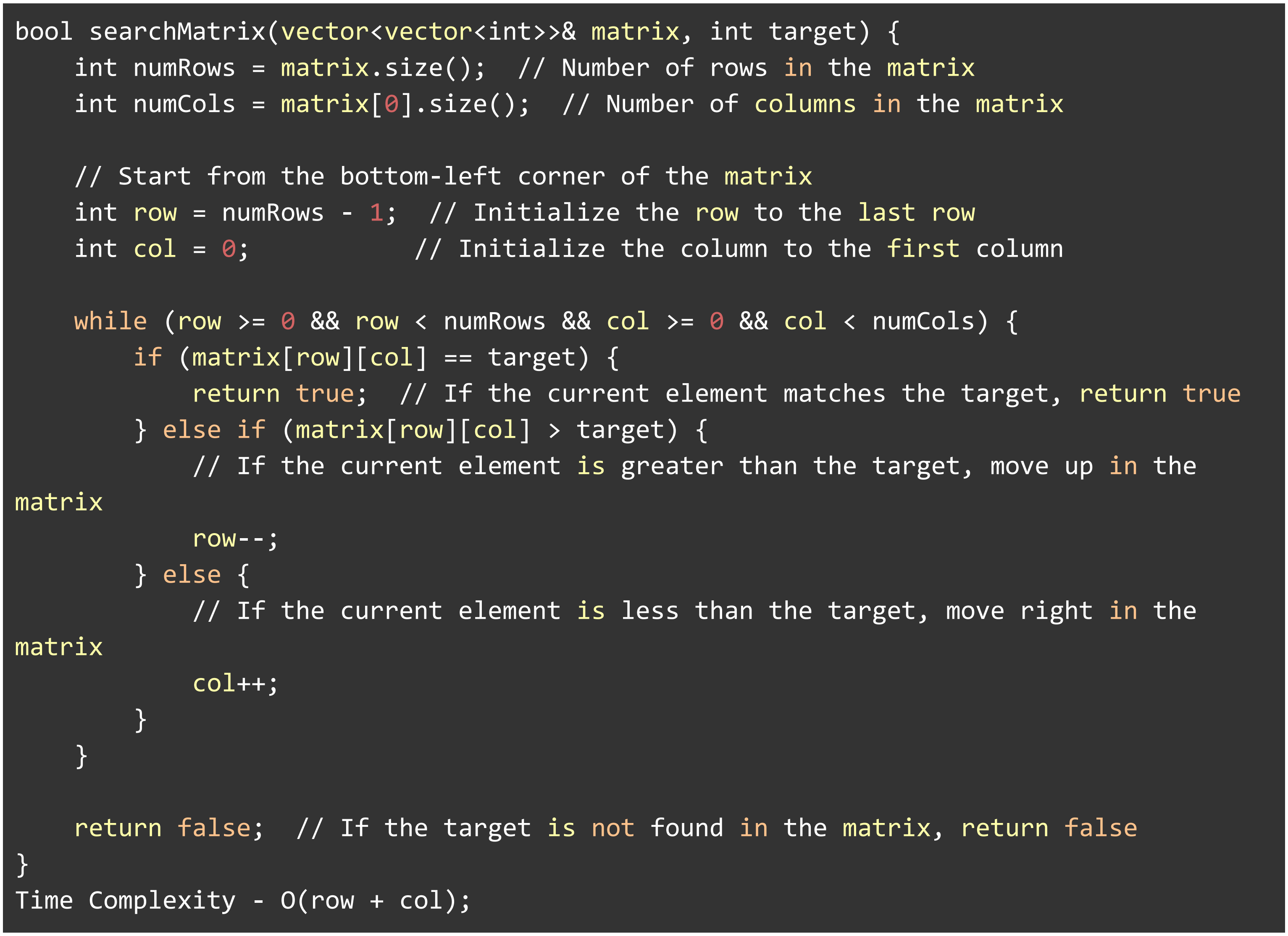
Day 46/180 Binary Search in 2D Arrays

1:Binary Search:

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int n = matrix.size();      // Number of rows in the matrix
    int m = matrix[0].size();   // Number of columns in the matrix

    for (int row = 0; row < n; row++) {
        int l = 0;              // Initialize the left pointer for binary search
        int r = m - 1;          // Initialize the right pointer for binary search

        while (l <= r) {
            int mid = (l + r) / 2;  // Calculate the middle index in the current 
row

            if (matrix[row][mid] == target) {
                return true;      // If the middle element in the current row 
matches the target, return true
            } else if (matrix[row][mid] < target) {
                l = mid + 1;       // If the middle element is less than the 
target, search in the right half
            } else {
                r = mid - 1;       // If the middle element is greater than the 
target, search in the left half
            }
        }
    }

    return false;  // If the target is not found in the matrix, return false
}

Time Complexity - O(row * log(col));
O(row) for traversing each row * O(log(col)) for searching element in each row;
```

Optimal Solution:



```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int numRows = matrix.size();  // Number of rows in the matrix
    int numCols = matrix[0].size();  // Number of columns in the matrix

