Class Notes 5

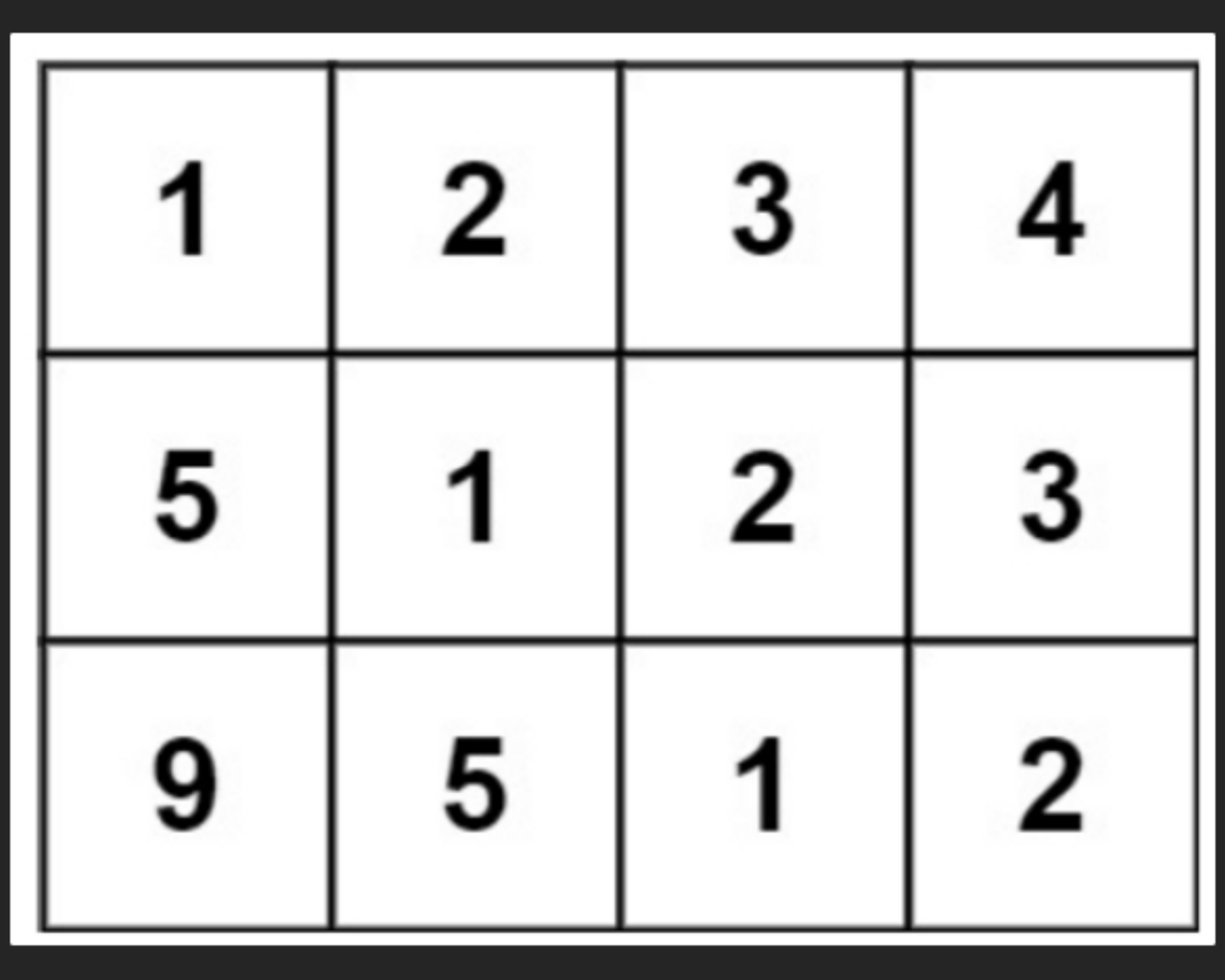
▸ Python

▸ JavaScript

▾ Java

💡 **Question 1**

Given an m x n matrix, return true if the matrix is Toeplitz. Otherwise, return false.
A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same elements.



Example 1:

Input: matrix = [[1,2,3,4],[5,1,2,3],[9,5,1,2]]
Output: true
Explanation:
In the above grid, the diagonals are:
"[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]".
In each diagonal all elements are the same, so the answer is True.

Solution:

Intuition and Algorithm

For each diagonal with elements in order $a_1, a_2, a_3, \ldots, a_k$, we can check $a_1 = a_2, a_2 = a_3, \ldots, a_{k-1} = a_k$. The matrix is *Toeplitz* if and only if all of these conditions are true for all (top-left to bottom-right) diagonals.

