TREES

- **Introduction to tree traversals :**

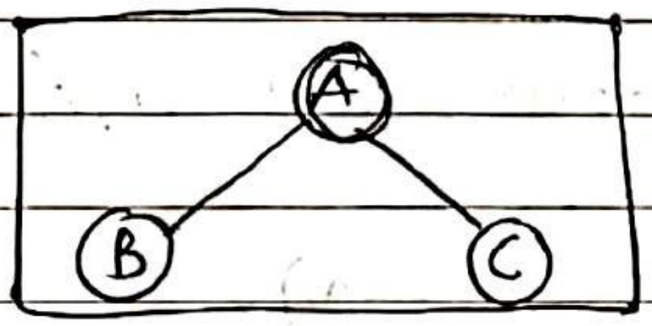
- **Traversing means** → We want to see what is information present in Every node.

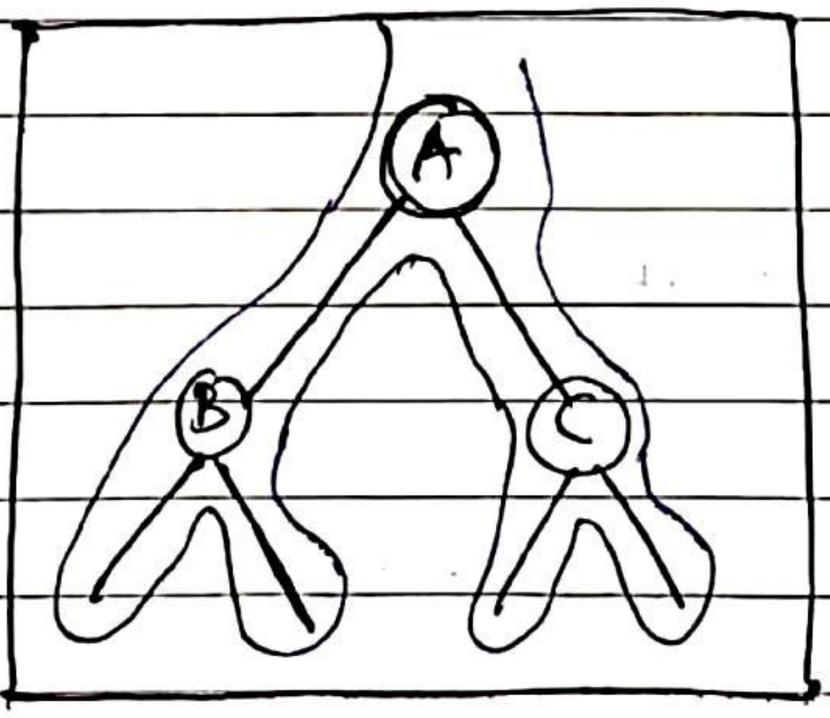
- **Searching means** → searching for a particular node.

Methods of traversing

visit

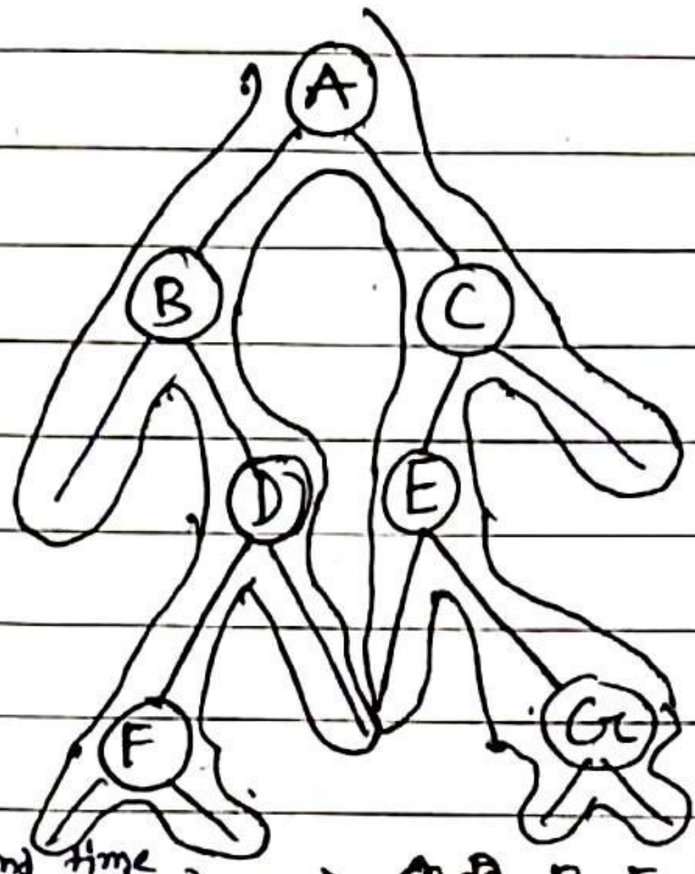
- 2nd time → Inorder traversing : (Left - Root - Right) — BAC
- 1st time → preorder traversing (Root - Left - Right) — ABC
- 3rd time → postorder traversing : (Left - Right - Root) — BCA





1st time — ABC — (pre)
2nd time — BAC — (Inorder)
3rd time — BCA — (post)

example :



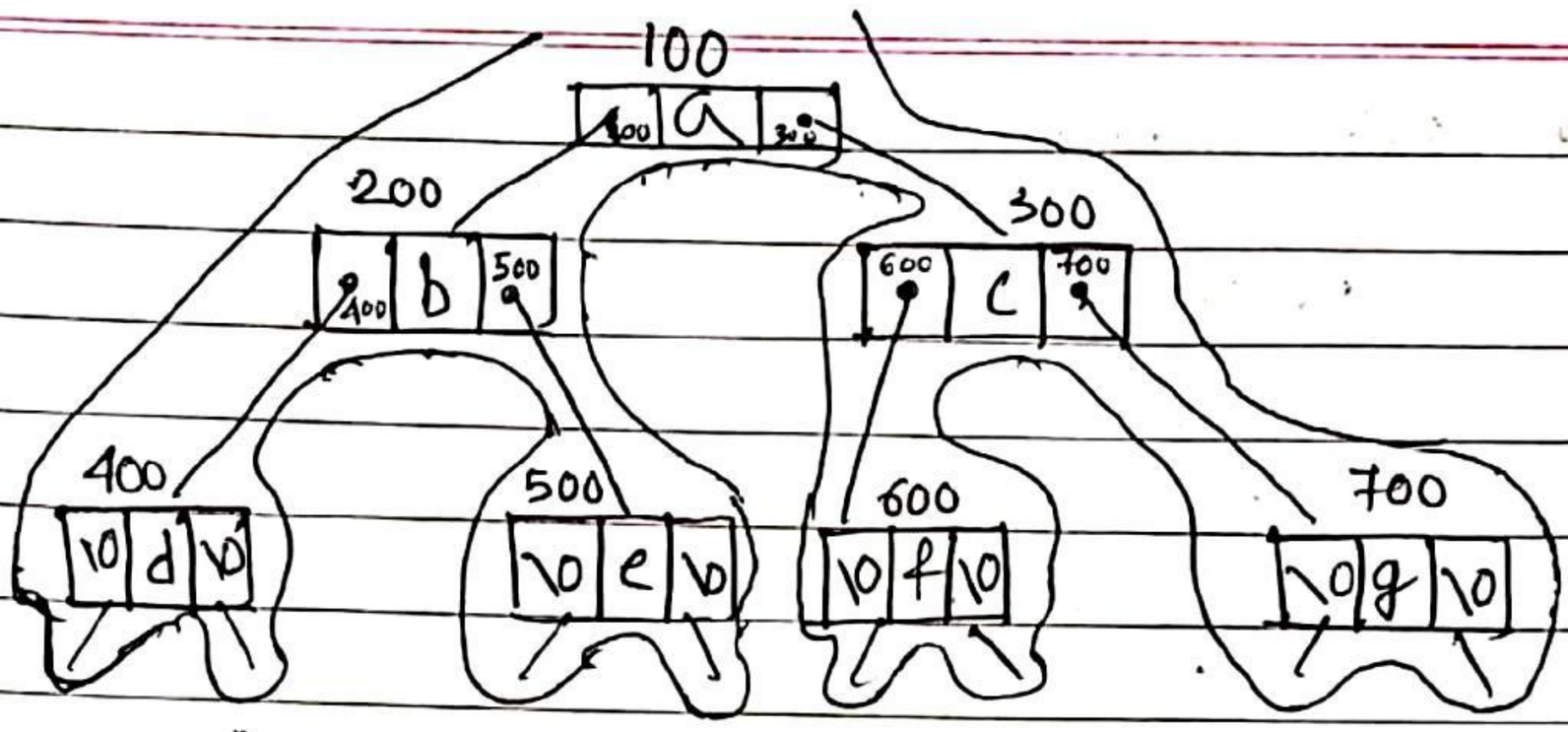
- Inorder (2nd time) → B F D A E G C
- ~~post~~ preorder (1st time) → A B D F G E C
- postorder (3rd time) → F D B G E C A

- Implementation of traversals and time and space analysis -

Inorder traversing → (left — root - Right).

Program : (Using recursion)

```
struct node
{   char data;
    struct node* left
                 * Right;
};

void Inorder (struct node * t)
{
    if (t != NULL)
    {
    1. Inorder (t → left);
    2. printf ("%c", t → data);
    3. Inorder (t → Right);
    }
}
```



main | inorder | I | I | I | I

t=100	t=200	t=400	t=X0	t=X0	t=500	t=X0	t=X0	t=300	...
4	4	4			4			2	

Stack

O/p : d b e a f c g

In/pre/post order traversal all take

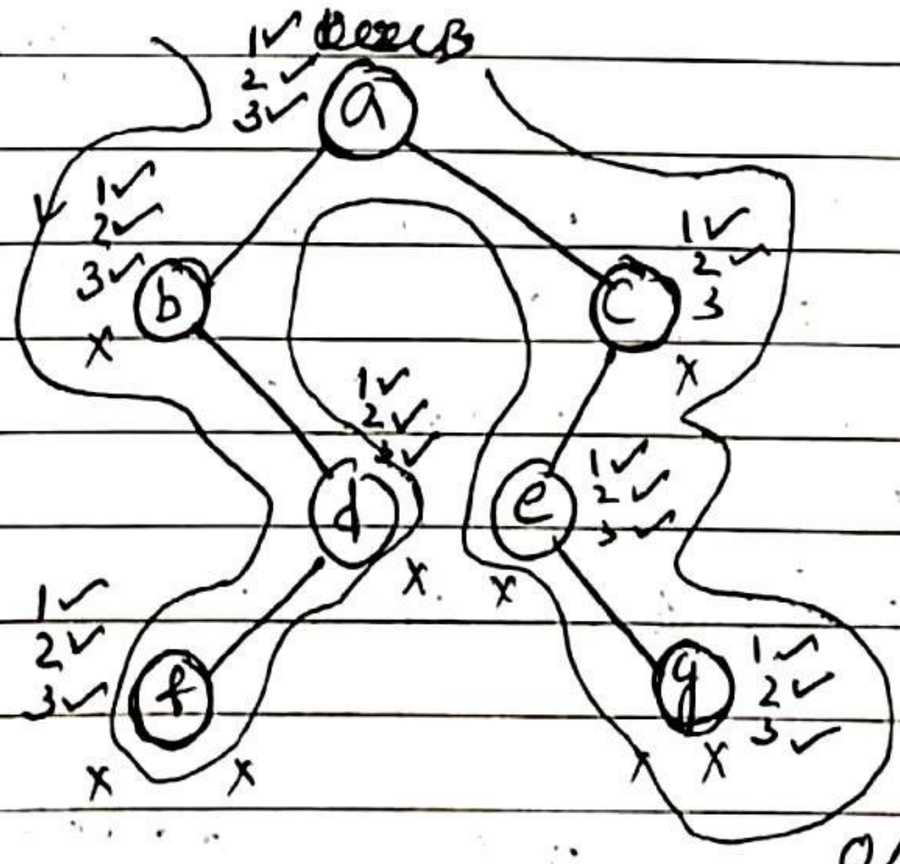
Time complexity = Time and Space complexity = O(n).

