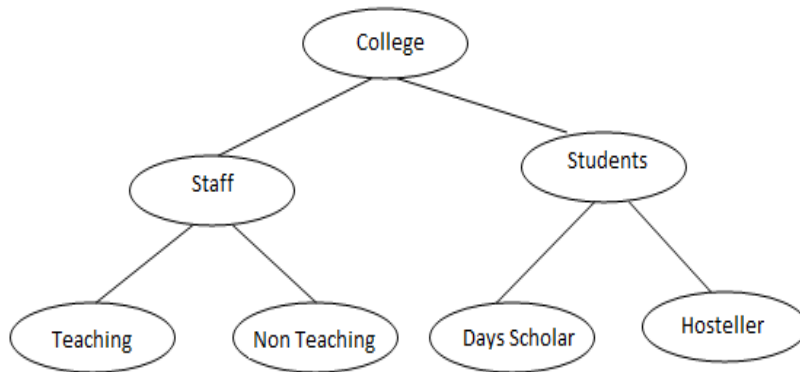**Q1) a)Define Tree.(2)**

<div align="center">Or</div>

Q1) b) What do you mean by Tree data Data Structure.(2)
ANSWER

A TREE is a finite set of one or more nodes such that there is a specially designated node called the ROOT and zero or more non empty subtrees T1,T2,T3,...............Tk each of whose roots are connected by a directed edge from ROOT R.

Q2) a) Write about the terms of Tree Data Structure.(8)
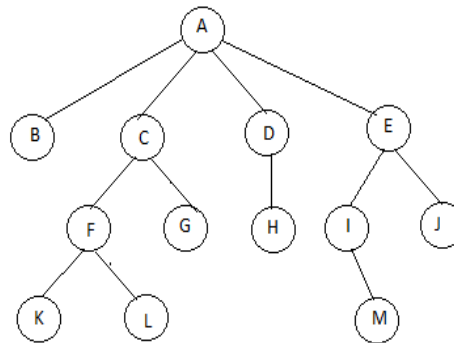
<div align="center">Or</div>

Q3) b) Define Basic Terminologies of Tree with examples.(13)
ANSWER
Basic Terminologies
Node: Item of information.
Root or Parent: A node which doesn't have a parent. A is a ROOT node.

Leaf Or Child:

A node which doesn't have child is called leaf or terminal node.b,K,L,G,H,M,J are leaf nodes.

Siblings:

Children of the same parents are said to be siblings.

B,C,D,E are siblings.

F,g are siblings

I,J, are siblings

K,L,are siblings.

Path:

- A path from node n1 to nk is defined as a sequence of nodes n1,n2,n3..............nk such that ni is the parent of ni+1 for 1<i<k.
- In a tree there is exactly only one path from root to each node.
- Path from A to L is A-C-F-L.where A is a parent for C,C is a parent for F and f is a parent for L.

Length: The length is defined as a number of edges on the path.The length for path a to l is 3.

Degree: The number of subtrees of anode is called its degree.

Degree of A is 4.

Degree of C is 2.

Degree of D is 1.

Degree of H is 0.

The degree of the tree is the maximum degree of any node in tree.

The degree of tree is 4.

Level:

The level of a node is defined by initially letting the root be at level one,if a node is at level L then its children are at level   L+1.

Level of A is 1.

Level of B,C,D are 2.

Level of F,G,H,I,J is 3.

Level of K,L,M, is 4.

Depth: For any node n, the depthof n is the length of the unique path from root to n.

The depth of the root is 0.

Depth of node F is 2.

Depth of node L is 3.

Height: For any node n,the height of the node n is the length of the longest path from n to the leaf.

The height of the leaf is 0.

Height of node F is 1 and L is 0.

Ancestor And Descendant Nodes:

If there is a path from n1 to n2,then n1 is an ancestor of n2 and n2 is a descendant of n1. If n1=n2 then n1 is a proper ancestor of n2 and n2 is a proper descendane of n1.

A is a ancestor of B,C,D,E.

B,C,D,E is descendant of A.

Note:
- The height of the tree=height of the root
- Depth of the tree= Height of the tree.

Q3) a) Explain the tree traversals with algorithms and examples.(13)

Or

**Q3) b) Explain the tree traversals. Give all the essential aspects.(15)**

ANSWER:

TREE TRAVERSAL
- ➢ Traversing means visiting each and every nodes only once.
- ➢ Tree Traversal is a method for visiting all the nodes in the tree exactly once.
- ➢ There are three types of tree traversal techniques namely,
  - ✓ Inorder traversal
  - ✓ Preorder traversal
  - ✓ Postorder traversal

INORDER TRAVERSAL:

The inorder Traversal of a binary tree is performed as,
- Traverse the leftsubtree in order.
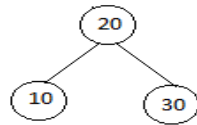- Visit the root or parent
- Traverse the right subtree in order.
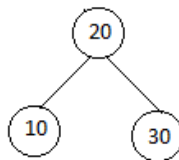
- ➢ Inorder Traversal : 10 – 20 – 30
- ➢ The inorder traversal of the binary tree for an arithmetic expression is in infix form.

Routine to perform in order traversal

```
void inorder (Tree T)
        {
                if(T!=NULL)
                {
                        inorder(T→left);
                        printf("%d",T→data);
                        inorder(T→right);
                }
        }
```

PREORDER TRAVERSAL:

➢ The preorder Traversal of a binary tree is performed as,
- Visit the root or parent.
- Traverse the left subtree in pre order.
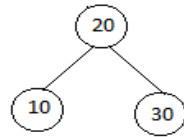- Traverse the right subtree in preorder.

➢ preorder Traversal 20 – 10- 30
➢ Pre order traversal of the binary tree for an arithmetic expression is in prefix form.

Routine to perform Pre order traversal

```
void preorder (Tree T)
        {
                if(T!=NULL)
                {
                        printf("%d",T→data);
                        preorder(T→left);
                        preorder(T→right);
                }
        }
```

POSTORDER TRAVERSAL:

➢ The postorder Traversal of a binary tree is performed as,
- Traverse the leftsubtree in pre order.
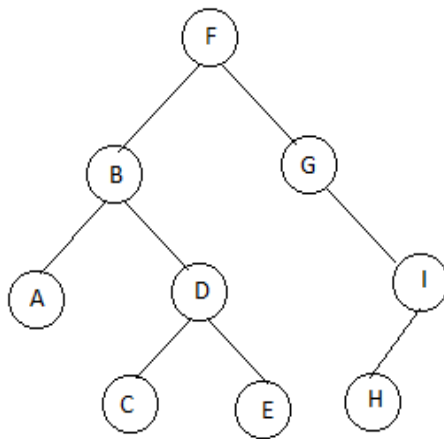- Traverse the right subtree in preorder.
- Visit the root or parent.

➤ Post order traversal of the binary tree for an arithmetic expression is in postfix form.

Routine to perform post order traversal

```
void postorder (Tree T)
        {
                        if(T!=NULL)
                        {
                                postorder(T→left);
                                postorder(T→right);
                                printf("%d",T→data);


                        }
                }
```

Example: Binary Tree

➤ Inorder Traversal of the fig2 is A,B,C,D,E,F,G,H,I.
➤ preorder Traversal of the fig2 is F,B,A,D,C,E,G,I,H.
➤ Postorder Traversal of the fig2 is F,B,A,D,C,E,G,I,H

Q4) a) i) Define Binary tree.(2)

   ii) What are the types of Binary Tree?(2)

   iii) How do you represent Binary Tree?(2)

   iv) What are the operations we can perform on Binary Tree?(2)

                                          Or

Q4) b) Explain about Binary Tree in detail.(13)

ANSWER:

BINARY TREE ADT

➢ Binary tree is a tree in which no node can have more than two children
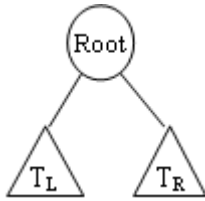
➢ A binary tree is a finite set of data items which is either empty or consists of a single item called the root and two disjoint binary trees called left subtree and right subtrees. The minimum degree of any node is atmost two and the depth of an average binary tree is smaller than N Average depth is O(n). Maximum number of nodes of level i of a binary tree is 2i+1

TYPES OF BINARY TREE:

      1) Full Binary Tree

      2) Complete Binary Tree

1) Full Binary Tree:

➢ A full binary tree of height h has $2^{h+1}-1$ nodes (ie) it contains maximum possible number of nodes in all level.
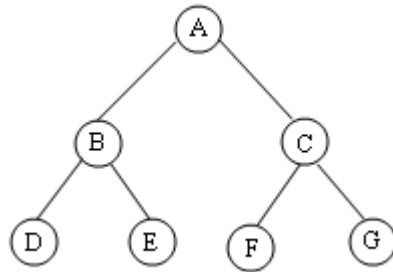
      Here height is 2

      No of nodes are $2^{h+1}-1 = 2^{2+1}-1$

                  $= 2^3-1 = 8-1 = 7$

              OR

Level	Nodes
0	1
1	2
2	4

7

2) Complete binary tree:

➢ A complete binary tree of height h has nodes between 2^h and 2^{h+1}. In the bottom level the elements should be filled from left to right.
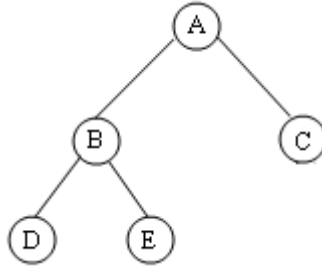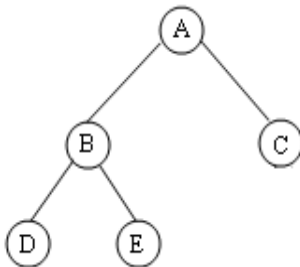

Here height is 2.

Nodes are between 2^h and 2^{h+1} = 2^2 and 2^3

= 4 and 8

➢ Here there are nodes.

➢ *Strict Binary Tree:*

   o Every non terminal node in a binary tree consists of a non empty left subtre and right subtree then such a tree is called strict binary tree.

➢ *Extended Binary Tree*:

   o In each node of a tree has either o0 or 2 children.In that case the nodes with 2 children are called internal node and the nodes with 0 children are called external nodes.
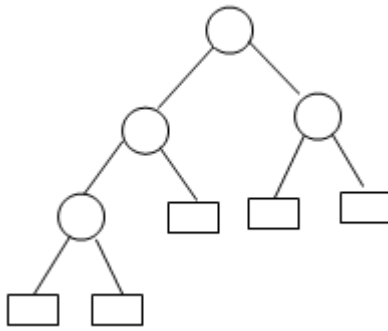
REPRESENTATION OF A BINARY TREE:
- ➤ There are two ways for representing binary tree .They are,
  - 1) Linear representation
  - 2) linked Representation

1) Linear Representation
- ➤ The elements are represented using arrays. For any element in position i, the left child is in position 2i,the right child is i position (2i+1), and the parent is in position (i/2)
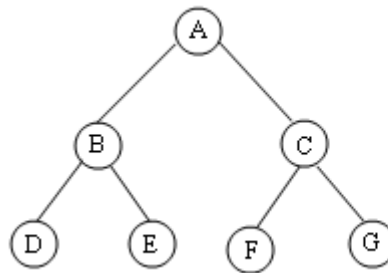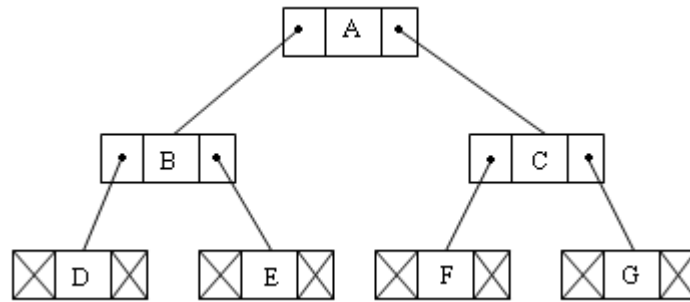

A	B	C	D	E	F	G
1	2	3	4	5	6	7

For eg,i=1 for node a
Its left child at 2i=2*1=2==== B
Its right child at 2i+1=2*1+1=2+1=3====C

2) Linked Representation:
- ➤ The elements are represented using pointers.Each node has 3 fields.
  - 1) Pointer to left subtree
  - 2) Data
  - 3) Pointer to right subtree.
- ➤ In leaf nodes,both pointers are NULL.

