Problem 3682: Minimum Index Sum of Common Elements

Problem Information

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

Problem Description

You are given two integer arrays

nums1

and

nums2

of equal length

n

.

We define a pair of indices

(i, j)

as a

good pair

if

nums1[i] == nums2[j]

.

Return the

minimum index sum

i + j

among all possible good pairs. If no such pairs exist, return -1.

Example 1:

Input:

nums1 = [3,2,1], nums2 = [1,3,1]

Output:

1

Explanation:

Common elements between

nums1

and

nums2

are 1 and 3.

For 3,

[i, j] = [0, 1]

, giving an index sum of

i + j = 1

.

For 1,

[i, j] = [2, 0]

, giving an index sum of

i + j = 2

.

The minimum index sum is 1.

Example 2:

Input:

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

Output:

2

Explanation:

Common elements between

nums1

and

nums2

are 1 and 2.

For 1,

[i, j] = [1, 1]

, giving an index sum of

i + j = 2

.

For 2,

[i, j] = [2, 0]

, giving an index sum of

i + j = 2

.

The minimum index sum is 2.

Example 3:

Input:

nums1 = [6,4], nums2 = [7,8]

Output:

-1

Explanation:

Since no common elements between

nums1

and

nums2

, the output is -1.

Constraints:

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

5

-10

5

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

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Index Sum of Common Elements
* 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 minimumSum(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Index Sum of Common Elements
* 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 minimumSum(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Index Sum of Common Elements
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 minimumSum(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Index Sum of Common Elements
* 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[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var minimumSum = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Index Sum of Common Elements
* 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 minimumSum(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Index Sum of Common Elements
* 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 MinimumSum(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Index Sum of Common Elements
* 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 minimumSum(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go Solution:

```
// Problem: Minimum Index Sum of Common Elements
// 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 minimumSum(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Index Sum of Common Elements
// 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 minimum_sum(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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