Problem 3040: Maximum Number of Operations With the Same Score II

Problem Information

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

Problem Description

Given an array of integers called

nums

, you can perform

any

of the following operation while

nums

contains

at least

2

elements:

Choose the first two elements of

nums

and delete them.

Choose the last two elements of

nums

and delete them.

Choose the first and the last elements of

nums

and delete them.

The

score

of the operation is the sum of the deleted elements.

Your task is to find the

maximum

number of operations that can be performed, such that

all operations have the same score

.

Return

the

maximum

number of operations possible that satisfy the condition mentioned above

.

Example 1:

Input:

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

Output:

3

Explanation:

We perform the following operations: - Delete the first two elements, with score 3 + 2 = 5, nums = [1,2,3,4]. - Delete the first and the last elements, with score 1 + 4 = 5, nums = [2,3]. - Delete the first and the last elements, with score 2 + 3 = 5, nums = []. We are unable to perform any more operations as nums is empty.

Example 2:

Input:

nums = [3,2,6,1,4]

Output:

2

Explanation:

We perform the following operations: - Delete the first two elements, with score 3 + 2 = 5, nums = [6,1,4]. - Delete the last two elements, with score 1 + 4 = 5, nums = [6]. It can be proven that we can perform at most 2 operations.

Constraints:

2 <= nums.length <= 2000

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 II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

```
}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Operations With the Same Score II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Operations With the Same Score II
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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 II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Operations With the Same Score II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Operations With the Same Score II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Operations With the Same Score II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxOperations(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Operations With the Same Score II
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(n) or O(n * m) for DP table

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 II
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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?)
)
```