Problem 1551: Minimum Operations to Make Array Equal

Problem Information

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

Problem Description

You have an array

arr

of length

n

where

arr[i] = (2 * i) + 1

for all valid values of

i

(i.e.,

0 <= i < n

).

In one operation, you can select two indices

x

and

y

where

0 <= x, y < n

and subtract

1

from

arr[x]

and add

1

to

arr[y]

(i.e., perform

arr[x] -=1

and

arr[y] += 1

). The goal is to make all the elements of the array

equal

. It is

guaranteed

that all the elements of the array can be made equal using some operations.

Given an integer

n

, the length of the array, return

the minimum number of operations

needed to make all the elements of arr equal.

Example 1:

Input:

n = 3

Output:

2

Explanation:

arr = [1, 3, 5] First operation choose x = 2 and y = 0, this leads arr to be [2, 3, 4] In the second operation choose x = 2 and y = 0 again, thus arr = [3, 3, 3].

Example 2:

Input:

n = 6

Output:

9

Constraints:

1 <= n <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int minOperations(int n) {

}
};
```

Java:

```
class Solution {
public int minOperations(int n) {

}
}
```

Python3:

```
class Solution:
def minOperations(self, n: int) -> int:
```

Python:

```
class Solution(object):
def minOperations(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var minOperations = function(n) {

};
```

TypeScript:

```
function minOperations(n: number): number {

};
```

C#:

```
public class Solution {
public int MinOperations(int n) {

}
}
```

C:

```
int minOperations(int n) {

}
```

Go:

```
func minOperations(n int) int {

}
```

Kotlin:

```
class Solution {
fun minOperations(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func minOperations(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_operations(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def min_operations(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function minOperations($n) {

}
}
```

Dart:

```
class Solution {
int minOperations(int n) {

}
}
```

Scala:

```
object Solution {
def minOperations(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations(n :: integer) :: integer
def min_operations(n) do

end
end
```

Erlang:

```
-spec min_operations(N :: integer()) -> integer().
min_operations(N) ->
.
```

Racket:

```
(define/contract (min-operations n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make Array Equal
* 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 minOperations(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make Array Equal
* 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 minOperations(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make Array Equal
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 minOperations(self, n: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def minOperations(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make Array Equal
* 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} n
* @return {number}
*/
var minOperations = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make Array Equal
* 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 minOperations(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make Array Equal
* 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 MinOperations(int n) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make Array Equal
* 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 minOperations(int n) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make Array Equal
// 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 minOperations(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minOperations(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make Array Equal
// 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_operations(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def min_operations(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function minOperations($n) {

}
}
```

Dart Solution:

```
class Solution {
int minOperations(int n) {

}
}
```

Scala Solution:

```
object Solution {
def minOperations(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations(n :: integer) :: integer
def min_operations(n) do

end
end
```

Erlang Solution:

```
-spec min_operations(N :: integer()) -> integer().
min_operations(N) ->
.
```

Racket Solution:

```
(define/contract (min-operations n)
(-> exact-integer? exact-integer?)
)
```