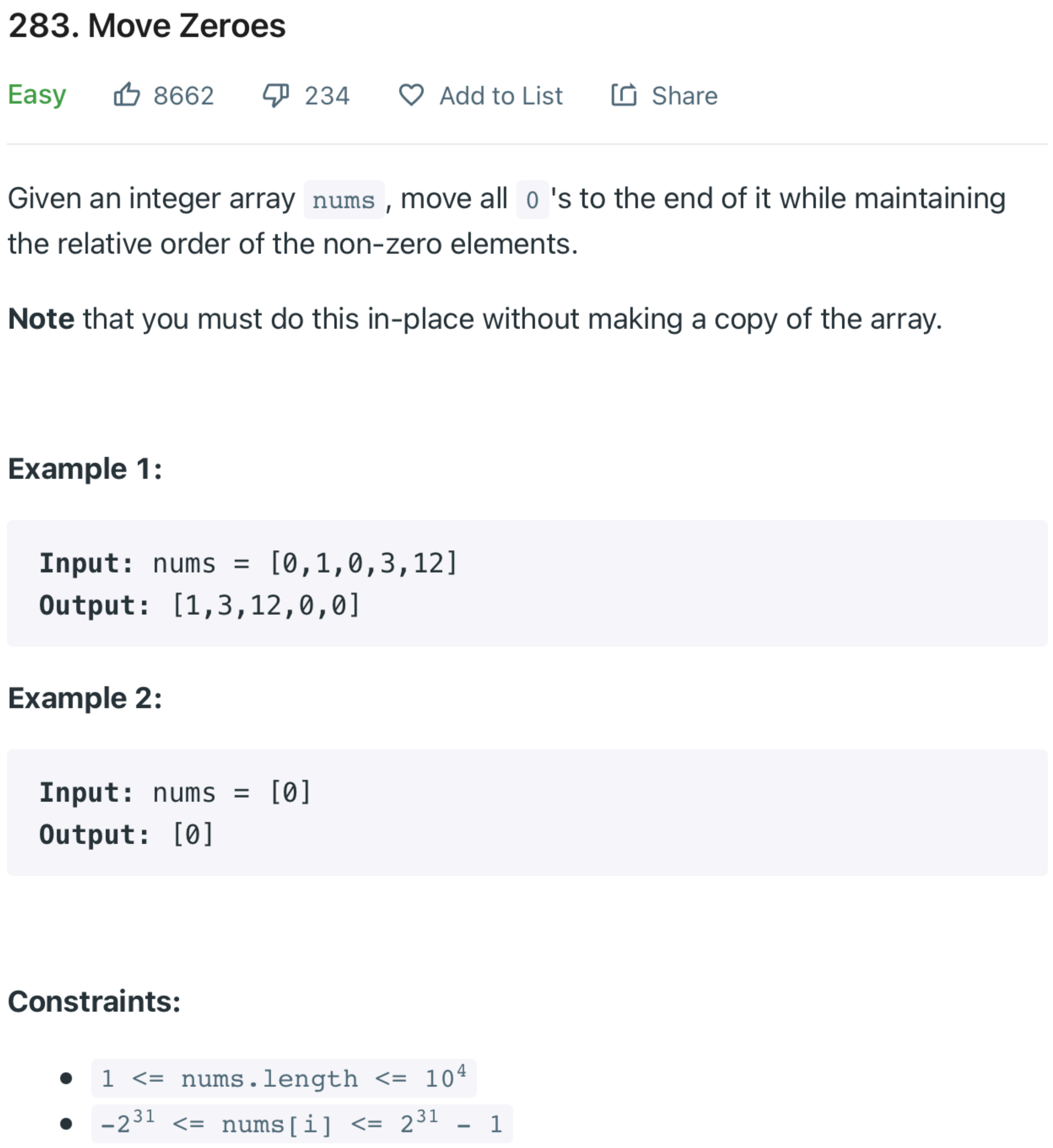


283. Move Zeroes

Given an integer array `nums`, move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:

```
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Example 2:

```
Input: nums = [0]
Output: [0]
```

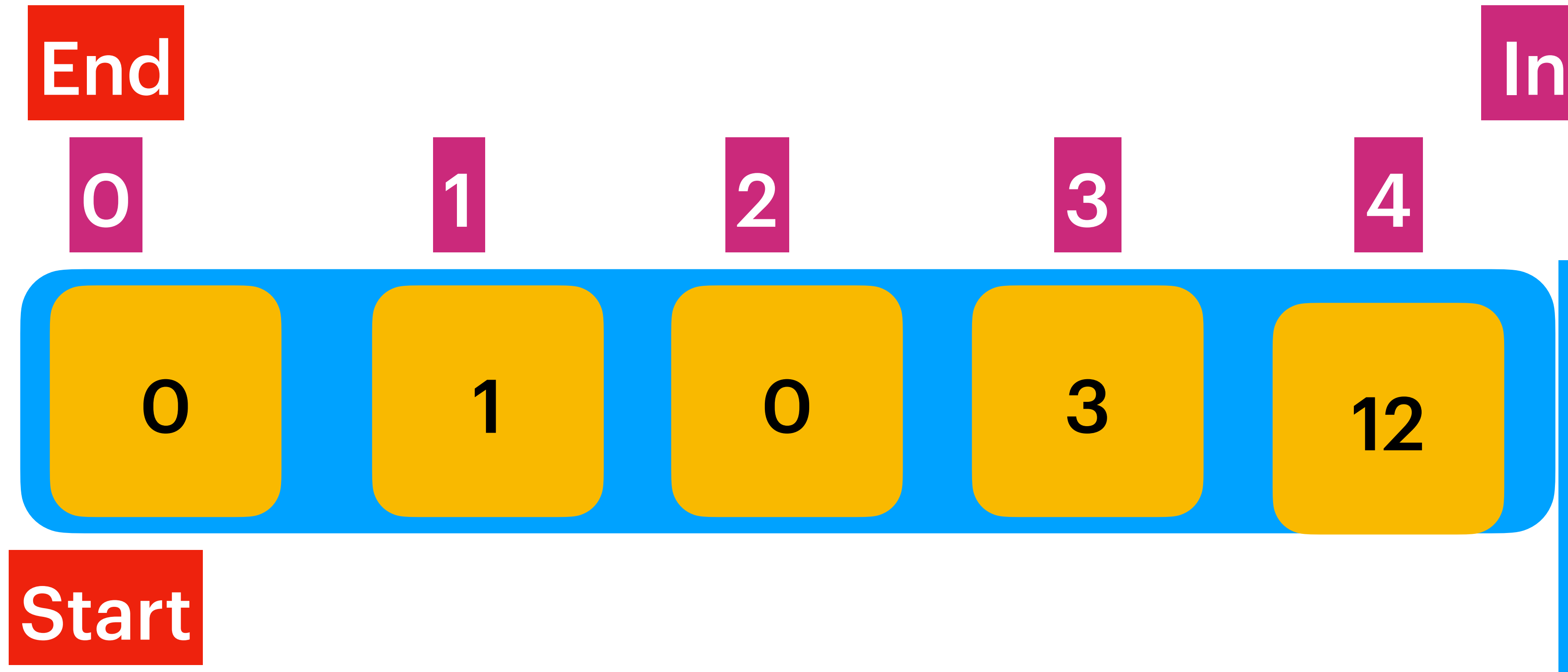
Constraints:

- `1 <= nums.length <=` 10^4
- -2^{31} `<= nums[i] <=` $2^{31} - 1$

In Place Algorithm

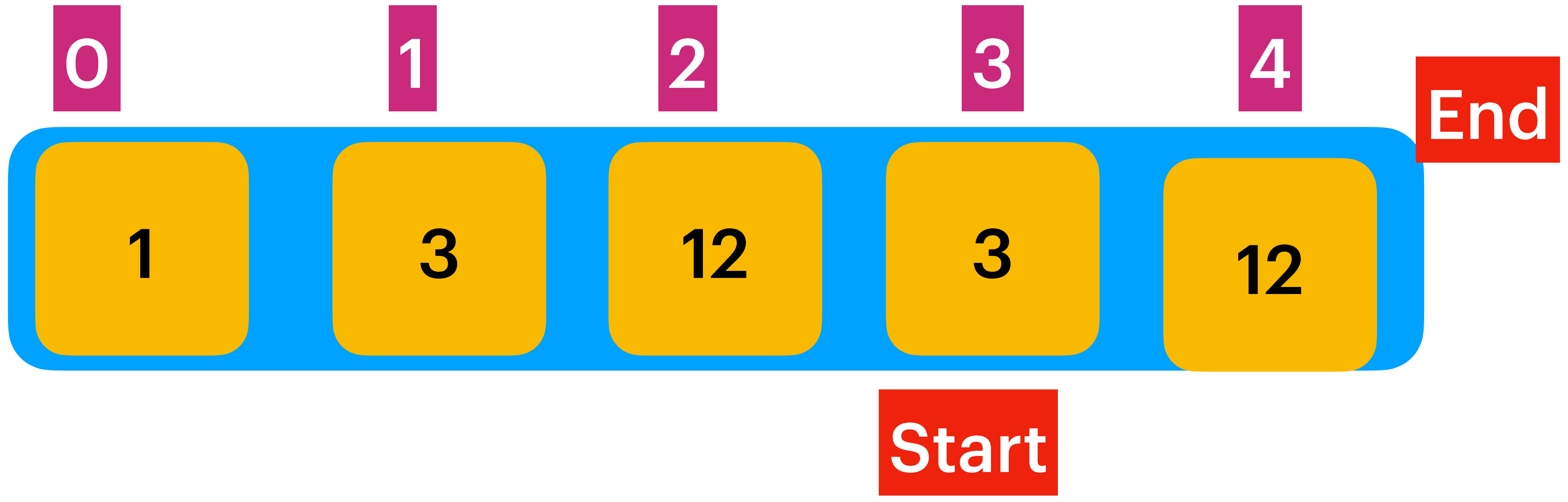
It’s a two pointer technique.

LeetCode 283. Move Zeros



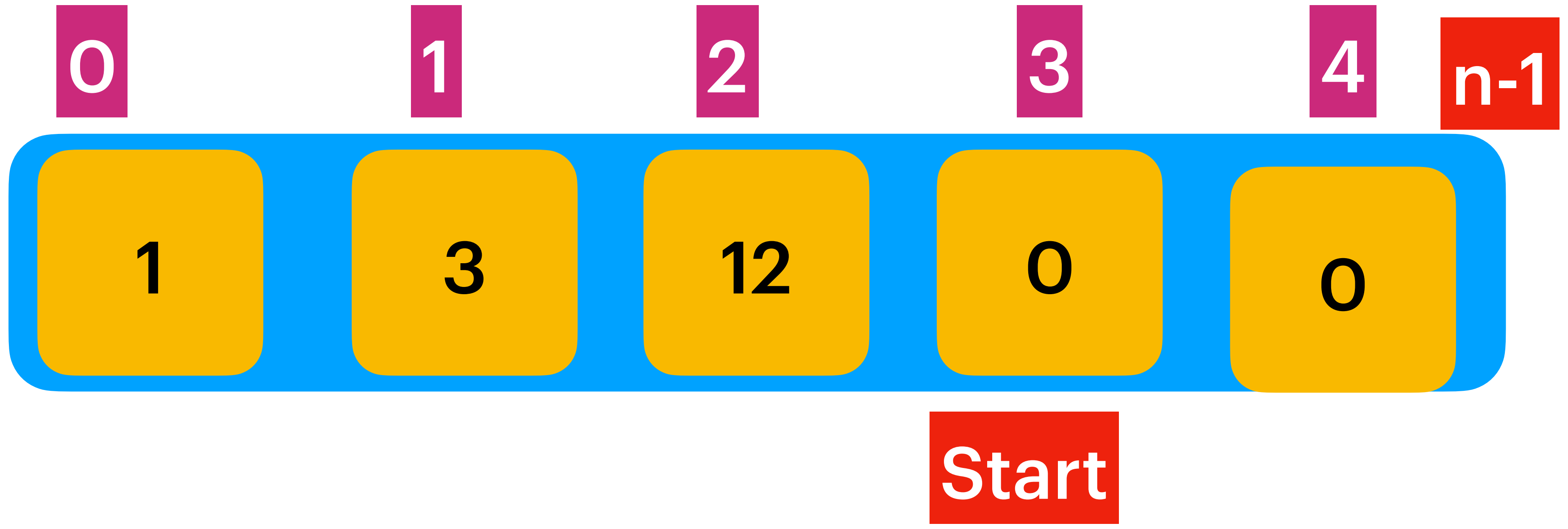
-> Take two pointers start & end starting from index zero.
-> move end pointer till length ’n-1‘
-> When ever end pointer pointing to non-zero value replace with start pointer then increment start pointer.
-> In 2nd pass from index ‘start’ to n-1 replace with zeros.

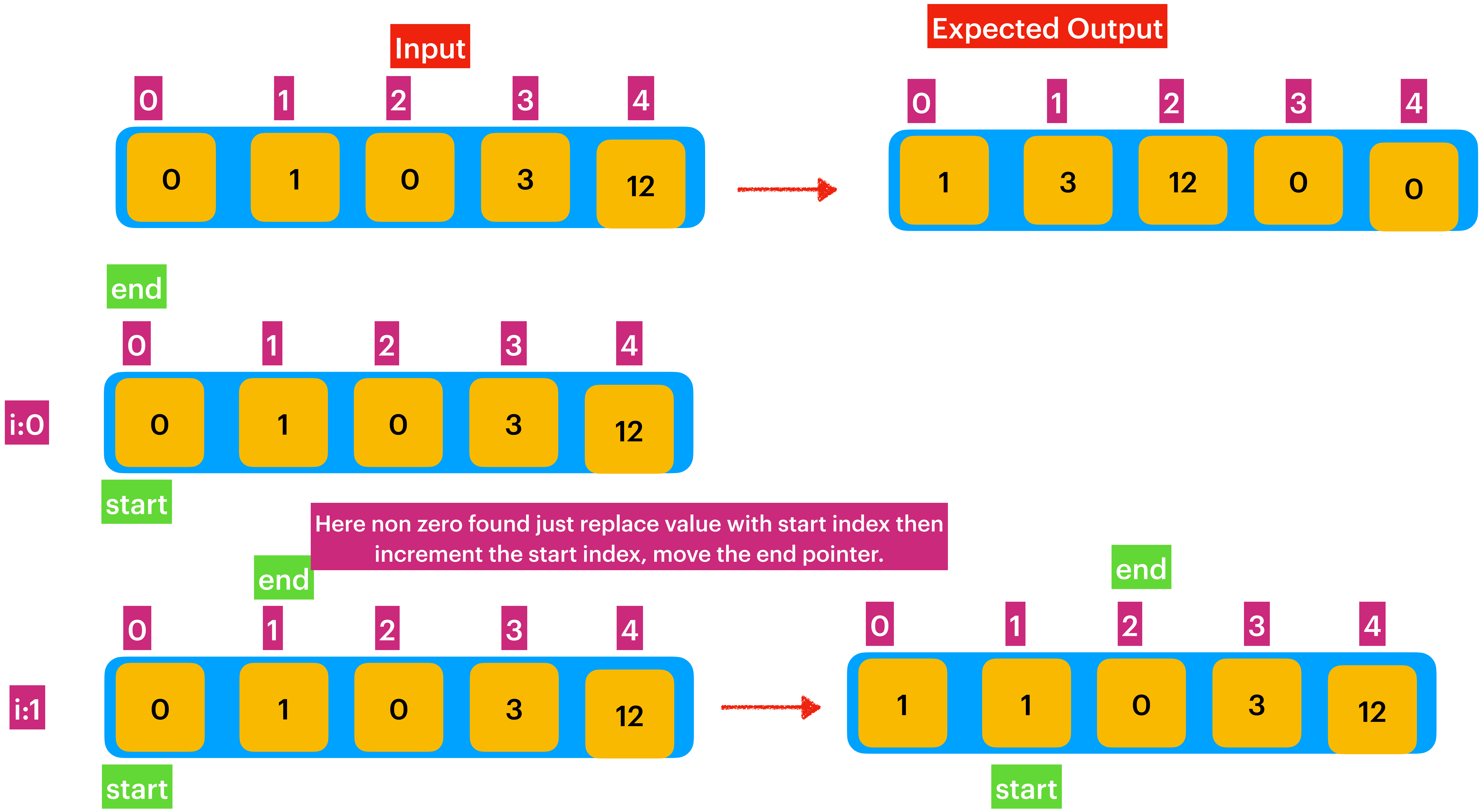
