**Infinite Champions Programme – Day 3 (Assignment Sheet)**

**Instructions  
• Deadline: Submit your solutions by 29th September, 2025, EOD.  
• Platform: Test your solutions on LeetCode  
• Collaboration: Discussing concepts is encouraged, but all code must be your own.**

1. [Search a 2D Matrix](https://leetcode.com/problems/search-a-2d-matrix/)  
   • **Problem**: You are given an m × n matrix with the following properties:
   1. Each row is sorted in ascending order.
   2. The first integer of each row is greater than the last integer of the previous row.  
      Given a target integer, return true if it exists in the matrix, otherwise false.

• **Objective**: Use binary search by treating the 2D matrix as a 1D sorted array (index mapping).  
• **YouTube Solution (Java)**: [Search a 2D Matrix – Java Solution](https://www.youtube.com/watch?v=eT0UqrYuqbg)

class Solution {

    public boolean searchMatrix(int[][] matrix, int target) {

        int n=matrix.length;

        int m=matrix[0].length;

        int l=0;

        int r=n\*m -1;

        while(l<=r){

            int mid=l+(r-l)/2;

            int rows=mid/m;

            int cols=mid%m;

            if(matrix[rows][cols]==target){

                return true;

            }

            else if(matrix[rows][cols]<target){

                l=mid+1;

            }

            else{

                r=mid-1;

            }

        }

        return false;

    }

}

1. [Spiral Matrix](https://leetcode.com/problems/spiral-matrix/)  
   • **Problem**: Given an m × n matrix, return all elements in **spiral order** (clockwise). You must traverse the matrix layer by layer, shrinking the boundaries after each pass.  
   • **Objective**: Implement a solution using four boundary variables (top, bottom, left, right) and loop until traversal completes.  
   • **YouTube Solution (Java)**: [Spiral Matrix – Java Solution](https://www.youtube.com/watch?v=w3iZyl3fHmU)

class Solution {

public List<Integer> spiralOrder(int[][] matrix) {

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

int rows=matrix.length;

int cols=matrix[0].length;

int top=0;

int bottom=rows-1;

int left=0;

int right=cols-1;

while(top<=bottom && left<=right){

for(int i=left;i<=right;i++){

arr.add(matrix[top][i]);

}

top++;

for(int i=top;i<=bottom;i++){

arr.add(matrix[i][right]);

}

right--;

if(top<=bottom){

for(int i=right;i>=left;i--){

arr.add(matrix[bottom][i]);

}

bottom--;

}

if(left<=right){

for(int i=bottom;i>=top;i--){

arr.add(matrix[i][left]);

}

left++;

}

}

return arr;

}

}

1. [Number of Islands](https://leetcode.com/problems/number-of-islands/)  
   • **Problem**: You are given a 2D grid of '1' (land) and '0' (water). An island is formed by connecting adjacent lands horizontally or vertically. Return the total number of islands.  
   • **Objective**: Implement DFS or BFS to traverse each connected component and count islands efficiently.  
   • **YouTube Solution (Java)**: [Number of Islands – Java Solution](https://www.youtube.com/watch?v=U6-X_QOwPcs)

class Solution {

    public int numIslands(char[][] grid) {

        int n=grid.length;

        int m=grid[0].length;

        int c=0;

        for(int i=0;i<n;i++){

            for(int j=0;j<m;j++){

                if(grid[i][j]=='1'){

                    c++;

                    dfs(grid,i,j,n,m);

                }

            }

        }

        return c;

    }

    public void dfs(char[][] grid,int i,int j,int n,int m ){

        if(i < 0 || j < 0 || i >= n || j >= m || grid[i][j] == '0'){

            return;

        }

        grid[i][j]='0';

        dfs(grid,i+1,j,n,m);

        dfs(grid,i-1,j,n,m);

        dfs(grid,i,j+1,n,m);

        dfs(grid,i,j-1,n,m);

    }

}

**Submission Checklist  
• Time and space complexity analysis for each solution.  
• Test cases demonstrating the correctness of your solutions.**