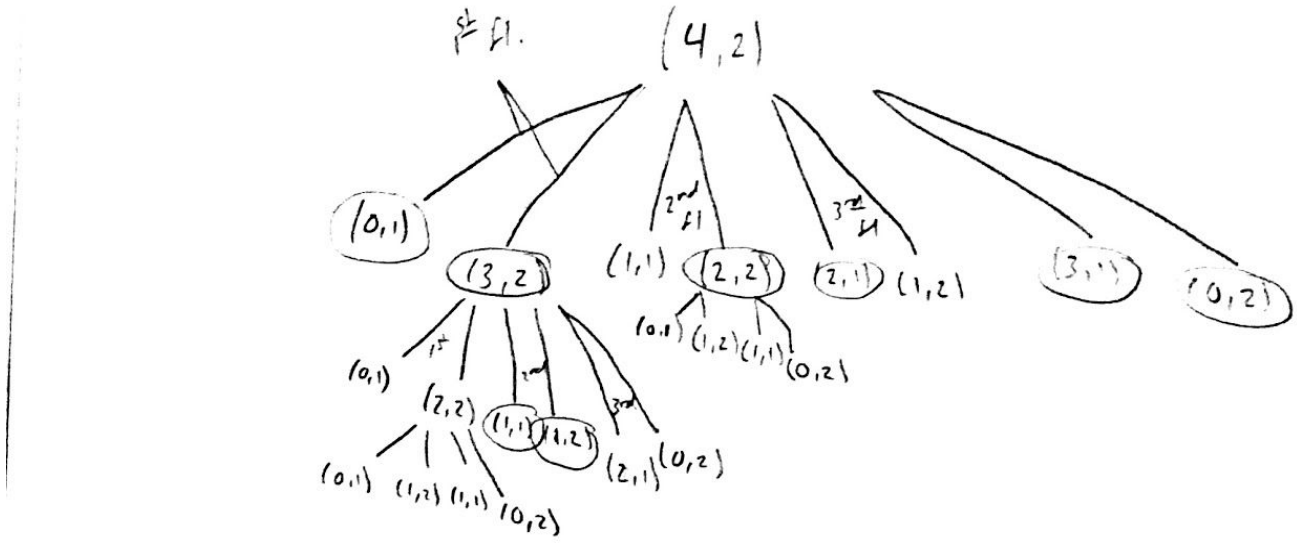
Nyran Bonilla

Glass Falling

1b) In order to find the minimum amount of trials we must drop sheets from each floor to find optimal solution. If there is only one sheet to test with then the worst case scenario is that we test each floor once. So we return the number of floors. If there are only one or zero floors, then we must return the floor number as the minimum number of trials. If we were to drop the sheet from floor n, then there can only be two possibilities. Either the sheet breaks or it survives the fall. If the sheet breaks then we have to check each floor below it. If the sheet doesn't break , then we must check the floors above the current one. This leaves us with problems of (n-1) floors and sheets-1 if the sheet breaks and if it doesn't break the subproblem becomes (n-current floor) and the same number of sheets. We will take the maximum value of each of the two subproblems per floor and use that as the minimum number of trials up to that floor.

If sheets=1, return floor
If floor=0 or floor=1, return floor
If sheet breaks, find [floor-1][sheet-1]
If sheet does not break, find [floor-i][sheet]

1b)



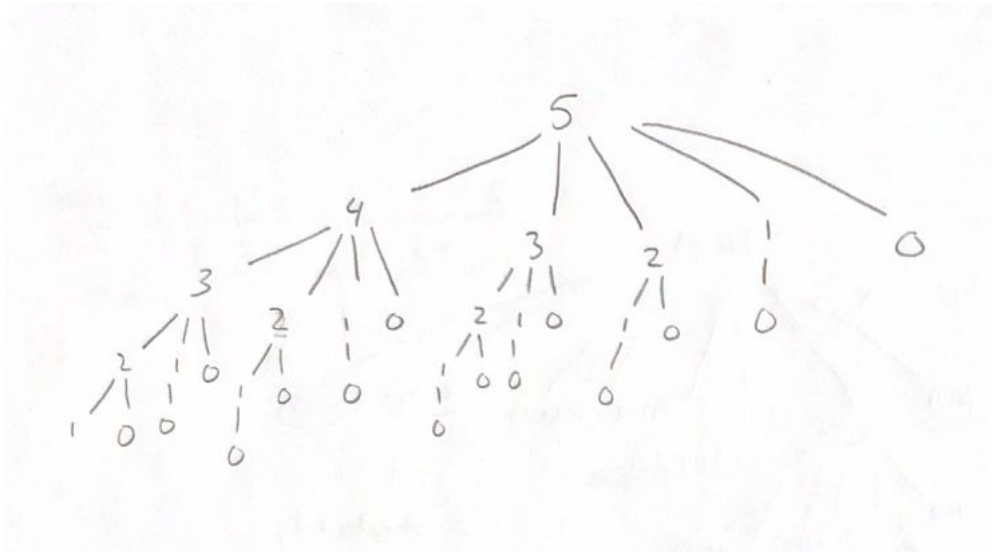
Nyran Bonilla

1d) 8 distinct subproblems :
(0,1), (3,2), (1,1), (1,2), (2,2), (2,1), (3,1), (0,2)

1e) n*m distinct subproblems
1f) In order to memoize GlassFalling we will initialize an array to be used to store the values of each subproblem. Then we will call a helper function that will help us find the minimum amount of trials recursively. First we will check the value at memo[floor][sheet] and see if that contains value we are looking for. If not then we will continue the algorithm like in the non-memoized version to calculate the minimum trial. However , if the answer is already within the memo array then we will return that value. This helps eliminate the repetitions that is found in the first GlassFalling algorithm.

Rod Cutting

1a)



1b)

Length (i)	1	2	3	4	5
Price (pi)	2	16	12	20	22
Price Density (pi/i)	2	4	4	5	4.4

In greedy algorithms, you look to use the highest density first. So in this example of length 5, greedy will choose length 4 first since it has the highest density then with the remaining length it has to choose 1. That gives the final price to be 22. But with dynamic programming, it will

choose length 2 twice then length 1. That would result in the price to be 34, which would be the optimal solution. This shows that greedy will not always give the optimal solution.