Unit – 4

Trees

Content

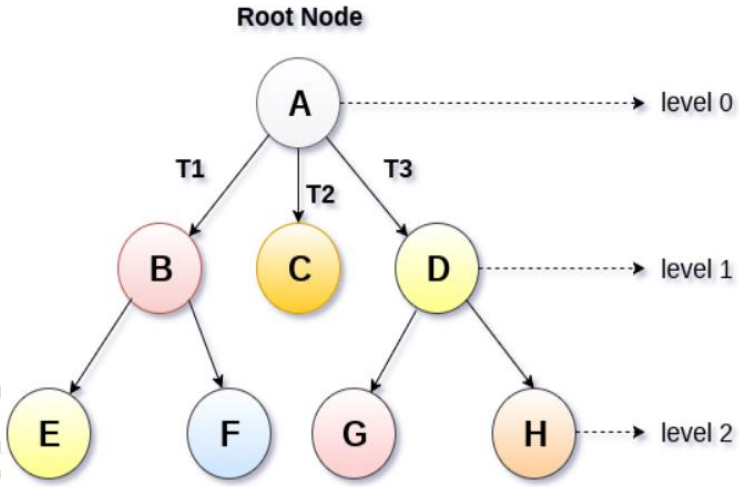
Basic Terminology in Trees, Binary Trees-Properties, Representation of Binary Trees using Arrays and Linked lists, Traversing a Binary Tree (In-Order, Pre-Order, Post-Order).

Binary Search Trees: Definition, **Operations:** Searching, Insertion, Deletion, Applications Expression Trees, Heap Sort,

Balanced Binary Trees: AVL Trees, Insertion, Deletion and Rotations.

Tree Data Structure Introduction

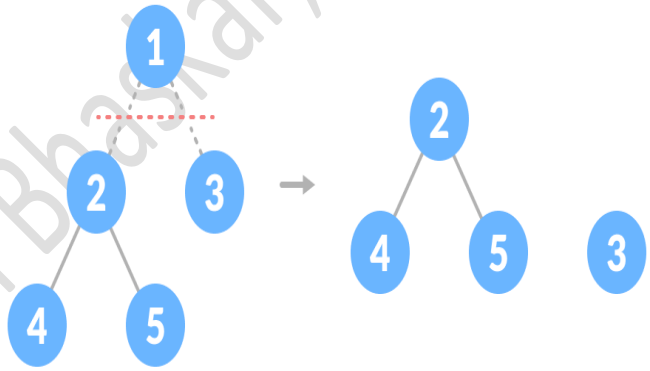
- ✓ Tree is a hierarchical data structure which stores the information naturally in the form of hierarchy style.
- ✓ Tree is one of the most powerful and advanced data structures.
- ✓ It is a non-linear data structure compared to arrays, linked lists, stack and queue.
- ✓ It represents the nodes connected by edges.
- ✓ A Tree is a recursive data structure containing the set of one or more data nodes, where one node is designated as the root of the tree while the remaining nodes are called as the children of the root.



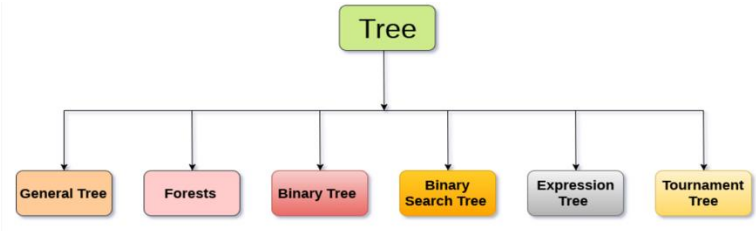
Basic terminology:

- ✓ Root Node :- The root node is the topmost node in the tree hierarchy. In other words, the root node is the one which doesn't have any parent.
- ✓ Sub Tree :- If the root node is not null, the tree T1, T2 and T3 is called sub-trees of the root node.
- ✓ Leaf Node :- The node of tree, which doesn't have any child node, is called leaf node. Leaf node is the bottom most node of the tree. There can be any number of leaf nodes present in a general tree. Leaf nodes can also be called external nodes.

- ✓ Path :- The sequence of consecutive edges is called path. In the tree shown in the above image, path to the node E is A→ B → E.
- ✓ **Ancestor node** :- An ancestor of a node is any predecessor node on a path from root to that node. The root node doesn't have any ancestors. In the tree shown in the above image, the node F have the ancestors, B and A.
- ✓ **Degree :-** Degree of a node is equal to number of children, a node have. In the tree shown in the above image, the degree of node B is 2. Degree of a leaf node is always 0 while in a complete binary tree, degree of each node is equal to 2.
- ✓ **Level Number :-** Each node of the tree is assigned a level number in such a way that each node is present at one level higher than its parent. Root node of the tree is always present at level 0.
- ✓ **Parent node** :- A parent node is immediate predecessor of a node.
- ✓ **Child node :-** All immediate successors of a nodes are its children.
- ✓ **Siblings :-** A node with same parent are called siblings.
- ✓ **Height of node :-** The height of a node is the number of edges from the node to the deepest leaf (ie. the longest path from the node to a leaf node).
- ✓ **Height of Tree :-** The height of a Tree is the height of the root node or the depth of the deepest node.
- ✓ **Edge :-** It is the link between any two nodes.
- ✓ **Forest :-** A collection of disjoint trees is called a forest.



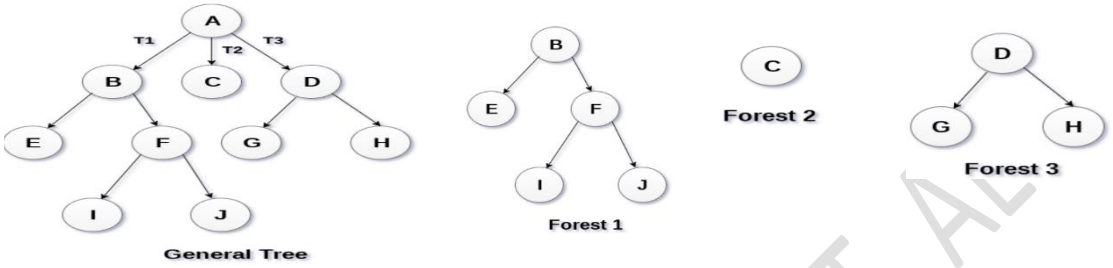
Types of Trees:



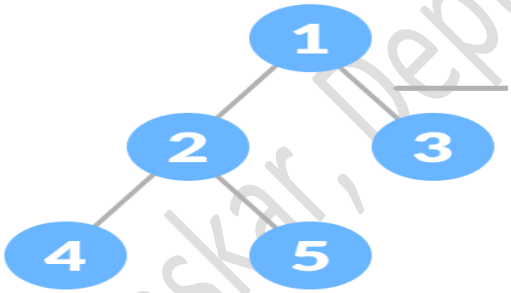
1. **General Tree:**
   - General tree stores the elements in a hierarchical order in which the top-level element is always present at level 0 as the root element.
   - All the nodes except the root node are present at number of levels.

- The nodes which are present on the same level are called siblings while the nodes which are present on the different levels exhibit the parent-child relationship among them.
- A node may contain any number of sub-trees.
- The tree in which each node contain 3 sub-tree, is called ternary tree.

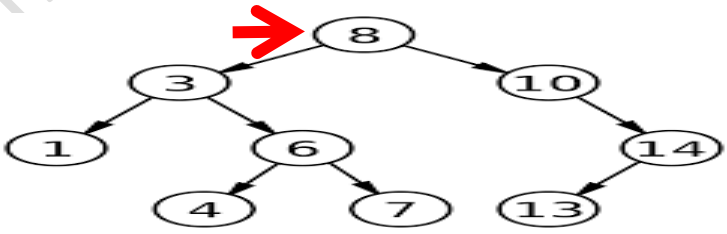
2. **Forest:**
   Forest can be defined as the set of disjoint trees which can be obtained by deleting the root node and the edges which connects root node to the first level node.



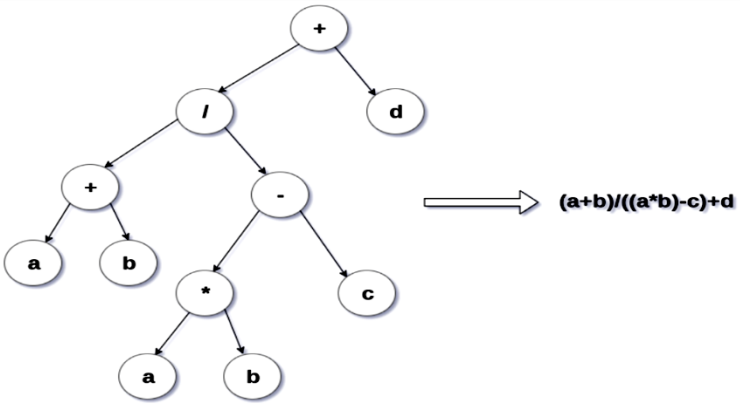
3. **Binary Tree:**
   - Binary tree is a data structure in which each node can have at most 2 children.
   - The node present at the top most level is called the root node.
   - A node with the 0 children is called leaf node.



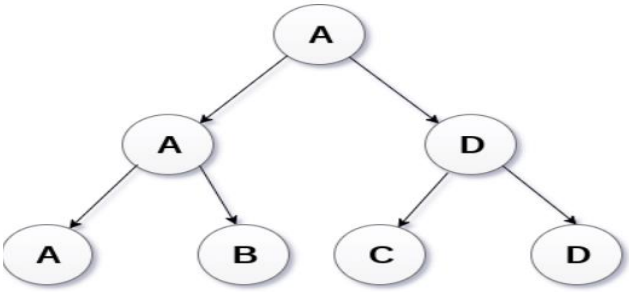
4. **Binary Search Tree :**
   - Binary search tree is an ordered binary tree.
   - All the elements in the left sub-tree are less than the root while elements present in the right sub-tree are greater than or equal to the root node element.



5. **Expression trees:**
   - Expression trees are used to evaluate the simple arithmetic expressions.
   - Expression tree is basically a binary tree where internal nodes are represented by operators while the leaf nodes are represented by operands.
   - Expression trees are widely used to solve algebraic expressions like (a+b)*(a-b).



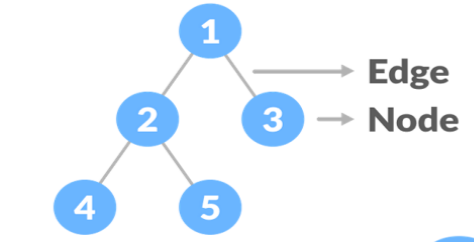
6. **Tournament tree :**
   - Tournament tree are used to record the winner of the match in each round being played between two players.
   - Tournament tree can also be called as selection tree or winner tree.
   - For example, tree of a chess tournament being played among 4 players is shown as follows.
   - However, the winner in the left sub-tree will play against the winner of right sub-tree.



