Problem 75: Sort Colors

Problem Information

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

Problem Description

Given an array

nums

with

n

objects colored red, white, or blue, sort them

in-place

so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers

0

,

1

, and

2

to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

Input:

nums = [2,0,2,1,1,0]

Output:

[0,0,1,1,2,2]

Example 2:

Input:

nums = [2,0,1]

Output:

[0,1,2]

Constraints:

n == nums.length

1 <= n <= 300

nums[i]

is either

0

,

1

, or

2

.

Follow up:

Could you come up with a one-pass algorithm using only constant extra space?

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public void sortColors(int[] nums) {

}
}
```

Python3:

```
class Solution:
def sortColors(self, nums: List[int]) -> None:
"""
Do not return anything, modify nums in-place instead.
"""
```

Python:

```
class Solution(object):
def sortColors(self, nums):
"""
:type nums: List[int]
:rtype: None Do not return anything, modify nums in-place instead.
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {void} Do not return anything, modify nums in-place instead.
*/
var sortColors = function(nums) {

};
```

TypeScript:

```
/**
Do not return anything, modify nums in-place instead.
*/
function sortColors(nums: number[]): void {

};
```

C#:

```
public class Solution {
public void SortColors(int[] nums) {

}
}
```

C:

```
void sortColors(int* nums, int numsSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun sortColors(nums: IntArray): Unit {

}
}
```

Swift:

```
class Solution {
func sortColors(_ nums: inout [Int]) {

}
}
```

Rust:

```
impl Solution {
pub fn sort_colors(nums: &mut Vec<i32>) {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Void} Do not return anything, modify nums in-place instead.
def sort_colors(nums)

end
```

PHP:

```
class Solution {

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

```
*/
function sortColors(&$nums) {

}
}
```

Dart:

```
class Solution {
void sortColors(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Sort Colors
* 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:
void sortColors(vector<int>& nums) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Sort Colors
* 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 void sortColors(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort Colors
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 sortColors(self, nums: List[int]) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def sortColors(self, nums):
"""
:type nums: List[int]
:rtype: None Do not return anything, modify nums in-place instead.
"""
```

JavaScript Solution:

```
/**
* Problem: Sort Colors
* 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 {void} Do not return anything, modify nums in-place instead.
*/
var sortColors = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort Colors
* 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
*/

/**
Do not return anything, modify nums in-place instead.
*/
```

```
function sortColors(nums: number[]): void {

};
```

C# Solution:

```
/*
* Problem: Sort Colors
* 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 void SortColors(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sort Colors
* 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
*/

void sortColors(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sort Colors
// 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 sortColors(nums []int) {

}
```

Kotlin Solution:

```
class Solution {
fun sortColors(nums: IntArray): Unit {

}
}
```

Swift Solution:

```
class Solution {
func sortColors(_ nums: inout [Int]) {

}
}
```

Rust Solution:

```
// Problem: Sort Colors
// 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 sort_colors(nums: &mut Vec<i32>) {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Void} Do not return anything, modify nums in-place instead.
def sort_colors(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return NULL
*/
function sortColors(&$nums) {

}
}
```

Dart Solution:

```
class Solution {
void sortColors(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```