//team ice cream--James Huang, Shafali Gupta, Kevin Feng
//apcs pd01
//hw2--speaking in pseudocode
//2018-01-31
the Situation:
Consider an *n* x *n* 2D array of ints, wherein numbers increase across any row (L->R) and down any column...
e.g.,
| 1 3 5 |
| 3 7 8 |
| 5 12 15 |
Your Goal:
Devise a O(n) search algorithm, expressed as one or more of these:
PLAN
We used recursion to go through each element in the 2D array. The runtime is O(n) because there are no nested for loops. The runtime does depend on the size of the 2D array of ints, as n comparisons are needed in the worst case scenario!
Psuedocode?

```
a=new int[n][n])
rowValue=n
colValue=n
}
isThere(row,col,number) { //recursion
        if(a[row][col]==number) {
                Return {row,col} //number is found
}
        Else if (rowValue>row && colValue>col) {
                IsThere(row, col + 1, number) //look for it in the position to the right THE
        POSITION ON THE RIGHT IS COL + 1
}

        Else if(ColValue<=col) {
                col=0//resets column
                isThere(row+1,col,number) //recursive call to compare to the next element in the
        matrix
}
        Else {
                System.out.println("could not find, returning -1 as coordinate");
                Return {-1,-1}; //if not in the matrix, return -1

}

Main method{
```

```
isThere(0,0, value);
}
}
```
// In this plan, we use recursion to find the value given. We compare the value with each element of the matrix until we find it, unless it is not in it, then wwe return -1. Each time, either the row or column increases by 1 until it is at the end of the row and it resets the column.
this