CSSE 220 Day 25

Sorting Algorithms
Algorithm Analysis and Big-O
Function Objects and the Comparator Interface

Checkout *SortingAndSearching* project from SVN

Questions

- Exam results

Remember Selection Sort?

- Let’s see…

Why study sorting?

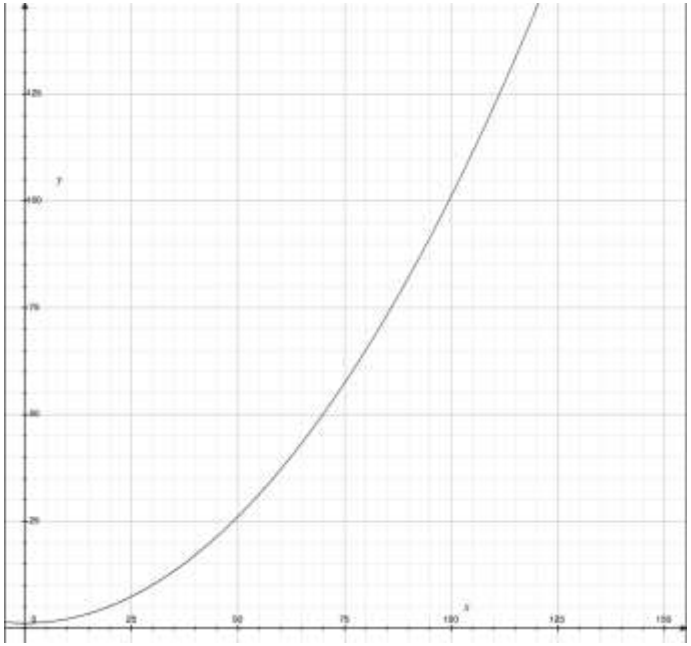
- Remember Shlemiel the Painter

Course Goals for Sorting: You should…

- Be able to describe basic sorting algorithms:
  - Selection sort
  - Insertion sort
  - Merge sort
  - Quicksort
- Know the run-time efficiency of each
- Know the best and worst case inputs for each

Profiling Selection Sort

- Profiling: collecting data on the run-time behavior of an algorithm
- How long does selection sort take on:
  - 10,000 elements?
  - 20,000 elements?
  - …
  - 80,000 elements?



- $O(n^2)$

Q1-3

Big-Oh Notation

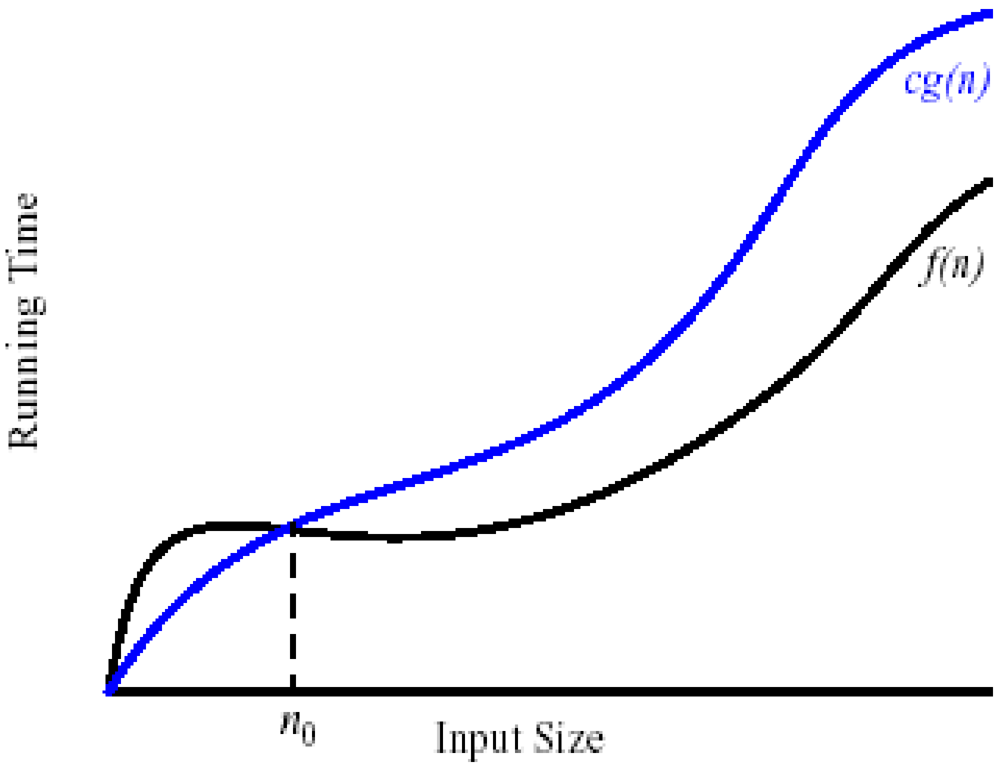
- In analysis of algorithms we care about differences between algorithms on very large inputs

