Rishith Reddy-B
23211A05R3
CSE-D

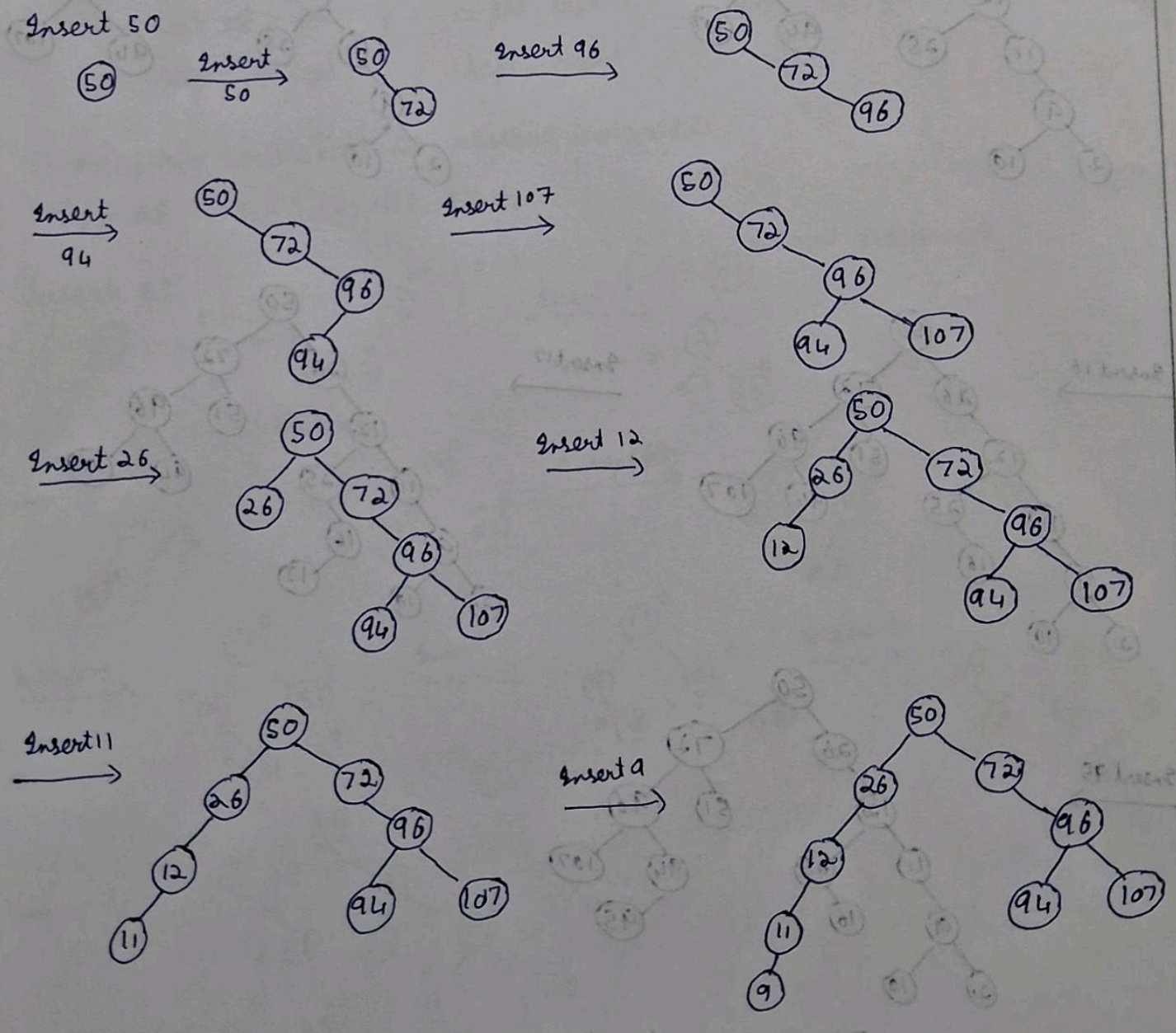
ADS Assignment-1

1) Construct the binary search tree whose elements are inserted in the following order
50, 72, 96, 94, 107, 26, 12, 11, 9, 2, 10, 25, 51, 16, 17, 95

