AVL Tree(Balanced Tree)

As we have seen in binary search tree, optimization of searching is totally dependent on the order of inserted element. If the binary search tree is complete balanced tree then this optimization can be achieved. We know that complete balanced tree is an ideal situation but we can be near to this optimization of search time for insertion and deletion. The Russian mathematicians G.M Adel'son.Vel'skii and E.M.Landis came with the new technique for balancing binary search tree called AVL tree on their names. This tree has technique for efficient search and insertion, so AVL tree behaves near to complete binary search tree.

So now our purpose is to maintain the binary search tree as complete binary search tree. In binary search tree it is not possible to maintain the same height of left and right subtree of any nodes as in complete binary tree. But with AVL tree we can get the binary search tree where height of left and right subtree will be with maximum difference 1 which is close to complete binary search tree. So we can define AVL tree as

An AVL tree is a binary search tree where height of left and right subtree of any node will be with maximum difference 1.

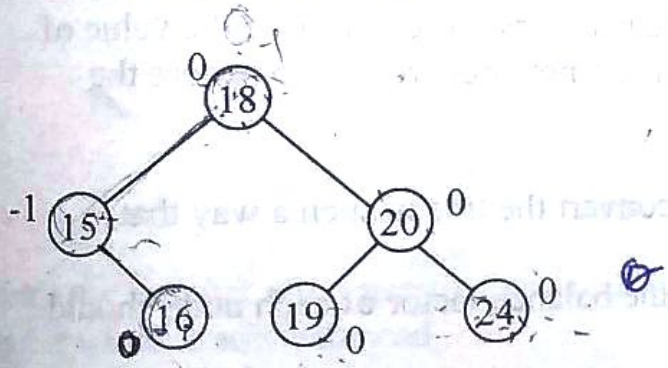
Each node of AVL tree has a balance factor. Balance factor of a node is defined as the difference between the height of left subtree and right subtree of a node.

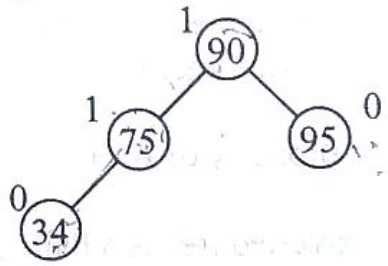
Balance factor = Height of left subtree - Height of right subtree

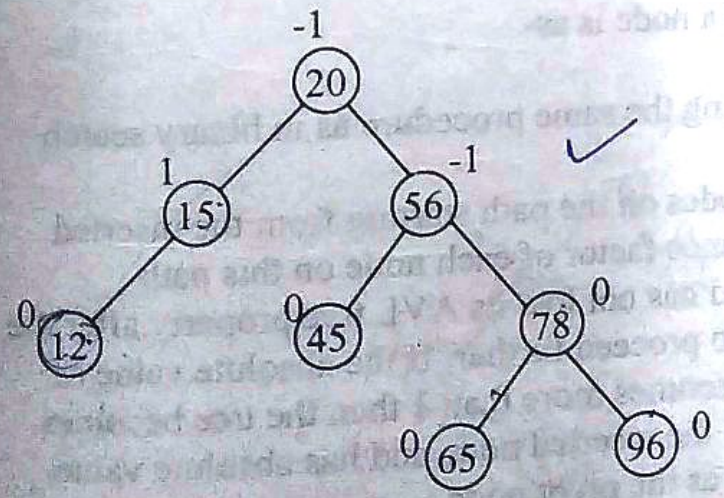
A node is called right heavy or right high if height of its right subtree is one more than height of its left subtree. A node is called left heavy or left high if height of its left subtree is one more than height of its right subtree.A node is called balanced if the heights of right and left subtree are equal.

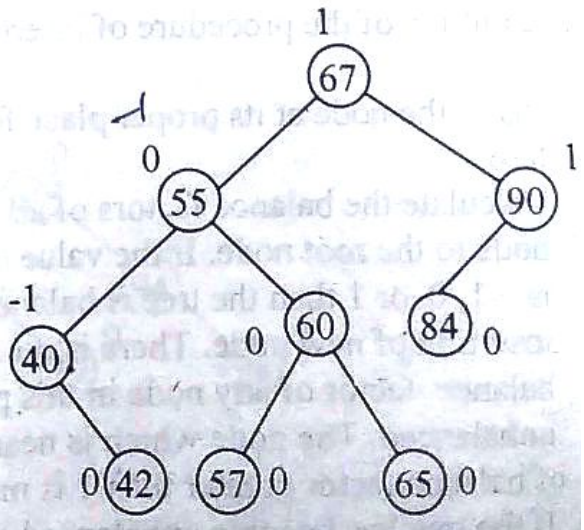
The balance factor will be 1 for left high, -1 for right high and 0 for balanced node. So in an AVL tree each node can have only 3 values of balance factor which are –1, 0, 1 or we can say the the absolute value of balance factor should be less than or equal to 1.

Let's take some AVL trees

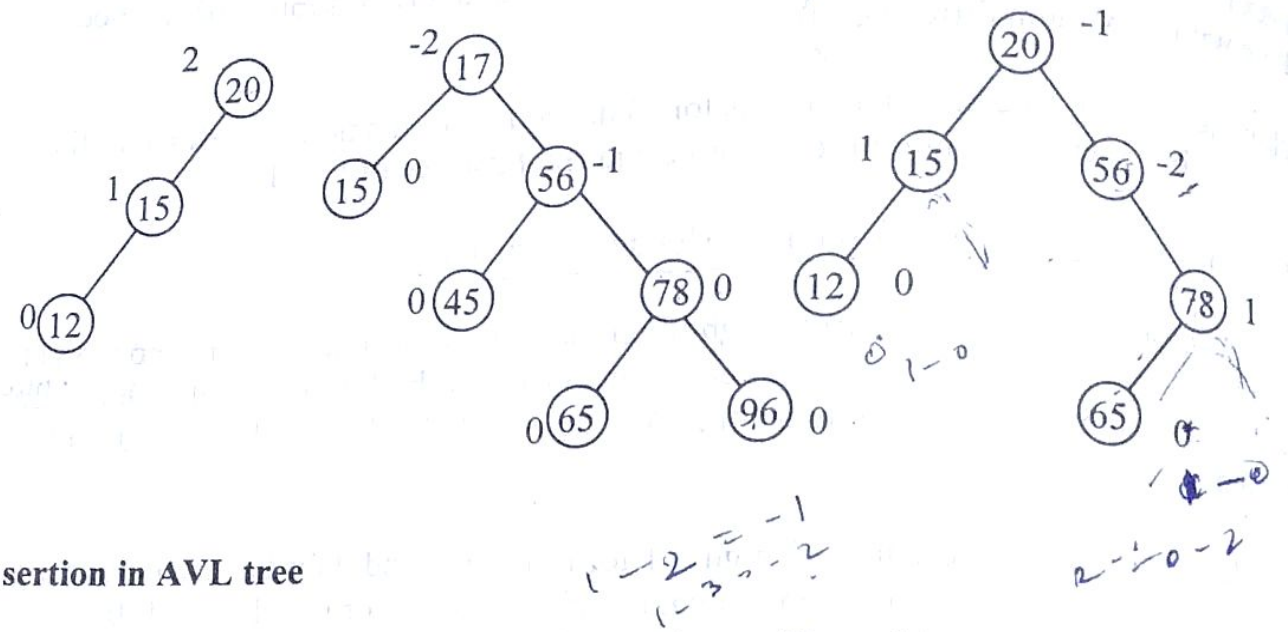








Let us see some trees which are binary search trees but are not AVL trees.



Insertion in AVL tree

Insertion in AVL tree is same as in binary search tree. Here also we will search for the position where the new node is to be inserted and then insert the node. But AVL tree has a property that the height of left and right subtree will be with maximum difference 1. Suppose after inserting new node, this difference becomes more than 1 i.e the value of balance factor has some value other than –1, 0, 1. So now our work is to restore the property of AVL tree again.

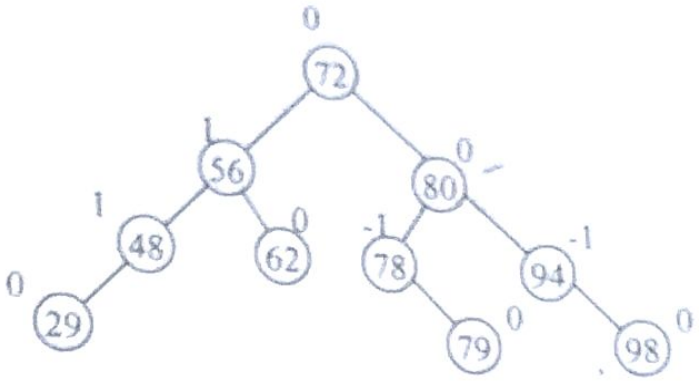
To restore the property of AVL tree we should convert the tree in such a way that

1. The new converted tree is a balanced tree i.e the balance factor of each node should be -1, 0, or 1.
2. The new converted tree should be a binary search tree with inorder traversal same as that of the original tree.

The outline of the procedure of insertion of a node is as-

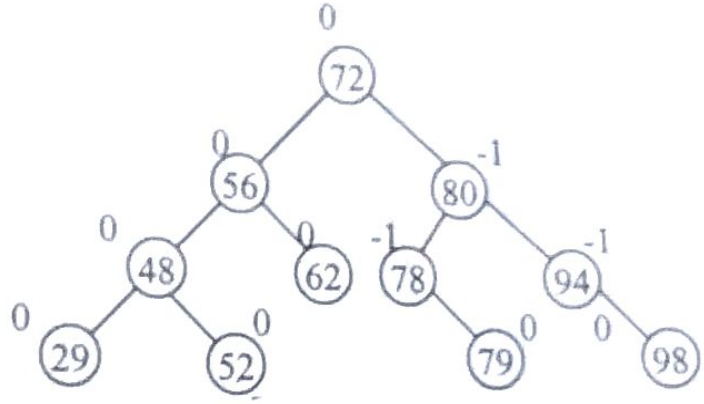
1. Insert the node at its proper place following the same procedure as in binary search tree.
2. Calculate the balance factors of all the nodes on the path starting from the inserted node to the root node. If the value of balance factor of each node on this path is –1, 0, or 1 then the tree is balanced and has not lost its AVL tree property after the insertion of new node. There is no need to proceed further. If the absolute value of balance factor of any node in this path becomes more than 1 then the tree becomes unbalanced. The node which is nearest to the inserted node and has absolute value of balance factor greater than 1 is marked as the pivot node.
3. If the tree has become unbalanced after the insertion then there is a need to convert the above tree by performing rotations about the pivot node such the converted tree has all the properties of the AVL tree. This rotation is simple manipulation of pointer and is discussed in next section.

Let us take an AVL tree-



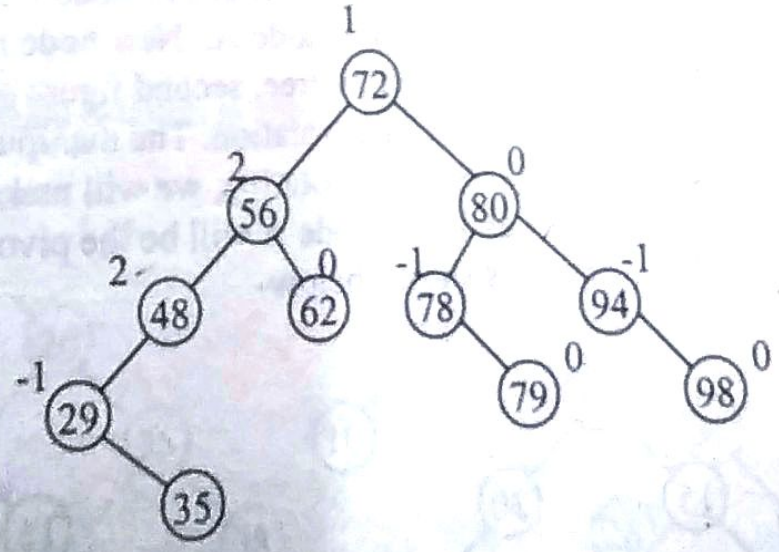
AVL Tree (T)

Insert node 52 in the above tree.



Here we see that after inserting node 52, the balance factors of some nodes have changed but the tree is still balanced.

Now we insert node 35 in the AVL tree T.



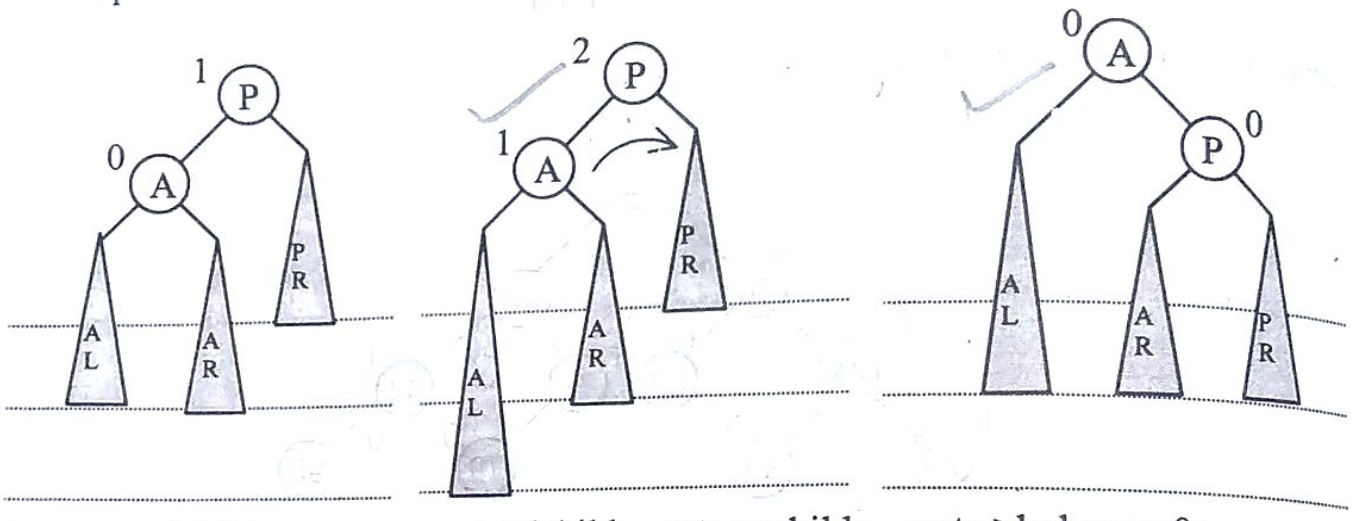
Here we see that the new tree is unbalanced. Node 48 is unbalanced and is nearest to the inserted node, so it will be the pivot node.

AVL Rotations

There can be four types of rotations depending upon where the new node is inserted.
1. Left to left rotation -- Insertion in left subtree of left child of pivot node.
2. Right to right rotation -- Insertion in right subtree of right child of pivot node.
3. Right to left rotation -- Insertion in left subtree of right child of pivot node.
4. Left to right rotation -- Insertion in right subtree of left child of pivot node.

Left to Left rotation-

When the pivot node is left heavy and the new node is inserted in left subtree of the left child of pivot node then the rotation performed is left to left rotation.

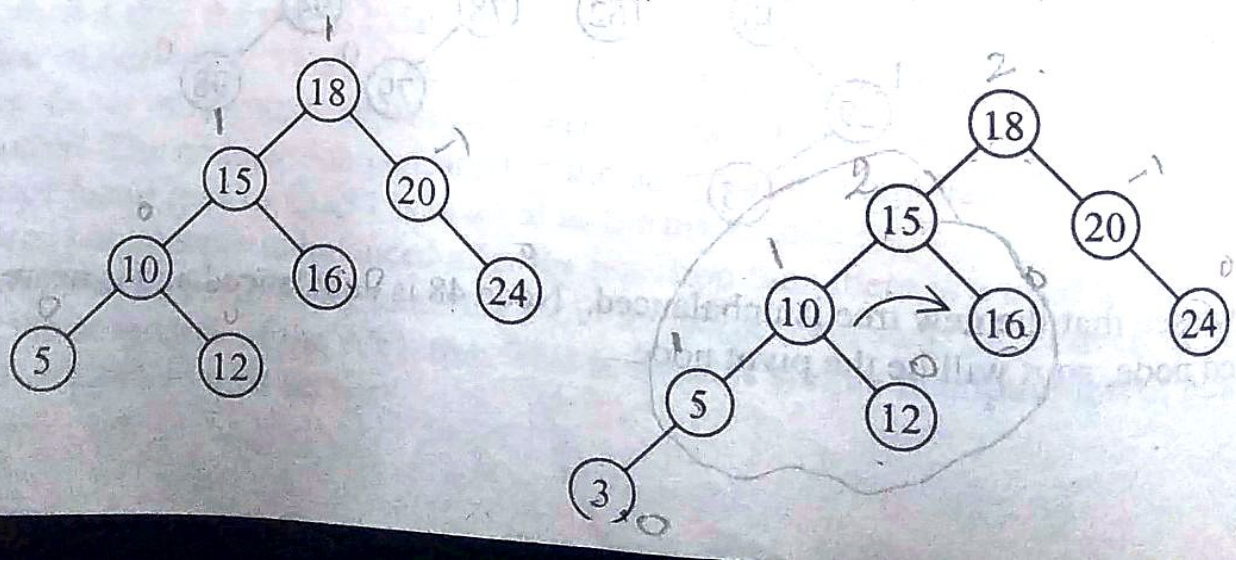


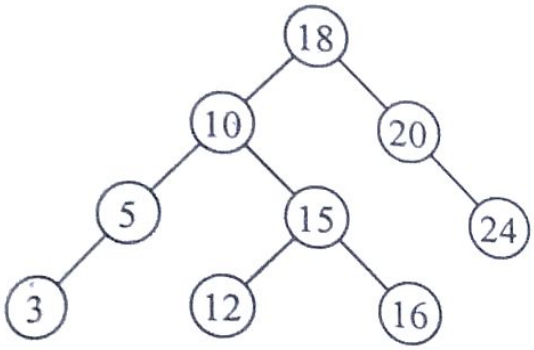
aptr = pptr->lchild;

pptr->lchild = aptr->rchild;
aptr->rchild=pptr;

pptr->balance=0;
aptr->balance=0;
pptr=aptr;

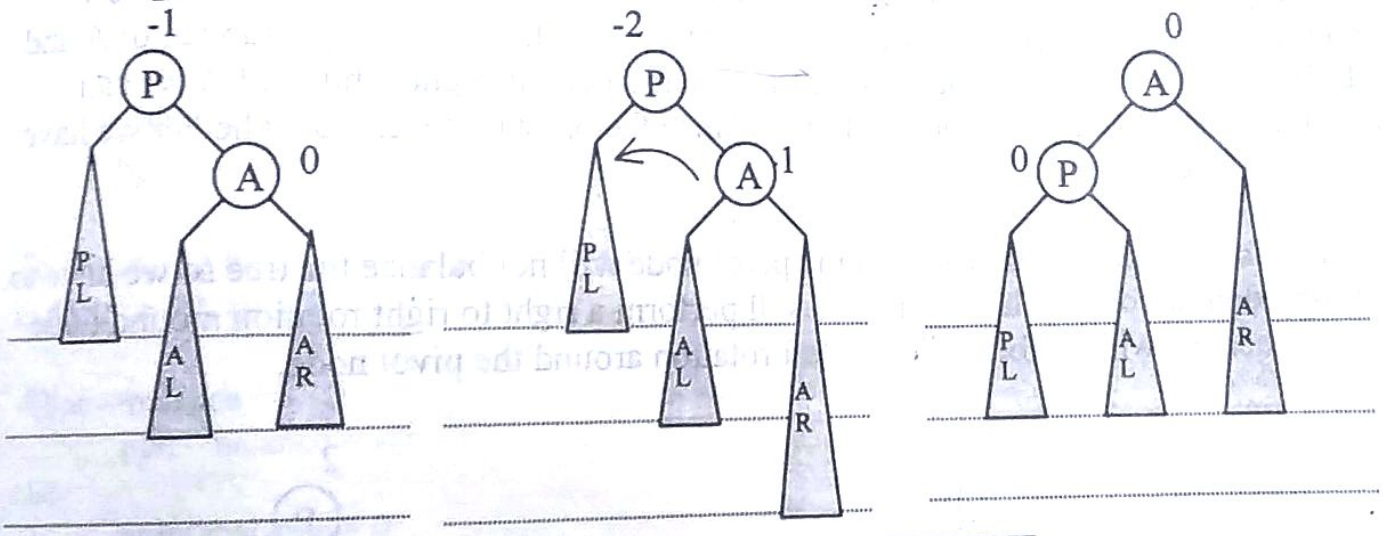
Here P represents the pivot node, A is left child of pivot node and AL and AR represent the left and right subtrees of node A. The right subtree of node P is PR. Pointer pptr points to the pivot node and aptr points to the node A. New node is inserted in the left subtree of A. The first figure shows the initial tree, second figure shows the tree after insertion and third figure shows the tree after rotation. The manipulation of pointers for the rotation is given below the figures. After rotating, we will make the balance factors of both pivot node and node A zero, and now node A will be the pivot node.
An example of left to left rotation is shown below-





Right to Right rotation-

When the pivot node is right heavy and the new node is inserted in right subtree of the right child of pivot node then the rotation performed is right to right rotation. This is the mirror image of left to left rotation.

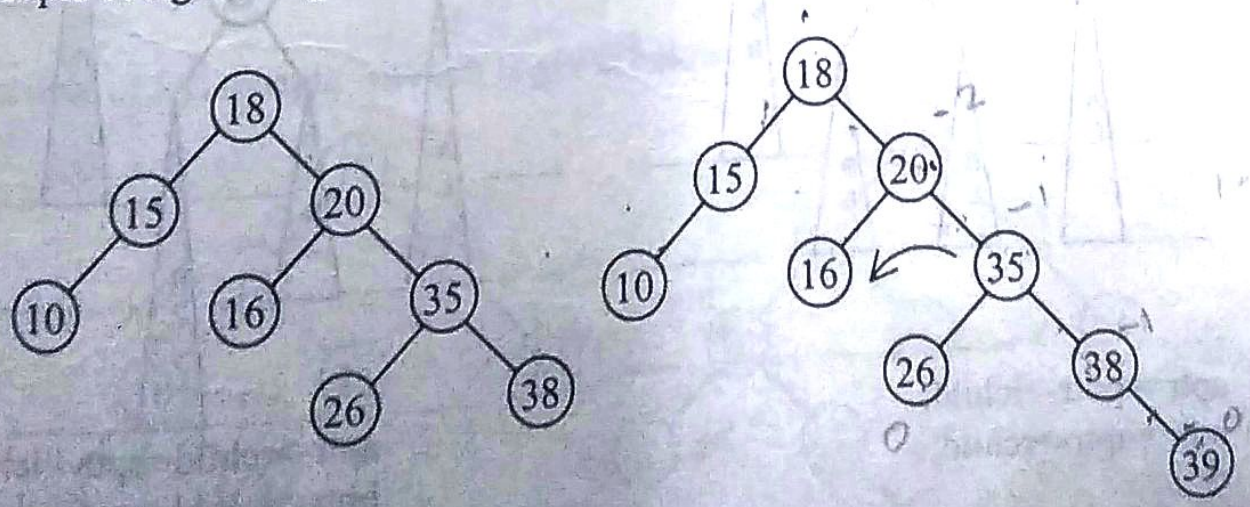


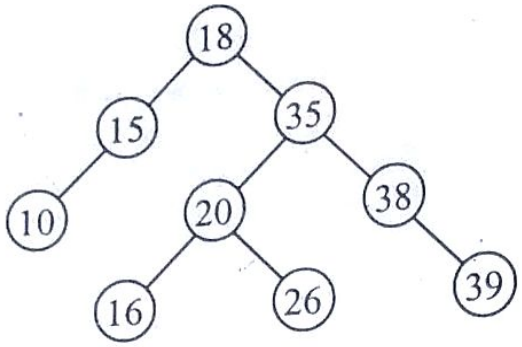
aptr=pptr->rchild;

pptr->rchild = aptr->lchild;
aptr->lchild=pptr;

pptr->balance=0;
aptr->balance=0
pptr=aptr;

An example of right to right rotation is shown below.

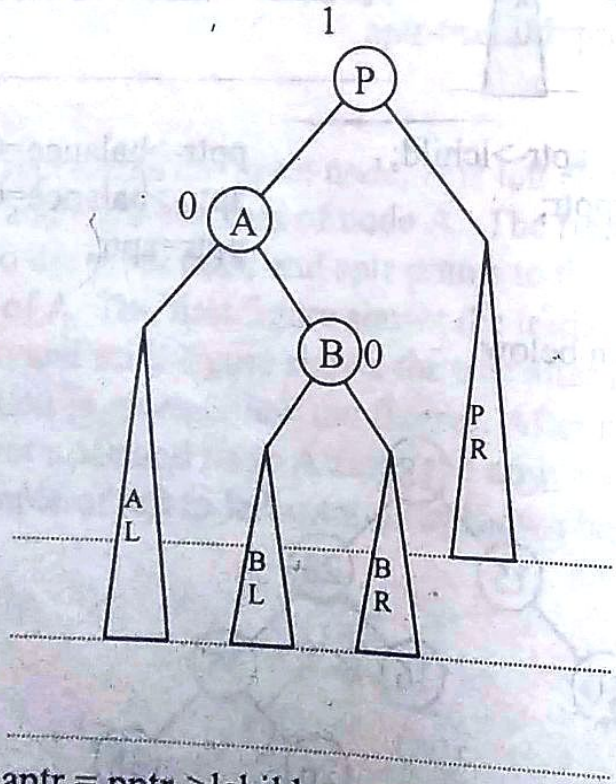




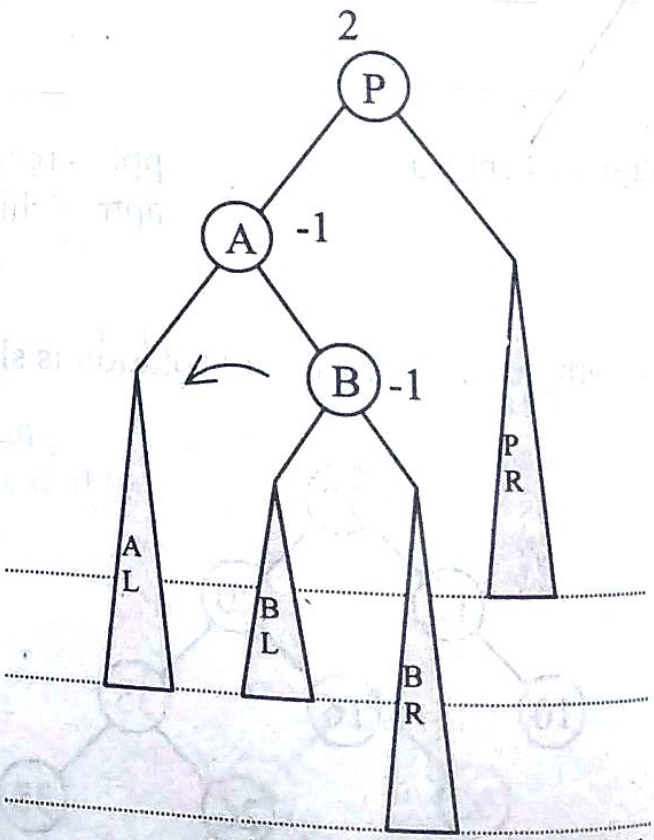
Left to right rotation

When the pivot node is left heavy and the new node is inserted in right subtree of the left child of pivot node then the rotation performed is left to right rotation. Here the new node is inserted in the right subtree of A. Suppose B is the root of right subtree of A and BL, BR are left and right subtrees of B. To insert a node in right subtree of A we can insert in either BL or BR. The resulting balance factors will depend on whether we have inserted in BL or BR.

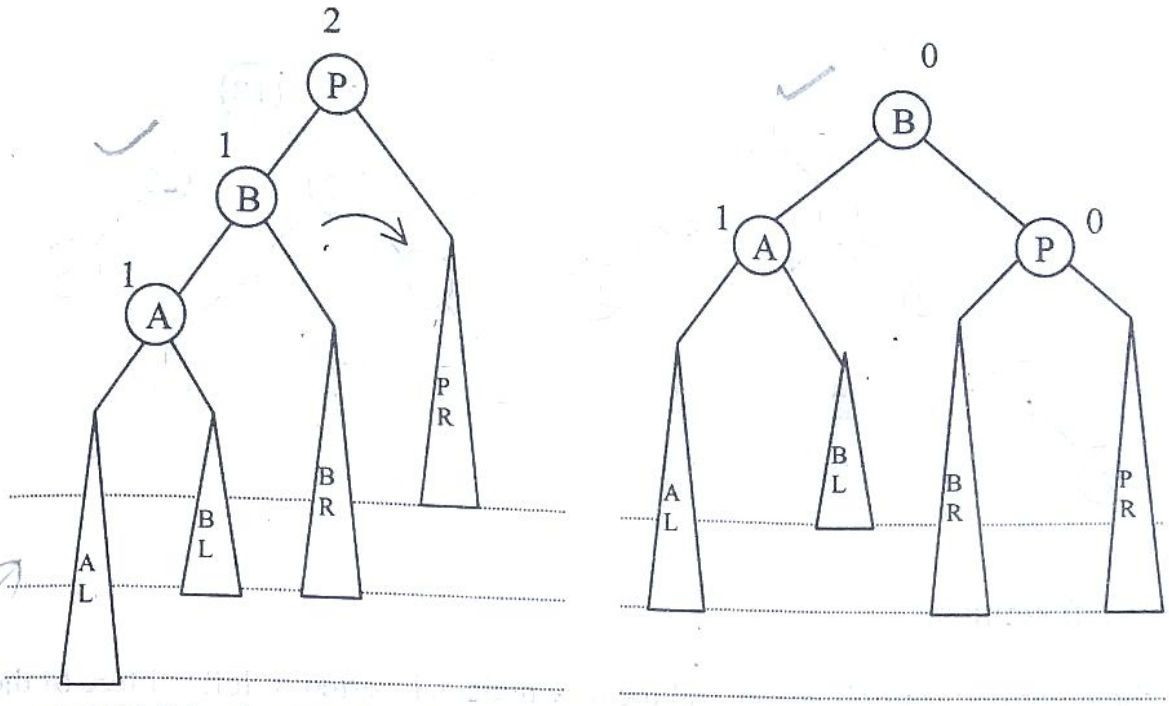
In this case a single rotation around the pivot node will not balance the tree so we have to perform double rotation here. First we will perform a right to right rotation around node A, and then we will perform a left to left rotation around the pivot node.



aptr = pptr->lchild;
bptr =aptr->rchild;



aptr->rchild=bptr->lchild;
bptr->lchild=aptr;

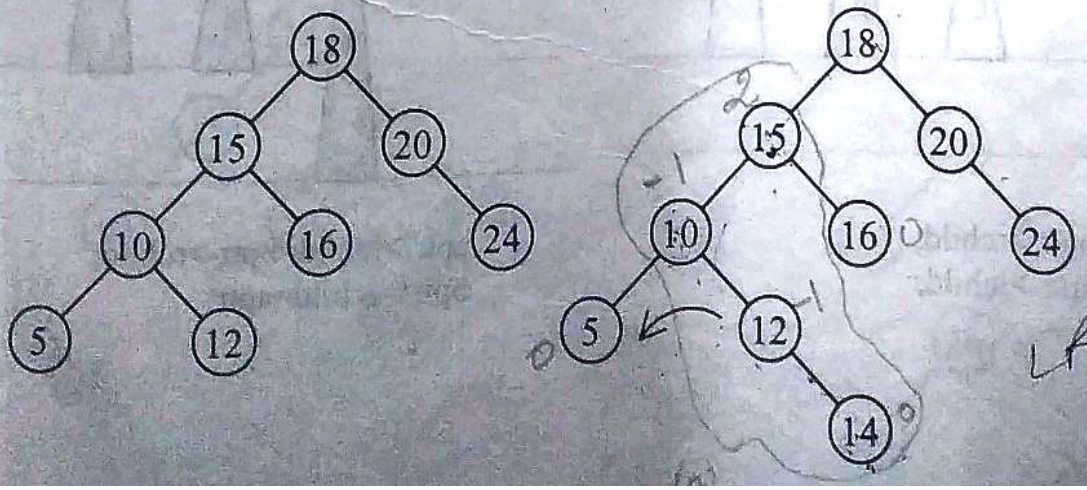


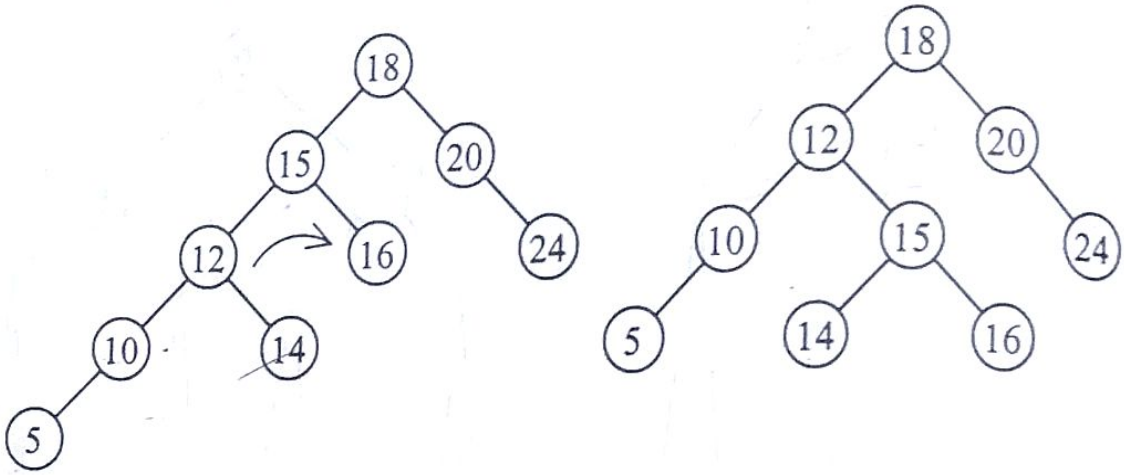

```
pptr->lchild = bptr->rchild;
bptr->rchild=pptr;
```

To adjust the balance factors we have to consider both the situations i.e whether insertion is done in BL or BR.

```
if(bptr->balance == 1 )
        pptr->balance = -1;
else
        pptr->balance = 0;
if(bptr->balance == -1)
        aptr->balance = 1;
else
        aptr->balance = 0;
bptr->balance=0;
pptr=bptr;
```

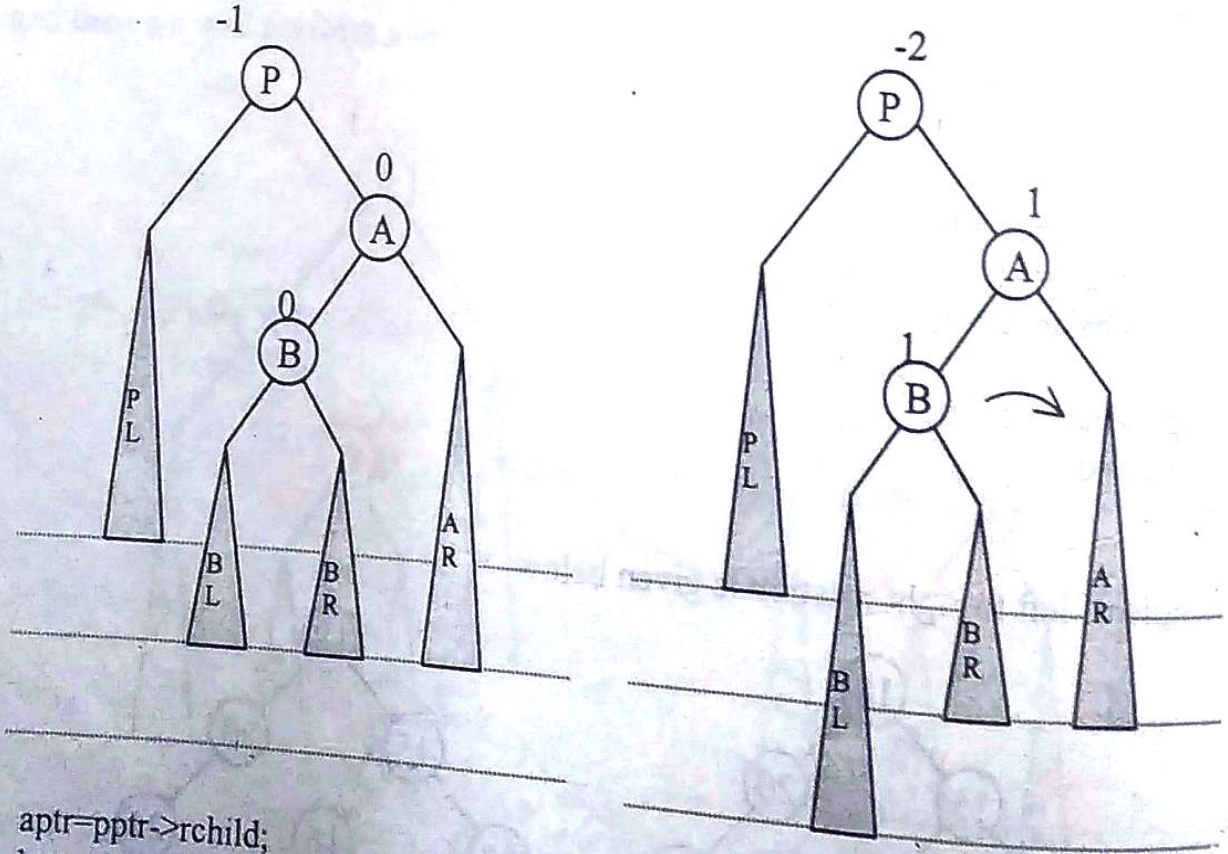
An example of left to right rotation is given below.





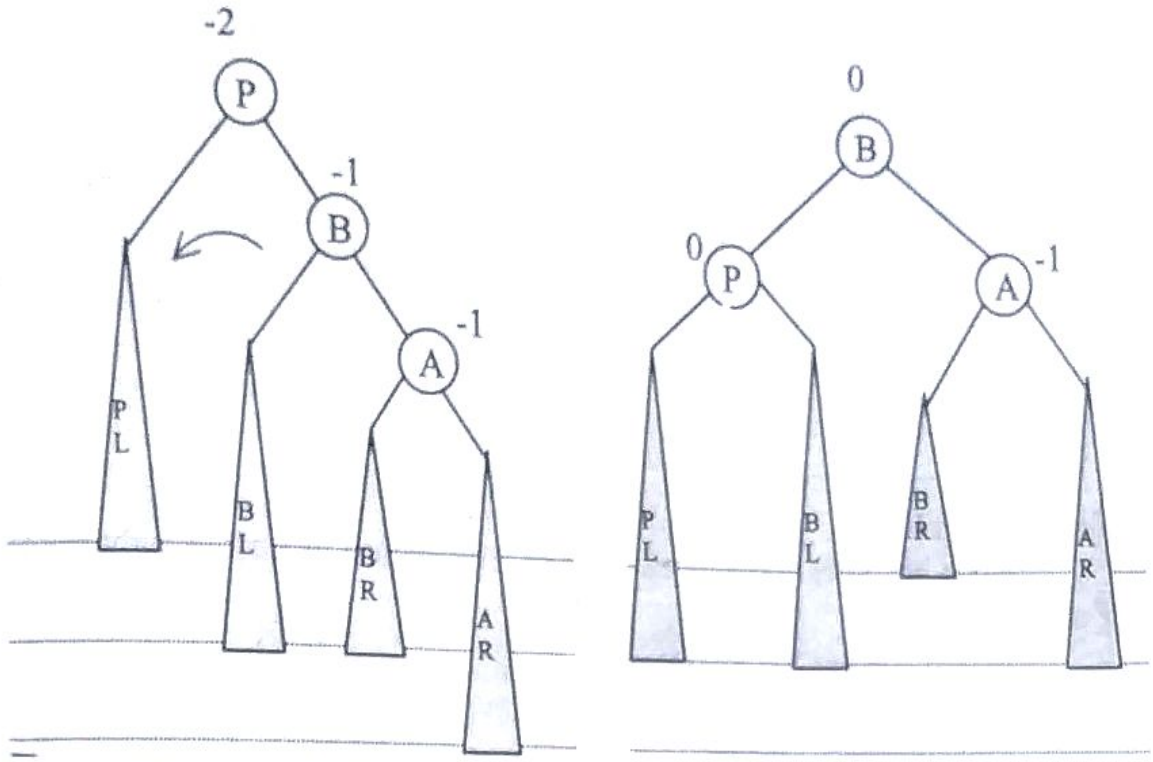
Right to left rotation-

When the pivot node is right heavy and the new node is inserted in left subtree of the right child of pivot node then the rotation performed is right to left rotation. Here also we have to perform double rotation. First we will perform a left to left rotation around node A, then right to right rotation around pivot node. We can see that this is the mirror image of the above case of left to right rotation.




```
aptr=pptr->rchild;
bptr=aptr->lchild;
```

```
aptr->lchild=bptr->rchild;
bptr->rchild=aptr;
```



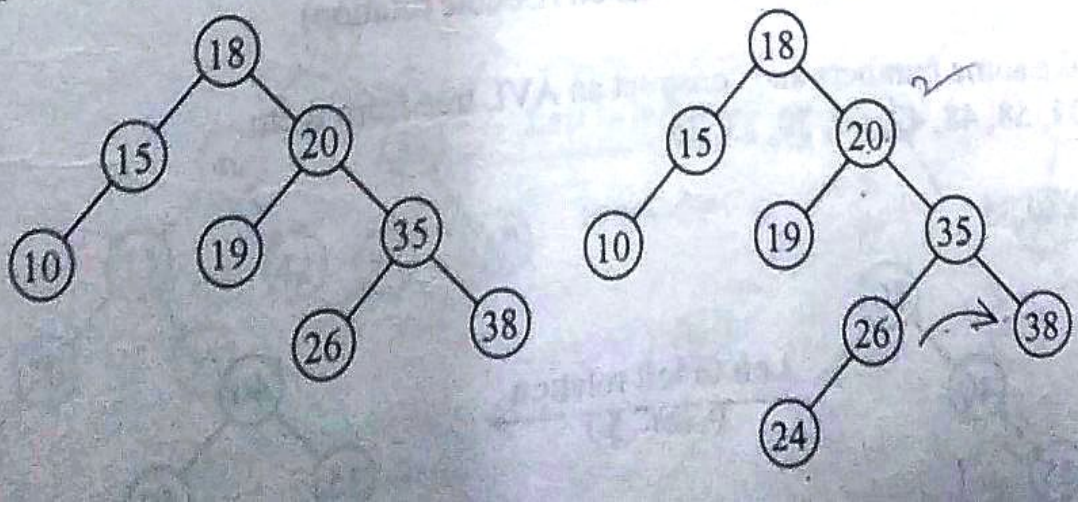

```
pptr->rchild=bptr->lchild;
bptr->lchild=pptr;
```

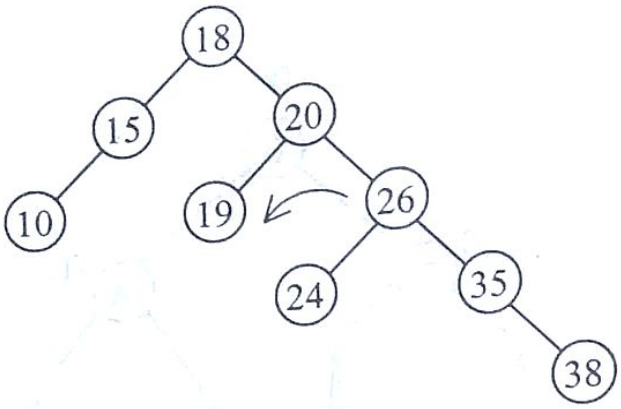
Here also we can insert in BL or BR so we have to adjust the balance factors accordingly.

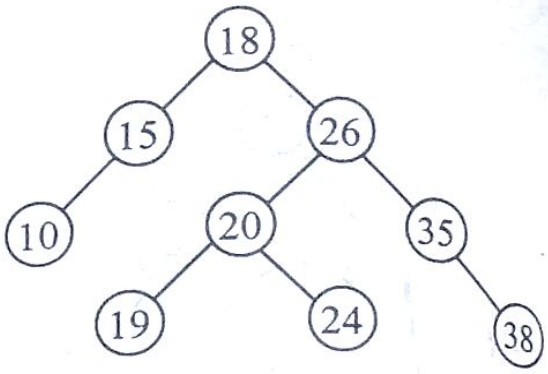
```
if(bptr->balance == -1)
        pptr->balance = 1;
else
        pptr->balance = 0;

if(bptr->balance == 1)
        aptr->balance = -1;
else
        aptr->balance = 0;
bptr->balance=0;
pptr = bptr;
```

An example of right to left rotation is shown below.







Insertion in left subtree of a node

If node is left heavy

Node becomes unbalanced (Left to left rotation or Left to right rotation)

If node is balanced

Node becomes left heavy

If node is right heavy

Node becomes balanced

Insertion in right subtree of a node

If node is left heavy

Node becomes balanced

If node is balanced

Node becomes right heavy

If node is right heavy

Node becomes unbalanced (Right to right rotation or Right to left rotation)

When the pivot node is left heavy and insertion in left subtree of pivot node.

Insertion in left subtree of left child of pivot node.

Left to left rotation.(single rotation)

Insertion in right subtree of left child of pivot node

Left to right rotation (Double rotation).

When the pivot node is right heavy and insertion in right subtree of pivot node.

Insertion in right subtree of right child of pivot node.

Right to right rotation.(single rotation)

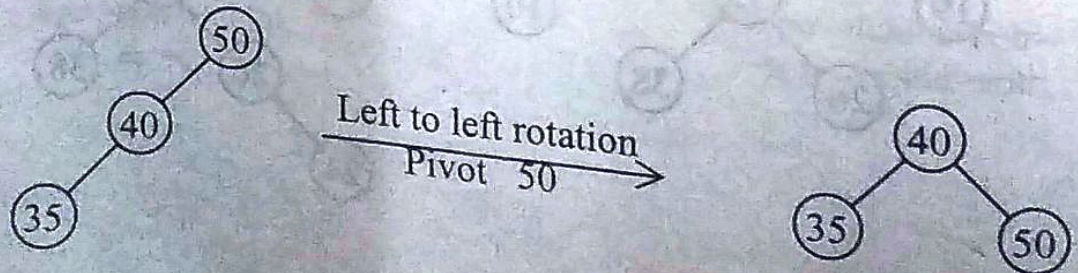
Insertion in left subtree of right child of pivot node

Right to left rotation (Double rotation).

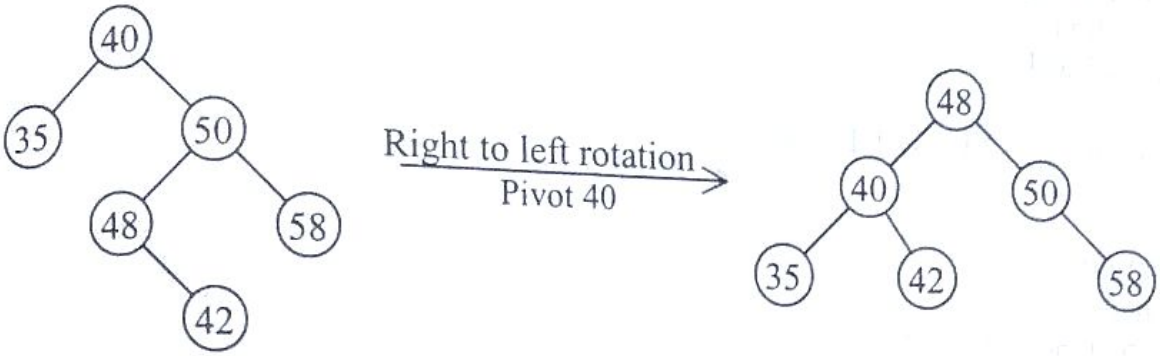
Let us take some numbers and construct an AVL tree from them.

50, 40, 35, 58, 48, 42, 60, 30, 33, 25

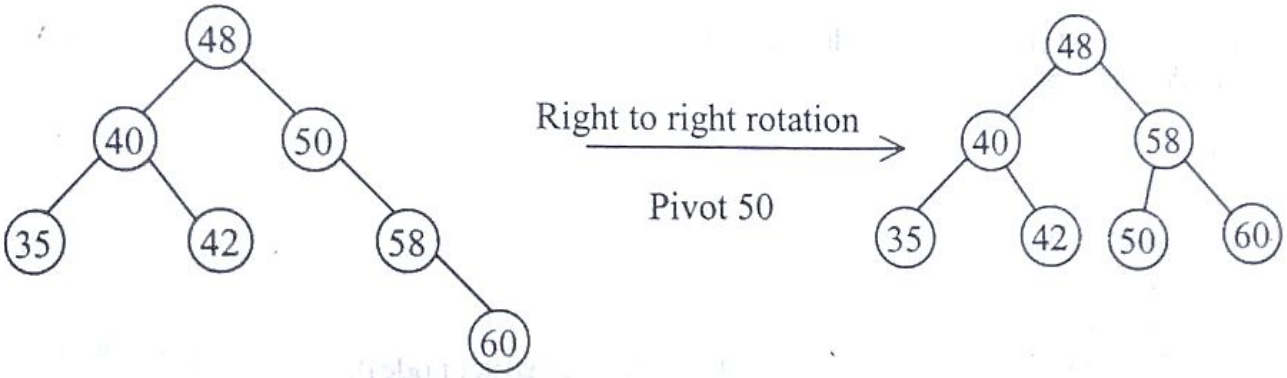
Insert 50,40,35



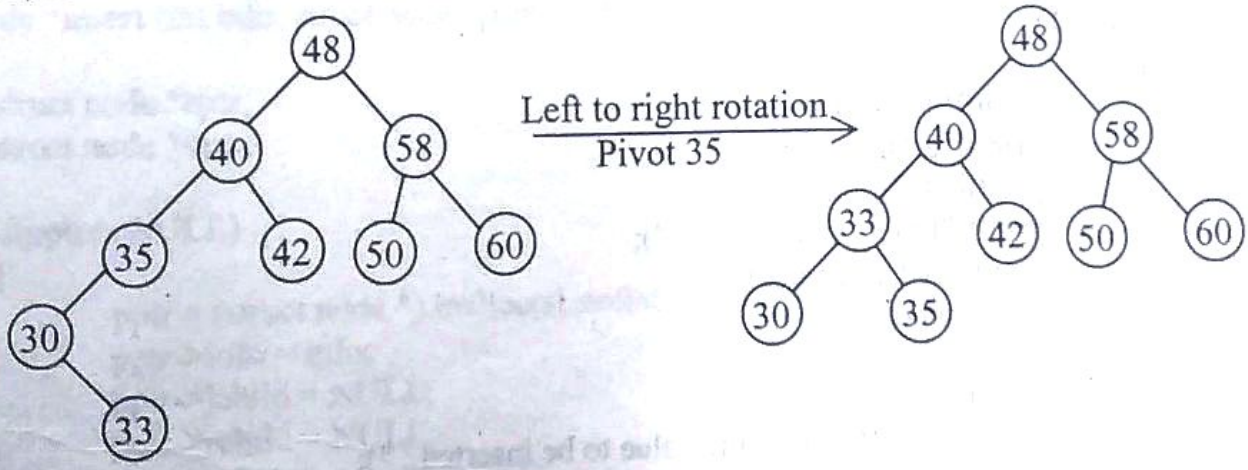
Insert 58,48,42



Insert 60



Insert 30, 33



Insert 25