PDS_MID_1_ANSWERS

1.	Maximum SubArray, Kadanes's Algorithm	[Easy]
	Given an integer array nums, find the subarray with the largest sum, and return its sum. Example 1: Input: nums = [-2,1,-3,4,-1,2,1,-5,4] Output: 6 Explanation: The subarray [4,-1,2,1] has the largest sum 6. Example 2: Input: nums = [1] Output: 1 Explanation: The subarray [1] has the largest sum 1. Example 3: Input: nums = [5,4,-1,7,8] Output: 23 Explanation: The subarray [5,4,-1,7,8] has the largest sum 23. Constraints: ● 1 <= nums.length <= 105 ● -104 <= nums[i] <= 104 Follow up: If you have figured out the O(n) solution, try coding another solution using the divide and conquer approach, which is more subtle. 32	

Practice link: https://leetcode.com/problems/maximum-subarray/description/
Solution :

```
class Solution {
    public int maxSubArray(int[] nums) {
        int[] prefixArray = new int[nums.length];
        prefixArray[0] = nums[0];
        int maximumSum = prefixArray[0];
        for (int i = 1; i < nums.length; i++) {
            if (prefixArray[i - 1] < 0) {
                prefixArray[i] = nums[i];
            } else {
                prefixArray[i] = prefixArray[i - 1] + nums[i];
            }
            maximumSum = Math.max(maximumSum, prefixArray[i]);
        }

        return maximumSum;
    }
}
```

2.	**Move Zeros**	[Eas y]
	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] Constraints: • $1 <= nums.length <= 10^4$ • $-2^{31} <= nums[i] <= 2^{31} - 1$	

Practice link:
https://leetcode.com/problems/move-zeroes/description/
Solution:

```
class Solution {
    public void moveZeroes(int[] nums) {
        int left = 0;

        for (int right = 0; right < nums.length; right++) {
            if (nums[right] != 0) {
                int temp = nums[right];
                nums[right] = nums[left];
                nums[left] = temp;
                left++;
            }
        }
    }
}
```

3.

Missing Number

[Easy]

Given an array *‘a’* of size *‘n’*-1 with elements of range 1 to ‘n’. The array does not contain any duplicates. Your task is to find the missing number.

For example:
Input:
'a' = [1, 2, 4, 5], 'n' = 5

Output :
3

Explanation: 3 is the missing value in the range 1 to 5.
Detailed explanation (Input/output format, Notes, Images)
Sample Input 1 :
4
1 2 3
Sample Output 1:
4
Explanation Of Sample Input 1:
4 is the missing value in the range 1 to 4.
Sample Input 2:
8
1 2 3 5 6 7 8
Sample Output 2:
4
Explanation Of Sample Input 2:
4 is the missing value in the range 1 to 8.
Expected time complexity:
The expected time complexity is O(n).
Constraints:
1 <= 'n' <= 10^6
1 <= 'a'[i] <= 'n'
Time Limit: 1 sec

Practice link:
https://www.naukri.com/code360/problems/missing-number_6680467?leftPanelTabValue=DISCUSS

Solution:

```
public class Solution {
    public static int missingNumber(int []a, int N) {
        int hash[]=new int[N+1];
        int idx=0;
        for(int i=0;i<N-1;i++){
            hash[a[i]]=1;
        }
        for(int i=1;i<=N;i++){
            if(hash[i]==0){
                idx=i;
                break;
            }
        }
        return idx;
    }
}
```

4.	Two Sum	[Easy]
	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] Constraints:	

2 <= nums.length <= 104
-109 <= nums[i] <= 109
-109 <= target <= 109
Only one valid answer exists.

Follow-up: Can you come up with an algorithm that is less than O(n2) time complexity?

Practice link:
https://leetcode.com/problems/two-sum/description/

Solution:

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        int n = nums.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                if (nums[i] + nums[j] == target) {
                    return new int[]{i, j};
                }
            }
        }
        return new int[]{}; // No solution found
    }
}
```

5.

Remove Duplicates

[Easy]

You are given a sorted integer array 'arr' of size 'n'.
You need to remove the duplicates from the array such that each element appears only once.
Return the length of this new array.
Note:
Do not allocate extra space for another array. You need to do this by modifying the given input array in place with O(1) extra memory.
For example:

	'n' = 5, 'arr' = [1 2 2 2 3]. The new array will be [1 2 3]. So our answer is 3. Sample input 1: 10 1 2 2 3 3 3 4 4 5 5 Sample output 1: 5 Explanation of sample input 1: The new array will be [1 2 3 4 5]. So our answer is 5. Sample input 2: 9 1 1 2 3 3 4 5 5 5 Sample output 2: 5 Expected time complexity: The expected time complexity is O(n). Constraints : 1 <= 'n' <= 10^6 -10^9 <= 'arr[i]' <=10^9 Where 'arr[i]' is the value of elements of the array.

Practice link:
https://www.naukri.com/code360/problems/remove-duplicates-from-sorted-array_1102307?leftPanelTabValue=DISCUSS

Solution :