Declaration Of A Binary Tree

```
Struct TreeNode
{
int Element;
Struct TreeNode *Left;
Struct TreeNode * right;
};
```

OPERATIONS OF BINARY TREE:

➢ The basic operations that are performed onn binary tree are,

1) CREATE

It create an empty binary tree.

2) MAKEBT

It create a new binary tree having a single node with data field set to some value.

3) EMPTYBT

Return true,if the binary tree is empty else return false

4) LCHILD

Return a pointer to the left child of the node else return NULL pointer.

5) RCHILD

It rerurns a pointer to the right child of the node else return NULL pointer.

6) FATHER

It returns a pointer to the father of the node else it returns a NULL pointer.

7) SIBLING

Reurns a pointer to the brother of the node else it return a NULL pointer.

8) DATA

Returns the content of the node.

Various operations are,

1) Tree Traversal

2) searching for a node

3) Insertion of a node

4) Deletion of a node

5) Copying the binary tree.

Q5) a) Define Expression Tree. How do you construct a expression tree.(13)

Or

Q5) b) Construct a Expression Tree for the expression ab+c*

ANSWER:

EXPRESSION TREE:

Expression tree is a binary tree in which the leaf node is operands and the interior nodes are operators. Like binary tree, expression tree can also be traversed by inorder, preorder and postorder traversal.

Constructing an Expression Tree:

➤ Let us consider postfix expression given as an input for constructing an expression tree by performing the following steps:

1) Read one symbol at a time from the postfix expression

2) Check whether the symbol is an operand or operator.

a) If the symbol is an operand, create a one node tree and push a pointer on to the stack.

b) If the symbol is an operator pop two pointers from the stack namely T1 and T2 and from a new tree with root as the operator and T2 as a left child and T1 as a right child a pointer to this new tree is then pushed onto the stack.
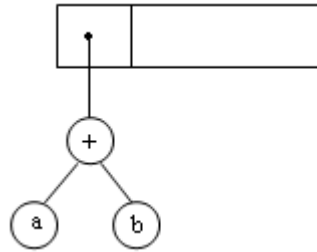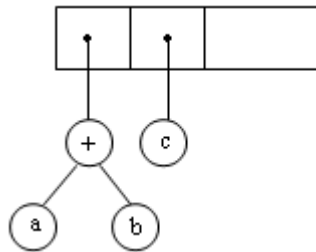
Eg:

ab+c*

**Step 1:** The first two symbols are operand, so create a one node tree and push the pointer onto the stack.
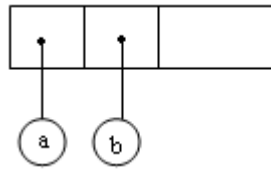
Step 2: Next + symbol is read,so two pointers are popped ,a new tree is formed and a pointer to this is pushed onto the stack.

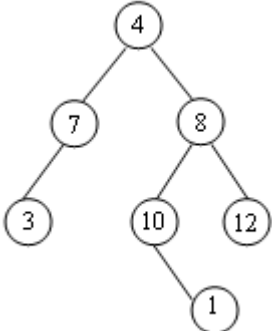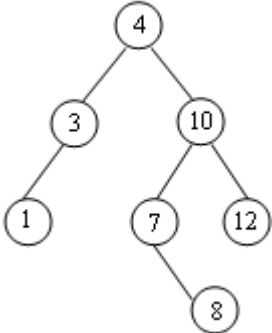**Step 3:** Next the operand C is read, so one node tree is created and the pointer is pushed onto the stack.

Step 4: Now * is read, so two trees are merged and the pointer to the final tree is pushed onto the stack.
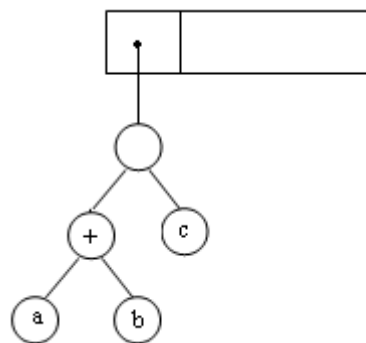
Q6) a) Differentiate Binary Tree and Binary Search Tree.
ANSWER:
Comparison Between Binary Tree & Binary Search Tree

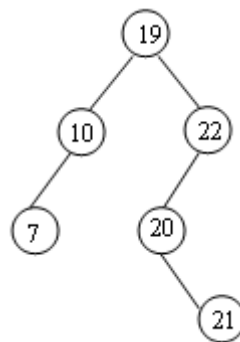| BINARY TREE	BINARY SEARCH TTREE
1)It is a tree which has atmost 2 children	It is a binary tree in which the key values in the left node is less than the root and the key values in the right node is greater than the root.
2)It doesn't have any order	

Q7) a) Explain Binary Search Tree ADT in detail. (15)

Or

Q7) b) Define Binary Search Tree.Write an algorithm for binary search tree operations. Give example.(15)
ANSWER
 BINARY SEARCH TREE ADT
Definition:
 ➢ Binary search tree is a binary tree in which for every node X in the tree,the values of all the keys in its left subtree are smaller than the key value in X and the values of all the keys in its right subtree and larger than the key value in X.

Declaration Of A Binary Search Tree

```
Struct TreeNode
{
int Element;
Struct  SearchTreeNode *Left;
Struct  SearchTreeNode * right;
};
```

OPERATIONS ON BINARY SEARCH TREE

1) Make Empty:
  ➢ This operation is mainly for initialization when the programmer prefer to initialize the first element as a one node tree.

Routine To Make A Empty Tree

```
SearchTree MakeEmpty (SearchTree T)
{
If(T!= NULL)
{
MakeEmpty ( T→ Left);
MakeEmpty (T→Right);
Free(T);
}
Return NULL;
}
```

2) Insert:
  ➢ To insert a element X into the tree,

> 1) Check with the root node T
> 2) If it is less than the root,
>> Traverse the left subtree recursively until; it reaches the T→left equals to NULL.Then X is placed in T→Left.
> 3) If X is greater than the root,

Traverse the right subtree recursively until it reaches the T →right equals to NULL .then X is placed in T→Right.

**Routine To Insert Into A Binary Search Tree**

```
SearchTree Insert(int X, SearchTree T)
{
If(T == NULL)
{
T= malloc ( size of (Struct TreeNode));
If(T!= NULL) // First element is placed in the root.
{
T→ Element = X;
T→ Left = NULL:
T→ Right = NULL;
}
}
Else if(X< T → Element)
T → Left = Insert ( X, T→ Left);
Else if(X> T → Element)
T → Right = Insert ( X, T→ Right);
// Else X is in the tree already
Return T;
}
```

Example:

As 5 < 8, Traverse towards left

10 > 8, Traverse towards right
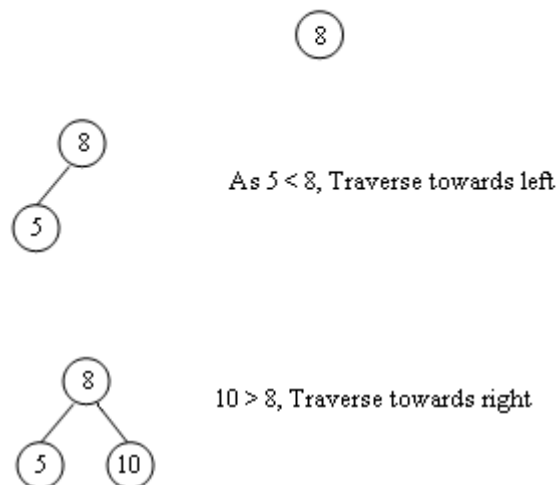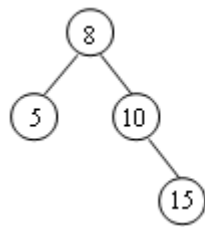
Similarly the rest of the elements are traversed
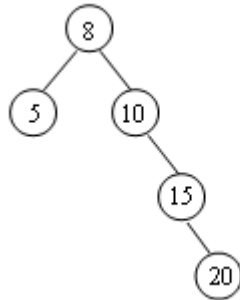
After 20
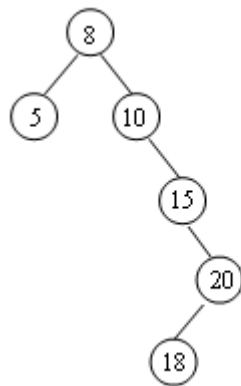
After 18

3)Find

1) Check whether the root is NULL if so then return NULL.

2) Otherwise,check the value X with root node value(ie)T→data

* If X is equal to T→datya,return T

* If x is less than T→data,

Traverse the left of T recursively.

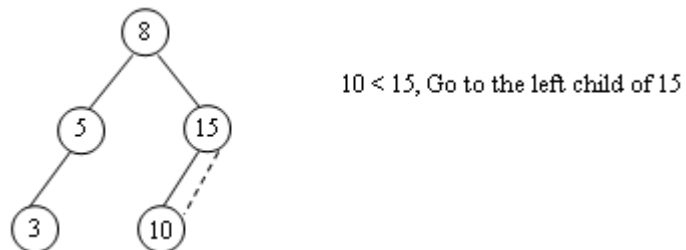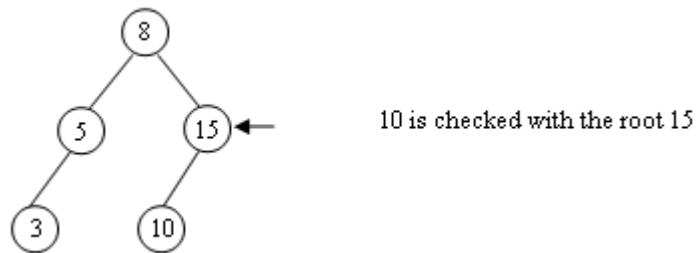* If x is greater than T→data,traverse the right of T recursively.

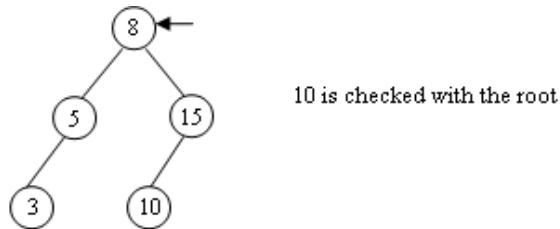### Routine For Find Operation

```
Int find(int X, SerachTree T)
{
If (T == NULL)
Return NULL:
If( X < T → Element)
```
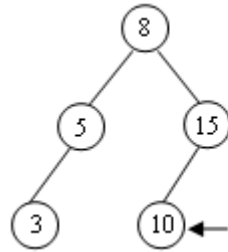
> Return Find (X , T → Left);
> Else if (X > T→ Element)
> Return Find (X, T→ Right);
> Else
> Return T;   // returns the position of the search element
> }

Example:

➢ To find an element 10 (consider, X= 10)

10 is checked with the root

10 > 8, Go to the right child of 8

10 is checked with the root 15

10 < 15, Go to the left child of 15
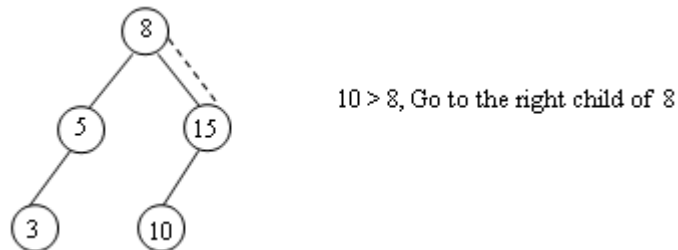
10 is checked with the root 10 (Found)