Step1 : Move all the non zero values to front



Time Complexity : O(n)
Space Complexity : O(1)

Step2 : Replace From index start to n-1 with zero





0	1	2	3	4
0	1	0	3	12

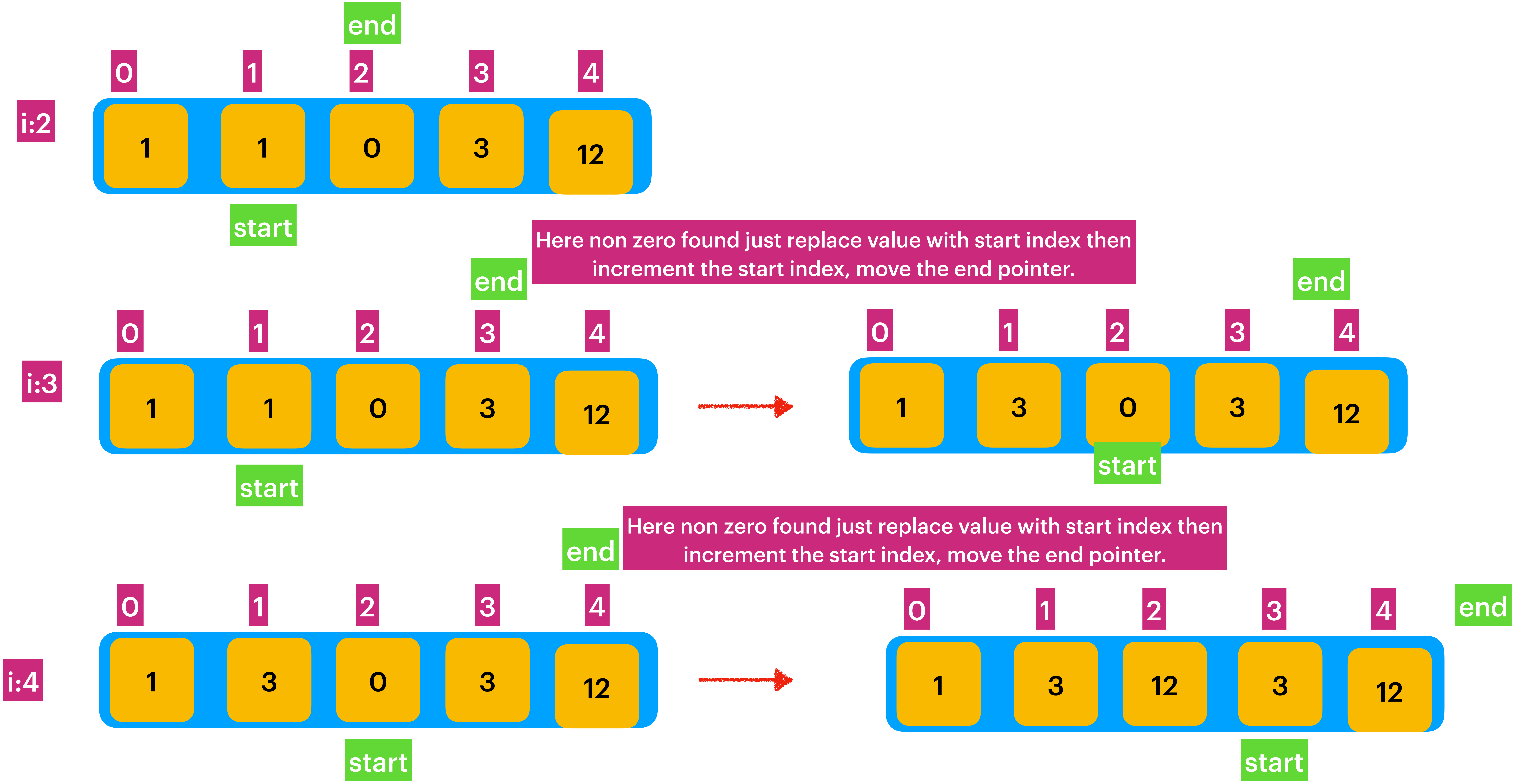
0	1	2	3	4
1	3	12	0	0

0	1	2	3	4
0	1	0	3	12

Here non zero found just replace value with start index then increment the start index, move the end pointer.

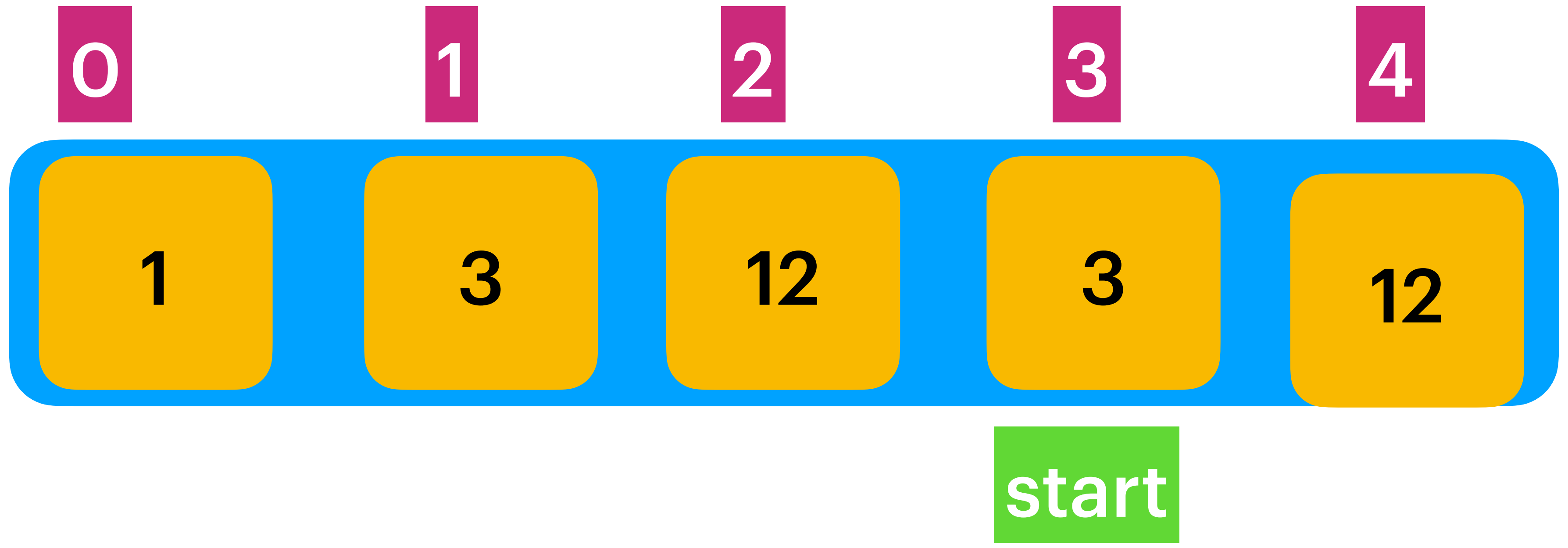
0	1	2	3	4
0	1	0	3	12

0	1	2	3	4
1	1	0	3	12



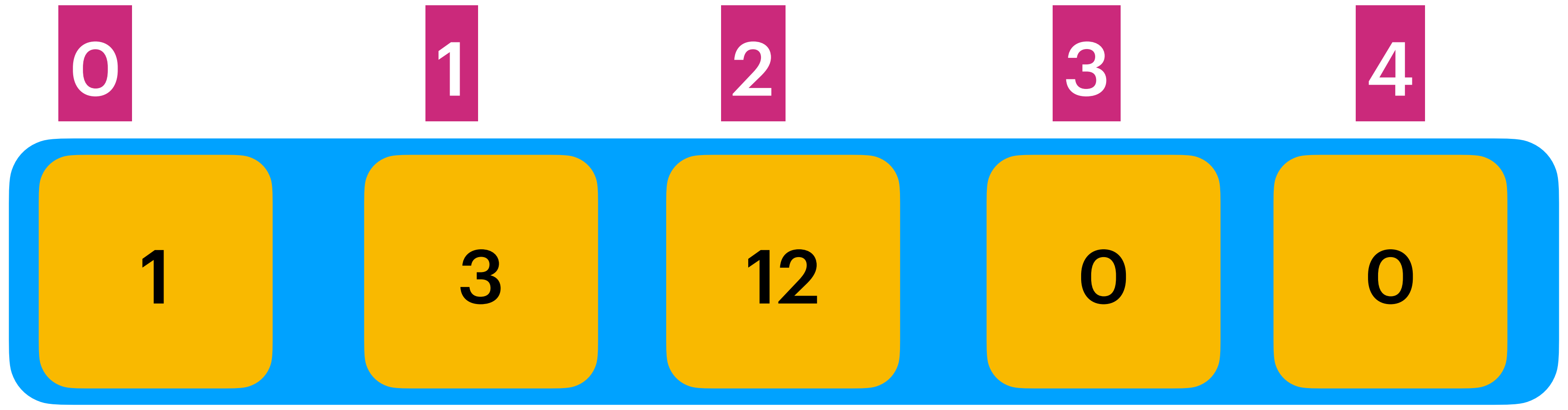
Here non zero found just replace value with start index then increment the start index, move the end pointer.

