Problem 2499: Minimum Total Cost to Make Arrays Unequal

Problem Information

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

Problem Description

You are given two

0-indexed

integer arrays

nums1

and

nums2

, of equal length

n

.

In one operation, you can swap the values of any two indices of

nums1

. The

cost

of this operation is the

sum

of the indices.

Find the

minimum

total cost of performing the given operation

any

number of times such that

nums1[i] != nums2[i]

for all

0 <= i <= n - 1

after performing all the operations.

Return

the

minimum total cost

such that

nums1

and

nums2

satisfy the above condition

. In case it is not possible, return

-1

.

Example 1:

Input:

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

Output:

10

Explanation:

One of the ways we can perform the operations is: - Swap values at indices 0 and 3, incurring cost = 0 + 3 = 3. Now, nums1 = [4,2,3,1,5] - Swap values at indices 1 and 2, incurring cost = 1 + 2 = 3. Now, nums1 = [4,3,2,1,5]. - Swap values at indices 0 and 4, incurring cost = 0 + 4 = 4. Now, nums1 =[5,3,2,1,4]. We can see that for each index i, nums1[i] != nums2[i]. The cost required here is 10. Note that there are other ways to swap values, but it can be proven that it is not possible to obtain a cost less than 10.

Example 2:

Input:

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

Output:

10

Explanation:

One of the ways we can perform the operations is: - Swap values at indices 2 and 3, incurring cost = 2 + 3 = 5. Now, nums1 = [2,2,1,2,3]. - Swap values at indices 1 and 4, incurring cost = 1 + 4 = 5. Now, nums1 = [2,3,1,2,2]. The total cost needed here is 10, which is the minimum possible.

Example 3:

Input:

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

Output:

-1

Explanation:

It can be shown that it is not possible to satisfy the given conditions irrespective of the number of operations we perform. Hence, we return -1.

Constraints:

n == nums1.length == nums2.length

1 <= n <= 10

5

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

Code Snippets

C++:

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

}
```

```
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
```

```
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Total Cost to Make Arrays Unequal
* Difficulty: Hard
* 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:
long long minimumTotalCost(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Total Cost to Make Arrays Unequal
* Difficulty: Hard
* 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 long minimumTotalCost(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Total Cost to Make Arrays Unequal
Difficulty: Hard
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 minimumTotalCost(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Total Cost to Make Arrays Unequal
* Difficulty: Hard
* 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 minimumTotalCost = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Total Cost to Make Arrays Unequal
* Difficulty: Hard
* 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 minimumTotalCost(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Total Cost to Make Arrays Unequal
* Difficulty: Hard
* 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 long MinimumTotalCost(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Total Cost to Make Arrays Unequal
* Difficulty: Hard
* 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
*/

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

}
```

Go Solution:

```
// Problem: Minimum Total Cost to Make Arrays Unequal
// Difficulty: Hard
// 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 minimumTotalCost(nums1 []int, nums2 []int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Total Cost to Make Arrays Unequal
// Difficulty: Hard
// 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 minimum_total_cost(nums1: Vec<i32>, nums2: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
```

```
# @return {Integer}
def minimum_total_cost(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

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