Problem 1887: Reduction Operations to Make the Array Elements Equal

Problem Information

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

Problem Description

Given an integer array

nums

, your goal is to make all elements in

nums

equal. To complete one operation, follow these steps:

Find the

largest

value in

nums

. Let its index be

i

(

0-indexed

) and its value be

largest

. If there are multiple elements with the largest value, pick the smallest

i

.

Find the

next largest

value in

nums

strictly smaller

than

largest

. Let its value be

nextLargest

.

Reduce

nums[i]

to

nextLargest

.

Return

the number of operations to make all elements in

nums

equal

.

Example 1:

Input:

nums = [5,1,3]

Output:

3

Explanation:

It takes 3 operations to make all elements in nums equal: 1. largest = 5 at index 0. nextLargest = 3. Reduce nums[0] to 3. nums = [

3

,1,3]. 2. largest = 3 at index 0. nextLargest = 1. Reduce nums[0] to 1. nums = [

1

,1,3]. 3. largest = 3 at index 2. nextLargest = 1. Reduce nums[2] to 1. nums = [1,1,

1

].

Example 2:

Input:

nums = [1,1,1]

Output:

0

Explanation:

All elements in nums are already equal.

Example 3:

Input:

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

Output:

4

Explanation:

It takes 4 operations to make all elements in nums equal: 1. largest = 3 at index 4. nextLargest = 2. Reduce nums[4] to 2. nums = [1,1,2,2,

2

]. 2. largest = 2 at index 2. nextLargest = 1. Reduce nums[2] to 1. nums = [1,1,

1

,2,2]. 3. largest = 2 at index 3. nextLargest = 1. Reduce nums[3] to 1. nums = [1,1,1,

1

,2]. 4. largest = 2 at index 4. nextLargest = 1. Reduce nums[4] to 1. nums = [1,1,1,1,

1

].

Constraints:

1 <= nums.length <= 5 * 10

4

1 <= nums[i] <= 5 * 10

4

Code Snippets

C++:

```
class Solution {
public:
int reductionOperations(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int reductionOperations(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function reductionOperations(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int ReductionOperations(int[] nums) {

}
}
```

C:

```
int reductionOperations(int* nums, int numsSize) {

}
```

Go:

```
func reductionOperations(nums []int) int {
```

```
}
```

Kotlin:

```
class Solution {
fun reductionOperations(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func reductionOperations(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn reduction_operations(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def reduction_operations(nums)

end
```

PHP:

```
class Solution {

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

```
function reductionOperations($nums) {

}
}
```

Dart:

```
class Solution {
int reductionOperations(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def reductionOperations(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec reduction_operations(nums :: [integer]) :: integer
def reduction_operations(nums) do

end
end
```

Erlang:

```
-spec reduction_operations(Nums :: [integer()]) -> integer().
reduction_operations(Nums) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Reduction Operations to Make the Array Elements Equal
* Difficulty: Medium
* Tags: array, sort
*
* 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:
int reductionOperations(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Reduction Operations to Make the Array Elements Equal
* Difficulty: Medium
* Tags: array, sort
*
* 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 int reductionOperations(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Reduction Operations to Make the Array Elements Equal
Difficulty: Medium
Tags: array, sort

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 reductionOperations(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Reduction Operations to Make the Array Elements Equal
* Difficulty: Medium
* Tags: array, sort
*
* 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
* @return {number}
*/
var reductionOperations = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Reduction Operations to Make the Array Elements Equal
* Difficulty: Medium
* Tags: array, sort
*
* 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 reductionOperations(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Reduction Operations to Make the Array Elements Equal
* Difficulty: Medium
* Tags: array, sort
*
* 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 int ReductionOperations(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Reduction Operations to Make the Array Elements Equal
* Difficulty: Medium
```

```
* Tags: array, sort
*
* 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
*/

int reductionOperations(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Reduction Operations to Make the Array Elements Equal
// Difficulty: Medium
// Tags: array, sort
//
// 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 reductionOperations(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun reductionOperations(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func reductionOperations(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Reduction Operations to Make the Array Elements Equal
// Difficulty: Medium
// Tags: array, sort
//
// 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 reduction_operations(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def reduction_operations(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function reductionOperations($nums) {

}
}
```

Dart Solution:

```
class Solution {
int reductionOperations(List<int> nums) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def reductionOperations(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec reduction_operations(nums :: [integer]) :: integer
def reduction_operations(nums) do

end
end
```

Erlang Solution:

```
-spec reduction_operations(Nums :: [integer()]) -> integer().
reduction_operations(Nums) ->
.
```

Racket Solution:

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