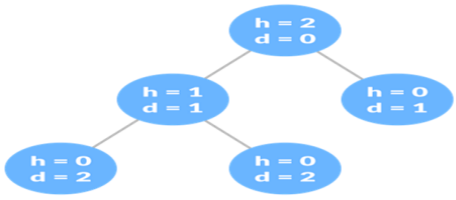
Basic terminology:

- ✓ **Root Node** :- The root node is the topmost node in the tree hierarchy. In other words, the root node is the one which doesn't have any parent.
- ✓ **Sub Tree** :- If the root node is not null, the tree T1, T2 and T3 is called sub-trees of the root node.
- ✓ **Leaf Node** :- The node of tree, which doesn't have any child node, is called leaf node. Leaf node is the bottom most node of the tree. There can be any number of leaf nodes present in a general tree. Leaf nodes can also be called external nodes.
- ✓ **Path** :- The sequence of consecutive edges is called path. In the tree shown in the above image, path to the node E is A→ B → E.
- ✓ **Ancestor node** :- An ancestor of a node is any predecessor node on a path from root to that node. The root node doesn't have any ancestors. In the tree shown in the above image, the node F have the ancestors, B and A.
- ✓ **Degree :-** Degree of a node is equal to number of children, a node have. In the tree shown in the above image, the degree of node B is 2. Degree of a leaf node is always 0 while in a complete binary tree, degree of each node is equal to 2.
- ✓ **Level Number :-** Each node of the tree is assigned a level number in such a way that each node is present at one level higher than its parent. Root node of the tree is always present at level 0.
- ✓ **Parent node** :- A parent node is immediate predecessor of a node.
- ✓ **Child node :-** All immediate successors of a nodes are its children.
- ✓ **Siblings :-** A node with same parent are called siblings.

- ✓ **Height of node :-** The height of a node is the number of edges from the node to the deepest leaf (ie. the longest path from the node to a leaf node).
- ✓ **Height of Tree :-** The height of a Tree is the height of the root node or the depth of the deepest node.
- ✓ **Edge :-** It is the link between any two nodes.
- ✓ **Forest :-** A collection of disjoint trees is called a forest.



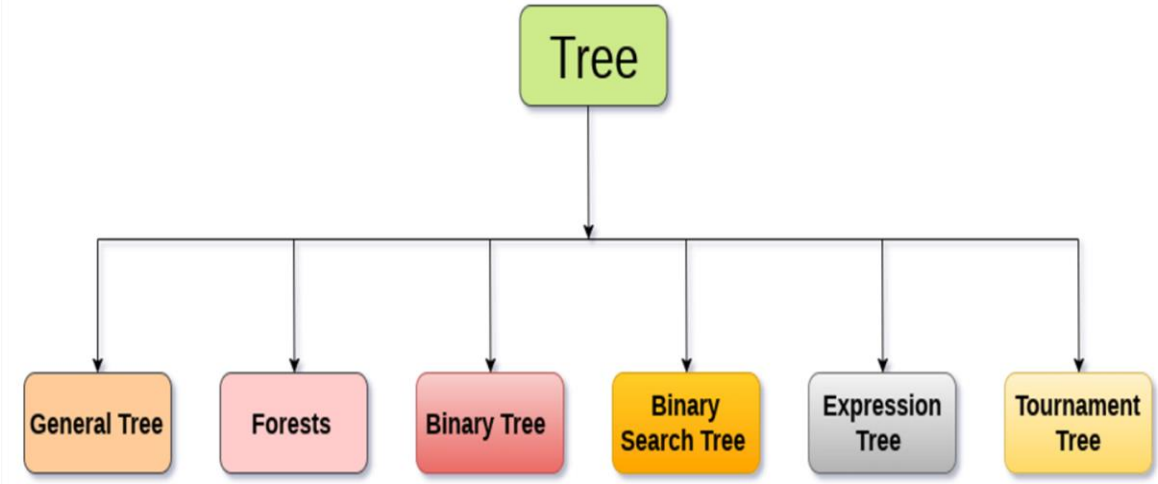
Nodes and Edges of a Tree



Height and depth of each node in a tree

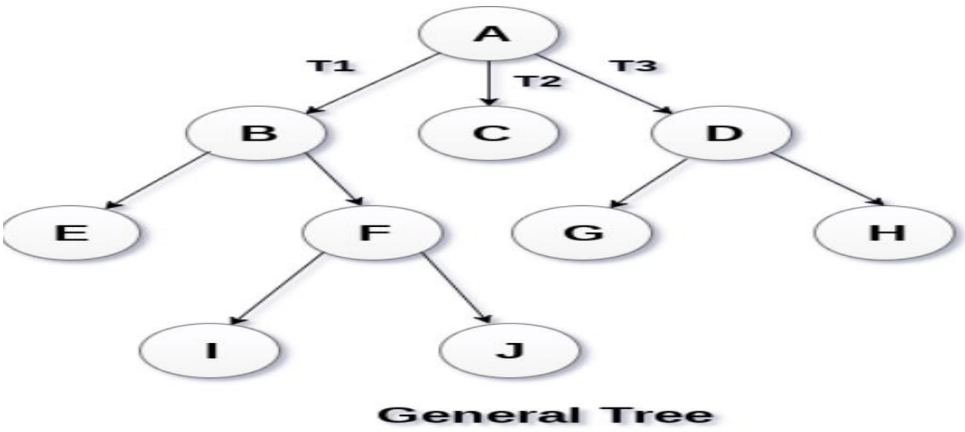
Forest

Types of Trees:



1. General Tree:

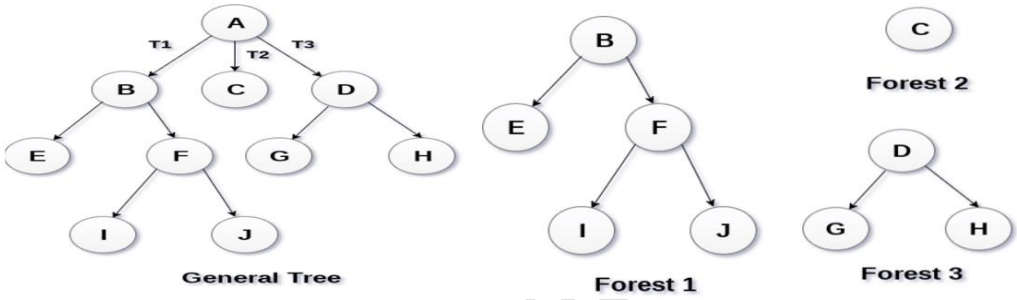
- ✓ General tree stores the elements in a hierarchical order in which the top-level element is always present at level 0 as the root element.
- ✓ All the nodes except the root node are present at number of levels.
- ✓ The nodes which are present on the same level are called siblings while the nodes which are present on the different levels exhibit the parent-child relationship among them.
- ✓ A node may contain any number of sub-trees.
- ✓ The tree in which each node contain 3 sub-tree, is called ternary tree.



General Tree

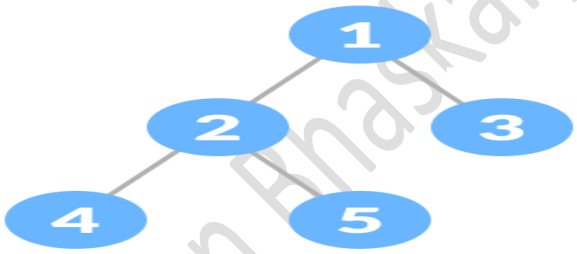
2. Forest:

✓ Forest can be defined as the set of disjoint trees which can be obtained by deleting the root node and the edges which connects root node to the first level node.



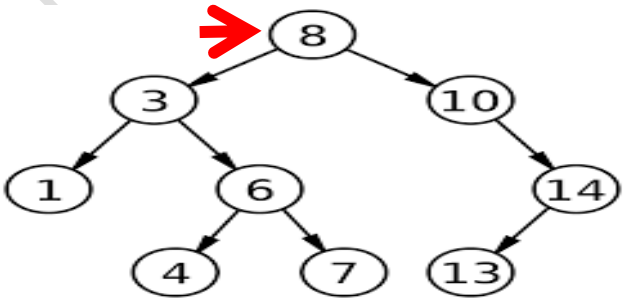
3. **Binary Tree :**

Binary tree is a data structure in which each node can have at most 2 children. The node present at the top most level is called the root node. A node with the 0 children is called leaf node.



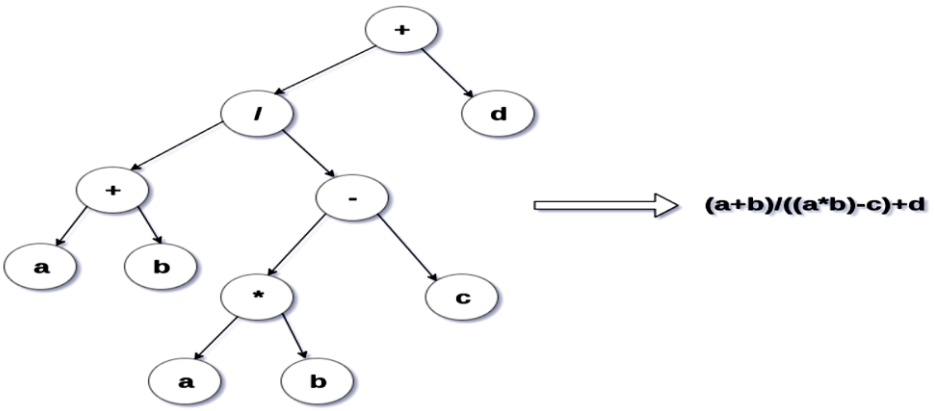
4. **Binary Search Tree :**

Binary search tree is an ordered binary tree. All the elements in the left sub-tree are less than the root while elements present in the right sub-tree are greater than or equal to the root node element.



5. **Expression trees:**

Expression trees are used to evaluate the simple arithmetic expressions. Expression tree is basically a binary tree where internal nodes are represented by operators while the leaf nodes are represented by operands. Expression trees are widely used to solve algebraic expressions like (a+b)*(a-b).

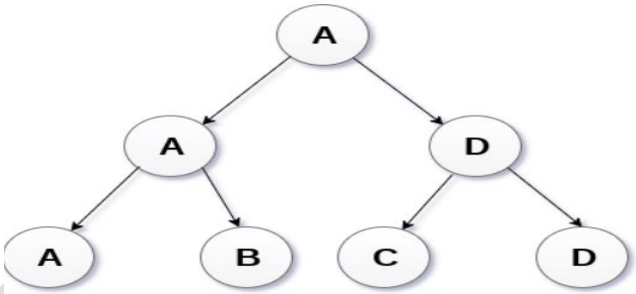


6. **Tournament tree :**

Tournament tree are used to record the winner of the match in each round being played between two players. Tournament tree can also be called as selection tree or winner tree.

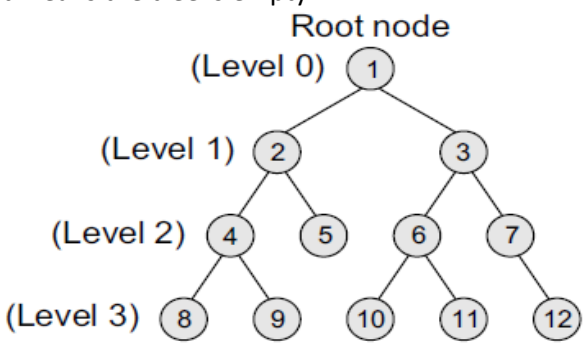
For example, tree of a chess tournament being played among 4 players is shown as follows.