4) FindMin

➤ This operation returns the position of the smallest element in the tree. To perform FINDMIN, start at the root and go left as long as there is a left child. The stopping point is the smallest element.
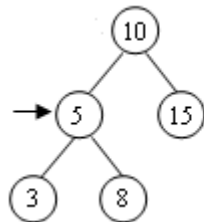
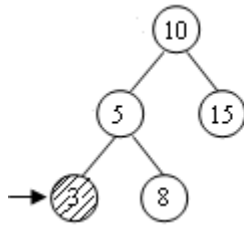**Recursive Routine For Findmin Operation**

```
Int FindMin(int X, SerachTree T)
{
If (T == NULL)
Return NULL;
Else If( T → Left = = NULL)
Return T;
Else
Return  FindMin ( T→ Left);
}I
```
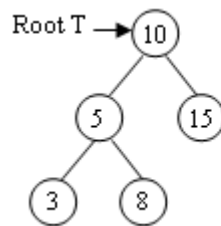

T!= NULL and T--> left != NULL, Traverse left

T != NULL and T --> left != NULL, Traverse left

Since T --> left is NULL, Return T as a minimum element

Non Recursive Routine For Findmin Operation

```
Int FindMin(int X, SerachTree T)
{
If (T != NULL)
while( T → Left1! = NULL)
T = T → Left;
Return  T;
}I
```
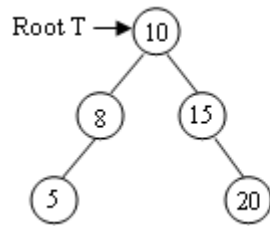
5) Find Max:

> Find max routine return the position of largest element in the tree. To perform a FINDMAX start at the root and go right as long s there is a right child.The stopping point is the largest element.
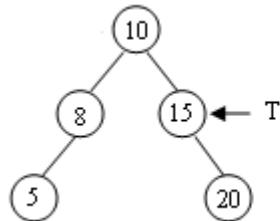
Recursive Routine For Findmax Operation

```
Int FindMax(int X, SerachTree T)
{
If (T == NULL)
Return NULL;
Else If( T → Right = = NULL)
Return T;

Else
Return  FindMax ( T→ Right);
}I
```
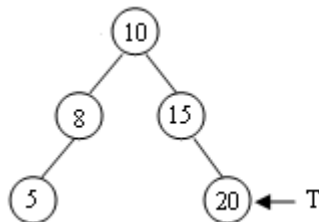
Root T → (10)
(8) (15)
(5) (20)

T != NULL and T --> Right != NULL, Traverse Right

(10)
(8) (15) ← T
(5) (20)

T != NULL and T --> Right != NULL, Traverse Right

(10)
(8) (15)
(5) (20) ← T

Since T --> Right is NULL, Return T as a Maximum Element

Non Recursive Routine For Findmin Operation

```
Int FindMax(int X, SerachTree T)
{
If (T != NULL)
while( T → Right1! = NULL)
T = T → Right;
Return   T;
}I
```

6) DELETE

➢ Deletion peartion is a complex operation in the binary search tree. To delete   an element consider the following three possibilities

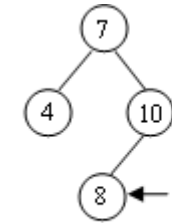*Case 1: Node to be deleted is a leaf noe(ie)No children*

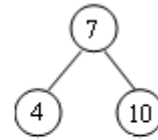*Case 2 : Node with one child*

Case 3 : Node with two children

Case 1   : Node with no children

➢ If the node is a leaf node,it can be deleted immediately.
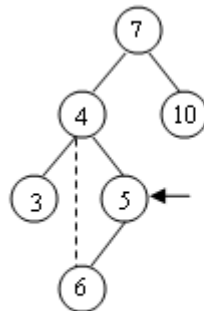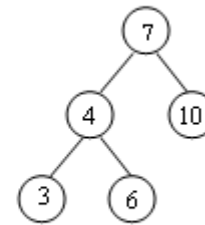
Delete 8:

Before Deletion

After Deletion

Case 2: Node with one child

➢ If the node has one child ,it can be deleted by adjusting its parant pointer that points to its child
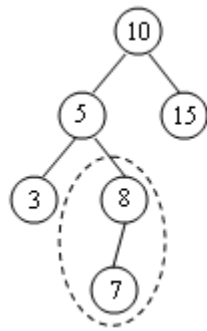node.

Delete 5

Before Deletion

After Deletion

Case 3: Node with two children

➢ It is difficult to delete a node which has two children. The general strategy is to replace the data of node to be deleted with its smallest data of the right subtree and recursively delete that node
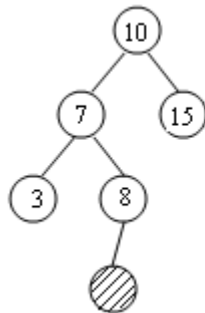
Deletion Routine for Binary Search Trees:

```
SerachTree Delete (int X, SearchTree T)
{
Int Tmpcell;
If( t == NULL)
Error ( " Element not found');
Else
If(X < T → Element) // Traverse towards left
T → Left = Delete ( X, T → Left);
Else if( X > T → Element) // Traverse towards right
T --.> Right= Delete ( X , T→ Right);
};
```
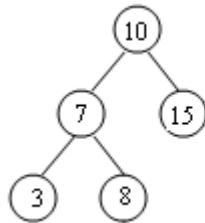
To delete 5

The maximum element at the right subtree is 7.

- Now the value 7 is replaced in the position of 5.
- Since the position of 7 is the leaf delete immediately

After deleting the node 5.

Q8) a) Write down the applications of Trees.(2)
ANSWER:

　　　1) A binary tree is used for manipulation of arithmetic expression.

　　　2) A binary search tree is an appropriate data structure for searching a value and sorting the records.

　　　3) It is also used for construction and maintenance of symbol tables.

　　　4) It plays important role in the area of syntax analysis.

　　　5) It is also used for playing games such as tic-tac-toe, chess, etc .

　　　6) Trees are used in data compression techniques for encoding the compressed data.

Q9) a) Explain clearly the logic behind using the THREADED BINARY TREES in Dta Structure.Draw a labeled diagram to show the working of the Threaded Binary Tree.(15)
Or
Q9) b) What is the need for Threaded Binary Tree.Explain it witn an examples(15)

ANSWER:

Need For Threaded Binary Tree:

In a Binary Tree or binary search tree, there are many nodes that have an empty left child or empty right child or both.• You can utilize these fields in such a way so that the empty left child of a node points to its inorder predecessor and empty right child of the node points to its inorder successor.

Threaded Binary Tree definition

The structure of a node in a threaded binary tree is a bit different from that of a normal binary tree.

Unlike a normal binary tree, each node of a threaded binary tree contains two extra pieces of information, namely left thread and right thread.

We have the pointers reference the next node in an inorder traversal; called threads The left and right thread fields of a node can have two values:

1: Indicates a normal link to the child node

0: Indicates a thread pointing to the inorder predecessor or inorder
   successor

Two types of Threading

- ✓ *One way Threading*
- ✓ *Two way Threading*

1) One Way Threading:

Thread appears only on the RLINK of a node, pointing to the inorder successor of the node.

2) Two Way Threading

Threads are appear in both the links LLINK and RLINK and points to the inorder predecessor and inorder successor respectively.

Logic Explanation:

➢ In threaded binary tree,the NULL pointer are avoided.Instead of left NULL pointer the link points to inorder predecessor of that node. Similarly, instead of right NULL pointer the link points to the inorder successor of that node.

➢ There are two additional fields in each node named as LThread or RThread these fields indicates whether left or right thread is present. Thread present means the NULL link is replaced by a link to inorder predecessor or inorder successor. To represent that there exists thread the Lth or Rth fields are set to 0.

➢ The basic idea in inorder threading is that the left thread should point to the inorder predecessor and the right thread points to the inorder successor. Here we assuming the head(dummy) node as the starting node and the root node of the tree is attached to left of head(dummy) node.

Node representation:

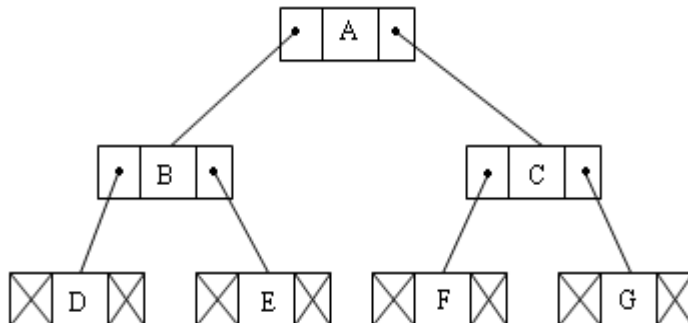| TLPOINT	LLINK	DATA	RLINK	TRPOINT

Example :

Consider the following Binary Tree and create Threaded Binary Tree
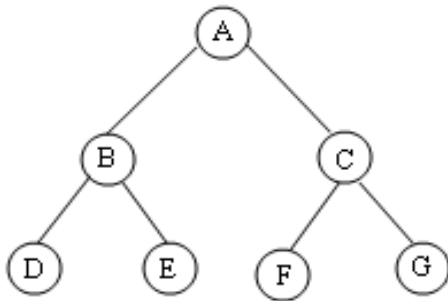
Solution:

Step1: Represent the binary tree by using linked list representation.

Step 2:Inorder Traversal of Binary Tree

D B E A C F G

Step 3: Nodes D, E, F and G contain left and right null poniters. In threaded binary tree,the NULL pointer are avoided.So find inorder predecessor and inorder successor

1) Inorder predecessor of D is dummy node.

Inorder successor of D is B

2) Inorder predecessor of E is B.

Inorder successor of E is A

3) Inorder predecessor of F is C.

Inorder successor of F is G

4) Inorder predecessor of G is F

Inorder successor of G is dummy node

Step4: .Instead of left NULL pointer the link points to inorder predecessor of that node. Similarly, instead of right NULL pointer the link points to the inorder successor of that node.Hence we will get the theaded binary tree as follows
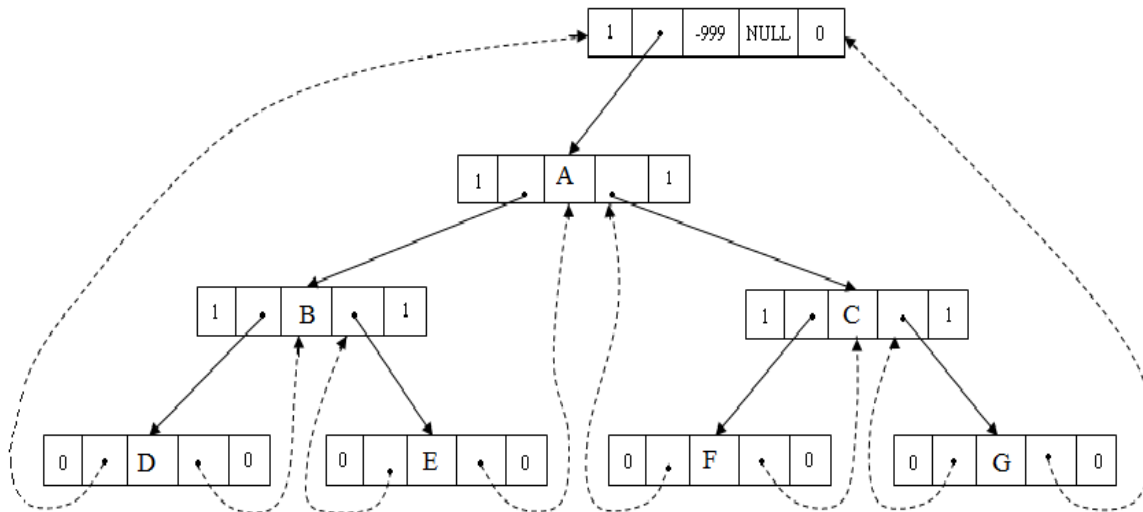
Advantages:

1) IN THREADED BINARY TREE is avoid NULL pointer. Hence memory wastage in occupying predecessor and successor nodes.