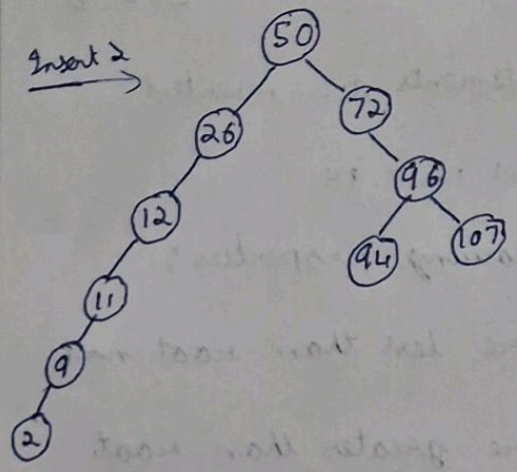
Ans:- Insertion in Binary search tree follows following properties:

1) All the left subtree values are less than root node value
2) All the right subtree values are greater than root node value
3) Every subtree should satisfy BST properties
4) No duplicate values allowed

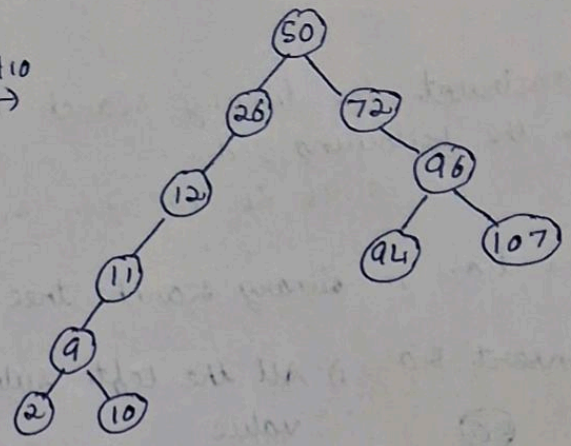
Insert 50



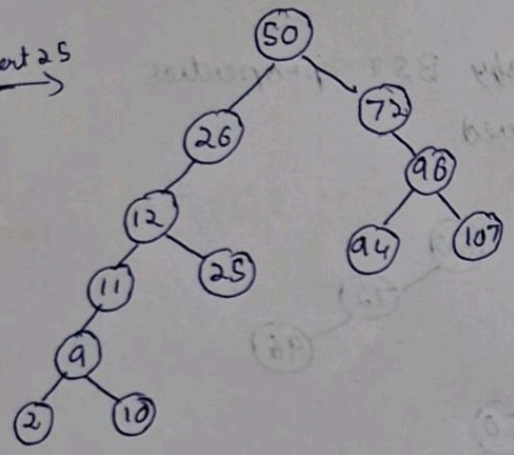
Insert 2 →



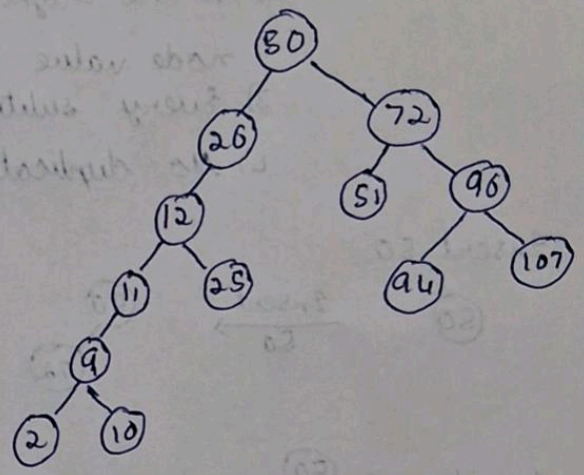
Insert 10 →



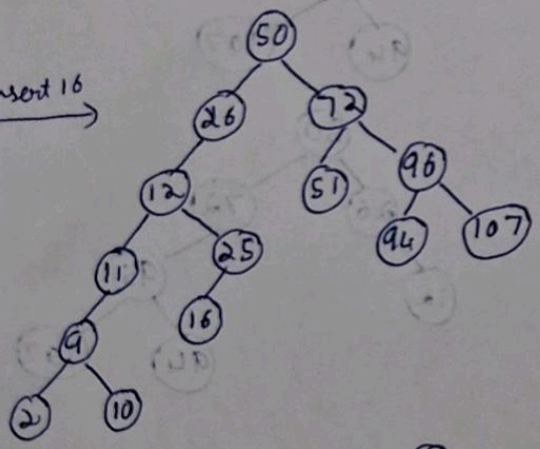
Insert 25 →