Every element belongs to some diagonal, and it's previous element (if it exists) is it's top-left neighbor. Thus, for the square `(r, c)`, we only need to check `r == 0 OR c == 0 OR matrix[r-1][c-1] == matrix[r][c]`.

Time Complexity: *O*(*M*∗*N*), as defined in the problem statement.

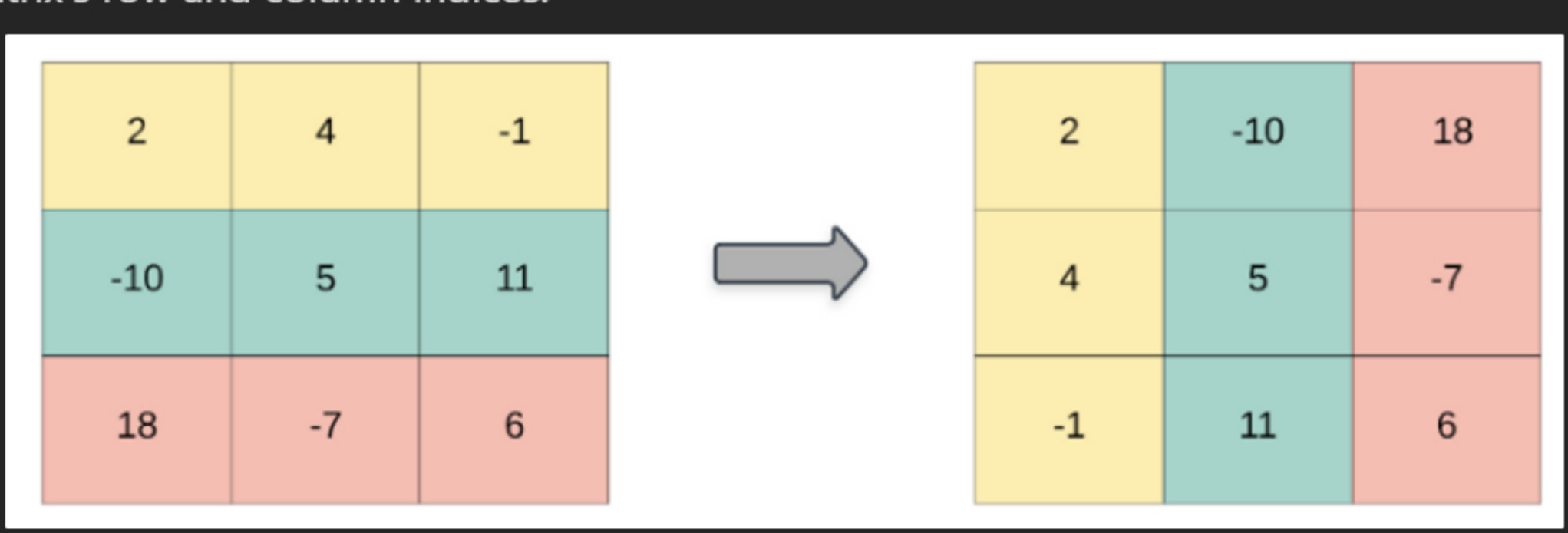
Space Complexity: *O*(1).

```java
class Solution {
    public boolean isToeplitzMatrix(int[][] matrix) {
        for (int r = 0; r < matrix.length; ++r)
            for (int c = 0; c < matrix[0].length; ++c)
                if (r > 0 && c > 0 && matrix[r-1][c-1] != matrix[r][c])
                    return false;
        return true;
    }
}
```

💡 **Question 2**

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]]

