# Recursion

Some programming languages allow to call a module (function) itself is called recursion.

* Base case - There must be at least 1 base criteria or condition such that when this condition is met, the $f^n$ stops calling itself.

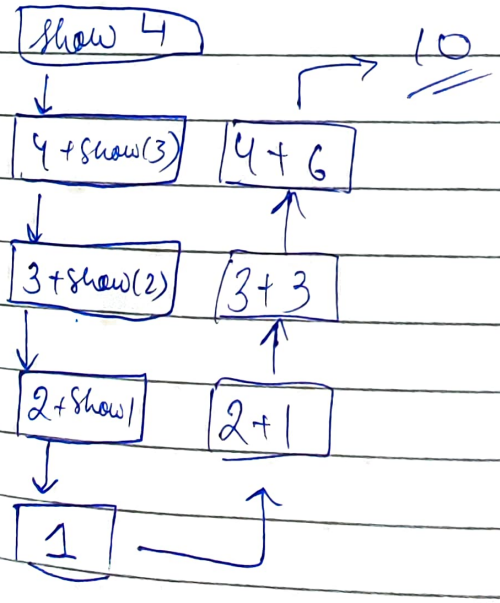
* Recursive calls - The recursive call should progress in such a way that each time a recursive call is made it comes closer to the base case.

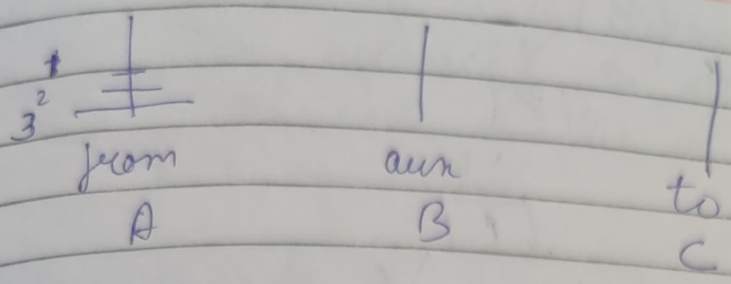
---

Q1 Recursive $f^n$ to print number from n to 1

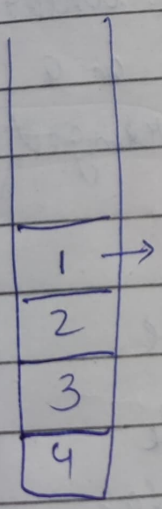
Q2 WAP to add n numbers using recursion

let n = 4



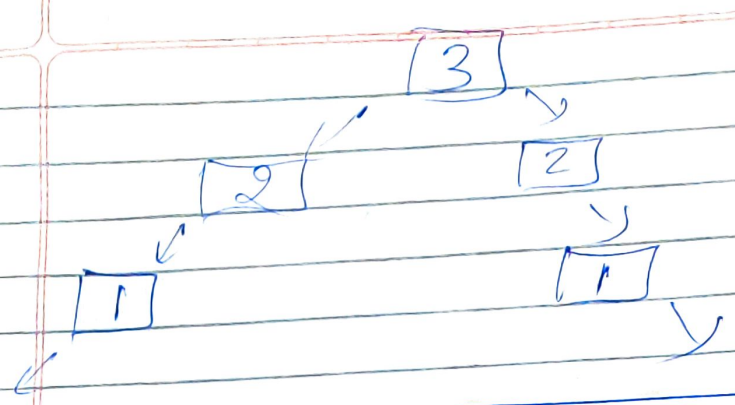



Move disk 1 from A to C
" " 2 from A to B
" " 1 from C to B
" " 3 from A to C
" " 1 from B to A
" " 2 from B to C
" " 1 from A to C




| disk ~~rod~~ | from | to | Aux |
|---|---|---|---|
| 3 | A | C | B |
| 2 | A | B | C |

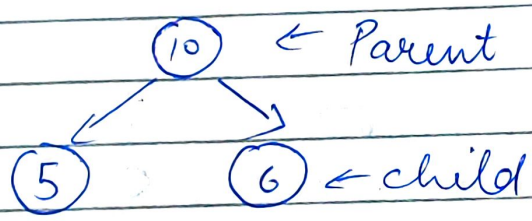
1 → A to C
2 → A to B
1 → C → to B.
3 →



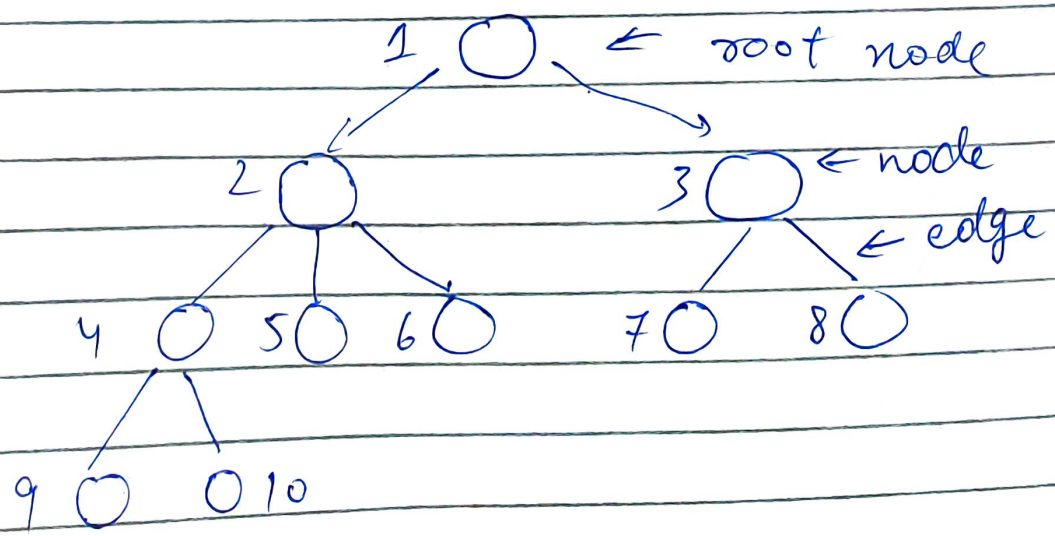

25/11/21

# Trees

Tree is a hierarichial data structure which stores data in parent-child format.




A tree data structure is a non linear data structure because its stores data in a non sequential manner. Trees are arranged in multiple levels



Date: / / Page no:

1. Root → Root node is the Topmost node in the tree hierarchy. In other words the root node is the one that does not have any parent.

2. Child node → If the node is a descendant of any node then the node is k/as a child node
   eg → 9, 10 are child nodes of 4.
   → 2, 3 are child nodes of 1.

3. Parent node → If the node contains any sub-node then that node is said to be the parent of that sub-node

4. Siblings → The node that have the same parents are k/as siblings (4, 5, 6)

5. Leaf node → The node of the tree which does not have any child node is called a leaf node.
   Leaf nodes can also be called as external nodes.
   eg 5, 6, 7, 8, 9, 10.

6. Internal node → a node has at least one child node is k/as an internal node.
   eg → 2, 3, 4.

7 Ancestor nodes - an ancestor of a node is any predecessor node on a path from the root to that node

eg -> 2, 4 are ancestor node for 10

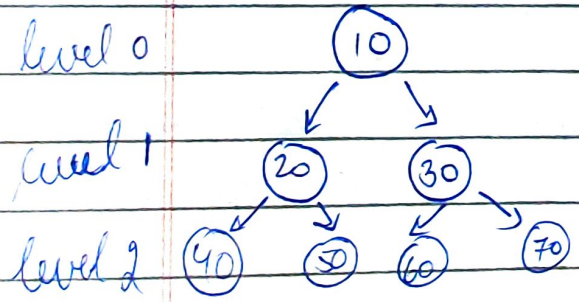
8 Depth of node 'x' -> the depth of node 'x' can be defined as the length of the path from the root to the node 'x'. one edge contributes one unit length in the path.

eg -> depth of 4 is 2.

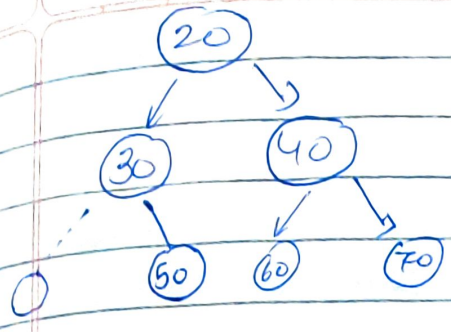
9 Height of node 'x' -> The height of node x can be defined as the longest path from the node x to the leaf node

## Representation of tree in Memory

-> Binary tree -> a tree in which any node can have at max two child node are k/as binary trees.




| 10 | 20 | 30 | 40 | 50 | 60 | 70 |
|---|---|---|---|---|---|---|



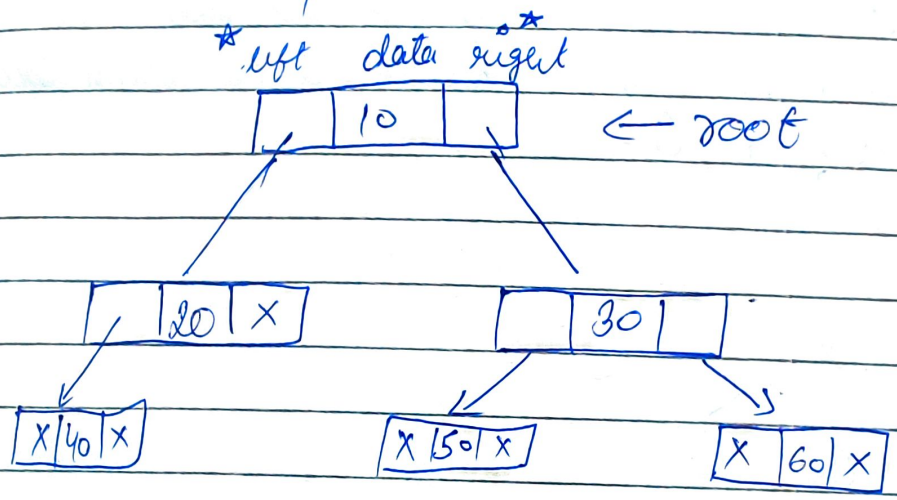

| 20 | 30 | 40 | -1 | 50 | 60 | 70 |
|---|---|---|---|---|---|---|

↑

pre-decided

such element which is not part

## Linked Representation.




```
typedef struct treenode
{
    struct treenode * left;
    int data;
    struct treenode * right;
} node;

node root;
```

# Types of Binary Trees -

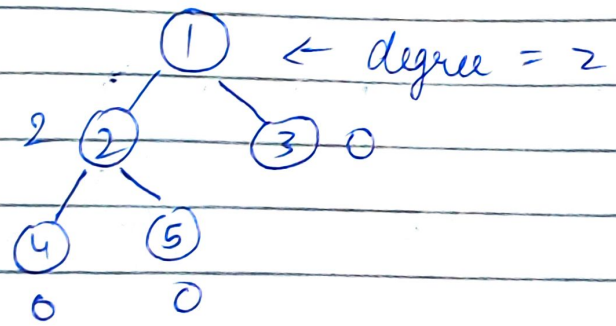
There are 4 types of Binary trees -

## 1 Full / Proper / Strict Binary tree.

→ The full binary tree is a tree in which each node must contain either 0 or two ~~children~~ children.

→ It can also be defined as the tree in which each node must contain 2 children except the leaf node

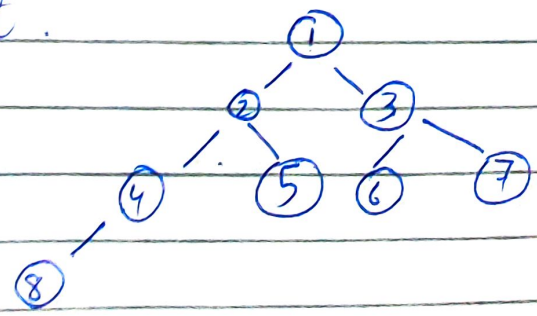
eg




## 2 Complete Binary Tree →

It is a tree in which all the nodes are completely filled except the last level. In the last level all the nodes must be as left as possible. In a complete binary tree the nodes should be added from the left.

eg →



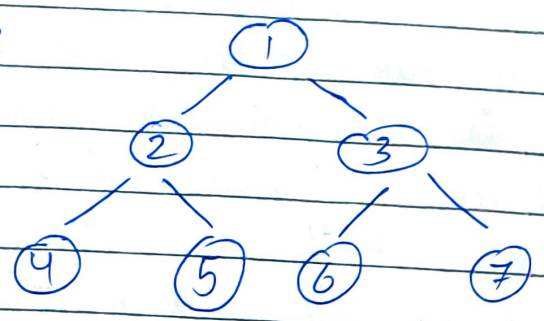
26/11/21

Date: Page no:

## 3 Perfect Binary Tree –

In the Perfect binary tree all the internal nodes have 2 children & all the leaf nodes are at the same level

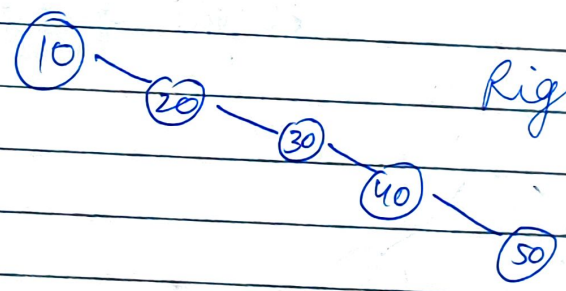
eg




## 4 Degenerated Binary Trees →

In these, all the internal node have only one child.

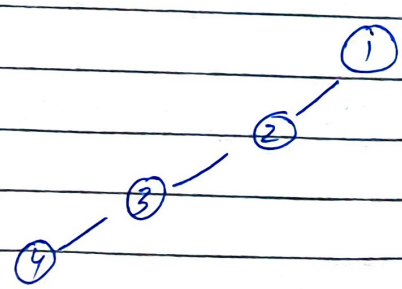
eg →




Right skewed Tree
skewed

the above degenerated tree is also k/as right skewed tree as all the nodes have a right child only

Left skewed →




all the ~~child nodes~~ ~~are~~ nodes are having left child only

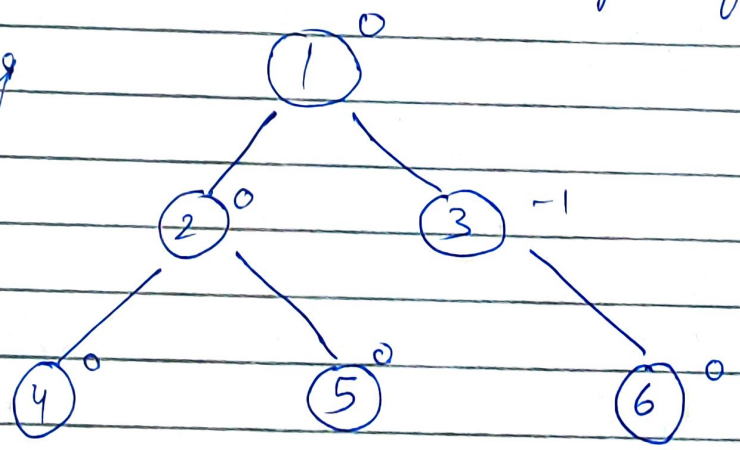
# 5 Balanced Binary Tree -

It is a tree in which both the left and right tree differ by at most 1.

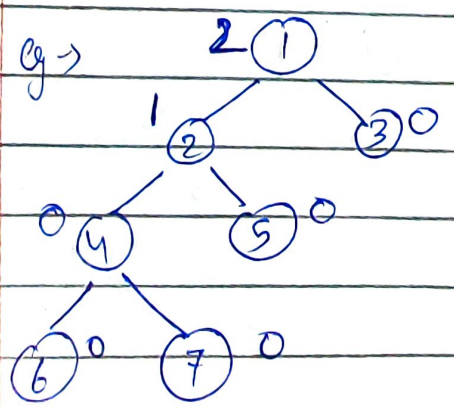
To find out whether a Tree is a balanced tree a balanced tree or not, we find the balance factor of all the nodes of tree. If the balance factor is -1, 0, +1 then the trees are considered as balanced trees. If the balance factor is < -1 or > +1 then the tree is considered to be unbalanced binary tree.

