

Seat no: S204156

Name: Sheuti Deepak Dhumre

PRN: 0220200161

Roll: 227

Batch: E1

ADS Open ended assignment

Q1. BST

- Binary search tree is a special type of tree data structure in which each node can have at most 2 children.
- Thus, in a BST each node has either 0 child or 1 child or 2 children
- The value of the key (~~root~~) of the left sub-tree is less than the value of its parent (root) node's key.
- The value of the key of the right sub-tree is greater than the value of its parent (root) node's key.

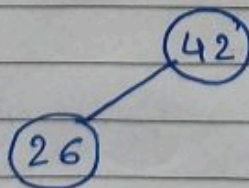
⇒ Construction of BST for the following values: 42, 26, 65, 21, 30, 50, 67, 15, 28, 37, 61, 69, 27. Assuming root node as 42.

Insert 42 →

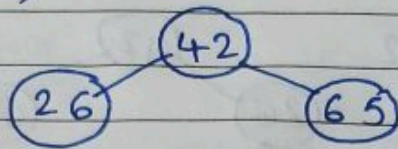
(42)

Insert 26 →

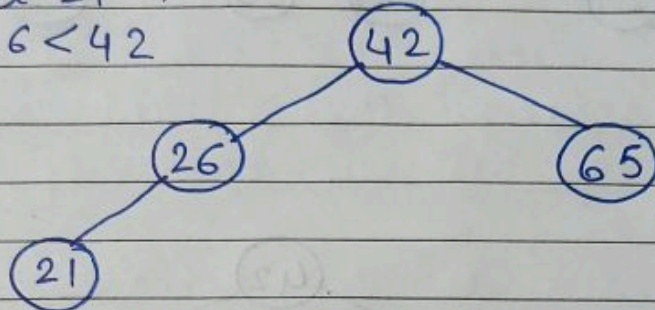
$42 < 26$



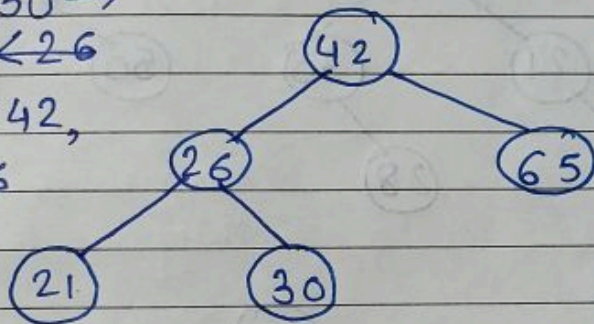
Insert 65 →
65 > 42



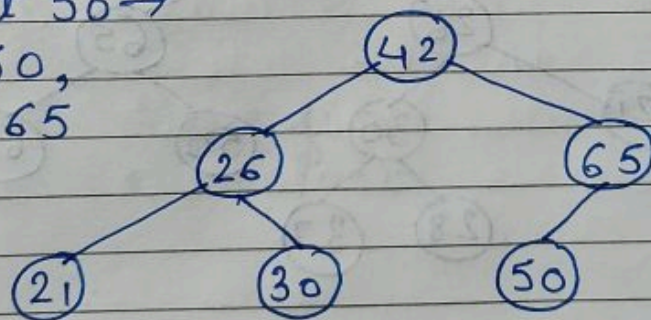
Insert 21 →
21 < 26 < 42



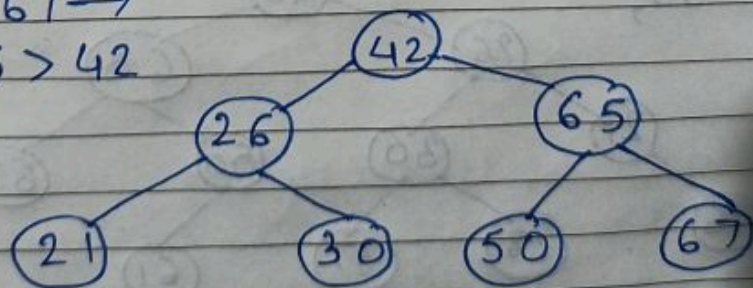
Insert 30 →
42 > 30 < 26
~~42~~ 30 < 42,
30 > 26



