

2D Array questions



explain problem with example and simple and optimized java code and also give time complexity and do not give enhanced for loop

Basic to Intermediate Level:

Matrix Addition and Subtraction: Given two matrices, perform addition and subtraction.

Matrix Transpose: Find the transpose of a given matrix.

Matrix Multiplication: Multiply two matrices, if possible, and return the result.

Search in a 2D Matrix: Search for a given element in a sorted 2D matrix where each row and column is sorted.

Rotate Matrix by 90 Degrees: Rotate a given N x N matrix by 90 degrees clockwise.

Find the Row with Maximum 1s: In a binary matrix where each row is sorted, find the row with the maximum number of 1s.

Spiral Order Traversal: Traverse a 2D matrix in spiral order and print the elements.

Diagonal Traversal: Print all elements of the matrix in diagonal order.

Find the Sum of all Elements: Find the sum of all elements in the given matrix.

Set Matrix Zeros: If an element in a matrix is zero, set the entire row and column to zero.



Here are solutions to the matrix-based problems with examples and optimized Java code. Each solution includes an explanation, Java code, and time complexity analysis.

1. Matrix Addition and Subtraction

Problem:

Given two matrices, perform both addition and subtraction element-wise. The matrices must be of the same dimensions.

Example:

text

Matrix A:	Matrix B:
1 2 3	4 5 6
4 5 6	7 8 9

Addition Result:

text

```
5 7 9
11 13 15
```

Subtraction Result:

text

```
-3 -3 -3
-3 -3 -3
```

Java Code:

java

```
public class MatrixOperations {
    public static int[][] addMatrices(int[][] A, int[][] B) {
        int rows = A.length;
        int cols = A[0].length;
        int[][] result = new int[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = A[i][j] + B[i][j];
            }
        }
        return result;
    }

    public static int[][] subtractMatrices(int[][] A, int[][] B) {
        int rows = A.length;
        int cols = A[0].length;
        int[][] result = new int[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = A[i][j] - B[i][j];
            }
        }
        return result;
    }
}
```

Time Complexity:

- **Addition and Subtraction:** $O(n * m)$, where `n` is the number of rows and `m` is the number of columns.

2. Matrix Transpose

Problem:

Find the transpose of a matrix (rows become columns and vice versa).

Example:

text

```
Matrix A:
1 2 3
```

4 5 6

Transpose:

text

```
1 4
2 5
3 6
```

Java Code:

java

```
public class MatrixTranspose {
    public static int[][] transpose(int[][] A) {
        int rows = A.length;
        int cols = A[0].length;
        int[][] result = new int[cols][rows];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[j][i] = A[i][j];
            }
        }
        return result;
    }
}
```

Time Complexity:

- **Transpose:** $O(n * m)$, where `n` is the number of rows and `m` is the number of columns.

3. Matrix Multiplication

Problem:

Multiply two matrices A (n x m) and B (m x p) to get the result matrix (n x p).

Example:

text

Matrix A:	Matrix B:
1 2	5 6
3 4	7 8

Multiplication Result:

text

```
19 22
43 50
```

Java Code:

java

```
public class MatrixMultiplication {
    public static int[][] multiply(int[][] A, int[][] B) {
        int rowsA = A.length;
        int colsA = A[0].length;
        int colsB = B[0].length;
        int[][] result = new int[rowsA][colsB];

        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < colsB; j++) {
                for (int k = 0; k < colsA; k++) {
                    result[i][j] += A[i][k] * B[k][j];
                }
            }
        }
        return result;
    }
}
```

Time Complexity:

- **Multiplication:** $O(n * m * p)$, where `n` is rows in A, `m` is columns in A (or rows in B), and `p` is columns in B.

4. Search in a 2D Matrix

Problem:

Search for a target value in a sorted 2D matrix where each row and column is sorted.

Example:

text

```
Matrix:
1  4  7
2  5  8
3  6  9
```

Target = 5 → Found.

Java Code:

java

```

public class MatrixSearch {
    public static boolean searchMatrix(int[][] matrix, int target) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int row = 0, col = cols - 1;

        while (row < rows && col >= 0) {
            if (matrix[row][col] == target) {
                return true;
            } else if (matrix[row][col] > target) {
                col--;
            } else {
                row++;
            }
        }
        return false;
    }
}

```

Time Complexity:

- **Search:** $O(n + m)$, where `n` is rows and `m` is columns.

