1. Announcements:
   (a) Labs due on Tuesday; the cycle begins!
   (b) Questions?

2. Complexity (left over from Wednesday!):
   (a) Formal definition of what it means for $f(x)$ to be $O(g(x))$. A definition worth remembering. Write it here:
      i. $f(x)$ is bounded above by some constant times $g(x)$.
      ii. $f(x)$ need not ever be equal to $c \cdot g(x)$. The bound need not be tight.
      iii. It only needs to be bounded above to the right of some $x_0$.
      iv. Really, we’re only concerned about the magnitude of $f(x)$. In practice, $f(x) \geq 0$ since $f$ is often a measure of time or space utilization.
   (b) Typical approach to “simplifying” a complex function to its big-O equivalent:
      i. If a function is a sum of terms, keep the term that grows the fastest. For example, $\frac{1}{2}n^2 - \frac{n}{2}$, you keep just the $\frac{1}{2}n^2$ term; in the long run, the polynomial’s trend is governed by this term.
      ii. Thus, the function has a single term. If this term has a coefficient that is not 1, cross out the coefficient. Thus, $\frac{1}{2}n^2$ is $O(n^2)$—a quadratic—and $f = 1000$ is written $O(1)$—a constant function.
      iii. Logarithms of all bases are related by a constant. Thus, it does not matter if $f(n) = \log_2 n$ or $f(n) = \ln n$: they’re both written $O(\log n)$.
   (c) E.g.: Vectors that extend to size $n$ by 1 requires $\frac{n(n-1)}{2}$ copies of old data to new, or $O(n^2)$ time. *When you double the Vector’s length, the time to expand increases by 4.*
   (d) E.g.: Vectors that extend to size $n$ by doubling takes copies old data new new in $2^{\log_2 n} - 1 = n - 1$ steps, or $O(n)$ time. *When you double the Vector’s length, the time to expand increases by 2.*

3. Recursion: big ideas, little code.
   (a) Basic idea: Reduce hard problem to simpler problem plus a little work.
      i. Base case. The simplest problem(s) you can solve. Think zero.
      ii. Progress. If not a base case, a little bit of work necessary to reduce problem to a simpler problem.
      iii. Recursion. A call to (perhaps another) method that solves the subproblem.
   (b) Counting down to zero from $n$:
      i. Base case: if $n == 0$: count 0, we’re done.
      ii. Progress: Count $n$. Now only $n-1$ to zero is left.
      iii. Recursion: Count down from $n-1$.
   (c) Reversing a string: `reverse("bard")` is "drab" and `reverse("grub")` is "burg". (This, of course, is helpful for finding palindromes, but just as helpful in finding even more exotic semordnilaps.)
   (d) The hello, world of recursion: Towers of Hanoi.
   (e) Print all substrings of a string (!), `printSubstrings(s)`.

4. Challenge: Can you write down the $n$ distinct values $x_i \in \{1...n\}$ in an order such that that makes $x_i + x_{i+1}$ a perfect square for $0 \leq i < n$?

---

**YOUR PARTY ENTRIES THE TAVERN.**

I gather everyone around a table. I have the elves start whittling dice and get out some parchment for character sheets.

HeY, no reCURsING.

**XKCD NUMBER 244**