Insert 50 →
42 < 50,
50 < 65

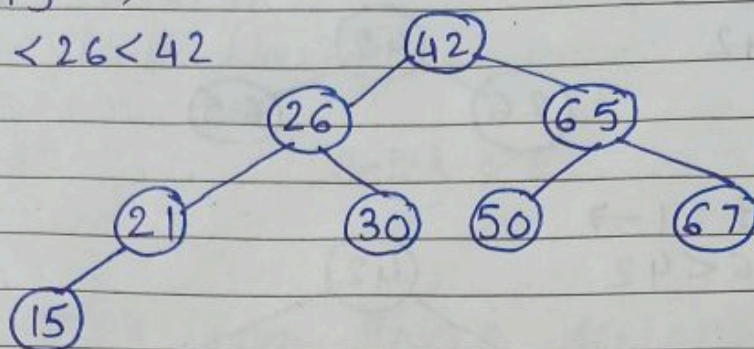


Insert 67 →
67 > 65 > 42



Insert 15 →

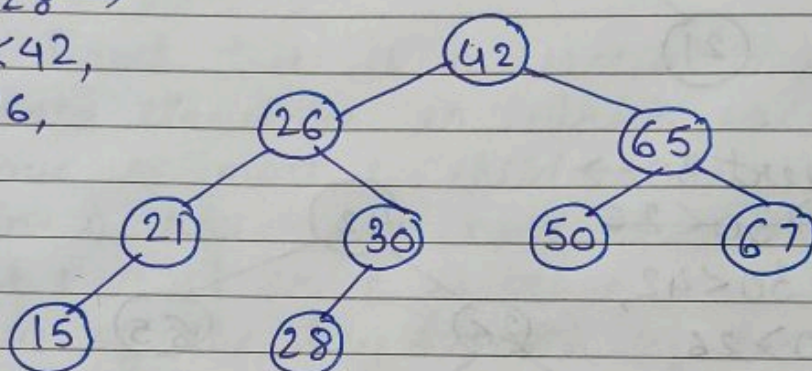
$$15 < 21 < 26 < 42$$



Insert 28 →

$$28 < 30 < 42,$$

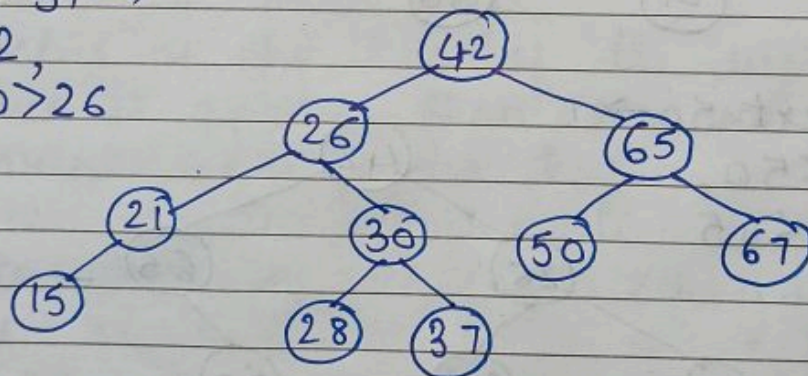
$$28 > 26,$$



Insert 37 →

$$37 < 42,$$

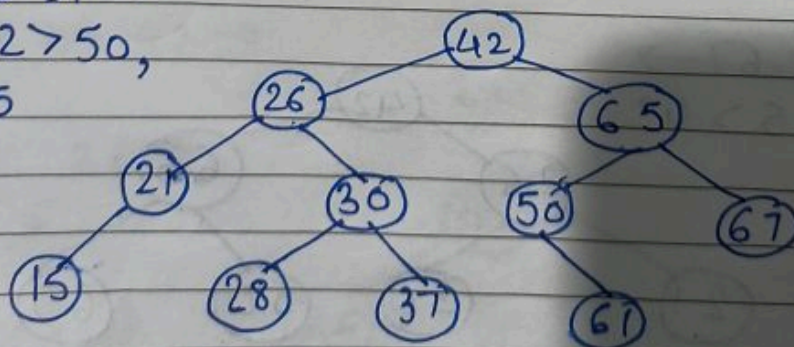
$$37 > 30 > 26$$



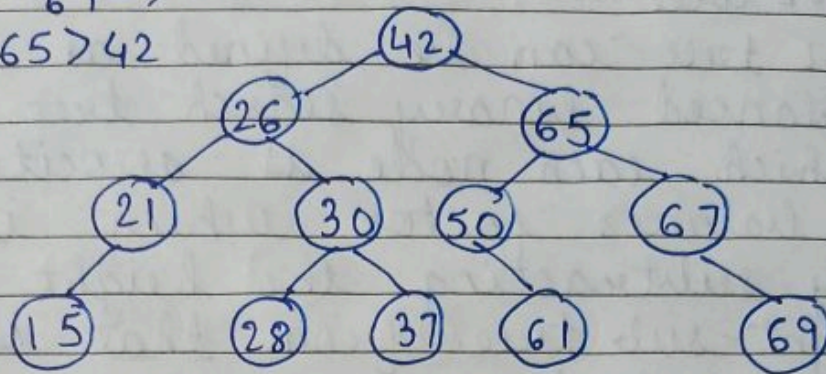
Insert 61 →

$$61 > 42 > 50,$$

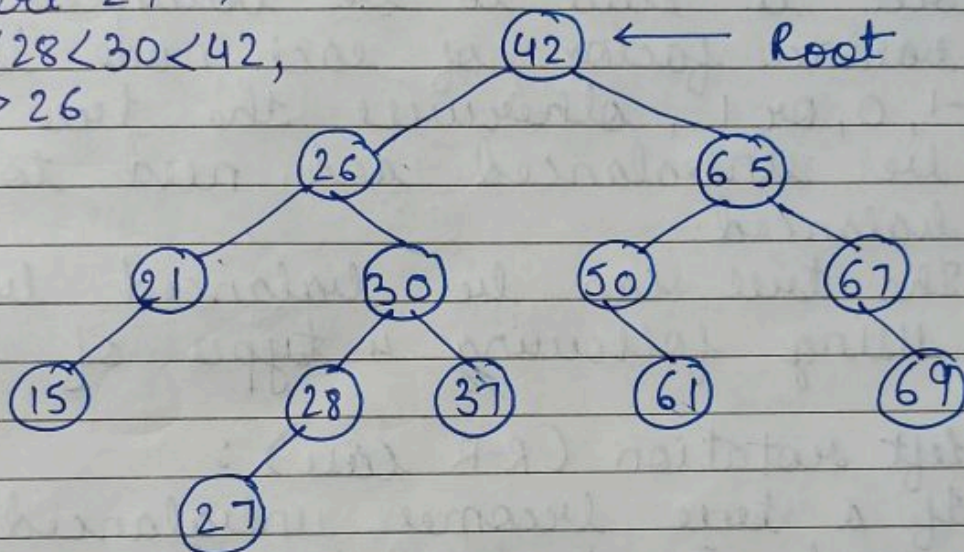
$$61 < 65$$



Insert 69 →
69 > 67 > 65 > 42



Insert 27 →
27 < 28 < 30 < 42,
27 > 26



This is the final result of the given BST values.

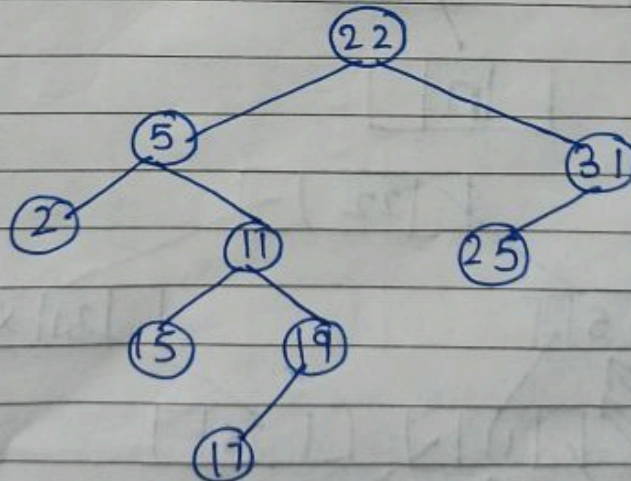
Q2.

TBT

- Threaded binary tree is a simple binary tree but they have a speciality that null pointers of leaf node of the binary tree is set to inorder predecessor or inorder successor.
- There are 2 types of TBT they are:
 - ① Single threaded :- each node is threaded towards either the inorder predecessor or successor (left or right)
 - ② Double threaded :- each node is threaded towards both the inorder predecessor and successor (left and right)

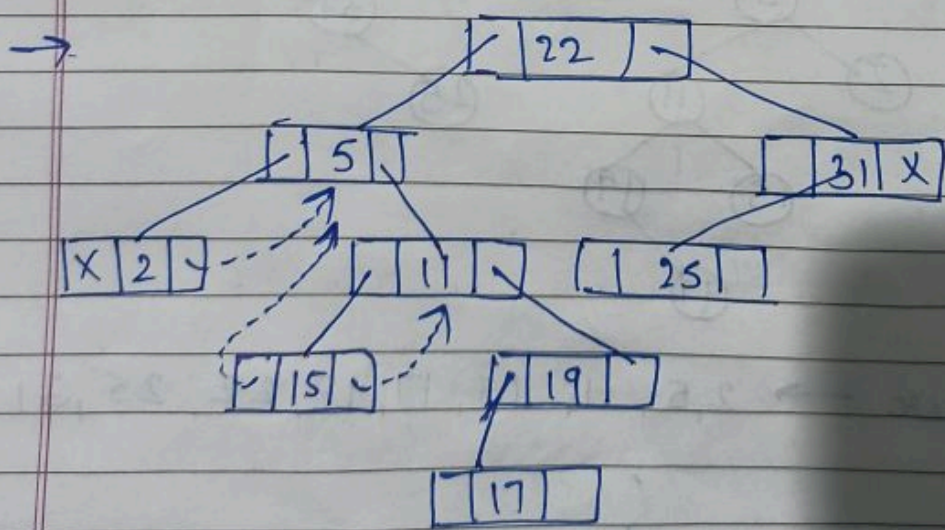
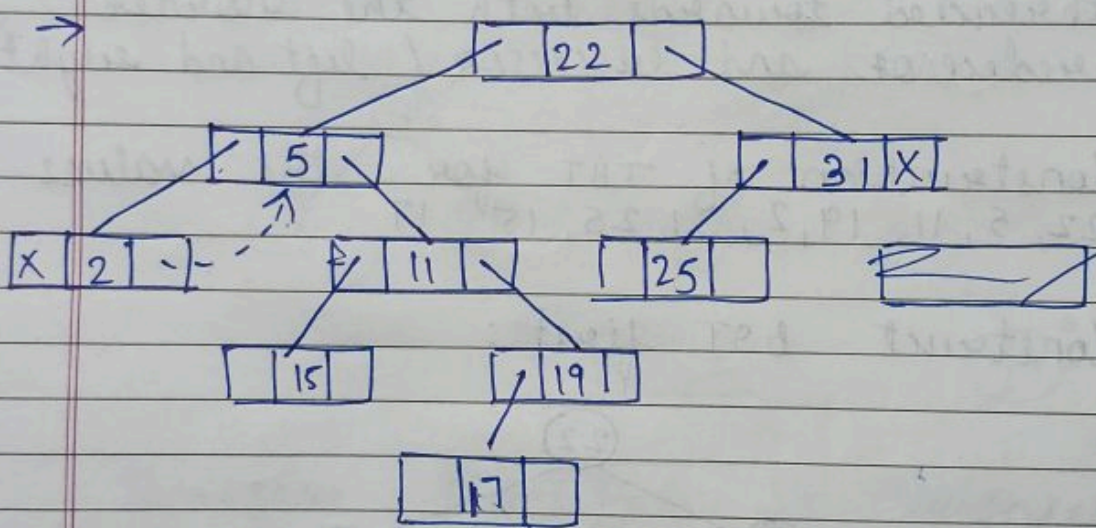
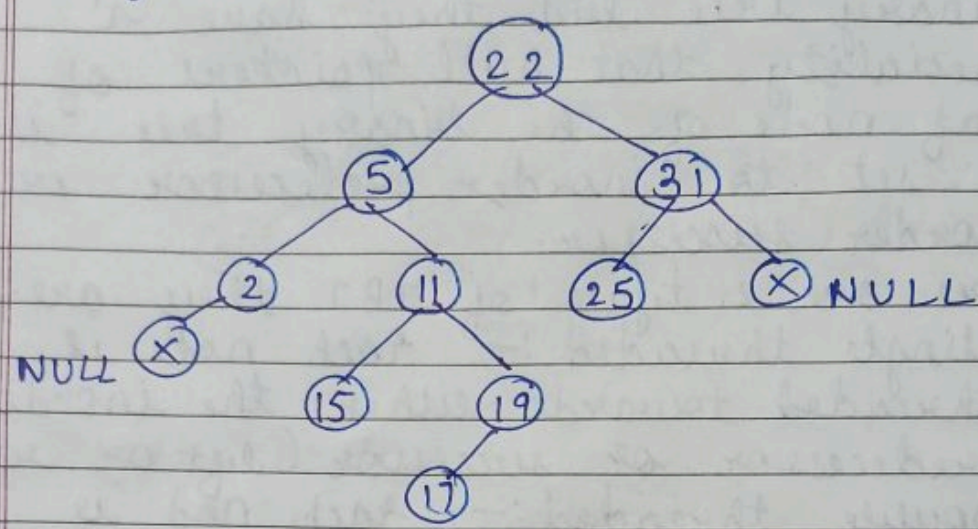
⇒ Construction of TBT for the values 22, 5, 11, 19, 2, 31, 25, 15, 17

