Staircase Search / Search in Sorted Matrix

> Staircase Search is a smart algorithm to find an element in a *sorted matrix* (a grid where rows and columns are sorted).

Imagine a staircase where each step (row) has numbers in order, and each column also goes up or down in order. This method uses that sorted property to skip large chunks of the matrix, making it faster than checking every item! 🎯

Real-Life Example:

Imagine searching for a book in a library where each shelf (row) is sorted by genre, and each row's positions (columns) are sorted by publication year. You don't check every shelf—you start from the top-right corner and move strategically! 📚🔍

🔍 How Staircase Search Works

The algorithm starts at the **top-right corner** of the matrix. Based on comparisons, it moves:

- **Down** if the current element is smaller than the target (to explore larger values in the next row).
- **Left** if the current element is larger (to explore smaller values in the previous column).

Why Start at Top-Right?

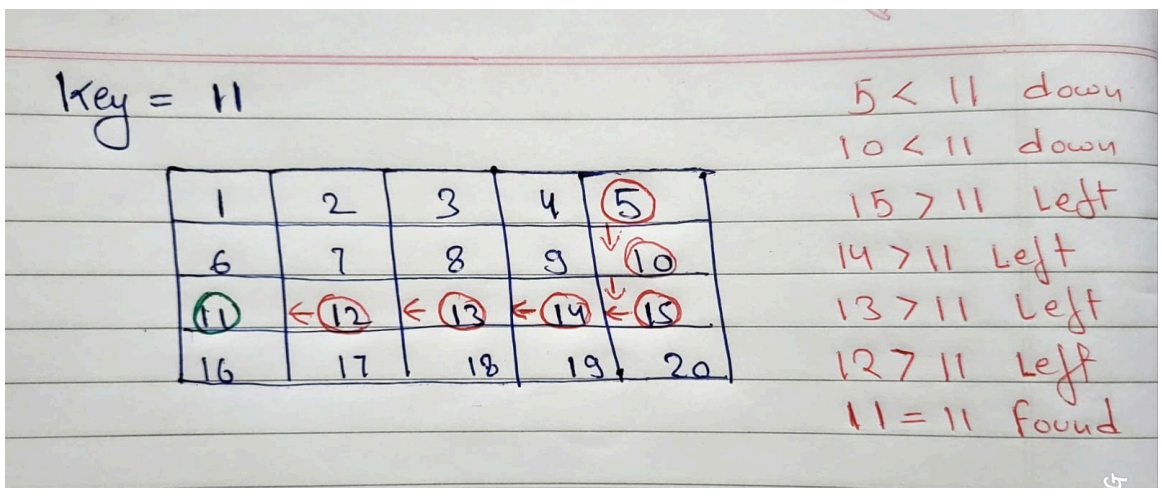
Because it balances the sorted rows and columns, allowing efficient elimination of rows or columns at each step! 💡

Steps in Detail:

1. Begin at the top-right corner of the matrix.
2. Compare the current element with the target.
3. If equal, you found it!
4. If smaller, move down (to a larger value).
5. If larger, move left (to a smaller value).
6. Repeat until you find the target or go out of bounds.

Time Complexity: O(m + n) – where *m* is rows and *n* is columns.

This is much faster than checking every element (O(m*n))!



💻 Java Code Example

Let's see how to implement this in Java. We'll write a function to search for a target in a sorted matrix!

```
public class StaircaseSearch {
    public static boolean search(int[][] matrix, int target) {
        int rows = matrix.length;
        if (rows == 0) return false; // No rows? No target!
        int cols = matrix[0].length;
        int row = 0; // Start at top row
        int col = cols - 1; // Start at rightmost column
```

```
        while (row < rows && col >= 0) { // Keep moving while in bounds
            if (matrix[row][col] == target) {
                return true; // Found it! 🎉
            } else if (matrix[row][col] < target) {
                row++; // Move down to a larger value
            } else {
                col--; // Move left to a smaller value
            }
        }
        return false; // Target not found 😕
    }
}
```

Code Breakdown:

- `row` and `col` track our current position.
- The `while` loop runs as long as we're within matrix limits.
- Each comparison directs us **down** or **left**, shrinking the search area.

🧪 Self-Check Questions

1. **Why start at the top-right corner?**

   (Think: How does this help eliminate rows/columns?)

2. **What happens if the target is smaller than all elements?**

   (Hint: Which direction would you move?)

🧤 Real-World Use Cases

- Searching in sorted datasets (e.g., databases).
- Pathfinding in grids (like games).

📚 Quick Recap 🎯

- Staircase Search works on **sorted matrices** (rows/columns sorted).
- Start at the **top-right** corner.
- Use comparisons to move **down** or **left**.
- Time-efficient: O(m + n) time.