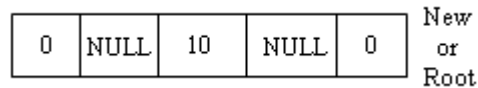**Q10) a) Create a threaded binary tree for the nodes having values. 10,8,6,12,9,11,14(13)**

 ANSWER

1) *initially we will create a dummy node which will act as a header of the tree.*
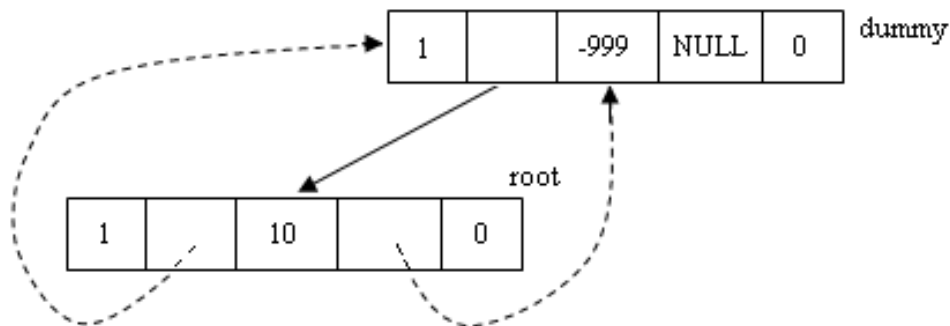
2) Now let us take first value(ie)10.The node with value 10 will be the root node.we will attach this root node as left child of dummy node.

| 0 | NULL | 10 | NULL | 0 | New or Root |

**3) The NULL links of root's left and right child will be pointed to dummy.**

4) Now next comes 8.The 8 will be compared with 10,as the value is less than 10,we will attach 8 as left child of 10 and we will set root's Lth field to 1,indicating that the node 10 is having the child. Then links are arranged are,
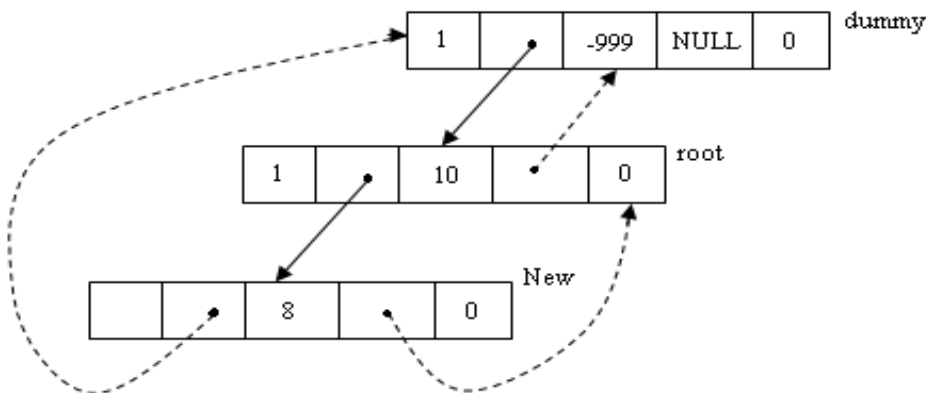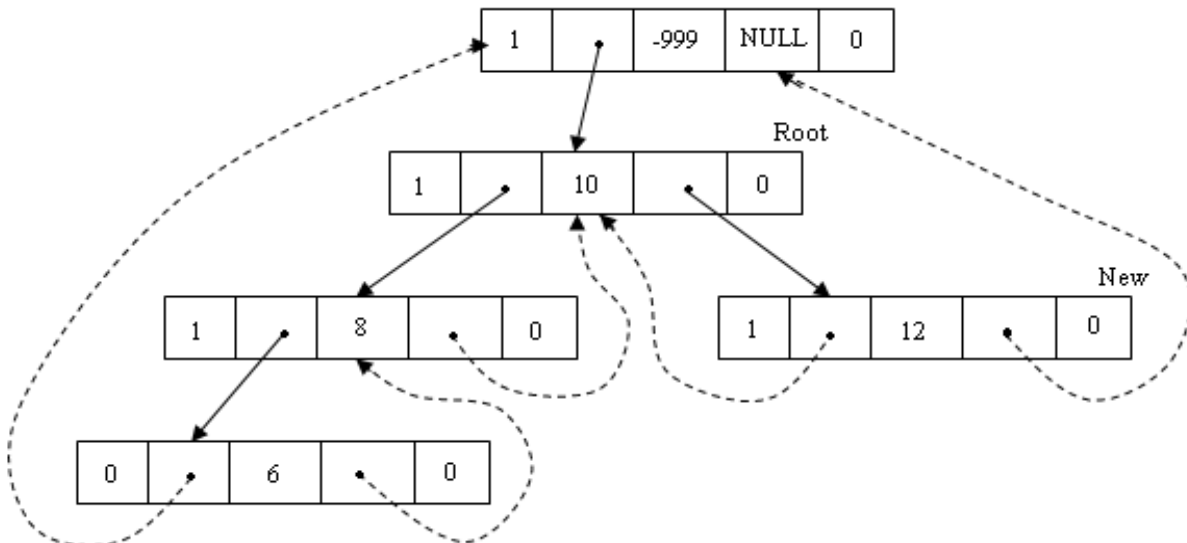

Note:

The left link of node 8 points to its inorder predecessor and right link of node 8 points to its inorder successor.

5) Next comes 6.The value 6 will be first compared with root(ie) with 10. As 6 is less than 10,we will move on left subbranch. The 6 will then be compared with 8,so we have to move on left subbranched of 8 but lth of node 8is 0,indicating that there is no left child of 8,so we will attach 6 as left child to 8.
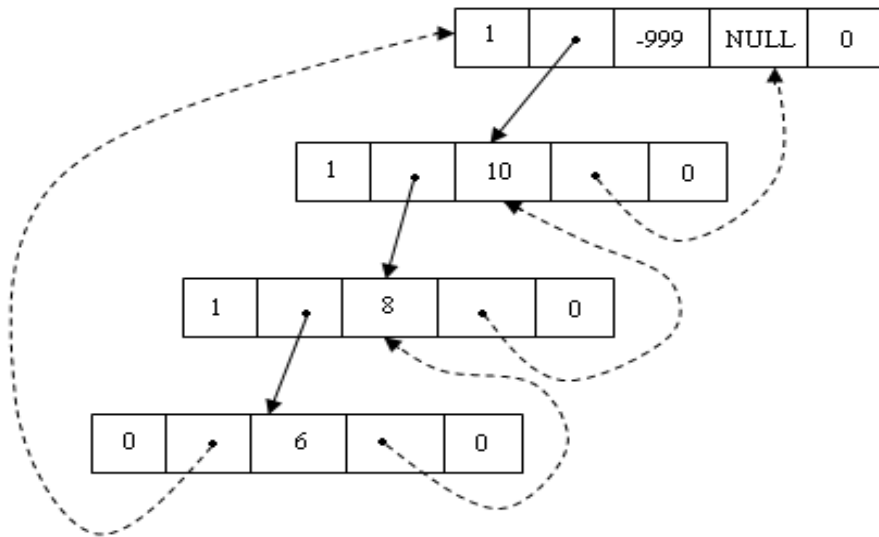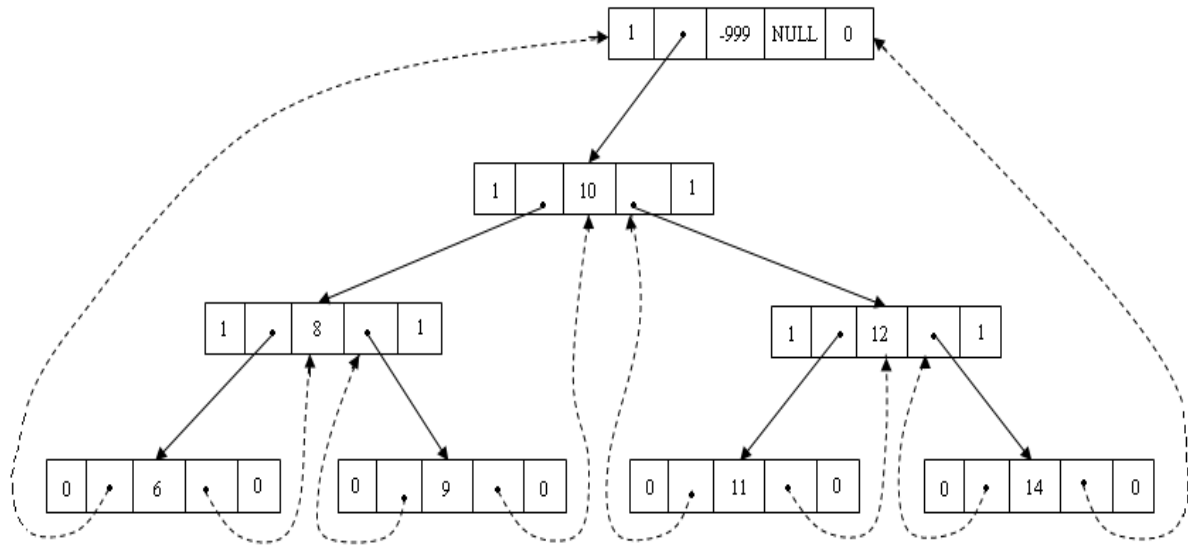
6) Then comes 12.We will compare 12 with10.As 12 is greater than 10,we will attach the node 12 as right child of 10.

Note that the left field of node 12 points to its inorder predecessor and right field of node 12 points to its inorder successor.

7) Thus by attaching 9,11,14 as appropriate children, the threaded binary tree will look like this

27

Q11) a) Define AVL Tree. Explain its rotation operations with example.(15)

Or

Q11) b) Show the results of inserting43, 11, 69, 72 and 30 into an initially empty AVL tree.

ANSWER:

AVL TREE:

➢ An AVL (Adelson –Velskii and Landis) tree is a binary search tree with a ***balance condition***. The ***balance information*** is termed as **Balance Factor (BF)**, it is the height of the left sub tree minus height of the right sub tree.

> **BF=Height of left sub tree-Height of right sub tree**

➢ For an AVL tree all ***balance factor*** value will be ***+1,0,-1.***
➢ For example, consider the AVL tree. The balance factor of each node will be calculated as follows,


28
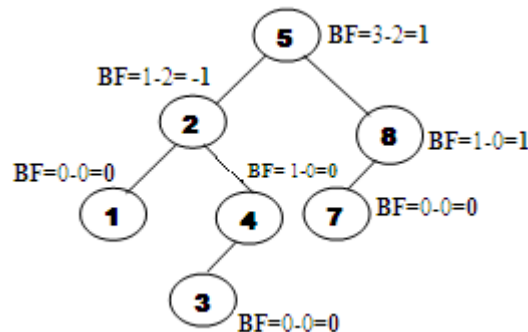
Balanced AVL tree with Balance Factor Value

- ➤ (ie) An AVL Tree is identical to a binary search tree, except that for every node in the tree, must have balanced factor of 1 or 0 or -1
- ➤ Insertion in AVL tree is performed in the same way as it is performed in a binary search tree. However, it may lead to violation in the AVL tree property and therefore the tree may need balancing. The tree can be balanced by applying rotations.
- ➤ Then any one of the following conditions would occur due to an insertion in node *a*.
  - ❖ **CASE 1:** *An insertion into the left subtree of the left child of node a.*
  - ❖ **CASE 2:** *An insertion into the right subtree of the left child of node a.*
  - ❖ **CASE 3:** *An insertion into the left subtree of the right child of node a.*
  - ❖ **CASE 4:** *An insertion into the right subtree of right child of node.*
- ➤ To rebalance the AVL tree we need to perform rotation on the imbalanced subtree. There are two types of rotation are available
  - • *Single Rotation*
  - • *Double Rotation*
- ➤ For case 1 and case 4 we need to perform single rotation. And for case 2 and case 3 we need to perform double rotation