Balance factor = Height of left subtree
− Height of right subtree.

eg




eg →



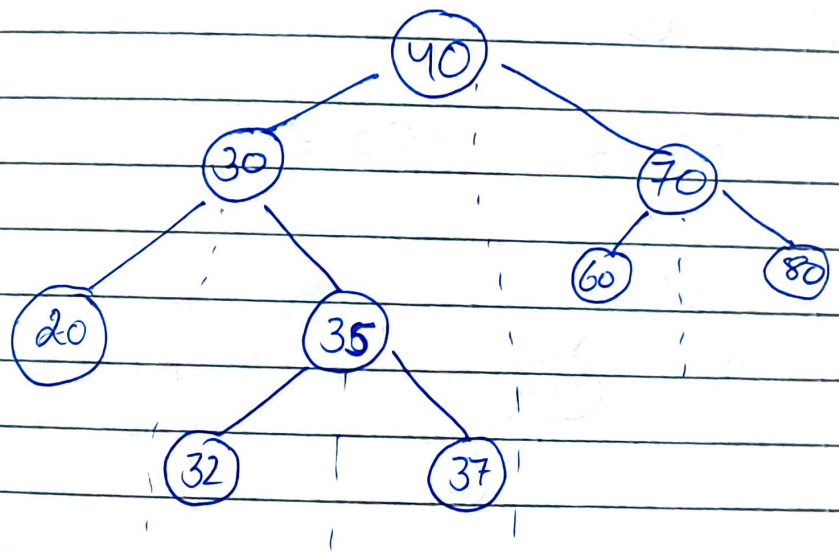

Unbalanced

# Binary Search Tree -

→ It is also called as ordered binary Tree as the nodes are arranged in a specific order.

→ In a BST the value of all the nodes in the left sub tree is less than the value of root node

→ Similarly value of all the nodes in right sub tree is greater than or equal to the value of the root node



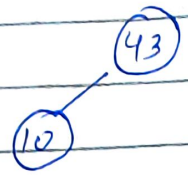

Q Create the binary search tree using the foll. data elements -

43, 10, 79, 90, 12, 54, 11, 9, 50

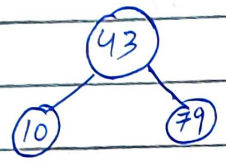
43, 10, 79, 90, 12, 54, 11, 9, 50

Step 1: Insert 43

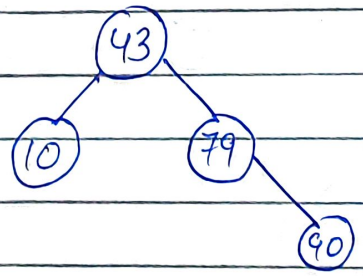
Step 2: Insert 10  10 < 43  left



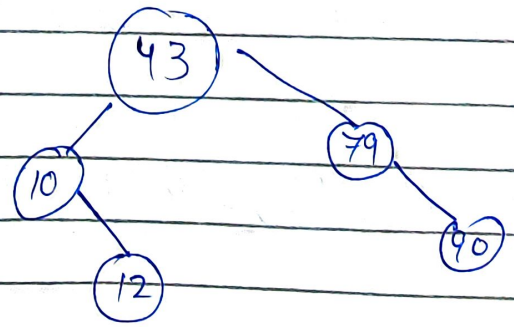

Step 3: Insert 79  79 > 43  right




Step 4: Insert 90  90 > 43 , 90 > 79

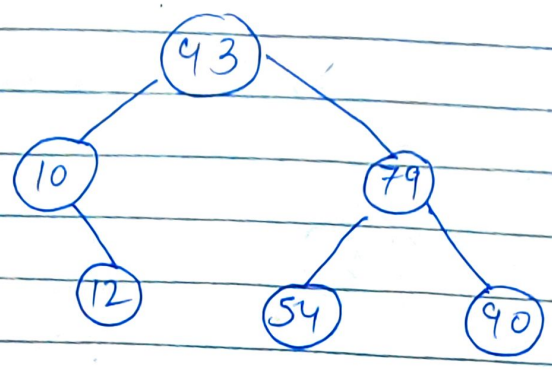



Step 5: Insert 12  12 < 43  12 > 10



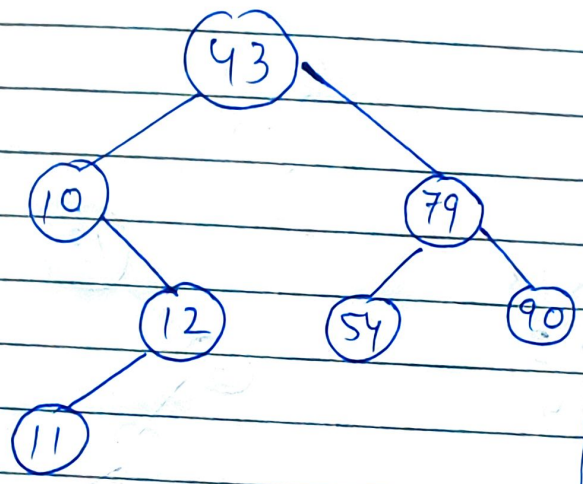
Step 6 Insert 54

54 > 40
54 < 79



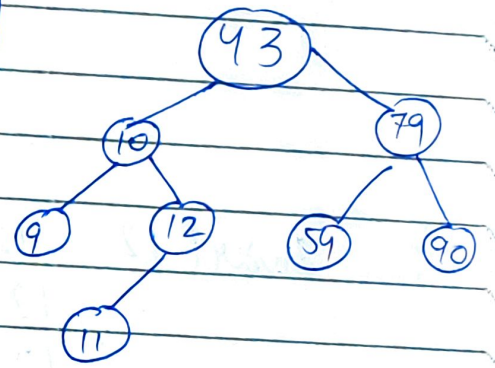

Step 7 Insert 11

11 < 43
11 > 10
11 < 12



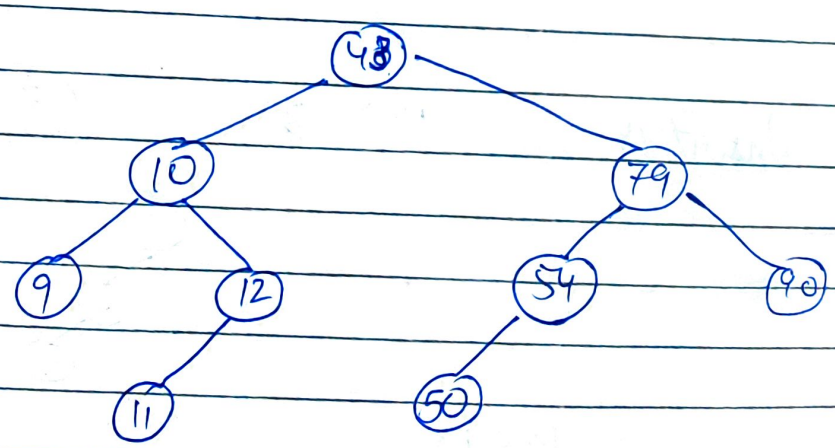

Step 8 insert 9

9 < 43
9 < 10




Step 9 insert 50

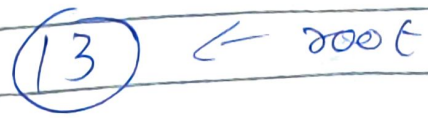
50 > 43
50 < 79
50 < 54



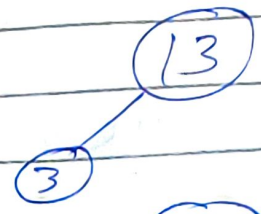
13, 3, 4, 12, 14,

13, 3, 4, 12, 14, 10, 5, 1, 8, 2, 7, 9, 11, 6, 18

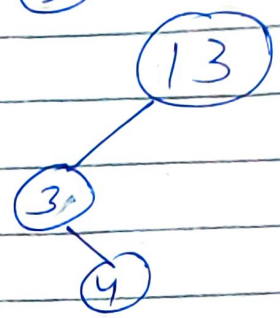
Step 1 Insert 13



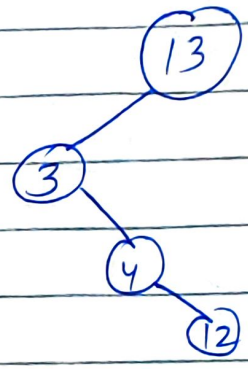

Step 2 Insert 3, 3<13



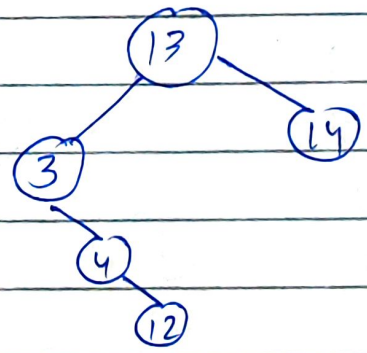

Step 3 Insert 4, 4<13
4>3



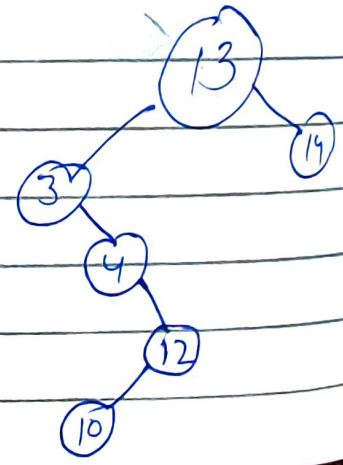

☆ Insert 12 12<13
12>3
12>4




☆ Insert 14 14>13



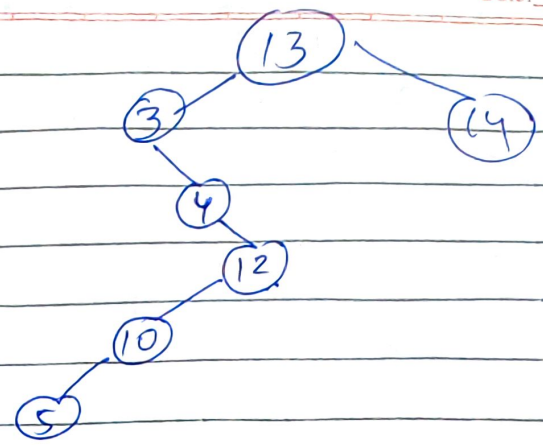

☆ Insert 10 10<13
10>3
10>4
10<12



Date: / / Page no:

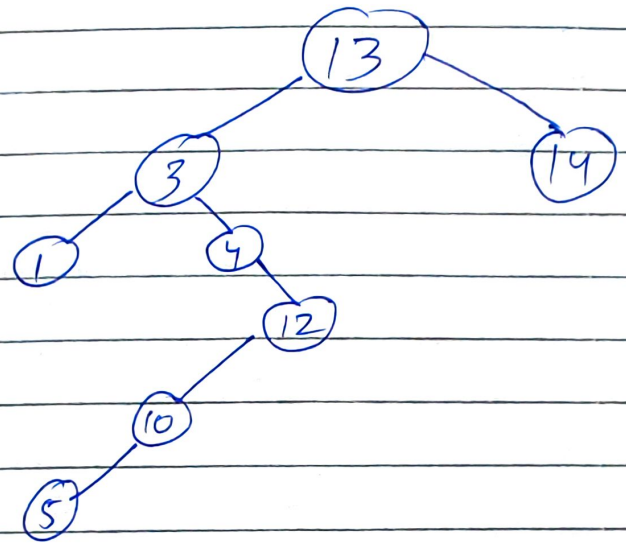
* Insert 5

5<13
5>3
5>4
5<12
5<10



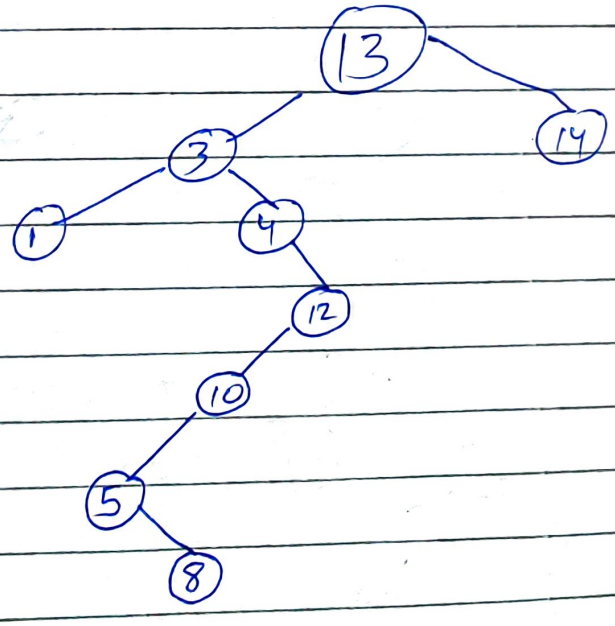

* Insert 1

1<13
1<3




* Insert 8

8<13
8>3
8>4
8<12
8 ~~<~~ 10
8>5



2, 7, 9, 11, 6, 18

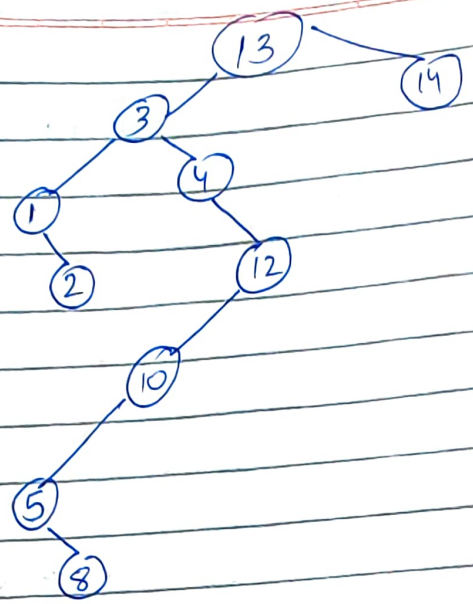
Date: / / Page no:

★ Insert 2

2<13

2<3

2>1




★ Insert 7

7<13

7>3

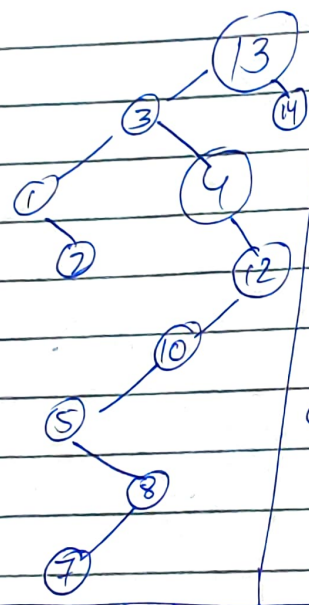
7>4

7<12

7<10

7>5

7<8




★ Insert 9

9<13

9>3

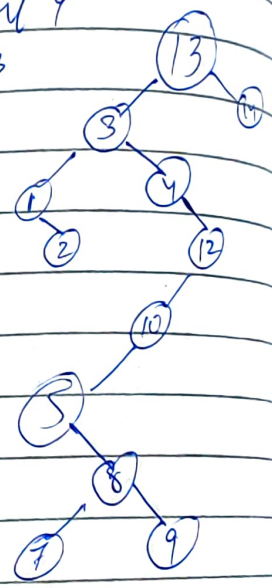
9>4

9<12

9<10

9>5

9>8




★ Insert 11

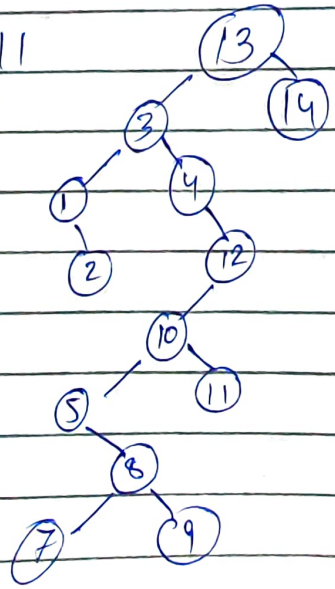
11<13

11>3

11>4

11<12

11>10




★ Insert 6

6<13

6>3

6>4

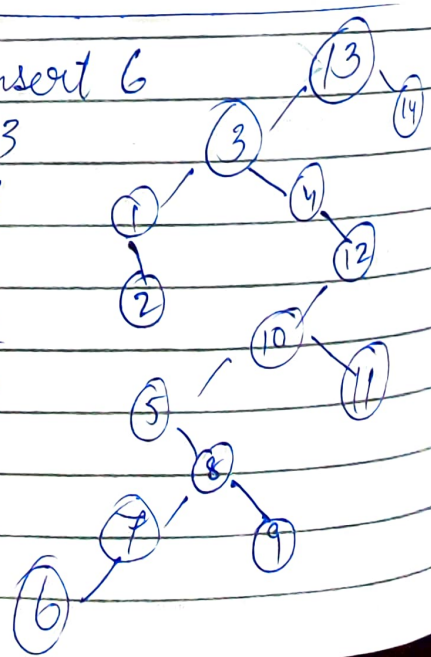
6<12

6<10

6>5

6<8

6<7

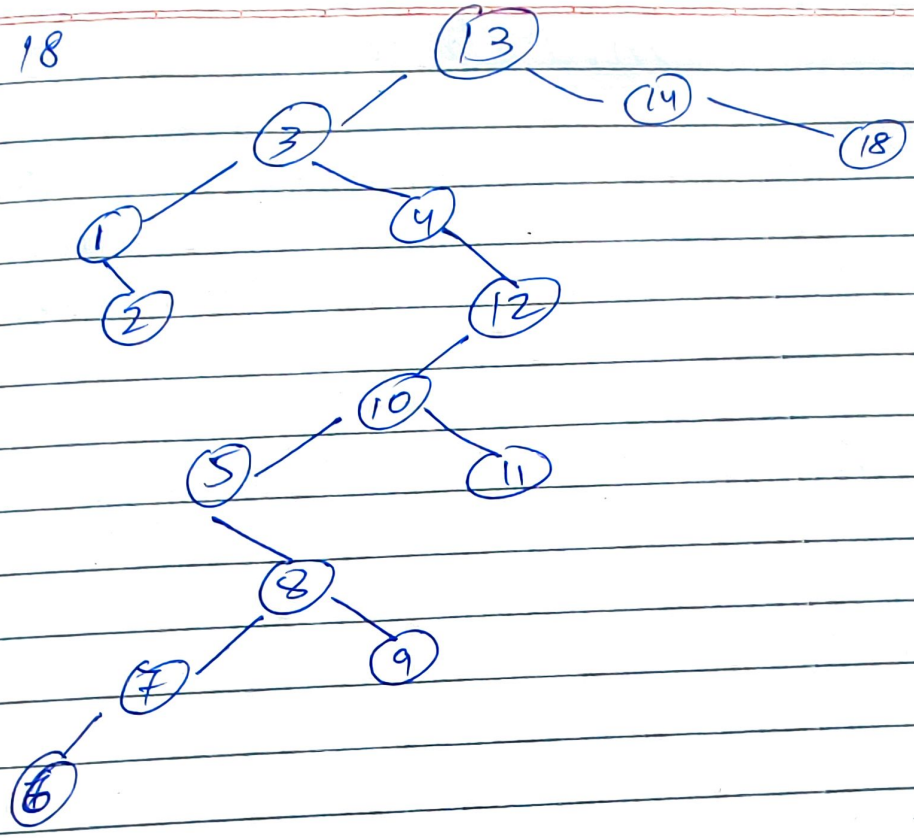


Date: / / Page no:

Insert 18

18>13

18>14



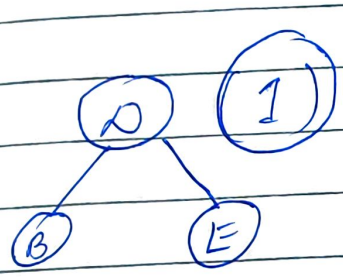
Date: / / Page no:

# Traversal of Binary Search Tree or Binary Tree

Three types of Traversal →

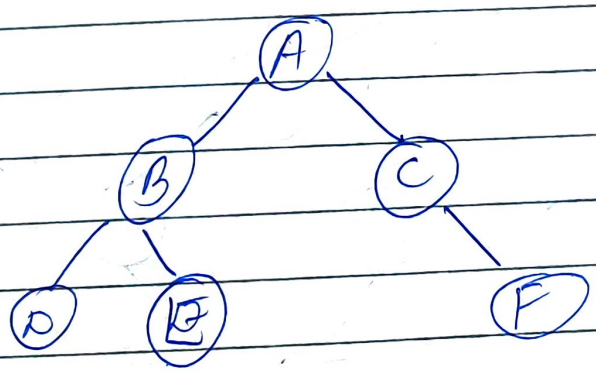
1. Preorder (NLR) — N is Root node
2. Inorder (LNR) — L is Left
3. Post order (LRN) — R is Right.

Q1




Pre - D B E
In - B D E
Post - B E D

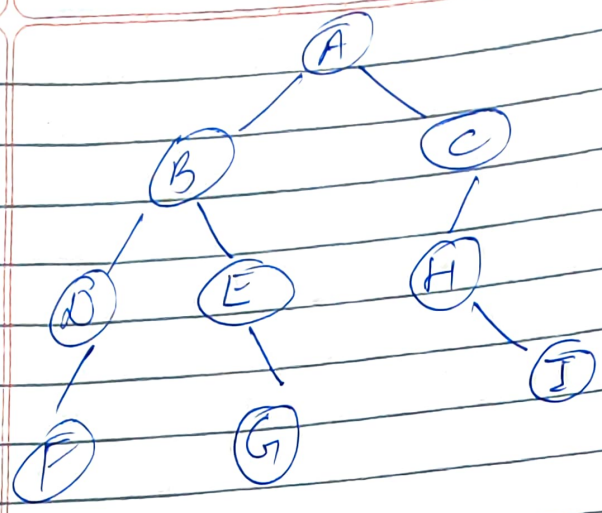
Q2




Pre - A B D E C F
In - D B E A C F
Post - D E B F C A

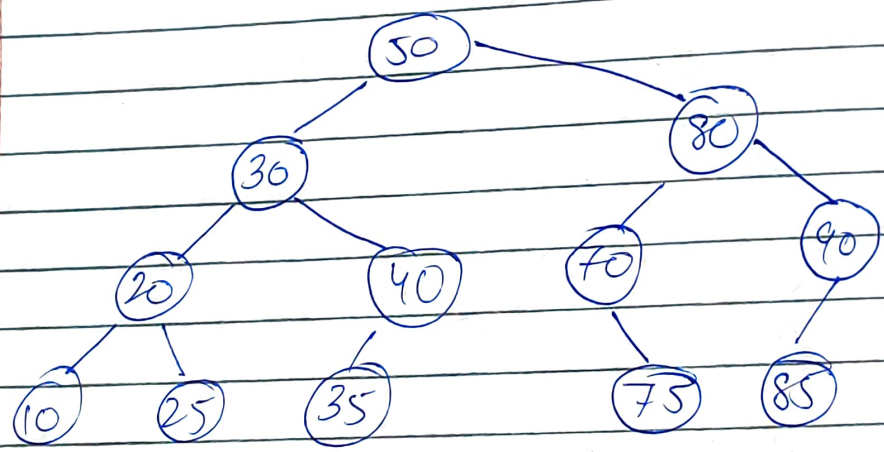
Date: / / Page no:

Q3




NLR Pre -> A B D F E G C H I
LNR In -> F D B E G A H I C
LRN POST -> F D G E B I H C A

Q4

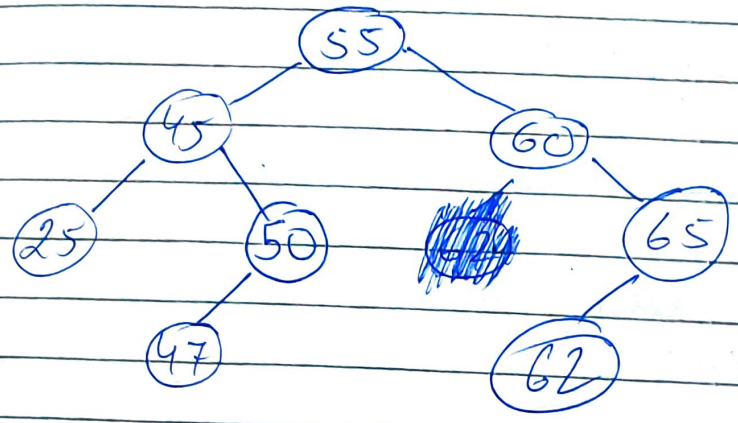



NLR PRE -> 50, 30, 20, 10, 25, 40, 35, 80, 70, 75, 90, 85
LNR In -> 10, 20, 25, 30, 35, 40, 50, 70, 75, 80, 85, 90
LRN POST -> 10, 25, 20, 35, 40, 30, 75, 70, 85, 90, 80, 50

Q5

PRE -> 55 45 25 50 47 60 65 62

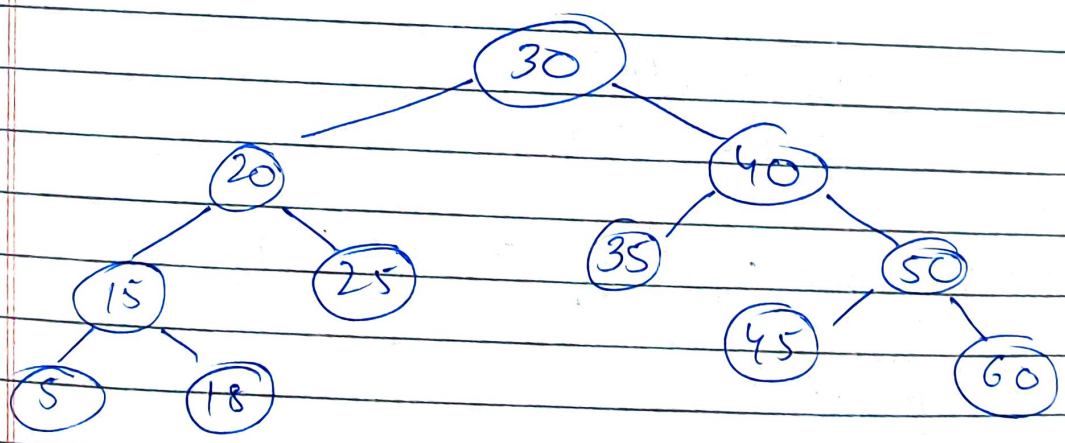
IN -> 25 45 47 50 55 60 62 65




Q6

In 5, 15, 18, 20, 25, 30, 35, 40, 45, 50, 60

POST 5, 18, 15, 25, 20, 35, 45, 60, 50, 40, 30.



Date: / / Page no:

# Application of Traversing

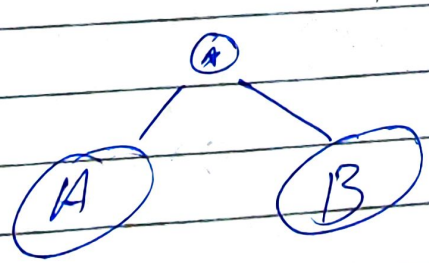
1. Conversion of an infix expression into Postfix notation of prefix notation can be done through binary tree Traversal.

we know that arthmatic operators are binary operators which means a binary tree can represent an algebraic expression.

## Precedance of operators

1. all expressions in Parenthesis are to be evaluated first
2. Exponential will come next **
3. divide & multiplication / *
4. + & -

A * B

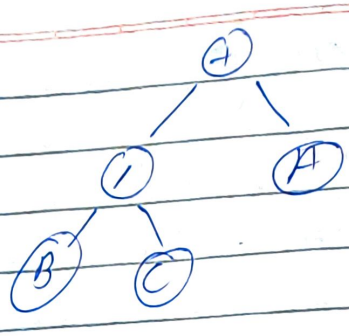



Inorder -> A * B  LNR
Prefix -> * AB  NLR
Postfix -> AB *  LRN

6/12/21

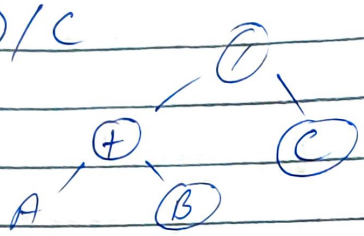
Date: / / Page no:

Inorder -> A+B/C




Pre -> +/BCA
Post -> BC/A+

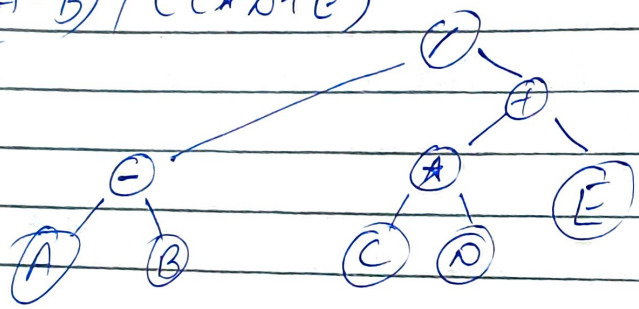
(A+B)/C




Pre -> /+ABC
Post -> AB+C/

---

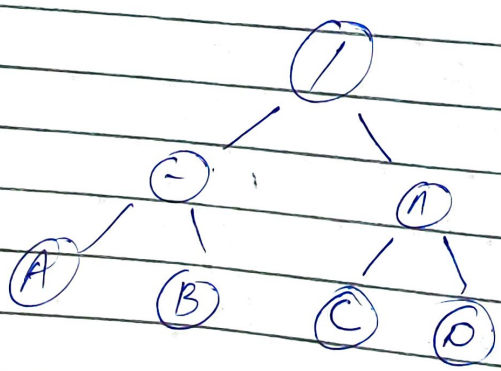
(A-B)/(C*D+E)




Pre -> /-AB+*CDE
Post -> AB-CD*E+/

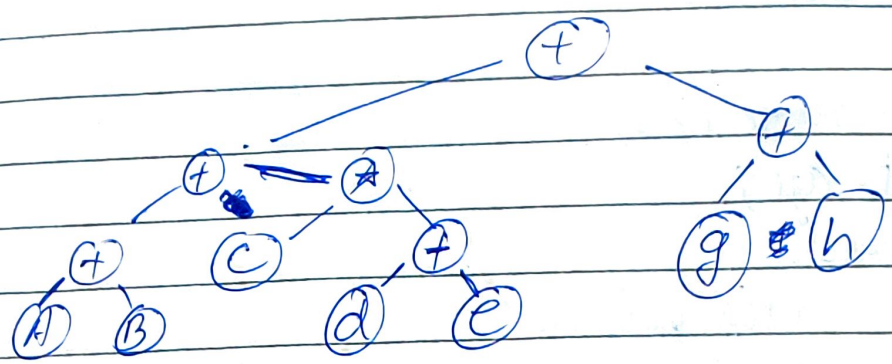
---

(A-B)/(C^D)




