CompSci 100  
Prog Design and Analysis II

Announcements

• APT-ONE due today
• Markov and APT-TWO out today
• Recitation this week focuses on Markov. You should start it before then

Sept 14, 2010  
Prof. Rodger

What is the plan for the day

• How are objects compared in Java?
  – When would you want to compare?
  – What can’t be compared?

• Empirical and Analytical Analysis
  – Why are some lists different?
  – Why is adding in the middle fast?
  – Why is adding in the middle slow?

From Comparable to Comparator

• When a class implements Comparable then ...
  – Instances are comparable to each other
    • “apple” < “zebra”, 6 > 2
    • Sorting Strings, Sorting WordPairs, ...
    • Method compareTo invoked when ...
      – Comparable<...> types the parameter to compareTo
      – Return < 0, == 0, > 0 according to results of comparison

• Suppose we want to change how Strings compare
  – Or change class Foo implements Comparable<Foo>
    – What if we need more than one way to compare Foo’s?
java.util.Comparator

• How does sorting work in general and in Java?
  – Characteristics of Java library sort methods
  – What can be sorted?
  – How do you change how sorting works?
• APT ClientsList: example to explore Comparator
  – Creating new Comparator: nested class
    • Should it be public? Private? Matter?
    – Comparator could anonymous, but then issues.
• What does it mean to implement Comparable?
  – Other Java interfaces: cloneable, serializable, ...

What is a list in Java?

• Collection of elements, operations?
  – Add, remove, traverse, ...
  – What can a list do to itself?
  – What can we do to a list?
• Why more than one kind of list: Array and Linked?
  – Useful in different applications
  – How do we analyze differences?

Analyze Data Structures

```java
public double removeFirst(List<String> list) {
    double start = System.currentTimeMillis();
    while (list.size() != 1){
        list.remove(0);
    }
    double end = System.currentTimeMillis();
    return (end-start)/1000.0;
}
```

- Time taken to remove the first element?

<table>
<thead>
<tr>
<th>size</th>
<th>link</th>
<th>array</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.003</td>
<td>0.045</td>
</tr>
<tr>
<td>20</td>
<td>0.001</td>
<td>0.173</td>
</tr>
<tr>
<td>30</td>
<td>0.001</td>
<td>0.383</td>
</tr>
<tr>
<td>40</td>
<td>0.002</td>
<td>0.680</td>
</tr>
<tr>
<td>50</td>
<td>0.002</td>
<td>1.074</td>
</tr>
<tr>
<td>60</td>
<td>0.002</td>
<td>1.530</td>
</tr>
<tr>
<td>70</td>
<td>0.003</td>
<td>2.071</td>
</tr>
<tr>
<td>80</td>
<td>0.003</td>
<td>2.704</td>
</tr>
<tr>
<td>90</td>
<td>0.004</td>
<td>3.449</td>
</tr>
<tr>
<td>100</td>
<td>0.007</td>
<td>4.220</td>
</tr>
</tbody>
</table>

Removing first element
public double removeMiddleIndex(List<String> list) {
    double start = System.currentTimeMillis();
    while (list.size() != 1) {
        list.remove(list.size() / 2);
    }
    double end = System.currentTimeMillis();
    return (end - start) / 1000.0;
}

• What operations could be expensive here?
  – Explicit: size, remove
  – Implicit: find $n^{th}$ element