SINGLE ROTATION:
- ➤ There are two types of single rotations
  1. *Single rotation with Left (Case 1)*
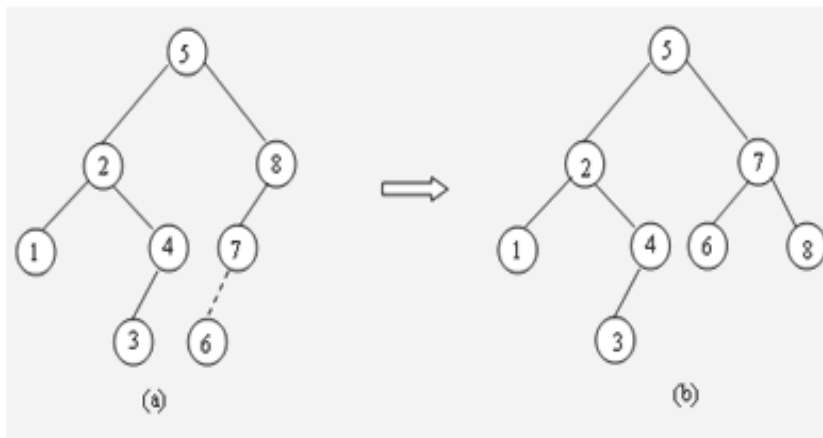 2. *Single rotation with Right (Case 4)*

SINGLE ROTATION WITH LEFT:
General Representation of **Single rotation with Left(*case 1*):**

(a) Before Rotation   (b) After Rotation
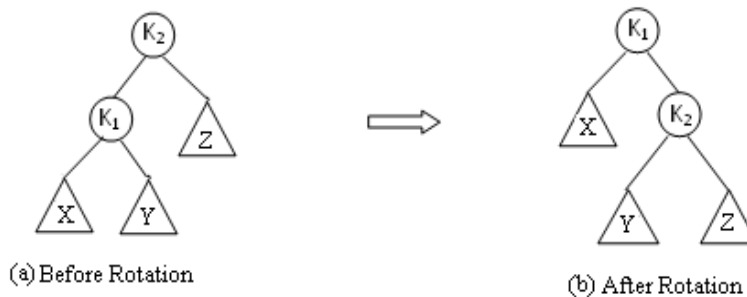
Routine to perform single rotation with left

```
Static Position  SingleRotateWithLeft (Position K2)
{
        Position K1;
        K1 = K2 → Left;
        K2 → Left = K1 → Right;
        K1 → Right = K2;
        K2 → Height = Max (Height (K2 → Left), Height(K2 → Right))+1;
        K1 → Height = Max (Height (K1 → Left), K2 → Height) +1;
        Return K1; /* New Root */
}
```

Example:-

| **Before Rotation** | **After Rotation** |

➢ In the above example, when we try to insert a new node **6** in an AVL tree, imbalance will occur at node **8** (BF=2-0=2). So rotation should be performed.

➢ Here imbalance is occurred due to the insertion at the left subtree of the left child of node **8**. So *single rotation with left should be performed*.

➢ Now we want to rotate the subtree a, based on this the node **7** should be placed as root and the node **8** should be placed as right child of 7.

SINGLE ROTATION WITH RIGHT :
➢ An insertion into the right subtree of the right child of K1

General Representation of Single rotation with Right (Case 4):

(a) Before Rotation   (b) After Rotation
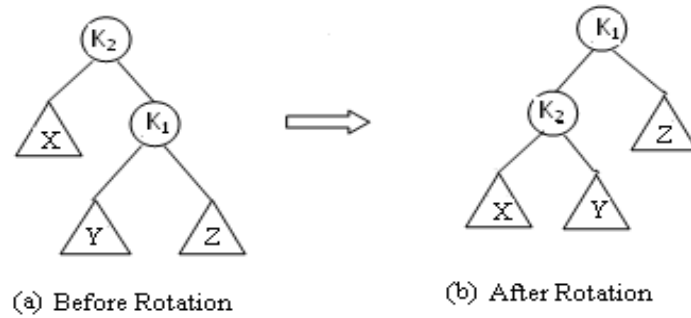
Figure 3.7: Single rotation with Right

<u>Routine to perform single rotation with right</u>

Static Position SingleRotateWithRight(Position K2)
{
    Position K1;
    $K_1 = K_2 \rightarrow Right;$
    $K_2 \rightarrow Right = K_1 \rightarrow Left;$
    $K_1 \rightarrow Left = K_2;$
    $K_2 \rightarrow Height = Max(Height(K_2 \rightarrow Left), Height(K_2 \rightarrow Right))+1;$
    $K_1 \rightarrow Height = Max(Height(K_1 \rightarrow Right), K_2 \rightarrow Height)+1;$
   *return K1; /* New Root */*
}

Example:-

➢ Inserting node **13** in the following AVL tree will leads to imbalance (violate AVL tree property) at the node **6 (BF= -2)**. So we want to rebalance it by performing rotation.

➢ As the imbalance happen due to the insertion at the *right subtree of the right child of 6,* we want to perform *single rotation with right*.

➢ Now if we rotate the tree in single rotation with right, we get the balanced AVL tree as follows
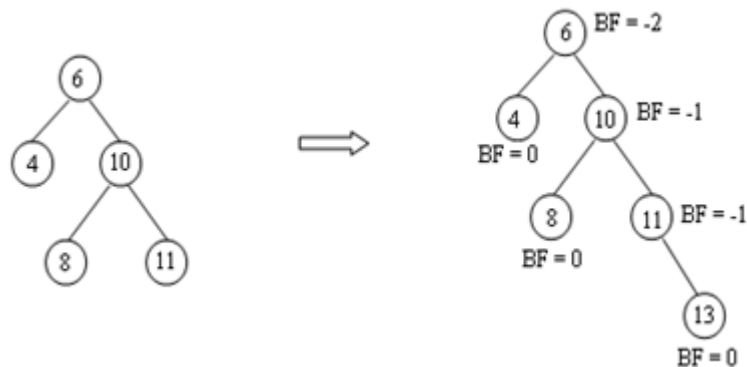
(a): AVL tree with imbalance

Rotation with Right
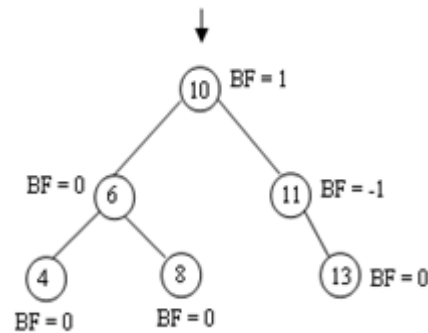
(b) Balanced AVL Tree

DOUBLE ROTATION:

➢ Double rotation is performed for case 2 and case 3.ie) imbalance due to insertion at the left subtree of the right child of node **a** and vice-versa**.**

➢ There are two types of double rotation. They are,

▪ *Left-Right Double Rotation (Case 2)*
▪ *Right-Left Double Rotation (Case 3)*

LEFT – RIGHT DOUBLE ROTATION:

➢ The Left-Right double rotation is performed, when an imbalance occur due to the insertion at the *right subtree of the left child of an node.*

General Representation of Left-Right Double Rotation:

(a) Before Rotation   (b) Afer Rotation
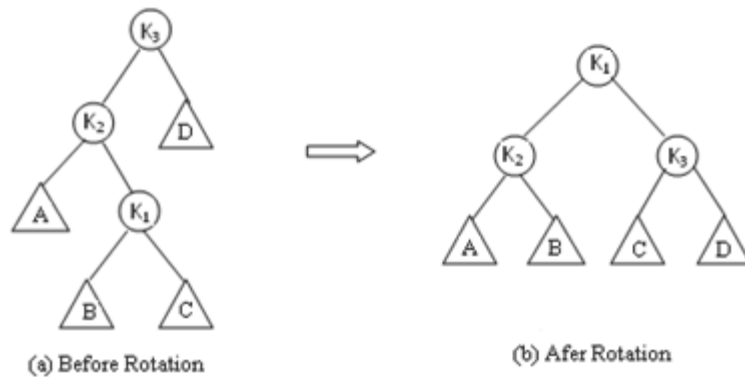
Figure 3.9: Left-Right Double Rotation

➢ The above double rotation can be done by using the following two single rotation,

1. Single rotation with right, then
2. Single rotation with left.

***Routine to perform Left-Right Double Rotation:***

Static Position DoubleRotateWithLeft (Position K3)
{
       /* Rotate between K1 and K2 */
       K3 → Left = SingleRotateWithRight(K3→Left);
       /* Rotate between K3 and K1 */
       return SingleRotateWithLeft(K3);

}

Example:-

➢ Consider the following AVL tree in the figure 4.10.(a). If we insert **20** in the tree, then imbalance will occur at the node **22** as follows
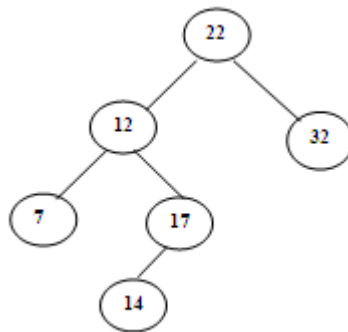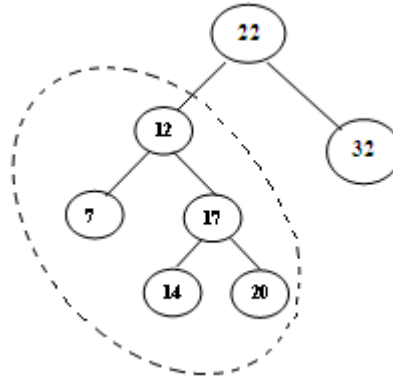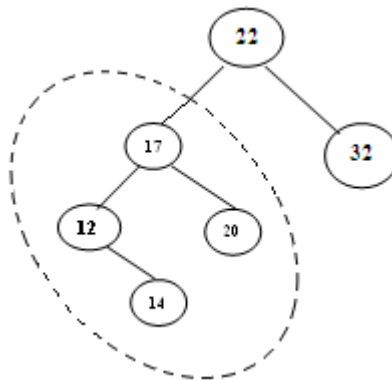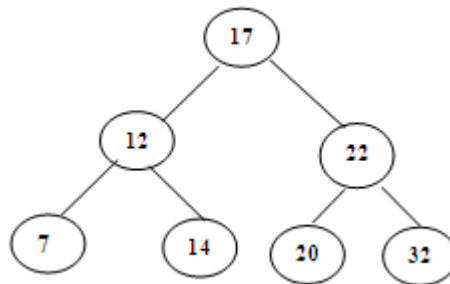

AVL Tree
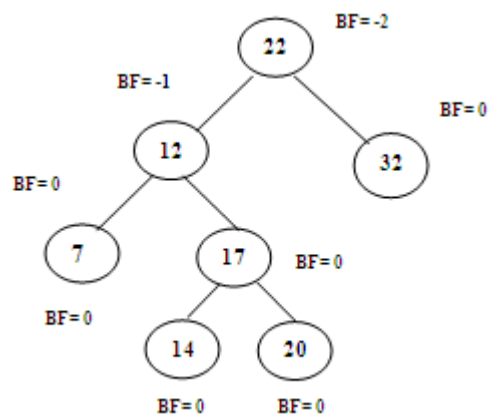
Imbalanced AVL Tree after inserting 20.