Pre -> /-AB^CD
Post -> AB-CD^/

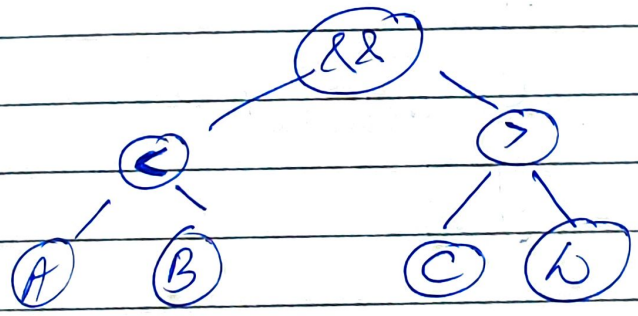
$((A+B)+(C*(d+e))+(g+h))$




Pre -> + + + AB * C + de + gh

Post -> AB + C de + * + gh + +

d $(A<B) \&\& (C>D)$



8/12/21

Date: / / Page no:

# BS Tree Code

```
typedef struct B_node
{
  struct B_node *left;
   int data;
   struct B_node *right;
} node;

node *root, *temp , *p, *q;

int main()
{
   int n;
   printf("Enter num of nodes in tree ");
   scanf("%d", &n);
   while(n)
   {
      temp = (node *) malloc (sizeof(node));
      printf("Enter new element: ");
      scanf("%d", &temp->data);
      temp->right = Null;
      temp->left = Null;

      if (root == Null) root = temp;
      else
       {
         p = root;
         while (p != Null)
         {
            if (p->data > temp->data)
```

```
{ q = p;
  p = p -> left;
}
else
{
  q = p;
  p = p -> right;
}
}
if (q -> data > temp -> data) q -> left = temp;
else q -> right = temp;
}
n--;
}
return 0;
}
```

9/12/21

```
void search()
{
int num;
if (root == Null) printf(" Tree is empty ");
else
{ printf(" Enter element to search = ");
  scanf("%d", &num);
  p = root;
  while (p != Null)
  { if (p -> data == num)
      { printf(" element found ");
        break;
      }
    else if (p -> data > num) p = p -> left;
```

```
        else p = p -> right
    }
    if (p == Null) printf("Not found");
  }
}
```

Traversals

```
void Preorder (node *root)
{
    if (root == Null) return;

    printf("%d ", root -> data);        NLR
    preorder (root -> left);
    preorder (root -> right);
}

void Inorder (node * root)
{
    If (root == Null) return;           LNR

    Inorder (root -> left);
    printf("%d ", root -> data);
    Inorder (root -> right);
}

void Postorder (node *root)             LRN
{   If (root == Null) return;
    Postorder (root -> left);
    Postorder (root -> right);
    Printf("%d ", root -> data);
}
```

DSA

Date: / / Page no:

# Assignment 3

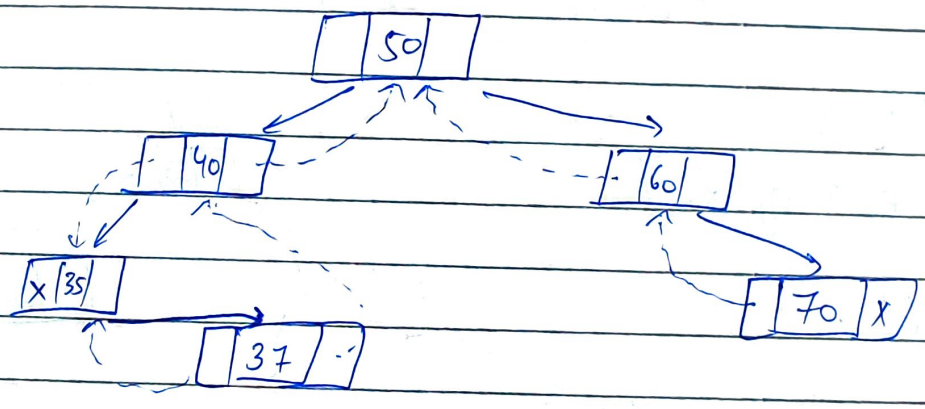
1. ~~WAP~~ Write an algo to delete nodes from binary search tree

2. Explain the Representation of graphs in memory adjacency matrix

3. Explain BFS (Breadth first search) & DFS (depth first search) methods in graphs.

---

## Threadded Binary Tree




right null pointers represent successors
left " " " predecessor

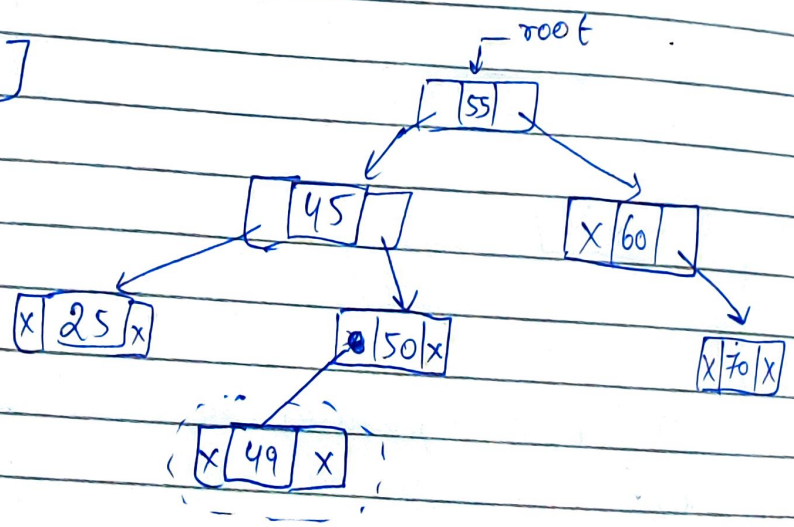
13/12/21

Date: / / Page no:

# Deletion In Binary Search Tree

Case :- 1 If the node 'X' To be deleted has no child.

1 Begin
2 If x -> left Null & X -> Right = Null then
3 if Parent -> Right = 'x' then
set Parent -> right = Null
else
set Parent -> left = Null
[end of if else]
4 delete X
[end of if]
5 Exit.




Case 2 If the node X to be deleted has only Right Child

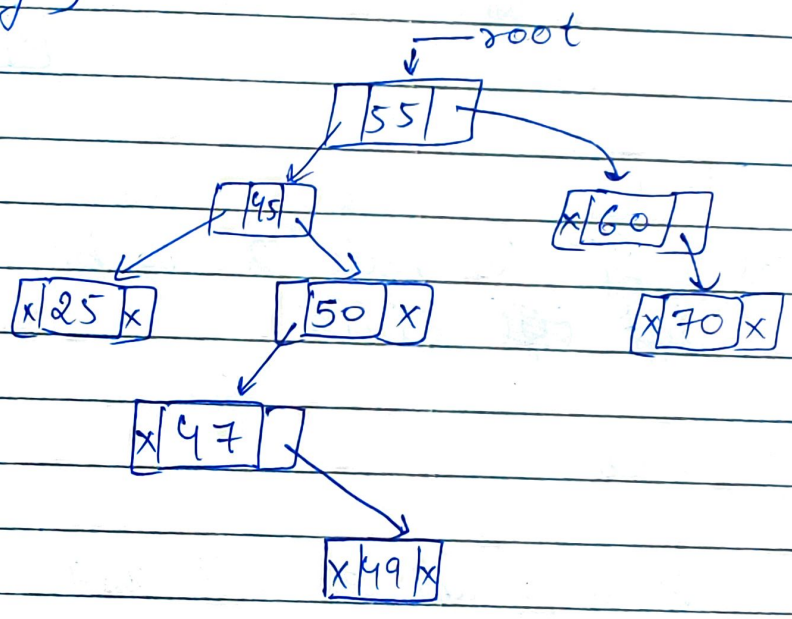
1 Begin

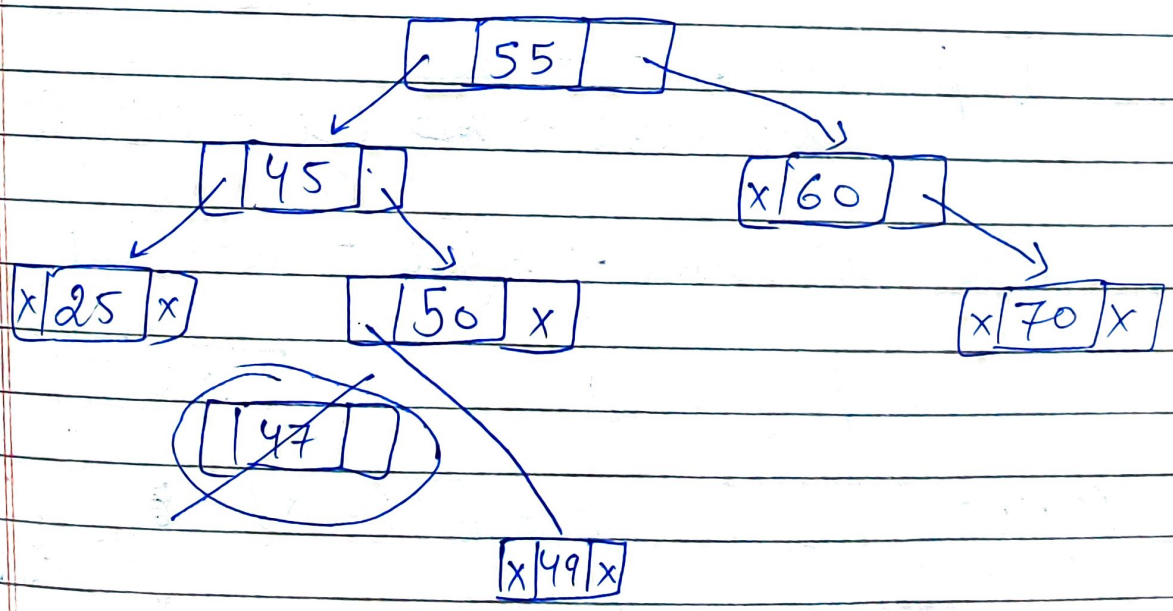
2 If X -> left = Null and X -> Right ≠ Null then

3. if parent -> left = X then
set parent -> left = X -> Right
else set parent -> Right = X -> Right
[End of if else]

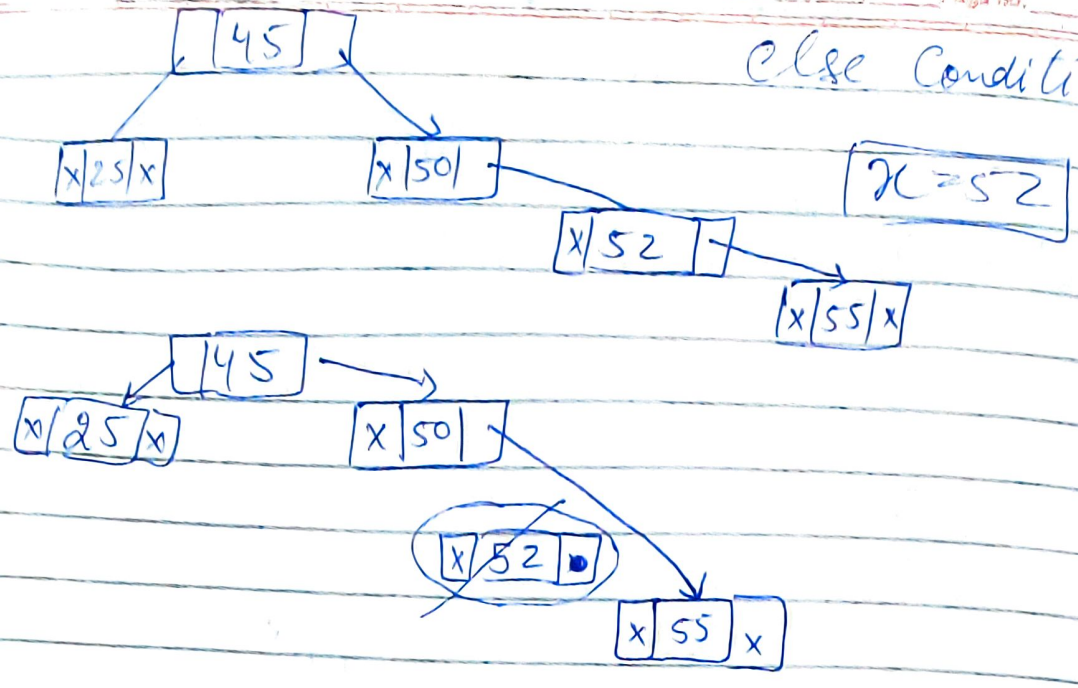
4 Delete X
[End of if]

5 Exit



Else Condition




Case 3 If the node x to be deleted has only left ~~left~~ Child

1 Begin

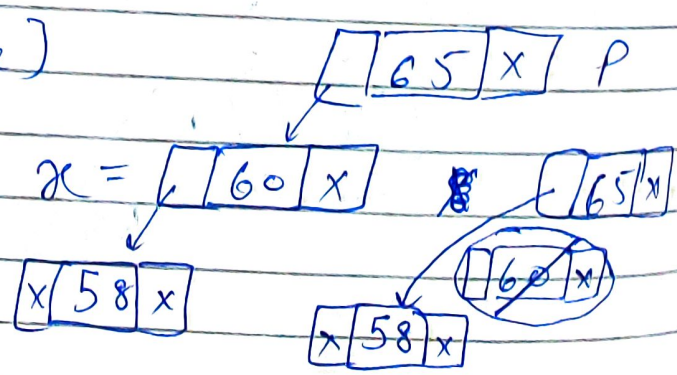
2 If x -> left ≠ Null and x -> right = Null Then

3 If parent -> left = x then
set parent -> left = x -> left

Else set parent -> right = x -> left

[end of if else]

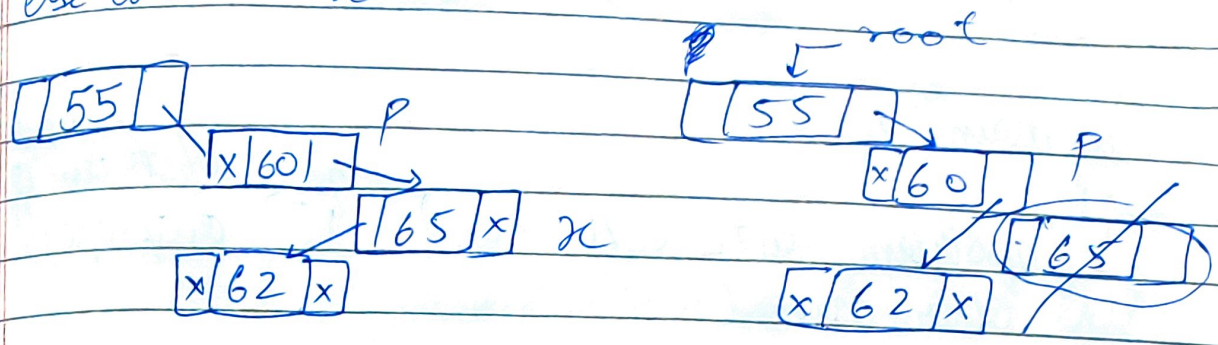
4 Delete x
[end of if]




5 Exit

Date: __/__/__ Page no: ____

else cond$^n$ $x = 65$




Case 4 If the node X to be deleted has both childs.

1 Begin
2 If $x \to$ left $\neq$ Null and $x \to$ right $\neq$ Null then
3 set parent $= x$
4. set x succ $= x \to$ Right
5 Repeat while x succ $\to$ left $\neq$ Null
   a) set Parent $=$ x succ
   b) set x succ $=$ x succ $\to$ left
   [End of while loop]
6 set $x \to$ data $=$ x succ $\to$ data
7. set $x =$ x succ.
   [End of if]
8 End

delete X

Date: / / Page no:

# M-WAy Tree (Search tree)

## B-Tree →

A B-tree is a balanced M-Way tree. B-Tree of order M has following properties -

1. It should be perfectly balanced, i.e., every leaf node is at same level.

2. Each node (except the root) contains minimum M/2 children and maximum M children

3. Each node has one fewer value (or key) than children. No node can contain more than M-1 values

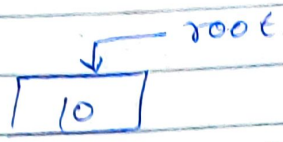
4. Keys are arranged in a defined order within the node.