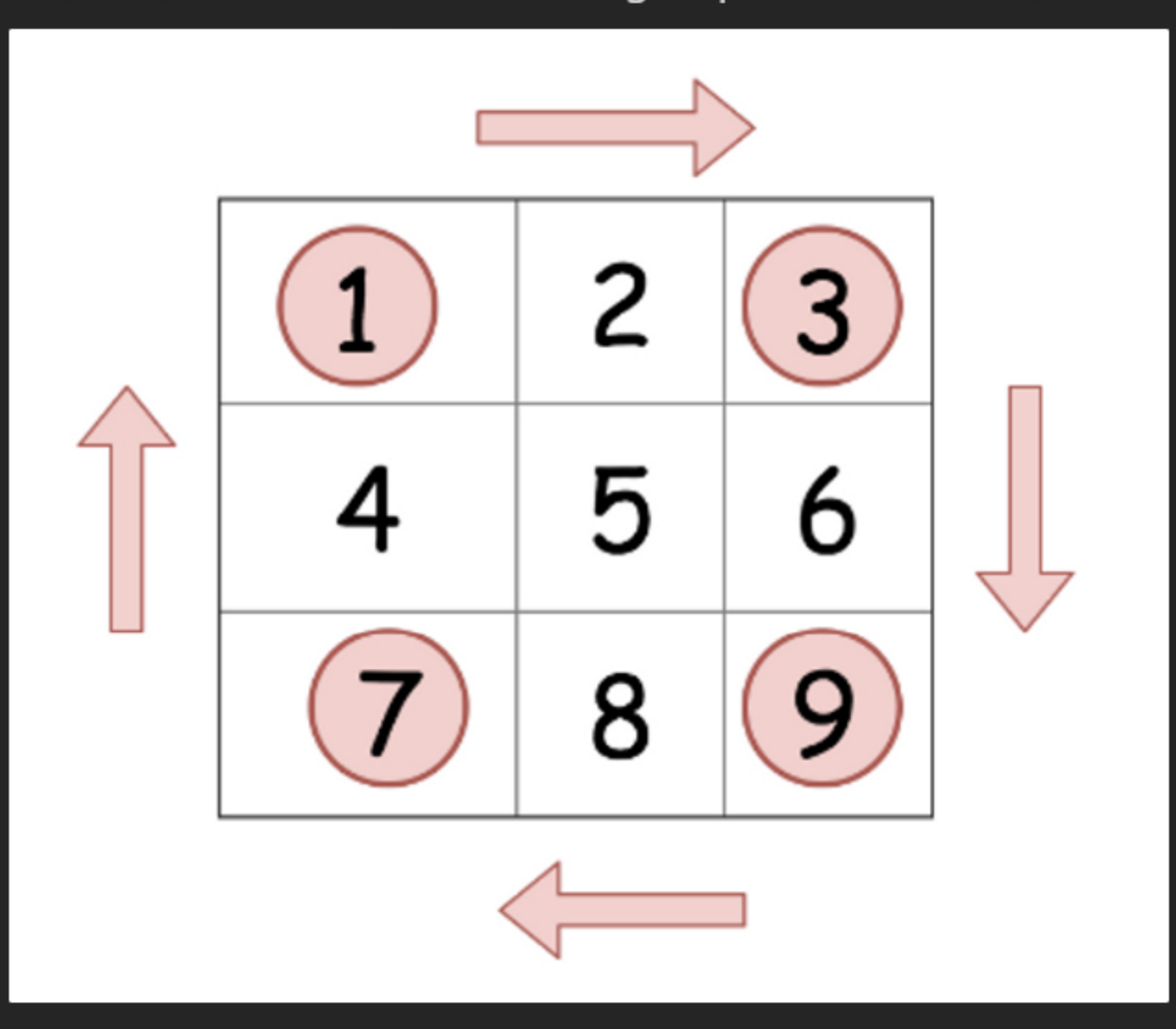
Output:

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

Solution:

Intuition

Observe how the cells move in groups when we rotate the image.



We can iterate over each group of four cells and rotate them.

Complexity Analysis

Let *M* be the number of cells in the matrix.

Time complexity: O(*M*), as each cell is getting read once and written once.

Space complexity: O(1) because we do not use any other additional data structures.

