CSSE 220 Day 15

Searching
Function Objects and the Comparator Interface
Merge Sort
Fork/Join Framework

Checkout *SortingAndSearching* and *ForkJoinIntro* projects from SVN

Questions?

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 value 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, Q1–10

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

Ask for help if you're stuck!

Q1–Q10

Searching

- Consider:
  - Find Royal Mandarin Express's number in the phone book
  - Find who has the number 208-0521
- Is one task harder than the other? Why?
- 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
  - Decide whether result should be in upper or lower half
  - Recursively search that half

Analyzing Binary Search

- What's the best case?
- What's the worst case?
- We use recurrence relations to analyze recursive algorithms:
  - Let T(*n*) count the number of comparisons to search an array of size *n*
  - Examine code to find recursive formula of T(*n*)
  - Solve for *n*

Q11

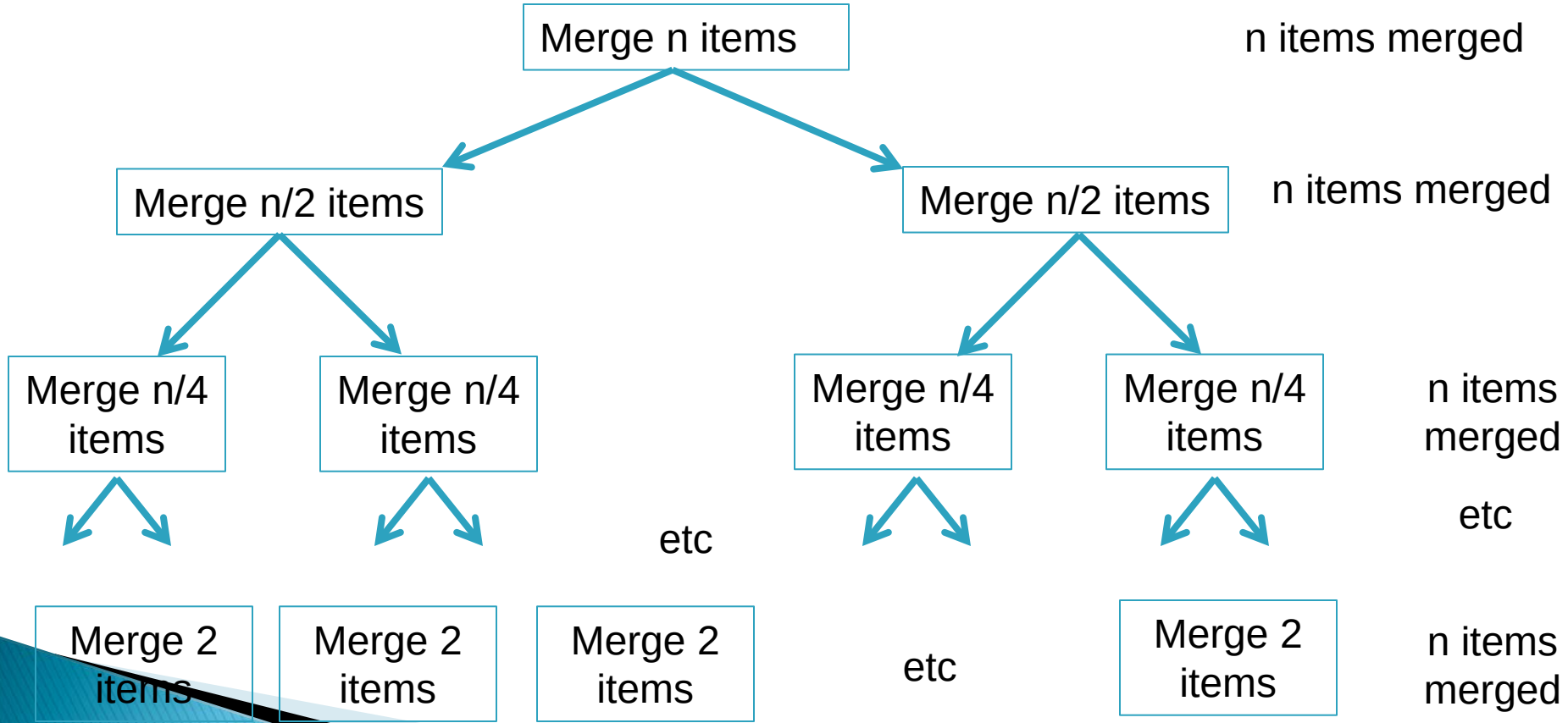
Merge Sort

- 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

Analyzing Merge Sort

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



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
- For sorting we can create a function object that implements `Comparator`
- `Let's try it!`

Intro. to Fork-Join Parallelism

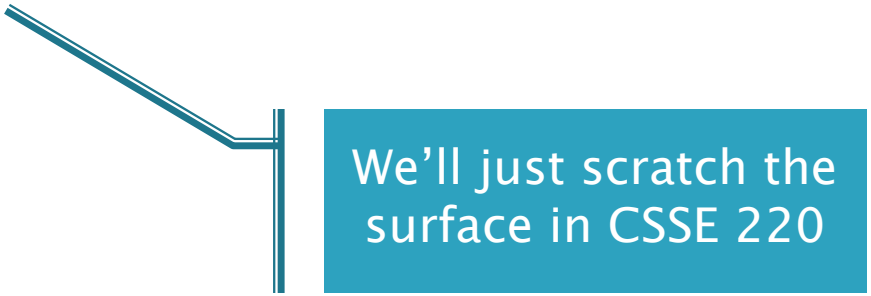
- Function objects and recursion meet multicore computers

Some slides and examples derived from Dan Grossman's materials at http://www.cs.washington.edu/homes/djg/teachingMaterials/

Changing a Major Assumption

- *Sequential programming*: one thing happens at a time
  - No longer the case!
- *Parallel programming*: multiple things happen simultaneously

- Major challenges and opportunities
  - Programming
  - Algorithms
  - Data



Simplified View of History

- Parallel code is often much harder to write than sequential
- Free ride from the CPEs
  - From 1980–2005 performance of same sequential code doubled every two years
- No one knows how to continue this!
  - Speed up clock rate?
    - Two much heat
    - Memory can't keep up
  - But the "wires" keep getting smaller, so…
    - Put multiple processors on same chip!

What do we do with all of them?

- Run multiple totally different programs
  - Operating system handles this
  - Uses *time-slicing* plus multiple cores
- Multiple things at once in one program
  - We'll play with this today!

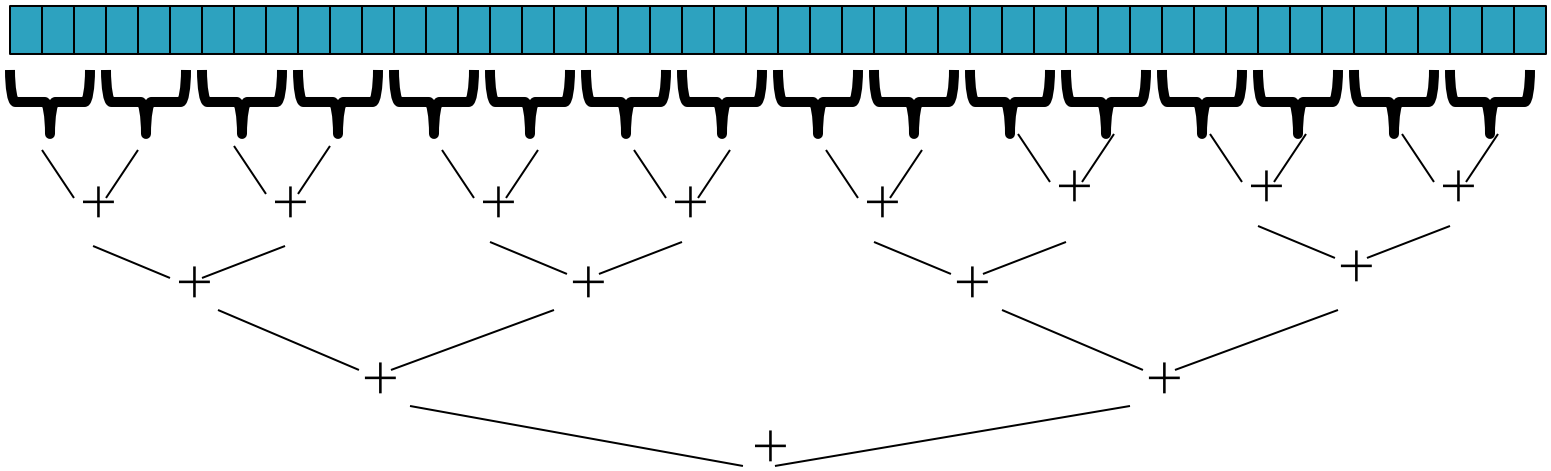
Parallelism vs. Concurrency

- *Parallelism*: Use more resources for a faster answer
- *Concurrency*: Correctly and efficiently allow simultaneous access to data

An analogy

- CS1 idea: Writing a program is like writing a recipe for a cook
- Parallelism: slicing lots of potatoes
- Concurrency: sharing stove burners

Parallelism Idea



- Example: Sum elements of a large array
- Use divide-and-conquer!
  - Parallelism for the recursive calls

Fork-Join Framework

- Specifically for recursive, divide-and-conquer parallelism
  - Is in Java 7 standard libraries, but available in Java 6 as a downloaded `.jar` file
- *Fork*: splitting off some code that can run in parallel with the original code
  - Like handing a potato to a helper
- *Join*: waiting for some forked code to finish
  - Like waiting for the potato slices from the helper

Getting good results in practice

- Set a *sequential threshold*
  - A size below which we just "slice 'em ourselves"
- Library needs to "warm up"
  - Java Virtual Machine optimizes as it runs
- Wait until your computer has more processors ☺

- Here there be dragons!
  - Memory-hierarchy issues
  - Race conditions
  - We're ignoring lots of gory details!

Fork-Join Lab

- Find a partner for HW15b (today's homework)
- You'll:
  - Write some code
  - Run some experiments
  - Write a lab report

Follow the written homework instructions carefully. There's much more independent learning here than we've been doing so far.

Work Time

- Review Homework.

Q12-Q13