Problem 525: Contiguous Array

Problem Information

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

Problem Description

Given a binary array

nums

, return

the maximum length of a contiguous subarray with an equal number of

0

and

1

.

Example 1:

Input:

nums = [0,1]

Output:

2

Explanation:

[0, 1] is the longest contiguous subarray with an equal number of 0 and 1.

Example 2:

Input:

nums = [0,1,0]

Output:

2

Explanation:

[0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.

Example 3:

Input:

nums = [0,1,1,1,1,1,0,0,0]

Output:

6

Explanation:

[1,1,1,0,0,0] is the longest contiguous subarray with equal number of 0 and 1.

Constraints:

1 <= nums.length <= 10

5

nums[i]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-max-length nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Contiguous Array
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Contiguous Array
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int findMaxLength(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Contiguous Array
Difficulty: Medium
Tags: array, hash
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Contiguous Array
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Contiguous Array
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function findMaxLength(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Contiguous Array
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int FindMaxLength(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Contiguous Array
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

int findMaxLength(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Contiguous Array
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func findMaxLength(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Contiguous Array
// Difficulty: Medium
// Tags: array, hash
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-max-length nums)
(-> (listof exact-integer?) exact-integer?)
)
```