Problem 27: Remove Element

Problem Information

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

Problem Description

Given an integer array

nums

and an integer

val

, remove all occurrences of

val

in

nums

in-place

. The order of the elements may be changed. Then return

the number of elements in

nums

which are not equal to

val

.

Consider the number of elements in

nums

which are not equal to

val

be

k

, to get accepted, you need to do the following things:

Change the array

nums

such that the first

k

elements of

nums

contain the elements which are not equal to

val

. The remaining elements of

nums

are not important as well as the size of nums

.

Return

k

.

Custom Judge:

The judge will test your solution with the following code:

int[] nums = [...]; // Input array int val = ...; // Value to remove int[] expectedNums = [...]; // The expected answer with correct length. // It is sorted with no values equaling val.

int k = removeElement(nums, val); // Calls your implementation

assert k == expectedNums.length; sort(nums, 0, k); // Sort the first k elements of nums for (int i = 0; i < actualLength; i++) { assert nums[i] == expectedNums[i]; }

If all assertions pass, then your solution will be

accepted

.

Example 1:

Input:

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

Output:

2, nums = [2,2,_,_]

Explanation:

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

Example 2:

Input:

nums = [0,1,2,2,3,0,4,2], val = 2

Output:

5, nums = [0,1,4,0,3,_,_,_]

Explanation:

Your function should return k = 5, with the first five elements of nums containing 0, 0, 1, 3, and 4. Note that the five elements can be returned in any order. It does not matter what you leave beyond the returned k (hence they are underscores).

Constraints:

0 <= nums.length <= 100

0 <= nums[i] <= 50

0 <= val <= 100

Code Snippets

C++:

```
class Solution {
public:
int removeElement(vector<int>& nums, int val) {

}
};
```

Java:

```
class Solution {
public int removeElement(int[] nums, int val) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function removeElement(nums: number[], val: number): number {

};
```

C#:

```
public class Solution {
public int RemoveElement(int[] nums, int val) {

}
}
```

C:

```
int removeElement(int* nums, int numsSize, int val) {

}
```

Go:

```
func removeElement(nums []int, val int) int {

}
```

Kotlin:

```
class Solution {
fun removeElement(nums: IntArray, `val`: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} val
# @return {Integer}
def remove_element(nums, val)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $val
* @return Integer
*/
function removeElement(&$nums, $val) {

}
}
```

Dart:

```
class Solution {
int removeElement(List<int> nums, int val) {

}
}
```

Scala:

```
object Solution {
def removeElement(nums: Array[Int], `val`: Int): Int = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Element
* Difficulty: Easy
* 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 removeElement(vector<int>& nums, int val) {

}
};
```

Java Solution:

```
/**
* Problem: Remove Element
* Difficulty: Easy
* 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 removeElement(int[] nums, int val) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Element
Difficulty: Easy
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 removeElement(self, nums: List[int], val: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove Element
* Difficulty: Easy
* 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
* @param {number} val
* @return {number}
*/
var removeElement = function(nums, val) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove Element
* Difficulty: Easy
* 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 removeElement(nums: number[], val: number): number {

};
```

C# Solution:

```
/*
* Problem: Remove Element
* Difficulty: Easy
* 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 RemoveElement(int[] nums, int val) {

}
}
```

C Solution:

```
/*
* Problem: Remove Element
* Difficulty: Easy
* 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 removeElement(int* nums, int numsSize, int val) {

}
```

Go Solution:

```
// Problem: Remove Element
// Difficulty: Easy
// 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 removeElement(nums []int, val int) int {

}
```

Kotlin Solution:

```
class Solution {
fun removeElement(nums: IntArray, `val`: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Remove Element
// Difficulty: Easy
// 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_element(nums: &mut Vec<i32>, val: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} val
# @return {Integer}
def remove_element(nums, val)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $val
* @return Integer
*/
function removeElement(&$nums, $val) {

}
}
```

Dart Solution:

```
class Solution {
int removeElement(List<int> nums, int val) {

}


}
```

Scala Solution:

```
object Solution {
def removeElement(nums: Array[Int], `val`: Int): Int = {

}
}
```