Here non zero found just replace value with start index then increment the start index, move the end pointer.



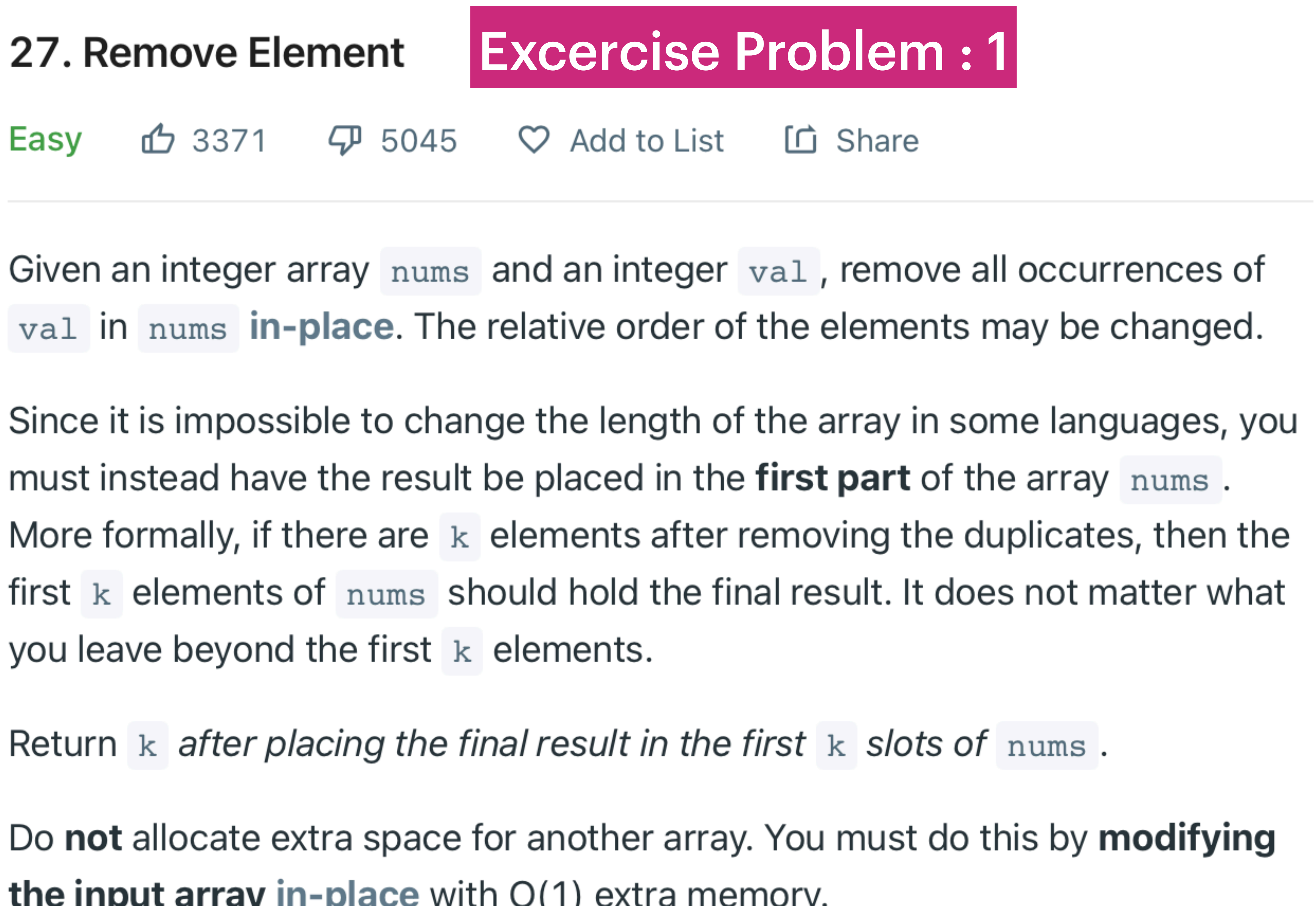
0	1	2	3	4
1	3	12	3	12
			start	

Step2 : Replace From index start to n-1 with zero



0	1	2	3	4
1	3	12	0	0

Time Complexity : O(n)
Space Complexity : O(1)



27. Remove Element

Excercise Problem : 1

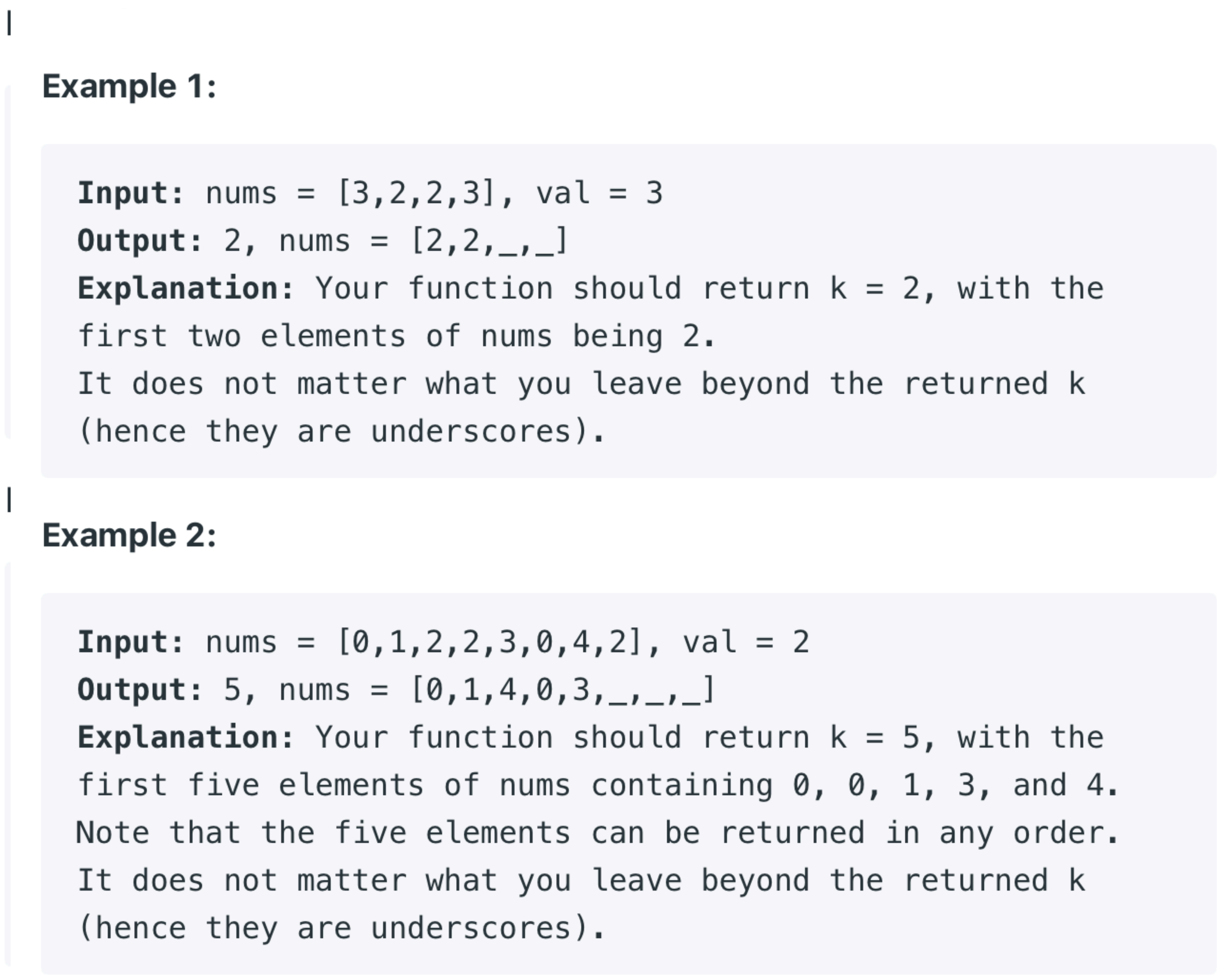
Easy 👍 3371 👎 5045 ♡ Add to List ⎘ Share

Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` **in-place**. 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** with O(1) extra memory.

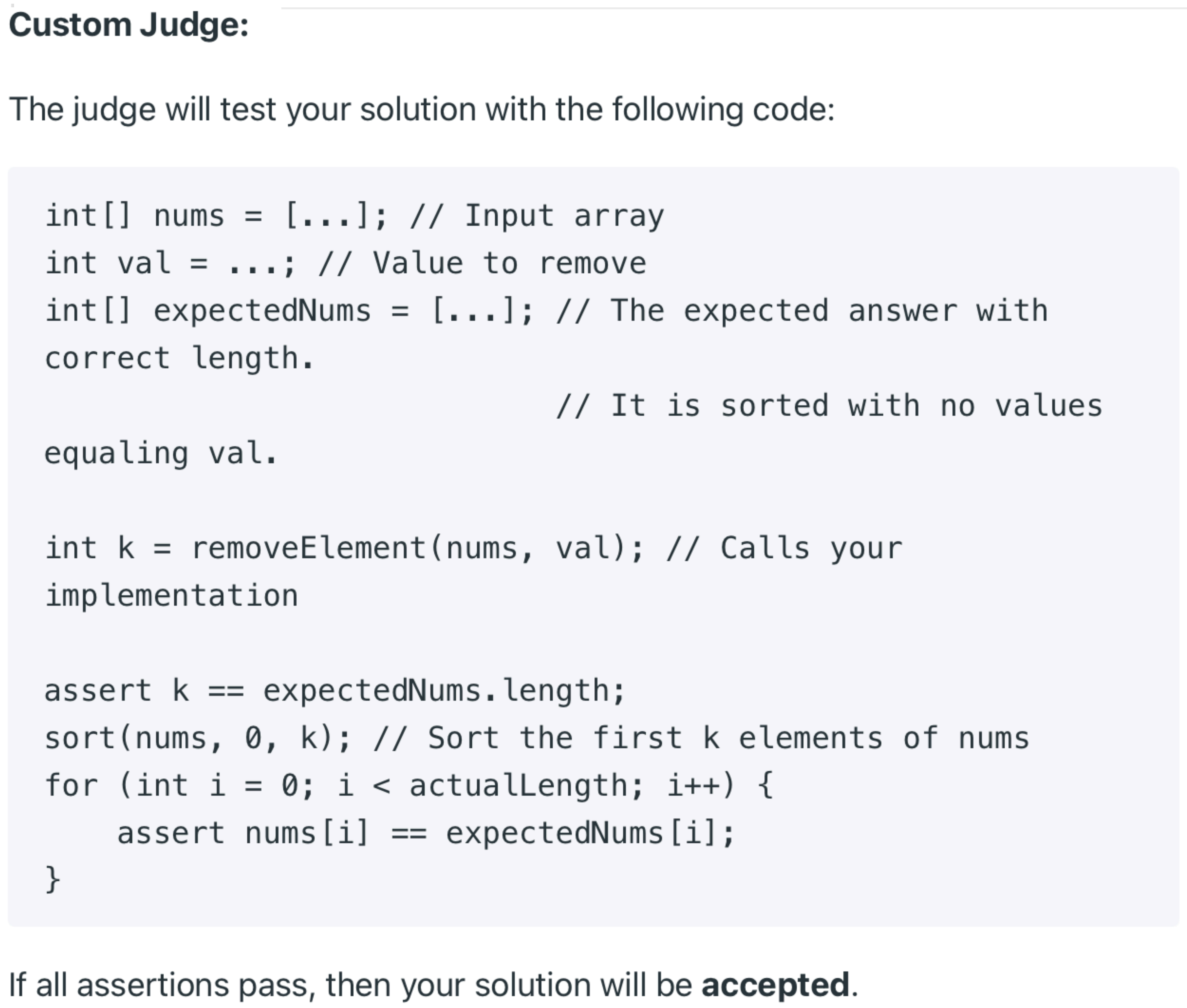


Example 1:

```
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).
```

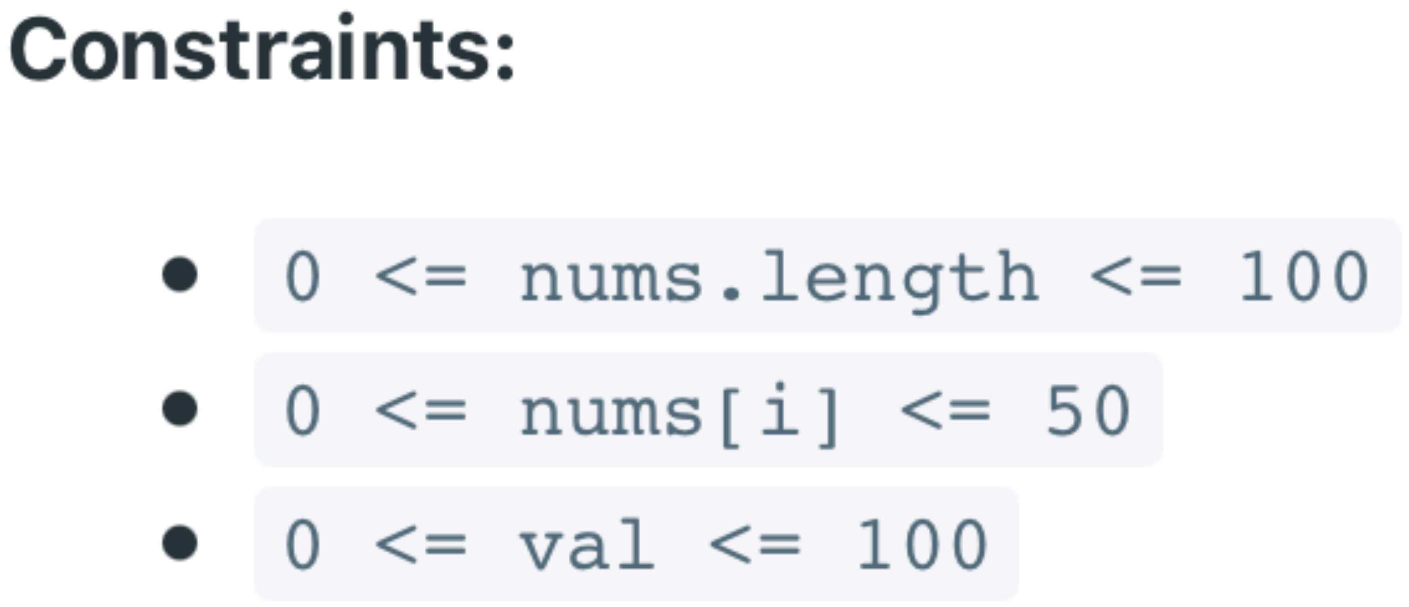


Custom Judge:

The judge will test your solution with the following code:

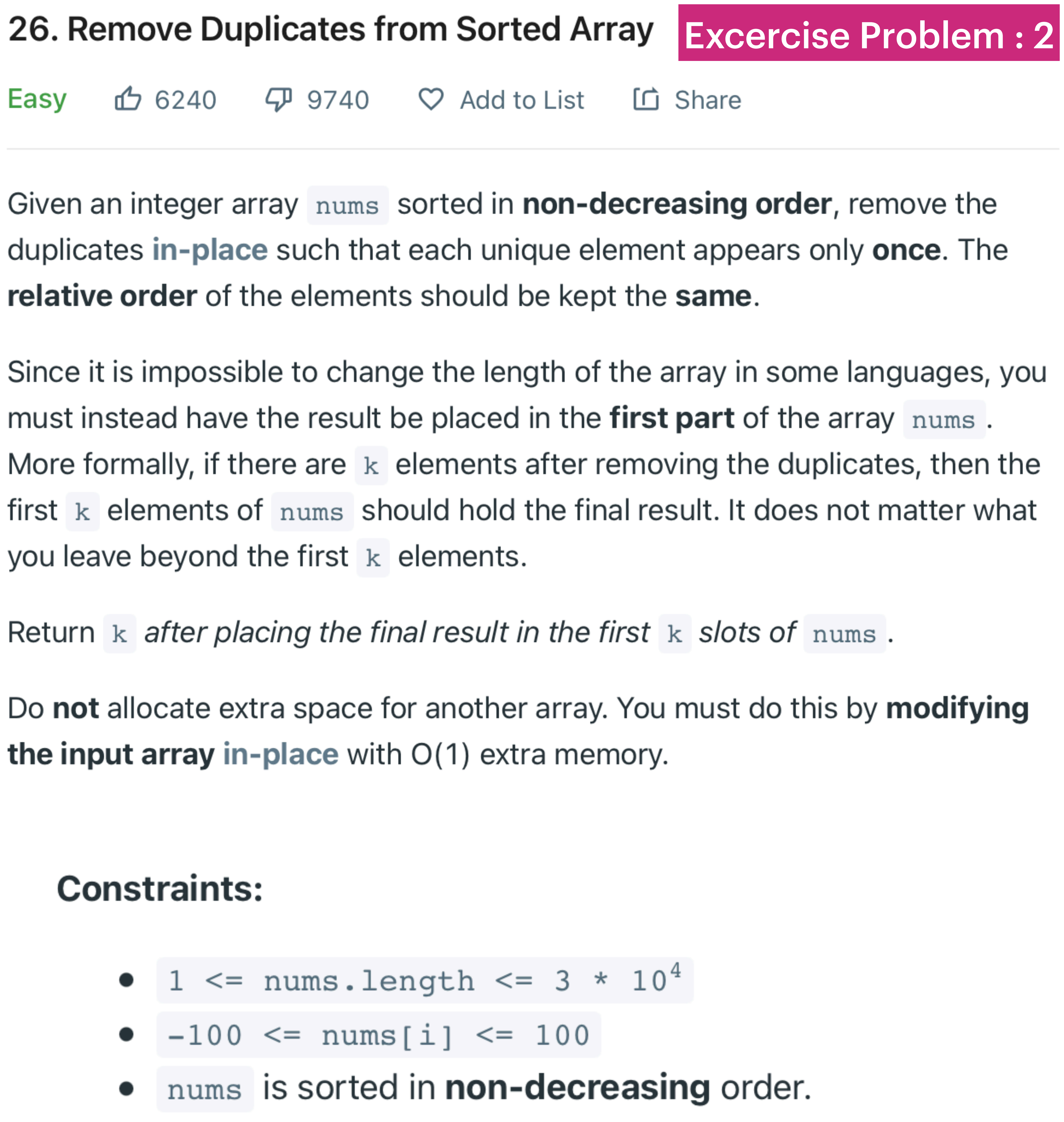
```
int[] nums = [...]; // Input array
int val = ...; // Value to remove
int[] expectedNums = [...]; // The expected answer with
correct length.
                            // It is sorted with no values
equaling val.

