**27. Remove Element**

Given an integer array nums and an integer val, remove all occurrences of val in nums [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm). The relative order of the elements may be changed.

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**](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

**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).

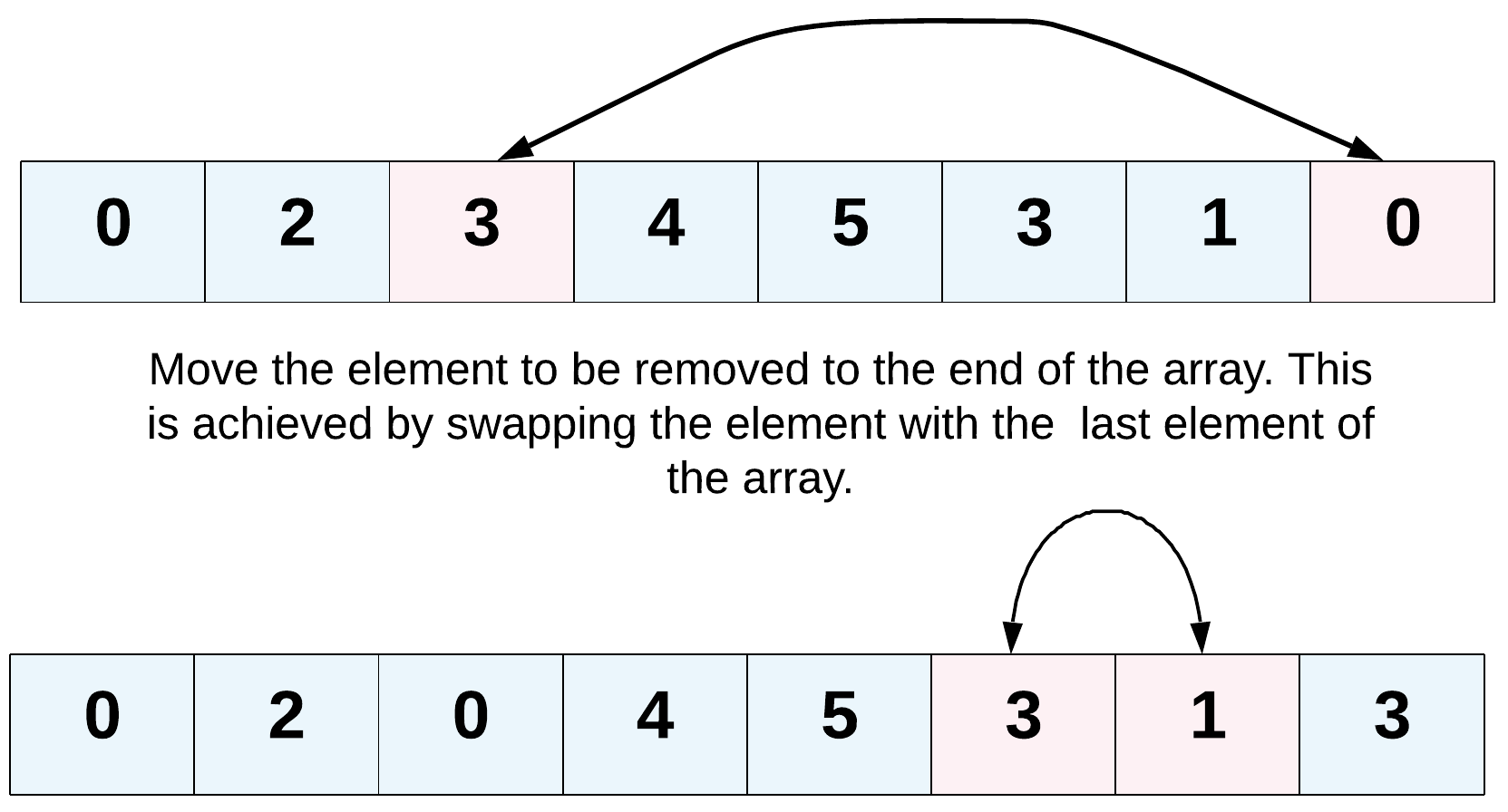
**Constraints:**

* 0 <= nums.length <= 100
* 0 <= nums[i] <= 50
* 0 <= val <= 100

Hide Hint 1

The problem statement clearly asks us to modify the array in-place and it also says that the element beyond the new length of the array can be anything. Given an element, we need to remove all the occurrences of it from the array. We don't technically need to **remove** that element per-say, right?

Hide Hint 2

We can move all the occurrences of this element to the end of the array. Use two pointers!  


Hide Hint 3

Yet another direction of thought is to consider the elements to be removed as non-existent. In a single pass, if we keep copying the visible elements in-place, that should also solve this problem for us.