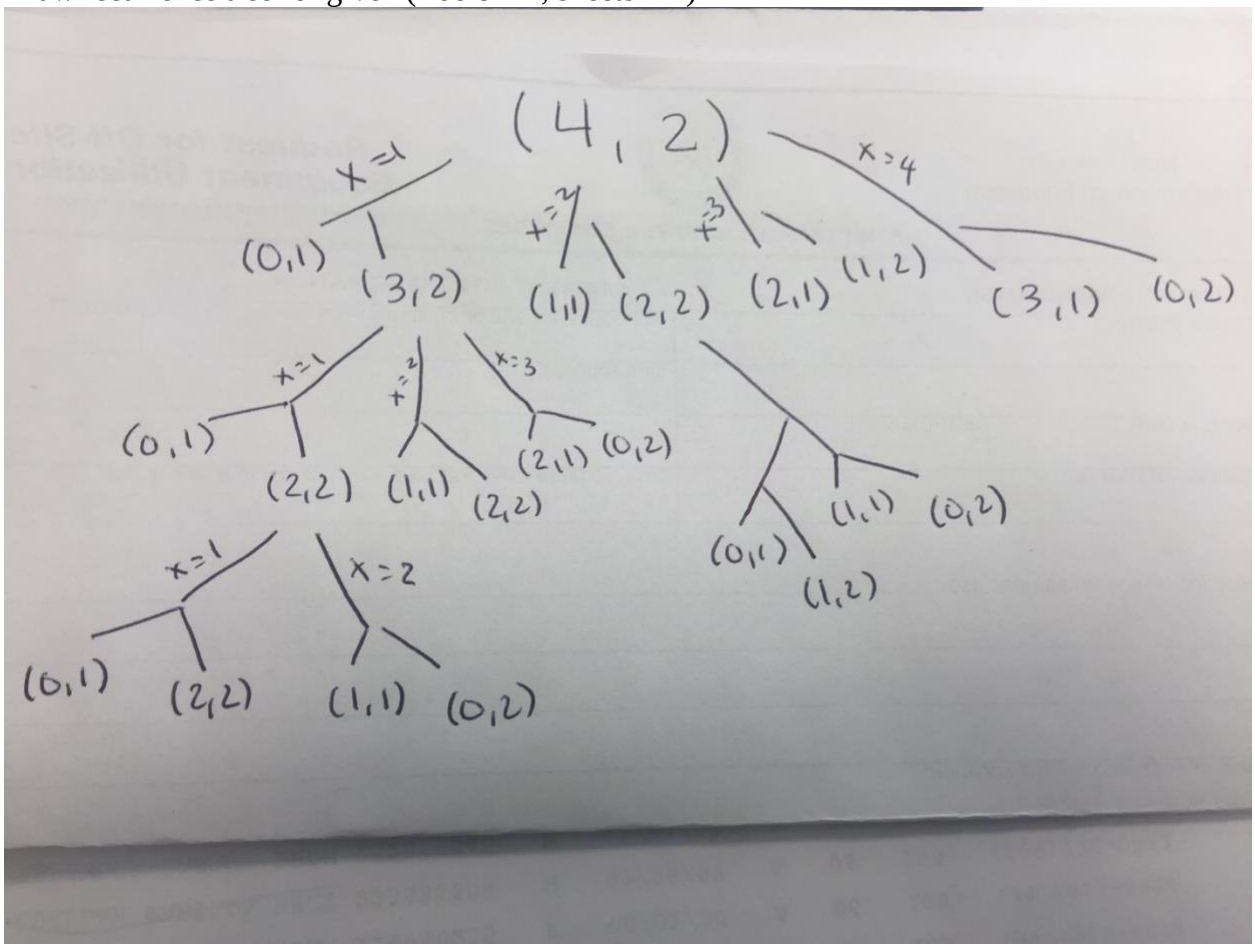
1. Falling Glass

a) Describe the optimal substructure/recurrence that would lead to a recursive solution
   - When we drop the glass from a floor x, it can either break or not break. If the glass doesn't break from the selected xth floor, we would check from the floors above x, which leads to a recursive solution.

b) Draw recurrence tree for given (floors = 4, sheets = 2)



c) Code your recursive solution under GlassFallingRecur(int n numFloors, int m numGlass)

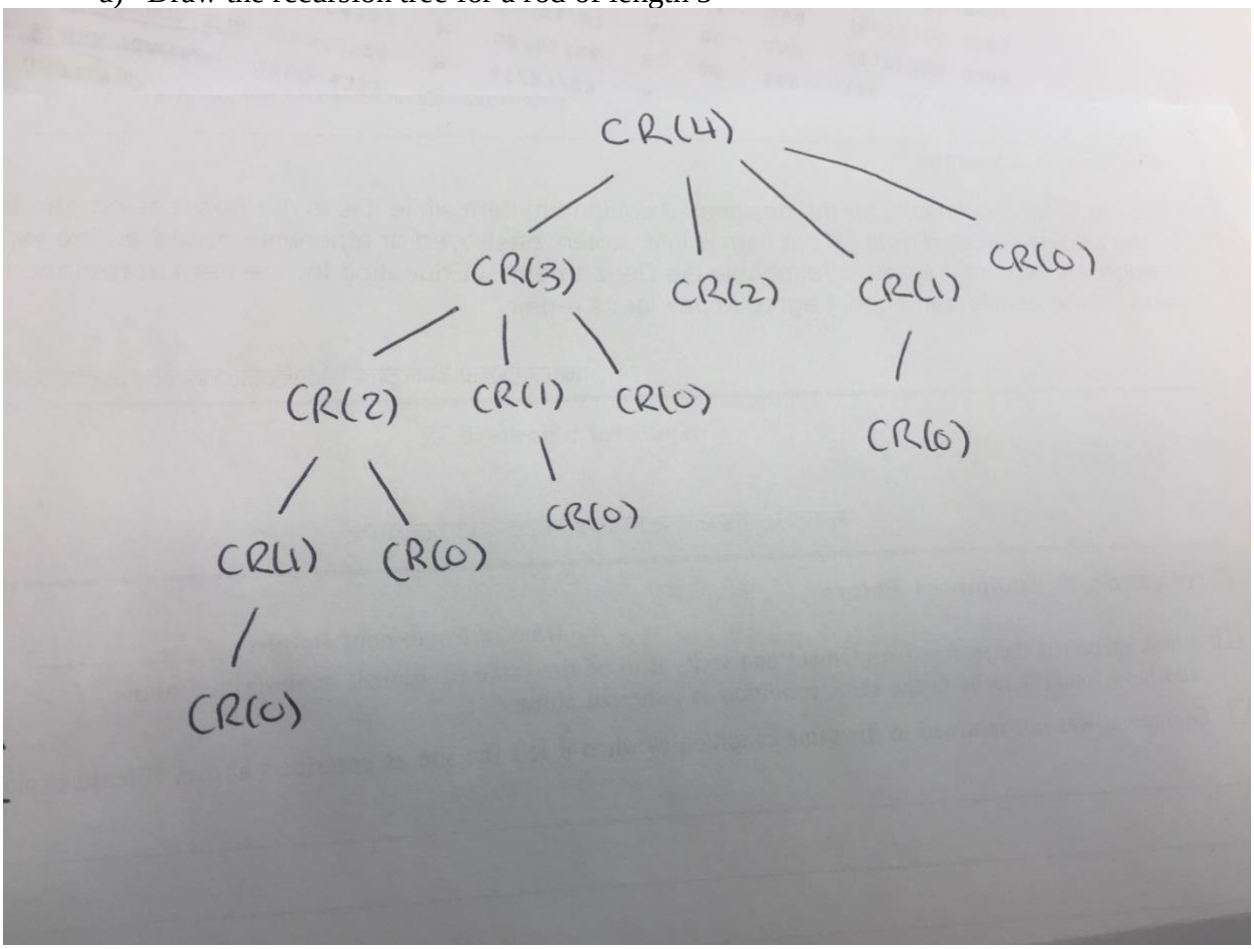
d) How many distinct subproblems do you end up with given 4 floors and 2 sheets?
   - 9

e) How many distinct subproblems for n floors and m sheets?
   - n*m+n-m-1

f) Describe how you would memoize GlassFallingRecur
   - In order to memorize the GlassFallingRecur, I would make a new 2D array and for each arr[i] [j] I would store minimum number of trails needed for i sheets and j floors.

2. Rod Cutting

a) Draw the recursion tree for a rod of length 5



b) On page 370: answer 15.1-2 by coming up with a counterexample, meaning come up with a situation / some input that shows we can only try all the options via dynamic programming instead of using a greedy choice.
   - If the rod length is 4, the optimal way of solving the problem would be to cut the rod in two lengths of 2, which would give a total of $44. If you do the greedy strategy, you would cut a rod length of 3 with revenue 33 and then a length of 1 for a price of 1, which would make the total into 37. Through this, it can be seen that the greedy strategy does not always give the optimal solution to cutting rods.
   - I referenced these website to answer the question:
     http://ranger.uta.edu/~huang/teaching/CSE5311/HW3_Solution.pdf
     https://gradebuddy.com/doc/3068224/my-assignment4

c) Code the memoized recursive version in RodCutting.java under rodCuttingRecur

d) Code the bottom-up solution in RodCutting.java under rodCuttingBottomUp