However, the winner in the left sub-tree will play against the winner of right sub-tree.



2. Binary tree:

- ✓ A binary tree is a data structure that is defined as a collection of elements called nodes.
- ✓ In a binary tree, the topmost element is called the root node, and each node has 0, 1, or at the most 2 children.
- ✓ A node that has zero children is called a leaf node or a terminal node.
- ✓ Every node contains a data element, a left pointer which points to the left child, and a right pointer which points to the right child.
- ✓ The root element is pointed by a 'root' pointer.
- ✓ If root = NULL, then it means the tree is empty.



TERMINOLOGY:

Parent : If N is any node in T that has left successor S1 and right successor S2, then N is called the parent of S1 and S2. Correspondingly, S1 and S2 are called the left child and the right child of N. Every node other than the root node has a parent.

Level number: Every node in the binary tree is assigned a level number . The root node is defined to be at level 0. The left and the right child of the root node have a level number 1. Similarly, every node is at one level higher than its parents. So all child nodes are defined to have level number as parent's level number + 1.

Degree of a node: It is equal to the number of children that a node has. The degree of a leaf node is zero. For example, in the tree, degree of node 4 is 2, degree of node 5 is zero and degree of node 7 is 1.

Leaf node : A node that has no children is called a leaf node or a terminal node. The leaf nodes in the tree are: 8, 9, 5, 10, 11, and 12

Sibling: All nodes that are at the same level and share the same parent are called siblings (brothers). For example, nodes 2 and 3; nodes 4 and 5; nodes 6 and 7; nodes 8 and 9; and nodes 10 and 11 are siblings.

Edge: It is the line connecting a node N to any of its successors. A binary tree of n nodes has exactly n–1 edges because every node except the root node is connected to its parent via an edge.

Path: A sequence of consecutive edges.

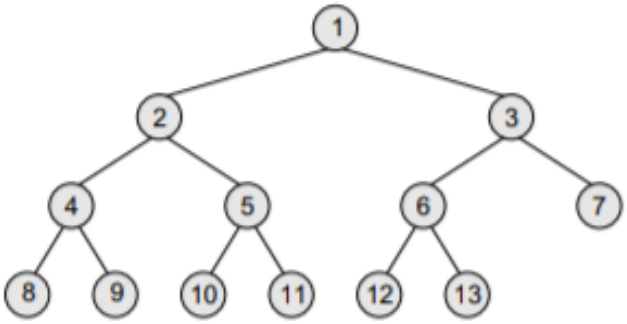
Height of a tree : It is the total number of nodes on the path from the root node to the deepest node in the tree. A tree with only a root node has a height of 1. A binary tree of height h has at least h nodes and at most 2 power h–1 nodes. This is because every level will have at least one node and can have at most 2 nodes.

In-degree/out-degree of a node :

It is the number of edges arriving at a node. The root node is the only node that has an in-degree equal to zero. Similarly, out-degree of a node is the number of edges leaving that node.

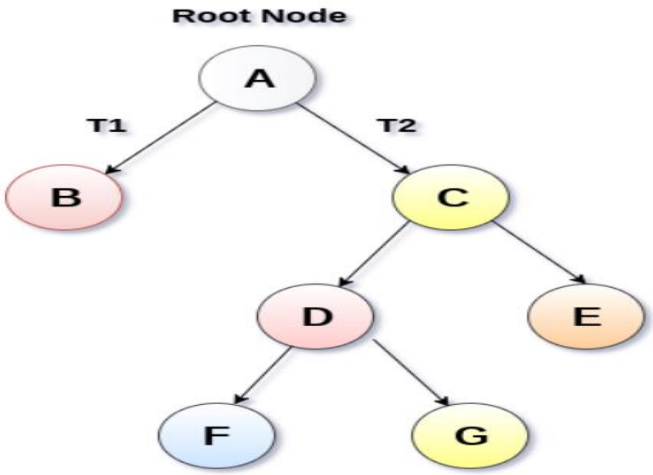
Complete Binary Tree

- ✓ A complete binary tree is a binary tree that satisfies two properties.
- ✓ First, in a complete binary tree, every level, except possibly the last, is completely filled.
- ✓ Second, all nodes appear as far left as possible.
- ✓ In a complete binary tree T_n, there are exactly n nodes and level r of T can have at most 2^r nodes. The Figure shows a complete binary tree.
- ✓ level 0 has 2 power 0 = 1 node, level 1 has 2 power1 = 2 nodes, level 2 has 2power2 = 4 nodes, level 3 has 6 nodes which is less than the maximum of 2 power 3 = 8 nodes



Strictly Binary Tree

- ✓ In Strictly Binary Tree, every non-leaf node contain non-empty left and right sub-trees.
- ✓ In other words, the degree of every non-leaf node will always be 2. A strictly binary tree with n leaves, will have (2^n - 1) nodes.



Extended Binary Tree

- ✓ A binary tree T is said to be an extended binary tree (or a 2-tree) if each node in the tree has either no child or exactly two children.
- ✓ Figure shows how an ordinary binary tree is converted into an extended binary tree.
- ✓ In an extended binary tree, nodes having two children are called internal nodes and nodes having no children are called external nodes.
- ✓ In Fig., the internal nodes are represented using circles and the external nodes are represented using squares.

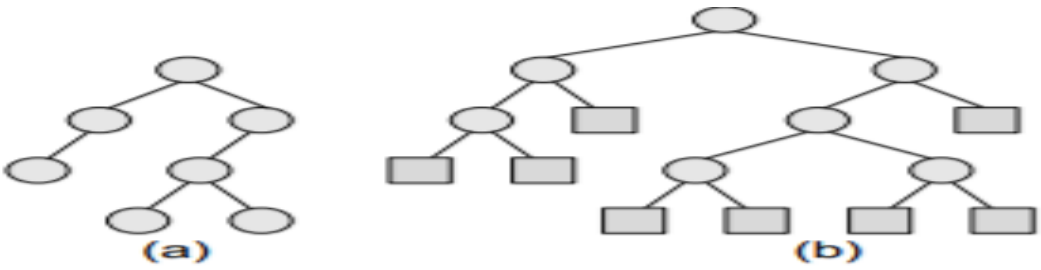


Figure 9.8 (a) Binary tree and (b) extended binary tree

Memory representation of binary tree

- ✓ In the computer's memory, a binary tree can be maintained either by using a linked representation or by using a sequential representation.
- ✓ In linked representation of a binary tree, every node will have three parts: the ***data element***, ***a pointer to the left node***, and ***a pointer to the right node***.
- ✓ So in C, the binary tree is built with a node type given below.

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

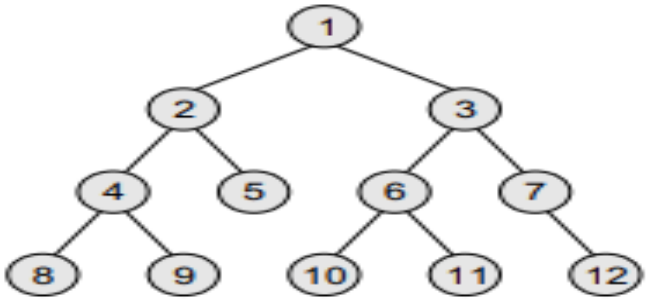


Figure 9.10 Binary tree T

		LEFT	DATA	RIGHT
ROOT 3	1	–1	8	–1
	2	–1	10	–1
	3	5	1	8
	4			
	5	9	2	14
	6			
	7			
	8	20	3	11
	9	1	4	12
	10			
	11	–1	7	18
	12	–1	9	–1
	13			
	14	–1	5	–1
15 AVAIL	15			
	16	–1	11	–1
	17			
	18	–1	12	–1
	19			
	20	2	6	16

Figure 9.11 Linked representation of binary tree T

Linked representation of binary tree

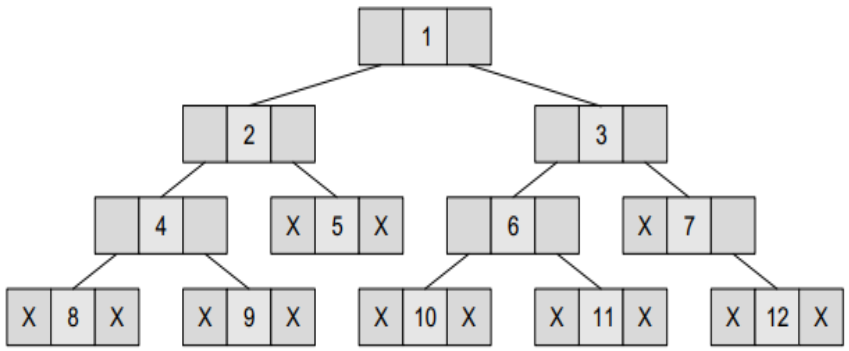


Figure 9.9 Linked representation of a binary tree

Traversing a binary tree

- ✓ Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way.
- ✓ Unlike linear data structures in which the elements are traversed sequentially, tree is a nonlinear data structure in which the elements can be traversed in many different ways.
- ✓ There are different algorithms for tree traversals.

- ✓ These algorithms differ in the order in which the nodes are visited.
- ✓ Three types of tree traversing mechanisms are there:
    a) Pre-order traversing – ParentNode – Left – Right
    b) In-Order traversing – Left – ParentNode – Right
    c) Post-order traversing – Left – Right – ParentNode

a. PRE-ORDER TRAVERSAL (NLR)

To traverse a non-empty binary tree in pre-order, the following operations are performed recursively at each node. The algorithm works by:

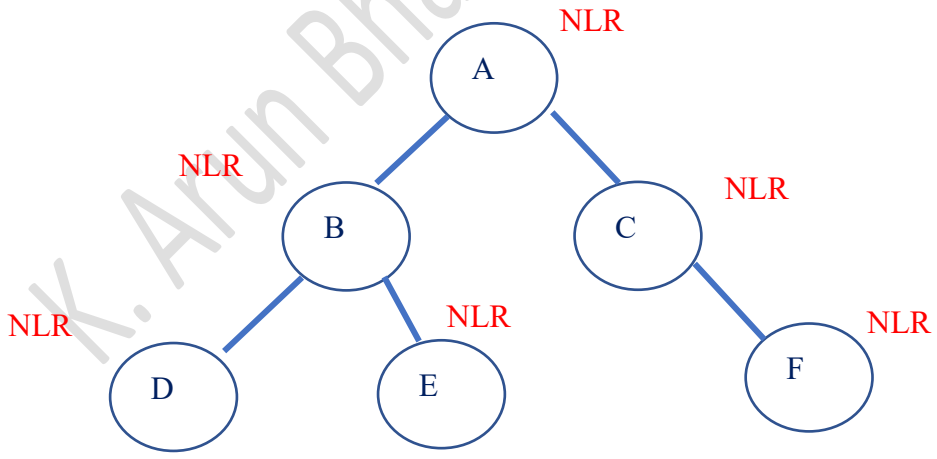
1. *Visiting the root node*
2. *Traversing the left sub-tree, and finally*
3. *Traversing the right sub-tree.*

Pre-order traversal is also called as depth-first traversal.

Pre-order traversal algorithm

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:          Write TREE -> DATA
Step 3:          PREORDER(TREE -> LEFT)
Step 4:          PREORDER(TREE -> RIGHT)
      [END OF LOOP]
Step 5: END
```

Pre-order traversal Example (NLR) Example:



Traversal Order is → A B D E C F

b. In-order Traversal (LNR)

To traverse a non-empty binary tree in in-order, the following operations are performed recursively at each node. The algorithm works by:

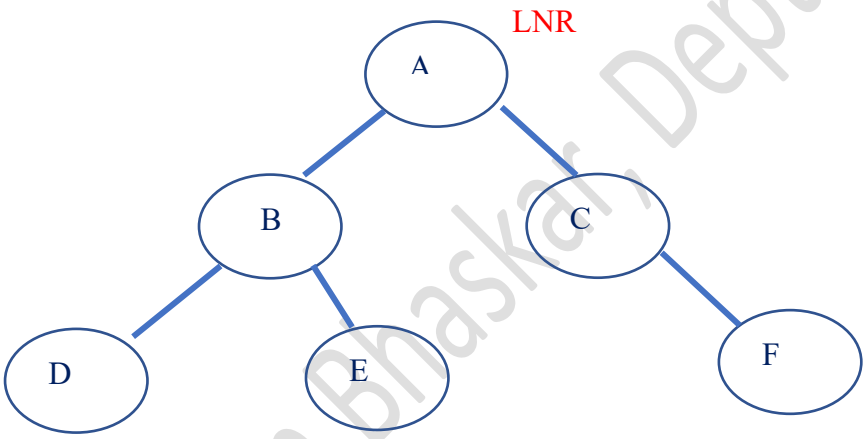
1. ***Traversing the left sub-tree***
2. ***Visiting the root node, and finally***
3. ***Traversing the right sub-tree.***

In-order traversal is also called as symmetric traversal

In-order Traversal (LNR) algorithm

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:          INORDER(TREE -> LEFT)
Step 3:          Write TREE -> DATA
Step 4:          INORDER(TREE -> RIGHT)
      [END OF LOOP]
Step 5: END
```

Example:



Traversal Order is → D B E A C F

c. Post-Order Traversal (LRN)

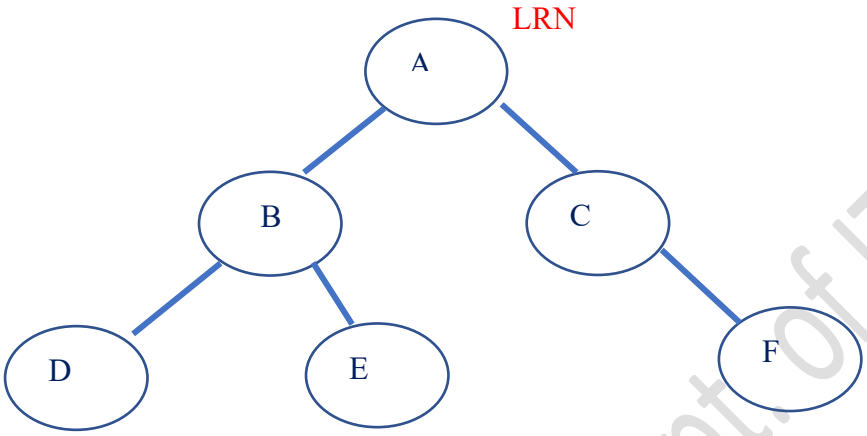
To traverse a non-empty binary tree in post-order, the following operations are performed recursively at each node. The algorithm works by:

1. ***Traversing the left sub-tree***
2. ***Traversing the right sub-tree***
3. ***Visiting the root node.***

Algorithm :

