Problem 3229: Minimum Operations to Make Array Equal to Target

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 a single operation, you can select any subarray of

nums

and increment each element within that subarray by 1 or decrement each element within that subarray by 1.

Return the

minimum

number of operations required to make

nums

equal to the array

target

.

Example 1:

Input:

nums = [3,5,1,2], target = [4,6,2,4]

Output:

2

Explanation:

We will perform the following operations to make

nums

equal to

target

:

- Increment

nums[0..3]

by 1,

nums = [4,6,2,3]

.

- Increment

nums[3..3]

by 1,

nums = [4,6,2,4]

.

Example 2:

Input:

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

Output:

5

Explanation:

We will perform the following operations to make

nums

equal to

target

:

- Increment

nums[0..0]

by 1,

nums = [2,3,2]

.

- Decrement

nums[1..1]

by 1,

nums = [2,2,2]

.

- Decrement

nums[1..1]

by 1,

nums = [2,1,2]

.

- Increment

nums[2..2]

by 1,

nums = [2,1,3]

.

- Increment

nums[2..2]

by 1,

nums = [2,1,4]

.

Constraints:

1 <= nums.length == target.length <= 10

5

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

8

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def minimumOperations(self, nums, target):
```

```
"""
:type nums: List[int]
:type target: List[int]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $target
```

```
* @return Integer
*/
function minimumOperations($nums, $target) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-operations nums target)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make Array Equal to Target
* Difficulty: Hard
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make Array Equal to Target
* Difficulty: Hard
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make Array Equal to Target
Difficulty: Hard
Tags: array, dp, greedy, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make Array Equal to Target
* Difficulty: Hard
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make Array Equal to Target
* Difficulty: Hard
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumOperations(nums: number[], target: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make Array Equal to Target
* Difficulty: Hard
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make Array Equal to Target
* Difficulty: Hard
* Tags: array, dp, greedy, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

long long minimumOperations(int* nums, int numsSize, int* target, int
targetSize) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make Array Equal to Target
// Difficulty: Hard
// Tags: array, dp, greedy, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimumOperations(nums []int, target []int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make Array Equal to Target
// Difficulty: Hard
// Tags: array, dp, greedy, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $target
* @return Integer
*/
```

```
function minimumOperations($nums, $target) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-operations nums target)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
```

```
)
```