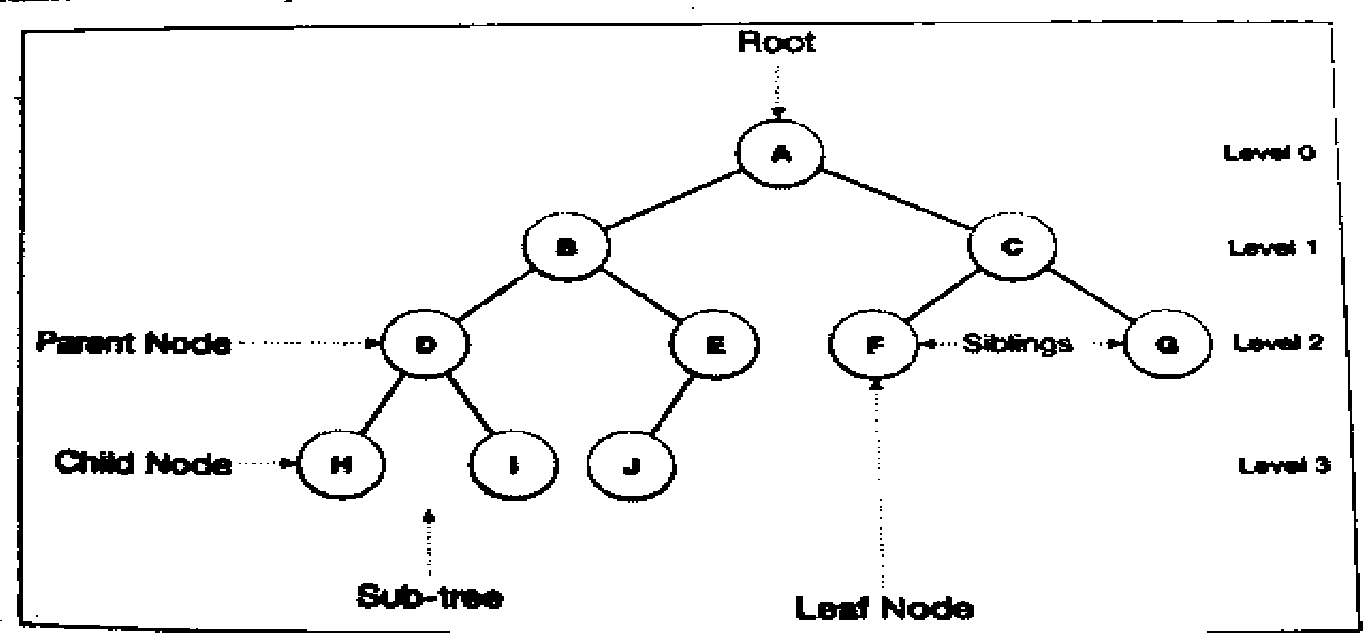
Tree

1.1 Definition and Concept :

The tree is a non-linear data structure that allows you to associate a parent-child relationship between various pieces of data and thus allows us to arrange our records, data, and files in a hierarchical fashion. It represents the nodes connected by edges.



Root: Root is a special node in a tree. The entire tree is referenced through it. It does not have a parent.

Parent Node: Parent node is an immediate predecessor of a node.

Child Node : All immediate successors of a node are its children.

Leaf : The node which does not have any child node is called the leaf node.

Siblings: Nodes with the same parent are called Siblings.

Level : Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.

Path : Path is a number of successive edges from source node to destination node.

Height of Node: Height of a node represents the number of edges on the longest path between that node and a leaf.

Height of Tree: Height of tree represents the height of its root node.

Depth of Node : Depth of a node represents the number of edges from the tree's root node to the node.

Degree of Node: Degree of a node represents a number of children of a node.

Edge: Edge is a connection between one node to another. It is a line between two nodes or a node and a leaf.

1.2 Binary Tree:

In a binary tree, each node can have at most two children, such that:

i. a binary tree can be empty called the NULL tree or
ii. consist of a root node and two disjointed binary trees termed as left subtree and right subtree.

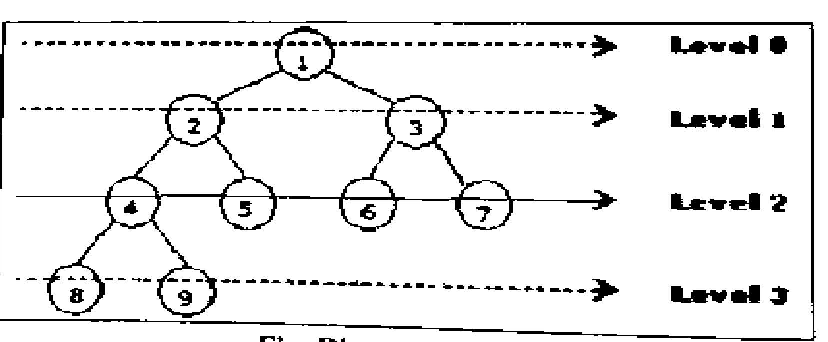


Fig. Binary tree

A simple binary tree of **size 9** and **height 4**, with a **root node** whose value is 1

Properties of Binary Trees:

Some of the important properties of a binary tree are as follows:

1. If a binary tree contains n nodes, then it contains exactly n-1 edges.
2. A binary tree of height h has 2^h-1 nodes or less.
3. If a binary tree contains m nodes at level l, it contains at most 2m nodes at level l + 1.

Ex. Draw all possible non-similar trees T with 3 nodes.

Fig. non-similar trees T with 3 nodes

There are four types of binary tree:

1. Full Binary Tree
2. Complete Binary Tree
3. Skewed Binary Tree
4. Extended Binary Tree

1. **Full Binary Tree**

If each node of the binary tree has either two children or no child at all, is said to be a Full Binary Tree. A full binary tree is also called a Strictly Binary Tree.

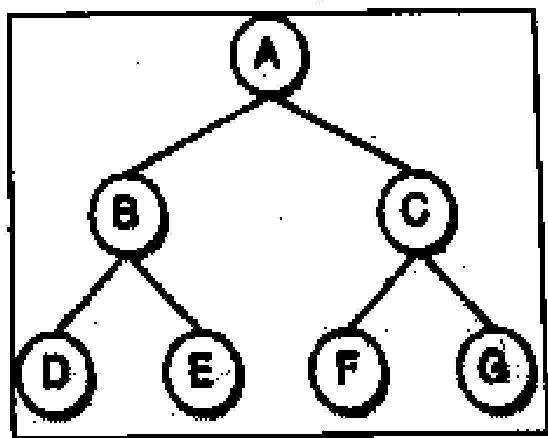


Fig. Full Binary Tree

Every node in the tree has either 0 or 2 children.

2. **Complete Binary Tree**

If all levels of the tree are completely filled except the last level and the last level has all keys as left as possible, is said to be a Complete Binary Tree. A complete binary tree is also called Perfect Binary Tree.

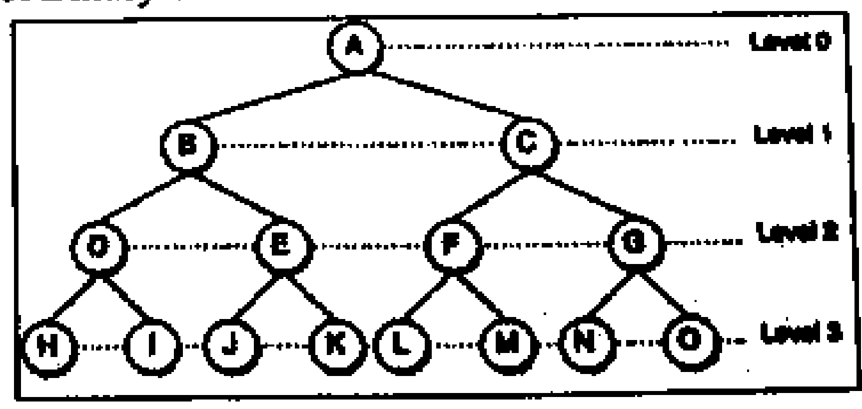


Fig. Complete Binary Tree

3. **Skewed Binary Tree**

If a tree that is dominated by the left child node or right child node is said to be a Skewed Binary Tree. In a skewed binary tree, all nodes except one have only one child node. The remaining node has no child.

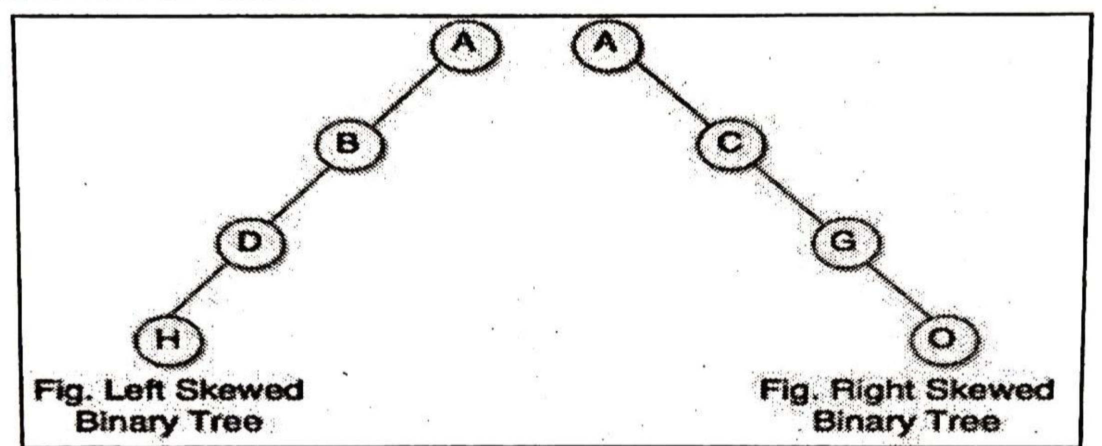


Fig. Left Skewed Binary Tree

Fig. Right Skewed Binary Tree

In a left-skewed tree, most of the nodes have the left child without corresponding right child.

In the right-skewed tree, most of the nodes have the right child without corresponding left child.

Extended Binary Tree

An extended binary tree consists of replacing every null subtree of the original tree with special nodes. The empty circle represents the internal node and the filled circle represents external node.

