CS 367 - Introduction to Data Structures  
Week 9, Spring 2017

Homework 6 due 10 pm Wednesday, March 15th

Program 3  
SPRING BREAK: next week, no classes, no consulting

Last Week  
PriorityQueue, array-based heap, recursion, practice, complexity analysis,

This Week  
Read: Trees, Binary Search Trees  
Recursion  
• Towers of Hanoi  
• more practice writing/analyzing recursion (from last time)  
• execution tree tracing  
Searching  
Categorizing ADTs Part 1  
General Trees  
• implementing  
• determining tree height  
Binary Trees  
• implementing  
Tree Traversals  
Categorizing ADTs Part 2  
Binary Search Tree (BST)  
• BSTnodes  
BST class

Next Week: (after Spring Break)  
Read: Module 10, continue Binary Search Trees  
Binary Search Tree (BST)  
• implementing print  
• implementing lookup, insert, delete  
• complexities of BST methods  
• CS Options/Courses
Algorithm

solveTowers( count, src, dest, spare ) {

}

Complexity
Problem size N is

1. Equations

2. Table

3. Verify

4. Complexity
Picking Lottery Numbers

What are your odds of winning the lottery? It depends on the number of possible combinations given how many numbers you have to pick and over what range:

- **Supercash** - choose 6 out of 39 numbers (range 1 – 39)
- **Megabucks** - choose 6 out of 49 numbers (range 1 – 49)

**N Choose K:** How many combinations of K things can you make from N things?

**Recursive Definition:**

\[ c(n,k) = \]

→ Implement the c(n,k) method.
Execution Tree Tracing of c(n,k)
Searching

Linear Search:

Binary Search:
Categorizing ADTs Part 1: based on Layout

LINEAR

HIERARCHICAL

GRAPHICAL
General Tree

The Tree Node Class:
```java
class Treenode<T> {
    private T data;
    private ListADT<Treenode<T>> children;
    ...
}
```

→ **Draw a picture** of the memory layout of a Treenode
(assume an ArrayList is used for the ListADT):

The Tree Class:
```java
public class Tree<T> {
    private Treenode<T> root;
    private int size;

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

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

→ **Draw a picture** of the memory layout
of a general tree with a root node having 3 children:
Determining Height of a General Tree

Recall the height of a tree is the length of a path from the root to the deepest leaf.

→ **Write a recursive definition** for the height of a general tree.

→ **Complete the recursive height method** based on the recursive definition. Assume the method is added to a Tree class having a root instance variable.

```java
public int height() {
    // Your implementation here
}
```
Binary Tree

The Tree Node 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 Tree 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

Pre-order

Post-order

In-order
Practice – Binary Tree Traversals

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

```
H
  A
  B
  C
  G
  E
  J
  D
  I
  F
```

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

```
H
  A
  B
  C
  G
  E
  J
  D
  I
  F
```

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

```
H
  A
  B
  C
  G
  E
  J
  D
  I
  F
```
Categorizing ADTs Part 2
Binary Search Tree (BST)

Example

Ordering Constraint
Practice - Identifying Binary Search Trees

→ Identify which trees below are valid BSTs.

A

B

C

D

E

F
→ Draw a picture of the memory layout of a Treenode:

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; }
}
BST Class

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 ...