Arrays: Two Pointer Approach

The two-pointer technique is a method used to solve problems on arrays or lists by using two pointers to traverse the data structure

1.Two Sum:
Problem Link: https://leetcode.com/problems/two-sum/description/
Level:Easy

Solution1:
Code:
```
var twoSum = function(nums, target) {
     var array=[];
     for(let i=0;i<nums.length;i++){
        for(let j=i+1;j<nums.length;j++){
            if(nums[i]+nums[j]==target){
                array.push(i);
                array.push(j);
            }
        }
     }
     return array;
};
```
Time Complexity:O(n^2)
Space Complexity:O(1) excluding the answer array space

Solution 2:
Code:

```
var twoSum = function(nums, target) {
    var array=[];
    var mp=new Map();
    for(let i=0;i<nums.length;i++){
        let difference=(target-nums[i]);
        if(mp.has(difference)){
            array.push(mp.get(difference));
            array.push(i);
            return array;
        }
        else{
            mp.set(nums[i],i);
        }
    }
    return array;
};
```

Time Complexity:O(n)
Space Complexity:O(n)

2. Sort Colors:
Level:Easy
Problem Link:https://leetcode.com/problems/sort-colors/

Level:Medium
Solution1:
Code:

```
var sortColors = function(nums) {
    var zeroes=[];
    var ones=[];
    var twos=[];
    for(let i=0;i<nums.length;i++){
        if(nums[i]===0){
            zeroes.push(nums[i]);
        }
        else if(nums[i]===1){
            ones.push(nums[i]);
        }
        else if(nums[i]===2){
            twos.push(nums[i]);
        }
    }
    var ansArray = zeroes.concat(ones, twos);
    return ansArray;
};
```

Time Complexity:O(n)
Space Complexity:O(n)

Solution2:
Code:

```
var sortColors = function(arr) {
    var low=0;
    var mid=0;
    var high=arr.length-1;
    while(mid<=high){
        if(arr[mid]===0){
             [arr[low], arr[mid]] = [arr[mid], arr[low]];
             low++;
             mid++;
         }
         else if(arr[mid]===1){
            mid++;
         }
         else{
             [arr[mid], arr[high]] = [arr[high], arr[mid]]
             high--;
         }
     }
     return arr;
};
```

Time Complexity:O(n)
Space Complexity:O(1)

Solution3:
Code:

```
var sortColors = function(nums) {
    nums.sort((a,b)=>(a-b))
    return nums;
};
```

Time Complexity:O(n)
Space Complexity:O(n)

3.Find Common Elements Between Two Arrays:
Problem
Level:Easy
Link:https://leetcode.com/problems/find-common-elements-between-two-arrays/description/

Solution1:
Code:

```
var findIntersectionValues = function(nums1, nums2) {
    var array=[];
    let count1=0;
    let count2=0;
    for(let i=0;i<nums1.length;i++){
        var flag=false;
        for(let j=0;j<nums2.length;j++){
            if(nums1[i]==nums2[j]){
                flag=true;
            }
        }
        if(flag==true){
            count1++;
        }
    }
    for(let i=0;i<nums2.length;i++){
        var flag=false;
        for(let j=0;j<nums1.length;j++){
            if(nums2[i]==nums1[j]){
                flag=true;
            }
        }
        if(flag==true){
            count2++;
        }
    }
    array.push(count1);
    array.push(count2);
    return array;
};
```

Time Complexity:O(n1*n2)

Space complexity:O(1) excluding the answer array space

Solution2:

Code:

```
var findIntersectionValues = function(nums1, nums2) {
    var array=[];
    var set1=new Set();
    var set2=new Set();

    for(let i=0;i<nums1.length;i++){
        set1.add(nums1[i]);
    }

    for(let i=0;i<nums2.length;i++){
        set2.add(nums2[i]);
    }

    var count1=0;
    var count2=0;

    for(let i=0;i<nums1.length;i++){
        if(set2.has(nums1[i])){
            count1++;
        }
    }
    for(let i=0;i<nums2.length;i++){
        if(set1.has(nums2[i])){
            count2++;
        }
    }
    array.push(count1);
    array.push(count2);
    return array;
};
Time Complexity:O(n1)+O(n2)
Space complexity:O(nums1.length)+O(nums2.length)
```

4.No. of good pairs

5.Merge two Sorted arrays