SMVS
Page No.
Date: / /

* Tree: A tree is a non-linear data structure in which data elements are arranged in a sorted sequence.

=> Basic Terminology of Tree:

1 Root - Origin of the tree.

2 Node - Each data item in a tree.

3 Degree of Node: Number of subtrees of a node.

4 Degree of Tree: Maximum degree of nodes in tree.

5 Height of the tree: Maximum number of nodes in a branch.

Height = level + 1

6 Leaf Node: A node that has no child.

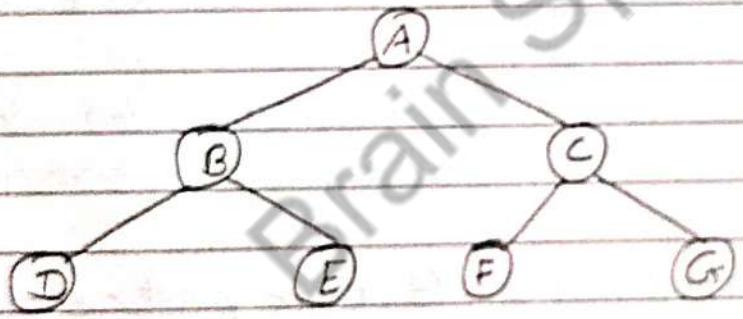
7 Levels:

* Binary Tree:

A tree is a bim binary tree if each node of it can have at most two branch.

Since each element in a binary tree can have only 2 children. This two children is called Left child and right child.

Ex.



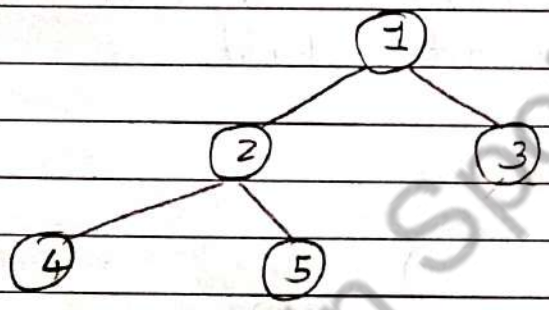
=> Types of Binary Tree:

- Complete Binary Tree
- Full Binary Tree
- Almost complete Binary Tree.

=> Complete Binary Tree:

A Binary tree is called complete Binary tree if each node of a tree have either have two child and all nodes are at same level, except the last level.

Ex.



-> level-0

-> level-1

-> Level-2

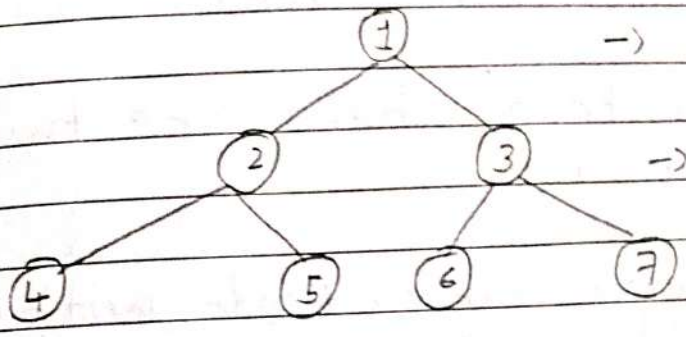
If Complete Binary tree have L Level than

number of nodes = 2^L.

=> Full Binary Tree:

A Binary tree is called Full Binary tree if all the non-leaf node has two child at a same Level.

Ex.



If Full Binary tree have L level than number of node = 2^L.

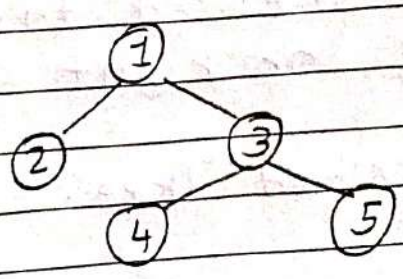
In Full Binary tree every non-leaf node have 2 child.

=> Almost Complete Binary tree.

There are four types of almost complete Binary tree.

1 Strictly Binary tree: In this Binary tree every parent node either have two child or no child.

Ex.



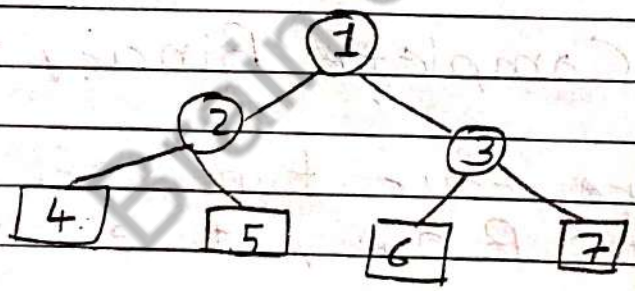
2. Extended Binary tree:

In this tree, there are two type of node.

1) Internal node: Node with 2 child that is represent by O.

2) External node: Node with no child that is represent by □.

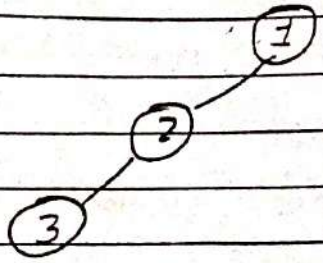
Ex.



