Homework 7 due 10pm Friday, November 4th (5 hours to write code and complete quiz)
Program 3 due 10 pm Sunday, November 6th

Last Time
Recursion
• more practice writing/analyzing recursion
• execution tree tracing
Searching
Categorizing ADTs Part 1
General Trees
• implementing

Today
General Trees
• determining tree height (from last time)
Binary Trees
• implementing
Tree Traversals
Categorizing ADTs Part 2
Binary Search Tree (BST)
• BSTnodes
• BST class

Next, Next, Next Time
Read: continue Binary Search Trees
Binary Search Tree (BST)
• implementing print
• implementing lookup, insert, delete
• complexities of BST methods
CS Options/Courses
Binary Tree

The BinaryTreenode Class:

```java
class BinaryTreenode<T> {
    private T data;
    private BinaryTreenode<T> leftChild;
    private BinaryTreenode<T> rightChild;

    public BinaryTreenode(T item) {
        data = item;
        leftChild = null;
        rightChild = null;
    }
    ...
}
```

→ Draw a picture of the memory layout of a BinaryTreenode:

The BinaryTree Class:

```java
public class BinaryTree<T> {
    private BinaryTreenode<T> root;
    private int size;

    public BinaryTree() {
        root = null;
        size = 0;
    }
    ...
}
```

→ Draw a picture of the memory layout of an empty binary tree:

→ Draw a picture of the memory layout of a binary tree with a root node having 2 children:
Tree Traversals

Goal: visit every node in the tree exactly once

Level-order

General Tree

Binary Tree

Pre-order

Post-order

In-order
Practice – Binary Tree Traversals

→ **List the nodes** using a pre-order traversal.

![Binary Tree Diagram](pre-order-diagram.png)

→ **List the nodes** using a post-order traversal.

![Binary Tree Diagram](post-order-diagram.png)

→ **List the nodes** using an in-order traversal.

![Binary Tree Diagram](in-order-diagram.png)
Categorizing ADTs Part 2

Position-Oriented

Value-Oriented
Binary Search Tree (BST)

Goal

Example

Ordering Constraint
Practice - Identifying Binary Search Trees

→ Identify which trees below are valid BSTs.

A

B

C

D

E

F
class BSTnode<K> { 

    private K key;
    private BSTnode<K> left, right;

    public BSTnode(K key, BSTnode<K> left, BSTnode<K> right) { 
        this.key = key;
        this.left = left;
        this.right = right;
    }

    public K getKey() { return key; }
    public BSTnode<K> getLeft() { return left; }
    public BSTnode<K> getRight() { return right; }

    public void setKey(K newK) { key = newK; }
    public void setLeft(BSTnode<K> newL) { left = newL; }
    public void setRight(BSTnode<K> newR) { right = newR; }
}

→ Draw a picture of the memory layout of a BSTnode:
import java.io.*;  //for PrintStream

public class BST<K extends Comparable<K>> {

    private BSTnode<K> root;
    public BST() { root = null; }

    public void insert(K key) throws DuplicateException {
    }

    public void delete(K key) {
    }

    public boolean lookup(K key) {
    }

    public void print(PrintStream p) {
    }

    //add helpers ...
}