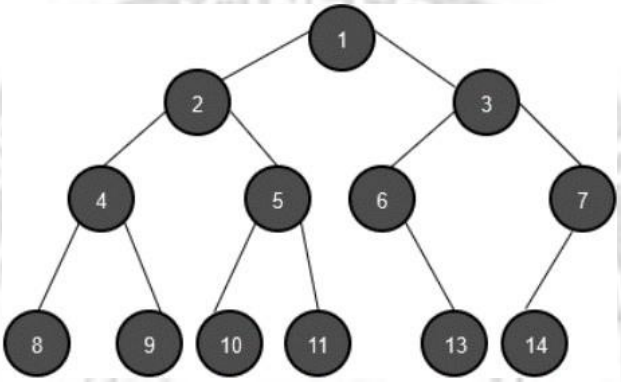
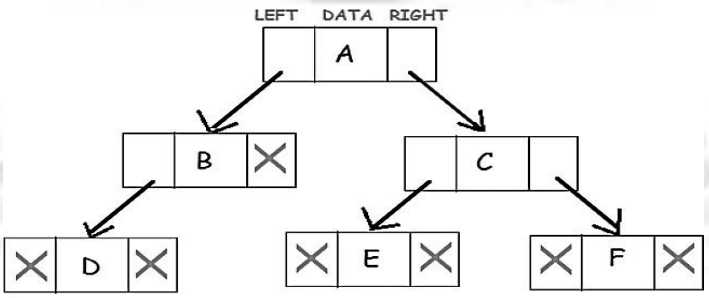
BINARY TREES

- Binary Tree is defined as a Tree data structure with at most 2 children. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.



Binary Tree Representation

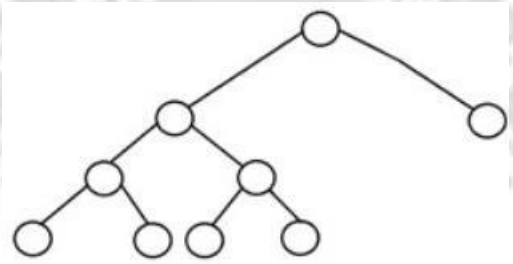
- A Binary tree is represented by a pointer to the topmost node of the tree. If the tree is empty, then the value of the root is NULL.
- Binary Tree node contains the following parts:
  - Data
  - Pointer to left child
  - Pointer to right child



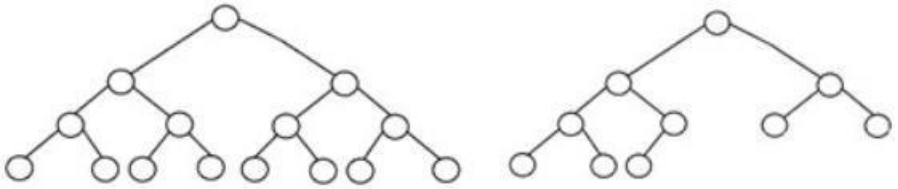
Types of Binary Trees

- There are various types of binary trees, and each of these binary tree types has unique characteristics. Here are each of the binary tree types in detail:

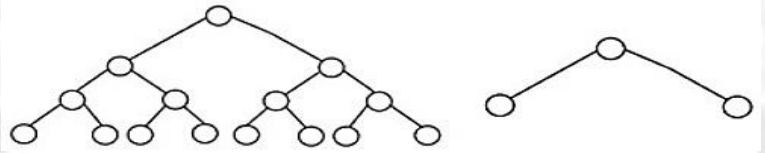
- **Full Binary Tree**
  - It is a special kind of a binary tree that has either zero children or two children. It means that all the nodes in that binary tree should either have two child nodes of its parent node or the parent node is itself the leaf node or the external node.



- **Complete Binary Tree**
  - A complete binary tree is another specific type of binary tree where all the tree levels are filled entirely with nodes, except the lowest level of the tree. Also, in the last or the lowest level of this binary tree, every node should possibly reside on the left side.

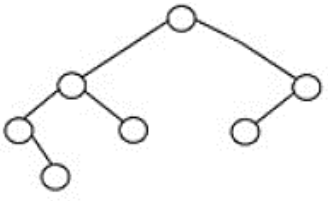


- **Perfect Binary Tree**
  - ✓ A binary tree is said to be 'perfect' if all the internal nodes have strictly two children, and every external or leaf node is at the same level or same depth



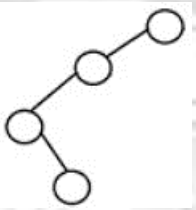
within a tree. A perfect binary tree having height 'h' has 2h – 1 node.

- **Balanced Binary Tree**
  - ✓ A balanced binary tree, also referred to as a height-balanced binary tree, is



defined as a binary tree in which the height of the left and right subtree of any node differs by not more than 1.

- **Degenerate Binary Tree**
  - ✓ A binary tree is said to be a degenerate binary tree or pathological binary tree if every internal node has only a single child.



