Problem 2449: Minimum Number of Operations to Make Arrays Similar

Problem Information

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

Problem Description

You are given two positive integer arrays

nums

and

target

, of the same length.

In one operation, you can choose any two

distinct

indices

i

and

j

where

0 <= i, j < nums.length

and:

set

nums[i] = nums[i] + 2

and

set

nums[j] = nums[j] - 2

.

Two arrays are considered to be

similar

if the frequency of each element is the same.

Return

the minimum number of operations required to make

nums

similar to

target

. The test cases are generated such that

nums

can always be similar to

target

.

Example 1:

Input:

nums = [8,12,6], target = [2,14,10]

Output:

2

Explanation:

It is possible to make nums similar to target in two operations: - Choose i = 0 and j = 2, nums = [10,12,4]. - Choose i = 1 and j = 2, nums = [10,14,2]. It can be shown that 2 is the minimum number of operations needed.

Example 2:

Input:

nums = [1,2,5], target = [4,1,3]

Output:

1

Explanation:

We can make nums similar to target in one operation: - Choose i = 1 and j = 2, nums = [1,4,3].

Example 3:

Input:

nums = [1,1,1,1,1], target = [1,1,1,1,1]

Output:

0

Explanation:

The array nums is already similiar to target.

Constraints:

n == nums.length == target.length

1 <= n <= 10

5

1 <= nums[i], target[i] <= 10

6

It is possible to make

nums

similar to

target

.

Code Snippets

C++:

```
class Solution {
public:
long long makeSimilar(vector<int>& nums, vector<int>& target) {

}
};
```

Java:

```
class Solution {
public long makeSimilar(int[] nums, int[] target) {

}
}
```

Python3:

```
class Solution:
def makeSimilar(self, nums: List[int], target: List[int]) -> int:
```

Python:

```
class Solution(object):
def makeSimilar(self, nums, target):
"""
:type nums: List[int]
:type target: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number[]} target
* @return {number}
*/
var makeSimilar = function(nums, target) {

};
```

TypeScript:

```
function makeSimilar(nums: number[], target: number[]): number {

};
```

C#:

```
public class Solution {
public long MakeSimilar(int[] nums, int[] target) {

}
}
```

C:

```
long long makeSimilar(int* nums, int numsSize, int* target, int targetSize) {

}
```

Go:

```
func makeSimilar(nums []int, target []int) int64 {

}
```

Kotlin:

```
class Solution {
fun makeSimilar(nums: IntArray, target: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func makeSimilar(_ nums: [Int], _ target: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn make_similar(nums: Vec<i32>, target: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} target
# @return {Integer}
def make_similar(nums, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $target
* @return Integer
*/
function makeSimilar($nums, $target) {

}
}
```

Dart:

```
class Solution {
int makeSimilar(List<int> nums, List<int> target) {

}
}
```

Scala:

```
object Solution {
def makeSimilar(nums: Array[Int], target: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec make_similar(nums :: [integer], target :: [integer]) :: integer
def make_similar(nums, target) do
```

```
end
end
```

Erlang:

```
-spec make_similar(Nums :: [integer()], Target :: [integer()]) -> integer().
make_similar(Nums, Target) ->
.
```

Racket:

```
(define/contract (make-similar nums target)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Operations to Make Arrays Similar
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 makeSimilar(vector<int>& nums, vector<int>& target) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Operations to Make Arrays Similar
```

```
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 makeSimilar(int[] nums, int[] target) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Operations to Make Arrays Similar
Difficulty: Hard
Tags: array, greedy, sort

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 makeSimilar(self, nums: List[int], target: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def makeSimilar(self, nums, target):
"""
:type nums: List[int]
:type target: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Operations to Make Arrays Similar
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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[]} nums
* @param {number[]} target
* @return {number}
*/
var makeSimilar = function(nums, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Operations to Make Arrays Similar
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 makeSimilar(nums: number[], target: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Operations to Make Arrays Similar
* Difficulty: Hard
* Tags: array, greedy, sort
```

```
*
* 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 MakeSimilar(int[] nums, int[] target) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Operations to Make Arrays Similar
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 makeSimilar(int* nums, int numsSize, int* target, int targetSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of Operations to Make Arrays Similar
// Difficulty: Hard
// Tags: array, greedy, sort
//
// 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 makeSimilar(nums []int, target []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun makeSimilar(nums: IntArray, target: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func makeSimilar(_ nums: [Int], _ target: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Operations to Make Arrays Similar
// Difficulty: Hard
// Tags: array, greedy, sort
//
// 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 make_similar(nums: Vec<i32>, target: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} target
# @return {Integer}
def make_similar(nums, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $target
* @return Integer
*/
function makeSimilar($nums, $target) {

}
}
```

Dart Solution:

```
class Solution {
int makeSimilar(List<int> nums, List<int> target) {

}
}
```

Scala Solution:

```
object Solution {
def makeSimilar(nums: Array[Int], target: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec make_similar(nums :: [integer], target :: [integer]) :: integer
def make_similar(nums, target) do

end
end
```

Erlang Solution:

```
-spec make_similar(Nums :: [integer()], Target :: [integer()]) -> integer().
make_similar(Nums, Target) ->
.
```

Racket Solution:

```
(define/contract (make-similar nums target)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```