在一个二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

记住：这个第一行的最后一个数字不一定小于下一行的第一个数字

```
public class B1 {
/*
 * 暴力时，从第一行第一列开始查找，查找时按行优先，由于数组向下和向右时递增的，故如果从第一行第一列查找如果target比当前元素大，
 * 那么向下走和向右走都是可以的，这样很麻烦，但是如果从左下角或右上角开始查找，问题就变得简单了，以从左下角开始查找为例，如果当前
 * 元素比target大，那么就往上走；如果target比当前元素小，就往右走。如果走出了这个矩阵还没有找到target那么返回false。
这样做为什么是可行的呢？把数组想象成一个棋盘，起点是左下角，终点是target(棋盘中的某个位置)。其实就是找一条最短路从起点到终点
，那么判断的条件是如果当前元素比target大，那么就往上走；如果target比当前元素小，就往右走，可能你会想，当前元素比target ，
往左走也是可以的啊，确实可以，但是你这样不是多走了一些路吗，如果最短路走不到target，那么你往左走也是行不通的。
时间复杂度：O(2n)O(2n) 。
 */
    public static boolean Find(int [][] array, int target) {
        int n = array.length;
        int m = array[0].length;
        //从左下角开始
        int r = n - 1, c = 0;
        while (r >= 0 && c < m) {
            if(target == array[r][c]) {
                return true;
            }else if (target < array[r][c]) {
                r --;
            }else {
                c ++;
            }
        }
        return false;
    }

    public static void main(String [] args) {
        int [][] array = new int [][] {{1,2,8,9},{2,4,9,12},{4,7,10,13},{6,8,11,15}};
        System.out.println(Find(array, 14));
    }
}
```