Question1:

Falling Glass

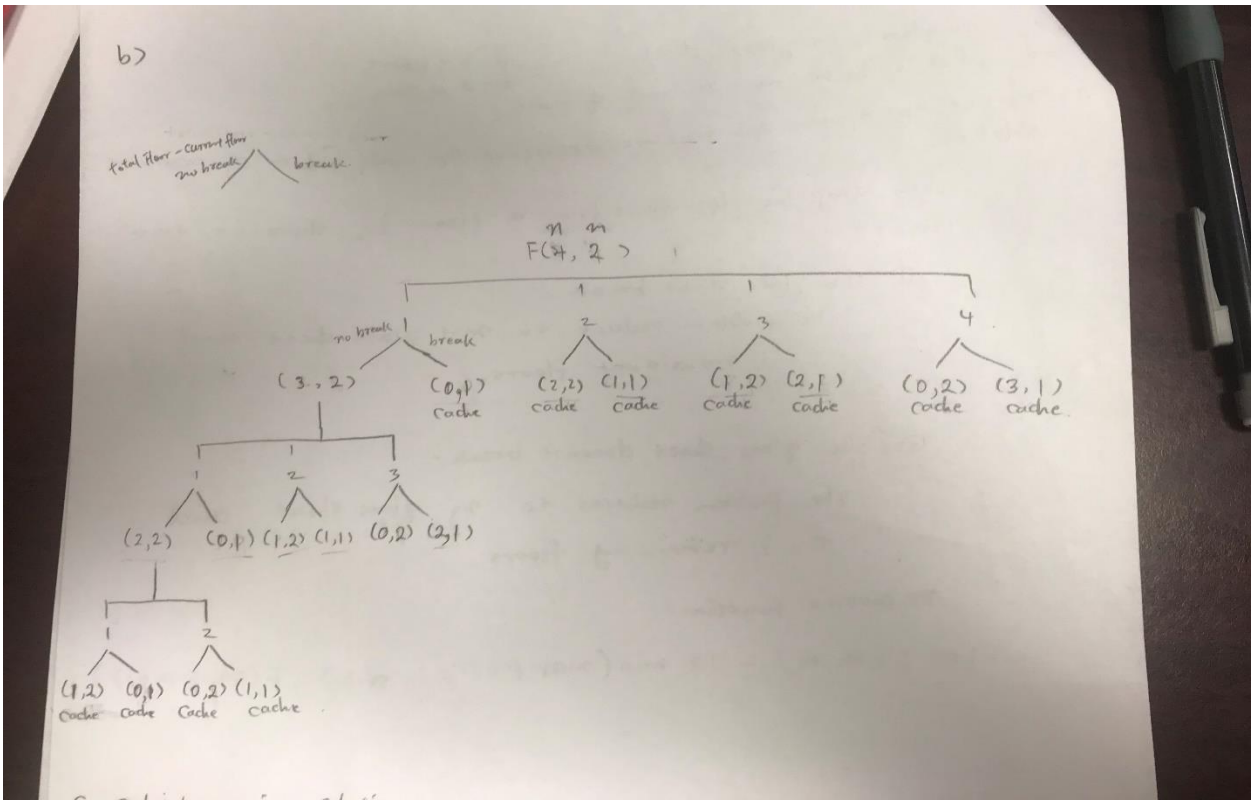
(a) Describe the optimal substructure/recurrence that would lead to a recursive solution

m is number of glass sheets
n is the number of floors.

When drop the glass sheet from a floor i, there is two cases:
Case1: The glass sheet break:
The problem reduces to m -1 glass sheet and i -1 remaining floors.

Case2: The glass sheet doesn't break:
Then problem reduces to m glass sheet and n – i remaining floors.

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



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

github

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

Answer: 4*2 = 8 distinct subproblems

(e) How many distinct subproblems for n floors and m sheets?

Answer: n*m distinct subproblems

(f) Describe how you would memoize GlassFallingRecur

Answer: step 1 create a cache to store temporary results.

