Binary Trees
Tree Structures

• A tree is
  – A hierarchical data structure whose point of entry is the root node
  – This structure can be partitioned into disjoint subsets
  – These subsets are themselves trees and are also subtrees of the tree
Definitions

• A *tree* is an abstract data type
  – one entry point, the *root*
  – Each node is either a *leaf* or an *internal node*
  – An internal node has 1 or more *children*, nodes that can be reached directly from that internal node.
  – The internal node is said to be the *parent* of its child nodes
Properties of Trees

- Only access point is the root
- All nodes, except the root, have one parent
  - like the inheritance hierarchy in Java
- Traditionally trees drawn upside down
Properties of Trees and Nodes

- **siblings**: two nodes that have the same parent
- **edge**: the link from one node to another
- **path length**: the number of edges that must be traversed to get from one node to another

Path length from root to this node is 3
More Properties of Trees

• *depth*: the path length from the root of the tree to this node

• *height of a node*: The maximum distance (path length) of any leaf from this node
  – a leaf has a height of 0
  – the height of a tree is the height of the root of that tree
Binary Trees

- There are many variations on trees but we will work with binary trees.
- binary tree: a tree with at most two children for each node.
  - the possible children are normally referred to as the left and right child.

```
  parent
    /   \
   /     \
left child right child
```
Binary Trees

• A hierarchical data structure which
  – May be empty (empty tree or null tree)
  – All nodes can have degree of 0, 1 or 2
  – A node is explicitly defined as a left child or a right child
  – Consists of subtrees called left subtree and right subtree
Binary Tree - concept

• A *binary tree* structure is either empty or consists of an element,
  – called the *Root element*
• and two distinct *branches*  
  – a *left subtree*
  – and *right subtree*
• The left and right subtrees may be empty or contain elements  
  – a *left element*  
  – and *right element*
The Binary Tree grows…

- The left and right subtrees may also contain distinct branches
- And each of those subtrees may contain elements
Binary Tree taxonomy

- A **leaf** is an element whose left and right subtrees are empty.
- A **node** is an element containing at least one subtree
In a binary tree $t$, *height* is the number of branches from the root to the farthest leaf.

-depth($x$), the depth of an element $x$ is the number of branches from the root element to $x$. If $x$ is the root element, depth($x$) = 0

-level($x$), the level of $x$, is the same as the depth of $x$. 

![Diagram of a binary tree with indicated depths and heights]
(a) Height 2
(b) Height 3
(c) Height 5
Full Binary Tree

- *full binary tree*: a binary tree is which each node was exactly 2 or 0 children
Complete Binary Tree

- complete binary tree: a binary tree in which every level, except possibly the deepest is completely filled. At depth $n$, the height of the tree, all nodes are as far left as possible
Perfect Binary Tree

• *perfect binary tree*: a binary tree with all leaf nodes at the same depth. All internal nodes have exactly two children.

• a perfect binary tree has the maximum number of nodes for a given height

• a perfect binary tree has $2^{(n+1)} - 1$ nodes where $n$ is the height of a tree
  – height = 0 -> 1 node
  – height = 1 -> 3 nodes
  – height = 2 -> 7 nodes
  – height = 3 -> 15 nodes
Traversals of a Binary Tree
Tree Traversal

• Traversing a tree means visiting each node in a specified order

• This process is not as commonly used as finding, inserting, and deleting nodes. One reason for this is that traversal is not particularly fast.

• But traversing a tree is useful in some circumstances and the algorithm is interesting
• There are generally two types of traversal:
  – breadth first
  – depth first.

• There are three variants for depth first traverse a tree. They're called preorder, inorder, and postorder.
1-Inorder Traversal

- An inorder traversal of a binary search tree will cause all the nodes to be visited in ascending order, based on their key values. If you want to create a sorted list of the data in a binary tree, this is one way to do it.
The simplest way to carry out a traversal is the use of recursion. A recursive method to traverse the entire tree is called with a node as an argument. Initially, this node is the root. The method needs to do only three things:

1. Call itself to traverse the node's left subtree
2. Visit the node
3. Call itself to traverse the node's right subtree
Java Code for Traversing

private void inOrder(node localRoot)
{
    if(localRoot != null)
    {
        inOrder(localRoot.leftChild);
        localRoot.displayNode();
        inOrder(localRoot.rightChild);
    }
}
Determine the order in which the elements would be accessed during an in-order traversal.
Answer: 12, 30, 40, 50, 86, 90, 100
2-Postorder

1. Call itself to traverse the node's left subtree.
2. Call itself to traverse the node's right subtree.
3. Visit the node.
2. **postOrder** (t)
   
   ```
   if (t is not empty) {
       postOrder(leftTree(t));
       postOrder(rightTree(t));
       access the root element of t;
   } // if
} // postOrder traversal
```

**Left – Right – Root**
3-Preorder

1. Visit the node.
2. Call itself to traverse the node's left subtree.
3. Call itself to traverse the node's right subtree.
preOrder (t)  
  
  {  
    if (t is not empty)  
    {  
      access the root element of t;  
      preOrder (leftTree (t));  
      preOrder (rightTree (t));  
    } // if  
  } // preOrder traversal

**Root – Left – Right**
Breadth-first traversal  
(level order)

• To perform a breadth-first traversal of a non-empty binary tree, first access the root element, then the children of the root element, from left to right, then the grandchildren of the root element, from left to right, and so on.
Perform a breadth-first traversal of the following binary tree.
Answer: A, B, C, D, E, R, S, F, G, L
Perform the other traversals of that tree:

inOrder (Left-Root-Right)
postOrder (Left - Right - Root)
preOrder (Root - Left - Right)
Perform the other traversals of that tree:

inOrder (Left-Root-Right)
postOrder (Left - Right - Root)
preOrder (Root - Left - Right)