➢ After performing single rotation with right on the subtree we get the intermediate result as shown below,

After single rotation with Right

➢ Now perform single rotation with left on the above tree. The values are, **K1= 17, K3= 22, D= 32.** We get the balanced AVL Tree as in figure 4.10 (d)
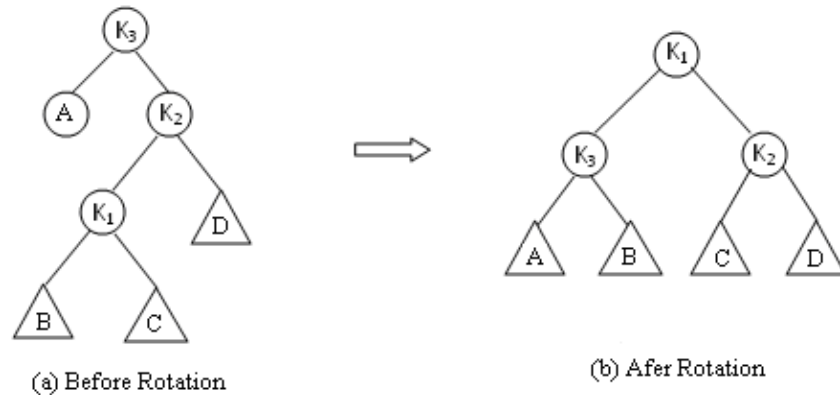

After Left-Right Double Rotation

RIGHT – LEFT ROTATION:

➢ The Right-Left double rotation is performed, when an imbalance occur due to the insertion at the *left subtree of the right child of an node.*

General Representation of Right- Left Double Rotation:

(a) Before Rotation   (b) Afer Rotation

General Representation of Right- Left Double Rotation

➢ The above double rotation can be done by using the following two single rotation,

       1. Single rotation with left, then

   2. Single rotation with right.

Routine to perform Right-Left Double Rotation:

> *Static Position DoubleRotateWithRight (Position K3)*
> *{*
>   */* Rotate between K1 and K2 */*
> *K3 → Right = SingleRotateWithLeft(K3→Right);*
> */* Rotate between K3 and K1 */*
> *return SingleRotateWithRight(K3);*
> *}*

➢ For example, consider the following AVL tree in figure 4.12 (a), if we insert a node **13** in that AVL tree it will become imbalanced it node **12** as shown in figure 4.12 (b)

     (a)   **(b)**

    AVL tree   **Imbalanced AVL Tree.**
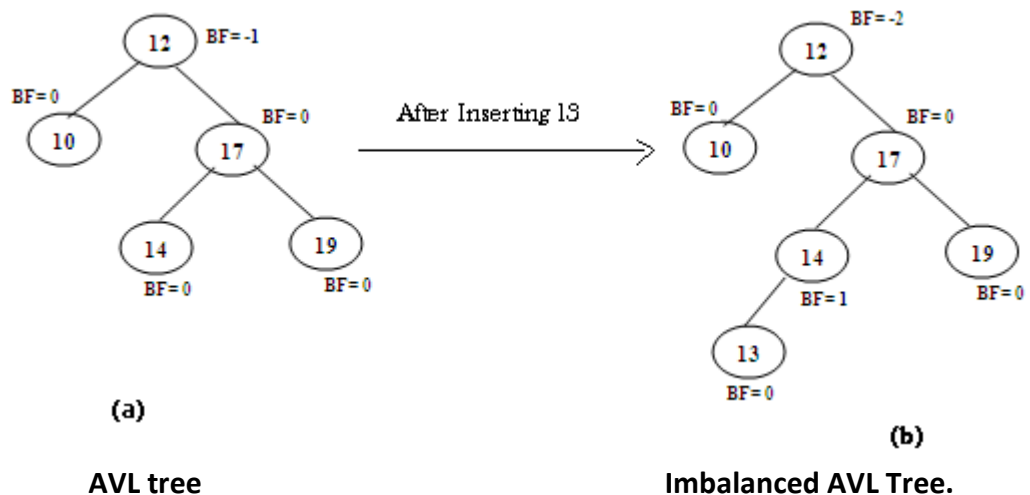
➢ To make it as a balanced tree, perform Right-Left double rotation.
➢ For that, first perform single rotation with left on the subtree (17, 14, 19, 13), in which the vaules of keys are, **K2= 17, K1= 14, B= 13, D=19,** and no value for **C.**
➢ After the single rotation with left, we obtain the tree as,
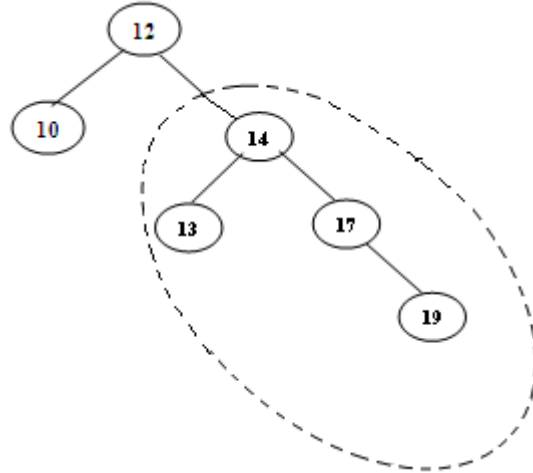
➢ The balanced AVL tree will be as follows
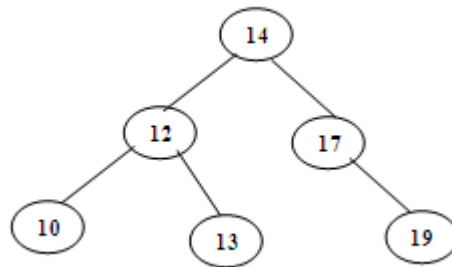
AVL tree after Right-Left double rotation.

Q12) Show the results of inserting 2, 1, 4, 5, 9, 3, 6, 7into an initially empty AVL tree.
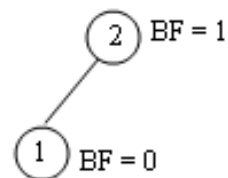**ANSWER:**
Example
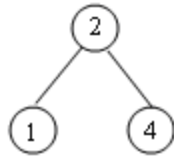Let us consider how to balance a tree while inserting the numbers 2, 1, 4, 5, 9, 3, 6, 7
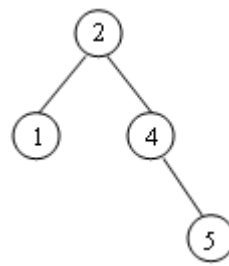
1. Insert the value 2   **2. Insert 1**
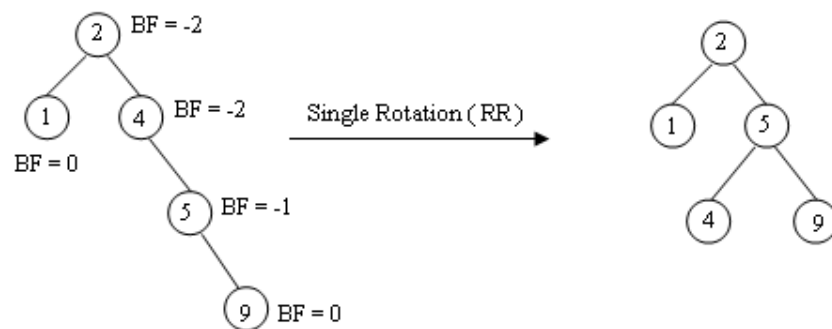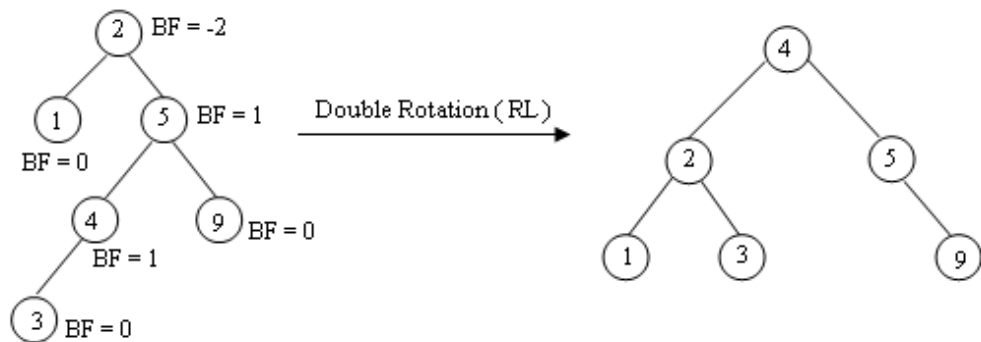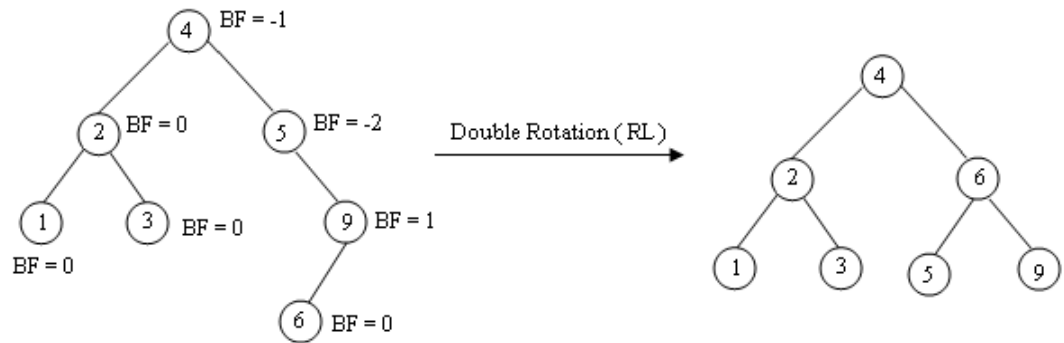
3. Insert 4
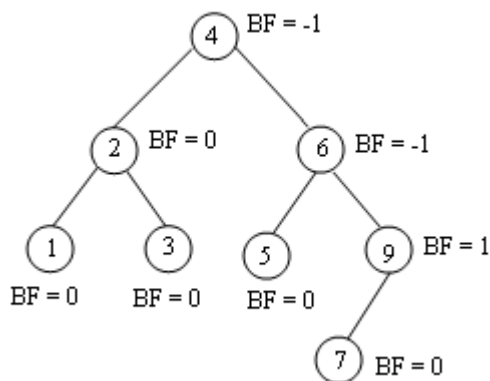

4. Insert 5

5. Insert 9

6. Insert 3

7. Insert 6

8. Insert 7

Q13)a) Discuss in detail the B -tree. What are its advantages? (15)

Or

**Q13) b) Explain the oprations which are done in B Tree with examples.(15)**

ANSWER

B Tree

➤ B-tree is a specialized multiway tree used to store the records in a disk. There are number of subtrees to each node. So that the height of the tree is relatively small. So that only small number of nodes must be read from disk to retrieve an item. The goal of B-trees is to get fast access of the data.

➤ A multiway search tree of order m is an ordered tree where each node has at the most m children. If there are n number of children in a node then (n — 1) is the number of keys in the node.

For example:

➤ Following is a tree of order 4.
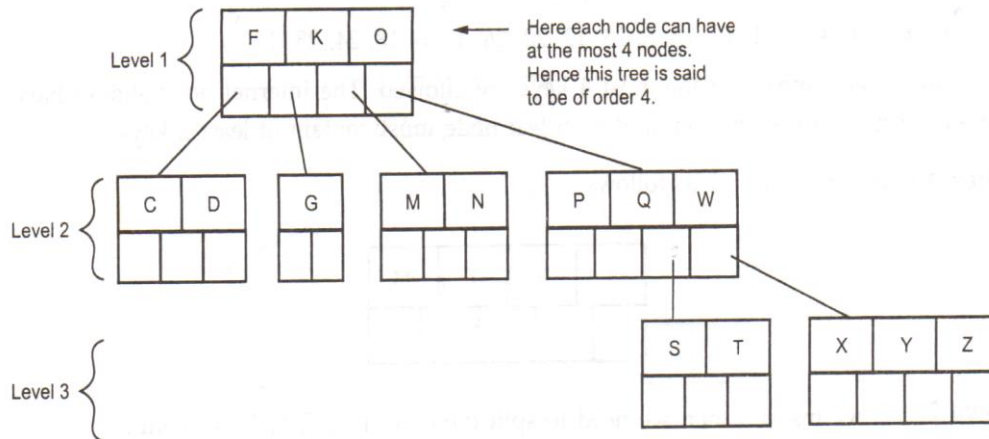
➢ From above tree following observations can be made -

1. The node which has n children posses (n - 1) keys.

2. The keys in each node are in ascending order.

3. For every node Node.child [0] has only keys which are less then Node. key [0] similarly, Node. child [1] has only keys which are greater than Node. key [0]

➢ In other words, the node at level 1 has F, K and 0 as keys. At level 2 the node containing keys C and D are arranged as child of key F (the cell before F). Similarly the node containing key G will be child of F but should be attached after F and before K, as G is between F and K. Here the alphabets F, K, 0 are called keys and branches are called children.

Properties of B+ Tree

- All the leaf nodes are on the botom level.

- The root node should have at least two children.

- All the internal nodes except root node have at least ceil (m/2) nonempty children. The ceil is a function such that ceil (3.4) = 4, ceil (9.3) = 10, ceil (2.98) = 3 ceil (7) = 7.

- Each leaf node must contain at least ceil (m/2) — 1 keys.