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:          POSTORDER(TREE -> LEFT)
Step 3:          POSTORDER(TREE -> RIGHT)
Step 4:          Write TREE -> DATA
      [END OF LOOP]
Step 5: END
```

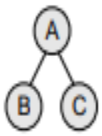
Example:



Traversal Order is → D E B F C A

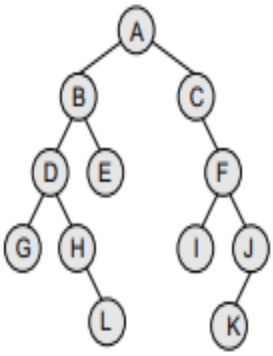
d. Level-order Traversal

- ✓ In level-order traversal, all the nodes at a level are accessed before going to the next level.
- ✓ This algorithm is also called as the breadth-first traversal algorithm.



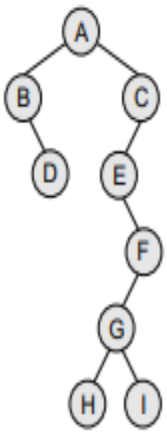
(a)

TRAVERSAL ORDER:
A, B, and C



(b)

TRAVERSAL ORDER:
A, B, C, D, E, F, G, H, I, J, L, and K



(c)

TRAVERSAL ORDER:
A, B, C, D, E, F, G, H, and I

Constructing a Binary Tree from Traversal Results

- ✓ We can construct a binary tree if we are given at least two traversal results.
- ✓ The first traversal must be the in-order traversal and the second can be either pre-order or post-order traversal.
- ✓ The in-order traversal result will be used to determine the left and the right child nodes, and the pre-order/post-order can be used to determine the root node.
- ✓ For example, consider the traversal results given below:

  In–order Traversal: D B E A F C G

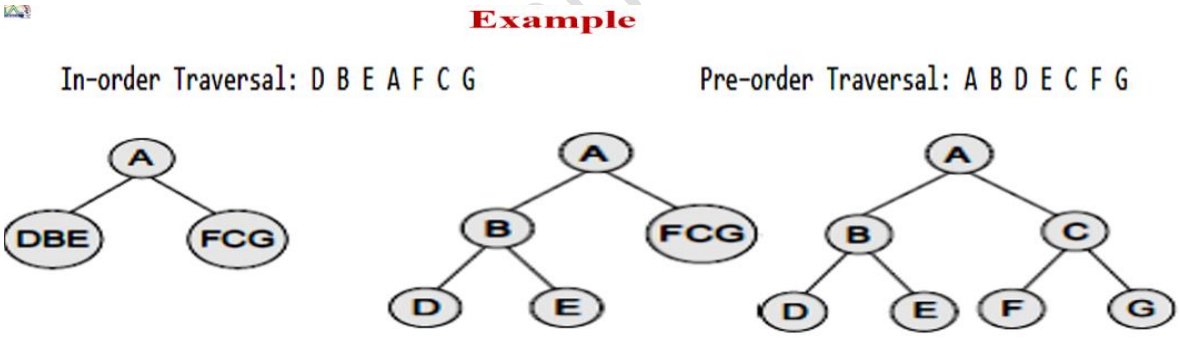
  Pre–order Traversal: A B D E C F G

- ✓ We have the in-order traversal sequence and pre-order traversal sequence.
- ✓ Follow the steps given below to construct the tree:

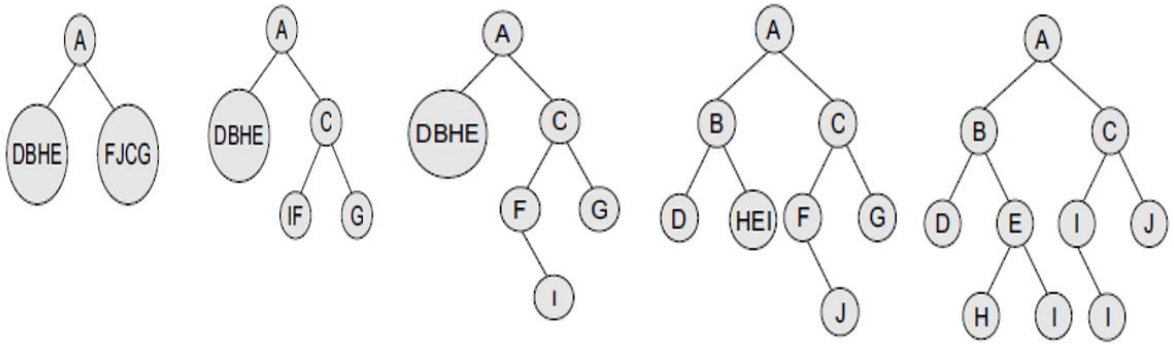
Step 1: Use the pre-order sequence to determine the root node of the tree. The first element would be the root node.

Step 2: Elements on the left side of the root node in the in-order traversal sequence form the left sub-tree of the root node. Similarly, elements on the right side of the root node in the in-order traversal sequence form the right sub-tree of the root node.

Step 3: Recursively select each element from pre-order traversal sequence and create its left and right sub-trees from the in-order traversal sequence



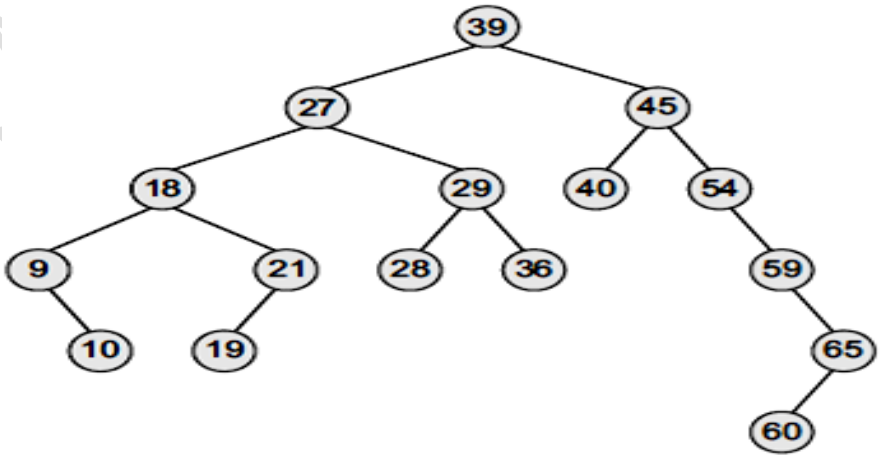
In–order Traversal: D B H E I A F J C G Post order Traversal: D H I E B J F G C A



Binary Search Tree:

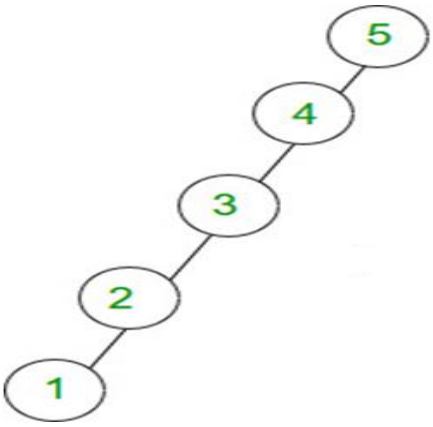
- ✓ A binary search tree, also known as an ordered binary tree, it is a variant of binary trees in which the nodes are arranged in an order.
- ✓ In a binary search tree, all the nodes in the left sub-tree have a value less than that of the root node.
- ✓ Correspondingly, all the nodes in the right sub-tree have a value either equal to or greater than the root node.
- ✓ The same rule is applicable to every sub-tree in the tree.
- ✓ Binary search trees also speed up the insertion and deletion operations.
- ✓ The tree has a speed advantage when the data in the structure changes rapidly.
- ✓ Binary search trees are considered to be efficient data structures especially when compared with sorted linear arrays and linked lists.
- ✓ Since the nodes in a binary search tree are ordered, the time needed to search an element in the tree is greatly reduced.

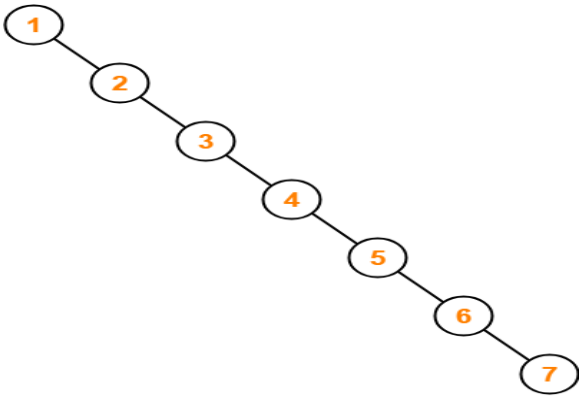
Example:



Binary Search Tree (BST) example

Skewed BST





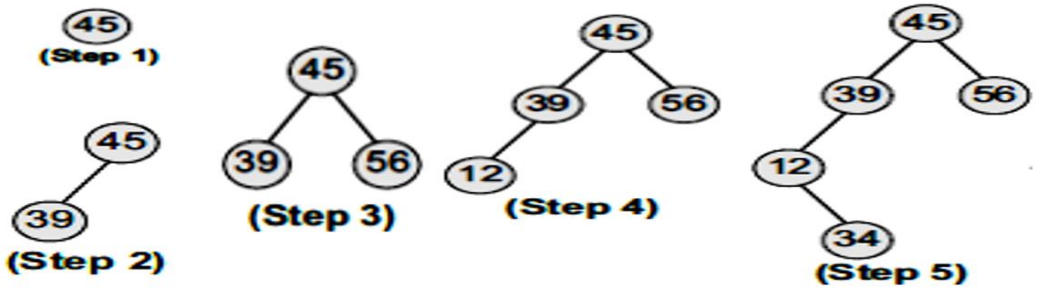
Left Skewed BST **Right Skewed BST**

- ✓ To summarize, a binary search tree is a binary tree with the following properties:
- ➢ *The left sub-tree of a node N contains values that are less than N's value.*
- ➢ *The right sub-tree of a node N contains values that are greater than N's value.*
- ➢ *Both the left and the right binary trees also satisfy these properties and, thus, are binary search trees.*

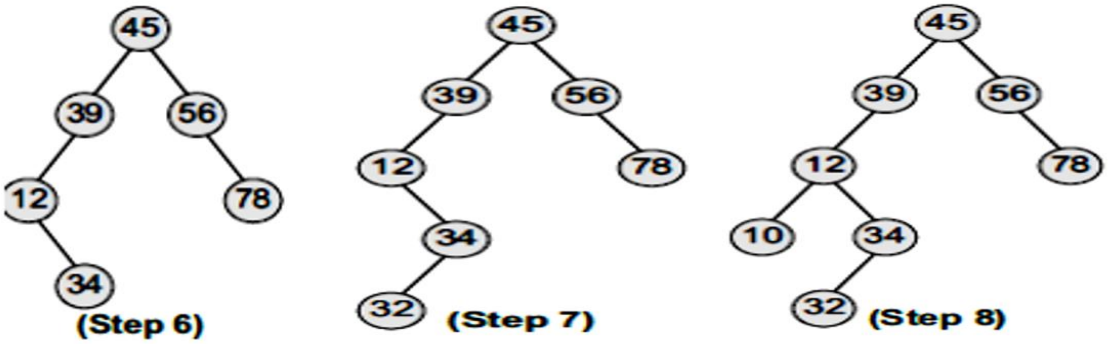
Create a binary search tree :

Create a binary search tree using the following data elements

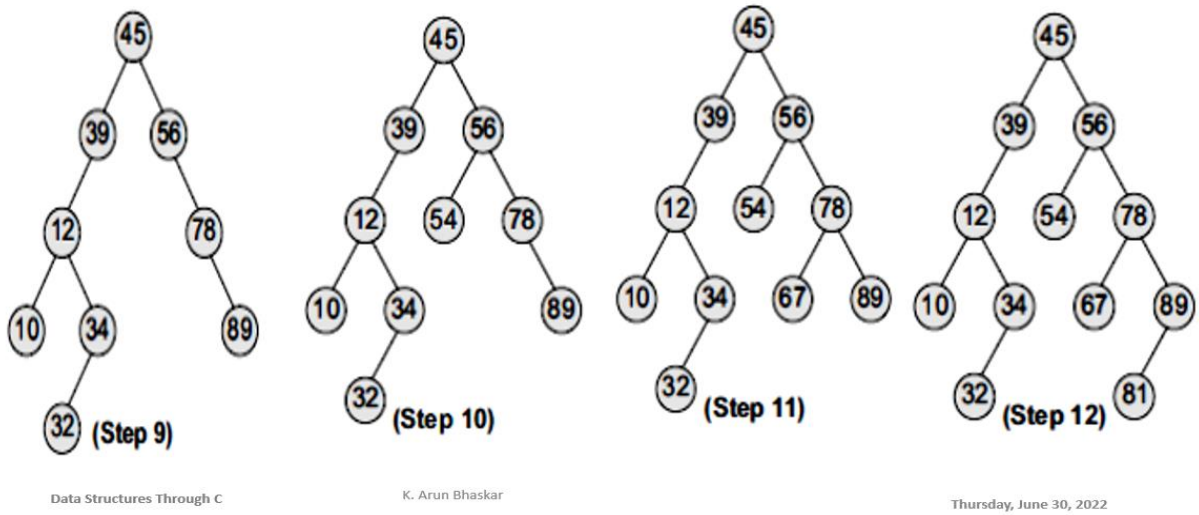
45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81



Data Structures Through C K. Arun Bhaskar Thursday, June 30, 2022



Data Structures Through C K. Arun Bhaskar Thursday, June 30, 2022



Operations on BST

- ***Searching for a Node in a Binary Search Tree***
- ***Inserting a New Node in a Binary Search Tree***
- ***Deleting a Node from a Binary Search Tree***

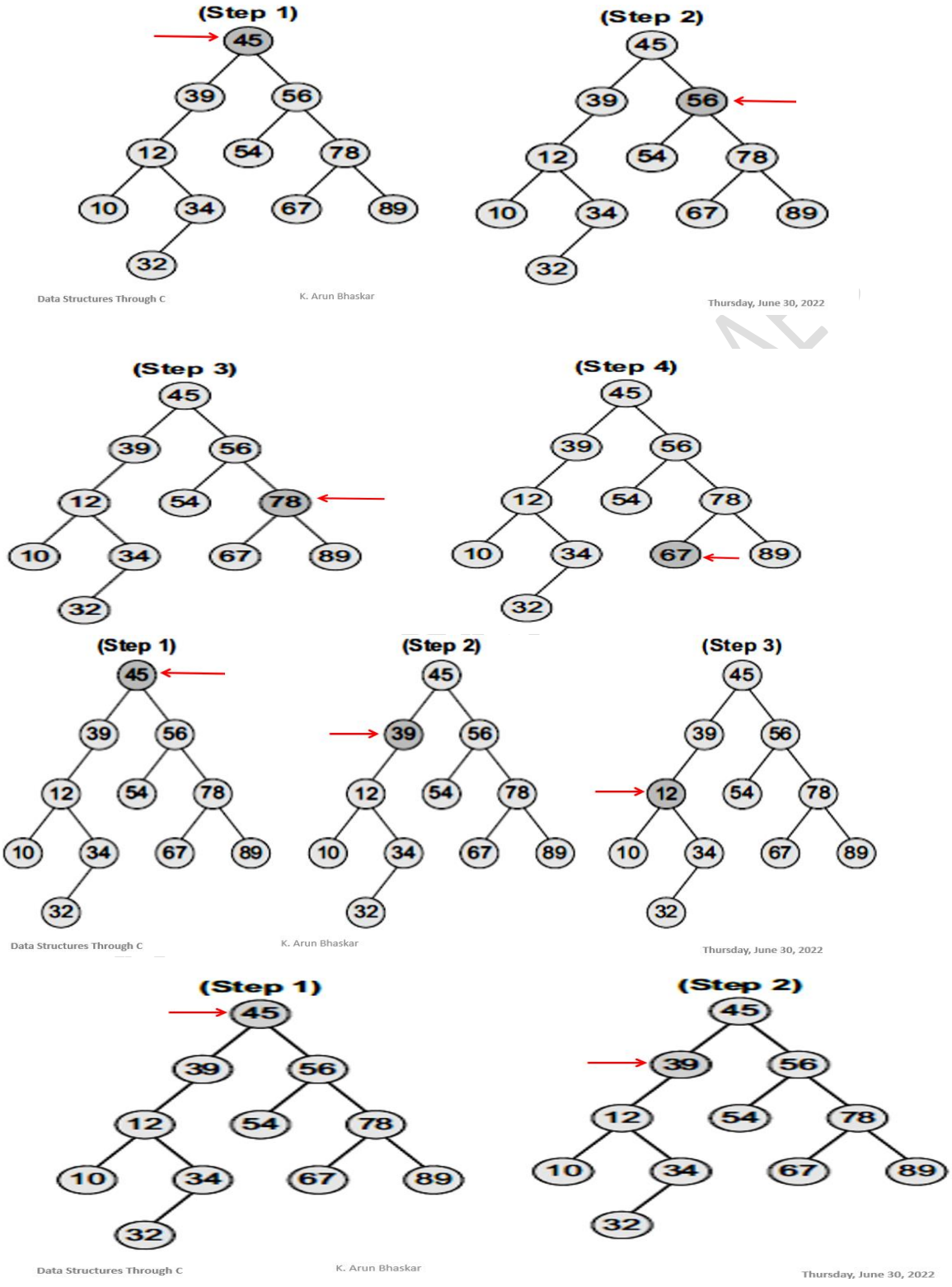
A. Searching for a Node in a Binary Search Tree:

- ✓ The searching process begins at the root node.
- ✓ The function first checks if the binary search tree is empty.
- ✓ If it is empty, then the value we are searching for is not present in the tree.
- ✓ So, the search algorithm terminates by displaying an appropriate message.
- ✓ However, if there are nodes in the tree, then the search function checks to see if the key value of the current node is equal to the value to be searched.
- ✓ If not, it checks if the value to be searched for is less than the value of the current node, in which case it should be recursively called on the left child node.
- ✓ In case the value is greater than the value of the current node, it should be recursively called on the right child node.

