2D-Matrix

Q. 1 Given a 2-D array of size N * M. Print values row by row.

```
void PrintMatrix(vector<vector<int>> matrix){
	for(int row = 0; row < matrix.size(); row++){
		for(int col = 0; col < matrix[row].size(); col++){
			cout<<matrix[row][col]<<" ";
		}
		cout<<endl;
	}
}
```

Time Complexity O(N*M)

Q. 2 Given a 2-D array of size N * M. Print sum of rows.

```
void PrintMatrixRowSum(vector<vector<int>> matrix){
	for(int row = 0; row < matrix.size(); row++){
		int row_sum = 0;
		for(int col = 0; col <matrix[row].size(); col++){
			row_sum += matrix[row][col];
		}
		cout<<"Sum of row "<<row<<" :: "<<row_sum<<endl;
	}
}
```

Time Complexity O(N*M)

Q. 3 Given a 2-D array of size N * M. Print sum of columns.

```
void PrintMatrixColumnSum(vector<vector<int>> matrix){
	int N = matrix.size();
	int M = matrix[0].size();
	for(int col = 0; col < M;  col++){
		int col_sum = 0;
		for(int row = 0; row < N; row++){
			col_sum += matrix[row][col];
		}
		cout<<"Sum of columns "<<col<<" :: "<<col_sum<<endl;
	}
}
```

Time Complexity O(N*M)

Q. 4 Given two 2-D matrices. Return the sum.

```
vector<vector<int>> AddMatrices(vector<vector<int>> matrix1, vector<vector<int>> matrix2){
	vector<vector<int>> resultant;
	int matrix1_row_size = matrix1.size();
	int matrix1_col_size = matrix1[0].size();
```

```
        int matrix2_row_size = matrix2.size();
        int matrix2_col_size = matrix2[0].size();

        if(matrix1_row_size == matrix2_row_size && matrix1_col_size = matrix2_col_size){
                for(int row = 0; row < matrix1.size(); row++){
                        vector<int> v;
                        for(int col = 0; col < matrix1[0].size(); col++){
                                v.push_back(matrix1[row][col] + matrix2[row][col]);
                        }
                        resultant.push_back(v);
                }
        }
        return resultant;
}
```
Time Complexity O(N*M) Space Complexity O(N*M)

Q. 5 Given a square matrix. Print the diagonal elements (Left to Right).

```
void PrintDiagonalElementsLR(vector<vector<int>> matrix){
        for(int row = 0; row < matrix[0].size(); row++){
                cout<<matrix[row][row]<<" ";
        }
}
```
Time Complexity O(N)

Q. 6 Given a square matrix. Print the diagonal elements (Right to Left).

```
void PrintDiagonalElementsRL(vector<vector<int>> matrix){
        int row = 0;
        int col = matrix[0].size() - 1;
        while(row < matrix[0].size() && col >= 0){
                cout<<matrix[row][col]<<" ";
                row++;
                col--;
        }
        //or with for loop
        for(int row = 0; row < matrix[0].size(); row++){
                cout<<matrix[row][col--]<<" ";
        }
}
```
Time Complexity O(N)

Q. 7 Given a 2-D matrix of N*M. Print all the diagonal elements (Left to Right).

```
void PrintDiagonals(vector<vector<int>> matrix, int row, int col, int N){
        int j = col;
        int i = row;
        while (i < N && j >= 0){
```

```
                cout<<matrix[i++][j--]<<" ";
        }
}

void PrintAllDiagonals(vector<vector<int>> matrix){
        //for first row
        for(int col = 0; col < matrix[0].size(); col++){
                PrintDiagonals(matrix, 0, col, matrix.size());
        }
        //for first col
        for(int row = 0; row < matrix[0].size(); row++){
                PrintDiagonals(matrix, row, matrix[0].size() - 1,  matrix.size());
        }
}
```

Time Complexity O(N*M)

Q. 8 Given a square matrix. Convert into transpose(upper triangle) matrix without using extra space.

```
void TransposeMatrix(vector<vector<int>>& matrix){
        for(int row = 0; row < matrix.size(); row++){
                for(int col = row + 1; col < matrix[row].size(); col++){
                        int temp = matrix[col][row];
                        matrix[col][row] = matrix[row][col];
                        matrix[row][col] = temp;
                }
        }
}
```

OR

```
vector<vector<int>> TransposeMatrix(vector<vector<int>>& matrix){
        int rows = A.size();
        if(rows == 0) return {{}};
        int cols = A[0].size();
        vector<vector<int>> out(cols, vector<int>(rows));
        for(int row = 0; row < rows; row++){
                for(int col = 0; col < cols; col++){
                        out[col][row] = matrix[row][col];
                }
        }
        return out;
}
```

Time Complexity O(N*N)

