Team: Anything (Ricky Lin, Susan Lin, Bo Hui Lu)
APCS2 pd2
Hw02: Speaking in Pseudocode
2018-01-31

Ex:
| 1 3 5 |
| 3 7 8 |
| 5 12 15 |

Looking for the number 8.

1.Start off at either the top right corner or the bottom left corner.

- **For this example, we'll start at the bottom left. If you want to use the top right corner, just do the reverse for the decreasing/increasing of each row/column, and set the start to that corner instead.**

2.Have **two counter**s, one for **row number**, and one for **column number**. Set row number to the # of rows -1, and set column number to 0.

3. Have a while loop set to while numWeWant != elem.
   - Include a conditional, to check if either the column number or row number exceeds/is lower than the possible range of indexes. This is to prevent our loop from running forever if the number doesn't exist in the 2d array, since either the row/column counter will eventually go over/below the range of indexes.

4.Compare the element at [row counter][column counter], with the number we're looking for. **(The main part of our method) If the elem at that corner is greater than the number we're looking for, then increase row counter by 1. If the elem at the corner is less than the number we're looking for, then add one to the column counter. The new element we're looking at is at the updated row and column. Compare this element to the number we're looking for.**

5.Repeat step 4 till number is found.

In our example, the first number we start off with will be 5. Since 5 is less than the number 8, we will shift over to the next column, while keeping our row the same. The new element is 12. Since 12>8, we move up one row instead, while maintaining our column constant. The new element is then 7. Since 7<8, we shift the column to the right 1, which finally brings us to 8, the number we're looking for.

This code works, since we're relying on the fact that numbers to the right will be > the numbers on the left, and that numbers above will always be < the number below. It's able to quickly eliminate rows/columns that can't possibly be where the number is, since it would be either too large/not big enough if we continued in that direction. This is O(n), because it only needs to iterate through the 2D-array once to find the number.

The one issue with this code is that, it only works if we're trying to confirm if a number exists within our 2D-array. If we were instead asked to look for the first occurence of that number, this method would not always work.