- We say, “selection sort takes on the order of n^2 steps”

- Big-Oh gives a formal definition for “on the order of”

Formally

- We write $f(n) = O(g(n))$, and say "f is big-Oh of g"
- if there exists positive constants c and n_0 such that
- $0 \leq f(n) \leq c\, g(n)$ for all $n > n_0$
- g is a ceiling on f



Rule of Thumb

- Suppose the number of operations is given by a polynomial:

$$a_k * n^k + a_{k-1} * n^{k-1} + \ldots + a_2 * n^2 + a_1 * n + a_0$$

- Then the algorithm is $O(n^k)$.
- That is, **take the highest order term and drop the coefficient**

Insertion Sort

- Basic idea:
  - Think of the list as having a sorted part (at the beginning) and an unsorted part (the rest)
  - Get the first number in the unsorted part
  - Insert it into the correct location in the sorted part, moving larger values up to make room

Repeat until unsorted part is empty

Insertion Sort Exercise, Q4-11b

- Profile insertion sort
- Analyze insertion sort assuming the inner while loop runs that maximum number of times (count the array accesses)
- What input causes the worst case behavior? The best case?
- Does the input affect selection sort?

Ask for help if you're stuck!

Searching

- For searching unsorted data, what's the worst case number of comparisons we would have to make?