Benefits of Binary Trees:

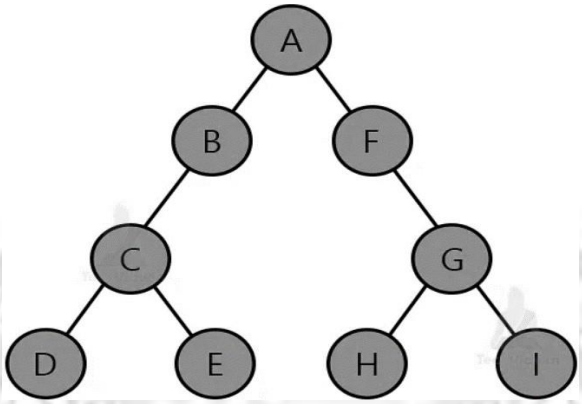
- ❖ The search operation in a binary tree is faster as compared to other trees
- ❖ Only two traversals are enough to provide the elements in sorted order
- ❖ It is easy to pick up the maximum and minimum elements
- ❖ Graph traversal also uses binary trees
- ❖ Converting different postfix and prefix expressions are possible using binary trees

Types of Tree Traversal

- **Preorder traversal**
  - ✓ In a preorder traversal, we process/visit the root node first. Then we traverse the left subtree in a preorder manner. Finally, we visit the right subtree again in a preorder manner.
  - ✓ For example, consider the following tree:

TREE TRAVERSAL

- ➢ Tree traversal means visiting each node of the tree. The tree is a non-linear data structure, and therefore its traversal is different from other linear data structures.
- ➢ There is only one way to visit each node/element in linear data structures, i.e. starting from the first value and traversing in a linear order.



- ✓ Here, the root node is A. All the nodes on the left of A are a part of the left subtree whereas all the nodes on the right of A are a part of the right subtree. Thus, according to preorder traversal, we will first visit the root node, so A will print first and then move to the left subtree.
- ✓ B is the root node for the left subtree. So B will print next, and we will visit the left and right nodes of B. In this manner, we will traverse the whole left subtree and then move to the right subtree. Thus, the order of visiting the nodes will be **A→B→C→D→E→F→G→H→I**.

❖ **Algorithm for Preorder Traversal**

- for all nodes of the tree:
  - Step 1: Visit the root node.
  - Step 2: Traverse left subtree recursively.
  - Step 3: Traverse right subtree recursively.

❖ **Pseudo-code for Preorder Traversal**

```
void Preorder(struct node* ptr)
{
  if(ptr != NULL)
  {
    printf("%d", ptr-
    >data);Preorder(ptr-
    >left); Preorder(ptr-
    >right);
  }
```

```
}
```

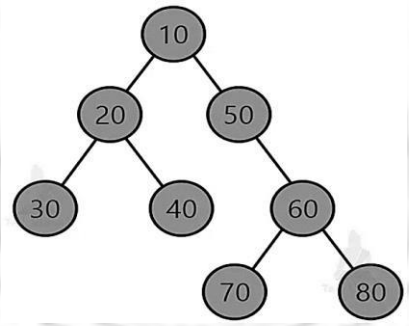
❖ **Uses of Preorder Traversal**

- If we want to create a copy of a tree, we make use of preorder traversal.
- Preorder traversal helps to give a prefix expression for the expression tree.

- **Inorder Traversal**

✓ In an inorder traversal, we first visit the left subtree, then the root node and then the right subtree in an inorder manner.

✓ Consider the following tree:



✓ In this case, as we visit the left subtree first, we get the node with the value 30 first, then 20 and then 40. After that, we will visit the root node and print it. Then comes the turn of the right subtree. We will traverse the right subtree in a similar manner. Thus, after performing the inorder traversal, the order of nodes will be **30→20→40→10→50→70→60→80.**

❖ **Algorithm for Inorder Traversal**

- for all nodes of the tree:
  - Step 1: Traverse left subtree recursively.
  - Step 2: Visit the root node.
  - Step 3: Traverse right subtree recursively.

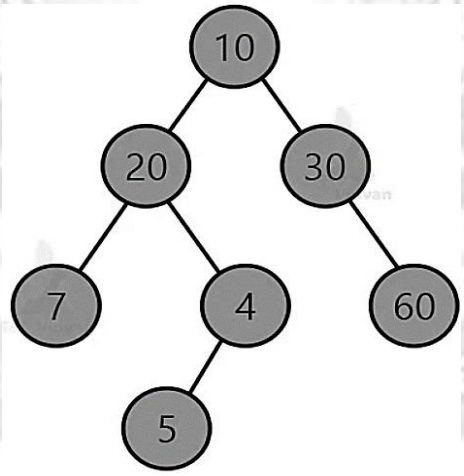
❖ **Pseudo-code for Inorder Traversal**

```
void Inorder(struct node* ptr)
{
  if(ptr != NULL)
  {
```

```
        Inorder(ptr->left);
        printf("%d", ptr->data);Inorder(ptr->right);
      }
    }
```

- ❖ **Uses of Inorder Traversal**
  - It helps to delete the tree.
  - It helps to get the postfix expression in an expression tree.

- **Postorder Traversal**
  - ✓ Postorder traversal is a kind of traversal in which we first traverse the left subtree in a postorder manner, then traverse the right subtree in a postorder manner and at the end visit the root node.
  - ✓ For example, in the following tree:



- ➢ The postorder traversal will be **7→5→4→20→60→30→10**.
  - ❖ **Algorithm for Postorder Traversal**
    - or all nodes of the tree:
      - Step 1: Traverse left subtree recursively.
      - Step 2: Traverse right subtree recursively.
      - Step 3: Visit the root node.
  - ❖ **Pseudo-code for Postorder Traversal**

```
void Postorder(struct node* ptr)
{
   if(ptr != NULL)
 {
      Postorder(ptr->left);
     Postorder(ptr->right);
     printf("%d", ptr->data);
   }
 }
```

- **Uses of Postorder Traversal**
  - It helps to delete the tree.
  - It helps to get the postfix expression in an expression tree.