CST8130 – Data Structures

Professor : Dr. Anu Thomas
Email: thomasa@algonquincollege.com
Office: T314

Trees

Linked Lists vs Arrays

Storing data in a linked lists was an improvement over arrays:

- we had to have a solid block of consecutive memory addresses to hold the elements (or pointers to the elements);
- we were not constrained by size (arrays needs us to declare a fixed size – which we could then accommodate by reallocating larger array and copying existing array to new larger one – overhead)

Operation	Arrays – best case	Linked list – best case
Insertion in order	O(n)	O(n)
Searching	O(log n)	O(n)
Deletion	O(n)	O(n)
Sorting	O(n log n)	

Linked List - problems

Storing data in a linked lists was NOT an improvement over arrays because:

- we lost our ability to access any element directly – we had to “follow the links” to find an element – hence we lost our ability to perform binary search

Trees will give us back the ability to “binary search”

Trees Terminology

Tree – consists of finite set of elements, called nodes, and a finite set of directed lines, called branches, that connect the nodes.

Root – first node of a tree

Parent node – node with successors

Child node – node with a predecessor

Leaf node – node with no successor

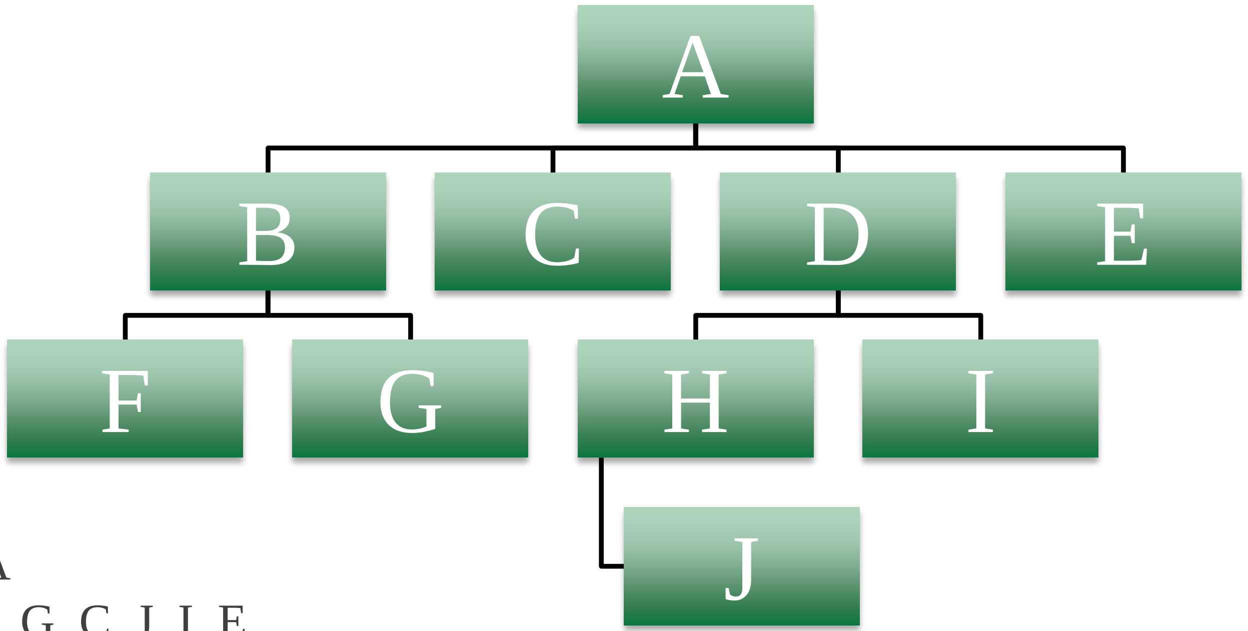
Internal node – node that is not root or leaf

Sibling nodes – have same parent

Level of a node – its distance from root

Height of tree – level of the leaf in the longest path from root plus 1

Example



Root – A
Leaf – F, G, C, J, I, E
Internal nodes – B, D, H
B is parent of F, G
F is child of B
F and G are siblings
Level of I is 2
Height of tree is 4
Subtree starting at D consists of nodes D, H, I, J

Binary Tree

Tree in which no node can have more than two subtrees. So, a node can have at most two children –the left child and the right child

Properties:

Height of tree (max) = N (number of nodes)   How???

