Why dynamic programming:

- Dynamic programming applies to optimization problems that have overlapping subproblems.
- Dynamic programming avoid the bad running time of brute-force ("naïvely recursive") solutions by recording previously computed results in a table (*memoization*)

The anatomy of the dynamic programming approach from the programmer's perspective (compare CLRS pg 359):

- *Characterize the substructure*: Determine what the subproblems are and how they relate to the larger problem. (Determine the meaning of the tables.)
- Recursively define the problem.
- Devise an algorithm to populate the tables of subproblem solutions. (Find *how good* the best way is.)
- Devise an algorithms to reconstruct a solution from the tables. (Find *the best way*.)

A lumberjack has an k-yard long log of wood he wants cut at n specific places j_1, j_2, $\ldots j_n$, represented as the distance of that cut point from one end of the log. (We can also consider the ends as trivial "cut points" $j_0 = 0$ and $j_{n+1} = k$.) The sawmill charges \$$x$ to cut a log that is x yards long (regardless of where that cut is). The sawmill also allows the customer to specify the ordering and location of the cuts.

For example, if $k = 20$ and we want cuts at 3 yards, 6 yards, and 10 yards from the left end, then if we cut them from left to right the cost would be

$$20 + (20 - 3) + (20 - 6) = 20 + 17 + 14 = 51$$

But making the same cuts from right to left would cost

$$20 + 10 + 6 = 36$$

Devise and implement an algorithm to minimize the cost, and analyze its running time.