Problem 3719: Longest Balanced Subarray I

Problem Information

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

Problem Description

You are given an integer array

nums

.

A

subarray

is called

balanced

if the number of

distinct even

numbers in the subarray is equal to the number of

distinct odd

numbers.

Return the length of the

longest

balanced subarray.

Example 1:

Input:

nums = [2,5,4,3]

Output:

4

Explanation:

The longest balanced subarray is

[2, 5, 4, 3]

.

It has 2 distinct even numbers

[2, 4]

and 2 distinct odd numbers

[5, 3]

. Thus, the answer is 4.

Example 2:

Input:

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

Output:

5

Explanation:

The longest balanced subarray is

[3, 2, 2, 5, 4]

.

It has 2 distinct even numbers

[2, 4]

and 2 distinct odd numbers

[3, 5]

. Thus, the answer is 5.

Example 3:

Input:

nums = [1,2,3,2]

Output:

3

Explanation:

The longest balanced subarray is

[2, 3, 2]

.

It has 1 distinct even number

[2]

and 1 distinct odd number

[3]

. Thus, the answer is 3.

Constraints:

1 <= nums.length <= 1500

1 <= nums[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int longestBalanced(vector<int>& nums) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Balanced Subarray I
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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 longestBalanced(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Balanced Subarray I
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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 longestBalanced(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Balanced Subarray I
Difficulty: Medium
Tags: array, tree, hash

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Balanced Subarray I
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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
* @return {number}
*/
var longestBalanced = function(nums) {
};
```

TypeScript Solution:

```
/**
* Problem: Longest Balanced Subarray I
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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 longestBalanced(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Longest Balanced Subarray I
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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 LongestBalanced(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Longest Balanced Subarray I
* Difficulty: Medium
* Tags: array, tree, hash
*
* 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 longestBalanced(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Longest Balanced Subarray I
// Difficulty: Medium
// Tags: array, tree, hash
//
// 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 longestBalanced(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Balanced Subarray I
// Difficulty: Medium
// Tags: array, tree, hash
//
// 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 longest_balanced(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int longestBalanced(List<int> nums) {
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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