Big Oh, Linked Lists and Trees

7/2/2009

Today

• Big-Oh
• Lists
• Trees
• Intro to Project 2

<table>
<thead>
<tr>
<th>Activity</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn on the computer</td>
<td>3 minutes</td>
</tr>
<tr>
<td>Open Firefox</td>
<td>30 seconds</td>
</tr>
<tr>
<td>Go to gmail.com</td>
<td>10 seconds</td>
</tr>
<tr>
<td>Type in password</td>
<td>10 seconds</td>
</tr>
<tr>
<td>Read an email</td>
<td>30 seconds</td>
</tr>
</tbody>
</table>

Example

• An algorithm for processing a retail store's inventory takes:
  
  – 10,000 ms to read the initial inventory from disk, and then
  
  – 10 ms to process each transaction
  
  – Total time takes \((10,000 + 10 \cdot n)\) ms.
  
  – \(10 \cdot n\) is more important if \(n\) is very large.

Asymptotic Cost

• The constant coefficients can change:
  
  – If we had a faster computer or,
  
  – Use a different language or compiler.

• The constant factors can get smaller as technology improves.

• We want to express the speed of an algorithm independently of a specific implementation on a specific machine.

• We examine the cost of the algorithms for large input sets i.e. the asymptotic cost.
\[ T(n) \in O(f(n)) \quad \text{if and only if} \quad T(n) \leq c \cdot f(n) \quad \text{for all } n > N \]

**Big-Oh Summary**

- Specify bounding from above:
  - Big-Oh says how slowly code might run as its input size grows.
- Let \( n \) be the size of a program’s input. Let \( T(n) \) be a function that equals to the algorithm’s running time, given an input of size \( n \).
- Let \( f(n) \) be another function. We say that \( T(n) \in O(f(n)) \) if and only if \( T(n) \leq c \cdot f(n) \) whenever \( n \) is big, and for some constant \( c \).

**Simplifying stuff is important**

\[ f(n) \in O(5n^3 + 10n^2 + 1000n) \]

**Important Big-Oh Sets**

<table>
<thead>
<tr>
<th>Function</th>
<th>Common Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O(1) )</td>
<td>Constant</td>
</tr>
<tr>
<td>( O(\log n) )</td>
<td>Logarithmic</td>
</tr>
<tr>
<td>( O(\log^2 n) )</td>
<td>Log-squared</td>
</tr>
<tr>
<td>( O(\sqrt{n}) )</td>
<td>Root-n</td>
</tr>
<tr>
<td>( O(n) )</td>
<td>Linear</td>
</tr>
<tr>
<td>( O(n \log n) )</td>
<td>n log n</td>
</tr>
<tr>
<td>( O(n^2) )</td>
<td>Quadratic</td>
</tr>
<tr>
<td>( O(n^3) )</td>
<td>Cubic</td>
</tr>
<tr>
<td>( O(n^4) )</td>
<td>Quartic</td>
</tr>
<tr>
<td>( O(2^n) )</td>
<td>Exponential</td>
</tr>
<tr>
<td>( O(e^n) )</td>
<td>Bigger exponential</td>
</tr>
</tbody>
</table>

**What is the running time to do pair-wise comparison of all student solutions?**

- Assume that \( n \) is the number of students that submit project 1.
What is the running time to do pair-wise comparison of all student solutions and every student solution with every old solution?

- Assume that \( n \) is the number of students that submit project 1
- Assume that \( m \) is the number of previous semester solutions

Linked Lists

```java
abstract public class ListNode {
    abstract public Object first();
    abstract public ListNode rest();
    abstract public boolean isEmpty();
}
```

EmptyListNode

```java
class EmptyListNode extends ListNode {
    public EmptyListNode () {
        throw new IllegalArgumentException ("EmptyListNode constructor takes no arguments.");
    }
    public ListNode rest () {
        throw new IllegalArgumentException ("EmptyListNode constructor takes no arguments.");
    }
    public boolean isEmpty () {
        return true;
    }
}
```

NonemptyListNode

```java
class NonemptyListNode extends ListNode {
    private Object myFirst;
    private ListNode myRest;

    // cons in Scheme.
    public NonemptyListNode (Object first, ListNode rest) {
        myFirst = first;
        if (rest == null) {
            myRest = new EmptyListNode ( );
        } else {
            myRest = rest;
        }
    }
}
```

```
this(  ) ?!?!?
```

```java
public NonemptyListNode (Object first) {
    this (first, new EmptyListNode ( ));
}
```

- Appears in a constructor
- Calls a different constructor
- Must be the FIRST LINE of the constructor
Abstract ListNode stuff

EmptyListNode
NonemptyListNode
myFirst
myRest

public class List {
    private ListNode myHead;
    public List () { myHead = null; }
    public boolean isEmpty () { return myHead == null; }
}

private static class ListNode {
    private Object myFirst;
    private ListNode myRest;
    private ListNode (Object first, ListNode rest) {
        myFirst = first;
        myRest = rest;
    }
    private ListNode (Object first) {
        myFirst = first;
        myRest = null;
    }
}

Trees

Lists and ListNodes