5. When a new key is to be inserted into a full node, then it split into two nodes and the key with the median value is inserted in the parent node. In case, the parent node is the root, a new node is created.

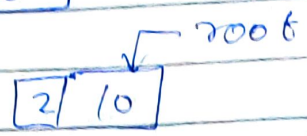
Eg Draw a 3-WAY B-Tree by inserting the foll. data:

10, 2, 61, 25, 18, 16, 35, 30

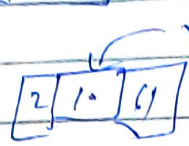
Step 1 Insert 10



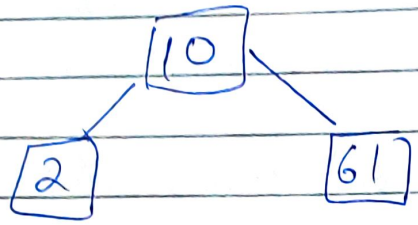

Step 2 Insert 2



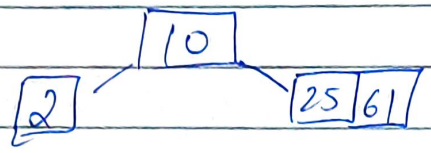

Step 3 Insert 61 ->



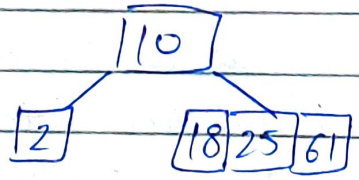

M keys -> split in root node



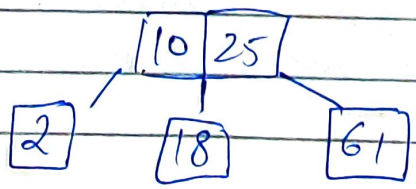

Step 4 Insert 25




Step 5 Insert 18

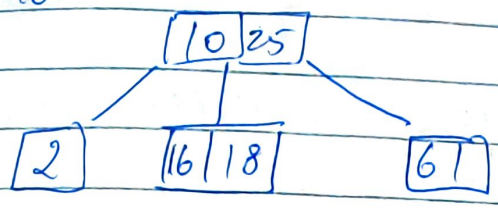



M keys in a node -> split.

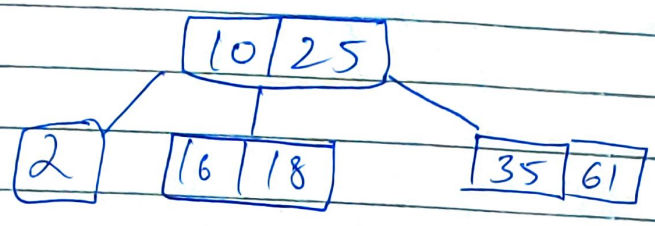


Date: / / Page no:

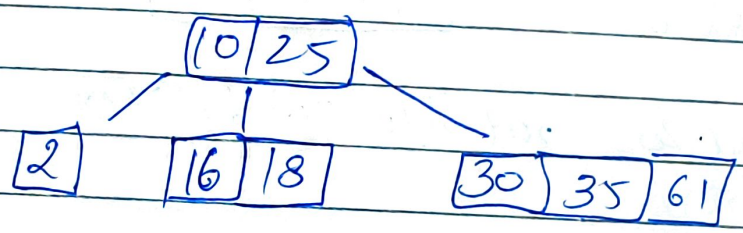
Step 6: Insert 16



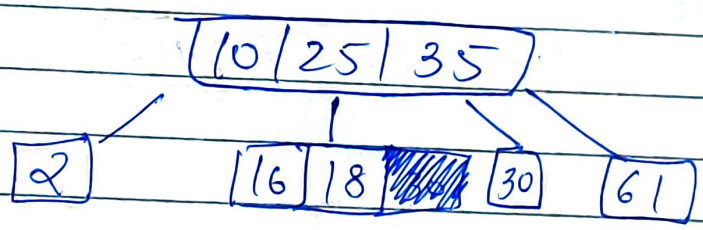

Step 7 Insert 35



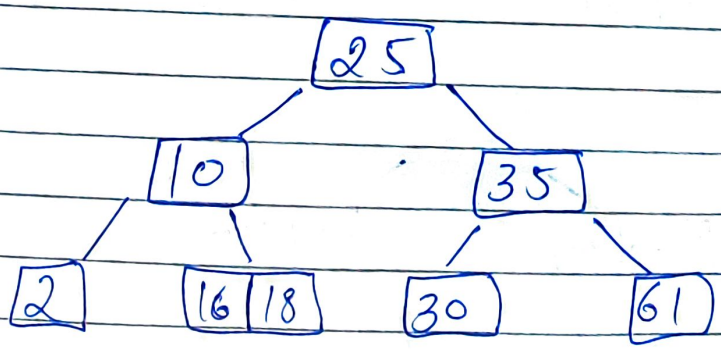

Step 8 Insert 30




M nodes




root has 3 (M) keys. now split

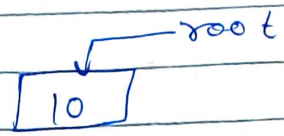


Date: / / Page no:

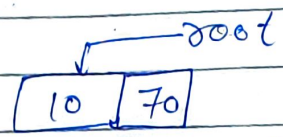
Q Create a 5-way tree using foll. data

10, 70, 60, 20, 110, 40, 80, 130, 100, 50, 190, 90, 180, 240, 30, 120, 140, 160.

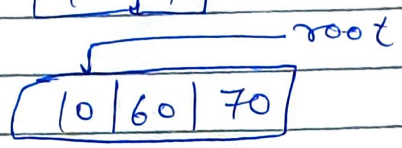
1 Insert 10




2 Insert 70

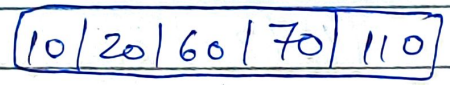



3 Insert 60

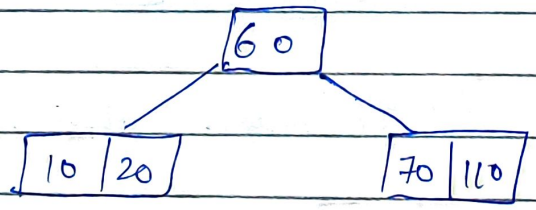



4 Insert 20

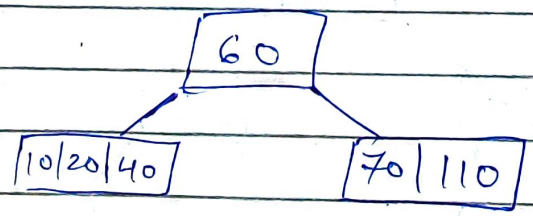
5 Insert 110



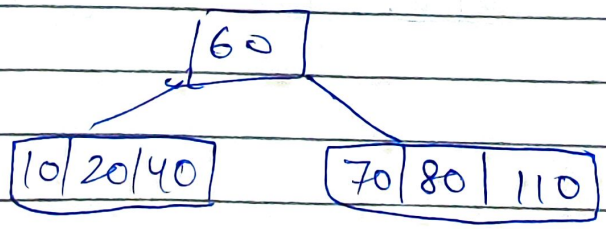

M=5 ~~root~~ Split



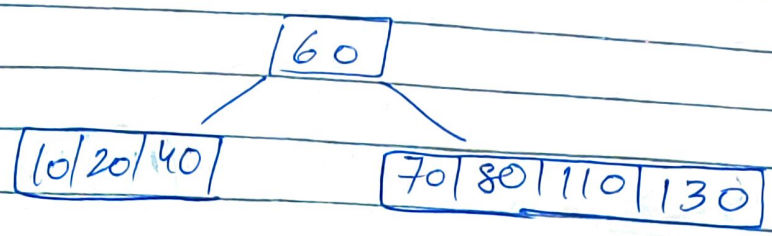

6 Insert 40



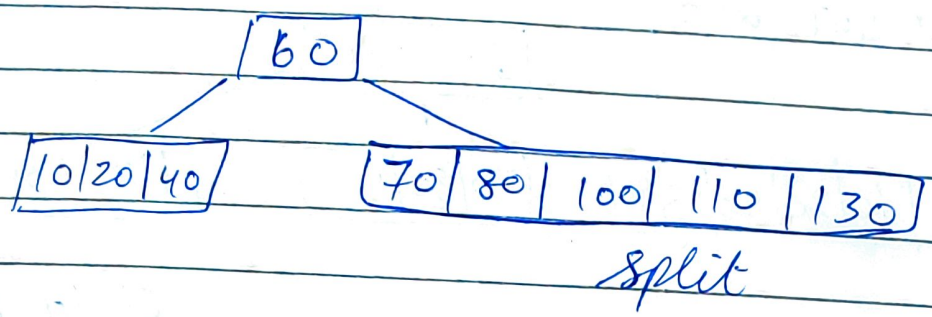

7 Insert 80



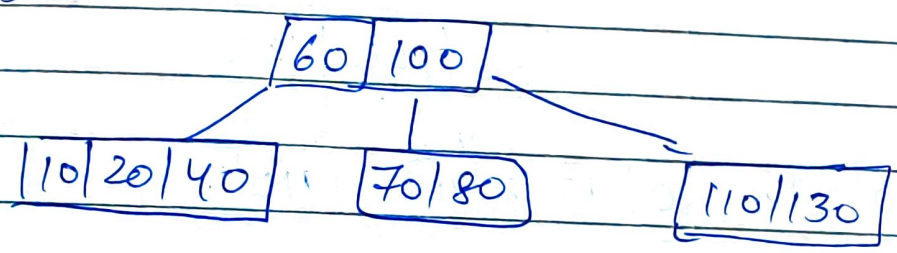
8 Insert 130



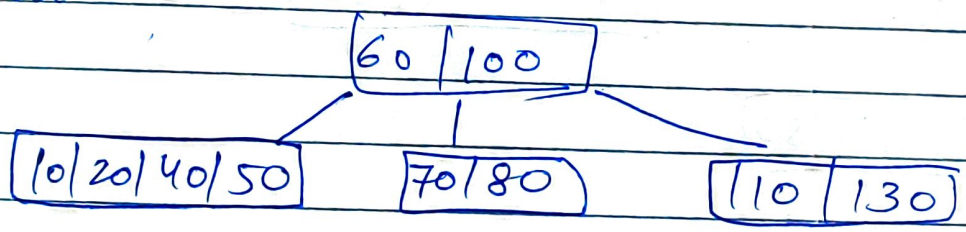

9 Insert 100



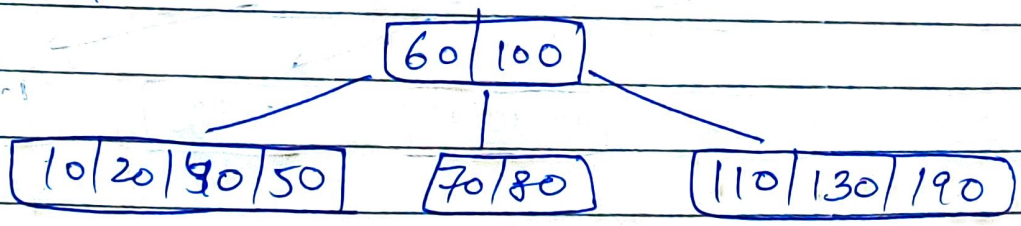

~~10 Insert 50~~



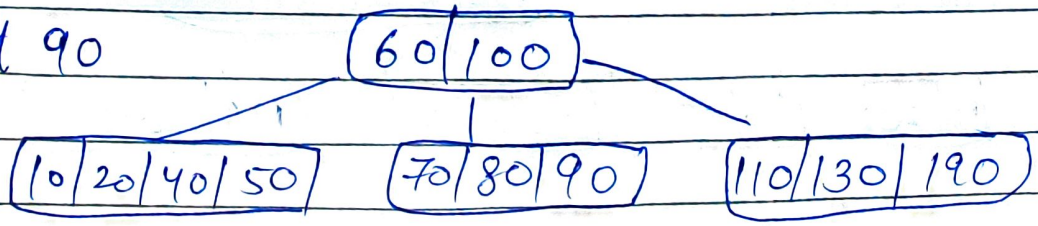

10 Insert 50




11 Insert 190



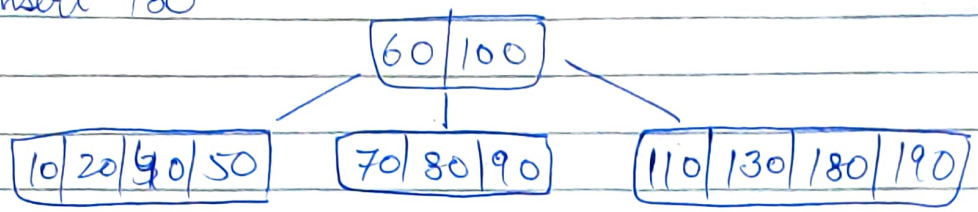

12 Insert 90



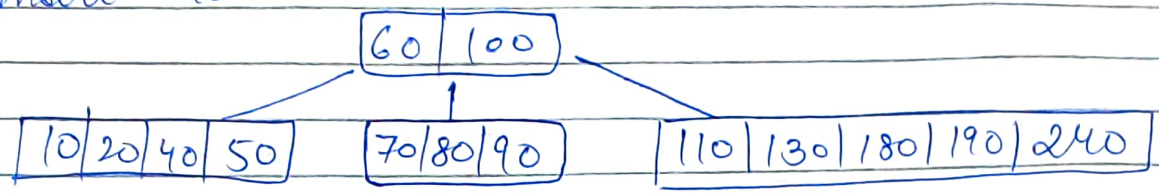
120, 140, 160

Date: / / Page no:

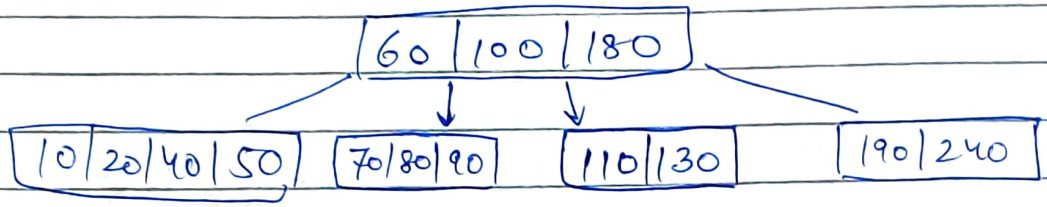
13 Insert 180



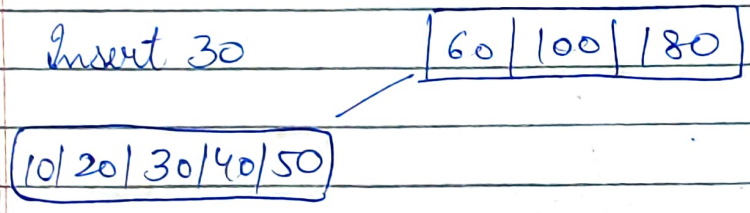

14 Insert 240



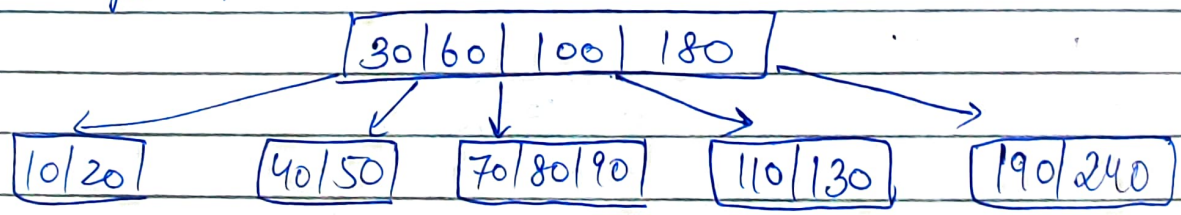

Child has M Keys -> split



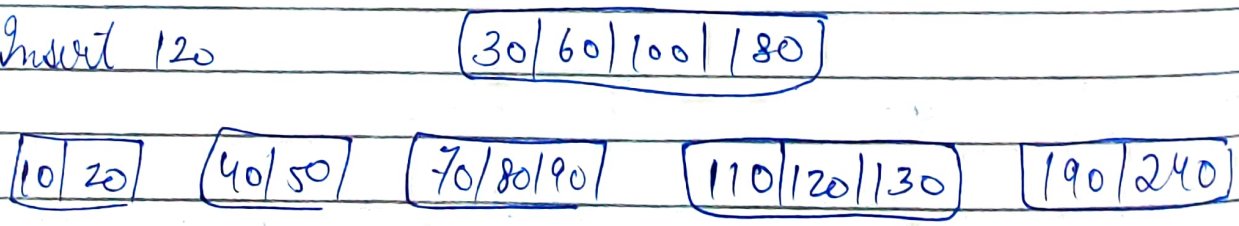

15 Insert 30




M Keys -> split

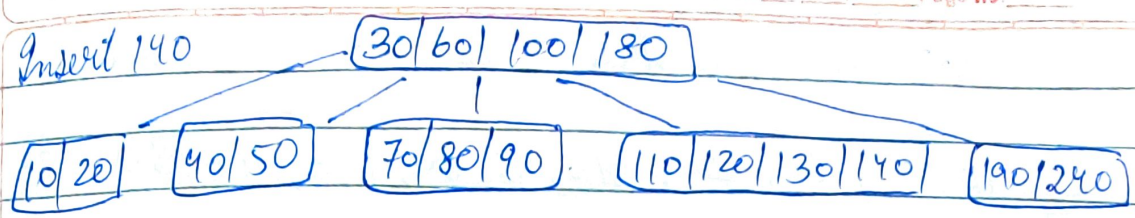



16 Insert 120

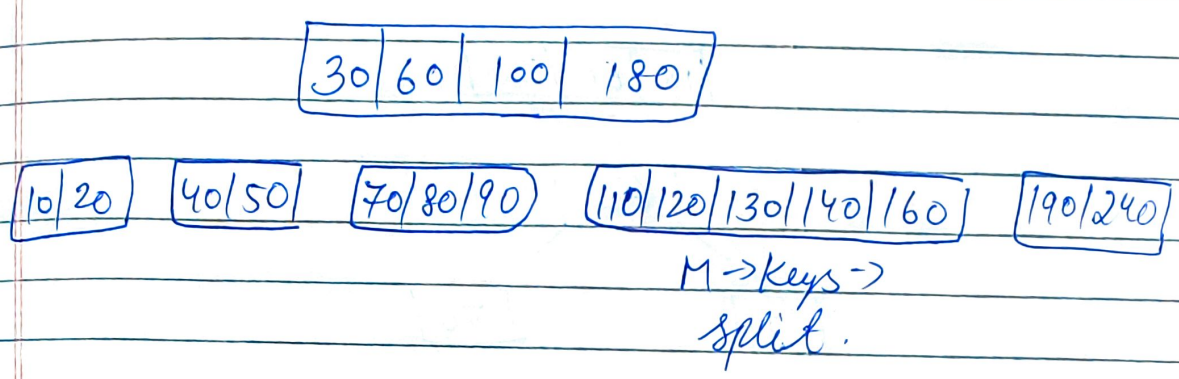


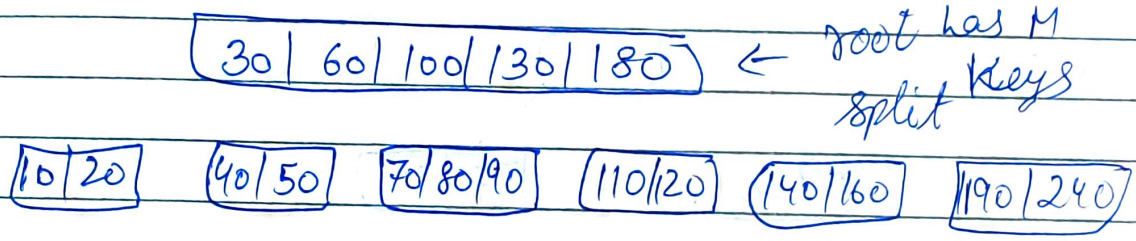
Date: __/__/__ Page no:

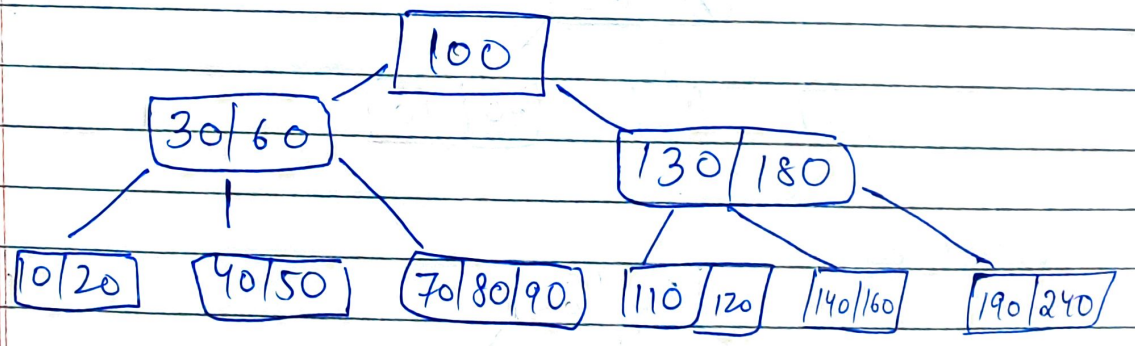
17 Insert 140




18 Insert 160



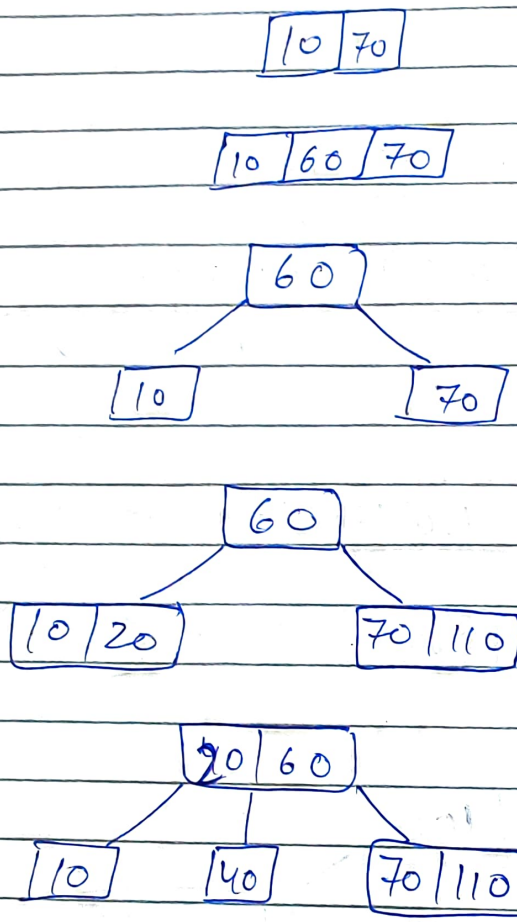



Date: __/__/____ Page no:

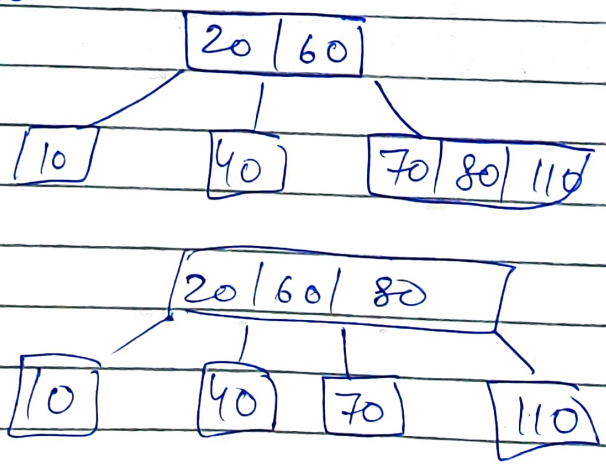
Q Create a 3-way tree using foll. elements

10, 70, 60, 20, 110, 40, 80, 130, 100, 50, 190

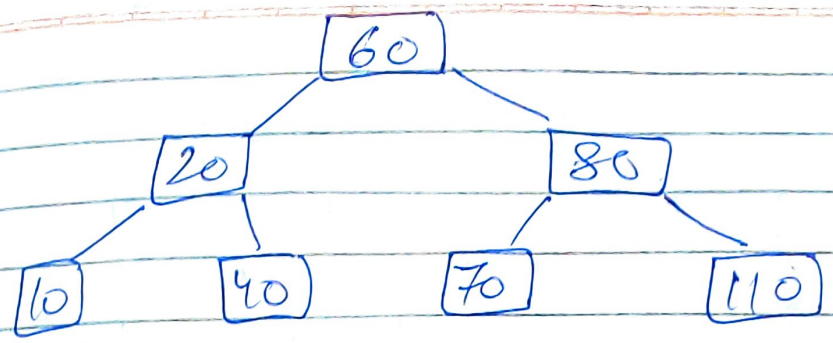
Ans



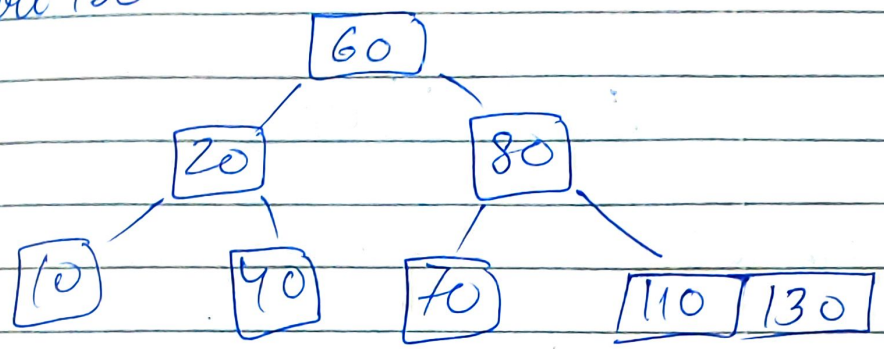

Insert 80



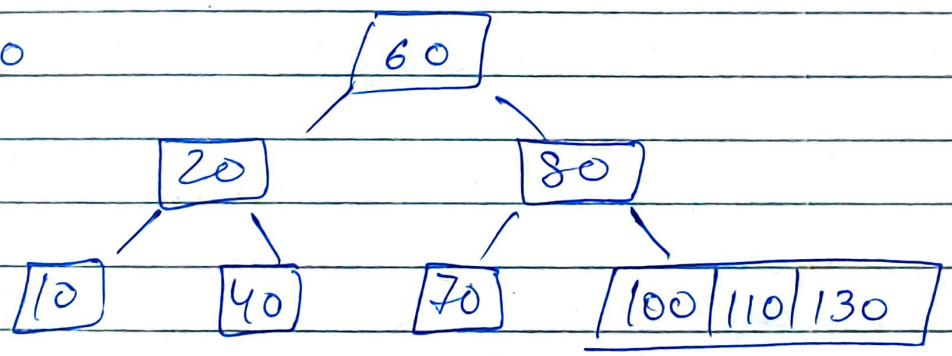

split



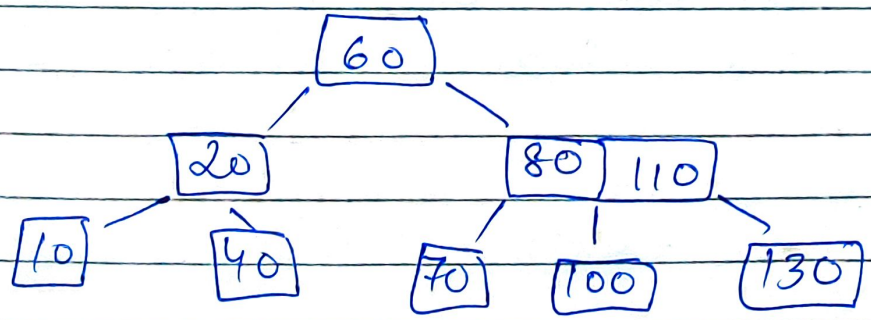

Insert 130



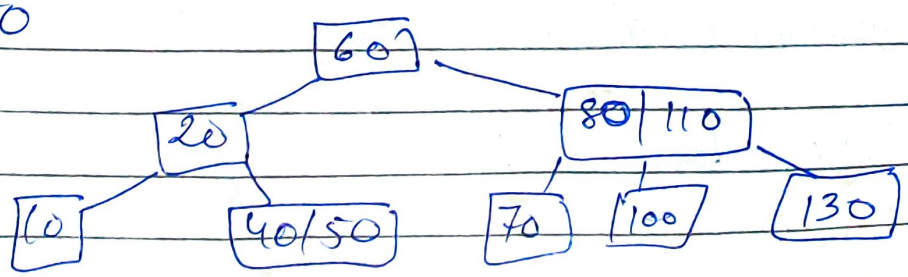

Insert 100



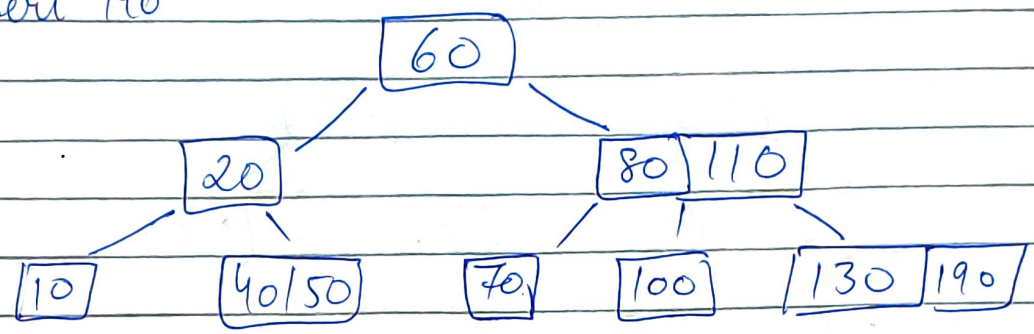

split




Insert 50



Insert 190




16/12/21

# AVL Tree

## Problem with Binary Search Tree –

If the elements to be inserted in BST follows some kind of order then the tree we will get is either a left skewed tree or a right skewed tree.

To remove this problem we use balanced binary trees.

AVL is a self balancing binary search tree, where difference of right sub tree and left subtree height to a node is at most 1

## Balancing & Balance factor

The balancing condition of AVL tree is

Balance factor = Height of left subtree − Height of Right subtree

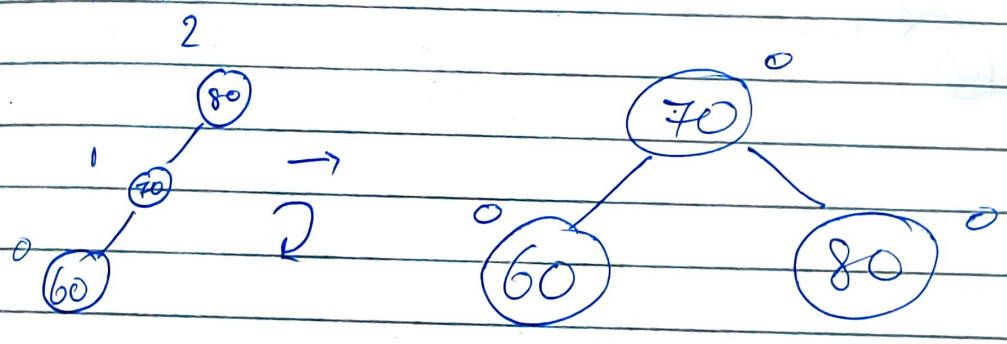
Balance factor = Height of left subtree − Height of Right subtree

The balance factor should be (-1, 0 or 1) other than this will cause Re-structuring (balancing) the tree.

