Problem 1664: Ways to Make a Fair Array

Problem Information

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

Problem Description

You are given an integer array

nums

. You can choose

exactly one

index (

0-indexed

) and remove the element. Notice that the index of the elements may change after the removal.

For example, if

nums = [6,1,7,4,1]

:

Choosing to remove index

1

results in

nums = [6,7,4,1]

.

Choosing to remove index

2

results in

nums = [6,1,4,1]

.

Choosing to remove index

4

results in

nums = [6,1,7,4]

.

An array is

fair

if the sum of the odd-indexed values equals the sum of the even-indexed values.

Return the

number

of indices that you could choose such that after the removal,

nums

is

fair

.

Example 1:

Input:

nums = [2,1,6,4]

Output:

1

Explanation:

Remove index 0: [1,6,4] -> Even sum: 1 + 4 = 5. Odd sum: 6. Not fair. Remove index 1: [2,6,4] -> Even sum: 2 + 4 = 6. Odd sum: 6. Fair. Remove index 2: [2,1,4] -> Even sum: 2 + 4 = 6. Odd sum: 1. Not fair. Remove index 3: [2,1,6] -> Even sum: 2 + 6 = 8. Odd sum: 1. Not fair. There is 1 index that you can remove to make nums fair.

Example 2:

Input:

nums = [1,1,1]

Output:

3

Explanation:

You can remove any index and the remaining array is fair.

Example 3:

Input:

nums = [1,2,3]

Output:

0

Explanation:

You cannot make a fair array after removing any index.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (ways-to-make-fair nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Ways to Make a Fair Array
* 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 waysToMakeFair(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Ways to Make a Fair Array
* 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 waysToMakeFair(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Ways to Make a Fair Array
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 waysToMakeFair(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Ways to Make a Fair Array
* 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 waysToMakeFair = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Ways to Make a Fair Array
* 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 waysToMakeFair(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Ways to Make a Fair Array
* 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 WaysToMakeFair(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Ways to Make a Fair Array
* 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 waysToMakeFair(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Ways to Make a Fair Array
// 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 waysToMakeFair(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Ways to Make a Fair Array
// 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 ways_to_make_fair(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (ways-to-make-fair nums)
(-> (listof exact-integer?) exact-integer?)
)
```