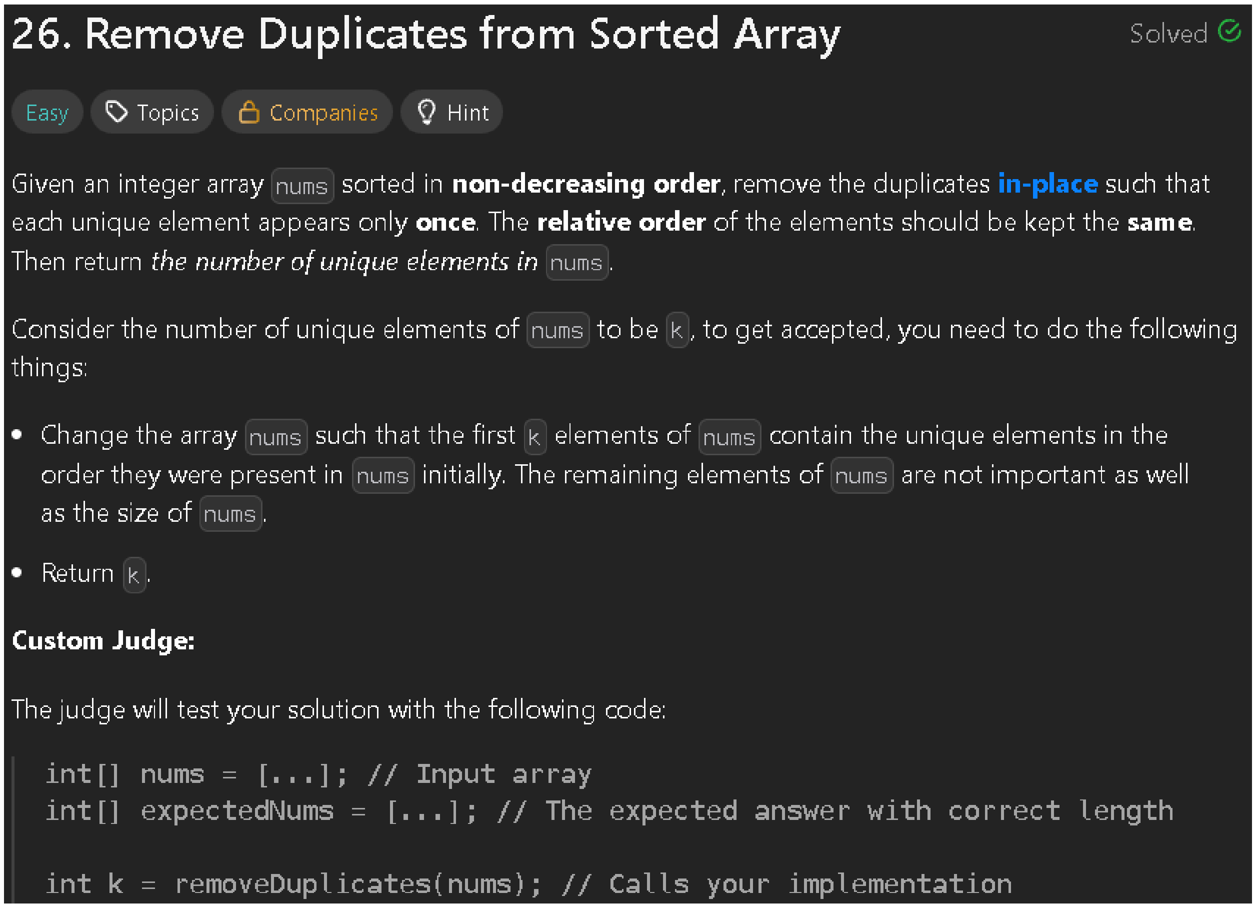


26. Remove Duplicates from Sorted Array

Solved

Easy · Topics · Companies · Hint

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**. Then return *the number of unique elements in* `nums`.

Consider the number of unique elements of `nums` to be `k`, to get accepted, you need to do the following things:

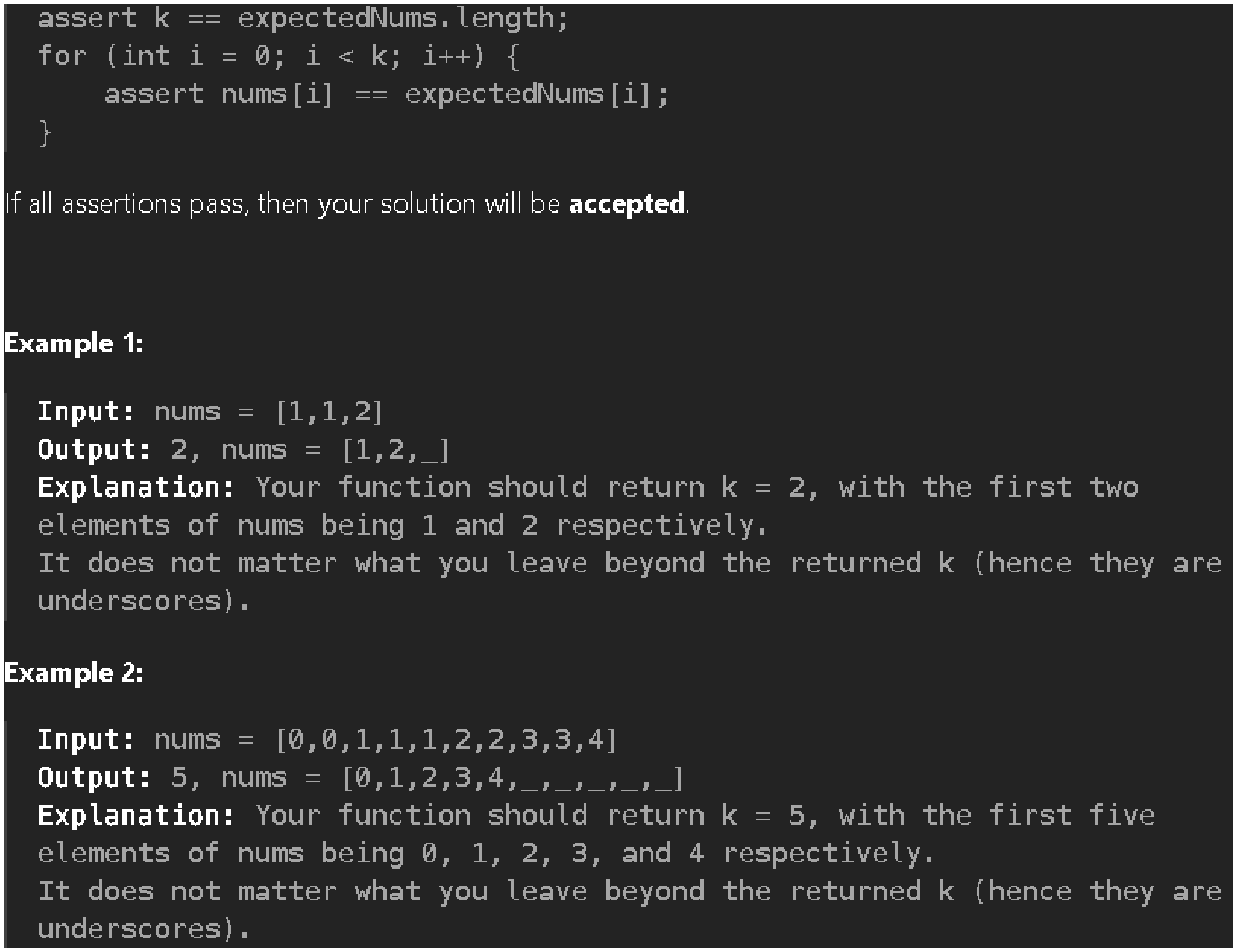
- Change the array `nums` such that the first `k` elements of `nums` contain the unique elements in the order they were present in `nums` initially. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] 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];
}
```

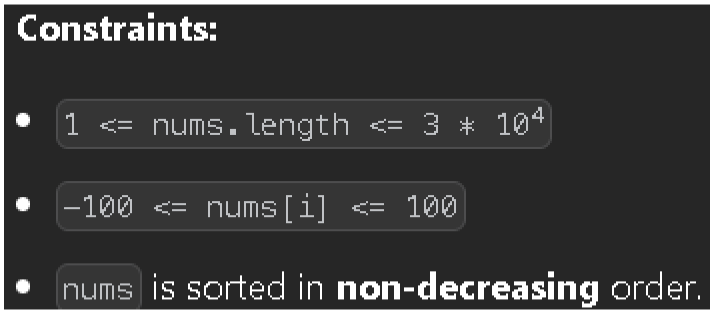
If all assertions pass, then your solution will be **accepted**.

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).
```



Constraints:

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

Python:

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        # Edge case: if array has 0 or 1 element
        if not nums:
            return 0

        # Pointer for placing unique elements
        k = 1
        # Iterate from 2nd element to end
        for i in range(1, len(nums)):
            if nums[i] != nums[i - 1]:  # Found a unique element
                nums[k] = nums[i]       # Place it at index k
                k += 1                  # Move the pointer

        return k
```

JavaScript:

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var removeDuplicates = function(nums) {
    if (nums.length === 0) return 0;

    let k = 1; // pointer for unique elements

    for (let i = 1; i < nums.length; i++) {
        if (nums[i] !== nums[i - 1]) {
            nums[k] = nums[i]; // move unique element forward
            k++;
        }
    }

    return k;
};
```

Java:

```
class Solution {
    public int removeDuplicates(int[] nums) {
        // If array has 0 or 1 element, return its length
        if (nums.length == 0) {
            return 0;
        }

        int k = 1; // Pointer for placing unique elements

        for (int i = 1; i < nums.length; i++) {
            // Compare current element with the last unique one
            if (nums[i] != nums[k - 1]) {
                nums[k] = nums[i]; // Place unique element
                k++;
            }
        }
        return k; // Number of unique elements
    }
}
```