Day 45/180 Rotate Matrix in C++

1: Rotate Image:

Solution - 1

```
void rotate(vector<vector<int>> &matrix) {
    int row = matrix.size();  // Get the number of rows in the matrix
    int col = matrix[0].size();  // Get the number of columns in the matrix

    vector<vector<int>> temp = matrix;  // Create a temporary matrix to store the
rotated values

    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            // Rotate the values by 90 degrees clockwise and update the original
matrix
            matrix[j][row - i - 1] = temp[i][j];
        }
    }
}
```

Space Complexity:

The code uses an additional matrix "temp" to store the rotated values, which has the same dimensions as the input matrix. Therefore, the space complexity is O(N^2), where N is the size of the matrix (N x N).

Time Complexity:

The code uses two nested loops to iterate through the entire matrix, so the time complexity is O(N^2), where N is the size of the matrix. This is because each element in the matrix is visited exactly once to perform the rotation.

Optimal Soution in O(1) Space

```
void rotate(vector<vector<int>& matrix) {
    int row = matrix.size();  // Get the number of rows in the matrix
    int col = matrix[0].size();  // Get the number of columns in the matrix

    // Step 1: Transpose the matrix
    for (int i = 0; i < row; i++) {
        for (int j = i; j < col; j++) {
            swap(matrix[i][j], matrix[j][i]);  // Swap elements to perform the
transpose
        }
    }

    // Step 2: Reverse the columns
    for (int i = 0; i < row; i++) {
        reverse(matrix[i].begin(), matrix[i].end());  // Reverse the elements in
each row
    }
}
```

Time Complexity:

- The code uses two nested loops in the first step to perform the matrix transpose, which takes O(N^2) time, where N is the size of the matrix (N x N).In the second step, it reverses each row, which also takes O(N^2) time.Overall, the time complexity of this code is O(N^2), which is linear with respect to the number of elements in the matrix.

Space Complexity:

- The code performs the rotation in-place without using any additional data structures. It only uses a constant amount of extra space for variables. Therefore, the space complexity is O(1), which is constant.

2: Matrix Rotation by 180 degree:

```
void rotateby90(vector<vector<int> >& matrix) {
    int n = matrix.size(); // Get the size of the square matrix

    // Step 1: Swap elements along the main diagonal
    // This operation mirrors the matrix along the diagonal
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            swap(matrix[i][j], matrix[n - j - 1][n - i - 1]);
        }
    }

    // Step 2: Reverse each row of the mirrored matrix
    for (int i = 0; i < n; i++) {
        reverse(matrix[i].begin(), matrix[i].end());
    }
}

void rotate(vector<vector<int> >& matrix) {
    // Rotate the matrix by 90 degrees twice to achieve a 180-degree rotation
    // This is equivalent to a 90-degree counterclockwise rotation
    rotateby90(matrix); // First 90-degree rotation
    rotateby90(matrix); // Second 90-degree rotation
}
```

3: Rotate by 90 degree anticlockwise:

```
void rotateby90(vector<vector<int> >& matrix, int n) {
    // Step 1: Swap elements along the secondary diagonal
    // This operation mirrors the matrix along the secondary diagonal (from
top-right to bottom-left).
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            swap(matrix[i][j], matrix[n - j - 1][n - i - 1]);
        }
    }

    // Step 2: Reverse rows
```

```
    // After mirroring along the secondary diagonal, the function reverses each
row to complete the 90-degree counterclockwise rotation.
    for (int i = 0; i < n; i++) {
        reverse(matrix[i].begin(), matrix[i].end());
    }
}
```

4:Left Rotate Matrix K times (This is different Problem which was taught in class)

Solution in O(N * N) Space

```
vector<vector<int>> rotateMatrix(int N, int M, int K, vector<vector<int>> Mat) {
    vector<vector<int>> res(N); // Initialize the result matrix with N rows

    for (int i = 0; i < N; i++) {
        K %= M; // Ensure K is within the range of the number of columns (M)

        // Step 1: Copy elements from column K to M-1
        for (int j = K; j < M; j++) {
            res[i].push_back(Mat[i][j]); // Copy the element to the result matrix
        }

        // Step 2: Copy elements from column 0 to K-1
        for (int j = 0; j < K; j++) {
            res[i].push_back(Mat[i][j]); // Copy the element to the result matrix
        }
    }

    return res; // Return the rotated matrix
}
```

Solution In O(1) Space

```cpp
vector<vector<int>> rotateMatrix(int N, int M, int K, vector<vector<int>> Mat) {
    for (int i = 0; i < N; i++) {
        K %= M; // Ensure K is within the range of the number of columns (M)

        // Left Rotate Mat[i] by K positions
        // Step 1: Reverse the first K elements
        reverse(Mat[i].begin(), Mat[i].begin() + K);

        // Step 2: Reverse the remaining elements after the first K elements
        reverse(Mat[i].begin() + K, Mat[i].end());

        // Step 3: Reverse the entire row, effectively achieving the rotation
        reverse(Mat[i].begin(), Mat[i].end());
    }

    return Mat; // Return the rotated matrix
}
```