Class Notes 4

- ▶ Python
- ▶ JavaScript
- ▼ Java

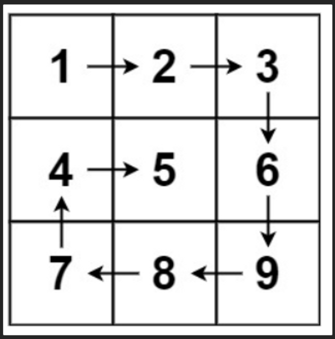
💡 **Question 1**

Given an m x n matrix, return *all elements of the* matrix *in spiral order*.

Example 1:

Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [1,2,3,6,9,8,7,4,5]



Solution:

TC: O(n)

SC: O(1)

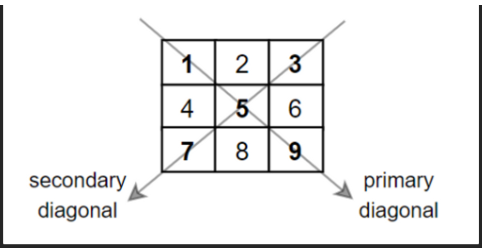
```
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> result = new ArrayList<>();
        int rows = matrix.length;
        int columns = matrix[0].length;
        int up = 0;
        int left = 0;
        int right = columns - 1;
        int down = rows - 1;


        while (result.size() < rows * columns) {
            // Traverse from left to right.
            for (int col = left; col <= right; col++) {
                result.add(matrix[up][col]);
            }
            // Traverse downwards.
            for (int row = up + 1; row <= down; row++) {
                result.add(matrix[row][right]);
            }
            // Make sure we are now on a different row.

                // Traverse from right to left.
                for (int col = right - 1; col >= left; col--) {
                    result.add(matrix[down][col]);
                }
                // Traverse upwards.
                for (int row = down - 1; row > up; row--) {
                    result.add(matrix[row][left]);
                }
            left++;
            right--;
            up++;
            down--;
        }
        return result;
    }
}
```

💡 **Question 2**

Given a square matrix mat, return the sum of the matrix diagonals.

Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

Example 1:



Input: mat = [[1,2,3],

[4,5,6],

[7,8,9]]

Output: 25

Explanation: Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25

Notice that element mat[1][1] = 5 is counted only once.

Solution:

TC: O(m+n)

SC: O(1)

```
class Solution {
    public int diagonalSum(int[][] mat) {
        int n = mat.length;
        int ans = 0;

        for (int i = 0; i < n; i++) {
            // Add elements from primary diagonal.
            ans += mat[i][i];
            // Add elements from secondary diagonal.
            ans += mat[n - 1 - i][i];
        }

        // If n is odd, subtract the middle element as it's added twi
ce.
        if (n % 2 != 0) {
            ans -= mat[n / 2][n / 2];
        }

        return ans;
    }
}
```

Question 3

Given a m x n matrix grid which is sorted in non-increasing order both row-wise and column-wise, return *the number of negative numbers in* grid.

Example 1:

Input: grid = [[4,3,2,-1],[3,2,1,-1],[1,1,-1,-2],[-1,-1,-2,-3]]

Output: 8

Explanation: There are 8 negatives number in the matrix.

TC: O(m*n)

SC : O(1)

```
class Solution {
    public int countNegatives(int[][] grid) {
        int count = 0;
    int n = grid[0].length;
    int currRowNegativeIndex = n - 1;


    // Iterate on all rows of the matrix one by one.
    for (int[] row : grid) {
        // Decrease 'currRowNegativeIndex' so that it points to curre
nt row's last positive element.
        while (currRowNegativeIndex >= 0 && row[currRowNegativeIndex]
< 0) {
            currRowNegativeIndex--;
        }
        // 'currRowNegativeIndex' points to the last positive elemen
t,
        // which means 'n - (currRowNegativeIndex + 1)' is the number
of all negative elements.
        count += (n - (currRowNegativeIndex + 1));
    }
    return count;
    }
}
```

Question 4

You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the ith customer has in the jth bank. Return *the wealth that the richest customer has.*

customer has.

A customer's wealth is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum wealth.

Example 1:

Input: accounts = [[1,2,3],[3,2,1]]

Output: 6

Explanation:

1st customer has wealth = 1 + 2 + 3 = 6

2nd customer has wealth = 3 + 2 + 1 = 6

Both customers are considered the richest with a wealth of 6 each, so return 6.

Solution:

TC: O(m*n)

SC : O(1)

```
class Solution {
    public int maximumWealth(int[][] accounts) {
        // Initialize the maximum wealth seen so far to 0 (the minimu
m wealth possible)
        int maxWealthSoFar = 0;

        // Iterate over accounts
        for (int[] account : accounts) {
            // For each account, initialize the sum to 0
            int currCustomerWealth = 0;
            // Add the money in each bank
            for (int money : account) {
                currCustomerWealth += money;
            }
            // Update the maximum wealth seen so far if the current w
ealth is greater
            // If it is less than the current sum
            maxWealthSoFar = Math.max(maxWealthSoFar, currCustomerWea
lth);
        }

        // Return the maximum wealth
        return maxWealthSoFar;
    }
}
```