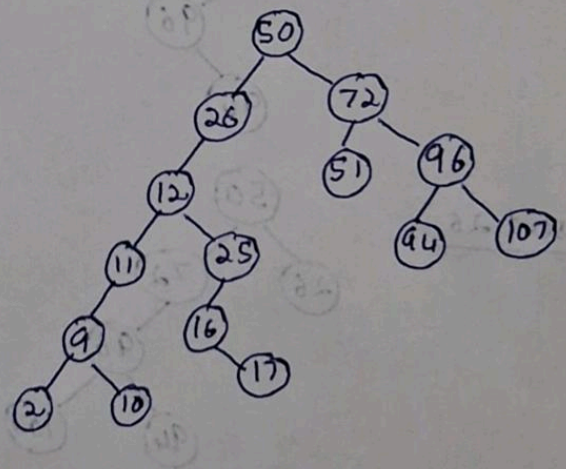
Insert 51 →



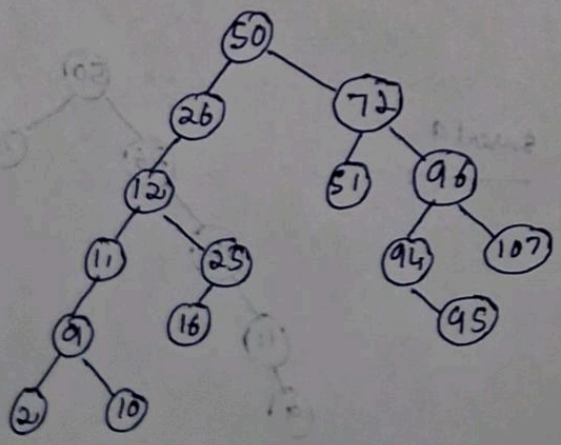
Insert 16 →



Insert 17 →



Insert 95 →



Binary search Tree

2) Discuss the procedure to insert a node for a given AVL tree
Construct AVL Tree for given list of keys: 63, 9, 19, 27, 18, 108, 99, 81

Algorithm:

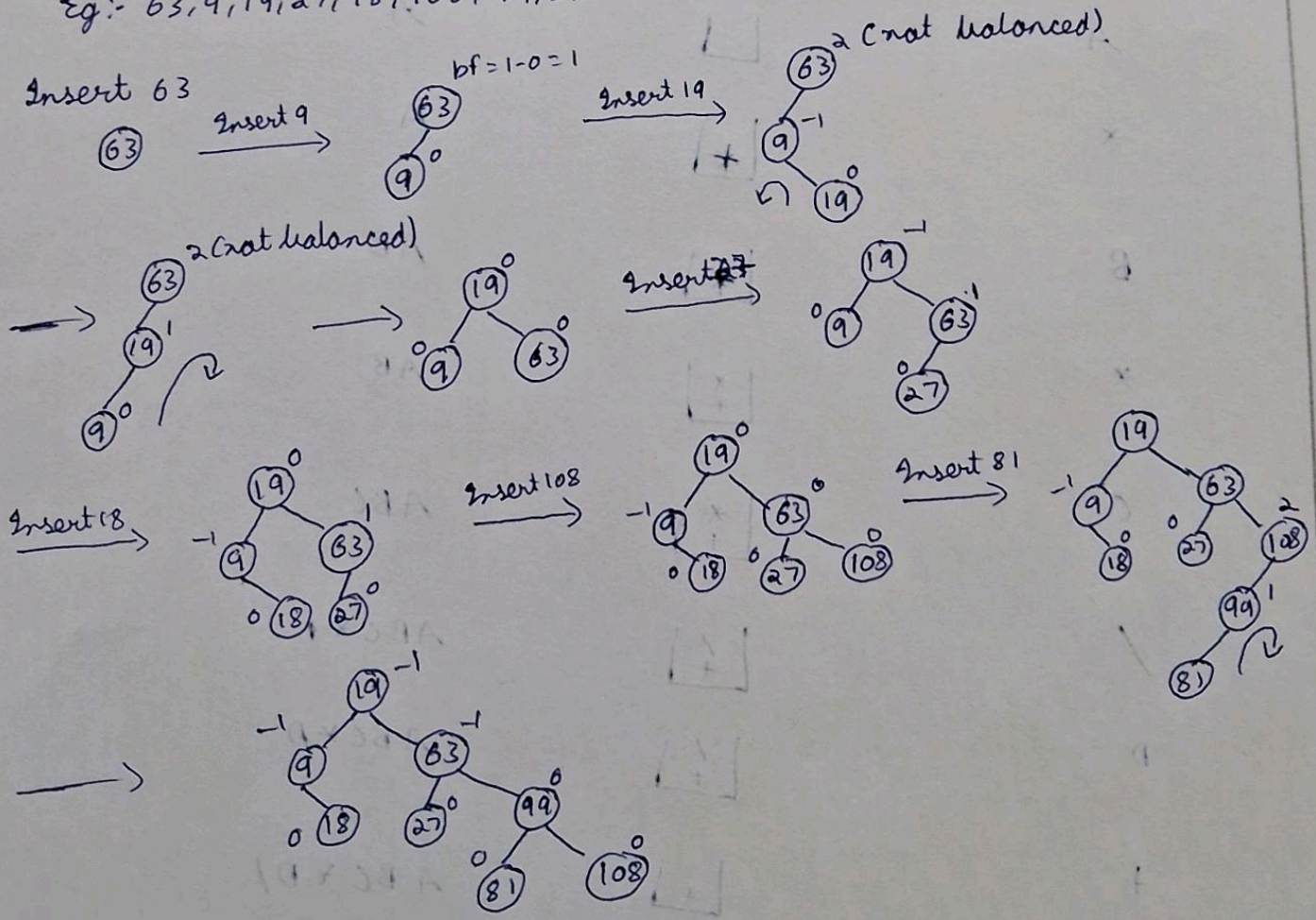
1) Insert node in normal BST
2) Check Balance factor for each ancestor node:
   Balance factor = height(left) - height (right)
   if (balance factor = 0 or 1 or -1 then tree is AVL
   otherwise we have to apply rotations

3)

case	Rotation
Right of Right	Left
Left of Left	Right
Left of right	right, left
right of left	left, right

4) Perform rotations to restore balance

Eg:- 63, 9, 19, 27, 18, 108, 99, 81



AVL Tree

3) Write Algorithm for expression tree. construct expression tree for the given infix expression : A+B*C/D+E/F+G*H

Algorithm :

1) Convert infex to postfix
2) Read all characters from postfix expressions
3) If character is operand then push it into stack
4) If character is operator
   - pop the top most two elements from
   - Consider operator as root node and constructor tree with elements as right child & left child
   - Push the tree as element onto stack
5) Repeat the process until reading all the from postfix expression.

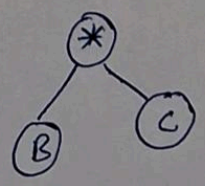
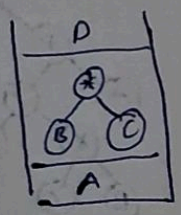
A + B * C / D + E / F + G * H

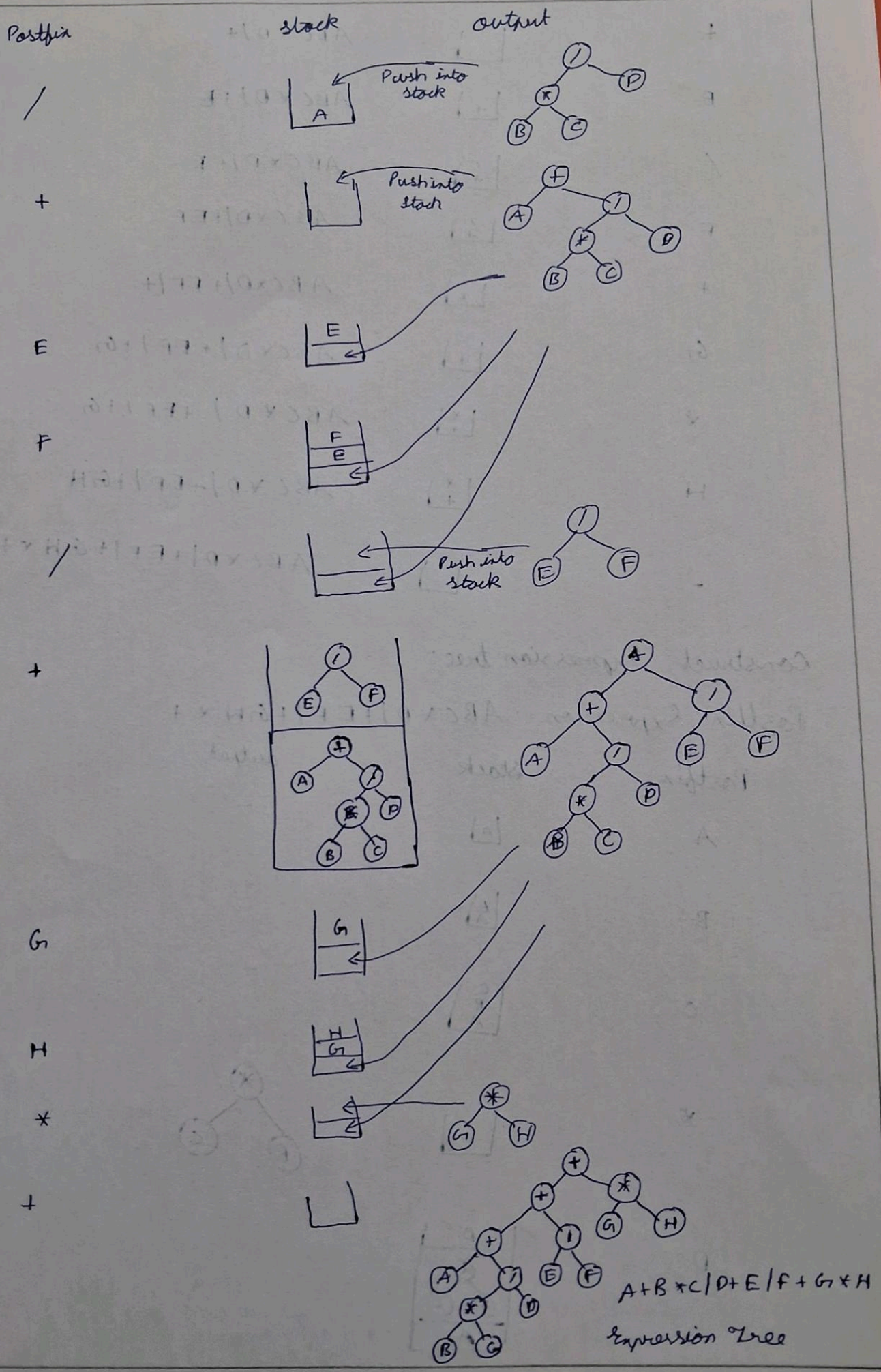
Infix	stack	output
A	⌊ ⌋	A
*	⌊ * ⌋	A
B	⌊ * ⌋	AB
*	⌊ * + ⌋	AB
C	⌊ * + ⌋	ABC
/	⌊ / + ⌋	ABC*
D	⌊ / + ⌋	ABC*D
+	⌊ + ⌋	ABC*D/

+		ABC*D/+
E	+	ABC*D/+E
/	/ +	ABC*D/+E
F	/ +	ABC*D/+EF
+	+	ABC*D/+EF/+
G	+	ABC*D/+EF/+G
*	* +	ABC*D/+EF/+G
H	* +	ABC*D/+EF/+GH
-		ABC*D/+EF/+GH*+

Construct Expression tree:

Postfix Expression : ABC*D/+EF/+GH*+

Postfix	stack	output
A	A	–
B	B A	–
C	C B A	–
*	A	
D	D  A	


Postfix
stack
output
/
Push into stack
+
Push into stack
E
F
/
Push into stack
+
G
H
*
+
A+B*C/D+E/F+G*H
Expression Tree

4) Explain Insertion Operation in red black search trees with suitable Example.

A:- Algorithm:-

1) If the tree is empty, then we create a new node as root node with the color black.
2) If the tree is not empty, then create a new node as a leaf node with a color red.
3) If the parent of a new node is black, then exit.
4) If the parent of a new node is red, then we have to check the color of the parent's sibling of new node.
   a) If the color is black or NIL, then we perform suitable rotation and recolouring.
   b) If the color is red, then we recolor the node. We will also check whether the parents of new node is the root node or not. if it is not a root node, we will recolor and recheck the node.

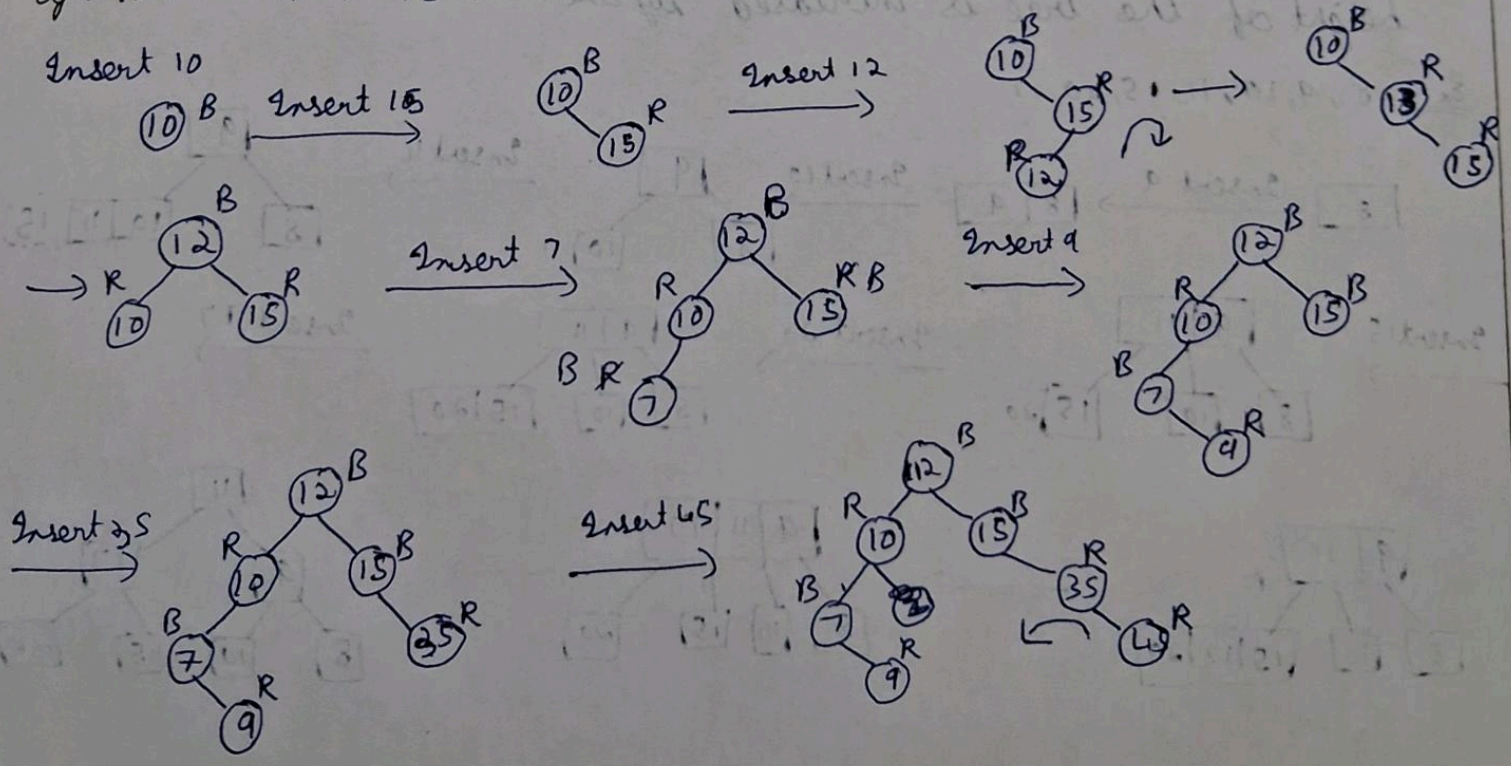
<u>Note:-</u>

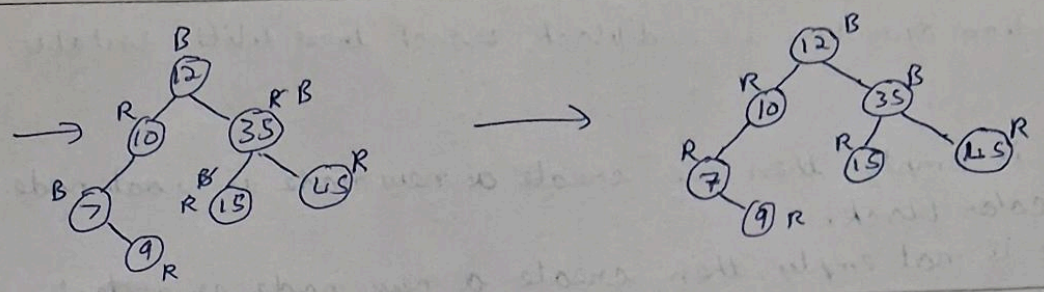
→ Root node is always black

→ No two adjacent red nodes

→ Count no. of black nodes in each path should be same.

