Problem 2602: Minimum Operations to Make All Array Elements Equal

Problem Information

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

Problem Description

You are given an array

nums

consisting of positive integers.

You are also given an integer array

queries

of size

m

. For the

i

th

query, you want to make all of the elements of

nums

equal to

queries[i]

. You can perform the following operation on the array

any

number of times:

Increase

or

decrease

an element of the array by

1

.

Return

an array

answer

of size

m

where

answer[i]

is the

minimum

number of operations to make all elements of

nums

equal to

queries[i]

.

Note

that after each query the array is reset to its original state.

Example 1:

Input:

nums = [3,1,6,8], queries = [1,5]

Output:

[14,10]

Explanation:

For the first query we can do the following operations: - Decrease nums[0] 2 times, so that nums = [1,1,6,8]. - Decrease nums[2] 5 times, so that nums = [1,1,1,8]. - Decrease nums[3] 7 times, so that nums = [1,1,1,1]. So the total number of operations for the first query is 2 + 5 + 7 = 14. For the second query we can do the following operations: - Increase nums[0] 2 times, so that nums = [5,1,6,8]. - Increase nums[1] 4 times, so that nums = [5,5,6,8]. - Decrease nums[2] 1 time, so that nums = [5,5,5,8]. - Decrease nums[3] 3 times, so that nums = [5,5,5,5]. So the total number of operations for the second query is 2 + 4 + 1 + 3 = 10.

Example 2:

Input:

nums = [2,9,6,3], queries = [10]

Output:

[20]

Explanation:

We can increase each value in the array to 10. The total number of operations will be 8 + 1 + 4 + 7 = 20.

Constraints:

n == nums.length

m == queries.length

1 <= n, m <= 10

5

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

9

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Long> minOperations(int[] nums, int[] queries) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minOperations(nums: number[], queries: number[]): number[] {

};
```

C#:

```
public class Solution {
public IList<long> MinOperations(int[] nums, int[] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* minOperations(int* nums, int numsSize, int* queries, int
queriesSize, int* returnSize){

}
```

Go:

```
func minOperations(nums []int, queries []int) []int64 {

}
```

Kotlin:

```
class Solution {
fun minOperations(nums: IntArray, queries: IntArray): List<Long> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} queries
# @return {Integer[]}
def min_operations(nums, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $queries
* @return Integer[]
*/
function minOperations($nums, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> minOperations(List<int> nums, List<int> queries) {

}
}
```

Scala:

```
object Solution {
def minOperations(nums: Array[Int], queries: Array[Int]): List[Long] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations(nums :: [integer], queries :: [integer]) :: [integer]
def min_operations(nums, queries) do
```

```
end
end
```

Erlang:

```
-spec min_operations(Nums :: [integer()], Queries :: [integer()]) ->
[integer()].
min_operations(Nums, Queries) ->
.
```

Racket:

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

)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make All Array Elements Equal
* Difficulty: Medium
* Tags: array, 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:
vector<long long> minOperations(vector<int>& nums, vector<int>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make All Array Elements Equal
* Difficulty: Medium
* Tags: array, 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 List<Long> minOperations(int[] nums, int[] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make All Array Elements Equal
Difficulty: Medium
Tags: array, 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 minOperations(self, nums: List[int], queries: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make All Array Elements Equal
* Difficulty: Medium
* Tags: array, 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[]} queries
* @return {number[]}
*/
var minOperations = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make All Array Elements Equal
* Difficulty: Medium
* Tags: array, 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 minOperations(nums: number[], queries: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make All Array Elements Equal
* Difficulty: Medium
```

```
* Tags: array, 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 IList<long> MinOperations(int[] nums, int[] queries) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make All Array Elements Equal
* Difficulty: Medium
* Tags: array, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* minOperations(int* nums, int numsSize, int* queries, int
queriesSize, int* returnSize){

}
```

Go Solution:

```
// Problem: Minimum Operations to Make All Array Elements Equal
// Difficulty: Medium
// Tags: array, 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 minOperations(nums []int, queries []int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(nums: IntArray, queries: IntArray): List<Long> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make All Array Elements Equal
// Difficulty: Medium
// Tags: array, 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_operations(nums: Vec<i32>, queries: Vec<i32>) -> Vec<i64> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} queries
# @return {Integer[]}
def min_operations(nums, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $queries
* @return Integer[]
*/
function minOperations($nums, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> minOperations(List<int> nums, List<int> queries) {

}
}
```

Scala Solution:

```
object Solution {
def minOperations(nums: Array[Int], queries: Array[Int]): List[Long] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations(nums :: [integer], queries :: [integer]) :: [integer]
def min_operations(nums, queries) do
```

```
end
end
```

Erlang Solution:

```
-spec min_operations(Nums :: [integer()], Queries :: [integer()]) ->
[integer()].
min_operations(Nums, Queries) ->
.
```

Racket Solution:

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

)
```