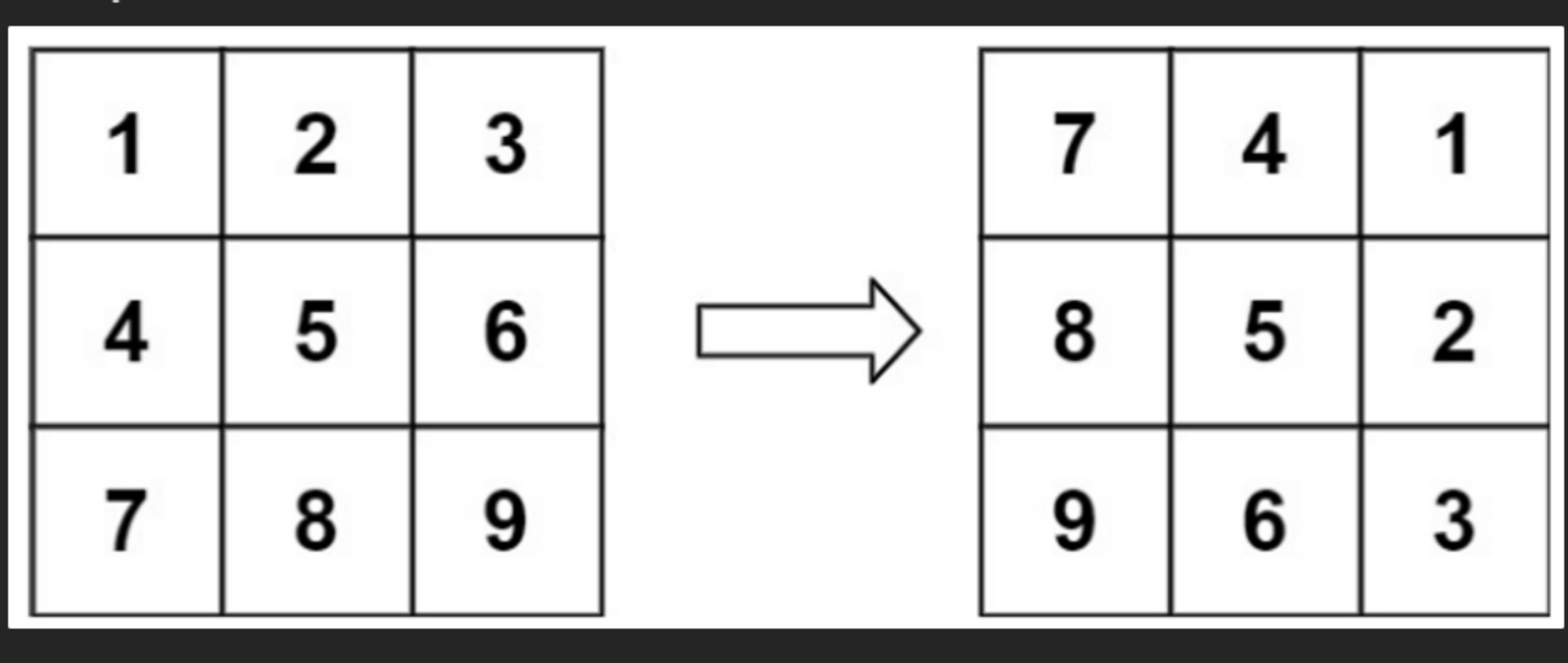
```java
class Solution {
    public int[][] transpose(int[][] A) {
        int R = A.length, C = A[0].length;
        int[][] ans = new int[C][R];
        for (int r = 0; r < R; ++r)
            for (int c = 0; c < C; ++c) {
                ans[c][r] = A[r][c];
            }
        return ans;
    }
}
```

💡 **Question 3**

You are given an n x n 2D matrix representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

Example 1:



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

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

Solution:

Intuition and Algorithm

The transpose of a matrix A with dimensions R x C is a matrix *ans* with dimensions C x R for which ans[c][r] = A[r][c].

We initialize a new matrix *ans* representing the answer. Then, we'll copy each entry of the matrix as appropriate.

Complexity Analysis

Time Complexity: *O*(*R*∗*C*), where *R* and *C* are the number of rows and columns in the given matrix A.

Space Complexity: *O*(*R*∗*C*), the space used by the answer.

```java
class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        for (int i = 0; i < (n + 1) / 2; i ++) {
            for (int j = 0; j < n / 2; j++) {
                int temp = matrix[n - 1 - j][i];
                matrix[n - 1 - j][i] = matrix[n - 1 - i][n - j - 1];
                matrix[n - 1 - i][n - j - 1] = matrix[j][n - 1 -i];
                matrix[j][n - 1 - i] = matrix[i][j];
                matrix[i][j] = temp;
            }
        }
    }
}
```

💡 **Question 4**

Given a non-empty array of non-negative integers nums, the **degree** of this array is defined as the maximum frequency of any one of its elements.

Your task is to find the smallest possible length of a (contiguous) subarray of nums, that has the same degree as nums.

Example 1:

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

Output: 2

Explanation:

The input array has a degree of 2 because both elements 1 and 2 appear twice.

Of the subarrays that have the same degree:

[1, 2, 2, 3, 1], [1, 2, 2, 3], [2, 2, 3, 1], [1, 2, 2], [2, 2, 3], [2, 2]

The shortest length is 2. So return 2.

Complexity Analysis

- Time Complexity: O(m*n)
- Space Complexity: O(1)

Solution:

```java
class Solution {
    public int maximumWealth(int[][] accounts) {
        // Initialize the maximum wealth seen so far to 0 (the minimum wealth p
        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 wealth is g
            // If it is less than the current sum
            maxWealthSoFar = Math.max(maxWealthSoFar, currCustomerWealth);
        }

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

▸ C++