    // Start from the bottom-left corner of the matrix
    int row = numRows - 1;  // Initialize the row to the last row
    int col = 0;           // Initialize the column to the first column

    while (row >= 0 && row < numRows && col >= 0 && col < numCols) {
        if (matrix[row][col] == target) {
            return true;  // If the current element matches the target, return true
        } else if (matrix[row][col] > target) {
            // If the current element is greater than the target, move up in the
matrix
            row--;
        } else {
            // If the current element is less than the target, move right in the
matrix
            col++;
        }
    }

    return false;  // If the target is not found in the matrix, return false
}
Time Complexity - O(row + col);
```

2:Search in a sorted row-col wise Matrix:

```
//Function to search a given number in row-column sorted matrix.
    bool search(vector<vector<int> > matrix, int n, int m, int target)
    {

        // Start from the bottom-left corner of the matrix
        int row = n - 1;  // Initialize the row to the last row
        int col = 0;           // Initialize the column to the first column

        while (row >= 0 && row < n && col >= 0 && col < m) {
            if (matrix[row][col] == target) {
                return true;  // If the current element matches the target, return true
            } else if (matrix[row][col] > target) {
                // If the current element is greater than the target, move up in the matrix
                row--;
            } else {
                // If the current element is less than the target, move right in the matrix
                col++;
            }
        }

        return false;  // If the target is not found in the matrix, return false
    }
Time Complexity : O(n + m), Space complexity: O(1)
```

3: Count zeros in a sorted matrix

```
int countZeros(vector<vector<int>> a) {
    int ans = 0;  // Initialize a variable to store the count of zeros

    for (int i = 0; i < a.size(); i++) {
        int l = 0;              // Initialize the left pointer for binary search
        int r = a[0].size() - 1; // Initialize the right pointer for binary search
        int cnt = -1;           // Initialize a variable to store the index of the last zero in the row

        while (l <= r) {
            int mid = (l + r) / 2;  // Calculate the middle index in the current row

            if (a[i][mid] == 0) {
                cnt = mid;          // If the middle element is 0, update 'cnt' and move the left pointer to the right
                l = mid + 1;
            } else {
                r = mid - 1;         // If the middle element is 1, move the right pointer to the left
            }
        }

        ans += cnt + 1;  // Increment the count of zeros with the index of the last zero + 1
    }

    return ans;  // Return the total count of zeros in the matrix
}
Time complexity - n * log(n)
```

4: Row with max 1s

```
int rowWithMax1s(vector<vector<int>> a, int n, int m) {
    int ans = 0;  // Initialize a variable to store the maximum count of 1s found in a row
    int row = -1; // Initialize a variable to store the index of the row with the maximum count of 1s (initially set to -1)

    for (int i = 0; i < n; i++) {
        int l = 0;              // Initialize the left pointer for binary search
        int r = m - 1;          // Initialize the right pointer for binary search
        int cnt = -1;           // Initialize a variable to store the index of the
last 0 in the row

        while (l <= r) {
            int mid = (l + r) / 2;  // Calculate the middle index in the current
row

            if (a[i][mid] == 0) {
                cnt = mid;         // If the middle element is 0, update 'cnt' and
move the left pointer to the right
                l = mid + 1;
            } else {
                r = mid - 1;        // If the middle element is 1, move the right
pointer to the left
            }
        }

        if (ans < m - cnt - 1) {
            ans = m - cnt - 1;  // Update 'ans' with the count of 1s in the current
row (subtracting 1 to account for 0-based indexing)
            row = i;            // Update 'row' with the index of the row with the
maximum count of 1s
        }
    }

    return row;  // Return the index of the row with the maximum count of 1s, or -1
if no row contains 1s
}
Time Complexity - n*log(n)
```

5: Binary Search: (Solve it in log(n)+log(m) time, where n is number of row and m is number of columns)

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int m = matrix.size();          // Number of rows in the matrix
    int n = matrix[0].size();       // Number of columns in the matrix
    int left = 0;                   // Initialize the left pointer for binary search
    int right = m * n - 1;          // Initialize the right pointer for binary search

    while (left <= right) {
        int mid = left + (right - left) / 2;   // Calculate the middle index in the 1D representation of the matrix
        int mid_val = matrix[mid / n][mid % n];  // Calculate the value at the middle index in the 2D matrix

        if (mid_val == target) {
            return true;  // If the middle element matches the target, return true
        } else if (mid_val < target) {
            left = mid + 1;  // If the middle element is less than the target, move the left pointer to the right
        } else {
            right = mid - 1; // If the middle element is greater than the target, move the right pointer to the left
        }
    }

    return false;  // If the target is not found in the matrix, return false
}
Time Complexity - log(n) * log(m)
```

6: Binary Search in a 2D array which is sorted in decreasing order. N is the number of rows and M is the number of columns.

=> Same as first problem

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int n = matrix.size();      // Number of rows in the matrix
    int m = matrix[0].size();   // Number of columns in the matrix

    for (int row = 0; row < n; row++) {
        int l = 0;              // Initialize the left pointer for binary search
        int r = m - 1;          // Initialize the right pointer for binary search

        while (l <= r) {
            int mid = (l + r) / 2;  // Calculate the middle index in the current
row

            if (matrix[row][mid] == target) {
                return true;      // If the middle element in the current row
matches the target, return true
            } else if (matrix[row][mid] > target) {
                l = mid + 1;       // If the middle element is less than the
target, search in the right half
            } else {
                r = mid - 1;       // If the middle element is greater than the
target, search in the left half
            }
        }
    }

    return false;  // If the target is not found in the matrix, return false
}
Time complexity = n * log(m)
```