Big oh day 2

[From last time: one class got through all slides, the other class still needs to do last few slides]

Review:

* Big-oh is how we measure the running time of an algorithm.
* Most algorithms, as we increase their input size (the amount of data the algorithm is processing), will get slower and slower. In other words, the time the algorithm takes to run will grow, as the input size grows. This is pretty much inevitable.
* We can plot the input size of an algorithm and the time it takes on a graph. The time it takes is often the number of basic operations, T(n), which is an approximation of time, but works because basic operations don't depend on the particular computer or programming language.
* The best algorithms have graphs that grow slowly. That is, as we increase the amount of data the algorithm is processing, the time the algorithm takes grows, but grows slowly.
* In general, "bad" algorithms tend to grow quickly, which means at some point, the algorithm will be too slow to be practical, perhaps even at small input sizes.
* Big-oh is a way to categorize these graphs. The categories are based on the "order of growth" or "rate of growth."
* The categories are very broad, and only consider, for polynomial graphs, the largest exponent. So all linear graphs are grouped together, all quadratic graphs, all cubic graphs, etc.
* Other categories: constant O(1), logarithmic O(log n), exponential (2^n). There are others as well, like O(n log n).

Begin today with Fibonacci.

* Run the slow Fibonacci code.
* Explain with pictures why this is exponential time.
* Run the fast fib code.
* Explain why this is linear time.

Go on to Arraylist timings.

* Run code for add and remove.
* Explain why this is linear.
* Switch to a linked list – goes to constant time. Whoa!
* Run code for arraylist access.