Last Time

• Linked Lists
  – Singly-linked, doubly-linked, and circular
  – Adding items, removing items, and retrieving values
  – Using a Node class within a SinglyLinkedList class
  – Adding support for generics
Back to Iterative/Recursive Fibonacci

• We saw that the Iterative Fibonacci code ran significantly faster than the Recursive Fibonacci.

• How much faster? Can we quantify this performance difference?
Timing Algorithms

```java
long startTime = System.nanoTime();
/* run the algorithm */
long endTime = System.nanoTime();
long timeElapsed = endTime - startTime;
```

• Let’s try it on the Fibonacci example.
Showing Results

• **Tabular form** – Raw data presented, exact values, tough to show relationships without explicit formulas

• **Chart form** – Data graphed, approximate values, easy to show relationships based on slopes and curves
Challenges

• Experimental running times are difficult to compare unless on the same hardware and software. (why?)
• Experiments only cover the inputs tested, might leave out an important outlier condition. (why?)
• Need the full algorithm implementation – can’t guarantee good predictions with a partial algorithm. (why?)
Solving These Challenges

• We want to be able to:
  – Compare the efficiency of two algorithms.
  – Consider all possible inputs.
  – Evaluate at the algorithm-level rather than at the implementation level.

• Solution: Count primitive operations!
Primitive Operations

• Assigning a value to a variable.
• Following an object reference.
• Performing an arithmetic operation.
• Comparing two numbers.
• Accessing an array index.
• Calling a method / returning from a method.

• This operation count will correlate the number of instructions to the actual running time in a computer.
Any Questions?