Problem 1674: Minimum Moves to Make Array Complementary

Problem Information

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

Problem Description

You are given an integer array

nums

of

even

length

n

and an integer

limit

. In one move, you can replace any integer from

nums

with another integer between

1

and

limit

, inclusive.

The array

nums

is

complementary

if for all indices

i

(

0-indexed

),

nums[i] + nums[n - 1 - i]

equals the same number. For example, the array

[1,2,3,4]

is complementary because for all indices

i

,

nums[i] + nums[n - 1 - i] = 5

.

Return the

minimum

number of moves required to make

nums

complementary

.

Example 1:

Input:

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

Output:

1

Explanation:

In 1 move, you can change nums to [1,2,

2

,3] (underlined elements are changed). nums[0] + nums[3] = 1 + 3 = 4. nums[1] + nums[2] = 2 + 2 = 4. nums[2] + nums[1] = 2 + 2 = 4. nums[3] + nums[0] = 3 + 1 = 4. Therefore, nums[i] + nums[n-1-i] = 4 for every i, so nums is complementary.

Example 2:

Input:

nums = [1,2,2,1], limit = 2

Output:

2

Explanation:

In 2 moves, you can change nums to [

2

,2,2,

2

]. You cannot change any number to 3 since 3 > limit.

Example 3:

Input:

nums = [1,2,1,2], limit = 2

Output:

0

Explanation:

nums is already complementary.

Constraints:

n == nums.length

2 <= n <= 10

5

1 <= nums[i] <= limit <= 10

5

n

is even.

Code Snippets

C++:

```
class Solution {
public:
int minMoves(vector<int>& nums, int limit) {

}
};
```

Java:

```
class Solution {
public int minMoves(int[] nums, int limit) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minMoves(nums: number[], limit: number): number {

};
```

C#:

```
public class Solution {
public int MinMoves(int[] nums, int limit) {

}
}
```

C:

```
int minMoves(int* nums, int numsSize, int limit) {

}
```

Go:

```
func minMoves(nums []int, limit int) int {

}
```

Kotlin:

```
class Solution {
fun minMoves(nums: IntArray, limit: Int): Int {

}
}
```

Swift:

```
class Solution {
func minMoves(_ nums: [Int], _ limit: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_moves(nums: Vec<i32>, limit: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} limit
# @return {Integer}
def min_moves(nums, limit)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $limit
* @return Integer
*/
function minMoves($nums, $limit) {

}
}
```

Dart:

```
class Solution {
int minMoves(List<int> nums, int limit) {
```

```
}
}
```

Scala:

```
object Solution {
def minMoves(nums: Array[Int], limit: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_moves(nums :: [integer], limit :: integer) :: integer
def min_moves(nums, limit) do

end
end
```

Erlang:

```
-spec min_moves(Nums :: [integer()], Limit :: integer()) -> integer().
min_moves(Nums, Limit) ->
.
```

Racket:

```
(define/contract (min-moves nums limit)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Moves to Make Array Complementary
* 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 minMoves(vector<int>& nums, int limit) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Moves to Make Array Complementary
* 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 minMoves(int[] nums, int limit) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Moves to Make Array Complementary
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 minMoves(self, nums: List[int], limit: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Moves to Make Array Complementary
* 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
* @param {number} limit
* @return {number}
*/
var minMoves = function(nums, limit) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Moves to Make Array Complementary
* 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 minMoves(nums: number[], limit: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Moves to Make Array Complementary
* 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 MinMoves(int[] nums, int limit) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Moves to Make Array Complementary
* 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 minMoves(int* nums, int numsSize, int limit) {

}
```

Go Solution:

```
// Problem: Minimum Moves to Make Array Complementary
// 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 minMoves(nums []int, limit int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minMoves(nums: IntArray, limit: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minMoves(_ nums: [Int], _ limit: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Moves to Make Array Complementary
// 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 min_moves(nums: Vec<i32>, limit: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} limit
# @return {Integer}
def min_moves(nums, limit)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $limit
* @return Integer
*/
function minMoves($nums, $limit) {

}
}
```

Dart Solution:

```
class Solution {
int minMoves(List<int> nums, int limit) {

}
}
```

Scala Solution:

```
object Solution {
def minMoves(nums: Array[Int], limit: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_moves(nums :: [integer], limit :: integer) :: integer
def min_moves(nums, limit) do

end
end
```

Erlang Solution:

```
-spec min_moves(Nums :: [integer()], Limit :: integer()) -> integer().
min_moves(Nums, Limit) ->
.
```

Racket Solution:

```
(define/contract (min-moves nums limit)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```