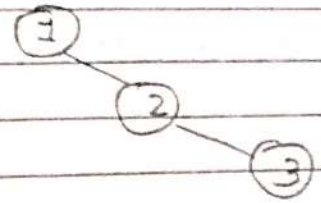
3 Skewed Tree:

A skewed tree is a binary tree that is skewed to the either left side or right side.

Ex. Left Skewed Tree:



-> Right Skewed Tree :



4 Similar tree : Binary tree T and T' are similar if they have same structures.

=> Storage Representation of Binary Tree :

There are two type of represent Binary Tree.

1) Array Representation
2) Linked List Representation.

1 Array Representation :

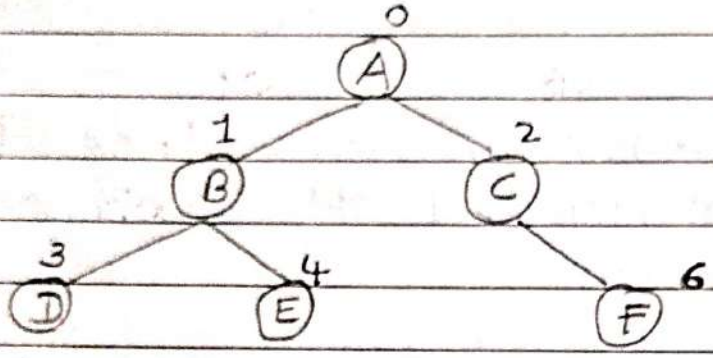
Array Representation is also called Sequential Representation.

In this representation uses a one dimensional array.

In this representation, root node is always number zero and then left child is give one number

and right child gives Two number

Ex.



Array Representation-

0	A
1	B
2	C
3	D
4	E
5	-
6	F

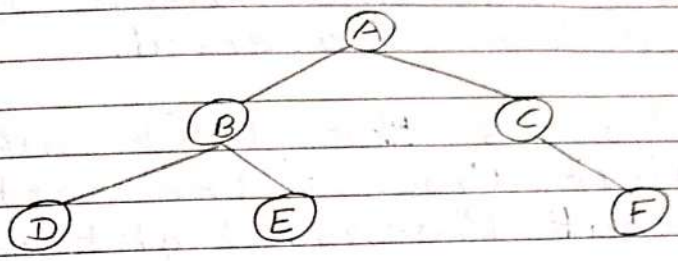
2 Linked list Representation:

In this representation, each node has three fields,

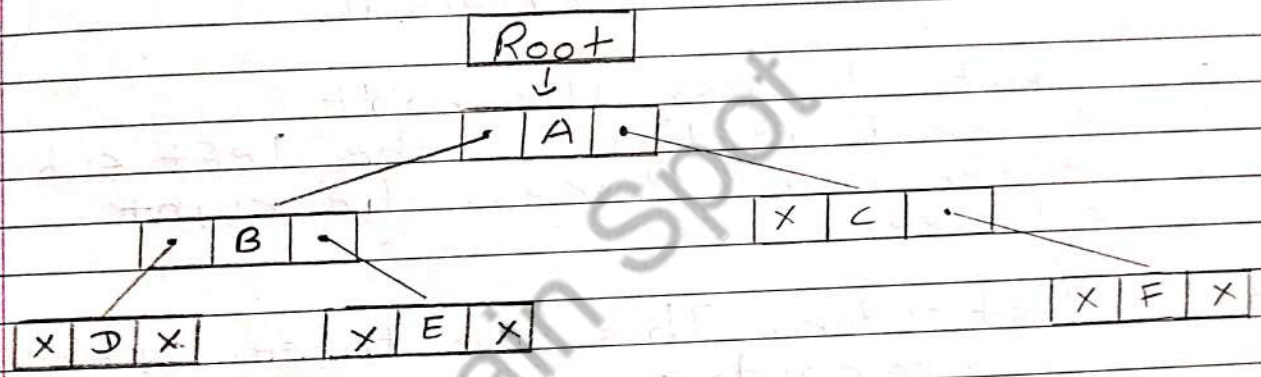
(i) Left child
(ii) Data
(iii) Right child.

Left Child	Data	Right Child

Ex.



-> Linked list Representation:



=> Traversal of a Binary Tree:

In tree, there are three Parameter

1) Node 2) Left Child 3) Right Child.

There are three type of Traversal of a Binary Tree.

(i) Inorder (ii) Preorder (iii) Postorder.

(ii) Inorder : This are three operation in Inorder Traversal.

1. First Process the left subtree.
2. After that Process the root.
3. After that Process right subtree.

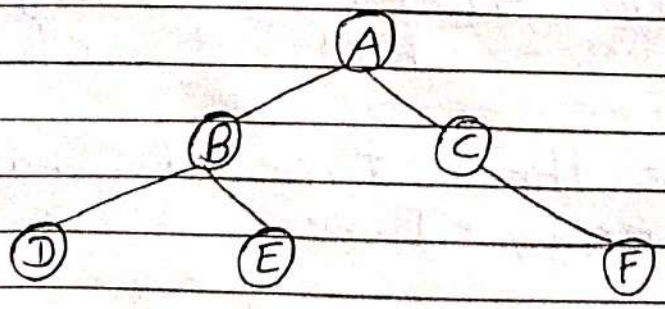
(iii) Preorder : This are three operation in Preorder Traversal.

