# CS W3134: Data Structures in Java

Lecture #7: Ordered lists, complexity, sorts
9/28/04
Janak J Parekh

## Administrivia

- HW#1 due Thursday!
  - Any general questions?
- Akash has replaced William as TA
  - Note changed office hours

## Agenda

- Finish ordered lists
- Big-Oh notation (complexity)
- Sorting algorithms
- Start stacks, if time allows

## Ordered lists

- Find( ), redux
  - What's the *stopping condition* for find( )?

## Costs

- How much do each of the previous operations cost in the *worst case*?
  - Most are linear, some are unit
- Binary search is special – it's better than linear time
  - Divide the range by half until too small to divide further == # of comparisons needed
  - Reverse: what's the range that can be covered with *n* steps? (Book page 63)
  - i.e., $r = 2^s$
  - What's this expressed as in terms of s?
    - $s = \log_2 r$
  - Algorithm grows *logarithmically*

## Formalizing costs

- We're going to approach this informally
- Time to insert one element is some constant *K*
  - e.g., T(N) = K
- Time to search for an element (linearly) is T(N) = K * N
- "Big-Oh Notation": upper-bound on worst-case time
  - We drop the constant K – for *sufficiently large N*, the constant is unimportant
  - To be precise, we find a function F(x), where T(x) is O(F(x)) if $|T(x)| \leq K|F(x)|$ for some $x > c$
  - The idea of doubling your computer's speed is embedded in K
  - T(N) = O(N), for example

## Examples of costs

- For lists using arrays?
  - Linear search: O(N)
  - Etc.
  - Draw a graph of the comparative costs, page 72
- What are bad about arrays?
  - Slow search in unordered, slow insert in ordered – can we speed both? Yes
  - Fixed size: can we change that? Yes

## Sorts

- Bubble (p. 85)
  - Sort pairwise repeatedly
  - Biggest placed each time
- Selection (p. 89)
  - Search for smallest, swap with first
  - Search for smallest, swap with second
- Insertion (p. 95)
  - Take the next one, and put it into the existing sorted subset
- All $O(n^2)$
  - But they're not the exact same performance
- Let's write out a little bit of psuedocode for each

## Sorts II

- Lexicographical comparisons?
- Stability of existing items?
- Sidebar: Comparable interface
  - All you have to do is implement boolean compareTo(Object o)
  - Generally a good thing to program to, I prefer to book's example
  - Arrays.sort()

## Stacks and Queues

- Useful programmer's tools, will encounter it in many places
  - Very easy and fast to implement
  - Runs very fast as well
- "Restricted access": no index – only manipulate one item at a time
- More abstract – the underlying implementation is unimportant or not remotely similar to the structure, unlike lists

## Stacks

- Basic operations: "LIFO" strategy
  - Push
  - Pop
  - Peek
- Analogy: mail basket
  - Not as rigorous as a real stack, of course
- Another analogy: life
  - Conversations
  - Workday
- Extraordinarily simple!

## Array-based stacks

- Limited size; ways to get around this
- Decoupled from array index!
- Very simple to implement
  - Keep *top* variable, initialized to -1
- Boundary conditions?
- Complexity bounds?
  - Apart from simplicity, biggest reason to use

## Next time…

- Reasons to use stacks
- Queues
- Arithmetic expression parsing