Problem 3542: Minimum Operations to Convert All Elements to Zero

Problem Information

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

Problem Description

You are given an array

nums

of size

n

, consisting of

non-negative

integers. Your task is to apply some (possibly zero) operations on the array so that

all

elements become 0.

In one operation, you can select a

subarray

[i, j]

(where

0 <= i <= j < n

) and set all occurrences of the

minimum

non-negative

integer in that subarray to 0.

Return the

minimum

number of operations required to make all elements in the array 0.

Example 1:

Input:

nums = [0,2]

Output:

1

Explanation:

Select the subarray

[1,1]

(which is

[2]

), where the minimum non-negative integer is 2. Setting all occurrences of 2 to 0 results in

[0,0]

.

Thus, the minimum number of operations required is 1.

Example 2:

Input:

nums = [3,1,2,1]

Output:

3

Explanation:

Select subarray

[1,3]

(which is

[1,2,1]

), where the minimum non-negative integer is 1. Setting all occurrences of 1 to 0 results in

[3,0,2,0]

.

Select subarray

[2,2]

(which is

[2]

), where the minimum non-negative integer is 2. Setting all occurrences of 2 to 0 results in

[3,0,0,0]

.

Select subarray

[0,0]

(which is

[3]

), where the minimum non-negative integer is 3. Setting all occurrences of 3 to 0 results in

[0,0,0,0]

.

Thus, the minimum number of operations required is 3.

Example 3:

Input:

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

Output:

4

Explanation:

Select subarray

[0,5]

(which is

[1,2,1,2,1,2]

), where the minimum non-negative integer is 1. Setting all occurrences of 1 to 0 results in

[0,2,0,2,0,2]

.

Select subarray

[1,1]

(which is

[2]

), where the minimum non-negative integer is 2. Setting all occurrences of 2 to 0 results in

[0,0,0,2,0,2]

.

Select subarray

[3,3]

(which is

[2]

), where the minimum non-negative integer is 2. Setting all occurrences of 2 to 0 results in

[0,0,0,0,0,2]

.

Select subarray

[5,5]

(which is

[2]

), where the minimum non-negative integer is 2. Setting all occurrences of 2 to 0 results in

[0,0,0,0,0,0]

.

Thus, the minimum number of operations required is 4.

Constraints:

1 <= n == nums.length <= 10

5

0 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Convert All Elements to Zero
* Difficulty: Medium
* Tags: array, greedy, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Convert All Elements to Zero
* Difficulty: Medium
* Tags: array, greedy, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

```
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Convert All Elements to Zero
Difficulty: Medium
Tags: array, greedy, hash, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Convert All Elements to Zero
* Difficulty: Medium
* Tags: array, greedy, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Convert All Elements to Zero
* Difficulty: Medium
* Tags: array, greedy, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function minOperations(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Convert All Elements to Zero
* Difficulty: Medium
* Tags: array, greedy, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MinOperations(int[] nums) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Convert All Elements to Zero
* Difficulty: Medium
* Tags: array, greedy, hash, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int minOperations(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Convert All Elements to Zero
// Difficulty: Medium
// Tags: array, greedy, hash, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func minOperations(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Convert All Elements to Zero
// Difficulty: Medium
// Tags: array, greedy, hash, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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