```
searchElement (TREE, VAL)

Step 1: IF TREE -> DATA = VAL OR TREE = NULL
          Return TREE
        ELSE
         IF VAL < TREE -> DATA
          Return searchElement(TREE -> LEFT, VAL)
         ELSE
          Return searchElement(TREE -> RIGHT, VAL)
         [END OF IF]
        [END OF IF]
Step 2: END
```

Search for a given value in a BST example

Searching a node with value 67 in the given binary search tree



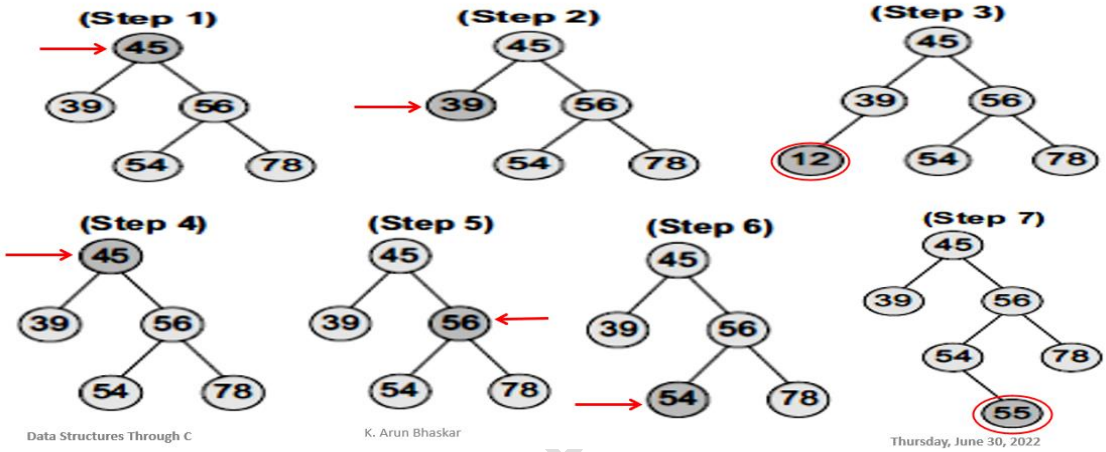
B. **Inserting a New Node in a Binary Search Tree:**

- ✓ The insert function is used to add a new node with a given value at the correct position in the binary search tree.
- ✓ Adding the node at the correct position means that the new node should not violate the properties of the binary search tree.

```
Insert (TREE, VAL)

Step 1: IF TREE = NULL
            Allocate memory for TREE
            SET TREE -> DATA = VAL
            SET TREE -> LEFT = TREE -> RIGHT = NULL
        ELSE
            IF VAL < TREE -> DATA
                  Insert(TREE -> LEFT, VAL)
            ELSE
                  Insert(TREE -> RIGHT, VAL)
            [END OF IF]
        [END OF IF]
Step 2: END
```

Inserting a New Node in a BST example

Insertion of given values in a given BST : Values are 12 and 55



C. **Deleting a Node from a Binary Search Tree:**

- ✓ The delete function deletes a node from the binary search tree.
- ✓ However, at most care should be taken that the properties of the binary search tree are not violated and nodes are not lost in the process.
- ✓ There are 3 cases :

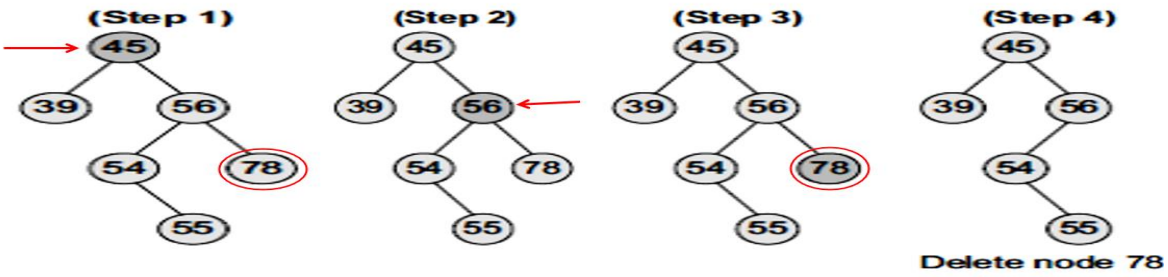
1) Deleting a Node that has No Children

2) Deleting a Node with One Child

3) Deleting a Node with Two Children

Case 1: Deleting a Node that has No Children

- ✓ If we have to delete node 78, we can simply remove this node without any issue. This is the simplest case of deletion.

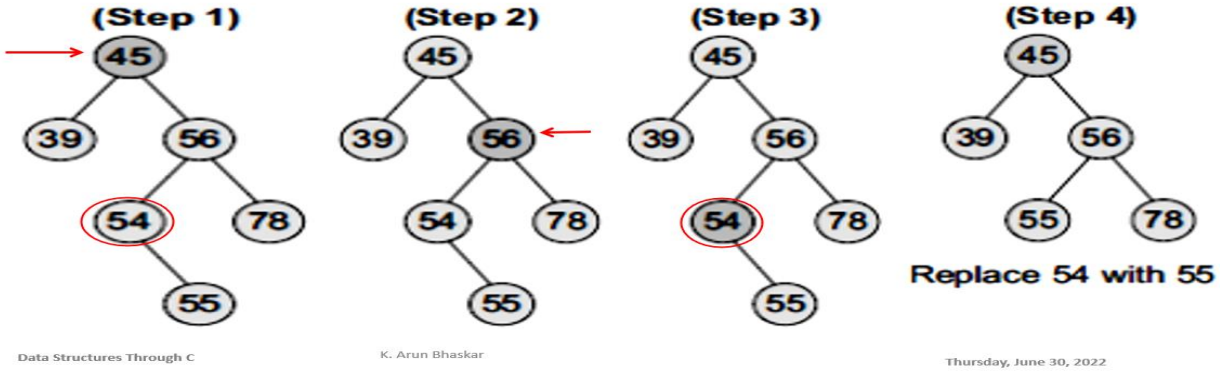


Case 2: Deleting a Node with One Child

- ✓ To handle this case, the node's child is set as the child of the node's parent.
- ✓ In other words, replace the node with its child.

- ✓ Now, if the node is the left child of its parent, the node's child becomes the left child of the node's parent.
- ✓ Correspondingly, if the node is the right child of its parent, the node's child becomes the right child of the node's parent.

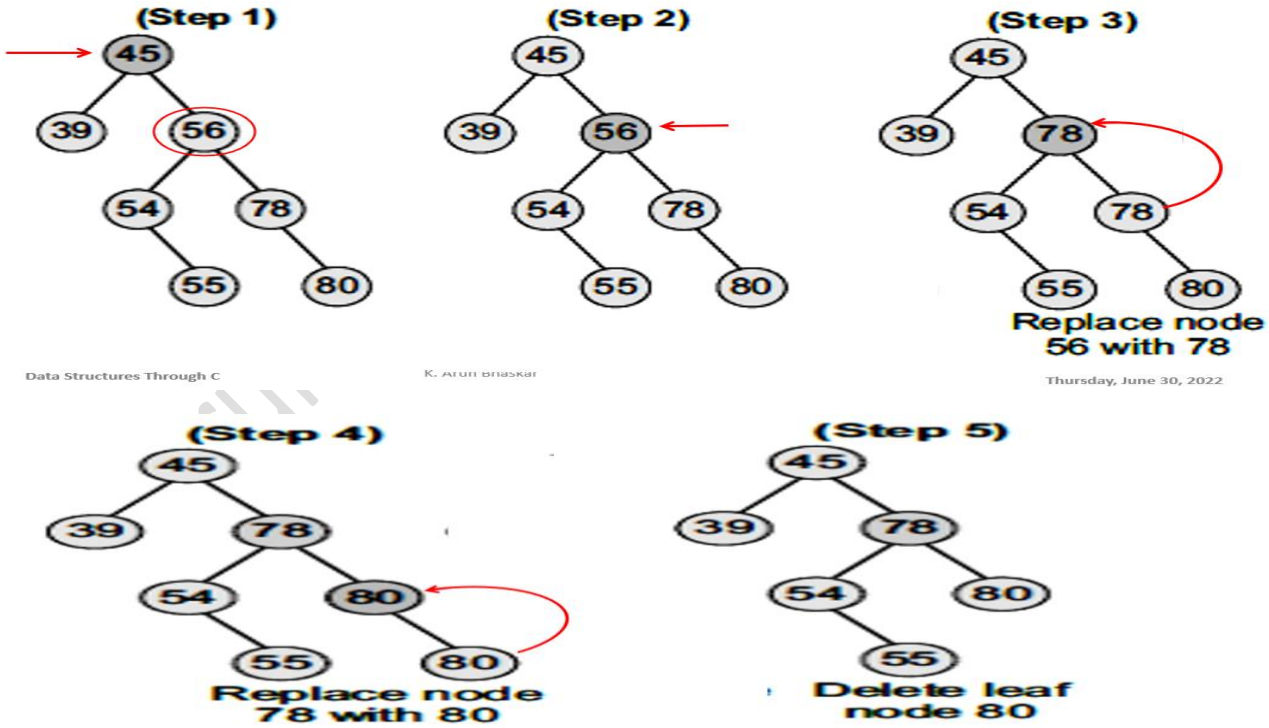
Deleting node 54 from the Binary Search Tree



Case 3: Deleting a Node with Two Children

- ✓ To handle this case, replace the node's value with its *in-order* ***predecessor*** (largest value in the left sub-tree) or *in-order successor* (smallest value in the right sub-tree).
- ✓ The in-order predecessor or the successor can then be deleted using any of the above cases.

Deleting node 56 from the Binary Search Tree



Algorithm to deleting a node from a Binary Search Tree

```
Delete (TREE, VAL)