Binary Search of Sorted Data

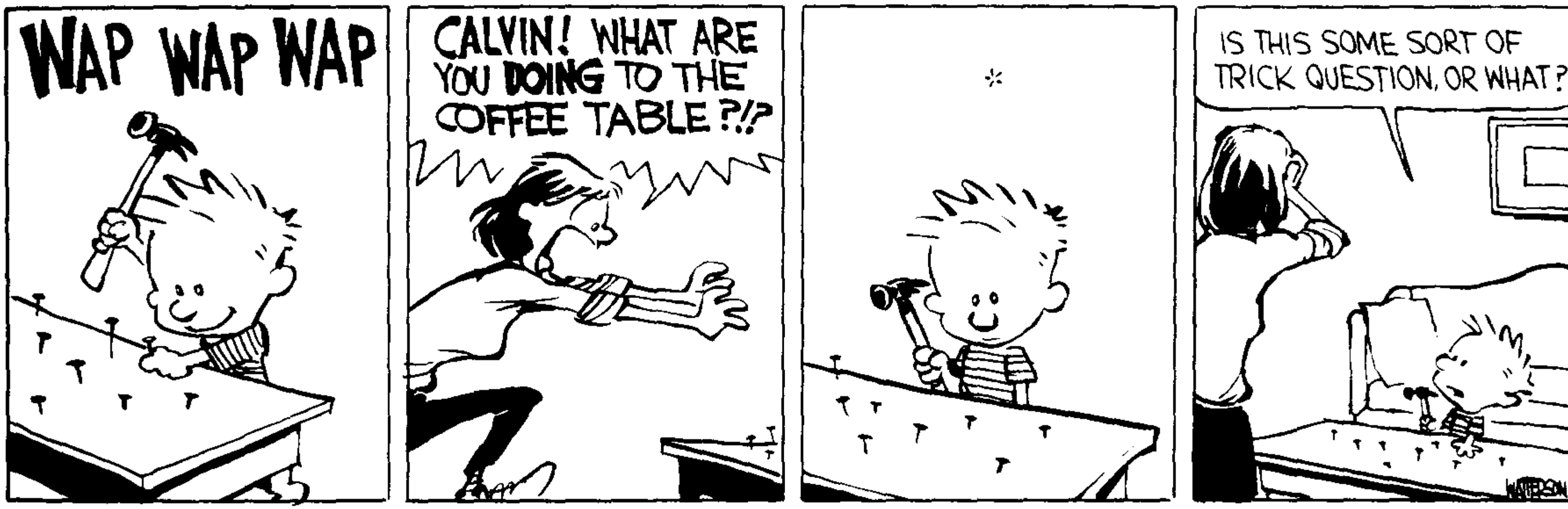
- A divide and conquer strategy

- Basic idea:
  - Divide the list in half
  - Should result be in first or second half?
  - Recursively search that half

Analyzing Binary Search

- What's the best case?
- What's the worst case?

Q12



Perhaps it's time for a break.

Merge Sort

- Basic recursive idea:
  - If list is length 0 or 1, then it's already sorted
  - Otherwise:
    - Divide list into two halves
    - Recursively sort the two halves
    - **Merge** the sorted halves back together
- Let's profile it…

Analyzing Merge Sort

- More trees

Q11c, 13

Quicksort

- Basic recursive idea:
  - If length is 0 or 1, then it's already sorted
  - Otherwise:
    - Pick a "pivot"
    - Shuffle the items around so all those less than the pivot are to its left and greater are to its right
    - Recursively sort the two "partitions"

- Let's profile it…

Analyzing Quicksort

- This one is trickier
- How should we choose the "pivot"

Q11d

Function Objects

- Another way of creating reusable code

A Sort of a Different Order

- Java libraries provide efficient sorting algorithms
  - `Arrays.sort(…)` and `Collections.sort(…)`

- But suppose we want to sort by something other than the "natural order" given by `compareTo()`

- Function Objects to the rescue!

Function Objects

- Objects defined to just "wrap up" functions so we can pass them to other (library) code

- We've been using these for awhile now
  - Can you think where?

- For sorting we can create a function object that implements Comparator

Data Structures

- Understanding the engineering trade-offs when storing data

Data Structures

- Efficient ways to store data based on how we'll use it

- So far we've seen `ArrayLists`
  - Fast addition **to end of list**
  - Fast access to any existing position
  - Slow inserts to and deletes from middle of list

Q14

Another List Data Structure

- What if we have to add/remove data from a list frequently?
- `LinkedLists` support this:
  - Fast insertion and removal of elements
    - Once we know where they go
  - Slow access to arbitrary elements