Construct BST first:

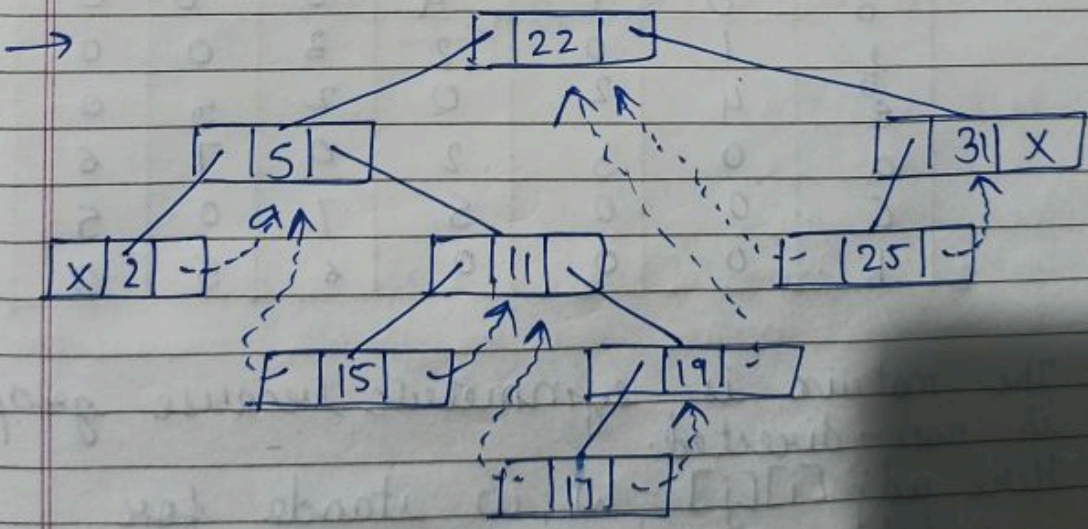
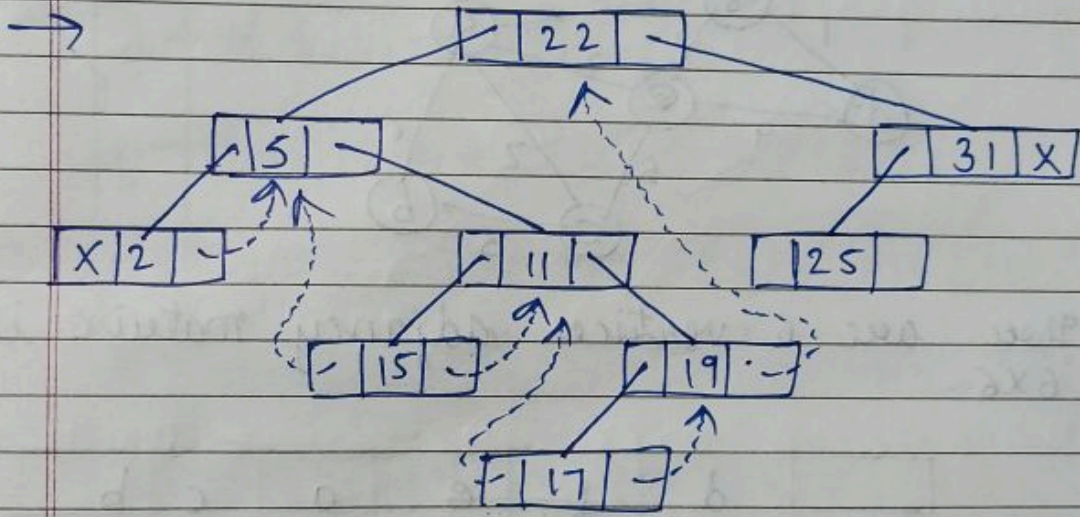
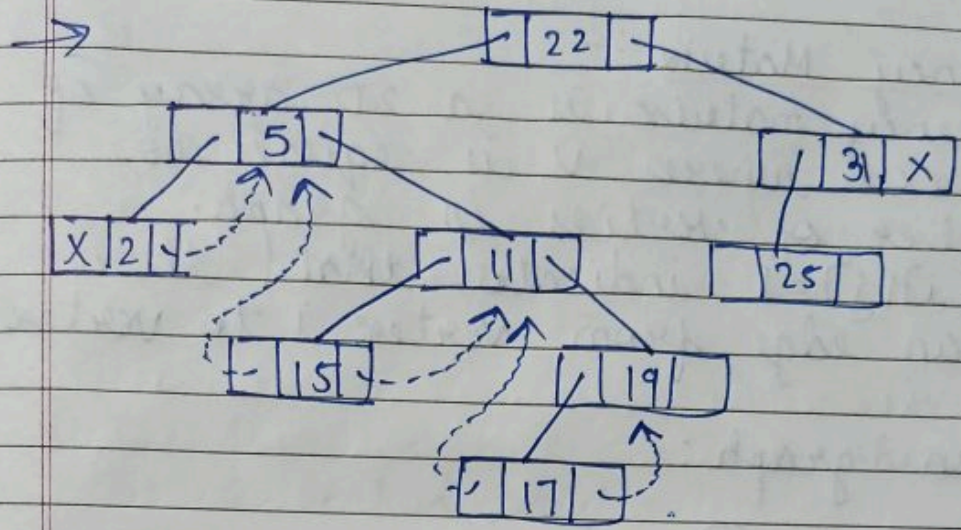


Inorder → 2, 5, 11, 15, 17, 19, 22, 25, 31

Q2. ^{& rightmost} goto the leftmost node and set the left link to null.



Q2



Hence, all the threads are corrected

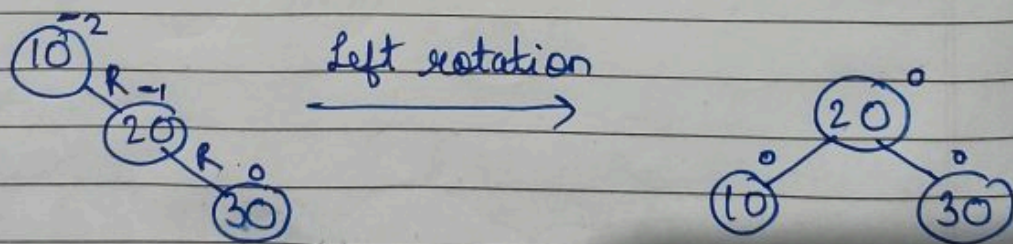
Q3. AVL Tree ---

→ - AVL tree can be defined as height balanced binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height of its right-sub tree from that of its left-sub tree.

- Tree is said to be balanced if balance factor of each node is ~~not~~ $-1, 0, \text{ or } 1$, otherwise the tree will be unbalanced and need to be balanced.
- The tree can be balanced by using following 4 types of rotation.

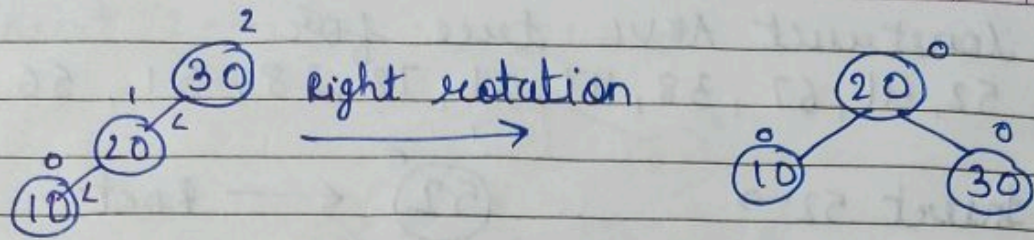
i Left rotation (R-R case):

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

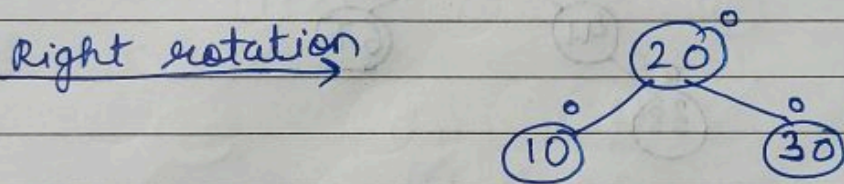
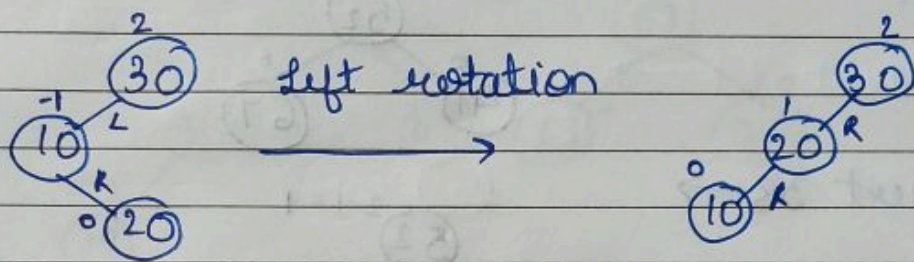


ii Right rotation (L-L case):

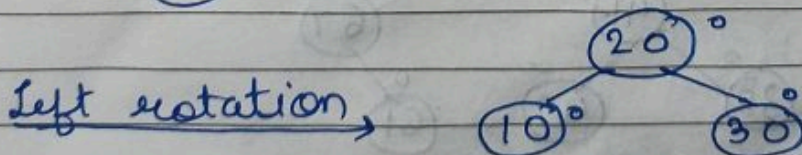
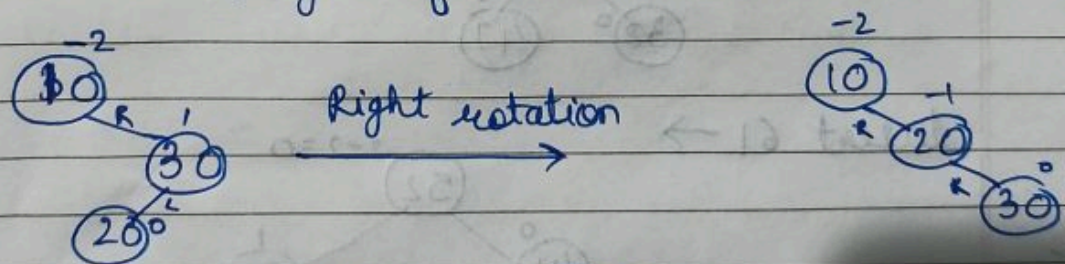
~~The~~ AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. Then the tree needs right rotation.



iii Left right rotation (LR case):
A left right rotation is a combination of left rotation followed by right rotation.

