**Assignment 1**

1. **Remove Duplicates from Sorted Array [2 points]**

Given an integer array nums sorted in **non-decreasing order**, remove the duplicates [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) 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 Evaluation:**

We 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 considered **correct**.

**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 \* 10000
* -100 <= nums[i] <= 100
* nums is sorted in **non-decreasing** order.

1. **Remove Element [2 points]**

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 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 Evaluation:**

We 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 considered **correct**.

**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

1. **Two Sum [1 point]**

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

**Example 1:**

**Input:** nums = [2,7,11,15], target = 9

**Output:** [0,1]

**Explanation:** Because nums[0] + nums[1] == 9, we return [0, 1].

**Example 2:**

**Input:** nums = [3,2,4], target = 6

**Output:** [1,2]

**Example 3:**

**Input:** nums = [3,3], target = 6

**Output:** [0,1]

**Constraints:**

* 2 <= nums.length <= 10000
* -1000000000 <= nums[i] <= 1000000000
* -1000000000 <= target <= 1000000000