Input: 2D array of distinct integers and a target integer

```
matrix = [
  [1   4   7   12  15  1000]
  [2   5   19  31  32  1001]
  [3   8   24  33  35  1002]
  [40  41  42  44  45  1003]
  [99  100 103 106 128 1004]
]
target = 44
```

Each row is sorted and each column is sorted
height != = width necessarily

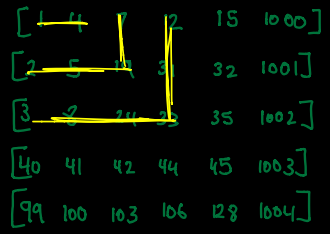
Output: An array of the row and column indices of the target integer if it's contained in the matrix, otherwise [-1,-1]
[3, 3]

---

Cannot do binary search because:

[1 4 7 12 15 1000]
[2 5 19 31 32 1001]
[3 8 24 33 35 1002]
[40 41 42 44 45 1003]
[99 100 103 106 128 1004]

If we start choose the mid (33) we'd eliminate the values above and to the left of it (since 33 < 44):



But where do we go from here? We cannot perform binary search again as we could be in any quadrant

---

Idea:
We will perform a stair search. Start in the first row, last column. Since this is the largest value in the row, if our target > this value, move down otherwise move left.
This is because all values to the left are smaller and all values below are larger since rows & columns are sorted

Time:
O(n+m) (where n and m are the rows and columns respectively) since at worst, we'd have the target in the last row, first column so we'd have to traverse n rows and m columns

Space:
O(1) since we are not using any more space as n and m grow

```
// O(n + m) time | O(1) space
function searchInSortedMatrix(matrix, target) {
  let i = 0;
  let j = matrix[0].length - 1;

  while (i < matrix.length && j >= 0) {
    let currValue = matrix[i][j];
    if (target === currValue) {
      return [i, j];
    } else if (target < currValue) {
      j--;
    } else {
      i++;
    }
  }
  return [-1, -1];
}
```