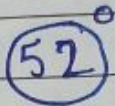


iv Right-left rotation (RL case):
It is a combination of right rotation followed by left rotation.

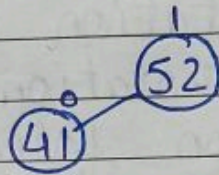


Note: Always keep the rules in your mind while doing the rotation.

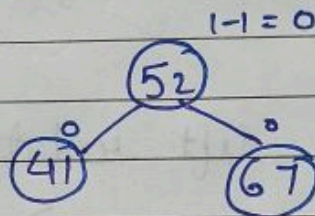
Q4. Construct AVL tree for
52, 41, 67, 38, 47, 61, 72, 28, 51, 56, 77

Insert 52 →  ← Root

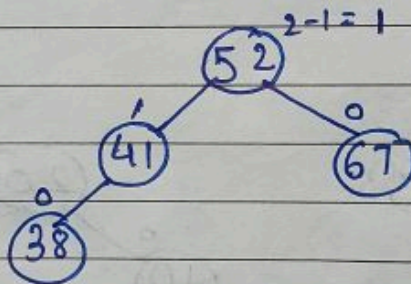
Insert 41 →



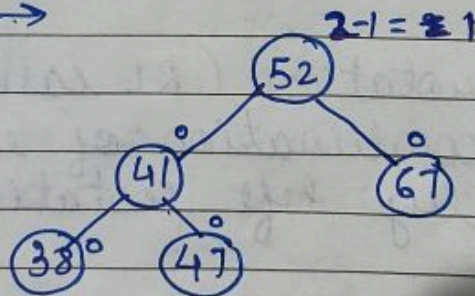
Insert 67 →



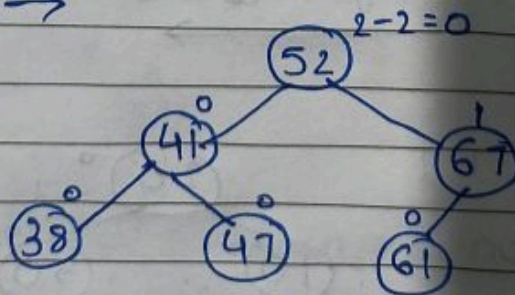
Insert 38 →



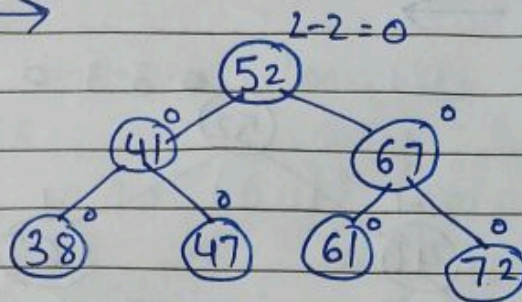
Insert 47 →



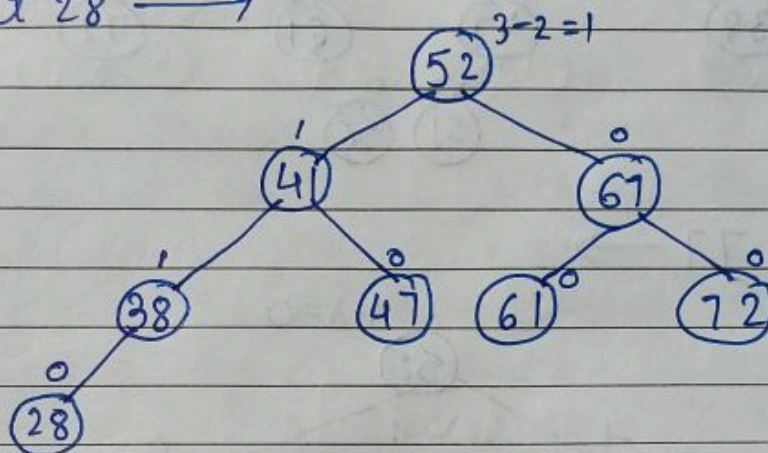
Insert 61 →



