CSCI 136 Data Structures & Advanced Programming

Jeannie Albrecht Lecture 11 Mar 5, 2014

Administrative Details

Lab 4 is today

- Extend Vector to sort with a Comparator
- More details on next slide...
- Midterm during lab next Wednesday (3/12)
 - Can everyone start at 1:00? Should be finished by 3:00.
 Lab 5 will still be posted next week but is **optional**
 - Extra credit opportunity
 Covers book, lab, lecture material through next Monday
 - I will post a sample exam and a study guide

Lab 4 Details

- Lab 4
 - Extend Vector to sort with a Comparator (described in textbook)
 - Create Student class and read in data into Vector of Students
 - Create Comparators to sort students in different ways (name, phone, mailbox, etc)
 - You might want to use a ComparableAssociation at some point
 - Same as normal Association, but has compareTo
 - I was unable to update the phone book data 🙁

Last Time

- Discussed searching
 - Linear searching
 - Binary searching

Today's Outline

- Wrap up searching
- Briefly learn about Comparables and Comparators
- Discuss sorting algorithms
- High-level Goals
 - Understand sorting algorithms
 - · Understand the tradeoffs between algorithms
 - Learn how to prove properties of recursive programs using induction

Recap: Binary Search

- · Find a name in the phonebook
- Guess a number between 1 and 100
- These are examples of binary search
- Why does it work?
- Rule out as much of search space as possible with each guess
- What assumption (about the data) does it rely on?
- Is it recursive? Let's look at the code...
- <u>http://www.cs.williams.edu/~jeannie/cs136/lectures/lecture9/SortSearchDemo/</u>

Binary Search

Complexity:

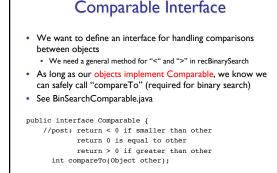
- · Each recursive call takes at most 2 array comparisons So how many calls when n=2^k? (list size is a power of 2...) Size of list during recursion: $[2^k, 2^{k-1}, ..., 2^0, 0]$ \rightarrow k+1 calls with 2 comparisons each → $2(k+1) = 2(\log n + 1) = O(\log n)$ for list of size n
- Show: recBinarySearch takes 2(log n + 1) comparisons. (Assume list size is power of 2...)
- Overall O(log n) comparisons
- Worst case: 2(log n + 1)
- Can we reduce the number of comparisons? Food for thought: BinSearchAlt.java

Is this version better?

- Worst case (element not found):
 - log n + 1 comparisons vs. 2(log n +1) comparisons
 - Twice as good!
- Average case
- · Only slight improvement in most cases
- Best case
 - · We now continue to recurse even if
 - a[mid] == value right away
 - This is actually worse than before...

Linear vs. Binary Search

- Which is better?
 - Linear is O(n) in average case • Binary is O(log n)
- So binary is better?? Yes, but...
- · Binary search requires ordering (i.e., pre-sorted data). We need the "<" and ">" operators
 - · Some objects already have these operators (ints, Strings) We want a uniform way of saying objects can be compared...
 - Make an interface! (Comparable interface)



compareTo in Card.java public class Card implements Comparable {

```
public int compareTo(Object obj) {
        Card other = (Card) obj;
if (suit != other.getSuit()) {
    return suit - other.getSuit();
         return rank - other.getRank();
  Note: The magnitude of the values returned is not important.
   We only care if it's +, -, or 0!

    compareTo defines a natural ordering of Objects
```

- · Can also use parameterized data types to avoid casting
- In lab this week, you'll explore another way to compare objects using Comparators

Comparators

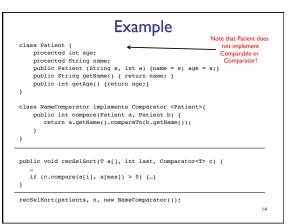
- Limitations with Comparable interface
- Only permits one order between objects
- What if it isn't the desired ordering?
- What if it isn't implemented?
- Solution: Comparators

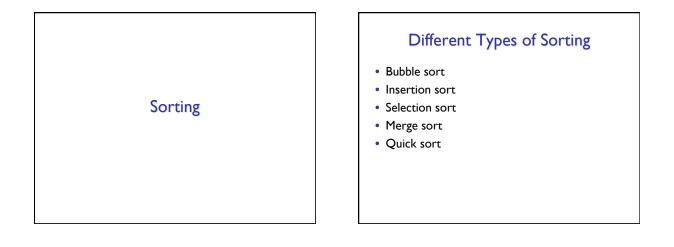
12

Comparators (Ch 6.8)

- A comparator is an object that contains a method that is capable of comparing two objects
- Sorting methods can apply a comparator to two objects when a comparison is to be performed
- Different comparators can be applied to the same data to sort in different orders or on different keys

public interface Comparator <E> { // pre: a and b are valid objects, likely of similar type // post: returns a value <, =, or > than 0 if a is less than, equal to, or greater than b public int compare(E a, E b); }





13

Bubble Sort

- Simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order
- Repeated until no swaps are needed
- Gets its name from the way smaller elements "bubble" to the front of the list
- Time complexity?
 O(n²)
- Space complexity?
 - O(n) total (no additional space is required)

Bubble Sort

• (|23<u>5</u>9)->(|23<u>5</u>9)

• (1235<u>9</u>) -> (1235<u>9</u>)

• First Pass: • $(5 \downarrow 329) \rightarrow (1 5 329)$ • $(1 5 \underline{3}29) \rightarrow (1 \underline{3}529)$ • $(1 2 359) \rightarrow (1 2 \underline{3}59)$ • $(1 2 \underline{3}59) \rightarrow (1 2 \underline{3}59)$

- $(135\underline{2}9) \rightarrow (13\underline{2}59)$ • $(132\underline{5}9) \rightarrow (132\underline{5}9)$ Second Pass:
- $(13259) \rightarrow (13259)$ • $(13259) \rightarrow (12359)$
- $(|23\underline{5}9) \rightarrow (|23\underline{5}9)$
- $(|235\underline{9}) \rightarrow (|235\underline{9})$
 - http://www.youtube.com/watch?v=lyZQPjUT5B4

Insertion Sort

- Simple sorting algorithm that works by building a sorted list one entry at a time
- Less efficient on large lists than more advanced sorting algorithms
- Advantages:
- Simple to implement and efficient on small lists
 Efficient on data sets which are already substantially sorted • Time complexity

19

- O(n²)
- Space complexity
 - O(n)

Insertion Sort								
• 5	7	0	3	4	2	6	Т	
• 5	7	0	3	4	2	6	I.	
• 0	5	7	3	4	2	6	I.	
• 0	3	5	7	4	2	6	I.	
• 0	3	4	5	7	2	6	I.	
• 0	2	3	4	5	7	6	I.	
• 0	2	3	4	5	6	7	I	
• 0	T	2	3	4	5	6	7	
								20