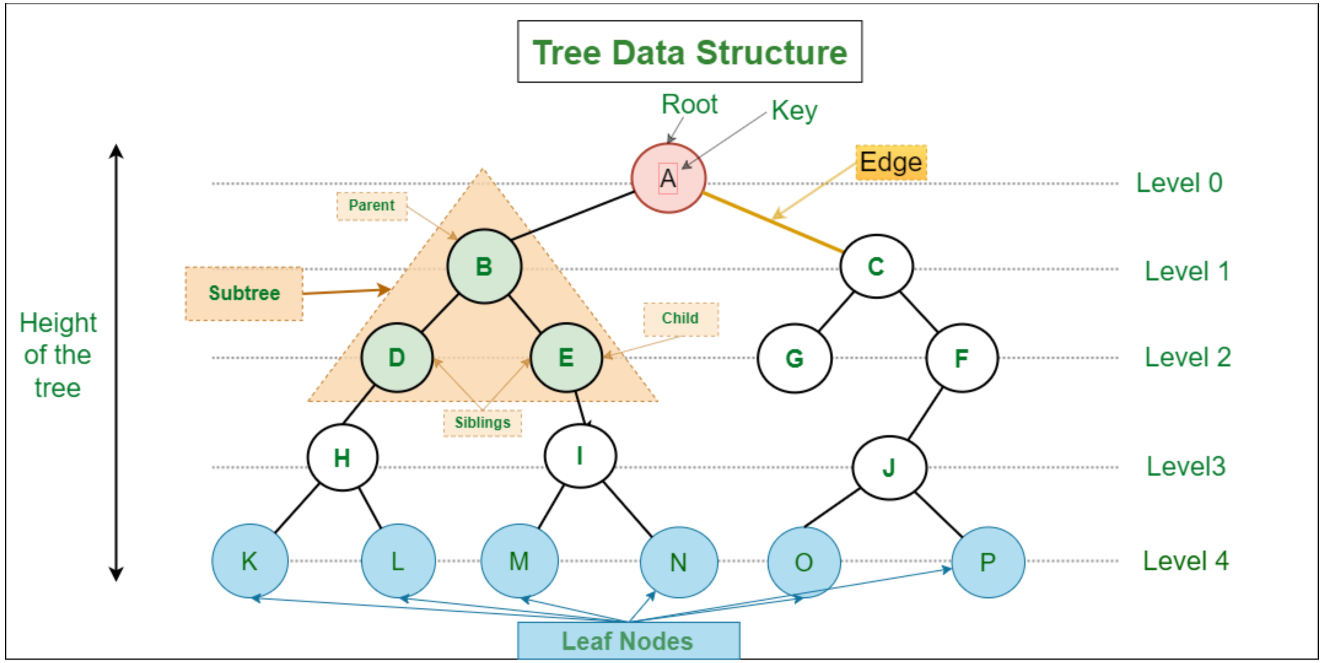
Unit - II
Trees

COURSE CODE: 21AI33

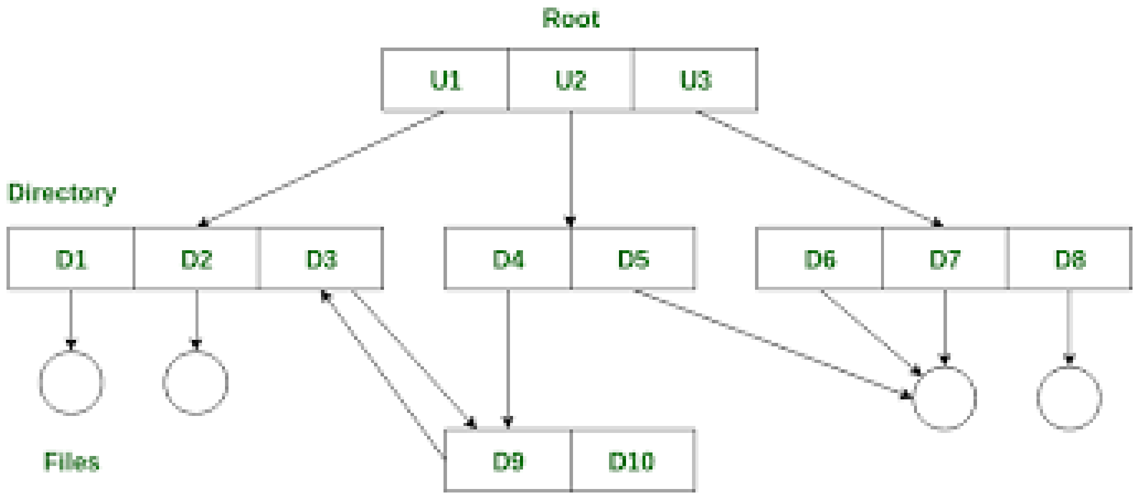
Trees

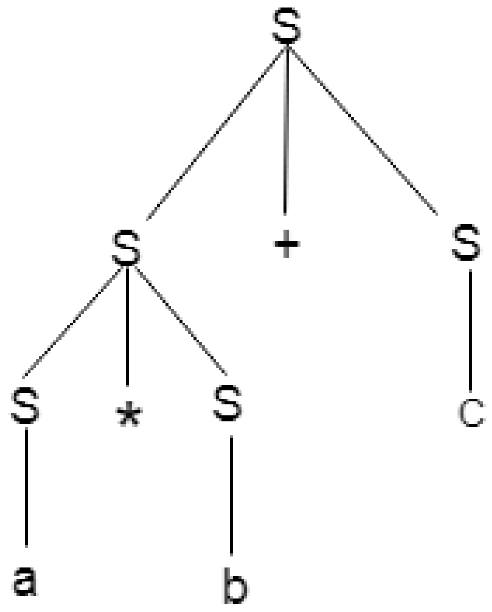


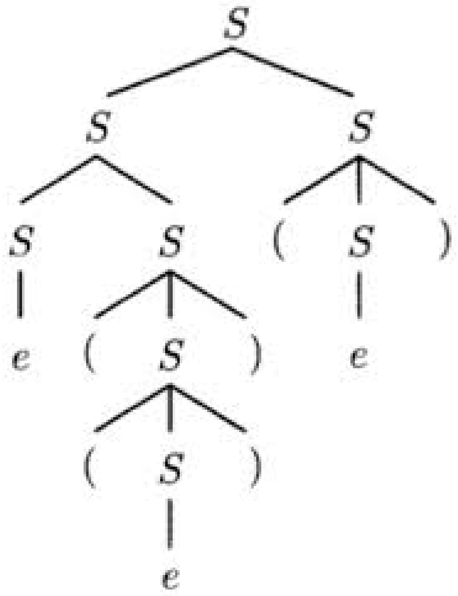
Tree Data Structure
Root
Key
Edge
A
Parent
B
C
Subtree
Child
Height of the tree
D
E
G
F
Siblings
H
I
J
K
L
M
N
O
P
Leaf Nodes
Level 0
Level 1
Level 2
Level3
Level 4

Trees Concepts

- Represents information in hierarchical format
- Examples: File Directory, Parse Trees, Expression Trees

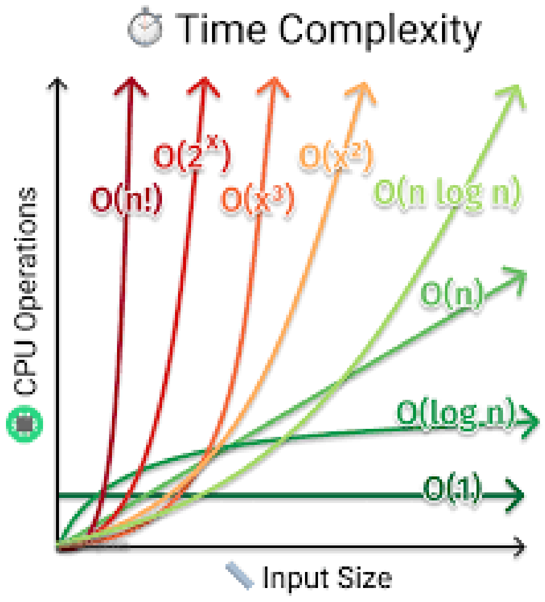






Trees Concepts

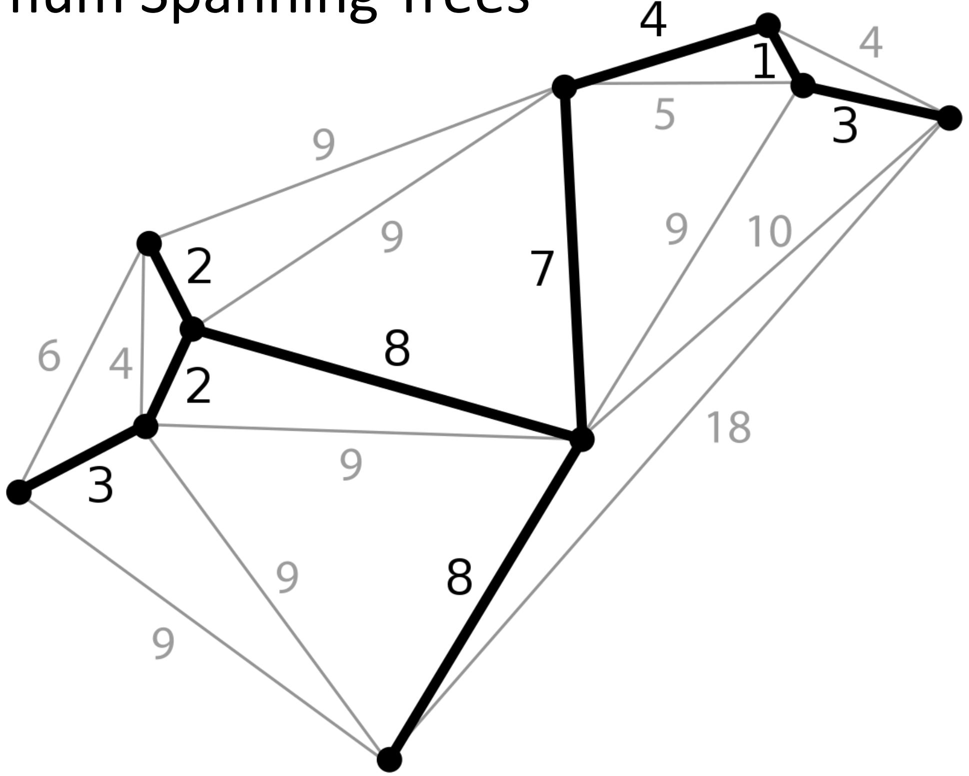
- Logarithmic time complexity

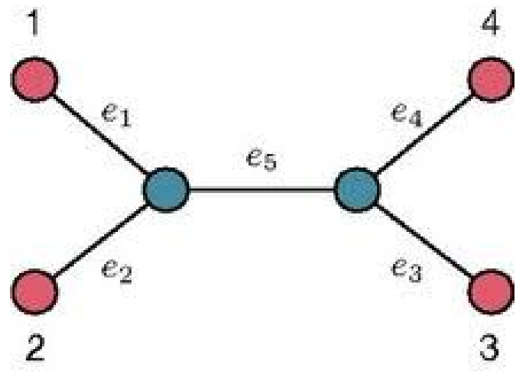


Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)
Tim Sort	O(n)	O(nlogn)	O(nlogn)	O(n)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)

Tree Types

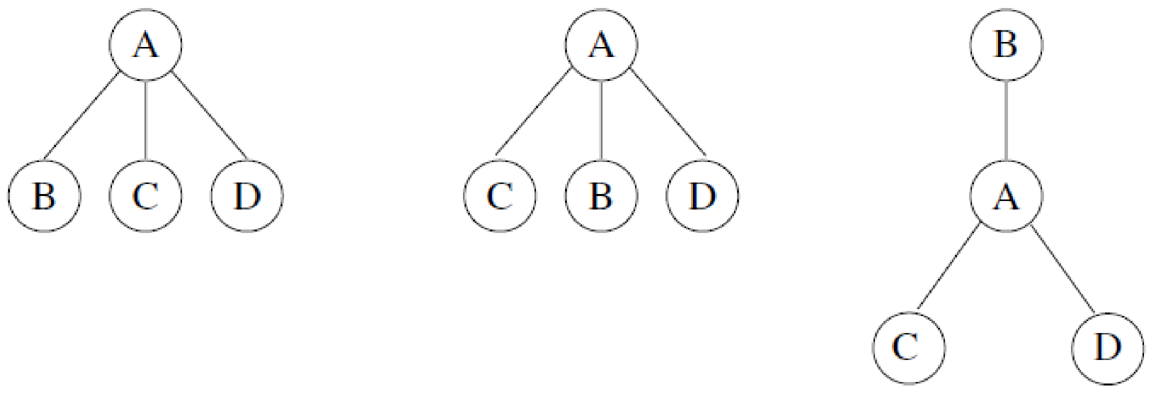
- Unrooted Tree
- Minimum Spanning Trees





Tree Types

- Rooted Tree
- Special Node called Root Node
- The remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1, ..., T_n$, where each of these sets is a tree. $T_1, ..., T_n$ are called the subtrees of the root.
- May be ordered or unordered based on the placement of subtrees



K-ary Tree

- A finite set of nodes that is either empty or consists of a root and the elements of *k* disjoint *k*-ary trees called the 1st, 2nd, ..., *k*th subtrees of the root.
- Example: Binary tree has K=2 (left and right branches)
- Binary tree can have no nodes (Empty trees), but tree cannot

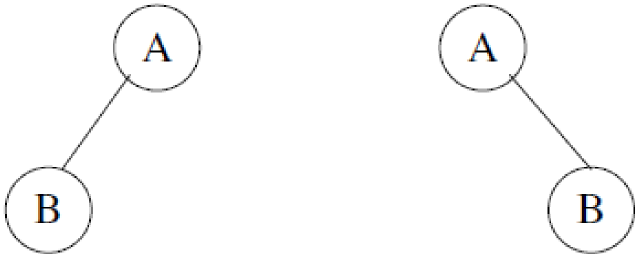
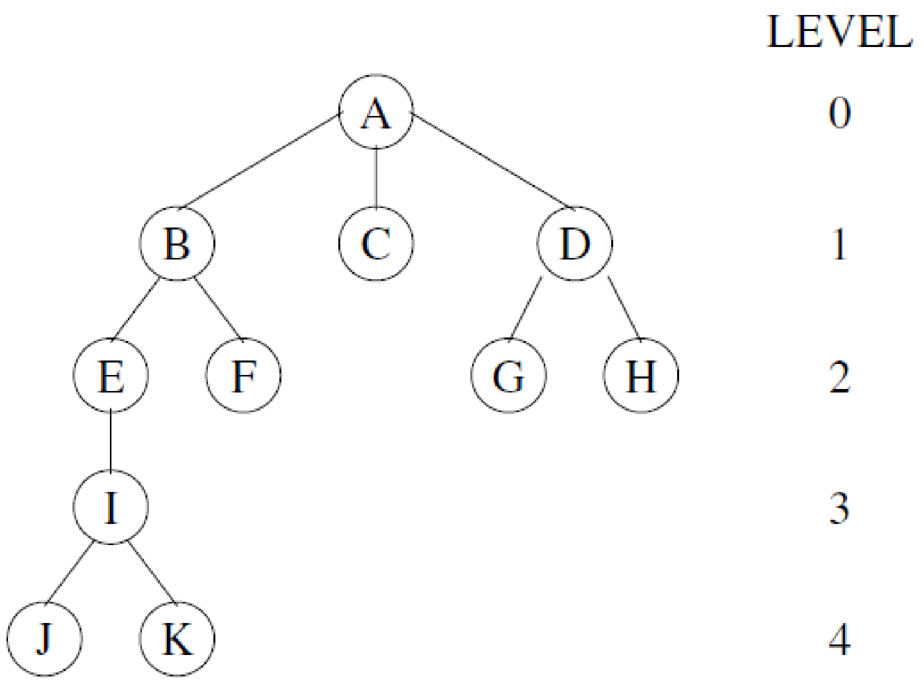


FIGURE 3.2: Different binary trees.

Tree Representations

- List Representation

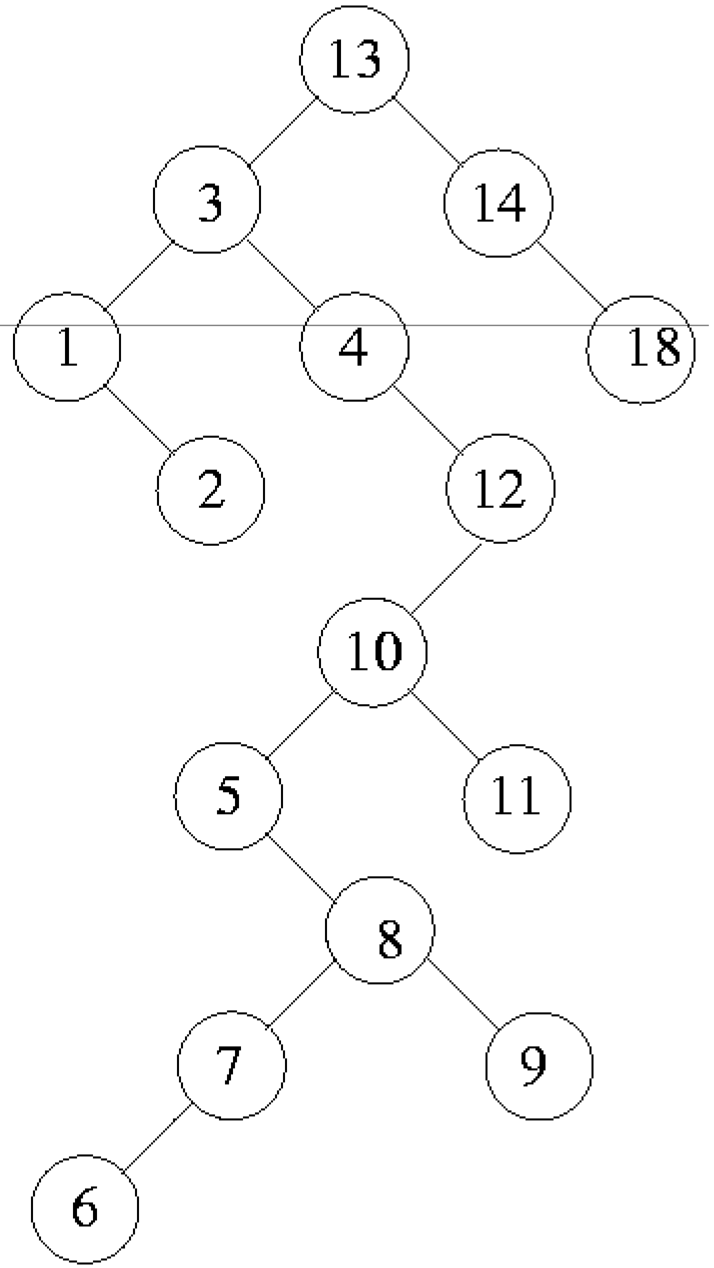
(A (B (E (I (J, K)), F), C, D(G, H)))



Tree Representations

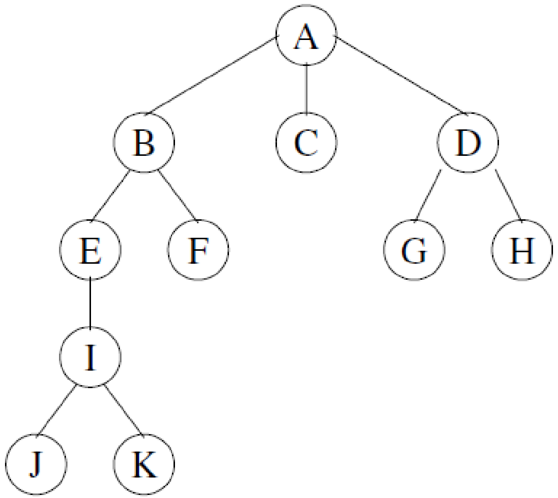
- List Representation

ANS: ??



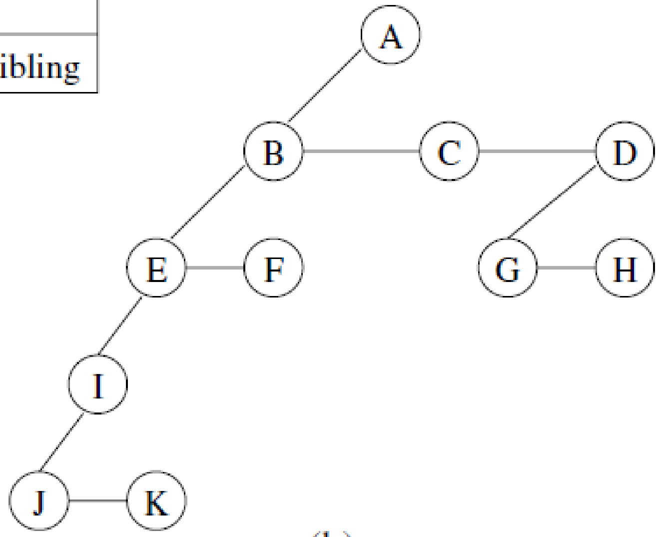
Tree Representations

- Left-Child and Right-Sibling Representation & Binary Tree Representation

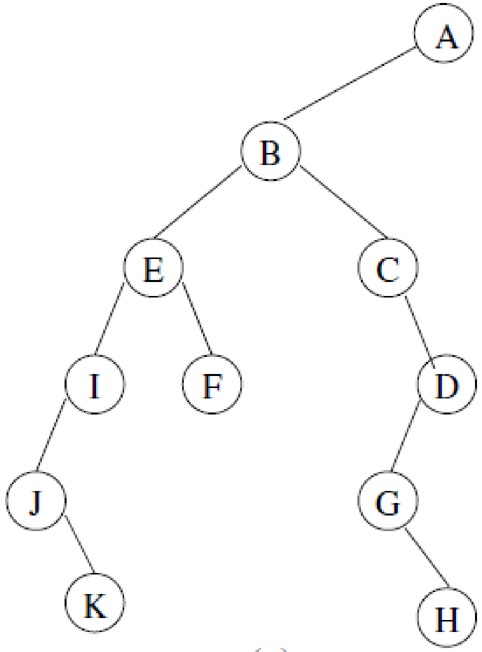


data	
left child	right sibling

(a)



(b)



(c)

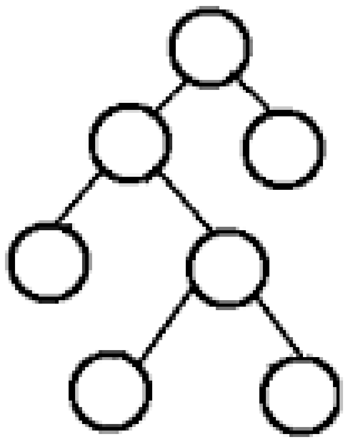
Binary Trees Types

A full Binary tree is **a special type of binary tree in which every parent node/internal node has either two or no children**. It is also known as a proper binary tree.