{ Inorder → 1, 2, 3 (lines)
pre order → 2, 1, 3
post order → 1, 3, 2

- Double order traversal :

(one other method)

example: (Inorder) (root as Input)

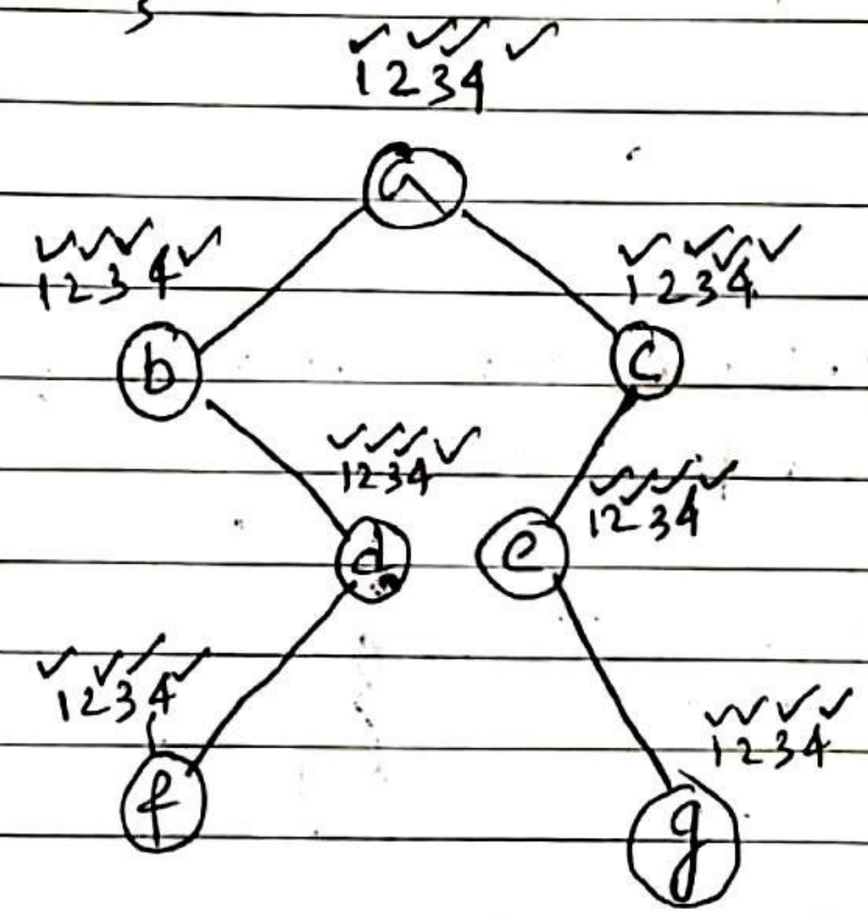



```
IINORDER(t)
{
1. INORDER(t→Left)
2. pf
3. INORDER(t→Right)
}
```

O/p : b f d a e g c

- Double order :

```
Do (t)
{ Pf (t)
  {
  1 PF (t → data)
  2 Do (t → left)
  3 Pf (t → data)
  4 Do (t → right)
  }
```

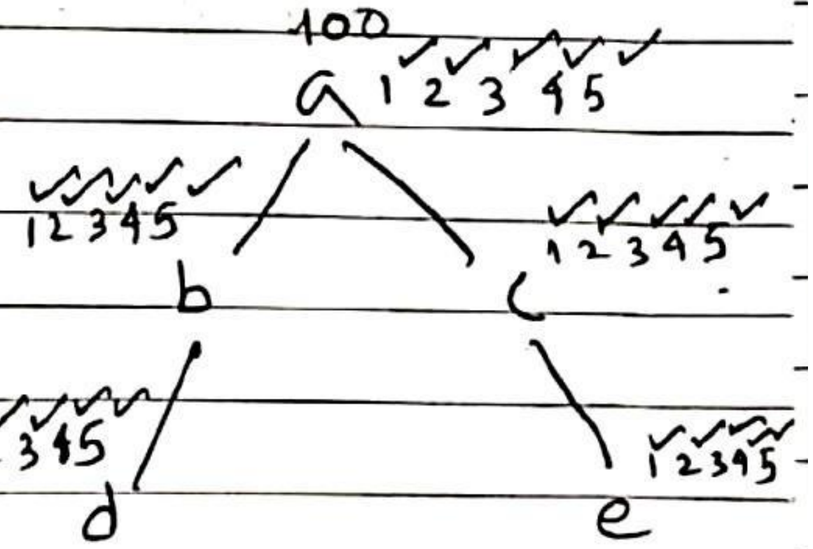


Output :

a b b d f f d a c e e g g e

- Triple order traversal :

```
                    (100)
void TO (struct node *t)
{
  if (t)
  {
  1. pf("%d", t → data);
  2. TO (t → Left);
  3. pf("%c", t → data);
  4. TO (t → Right);
  5. pf("%d", t → data);
  }
}
```

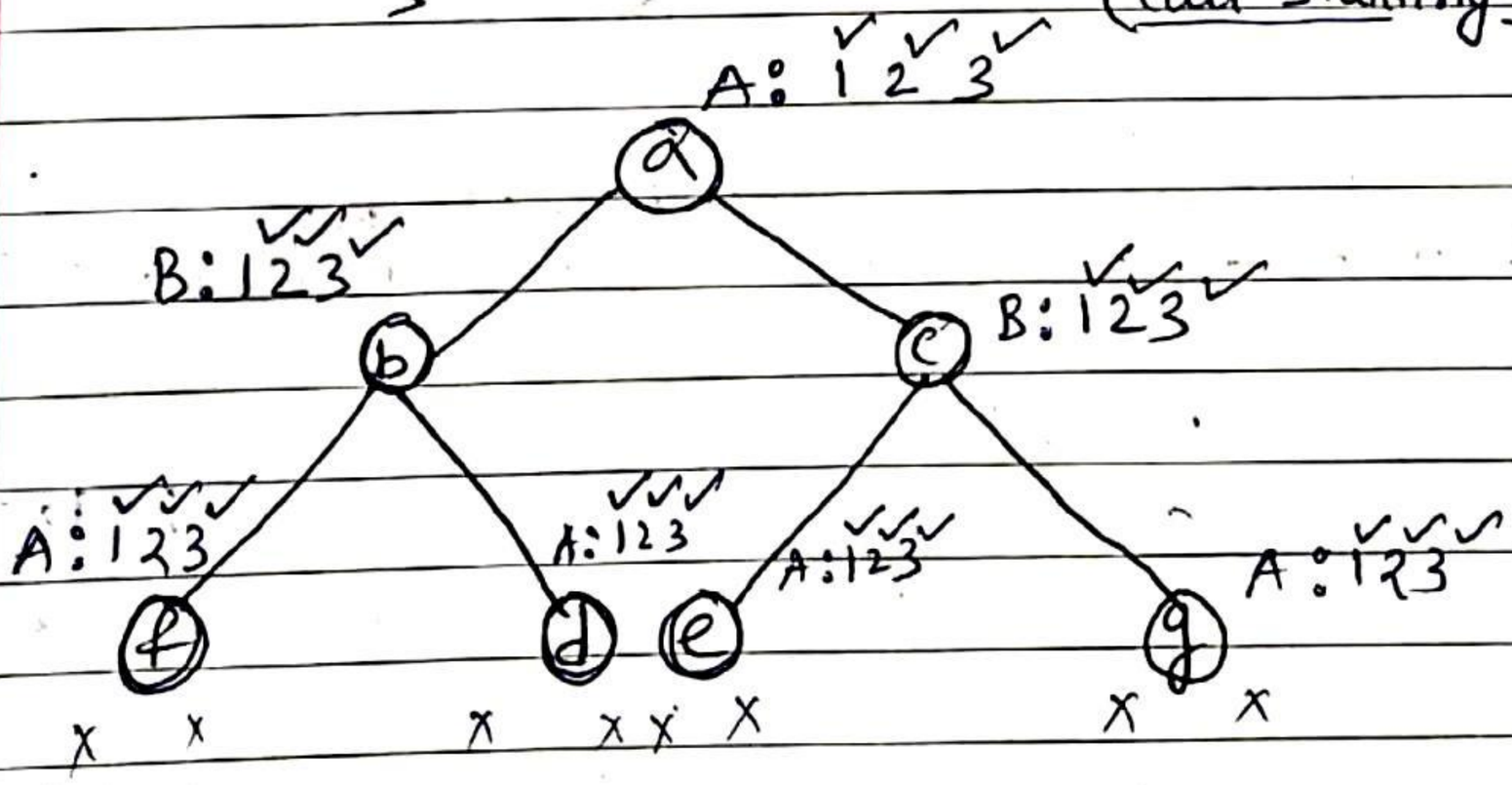


output : a b d d d b b a c c e e e c c a

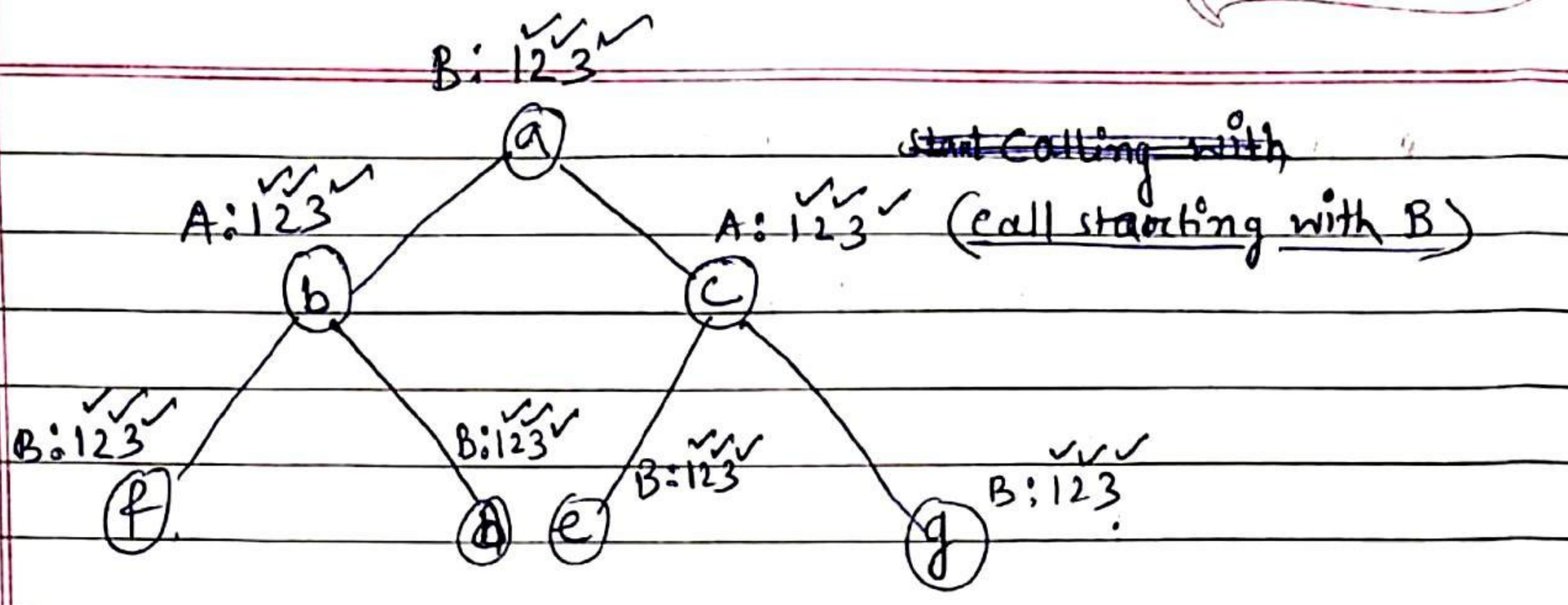
- Indirect recursion on trees:-

```
void A (struct Node *t)
{
    if(t)
    {1. pf("%d", t->data);
     2. B (t->left);
     3. B (t->right);
    }
}

void B (struct node *t)
{
    if(t)
    {
     1. A (t->left);
     2. pf("%d", t->data);
     3. A (t->Right);
    }
}
```

(call starting with A)



afbdecg — output.



bfdaceg → output.