Height of tree (min) = $\log_2 N + 1$

Conversely…….N (min) = H

N (max) = $2^H - 1$

Balance/Complete

Balance = difference in height between left and right subtrees

We say a tree is **balanced** when height of its subtrees differs by no more than 1.

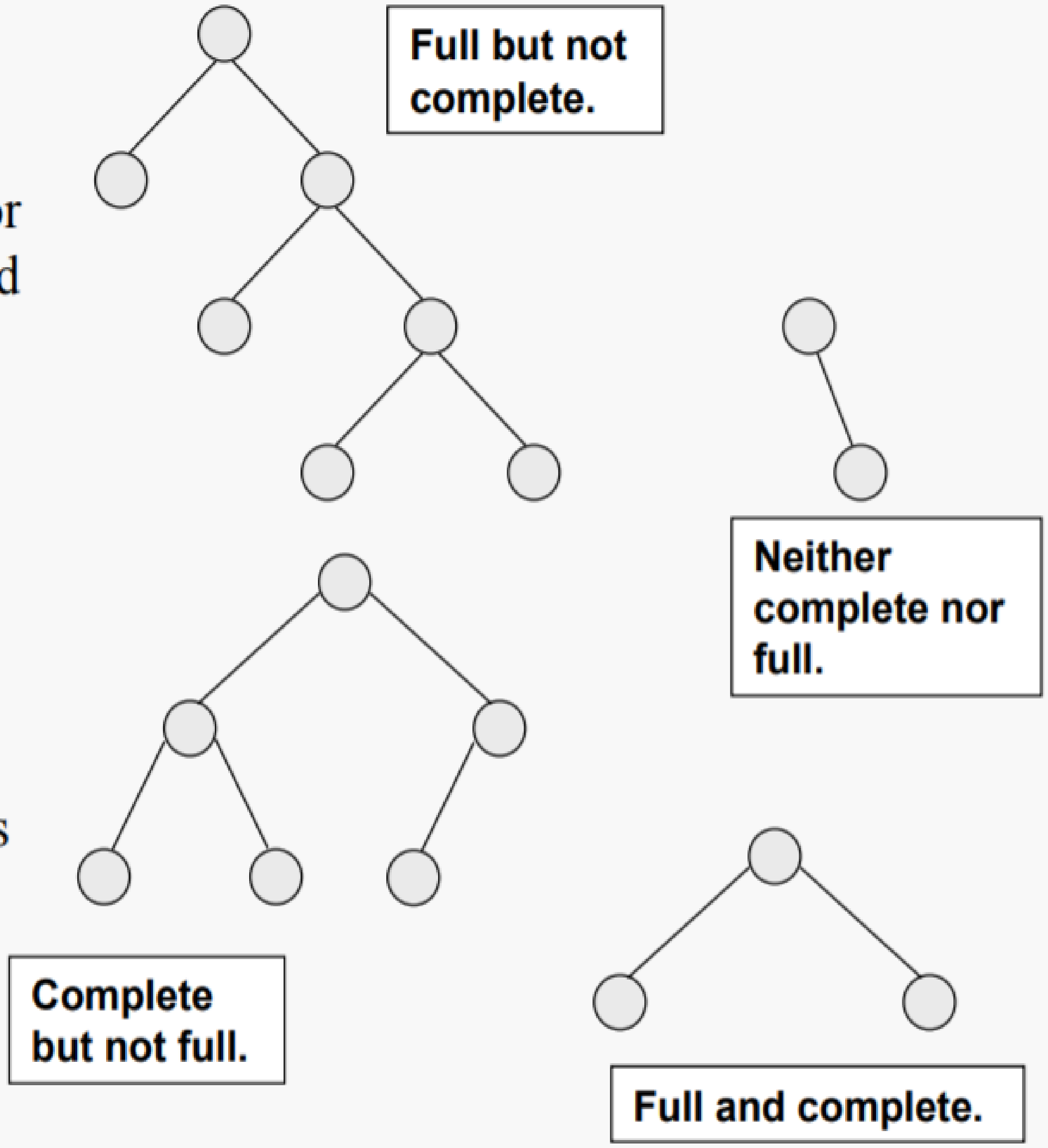
Full Binary tree - each node has exactly zero or two children.

