Assignment Questions 4

Question 1
Given three integer arrays arr1, arr2 and arr3 **sorted** in **strictly increasing** order, return a sorted array of **only** the integers that appeared in **all** three arrays.
Example 1:
Input: arr1 = [1,2,3,4,5], arr2 = [1,2,5,7,9], arr3 = [1,3,4,5,8]
Output: [1,5]
Explanation: Only 1 and 5 appeared in the three arrays.

code:-

```
public List<Integer> arraysIntersection(int[] arr1, int[] arr2, int[]
arr3) {
        List<Integer> intersection = new ArrayList<Integer>();
        int length1 = arr1.length, length2 = arr2.length, length3 =
arr3.length;
        int index1 = 0, index2 = 0, index3 = 0;
        while (index1 < length1 && index2 < length2 && index3 < length3)
{
            int num1 = arr1[index1], num2 = arr2[index2], num3 =
arr3[index3];
            if (num1 == num2 && num1 == num3) {
                intersection.add(num1);
                index1++;
                index2++;
                index3++;
            } else {
                int increment1 = 0, increment2 = 0, increment3 = 0;
                if (num1 < num2 || num1 < num3)
                    increment1 = 1;
                if (num2 < num1 || num2 < num3)
                    increment2 = 1;
                if (num3 < num1 || num3 < num2)
                    increment3 = 1;
                index1 += increment1;
                index2 += increment2;
                index3 += increment3;
            }
        }
        return intersection;
    }
```

Question 2
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].

code:-

```
public List<List<Integer>> findDifference(int[] nums1, int[] nums2) {
        List<List<Integer>> ans = new ArrayList<>();
        HashSet<Integer> set1 = new HashSet<>();
        HashSet<Integer> set2 = new HashSet<>();

        for(int i = 0 ; i < nums1.length; i ++){
            set1.add(nums1[i]);
        }
          for(int i = 0 ; i < nums2.length; i ++){
            set2.add(nums2[i]);
        }
         List<Integer> list1 = new ArrayList<>();
          for(int i = 0 ; i < nums1.length; i ++){
            if(!set2.contains(nums1[i]) && !list1.contains(nums1[i])){
                list1.add(nums1[i]);
            }
        }
         ans.add(list1);
          List<Integer> list2 = new ArrayList<>();
          for(int i = 0 ; i < nums2.length; i ++){
            if(!set1.contains(nums2[i]) && !list2.contains(nums2[i])){
                list2.add(nums2[i]);
            }
        }
        ans.add(list2);
        return ans;

    }
```

Question 3
Given a 2D integer array matrix, return the transpose of matrix.
The transpose of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.
Example 1:
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[1,4,7],[2,5,8],[3,6,9]]

code:-

```
 public int[][] transpose(int[][] matrix) {
        int row=matrix.length;
        int col=matrix[0].length;
        int arr[][]=new int[col][row];
        for(int i=0;i<col;i++)
        {
            for(int j=0;j<row;j++)
            {
            arr[i][j]=matrix[j][i];
            }
        }
        return arr;

    }
```

Question 4
Given an integer array nums of 2n integers, group these integers into n pairs (a1, b1), (a2, b2), ..., (an, bn) such that the sum of min(ai, bi) for all i is **maximized**. Return *the maximized sum*.
Example 1:
Input: nums = [1,4,3,2]
Output: 4
Explanation: All possible pairings (ignoring the ordering of elements) are:
1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3) = 1 + 2 =
2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4) = 1 + 2 = 3
3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4) = 1 + 3 = 4
So the maximum possible sum is 4.

code:-

```
public int arrayPairSum(int[] nums) {
        Arrays.sort(nums);
        int n = nums.length;
        int sum = 0;
        for (int i = 0; i < n; i += 2) {
            sum += nums[i];
        }
        return sum;
    }
```

Question 5
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.

code:-

```
public int arrangeCoins(int n) {
         int i = 1; // which row we are on
           while(n > 0){ // checking to see if we have used all our
coins
                 i++; // increasing our row
                 n = n-i; // adding coins to our row
           }
           return i-1;
    }
```

Question 6
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]

code:-

```
 public int[] sortedSquares(int[] nums) {
        int n = nums.length;
        int[] result = new int[n];
        int left = 0;
        int right = n - 1;
        int i = n - 1;

        while (left <= right) {
            int leftSquare = nums[left] * nums[left];
            int rightSquare = nums[right] * nums[right];

            if (leftSquare > rightSquare) {
                result[i] = leftSquare;
                left++;
            } else {
                result[i] = rightSquare;
                right--;
            }

            i--;
        }

        return result;
    }
```

Question 7
You are given an m x n matrix M initialized with all 0's and an array of operations ops, where ops[i] = [ai, bi] means M[x][y] should be incremented by one for all 0 <= x < ai and 0 <= y < bi.
Count and return *the number of maximum integers in the matrix after performing all the operations*
example 1:
Input: m = 3, n = 3, ops = [[2,2],[3,3]]
Output:4
Explanation:The maximum integer in M is 2, and there are four of it in M. So return 4.

code:-

```
public int maxCount(int m, int n, List<List<Integer>> ops) {
        Map<Integer, Integer> xs = new HashMap<>();
        Map<Integer, Integer> ys = new HashMap<>();

        for (List<Integer> op : ops) {
            for (int i = 1; i <= op.get(0); i++) {
                xs.put(i, xs.getOrDefault(i, 0) + 1);
            }
            for (int j = 1; j <= op.get(1); j++) {
                ys.put(j, ys.getOrDefault(j, 0) + 1);
            }
        }

        int maxX = m;
        int maxY = n;
        int maxXval = 0;
        int maxYval = 0;

        for (Map.Entry<Integer, Integer> entry : xs.entrySet()) {
            int k = entry.getKey();
            int v = entry.getValue();
            if (v >= maxXval) {
                maxX = k;
                maxXval = v;
            }
        }

        for (Map.Entry<Integer, Integer> entry : ys.entrySet()) {
            int k = entry.getKey();
            int v = entry.getValue();
            if (v >= maxYval) {
                maxY = k;
                maxYval = v;
            }
        }

        return maxX * maxY;
```

```
    }
```

Question 8
Given the array nums consisting of 2n elements in the form [x1,x2,...,xn,y1,y2,...,yn].
Return the array in the form* [x1,y1,x2,y2,...,xn,yn].
Example 1:
Input:nums = [2,5,1,3,4,7], n = 3
Output: [2,3,5,4,1,7]
Explanation: Since x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 then the answer is [2,3,5,4,1,7].

code:-

```
public int[] shuffle(int[] nums, int n) {
        int[] ans=new int[nums.length];
        int mid=nums.length/2;
        int j=0;
        for(int i=0;i<nums.length;i=i+2){
           ans[i]=nums[j];
           j++;
        }
        j=mid;
         for(int i=1;i<nums.length;i=i+2){
           ans[i]=nums[j];
           j++;
        }
        return ans;
    }
```