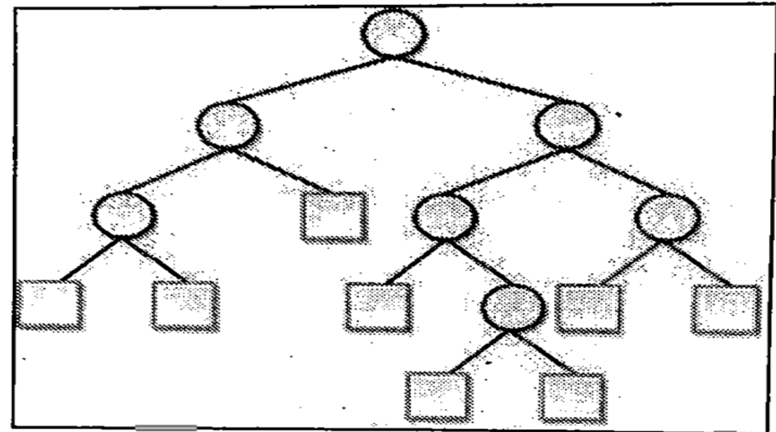


Fig. Extended Binary Tree

The nodes from the original tree are internal nodes and the special nodes are external nodes.

Every internal node in the extended binary tree has exactly two children and every external node is a leaf. It displays the result which is a complete binary tree.

1.3 Storage representation and Manipulation of Binary trees :

There are different storage representation technique is available for representing binary tree in computer memory.

a. Sequential representation of the binary tree
b. Linked storage representation for a binary tree.
c. Threaded storage representation for a binary tree.

a. Sequential representation of a binary tree:-

Let us consider that T is a binary tree that is complete or nearly complete. Then there is an efficient way of representing T in the memory called the sequential representation or array representation of T. This representation uses only a single linear array TREE as follows:

a) The root R of T stored in TREE[1].

b) If a node N occupies TREE[K], then its left child is stored in TREE[2 * K] and its right child is stored in TREE[2 * K + 1].

Null is used to indicate an empty subtree. In particular, TREE[1] = NULL indicates that the tree is empty.

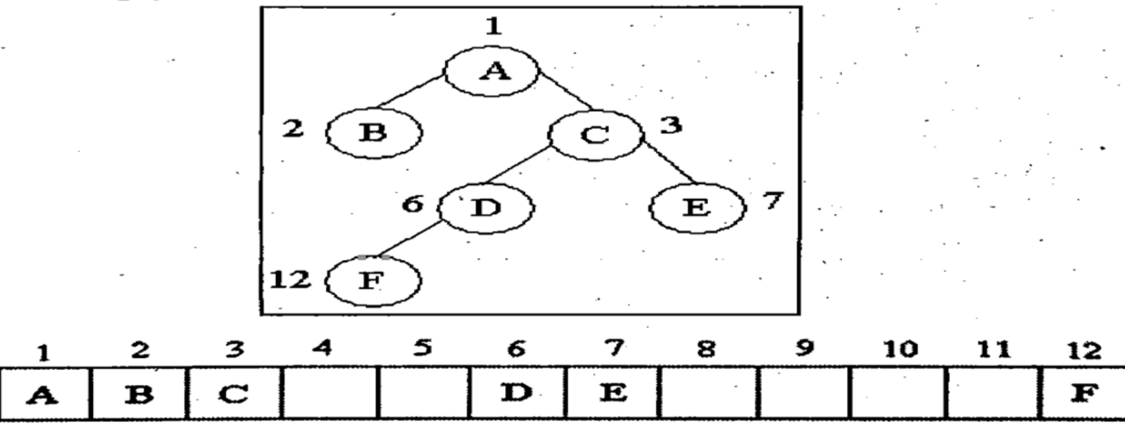


Fig. One-dimensional array representation

b. Linked storage representation:-

Since a binary tree consisting of nodes, which can have almost two children (subtree), the linked representation of such a tree involves the storage of nodes. Form as shown

LPTR	DATA	RPTR

Where LPTR and RPTR denote the address of the left and right subtrees respectively of a particular root node. Empty subtrees are represented by pointer values of NULL. Data specifies the information associated with a node.

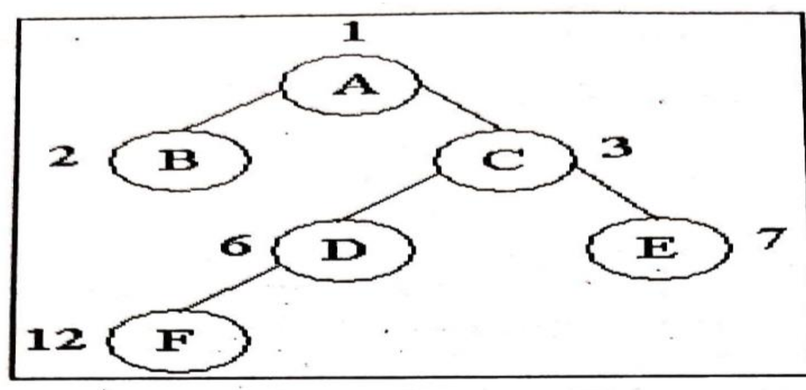


Fig. A Binary Tree

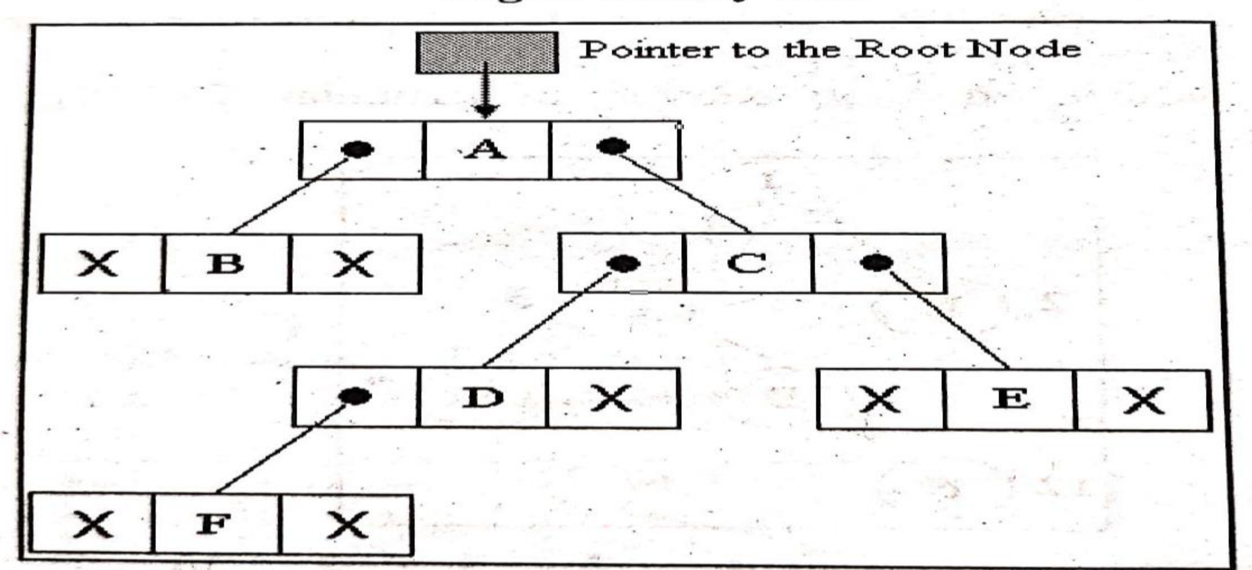


Fig. Linked storage representation of a binary tree

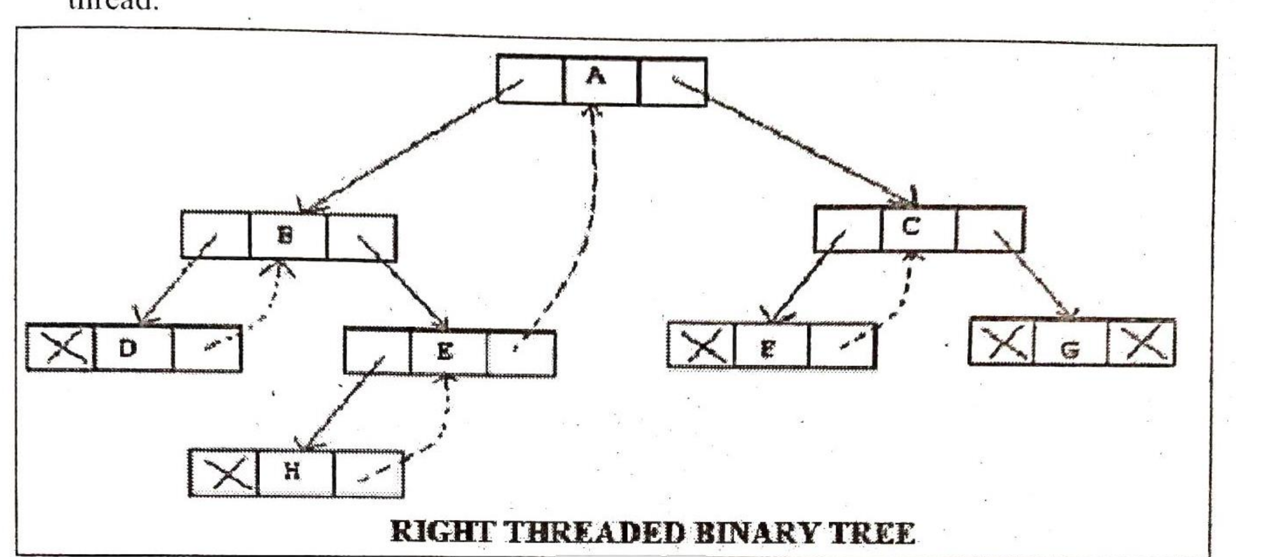
ed with
canner

c. **Threaded storage representation for a binary tree**

Consider the linked representation of a binary tree T, most of the entries in the left pointer field and right pointer field contain NULL elements. This space occupied by NULL entries can be efficiently utilized to store some kind of valuable information. These special pointers are called threads, and the binary tree having such pointers is called a threaded binary tree. Threads in a binary tree are represented by a dotted line.

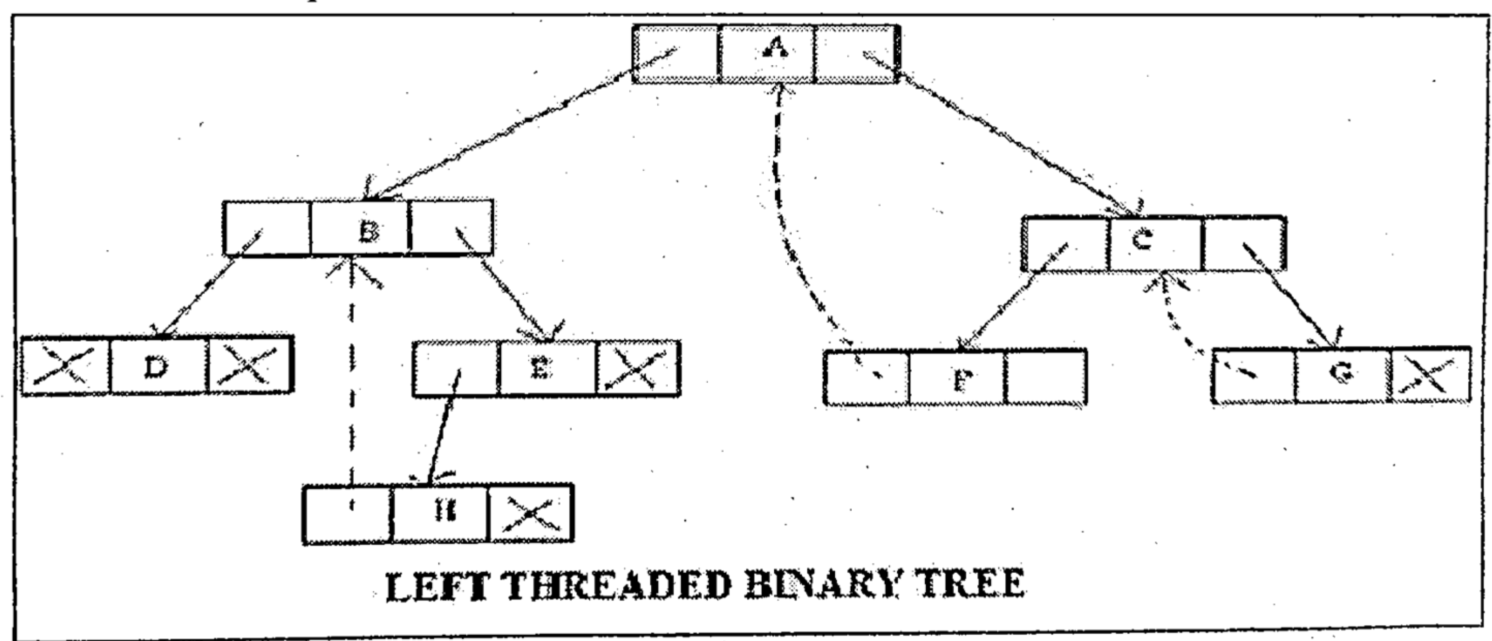
There are many ways to thread a binary tree these are—

1. **A right threaded binary tree-** The right NULL pointer of each leaf node can be replaced by a thread to the successor of that node under in order traversal called a right thread.



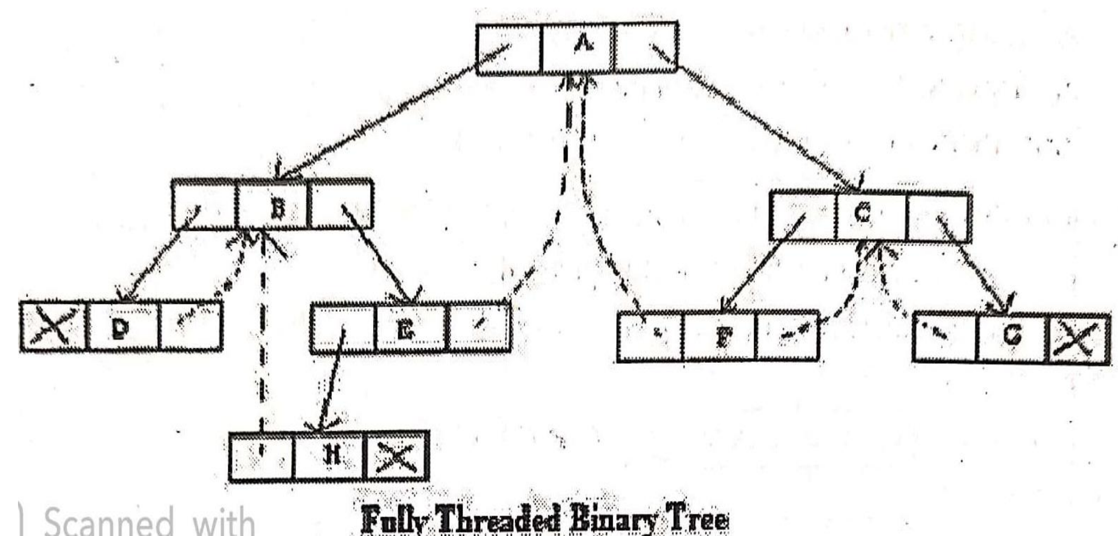
RIGHT THREADED BINARY TREE

2. **A left threaded binary tree-** The left NULL pointer of each node can be replaced by a thread to the predecessor of that node under in order traversal called left thread.



LEFT THREADED BINARY TREE

3. **A fully threaded tree-** Both left and right NULL pointers can be used to point to predecessor and successor of that node respectively, under in order traversal.
A threaded binary tree where only one thread is used is also known as one way threaded tree and where both threads are used is also known as two way threaded tree.



Scanned with

Fully Threaded Binary Tree

1.4 Operations on Binary tree – Traversing

One of the most common operations performed on tree structures is that of traversal. This is a procedure by which each node in the tree is processed exactly once in a systematic manner.

There are three main ways of traversing a binary tree with root :

1. Preorder Traversal
2. Inorder Traversal
3. Postorder Traversal

The easiest way to define each order is by recursion.

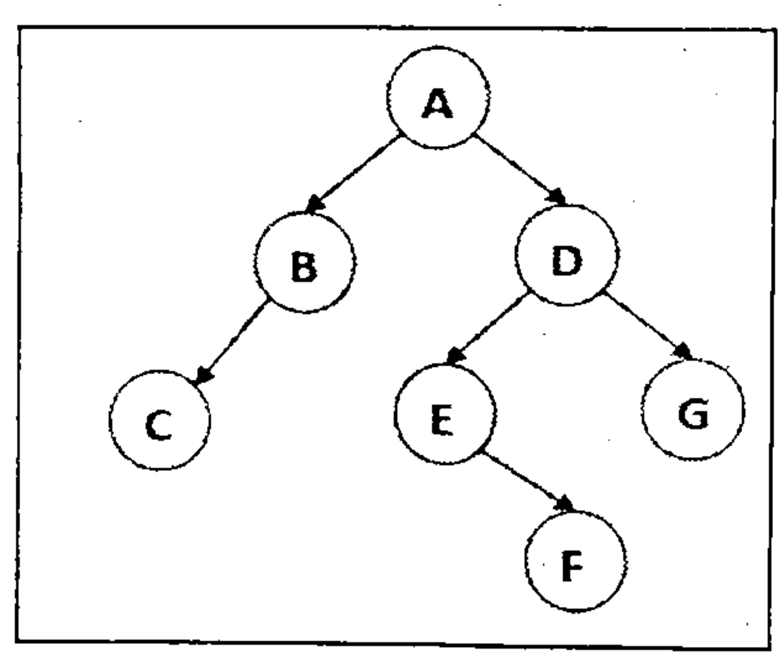


Fig. Binary Tree with Root

Preorder traversal : A B C D E F G

Inorder traversal : C B A E F D G

Postorder traversal : C B F E G D A

1. Preorder Traversal: -

i. Process the root node R.

ii. Traverse the left subtree of R in preorder.

iii. Traverse the right subtree of R in preorder.

From figure preorder traversal of tree T with root R is : **A B C D E F G**

2. Inorder Traversal:- The inorder traversal of binary tree is defined as follows:

i. Traverse the left subtree of R in inorder.

ii. Process the root R.

iii. Traverse the right subtree in inorder.

From figure Inorder traversal is : **C B A E F D G**

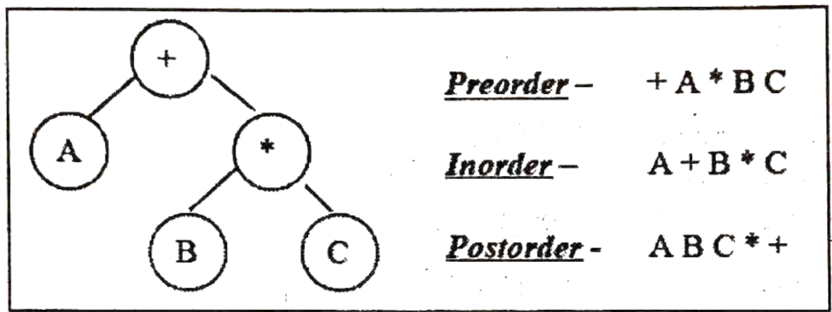
3. Postorder Traversal:- The postorder traversal of binary tree is defined as follows:

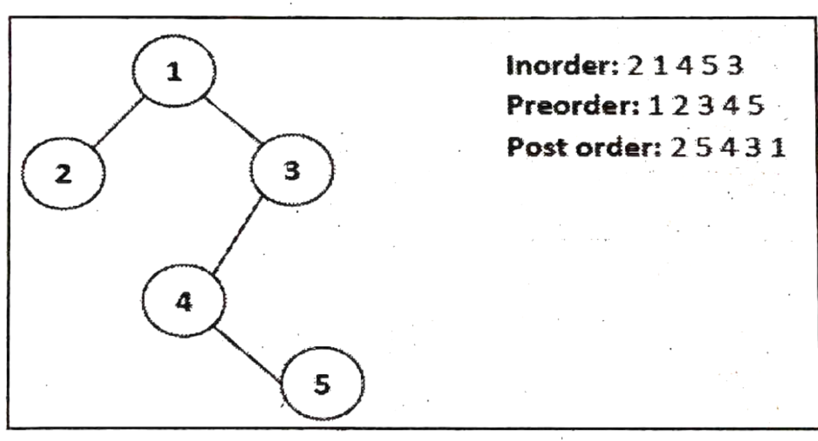
i. Traverse the left subtree of R in postorder.

ii. Traverse the right subtree of R in postorder.

iii. Process the root R.