Thus the B-tree is fairly balanced tree
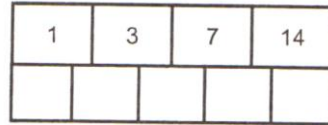
Insertion Example of B-tree

Construct a B-tree of order 5 by following numbers.

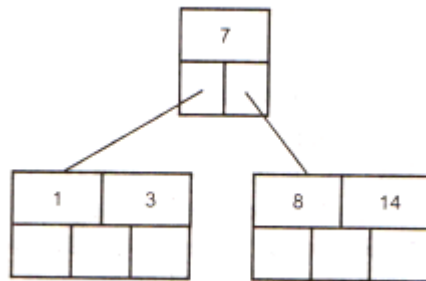➢ 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19.

Solution:

➤ The order 5 means at the most 4 keys are allowed.
➤ The internal node should have at least 3 non empty children and each leaf node must contain at least 2 keys.

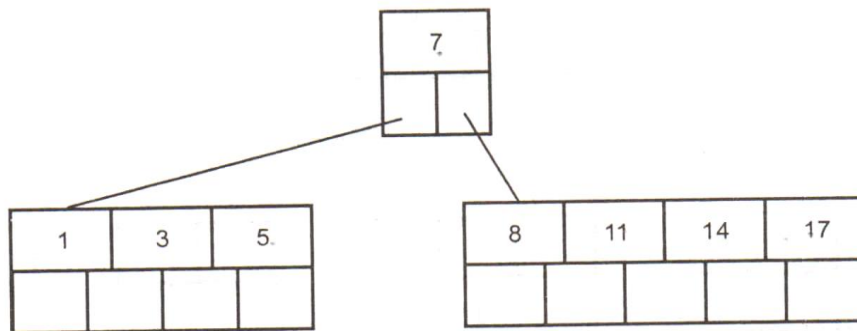**Step 1 :** Almost 4 keys are allowed.So Insert 3, 14, 7, 1 as follows

1	3	7	14

Step 2 : If we insert 8 then we need to split the node 1, 3, 7, 8, 4 at medium. Hence,
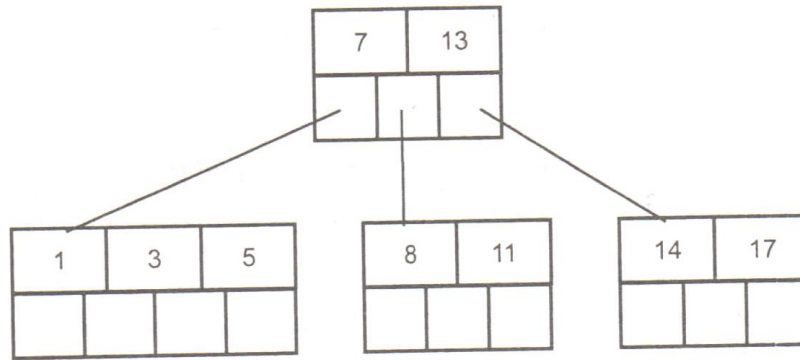

Here, 1 and 3 are smaller than 7.So they are in left and then 8 and 14 are greater than 7.So they are in right.
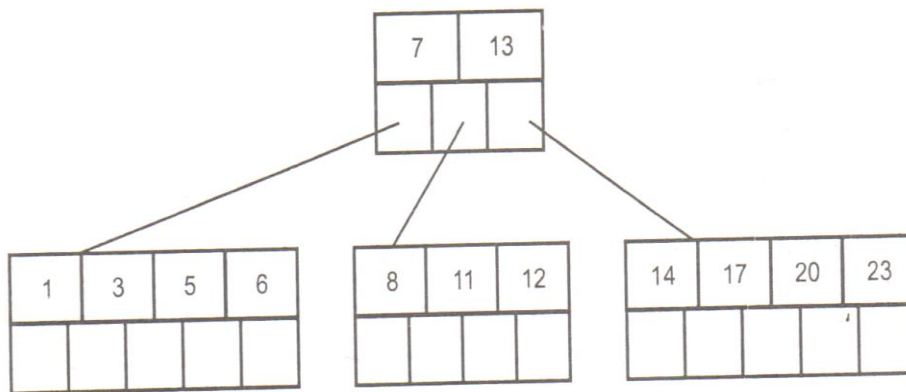
Step 3 : Insert 5, 11, 17 which can be easily inserted in a B-tree.Hence,
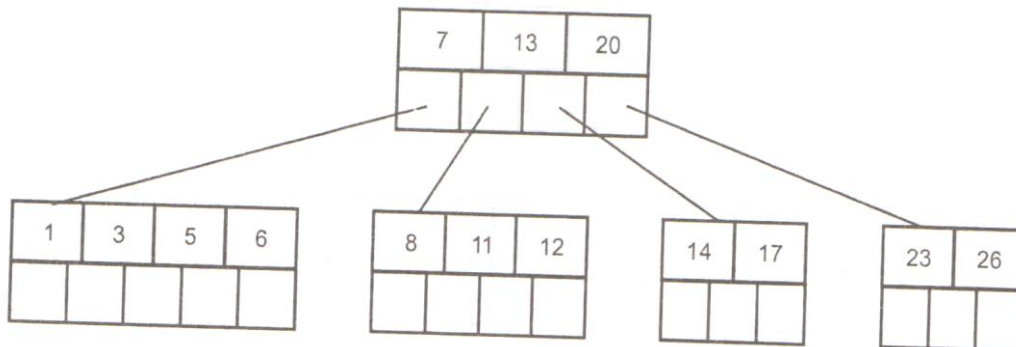

Step 4 : Now insert 13. But if we insert 13 then the leaf node will have 5 keys which is not allowed. Hence 8, 11, 13 , 14, 17 is split and medium node 13 is moved up.Hence,

Step 5 : Now insert 6, 23, 12, 20 without any split as folows

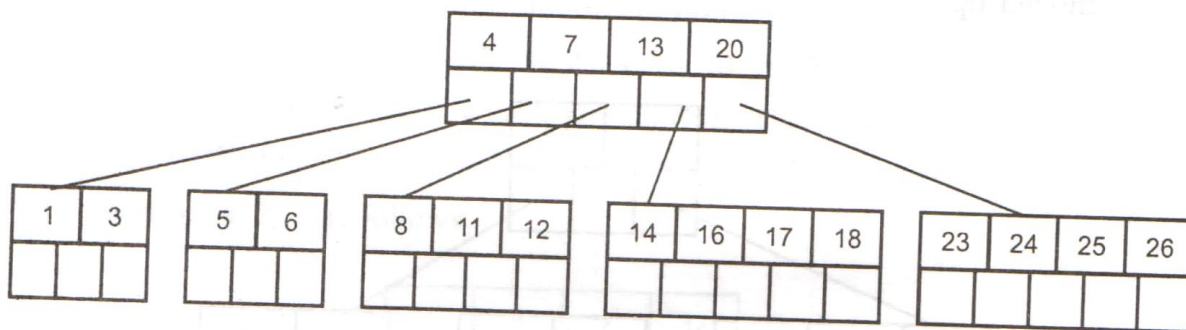**Step 6:** Now insert 26.If we insert 26 in rightmost leaf node then it will have 5 keys which is not allowed.Hence 14, 17, 20, 23, 26 the node is split and 20 will be moved up.

Step 7: Now insert 4.If we insert 4 in leftmost leaf node ,then it will have five keys which is not allowed.Hence, 1, 3, 4 , 5, 6 is split and medium node 4 is moved up and then insert 16, 18, 24, 25 without any split.

Step 8: Finally insert 19. If we insert 19 after 18 means that node contains 5 keyswhich is not allowed.Hence 14,16,17,18,19 is split and medium node 17 is moved up and the root node contains 5 keys which is not allowed.So split the root node 4,7,13,17,20 at medium and 13 is moved up as follows.
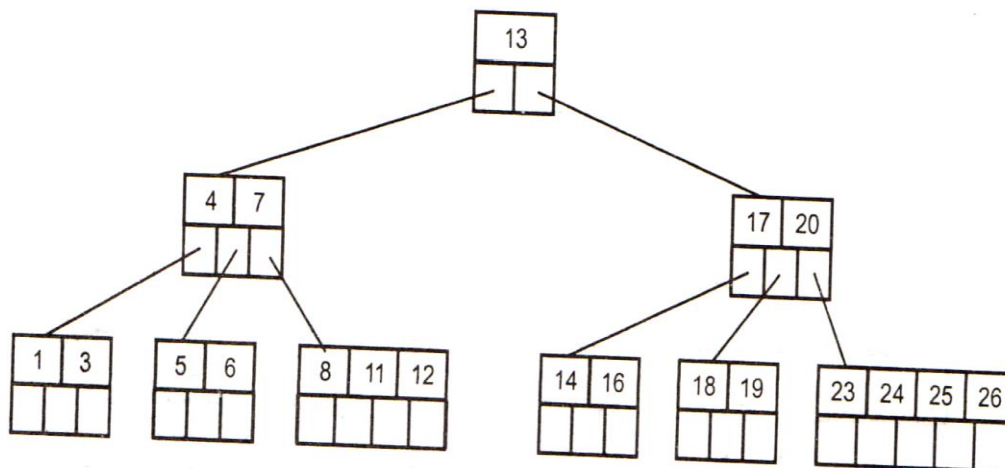
➢ Thus the B-tree of order 5 is constructed.

DELETION

➢ Consider a below B-tree and delete 8,20,18,5

Step 1:To delete 8, simply delete it from leaf node.Hence,

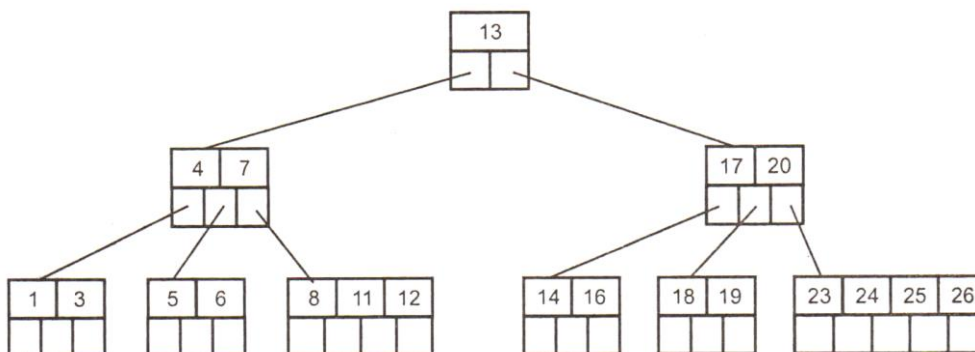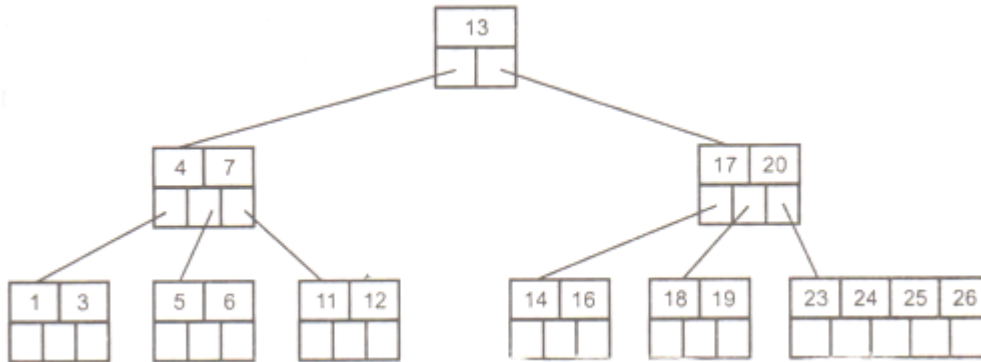Step2 : Now we will delete 20, the 20 is not in a leaf node so we will find successor of 20 which is 23. Hence 23 will be moved up to replace 20.Otherwise that node contain only one key 17.
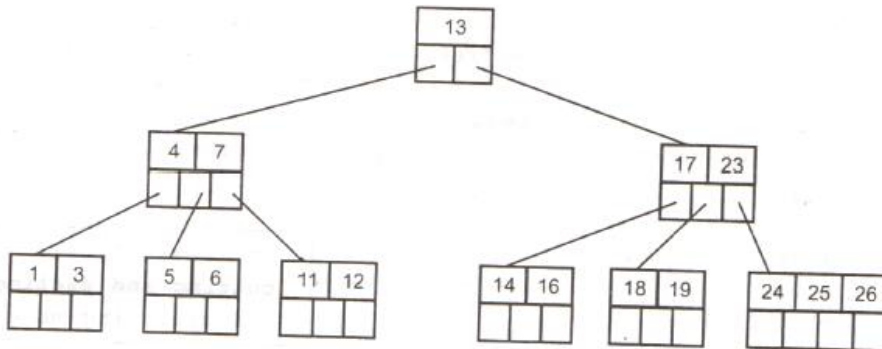
Step 3:Next we will delete 18. Deletion of 18 from the corresponding node causes the node with only one key, which is not allowed (as per property 4) in B-tree of order 5.

