Assignment Questions 5

Question 1

Convert 1D Array Into 2D Array

You are given a **0-indexed** 1-dimensional (1D) integer array original, and two integers, m and n. You are tasked with creating a 2-dimensional (2D) array with m rows and n columns using **all** the elements from original.

The elements from indices 0 to n - 1 (**inclusive**) of original should form the first row of the constructed 2D array, the elements from indices n to 2 * n - 1 (**inclusive**) should form the second row of the constructed 2D array, and so on.

Return *an* m x n *2D array constructed according to the above procedure, or an empty 2D array if it is impossible.*

Example 1:

Input: original = [1,2,3,4], m = 2, n = 2

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

Explanation: The constructed 2D array should contain 2 rows and 2 columns.

The first group of n=2 elements in original, [1,2], becomes the first row in the constructed 2D array.

The second group of n=2 elements in original, [3,4], becomes the second row in the constructed 2D array.

Solution:-

```
class Solution {
    public int[][] construct2DArray(int[] original, int m, int n) {

        if (m * n != original.length) {
            return new int[0][0];
        }
        int[][] ans = new int[m][n];
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                ans[i][j] = original[i * n + j];
            }
        }
        return ans;
```

```
    }
}
```

Question 2

You have n coins and you want to build a staircase with these coins. The staircase consists of k rows where the ith row has exactly i coins. The last row of the staircase **may be** incomplete.

Given the integer n, return *the number of **complete rows** of the staircase you will build*.

Example 1:

Input: n = 5

Output: 2

Explanation: Because the 3rd row is incomplete, we return 2.

Solution:-

```
class Solution {
    public int arrangeCoins(int n) {
        long i=0;
        for(i=1;i*(i+1)/2<=n;i++);
        return (int)i-1;
    }
}
```

Question 3

Given an integer array nums sorted in **non-decreasing** order, return *an array of **the squares of each number** sorted in non-decreasing order*.

Example 1:

Input: nums = [-4,-1,0,3,10]

Output: [0,1,9,16,100]

Explanation: After squaring, the array becomes [16,1,0,9,100].

After sorting, it becomes [0,1,9,16,100].

Solution:-

```
class Solution {
    public int[] sortedSquares(int[] A) {
        int n = A.length;
        int[] result = new int[n];
        int i = 0, j = n - 1;
        for (int p = n - 1; p >= 0; p--) {
            if (Math.abs(A[i]) > Math.abs(A[j])) {
                result[p] = A[i] * A[i];
                i++;
            } else {
                result[p] = A[j] * A[j];
                j--;
            }
        }
        return result;
    }
}
```

Question 4

Given two **0-indexed** integer arrays nums1 and nums2, return *a list* answer *of size* 2 *where:*

- answer[0] *is a list of all* ***distinct*** *integers in* nums1 *which are* ***not*** *present in* nums2*.*
- answer[1] *is a list of all* ***distinct*** *integers in* nums2 *which are* ***not*** *present in* nums1.

Note that the integers in the lists may be returned in **any** order.

Example 1:

Input: nums1 = [1,2,3], nums2 = [2,4,6]

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

Explanation:

For nums1, nums1[1] = 2 is present at index 0 of nums2, whereas nums1[0] = 1 and nums1[2] = 3 are not present in nums2. Therefore, answer[0] = [1,3].

For nums2, nums2[0] = 2 is present at index 1 of nums1, whereas nums2[1] = 4 and nums2[2] = 6 are not present in nums2. Therefore, answer[1] = [4,6].

Solution:-

```
class Solution {
    public List<List<Integer>> findDifference(int[] nums1, int[] nums2) {

     HashSet<Integer> set1=new HashSet<Integer>();
      HashSet<Integer> set2=new HashSet<Integer>();

     for(int ele: nums1){
         set1.add(ele);
     }

     for(int ele:nums2){
         set2.add(ele);
     }


     List<List<Integer>> list=new ArrayList<>();

      ArrayList<Integer> l1=new ArrayList<>();

      ArrayList<Integer> l2=new ArrayList<>();

     for(int ele:set2){

         if(set1.contains(ele)==false){
           l1.add(ele);
         }
     }


       for(int ele:set1){

         if(set2.contains(ele)==false){
           l2.add(ele);
         }
     }


       list.add(l2);
       list.add(l1);
       return list;
    }
}
```

Question 5

Given two integer arrays arr1 and arr2, and the integer d, *return the distance value between the two arrays*.

The distance value is defined as the number of elements arr1[i] such that there is not any element arr2[j] where |arr1[i]-arr2[j]| <= d.

