Problem 2448: Minimum Cost to Make Array Equal

Problem Information

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

Problem Description

You are given two

0-indexed

arrays

nums

and

cost

consisting each of

n

positive

integers.

You can do the following operation

any

number of times:

Increase or decrease

any

element of the array

nums

by

1

.

The cost of doing one operation on the

i

th

element is

cost[i]

.

Return

the

minimum

total cost such that all the elements of the array

nums

become

equal

.

Example 1:

Input:

nums = [1,3,5,2], cost = [2,3,1,14]

Output:

8

Explanation:

We can make all the elements equal to 2 in the following way: - Increase the 0

th

element one time. The cost is 2. - Decrease the 1

st

element one time. The cost is 3. - Decrease the 2

nd

element three times. The cost is 1 + 1 + 1 = 3. The total cost is 2 + 3 + 3 = 8. It can be shown that we cannot make the array equal with a smaller cost.

Example 2:

Input:

nums = [2,2,2,2,2], cost = [4,2,8,1,3]

Output:

0

Explanation:

All the elements are already equal, so no operations are needed.

Constraints:

n == nums.length == cost.length

1 <= n <= 10

5

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

6

Test cases are generated in a way that the output doesn't exceed 2

53

-1

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long minCost(int[] nums, int[] cost) {
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minCost(nums: number[], cost: number[]): number {

};
```

C#:

```
public class Solution {
public long MinCost(int[] nums, int[] cost) {

}
}
```

C:

```
long long minCost(int* nums, int numsSize, int* cost, int costSize) {

}
```

Go:

```
func minCost(nums []int, cost []int) int64 {

}
```

Kotlin:

```
class Solution {
fun minCost(nums: IntArray, cost: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} cost
# @return {Integer}
def min_cost(nums, cost)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $cost
* @return Integer
*/
function minCost($nums, $cost) {

}
}
```

Dart:

```
class Solution {
int minCost(List<int> nums, List<int> cost) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cost(nums :: [integer], cost :: [integer]) :: integer
def min_cost(nums, cost) do

end
```

```
end
```

Erlang:

```
-spec min_cost(Nums :: [integer()], Cost :: [integer()]) -> integer().
min_cost(Nums, Cost) ->
.
```

Racket:

```
(define/contract (min-cost nums cost)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Make Array Equal
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
* 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 minCost(vector<int>& nums, vector<int>& cost) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Make Array Equal
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
* 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 minCost(int[] nums, int[] cost) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Make Array Equal
Difficulty: Hard
Tags: array, greedy, sort, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Make Array Equal
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
* 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[]} cost
* @return {number}
*/
var minCost = function(nums, cost) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Make Array Equal
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
* 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 minCost(nums: number[], cost: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Make Array Equal
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
```

```
* 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 MinCost(int[] nums, int[] cost) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Make Array Equal
* Difficulty: Hard
* Tags: array, greedy, sort, search
*
* 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 minCost(int* nums, int numsSize, int* cost, int costSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost to Make Array Equal
// Difficulty: Hard
// Tags: array, greedy, sort, search
//
// 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 minCost(nums []int, cost []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minCost(nums: IntArray, cost: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Make Array Equal
// Difficulty: Hard
// Tags: array, greedy, sort, search
//
// 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_cost(nums: Vec<i32>, cost: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} cost
# @return {Integer}
def min_cost(nums, cost)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $cost
* @return Integer
*/
function minCost($nums, $cost) {

}
}
```

Dart Solution:

```
class Solution {
int minCost(List<int> nums, List<int> cost) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost(nums :: [integer], cost :: [integer]) :: integer
def min_cost(nums, cost) do

end
end
```

Erlang Solution:

```
-spec min_cost(Nums :: [integer()], Cost :: [integer()]) -> integer().
min_cost(Nums, Cost) ->
.
```

Racket Solution:

```
(define/contract (min-cost nums cost)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```