LeetCode Problem Solving

Week-1

Q. Merge Sorted Array → Number 88

=> Given Two Array. Array→1 Size is (m+n) Array→2 Size is (n). m is the number of element currently in Array→1.

Our Task Is To Merge This Two Array. There is two Method.

First Method: Copying The Elements Of Array→2 Into Array→1. Then Sort The Array Again. But For This Method We Need Nested Loops which cause O(n^2) Time Complexity. Here is the code:

```
int i,j,temp;
    for(i=m;i<m+n;i++){                → This Loop Copy Array-2 into 1
           nums1[i] = nums2[i-m];
    }
    for(i=0;i<m+n;i++){
            for(j=0;j<m+n;j++){
                if(nums1[j]<nums1[i]){
                     continue;
                  } else{
                        temp = nums1[j];
                        nums1[j] = nums1[i];
                        nums1[i] = temp;
                  }
            }
    }
```

So There Is A Solution And The Solution Is Second Method Which Cause O(n) Time Complexity.

Second Method: Two Pointer Method. Here We Will Use Two Pointer I,J. I will point to the last element of Array→1 and J will point to the last element of Array→2. Then we will use K which will indicate the array→1 last index. Then We Will Compare I & J and put the element into Array-1[k]. Here is the code of this method.

```
Int I = m-1;                       → This points the last element of Array-1
int j = n-1;                       → This points the last element of Array-2
int k = m+n-1;                     → This points the last index of Array-1

while(i>=0 && j>=0){               → Loop Until I & J = 0
        if(nums2[j] > nums1[i]){
                nums1[k--] = nums2[j--];
        } else{
                nums1[k--] = nums1[i--];
        }
}

while(j>=0){                       → Loop Until J = 0 But Here I already Less Than 0
        nums1[k--] = nums2[j--];   → Since here Array-1 now have no element so
}                                    directly putting the element of array-2 into array-1
```

Q. Remove Element → Number 27

=> Given An Array and An Integer Val. Target Is To Find The Val into Array And Delete That. Two Method Are There To Solve This.

First Method: We Can Delete The Val From The Array Using Shifting Method. First We Will Find The Val In array then Shift that into Last and Decrease the Size Of The Array. But Problem Is That this Takes O(n^2) Time Complexity Because Here We Need Nested Loop. One For Traverse The Array and One For Shifting. Here is the Code.

```
int removeElement(int* nums, int numsSize, int val) {

        int i = 0;
        int length = numsSize;

        while (i < length) {
                if (nums[i] == val) {
                        // shift elements left
                        for (int j = i; j < length - 1; j++) {
                                nums[j] = nums[j + 1];
                        }
                        length--; // decrease array size
                        // don't increase i, as new value comes in this place
                } else {
                        i++;
                }
        }
        return length;
}
```

To Avoid Nested Loop We Have A Solution Which Is Two Pointer Method. Here is the method.

Second Method: In Two Pointer Method We Will Use two pointer one is I and J. I will Point First Element And J will Point Last Element. When I == Val We Will Replace I value with J. So Now J will contain The Value of I and then decrease J. This Method Use O(n) Time Complexity. Here is the code.

```
int removeElement(int* nums, int numsSize, int val){
        int i = 0;
        int j = numsSize-1;

        while(i<=j){
                if(nums[i] == val){
                        nums[i] = nums[j];
                        j--;
                } else{
                        i++;
                }
        }
        return j+1;
}
```

This Method Is Perfect If Order Doesn't Matter But If Order Matter Then We Have Another Method But That Method Is More Easy And Readable. So For This Problem Always The 3rd Method Is Best. Here Is The Method.

Third Method: This Method Is Forward Only And Simple Copy Method. This Is The Most Easiest Method Ever. This Also A Two Pointer Method But This Time We Wont Move Two Pointer At Opposite Direction Rather This Time We Will Move Two Pointer In Same Direction. Here Will Traverse The Array And Check If Value == Val Then then just skip. If Value != Val Then Simply Add It Into New Array Using Int K. This Method Time Complexity O(n). Here is the code

```
int removeElement(int* nums, int numsSize, int val) {
        int k = 0;
        for (int i = 0; i < numsSize; i++) {
                if (nums[i] != val) {
                        nums[k++] = nums[i];
                }
        }
        return k;
}
```

This Method Is The Best Answer And It's Very Easy Also.

Q. Remove Duplicates From Sorted Array → Number 26

=> Here Give A Sorted Array. Here Some Numbers May Be Duplicated. Our Task Is Simple that We Have To Write Every Number One's Which Mean We Have to Delete The Duplicate Numbers. Here We Will Use Two Pointer Method But Not Opposite Direct Move Method Rather We Will Move the two Pointer In Same Direction. And Every time we will check with previous one. If Current Value == Previous one means It Duplicated so Skip If != then Just Push It Into array Using K. Here Is The Code.

```
int remove Duplicates(int* nums, int numsSize){

        int k = 0;
        int i;
        for(i=0;i<numsSize;i++){
                if(i==0){
                        nums[k++] = nums[i];
                } else{
                        if(nums[i]!=nums[k-1]{
                                nums[k++] = nums[i];
                        }
                }
        }
        return k;
}
```