CS 367 - Introduction to Data Structures
Thursday, March 17, 2016

Homework 6 due 10 pm tomorrow, March 18th

Program 3 due 10 pm Sunday, March 27th

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 Tree Node Class:

```
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:

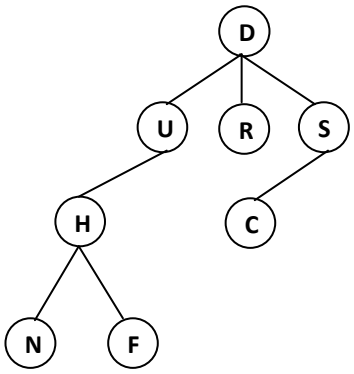
```
public class BinaryTree<T> {
   private BinaryTreenode<T> root;
   private int size;

   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

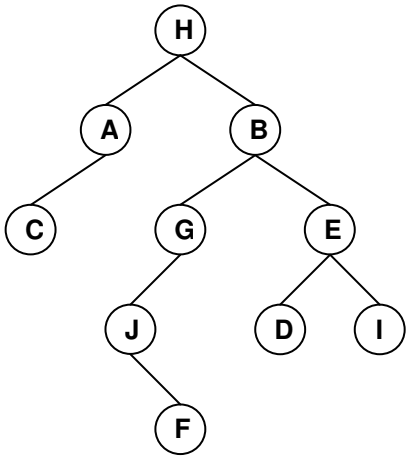
Pre-order

Post-order

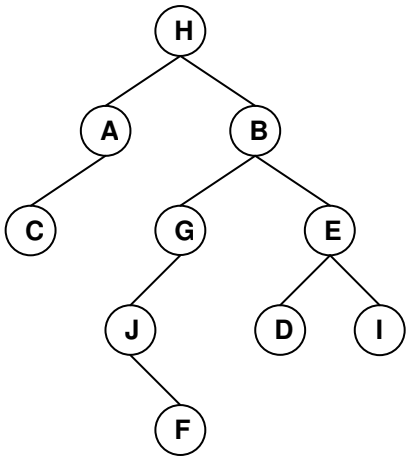
In-order

Practice – Binary Tree Traversals

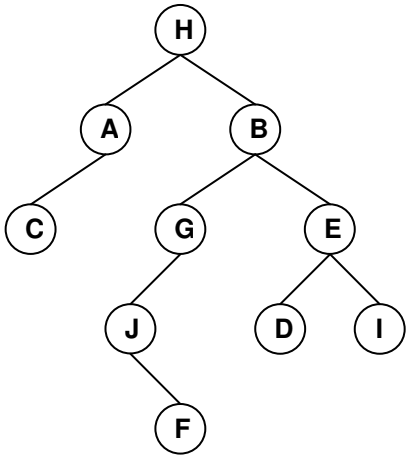
→ **List the nodes** using a pre-order traversal.



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



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



Categorizing ADTs Part 2

Binary Search Tree (BST)

-

Goal

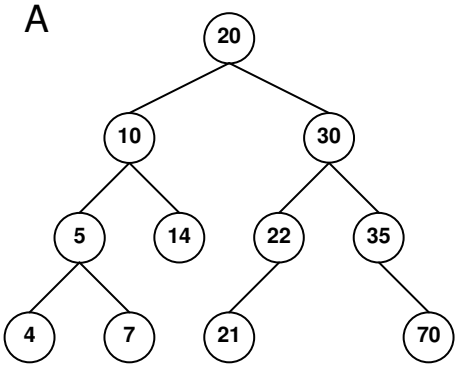
Example **2 3 6 7 10 12 13 15 17 19 22 24 26 27 30**

Ordering Constraint

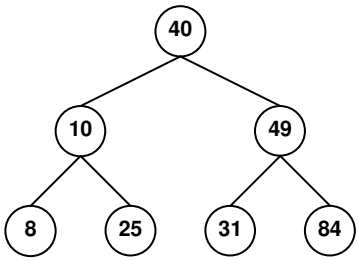
Practice - Identifying Binary Search Trees

→ Identify which trees below are valid BSTs.

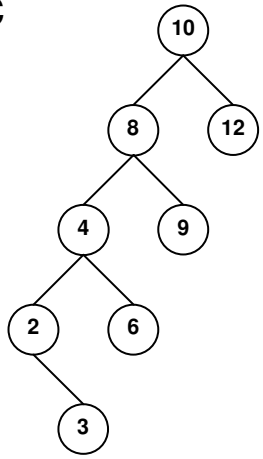
A



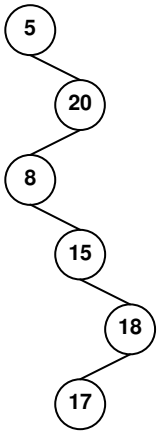
B



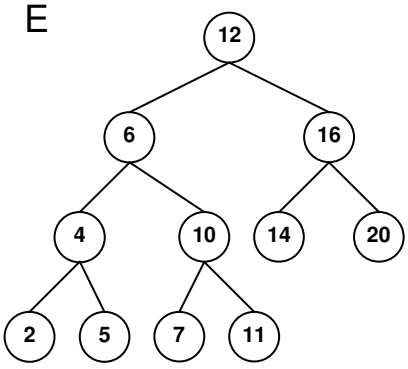
C



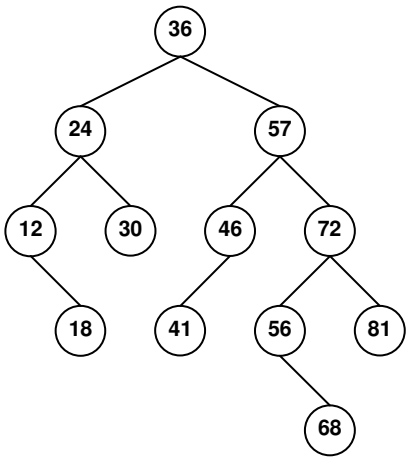
D



E



F



BSTnodeS

→ 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 ...




}
```