5. Rotate Matrix by 90 Degrees**Problem:**

Rotate an $N \times N$ matrix by 90 degrees clockwise.

Example:

text

```

Matrix:
1 2 3
4 5 6
7 8 9

```

Rotated Matrix:

text

```

7 4 1
8 5 2
9 6 3

```

Java Code:

```
java
```

```
public class RotateMatrix {
    public static void rotate(int[][] matrix) {
        int n = matrix.length;

        // Transpose the matrix
        for (int i = 0; i < n; i++) {
            for (int j = i; j < n; j++) {
                int temp = matrix[i][j];
                matrix[i][j] = matrix[j][i];
                matrix[j][i] = temp;
            }
        }

        // Reverse each row
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n / 2; j++) {
                int temp = matrix[i][j];
                matrix[i][j] = matrix[i][n - j - 1];
                matrix[i][n - j - 1] = temp;
            }
        }
    }
}
```

Time Complexity:

- **Rotate:** $O(n^2)$, where `n` is the dimension of the matrix.

6. Find the Row with Maximum 1s

Problem:

In a binary matrix where each row is sorted, find the row with the maximum number of 1s.

Example:

```
text
```

```
Matrix:
0 0 1 1
0 1 1 1
0 0 0 1
```

Output: Row 2 (index 1)

Java Code:

```
java
```

```
public class RowWithMaxOnes {
    public static int rowWithMaxOnes(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int maxRow = -1, j = cols - 1;
```

```

    for (int i = 0; i < rows; i++) {
        while (j >= 0 && matrix[i][j] == 1) {
            maxRow = i;
            j--;
        }
    }
    return maxRow;
}

```

Time Complexity:

- **Find Row:** $O(n + m)$, where `n` is rows and `m` is columns.

I will continue with the rest of the problems in a moment.



give remaining



Here are the remaining matrix-based problems with examples, optimized Java code, and time complexity analysis.

7. Spiral Order Traversal

Problem:

Traverse a 2D matrix in spiral order (clockwise).

Example:

text

```

Matrix:
1 2 3
4 5 6
7 8 9

```

Spiral Order:

text

```

1 2 3 6 9 8 7 4 5

```

Java Code:

```
java
```

```
import java.util.ArrayList;
import java.util.List;

public class SpiralTraversal {
    public static List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> result = new ArrayList<>();
        if (matrix.length == 0) return result;

        int top = 0, bottom = matrix.length - 1;
        int left = 0, right = matrix[0].length - 1;

        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++) {
                result.add(matrix[top][i]);
            }
            top++;

            for (int i = top; i <= bottom; i++) {
                result.add(matrix[i][right]);
            }
            right--;

            if (top <= bottom) {
                for (int i = right; i >= left; i--) {
                    result.add(matrix[bottom][i]);
                }
                bottom--;
            }

            if (left <= right) {
                for (int i = bottom; i >= top; i--) {
                    result.add(matrix[i][left]);
                }
                left++;
            }
        }

        return result;
    }
}
```

Time Complexity:

- **Spiral Traversal:** $O(n * m)$, where `n` is rows and `m` is columns.

8. Diagonal Traversal

Problem:

Print all elements of the matrix in diagonal order.

Example:

```
text
```

```
Matrix:
1  2  3
```



```
4 5 6
7 8 9
```

Diagonal Traversal:

text

```
1
2 4
3 5 7
6 8
9
```

Java Code:

java

```
import java.util.ArrayList;
import java.util.List;

public class DiagonalTraversal {
    public static List<Integer> diagonalOrder(int[][] matrix) {
        List<Integer> result = new ArrayList<>();
        int rows = matrix.length;
        int cols = matrix[0].length;

        for (int d = 0; d < rows + cols - 1; d++) {
            int row = (d < cols) ? 0 : d - cols + 1;
            int col = (d < cols) ? d : cols - 1;

            List<Integer> diagonal = new ArrayList<>();
            while (row < rows && col >= 0) {
                diagonal.add(matrix[row][col]);
                row++;
                col--;
            }
            result.addAll(diagonal);
        }

        return result;
    }
}
```

Time Complexity:

- **Diagonal Traversal:** $O(n * m)$, where `n` is rows and `m` is columns.