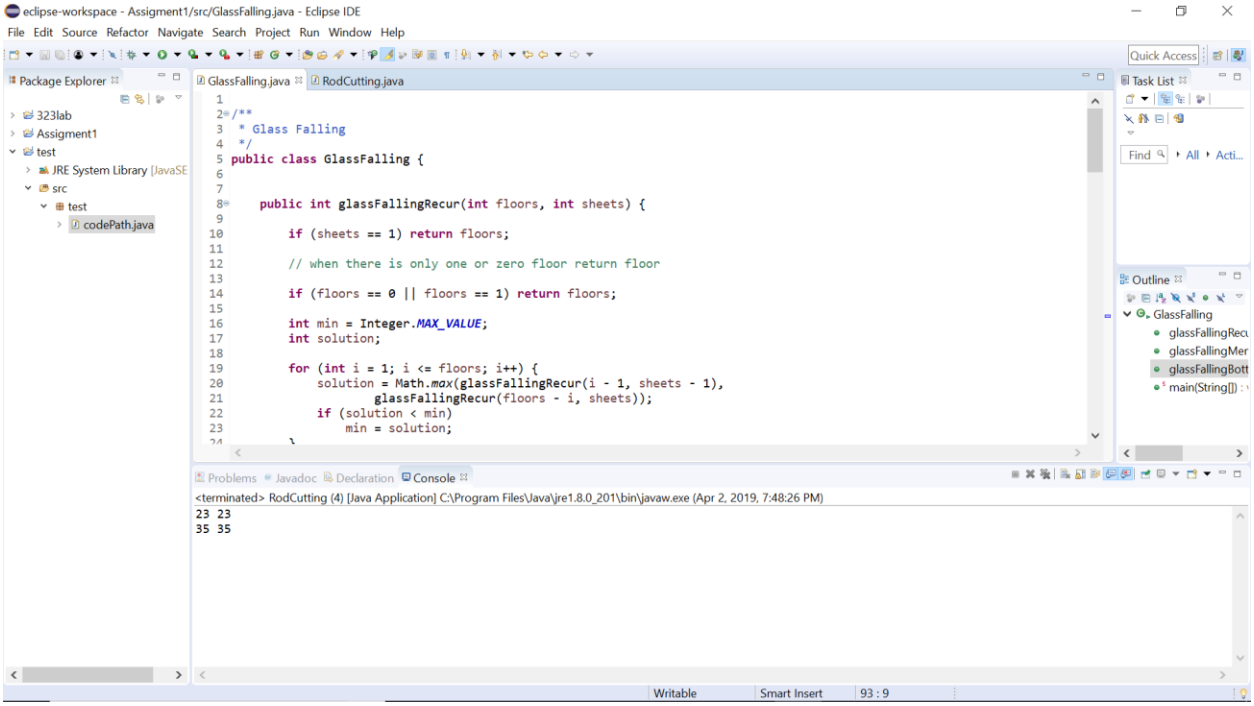
Step 2 Before performing a calculation, check whether the calculation has already been done, if its already done, use the store result.

Step 3 if the value is first time calculate, store the results for later use.

(g) Code a bottom-up solution GlassFallingBottomUp(int n numFloors, int m numGlass)

