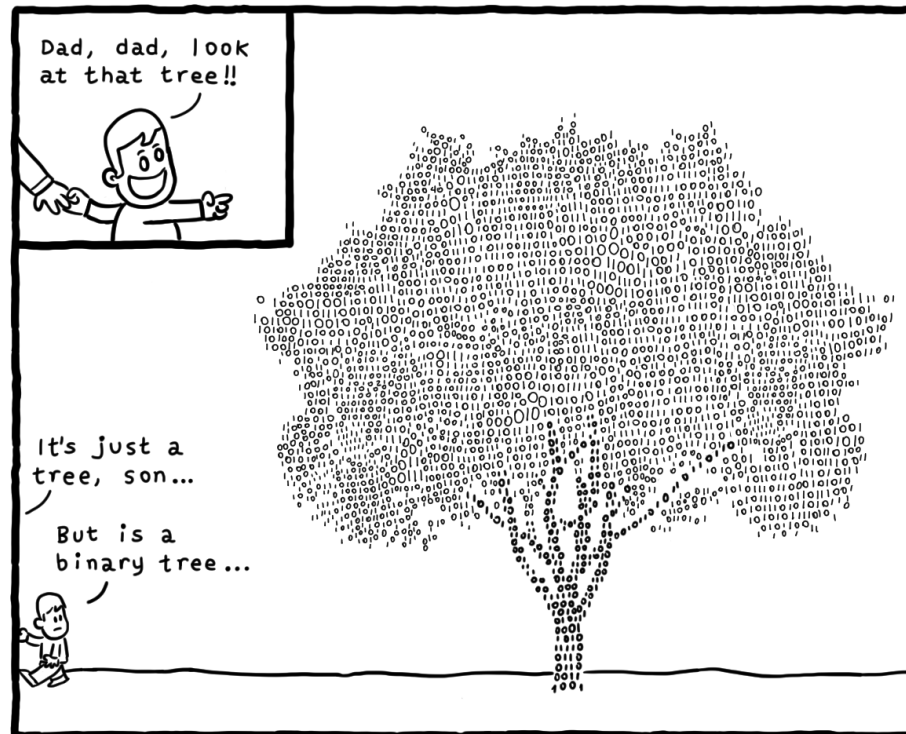


# BBM 201

## Data structures

### Lecture 11: Trees



# Content

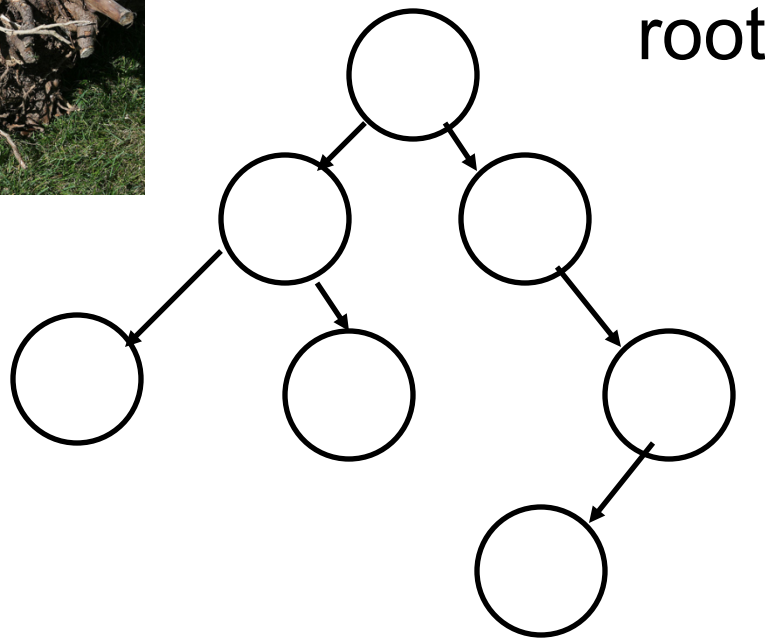
- Terminology
- The Binary Tree
- The Binary Search Tree

# Terminology

- Trees are used to represent relationships
- Trees are hierarchical in nature
  - “Parent-child” relationship exists between nodes in tree.
  - Generalized to ancestor and descendant
  - Lines between the nodes are called edges
- A **subtree** in a tree is any node in the tree together with all of its descendants

# Terminology

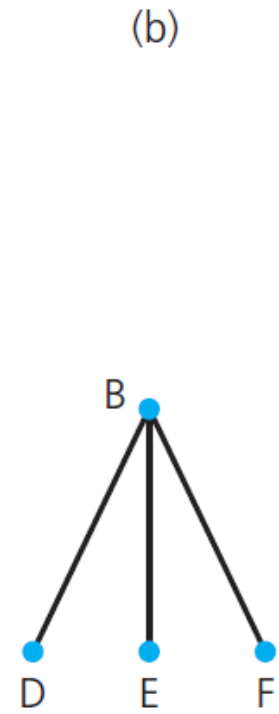
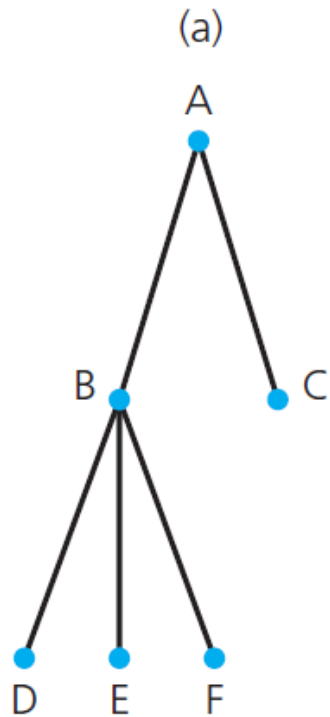
- Only access point is the root
- All nodes, except the root, have one parent
  - like the inheritance hierarchy in Java
- Traditionally trees are drawn upside down



leaves



# Terminology



(a) A tree;  
(b) a subtree of the tree in part a

# Terminology

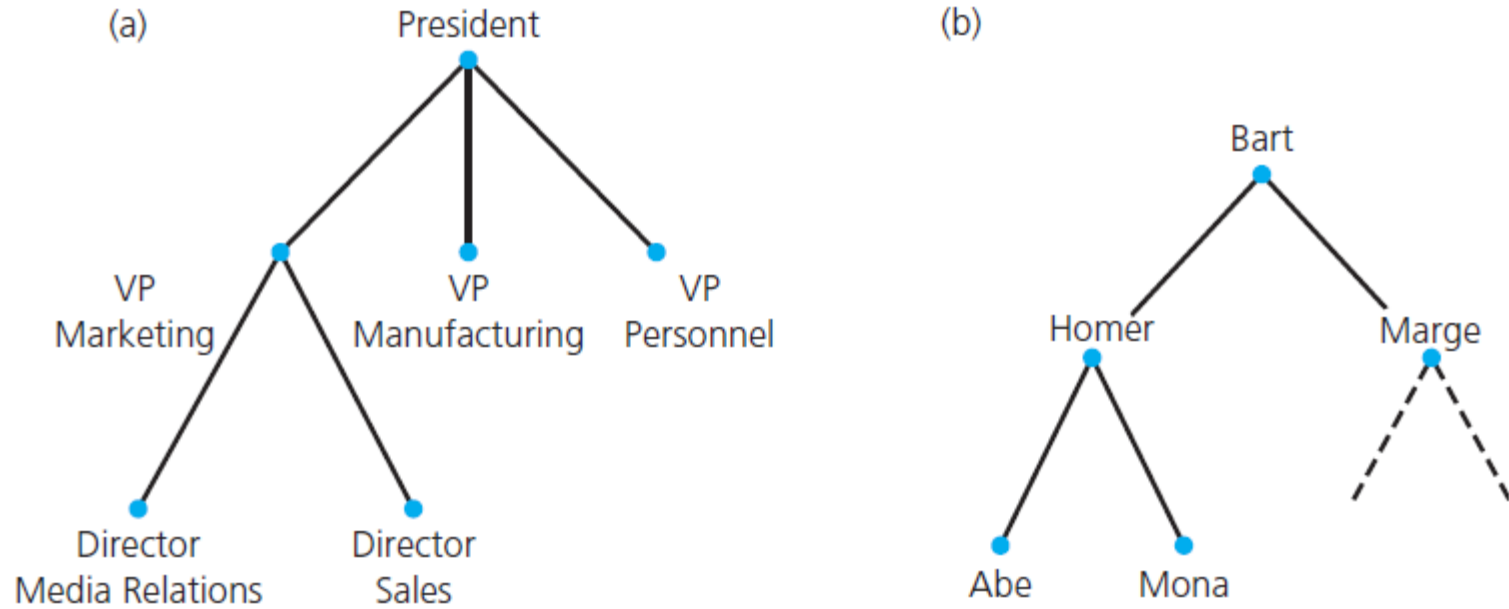
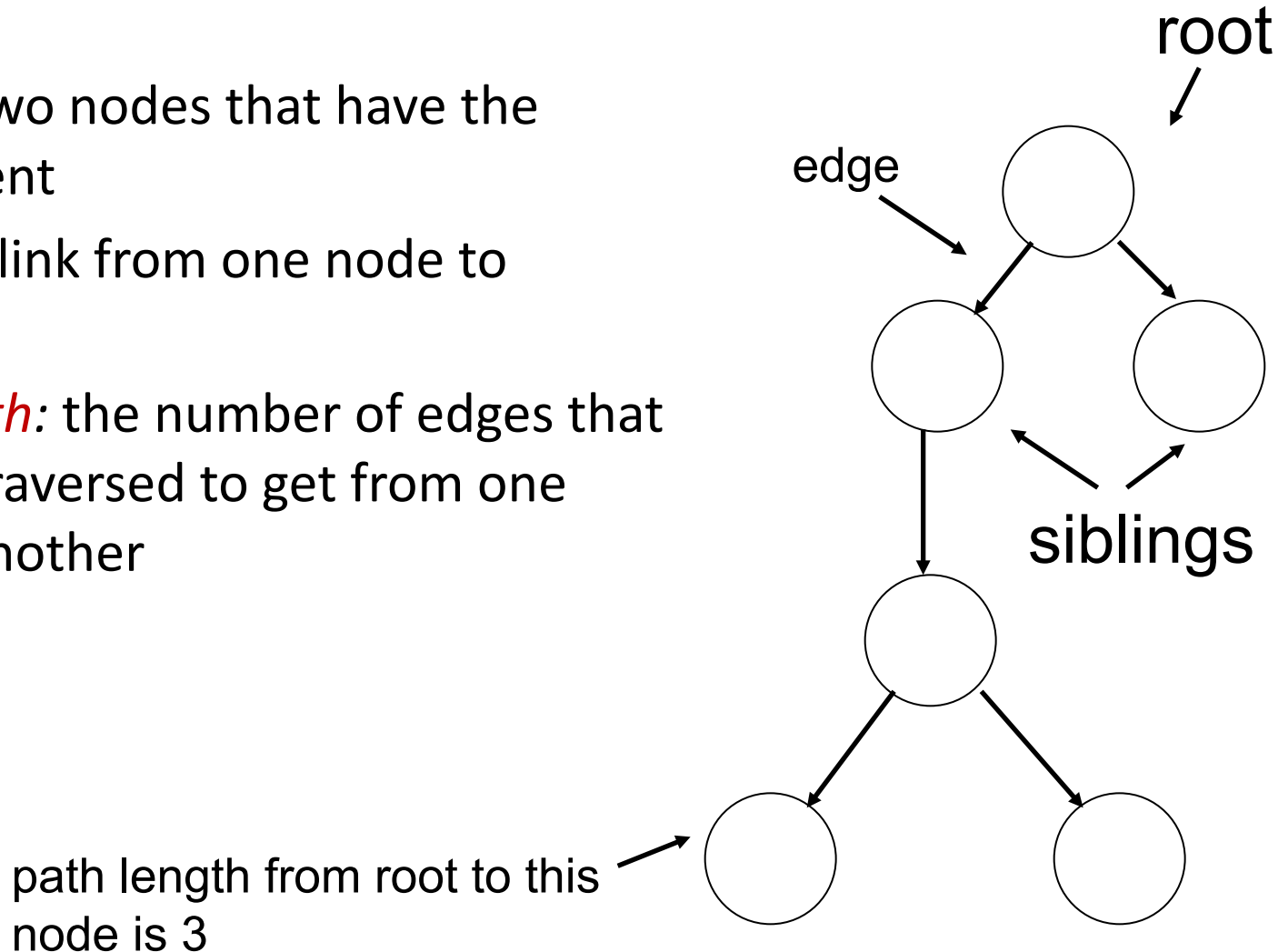