Complete tree – if all levels except possibly the last are completely full, and the last level has all its nodes to the left side.

Here are two important types of binary trees. Note that the definitions, while similar, are logically independent.

Definition: a binary tree T is *full* if each node is either a leaf or possesses exactly two child nodes.

Definition: a binary tree T with n levels is *complete* if all levels except possibly the last are completely full, and the last level has all its nodes to the left side.



Binary Search Tree

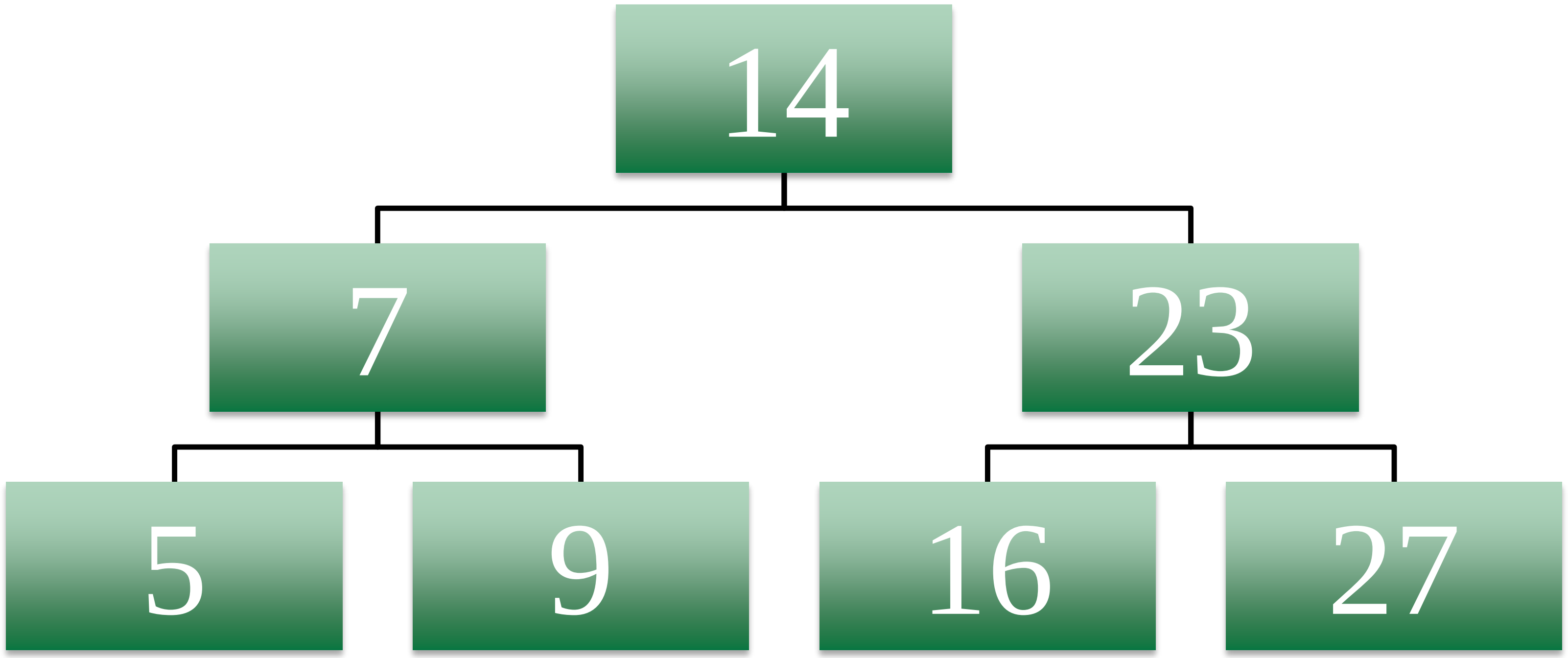
Binary search tree is a

- binary tree in which the left subtree contains key values less than the root and the right subtree contains key values greater than the root;