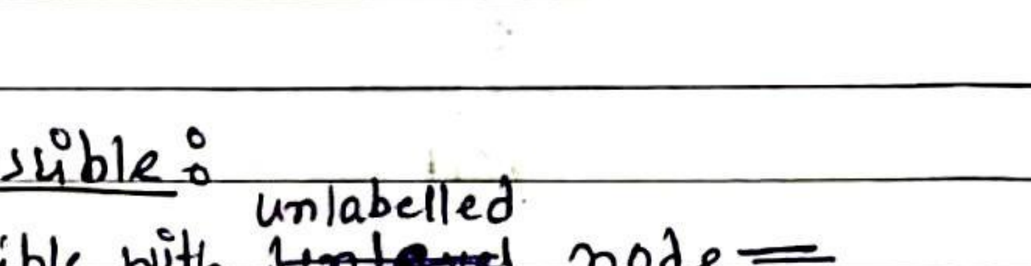
(structures)

- Number of Binary trees possible:
- Number of Binary tree possible with unlabelled node =

1 node no. of binary trees possible is: one

2 " " " " : two

3 " " " " : 5



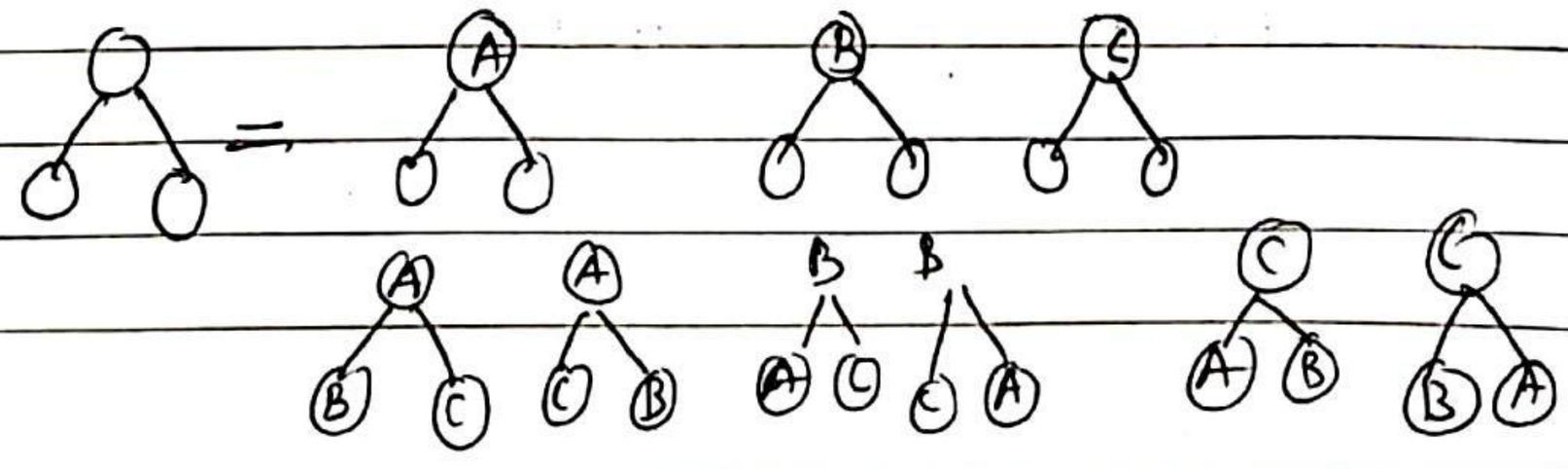
With 'n' unlabelled node no. of Binary trees possible is: (structure possible)

$$\frac{2nC_n}{(n+1)}$$

$$n=3$$

$$\frac{6C_3}{4} = \frac{\lfloor 6}{\lfloor 3 \lfloor 3 * 4} = \frac{6 \times 5 \times 4}{3 \times 2 \times 4} = 5$$

- Number of Binary tree possible with the labelled node 1 =



With 'n' labelled nodes no. of binary tree possible are =

$$= \frac{{}^{2n}C_n * n!}{(n+1)}$$

Ques

{ Question type =
{ How many node possible?

Question:

All the Binary Trees with 3 nodes A, B and C which have preordere ABC (1st time) How many binary tree possible?

→

With 3 unlabelled node trees possible : 5
" 3 labelled " " : 5 × 12 = 30.

→

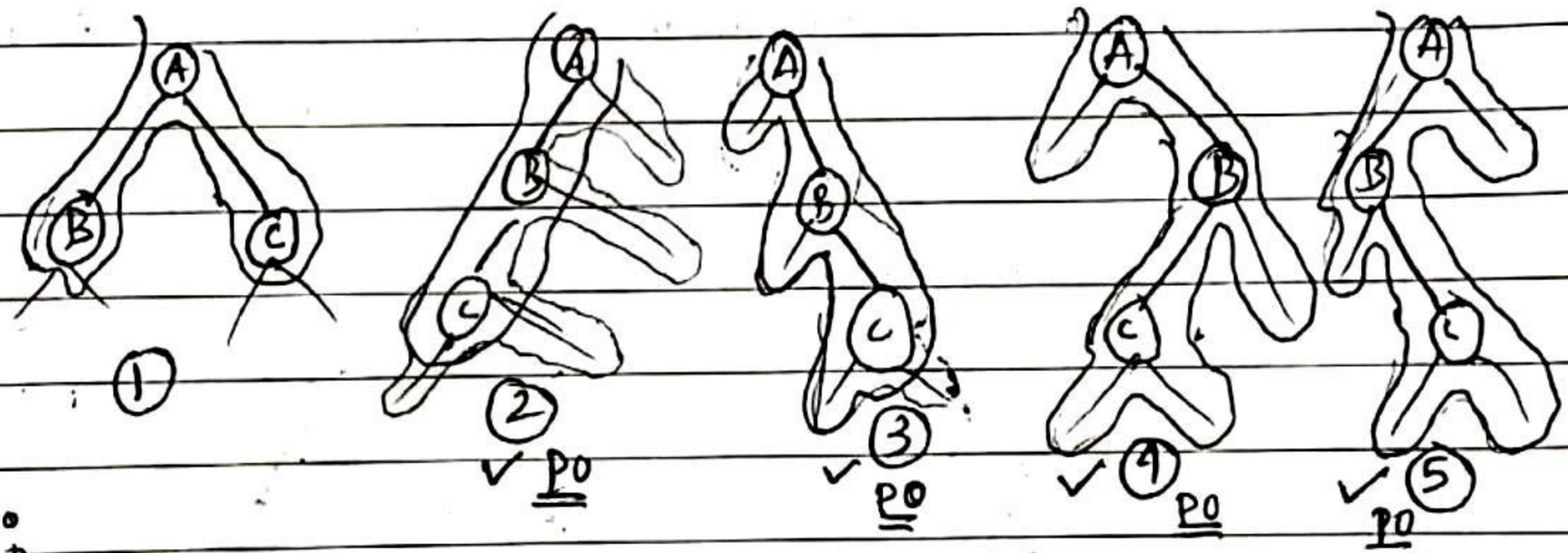


fig: ①

When:
preorder "ABC" && PostOrder is "CBA" =
(3rd time)

fig: ② ③ ④ ⑤ . (4 trees possible)

When: pre "ABC" (1) && Post "CBA" (3) && Inorder "BCA" = (2nd time visit)

fig: ③ ⑤ (only - one tree possible)

(here one tree that satisfy all the condition)

→ ~~no of tree possible with node~~

given { 'n' nodes
Preorder -

no. of trees possible is = $\frac{^{2n}C_n}{(n+1)}$

→ with given preorder and postorder you may get more than '1' binary tree.

*→ with given pre ~~or~~, post, and Inorder we always get only 'one' binary tree.

(~~unic~~)
unique

example: construction of ~~uniq~~ unique binary tree using In order and preorder.

(1st) preorder = ABC
(2nd) Inorder = BAC

→

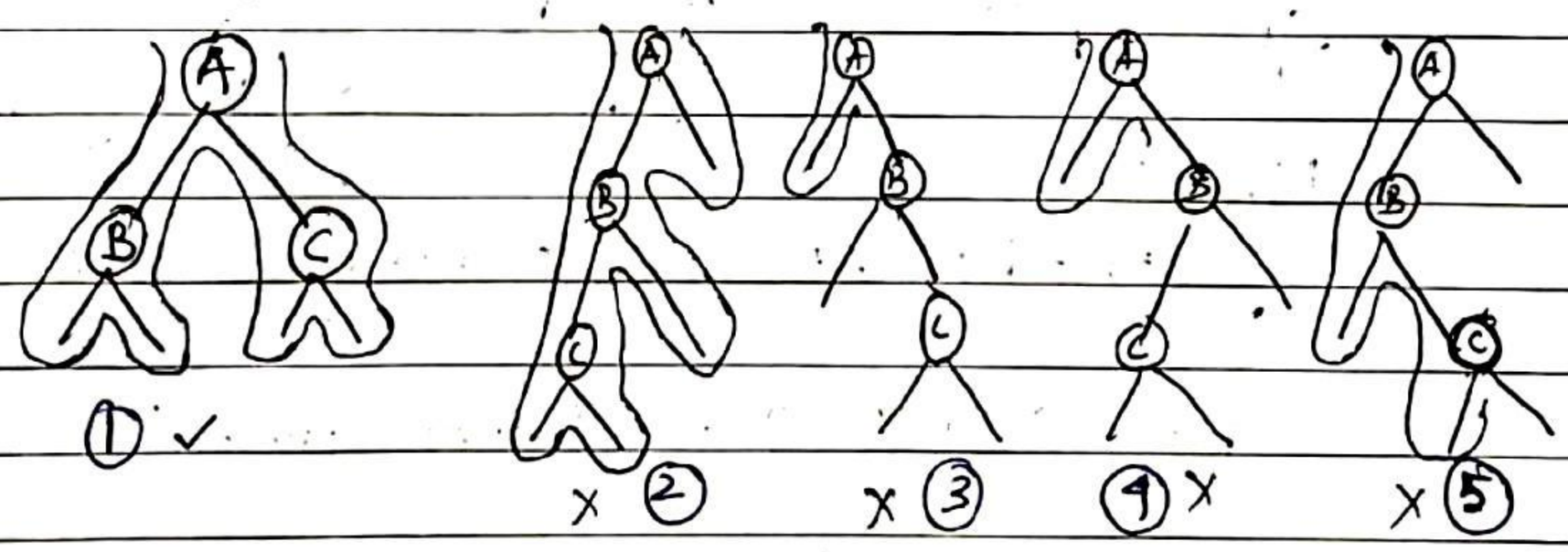


fig:

→ 5 binary tree possible with preorder ① ② ③ ④ ⑤.

→ ~~②~~ 1 BTS possible with (In and pre both order) fig - ①.

IN + PRE
IN + POST
} → possible

POST + PRE → not possible to find IN.

Question:

INORDER: 1, 2, 3, 4, 5, 6, 7, 8.

PREORDER: 5, 3, 1, 2, 4, 6, 8, 7.

POSTORDER: ?

