HW 11

- give the steps of the dynamic programming process from class, not just a statement of the algorithm or a narrative account of your reasoning process
  - establish the problem
    - specifications
    - examples
  - identify avenues of attack
    - targets
    - paradigms and patterns
    - the series of choices
  - define the algorithm
    - size
    - generalize / define subproblems: partial solution, alternatives, subproblem, memoization
    - base case(s)
    - main case
    - top level: initial subproblem, setup, wrapup
    - special cases
    - algorithm
  - show termination and correctness
    - termination
    - correctness: establish the base case, show the main case, final answer
  - determine efficiency
    - implementation
    - time and space
    - room for improvement
- only #2 was graded (or a different problem if you didn't hand in #2)
  - review those comments and consider their application to the other problems

HW 11

- #2 – the subproblem input in this case depends on the series of choices
  - process input – line break now or not?
    - the subproblem also requires the length of current line or the space left on current line – necessary for computing the cost (and for knowing if not breaking now is an option)
    - base case cost >= 0 because it is necessary to consider the cost of spaces on the current line if there are words on it
  - produce output – where is the next line break?
    - the subproblem only needs the remaining words because this decision is about the entire contents of the line – the cost of the space left can be computed from the length of the words between the current point and the next line break
    - base case cost is 0 because there are no words left to go on a line (the current line is empty)

HW 11

- avoid unnecessary work – even though the result may still be polynomial, it could be a better polynomial…
  - e.g. a `computeLineLength(i,j)` function to compute the cost of the remaining spaces for a line containing words i through j is a nice idea for organization, but consider its context

```
for j = i to n-1 do
  length ← computeLineLength(i,j)
  …
```

    - this adds up the lengths of words *i..i*, then *i..i*+1, *i..i*+2, *i..i*+3, … – a lot of repeated computation
    - instead incrementally add the length of word *j* to the running total in each iteration