(2) The Sibling of a node becomes its right child.

* Conversion of General Trees to Binary

Threaded Binary Trees :-

→ For Binary tree. Creation for the leaf node there is no subtree further so we Put NULL.

→ It is wastage of memory So, to avoid NULL values in the node we just Set the threads which are actually the links to the Predecessor and Successor nodes.

→ Three types of threading is Possible Inorder, Preorder and Postorder.

→ We will discuss inorder threading

→ Here left thread Should Point to the Predecessor and right thread Points to the Successor.

→ Assume Head node is starting node. and root node of tree is attach to left of head node.

Page
Date

* Application of Trees!

→ The Purpose of binary tree is to have efficient retrieval of data.

→ It is a useful data structure when we require to take two way decisions.

⇒ The use of binary tree is exploited by following data structure.

(1) BST (2) Expression trees
(3) Threaded binary trees.
(4) Game tree.

* Concept of Balanced Trees!

→ Balanced trees are useful data structures that are useful to store the data at some specific location.

Def → A balanced tree is a rooted tree where each subtree of the root has equal number of nodes.
→ The height of such binary tree is O(log N)

Page
Date

Many types of balanced tree is

(1) Height balanced tree or AVL-tree.

(2) Splay tree

(3) Red-black trees.

(4) B-trees.

* AVL Trees:-

Def An empty tree is height balanced if T is a non-empty binary tree with T_L and T_R as its left and Right Subtrees. The T is height balanced if and only if

i) T_L and T_R are height balanced

ii) $h_L - h_R <= 1$ where h_L and h_R are height of T_L and T_R.

The idea of balancing a tree is obtained by calculating the balance factor of a tree.

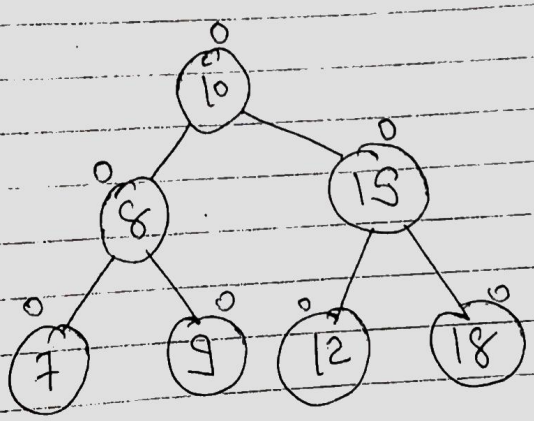
Page
Date

Defination of balance factor

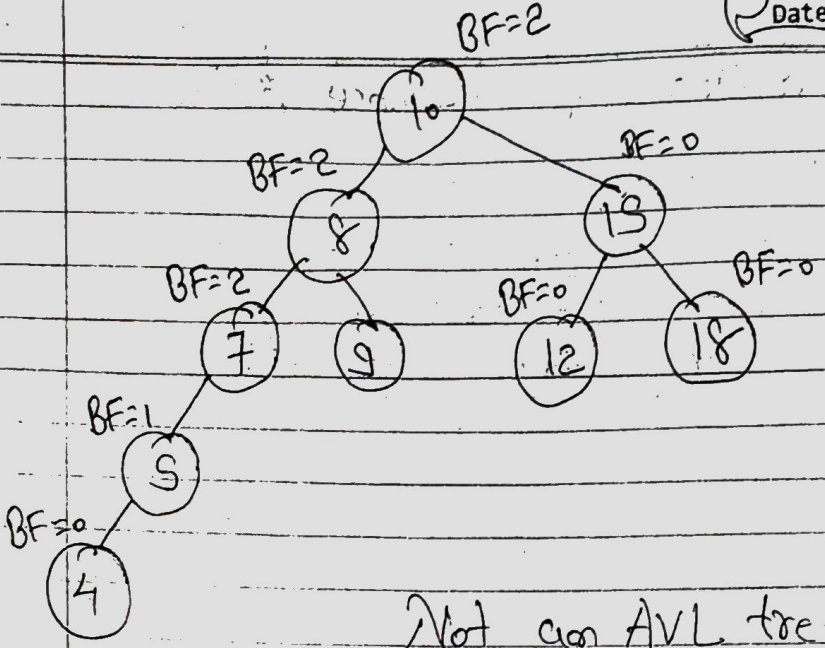
→ The balance factor BF(T) of a node in binary tree is defined to be $h_L - h_R$ where h_L & h_R are height of left and right subtrees of T.

* For any node in AVL tree the balance factor

BF(T) is -1, 0 or +1.



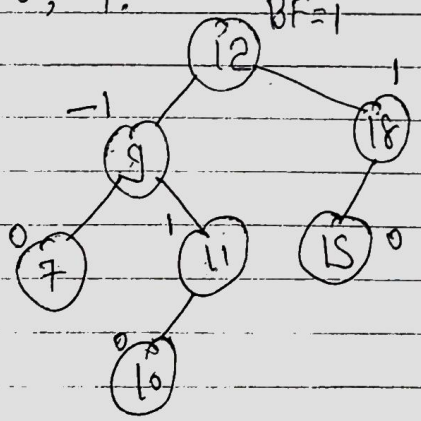
AVL Tree



Not an AVL tree

* Representation of AVL Tree

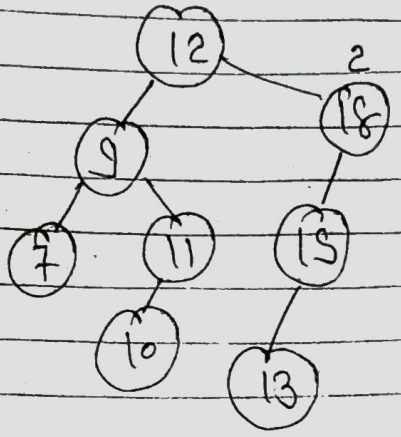
→ The AVL tree follows the Property of binary Search tree. Infact AVL trees are basically binary Search tree with balance factor as −1, 0, 1.



AVL tree.

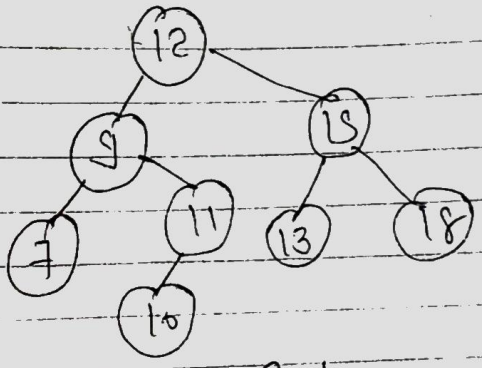
Date

Insert 13.



Property violated

So,



Node on that Path need to be readjust
- means only the affected subtree is to be rebalanced.

* Insertion

There are four different cases when rebalancing is required after insertion of new node.

1. An insertion of new node into left subtree of left child (LL)

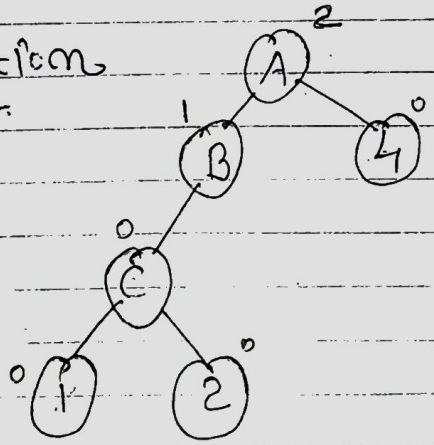
2. (LR)
3. (RL)
4. (RR).

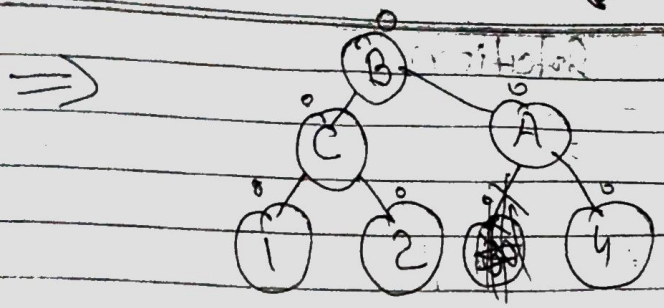
* Some modifications done on AVL tree in order to rebalance it is called rotations of AVL tree.

* There are two types of rotations.

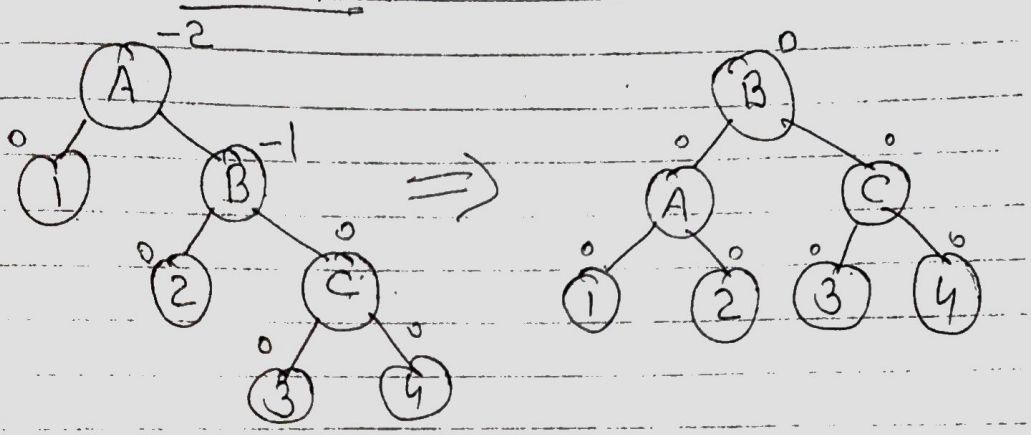
- Single rotation
  - left-left (LL)
  - Right-Right (RR)
- Double Rotation
  - left-Right (LR)
  - Right-left (RL)

(1) LL Rotation

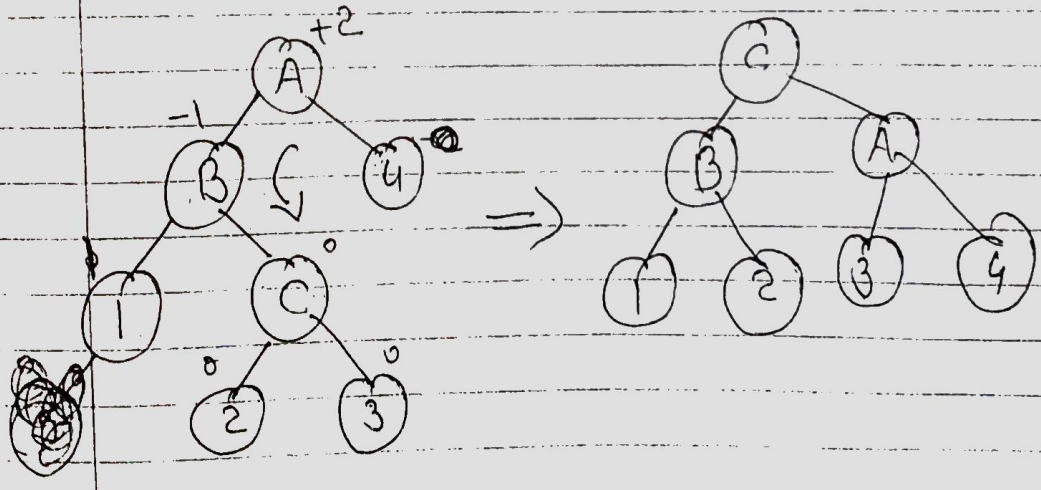




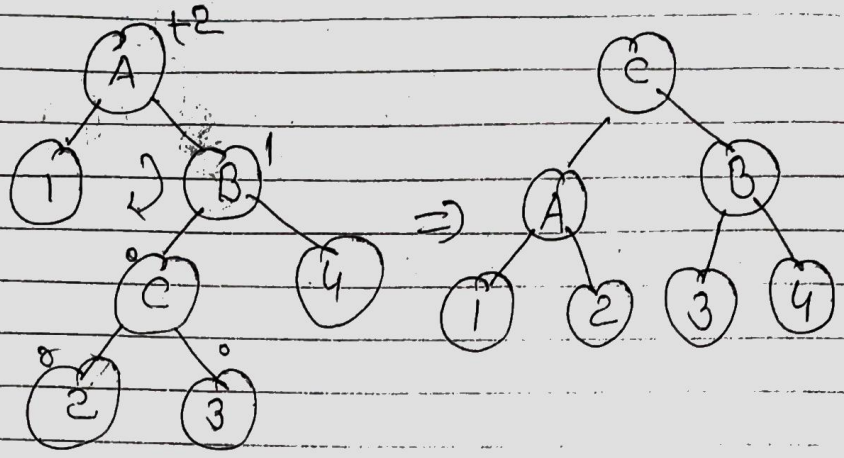
(2) RR Rotation



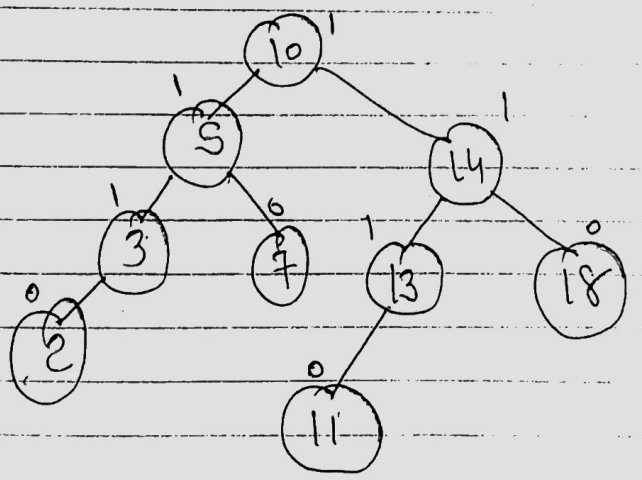
(3) LR Rotation



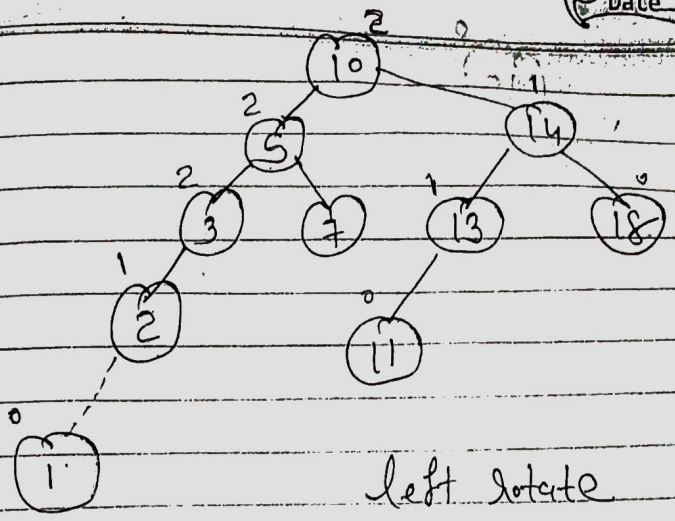
(4) RL Rotation



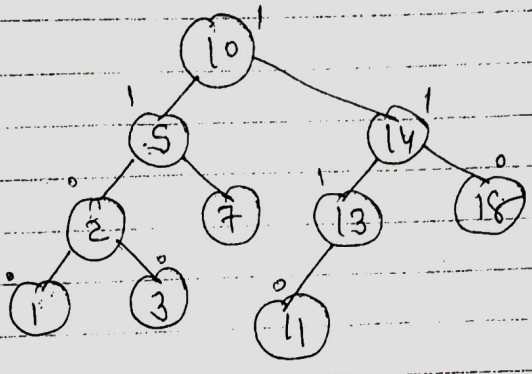
Ex:- Insert 1, 25, 28, 12 to build AVL tree.



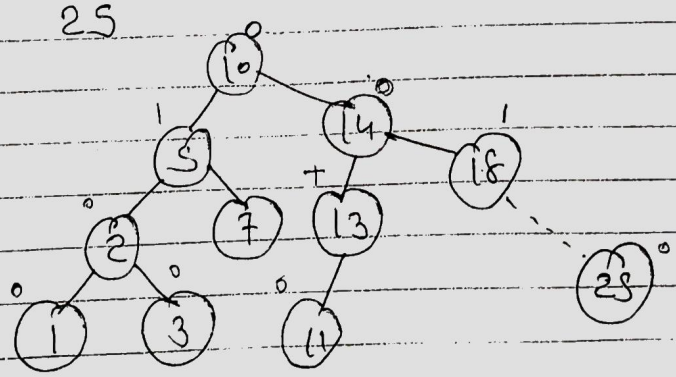
Insert 1



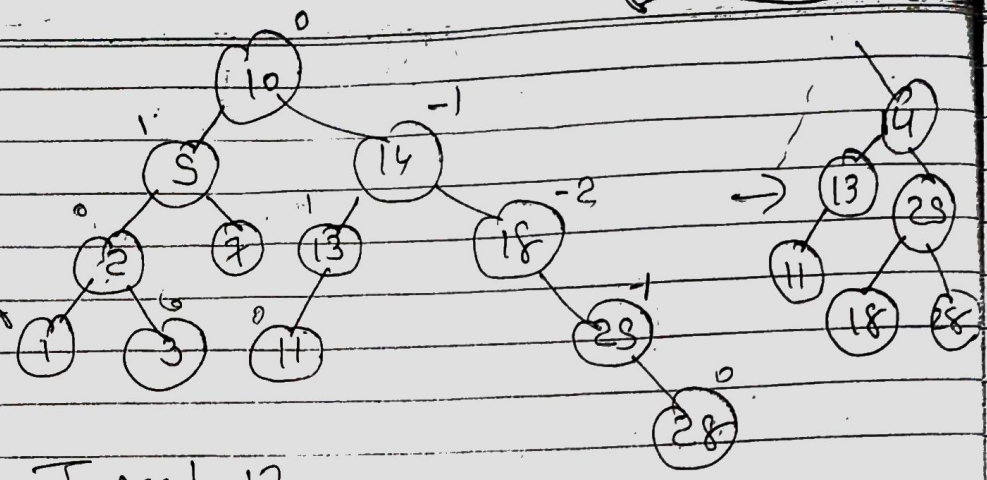
left rotate



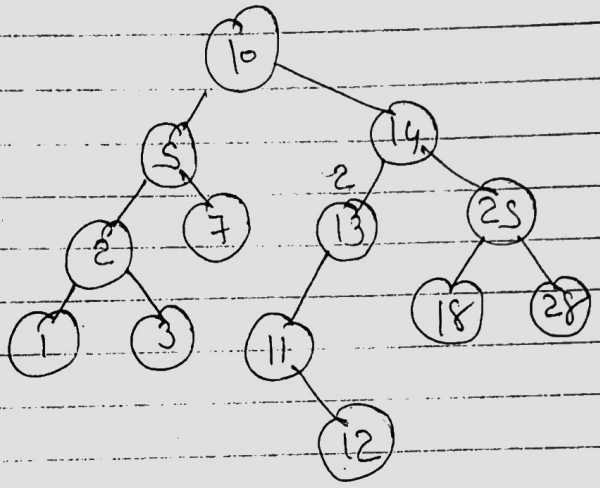
Insert 25

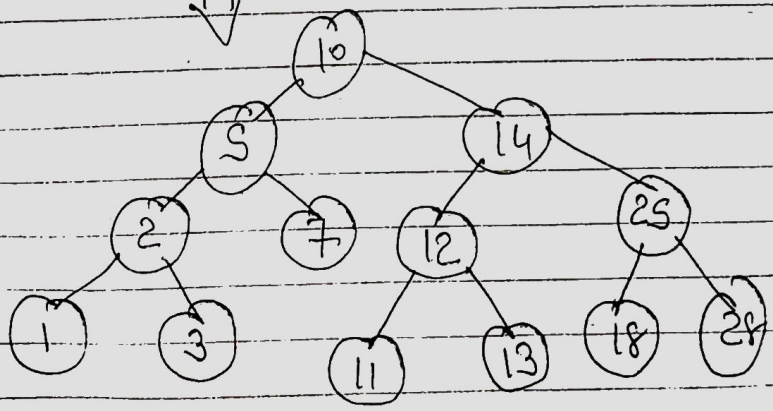


Insert 28


10
5
14
2
7
13
18
1
3
11
28
4
13
28
11
18
28

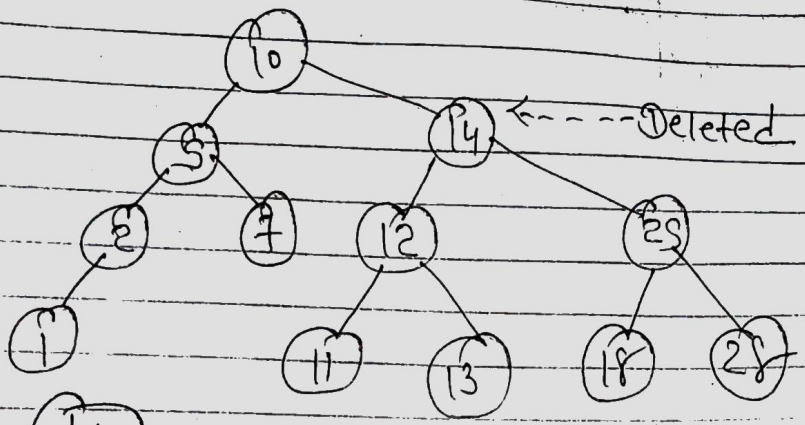
Insert 12


10
5
14
2
7
13
25
1
3
11
18
28
12


10
5
14
2
7
12
25
1
3
11
13
18
28

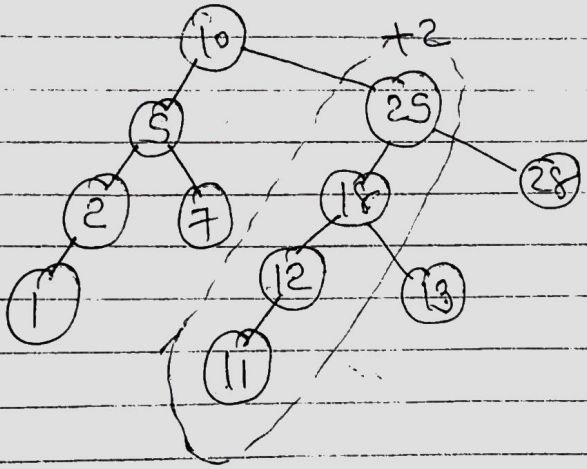
* Deletion

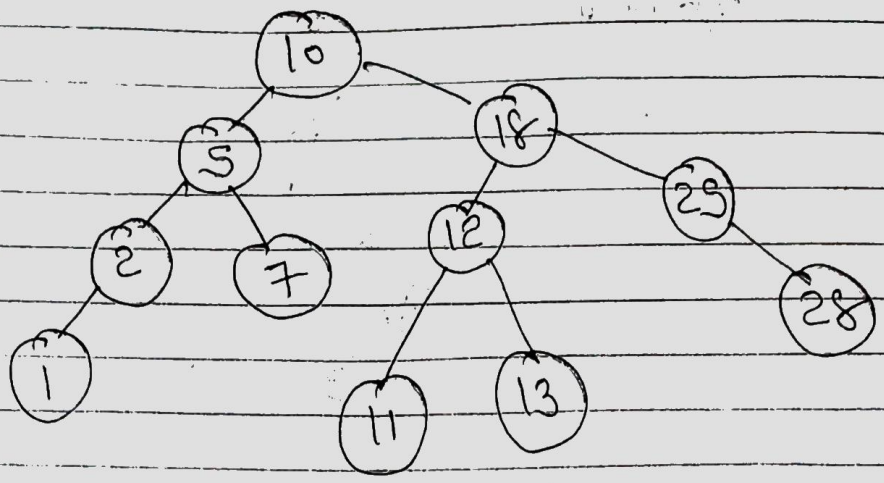
Consider AVL tree.



Delete (14)

So, tree become



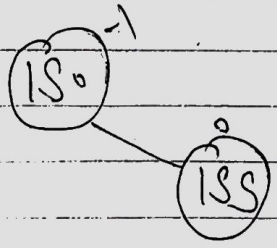


Ex1: Define an AVL tree. obtain AVL using following node insertion in sequence

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

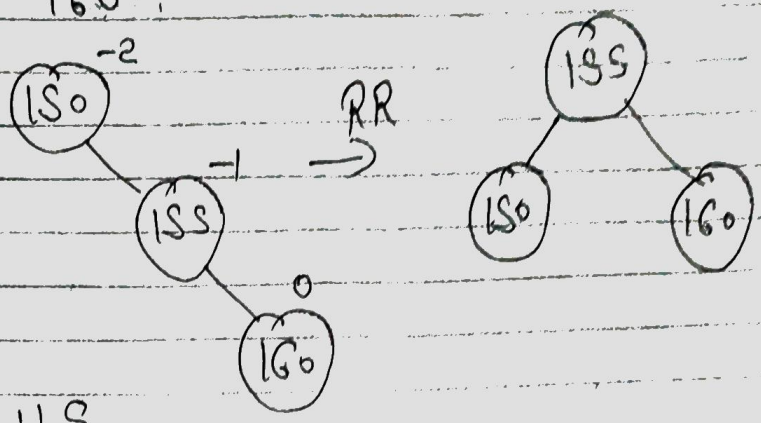
insert 150

insert 155

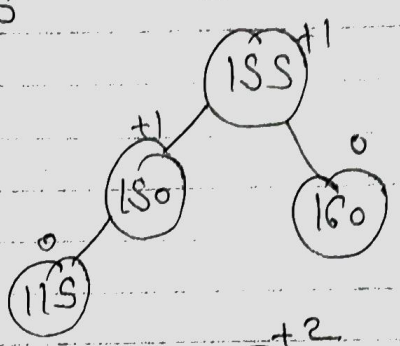


Page
Date

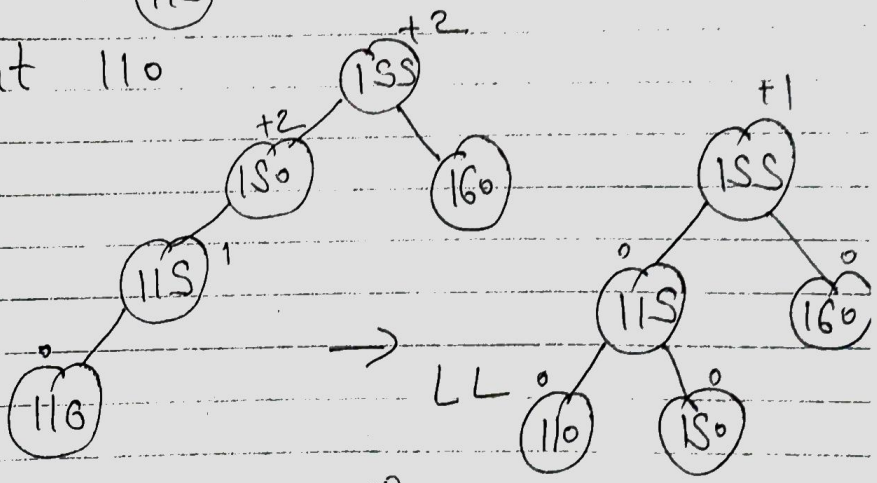
insert 160.



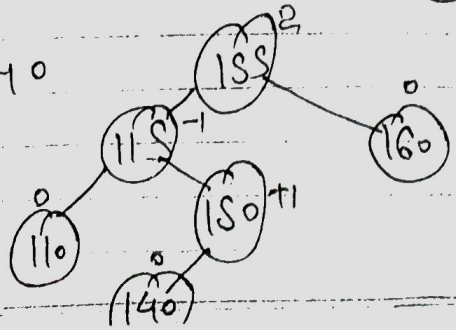
insert 115

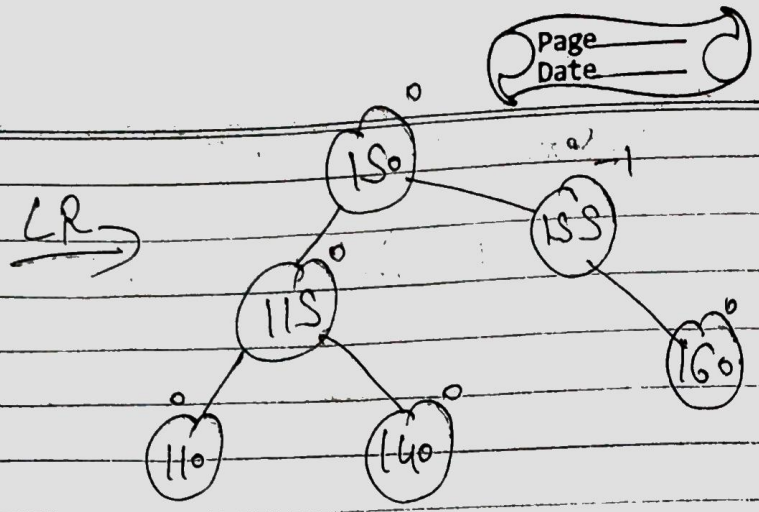


insert 110



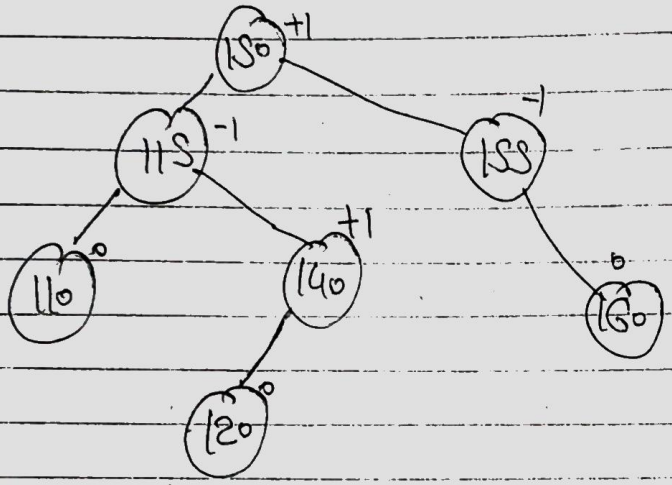
insert 140



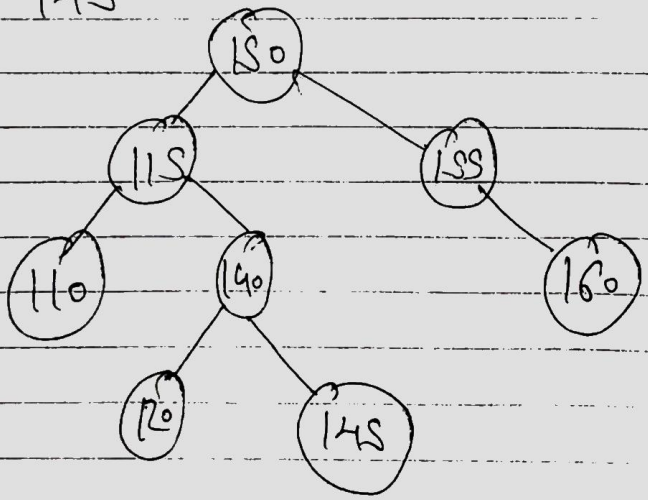


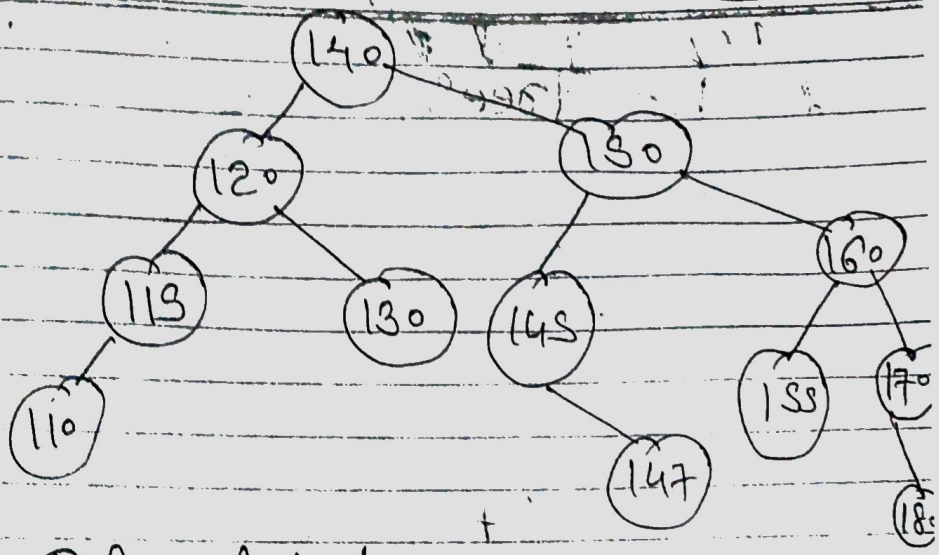
LR

insert 120



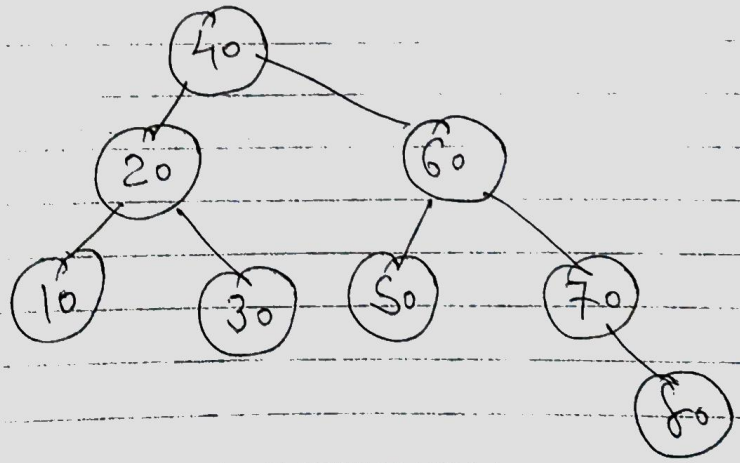
Insert 145





Ex1/ Define AVL tree.

10, 20, 30, 40, 50, 60, 70, 80



Page
Date

* B- Trees

⇒ B-tree is a specialized multiway tree used to store the record in a disk.

⇒ There are number of subtree to each node. So that the height of the tree is relatively small.

⇒ The goal of B-tree is to get fast access of the data.

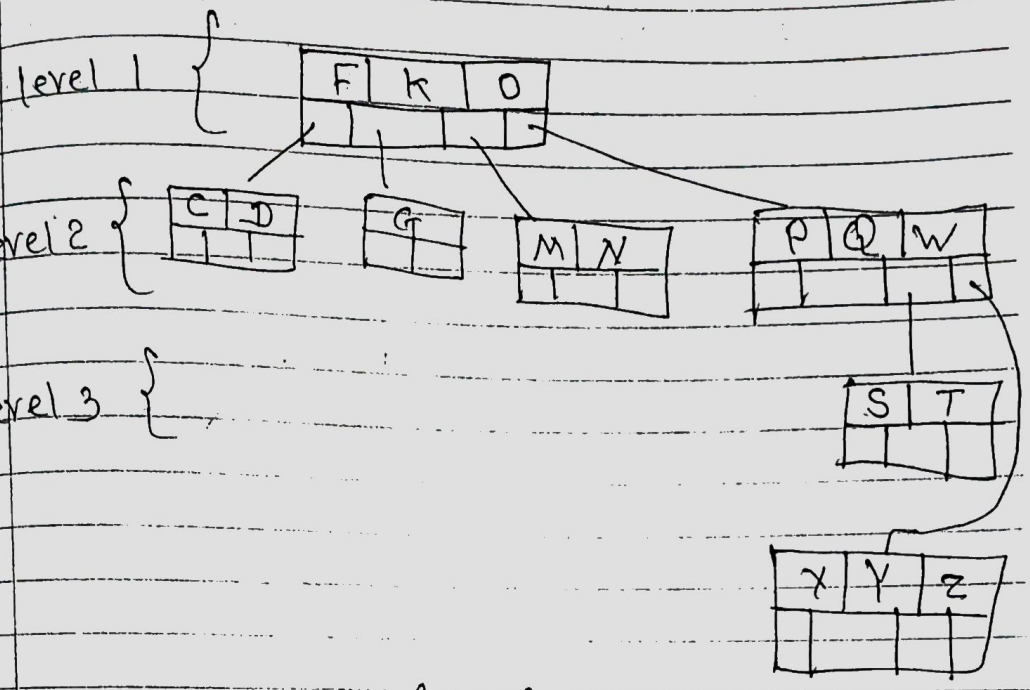
Multiway Search tree:-

⇒ A Multiway Search tree of order m is an ordered tree where each node has at the most m children.

⇒ if n number of children in a node then (n-1) is the number of key in the node.

Page
Date

Ex) Following is a tree of order 4.



From above fig following observation Cambe made.

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

(2) The key in each node are in ascending order.

(3) For every node Node. Child[0] has only keys which are less than Node. key[0], Child [i] has only keys which are greater than Node. key[0].

→ The node at level 1 has F, K and O as keys.

→ At level 2 the node containing keys C and D are arranged as child of key F.

→ Here F, K, O are called keys and branches are called Childre

* The B-tree of order m should be constructed such that.

Rule 1: All the leaf nodes are on the bottom level.

Rule 2: The root node should have atleast two children.

Rule 3: All the internal node except root node have atleast ceil (m/2) nonempty children. the ceil is a fun such that ceil (3.4) = 4,

ceil (9.3) = 10

ceil (2.93) = 3

ceil (7) = 7

Page
Date

Rule 4: Each leaf node must Contain atleast ceil (m/2) -1 keys.

Ex: Example of B-tree

* Insertion

We will Construct B-tree of order 5 following numbers.

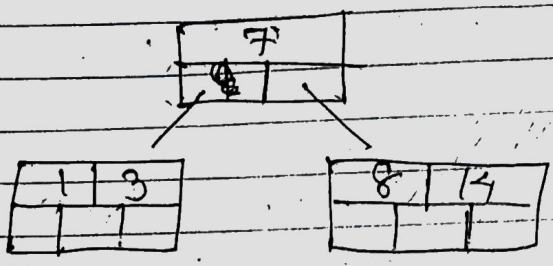
3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12
20, 26, 4, 16, 18, 24, 25, 19

- The order 5 means at the most 4 keys are allowed.

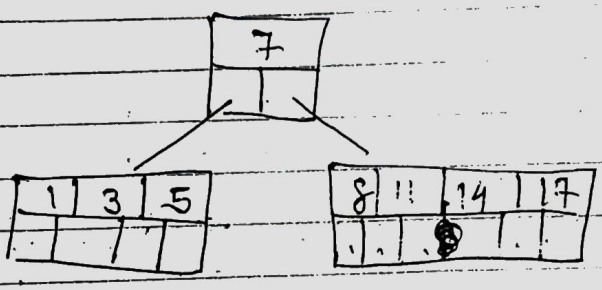
- The internal node should have atleast 3 noneempty children and each leaf node must Contain atleast 2 keys.

Step 1 Insert 3, 14, 7, 1 as follow

Step 2 If we insert 8 then we need to split the node 1, 3, 7, 8, 14 at medium.

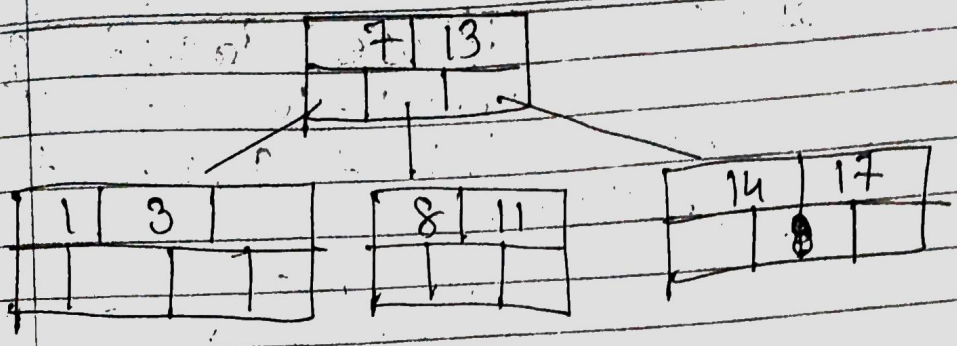


Step 3

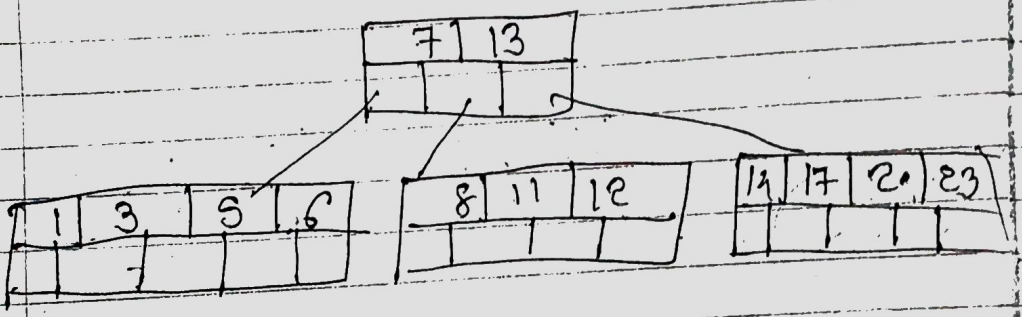


Step 4 Now insert 13. But if we insert 13 then the leaf node will have 5 keys which is not allowed.

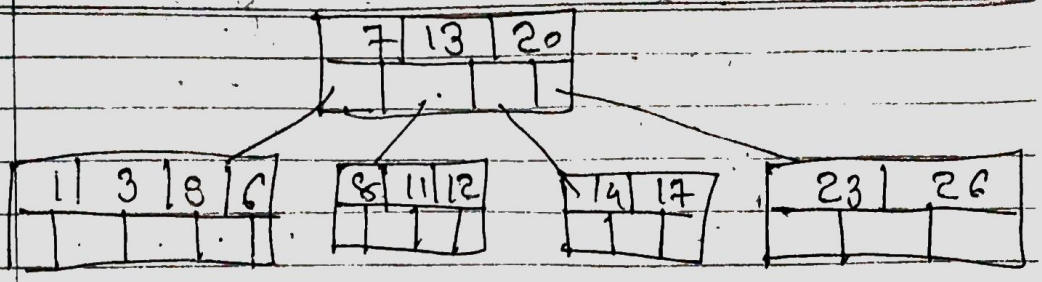
8, 11, (13), 14, 17 is split and medium node 13 is moved UP.



Step 5 :- Now insert 6, 23, 12, 20 without any split.

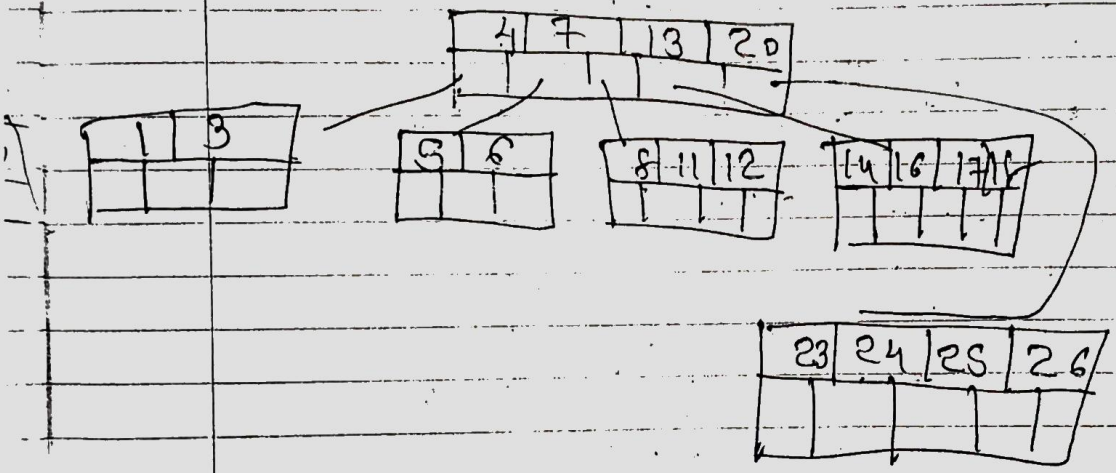


Step 6 :- The 26 is inserted to the rightmost leaf node. Hence 14, 17, 20, 23, 26 the node is split and 20 will moved up



Step 7

Insertion of node 4 cause leftmost node to split. The 1, 3, 4, 5, 6 causes key 4 to move up. Then insert 16, 18, 24, 25.



Step 8

Finally insert 19. Then 4, 7, 13, 19, 20 needs to be split. The ~~middle~~ median 13 will be moved upto form a root node.