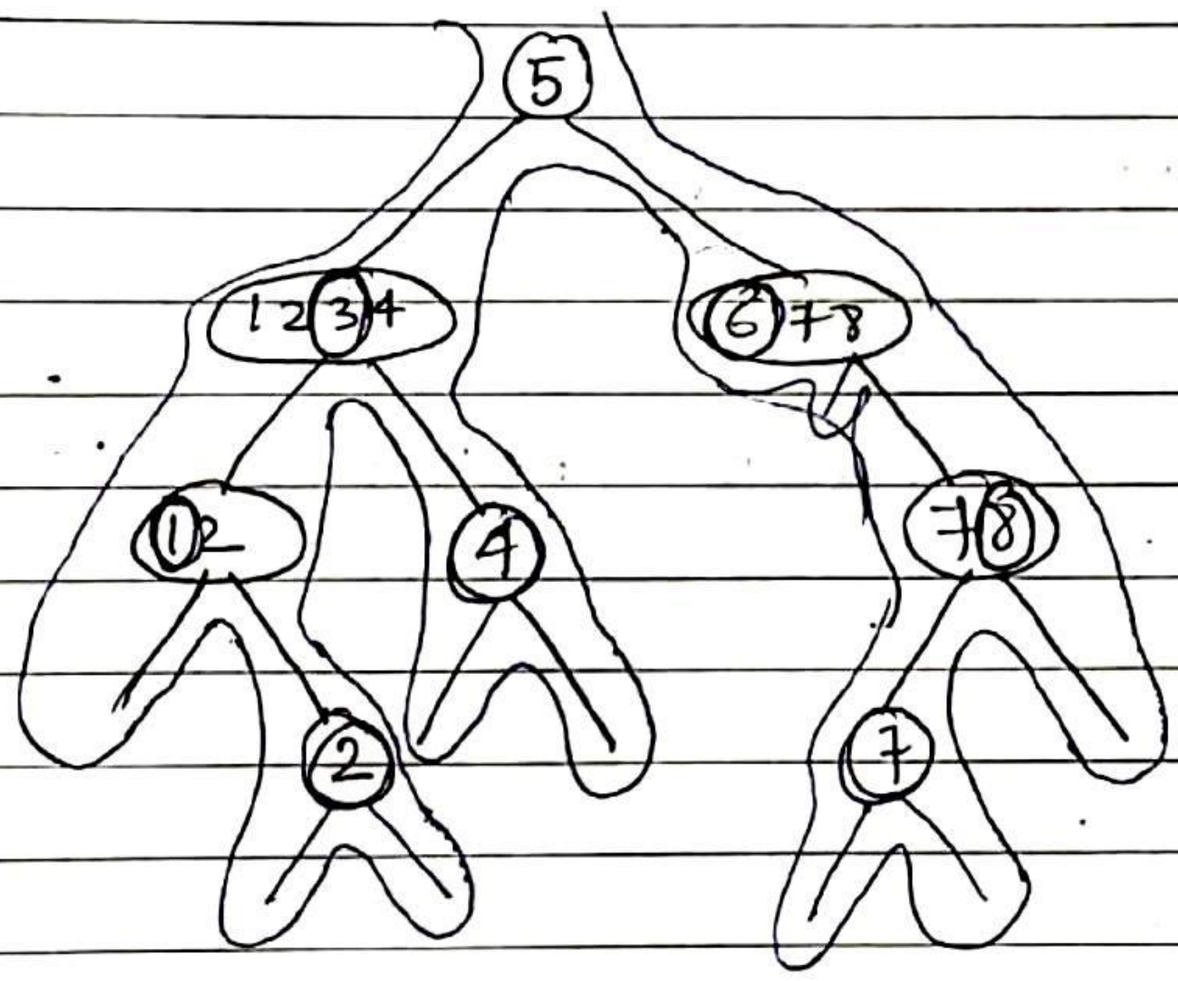
→

IN – L (Root) R

PRE – (Root) L R

→

1 2 3 4 5 6 7 8



POSTORDER: 2 1 4 3 7 8 6 5

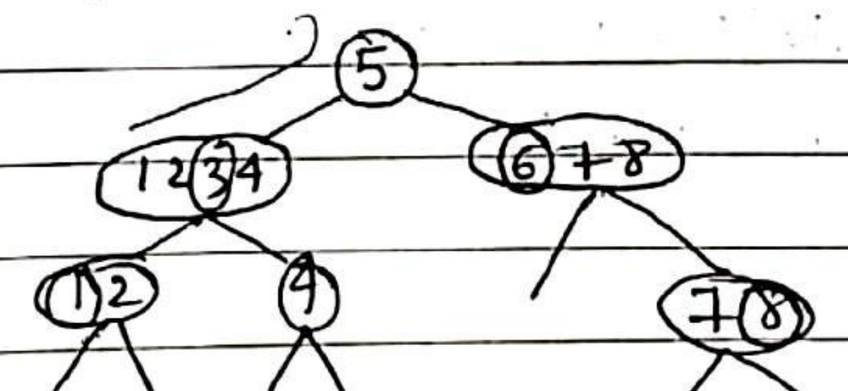
Question:

INORDER: 1 2 3 4 5 6 7 8 (L Root R)

POST: 2 1 4 3 7 8 6 5 (L R Root)

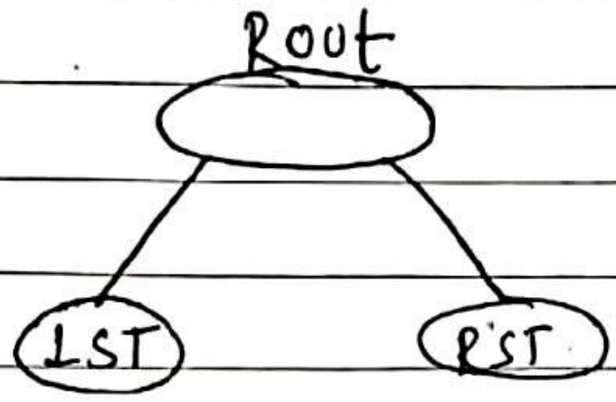
PRE: ?

→



PRE: 5 3 1 2 4 6 8 7.

- **Recursive program to count the number of nodes :-**



$NN(T) = 1 + NN(LST) + NN(RST)$

$= 0$; if T is NULL.

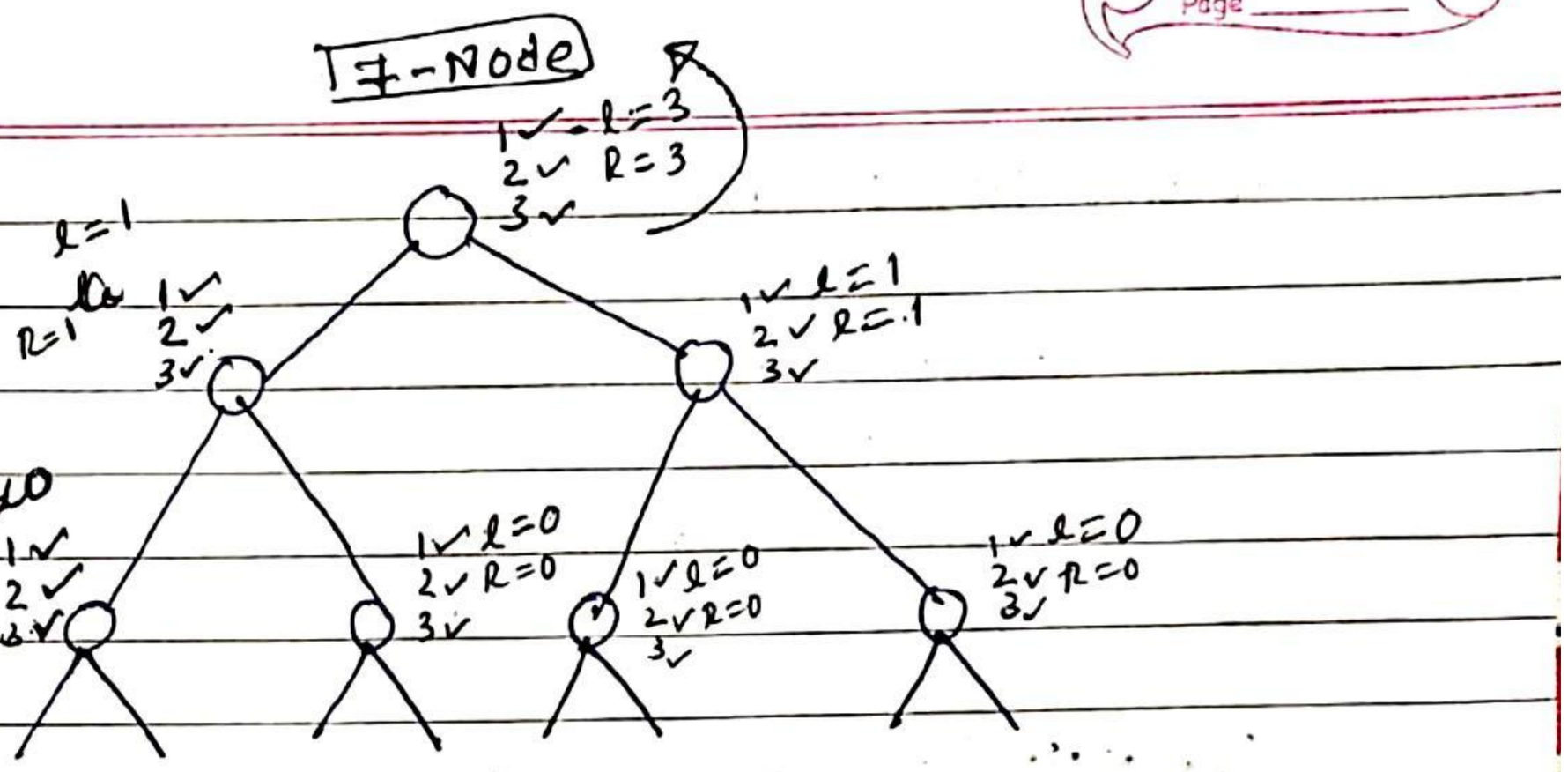
Program :-

```
struct node
{int i;
struct node *
    *left
    *Right;
}
```

```
Int NN (struct node *t)
  {
    if (t)
    { return (1+ NN (t→Left) + NN (t→Right));
    }

  else
      return 0;
  }
```

or

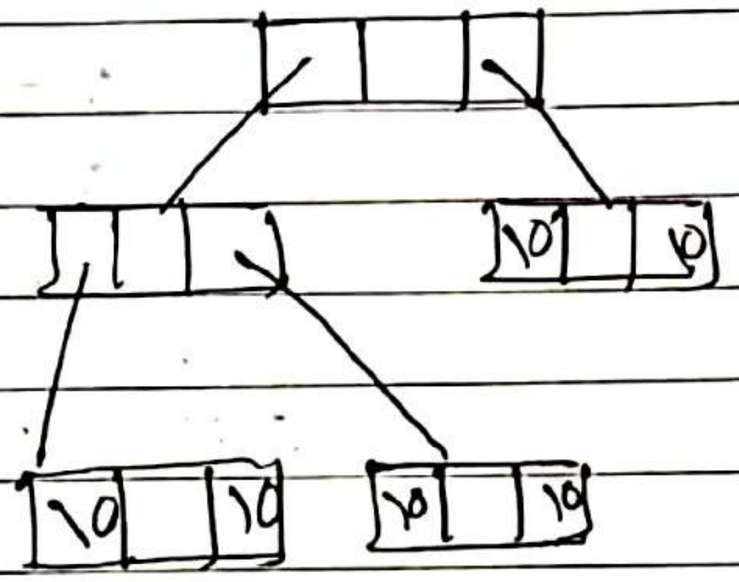
```
int NN (struct node *t)
  {
    if(t)
    int l, R;
    1. l = NN (t→left);
    2. R = NN(t→Right);
    3. return (1+l+R);
    }
  else
     return 0;
}
```



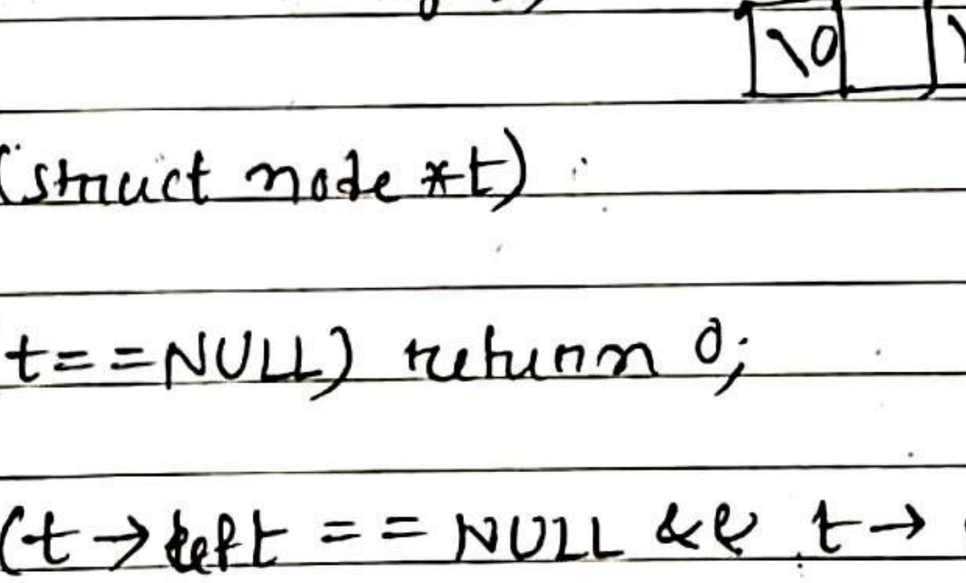
- Recursive program to count the no. of leaves and non leaves =