Example 1:

Input: arr1 = [4,5,8], arr2 = [10,9,1,8], d = 2

Output: 2

Explanation:

For arr1[0]=4 we have:

|4-10|=6 > d=2

|4-9|=5 > d=2

|4-1|=3 > d=2

|4-8|=4 > d=2

For arr1[1]=5 we have:

|5-10|=5 > d=2

|5-9|=4 > d=2

|5-1|=4 > d=2

|5-8|=3 > d=2

For arr1[2]=8 we have:

|8-10|=2 <= d=2

|8-9|=1 <= d=2

|8-1|=7 > d=2

|8-8|=0 <= d=2

Solution:-

```
class Solution {
    public int findTheDistanceValue(int[] arr1, int[] arr2, int d) {
        int ans=0;
        for (int i =0;i<arr1.length;i++){
            for (int j =0;j<arr2.length;j++){
                if (Math.abs(arr1[i]-arr2[j])<=d){
                   ans++;
                   break;
                }
            }
        }
        return (arr1.length-ans);

    }
}
```

Question 6

Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears **once** or **twice**, return *an array of all the integers that appears **twice***.

You must write an algorithm that runs in O(n) time and uses only constant extra space.

Example 1:

Input: nums = [4,3,2,7,8,2,3,1]

Output:

[2,3]

Solution:-

```
class Solution {
    public List<Integer> findDuplicates(int[] nums) {
        List<Integer> list = new ArrayList<>();
        int visited = -1;
        for(int i = 0;i<nums.length;i++){
            if(nums[Math.abs(nums[i])-1]<0){
                list.add(Math.abs(nums[i]));
            }
            else{
                nums[Math.abs(nums[i])-1] = nums[Math.abs(nums[i])-1]*visited;
            }
        }
```

```
        return list;
    }
}
```

Question 7

Suppose an array of length n sorted in ascending order is **rotated** between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

- [4,5,6,7,0,1,2] if it was rotated 4 times.
- [0,1,2,4,5,6,7] if it was rotated 7 times.

Notice that **rotating** an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in O(log n) time.

Example 1:

Input: nums = [3,4,5,1,2]

Output: 1

Explanation:

The original array was [1,2,3,4,5] rotated 3 times.

Solution:-

```
class Solution {
        public int findMin(int[] A) {
        final int N = A.length;
        if(N == 1) return A[0];
        int start = 0, end = N-1, mid;
        while(start < end){
            mid = (start+end) / 2;
            if(mid > 0 && A[mid] < A[mid-1]) return A[mid];
            if(A[start] <= A[mid] && A[mid] > A[end]) start = mid + 1;
            else end = mid - 1;
```

```
        }
        return A[start];
    }
}
```

Question 8

An integer array original is transformed into a **doubled** array changed by appending **twice the value** of every element in original, and then randomly **shuffling** the resulting array.

Given an array changed, return original *if* changed *is a* ***doubled*** *array. If* changed *is not a* ***doubled*** *array, return an empty array. The elements in* original *may be returned in* ***any*** *order*.

Example 1:

Input: changed = [1,3,4,2,6,8]

Output: [1,3,4]

Explanation: One possible original array could be [1,3,4]:

- Twice the value of 1 is 1 * 2 = 2.
- Twice the value of 3 is 3 * 2 = 6.
- Twice the value of 4 is 4 * 2 = 8.

Other original arrays could be [4,3,1] or [3,1,4].

Solution:-

```
class Solution {
    public int[] findOriginalArray(int[] changed) {

        int len = changed.length;
        if((len&1) != 0) return new int[0];

        // Sorting the array
        Arrays.sort(changed);

        // Store frequencies in map
        Map<Integer,Integer> map = new HashMap<>();
        for(int e : changed) map.put(e,map.getOrDefault(e,0)+1);

        int[] res = new int[len/2];
        int k = 0;
        for(int i=0; i<len; i++){
            int ele = changed[i];
```

```
            // if map contains 'ele'
            if(map.containsKey(ele)){

                // if map contains 'ele*2'
                if(map.containsKey(ele*2)){
                    res[k++] = ele;

                    // reduce frequency of 'ele' and 'ele*2'
                    map.put(ele,map.get(ele)-1);
                    map.put(ele*2,map.get(ele*2)-1);

                    // if freq of any key becomes <=0, remove it from map
                    if(map.get(ele)<=0) map.remove(ele);
                    if(map.containsKey(ele*2) && map.get(ele*2)<=0)
map.remove(ele*2);
                }
                else return new int[0];
            }

        }
        return res;
    }
}
```