9. Find the Sum of All Elements

Problem:

Find the sum of all elements in a given matrix.

Example:

text

```
Matrix:
1 2 3
4 5 6
7 8 9
```

Sum:

text

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45

Java Code:

java

```
public class MatrixSum {
    public static int sumOfElements(int[][] matrix) {
        int sum = 0;
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                sum += matrix[i][j];
            }
        }
        return sum;
    }
}
```

Time Complexity:

- **Sum of Elements:** $O(n * m)$, where `n` is rows and `m` is columns.

10. Set Matrix Zeros

Problem:

If an element in a matrix is zero, set the entire row and column to zero.

Example:

text

```
Matrix:
1 1 1
1 0 1
1 1 1
```

Result:

text

```
1 0 1
0 0 0
1 0 1
```

Java Code:

java

```
public class SetMatrixZeros {
    public static void setZeroes(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        boolean[] rowZero = new boolean[rows];
        boolean[] colZero = new boolean[cols];

        // First pass: record the rows and columns that are zero
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (matrix[i][j] == 0) {
                    rowZero[i] = true;
                    colZero[j] = true;
                }
            }
        }

        // Second pass: set rows and columns to zero
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (rowZero[i] || colZero[j]) {
                    matrix[i][j] = 0;
                }
            }
        }
    }
}
```

Time Complexity:

- **Set Matrix Zeros:** $O(n * m)$, where `n` is rows and `m` is columns.

Summary of Time Complexities:

- **Matrix Addition/Subtraction:** $O(n * m)$
- **Matrix Transpose:** $O(n * m)$
- **Matrix Multiplication:** $O(n * m * p)$
- **Search in 2D Matrix:** $O(n + m)$
- **Rotate Matrix by 90 Degrees:** $O(n^2)$
- **Row with Maximum 1s:** $O(n + m)$
- **Spiral Order Traversal:** $O(n * m)$
- **Diagonal Traversal:** $O(n * m)$
- **Sum of Elements:** $O(n * m)$
- **Set Matrix Zeros:** $O(n * m)$

These codes solve basic to intermediate matrix problems with time complexities provided for each solution.



Intermediate to Advanced Level:

Rotate Matrix by 180 Degrees: Rotate a given $N \times N$ matrix by 180 degrees.

Matrix Region Sum Queries: Given a matrix and multiple submatrix queries, return the sum of elements within each submatrix.

Count Islands in a Matrix: Given a matrix of 0s and 1s, count the number of islands (connected components of 1s).

Find the Largest Square Submatrix with All 1s: Given a binary matrix, find the largest square submatrix that contains only 1s.

Find the Maximum Path Sum in a Matrix: Given a matrix of integers, find the path with the maximum sum starting from any cell in the first row and moving downwards.

Search a 2D Matrix II: Search for a given element in a 2D matrix sorted row-wise and column-wise.

Find Common Elements in All Rows of a Matrix: Given an $N \times M$ matrix, find the common elements present in all rows.

Find the Shortest Path in a Binary Matrix: Find the shortest path from the top-left to the bottom-right in a binary matrix where you can move in all four directions.

Word Search in a Grid: Given a 2D board of characters and a word, determine if the word exists in the grid. The word can be constructed from letters of sequentially adjacent cells.

Maximal Rectangle in a Binary Matrix: Find the largest rectangle containing only 1s in a binary matrix and return its area.



 **Memory updated**

Let's go over the intermediate to advanced matrix-based problems, including examples, optimized Java code, and time complexity analysis.

1. Rotate Matrix by 180 Degrees

Problem:

Rotate a given $N \times N$ matrix by 180 degrees.

Example:

text

```
Matrix:
1 2 3
4 5 6
7 8 9
```

Rotated Matrix (180 degrees):

text

```
9 8 7
6 5 4
3 2 1
```

Java Code:

java

```
public class RotateMatrix180 {
    public static void rotateBy180(int[][] matrix) {
        int n = matrix.length;
        for (int i = 0; i < n / 2; i++) {
            for (int j = 0; j < n; j++) {
                int temp = matrix[i][j];
                matrix[i][j] = matrix[n - 1 - i][n - 1 - j];
                matrix[n - 1 - i][n - 1 - j] = temp;
            }
        }

        if (n % 2 != 0) {
            for (int j = 0; j < n / 2; j++) {
                int temp = matrix[n / 2][j];
                matrix[n / 2][j] = matrix[n / 2][n - 1 - j];
                matrix[n / 2][n - 1 - j] = temp;
            }
        }
    }
}
```

Time Complexity:

- **Rotation:** $O(n^2)$, where n is the dimension of the matrix.

2. Matrix Region Sum Queries

Problem:

Given a matrix and multiple submatrix queries, return the sum of elements within each submatrix.

Example:

text

```
Matrix:
1 2 3
4 5 6
7 8 9
```

Query: Sum of submatrix from (1,1) to (2,2) → Sum = 5 + 6 + 8 + 9 = 28.

Java Code:

```
java
```

```
public class MatrixRegionSum {
    private int[][] prefixSum;

    public MatrixRegionSum(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        prefixSum = new int[rows + 1][cols + 1];

        // Build the prefix sum matrix
        for (int i = 1; i <= rows; i++) {
            for (int j = 1; j <= cols; j++) {
                prefixSum[i][j] = matrix[i - 1][j - 1]
                    + prefixSum[i - 1][j]
                    + prefixSum[i][j - 1]
                    - prefixSum[i - 1][j - 1];
            }
        }

        public int querySum(int row1, int col1, int row2, int col2) {
            return prefixSum[row2 + 1][col2 + 1]
                - prefixSum[row1][col2 + 1]
                - prefixSum[row2 + 1][col1]
                + prefixSum[row1][col1];
        }
    }
}
```

Time Complexity:

- **Preprocessing:** $O(n * m)$ for prefix sum matrix.
- **Query:** $O(1)$ per query.

3. Count Islands in a Matrix

Problem:

Given a matrix of 0s and 1s, count the number of islands (connected components of 1s).

Example:

```
text
```

```
Matrix:
1 1 0 0
0 1 0 1
1 0 0 1
```

Islands: 3 (three disconnected clusters of 1s).

Java Code:

java

```
public class IslandCount {
    public static int countIslands(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        boolean[][] visited = new boolean[rows][cols];
        int count = 0;

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (matrix[i][j] == 1 && !visited[i][j]) {
                    dfs(matrix, visited, i, j);
                    count++;
                }
            }
        }
        return count;
    }

    private static void dfs(int[][] matrix, boolean[][] visited, int i, int j) {
        if (i < 0 || i >= matrix.length || j < 0 || j >= matrix[0].length || matrix[i][j]
        == 0 || visited[i][j]) {
            return;
        }

        visited[i][j] = true;
        dfs(matrix, visited, i + 1, j);
        dfs(matrix, visited, i - 1, j);
        dfs(matrix, visited, i, j + 1);
        dfs(matrix, visited, i, j - 1);
    }
}
```

Time Complexity:

- **DFS Traversal:** $O(n * m)$, where `n` is rows and `m` is columns.

4. Find the Largest Square Submatrix with All 1s

Problem:

Find the largest square submatrix containing only 1s in a given binary matrix.

Example:

text

```
Matrix:
1 1 1 0
1 1 1 1
1 1 1 1
0 1 1 1
```

Largest square submatrix: Size 3x3 of all 1s.

Java Code:

```
java
```

```
public class LargestSquareSubmatrix {
    public static int largestSquare(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int[][] dp = new int[rows][cols];
        int maxSize = 0;

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (i == 0 || j == 0) {
                    dp[i][j] = matrix[i][j];
                } else if (matrix[i][j] == 1) {
                    dp[i][j] = Math.min(dp[i - 1][j], Math.min(dp[i][j - 1], dp[i - 1][j - 1])) + 1;
                }
                maxSize = Math.max(maxSize, dp[i][j]);
            }
        }

        return maxSize;
    }
}
```

Time Complexity:

- **DP Traversal:** $O(n * m)$, where `n` is rows and `m` is columns.

5. Find the Maximum Path Sum in a Matrix

Problem:

Find the path with the maximum sum starting from any cell in the first row and moving downwards, choosing cells from adjacent columns.