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**.



Constraints:

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



26. Remove Duplicates from Sorted Array

Excercise Problem : 2

Easy 👍 6240 👎 9740 ♡ Add to List Share

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**.

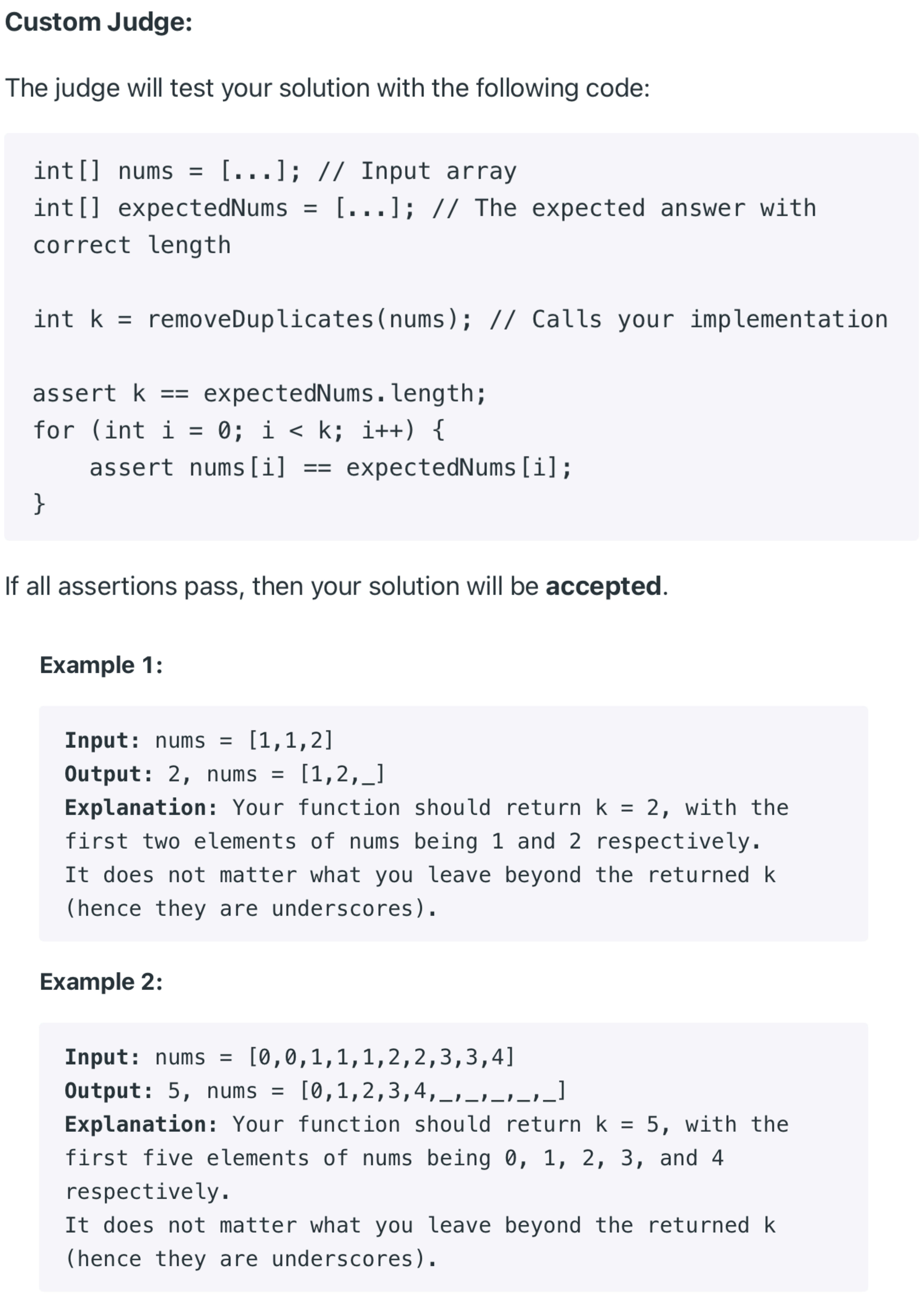
Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first part** of the array `nums`. More formally, if there are `k` elements after removing the duplicates, then the first `k` elements of `nums` should hold the final result. 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.

Constraints:

- `1 <= nums.length <= 3 * 10`^{4}
- `-100 <= nums[i] <= 100`
- `nums` is sorted in **non-decreasing** order.