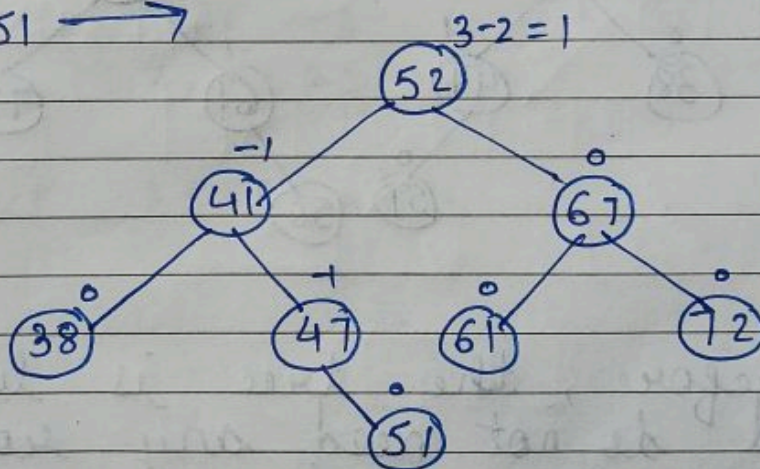
Q4 Insert 72 \rightarrow



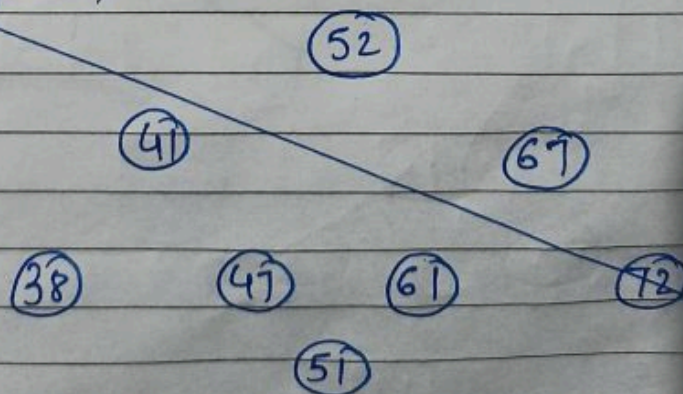
Insert 28 \rightarrow



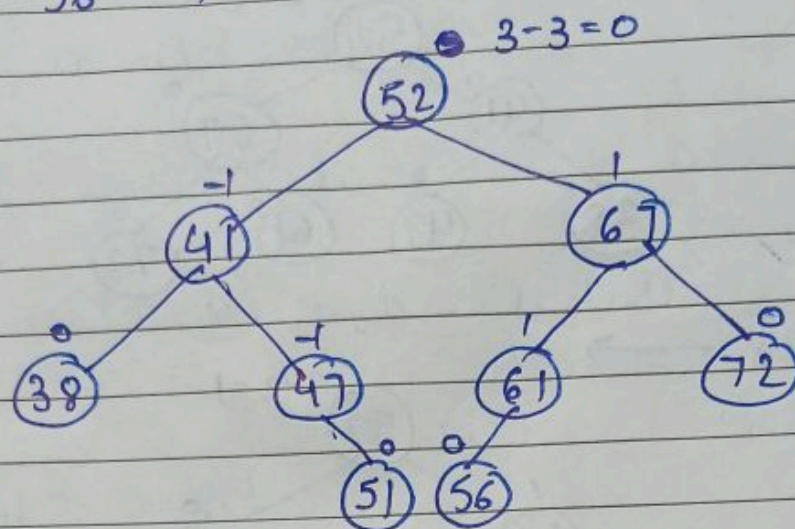
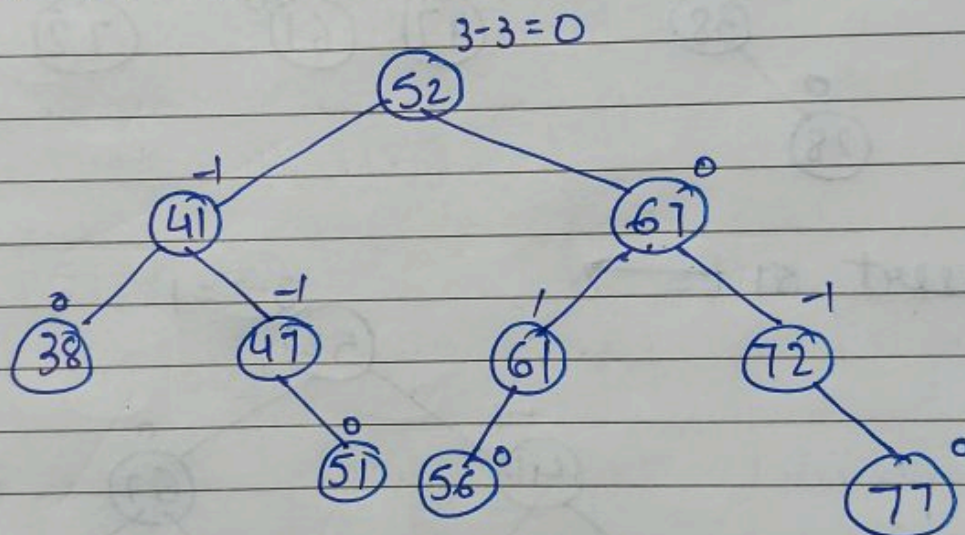
Insert 51 \rightarrow



~~Insert 56 \rightarrow~~



Q4

Insert 56 \rightarrow Insert 77 \rightarrow 

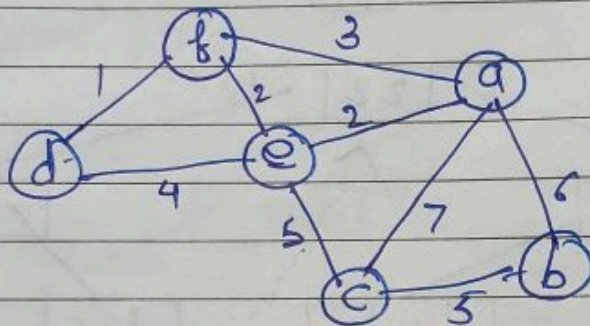
Therefore, the tree is balanced and do not need any rotation.

Q5.

Adjacency Matrix :-

- Adjacency matrix is a 2D array of size $V \times V$ where V is equal to number of vertices in graph.
- $adj[i][j] = 1$ indicates that there is an edge from vertex i to vertex j .

Given graph :



There are 6 vertices adjacency matrix is 6×6 .