- Each subtree is a binary search tree

Consider a binary search tree with node values:

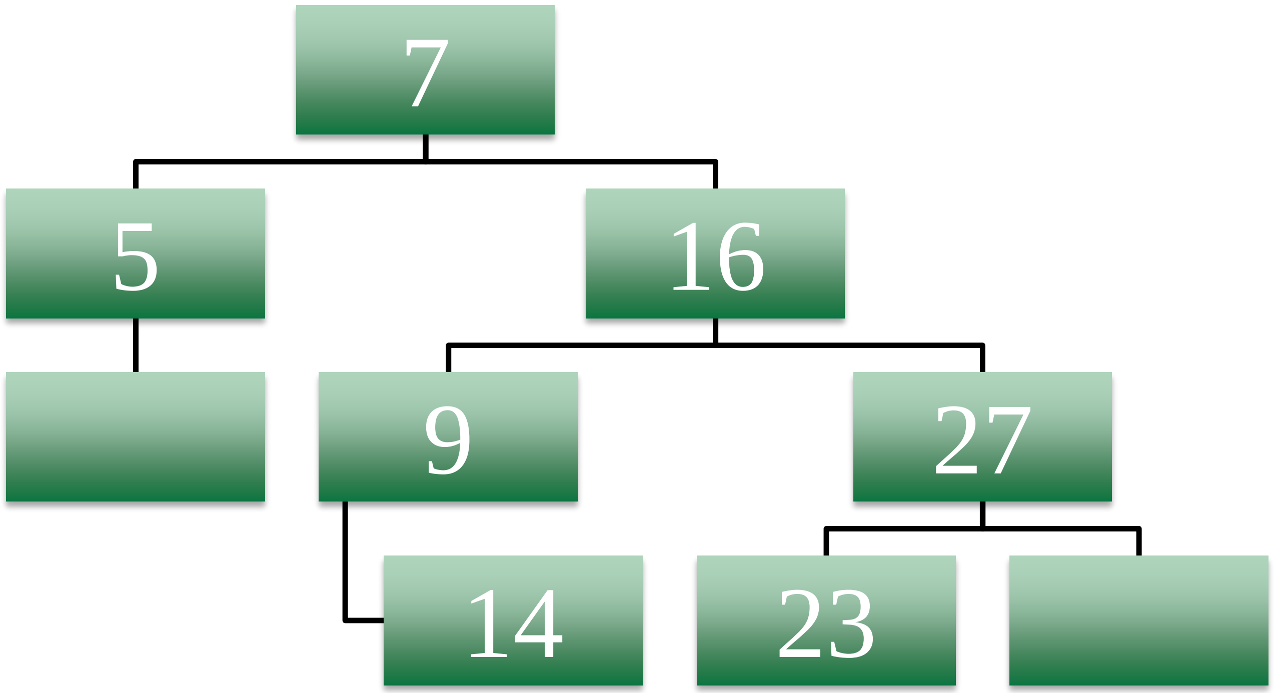
14, 23, 7, 16, 9, 27, 5

Example



Example 2

Consider a binary search tree with node values:
7, 16, 9, 27, 5, 14, 23



Code - BinaryTreeNode class

```
public class BinaryTreeNode {
    private int data;
    private BinaryTreeNode left;
    private BinaryTreeNode right;

    public BinaryTreeNode(int data) {
        left = null;   right = null; this.data = data;
    }
    public int getData() {   return data;      }
    public BinaryTreeNode getLeft() {     return left;      }
    public BinaryTreeNode getRight() {   return right;    }
```