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 = [1,1,2]
Output: 2, nums = [1,2,_]
Explanation: Your function should return k = 2, with the
first two elements of nums being 1 and 2 respectively.
It does not matter what you leave beyond the returned k
(hence they are underscores).
```

Example 2:

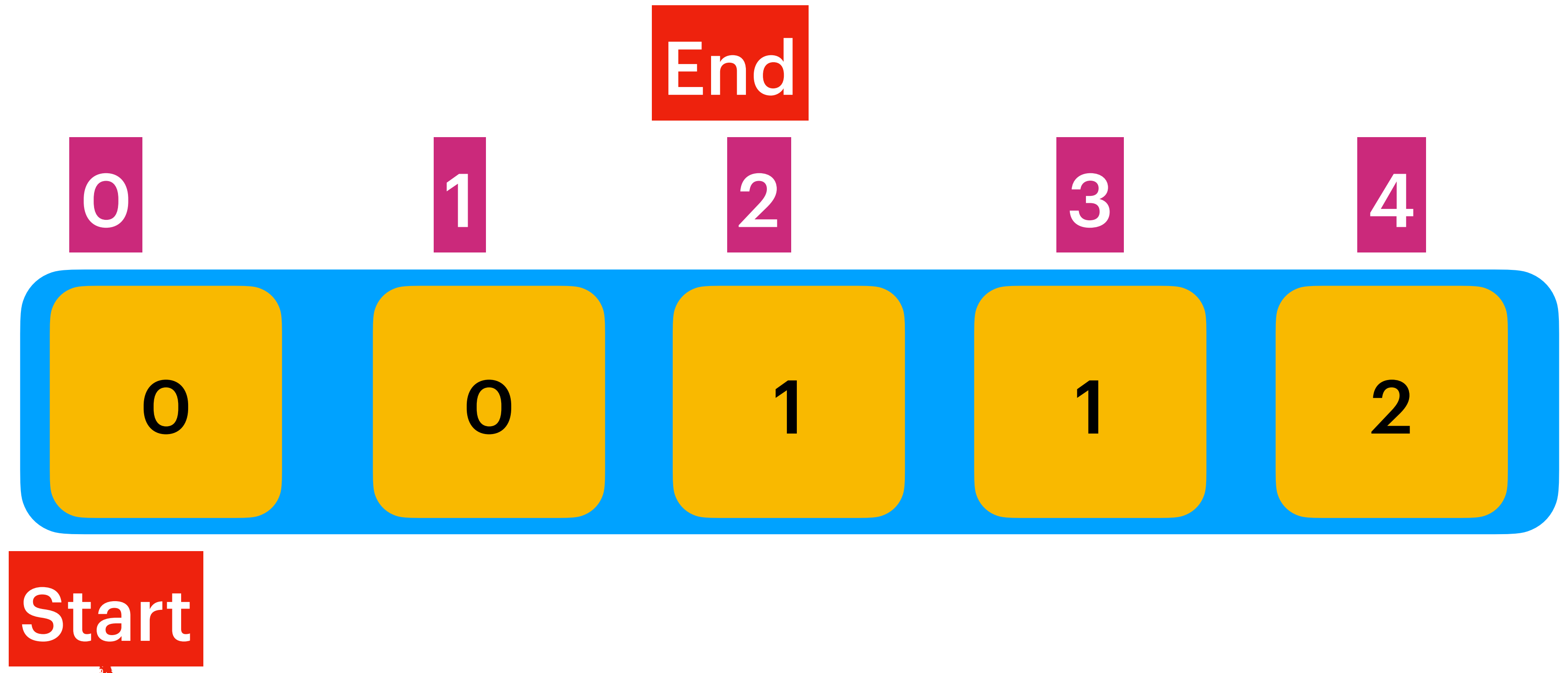
```
Input: nums = [0,0,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]
Explanation: Your function should return k = 5, with the
first five elements of nums being 0, 1, 2, 3, and 4
respectively.
It does not matter what you leave beyond the returned k
(hence they are underscores).
```

It’s a two pointer technique.

In Place Algorithm

LeetCode 26. Remove Duplicates in Sorted Array

-> Take two pointers starting from index start = 0 & end = 1.
-> end pointer moving till length ’n'
-> When ever ‘end’ value is greater then ‘start’ then increment the start then replace with end value.

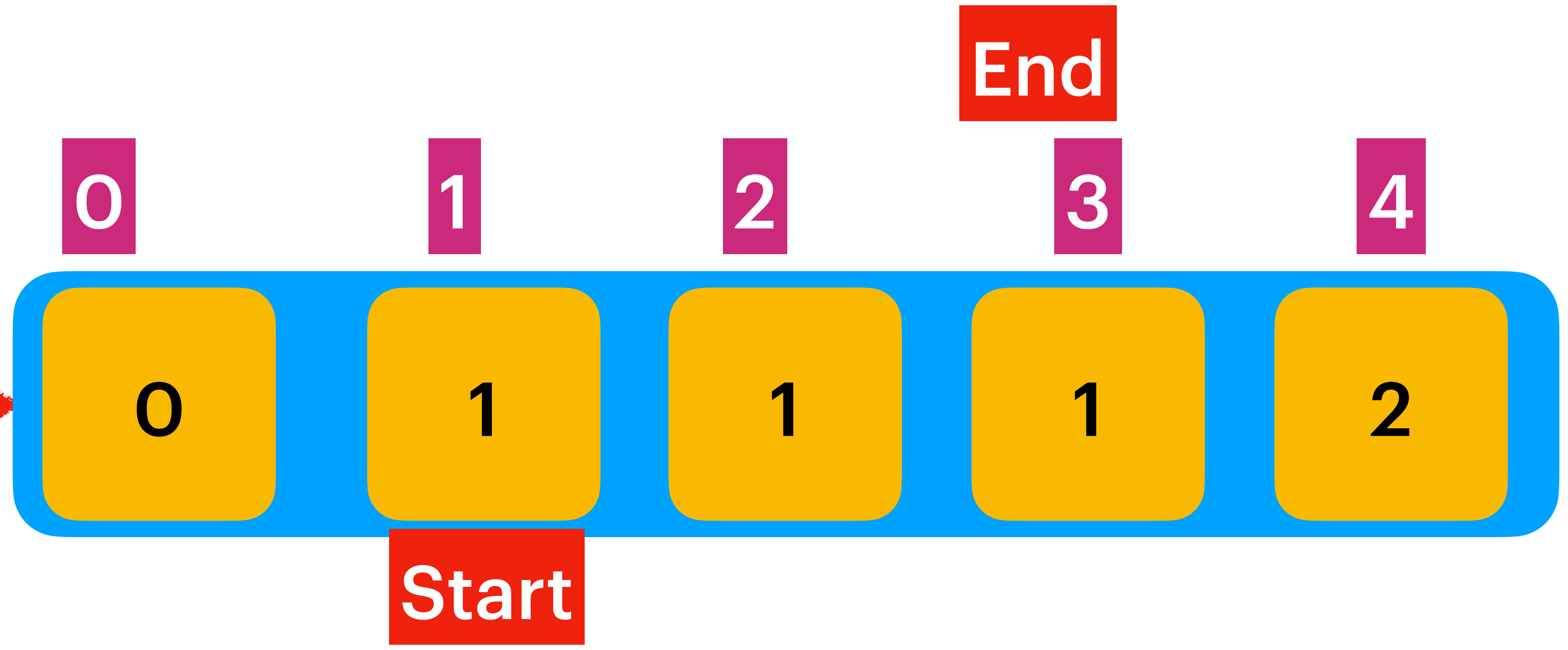


		End		
0	1	2	3	4
0	0	1	1	2
Start				

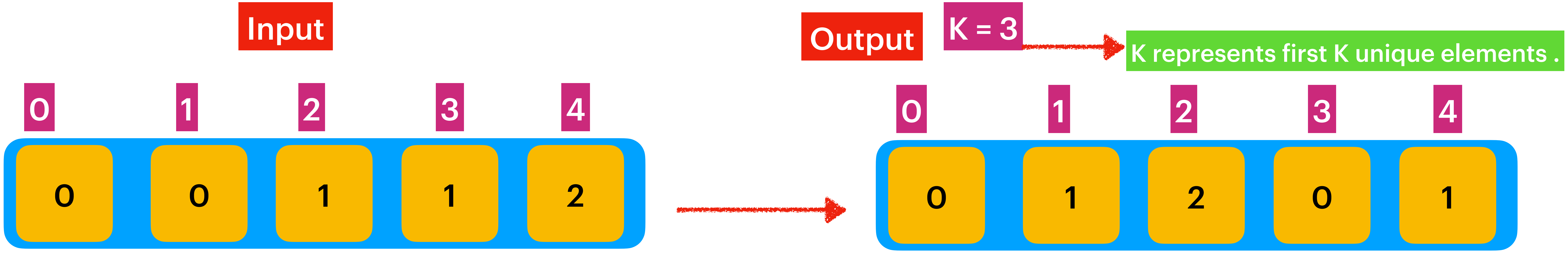