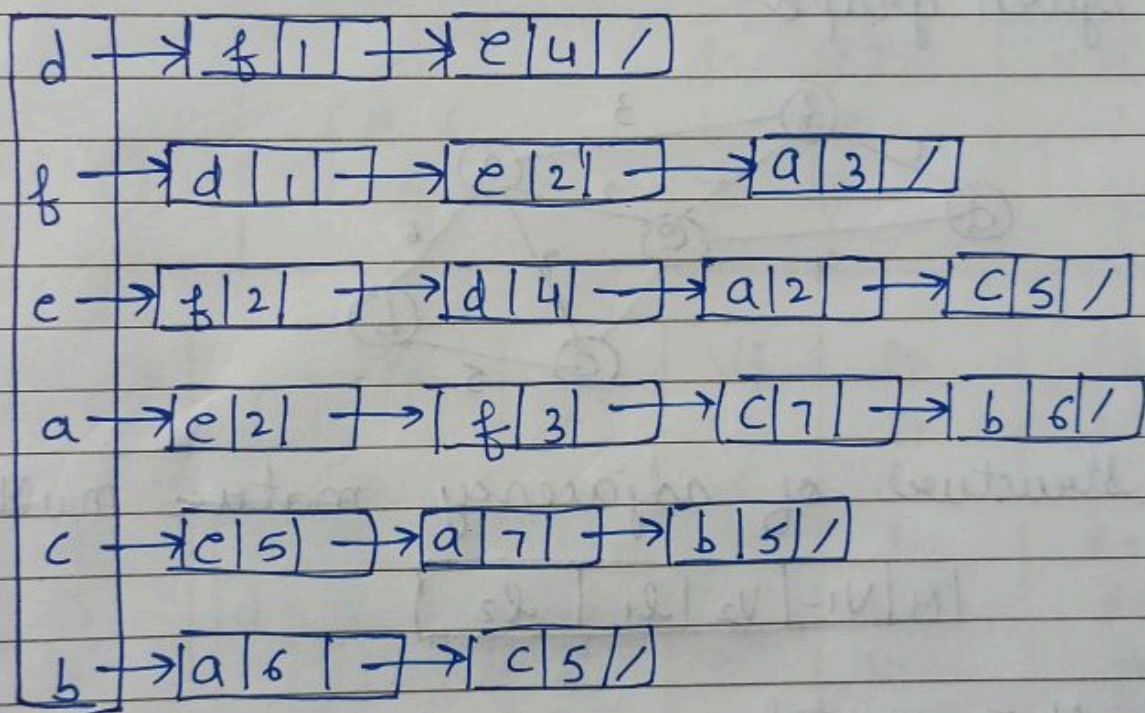
	d	f	e	a	c	b
d	0	1	4	0	0	0
f	1	0	2	3	0	0
e	4	2	0	2	5	0
a	0	3	2	0	7	6
c	0	0	5	7	0	5
b	0	0	0	6	5	0

- The matrix is symmetric because graph is non-directed.
- Here $adj[i][j] = w$, w stands for weight of edge.

adjacency list :-

- array of list is used.
- size of array is equal to number of vertices.
- each list contains the node which are adjacent to vertex.

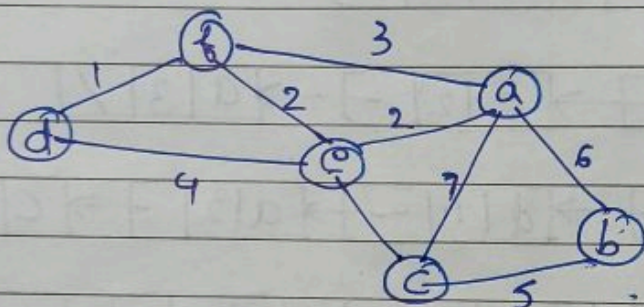
as there are 6 vertices there will be 6 linked list.



Q6. Adjacency Multi-list.

- It is a multi list representation of graph; a directory of node information
- * & set of linked list of edge information.
- There is one entry for each node in graph
- The directory entry for node i points to linked adjacency list for node i

given graph:-



Structure of adjacency matrix multi list:

M	V ₁	V ₂	l ₁	l ₂
---	----------------	----------------	----------------	----------------

M = visited

V₁ = Vertex 1

V₂ = Vertex 2

l₁ = list name where V₁ is present

l₂ = list name where V₂ is present

$$E = \{(a,b), (a,c), (a,e), (b,c), (c,e), (e,f), (e,d), (d,f), (a,f)\}$$

Vertex	list of edges	Name	Weight
a	(a,b)	l_1	6
b	(a,c)	l_2	7
c	(a,e)	l_3	2
d	(a,f)	l_4	3
e	(b,c)	l_5	5
f	(c,e)	l_6	5
	(e,f)	l_7	
	(d,e)	l_7	4
	(d,f)	l_8	1
	(e,f)	l_9	2

edges	Weight		u	V_i	V_j	l_i	l_j	
l_1	6	a		a	b	l_1	l_5	l_1
l_2	7	b		a	c	l_2	l_6	l_2
l_3	2	c		a	e	l_3	l_7	l_3
l_4	3	d		a	f	NULL	l_8	l_4
l_5	5	e		b	c	NULL	l_6	l_5
l_6	5	f		c	e	NULL	l_7	l_6
l_7	4			d	e	l_8	l_9	l_7
l_8	1			d	f	NULL	l_9	l_8
l_9	2			e	f	NULL	NULL	l_9

adjacency multi list.

vertex a : $l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4$
 vertex b : $l_1 \rightarrow l_5$
 vertex c : $l_2 \rightarrow l_5 \rightarrow l_6$
 vertex d : $l_7 \rightarrow l_8$
 vertex e : $l_3 \rightarrow l_6, l_7 \rightarrow l_9$
 vertex f : $l_4 \rightarrow l_8 \rightarrow l_9$