

# 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

$\text{nums1}[i]$

by

k

and decrement

$\text{nums1}[j]$

by

k

. In other words,

$$\text{nums1}[i] = \text{nums1}[i] + k$$

and

$$\text{nums1}[j] = \text{nums1}[j] - k$$

$\text{nums1}$

is said to be

equal

to

nums2

if for all indices

i

such that

$0 \leq i < n$

,

$\text{nums1}[i] == \text{nums2}[i]$

.

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:

$n == \text{nums1.length} == \text{nums2.length}$

$2 \leq n \leq 10$

5

$0 \leq \text{nums1}[i], \text{nums2}[j] \leq 10$

9

$0 \leq k \leq 10$

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 {  
        if (k < 0) return -1  
        if (k == 0) return 0  
        var count = 0  
        for (i in 0..nums1.size - 1) {  
            if (nums1[i] != nums2[i]) {  
                if (k >= 2) {  
                    if ((nums1[i] - nums2[i]).abs() % 2 != 0) return -1  
                    k -= 2  
                } else return -1  
                count++  
            }  
        }  
        if (k > 0) return -1  
        return count  
    }  
}
```

### Swift:

```
class Solution {  
    func minOperations(_ nums1: [Int], _ nums2: [Int], _ k: Int) -> Int {  
        if k < 0 {  
            return -1  
        }  
        if k == 0 {  
            return 0  
        }  
        var count = 0  
        for i in 0..            if nums1[i] != nums2[i] {  
                if k >= 2 {  
                    if (nums1[i] - nums2[i]).abs() % 2 != 0 {  
                        return -1  
                    }  
                    k -= 2  
                } else {  
                    return -1  
                }  
                count += 1  
            }  
        }  
        if k > 0 {  
            return -1  
        }  
        return count  
    }  
}
```

### Rust:

```
impl Solution {  
    pub fn min_operations(nums1: Vec<i32>, nums2: Vec<i32>, k: i32) -> i64 {  
        if k < 0 {  
            return -1  
        }  
        if k == 0 {  
            return 0  
        }  
        let mut count = 0;  
        for i in 0..nums1.len() {  
            if nums1[i] != nums2[i] {  
                if k >= 2 {  
                    if (nums1[i] - nums2[i]).abs() % 2 != 0 {  
                        return -1  
                    }  
                    k -= 2  
                } else {  
                    return -1  
                }  
                count += 1  
            }  
        }  
        if k > 0 {  
            return -1  
        }  
        return count  
    }  
}
```

### Ruby:

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

### PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums1  
     */  
    public function minOperations($nums1, $nums2, $k) {  
        if ($k < 0) {  
            return -1;  
        }  
        if ($k == 0) {  
            return 0;  
        }  
        $count = 0;  
        for ($i = 0; $i < count($nums1); $i++) {  
            if ($nums1[$i] != $nums2[$i]) {  
                if ($k >= 2) {  
                    if ((int)$nums1[$i] - (int)$nums2[$i] % 2 != 0) {  
                        return -1;  
                    }  
                    $k -= 2;  
                } else {  
                    return -1;  
                }  
                $count++;  
            }  
        }  
        if ($k > 0) {  
            return -1;  
        }  
        return $count;  
    }  
}
```

```

* @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 {  
        if (k == 0) return 0  
        if (nums1.size != nums2.size) return -1  
        var count = 0  
        var diffSum = 0  
        for (i in 0..nums1.size - 1) {  
            val diff = nums1[i] - nums2[i]  
            if (diff % k != 0) return -1  
            if (diff < 0) diffSum += -diff  
            else diffSum += diff  
            count += abs(diff) / k  
        }  
        if (diffSum % k != 0) return -1  
        return count  
    }  
}
```

### Swift Solution:

```
class Solution {  
    func minOperations(_ nums1: [Int], _ nums2: [Int], _ k: Int) -> Int {  
        if k == 0 {  
            return 0  
        }  
        if nums1.count != nums2.count {  
            return -1  
        }  
        var count = 0  
        var diffSum = 0  
        for i in 0..            let diff = nums1[i] - nums2[i]  
            if diff % k != 0 {  
                return -1  
            }  
            if diff < 0 {  
                diffSum += -diff  
            } else {  
                diffSum += diff  
            }  
            count += abs(diff) / k  
        }  
        if diffSum % k != 0 {  
            return -1  
        }  
        return count  
    }  
}
```

### 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 {  
        if k == 0 {  
            return 0  
        }  
        if nums1.len() != nums2.len() {  
            return -1  
        }  
        let mut count = 0;  
        let mut diffSum = 0;  
        for i in 0..nums1.len() {  
            let diff = nums1[i] - nums2[i];  
            if diff % k != 0 {  
                return -1  
            }  
            if diff < 0 {  
                diffSum += -diff  
            } else {  
                diffSum += diff  
            }  
            count += abs(diff) / k  
        }  
        if diffSum % k != 0 {  
            return -1  
        }  
        return count  
    }  
}
```

### Ruby Solution:

```
# @param {Integer[]} nums1  
# @param {Integer[]} nums2  
# @param {Integer} k  
# @return {Integer}  
def min_operations(nums1, nums2, k)  
    if k == 0  
        return 0  
    end  
    if nums1.length != nums2.length  
        return -1  
    end  
    count = 0  
    diffSum = 0  
    for i in 0..nums1.length - 1  
        diff = nums1[i] - nums2[i]  
        if diff % k != 0  
            return -1  
        end  
        if diff < 0  
            diffSum += -diff  
        else  
            diffSum += diff  
        end  
        count += abs(diff) / k  
    end  
    if diffSum % k != 0  
        return -1  
    end  
    return count  
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) {
        return 0;
    }
}

```

### Scala Solution:

```

object Solution {
    def minOperations(nums1: Array[Int], nums2: Array[Int], k: Int): Long = {
        if (k == 0) 0
        else if (k < 0) -1
        else if (nums1.length == 0 || nums2.length == 0) -1
        else {
            val sum = (nums1.sum + nums2.sum) / k
            if (sum % k != 0) -1
            else {
                val count = 0
                for (i <- 0 to nums1.length - 1) {
                    if ((nums1(i) - sum) % k != 0) count += 1
                }
                for (i <- 0 to nums2.length - 1) {
                    if ((nums2(i) - sum) % k != 0) count += 1
                }
                if (count >= k) -1
                else count
            }
        }
    }
}

```

### Elixir Solution:

```

defmodule Solution do
  @spec min_operations(list :: [integer], list :: [integer], integer :: integer) :: integer
  def min_operations(list1, list2, k) do
    if k == 0 do
      0
    else
      if k < 0 do
        -1
      else
        if length(list1) == 0 or length(list2) == 0 do
          -1
        else
          sum = Enum.sum(list1) + Enum.sum(list2)
          if rem(sum, k) != 0 do
            -1
          else
            count = 0
            for i in 0..length(list1)-1 do
              if rem(list1[i] - sum, k) != 0 do
                count = count + 1
              end
            end
            for i in 0..length(list2)-1 do
              if rem(list2[i] - sum, k) != 0 do
                count = count + 1
              end
            end
            if count >= k do
              -1
            else
              count
            end
          end
        end
      end
    end
  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?))
```