Binary Search in a 2D Matrix

The goal is to search for a **target value** in a **row-wise and column-wise sorted matrix**. In such a matrix:

- Each row is sorted in ascending order.
- Each column is sorted in ascending order.

Approach: Optimized Search

1. Start at the **top-right corner** of the matrix:
    - **Advantages**:
        - You can move left if the current value is greater than the target.
        - You can move down if the current value is less than the target.
2. Compare the value at the current position with the target:
    - If the value equals the target, return true (or the position).
    - If the value is greater than the target, move left (col--).
    - If the value is less than the target, move down (row++).
3. Stop when the indices go out of bounds (row >= m or col < 0).

Algorithm

```
1. Start at the top-right corner: (row = 0, col = n - 1).
2. While row < m and col >= 0:
  a. If matrix[row][col] == target, return true.
  b. If matrix[row][col] > target, move left: col--.
  c. If matrix[row][col] < target, move down: row++.
3. If the loop ends, return false (target not found).
```

C++ implementation

```
#include <iostream>
#include <vector>
using namespace std;

bool searchMatrix(const vector<vector<int>>& matrix, int target) {
    int m = matrix.size(), n = matrix[0].size();
    int row = 0, col = n - 1; // Start at the top-right corner

    while (row < m && col >= 0) {
        if (matrix[row][col] == target) {
            return true;
        } else if (matrix[row][col] > target) {
            col--; // Move left
        } else {
            row++; // Move down
        }
    }

    return false; // Target not found
}

int main() {
    int m, n, target;
    cin >> m >> n;
    vector<vector<int>> matrix(m, vector<int>(n));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> matrix[i][j];
        }
    }

    cin >> target;

    if (searchMatrix(matrix, target)) {
        cout << "Target found in the matrix." << endl;
    } else {
        cout << "Target not found in the matrix." << endl;
    }

    return 0;
}
```

Java

```
import java.util.Scanner;

public class SearchIn2DMatrix {

    public static boolean searchMatrix(int[][] matrix, int target) {
        int m = matrix.length, n = matrix[0].length;
        int row = 0, col = n - 1; // Start at the top-right corner

        while (row < m && col >= 0) {
            if (matrix[row][col] == target) {
                return true;
            } else if (matrix[row][col] > target) {
                col--; // Move left
            } else {
                row++; // Move down
            }
        }

        return false; // Target not found
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int m = sc.nextInt();
        int n = sc.nextInt();

        int[][] matrix = new int[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }

        int target = sc.nextInt();

        if (searchMatrix(matrix, target)) {
            System.out.println("Target found in the matrix.");
        } else {
            System.out.println("Target not found in the matrix.");
        }

        sc.close();
    }
}
```

Example Input/Output

Input 1:

CopyEdit

```
3 4
1 4 7 11
2 5 8 12
3 6 9 16
5
```

Output:

CopyEdit

```
Target found in the matrix.
```

Explanation:

- The target 5 is located at (1, 1).

Input 2:

CopyEdit

```
3 4
1 4 7 11
2 5 8 12
3 6 9 16
10
```

Output:

```
Target not found in the matrix.
```

Complexity

1. **Time Complexity**: O(m+n)
    - At most, m+n moves are made (either down or left).
2. **Space Complexity**: O(1)
    - Only a few variables are used.