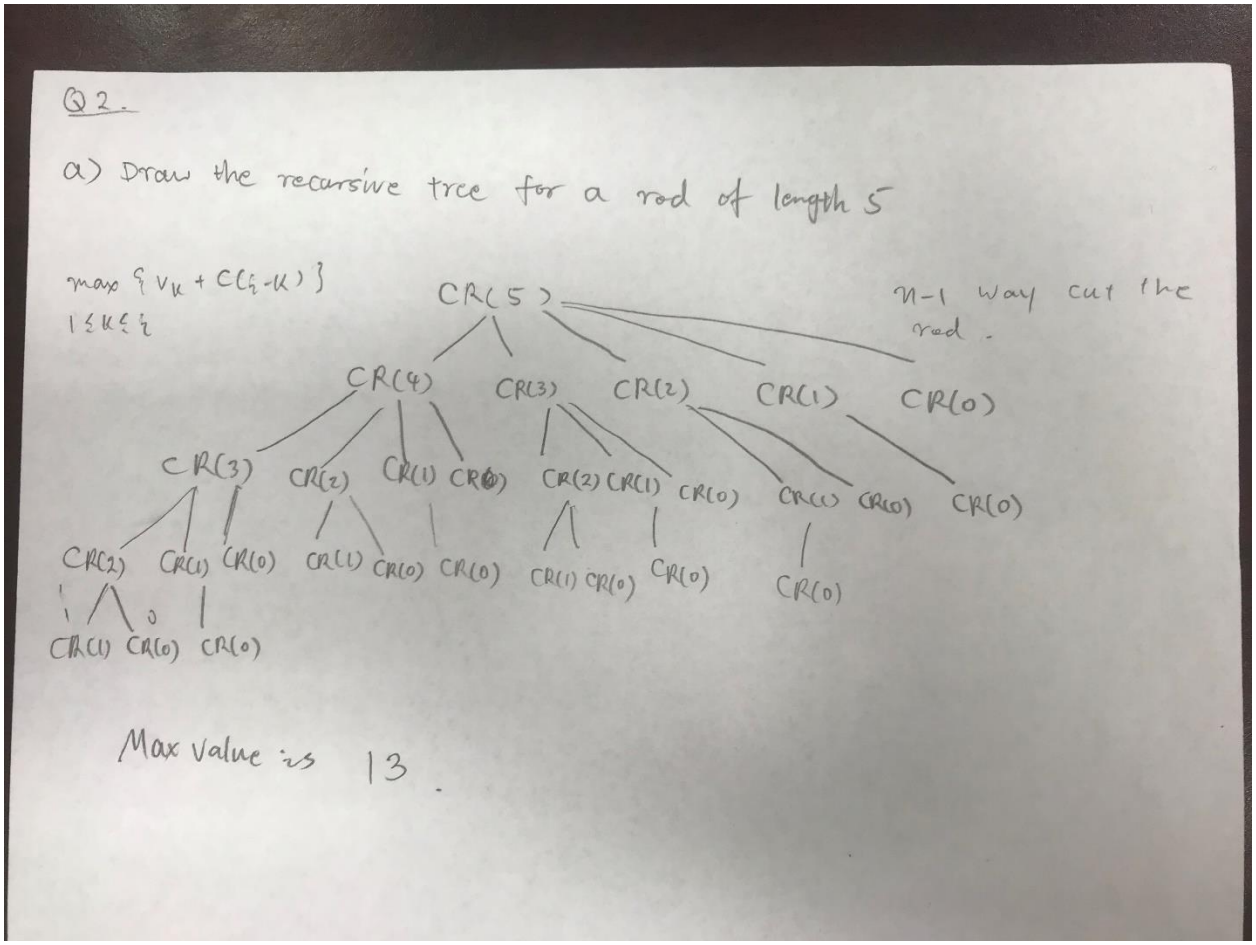
github

GlassFalling Output:


eclipse-workspace - Assigment1/src/GlassFalling.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
323lab
Assigment1
test
JRE System Library [JavaSE
src
test
codePath.java
GlassFalling.java
RodCutting.java
1
2 /**
3 * Glass Falling
4 */
5 public class GlassFalling {
6
7
8 public int glassFallingRecur(int floors, int sheets) {
9
10 if (sheets == 1) return floors;
11
12 // when there is only one or zero floor return floor
13
14 if (floors == 0 || floors == 1) return floors;
15
16 int min = Integer.MAX_VALUE;
17 int solution;
18
19 for (int i = 1; i <= floors; i++) {
20 solution = Math.max(glassFallingRecur(i - 1, sheets - 1),
21 glassFallingRecur(floors - i, sheets));
22 if (solution < min)
23 min = solution;
Task List
Find
All
Acti...
Outline
GlassFalling
glassFallingRecu
glassFallingMer
glassFallingBott
main(String[]) :
Problems
Javadoc
Declaration
Console
<terminated> RodCutting (4) [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (Apr 2, 2019, 7:48:26 PM)
23 23
35 35
Writable
Smart Insert
93 : 9

Q2. 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.

Answer: let p1 = 1, p2 = 5, p3 = 8, p4=9 and n =4
Density: 1/1 =1, 5/2 = 2.5, 8/3 = 2.667, 9/4=2.25.
Max density is 8/3 so the greedy first cut off a piece of length of 3, the remaining will be 1
8 + 1 = 9 total profit. But if cut the rob with length of 2 will give 5 + 5 =10 which is higher than 9.

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

github

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

Github

Rod cutting output: