CS W3134: Data Structures in Java

Lecture #14: Recursion and sorts
10/26/04
Janak J Parekh

Administrivia

- Exams returned
  - Mean: 37.93
  - Median: 41
  - StDev: 9.69
  - Max: 50
  - Min: 11
- We'll go over the exam now
- HW#3 due Thursday
  - Questions?
  - Matthew's holding OH tomorrow morning because he was ill Monday morning; check webboard for details
- HW#2 returned next Tuesday

Agenda

- Recursion, continued

FindMax, revisited

- Last time, we divided in half and searched both halves
- *Double* recursion
- We can something similar with only one recursive call…

Towers of Hanoi

- Three pegs
- Disks all on one peg
- Want to move it to third peg
- Second peg is a "work peg"
- Can't move a disk until all smaller disks have been moved
- Basic intuition
  - Move the top disks from start to intermediate
  - Move the largest disk to destination
  - Move top disks from intermediate to destination

Hanoi (II)

- Three steps:
  - First, move pile from "from" to "inter", using "to" as a work peg
  - Then, move disk from "from" to "to"
  - Then, move remainder of pile from "inter" to "to", using "from" as a work peg
- This works because we don't have to put things consecutively, just that larger disks must go on top of smaller disks
- Page 278 for code
- Emacs for visualization (really!)

Mergesort

- Classic recursive algorithm
- Split arrays in half, sort each half, and then merge them together
  - "Divide and conquer"
- Sort is the "recursive" call
- Let's do it intuitively first
- Now, psuedocode…

Mergesort (II)

- Key aspect of code on page 287
- The header of the method contains enough information to perform the recursive call
  - In this case, partition information
- Efficiency?
  - Partition: O(1)
  - Merge: O(n)
  - How many times each have to be done? O(log n)
  - Ergo, O(n*log n)
- Disadvantage: lots of memory required

Radix Sort

- Radix is the "base" of a system of numbers
- Very simple, fast algorithm
- Sort by *digit*, one at a time
  - Sort on the 1s digit
  - Sort on the 10s digit; keep relative order of equal 10s the same, i.e., go left-to-right on the 1s digit
  - Sort the 100s digit
  - Etc.
- Problem: where to store intermediate results?
- Can sort 100 numbers in 2 passes! ~ O(2n)
- But… that's essentially O(n log n)!
- There's no free lunch, but this works very well for specialized keys

Quicksort: Partition

- Relies on concept of *partition*
  - A number s.t. two groups are formed: those smaller than the number, and those larger than the number
  - "Pivot"
  - Walk from both edges
    - If left is smaller than pivot, walk left
    - If right is larger than pivot, walk right
    - Otherwise, swap the two
    - What if we cross?
  - Last element is the pivot?
- Code? p. 338

Quicksort: Recursion

- Given pivot, we:
  - Partition the array in two;
  - Quicksort the left "half";
  - Quicksort the right "half".
- And recurse!
- That's it (p. 338)
  - Well, must be very, very careful
- Analysis?
  - *Usually* O(n log n), and in-memory
- But there are some problems…

Next time

- Finish Quicksort
- Start trees