Leet Code Week :-2

1.Two Sum

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]

Java Solution:-

```
class Solution {
public int[] twoSum(int[] nums, int target) {
int i,j;
for(i=0;i<nums.length;i++){
for(j=0;j<nums.length;j++)
{
if( i== j){
continue;
}
if(nums[i]+ nums [j]== target){
return new int[]{i,j};
}
}
```

```
}
return new int[0];
}
}
```

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]

Java Solutions:-

```
class Solution {
public void moveZeroes(int[] nums) {
int i=0;
for(int n:nums){
if(n!=0)
nums[i++]=n;
}
while(i<nums.length){
nums[i++]=0;
}
}
}
```

27. Remove Element

Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. 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 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).

Java Solution:-

```
class Solution {
public int removeElement(int[] nums, int val) {
int i=0;
for(int a:nums){
if(a!=val)
nums[i++]=a;
}
return i;
}
}
```

80. Remove Duplicates from Sorted Array II

Given an integer array nums sorted in **non-decreasing order**, remove some duplicates **in-place** such that each unique element appears **at most twice**. 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.

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 = [1,1,1,2,2,3]

Output: 5, nums = [1,1,2,2,3,_]

Explanation: Your function should return k = 5, with the first five elements of nums being 1, 1, 2, 2 and 3 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

Example 2:

Input: nums = [0,0,1,1,1,1,2,3,3]

Output: 7, nums = [0,0,1,1,2,3,3,_,_]

Explanation: Your function should return k = 7, with the first seven elements of nums being 0, 0, 1, 1, 2, 3 and 3 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

Java Solutions:-

```
class Solution {

public int removeDuplicates(int[] nums) {

int k = 2;

for (int i = 2; i < nums.length; i++) {

if (nums[i] != nums[k - 2]) {

nums[k] = nums[i];

k++;

}

}

return k;

}

}
```

26. Remove Duplicates from Sorted Array

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).

Java Solution:-

class Solution {

public int removeDuplicates(int[] nums) {

int k = 0;

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

if (nums[i] != nums[k]) {

```
k++;
nums[k] = nums[i];
}
}
return k+1;
}
}
```

2235. Add Two Integers

Given two integers num1 and num2, return *the **sum** of the two integers*.

Example 1:

Input: num1 = 12, num2 = 5

Output: 17

Explanation: num1 is 12, num2 is 5, and their sum is 12 + 5 = 17, so 17 is returned.

Example 2:

Input: num1 = -10, num2 = 4

Output: -6

Explanation: num1 + num2 = -6, so -6 is returned.

Java Solution:-

```
class Solution {
public int sum(int num1, int num2) {
return num1+num2;
}
}
```