F20 – 2nd half of day 2 (needed b/c 1st day took too long, spent extra time discussing racism).

* Do best/average/worst case. (prev notes)
* Talk about coefficients.

**Hidden coefficients.**

* Big-oh comes with a caveat, sometimes a large caveat. Recall that we had two rules for big oh—always take the term that grows the fastest, and drop coefficients. Sometimes this can lead to intentional consequences.
* Take for instance, T1(n) = n^2 + 2. A reasonable T(n). What is the big-oh? O(n^2).  
  And now, T2(n) = 1,000,000n + 1,000,000. big oh? O(n). So we have a quadratic algorithm in T1, and a linear-time algorithm in T2. By our rule of thumb, we should prefer T2 over T1.
* But for all n less than one million, T1(n) < T2(n). so in the real world, we’d probably prefer T1.
* Now these numbers are made up, but here are some that are not.
* Let’s go back to the matrix multiplication problem.
  + Explain how the Naïve matrix mult algorithm is O(n^3).
    - New matrix is n by n. go through each square (n^2 operations).
    - For each square, you need to mult & add an entire row and column.
  + So one of the big breakthroughs in matrix multiplication was called Strassen’s algorithm (1969), which works in O(n^2.81). Only faster for matrices greater than 100 by 100.
  + Coopersmith-Winograd alg (1990) O(n^2.376). Coefficients are so big only worthwhile for matrices that are so big they can’t be stored on modern computers anyway.

Analyze big-oh of RArrayList operations.

* Access (get/set one item) – O(1)
* Expand – O(n)
* Append – for us, worst & avg O(n). Can get average down to O(1) with a geometric progression where we double the array in capacity every time.
* Prepend – always need a shift of the whole array, with is O(n). [worst & average]
* Table of Vector operations:
* Worst case Best? Average
* Get O(1) 1
* Set O(1) 1
* Append O(n) 1 n
* Prepend O(n) n
* [linear] Search O(n) 1 n/2 = O(n)
* size O(1) 1
* insert an element
* front n 1
* back n
* middle n
* remove an element
* front n
* back 1
* We do this for all ADTs.

Analyzing Recursive Functions.