Code – BinaryTree class

```
public class BinaryTree {
	BinaryTreeNode root = null;
```

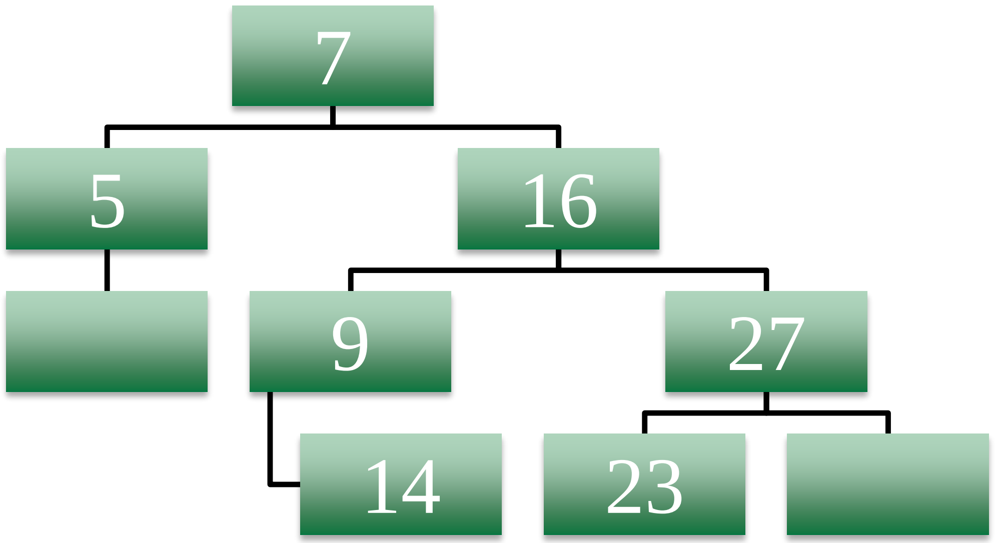
Insert …inside BinaryTree class

```
public void insertInTree (int newData) {
    if (root == null)
        root = new BinaryTreeNode(newData);
    else root.insert(newData);
}
```

Insert …inside BinaryTreeNode class

```
public void insert (int newData) {
    if (newData  <  data) {
        if (left == null)
            left = new BinaryTreeNode(newData);
        else left.insert(newData);
    } else if (newData >  data) {
        if (right == null)
            right = new BinaryTreeNode(newData);
        else right.insert(newData);
    } else
        System.out.println("Duplicate – not adding" +
                                                        newData);
}
```

Traversals



PreOrder Traversal – root processed first, then left subtree, then right subtree

7, 5, 16, 9, 14, 27, 23

InOrder Traversal – the left subtree is processed, then the root, then right right subtree

5, 7, 9, 14, 16, 23, 27

PostOrder Traversal – left subtree is processed, then right subtree, then the root

5, 14, 9, 23, 27, 16, 7

InOrder Traversal – BinaryTree Code

```
public void displayInOrder () {
    displayInOrder (root);
}
public void displayInOrder (BinaryTreeNode subRoot){
    if (subRoot == null)   return;
    displayInOrder (subRoot.getLeft());
    System.out.print(" " + subRoot.getData() + " ");
    displayInOrder (subRoot.getRight());
}
```

Example…

```
public static void main(String[] args) {
    BinaryTree tree = new BinaryTree();

    tree.insertInTree(6);
    tree.insertInTree(3);
    tree.insertInTree(9);
    tree.insertInTree(1);
    tree.insertInTree(15);
    tree.insertInTree(7);
    tree.displayInOrder();
```

Displays: 1 3 6 7 9 15

Questions?