Q. 9 Given a square matrix. Convert into transpose(lower triangle) matrix without using extra space.

```
void TransposeMatrix(vector<vector<int>>& matrix){
```

```
        for(int col = 0; col < matrix[0].size(); col++){
                for(int row = col + 1; row < matrix.size(); row++){
                        int temp = matrix[col][row];
                        matrix[col][row] = matrix[row][col];
                        matrix[row][col] = temp;
                }
        }
}
```
Time Complexity O(N*N)

Q. 10 Given a square matrix. Rotate it by 90 degree in clockwise direction.

```
void ReverseRows(vector<int>& row){
        int s = 0;
        int e = row.size() - 1;
        while(s < e){
                int t = row[s];
                row[s] = row[e];
                row[e] = t;
                s++;
                e--;
        }
}
void RotateMatrix(vector<vector<int>>& matrix){
        //1. Transpose given matrix
        //2. Reverse matrix rows
        TransposeMatrix(matrix);
        for(int i = 0; i < matrix.size(); i++){
                ReverseRows(matrix[i]);
        }
}
```
Time Complexity O(N*N)

Q. 11 Given a 2-D matrix. Print boundaries of matrix.

```
void PrintMatrixBoundary(vector<vector<int>> matrix){
        //Print upper row
        for(int i = 0; i < matrix[0].size(); i++){
                cout<<matrix[0][i]<<" ";
        }
        cout<<endl;
        //Print right most column
        int col = matrix[0].size() - 1;
        for(int i = 1; i < matrix.size(); i++){
                cout<<matrix[i][col]<<" ";
        }
        cout<<endl;
        //Print bottom row
```

```
        int row = matrix.size() - 1;
        col = matrix[0].size() - 1;
        for(int i = col - 1; i >= 0; i--){
                cout<<matrix[row][i]<" ";
        }
        cout<<endl;
        //Print left lost column
        row = matrix.size() - 1;
        for(int i = row - 1; i > 0; i--){
                cout<<matrix[i][0]<" ";
        }
}
```

Q. 12 Print matrix in spiral order.

```
vector<int> PrintSpiral(vector<vector<int>> matrix){
        vector<int> out;
        for(int left = 0, right = matrix[0].size() - 1, int top = 0, int bottom = matrix.size() - 1;
                left <= right && top <= bottom;
                ++left, --right, ++top, --bottom){
                        for(int j = left; j <= right; ++j){
                                out.push_back(matrix[top][j]);
                        }

                        for(int i = top + 1; i < bottom; ++i){
                                out.push_back(matrix[i][right]);
                        }

                        for(int j = right; top < bottom && j >= left; --j){
                                out.push_back(matrix[bottom][j]);
                        }

                        for(int i = bottom - 1; left < right && i > top; --i){
                                out.push_back(matrix[i][left]);
                        }
                }
        return out;
}
```

Q. 13 Matrix multiplication

1. take the row of mat1 and take the column of mat
2. do A[i][j] * B[i][j] + A[i][j+1] * B[i+][j] .... + A[i][N-1]*B[N-1][j]

```
vector<vector<int>> MultiplyMatrix(vector<vector<int>> A, vector<vector<int>> B){
        vector<vector<int>> resultant;
        int A_Col = A[0].size();
        int B_Row = B.size();
        if(A_Col == B_Row){
```

```
            for(int i = 0; i < A.size(); i++){
                vector<int> v;
                for(int j = 0; j < A[i].size(); j++){
                    int sum = 0;
                    for(int k = 0; k < B[0].size(); k++){
                        sum += A[i][k] * B[k][j];
                    }
                    v.push_back(sum);
                }
                resultant.push_back(v);
            }
        }
        return resultant;
}
```

Q. 14 Anti Diagonals. Given a N*N matrixA, return an array of its anti-diagonals. Look at example for more details:

Input 1

1 2 3

4 5 6

7 8 9

output 1

1 0 0

2 4 0

3 5 7

6 8 0

9 0 0

```
vector<vector<int>> AntiDiagonal(vector<vector<int>>& A){
   vector<vector<int>> out;
   //for first row
        for(int col = 0; col < A[0].size(); col++){
            int i = 0;
            int j = col;
            vector<int> v(A[0].size(), 0);
            while(i < A.size() && j >= 0){
                v[i] = A[i][j];
                i++;
                j--;
      }
      out.push_back(v);
   }
   //for first col
        for(int row = 1; row < A.size(); row++){
            int i = row;
            int j = A[0].size() - 1;
            vector<int> v(A[0].size(), 0);
            int k =0;
            while(i < A[0].size() && j >= 0){
```

```
                v[k] = A[i][j];
                i++;
                j--;
                k++;
        }
        out.push_back(v);
    }
    return out;
}
```