Assignment 4 (2D-Arrays)

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.

Sol:-

```java
// Java program to find common elements in three arrays
import java.io.*;
class FindCommon {
    // This function prints common elements in ar1
    void findCommon(int arr1[], int arr2[], int arr3[])
    {
        // Initialize starting indexes for ar1[], ar2[] and
        // ar3[]
        int i = 0, j = 0, k = 0;

        // Iterate through three arrays while all arrays
        // have elements
        while (i < arr1.length && j < arr2.length
            && k < arr3.length) {
            // If x = y and y = z, print any of them and
            // move ahead in all arrays
            if (arr1[i] == arr2[j] && arr2[j] == arr3[k]) {
                System.out.print(arr1[i] + " ");
                i++;
                j++;
                k++;
            }

            // x < y
            else if (arr1[i] < arr2[j])
                i++;

            // y < z
            else if (arr2[j] < arr3[k])
                j++;

            // We reach here when x > y and z < y, i.e., z
            // is smallest
            else
                k++;
        }
    }

    // Driver code to test above
    public static void main(String args[])
    {
        FindCommon  ob = new FindCommon ();

        int arr1[] = { 1, 2,3,4,5 };
        int arr2[] = {1,2,5,7,9};
        int arr3[] = { 1,3,4,5,8 };

        System.out.print("Common elements are: ");
        ob.findCommon(arr1, arr2, arr3);
    }
}
```

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]]

Sol:-

```
      class Solution {

    public List<List<Integer>> findDifference(int[] nums1, int[] nums2) {
        List<List<Integer>> ans = new ArrayList<>();
        Set<Integer> s1 = new HashSet<>();
        Set<Integer> s2 = new HashSet<>();

        for(int ele : nums1){
            s1.add(ele);
        }
        for(int ele : nums2){
            s2.add(ele);
        }
        List<Integer> list1 = new ArrayList<>();
        for(int ele : s1){
            if(!s2.contains(ele)){
                list1.add(ele);
            }
        }
        ans.add(list1);

        List<Integer> list2 = new ArrayList<>();
        for(int ele : s2){
            if(!s1.contains(ele)){
                list2.add(ele);
            }
        }
        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]]

Sol:-

```
// Time Complexity: O(m*n)
// Space Complexity: O(m*n)
```

```
class Solution {
    public int[][] transpose(int[][] matrix) {
        // m - number of rows
        // n - number of cols
        int m = matrix.length;
        int n = matrix[0].length;
        // result - final transposed matrix
        int[][] result = new int[n][m];//here we create seprate array so, space will increase
        for(int j =0; j<n; j++)
            for(int i=0; i<m; i++)
            //swap between matrix[i][j] and matrix[j][i]
                result[j][i] = matrix[i][j];
        return result;
    }
 }
```

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

Sol:-

```
// Time Complexity: O(nlogn)
// Space Complexity: O(n)
class Solution {
    public int arrayPairSum(int[] nums) {
        //Sort the array in ascending order
        Arrays.sort(nums);
        // initialize sum to zero
        int sum = 0;
        for(int i=0;i<nums.length; i +=2){
            // Add everey element at even positions(0-indexed)
            sum = sum + Math.min(nums[i], nums[i+1]);
        }
        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*.

Sol:-

```
// Time Complexity: O(n)
// Space Complexity: O(1)
class Solution {
    public int arrangeCoins(int n) {
       int i = 1;
       int res = 0;
       while(n >= i){
           n = n - i;
```

```
            res++;
            i++;
        }
        return res;
    }
}
```

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]

Sol:-

```
// Time Complexity: O(n)
// Space Complexity: O(n)
class Solution {
    public int[] sortedSquares(int[] nums) {
        int n = nums.length;
        int[] result = new int[n];
        int start = 0, end = n - 1, i = n - 1;
        while(i >= 0){
            if(nums[start] * nums[start] > nums[end] * nums[end]){
                result[i--] = nums[start] * nums[start];
                start++;
            }
            else{
                result[i--] = nums[end] * nums[end];
                end--;
            }
        }
        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*

Sol:-

```
// Time Complexity: O(n)
// Space Complexity: O(1)
class Solution {
    public int maxCount(int m, int n, int[][] ops) {
        int[] common = new int[]{m, n};
        for(int[] op : ops){
            common[0] = Math.min(common[0], op[0]);
            common[1] = Math.min(common[1], op[1]);
        }
        return (common[0] * common[1]);
    }
}
```

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]

Sol:-

```
// Time Complexity: O(n)
// Space Complexity: O(n)
class Solution {
    public int[] shuffle(int[] nums, int n) {
        int[] result = new int[2 * n];
        for (int i = 0; i < n; i++) {
            result[2 * i] = nums[i];
            result[2 * i + 1] = nums[n + i];
        }
        return result;
    }
}
```