Examples:-





Algorithms:

1) Preorder Traversal :-

A recursive preorder traversal algorithm is as follows-

Procedure RPREORDER(T) Given a binary tree whose root node by a pointer variable T and whose node structure is the LPTR & RPTR denotes the address of the left and right subtrees respectively. DATA specifies the information. EMPTY subtree are represented by a pointer values of NULL. This algorithm traverse the tree in preorder in a recursive manner.

1. [Process the root node]

   IF $T \neq$ NULL
   then Write('DATA(T))
   else Write('EMPTY TREE')
   Return

2. [Process the left subtree]

   If LPTR(T) $\neq$ NULL
   then Call RPREORDER(LPTR(T))

3. [Process the right subtree]

   If RPTR(T) $\neq$ NULL
   then Call RPREORDER(RPTR(T))

4. [Finished]

   Return

2) Inorder Traversal :-

A recursive inorder traversal algorithm is as follows-

Procedure RINORDER(T) Given a binary tree whose root node by a pointer variable T and whose node structure is the LPTR & RPTR denotes the address of the left and right subtrees respectively. DATA specifies the information. EMPTY subtree are represented by a pointer values of NULL. This algorithm traverse the tree in inorder in a recursive manner.

```
1.  [Check for empty tree]
        IF      T = NULL
        then    Write('EMPTY TREE')
                Return
2.  [Process the left subtree]
        If      LPTR(T) ≠ NULL
        then    Call RINORDER(LPTR(T))
3.  [Process the root node]
        Write('DATA(T))
4.  [Process the right subtree]
        If      RPTR(T) ≠ NULL
        then    Call RINORDER(RPTR(T))
5.  [Finished]
        Return
```

A recursive postorder traversal algorithm is as follows-

Procedure RPOSTORDER(T) Given a binary tree whose root node by a pointer variable T and whose node structure is the LPTR & RPTR denotes the address of the left and right subtrees respectively. DATA specifies the information. EMPTY subtree are represented by a pointer values of NULL. This algorithm traverse the tree in postorder in a recursive manner.

```
1.  [Check for empty tree]
        IF      T = NULL
        then    Write('EMPTY TREE')
                Return
2.  [Process the left subtree]
        If      LPTR(T) ≠ NULL
        then    Call RPOSTORDER(LPTR(T))
3.  [Process the right subtree]
        If      RPTR(T) ≠ NULL
        then    Call RPOSTORDER(RPTR(T))
4.  [Process the root node]
        Write ('DATA(T))
5.  [Finished]
        Return
```

1.5 Operations & Algorithms on BST – Create, Insert, Delete

Binary Search Tree :-

Suppose T is a binary tree. Then T is called a binary search tree (or binary sorted tree) if each node N of T has the following property

The value of N is greater than every value in the left subtree of N and is less than every value in the right subtree of N.

Note that - this property guarantees that the inorder traversal of T will yield a sorted listing of the element of T.

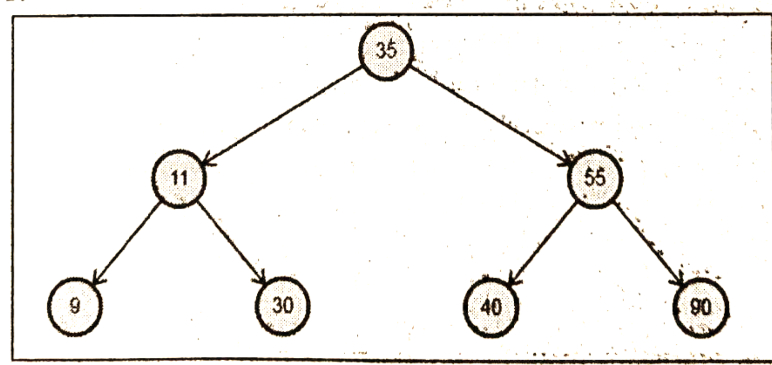


Figure 4.6

Traverse this tree in inorder method. It will give Inorder traversal : 9 11 30 35 40 55 90

This shows that, the inorder traversal of binary search tree gives the elements is ascending order, therefore we can define binary search tree. In other word, If a particular tree is traverse in inorder & if it gives the elements in a ascending order it is called as binary search tree.

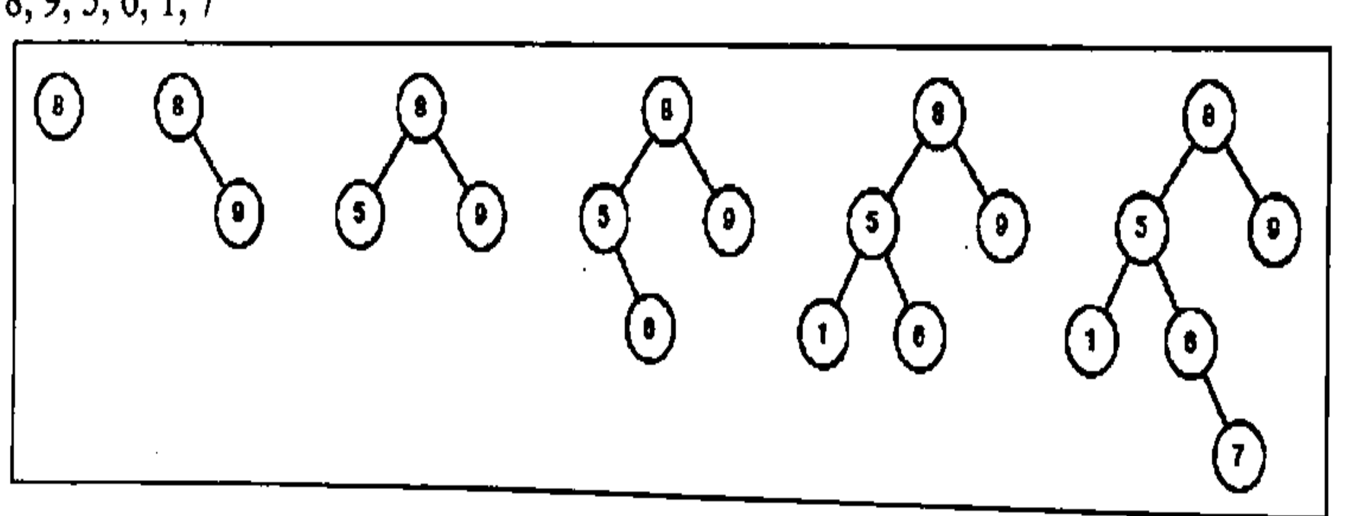
Algorithm to construct a binary search tree:

Following is algorithm to insert a new node into a binary search tree. Trees can be created through the repeated use of insertion operation a general algorithm for performing such an inserting into an existing lexically ordered binary tree is as follows:

1) If the existing the new node as the root of the tree & exist.
2) Compare the new name (element) with the name of the root node if the new name is lexically less than the root node name then if the left subtree is not empty then repeat step 2 on the left subtree ale append the new name as a lest leaf to the present tree and exist else if the right subtree is not empty then repeat step 2 on the right subtree else append the new name as the right leaf to the present tree exist.

Ex. The following six numbers are inserted in order into an empty binary search tree:

8, 9, 5, 6, 1, 7

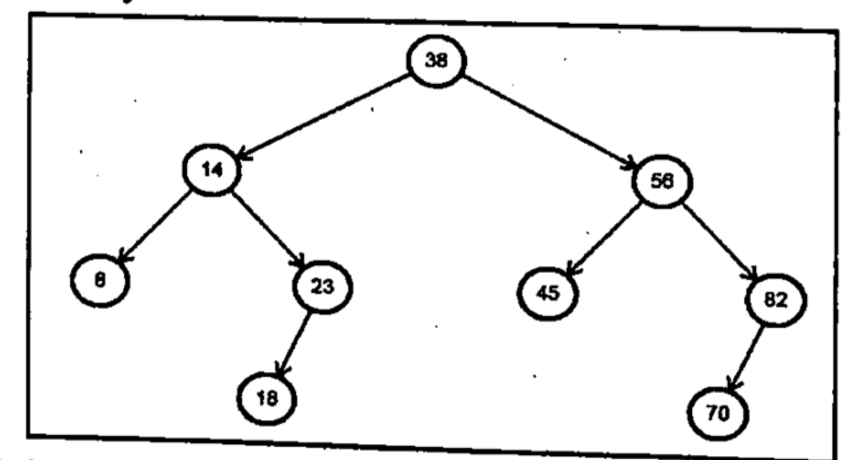


Searching and inserting binary search trees :

