Problem 2826: Sorting Three Groups

Problem Information

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

Problem Description

You are given an integer array

nums

. Each element in

nums

is 1, 2 or 3. In each operation, you can remove an element from

nums

. Return the

minimum

number of operations to make

nums

non-decreasing

.

Example 1:

Input:

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

Output:

3

Explanation:

One of the optimal solutions is to remove

nums[0]

,

nums[2]

and

nums[3]

.

Example 2:

Input:

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

Output:

2

Explanation:

One of the optimal solutions is to remove

nums[1]

and

nums[2]

.

Example 3:

Input:

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

Output:

0

Explanation:

nums

is already non-decreasing.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 3

Follow-up:

Can you come up with an algorithm that runs in

O(n)

time complexity?

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int minimumOperations(List<Integer> nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int MinimumOperations(IList<int> nums) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun minimumOperations(nums: List<Int>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def minimumOperations(nums: List[Int]): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Sorting Three Groups
* Difficulty: Medium
* Tags: array, dp, search
*
* 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:
int minimumOperations(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sorting Three Groups
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 int minimumOperations(List<Integer> nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sorting Three Groups
Difficulty: Medium
Tags: array, dp, search

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Sorting Three Groups
* Difficulty: Medium
* Tags: array, dp, search
*
* 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
* @return {number}
*/
var minimumOperations = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sorting Three Groups
* Difficulty: Medium
* Tags: array, dp, search
*
* 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[]): number {

};
```

C# Solution:

```
/*
* Problem: Sorting Three Groups
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 int MinimumOperations(IList<int> nums) {

}
}
```

C Solution:

```
/*
* Problem: Sorting Three Groups
* Difficulty: Medium
* Tags: array, dp, search
*
* 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
*/

int minimumOperations(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sorting Three Groups
// Difficulty: Medium
// Tags: array, dp, search
//
// 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) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumOperations(nums: List<Int>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sorting Three Groups
// Difficulty: Medium
// Tags: array, dp, search
//
// 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>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def minimumOperations(nums: List[Int]): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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