Problem 3038: Maximum Number of Operations With the Same Score I

Problem Information

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

Problem Description

You are given an array of integers

nums

. Consider the following operation:

Delete the first two elements

nums

and define the

score

of the operation as the sum of these two elements.

You can perform this operation until

nums

contains fewer than two elements. Additionally, the

same

score

must be achieved in

all

operations.

Return the

maximum

number of operations you can perform.

Example 1:

Input:

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

Output:

2

Explanation:

We can perform the first operation with the score

3 + 2 = 5

. After this operation,

nums = [1,4,5]

.

We can perform the second operation as its score is

4 + 1 = 5

, the same as the previous operation. After this operation,

nums = [5]

.

As there are fewer than two elements, we can't perform more operations.

Example 2:

Input:

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

Output:

2

Explanation:

We can perform the first operation with the score

1 + 5 = 6

. After this operation,

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

.

We can perform the second operation as its score is

3 + 3 = 6

, the same as the previous operation. After this operation,

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

.

We cannot perform the next operation as its score is

4 + 1 = 5

, which is different from the previous scores.

Example 3:

Input:

nums = [5,3]

Output:

1

Constraints:

2 <= nums.length <= 100

1 <= nums[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Operations With the Same Score I
* Difficulty: Easy
* 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 maxOperations(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Operations With the Same Score I
* Difficulty: Easy
* 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 maxOperations(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Operations With the Same Score I
Difficulty: Easy
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 maxOperations(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Operations With the Same Score I
* Difficulty: Easy
* 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 maxOperations = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Operations With the Same Score I
* Difficulty: Easy
* 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 maxOperations(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Operations With the Same Score I
* Difficulty: Easy
* 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 MaxOperations(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Operations With the Same Score I
* Difficulty: Easy
* 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 maxOperations(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Operations With the Same Score I
// Difficulty: Easy
// 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 maxOperations(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Maximum Number of Operations With the Same Score I
// Difficulty: Easy
// 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 max_operations(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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