Step 1: IF TREE = NULL
          Write "VAL not found in the tree"
        ELSE IF VAL < TREE -> DATA
          Delete(TREE->LEFT, VAL)
        ELSE IF VAL > TREE -> DATA
          Delete(TREE -> RIGHT, VAL)
        ELSE IF TREE -> LEFT AND TREE -> RIGHT
          SET TEMP = findLargestNode(TREE -> LEFT)
          SET TREE -> DATA = TEMP -> DATA
          Delete(TREE -> LEFT, TEMP -> DATA)
        ELSE
          SET TEMP = TREE
          IF TREE -> LEFT = NULL AND TREE -> RIGHT = NULL
              SET TREE = NULL
          ELSE IF TREE -> LEFT != NULL
              SET TREE = TREE -> LEFT
          ELSE
              SET TREE = TREE -> RIGHT
          [END OF IF]
          FREE TEMP
        [END OF IF]
Step 2: END
```

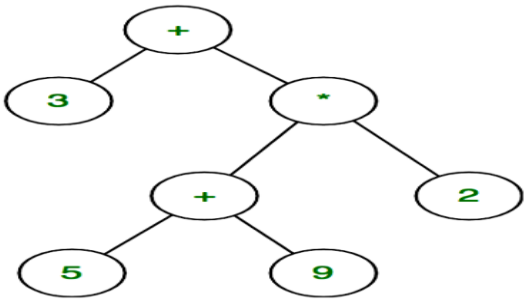
BST Applications

1. *3D game designing*
2. *Compilers designing*

Expression Tree:

The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand.

Example expression tree for **3 + ((5+9)*2)** would be:



- ✓ There are different types of expression formats:
  - ➢ *Prefix expression*
  - ➢ *Infix expression and*
  - ➢ *Postfix expression*
- ✓ Expression Tree is a special kind of binary tree with the following properties:
  - ➢ Each leaf is an operand. Examples: a, b, c, 6, 100
  - ➢ The root and internal nodes are operators. Examples: +, -, *, /, ^
  - ➢ Sub-trees are sub-expressions with the root being an operator.

Traversal Techniques

- ✓ There are three standard traversal techniques to represent the three different expression formats.

a. In-order Traversal :

We can produce an infix expression by recursively printing out

- ✓ *Left expression,*
- ✓ *The root, and*
- ✓ *Right expression*.

b. Post-order Traversal :

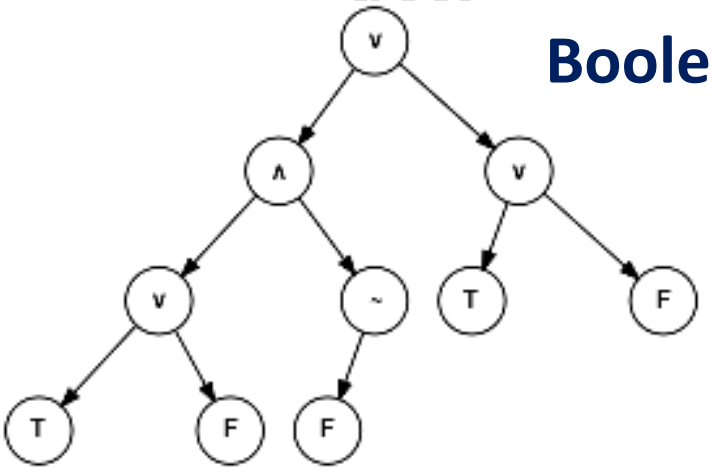
The postfix expression can be evaluated by recursively printing out

- ✓ *Left expression,*
- ✓ *Right expression and*
- ✓ *The root*

c. Pre-order Traversal :

We can also evaluate prefix expression by recursively printing out:

- ✓ *Root,*
- ✓ *Left expression and*
- ✓ *Right expression*.



Boolean Expression Tree

((T V F) ^ ~ F) V (T V F)

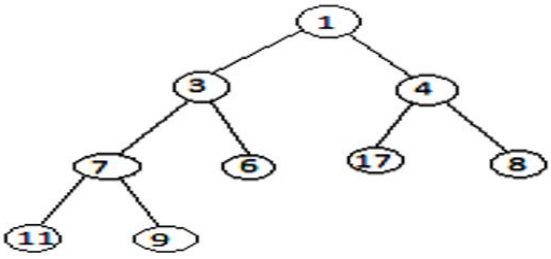
Expression Tree Applications

1. *Storing algebraic expression (or) Boolean expression data*
2. *Evaluation of algebraic expressions.*
3. *Evaluation of Boolean expressions.*
4. *Conversion of expression from one form to other, like in-fix to post (or) pre fix notation.*

Heap Sort

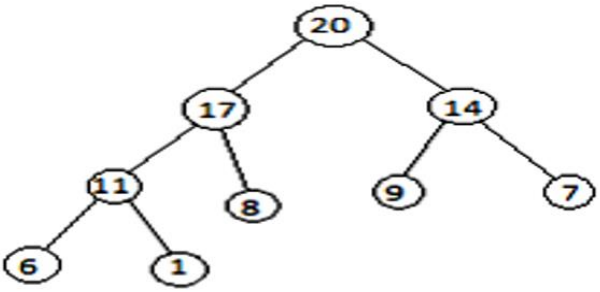
- ✓ A Heap is a Complete Binary Tree.
- ✓ A sorting algorithm that works by first organizing the data to be sorted into a special type of Binary Tree called a **heap**.
- ✓ The **heap** itself has, by definition, the largest value at the top of the tree.
- ✓ A Binary Heap is a Complete Binary Tree where items are stored in a special order such that value in a parent node is greater (or smaller) than the values in its two children nodes.
- ✓ So it is called as max heap and the latter is called min heap.
- ✓ The heap can be represented by binary tree or array.

Heap Sort Cont…



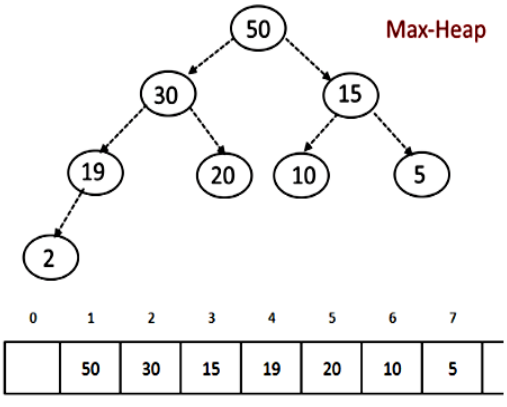
Min-Heap

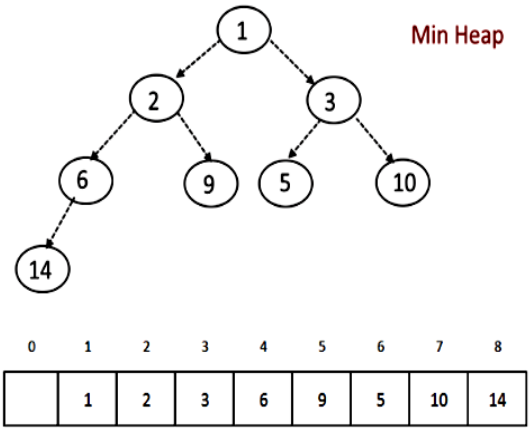
In min-heap, first element is the smallest. So when we want to sort a list in ascending order, we create a Min-heap from that list, and picks the first element, as it is the smallest, then we repeat the process with remaining elements.



Max-Heap

In max-heap, the first element is the largest, hence it is used when we need to sort a list in descending order.





Heap Sort Algorithm

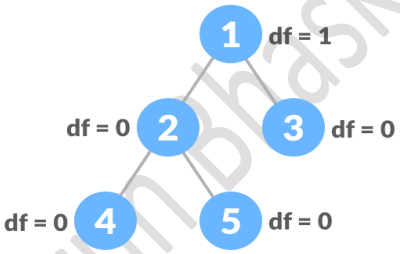
```
HEAPSORT(ARR, N)

