Problem 2541: Minimum Operations to Make Array Equal II

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

and an integer

k

. You can perform the following operation on

nums1

:

Choose two indexes

i

and

j

and increment

nums1[i]

by

k

and decrement

nums1[j]

by

k

. In other words,

nums1[i] = nums1[i] + k

and

nums1[j] = nums1[j] - k

.

nums1

is said to be

equal

to

nums2

if for all indices

i

such that

0 <= i < n

,

nums1[i] == nums2[i]

.

Return

the

minimum

number of operations required to make

nums1

equal to

nums2

. If it is impossible to make them equal, return

-1

.

Example 1:

Input:

nums1 = [4,3,1,4], nums2 = [1,3,7,1], k = 3

Output:

2

Explanation:

In 2 operations, we can transform nums1 to nums2. 1

st

operation: i = 2, j = 0. After applying the operation, nums1 = [1,3,4,4]. 2

nd

operation: i = 2, j = 3. After applying the operation, nums1 = [1,3,7,1]. One can prove that it is impossible to make arrays equal in fewer operations.

Example 2:

Input:

nums1 = [3,8,5,2], nums2 = [2,4,1,6], k = 1

Output:

-1

Explanation:

It can be proved that it is impossible to make the two arrays equal.

Constraints:

n == nums1.length == nums2.length

2 <= n <= 10

5

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

9

0 <= k <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:type nums2: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

```
}
```

Kotlin:

```
class Solution {
fun minOperations(nums1: IntArray, nums2: IntArray, k: Int): Long {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make Array Equal II
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 minOperations(vector<int>& nums1, vector<int>& nums2, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make Array Equal II
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 minOperations(int[] nums1, int[] nums2, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make Array Equal II
Difficulty: Medium
Tags: array, greedy, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make Array Equal II
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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
* @param {number} k
* @return {number}
*/
var minOperations = function(nums1, nums2, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make Array Equal II
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 minOperations(nums1: number[], nums2: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make Array Equal II
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 MinOperations(int[] nums1, int[] nums2, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make Array Equal II
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 minOperations(int* nums1, int nums1Size, int* nums2, int nums2Size,
int k) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make Array Equal II
// Difficulty: Medium
// Tags: array, greedy, math
//
// 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 minOperations(nums1 []int, nums2 []int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(nums1: IntArray, nums2: IntArray, k: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make Array Equal II
// Difficulty: Medium
// Tags: array, greedy, math
//
// 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_operations(nums1: Vec<i32>, nums2: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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