Time Complexity : O(n)
Space Complexity : O(1)

Clue

Current start position is valid till the arr[end] == arr[start].
Whenever arr[end] > arr[start] then just increment the start , replace with end value, then move the end pointer.



			End	
0	1	2	3	4
0	1	1	1	2
	Start			



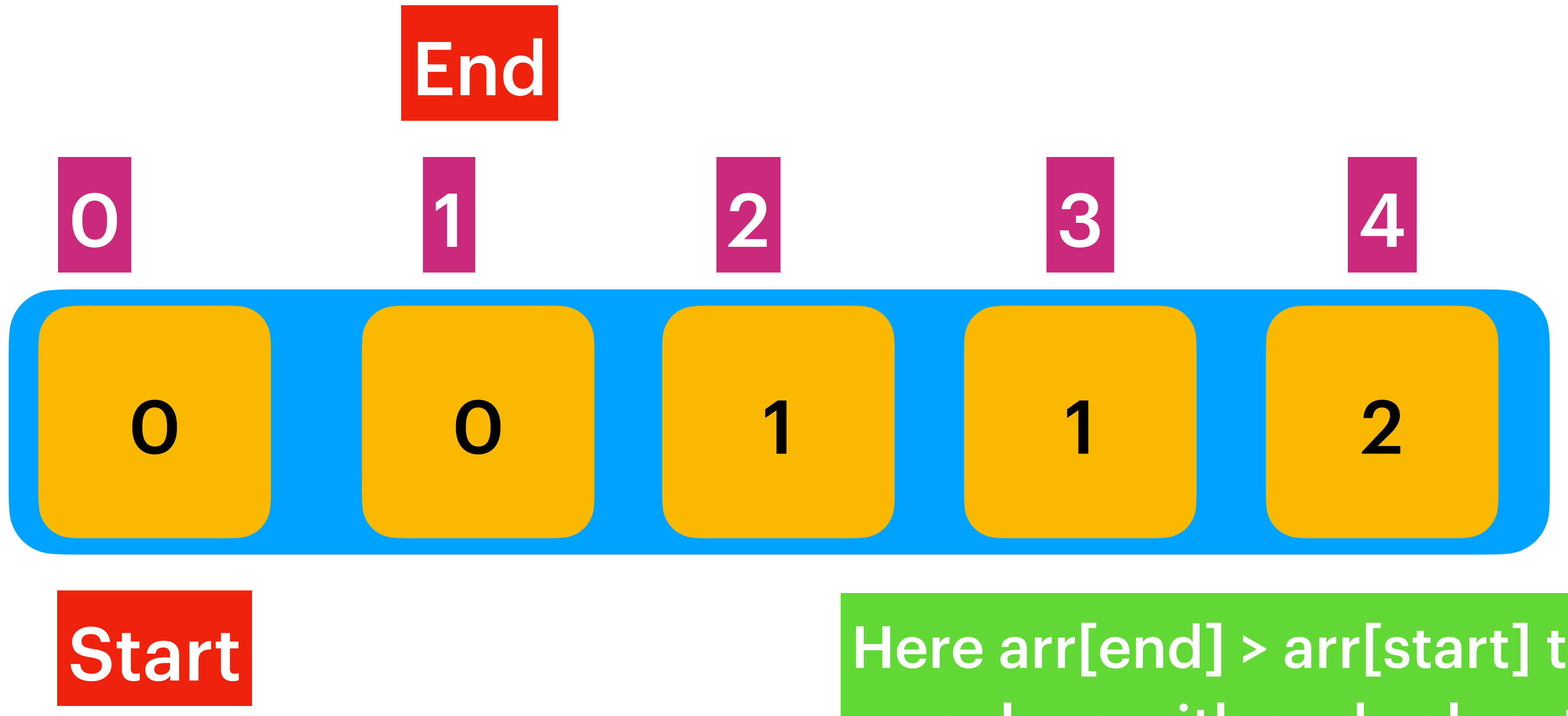
Input

0	1	2	3	4
0	0	1	1	2

Output

K = 3 → K represents first K unique elements .

0	1	2	3	4
0	1	2	0	1



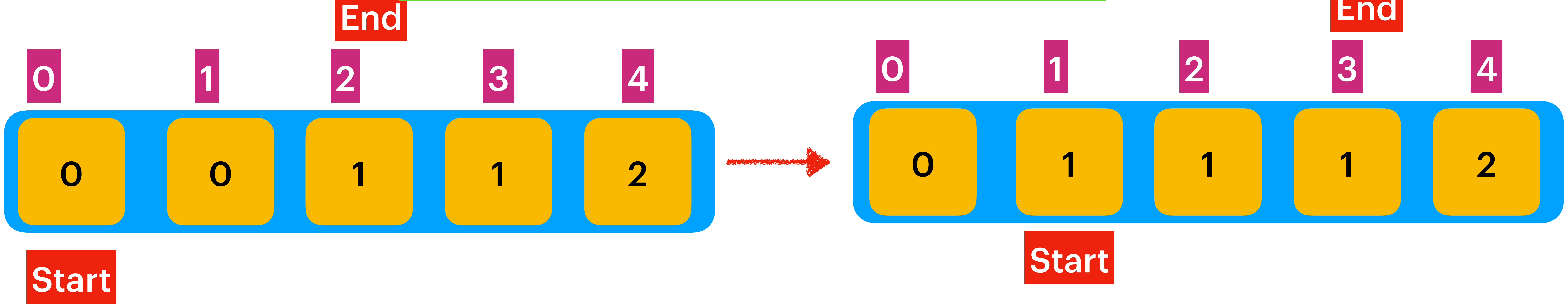
End

0	1	2	3	4
0	0	1	1	2

Start

Time Complexity : O(n)
Space Complexity : O(1)

Here arr[end] > arr[start] then just increment the start , replace with end value, then move the end pointer.



End

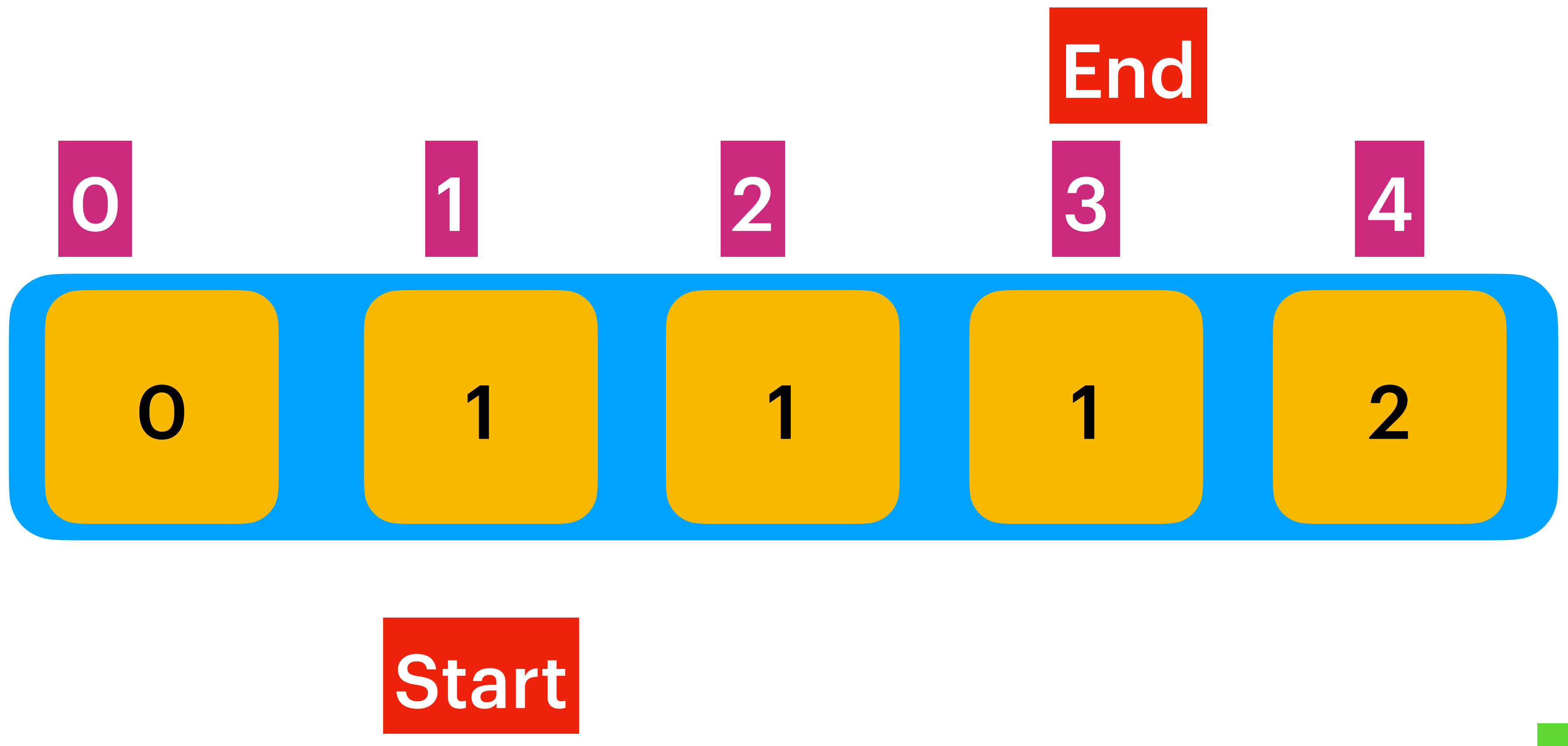
0	1	2	3	4
0	0	1	1	2

Start

End

0	1	2	3	4
0	1	1	1	2

Start

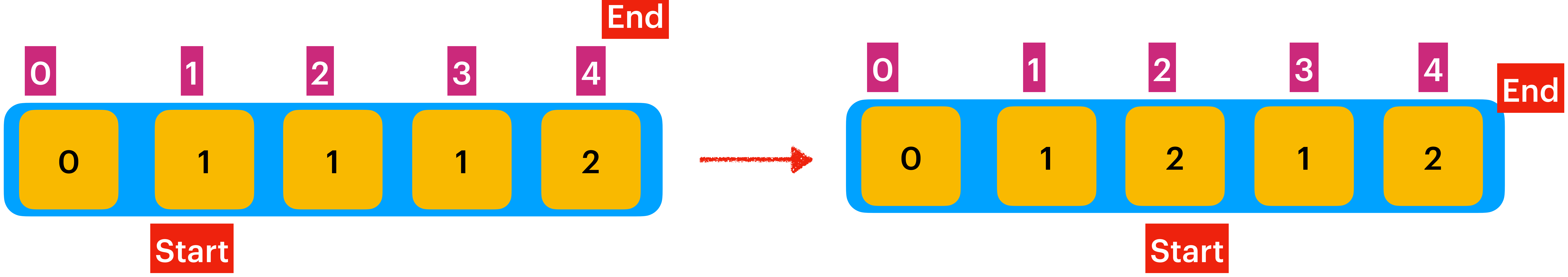


End

0	1	2	3	4
0	1	1	1	2

Start

Here arr[end] > arr[start] then just increment the start , replace with end value, then move the end pointer.



End

0	1	2	3	4
0	1	1	1	2

Start

0	1	2	3	4
0	1	2	1	2

End

Start

return start+1