Problem 1775: Equal Sum Arrays With Minimum Number of Operations

Problem Information

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

Problem Description

You are given two arrays of integers

nums1

and

nums2

, possibly of different lengths. The values in the arrays are between

1

and

6

, inclusive.

In one operation, you can change any integer's value in

any

of the arrays to

any

value between

1

and

6

, inclusive.

Return

the minimum number of operations required to make the sum of values in

nums1

equal to the sum of values in

nums2

.

Return

-1

if it is not possible to make the sum of the two arrays equal.

Example 1:

Input:

nums1 = [1,2,3,4,5,6], nums2 = [1,1,2,2,2,2]

Output:

3

Explanation:

You can make the sums of nums1 and nums2 equal with 3 operations. All indices are 0-indexed. - Change nums2[0] to 6. nums1 = [1,2,3,4,5,6], nums2 = [

6

,1,2,2,2,2]. - Change nums1[5] to 1. nums1 = [1,2,3,4,5,

1

], nums2 = [6,1,2,2,2,2]. - Change nums1[2] to 2. nums1 = [1,2,

2

,4,5,1], nums2 = [6,1,2,2,2,2].

Example 2:

Input:

nums1 = [1,1,1,1,1,1,1], nums2 = [6]

Output:

-1

Explanation:

There is no way to decrease the sum of nums1 or to increase the sum of nums2 to make them equal.

Example 3:

Input:

nums1 = [6,6], nums2 = [1]

Output:

3

Explanation:

You can make the sums of nums1 and nums2 equal with 3 operations. All indices are 0-indexed. - Change nums1[0] to 2. nums1 = [

2

,6], nums2 = [1]. - Change nums1[1] to 2. nums1 = [2,

2

], nums2 = [1]. - Change nums2[0] to 4. nums1 = [2,2], nums2 = [

4

].

Constraints:

1 <= nums1.length, nums2.length <= 10

5

1 <= nums1[i], nums2[i] <= 6

Code Snippets

C++:

```
class Solution {
public:
int minOperations(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java:

```
class Solution {
public int minOperations(int[] nums1, int[] nums2) {

}
}
```

Python3:

```
class Solution:
def minOperations(self, nums1: List[int], nums2: List[int]) -> int:
```

Python:

```
class Solution(object):
def minOperations(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var minOperations = function(nums1, nums2) {

};
```

TypeScript:

```
function minOperations(nums1: number[], nums2: number[]): number {

};
```

C#:

```
public class Solution {
public int MinOperations(int[] nums1, int[] nums2) {

}
}
```

C:

```
int minOperations(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go:

```
func minOperations(nums1 []int, nums2 []int) int {

}
```

Kotlin:

```
class Solution {
fun minOperations(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minOperations(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_operations(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def min_operations(nums1, nums2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function minOperations($nums1, $nums2) {

}
}
```

Dart:

```
class Solution {
int minOperations(List<int> nums1, List<int> nums2) {

}
}
```

Scala:

```
object Solution {
def minOperations(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations(nums1 :: [integer], nums2 :: [integer]) :: integer
def min_operations(nums1, nums2) do
```

```
end
end
```

Erlang:

```
-spec min_operations(Nums1 :: [integer()], Nums2 :: [integer()]) ->
integer().
min_operations(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (min-operations nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Equal Sum Arrays With Minimum Number of Operations
* Difficulty: Medium
* Tags: array, greedy, 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 minOperations(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Equal Sum Arrays With Minimum Number of Operations
* Difficulty: Medium
* Tags: array, greedy, 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 minOperations(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Equal Sum Arrays With Minimum Number of Operations
Difficulty: Medium
Tags: array, greedy, 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 minOperations(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minOperations(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Equal Sum Arrays With Minimum Number of Operations
* Difficulty: Medium
* Tags: array, greedy, 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[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var minOperations = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Equal Sum Arrays With Minimum Number of Operations
* Difficulty: Medium
* Tags: array, greedy, 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 minOperations(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Equal Sum Arrays With Minimum Number of Operations
* Difficulty: Medium
```

```
* Tags: array, greedy, 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 MinOperations(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Equal Sum Arrays With Minimum Number of Operations
* Difficulty: Medium
* Tags: array, greedy, 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 minOperations(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go Solution:

```
// Problem: Equal Sum Arrays With Minimum Number of Operations
// Difficulty: Medium
// Tags: array, greedy, 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 minOperations(nums1 []int, nums2 []int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun minOperations(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minOperations(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Equal Sum Arrays With Minimum Number of Operations
// Difficulty: Medium
// Tags: array, greedy, 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_operations(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def min_operations(nums1, nums2)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function minOperations($nums1, $nums2) {

}
}
```

Dart Solution:

```
class Solution {
int minOperations(List<int> nums1, List<int> nums2) {

}
}
```

Scala Solution:

```
object Solution {
def minOperations(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations(nums1 :: [integer], nums2 :: [integer]) :: integer
def min_operations(nums1, nums2) do

end
end
```

Erlang Solution:

```
-spec min_operations(Nums1 :: [integer()], Nums2 :: [integer()]) ->
integer().
min_operations(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (min-operations nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```