Announcements

• One APT next week – BSTCount
  – Will do in class
• Written Assignment lists/trees due March 31
• New assignment Boggle due April 7
  – Will do part of it in lab (last time, and next lab)
• Today
  – More on trees and analysis with trees
  – Recurrence relations
More on Trees

• Focus on binary trees
  – Includes binary search trees
  – Process tree: root (subtree) (subtree)
  – Analyze recursive tree functions
    • Recurrence relation
Review: Printing a search tree in order

• When is root printed?
  – After left subtree, before right subtree.

```java
void visit(TreeNode t){
    if (t != null) {
        visit(t.left);
        System.out.println(t.info);
        visit(t.right);
    }
}
```

• Inorder traversal
• How long for n nodes?
  – \(O(\_\_\_\_)\)?
Tree functions

• Compute height of a tree, what is complexity?

```java
int height(Tree root) {
    if (root == null) return 0;
    else {
        return 1 + Math.max(height(root.left),
                             height(root.right));
    }
}
```

• Modify function to compute number of nodes in a tree, does complexity change?
  – What about computing number of leaf nodes?
Balanced Trees and Complexity

- A tree is height-balanced if
  - Left and right subtrees are height-balanced
  - Left and right heights differ by at most one

```java
boolean isBalanced(Tree root){
    if (root == null) return true;
    return
        isBalanced(root.left) &&
        isBalanced(root.right) &&
        Math.abs(height(root.left) - height(root.right)) <= 1;
}
```
What is complexity?

• Consider worst case? What does the tree look like?
• Consider average case? Assume trees are “balanced” in analyzing complexity
  – Roughly half the nodes in each subtree
  – Leads to easier analysis

• How to develop recurrence relation?
  – What is T(n)?
  – What other work is done?

• How to solve recurrence relation – formula for recursion
• Plug, expand, plug, expand, find pattern
  – A real proof requires induction to verify correctness
Solving Recurrence Relation

• Recurrence relation is a formula that models how much time the method takes.
• T(n) – the time it takes to solve a problem of size n
• Basis – smallest case you know how to solve, such as n=0 or n=1
• If two recursive calls formula might be:
  – T(n) = T(smaller problem) + T(smaller problem) + work to put answer together...
• On the right side, replace T(smaller) by plugging it in to the formula
Solving Recurrence Relation (cont)

• Continue replacing the $T(\text{smaller})$ values until you see a pattern – use $k$ for the pattern

• Then solve for $k$ with respect to $N$ to get a basis case that has a constant value – this removes the $T$ term from the right hand side of the equation and you are left with $T(N) = \text{to terms of } N$ and can easily compute big-Oh
What is average big-Oh for height?

• Write a recurrence relation
• $T(0) =$
• $T(1) =$
• $T(n) =$
What is worst case big-Oh for height?

• Write a recurrence relation
• \( T(0) = \)
• \( T(1) = \)
• \( T(n) = \)
What is average case big-Oh for is-balanced?

• Write a recurrence relation
• $T(1) =$
• $T(n) =$
Recognizing Recurrences

• Solve once, re-use in new contexts
  – T must be explicitly identified
  – n must be some measure of size of input/parameter
    • T(n) is for quicksort to run on an n-element array

\[
\begin{align*}
T(n) &= T(n/2) + O(1) & \text{binary search} & O(\ ) \\
T(n) &= T(n-1) + O(1) & \text{sequential search} & O(\ ) \\
T(n) &= 2T(n/2) + O(1) & \text{tree traversal} & O(\ ) \\
T(n) &= 2T(n/2) + O(n) & \text{quicksort} & O(\ ) \\
T(n) &= T(n-1) + O(n) & \text{selection sort} & O(\ )
\end{align*}
\]

• Remember the algorithm, re-derive complexity
Recognizing Recurrences

• Solve once, re-use in new contexts
  – T must be explicitly identified
  – n must be some measure of size of input/parameter
    • T(n) is for quicksort to run on an n-element array

\[
\begin{align*}
T(n) &= T(n/2) + O(1) & \text{binary search} & \mathcal{O}(\log n) \\
T(n) &= T(n-1) + O(1) & \text{sequential search} & \mathcal{O}(n) \\
T(n) &= 2T(n/2) + O(1) & \text{tree traversal} & \mathcal{O}(n) \\
T(n) &= 2T(n/2) + O(n) & \text{quicksort} & \mathcal{O}(n \log n) \\
T(n) &= T(n-1) + O(n) & \text{selection sort} & \mathcal{O}(n^2)
\end{align*}
\]

• Remember the algorithm, re-derive complexity
BSTCount APT

• Given values for a binary search tree, how many unique trees are there?
  – 1 value = one tree
  – 2 values = two trees
  – 3 values = 5 trees
  – N values = ? trees

• Will memoize help?
Recurrences

• If \( T(n) = T(n-1) + O(1) \)… where do we see this?

\[
T(n) = T(n-1) + O(1)
\]

true for all \( X \) so,

\[
T(n-1) = T(n-2) + O(1)
\]

\[
T(n) = [T(n-2) + 1] + 1 = T(n-2) + 2
\]

\[
= [T(n-3) + 1] + 2 = T(n-3) + 3
\]

• True for 1, 2, so eureka! We see a pattern

\[
T(n) = T(n-k) + k, \quad \text{true for all } k, \quad \text{let } n=k
\]

\[
T(n) = T(n-n) + n = T(0) + n = n
\]

• We could solve, we could prove, or remember!

CompSci 100e, Spring 2011