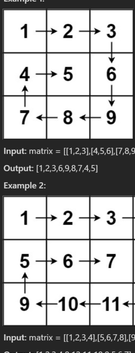
# Class Notes 4

- Python
- JavaScript
- Java
- Cpp

### Question 1

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

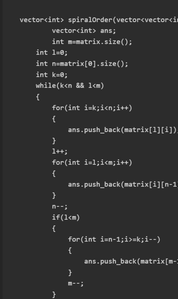
**Example 1:**



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

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

**Example 2:**



**Input:** matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]

**Output:** [1,2,3,4,8,12,11,10,9,5,6,7]

```
vector<int> spiralOrder(vector<vector<int>>& matrix) {
        vector<int> ans;
        int m=matrix.size();
    int l=0;
    int n=matrix[0].size();
    int k=0;
    while(k<n && l<m)
    {
        for(int i=k;i<n;i++)
        {
            ans.push_back(matrix[l][i]);
        }
        l++;
        for(int i=l;i<m;i++)
        {
            ans.push_back(matrix[i][n-1]);
        }
        n--;
        if(l<m)
        {
            for(int i=n-1;i>=k;i--)
            {
                ans.push_back(matrix[m-1][i]);
            }
            m--;
        }
        if(k<n)
        {
            for(int i=m-1;i>=l;i--)
            {
                ans.push_back(matrix[i][k]);
            }
            k++;
        }
    }
    return ans;
    }
```

### Question 2

Given an n x n matrix mat, return the sum of all elements of the matrix diagonally from the upper left to the lower right (i.e., sum(mat[i][i] ) for all i).

**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(n)

SC: O(1)

```
class Solution {
public:
    int diagonalSum(vector<vector<int>>& mat) {
        int n = mat.size();
        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 twice.
        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(vector<vector<int>>& grid) {
        int count = 0;
        int n = grid[0].size();
        int currRowNegativeIndex = n - 1;

        for (vector<int>& row : grid) {
            // Decrease 'currRowNegativeIndex' so that it points to current
row's last positive element.
            while (currRowNegativeIndex >= 0 && row[currRowNegativeIndex] <
0) {
                currRowNegativeIndex--;
            }
            // 'currRowNegativeIndex' points to the last positive element,
            // 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.*

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(vector<vector<int>>& accounts) {
        // Initialize the maximum wealth seen so far to 0 (the minimum wealt
h possible)
        int maxWealthSoFar = 0;

        // Iterate over accounts
        for (vector<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 wealth i
s greater
            // If it is less than the current sum
            maxWealthSoFar = max(maxWealthSoFar, currCustomerWealth);
        }

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