Assignment 5 (2D Arrays) Solution

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*.

Solution:

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


      int[][] threed = new int[0][0];


       if(original.length != m*n){


           return threed;

       }


       int[][] twod = new int[m][n];
```

```
        int i  = 0;


        while(i<original.length){

            int j = 0;

            while(j<m){

                int k = 0;

                while(k<n){

                    twod[j][k] = original[i];

                    k++;

                    i++;

                }

                j++;

            }

        }


    return twod;


    }
```

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*.

Solution:

```
public int arrangeCoins(int n) {
        long s=1,e=n,mid,ans=0;
        while(s<=e){
            mid = s +(e-s)/2;
            if((mid*(mid+1))/2<=n){
                ans=mid;
                s=mid+1;
            }else{
                e=mid-1;
            }
        }
        return (int)ans;
    }
```

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*.

Solution:

```
public int[] sortedSquares(int[] nums) {
        for(int i = 0;i<nums.length;i++){
            nums[i] = nums[i] * nums[i];
        }
        Arrays.sort(nums);

        return nums;

    }
```

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.

Solution:

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




     HashSet<Integer> set1=new HashSet();

      HashSet<Integer> set2=new HashSet();


     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.

Solution:

```
 public int findTheDistanceValue(int[] arr1, int[] arr2, int d) {

        int count=0;

        int x=0;

        for(int i=0;i<arr1.length;i++){

            x=0;

            for(int j=0;j<arr2.length;j++){

                int diff=Math.abs(arr1[i]-arr2[j]);

                if(diff<=d){

                    j=arr2.length;

                }

                else{

                    x++;
                }
            }
            if(x==arr2.length){
                count++;
            }
        }
        return count;
    }
```

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:

```
 public List<Integer> findDuplicates(int[] nums) {
        ArrayList<Integer> al=new ArrayList<>();
                    HashMap<Integer,Integer> map=new HashMap<>();
                    if(nums.length==1){
```

```
            return al;
        }
        for(int i=0;i<nums.length;i++) {
            map.put(nums[i],map.getOrDefault(nums[i],0)+1);
        }
        for(int i:map.keySet()) {
            if(map.get(i)>1) {
                al.add(i);
            }
        }
        Collections.sort(al);
        return al;


    }
```

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:

```
public int findMin(int[] nums) {
    int l = 0;
    int r = nums.length - 1;

    while (l < r) {
      final int m = (l + r) / 2;
      if (nums[m] < nums[r])
        r = m;
      else
        l = m + 1;
    }

    return nums[l];
  }
```

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:

```
 public int[] findOriginalArray(int[] nums) {

      int[] vacarr = new int[0];

                // when we need to return vacant array

       int n= nums.length;

                        // size of the array

       if(n%2!=0)

       {

          return vacarr;

                        // when we will have odd number of integer in our
input(double array can't be in odd number)
```

```
        }
        HashMap<Integer, Integer> hm = new HashMap<Integer,
Integer>();
                      // for storing the frequencies of each input
        int[] ans = new int[(nums.length/2)];
        // answer storing array

        for(int i=0;i<n;i++)
        {
          hm.put(nums[i], hm.getOrDefault(nums[i],0)+1);
                     // storing the frequencies
        }
        int temp = 0;

        Arrays.sort(nums);
                // sorting in increasing order
        for(int i: nums)
        {

          if(hm.get(i)<=0)
          {
```

```
                              // if we have already decreased it's value when we
were checking y/2 value, like 2,4 we will remove 4 also when we will
check 2 but our iteration will come again on 4.


                continue;

            }


            if(hm.getOrDefault(2*i,0)<=0)

            {   // if we have y but not y*2 return vacant array

                return vacarr;

            }

            ans[temp++] = i;

                             // if we have both y and y*2, store in our ans array

           // decrease the frequency of y and y*2

            hm.put(i, hm.get(i)-1);

            hm.put(2*i, hm.get(2*i)-1);

        }


        return ans;

    }
```