  - Count NO. of leaves :

```
struct node {
  { int i;
    struct node *left
                *right;
  }

int NL (struct node *t)
  {
    if (t==NULL) return 0;

    if (t->left == NULL && t-> Right == NULL)
      return 1;

    else
    { return (NL(t->left) + NL(t->Right));
    }
  }
```

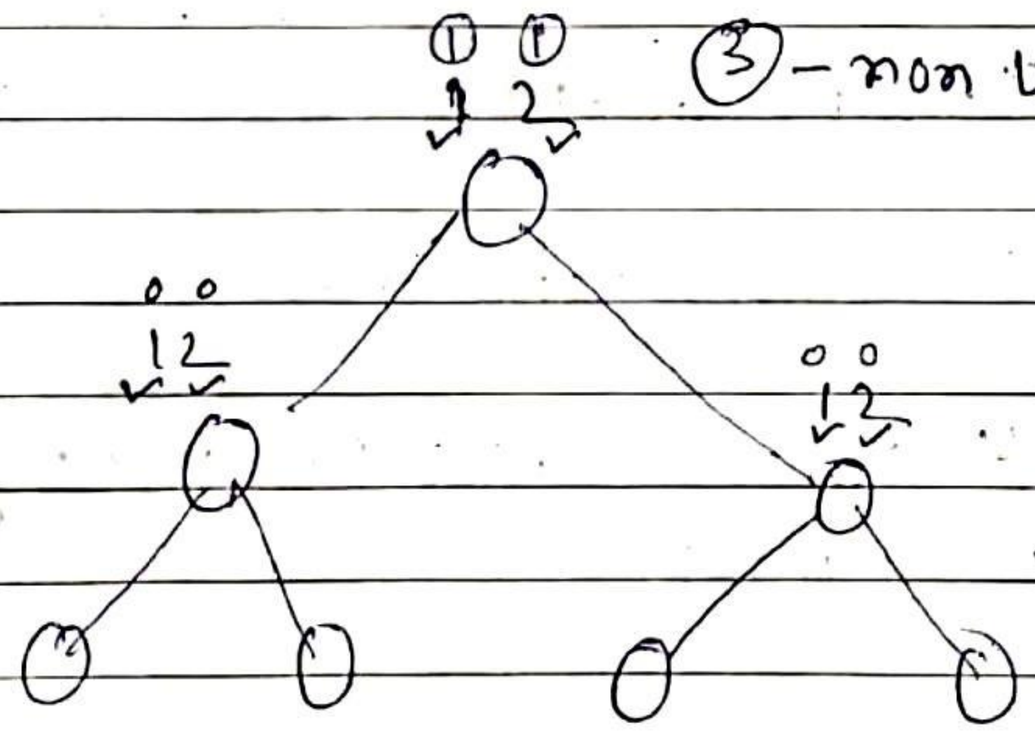


→ 3 leaves



• Count No. of non leaves :

```
struct node
{ int i;
  struct nod *left,
             *Right;
};

Int NNL(struct node *t)
{
   if (t == NULL) return 0;

   if (t->left == NULL &&
       t->Right == NULL)
       return 0;
   else
       return (1 + NNL(t->left) + NNL(t->Right);
}
```

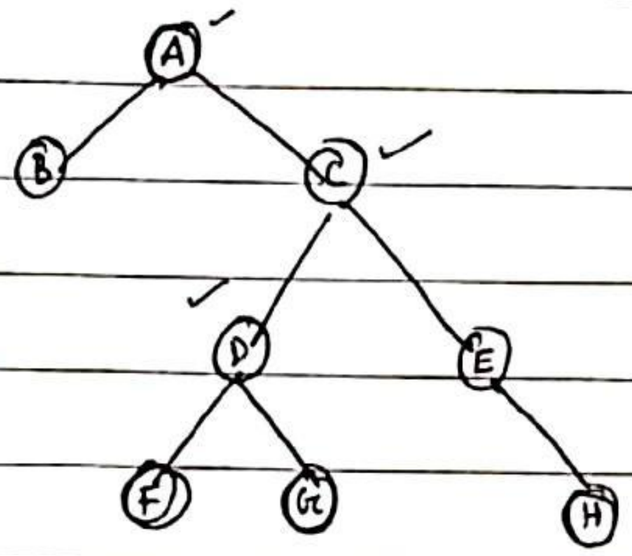


⇝ Total time and space complexity = O(n).

→ Space complexity = no. of levels of tree.
= 3 = O(n).

classmate
Date
Page

- Recursive program to find the full nodes =

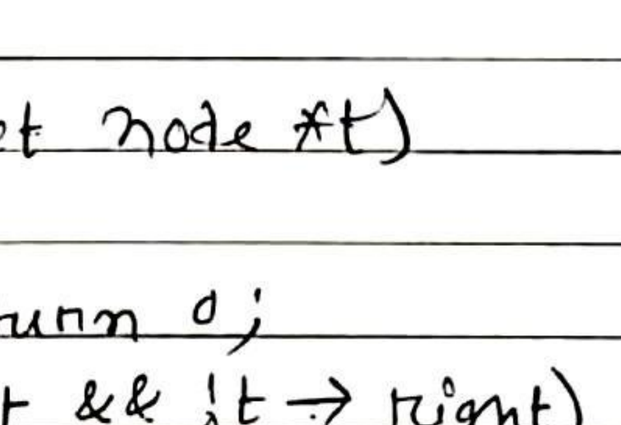


Total Full node : 3 (A,C,D)
Total node : 4 (A, C, D, E).

FN(T) = 0 ; T = NULL. (return)
= 0 ; T is a leaf.
= FN(T→LST) + FN(T→RgsT) ; if T has only one child.
= FN(T→LST) + FN(T→RST) + 1 ; if ~~kot~~ T is a full node.

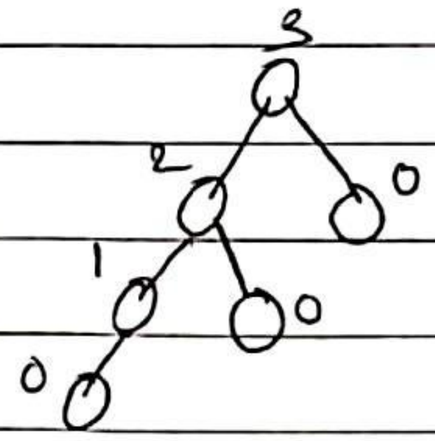
Program :

```
int FN (struct node *t)
{
   if (!t) return 0;
   if( !t→left && !t→right)
      return 0;
   return ( FN(t→left) + FN(t→right) + (t→left && t→right)
}                ①          ②              ③ ? 1 : 0;
```



→ Total Time && space complexity is = O(n).

- Recursive program to find the height of a tree :-

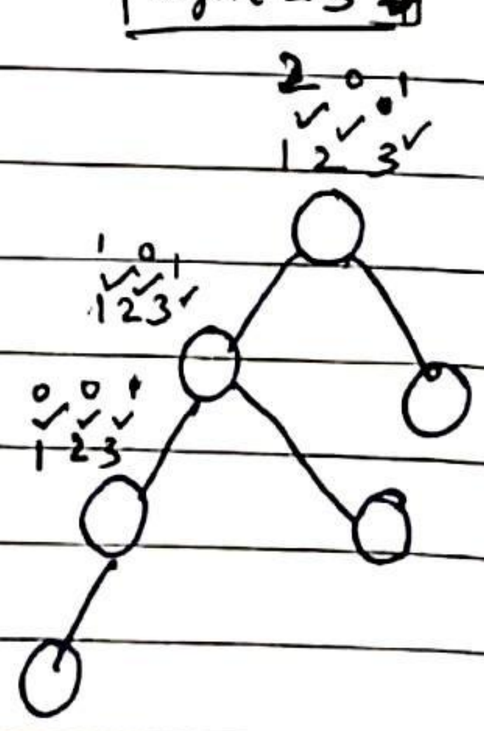


Recursive equation, $H(T) = \begin{cases} 0, & T \text{ is empty.} \\ 0, & T \text{ is leaf.} \\ 1 + \max(H(LST), H(RST)); & \text{Otherwise.} \end{cases}$

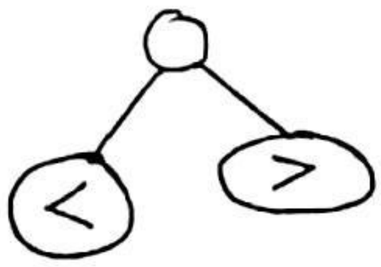
Program:

```
struct node
{ int i;
  struct node * left;
              * Right;
};

int H(struct node *t) {
    int l, r;
    if (!t) return 0;

    if ( (!(t->left))&&(!(t->Right)) )
      return 0;

    else
      1. l = H(t->left)
      2. r = H(t->Right)
      3. return (1+((l>r) ? l : r));
}
```

Hight = 3



→ Time and space complexity is = O(n).

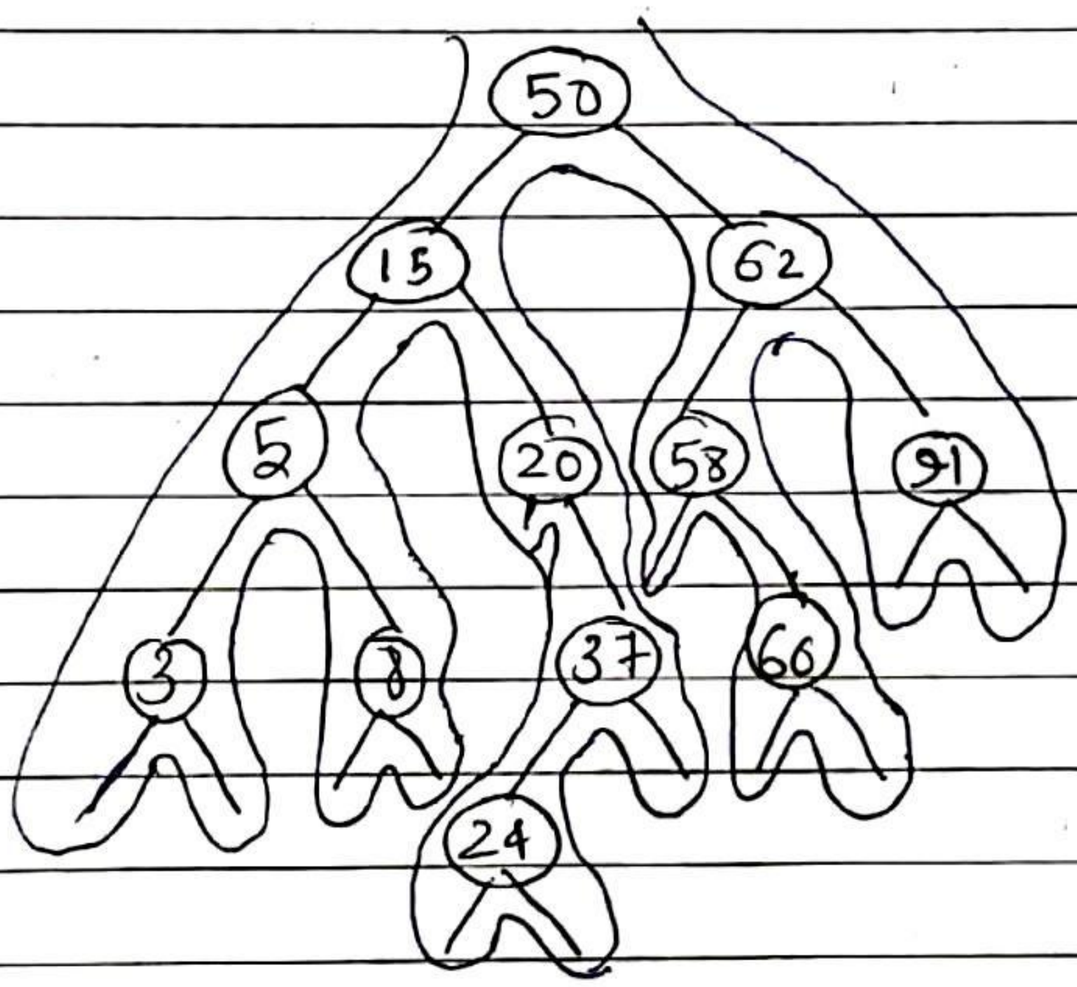


- **Introduction ~~of~~ to BST (Binary search Tree) =**

→ Binary search tree are a special kind of Binary tree. → Inorder is sorted order. (adv).

g-1996 [Question] - (1)

50, 15, 62, 5, 20, 58, 91, 3, 8, 37, 60, 24



→ The Inorder traversal of a BST is going to give sorted order of element.