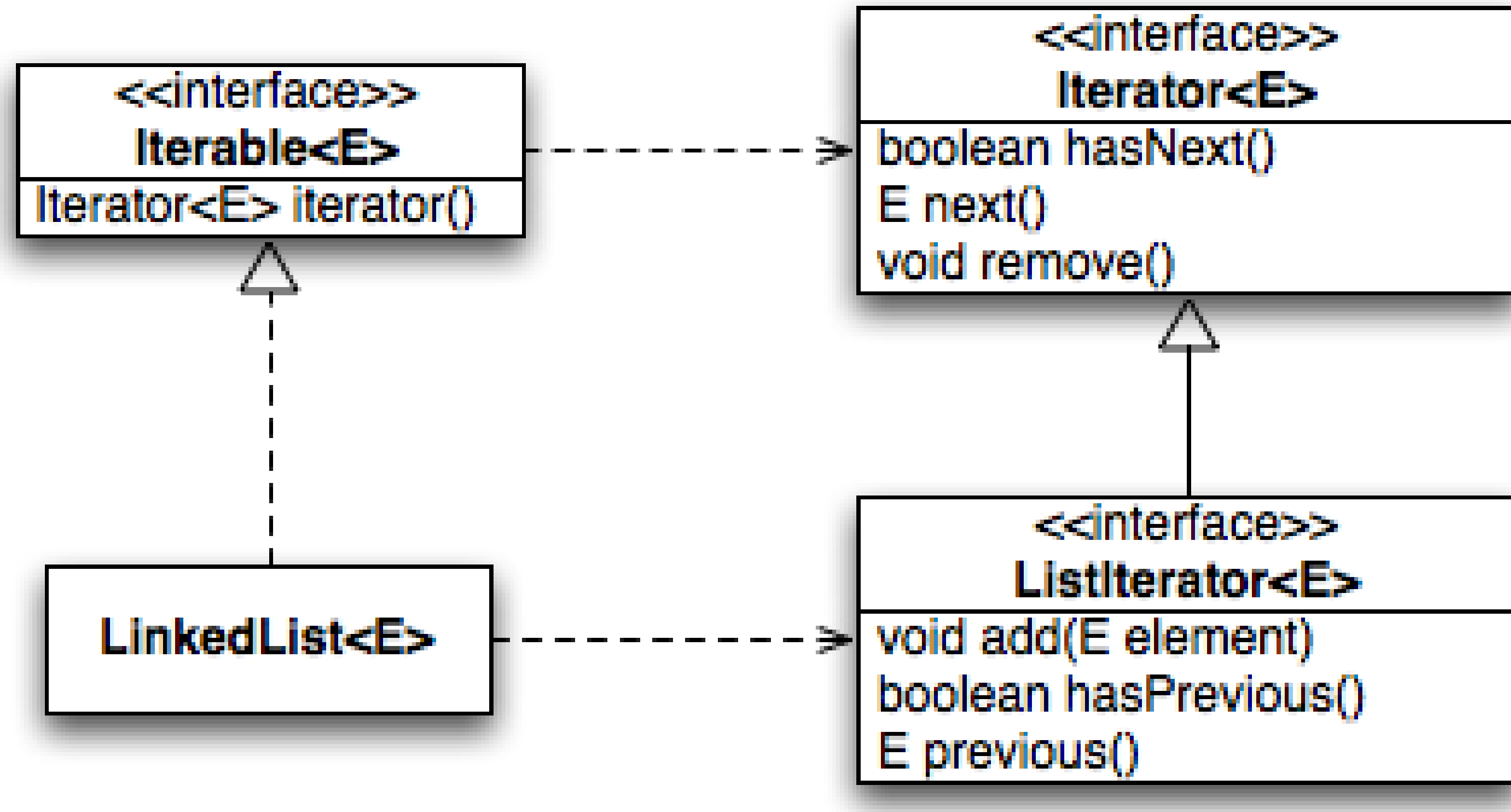
Q15,16

LinkedList<E> Methods

- `void addFirst(E element)`
- `void addLast(E element)`
- `E getFirst()`
- `E getLast()`
- `E removeFirst()`
- `E removeLast()`

- What about the middle of the list?
  - `LinkedList<E> implements Iterable<E>`

Accessing the Middle of a LinkedList



An Insider's View

```java
for (String s : list) {
  // do something
}
```

```java
Iterator<String> iter =
    list.iterator();

while (iter.hasNext()) {
  String s = iter.next();
  // do something
}
```

Enhanced For Loop

What Compiler Generates

Next Time

- Implementing `ArrayList` and `LinkedList`
- A tour of some data structures
- VectorGraphics work time