Problem 80: Remove Duplicates from Sorted Array II

Problem Information

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

Problem Description

Given an integer array

nums

sorted in

non-decreasing order

, remove some duplicates

in-place

such that each unique element appears

at most twice

. The

relative order

of the elements should be kept the

same

.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the

first part

of the array

nums

. More formally, if there are

k

elements after removing the duplicates, then the first

k

elements of

nums

should hold the final result. It does not matter what you leave beyond the first

k

elements.

Return

k

after placing the final result in the first

k

slots of

nums

.

Do

not

allocate extra space for another array. You must do this by

modifying the input array

in-place

with O(1) extra memory.

Custom Judge:

The judge will test your solution with the following code:

int[] nums = [...]; // Input array int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length; for (int i = 0; i < k; i++) { assert nums[i] == expectedNums[i]; }

If all assertions pass, then your solution will be

accepted

.

Example 1:

Input:

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

Output:

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

Explanation:

Your function should return k = 5, with the first five elements of nums being 1, 1, 2, 2 and 3 respectively. It does not matter what you leave beyond the returned k (hence they are underscores).

Example 2:

Input:

nums = [0,0,1,1,1,1,2,3,3]

Output:

7, nums = [0,0,1,1,2,3,3,_,_]

Explanation:

Your function should return k = 7, with the first seven elements of nums being 0, 0, 1, 1, 2, 3 and 3 respectively. It does not matter what you leave beyond the returned k (hence they are underscores).

Constraints:

1 <= nums.length <= 3 * 10

4

-10

4

<= nums[i] <= 10

4

nums

is sorted in

non-decreasing

order.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Duplicates from Sorted Array II
* 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 removeDuplicates(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Remove Duplicates from Sorted Array II
* 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 removeDuplicates(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Duplicates from Sorted Array II
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 removeDuplicates(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove Duplicates from Sorted Array II
* 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 removeDuplicates = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove Duplicates from Sorted Array II
* 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 removeDuplicates(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Remove Duplicates from Sorted Array II
* 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 RemoveDuplicates(int[] nums) {
}
}
```

C Solution:

```c
/*
* Problem: Remove Duplicates from Sorted Array II
* 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 removeDuplicates(int* nums, int numsSize) {

}
```

Go Solution:

```go
// Problem: Remove Duplicates from Sorted Array II
// 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 removeDuplicates(nums []int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun removeDuplicates(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Remove Duplicates from Sorted Array II
// 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 remove_duplicates(nums: &mut Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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


}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```