Inorder : 3, 5, 8, 15, 20, 24, 37, 50, 58, 60, 62, 91.
(2nd time) (assending order)

g-2005 [Question] - (2)

Post Order : 10, 9, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29.

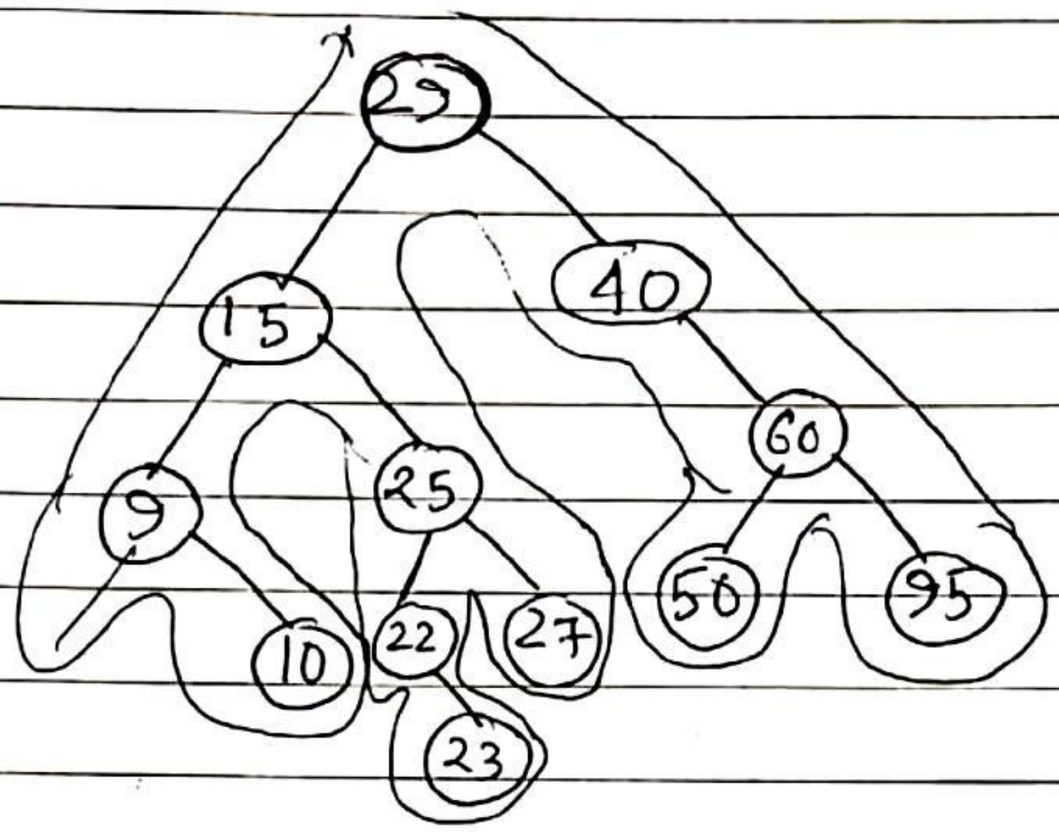
Inorder : ?

→ Inorder : 9, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 95.
(assending Order)

then find ~~post~~ preorder : ?

9 10 (15) 22 23 25 27 (29) (40) 50 60 95

Root



Pre order : 29, 15, 9, 10, 25, 22, 23, 27, 40, 60, 50, 95

g.-2005 Question-(3)

How many BSTs are possible with 4 distinct keys.
value

→ No. of BST possible with n distinct keys are,
→ (same as Binary Unlabelled tree).

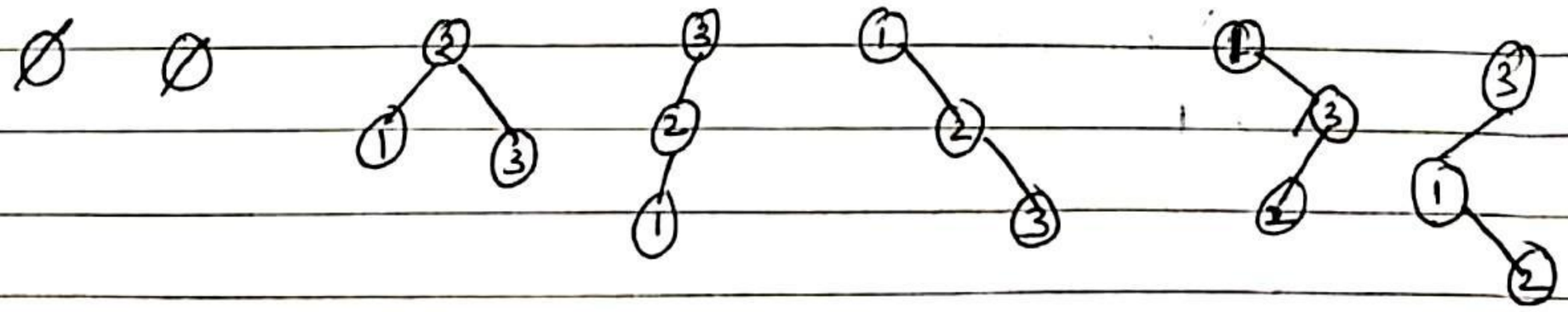
$\frac{2n C_n}{n+1}$

$$\frac{8C_4}{4+1} = \frac{|8}{|4 \cdot |4 \cdot 5}$$

$$= \frac{8*7*6*5}{4*3*2*5}$$

= 14 (BST) possible.

∅ BSTs 3 distinct keys = 1 2 3



g-2006

Question - ④

(1 – 100) in BST search for 55 which of the following sequences cannot be the sequence of nodes examined ?

a) {10, 75, 64, 43, 60, 57, 55}

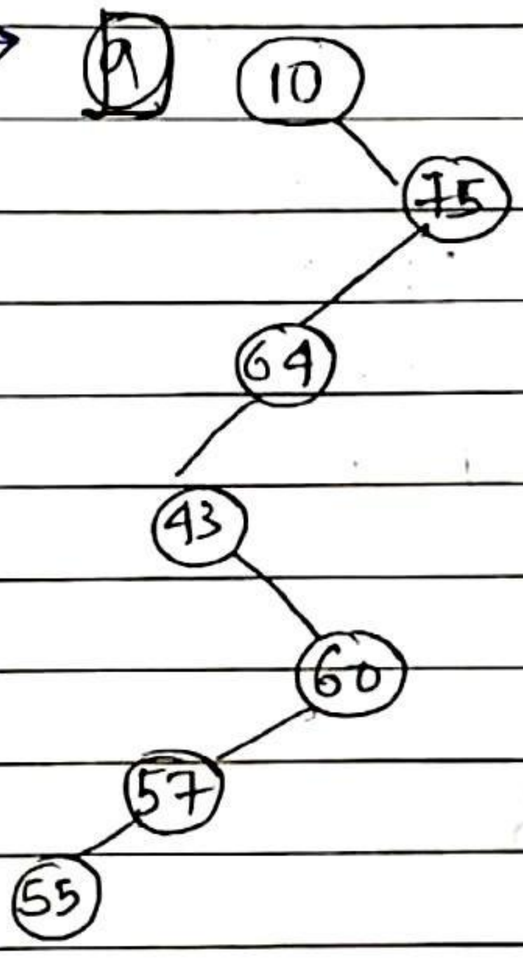
b) {90, 12, 68, 34, 62, 45, 55}

c) {9, 85, 47, 68, 43, 57, 55}

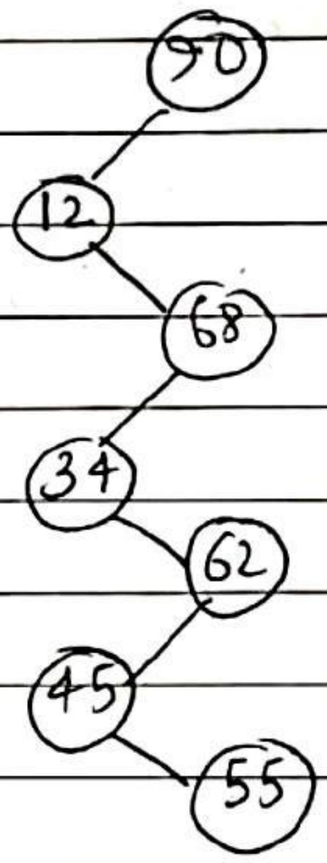
d) {79, 14, 72, 56, 16, 53, 55}

Use Inorder traversal ascending order

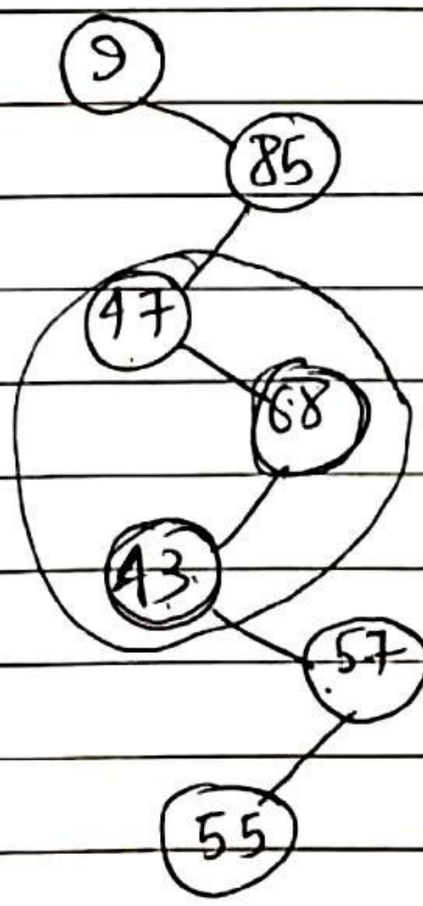
→ [a]



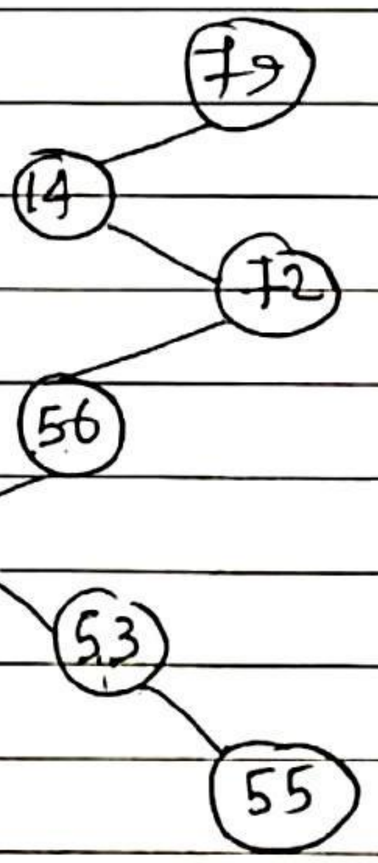
[b]



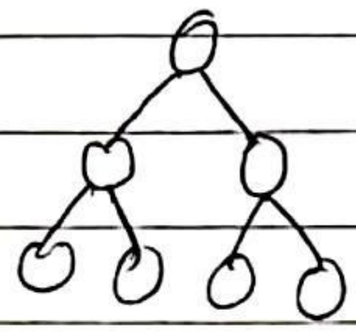
✗ [c]



[d]

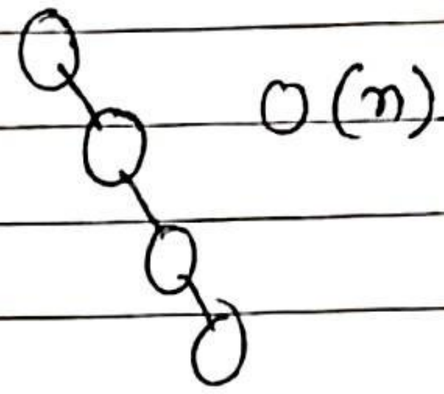


→ Time compurity of BST :



O(log n)
or
O(h)

h → hight.