Example:

```
text
```

```
Matrix:
1 2 3
9 8 7
4 5 6
```

Maximum Path Sum: $1 \rightarrow 9 \rightarrow 5 = 15$.

Java Code:

```
java
```

```
public class MaximumPathSum {
    public static int maxPathSum(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int[][] dp = new int[rows][cols];
```



```

for (int j = 0; j < cols; j++) {
    dp[0][j] = matrix[0][j];
}

for (int i = 1; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        int maxPrev = dp[i - 1][j];
        if (j > 0) maxPrev = Math.max(maxPrev, dp[i - 1][j - 1]);
        if (j < cols - 1) maxPrev = Math.max(maxPrev, dp[i - 1][j + 1]);
        dp[i][j] = matrix[i][j] + maxPrev;
    }
}

int maxSum = 0;
for (int j = 0; j < cols; j++) {
    maxSum = Math.max(maxSum, dp[rows - 1][j]);
}
return maxSum;
}
}

```

Time Complexity:

- **DP Traversal:** $O(n * m)$, where `n` is rows and `m` is columns.

I'll continue with the remaining problems in a moment.



give remaining



Here are the remaining intermediate to advanced matrix-based problems:

6. Search a 2D Matrix II

Problem:

Given a 2D matrix where each row and column is sorted, search for a target value.

Example:

text

```

Matrix:
1  4  7  11
2  5  8  12
3  6  9  16
10 13 14 17

```

Target: 5 → Found

Target: 15 → Not Found

Java Code:

```
java
```

```
public class Search2DMatrixII {
    public static boolean searchMatrix(int[][] matrix, int target) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int row = 0, col = cols - 1;

        while (row < rows && col >= 0) {
            if (matrix[row][col] == target) {
                return true;
            } else if (matrix[row][col] > target) {
                col--;
            } else {
                row++;
            }
        }

        return false;
    }
}
```

Time Complexity:

- **Search:** $O(n + m)$, where `n` is the number of rows and `m` is the number of columns.

7. Find Common Elements in All Rows of a Matrix

Problem:

Find the common elements present in all rows of a given NxM matrix.

Example:

```
text
```

```
Matrix:
1 2 1 4 8
3 7 8 5 1
8 7 7 3 1
8 1 2 7 9
```

Common Element: 8 (present in all rows).

Java Code:

```
java
```

```
import java.util.HashMap;

public class CommonElementsInRows {
    public static void findCommonElements(int[][] matrix) {
        HashMap<Integer, Integer> map = new HashMap<>();
    }
}
```

```
int rows = matrix.length;
int cols = matrix[0].length;

// Initialize map with first row
for (int j = 0; j < cols; j++) {
    map.put(matrix[0][j], 1);
}

// Traverse the rest of the matrix
for (int i = 1; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        if (map.containsKey(matrix[i][j]) && map.get(matrix[i][j]) == i) {
            map.put(matrix[i][j], i + 1);
            if (i == rows - 1 && map.get(matrix[i][j]) == rows) {
                System.out.println(matrix[i][j] + " ");
            }
        }
    }
}
}
```

Time Complexity:

- **Traversal:** $O(n * m)$, where `n` is rows and `m` is columns.

8. Find the Shortest Path in a Binary Matrix

Problem:

Find the shortest path from the top-left to the bottom-right in a binary matrix (0s and 1s), where you can move in 8 directions.

Example:

text

```
Matrix:
0 1 1 0
0 0 1 1
1 0 0 1
1 1 0 0
```

Shortest Path: Length = 4.

Java Code:

java

```
import java.util.LinkedList;
import java.util.Queue;

public class ShortestPathBinaryMatrix {
    public static int shortestPathBinaryMatrix(int[][] grid) {
        int n = grid.length;
        if (grid[0][0] != 0 || grid[n - 1][n - 1] != 0) return -1;
    }
}
```

```
int[][] directions = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}, {1, 1}, {-1, -1}, {1, -1},
{-1, 1}};
boolean[][] visited = new boolean[n][n];
Queue<int[]> queue = new LinkedList<>();
queue.add(new int[]{0, 0});
visited[0][0] = true;
int pathLength = 1;

while (!queue.isEmpty()) {
    int size = queue.size();
    for (int i = 0; i < size; i++) {
        int[] current = queue.poll();
        int row = current[0], col = current[1];

        if (row == n - 1 && col == n - 1) return pathLength;

        for (int[] dir : directions) {
            int newRow = row + dir[0], newCol = col + dir[1];
            if (newRow >= 0 && newRow < n && newCol >= 0 && newCol < n &&
grid[newRow][newCol] == 0 && !visited[newRow][newCol]) {
                queue.add(new int[]{newRow, newCol});
                visited[newRow][newCol] = true;
            }
        }
        pathLength++;
    }
    return -1;
}
```

