Problem 2918: Minimum Equal Sum of Two Arrays After Replacing Zeros

Problem Information

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

Problem Description

You are given two arrays

nums1

and

nums2

consisting of positive integers.

You have to replace

all

the

0

's in both arrays with

strictly

positive integers such that the sum of elements of both arrays becomes

equal

.

Return

the

minimum

equal sum you can obtain, or

-1

if it is impossible

.

Example 1:

Input:

nums1 = [3,2,0,1,0], nums2 = [6,5,0]

Output:

12

Explanation:

We can replace 0's in the following way: - Replace the two 0's in nums1 with the values 2 and 4. The resulting array is nums1 = [3,2,2,1,4]. - Replace the 0 in nums2 with the value 1. The resulting array is nums2 = [6,5,1]. Both arrays have an equal sum of 12. It can be shown that it is the minimum sum we can obtain.

Example 2:

Input:

nums1 = [2,0,2,0], nums2 = [1,4]

Output:

-1

Explanation:

It is impossible to make the sum of both arrays equal.

Constraints:

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

5

0 <= nums1[i], nums2[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
long long minSum(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go:

```
func minSum(nums1 []int, nums2 []int) int64 {

}
```

Kotlin:

```
class Solution {
fun minSum(nums1: IntArray, nums2: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn min_sum(nums1: Vec<i32>, nums2: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Equal Sum of Two Arrays After Replacing Zeros
* Difficulty: Medium
* Tags: array, greedy
*
* 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:
long long minSum(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Equal Sum of Two Arrays After Replacing Zeros
* Difficulty: Medium
* Tags: array, greedy
*
* 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 long minSum(int[] nums1, int[] nums2) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Equal Sum of Two Arrays After Replacing Zeros
Difficulty: Medium
Tags: array, greedy

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 minSum(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Equal Sum of Two Arrays After Replacing Zeros
* Difficulty: Medium
* Tags: array, greedy
*
* 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[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var minSum = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Equal Sum of Two Arrays After Replacing Zeros
* Difficulty: Medium
* Tags: array, greedy
*
* 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 minSum(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Equal Sum of Two Arrays After Replacing Zeros
* Difficulty: Medium
* Tags: array, greedy
*
* 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 long MinSum(int[] nums1, int[] nums2) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Equal Sum of Two Arrays After Replacing Zeros
* Difficulty: Medium
* Tags: array, greedy
*
* 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
*/

long long minSum(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go Solution:

```
// Problem: Minimum Equal Sum of Two Arrays After Replacing Zeros
// Difficulty: Medium
// Tags: array, greedy
//
// 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 minSum(nums1 []int, nums2 []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minSum(nums1: IntArray, nums2: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Equal Sum of Two Arrays After Replacing Zeros
// Difficulty: Medium
// Tags: array, greedy
//
// 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_sum(nums1: Vec<i32>, nums2: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

```
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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