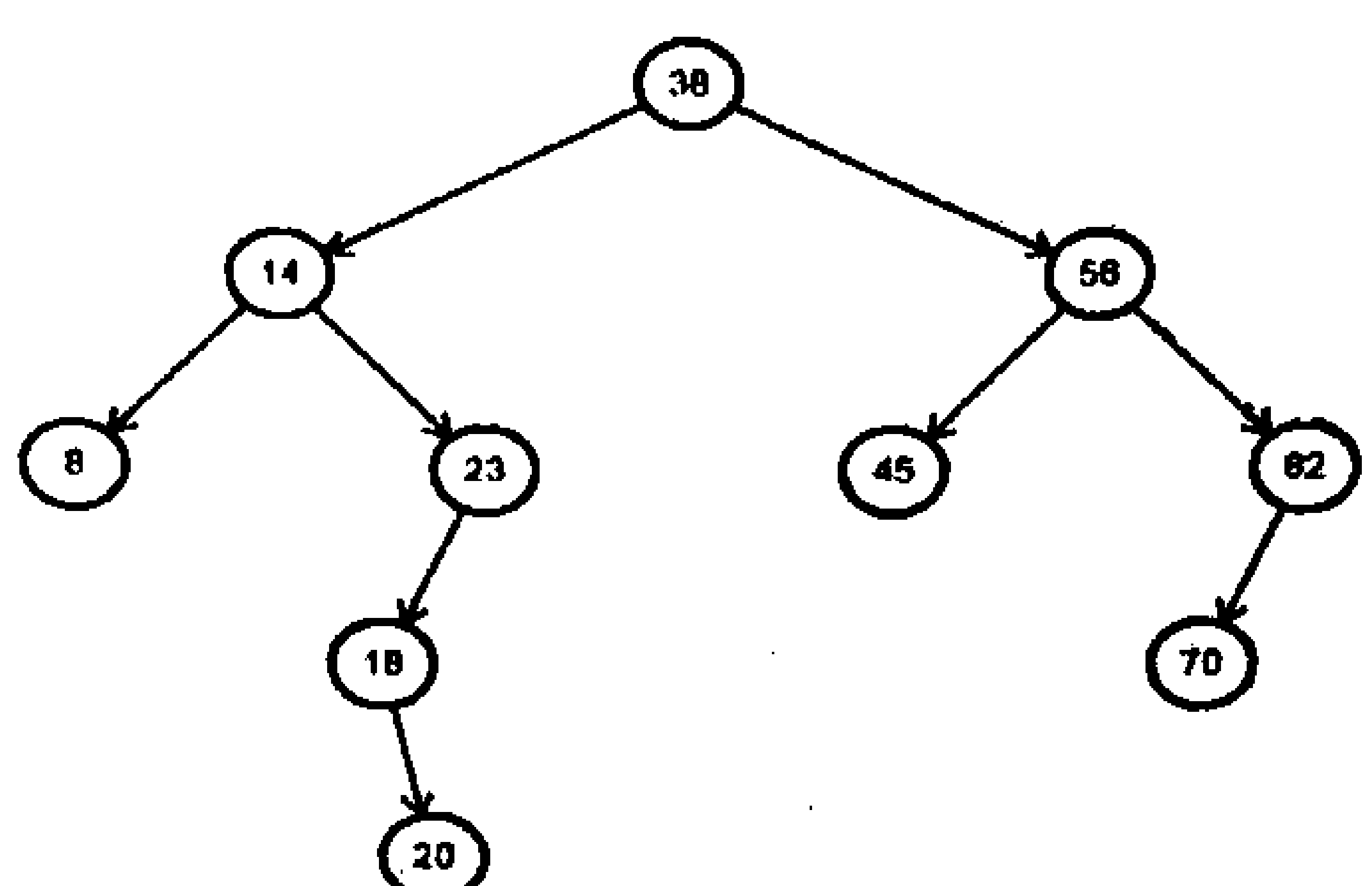
Suppose T is a binary search tree. An ITEM of information is given The following algorithm finds the location of ITEM in the binary search tree T, or inserts ITEM as a new node in its appropriate place in the tree.

1. [Compare ITEM with the root node N of the tree.]
   a) If ITEM < N, proceed to the left child of N
   b) If ITEM > N, proceed to the right child of N
2. [Repeat step (a) until one of the following occurs:
   a) We meet a node N such that ITEM = N, In this case the search is successful.
   b) We meet an empty subtree, which indicates that the search is unsuccessful, and we insert ITEM in place of the empty subtree.

Example. Consider a binary tree



Suppose ITEM =20 is given through search is unsuccessful and we can insert ITEM to its appropriate, place as shown in following figure.


38
14
56
8
23
45
82
18
70
20

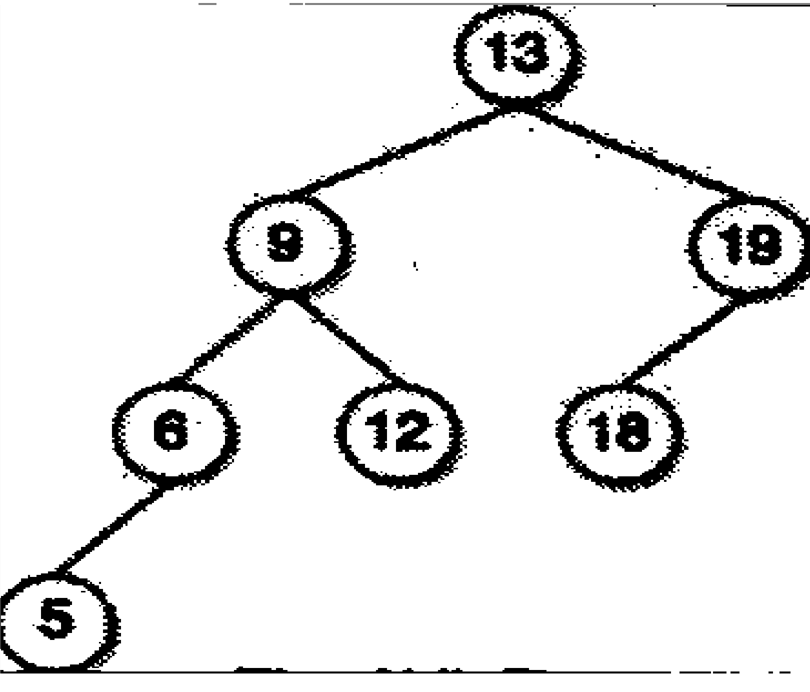
1.6 Concept: AVL tree. B- Tree

AVL Tree

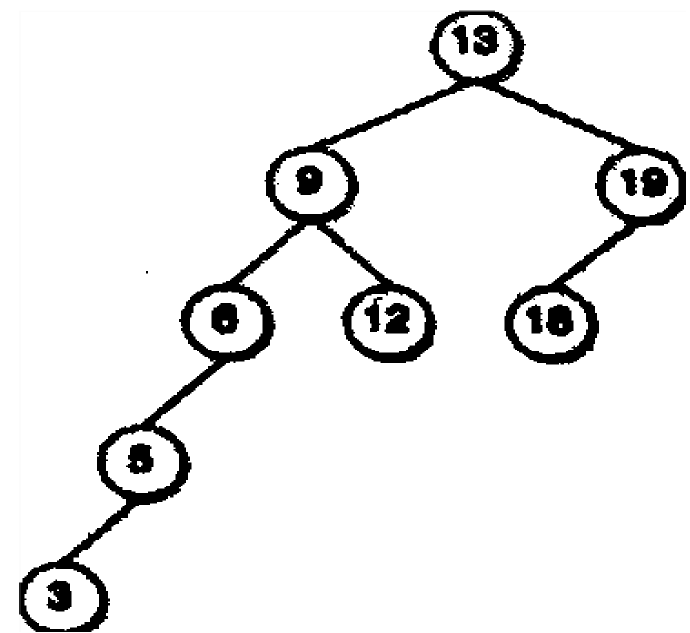
- AVL tree is a height-balanced tree.
- It is a self-balancing binary search tree.
- AVL tree is another balanced binary search tree.
- It was invented by Adelson-Velskii and Landis.
- AVL trees have faster retrieval.

It takes O(log N) time for addition and deletion operation.

In AVL tree, heights of left and right subtree cannot be more than one for all nodes.



The above tree is AVL tree because the difference between heights of left and right subtrees for every node is less than or equal to 1.



The above tree is not AVL because the difference between heights of left and right subtrees for 9 and 19 is greater than 1.

It checks the height of the left and right subtree and assures that the difference is not more than 1. The difference is called the balance factor.

Height Balanced Trees:

- An empty tree is height balanced.
- A binary tree with h_L and h_R as height of left and right sub-tree respectively is height balanced if $h_L - h_R <= 1$.
- A binary tree is height balanced if every sub-tree of the given tree is height balanced.
- An AVL tree is a height balanced binary search tree.

Balance Factor:

The balance factor, BF (T) of a node T in a binary tree is defined as $h_L - h_R$. where h_L and h_R are the heights of the left and the right sub-trees of T.

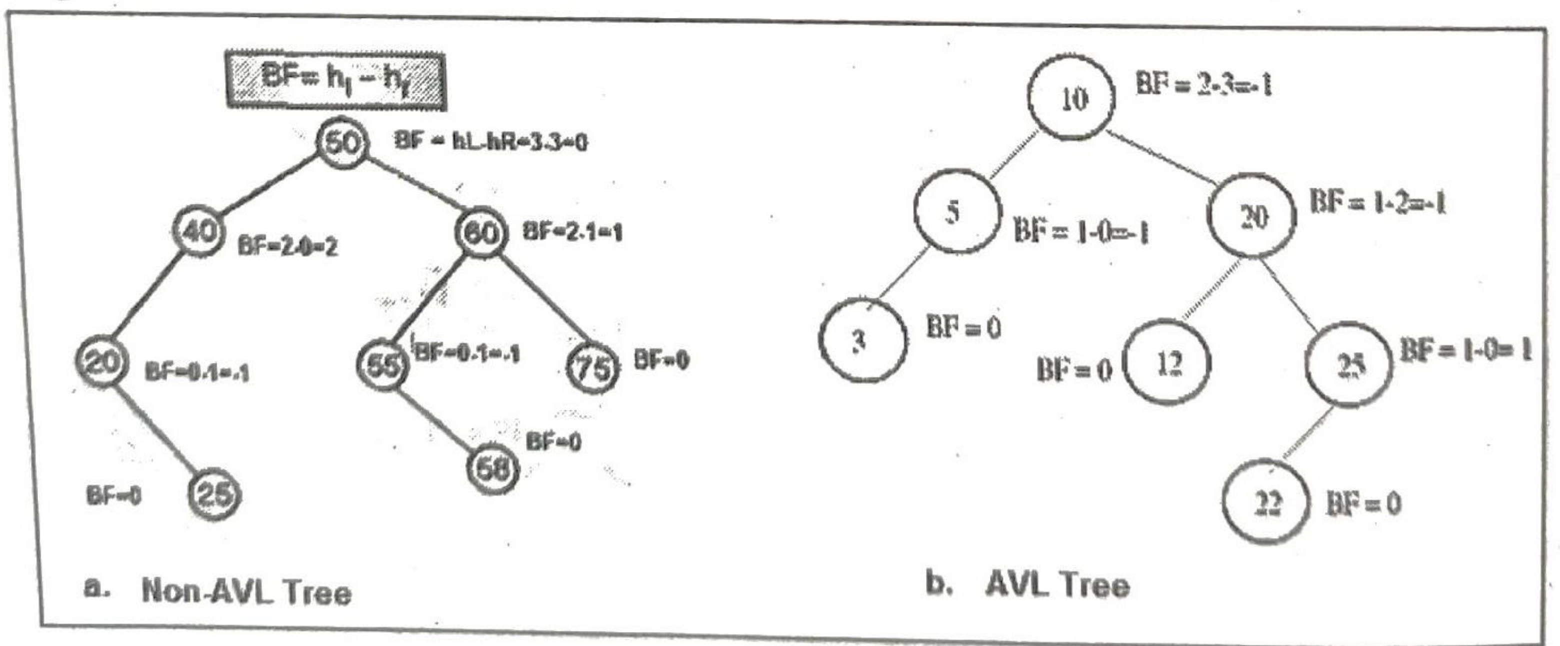
i.e. $BF(T) = h_L - h_R$

The balanced factor of every leaf node is 0.

The balance factor of a node in an **AVL tree could be -1, 0 or 1.**

- If the balance factor of a node is 0 then the heights of the left and rightsub-trees are equal.
- If the balance factor of a node is +1 then the height of the left sub-tree is one more than the height of the right sub-tree.
- If the balance factor of a node is -1 then the height of the left sub-tree is one less than the height of the right sub-tree.

Example –



In above tree, Fig. a is not an AVL tree. The balance factor of the node with data 40 is + 2. And Fig.b is AVL-tree.

AVL Rotations:

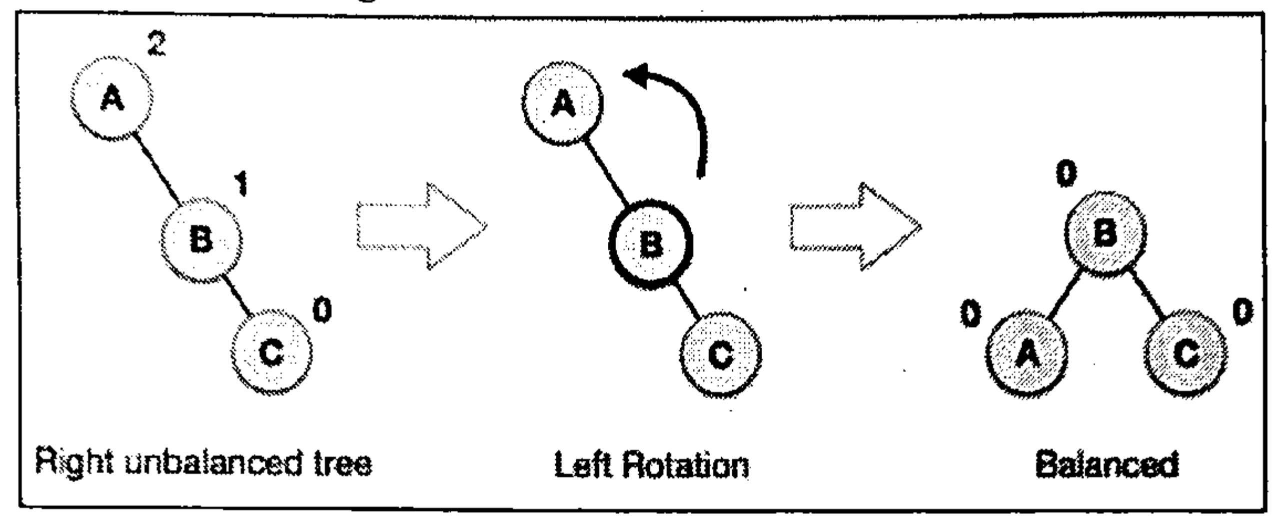
To make itself balanced, an AVL tree may perform four kinds of rotations –

1. Left rotation
2. Right rotation
3. Left-Right rotation
4. Right-Left rotation

First two rotations are single rotations and next two rotations are double rotations. Two have an unbalanced tree we at least need a tree of height 2.

1. **Left rotation:**

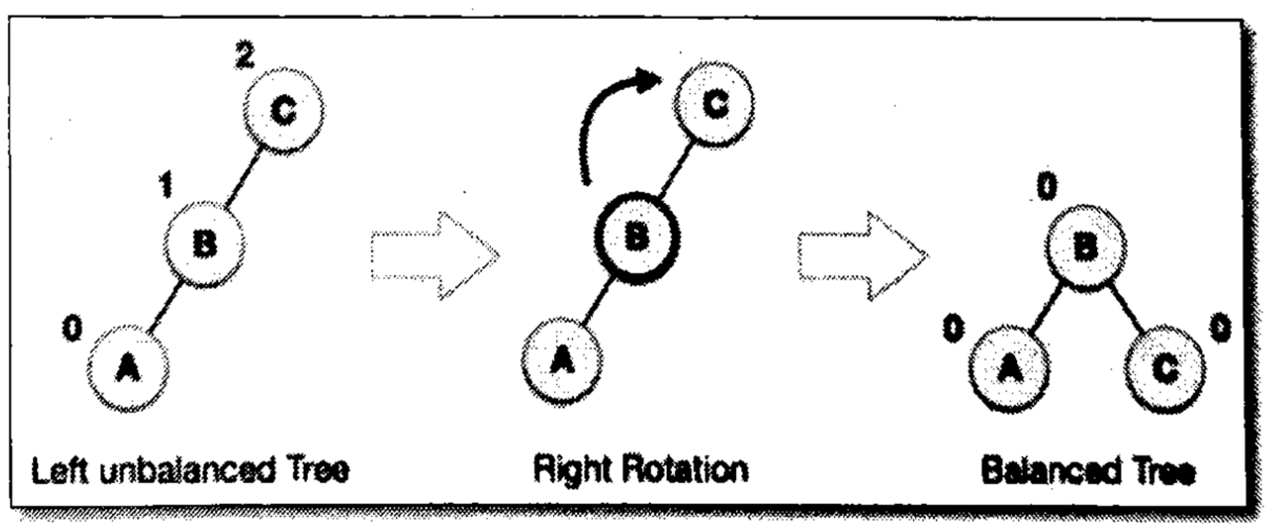
If a tree becomes unbalanced, when a node is inserted into the right sub-tree of right sub-tree, then we perform single left rotation.



In above, node A has become unbalanced as a node is inserted in right sub-tree of A's rightsub-tree. We perform left rotation by making A left sub-tree of B.

2. **Right rotation**

AVL tree may become unbalanced if a node is inserted in the left sub-tree of left sub-tree. The treethen needs a right rotation.



In above, node C is unbalanced node and it becomes right child of its left child by performing a right rotation.

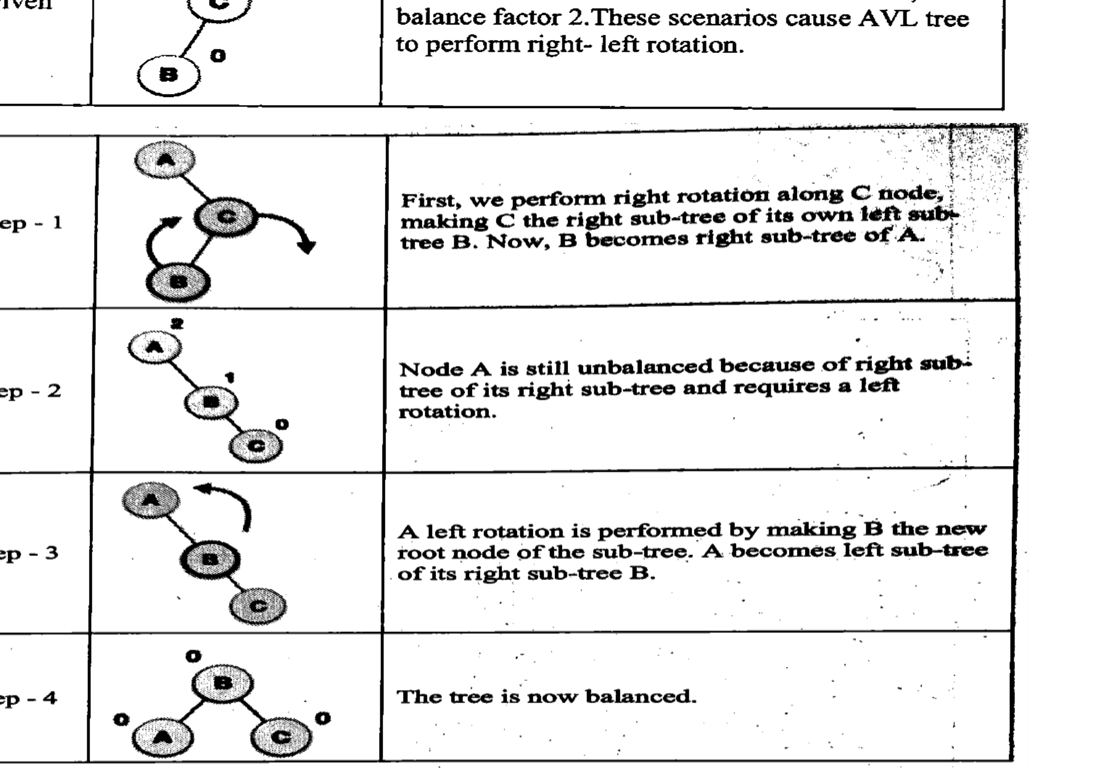
3. Left-Right rotation

The first type of double rotation is Left-Right Rotation. A left-right rotation is combination of left rotation followed by right rotation.

Steps	State	Action
Given		A node has been inserted into right sub-tree of left sub-tree. This makes C an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
Step -1		We first perform left rotation on left sub-tree of C. This makes A, left sub-tree of B.
Step -2		Node C is still unbalanced but now, it is because of left sub-tree of left sub-tree apply right rotation. i.e left sub-tree of B is A and left sub-tree of C is B.
Step -3		We shall now right-rotate the tree making B new root node of thissub-tree. C now becomes right sub-tree of its own left sub-tree.
Step - 4		The tree is now balanced.