Eg: 10, 15, 12, 7, 9, 35, 45, 6

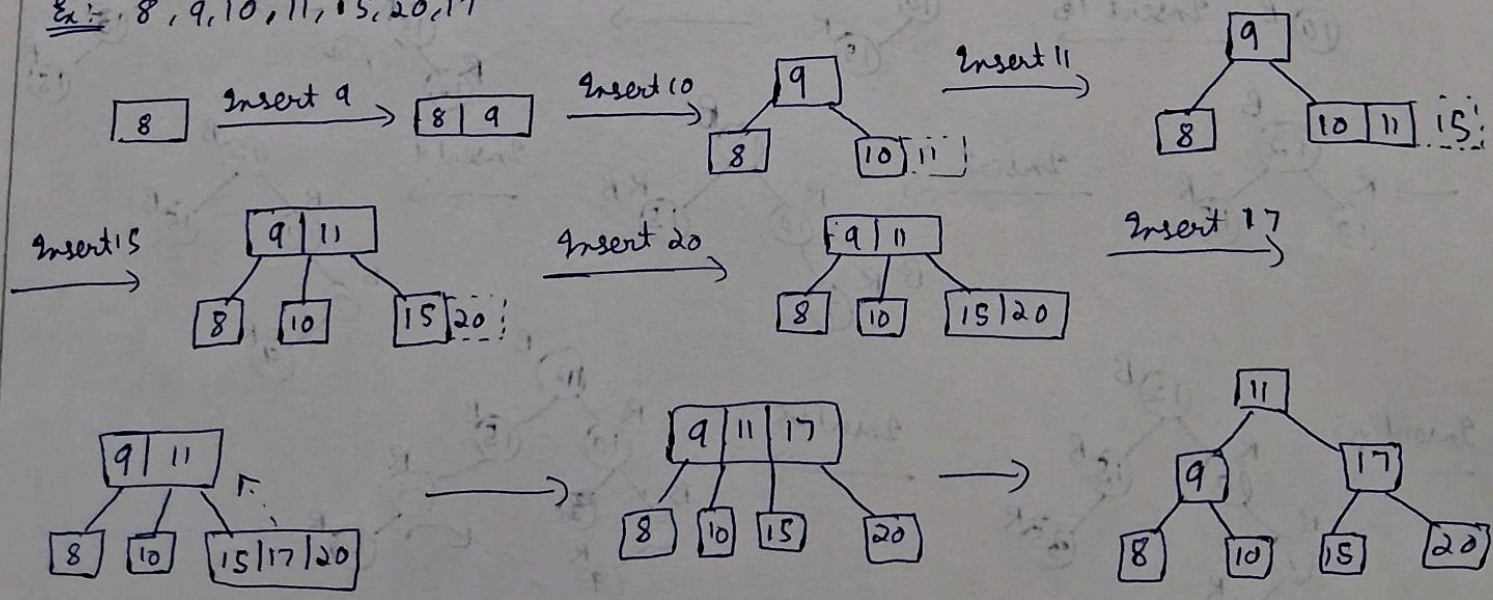




5) Explain insertion operation in B+ tree with suitable example.

1) Check whether tree is empty.
2) If tree is Empty, then create a new node with new key value and insert it into the tree as a root node.
3) If tree is not empty, then find the suitable leaf node to which the new key value is added using Binary Search tree logic
4) If that leaf node has empty position, add the new key value to that leaf node in ascending order of key value within the node.
5) If that leaf node is already full, split that leaf node by sending middle, value to its parent node. Repeat the same until the sending value is fixed onto a node.
6) If the spliting is performed at root node then the middle value becomes new root node for the tree and the height of the tree is increased by one.

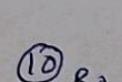
Ex:- 8, 9, 10, 11, 15, 20, 17

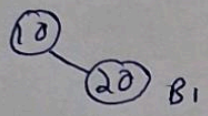


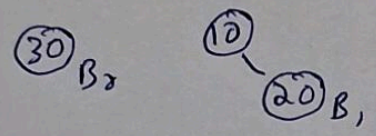
6) Write a short nodes on binomial queues. Perform insert and delete operation on binomial queues by taking an example.