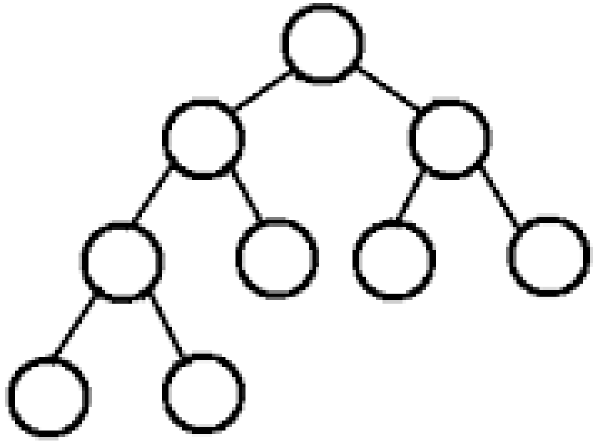
A complete binary tree is **a binary tree in which every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible**.

A perfect Binary Tree is **a binary tree in which each of the internal nodes has exactly two child nodes and all the leaf nodes are situated at the same level of the tree**.

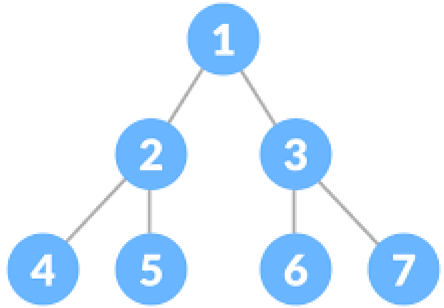
A balanced binary Tree where, **difference between the left and the right subtree for any node is not more than one.**



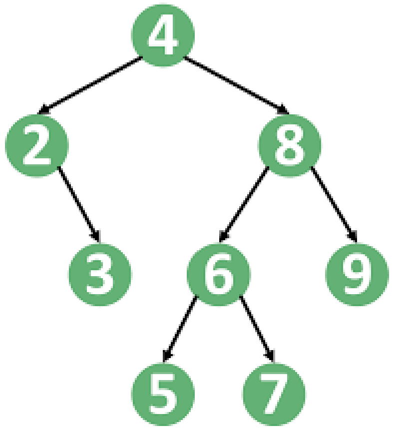
full tree



complete tree



Perfect BT

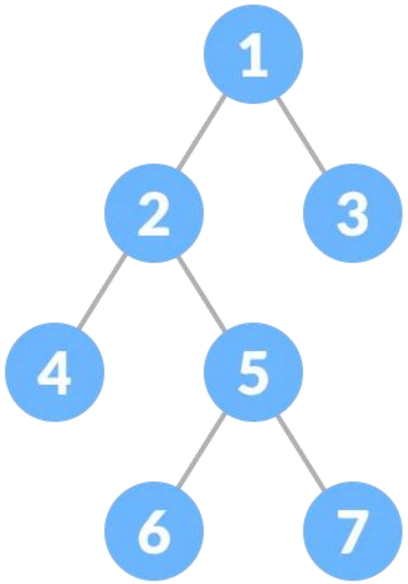


Balanced BT

Full BT Theorems

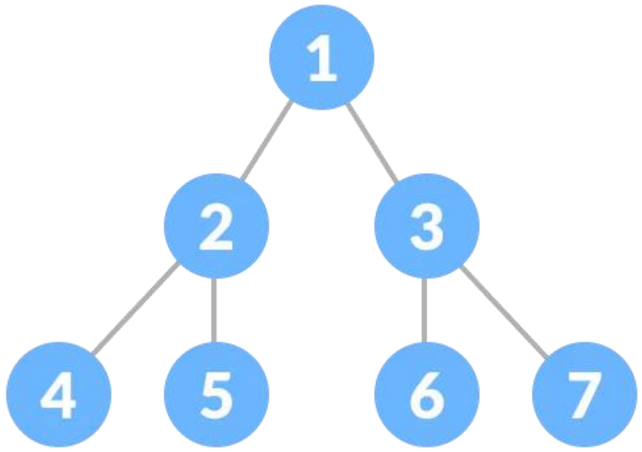
1. The number of leaves is `i + 1`.
2. The total number of nodes is `2i + 1`.
3. The number of internal nodes is `(n – 1) / 2`.
4. The number of leaves is `(n + 1) / 2`.
5. The total number of nodes is `2l – 1`.
6. The number of internal nodes is `l – 1`.
7. The number of leaves is at most $2^{\lambda - 1}$.

i = the number of internal nodes
n = be the total number of nodes
l = number of leaves
λ = number of levels



Perfect BT Theorems

1. A perfect binary tree of height h has $2^{h + 1} - 1$ node.
2. A perfect binary tree with n nodes has height $\log(n + 1) - 1 = \Theta(\ln(n))$.
3. A perfect binary tree of height h has 2^h leaf nodes.
4. The average depth of a node in a perfect binary tree is $\Theta(\ln(n))$.