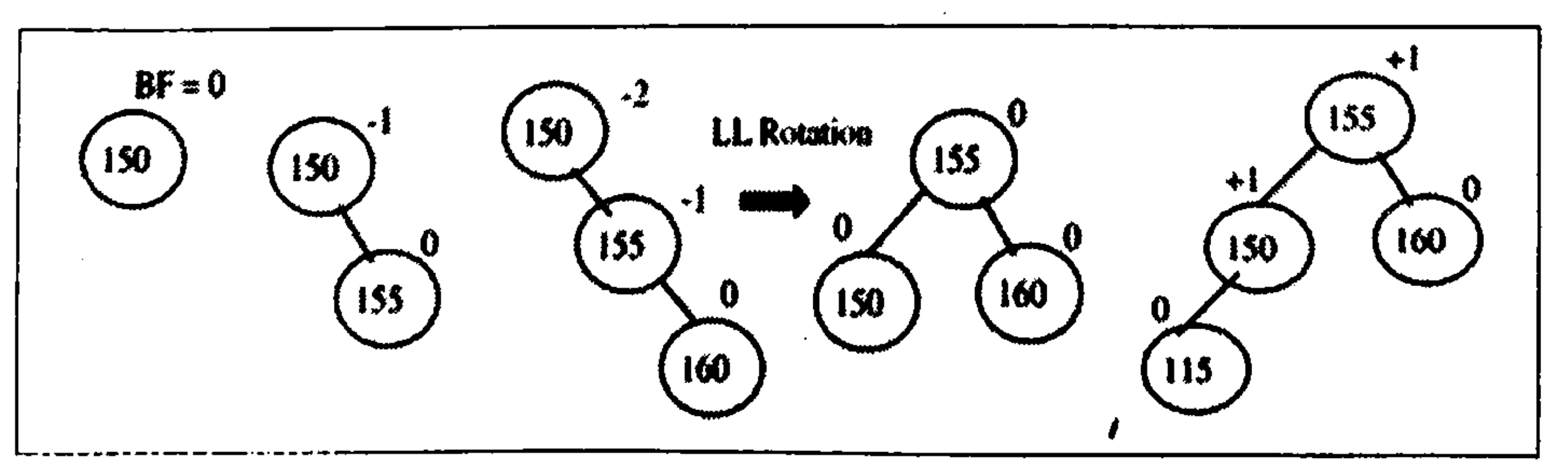
4. Right-Left rotation

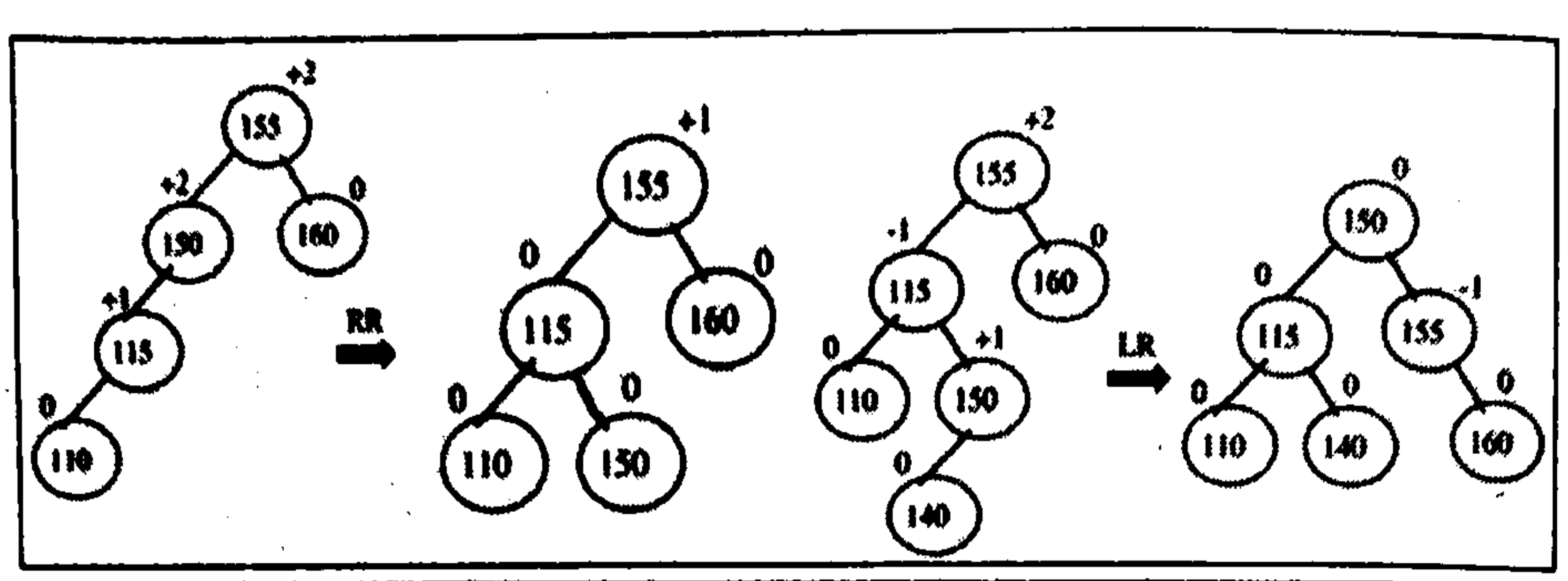
Second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

Steps	State	Action
Given		A node has been inserted into left sub-tree of right sub-tree. This makes A an unbalanced node, with balance factor 2.These scenarios cause AVL tree to perform right- left rotation.
Step - 1		First, we perform right rotation along C node, making C the right sub-tree of its own left sub-tree B. Now, B becomes right sub-tree of A.
Step - 2		Node A is still unbalanced because of right sub-tree of its right sub-tree and requires a left rotation.
Step - 3		A left rotation is performed by making B the new root node of the sub-tree. A becomes left sub-tree of its right sub-tree B.
Step - 4		The tree is now balanced.

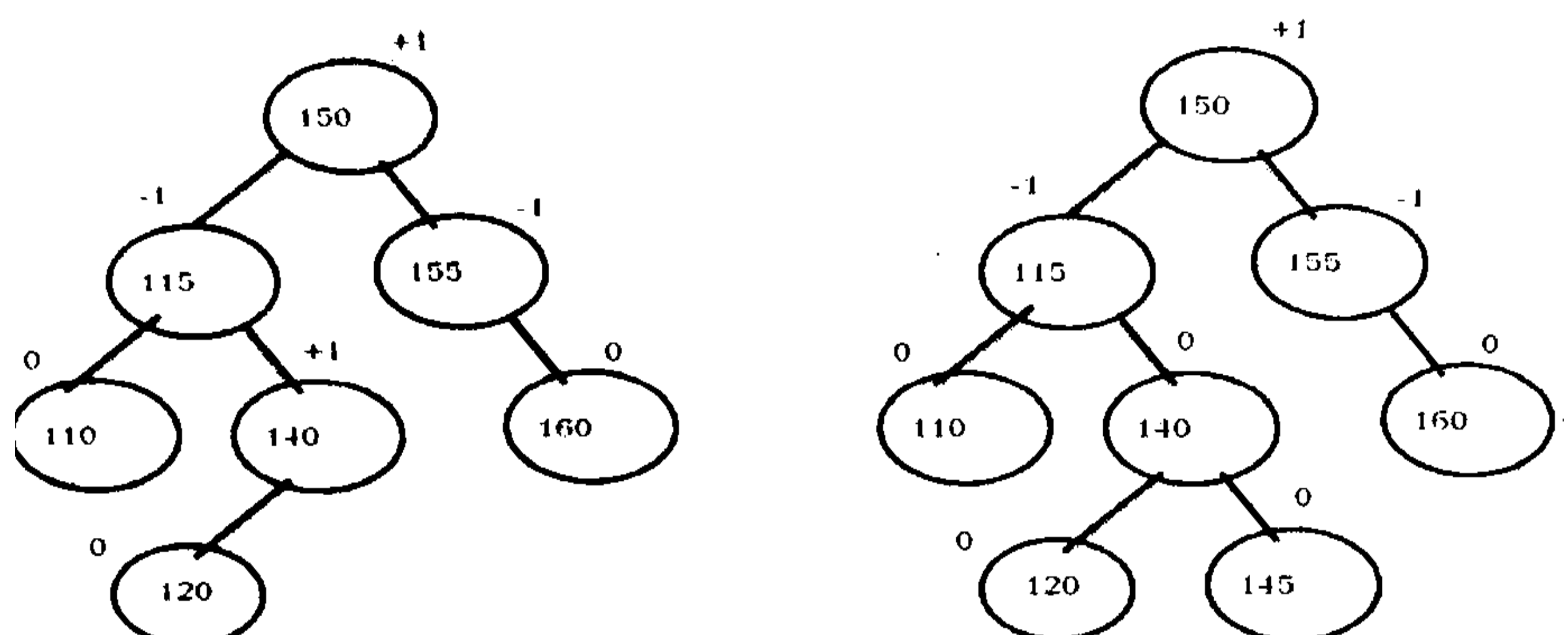
Q. –Construct Height balanced tree or AVL Tree using following data?

