Tianming Zhuang
100875151

SYSC2100B Assignment 4

For convenience, I will henceforth refer to the element we're searching for as *k* at (i,j), in the array and the element we're currently at as *a* at (x,y). The idea behind this search is that, we start at an element that is smallest in its row and largest in its column.

If that element is *k*, then we've found it; great!

Now if *k*>*a*, then we can eliminate the column *a* is in, since *a* is the largest in its column (and the element we're looking for is larger). Now we look at a at (x,y+1), moving right, since again, a will be the smallest in its row (since we eliminated the first column), but largest in its column.

Likewise, if *k*<*a*, then we can eliminate its row and move upwards, to *a* at (x-1, y). In this fashion, we move until either *k* is found, or we move out of bounds of the array, in which case it doesn't exist in the array.

The worst case scenario is if we have to zigzag from one corner to the other. Given an *n*x*m* array, where *n* is approximately equal to *m*, that would be *n*-1 steps up, and *m*-1 steps right. This gives O(n+m) = O(2n) = O(n) if our assumption holds. If *m*>>*n*, or vice versa, then the runtime complexity for the worst case scenario would just be O(m) or O(n) respectively.

Note, we could also start from top right corner, eliminating rows smaller than it, and columns larger than it.

Also note, but once you reach the boundary of the array in either direction, you could perform a binary search along the remaining elements of the row if you hit the end of the column, or vice versa. For *n* approximately equal to *m*, this makes no difference. But for cases where *m*>>*n*, or vice versa, our worst case becomes O(log(m)) and O(log(n)) respectively. This is not implemented, but would be an improvement.