A Helpful Link on Tree Traversals and Iterators

- Thanks to Austin Tam for this one
- http://nova.umuc.edu/~jarc/idsv/lesson1.html
- I put the link under Resources in Day 10 on the schedule page.

Strong Induction Practice

Analysis of a simple algorithm

Strong Induction

To prove that p(n) is true for all $n \geq n_0$, it is sufficient to show the following two things:

a) $p(n_0)$ is true.

b) for all $k > n_0$, if p(j) is true for all j with $n_0 \leq j < k$, then p(k) is also true.

Strong Induction Example

```
int i = n;
 while ( i > 1 )
     i = i/2;   //integer division
```

Let **T(n)** be the number of iterations of the above loop. Formula for T(n)?

P(*n*): T(n) is $\lfloor \log n \rfloor$ (Recall that "log n" means "$\log_2 n$")

Show that P(*n*) is true for all positive integers n.

Base case: n=1: Clearly T(1) = 0, and $\lfloor \log 1 \rfloor = 0$

Induction step: n>1:

Assume that P(j) is true for all j with $1 \leq j < n$, and show that P(n) is true

Strong Induction Example – page 2

```
int i = n;
 while ( i > 1 )
     i = i/2;
```

P(*n*): T(n) is floor(log *n*)
Show that P(*n*) is true for all positive integers *n*

Induction step: Assume that P(j) is true for all k with $1 \leq j < n$, and show that P(n) is true

Case 1. n is even. Then T(n) = 1 + T(n/2)

Now we can use the induction hypothesis, since $1 \leq n/2 < n$. Thus

T(n) = 1 + floor(log (n/2)) — How can we simplify?
= 1 + floor (log n - log 2)) — What is log 2?
= 1 + floor (log n - 1) — What can we do with the 1 inside the floor?
= 1 + floor (log n) - 1
= floor (log n)

Strong Induction Example – page 3

```
int i = n;
 while ( i > 1 )
     i = i/2;
```

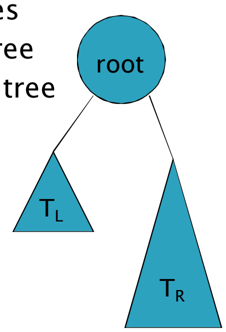
P(*n*): T(n) = floor(log *n*).
Show that P(*n*) is true for all positive integers *n*

Induction step. Assume that P(j) is true for all j with $1 \leq j < n$, and show that P(n) is true

Case 2. n is odd. You fill in the details. (on quiz, use handout)

Binary Tree: Recursive definition

- A Binary Tree is either
  - **empty**, or
  - **consists of**:
    - a distinguished node called the root, which contains an element, and two disjoint subtrees
    - A left subtree T_L, which is a binary tree
    - A right subtree T_R, which is a binary tree



Recap: Correct Merge Method

```
/**
 * Merge routine for BinaryTree class.
 * Forms a new tree from rootItem, t1 and t2.
 * Does not allow t1 and t2 to be the same.
 * Correctly handles other aliasing conditions.
 */
public void merge( AnyType rootItem,
                   BinaryTree<AnyType> t1, BinaryTree<AnyType> t2 )
{
    if( t1.root == t2.root && t1.root != null )
        throw new IllegalArgumentException( );

        // Allocate new node
    root = new BinaryNode<AnyType>( rootItem, t1.root, t2.root );

        // Ensure that every node is in one tree
    if( this != t1 )
        t1.root = null;
    if( this != t2 )
        t2.root = null;
}
```

Student suggestion: We could have used **duplicate** here to get different trees with the same content.

Can you see why we might not want to use **duplicate** for the normal case?

6

utine for the BinaryTree class

Properties of Binary Trees

Size vs Height

Size and Height of Binary Trees

- If T is a tree, we'll often write h(T) for the height of the tree, and N(T) for the number of nodes in the tree
- For a particular h(T), what are the upper and lower bounds on N(T)?
  - **Lower:** N(T) ≥ _______ (prove it by induction)
  - **Upper** N(T) ≤ _______ (prove it by induction)
  - Thus _______ ≤ N(T) ≤ _________
- Write bounds for h(T) in terms of N(T)
  - Thus _______ ≤ h(T) ≤ _________

Extreme Trees

- A tree with the maximum number of nodes for its height is a **full** tree.
  - Its height is O(log N)
- A tree with the minimum number of nodes for its height is essentially a _____ _____
  - Its height is O(N)
- Height matters!
  - We will see the the algorithms for search, insertion, and deletion in a Binary search tree are O(h(T))

Binary Search Trees

Definition
Algorithms
Properties

Binary Search Trees

- A Binary Search Tree (BST) is a Binary tree with the following additional properties:
  1. The elements are Comparable, and are not allowed to be null
  2. No duplicates are allowed
  3. If the tree T is non-empty , all elements in T's left subtree are less than the root element
  4. if the tree T is non-empty, all elements in T's right subtree are greater than the root element
  5. Both subtrees are BSTs
- Advantage: Easy (and fast?) lookup of items
  - O(height(T))

BST Algorithms

```
public class BinarySearchTree<T extends Comparable<T>> {

 private BinaryNode<T> root;

 public BinarySearchTree() {
   this.root = null;
 }

 // Does this tree contain obj?
 public boolean contains(T obj)

 // insert obj, if not already there
 public void insert(T obj)

 // delete obj, if it's there
 public void delete(T obj)
```

Check out the BST project from your repository, and join me in writing these methods.

Tree Balancing

- Algorithms are O(h(T)).
- What are the bounds on h(T)?
- Can we keep it at the best case?
  - Rebalance after every insertion?
  - D B F C E A G H
  - The problem with this …
  - Other alternatives?