Balancing performed is ~~cal~~ carried in the following ways ->

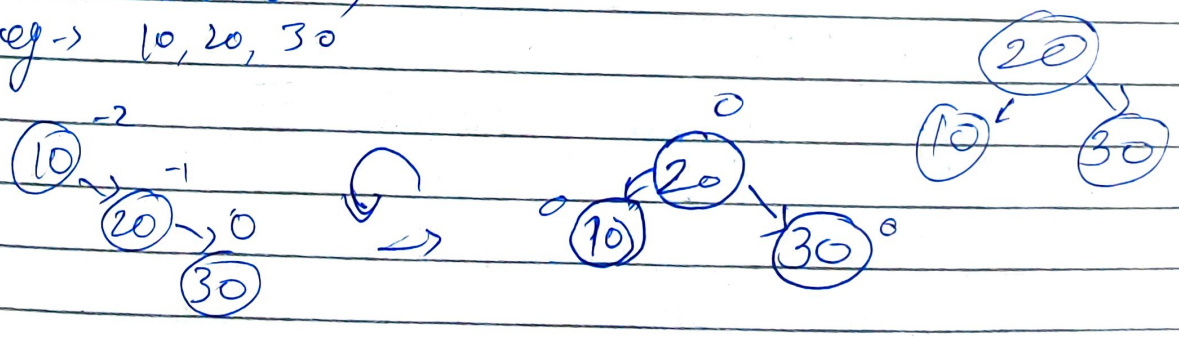
1 Right Rotation (RR)

eg -> 80, 70, 60



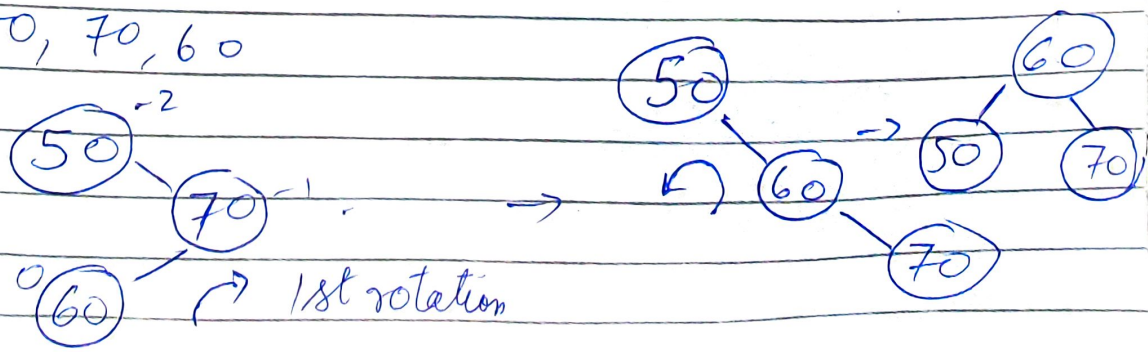

2 Left Rotation (LL)

eg -> 10, 20, 30




3 Left Right ~~Rotati~~ double Rotation (LR)

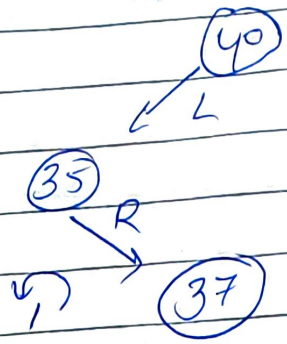
50, 70, 60

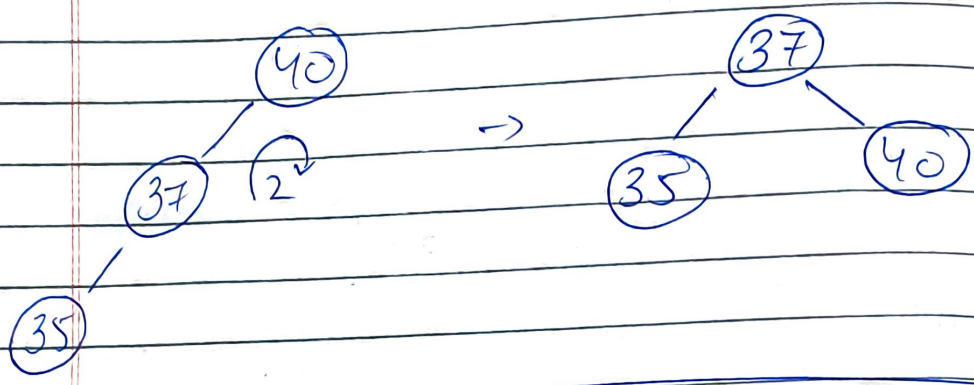


Date: / / Page no:

4 Right left double Rotation (RL)

40, 35, 37




Screenshot

Assignment - 3 Data

Q2 What is Graph representation?
Explaination
Types Adjacency matrix.

Undirected Graph representation
Directed " "
Undirected weighted " " .

Pros & Cons

Q3 BFS & DFS

17/12/21

Date: / / Page no:

# Algorithm for Right Rotation

Step 1: Begin  P -> critical node (

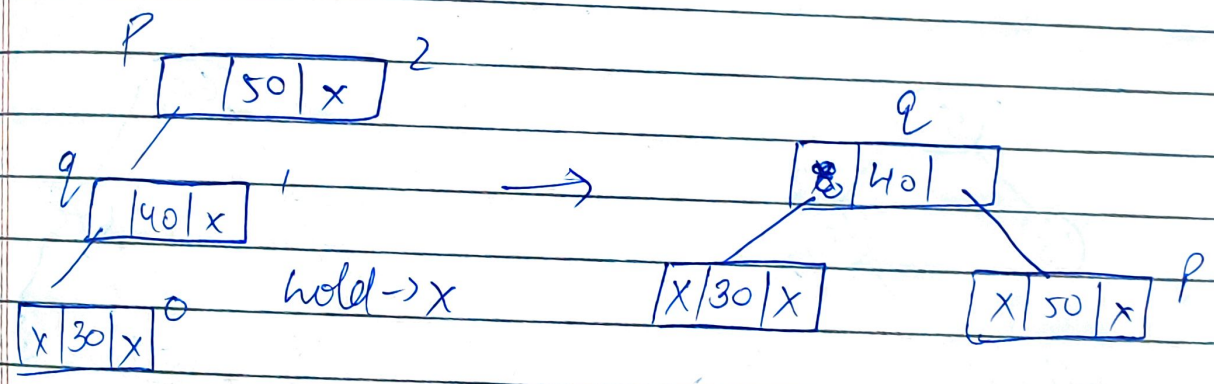
Step 2:- set q = p -> left

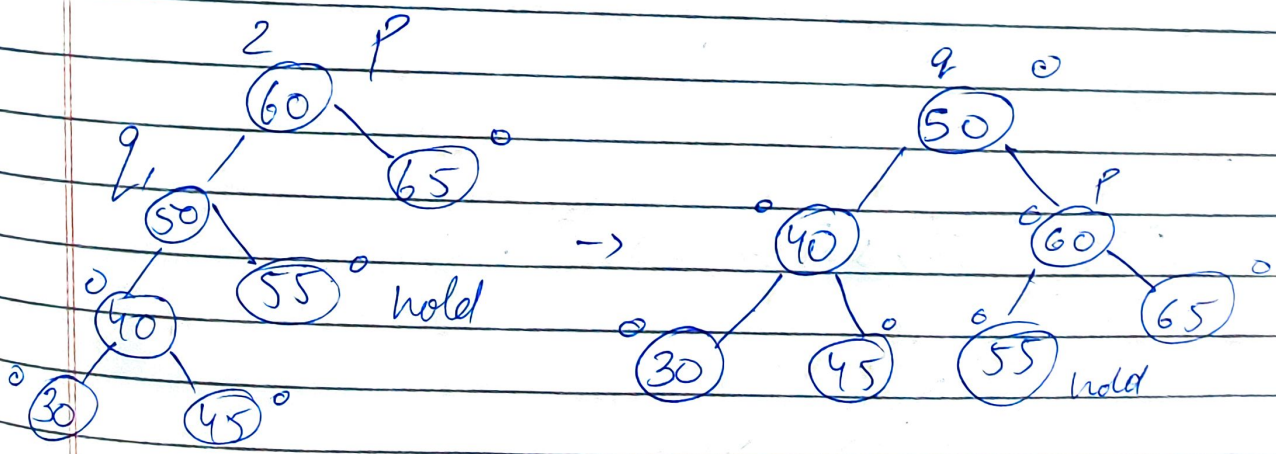
Step 3:- set hold = q -> right

Step 4:- set q -> right = p

Step 5:- set p -> left = hold

Step 6: return

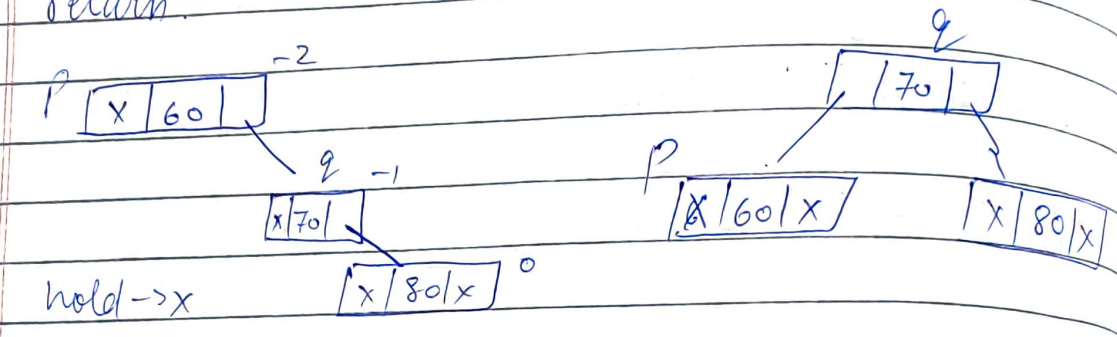


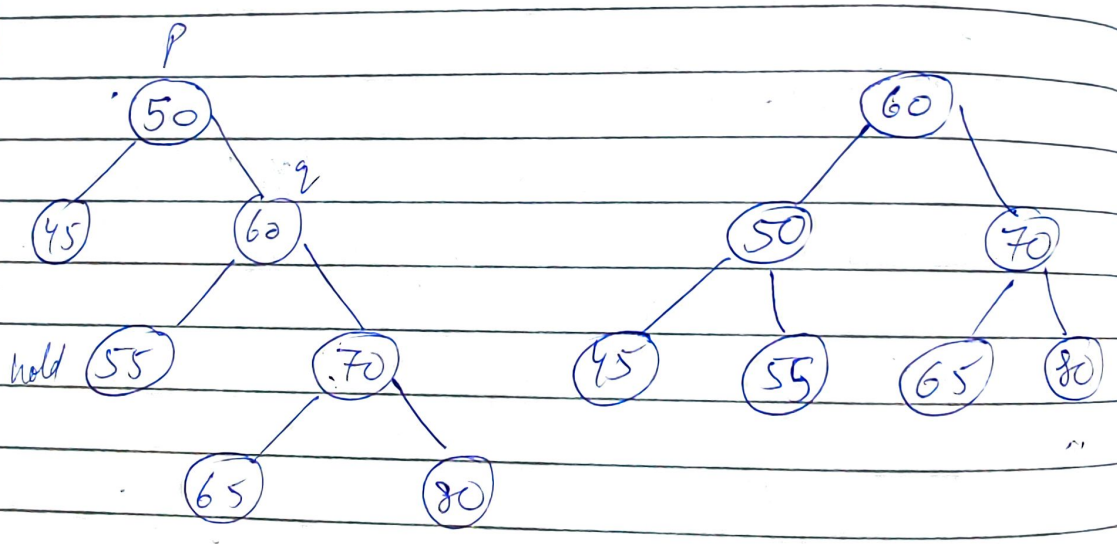



Date: / / Page no:

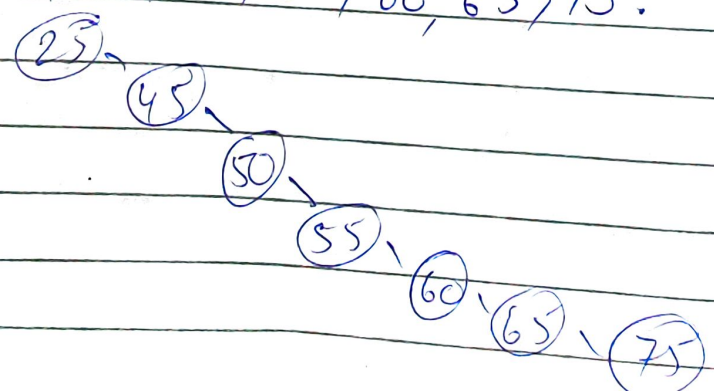
# Algorithm for left Rotation

Step 1: Begin
Step 2: set q = p -> right
Step 3: set hold = q -> left
Step 4: set q -> left = p
Step 5: set p -> right = hold
Step 6: return.

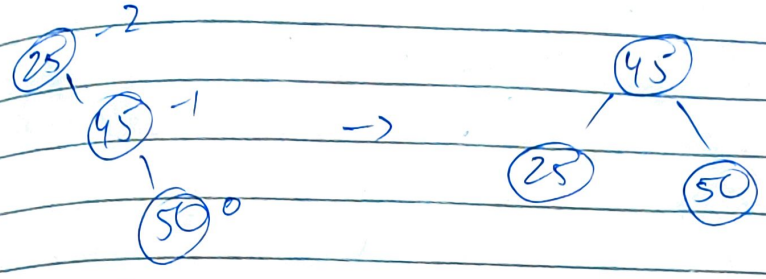


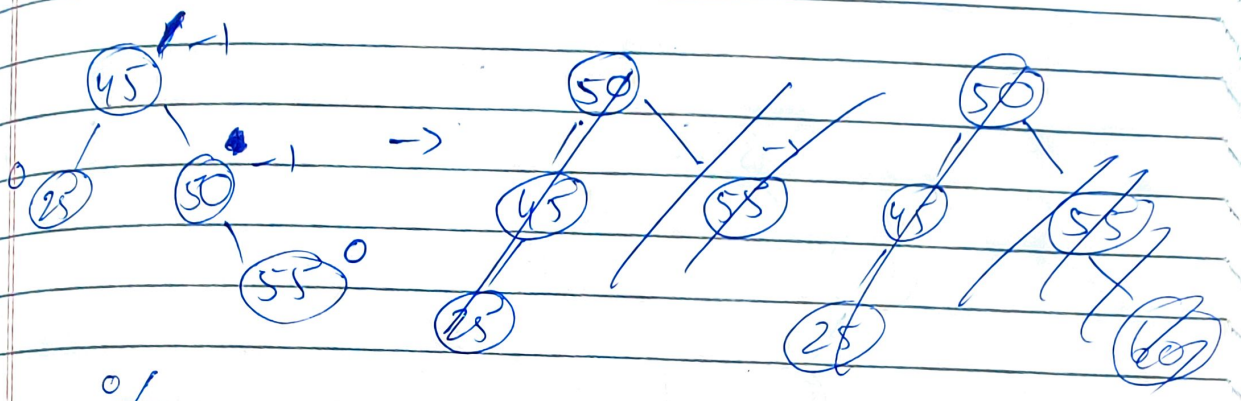



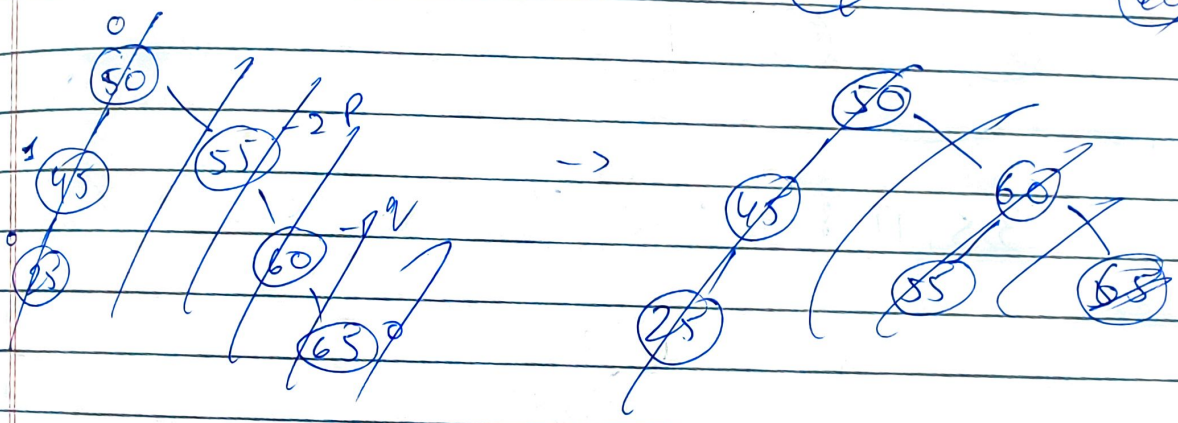

8 25, 45, 50, 55, 60, 65, 75.