FIGURE 15-2 (a) An organization chart; (b) a family tree

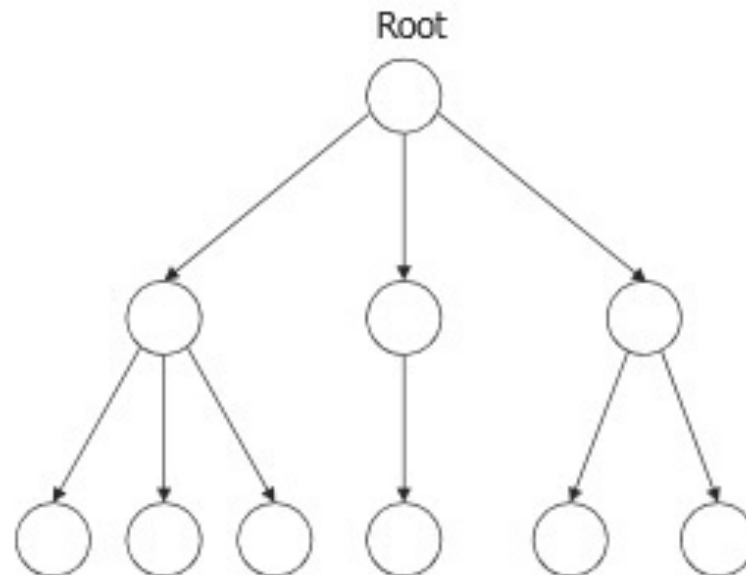
# Properties of Trees and Nodes

- *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



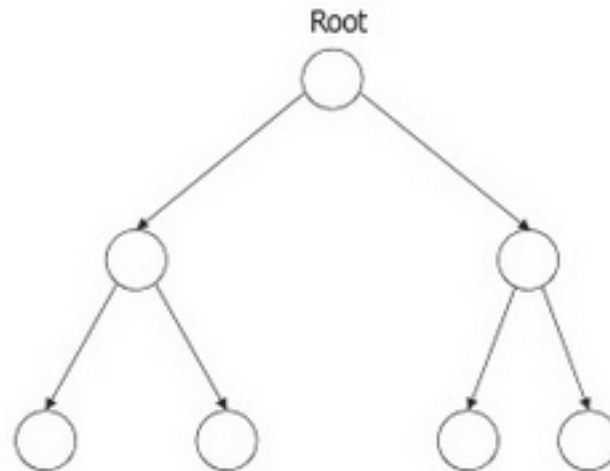
# General Tree

- A general tree is a data structure in that each node can have infinite number of children
- A general tree cannot be empty.



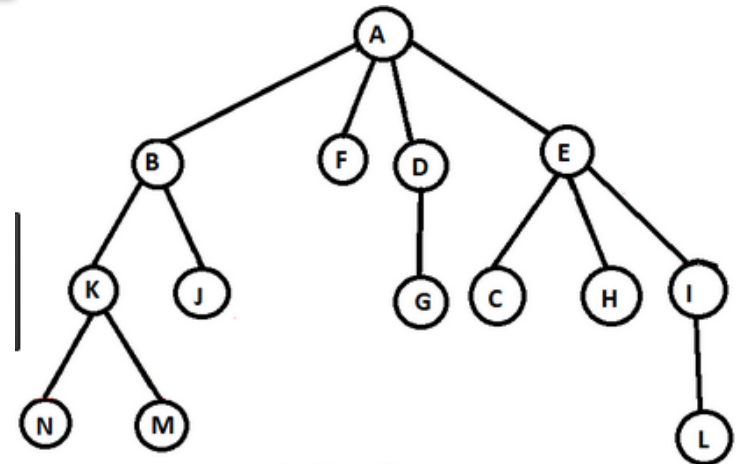
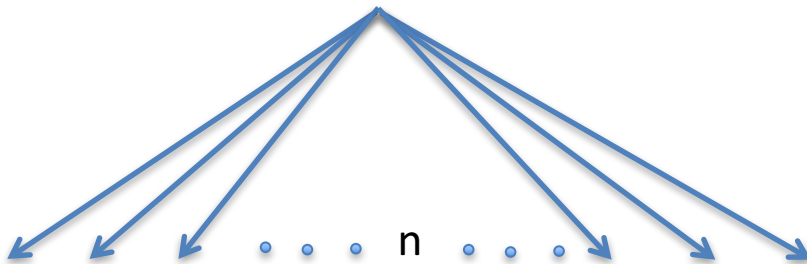
# Binary Tree

- A Binary tree is a data structure in that each node has at most **two nodes** left and right.
- A Binary tree can be **empty**.



# n -ary tree

- A generalization of a binary tree whose nodes each can have no more than n children.



An N-ary Tree.

# Example: Algebraic Expressions.

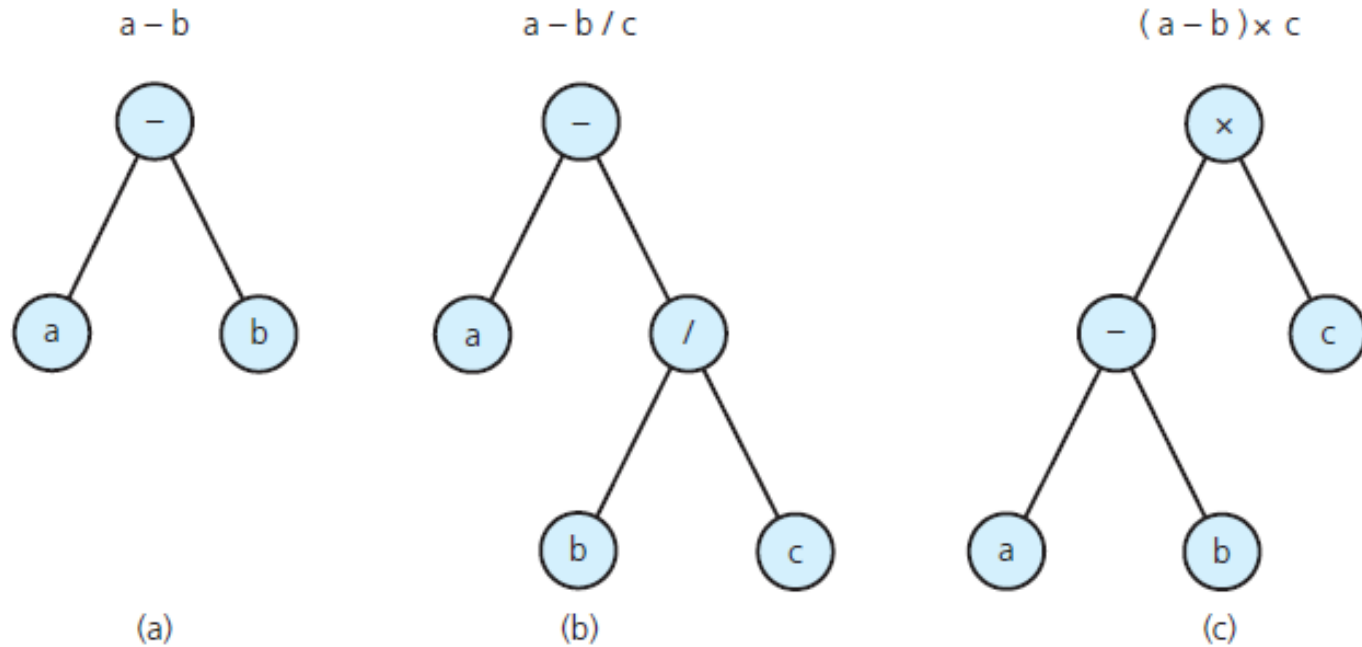
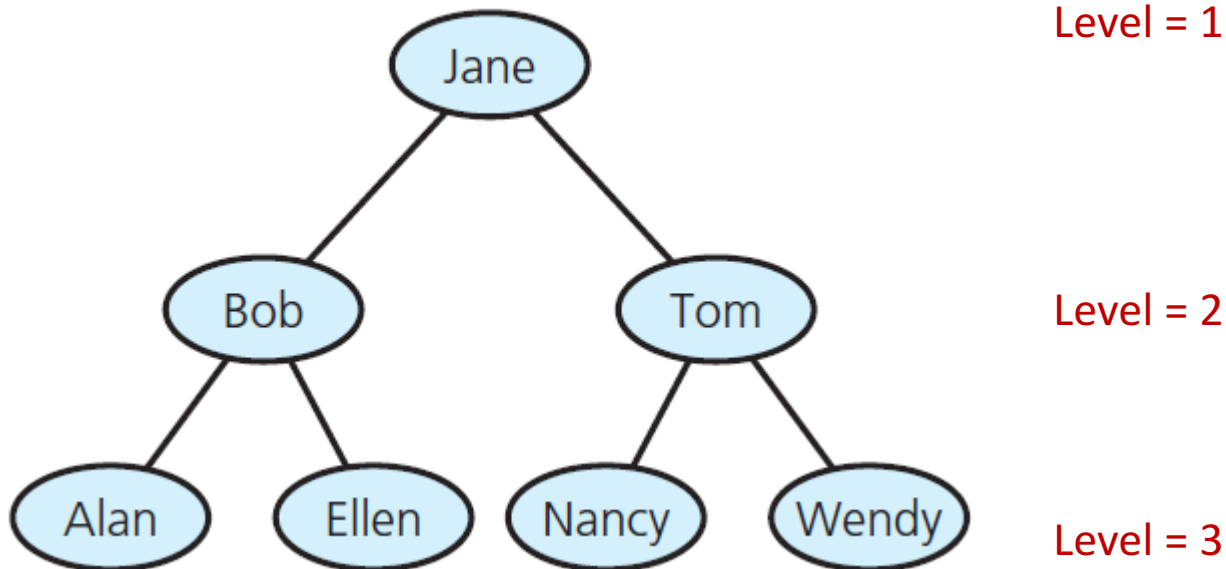


FIGURE 15-3 Binary trees that represent

# Level of a Node

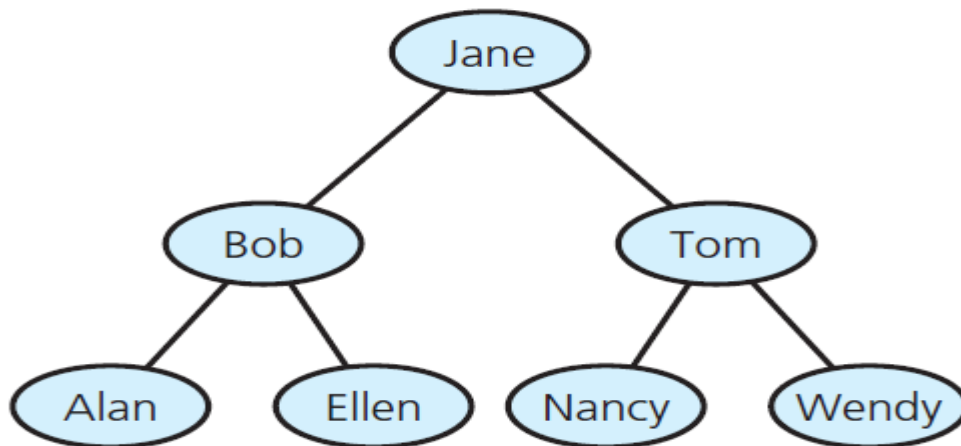
- Definition of the level of a node  $n$  :
  - If  $n$  is the root of  $T$ , it is at level 1.
  - If  $n$  is not the root of  $T$ , its level is 1 greater than the level of its parent.





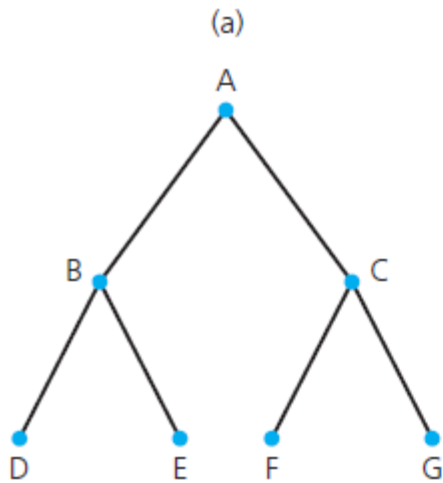
# Height of Trees

- Height of a tree  $T$  in terms of the levels of its nodes
  - If  $T$  is empty, its height is 0.
  - If  $T$  is not empty, its height is equal to the maximum level of its nodes.

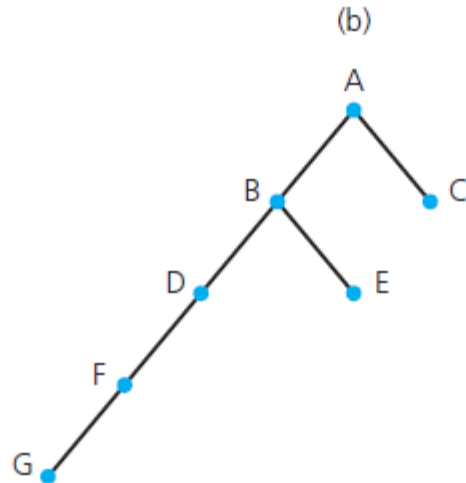


Height of tree = 3

# The Height of Trees



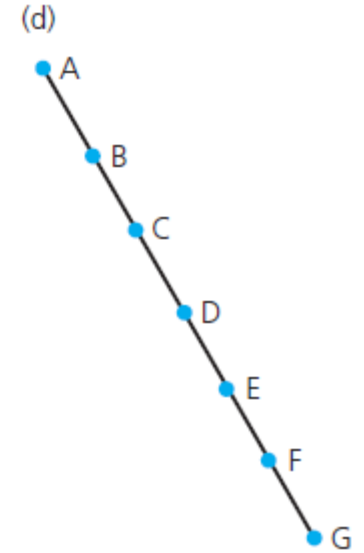
Height 3



Height 5



Height 7

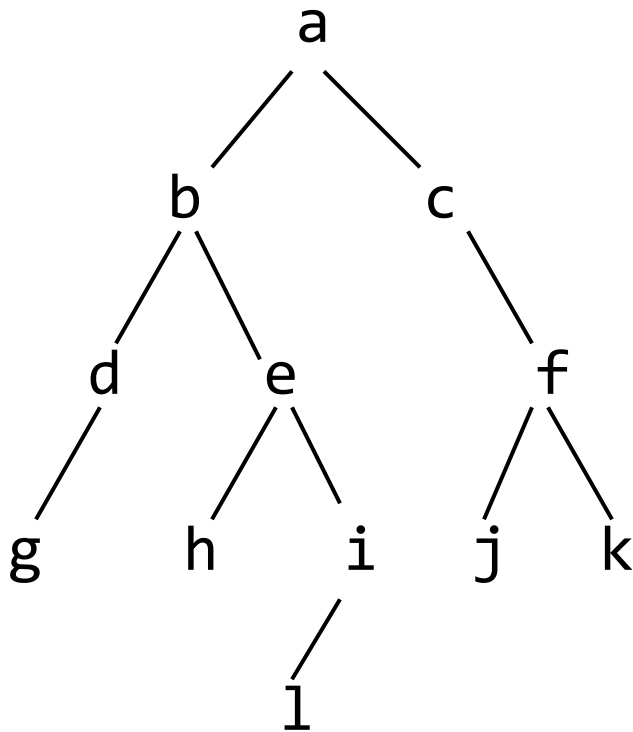


Height 7

Binary trees with the same nodes but different heights

# Depth of a Tree

- The path length from the root of the tree to this node.



The **depth** of a node is its distance from the root

**a** is at depth zero

**e** is at depth 2

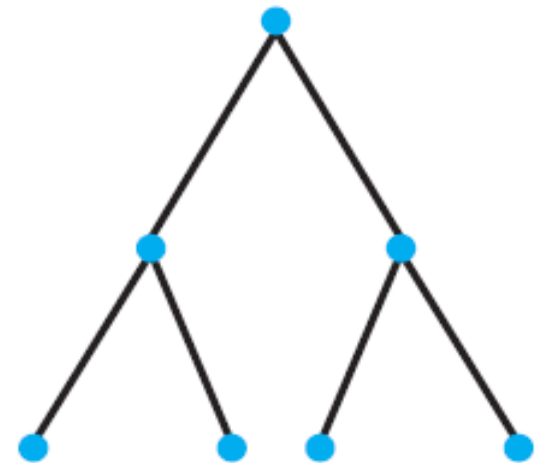
The **depth** of a binary tree is the depth of its deepest node

This tree has depth 4

# Full, Complete, and Balanced Binary Trees

# Full Binary Trees

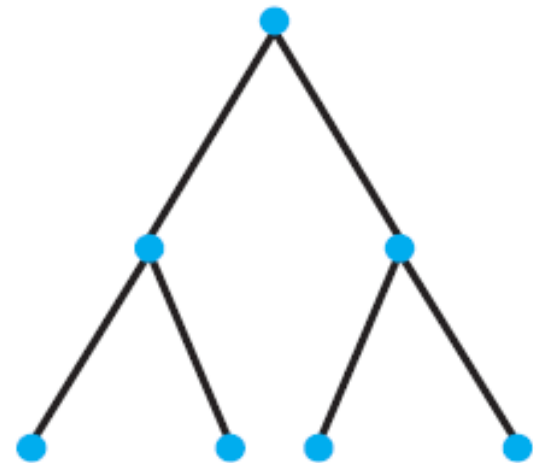
- Definition of a full binary tree
  - If  $T$  is empty,  $T$  is a full binary tree of height 0.
  - If  $T$  is not empty and has height  $h > 0$ ,  $T$  is a full binary tree if its root's subtrees are both full binary trees of height  $h - 1$ .
  - every node other than the leaves has two children.



# Facts about Full Binary Trees

- You cannot add nodes to a full binary tree without increasing its height.
- The number of nodes that a full binary tree of height  $h$  can have is  $2^h - 1$ .
- The height of a full binary tree with  $n$  nodes is

$$\lceil \log_2 (n + 1) \rceil$$



# Complete Binary Trees

Every level, except possibly the last, is completely filled, and all nodes are as far left as possible

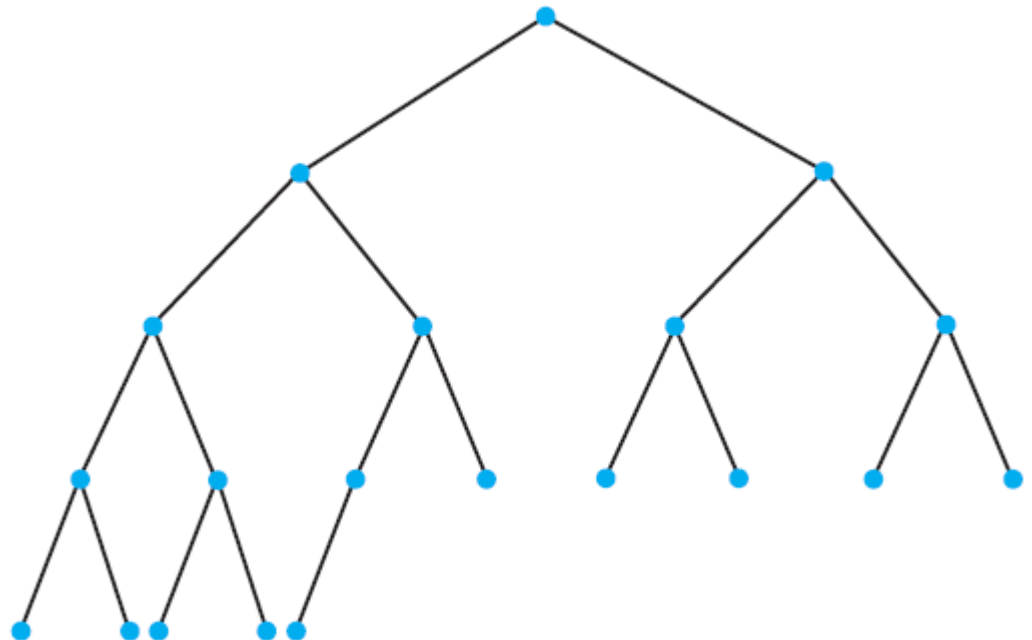
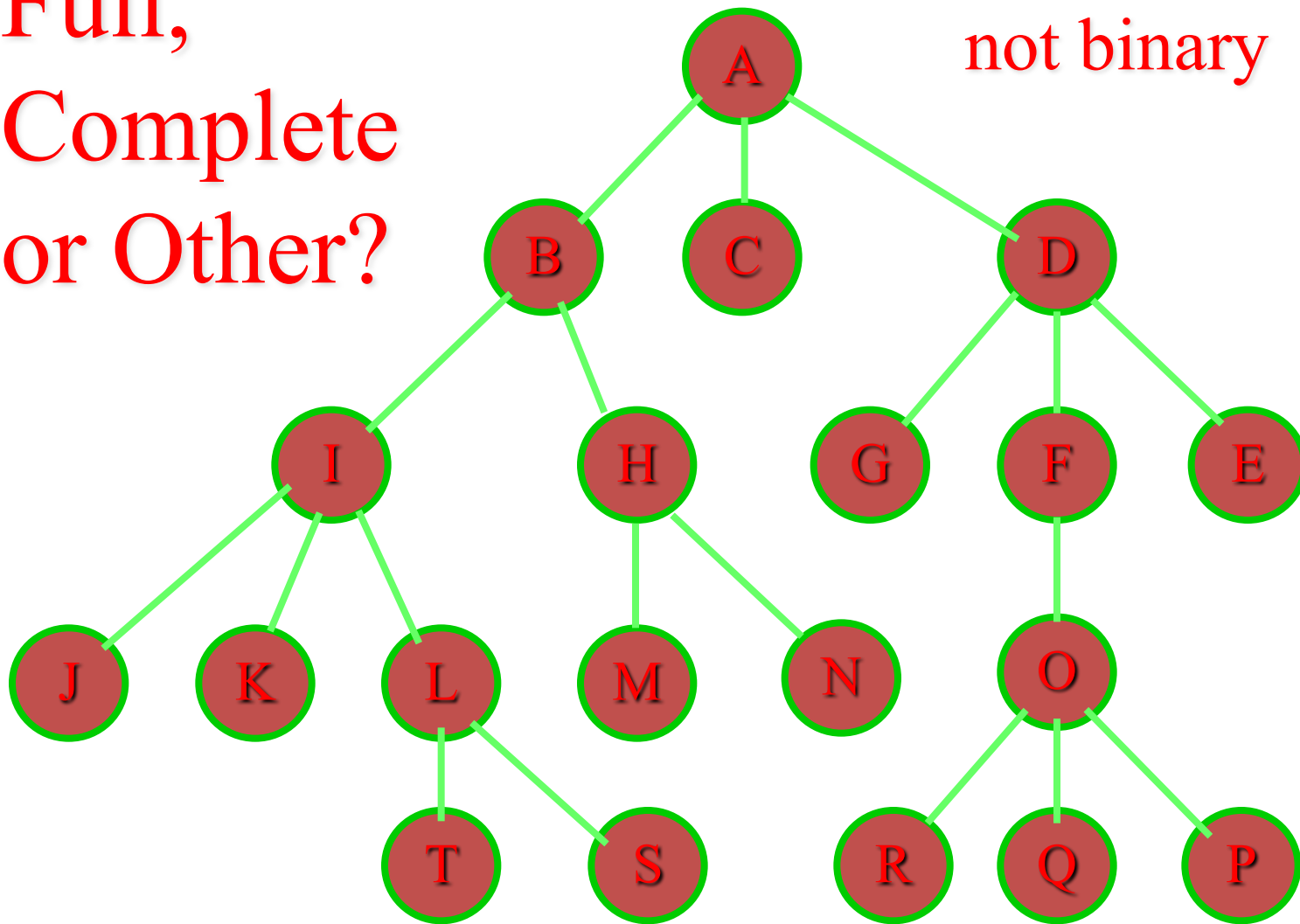


FIGURE 15-7 A complete binary tree

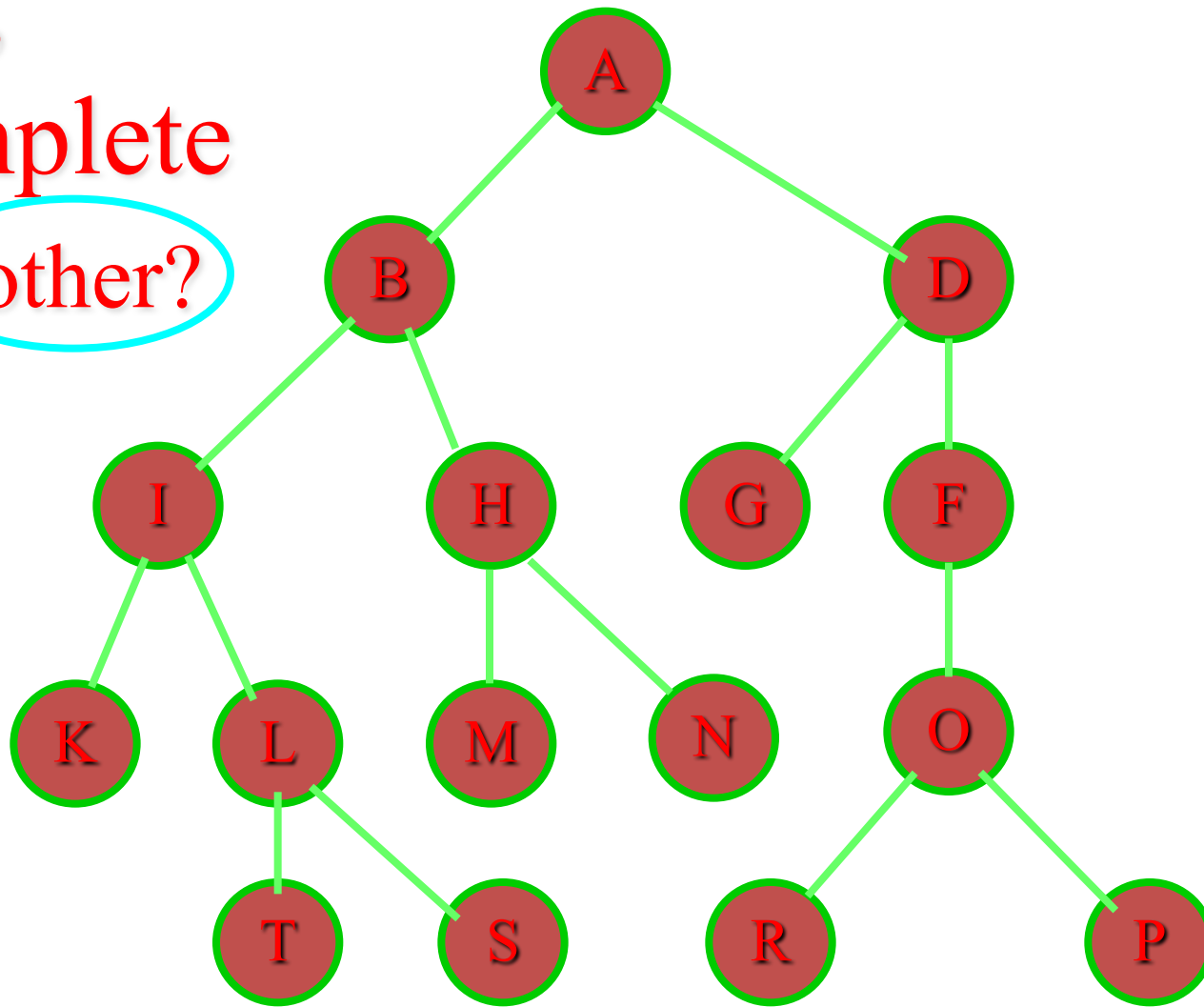
Full,  
Complete  
or Other?

not binary

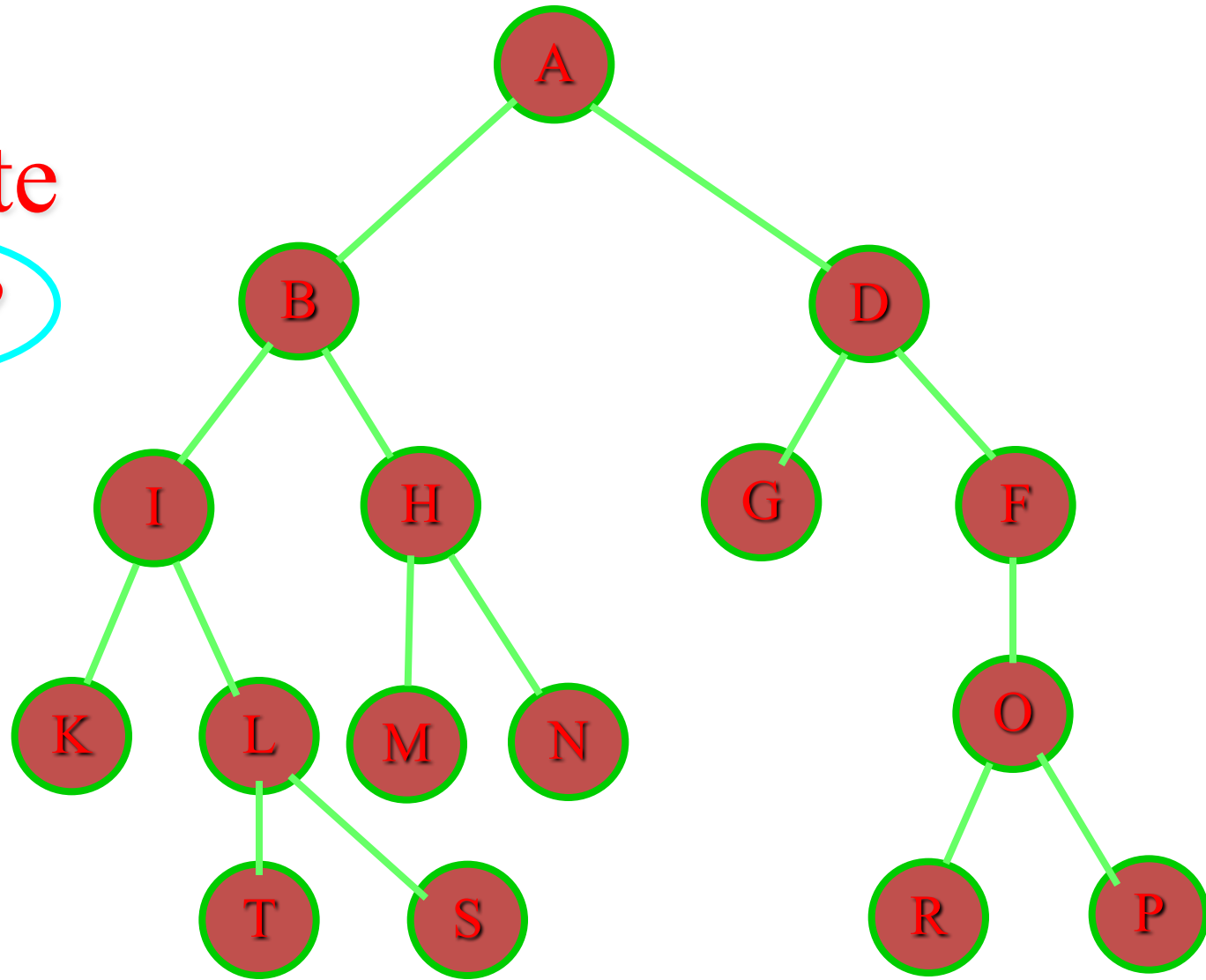




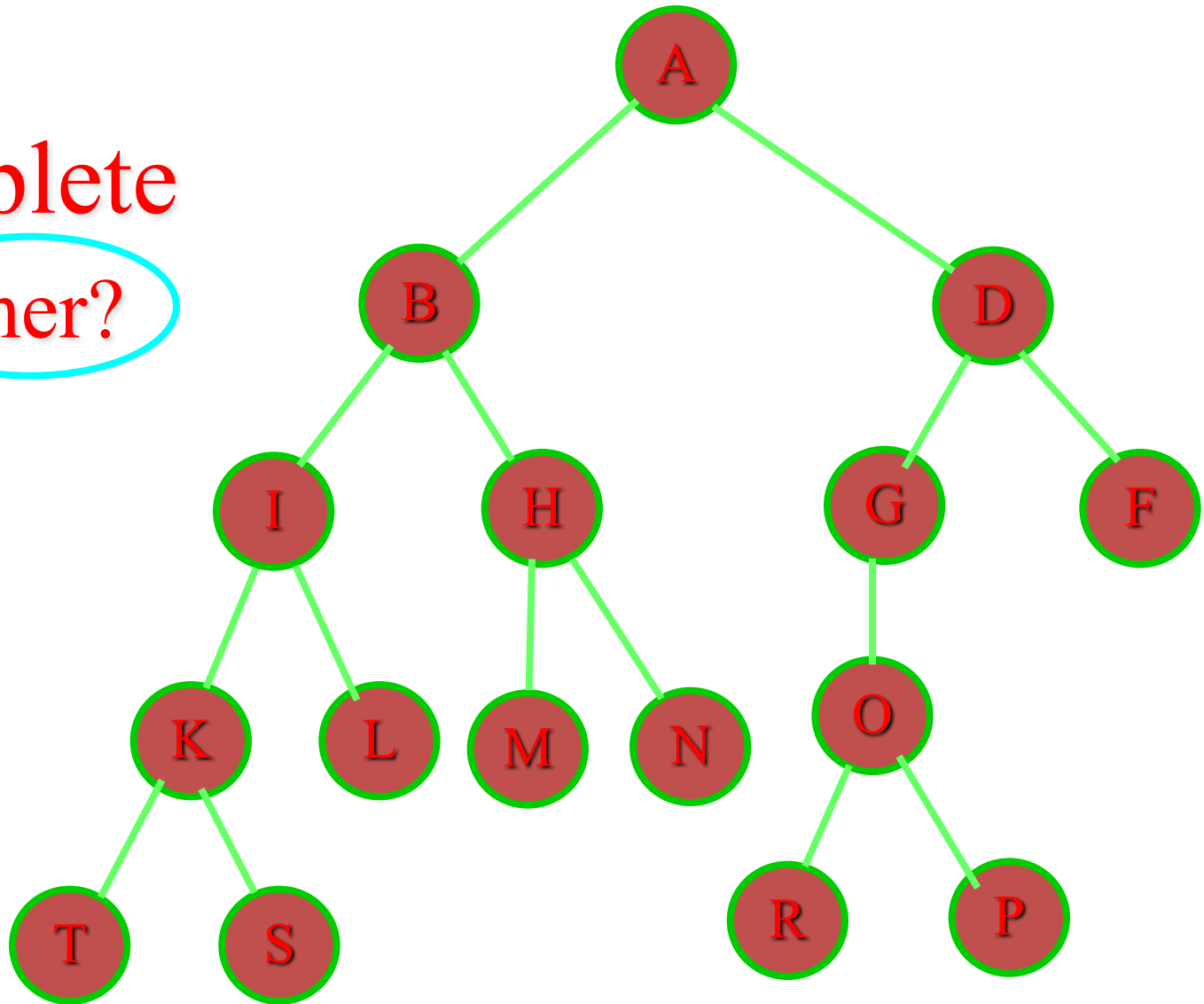
Full,  
Complete  
or other?



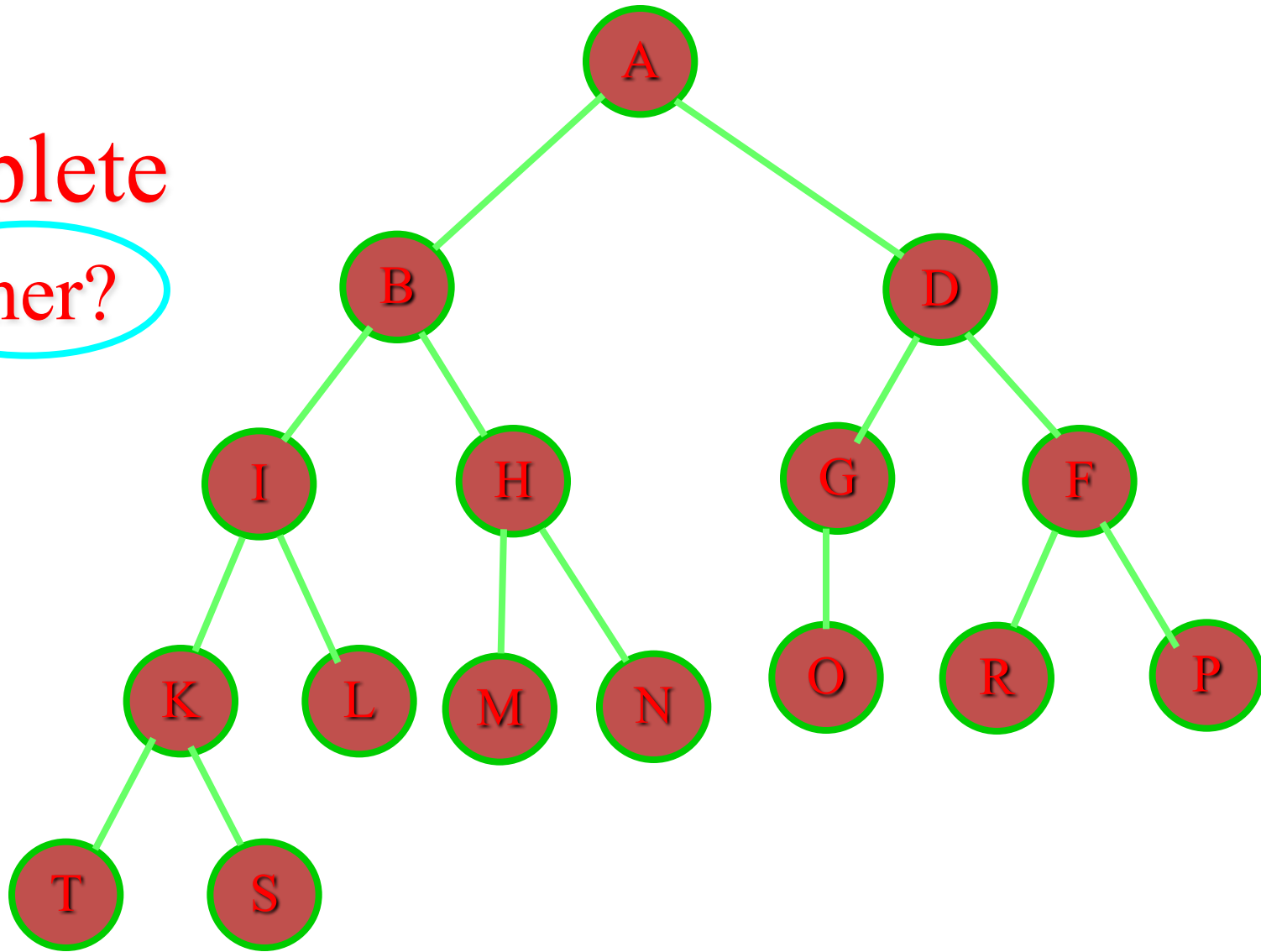
Full,  
Complete  
or other?



Full,  
Complete  
or other?

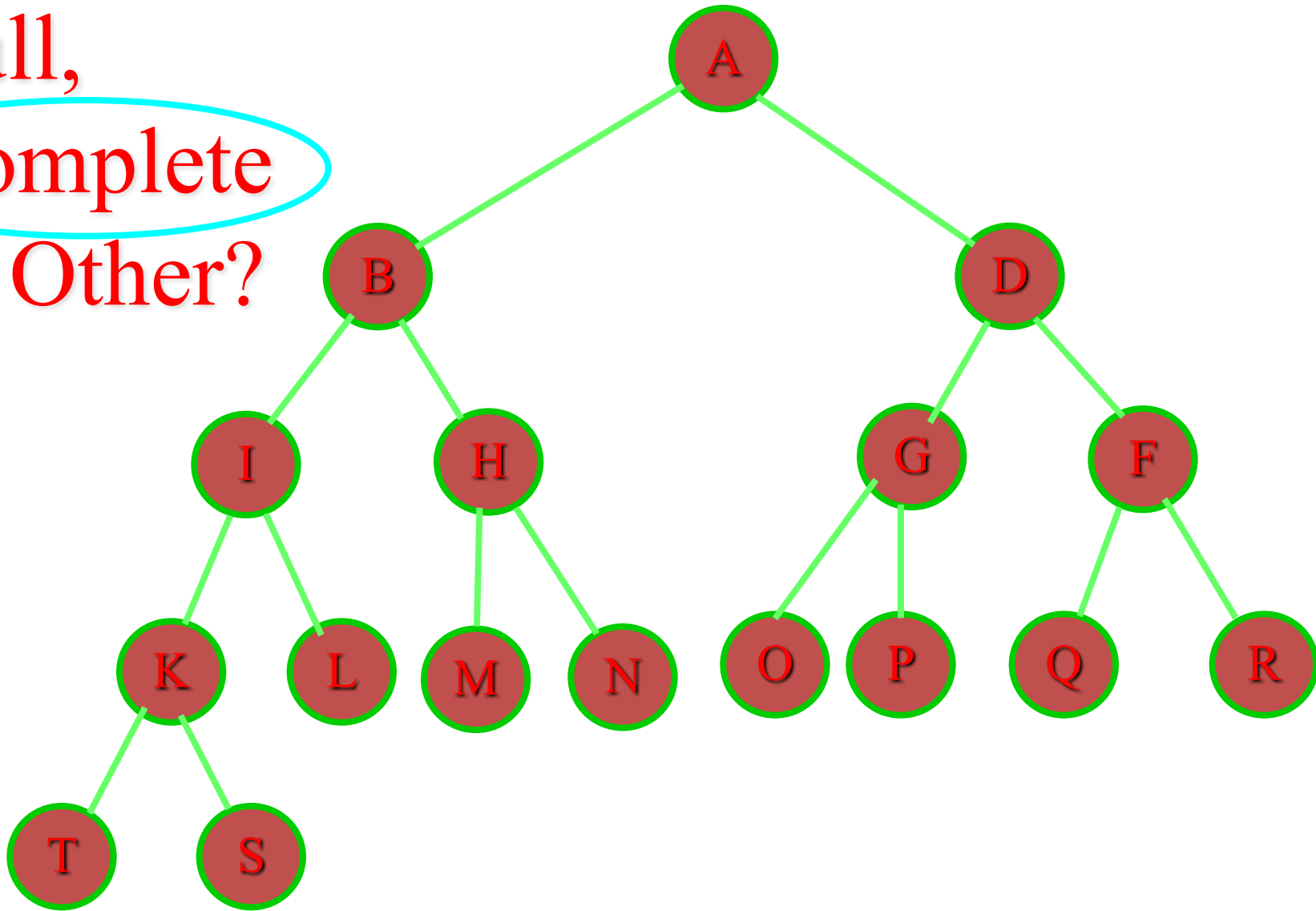


Full,  
Complete  
or other?

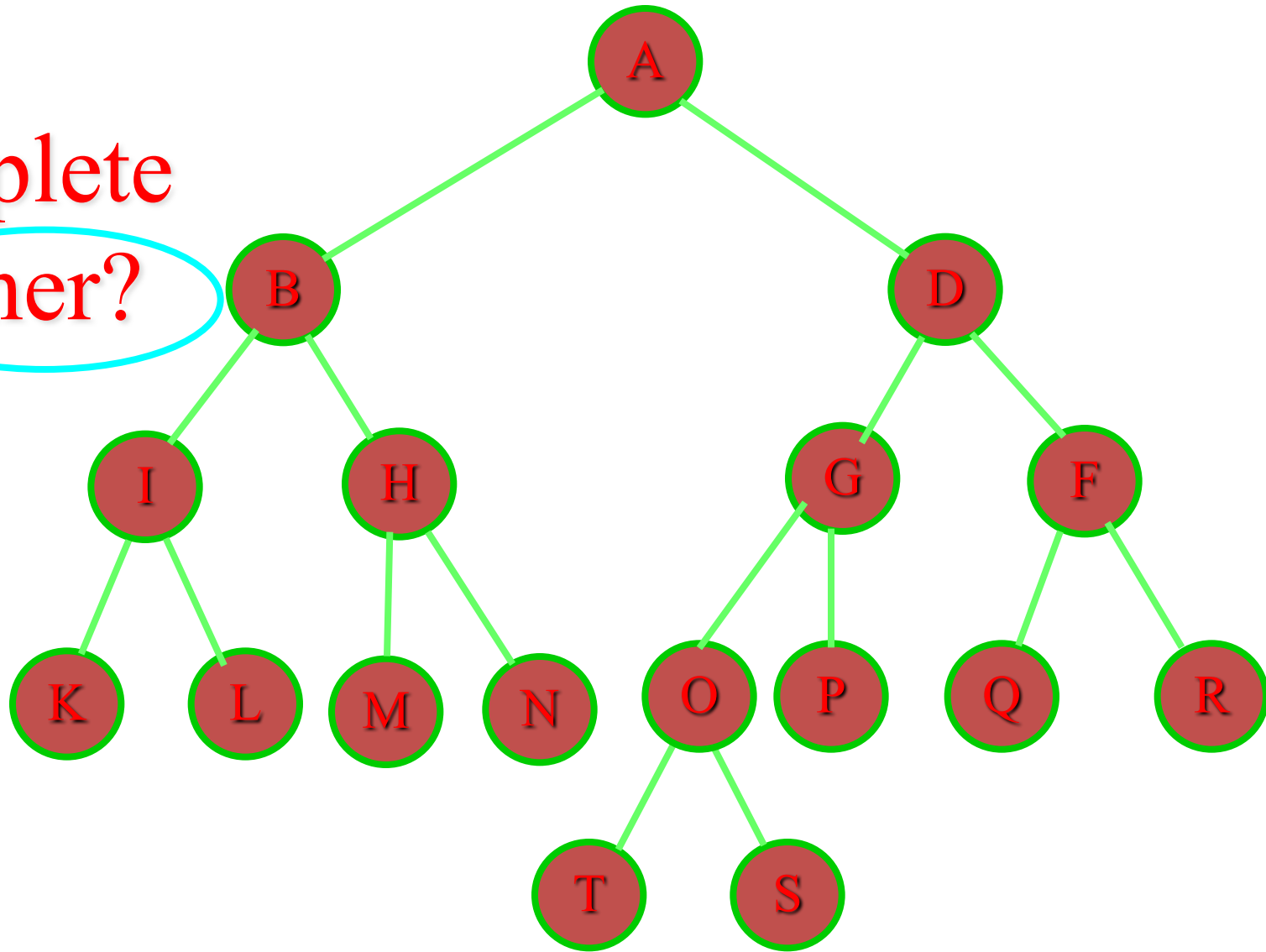


Full,

Complete  
or Other?

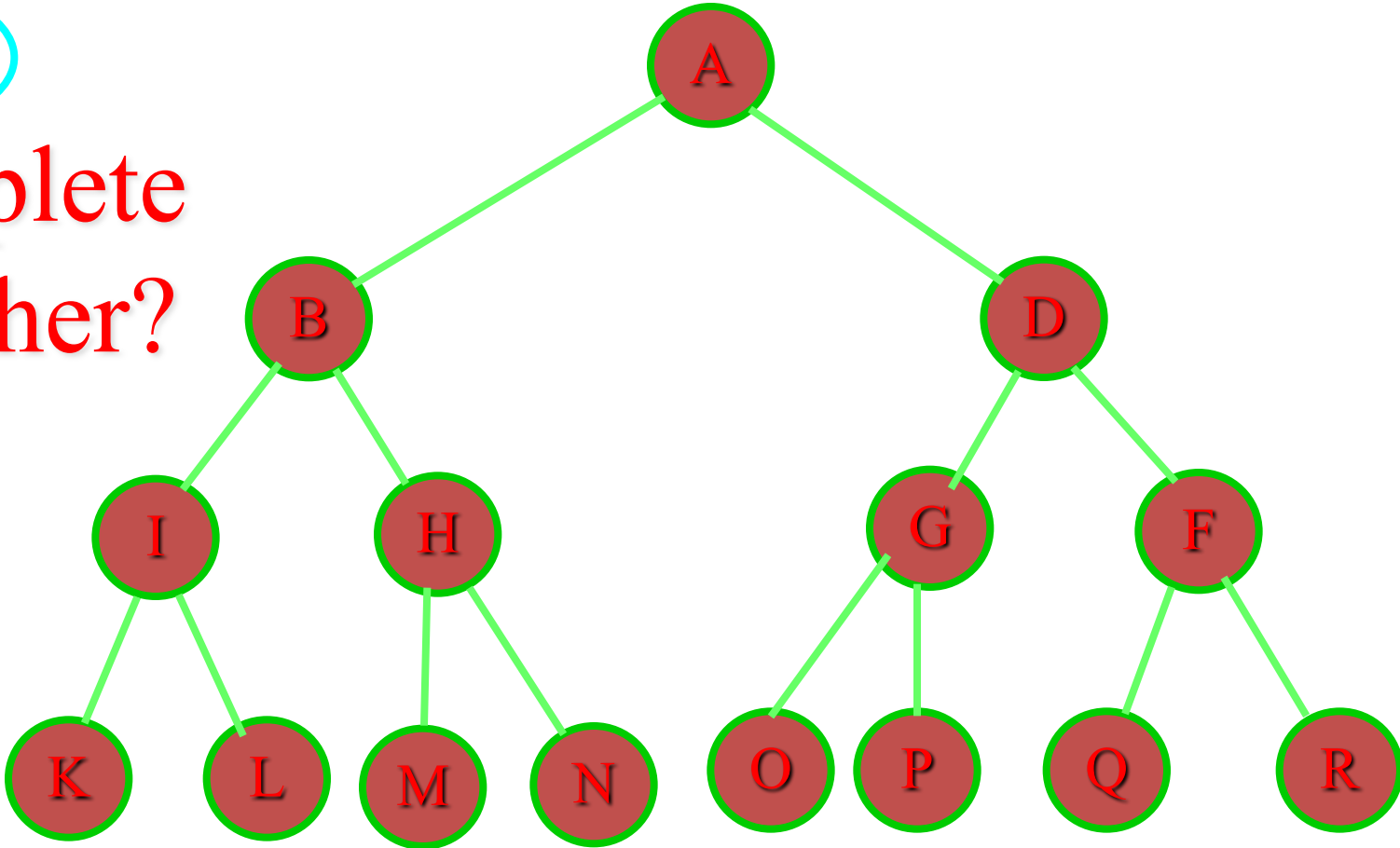


Full,  
Complete  
or other?

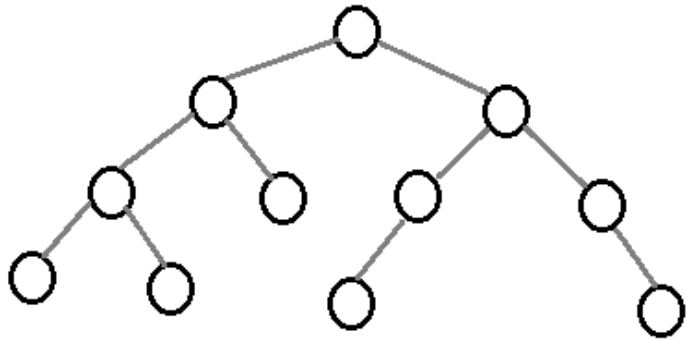


Full,

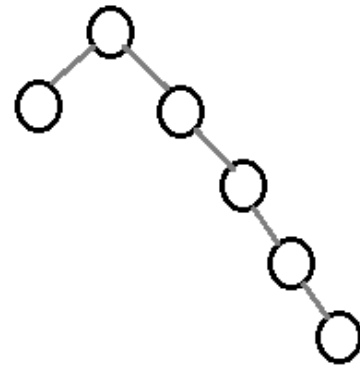
Complete  
or Other?



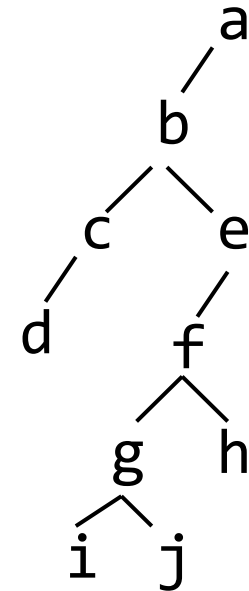
- A balanced binary tree has the minimum possible height for the leaves



Balanced Binary Tree



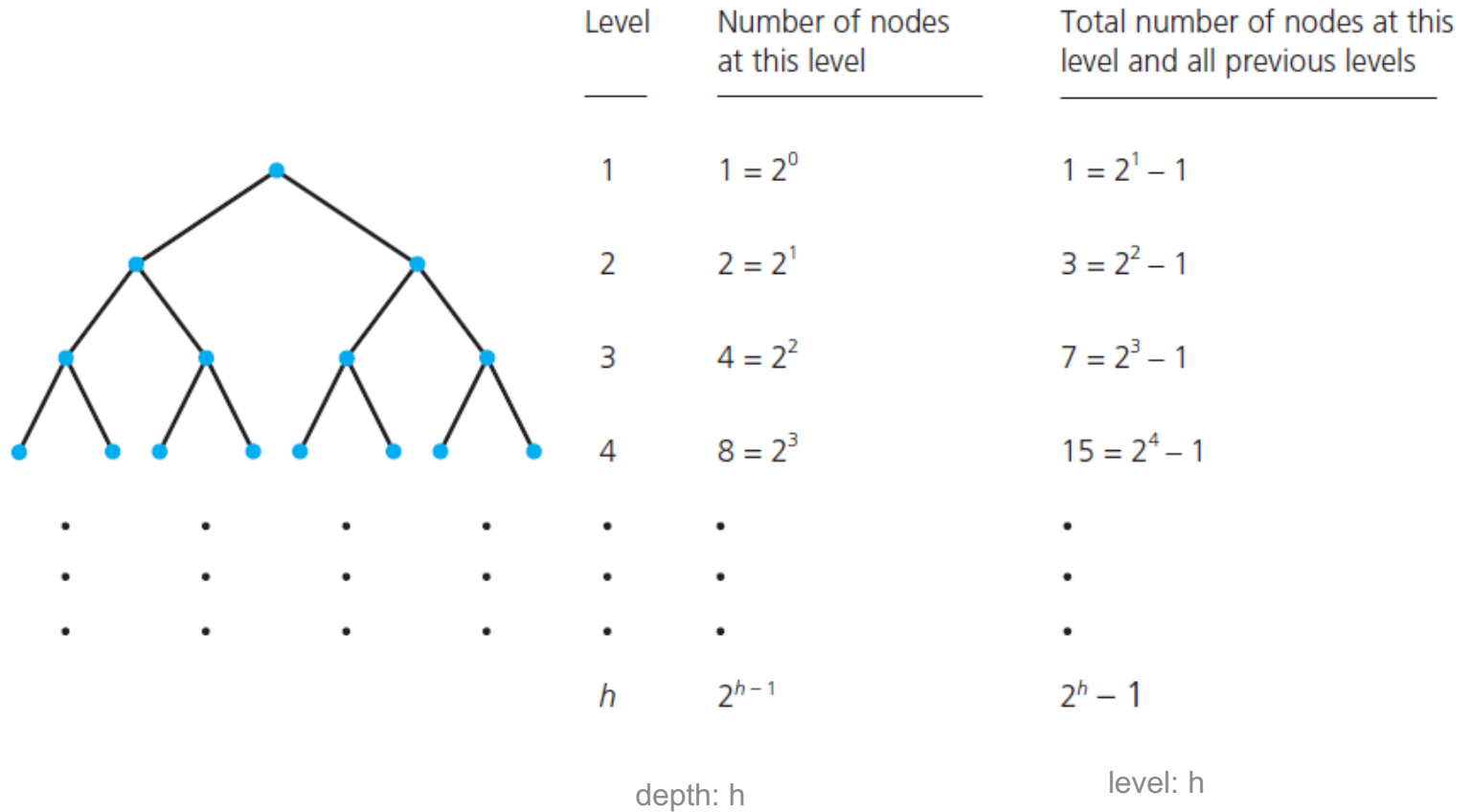
Unbalanced Binary Tree



An unbalanced binary tree



# Number of Nodes in a Binary Tree



# Traversals of a Binary Tree

- General form of recursive traversal algorithm

## 1. Preorder Traversal

Each node is processed before any node in either of its subtrees

## 2. Inorder Traversal

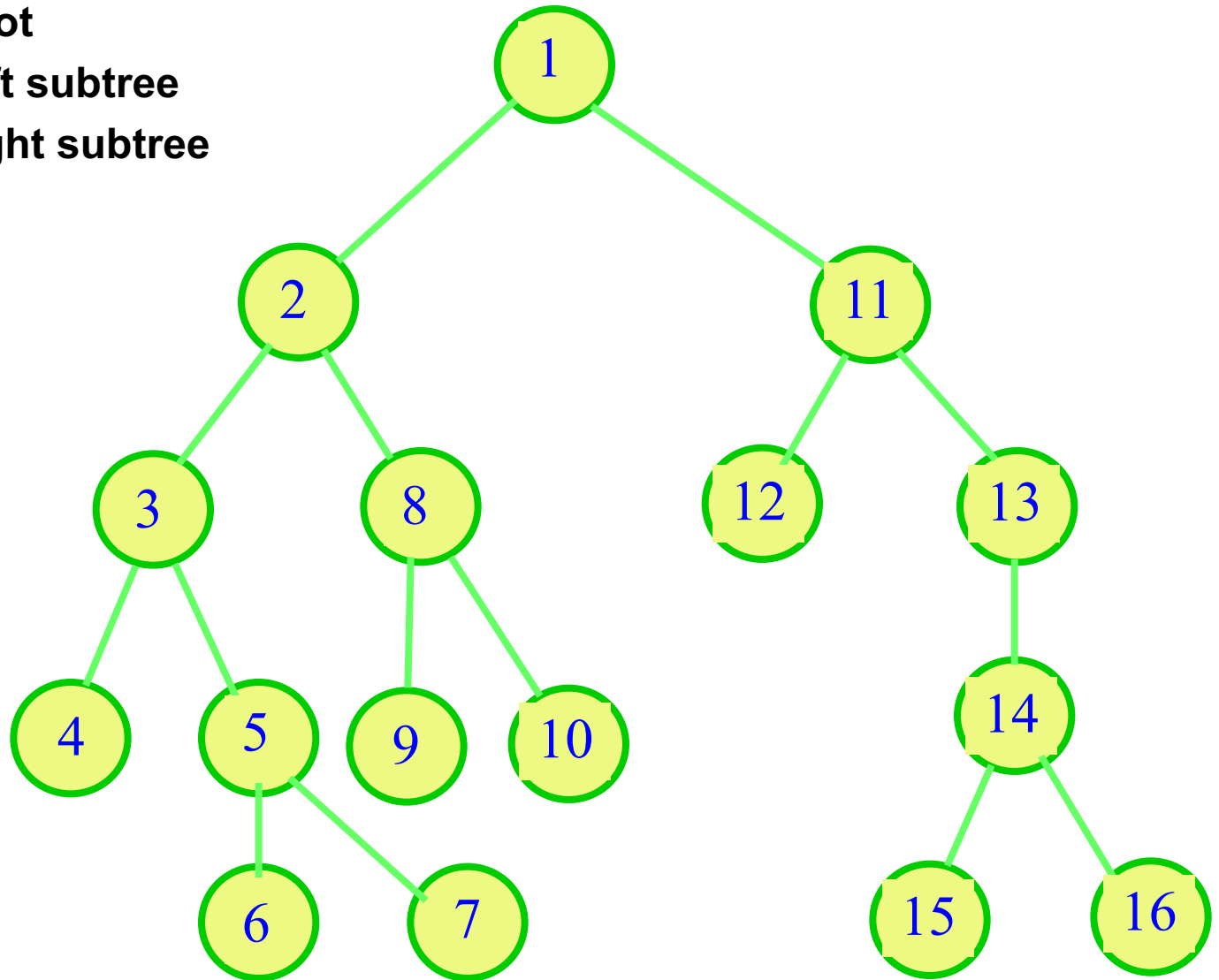
Each node is processed after all nodes in its left subtree and before any node in its right subtree

## 3. Postorder Traversal

Each node is processed after all nodes in both of its subtrees

# Preorder Traversal

1. Visit the root
2. Visit the left subtree
3. Visit the right subtree



**Algorithm TraversePreorder(n)**

**Process node n**

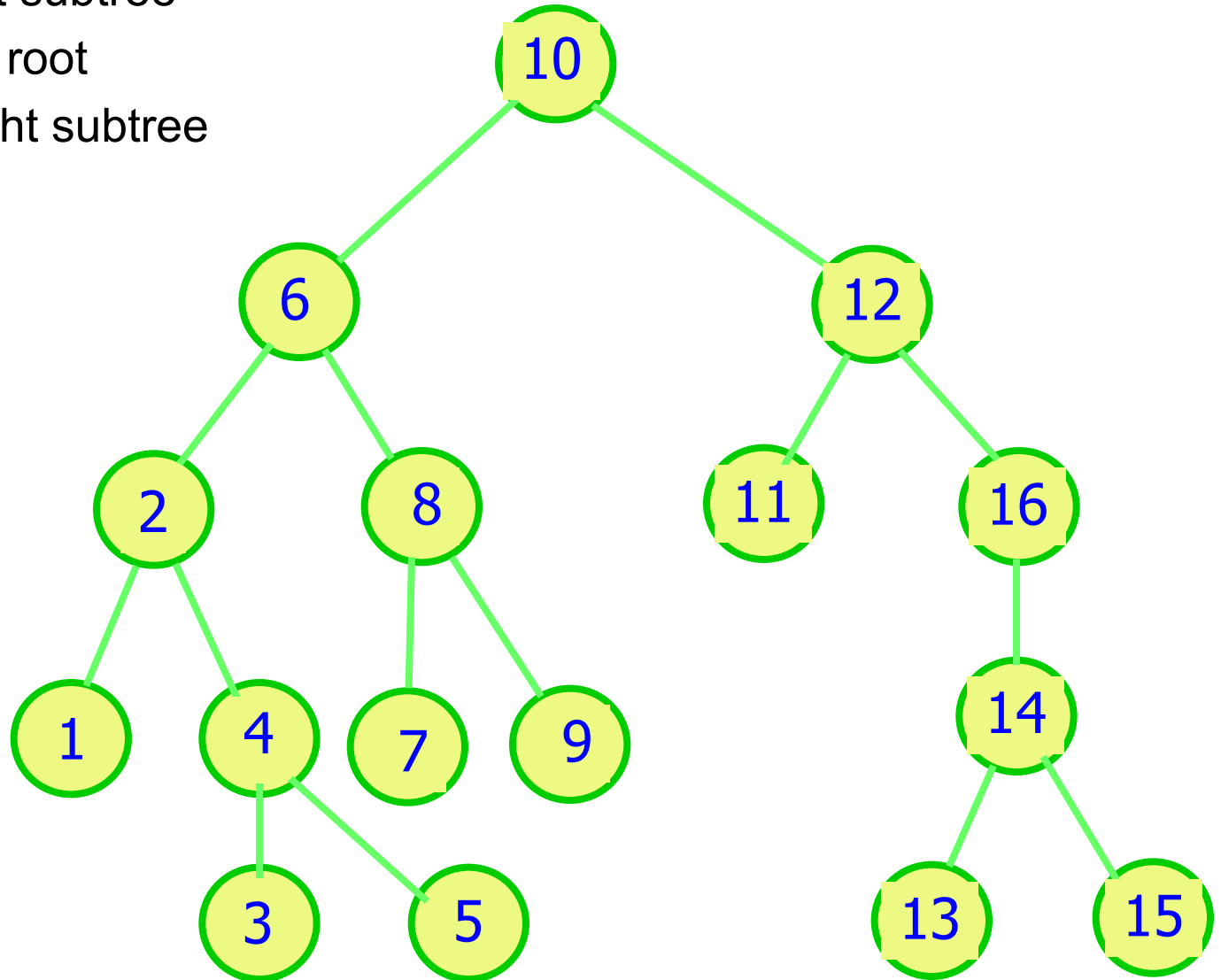
**if n is an internal node then**

**TraversePreorder( n -> leftChild)**

**TraversePreorder( n -> rightChild)**

# Inorder Traversal

1. Visit Left subtree
2. Visit the root
3. Visit Right subtree



**Algorithm TraverseInorder(n)**

**if n is an internal node then**

**TraverseInorder( n -> leftChild)**

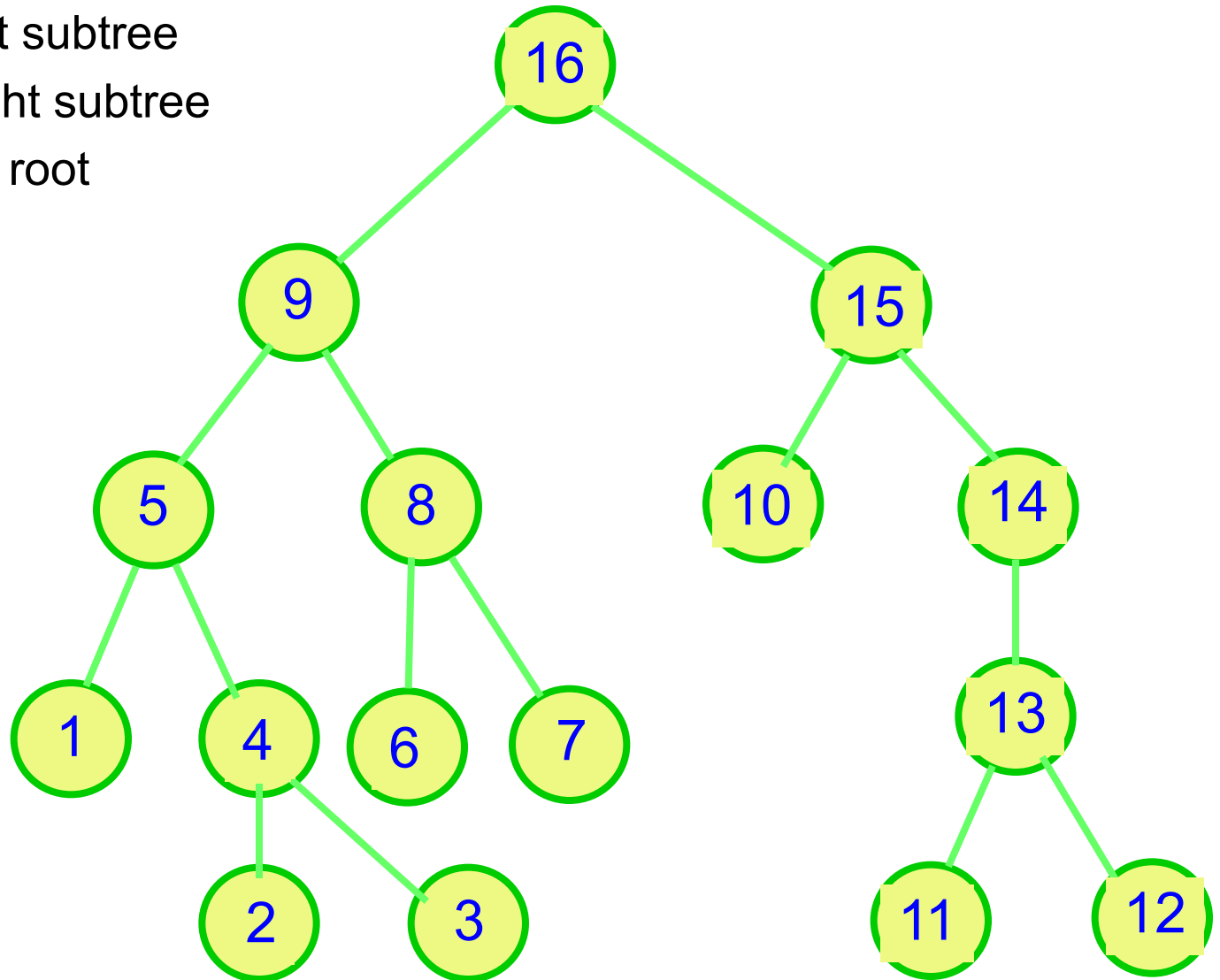
**Process node n**

**if n is an internal node then**

**TraverseInorder( n -> rightChild)**

# Postorder Traversals

1. Visit Left subtree
2. Visit Right subtree
3. Visit the root



```
Algorithm TraversePostorder(n)
```

```
  if n is an internal node then
```

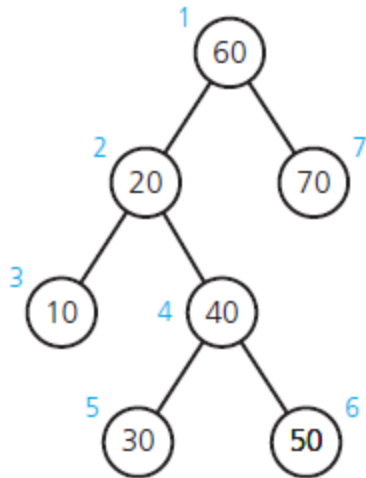
```
    TraversePostorder( n -> leftChild)
```

```
    TraversePostorder( n -> rightChild)
```

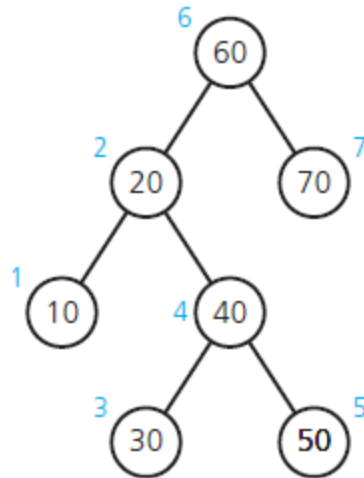
```
  Process node n
```



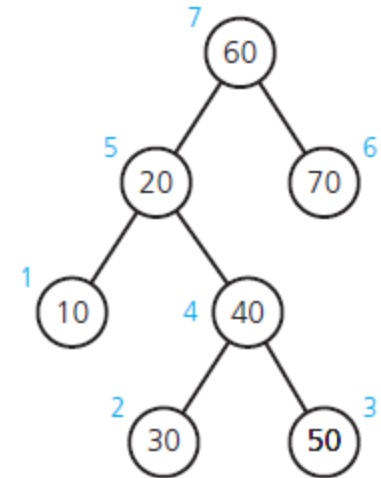
# Traversals of a Binary Tree



(a) Preorder: 60, 20, 10, 40, 30, 50, 70



(b) Inorder: 10, 20, 30, 40, 50, 60, 70



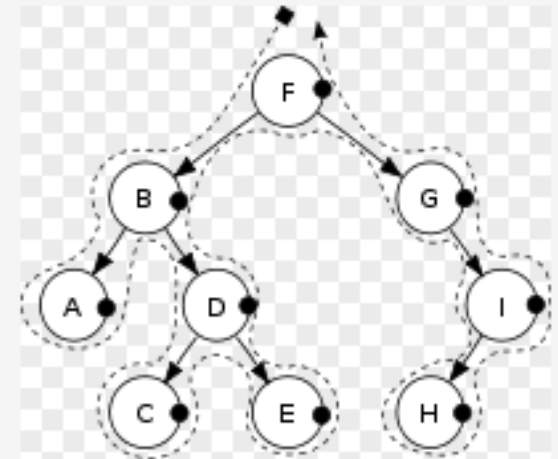
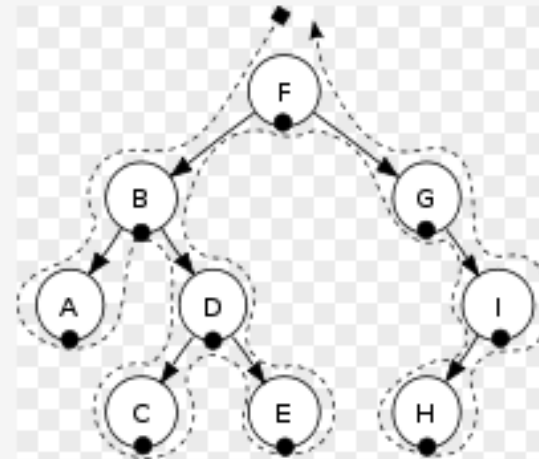
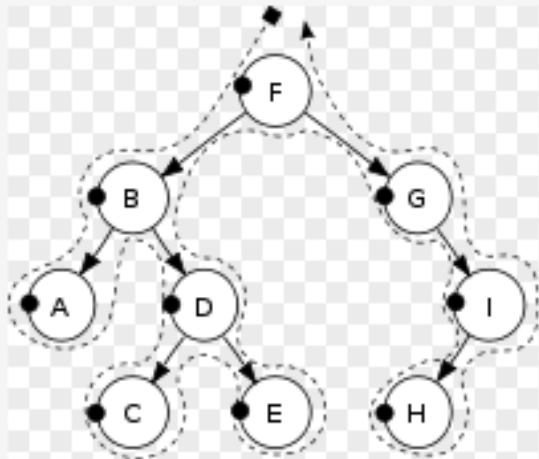
(c) Postorder: 10, 30, 50, 40, 20, 70, 60

(Numbers beside nodes indicate traversal order.)

FIGURE 15-11 Three traversals of a

binary tree

### The 3 different types of traversal



Pre-order Traversal  
FBADCEGIH

In-order Traversal  
ABCDEFGHI

Post-order Traversal  
ACEDBHIGF

# Binary Tree Operations

- Test whether a binary tree is empty.
- Get the height of a binary tree.
- Get the number of nodes in a binary tree.
- Get the data in a binary tree's root.
- Set the data in a binary tree's root.
- Add a new node containing a given data item to a binary tree.

# Binary Tree Operations

- Remove the node containing a given data item from a binary tree.
- Remove all nodes from a binary tree.
- Retrieve a specific entry in a binary tree.
- Test whether a binary tree contains a specific entry.
- Traverse the nodes in a binary tree in preorder, inorder, or postorder.

# Representation of Binary Tree ADT

A binary tree can be represented using

- **Linked List**
- **Array**

**Note** : Array is suitable only for full and complete binary trees

```
struct node
{
    int key_value;
    struct node *left;
    struct node *right;
};
```

```
struct node *root = 0;
```

```
void inorder(node *p)
{
    if (p != NULL)
    {
        inorder(p->left);
        printf(p->key_value);
        inorder(p->right);
    }
}
```

```
void preorder(node *p)
{
    if (p != NULL)
    {
        printf(p->key_value);
        preorder(p->left);
        preorder(p->right);
    }
}
```

```
void postorder(node *p)
{
    if (p != NULL)
    {
        postorder(p->left);
        postorder(p->right);
        printf(p->key_value);
    }
}
```

```
void destroy_tree(struct node *leaf)
{
    if( leaf != 0 )
    {
        destroy_tree(leaf->left);
        destroy_tree(leaf->right);
        free( leaf );
    }
}
```



# The Binary Search Tree

- Binary tree is ill suited for searching a specific item
- Binary *search* tree solves the problem
- Properties of each node,  $n$ 
  - $n$ 's value is **greater** than all values in the left subtree  $T_L$
  - $n$ 's value is **less** than all values in the right subtree  $T_R$
  - Both  $T_R$  and  $T_L$  are binary search trees.

# The Binary Search Tree

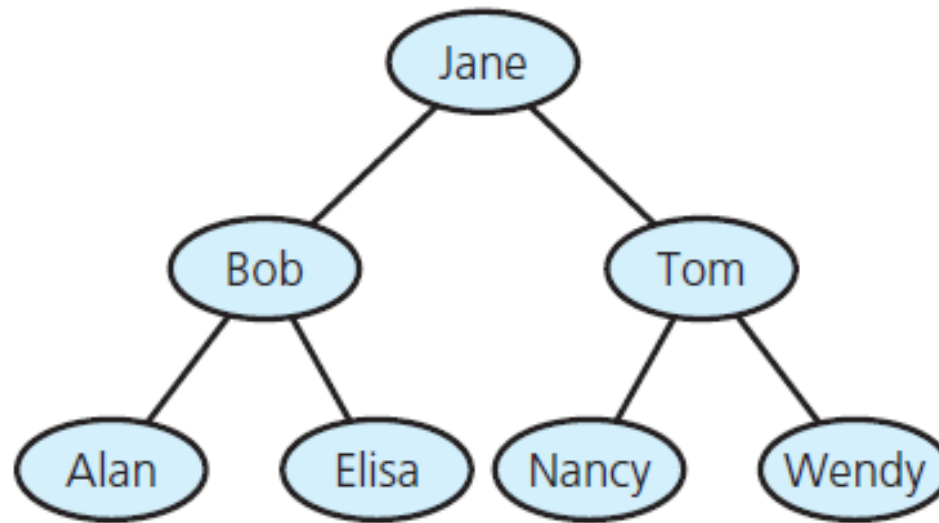


FIGURE 15-13 A binary search tree of

names

# The Binary Search Tree

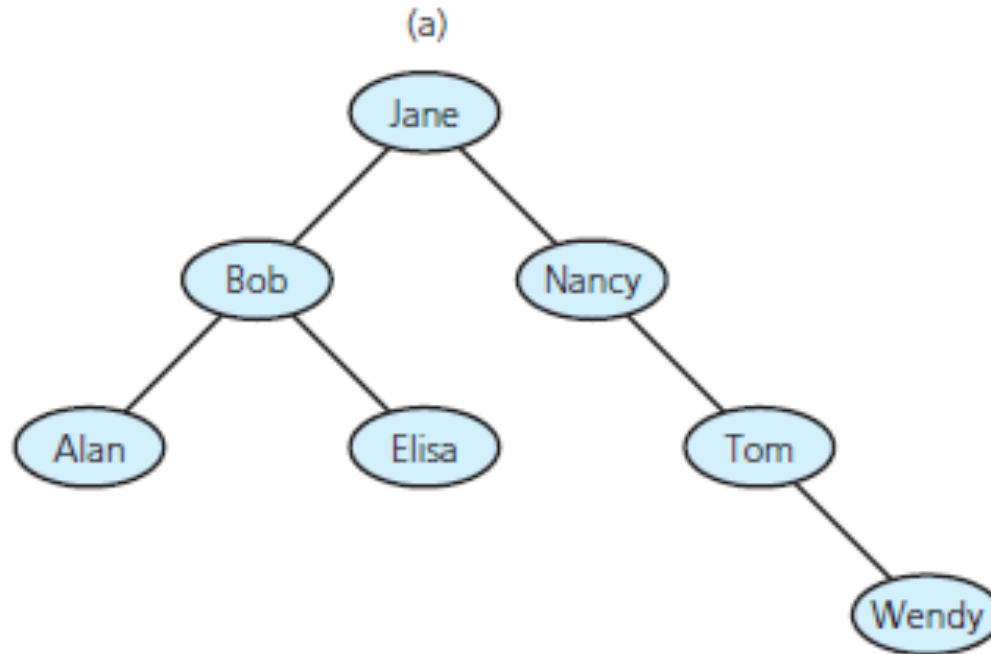


FIGURE 15-14 Binary search trees with the same data as in Figure 15-13

# The Binary Search Tree

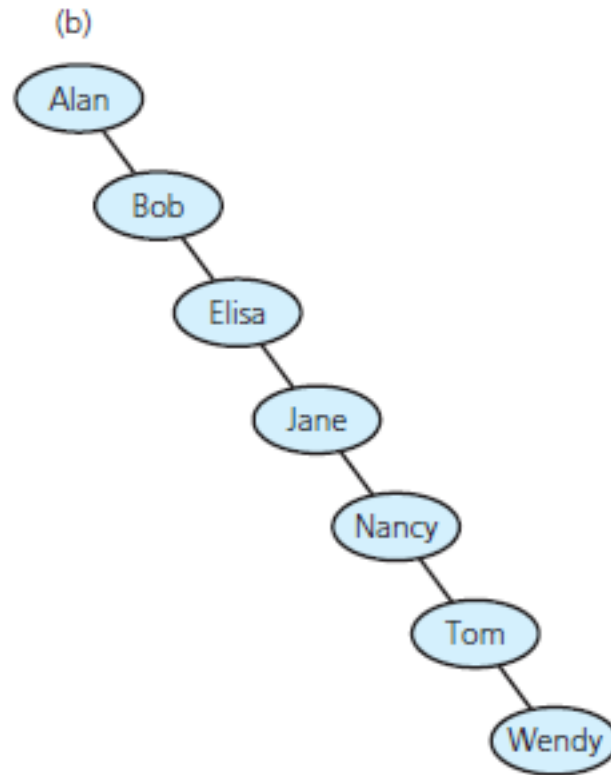
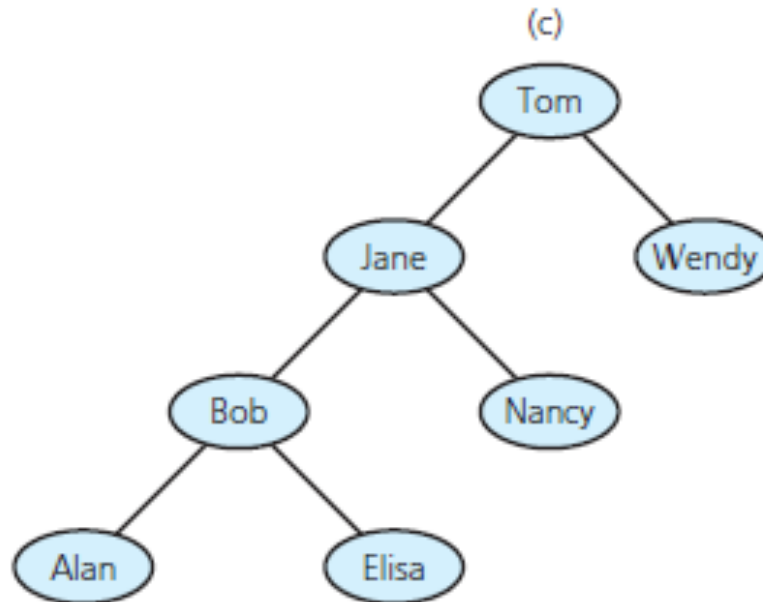


FIGURE 15-14 Binary search trees with the same data as in Figure 15-13

# The Binary Search Tree



Binary search trees with the same data as in Figure 15-13

# Binary Search Tree Operations

- Test whether a binary search tree is empty.
- Get height of a binary search tree.
- Get number of nodes in a binary search tree.
- Get data in binary search tree's root.
- Insert new item into the binary search tree.
- Remove given item from the binary search tree.

# Binary Search Tree Operations

- Remove all entries from a binary search tree.
- Retrieve given item from a binary search tree.
- Test whether a binary search tree contains a specific entry.
- Traverse items in a binary search tree in
  - Preorder
  - Inorder
  - Postorder.

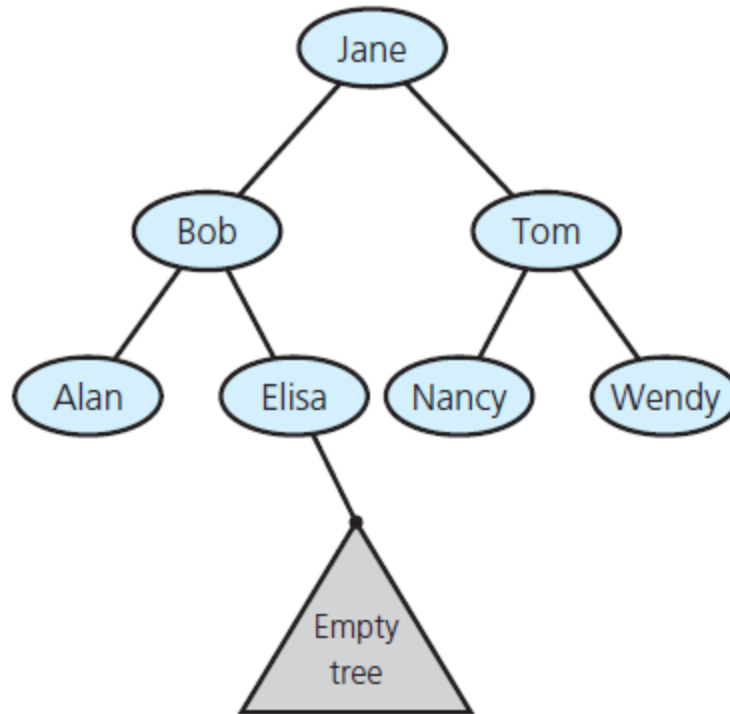
# Searching a Binary Search Tree

- Search algorithm for binary search tree



```
struct node *search(int key, struct node *leaf)
{
    if( leaf != 0 )
    {
        if(key==leaf->key_value)
        {
            return leaf;
        }
        else if(key<leaf->key_value)
        {
            return search(key, leaf->left);
        }
        else
        {
            return search(key, leaf->right);
        }
    }
    else return 0;
}
```

# Creating a Binary Search Tree



Empty subtree where the `search` algorithm terminates when looking for Frank

# Efficiency of Binary Search Tree Operations

<u>Operation</u>	<u>Average case</u>	<u>Worst case</u>
Retrieval	$O(\log n)$	$O(n)$
Insertion	$O(\log n)$	$O(n)$
Removal	$O(\log n)$	$O(n)$
Traversal	$O(n)$	$O(n)$

The Big O for the retrieval, insertion, removal, and traversal operations of the ADT binary search tree