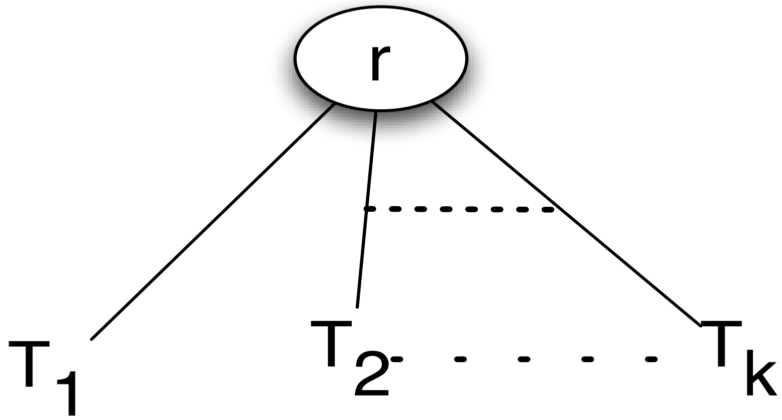
Trees

What is a Tree?

- T is a **tree** if either
  - T has no nodes, or
  - T is of the form:



where r is a node and T_1, T_2, ..., T_k are trees.

Tree Terminology

Parent – The parent of node n is the node directly above in the tree.

Child – The child of node n is the node directly below in the tree.

- If node m is the parent of node n, node n is the child of node m.

Root – The only node in the tree with no parent.

Leaf – A node with no children.

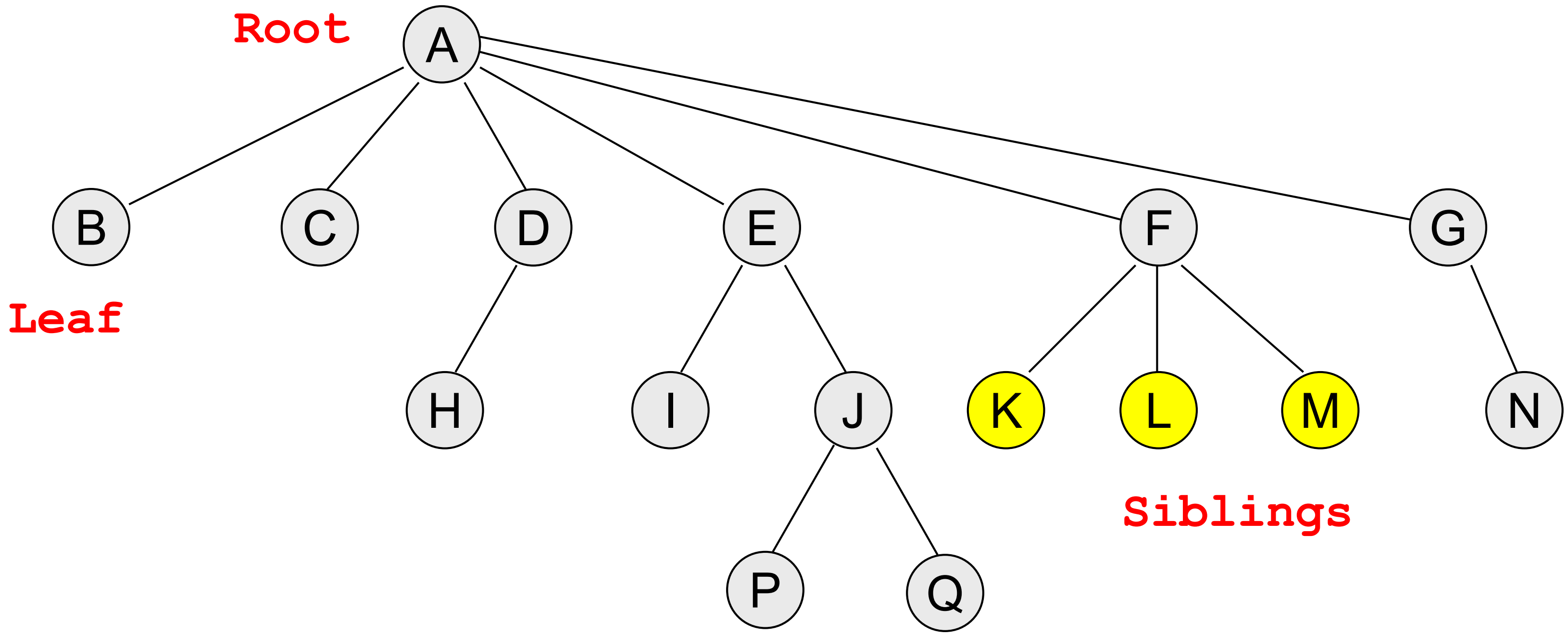
Siblings – Nodes with a common parent.

Ancestor – An ancestor of node n is a node on the path from the root to n.

Descendant – A descendant of node n is a node on the path from n to a leaf.

Subtree – A subtree of node n is a tree that consists of a child (if any) of n and the child's descendants (a tree which is rooted by a child of node n)

A Tree – Example



–Node *A* has 6 ***children***: B, C, D, E, F, G.

–B, C, H, I, P, Q, K, L, M, N are ***leaves*** in the tree above.

–K, L, M are ***siblings*** since F is parent of all of them.

What is a Tree?

- The root of each sub-tree is said to be ***child*** of r, and
  r is the ***parent*** of each sub-tree's root.

- If a tree is a collection of N nodes, then it has N-1 edges. **Why?**

- A ***path*** from node n_1 to n_k is defined as a sequence of nodes $n_1, n_2, \ldots, n_k$ such that n_i is parent of n_{i+1} $(1 \leq i < k)$
  - There is a path from every node to itself.
  - There is exactly one path from the root to each node. **Why?**

Level of a node

Level – The level of node n is the number of nodes on the path from root to node n.

Definition: *The level of node n in a tree T*

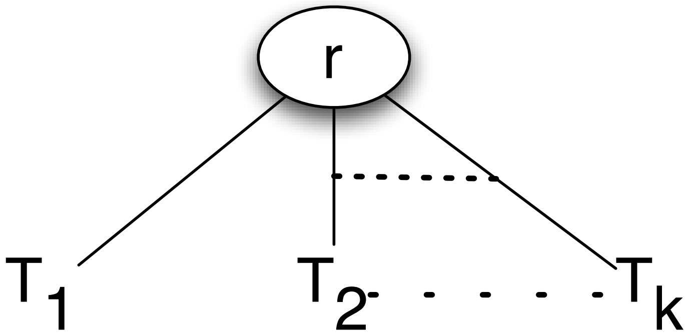
– If n is the root of T, the level of n is 1.

– If n is not the root of T, its level is 1 greater than the level of its parent.

Height of A Tree

Height – number of nodes on **longest path** from the root to any leaf.

- The height of a tree T in terms of the levels of its nodes is defined as:
  - If T is empty, its height is 0
  - If T is not empty, its height is equal to the maximum level of its nodes.
- Or, the height of a tree T can be defined as recursively as:
  - If T is empty, its height is 0.
  - If T is non-empty tree, then since T is of the form:

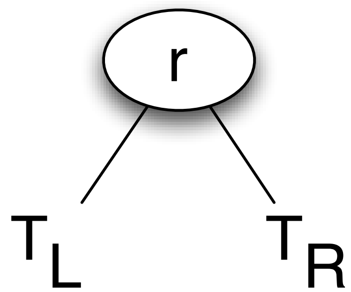


$$\textit{height}(T) = 1 + \max\{\textit{height}(T_1), \textit{height}(T_2), \ldots, \textit{height}(T_k)\}$$

Binary Tree

- *A binary tree T is a set of nodes with the following properties:*
  - *The set can be empty.*
  - *Otherwise, the set is partitioned into three disjoint subsets:*
    - *a tree consists of a distinguished node r, called **root**, and*
    - *two possibly empty sets are binary tree, called **left** and **right** **subtrees** of r.*

- T is a **binary tree** if either
  - T has no nodes, or
  - T is of the form:



where r is a node and T_L and T_R are binary trees.

Binary Tree Terminology

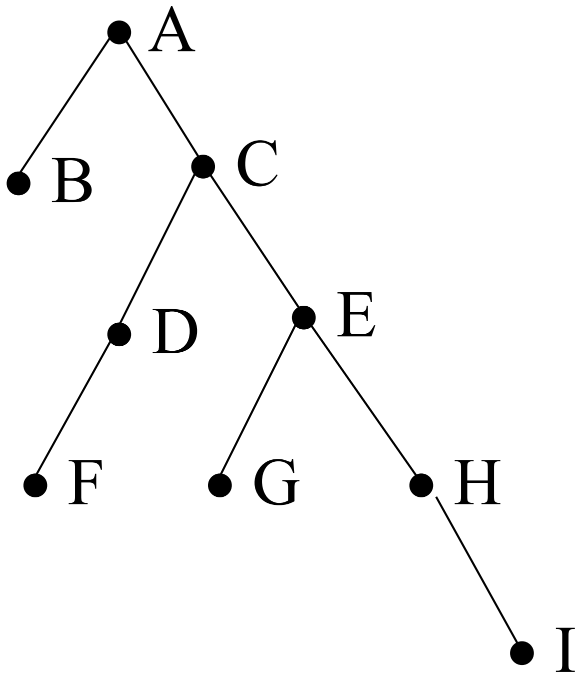
Left Child – The left child of node n is a node directly below and to the left of node n in a binary tree.

Right Child – The right child of node n is a node directly below and to the right of node n in a binary tree.

Left Subtree – In a binary tree, the left subtree of node n is the left child (if any) of node n plus its descendants.

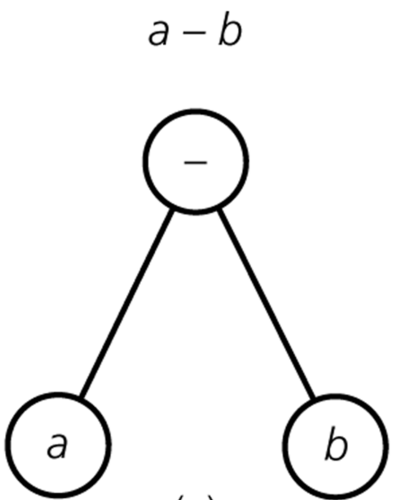
Right Subtree – In a binary tree, the right subtree of node n is the right child (if any) of node n plus its descendants.

Binary Tree -- Example

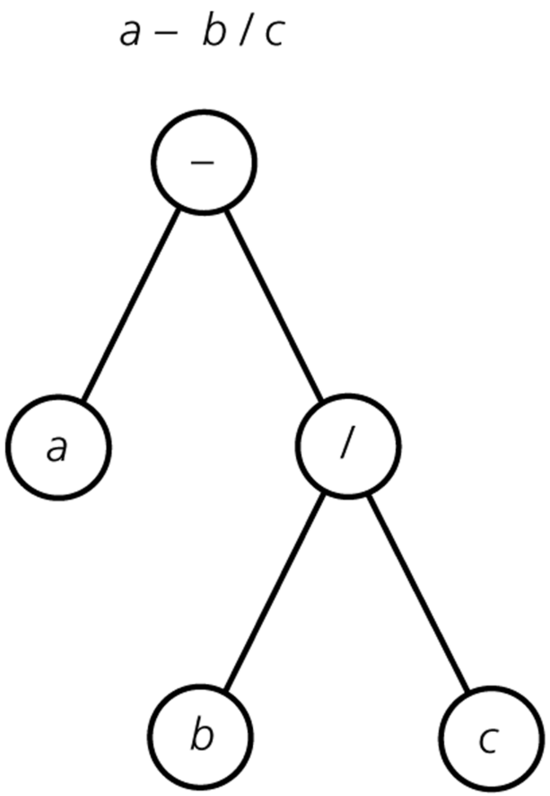


- A is the root.
- B is left child of A,
  C is right child of A.
- D doesn't have a right child.
- H doesn't have a left child.
- B, F, G and I are leaves.

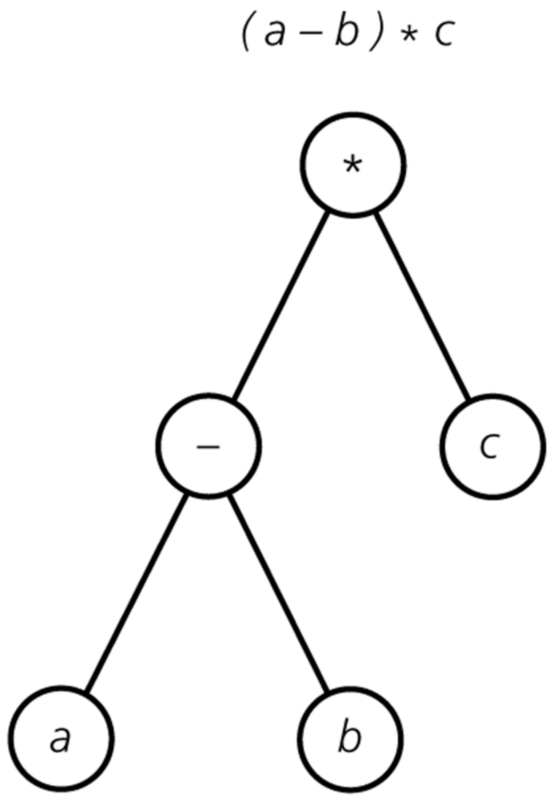
Binary Tree – Representing Algebraic Expressions



(a)



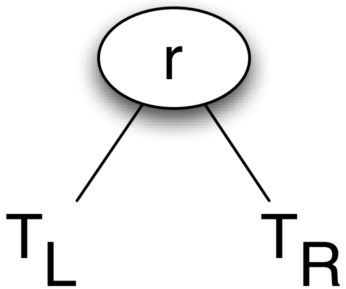
(b)



(c)

Height of Binary Tree

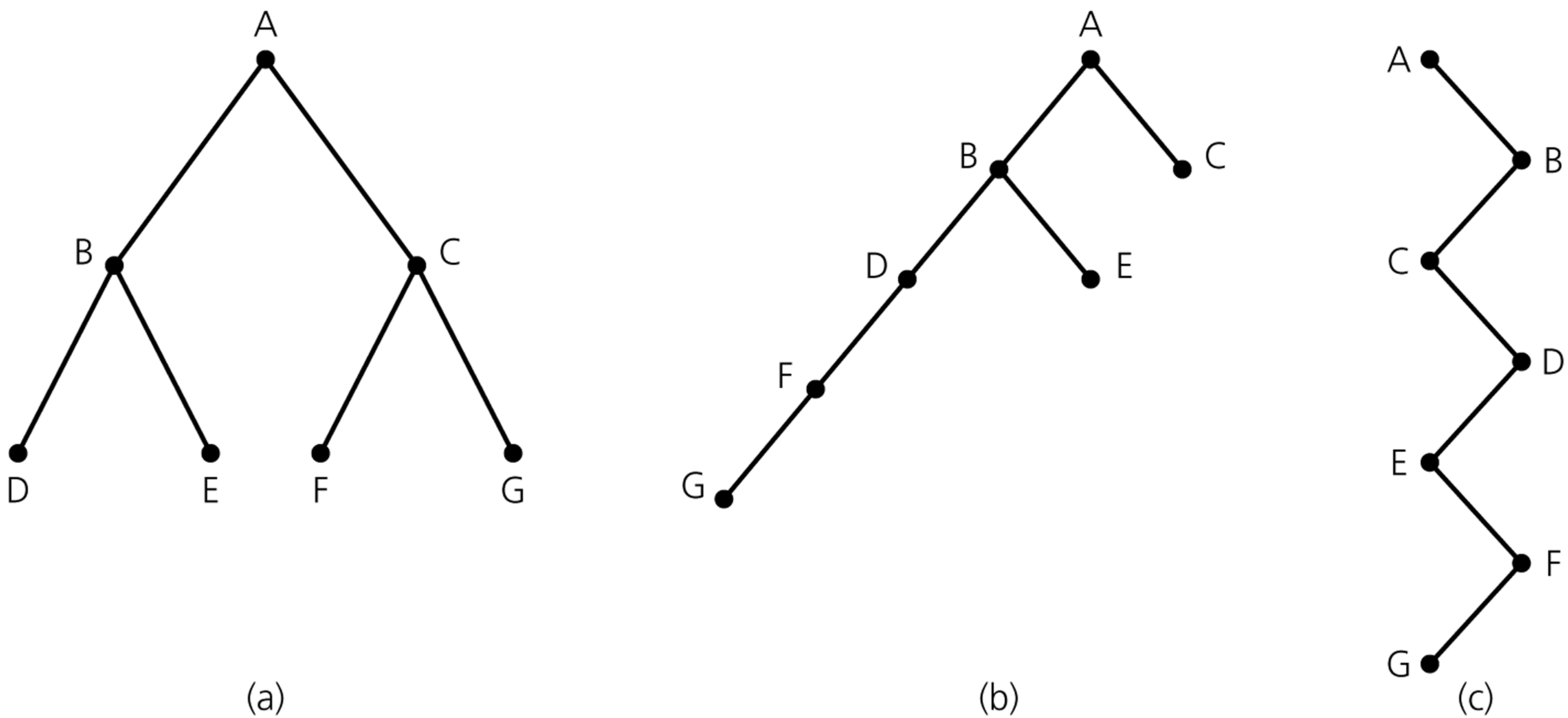
- The height of a binary tree T can be defined as recursively as:
  - If T is empty, its height is 0.
  - If T is non-empty tree, then since T is of the form ...



... height of T is 1 greater than height of its root's taller subtree; ie.

$$\textbf{height(T)} = 1 + \textbf{max}\{\text{height}(T_L), \text{height}(T_R)\}$$

Height of Binary Tree (cont.)

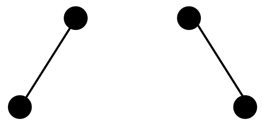


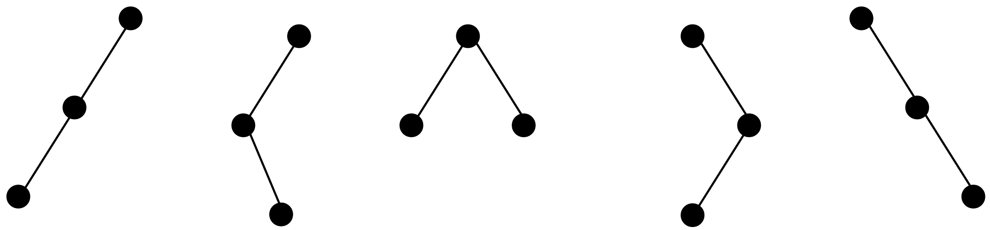
Binary trees with the same nodes but different heights

Number of Binary trees with Same # of Nodes

n=0 ➔ empty tree

n=1 ➔  (1 tree)

n=2 ➔  (2 trees)

n=3 ➔  (5 trees)

n is even ➔ $$NumBT(N) = 2\sum_{i=0}^{(n-1)/2}(NumBT(i)NumBT(n-i-1))$$

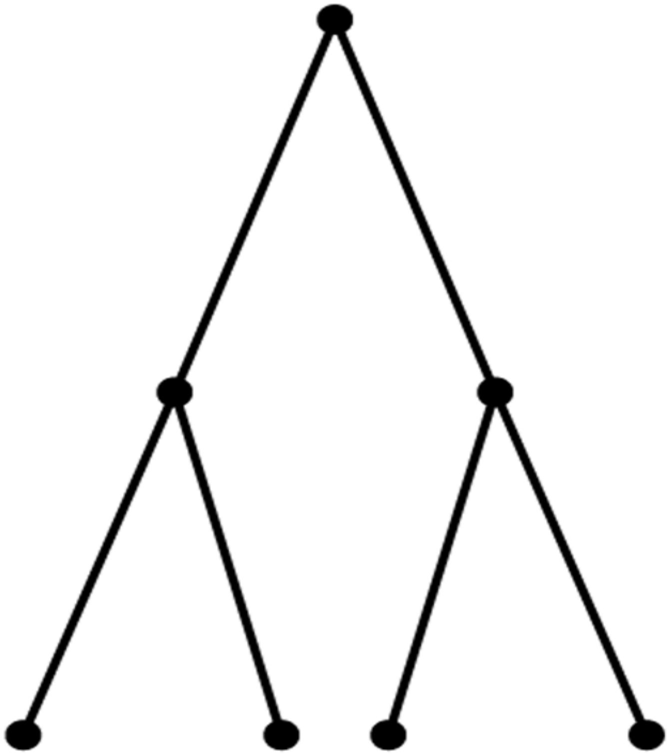
n is odd ➔ $$NumBT(N) = 2\sum_{i=0}^{((n-1)/2)-1}(NumBT(i)NumBT(n-i-1))$$

$$+ \ NumBT((n-1)/2)NumBT((n-1)/2)$$

Full Binary Tree

- In a ***full binary tree*** of height h, all nodes that are at a level less than h have two children each.
- Each node in a full binary tree has left and right subtrees of the same height.
- Among binary trees of height h, a full binary tree has as many leaves as possible, and **leaves all are at level h**.
- A full binary tree has **no missing nodes**.
- Recursive definition of 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.

Full Binary Tree – Example



A full binary tree of height 3

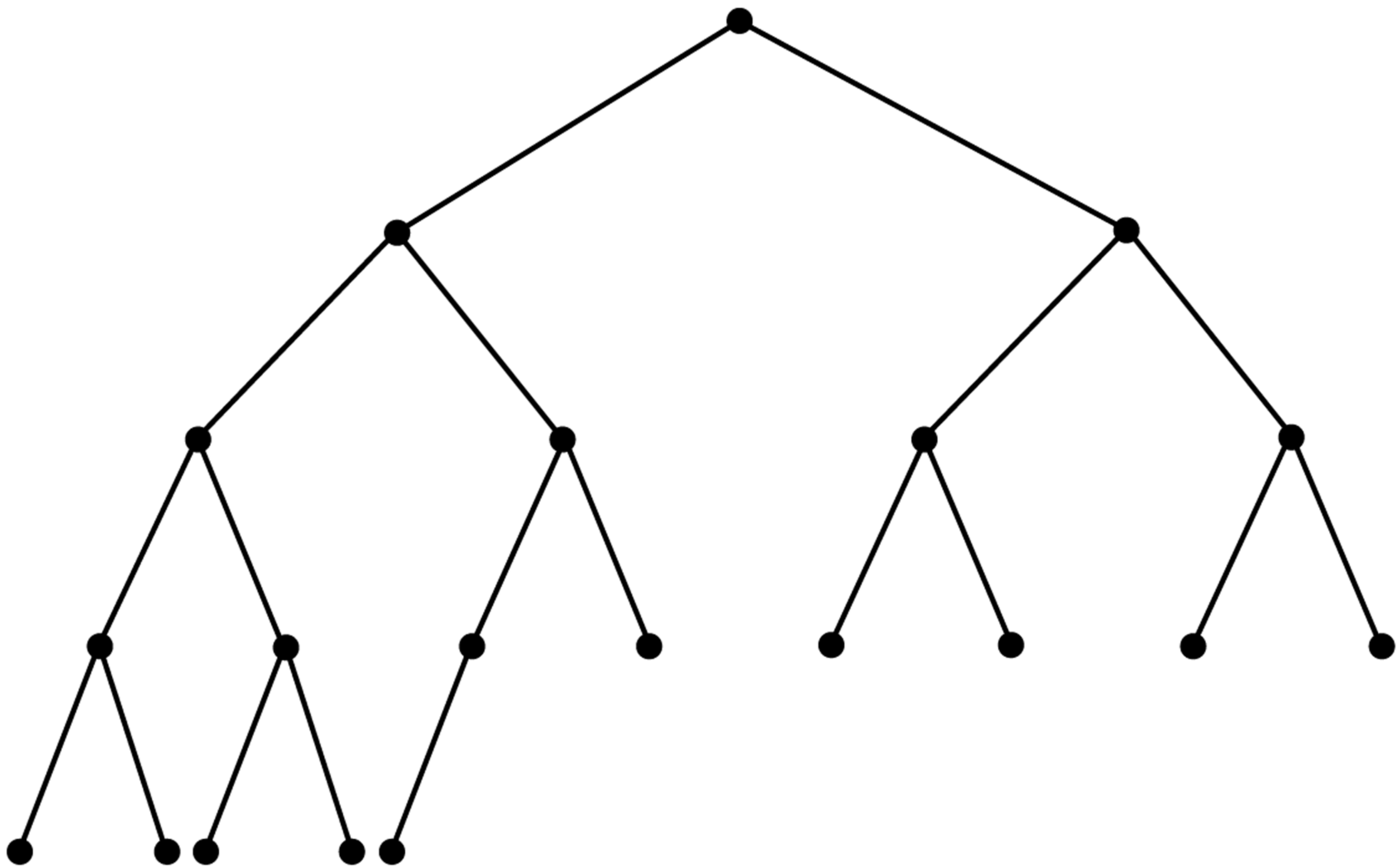
Complete Binary Tree

- A ***complete binary tree*** of height h is a binary tree that is **full down to level h-1**, with level h filled in from left to right.

- A binary tree T of height h is complete if
  1. All nodes at level h-2 and above have two children each, and
  2. When a node at level h-1 has children, all nodes to its left at the same level have two children each, and
  3. When a node at level h-1 has one child, it is a left child.

- A full binary tree is a complete binary tree.

Complete Binary Tree – Example



Balanced Binary Tree

- A binary tree is ***balanced*** (or ***height balanced***), if the height of any node's right subtree and left subtree **differ no more than 1**.

- A complete binary tree is a balanced tree. Why?

- Later, we look at other height balanced trees.
  - AVL trees
  - Red-Black trees, ...

Maximum and Minimum Heights of a Binary Tree

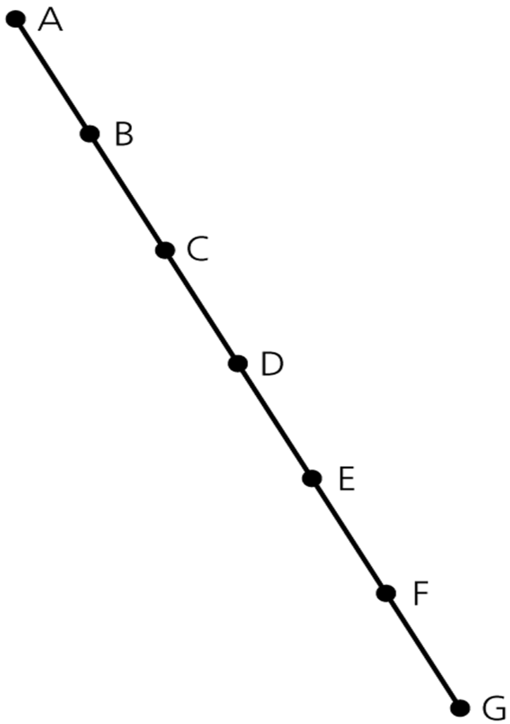
- **Efficiency** of most binary tree operations **depends on tree height**.

- **E.g. maximum number of key comparisons** for retrieval, deletion, and insertion operations for BSTs is the height of the tree.

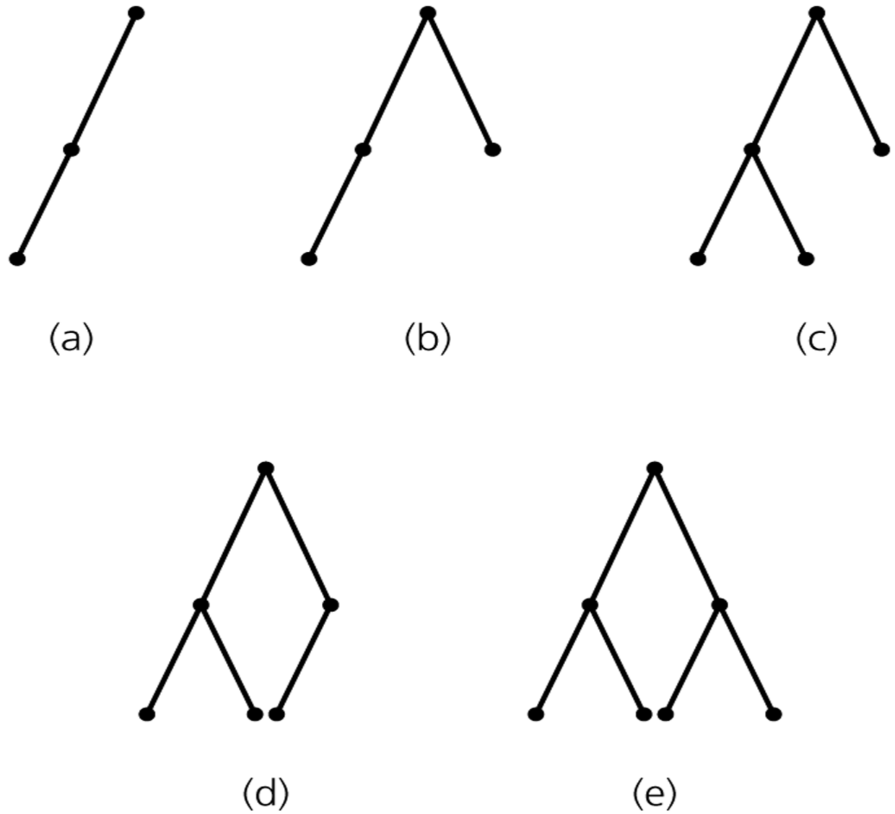
- The maximum of height of a binary tree with n nodes is n. How?

- Each level of a minimum height tree, except the last level, must contain as many nodes as possible.
  - Should the tree be a Complete Binary Tree?

Maximum and Minimum Heights of a Binary Tree

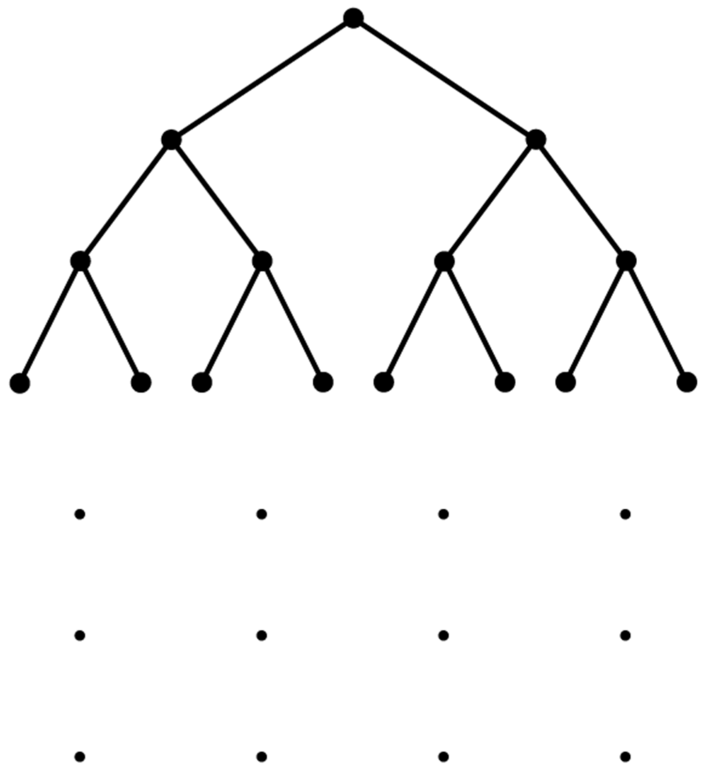


A maximum-height binary tree
with seven nodes



Some binary trees of height 3

Counting the nodes in a full binary tree of height *h*



Level	Number of nodes at this level	Number of nodes at this and previous levels
1	$1 = 2^0$	$1 = 2^1 - 1$
2	$2 = 2^1$	$3 = 2^2 - 1$
3	$4 = 2^2$	$7 = 2^3 - 1$
4	$8 = 2^3$	$15 = 2^4 - 1$
$\vdots$	$\vdots$	$\vdots$
h	2^{h-1}	$2^h - 1$

Some Height Theorems

Theorem: A full binary tree of height $h \geq 0$ has 2^h-1 nodes.

- The maximum number of nodes that a binary tree of height h can have is 2^h-1.

- We cannot insert a new node into a full binary tree without increasing its height.

Some Height Theorems

Theorem 10-4: The minimum height of a binary tree with n nodes is $\lceil \log_2(n+1) \rceil$.

Proof: Let h be the smallest integer such that $n \le 2^h-1$. We can establish following facts:

Fact 1 – A binary tree whose height is $\le$ h-1 has $<$ n nodes.

– Otherwise h cannot be smallest integer in our assumption.

Fact 2 – There exists a complete binary tree of height h that has exactly n nodes.

– A full binary tree of height h-1 has $2^{h-1}-1$ nodes.

– Since a binary tree of height h cannot have more than 2^h-1 nodes.

– At level h, we will reach n nodes.

Fact 3 – The minimum height of a binary tree with n nodes is the smallest integer h such that $n \le 2^h-1$.

So, $\Rightarrow$ $2^{h-1}-1 < n \le 2^h-1$

$\Rightarrow$ $2^{h-1} < n+1 \le 2^h$

$\Rightarrow$ $h-1 < \log_2(n+1) \le h$

Thus, $\Rightarrow$ $h = \lceil \log_2(n+1) \rceil$ is the minimum height of a binary tree with n nodes.

- UML Diagram for **BinaryTree ADT**

- What is an **ADT**?

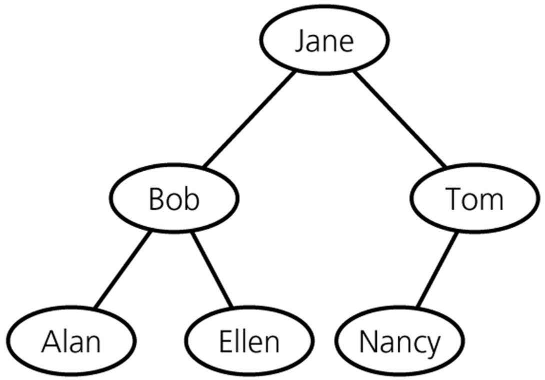
Binary tree
root *left subtree* *right subtree*
createTree() *destroyBinaryTree()* *isEmpty()* *getRootData()* *setRootData()* *attachRight()* *attachLeftSubtree()* *attachRightSubtree()* *detachLeftSubtree()* *detachRightSubtree()* *getLeftSubtree()* *getRightSubtree()* *preorderTraverse()* *inorderTraverse()* *postorderTraverse()*

An Array-Based Implementation of Binary Trees

```
public class BinaryTree <Value>

   private class Node
   {
      private Value val;
      private int left, right;
      public Node(Value val)
      {
          this.val = val;
          this.left = this.right = -1;
      }
   }


   const int MAX_NODES = 100;          // maximum number of nodes

   // An array of tree nodes
   Node[MAX_NODES] tree;
   int  root;
   int  free;

   …      // methods

}
```

An Array-Based Implementation (cont.)

(a)



(b)

`tree`

	`item`	`leftChild`	`rightChild`
0	Jane	1	2
1	Bob	3	4
2	Tom	5	–1
3	Alan	–1	–1
4	Ellen	–1	–1
5	Nancy	–1	–1
6	?	–1	7
7	?	–1	8
8	?	–1	9
⋮	⋮	⋮	⋮

`root`: 0

`free`: 6

Free list

- A ***free list*** keeps track of available nodes.
- To insert a new node into the tree, we first obtain an available node from the free list.
- When we delete a node from the tree, we have to place into the free list so that we can use it later.

An Array-Based Representation of a Complete Binary Tree

- If we know that our binary tree is a **complete binary tree**, we can use a simpler array-based representation for complete binary trees
  - **without** using **leftChild, rightChild** links

- We can number the nodes level by level, and left to right (starting from 0, the root will be 0). If a node is numbered as i, in the ith location of the array, `tree[i]`, contains this node without links.

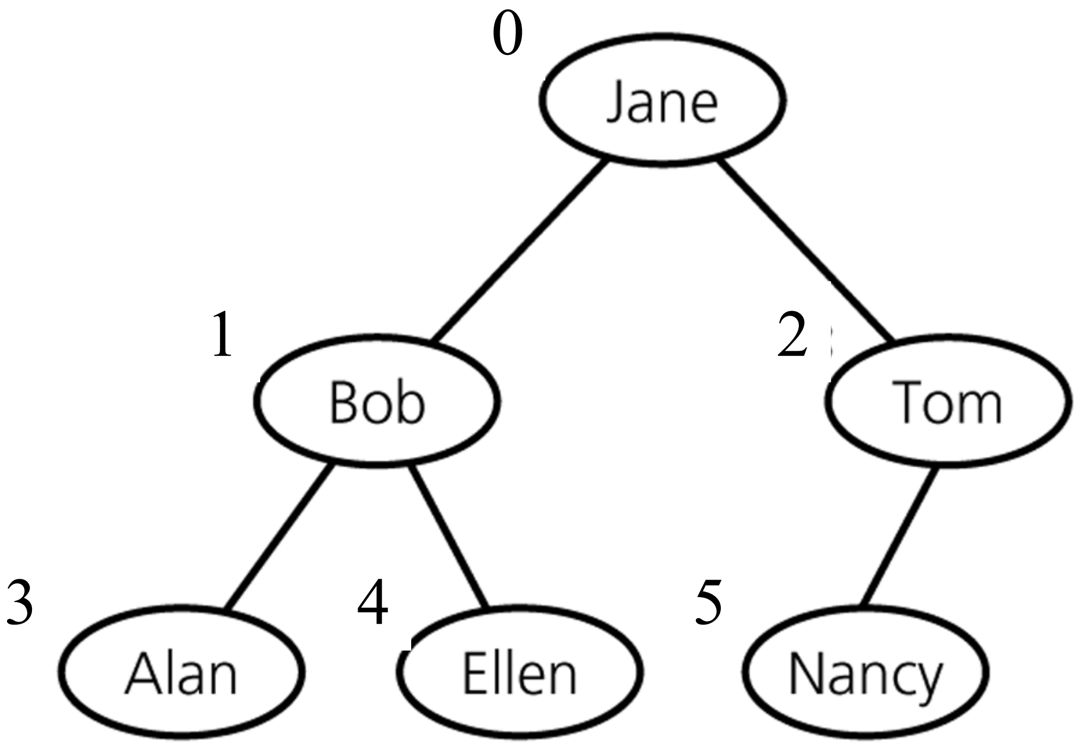
- Using these numbers we can find leftChild, rightChild, and parent of a node i.

The left child (if it exists) of node i is `tree[2*i+1]`

The right child (if it exists) of node i is `tree[2*i+2]`

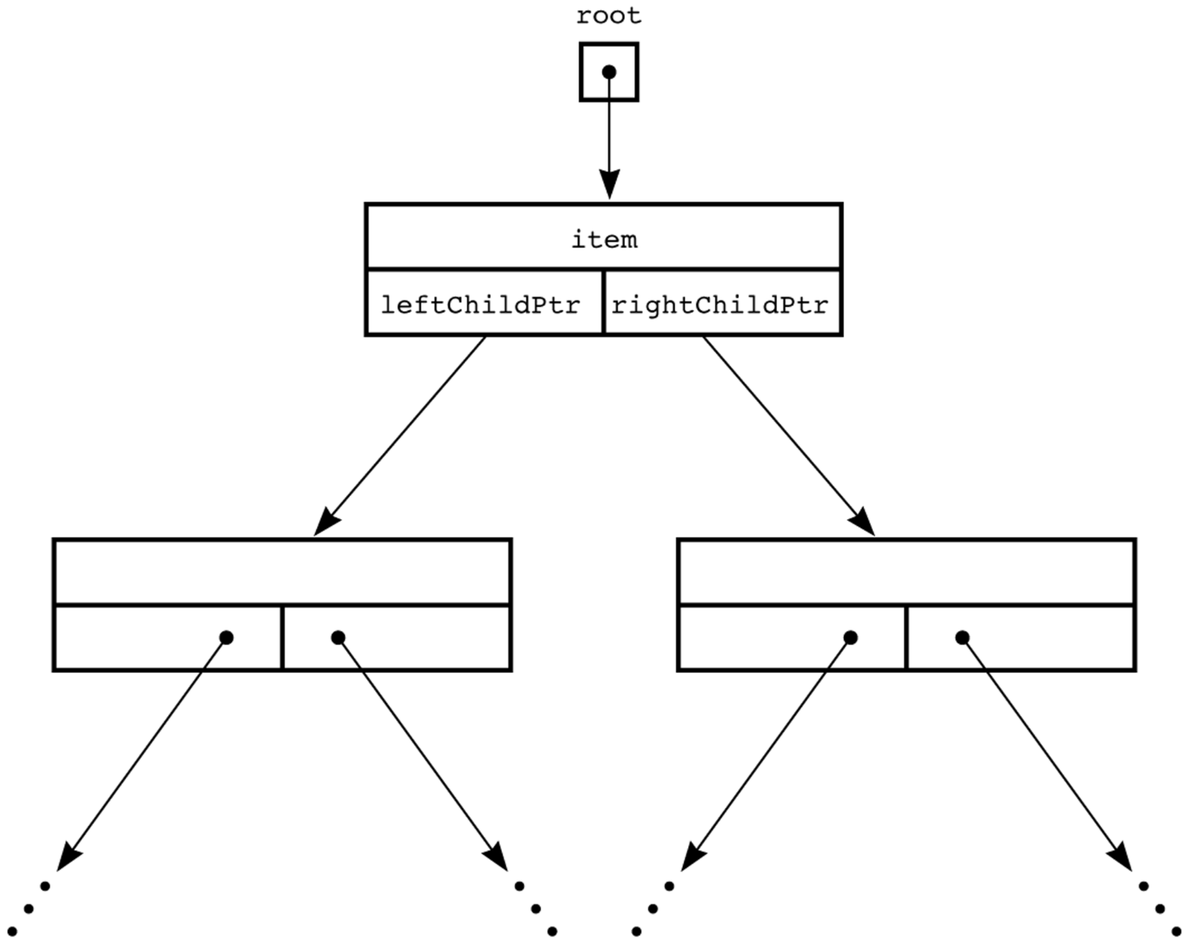
The parent (if it exists) of node i is `tree[(i-1)/2]`

An Array-Based Representation of a Complete Binary Tree (cont.)



0	Jane
1	Bob
2	Tom
3	Alan
4	Ellen
5	Nancy
6	
7	

Linked Implementation of Binary Trees



Linked Implementation of a Binary Tree Node

```
public class BinaryTree <Value>

   private class Node
   {
      private Value val;
      private Node left, right;
      public Node(Value val)
      {
         this.val = val;
         this.left= this.right = null;
      }

   }

   private Node root;

   …         // methods
}
```

Binary tree

root
left subtree
right subtree

createTree()
destroyBinaryTree()
isEmpty()
getRootData()
setRootData()
attachRight()
attachLeftSubtree()
attachRightSubtree()
detachLeftSubtree()
detachRightSubtree()
getLeftSubtree()
getRightSubtree()
preorderTraverse()
inorderTraverse()
postorderTraverse()

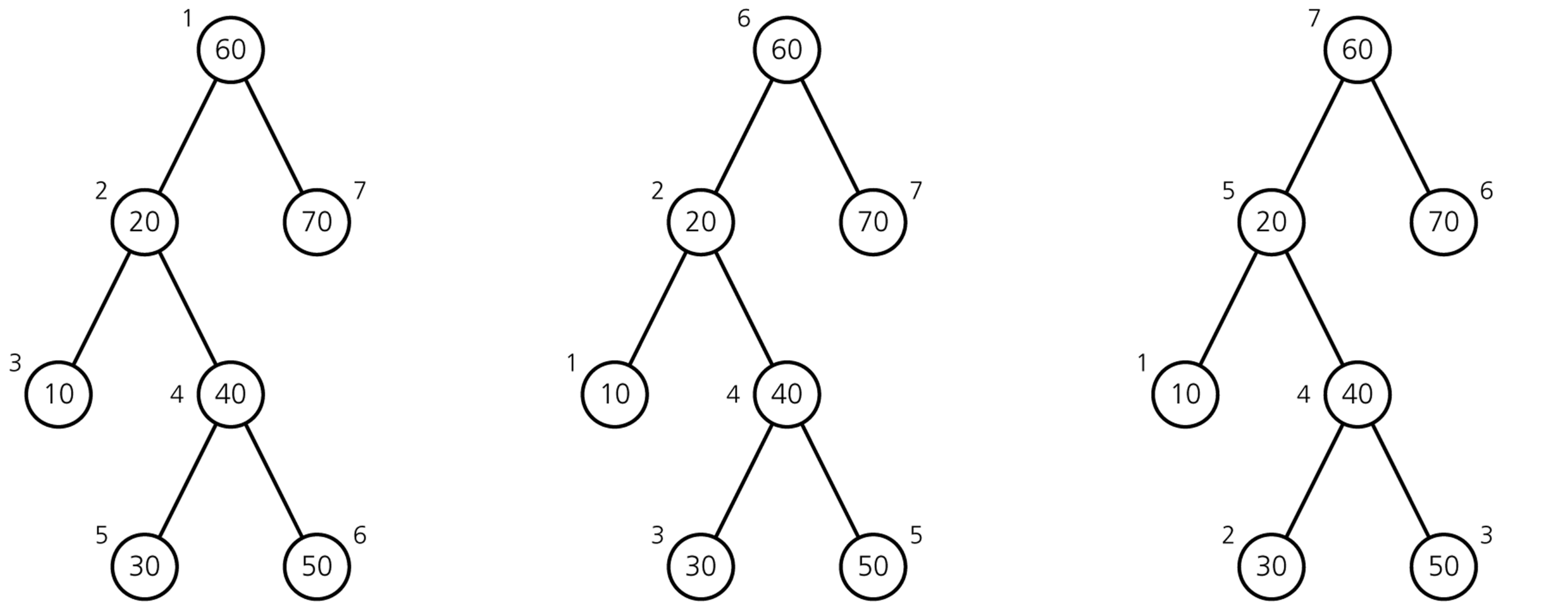
Binary Tree Traversals

- **Preorder Traversal**
  - The node is visited before its left and right subtrees,

- **Postorder Traversal**
  - The node is visited after both subtrees.

- **Inorder Traversal**
  - The node is visited between the subtrees,
  - Visit left subtree, visit the node, and visit the right subtree.

Binary Tree Traversals



(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.)

Complexity of Traversals

What is the complexity of each traversal type?

- Preorder traversal

- Postorder traversal

- Inorder traversal