BT Properties

- **Total number of nodes in a full binary tree = Number of non-leaf nodes + Number of leaf nodes**
  - **Number of leaf nodes = Number of non-leaf nodes + 1**
  - **Number of non-leaf nodes = Number of leaf nodes – 1**
- ***If number of leaf nodes in a full binary tree is x, how many nodes are present in that tree?***
- ***If number of non-leaf nodes in a full binary tree is x, how many nodes are present in that tree?***

BT Properties

- ***How many binary tree can be formed with 3 nodes?***

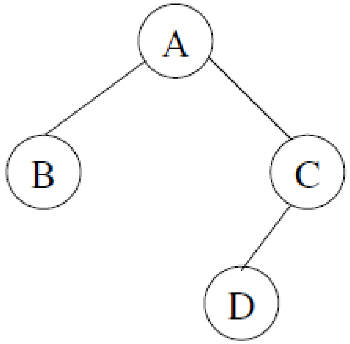
- ***How many ways you can allocate the data in 3 nodes Binary Tree?***

- ***What is the maximum and minimum number of leaf nodes of a binary tree with n nodes?***

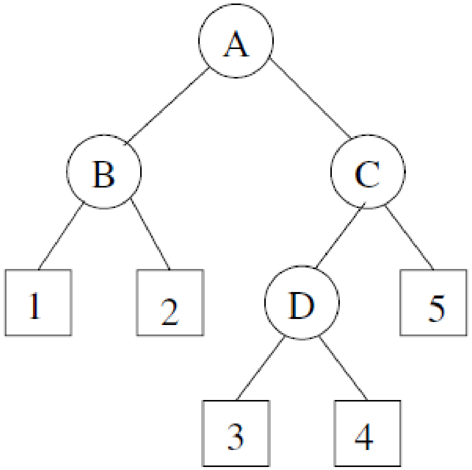
- ***How many levels will there be in a completely binary tree if it has n number of nodes?***

Binary Trees and Properties

A binary tree with *n* internal nodes has *n* + 1 external nodes.



(a)



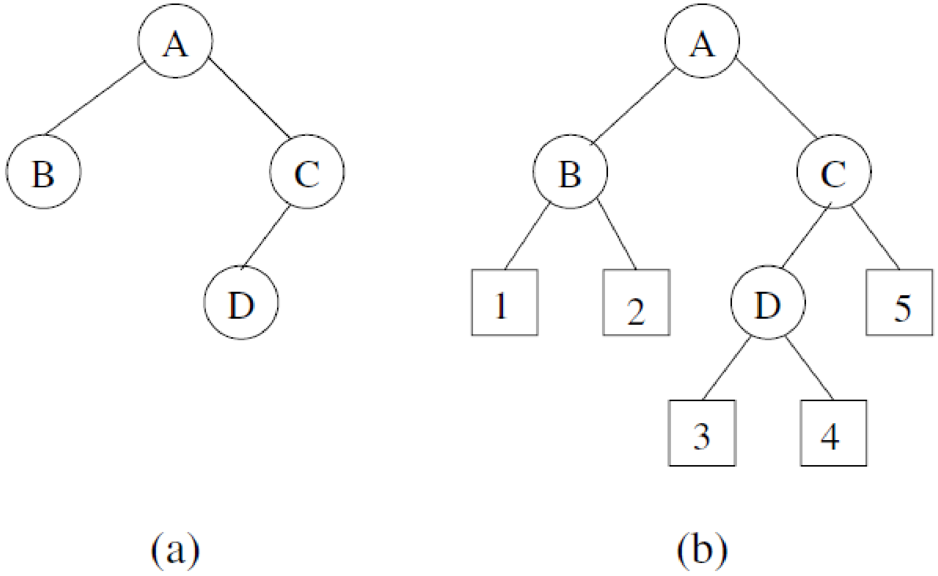
(b)

Proof:

- **Each internal node has maximum two children**
- **Total number of branches = 2n**
- **n-1 internal nodes have a single incoming branch**
- **2n-(n-1)= n +1**
- **n+1 branches points to external node**

Binary Trees and Properties

For any non-empty binary tree with n_0 leaf nodes and n_2 nodes of degree 2, $n_0 = n_2 + 1$.



Proof:

- Let n_1 be the number of nodes of degree 1
- Total number of nodes in the tree are
  $n = n_0 + n_1 + n_2$ (Eq. 1)
- The number of branches in a binary tree is $n-1$ since each non-root node has a branch leading into it.
- All branches are from nodes of degree 1 and 2. Thus, the number of branches is $n_1 + 2n_2$.
- Equating the two expressions for number of branches, we get $n = n_1 + 2n_2 + 1$ (Eq. 2)

From Eq.1 and Eq. 2 we get $n_0 = n_2 + 1$

Binary Trees and Properties

The height of a binary tree with n internal nodes is at least $\log_2(n + 1)$ and at most $n - 1$.