So, The sibling node to immediate right has an extra key(it contain 3 keys). In such a case we can borrow a key from parent and move spare key of sibling node to moved up (ie) move 24 up. See the following figure.

.
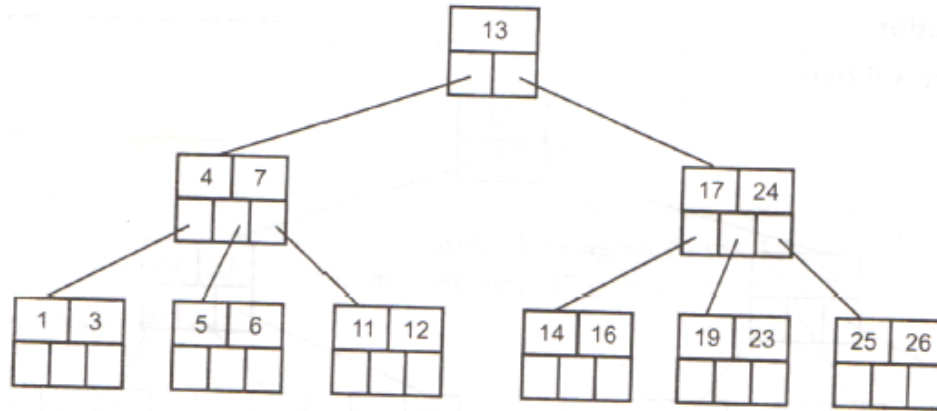
Step 4: Now delete 5. But deletion of 5 is not easy. The first thing is 5 is from leaf node. Secondly this leaf node has no extra keys nor siblings to immediate left or right. In such a situation we can combine this node with one of the siblings

➢ That means remove 5 and combine 6 with the node 1, 3. To make the tree balanced we have to move parent's key down. Hence we will move 4 down as 4 is between 1, 3 and 6. Then the tree will be
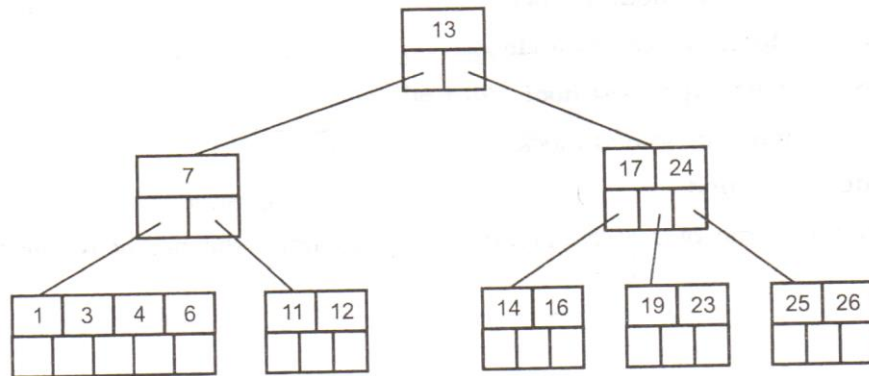
➢ But again internal node of 7 contains only one key which not allowed in B-tree (as per property 3). We then will try to borrow a key from sibling. But sibling 17, 24 has no spare key. Hence what we can do is that, combine 7 with 13 and 17, 24. Hence the B-tree will be
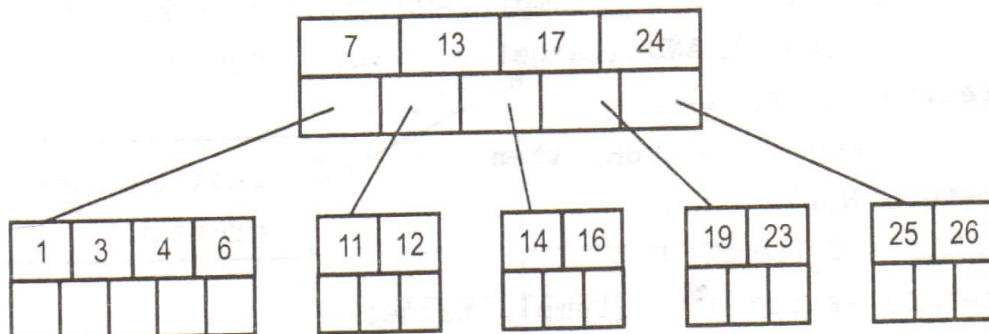
SEARCHING

➢ The search operation on B-tree is similar to a search on binary search tree. Instead of choosing between a left and right child as in binary tree, B-tree makes an m-way choice.

Example: Search 11 from a below B-tree
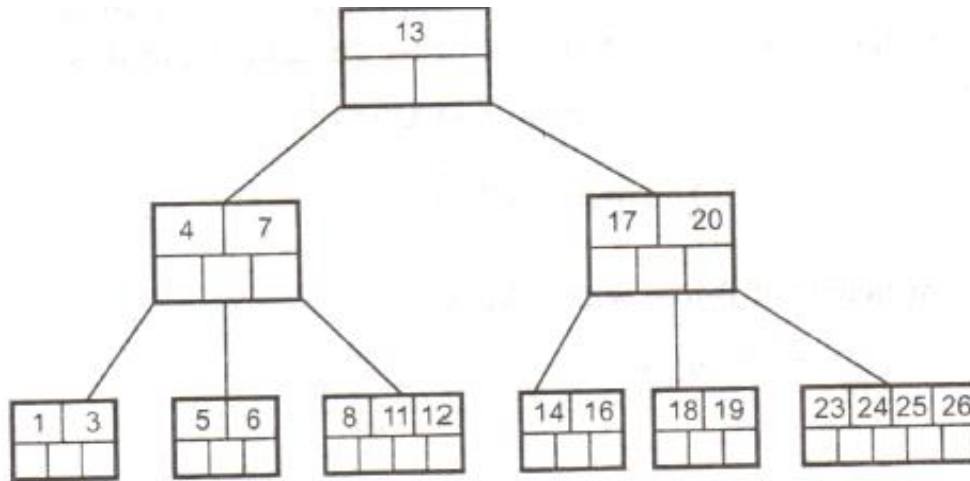
Solution: To search 11,

Step 1: Compare 11 and 13, 11 < 13 So go to left side nodes.

Step 2: Compare 11 and 7, 11 > 7 So go to right side nodes.

Step 3: Compare 11 nad 8, 11 > 8 So go to the second key.

Step 4: Second key 11.(ie) Found search element.

Viva QUESTIONS AND ANSWERS

1. Define tree?

Trees are non-liner data structure, which is used to store data items in a   shorted sequence. It represents any hierarchical relationship between any data   Item. It is a collection of nodes, which has a distinguish node called the root and zero or more non-empty sub trees T1, T2,....Tk. each of which are connected by a   directed edge from the root.

2. Define Height of tree?

The height of n is the length of the longest path from root to a leaf. Thus   all leaves have height zero. The height of a tree is equal to a height of a root.

3. Define Depth of tree?

For any node n, the depth of n is the length of the unique path from the root to node n. Thus for a root the depth is always zero.

4. Define Degree of a node?

It is the number of sub trees of a node in a given tree.

5. Define Degree of a tree?

It is the maximum degree of a node in a given tree.

6 .Define Terminal node or leaf?

Nodes with no children are known as leaves. A leaf will always have degree zero and is also called as terminal node.

7. Define Non-terminal node?

Any node except the root node whose degree is a non-zero value is called as a non-terminal node. Non-terminal nodes are the intermediate nodes in traversing the given tree from its root node to the terminal node.

8.Define sibling?

Nodes with the same parent are called siblings.

9.Define binary tree?

A Binary tree is a finite set of data items which is either empty or consists of a single item called root and two disjoin binary trees called left sub tree max degree of any node is two.

10. How to traverse a Tree?
- ➢ Traversing means visiting each and every nodes only once.
- ➢ Tree Traversal is a method for visiting all the nodes in the tree exactly once.
- ➢ There are three types of tree traversal techniques namely,
    - ✓ Inorder traversal
    - ✓ Preorder traversal
    - ✓ Postorder traversal
   - ✓

11. Define expression tree?
Expression tree is also a binary tree in which the leafs terminal nodes or operands and non-terminal intermediate nodes are operators used for traversal.

12.Construction of expression trees?
1.convert the given infix expression into postfix notation
2. Create a stack and read each character of the expression and Push into the stack, if operands are encountered.
3.when an operator is encountered pop 2 values from the stack. Form a tree using the operator.

13. Define Binary search tree.
- ➢ Binary search tree is a binary tree in which for every node X in the tree,the values of all the keys in its left subtree are smaller than the key value in X and the values of all the keys in its right subtree and larger than the key value in X.

.
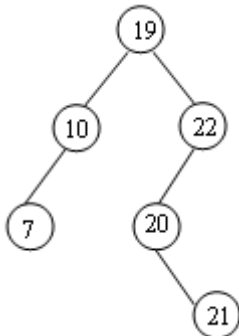
14. What are the various operation performed in the binary search tree ?

1.insertion

2.deletion

3.find

4.find min

5.find max

15. What is the Advantage of Threaded Binary Tree

1) in threaded binary tree is avoid NULL pointer. Hence memory wastage in occupying predecessor and successor nodes.

16.Define AVL tree?

AVL tree also called as height balanced tree .It is a height balanced tree in which every node will have a balancing factor of –1,0,1.

17. What do you mean by rotations in AVL Tree and what are the types?

The property can be restored with a simple modification in the tree known as *ROTATION*

There are two types of single rotations

Single rotation with Left (Left-Left rotation)

Single rotation with Right (Right to Right rotation)

There are two types of double rotation. They are,

Left-Right Double Rotation

Right-Left Double Rotation

18. Define balance information and balance factor.

➢ The *balance information* is termed as **Balance Factor (BF)**, it is the height of the left sub tree minus height of the right sub tree.

> **BF=Height of left sub tree-Height of right sub tree**

➢ For an AVL tree all *balance factor* value will be *+1,0,-1.*

➢ For example, consider the AVL tree in figure 4.2(a). The balance factor of each node will be calculated as follows,

.
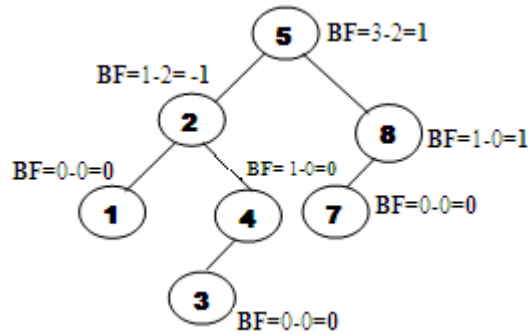
19.Define B tree:

➢ B-tree is a specialized multiway tree used to store the records in a disk. There are number of subtrees to each node. So that the height of the tree is relatively small. So that only small number of nodes must be read from disk to retrieve an item. The goal of B-trees is to get fast access of the data.

Multiway search tree

➢ A multiway search tree of order m is an ordered tree where each node has at the most m children. If there are n number of children in a node then (n — 1) is the number of keys in the node.

20. what is priority queue?

A priority queue is a data structure that allows at least the following   two operations: insert which does the obvious thing; and Deletemin, which finds, returns, and removes the minimum element in the priority queue. The Insert operation is the equivalent of Enqueue