O(n)

gate-2008

Question - (5)

A BST stores values in the range 37 to 573. Considere the following sequence of keys —

× a) 81, 537, 102, 439, 285, 376, 305.

× b) 52, 97, 121, 195, 242, 381, 472.

✓ c) 142, 248, 520, 386, 345, 270, 307.

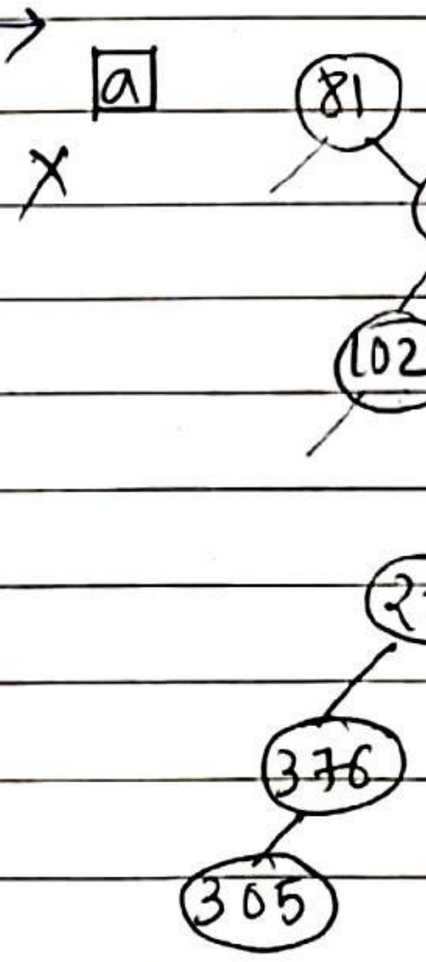
× d) 550, 149, 507, 395, 463, 402, 270.

search

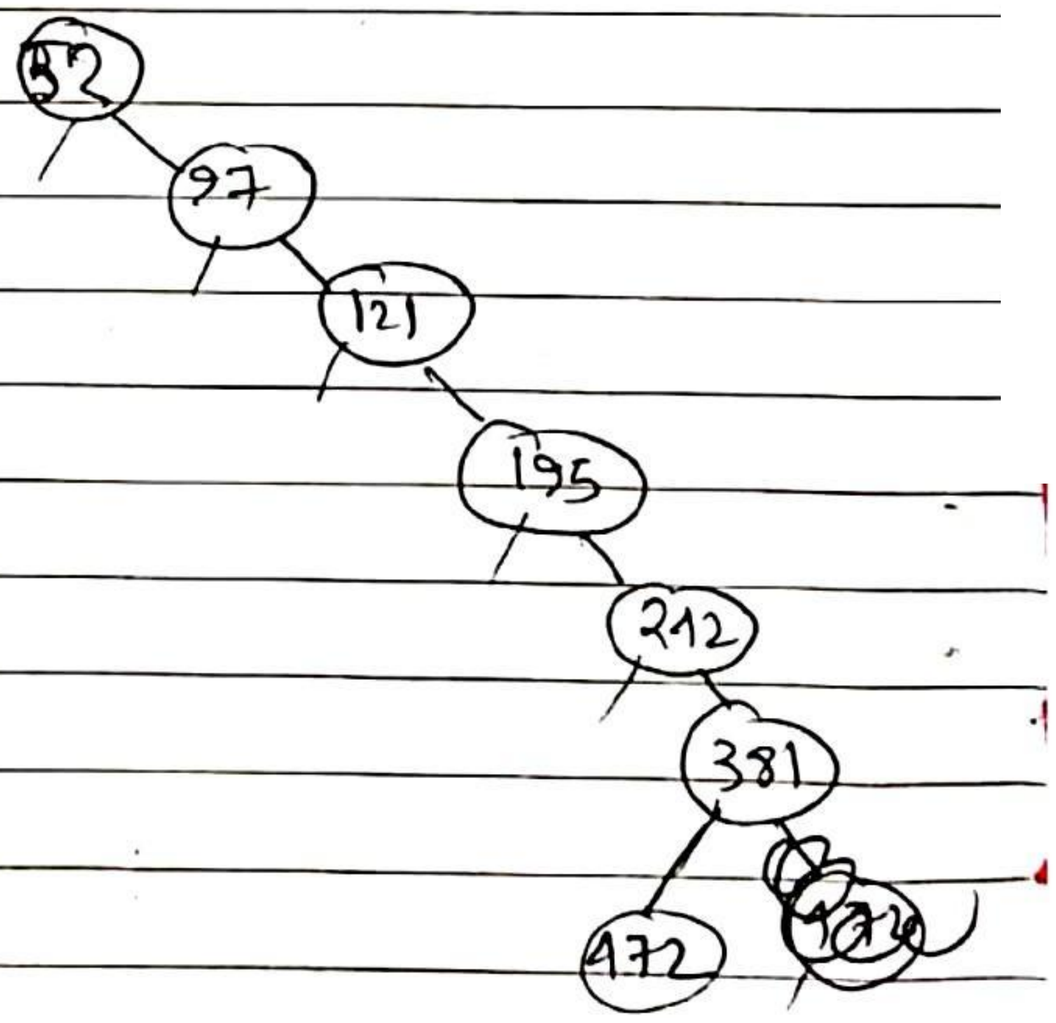
273 → Unsuccessful

→

a

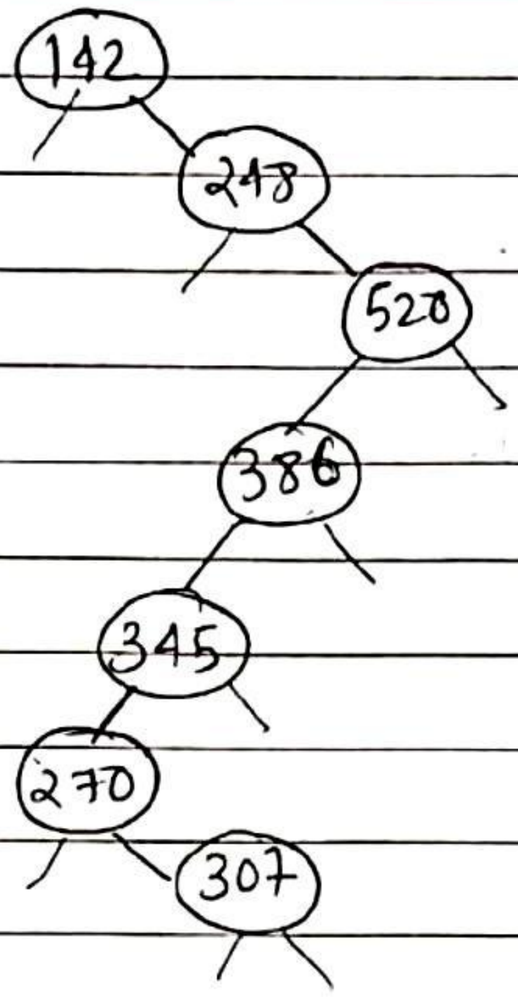


b

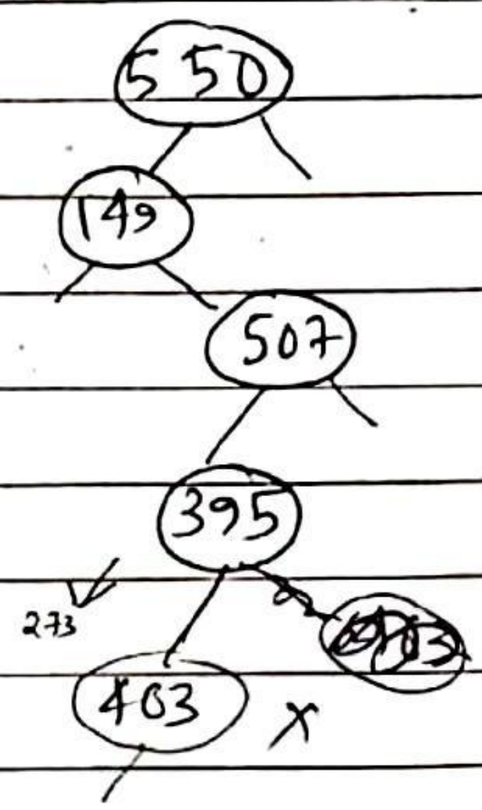


273

[C] ✓



[A] X



g- 2008

Question - (6)

A BST stores values in the range 37 to 573 consider the following sequence of key -

X I) 81, 537, 102, 439, 285, 376, 305.

II) 52, 97, 121, 195, 242, 381, 472.

III) 142, 248, 520, 386, 345, 270, 307.

X IV) 550, 149, 507, 395, 463, 402, 270.

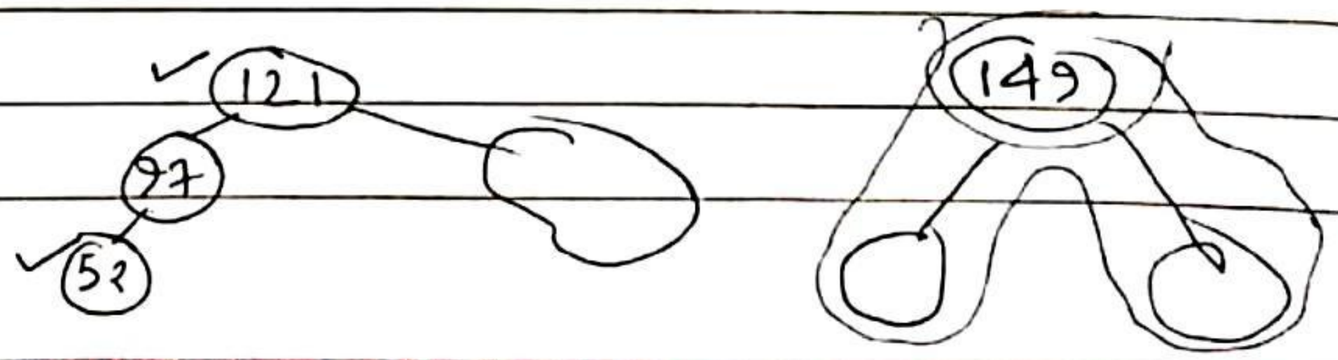
Which of the following is true ?

X (a) I, II and IV are Preorder sequences of 3 diff BSTs.

X (b) I is preorder sequence of some BST with 439 as the root.

✓ (c) II is an inorder of some BST with 121 as root and 52 as leaf.