1. First Process the root.
2. After that Process the left subtree.
3. After that Process the right subtree.

(iii) Postorder : This are three operation in postorder Traversal.

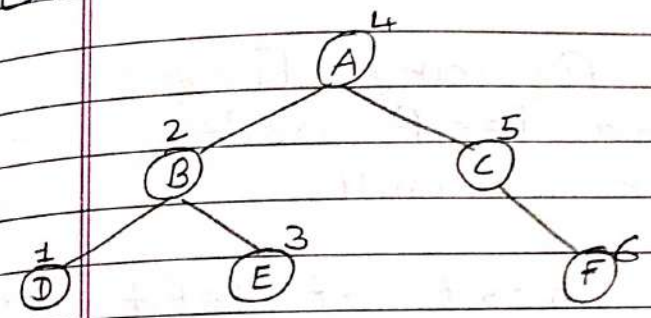
1. First Process the left subtree.
2. After that Process the right subtree.
3. After that Process root.

Ex.



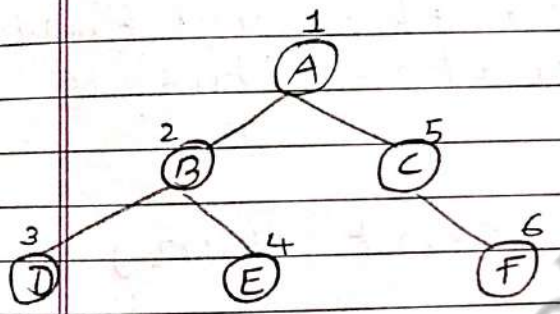
SMVS
Page No.
Date : / /

(i) Inorder:



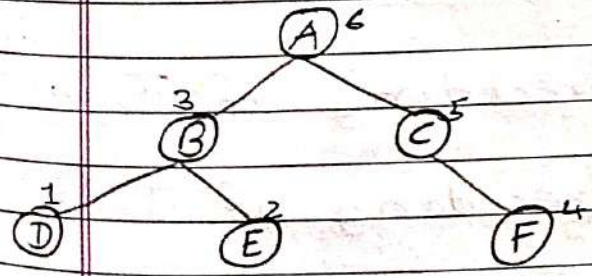
D B E A C F

(ii) Preorder:



A B D E C F

(iii) Postorder:



D E B F C A

* Threaded Binary Tree:

-> Threads: During Binary Tree Creations, for the leaf node we are just write null.

In Binary Tree most of left and Right Field are null.

So, we will replace the null field using special pointer, this is called Threads.

Threads are represent using dotted lines.

-> There are Three way to represent Threaded Binary Tree.

1) One way Threading of Tree
2) Two way Threading of Tree
3) Two way Threading of Tree with Header nodes.

1 One way Threading of Tree:

There are Two way to represent One way Threading of Tree.

(i) Right - In Threading of Tree.

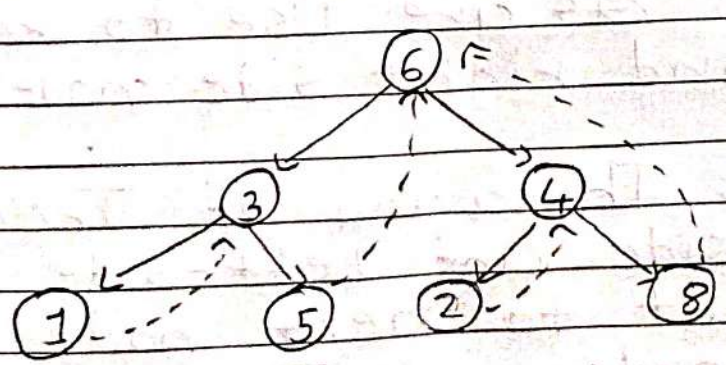
(ii) Left - In Threading of Tree.

(i) Right - In Threading of Tree:

In this Threading of Tree, threads are use only right side.

In this Threading of Tree, only right side node will point to the next node in the inorder traversal of Tree.

Ex.



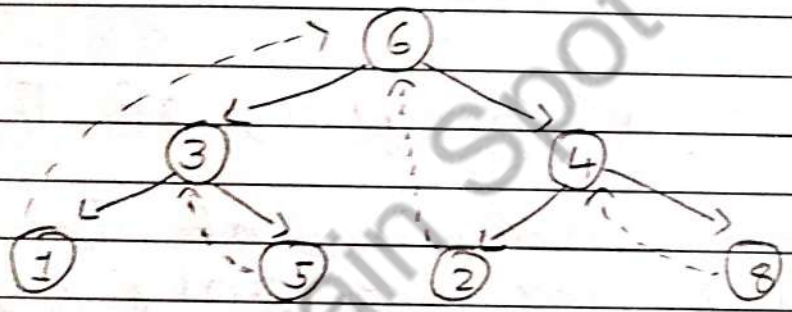
Page No.
Date :

(iii) Left-In Threading of Tree:

In this Threading of Tree, threads are use only left side.

