proven that no longer nice subarray can be obtained, so we return 3.

Example 2:

Input:

nums = [3,1,5,11,13]

Output:

1

Explanation:

The length of the longest nice subarray is 1. Any subarray of length 1 can be chosen.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
```

```
int longestNiceSubarray(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int longestNiceSubarray(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function longestNiceSubarray(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int LongestNiceSubarray(int[] nums) {

}
}
```

C:

```
int longestNiceSubarray(int* nums, int numsSize) {

}
```

Go:

```
func longestNiceSubarray(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun longestNiceSubarray(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func longestNiceSubarray(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn longest_nice_subarray(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def longest_nice_subarray(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function longestNiceSubarray($nums) {

}
}
```

Dart:

```
class Solution {
int longestNiceSubarray(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def longestNiceSubarray(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_nice_subarray(nums :: [integer]) :: integer
def longest_nice_subarray(nums) do
```

```
end
end
```

Erlang:

```
-spec longest_nice_subarray(Nums :: [integer()]) -> integer().
longest_nice_subarray(Nums) ->
.
```

Racket:

```
(define/contract (longest-nice-subarray nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Nice Subarray
* 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 longestNiceSubarray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Nice Subarray
```

```
* 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 longestNiceSubarray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Nice Subarray
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 longestNiceSubarray(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Nice Subarray
* 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[]} nums
* @return {number}
*/
var longestNiceSubarray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Nice Subarray
* 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 longestNiceSubarray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Longest Nice Subarray
* 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 LongestNiceSubarray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Longest Nice Subarray
* 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 longestNiceSubarray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Longest Nice Subarray
// 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 longestNiceSubarray(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestNiceSubarray(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func longestNiceSubarray(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Nice Subarray
// 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 longest_nice_subarray(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def longest_nice_subarray(nums)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $nums
* @return Integer
*/
function longestNiceSubarray($nums) {

}
}
```

Dart Solution:

```
class Solution {
int longestNiceSubarray(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def longestNiceSubarray(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_nice_subarray(nums :: [integer]) :: integer
def longest_nice_subarray(nums) do

end
end
```

Erlang Solution:

```
-spec longest_nice_subarray(Nums :: [integer()]) -> integer().
longest_nice_subarray(Nums) ->
.
```

Racket Solution:

```
(define/contract (longest-nice-subarray nums)
(-> (listof exact-integer?) exact-integer?)
)
```