Dennis Chang

Assignment 1

Glass Falling

A. Describe the optimal substructure/recurrence that would lead to a recursive solution.

Given *n* floors and *m* sheets, we will need to test all the floors as possible initial dropping floors; we'll call the initial floor *i*,

If we have 1 floor left, we will have to test the floor.
If we have 0 floors left, we do not need to test anything.
If we have 1 glass sheet left, we need to test each floor from bottom up, or i numbers of times.

Two outcomes can occur from a drop: the glass shatters or it does not shatter.
If the glass shatters, we run the recurrence with one less sheet of glass and with the number of possible floors being i-1. We know the critical floor is below i since the glass shattered at i.

If the glass does not shatter, we run the recurrence with n-i floors since we know the critical floor is between i and n.

B. Draw recurrence tree for given (floor=4, sheets=2)

| | | | | | | |
|---|---|---|---|---|---|---|
| DP(4,2) | 1 | DP(0,1) | | | | |
| | | DP(3,2) | 1 | DP(0,1) | | |
| | | | | DP(2,2) | 1 | DP(0,1) |
| | | | | | | DP(1,2) |
| | | | | | 2 | DP(1,1) |
| | | | | | | DP(0,2) |
| | | | 2 | DP(1,1) | | |
| | | | | DP(1,2) | | |
| | | | 3 | DP(2,1) | | |
| | | | | DP(0,2) | | |
| | 2 | DP(1,1) | | | | |
| | | DP(2,2) | 1 | DP(0,1) | | |
| | | | | DP(1,2) | | |
| | | | 2 | DP(1,1) | | |
| | | | | DP(0,2) | | |
| | 3 | DP(2,1) | | | | |
| | | DP(1,2) | | | | |
| | 4 | DP(3,1) | | | | |
| | | DP(0,2) | | | | |

D. How many distinct sub-problems do you end up with given 4 floors and 2 sheets?

There are 8 distinct sub-problems:

(0,1) , (0,2) , (1,1) , (1,2) , (2,1) , (2,2) , (3,1) , (3,2)

E. How many distinct sub-problems do you end up with given n floors and m sheets?

(n * m) distinct sub-problems

F. Describe how you would memorize GlassFallingRecur

A table of size [floors+1][sheets] should be created.
table[0][i] = 0;
table[1][i] = 1;
table[i][1] = i

The code would follow the recursive method except before calling GlassFallingRecur( floors, sheets), table[floor][sheets] should be checked. If a value exists, we just retrieve the value from the table. If the value does not exist, we will make a recursive call. The table will be completed thus fulfilling the memorization.

Rod Cutting

A. Draw a recursion tree for a rod length of 5.

```
5   0
    1   0
    2   0
        1   0
    3   0
        1   0
        2   0
            1   0
    4   0
        1   0
        2   0
            1   0
        3   0
            1   0
            2   0
                1   0
```

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.

| Length | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Price | 1 | 4 | 30 | 36 | 60 |
| Density | 1 | 2 | 10 | 9 | 12 |

Whenever given a rod of length 4, the greedy algorithm would select to cut a portion of size 3 because it has the highest density of available options. This would yield two pieces of size 3 and 1 with corresponding values of 30 and 1; sum value of 31. The greedy algorithm fails to see that a rod of length 4 already has a greater value of 36 compared to the greedy algorithm solution.