Step 1: [Build Heap H]
      Repeat for I = 0 to N-1
            CALL Insert_Heap(ARR, N, ARR[I])
      [END OF LOOP]
Step 2: (Repeatedly delete the root element)
      Repeat while N>0
            CALL Delete_Heap(ARR, N, VAL)
            SET N = N + 1
      [END OF LOOP]
Step 3: END
```

Balanced Binary Tree

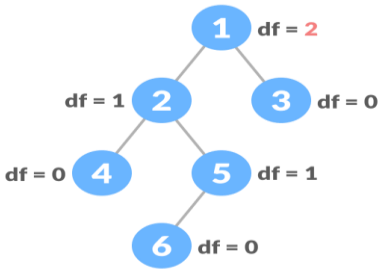
- ✓ A balanced binary tree, also referred to as a height-balanced binary tree, is defined as a binary tree in which the height of the left and right subtree of any node differ by not more than one.
- ✓ Following are the conditions for a height-balanced binary tree:

1. Difference between the left and the right subtree for any node is not more than one
2. The left subtree is balanced
3. The right subtree is balanced

Balanced Binary Tree with depth at each level



Unbalanced Binary Tree



Types of Self-Balancing Binary Search Trees :

Given below are a few types of BSTs that are self-balancing.

1. AVL trees
2. Red-black trees
3. Splay trees
4. Treaps

AVL Tree Introduction:

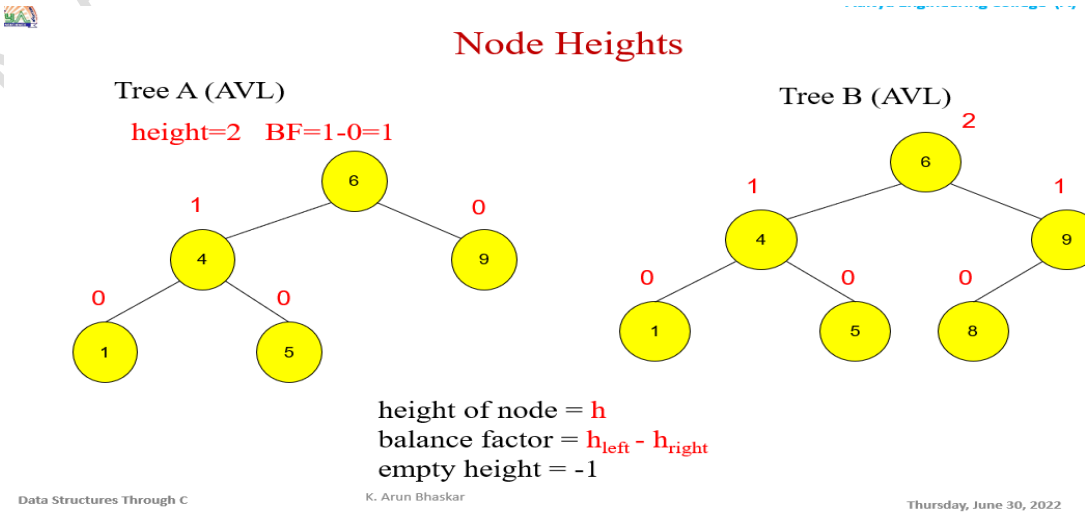
- ✓ AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.

- ✓ The technique of balancing the height of binary trees was developed by Adelson, Velskii, and Landi and hence given the short form as AVL tree or Balanced Binary Tree.
- ✓ The structure of AVL Tree is the same as that of a BST, with a little difference , It stores additional variable called “Balance Factor”.
- ✓ All BST operations are O(d), where d is tree depth
- ✓ Minimum d is d = ($\log_2$ N) for a binary tree with N nodes.
- ✓ Best case running time of BST operations is O(log N) and Worst case running time is O(N).

- ✓ Worst case running time is O(N) in BST
  - › What happens when you Insert elements in ascending order?
    - Insert: 2, 4, 6, 8, 10, 12 into an empty BST
  - › Problem: Lack of “balance”:
    - compare depths of left and right subtree
  - › Unbalanced degenerate tree

Height of an AVL Tree :

- N(h) = min. no. of nodes in an AVL tree of height h.
- Basis N(0) = 1, N(1) = 2 ..
- So, N(h) = N(h-1) + N(h-2) + 1
- Every node has a balance factor associated with it.
- Balance Factor of a node is calculated by subtracting the height of its right sub-tree from, the height of the Left sub-tree.
- BF = h_{left}-h_{right}
- (Balance factor = heightOfLeftSubtree – heightOfRightSubtree)
- A BST in which every node has a balance factor of -1,0, or 1.
- |BF |= | h_{left}-h_{right} | <= 1



Operations on AVL Tree

- ***Searching for a Node in a AVL Tree***
- ***Inserting a New Node in a AVL Tree***
- ***Deleting a Node from a AVL Tree***

Search operation in AVL Tree:

- ✓ Searching in an AVL tree is performed exactly same way as it is performed in the BST.
- ✓ Due to the height balancing of the tree, the search operation takes O(log n) time to complete.

Search algorithm in AVL Tree:

Step 1 - Read the search element.

Step 2 - Compare the search element with the value of root node in the tree.

Step 3-If both are matched, then display "Given node is found!!!" and terminate the function

Step 4 - If both are not matched, then check whether search element is smaller or larger than that node value.

Step 5 - If search element is smaller, then continue the search process in left subtree.

Step 6 - If search element is larger, then continue the search process in right subtree.

Step 7 - Repeat the same until we find the exact element or until the element is compared with the leaf node.

Step 8 - If we reach to the node having the value equal to the search value, then display "Element is found" and terminate the function.

Step 9 - If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.

Rotation operation in AVL Tree:

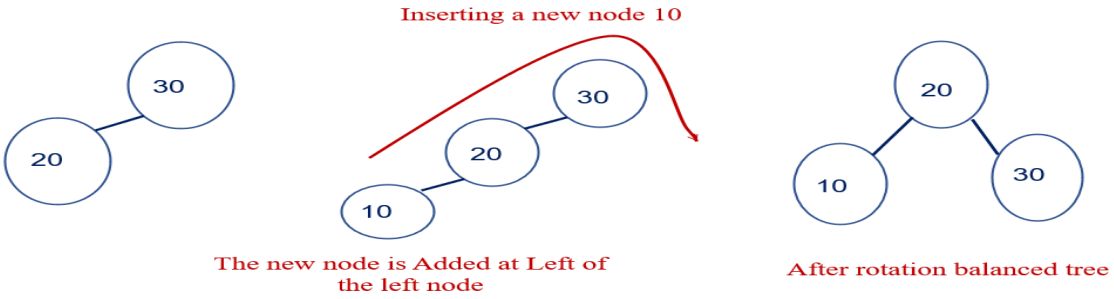
AVL Rotations :

To balance itself, an AVL tree may perform the following four kinds of rotations

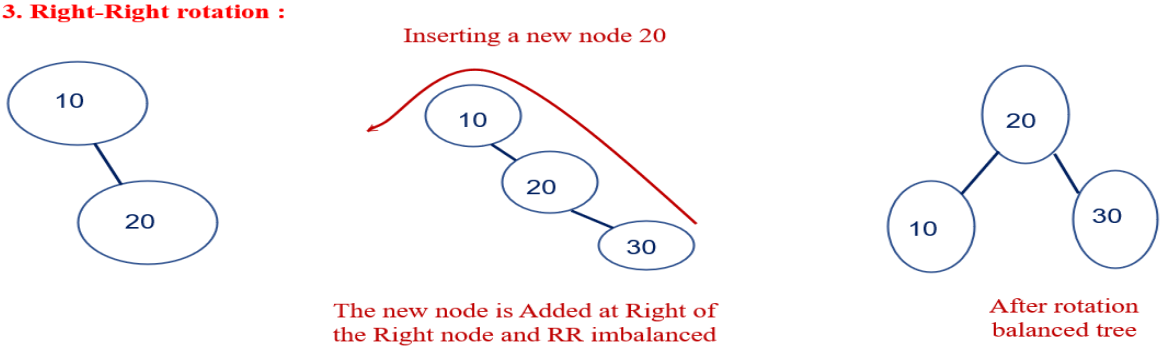
- ✓ *Left-Left rotation (or) Left rotation*
- ✓ *Right-Right rotation (or) Right rotation*
- ✓ *Left-Right rotation*
- ✓ *Right-Left rotation*

Rotation operation in AVL Tree Cont...

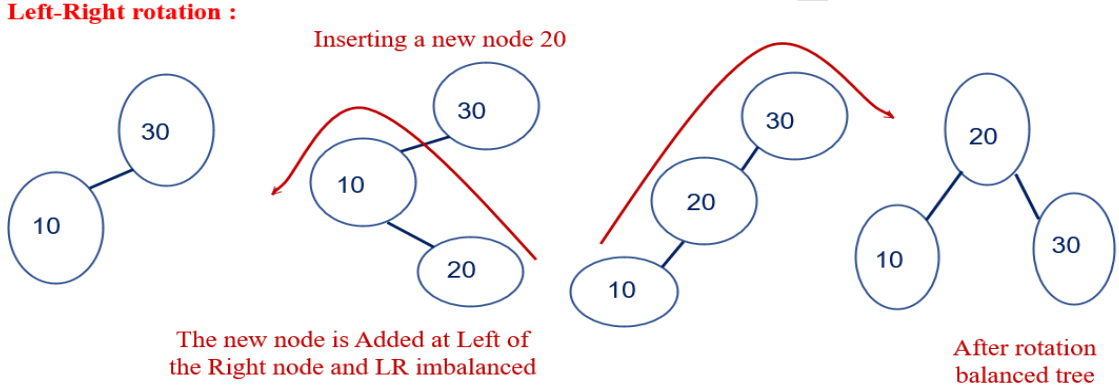
1. Left-Left rotation :



3. Right-Right rotation :

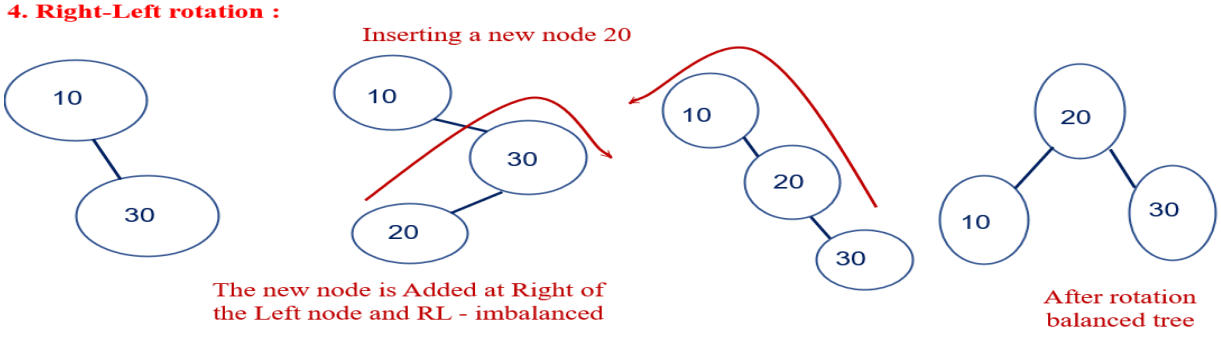


3. Left-Right rotation :



Two stage rotation need to execute

4. Right-Left rotation :



Two stage rotation need to execute

Insert operation in AVL Tree

- ✓ Insert in an AVL tree is performed exactly same way as it is performed in the BST.
- ✓ In the AVL tree the new node is always inserted as leaf node.
- ✓ But the step of insertion is usually followed by an additional step of rotational.

- ✓ Rotation is done to restore the balance of the tree.

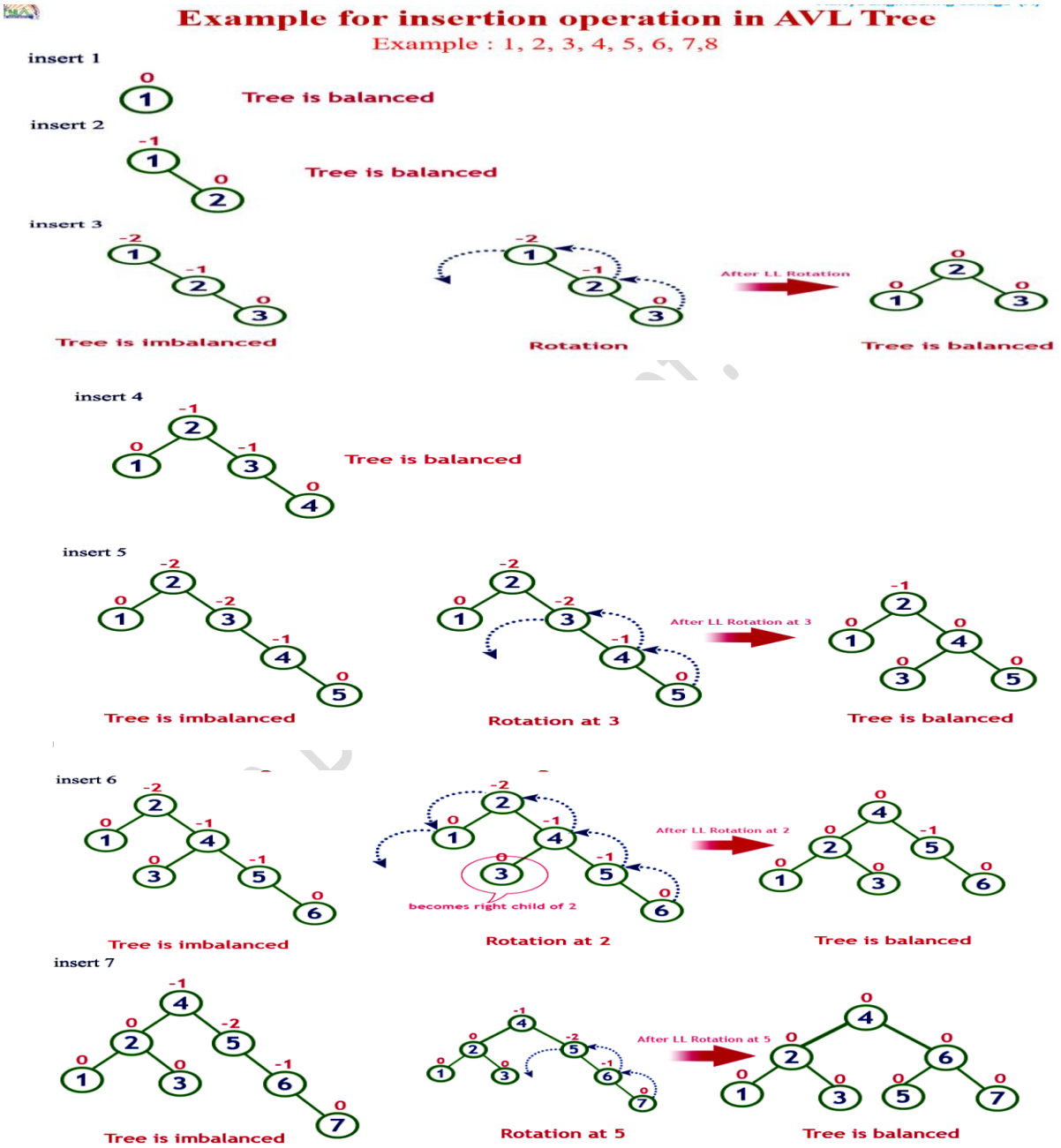
Insert algorithm in AVL Tree:

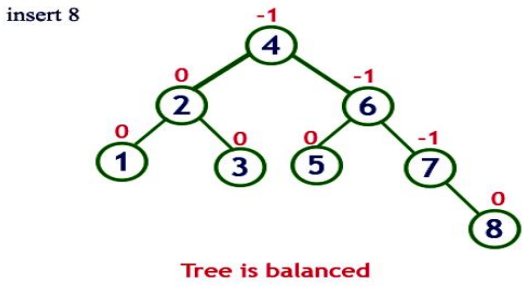
Step 1 - Insert the new element into the tree using Binary Search Tree insertion logic.

Step 2 - After insertion, check the **Balance Factor** of every node.

Step 3 - If the **Balance Factor** of every node is **0 or 1 or -1** then go for next operation.

Step 4 - If the **Balance Factor** of any node is other than **0 or 1 or -1** then that tree is said to be imbalanced. In this case, perform suitable **Rotation** to make it balanced and go for next operation.





Delete Operation in AVL Tree

- ✓ The deletion operation in AVL Tree is similar to deletion operation in BST.
- ✓ But after every deletion operation, we need to check with the Balance Factor checking.
- ✓ If the tree is balanced after deletion go for next operation otherwise perform suitable rotation to make the tree Balanced.

****