题目：

编写一个高效的算法来判断 *m* x *n* 矩阵中，是否存在一个目标值。该矩阵具有如下特性：

- 每行中的整数从左到右按升序排列。
- 每行的第一个整数大于前一行的最后一个整数。

示例 1：

```
输入：
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 3
输出：true
```

示例 2：

```
输入：
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 13
输出：false
```

思路：

方法一：二分法

看题目，可以看出，这个矩阵如果一行一行的放到数组里，是一个升序的数组

所有这个就相当于在一个m*n的升序数组里查找是否有target，可以用二分查找

数组中的下标和矩阵的下标的对应关系：

行：mid // n

列：mid % n

```python
def searchMatrix(self, matrix, target):
        """
        :type matrix: List[List[int]]
        :type target: int
        :rtype: bool
        """
        m = len(matrix)
        if m == 0:
            return False
         n = len(matrix[0])

         left, right = 0, m * n - 1
```

```
        while left <= right:
            mid = (left + right) // 2
            mid_num = matrix[mid // n][mid % n]
            if target == mid_num:
                return True
            else:
                if target < mid_num:
                    right = mid - 1
                else:
                    left = mid + 1
        return False

```

时间复杂度：O(log m *n)
空间复杂度：O(1)

方法二：
以行为单位，从左下角向右上角递归

左下角的数是那一列最大的，同时也是那一行最小的
所以，用左下角的数与target比较大小：
如果 > target：需要上移
如果 < target：需要➡移
如果相等，则放回true

```
def searchMatrix(self, matrix, target):
        """
        :type matrix: List[List[int]]
        :type target: int
        :rtype: bool
        """
        m = len(matrix)
        if m == 0 :
            return False
         n = len(matrix[0])
         if  n == 0:
             return False
         x = m-1
         y = 0
         while x >= 0 and y < n:
             if matrix[x][y] > target:
                 x -= 1
             elif matrix[x][y] < target:
                 y += 1
```

```
            else:
                return True
        return False
```

时间复杂度:O(m * n)

空间复杂度: O(1)