In this Threading of Tree, Left-side node will point to next node in the inorder of Traversal of Tree.

Ex.

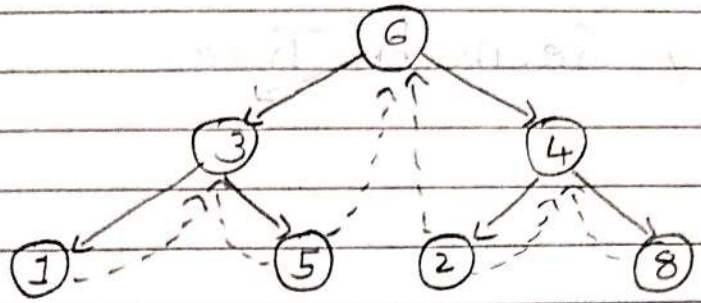


2 Two way Threading of Tree:

In this Threading of Tree, threads ~~as~~ are use to represent left and right side node.

In this Threading of Tree, Right-side and Left-side node will point to next node in the inorder of Traversal of Tree.

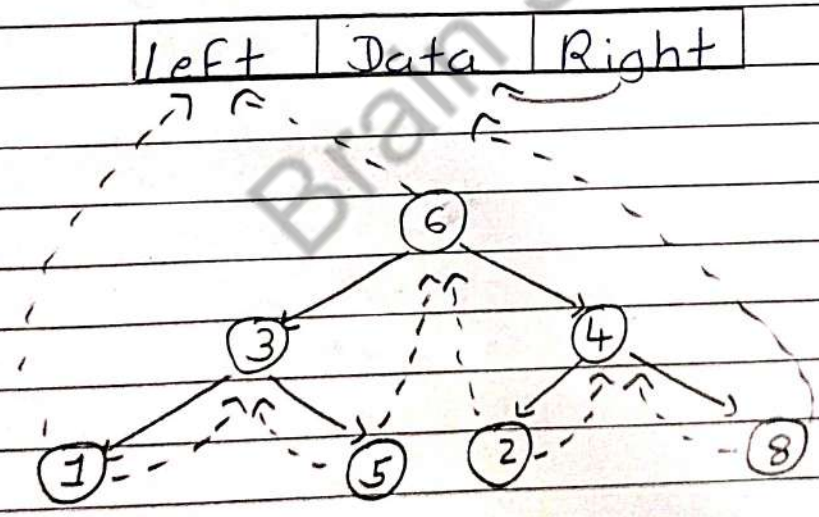
Ex.



3 Two way Threading of Tree with Header nodes.

In this Threading of Tree, one Header node is use. This Header node contain Left, node and Right side data field.

Ex



* Binary Search Tree:

For Binary Search Tree, Tree must follow This condition.

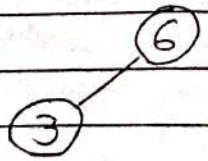
1) Left side element contain less value to the node element

2) Right side element contain greater value or equal value to the node element.

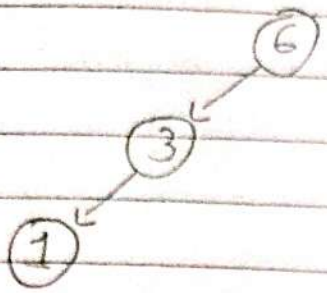
Ex. Create Binary Search Tree for the value:
6, 3, 4, 1, 5, 2, 8, 7,

=> Take First element : 6

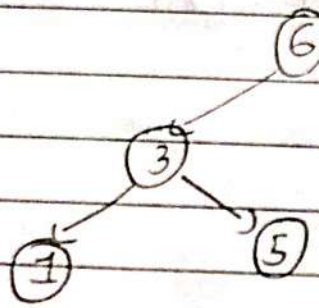
- Data = 3, Here 3 is less than 6. So, 3 will be insert at left side.



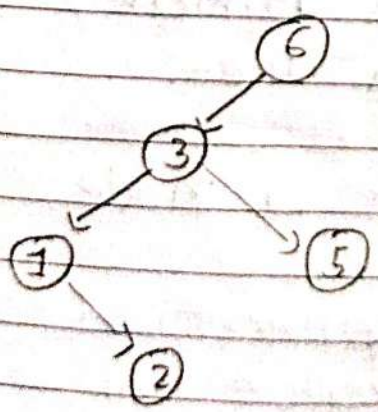
- Data = 1, Here 1 is less than 3, So, 1 will be insert at left side.



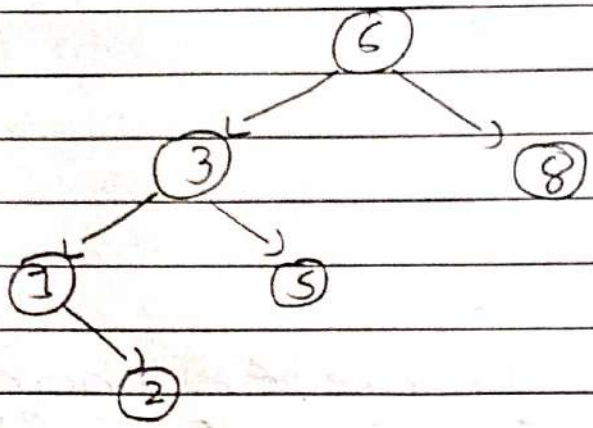
- Data = 5. 5 is greter than 3 but less than 6. So, 5 will insert at right side of 3.



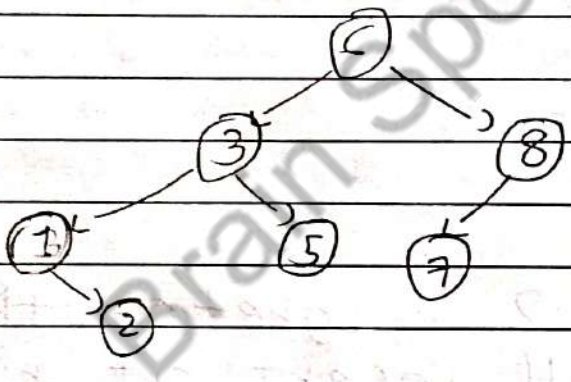
- Data = 2, 2 is greter than 1. So, 2 will insert at right side of 1.



- Data = 8, 8 is greter than 6. So, 8 will insert at right side of 86.



- Data = 7, 7 is less than 8. So, 7 is insert at left side of 8.



* Operation on Binary Tree or Binary Search Tree.

1 Insertion of a node:

Using this operation, we can insert the value in Binary Tree.

- Algorithm:

1 Start

2 Take First value as a node and called new.

3 If (root != Null)
then root = new

4 Again read the next value for node.

5 If (New -> value < root -> value)
then insert at left side
else
insert at right side.

6 Repeat step 4, 5 for complete the Binary Search Tree.

7 End.

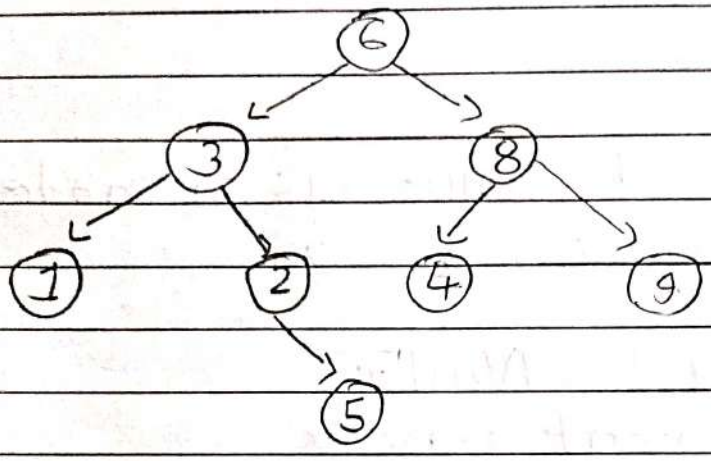
2 Deletion of a node:

Using this Operation, we can delete the node in the Binary Tree.

For Deletion of a node, there are for four case:

(i) If deleted node has no child.

Ex.



In this example, We have to delete 5 node value.

5 node is not have child.

→ Algorithm:

1 Start

2 If X -> left = Null and
X -> right = Null, Then

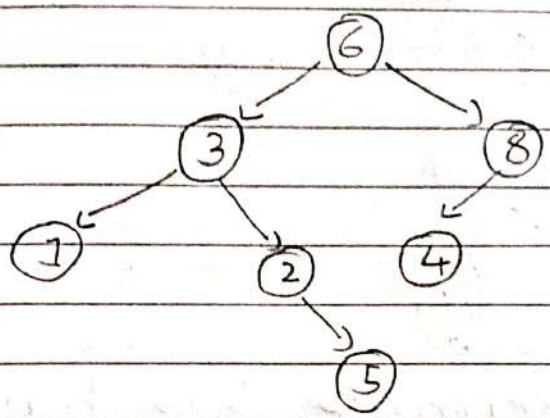
3 If Parent -> right = X, then
set Parent -> right = Null
else
set Parent -> left = Null

4 Free node X.

5 Exit.

(ii) If deleted node has only right child.

Ex.



In this example, We have to delete 2 node value.

2 node have only right child.

→ Algorithm:

1 Start

2 If X → left = Null and
X → right ≠ Null Then,

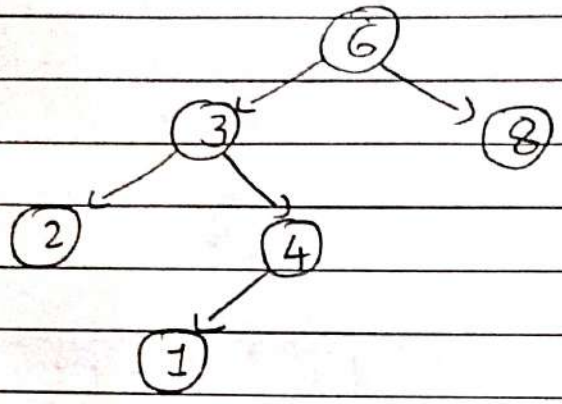
3 If Parent → left = X then,
set Parent → left = X → right
else
set Parent → right = X → right

4 Free node X.

5 Exit.

(iii) If delete node has only left child.

