Problem 3724: Minimum Operations to Transform Array

Problem Information

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

Problem Description

You are given two integer arrays

nums1

of length

n

and

nums2

of length

n + 1

.

You want to transform

nums1

into

nums2

using the

minimum

number of operations.

You may perform the following operations

any

number of times, each time choosing an index

i

:

Increase

nums1[i]

by 1.

Decrease

nums1[i]

by 1.

Append

nums1[i]

to the

end

of the array.

Return the

minimum

number of operations required to transform

nums1

into

nums2

.

Example 1:

Input:

nums1 = [2,8], nums2 = [1,7,3]

Output:

4

Explanation:

Step

i

Operation

nums1[i]

Updated

nums1

1

0

Append

-

[2, 8, 2]

2

0

Decrement

Decreases to 1

[1, 8, 2]

3

1

Decrement

Decreases to 7

[1, 7, 2]

4

2

Increment

Increases to 3

[1, 7, 3]

Thus, after 4 operations

nums1

is transformed into

nums2

.

Example 2:

Input:

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

Output:

4

Explanation:

Step

i

Operation

nums1[i]

Updated

nums1

1

1

Append

-

[1, 3, 6, 3]

2

0

Increment

Increases to 2

[2, 3, 6, 3]

3

1

Increment

Increases to 4

[2, 4, 6, 3]

4

2

Decrement

Decreases to 5

[2, 4, 5, 3]

Thus, after 4 operations

nums1

is transformed into

nums2

.

Example 3:

Input:

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

Output:

3

Explanation:

Step

i

Operation

nums1[i]

Updated

nums1

1

0

Increment

Increases to 3

[3]

2

0

Append

-

[3, 3]

3

1

Increment

Increases to 4

[3, 4]

Thus, after 3 operations

nums1

is transformed into

nums2

.

Constraints:

1 <= n == nums1.length <= 10

5

nums2.length == n + 1

1 <= nums1[i], nums2[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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



}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Transform Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Transform Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Transform Array
Difficulty: Medium
Tags: array, greedy

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]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Transform Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Transform Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Transform Array
* Difficulty: Medium
* Tags: array, greedy
*
* 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)
{

}
```

Go Solution:

```
// Problem: Minimum Operations to Transform Array
// Difficulty: Medium
// Tags: array, greedy
//
// 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) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Transform Array
// Difficulty: Medium
// Tags: array, greedy
//
// 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>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}

}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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