Problem 453: Minimum Moves to Equal Array Elements

Problem Information

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

Problem Description

Given an integer array

nums

of size

n

, return

the minimum number of moves required to make all array elements equal

.

In one move, you can increment

n - 1

elements of the array by

1

.

Example 1:

Input:

nums = [1,2,3]

Output:

3

Explanation:

Only three moves are needed (remember each move increments two elements): [1,2,3] => [2,3,3] => [3,4,3] => [4,4,4]

Example 2:

Input:

nums = [1,1,1]

Output:

0

Constraints:

n == nums.length

1 <= nums.length <= 10

5

-10

9

<= nums[i] <= 10

9

The answer is guaranteed to fit in a

32-bit

integer.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Moves to Equal Array Elements
* Difficulty: Medium
* Tags: array, math
*
* 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 minMoves(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Moves to Equal Array Elements
* Difficulty: Medium
* Tags: array, math
*
* 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 minMoves(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Moves to Equal Array Elements
Difficulty: Medium
Tags: array, math

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 minMoves(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Moves to Equal Array Elements
* Difficulty: Medium
* Tags: array, math
*
* 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 minMoves = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Moves to Equal Array Elements
* Difficulty: Medium
* Tags: array, math
*
* 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 minMoves(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Moves to Equal Array Elements
* Difficulty: Medium
* Tags: array, math
*
* 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 MinMoves(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Moves to Equal Array Elements
* Difficulty: Medium
* Tags: array, math
*
* 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 minMoves(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Moves to Equal Array Elements
// Difficulty: Medium
// Tags: array, math
//
// 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 minMoves(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Moves to Equal Array Elements
// Difficulty: Medium
// Tags: array, math
//
// 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 min_moves(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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