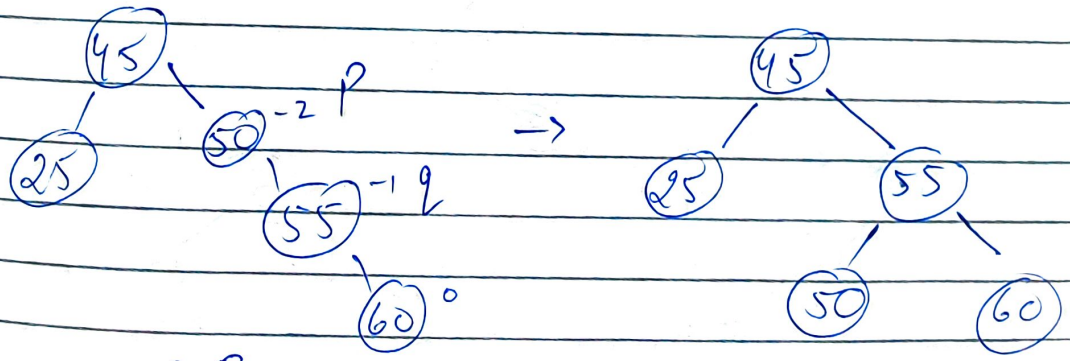


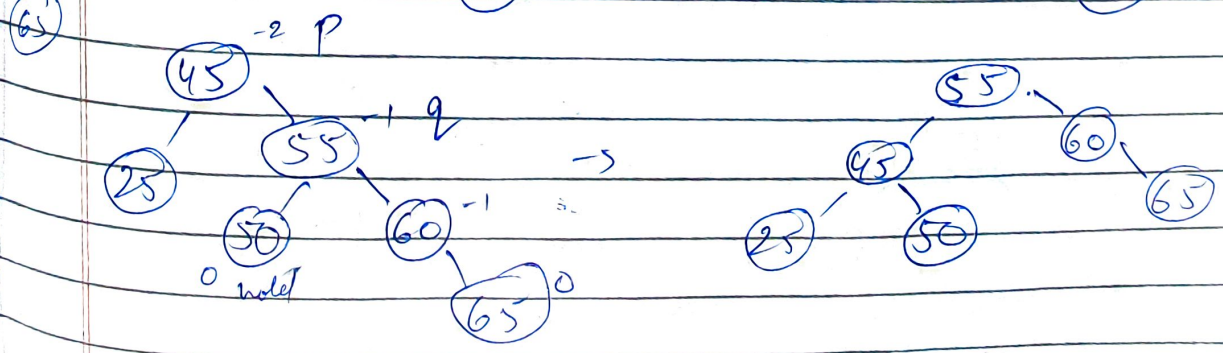
Date: / / Page no:


25 -2
45 -1
50 0
45
25
50

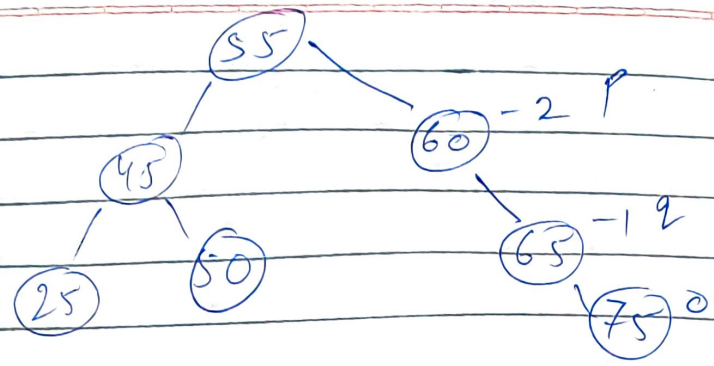

45 -1
25 0
50 -1
55 0
50
45
25
55
50
45
25
55
60


50 0
45 1
25 0
55 -2 p
60 -1 q
65 0
50
45
25
60
55
65

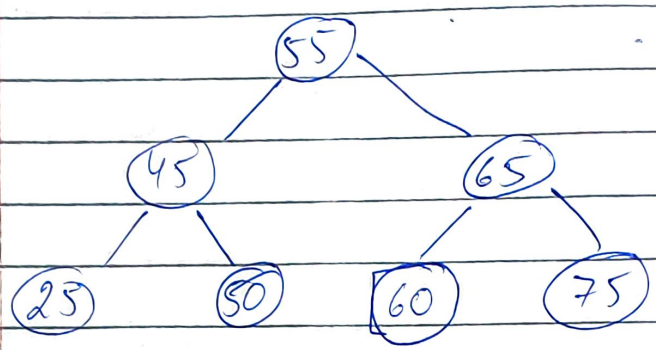

45
25
50 -2 p
55 -1 q
60 0
45
25
55
50
60


65
45 -2 p
25
55 -1 q
50 0 balef
60 -1
65 0
55
45
60
25
50
65

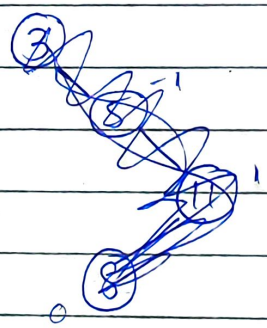
48

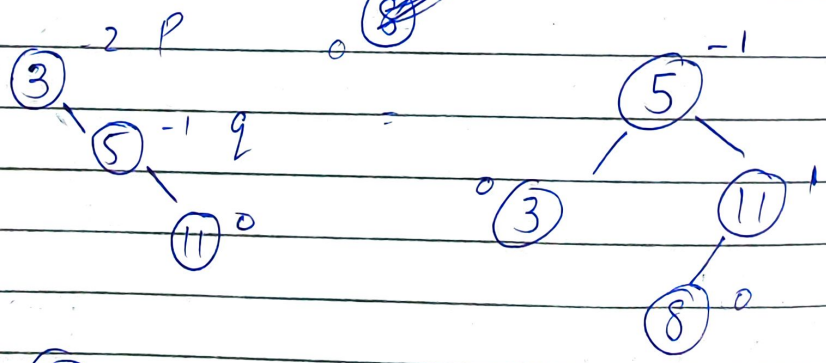


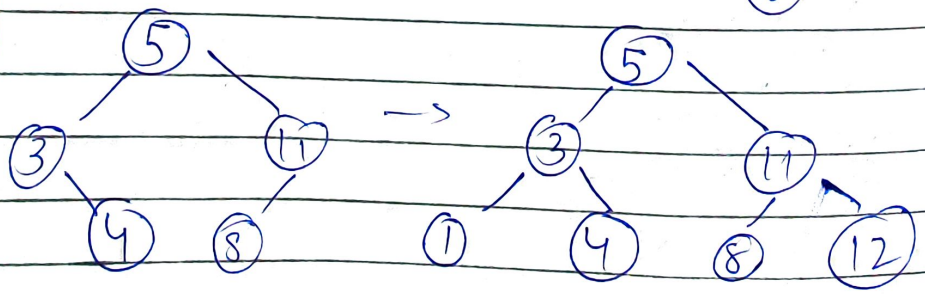

hold → X

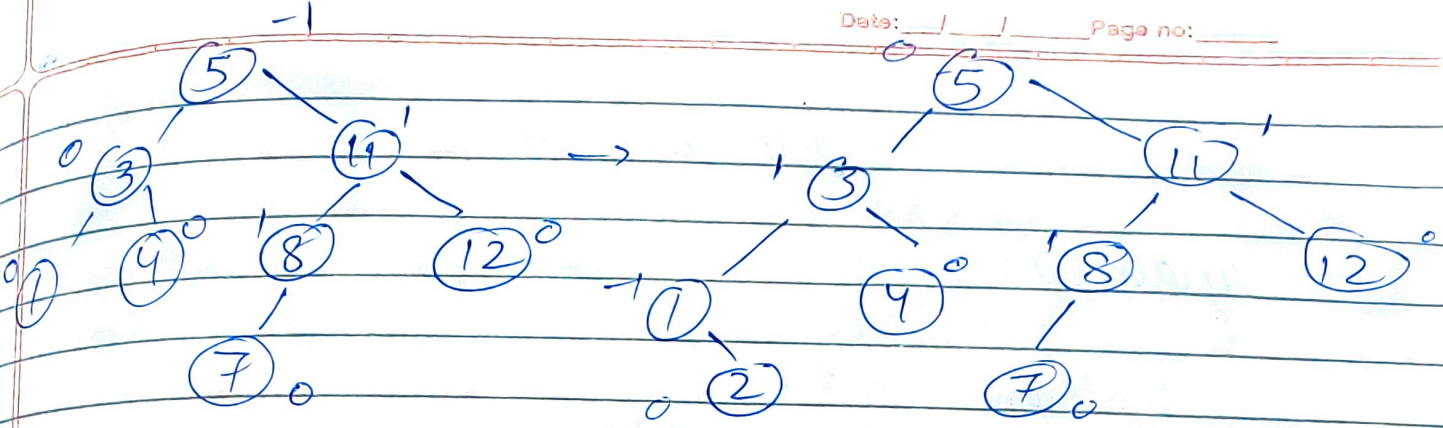


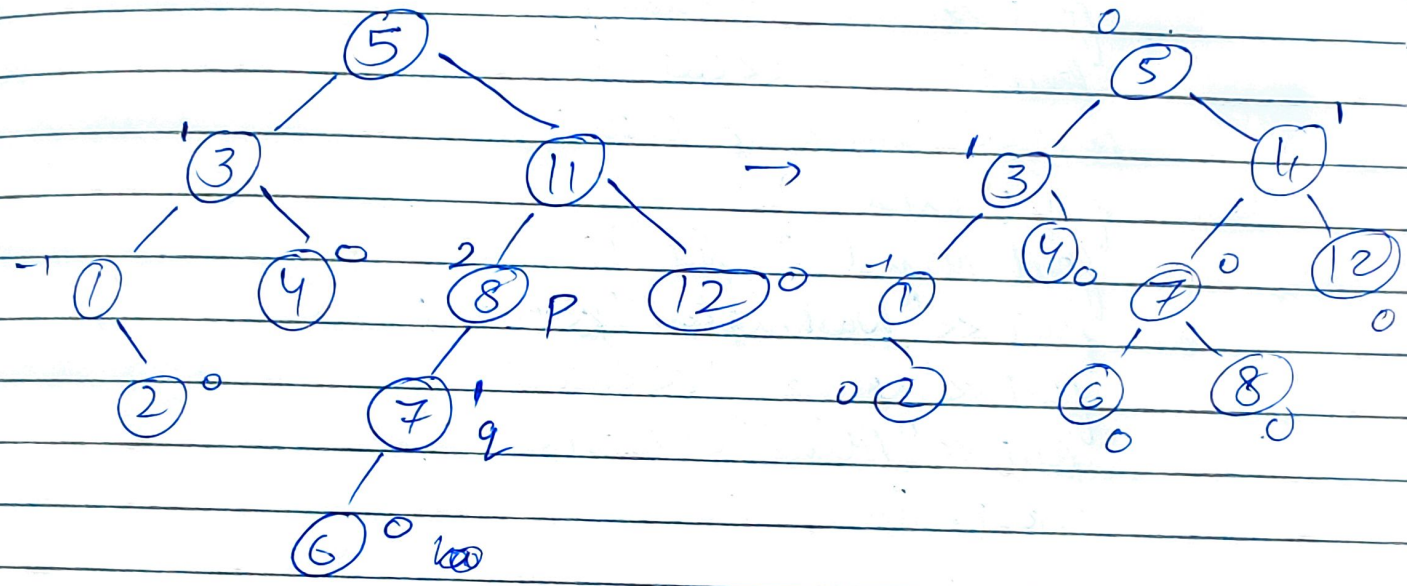

3, 5, 11, 8, 4, 1, 12, 7, 2, 6, 10.

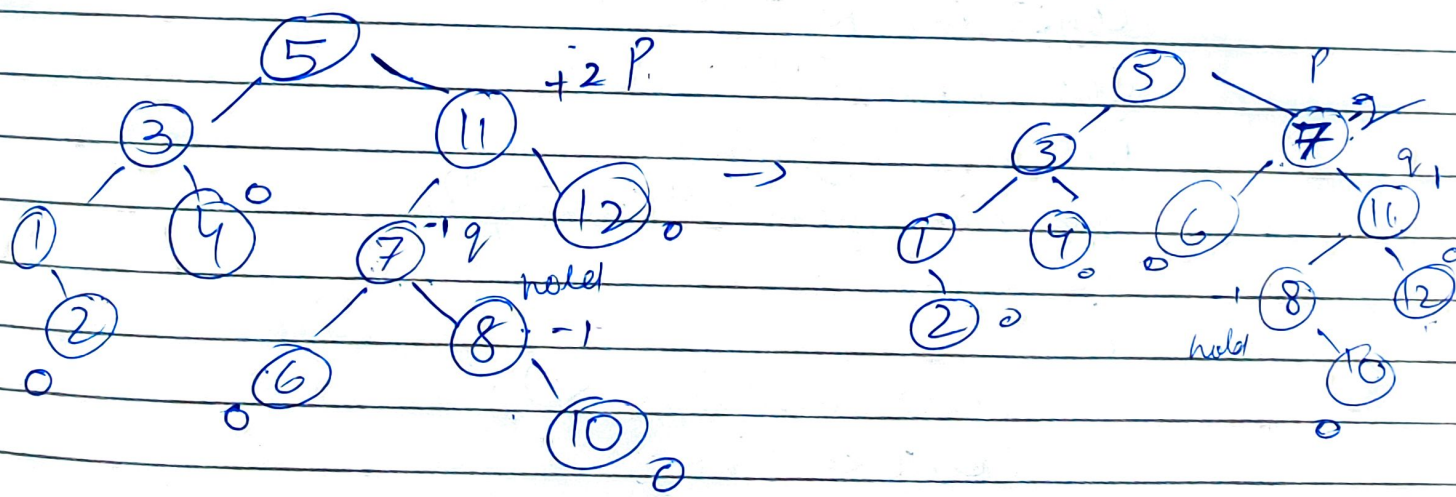







+2 p
p
q
q
q
p
hold
hold