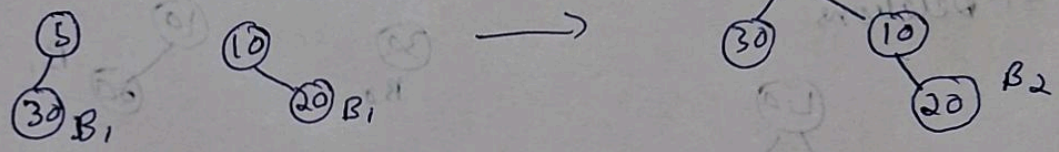
A binomial queue is a priority queue implemented as a collection of binomial heaps. The key at the root of each binomial tree is smaller than or equal to the keys of the children. It works on the basis of min heap. There is at most one binomial tree of any degree in the binomial heap. Binomial queues are efficient for merge operations.

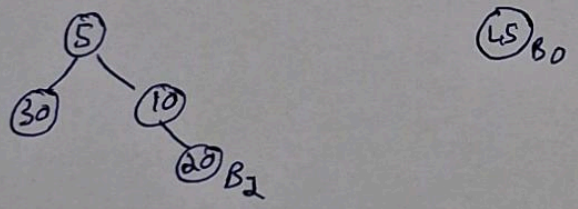
Example: 10, 20, 30, 5, 45, 40, 50, 65

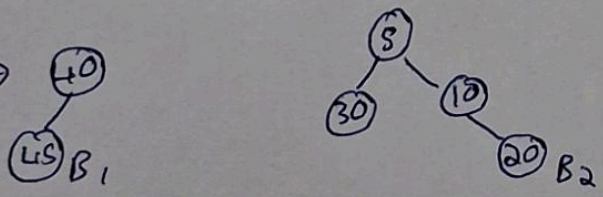
Insert 10 → 

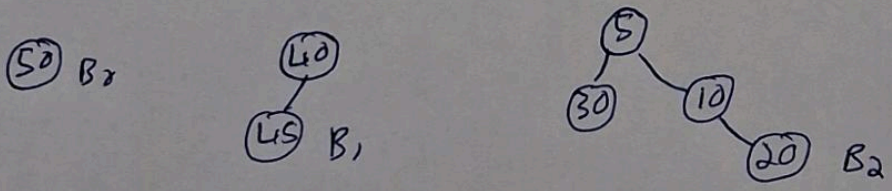
insert 20 → 

insert 30 → 

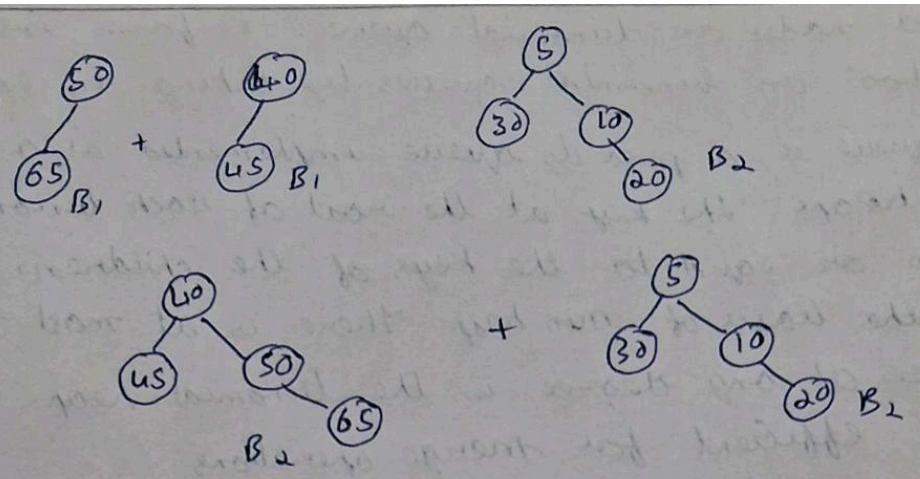
insert 5 → 

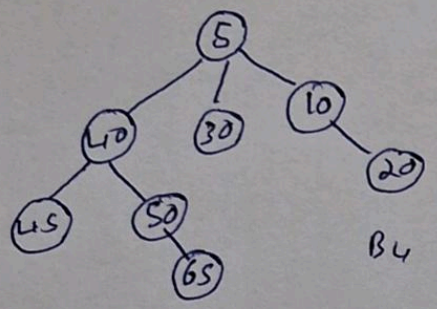
insert 45 → 

insert 40 → 

insert 50 → 

Insert 65 →





=> Deletions