Strivers-A2Z-DSA-Sheet-main\02.Binary Search\2D Arrays\3.Search_in_rowwise_sorted_matrix.cpp

```
/*
QUESTION:
Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:
- Integers in each row are sorted in ascending order from left to right.
- Integers in each column are sorted in ascending order from top to bottom.

Example 1:
Input: matrix = [
  [1, 4, 7, 11, 15],
  [2, 5, 8, 12, 19],
  [3, 6, 9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
], target = 5
Output: true
Explanation: The element 5 is present in the matrix.

Example 2:
Input: matrix = [
  [1, 4, 7, 11, 15],
  [2, 5, 8, 12, 19],
  [3, 6, 9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
], target = 20
Output: false
Explanation: The element 20 is not present in the matrix.

APPROACH:
We can start the search from the top-right element or the bottom-left element and move towards the target element based on the properties of the matrix.

1. Initialize the current position to the top-right element (row = 0, col = n-1), where n is the number of columns in the matrix.
2. While the current position is within the matrix boundaries:
     - If the current element is equal to the target, return true.
     - If the current element is greater than the target, move left to the previous column.
     - If the current element is less than the target, move down to the next row.
3. If the loop exits without finding the target, return false.

CODE:
*/

bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int row = 0, col = matrix[0].size() - 1;
    while (row < matrix.size() && col >= 0) {
        if (matrix[row][col] == target)
            return true;
        else if (matrix[row][col] > target)
            col--;
        else
```

```
            row++;
    }
    return false;
}

/*
Time Complexity: The time complexity of this algorithm is O(m + n), where m is the number of rows and n is the number of columns in the matrix.
                In the worst case, we may need to traverse through the entire row or column of the matrix.
Space Complexity: The space complexity is O(1) since we are using constant extra space.
*/
```