Assignment 1 (Arrays)

💡 **Q1.** 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: Input: nums = [2,7,11,15], target = 9 Output0 [0,1]

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

Sol:-

```
// Time Complexity: O(n)
// Space Complexity: O(n)
class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[] { map.get(complement), i };
            }
            map.put(nums[i], i);
        }
        return null;
    }
}
```

💡 **Q2.** 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.

Example : Input: nums = [3,2,2,3], val = 3 Output: 2, nums = [2,2,_ ,_]

Sol:-

```
      // Time Complexity: O(n)
// Space Complexity: O(1)
class Solution {
    public int removeElement(int[] nums, int val) {
        int i = 0;
        for(int j = 0; j < nums.length; j++){
            if(nums[j] != val){
                int temp = nums[i];
                nums[i] = nums[j];
```

```
                nums[j] = temp;
                i++;
            }
        }
        return i;
    }
}
```

💡 **Q3.** Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with O(log n) runtime complexity.

Example 1: Input: nums = [1,3,5,6], target = 5

Output: 2

Sol:-

```
// Time Complexity: O(logn)
// Space Complexity: O(1)
class Solution {
    public int searchInsert(int[] nums, int target) {
        int start = 0;
        int end = nums.length - 1;
        while(start<=end){
            int mid = start + (end-start)/2;
            if(nums[mid]==target){
                return mid;
            }
            else if(nums[mid]>target){
                end = mid-1;
            }
            else
            start = mid+1;
        }
        return start;
    }
}
```

💡 **Q4.** You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return the resulting array of digits.

Example 1: Input: digits = [1,2,3] Output: [1,2,4]

Explanation: The array represents the integer 123.

Incrementing by one gives 123 + 1 = 124. Thus, the result should be [1,2,4].

Sol:-

```
// Time Complexity: O(n)
// Space Complexity: O(1)
class Solution {
    public int[] plusOne(int[] digits) { // this is method signature
```

```
        for(int i=(digits.length)-1; i>=0;i--){ // loop iterates over elements of the
array in reverse order, starting from last digit.
            if(digits[i]<9){ // true
                digits[i] = digits[i]+1;//  adding one to the digits
                return digits;
            }else if(digits[i]==9){
                digits[i]=0; //
            }
        }
        int[] newarr = new int[digits.length+1]; // a new array newarr is created with a
length one greater than the original digits
        newarr[0] = 1;
        return newarr;
    }
}
// Approach:
// 1. As its already an int[] array given the last element of the array will be considered
as the units place of the number we need to add 1 for last digit if its less than 9
// 2. if the last element is greater than 9 we need to check for before element and
increment if its less than 9..
// 3. if all the elements are 9 we need add another element to the array
//for example: [9,9] --> [1,0,0]
```

💡 **Q5.** You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

Example 1: Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3 Output: [1,2,2,3,5,6]

Explanation: The arrays we are merging are [1,2,3] and [2,5,6]. The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

Sol:-

```
// Time Complexity: O(m+n)
// SPace COmplexity: O(1)
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        //variables to work as pointers
        int i=m-1; // will point at m-1 index of nums1 array
        int j=n-1; // will point at n-1 index of nums2 array
        int k=nums1.length-1; //will point at the last position of the nums1 array

        // Now traversing the nums2 array
        while(j>=0){
            // If element at i index of nums1 > element at j index of nums2
            // then it is largest among two arrays and will be stored at k position of
nums1
            // using i>=0 to make sure we have elements to compare in nums1 array
            if(i>=0 && nums1[i]>nums2[j]){
                nums1[k]=nums1[i];
                k--;
```

```
                i--; //updating pointer for further comparisons
            }else{
                // element at j index of nums2 array is greater than the element at i
index of nums1 array
                // or there is no element left to compare with the nums1 array
                // and we just have to push the elements of nums2 array in the nums1
array.
                nums1[k] = nums2[j];
                k--;
                j--; //updating pointer for further comparisons
            }
        }
    }
}
```

💡 **Q6.** Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

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

Output: true

Sol:-

```
// Time Complexity: O(n)
// Space Complexity: O(n)
class Solution {
    public boolean containsDuplicate(int[] nums) {
        HashMap<Integer,Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            if (map.containsKey(nums[i])) {
                return true;
            }
            map.put(nums[i],1);
        }
        return false;
    }
}
```

💡 **Q7.** Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the nonzero 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]

Sol:-

```
// Time Complexity: O(n)
// Space Complexity: O(1)
class Solution {
    public void moveZeroes(int[] nums) {
        int n = 0;
        for(int i = 0; i<nums.length;i++){
            if(nums[i]==0){
                n++;
            }
            else if(n > 0){
```

```
                int temp = nums[i];
                nums[i] = 0;
                nums[i-n] = temp;
            }
        }
    }
}
```

💡 **Q8.** You have a set of integers s, which originally contains all the numbers from 1 to n. Unfortunately, due to some error, one of the numbers in s got duplicated to another number in the set, which results in repetition of one number and loss of another number.

You are given an integer array nums representing the data status of this set after the error.

Find the number that occurs twice and the number that is missing and return them in the form of an array.

Example 1: Input: nums = [1,2,2,4] Output: [2,3]

Sol:-

```
// Time COmplexity: O(n)
// SPacr Complexity: O(n)
public class Solution {
    public int[] findErrorNums(int[] nums) {
        int[] arr = new int[nums.length + 1];
        int dup = -1, missing = 1;
        for (int i = 0; i < nums.length; i++) {
            arr[nums[i]] += 1;
        }
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] == 0)
                missing = i;
            else if (arr[i] == 2)
                dup = i;
        }
        return new int[]{dup, missing};
    }
}
```