Big Oh, Linked Lists and Trees

7/2/2009

Today

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

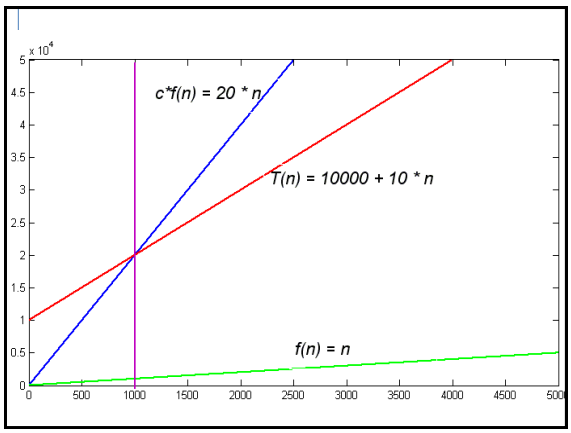
Turn on the computer	3 minutes
Open Firefox	30 seconds
Go to gmail.com	10 seconds
Type in password	10 seconds
Read an email	30 seconds

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 * n)* ms.
  - *10 * 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*.



$$T(n) \in O(f(n))$$

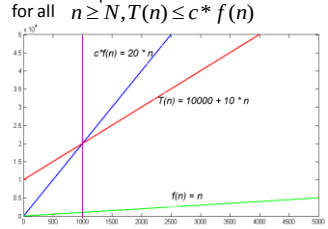
if and only if

$$T(n) \le c * f(n)$$

for all n > N

Definition

- *O(f(n))* is the **set of all** functions *T(n)* that satisfy:
  - There exist positive constants *c* and *N* such that, for all $n \ge N, T(n) \le c * f(n)$



- In the example above: *c = 20*, and *N = 1000*.

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 $T(n) \le c * f(n)$ whenever *n* is big, and for some constant *c*.

Simplifying stuff is important

$$f(n) \in O(5n^3 + 10n^2 + 1000\mathrm{n})$$

Important Big-Oh Sets

Subset of

Function	Common Name
$O(1)$	Constant
$O(\log n)$	Logarithmic
$O(\log^2 n)$	Log-squared
$O(\sqrt{n})$	Root-n
$O(n)$	Linear
$O(n \log n)$	n log n
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(n^4)$	Quartic
$O(2^n)$	Exponential
$O(e^n)$	Bigger exponential

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

```
abstract public class ListNode {

    abstract public Object first();
    abstract public ListNode rest();
    abstract public boolean isEmpty();

}
```

EmptyListNodes

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

NonemptyListNodes

```
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( ) ?!?!?

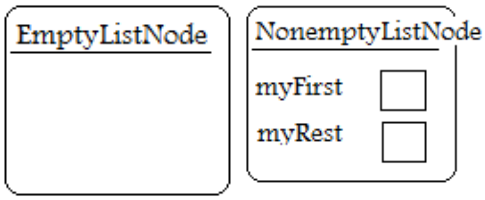
```
public NonemptyListNode (Object first) {
    this (first, new EmptyListNode ( ));
}
```

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

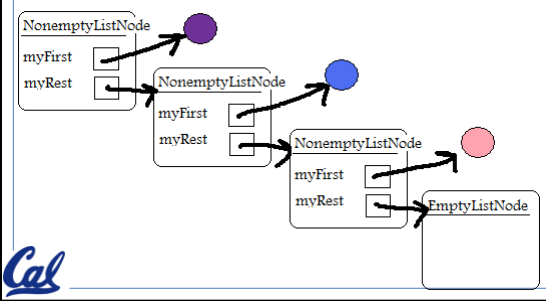
NonemptyListNodes

```
// car in Scheme.
public Object first ( ) {
    return myFirst;
}
// cdr in Scheme.
public ListNode rest ( ) {
    return myRest;
}
public boolean isEmpty ( ) {
        return false;
}
```

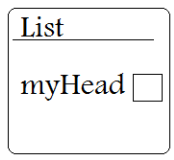
Abstract ListNode stuff



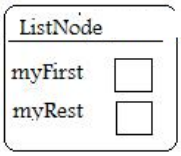
Abstract ListNode stuff



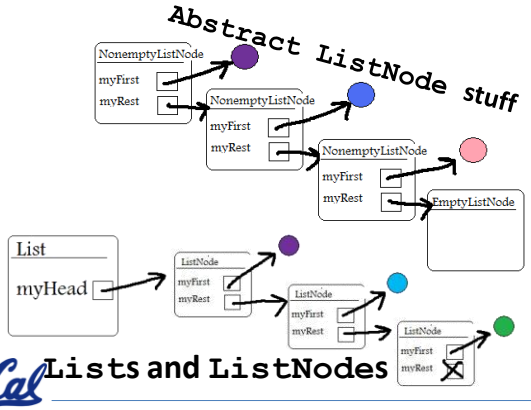

```
public class List {

    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;
    }
}
```



Abstract ListNode stuff



Lists and ListNodes

Trees