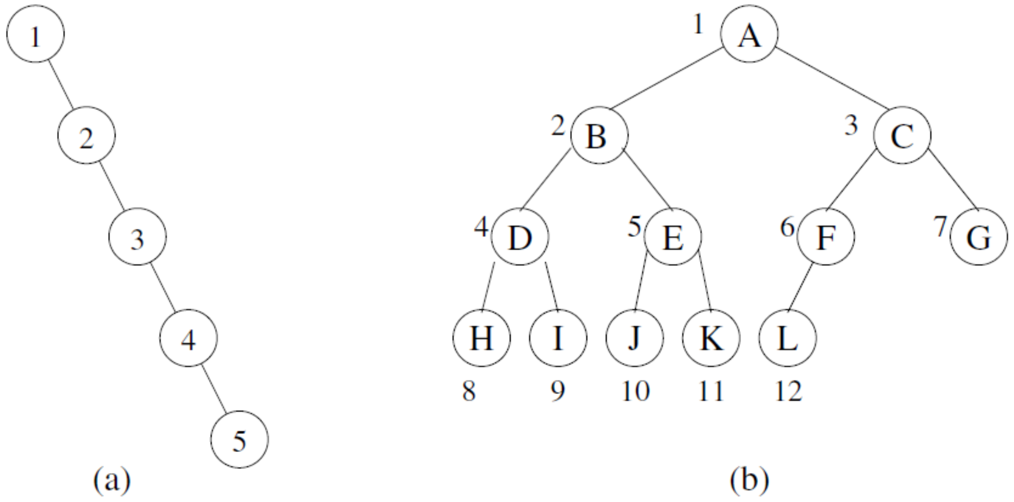


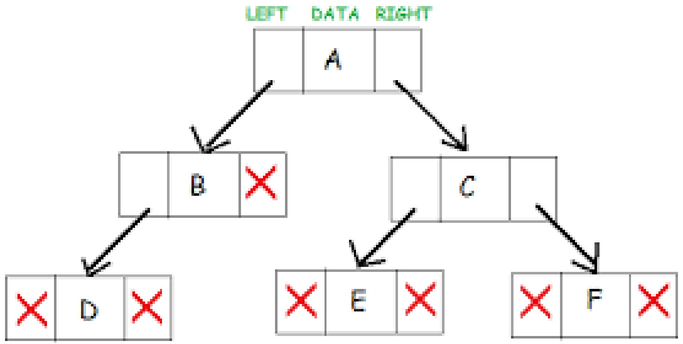
FIGURE 3.6: (a) Skewed and (b) complete binary trees.

Proof:

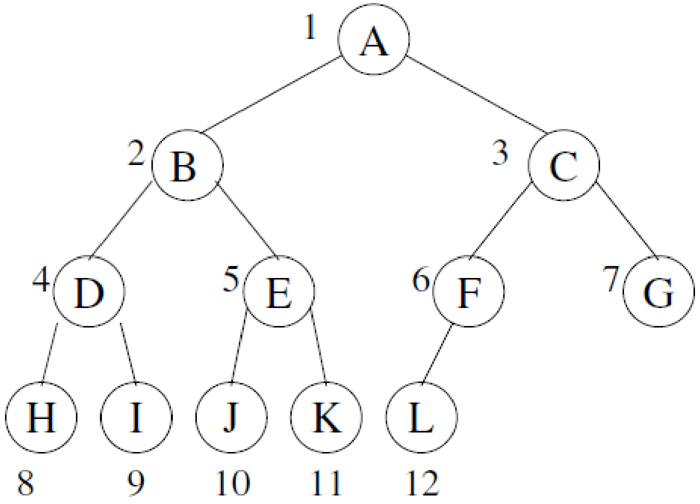
- The worst case is a skewed tree
- The best case is a tree with $2i$ nodes at every level i except possibly the bottom level.
- If the height is h, then $n + 1 \leq 2h$, where $n + 1$ is the number of external nodes.

Binary Trees Representations

Nodes and Pointers




```
struct treenode
{
    int data;
    struct node *lchild, * rchild;
};
```



Array Representation

1	2	3	4	5	6	7	8	9	10	11	12
[A	B	C	D	E	F	G	H	I	J	K	L]

Binary Tree Traversals

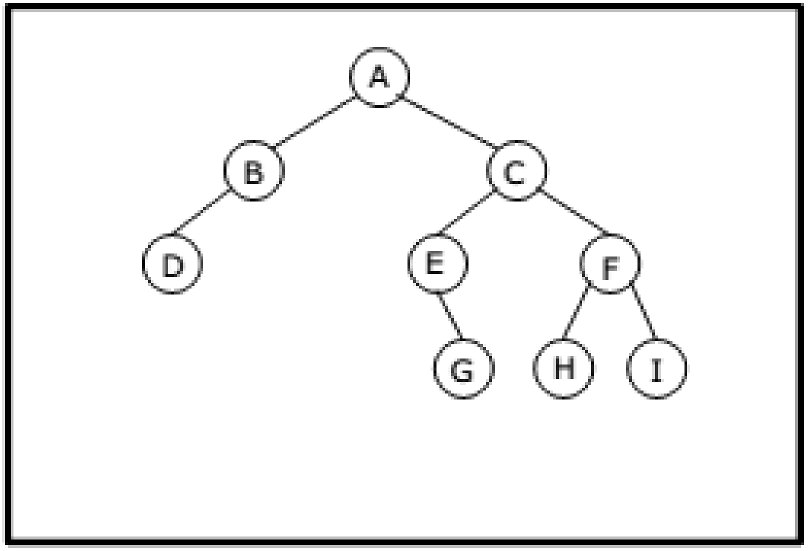
- Processing every node in the tree systematically is the purpose of traversal
- Starting at a node, we can do one of three things:

  visit the node (*V*),

  traverse the left subtree recursively (*L*), and

  traverse the right subtree recursively (*R*)
- inorder(LVR), preorder(VLR) and postorder(LRV)

Binary Tree Traversals

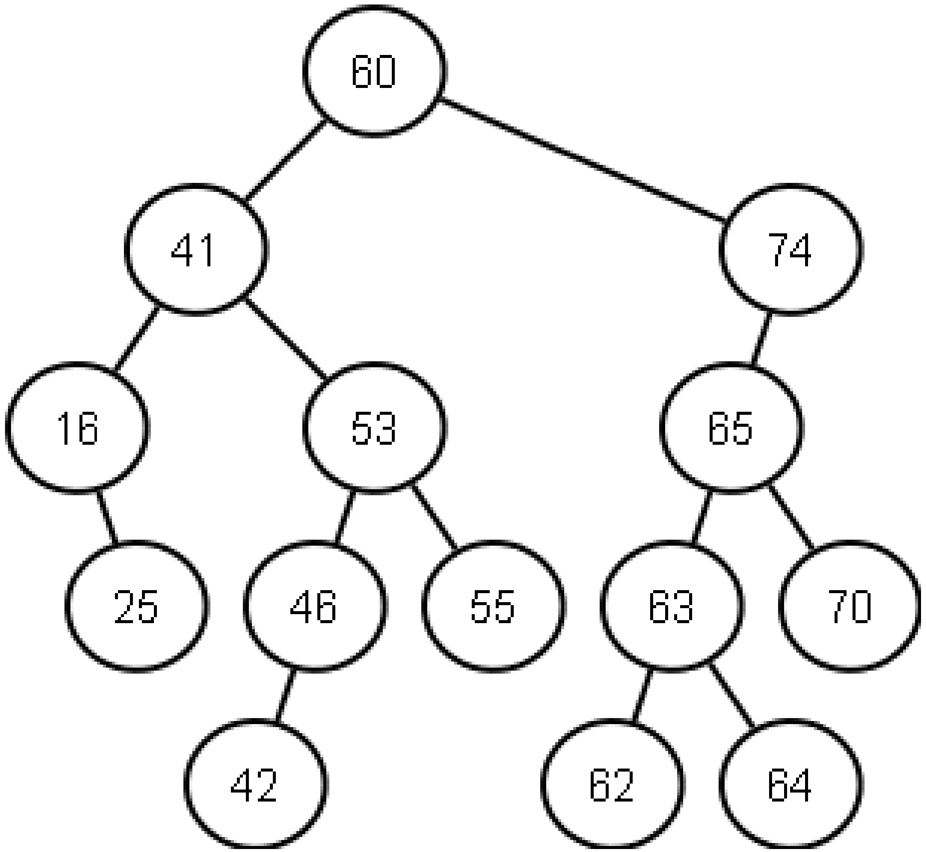


Binary Tree

- Preorder traversal yields:
  A, B, D, C, E, G, F, H, I
- Postorder traversal yields:
  D, B, G, E, H, I, F, C, A
- Inorder traversal yields:
  D, B, A, E, G, C, H, F, I
- Level order traversal yields:
  A, B, C, D, E, F, G, H, I

Pre, Post, Inorder and level order Traversing

Binary Tree Traversals

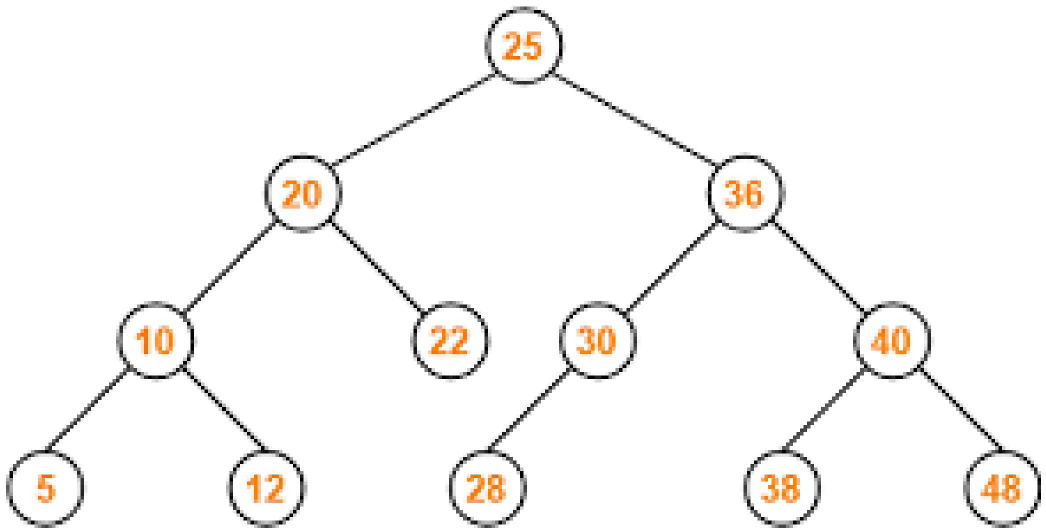


inorder logic

```
inorder (struct node *currentnode)
{
   if (currentnode)
  {
     inorder(currentnode->lchild);
     print(currentnode->data); //process the node
    inorder(currentnode->rchild);
  }
}
```

Binary Search Tree

- Creation of Tree
- Traversals
- Deletion of node with a Key
- Searching of a node with a Key
- Etc.



Binary Search Tree

Bina

- *Minimum* and *Maximum* that respectively find the minimum and maximum elements in the binary search tree. The minimum element is found by starting at the root and following `LeftChild` pointers until a node with a 0 `LeftChild` pointer is encountered. That node contains the minimum element in the tree.
- Another operation is to find the kth smallest element in the binary search tree. For this, each node must contain a field with the number of nodes in its left subtree. Suppose that the root has m nodes in its left subtree. If $k \leq m$, we recursively search for the kth smallest element in the left subtree. If $k = m+1$, then the root contains the kth smallest element. If $k > m+1$, then we recursively search the right subtree for the $k - m - 1$st smallest element.
- The Join operation takes two binary search trees A and B as input such that all the elements in A are smaller than all the elements of B. The objective is to obtain a binary search tree C which contains all the elements originally in A and B. This is accomplished by deleting the node with the largest key in A. This node becomes the root of the new tree C. Its `LeftChild` pointer is set to A and its `RightChild` pointer is set to B.
- The Split operation takes a binary search tree C and a key value k as input. The binary search tree is to be split into two binary search trees A and B such that all keys in A are less than or equal to k and all keys in B are greater than k. This is achieved by searching for k in the binary search tree. The trees A and B are created as the search proceeds down the tree as shown in Figure 3.11.
- An inorder traversal of a binary search tree produces the elements of the binary search tree in sorted order. Similarly, the inorder successor of a node with key k in the binary search tree yields the smallest key larger than k in the tree. (Note that we used this property in the Delete operation described in the previous section.)

Threaded Binary Tree