Leet Code Problems

Day 1 :

Two Sums:

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]

Solution:

public class Solution {

    public int[] TwoSum(int[] nums, int target) {

        int [] result=new int [2];

       int j=0;

        for(int i=0;i<nums.Length;i++){

            for(int k=i+1;k<nums.Length;k++){

            if(nums[i]+nums[k]==target){

               result[j]=i;

               j+=1;

               result[j]=k;

               return result;

            }

        }

    }

        return Array.Empty<int>();

    }

}

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).

Solution:

public class Solution {

    public int RemoveElement(int[] nums, int val) {

        int length=nums.Length;

       // int [] arr=new int[length];

        int j=0;

        for(int i=0;i<length;i++){

            if(nums[i]!=val){

                nums[j]=nums[i];

                j=j+1;

            }

        }

        return j;

    }

}