Example: 150, 155, 160, 115, 110, 140, 120, 145, 130, 147, 170, 180

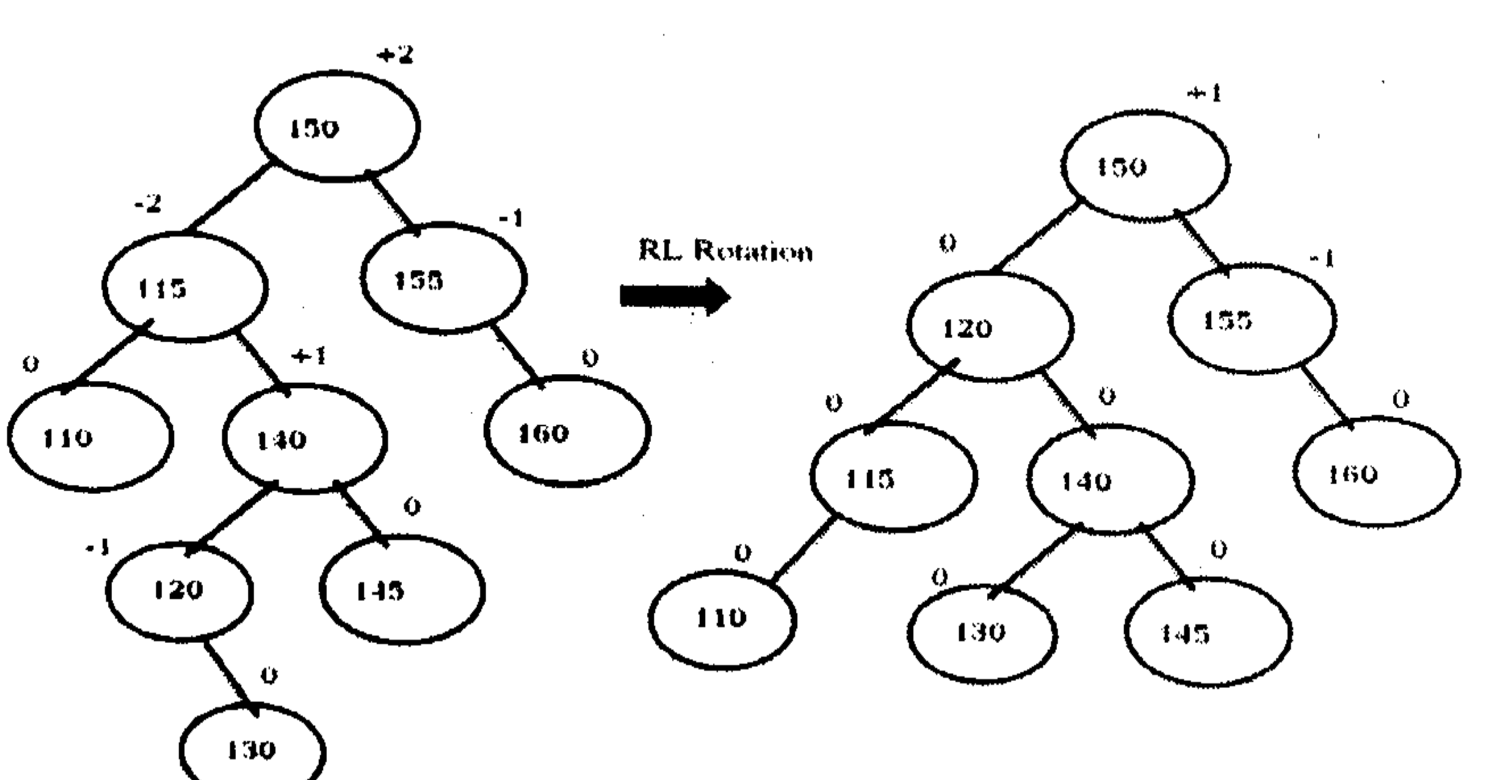




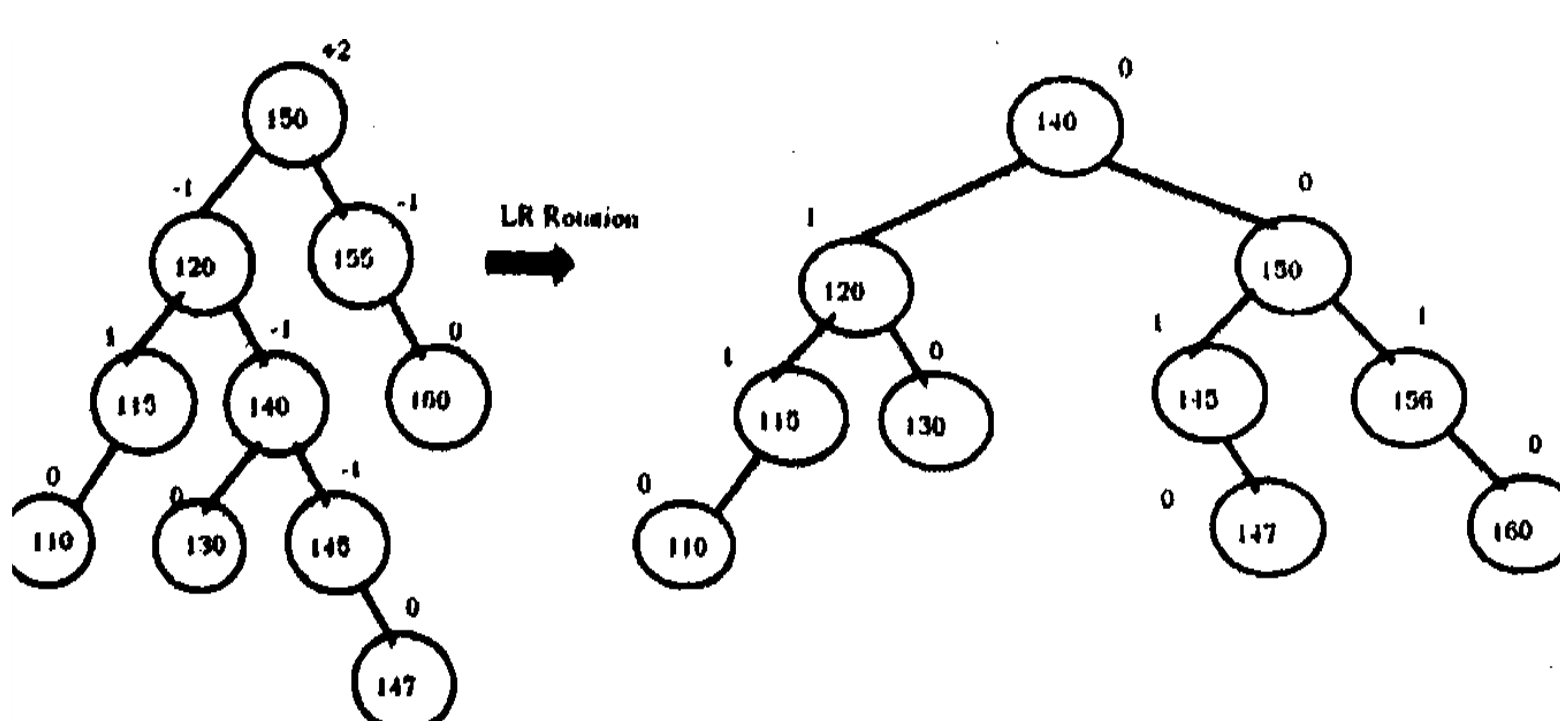
+2
155
+2
0
130
160
+1
115
0
110
RR
+1
155
0
0
115
160
0
0
110
150
+2
155
-1
0
115
160
0
+1
110
150
0
140
LR
0
150
0
-1
115
155
0
0
0
110
140
160



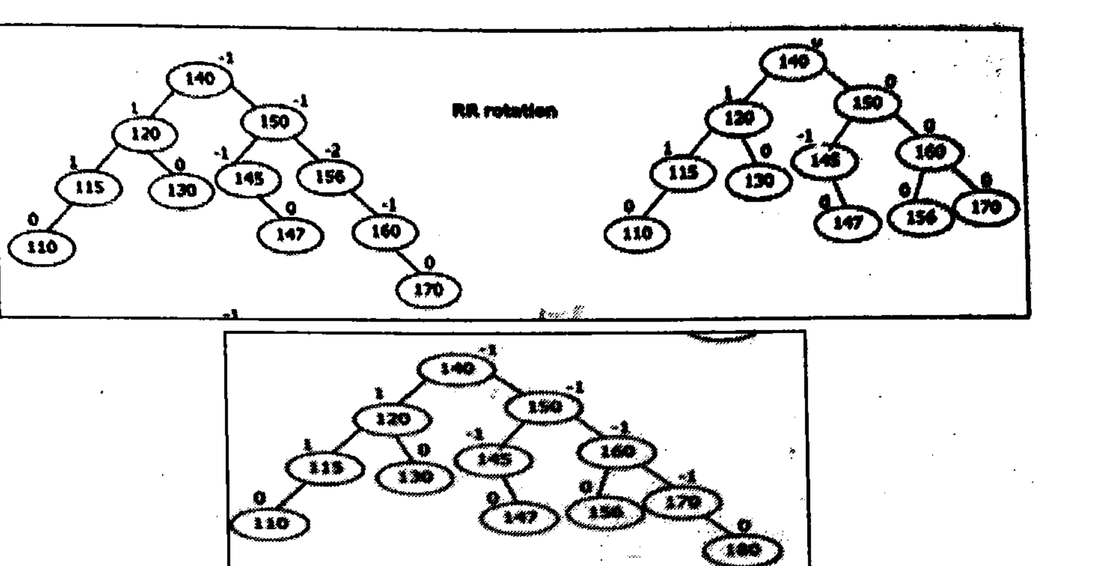
+1
150
-1
115
-1
155
0
110
+1
140
0
160
0
120
+1
150
-1
115
-1
155
0
110
0
140
0
160
0
120
0
145



+2
150
-2
115
-1
155
0
110
+1
140
0
160
-1
120
0
145
0
130
RL Rotation
+1
150
0
120
-1
155
0
115
0
140
0
160
0
110
0
130
0
145



LR Rotation
150
120
140
110
130
147
160



- Example:75, 8, 76, 79, 35, 77, 65, 72, 83, 06, 03 (H.W.)

B - Tree

Introduction of B-Tree

B-Tree was developed in the year 1972 by Bayer and McCreight with the name Height Balanced m-way Search Tree. Later it was named as B-Tree. B-Trees are multi-way search trees commonly used in database systems or other applications where data is stored externally on disks and keeping the tree shallow is important.

The B-tree is a generalization of a binary search tree in that a node can have more than two children. It is basically a self-balancing tree data structure that maintains sorted data and allows sequential access, searches, insertions, and deletions in logarithmic time.

Properties of B-Tree :

1. All leaf nodes must be at same level.
2. All nodes except root must have at least [m/2]-1 keys and maximum of m-1 keys.
3. All non leaf nodes except root (i.e. all internal nodes) must have at least m/2 children.
4. If the root node is a non leaf node, then it must have atleast 2 children.
5. A non leaf node with n-1 keys must have n number of children.
6. All the key values in a node must be in Ascending Order.