Time Complexity:

- **BFS Traversal:** $O(n^2)$, where `n` is the number of rows/columns.

9. Word Search in a Grid

Problem:

Given a 2D grid of characters and a word, determine if the word exists in the grid. The word can be constructed from letters of sequentially adjacent cells.

Example:

text

```
Grid:
A B C E
S F C S
A D E E
```

Word: "ABCCED" → Exists.

Java Code:

java

```
public class WordSearch {
    public static boolean exist(char[][] board, String word) {
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                if (dfs(board, word, i, j, 0)) {
                    return true;
                }
            }
        }
        return false;
    }

    private static boolean dfs(char[][] board, String word, int i, int j, int index) {
        if (index == word.length()) return true;
        if (i < 0 || i >= board.length || j < 0 || j >= board[0].length || board[i][j] !=
word.charAt(index)) {
            return false;
        }

        char temp = board[i][j];
        board[i][j] = '#'; // Mark as visited

        boolean found = dfs(board, word, i + 1, j, index + 1)
            || dfs(board, word, i - 1, j, index + 1)
            || dfs(board, word, i, j + 1, index + 1)
            || dfs(board, word, i, j - 1, index + 1);

        board[i][j] = temp; // Restore original value
        return found;
    }
}
```

Time Complexity:

- **DFS Traversal:** $O(n * m * 4^k)$, where `n` is rows, `m` is columns, and `k` is the length of the word.

10. Maximal Rectangle in a Binary Matrix

Problem:

Find the largest rectangle containing only 1s in a binary matrix and return its area.

Example:

text

```
Matrix:
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
```

Maximal Rectangle Area: 6.

Java Code:

java

```
import java.util.Stack;

public class MaximalRectangle {
    public static int maximalRectangle(char[][] matrix) {
        if (matrix.length == 0) return 0;
        int maxArea = 0;
        int[] heights = new int[matrix[0].length];

        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                if (matrix[i][j] == '1') {
                    heights[j]++;
                } else {
                    heights[j] = 0;
                }
            }
            maxArea = Math.max(maxArea, largestRectangleArea(heights));
        }

        return maxArea;
    }

    private static int largestRectangleArea(int[] heights) {
        Stack<Integer> stack = new Stack<>();
        int maxArea = 0;
        for (int i = 0; i <= heights.length; i++) {
            int h = (i == heights.length) ? 0 : heights[i];
            while (!stack.isEmpty() && h < heights[stack.peek()]) {
                int height = heights[stack.pop()];
                int width = stack.isEmpty() ? i : i - stack.peek() - 1;
                maxArea = Math.max(maxArea, height * width);
            }
            stack.push(i);
        }
        return maxArea;
    }
}
```

Time Complexity:

- **Traversal:** $O(n * m)$, where `n` is rows and `m` is columns.

Summary of Time Complexities:

- **Rotate Matrix by 180 Degrees:** $O(n^2)$
- **Matrix Region Sum Queries:** $O(n * m)$ preprocessing, $O(1)$ query
- **Count Islands in a Matrix:** $O(n * m)$
- **Largest Square Submatrix with All 1s:** $O(n * m)$
- **Maximum Path Sum in a Matrix:** $O(n * m)$
- **Search a 2D Matrix II:** $O(n + m)$
- **Find Common Elements in Rows:** $O(n * m)$
- **Shortest Path in a Binary Matrix:** $O(n^2)$
- **Word Search in a Grid:** $O(n * m * 4^k)$
- **Maximal Rectangle in a Binary Matrix:** $O(n * m)$

These solutions cover a variety of matrix-based problems with optimal approaches.

ChatGPT can make mistakes. Check important info.