LeetCode 240: Search in 2D Matrix

or Problem Statement

Given a 2D matrix where:

- ✓ Integers in each row are sorted in ascending order (left → right)
- ✓ Integers in each column are sorted in ascending order (top → bottom)

Find an efficient way to search for a target value.

💡 The Elegant Solution: Staircase Search

```
class Solution {
public:
  bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int rows = matrix.size();
    int cols = matrix[0].size();
    // Start from top-right corner
    int row = 0;
    int col = cols - 1;
    while (row < rows && col \geq 0) {
       if (matrix[row][col] == target) {
         return true;
                              // Found it! 🎉
       } else if (matrix[row][col] > target) {
                          // Too big? Move left 🔚
         col--;
       } else {
                              // Too small? Move down 🕕
         row++;
```

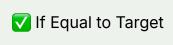
LeetCode 240: Search in 2D Matrix

```
}
return false; // Not found 😢
}
};
```

Visual Strategy

```
graph TD
A["Start at Top-Right"] \rightarrow B\{"Current = Target?"\}
B \rightarrow |"Yes"| C["Return True "s"]
B \rightarrow |"No"| D\{"Current > Target?"\}
D \rightarrow |"Yes"| E["Move Left "s"]
D \rightarrow |"No"| F["Move Down "s"]
E \rightarrow B
F \rightarrow B
```

M How It Works



Return true

Mission accomplished!

If Greater Than
Target

f Less Than
Target

Move left to find smaller values

Move down to find larger values

Performance Analysis

Metric	Value	Explanation
Time Complexity	O(m + n)	Linear path through matrix
Space Complexity	O(1)	Constant extra space

6 Key Takeaways

- Start from top-right corner for optimal search path
- Use matrix properties to eliminate search space
- Movement decisions based on current value comparison
- Linear time complexity makes it highly efficient

Pro Tip: This approach is also known as the "Search Space Reduction" technique, as each comparison eliminates either a row or a column from consideration.