Ex.



In this example, we have to delete 4 node value.

4 node have only left side child.

→ Algorithm:

1 Start

2 If X -> left ≠ Null and
X -> right = Null then

3 If Parent -> left = X then
set Parent -> left = X -> left
else
set Parent -> right = X -> left

4 Free node X

5 Exit.

(iv) If deleted node has both child.

→ Algorithm:

1 Start

2 If $X \rightarrow left \neq Null$ and
$X \rightarrow right \neq Null$ then

3 set Parent = X

4 Set $X_{succ} = X \rightarrow right$

5 Repeat while $X_{succ} \rightarrow left \neq Null$
(i) set Parent = X_{succ}
(ii) set & $X_{succ} = X_{succ} \rightarrow left$

6 Set $X \rightarrow Info = X_{succ} \rightarrow Info$
set $X = X_{succ}$

7 Free X node

8 Exit.

SMVS
Page No.
Date : / /

3. Searching a value in a Binary Tree:

-> Algorithm:

1 Start

2 If Root = Null
then Tree is empty
write: Number not found.

3 else if Num = Root -> Data
then write: Number is found.

4 else if Num < Root -> Data, then
call search Root -> left = Num

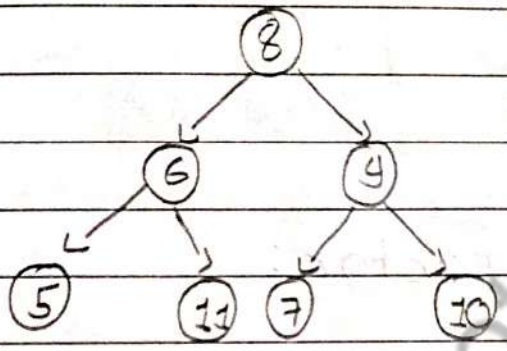
5 else
call search (Root -> right = Num)

6 Return.

* Height Balanced Tree:

A tree is perfectly height balaned if the left and right sub trees are at same level.

Ex.



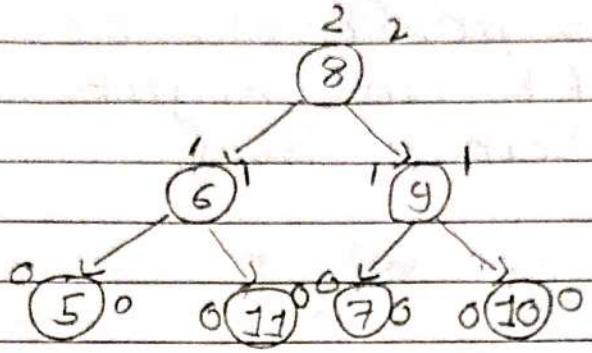
For Height Balanced Tree, we have to calculate Balaned factor.

Balance factor is a difference between right and left sub tree.

Balanced Factor = Height of Left Subtree - Height of right Subtree.

The Range of Balanced factor is 1, 0 and -1.

Ex. Find the Balanced Factor of each node.



→ Balanced factor:

B.F of 8 = 2 - 2 = 0
B.F of 6 = 1 - 1 = 0
B.F of 5 = 0 - 0 = 0
B.F of 11 = 0 - 0 = 0
B.F of 7 = 0 - 0 = 0
B.F of 10 = 0 - 0 = 0
B.F of 9 = 1 - 1 = 0

* AVL Tree:

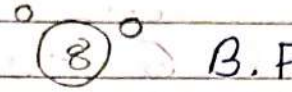
AVL - Adelson, Velskit, Landis.

In AVL tree, we use Height Balanced tree.

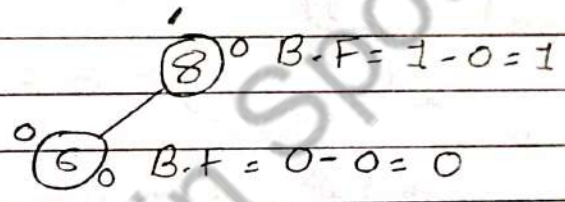
Ex. Draw AVL Tree:
8, 6, 9, 12, 13, 15, 2

=>

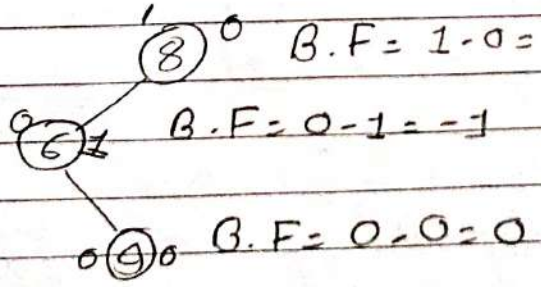
1 Start From First node : 8



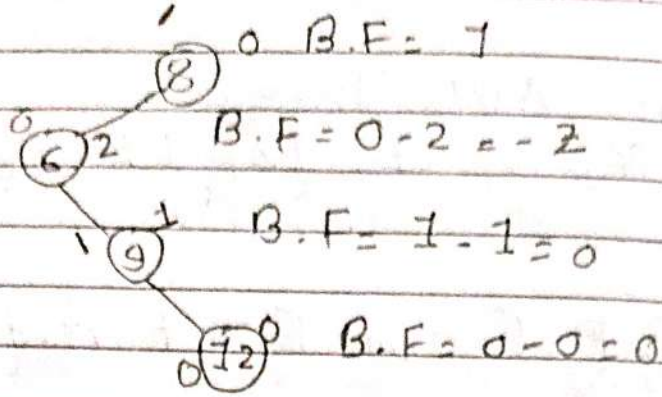
2 Insert second value : 6. 6 is less than 8. So, 6 is insert at left node of 8.



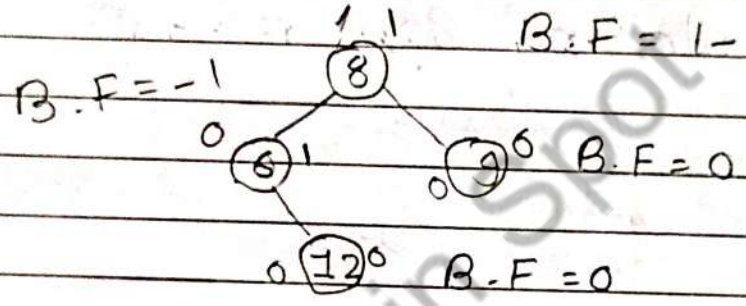
3 Insert third value : 9. 9 is greter than 6. So, 9 ~~we~~ will be insert at right node of 6.



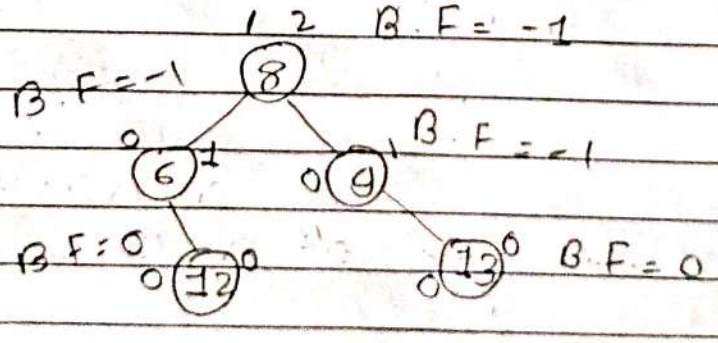
4 Insert Value : 12 . 12 is greter than 9. So 12 will be insert at right node of 9.



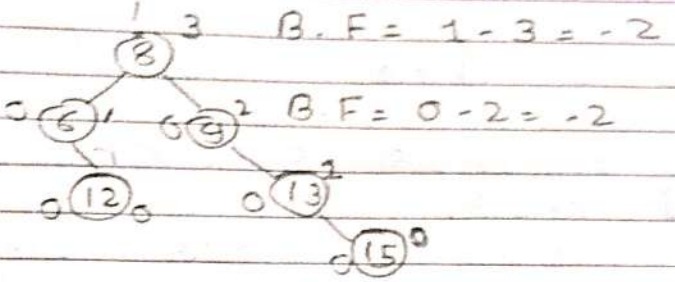
Here, B.F of 6 node is -2.
So this node is Critical node.
that's why we have to rearrage tree.



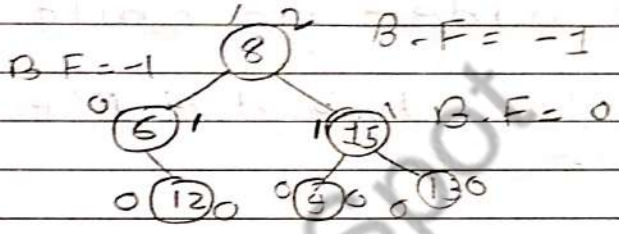
5 Insert value: 13. 13 is Greter than 9. So 13 is inserted at right side of the 9.



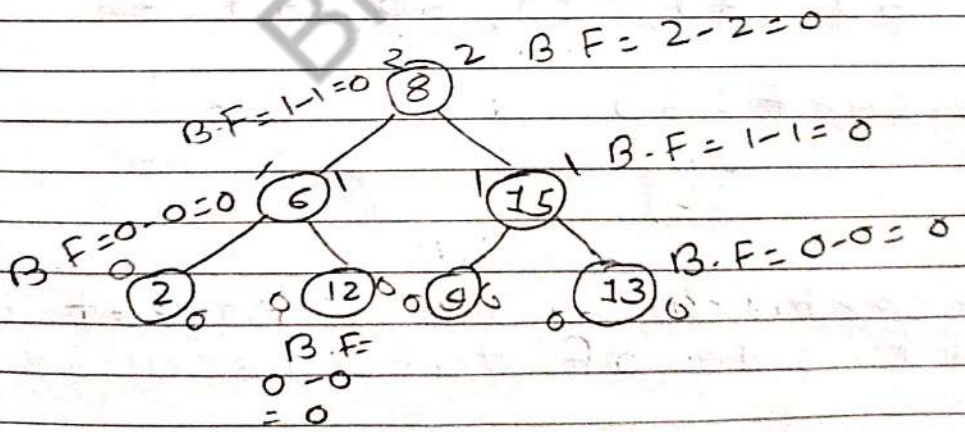
6 Insert value 15: . 15 is Greter than 13. So, 13 is inserted at right side of the 13.



Here, 8 and 9 be a critical node. So, we have to rearranged tree.



7 Insert value : 2. 2 is less than 6. So, 2 is inserted left side of the 6.



Here, All the nodes are Balanced. This is a AVL Tree.

* 2-3 Tree:

2-3 tree is a one kind of B-tree.

2-3 tree contain three type of node.

1 Leaf node - no child

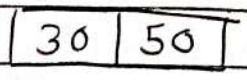
2 2-node - One data + Two child

3 3-node - Two data + Three child.

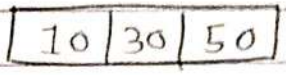
Ex. Create a 2-3 Tree.
50, 30, 10, 60, 80,

-> Insert: 50

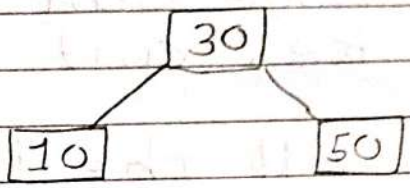
- Insert: 30 : 30 is insert at left side of data because 30<50



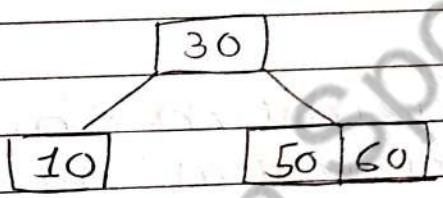
- Insert: 10 : 10 is insert at left side of data because 10<30



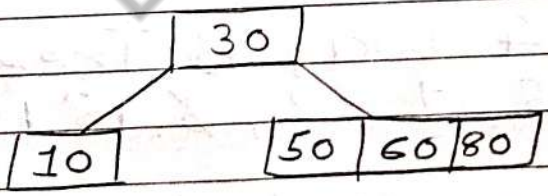
Here, We have to split this th tree into the 2-node.



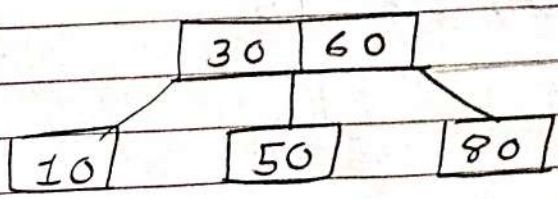
- Insert : 60 : 60 is insert at right side of the 50 data.



- Insert : 80 : 80 is insert at right side of the 60 data element.



Here, We have to split this tree into 3-node.



This is a 2-3 tree with 3-node.

SMVS
Page No.
Date : / /

* B Tree:

B Tree is balanced M-way tree

In B Tree, node can have more than one key and child.

In B Tree, All the leaf node at a same level.

IF B Tree have M-way or order

than, Maximum child = M
Maximum node = M-1
Internal node = $\frac{M}{2}$

Every node have m-1 key.

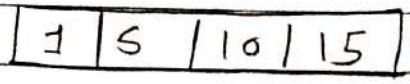
Ex. Create 4 way B Tree:
5, 10, 15, 1, 2, 8, 18, 19

-> Maximum child = 4
Maximum node = 4-1 = 3

- Insert 5, 10, 15

| 5 | 10 | 15 |

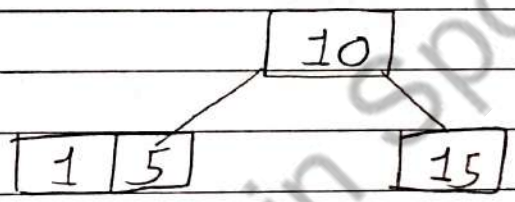
- Insert : 1



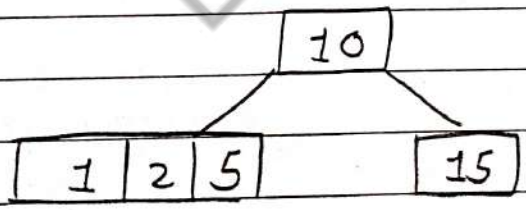
→ In this tree Maximum node Key = 3.

But in this tree, node key is 4. This is not allowed in 4-way B Tree.

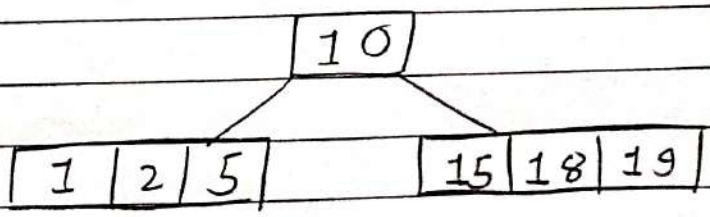
So, we have to split the tree.



- Insert : 2



- Insert : 18, 19



~~This is B+ Tree~~
This is a 4 way B Tree.