```
public class Solution {
    public static int removeDuplicates(int[] arr,int n) {
    // Write your code here.
    int i=0;
        for(int j=0; j<n; j++){
            if(arr[i]!= arr[j]){
                i++;
                arr[i]= arr[j];
            }else{
            continue;
        }
    }
    return i+1;
    }
}
```

6.	SubArray Sum Equals K	[Medium]
	Given an array of integers nums and an integer k, return the total number of subarrays whose sum equals to k. A subarray is a contiguous non-empty sequence of elements within an array. Example 1: Input: nums = [1,2,3], k = 3 Output: 2 Example 2: Input: nums = [1,2,3,-3, 1,1,1,4,2,-3], k = 3 Output: 2 Constraints: 1 <= nums.length <= 2 * 104 -1000 <= nums[i] <= 1000 -107 <= k <= 107	

Practice link:
https://leetcode.com/problems/subarray-sum-equals-k/description/

Solution :

```java
class Solution {
    public int subarraySum(int[] nums, int k) {
        int n=nums.length;
        int count=0;
        for(int i=0;  i<n;  i++){
            int sum=0;
            for(int j=i;  j<n;  j++){
                sum+=nums[j];//one subarray sum
                if(sum==k){
                    count++;
                }
            }
        }
        return count;
    }
}
```

7.

Check Sorted Array

[Easy]

Problem statement

Send feedback

You have been given an array *‘a’* of *‘n’* non-negative integers.You have to check whether the given array is sorted in the non-decreasing order or not.

Your task is to return 1 if the given array is sorted. Else, return 0.

Example :
Input: ‘n’ = 5, ‘a’ = [1, 2, 3, 4, 5]
Output: 1

The given array is sorted in non-decreasing order; hence the answer will be 1.
Detailed explanation (Input/output format, Notes, Images)
Sample Input 1 :
4
0 0 0 1
Sample Output 1 :
1
Explanation For Sample Input 1 :
The given array is sorted in non-decreasing order; hence the answer will be 1.
Sample Input 2 :
5
4 5 4 4 4
Sample Output 2 :
0
Expected Time Complexity:
O(n), Where ‘n’ is the size of an input array ‘a’.
Constraints:
1 ≤ 'n' ≤ 5*10^6
0 ≤ 'a'[i] ≤ 10^9

Time limit: 1 sec

practice link:
https://www.naukri.com/code360/problems/ninja-and-the-sorted-check_6581957?leftPanelTabValue=SOLUTION

Solution:

```
public class Solution {
    public static int isSorted(int n, int []a) {
        for(int i=1; i<n; i++){
            if (a[i] >= a[i-1]) {
            } else {
                return 0;
            }
        }
        return 1;
    }
}
```

8.	Sort Colors	[Medium]
	Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue. We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively. You must solve this problem without using the library's sort function. Example 1: Input: nums = [2,0,2,1,1,0] Output: [0,0,1,1,2,2] Example 2: Input: nums = [2,0,1] Output: [0,1,2] Constraints: • n == nums.length • 1 <= n <= 300 • nums[i] is either 0, 1, or 2. Follow up: Could you come up with a one-pass algorithm using only constant extra space?	

Practice link:
https://leetcode.com/problems/sort-colors/description/

Solution:

```
class Solution {
    public void sortColors(int[] nums) {
        int countRed=0;
        int countWhite=0;
        int countBlue=0;
        for(int i=0;i<nums.length;i++){
            if(nums[i]==0){countRed++;}
            if(nums[i]==1){countWhite++;}
            if(nums[i]==2){countBlue++;}
        }
        //modify the nums-array
        int i=0;
            while(countRed!=0){
                nums[i++]=0;countRed--;
            }
            while(countWhite!=0){
                nums[i++]=1;countWhite--;
            }
            while(countBlue!=0){
                nums[i++]=2;countBlue--;
            }
    }
}
```

9.	Maximum Consecutive Ones
	Given a binary array nums, return *the maximum number of consecutive* 1*'s in the array*. Example 1: Input: nums = [1,1,0,1,1,1] Output: 3

	Explanation: The first two digits or the last three digits are consecutive 1s. The maximum number of consecutive 1s is 3. Example 2: Input: nums = [1,0,1,1,0,1] Output: 2 Constraints: • $1 <= nums.length <= 10^5$ • nums[i] is either 0 or 1.

Practice link:https://leetcode.com/problems/max-consecutive-ones/description/

Solution:

```java
class Solution {
    public int findMaxConsecutiveOnes(int[] arr) {
        int max = 0;

        int count = 0;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == 0) {
                count = 0;
            } else {
                count++;
            }
            if (count > max) {
                max = count;
            }
        }
        return max;
    }
}
```

10.	**Find the number that appears ones and other numbers twice**	[Easy]
	Given a non-empty array of integers nums, every element appears *twice* except for one. Find that single one. You must implement a solution with a linear runtime complexity and use only constant extra space.	

	Example 1: Input: nums = [2,2,1] Output: 1 Example 2: Input: nums = [4,1,2,1,2] Output: 4 Example 3: Input: nums = [1] Output: 1 Constraints: • 1 <= nums.length <= 3 * 10^4 • -3 * 10^4 <= nums[i] <= 3 * 10^4 • Each element in the array appears twice except for one element which appears only once.

Practice link:https://leetcode.com/problems/single-number/description/

Solution:

```
class Solution {
    public int singleNumber(int[] nums) {
        int num=0;
        for(int number:nums){
            num=num^number;
        }
        return num;
    }
}
```

11.	Longest Subarray with given Sum K(Positives)	[Medium]
	Problem statement Send feedback You are given an array *'a'* of size *'n'* and an integer *'k'*. Find the length of the longest subarray of 'a' whose sum is equal to 'k'. Example :	

Input: ‘n’ = 7 ‘k’ = 3
‘a’ = [1, 2, 3, 1, 1, 1, 1]

Output: 3

Explanation: Subarrays whose sum = ‘3’ are:

[1, 2], [3], [1, 1, 1] and [1, 1, 1]

Here, the length of the longest subarray is 3, which is our final answer.
Detailed explanation (Input/output format, Notes, Images)
Sample Input 1 :
7 3
1 2 3 1 1 1 1

Sample Output 1 :
3

Explanation Of Sample Input 1 :
Subarrays whose sum = ‘3’ are:
[1, 2], [3], [1, 1, 1] and [1, 1, 1]
Here, the length of the longest subarray is 3, which is our final answer.

Sample Input 2 :
4 2
1 2 1 3

Sample Output 2 :
1

Sample Input 3 :
5 2
2 2 4 1 2

Sample Output 3 :
1

Expected time complexity :
The expected time complexity is O(n).

Constraints :
1 <= 'n' <= 5 * 10 ^ 6
1 <= 'k' <= 10^18

	0 <= 'a[i]' <= 10 ^ 9 Time Limit: 1-second

Practice link:https://www.naukri.com/code360/problems/longest-subarray-with-sum-k_6682399?leftPanelTabValue=DISCUSS

Solution:

```java
public class Solution {
    public static int longestSubarrayWithSumK(int []a, long k) {
        int max=0;
        int n=a.length;
        for(int i=0;i<n;i++){
            long sum=0;
            for(int j=i;j<n;j++){
                sum+=a[j];
                if(sum==k) max=Math.max(max,j-i+1);
            }
        }
        return max;
    }
}
```

12.	**Merge 2 Sorted Array**	[Medium]
	Problem statement Send feedback Given two sorted arrays, *'a'* and *'b'*, of size *'n'* and *'m'*, respectively, return the union of the arrays. The union of two sorted arrays can be defined as an array consisting of the common and the distinct elements of the two arrays. The final array should be sorted in ascending order. Note: 'a' and 'b' may contain duplicate elements, but the union array must contain unique elements. Example:	

Input: ‘n’ = 5 ‘m’ = 3
‘a’ = [1, 2, 3, 4, 6]
‘b’ = [2, 3, 5]

Output: [1, 2, 3, 4, 5, 6]

Explanation: Common elements in ‘a’ and ‘b’ are: [2, 3]
Distinct elements in ‘a’ are: [1, 4, 6]
Distinct elements in ‘b’ are: [5]
Union of ‘a’ and ‘b’ is: [1, 2, 3, 4, 5, 6]
Detailed explanation (Input/output format, Notes, Images)
Sample Input 1 :
5 3
1 2 3 4 6
2 3 5
Sample Output 1 :
1 2 3 4 5 6
Explanation Of Sample Input 1 :
Input: ‘n’ = 5 ‘m’ = 3
‘a’ = [1, 2, 3, 4, 6]
‘b’ = [2, 3, 5]

Output: [1, 2, 3, 4, 5, 6]

Explanation: Common elements in ‘a’ and ‘b’ are: [2, 3]
Distinct elements in ‘a’ are: [1, 4, 6]
Distinct elements in ‘b’ are: [5]
Union of ‘a’ and ‘b’ is: [1, 2, 3, 4, 5, 6]
Sample Input 2:
4 3
1 2 3 3
2 2 4
Sample Output 2:
1 2 3 4
Explanation Of Sample Input 2 :
Input: ‘n’ = 5 ‘m’ = 3
‘a’ = [1, 2, 3, 3]
‘b’ = [2, 2, 4]

Output: [1, 2, 3, 4]

Explanation: Common elements in ‘a’ and ‘b’ are: [2]
Distinct elements in ‘a’ are: [1, 3]

	Distinct elements in ‘b’ are: [4] Union of ‘a’ and ‘b’ is: [1, 2, 3, 4] Expected Time Complexity: O((N + M)), where 'N' and ‘M’ are the sizes of Array ‘A’ and ‘B’. Constraints : 1 <= 'n', 'm' <= 10^5 -10^9 <= 'a'[i], 'b'[i] <= 10^9 Time Limit: 1 sec

Practice link: https://www.naukri.com/code360/problems/sorted-array_6613259

Solution:

```java
import java.util.*;
public class Solution {
    public static List<Integer> sortedArray(int[] a, int[] b) {
        ArrayList<Integer> union = new ArrayList<>();
        int i = 0;
        int j = 0;
        int n = a.length;
        int m = b.length;
        while (i < n && j < m) {
            if (a[i] <= b[j]) {
                addUnique(union, a[i]);
                i++;
            } else {
                addUnique(union, b[j]);
                j++;
            }
        }
        while (i < n) {
            addUnique(union, a[i]);
            i++;
        }
        while (j < m) {
            addUnique(union, b[j]);
            j++;
        }
        return union;
    }
    private static void addUnique(List<Integer> list, int num) {
        if (list.size() == 0 || list.get(list.size() - 1) != num) {
            list.add(num);
        }
    }
}
```

13.	Rotate Array by k elements	[Easy]

Given an integer array nums, rotate the array to the right by k steps, where k is non-negative.

Example 1:
Input: nums = [1,2,3,4,5,6,7], k = 3
Output: [5,6,7,1,2,3,4]
Explanation:
rotate 1 steps to the right: [7,1,2,3,4,5,6]
rotate 2 steps to the right: [6,7,1,2,3,4,5]
rotate 3 steps to the right: [5,6,7,1,2,3,4]
Example 2:
Input: nums = [-1,-100,3,99], k = 2
Output: [3,99,-1,-100]
Explanation:
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]

Constraints:
- 1 <= nums.length <= 10^5
- -2^{31} <= nums[i] <= 2^{31} - 1
- 0 <= k <= 10^5

Follow up:
- Try to come up with as many solutions as you can. There are at least three different ways to solve this problem.
- Could you do it in-place with O(1) extra space?

Practice link:https://leetcode.com/problems/rotate-array/

Solution:

```java
class Solution {
    public void rotate(int[] nums, int k) {
        k %= nums.length;
        reverse(nums, 0, nums.length-1);
        reverse(nums, 0, k-1);
        reverse(nums, k, nums.length-1);
    }

    public void reverse(int[] nums, int start, int end){
        while(start < end){
            int temp = nums[start];
            nums[start] = nums[end];
            nums[end] = temp;
            start++;
            end--;
        }
    }
}
```

14.	Remove Duplicates from Sorted Array	[Easy]

	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).
	
	Constraints:
	• 1 <= nums.length <= 3 * 10^4
	• -100 <= nums[i] <= 100
	• nums is sorted in non-decreasing order.

Practice link:https://leetcode.com/problems/remove-duplicates-from-sorted-array/solutions/

Solution:

```java
class Solution {
    public int removeDuplicates(int[] nums) {
        int j = 1;
        for (int i = 1; i < nums.length; i++) {
            if (nums[i] != nums[i - 1]) {
                nums[j] = nums[i];
                j++;
            }
        }
        return j;
    }
}
```

15.	Rearrange Array Elements by Sign	[Medium]
	You are given a 0-indexed integer array nums of even length consisting of an equal number of positive and negative integers. You should return the array of nums such that the the array follows the given conditions: 1. Every consecutive pair of integers have opposite signs. 2. For all integers with the same sign, the order in which they were present in nums is preserved. 3. The rearranged array begins with a positive integer. Return *the modified array after rearranging the elements to satisfy the aforementioned conditions*. Example 1: Input: nums = [3,1,-2,-5,2,-4] Output: [3,-2,1,-5,2,-4] Explanation: The positive integers in nums are [3,1,2]. The negative integers are [-2,-5,-4]. The only possible way to rearrange them such that they satisfy all conditions is [3,-2,1,-5,2,-4]. Other ways such as [1,-2,2,-5,3,-4], [3,1,2,-2,-5,-4], [-2,3,-5,1,-4,2] are incorrect because they do not satisfy one or more conditions. Example 2: Input: nums = [-1,1] Output: [1,-1] Explanation: 1 is the only positive integer and -1 the only negative integer in nums. So nums is rearranged to [1,-1]. Constraints: • 2 <= nums.length <= 2 * 10^5 • nums.length is even • 1 <= |nums[i]| <= 10^5 • nums consists of equal number of positive and negative integers. It is not required to do the modifications in-place.	

Practice link:https://leetcode.com/problems/rearrange-array-elements-by-sign/solutions/

Solution:

```
class Solution {
    public int[] rearrangeArray(int[] nums) {
      int res[] = new int[nums.length];
      int pos =0, neg =1;
      for(int i=0;i<nums.length;i++){
          if(nums[i]>=0){
              res[pos] = nums[i];
              pos+=2;
          }
          else{
              res[neg] = nums[i];
              neg+=2;
          }
      }
      return res;
    }
}
```

16.	Best Time to Buy and Sell Stock	[Easy]
	You are given an array prices where prices[i] is the price of a given stock on the ith day. You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return 0. Example 1: Input: prices = [7,1,5,3,6,4] Output: 5 Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5. Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell. Example 2: Input: prices = [7,6,4,3,1] Output: 0 Explanation: In this case, no transactions are done and the max profit = 0. Constraints: • 1 <= prices.length <= 10^5	

	• 0 <= prices[i] <= 10^4

Practice link:https://leetcode.com/problems/best-time-to-buy-and-sell-stock/description/

Solution:

```java
class Solution {
    public int maxProfit(int[] prices) {
        int buyPrice = prices[0];
        int profit = 0;

        for (int i = 1; i < prices.length; i++) {
            if (buyPrice > prices[i]) {
                buyPrice = prices[i];
            }

            profit = Math.max(profit, prices[i] - buyPrice);
        }

        return profit;
    }
}
```

17.	3 Sum
	Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0. Notice that the solution set must not contain duplicate triplets. Example 1: Input: nums = [-1,0,1,2,-1,-4] Output: [[-1,-1,2],[-1,0,1]] Explanation: nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0. nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0. nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0. The distinct triplets are [-1,0,1] and [-1,-1,2]. Notice that the order of the output and the order of the triplets does not matter. Example 2: Input: nums = [0,1,1] Output: []

	Explanation: The only possible triplet does not sum up to 0. Example 3: Input: nums = [0,0,0] Output: [[0,0,0]] Explanation: The only possible triplet sums up to 0. Constraints: • $3 <= nums.length <= 3000$ • $-10^5 <= nums[i] <= 10^5$

Practice link:https://leetcode.com/problems/3sum/solutions/

Solution:

```java
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        Arrays.sort(nums);
        for (int i = 0; i < nums.length; i++) {
            if (i > 0 && nums[i] == nums[i-1]) {
                continue;
            }
            int j = i + 1;
            int k = nums.length - 1;
            while (j < k) {
                int total = nums[i] + nums[j] + nums[k];
                if (total > 0) {
                    k--;
                } else if (total < 0) {
                    j++;
                } else {
                    res.add(Arrays.asList(nums[i], nums[j], nums[k]));
                    j++;
                    while (nums[j] == nums[j-1] && j < k) {
                        j++;
                    }
                }
            }
        }
        return res;
    }
}
```

18.	4 Sum	
	Given an array nums of n integers, return *an array of all the unique quadruplets* [nums[a], nums[b], nums[c], nums[d]] such that: • 0 <= a, b, c, d < n • a, b, c, and d are distinct. • nums[a] + nums[b] + nums[c] + nums[d] == target You may return the answer in any order. Example 1: Input: nums = [1,0,-1,0,-2,2], target = 0 Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]] Example 2: Input: nums = [2,2,2,2,2], target = 8 Output: [[2,2,2,2]] Constraints: • 1 <= nums.length <= 200 • -10^9 <= nums[i] <= 10^9 • -10^9 <= target <= 10^9	

Practice link:https://leetcode.com/problems/4sum/description/

Solution:

```
class Solution {
    public List<List<Integer>> fourSum(int[] nums, int target) {
        if (nums == null || nums.length < 4) {
            return new ArrayList<>();
        }
        Arrays.sort(nums);
        HashSet<List<Integer>> result = new HashSet<>();
        for (int i = 0; i < nums.length - 3; i++) {
            for (int j = i + 1; j < nums.length - 2; j++) {
                int left = j + 1;
                int right = nums.length - 1;
                while (left < right) {
                    long sum = (long) nums[i] + (long) nums[j] + (long)
nums[left] + (long) nums[right];
                    if (sum == target) {
                        result.add(Arrays.asList(nums[i], nums[j], nums[left],
nums[right]));
                        left++;
                        right--;
                        }
                    else if (sum < target) {
                        left++;
                    }
                    else {
                        right--;
                    }
                }
            }
        }
        return new ArrayList<>(result);
    }
}
```

19.	Longest subarray with 0 sum	[Medium]
	given an array *‘Arr’* of size *‘N’*. You have to help him find the longest subarray of ‘Arr’, whose sum is 0. You must return the length of the longest subarray whose sum is 0. For Example: For N = 5, and Arr = {1, -1, 0, 0, 1}, We have the following subarrays with zero sums: {{1, -1}, {1, -1, 0}, {1, -1, 0, 0}, {-1, 0, 0, 1}, {0}, {0, 0}, {0}} Among these subarrays, {1, -1, 0, 0} and {-1, 0, 0, 1} are the longest subarrays with their sum equal to zero. Hence the answer is 4. Detailed explanation (Input/output format, Notes, Images) Sample Input 1: 4 1 0 -1 1 Sample Output 1: 3 Explanation of Sample Input 1: The subarrays with sums equal to zero are: {{1, 0, -1}, {0}, {0, -1, 1}, {-1, 1}}. Among these, {1, 0, -1} and {0, -1, 1} are the longest with length equal to 3.	

Hence the answer is 3.
Sample Input 2:
2
1 1
Sample Output 2:
0
Constraints:
1 <= N <= 10^5
-10^9 <= Arr[i] <= 10^9

The sum of ‘N’ over all test cases is less than or equal to 10^5.
Time Limit: 1 sec.

Practice link:
https://www.naukri.com/code360/problems/longest-subarray-with-zero-sum_6783450

Solution:

```
public class Solution {
    public static int getLongestZeroSumSubarrayLength(int[] arr) {
        int max = 0;
        int n = arr.length;
        for (int i = 0; i < n; i++) {
            int sum = 0;
            for(int j = i; j < n; j++){
                sum += arr[j];
                if(sum == 0){
                    max = Math.max(max, j-i+1);
                }
            }
        }
        return max;
    }
}
```

20.	Count the number of subarrays with given xor K	[Medium]
	Given an array *‘A’* consisting of *‘N’* integers and an integer *‘B’*, find the number of subarrays of array ‘A’ whose bitwise XOR(⊕) of all elements is equal to ‘B’. A subarray of an array is obtained by removing some(zero or more) elements from the front and back of the array.	

Example:
Input: ‘N’ = 4 ‘B’ = 2
‘A’ = [1, 2, 3, 2]

Output: 3

Explanation: Subarrays have bitwise xor equal to ‘2’ are: [1, 2, 3, 2], [2], [2].
Detailed explanation (Input/output format, Notes, Images)
Sample Input 1:
4 2
1 2 3 2
Sample Output 1 :
3
Explanation Of Sample Input 1:
Input: ‘N’ = 4 ‘B’ = 2
‘A’ = [1, 2, 3, 2]

Output: 3

Explanation: Subarrays have bitwise xor equal to ‘2’ are: [1, 2, 3, 2], [2], [2].
Sample Input 2:
4 3
1 2 3 3
Sample Output 2:
4
Sample Input 3:
5 6
1 3 3 3 5
Sample Output 3:
2
Constraints:
1 <= N <= 10^3
1 <= A[i], B <= 10^9

Time Limit: 1-sec

Practice link:https://www.naukri.com/code360/problems/subarrays-with-xor-k_6826258?leftPanelTabValue=PROBLEM

Solution:

```
import java.util.*;
```

```java
public class Solution {
    public static int subarraysWithSumK(int[] a, int b) {
        // Write your code here
        Map<Integer, Integer> freqMap = new HashMap<>();
        int count = 0;
        int xor = 0;
        for (int i = 0; i < a.length; i++) {
            xor ^= a[i];
            if (xor == b)
                count++;
            if (freqMap.containsKey(xor ^ b)) {
                count += freqMap.get(xor ^ b);
            }
            freqMap.put(xor, freqMap.getOrDefault(xor, 0) + 1);
        }
        return count;
    }
}
```

21.	Majority Element
	Given an array nums of size n, return *the majority element*. The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array. Example 1: Input: nums = [3,2,3] Output: 3 Example 2: Input: nums = [2,2,1,1,1,2,2] Output: 2 Constraints: • n == nums.length • 1 <= n <= 5 * 10^4 • -10^9 <= nums[i] <= 10^9 Follow-up: Could you solve the problem in linear time and in O(1) space?

Practice link:https://leetcode.com/problems/majority-element/description/

Solution:

```
class Solution {
    public int majorityElement(int[] nums) {
        Arrays.sort(nums);
        int n = nums.length;
        return nums[n/2];
    }
}
```

22.	Second Largest Number	[Easy]
	you have been given an array *'a'* of *'n'* unique non-negative integers. Find the second largest and second smallest element from the array. Return the two elements (second largest and second smallest) as another array of size 2. Example : Input: 'n' = 5, 'a' = [1, 2, 3, 4, 5] Output: [4, 2] The second largest element after 5 is 4, and the second smallest element after 1 is 2. Detailed explanation (Input/output format, Notes, Images) Sample Input 1 : 4 3 4 5 2 Sample Output 1 : 4 3 Explanation For Sample Input 1 : The second largest element after 5 is 4 only, and the second smallest element after 2 is 3. Sample Input 2 : 5 4 5 3 6 7 Sample Output 2 : 6 4 Expected Time Complexity:	

	O(n), Where ‘n’ is the size of an input array ‘a’. Constraints: 2 ≤ 'n' ≤ 10^5 0 ≤ 'a'[i] ≤ 10^9 Time limit: 1 sec Hints: 1. Sort the array. 2. More efficiently, can you use the largest and smallest elements to find the required elements?

Practice link:https://www.naukri.com/code360/problems/ninja-and-the-second-order-elements_6581960?leftPanelTabValue=PROBLEM

Solution:

```
import java.util.*;
public class Solution {
    public static int[] getSecondOrderElements(int n, int []a) {
        int  Slargest=slargest( n, a);
        int Ssmallest=ssmallest( n, a);
        return new int[]{Slargest, Ssmallest};
    }
    public static int slargest(int n , int arr[]){
        int Largest=Integer.MIN_VALUE;
        int Slargest=Integer.MIN_VALUE;
         for(int i=0;i<n;i++){
             if(arr[i]>Largest){
                 Slargest=Largest;
                 Largest=arr[i];
             }
             else if(Slargest<arr[i] && arr[i]!=Largest) {
                 Slargest=arr[i];
             }
         }
         return Slargest;
    }
    public static int ssmallest(int n,int arr[]){
         int smallest=Integer.MAX_VALUE;
        int Ssmallest=Integer.MAX_VALUE;
        for(int i=0;i<n;i++){
            if(arr[i]<smallest){
                Ssmallest=smallest;
                smallest=arr[i];
            }
            else if(Ssmallest>arr[i] && arr[i]!=smallest){
                Ssmallest=arr[i];
            }
        }
        return Ssmallest;
    }
}
```

23.	Rotate Array	[Easy]
	Given an integer array nums, rotate the array to the right by k steps, where k is non-negative. Example 1: Input: nums = [1,2,3,4,5,6,7], k = 3 Output: [5,6,7,1,2,3,4] Explanation: rotate 1 steps to the right: [7,1,2,3,4,5,6] rotate 2 steps to the right: [6,7,1,2,3,4,5] rotate 3 steps to the right: [5,6,7,1,2,3,4] Example 2: Input: nums = [-1,-100,3,99], k = 2 Output: [3,99,-1,-100] Explanation: rotate 1 steps to the right: [99,-1,-100,3] rotate 2 steps to the right: [3,99,-1,-100] Constraints: • 1 <= nums.length <= 10^5 • -2^{31} <= nums[i] <= 2^{31} - 1 • 0 <= k <= 10^5	

	Follow up: • Try to come up with as many solutions as you can. There are at least three different ways to solve this problem. • Could you do it in-place with O(1) extra space?

Practice link:https://leetcode.com/problems/rotate-array/description/

Solution:

```
class Solution {
    public void rotate(int[] nums, int k) {
        k %= nums.length;
        reverse(nums, 0, nums.length-1);
        reverse(nums, 0, k-1);
        reverse(nums, k, nums.length-1);
    }

    public void reverse(int[] nums, int start, int end){
        while(start < end){
            int temp = nums[start];
            nums[start] = nums[end];
            nums[end] = temp;
            start++;
            end--;
        }
    }
}
```

24.	Last occurrence in a sorted array	[Medium]
	Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value. If target is not found in the array, return [-1, -1]. You must write an algorithm with O(log n) runtime complexity. Example 1: Input: nums = [5,7,7,8,8,10], target = 8 Output: [3,4] Example 2: Input: nums = [5,7,7,8,8,10], target = 6 Output: [-1,-1] Example 3:	

	Input: nums = [], target = 0 Output: [-1,-1] Constraints: • $0 <= nums.length <= 10^5$ • $-10^9 <= nums[i] <= 10^9$ • nums is a non-decreasing array. • $-10^9 <= target <= 10^9$

Practice link:https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/description/

Solution:

```
class Solution {
// sarch from beginning
    public int searchindfornt(int[] arr,int target){
        for(int i=0;i<arr.length;i++){
            if(arr[i]==target){
                return i;
            }
        }
        return -1;
    }
// search from end
    public int searchindend(int[] arr,int target){
        for(int i=arr.length-1;i>=0;i--){
            if(arr[i]==target){
                return i;
            }
        }
        return -1;
    }
  public int[] searchRange(int[] nums, int target) {
        int x=searchindfornt(nums,  target);
        int y=searchindend(nums,  target);
        int[] ans={x,y};
        return ans;
    }
}
```

25.	Monk and his Friends Problem	[Easy]

	Monk is standing at the door of his classroom. There are currently N students in the class, i'th student got A_i candies. There are still M more students to come. At every instant, a student enters the class and wishes to be seated with a student who has exactly the same number of candies. For each student, Monk shouts YES if such a student is found, NO otherwise. Input: First line contains an integer T. T test cases follow. First line of each case contains two space-separated integers N and M. Second line contains N + M space-separated integers, the candies of the students. Output: For each test case, output M new line, Monk's answer to the M students. Print "YES" (without the quotes) or "NO" (without the quotes) pertaining to the Monk's answer. Constraints: $1 \leq T \leq 10$ $1 \leq N, M \leq 10^5$ $0 \leq A_i \leq 10^{12}$ Sample Input 1 2 3 3 2 9 11 2 Sample Output NO NO YES Time Limit: 3 Memory Limit: 256 Source Limit: Explanation Initially students with 3 and 2 candies are in the class. A student with 9 candies enters, No student with 9 candies in class. Hence, "NO" A student with 11 candies enters, No student with 11 candies in class. Hence, "NO" A student with 2 candies enters, Student with 2 candies found in class. Hence, "YES"

Practice link:https://www.hackerearth.com/practice/data-structures/trees/binary-search-tree/practice-problems/algorithm/monk-and-his-friends/?purpose=login&source=problem-page&update=google

Solution:

```java
import java.util.*;
class TestClass {
   public static void main(String args[] ) throws Exception {
       Scanner sc = new Scanner(System.in);
       int tc = sc.nextInt();
       while(tc-- != 0){
           int n = sc.nextInt();
           int m = sc.nextInt();
           Set<Long> hs = new HashSet<Long>();
           for(int i=0; i<n; i++){
               hs.add(sc.nextLong());
           }
           for(int i=0; i<m; i++){
               Long commingStud = sc.nextLong();
               if(hs.contains(commingStud)){
                   System.out.println("YES");
               }else{
                   System.out.println("NO");
                   hs.add(commingStud);
               }
           }
       }
   }
}
```