X (d) IV is a postorder sequence of some BST with 149 root.



g-2008

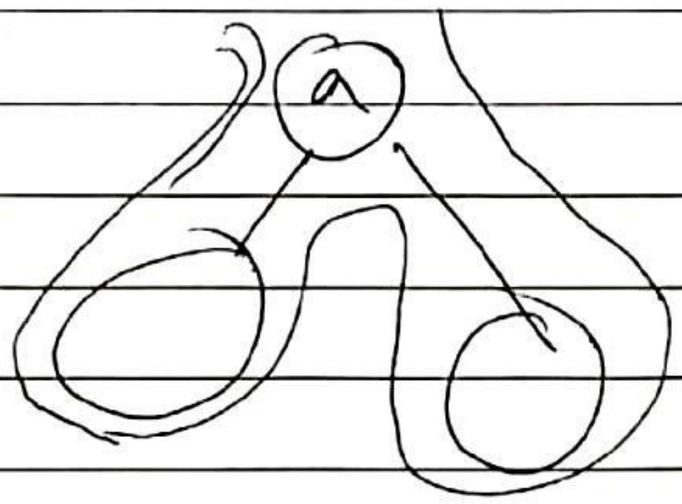
Question – (7)

I: MBCAFHPYK. — Post

✓II: KAMCBYPFH. — Pre

III: MABCKYFPH. — In

Which one is what order?.



pre: A ——

post: —— A

In: L Root R

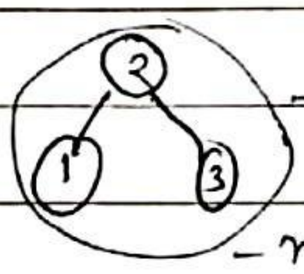
pre: Root L R

g-2011

Question – 8

We are given set of n distinct elements and an unlabeled binary tree with 'n' nodes. In how many ways can we populate the tree with the given set so that it becomes a BST =

a) 0 b) 1 c) $n!$ d) $\frac{2nC_n}{n+1}$.



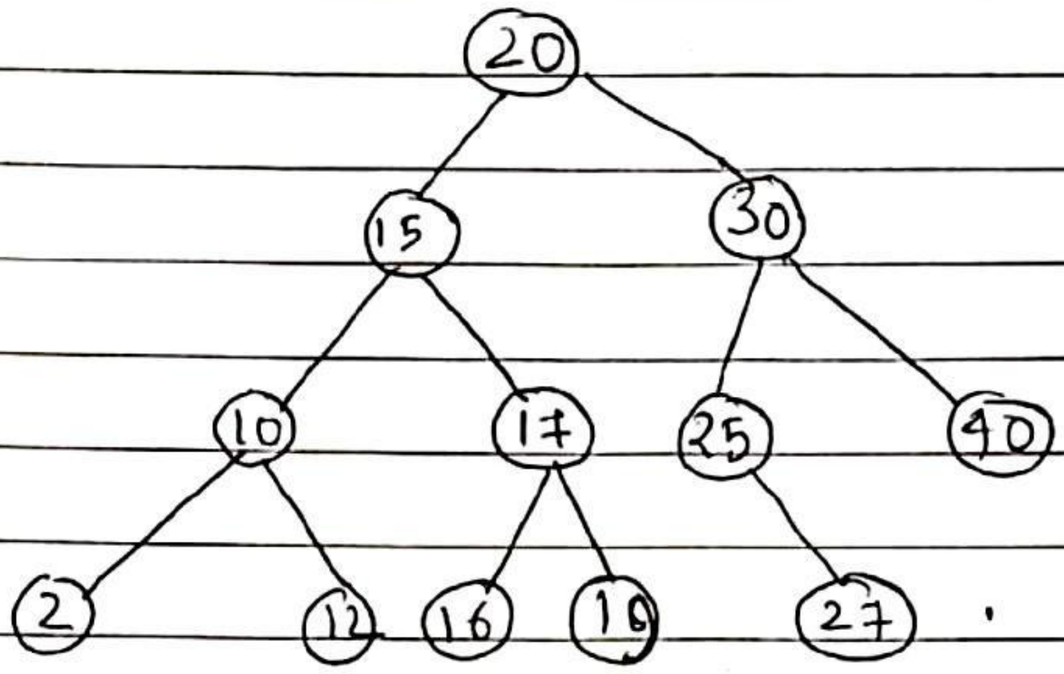
— 1, 2, 3

n-key

— n node

→ only one ways possible to becomes (BST).

- Deleting a node from BST (Binary search tree) :-

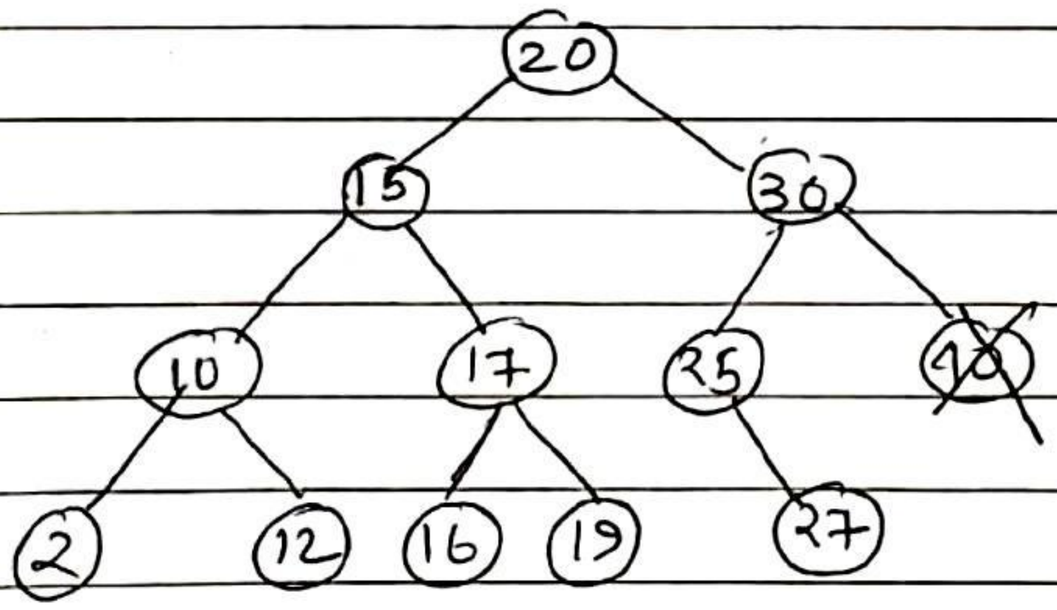


~~Fr Three types of deletion~~

Three types of element deletion possible in BST =

- leaf.
- Non-leaf
  - one child.
  - two children.

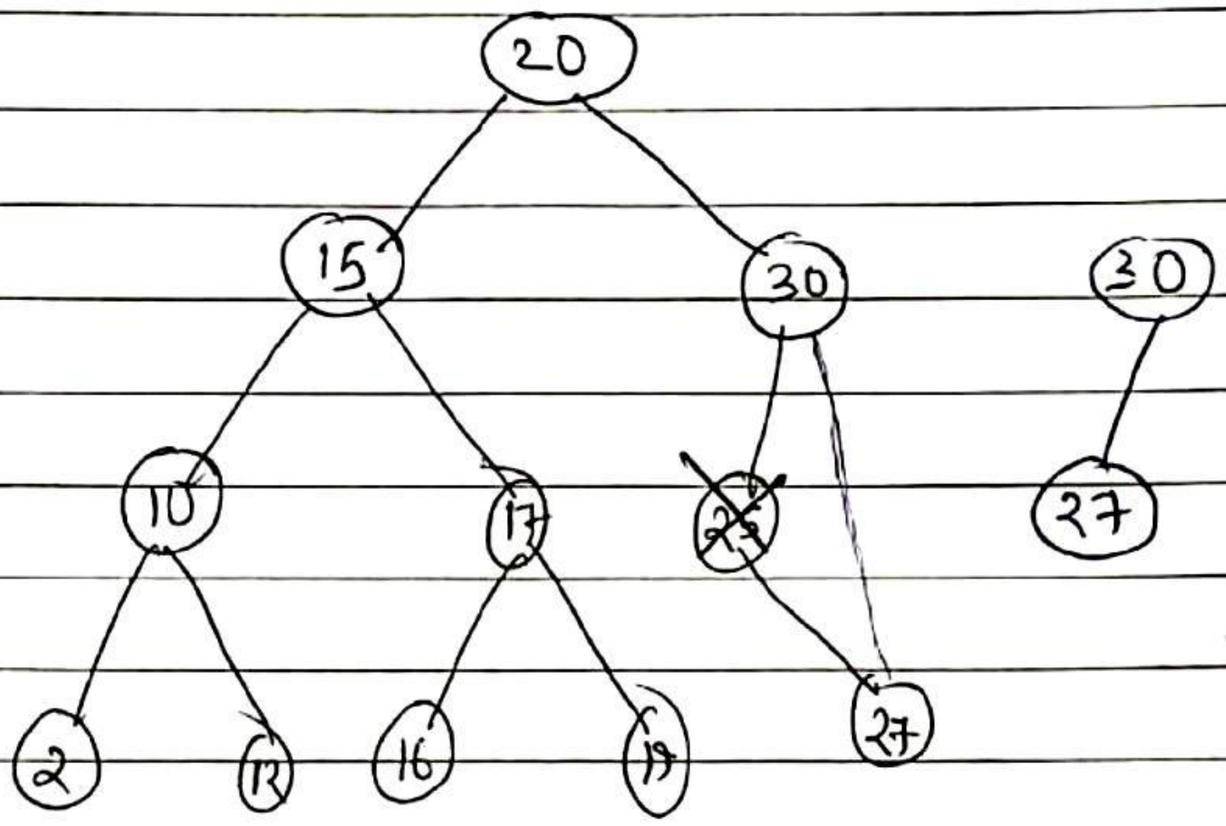
- Deleting leaf : 40



→ When deleting a leaf you did not worry about anything.

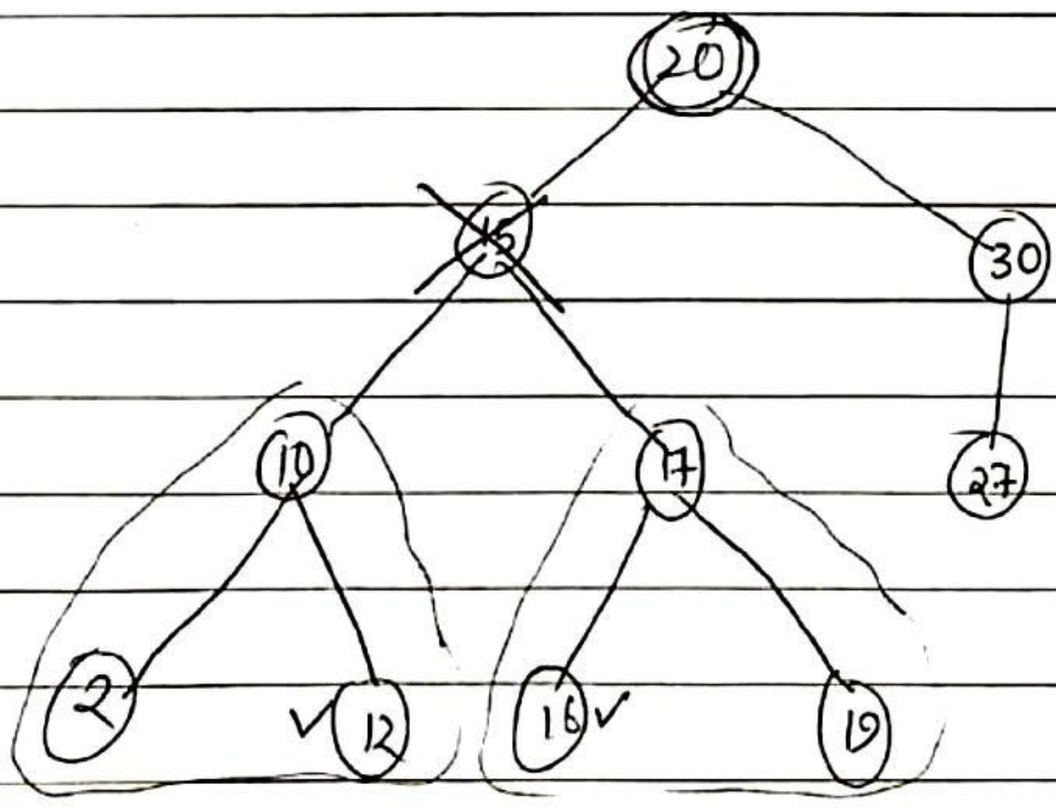
- Deleting Non-leaf :

(i) having one child : (35)



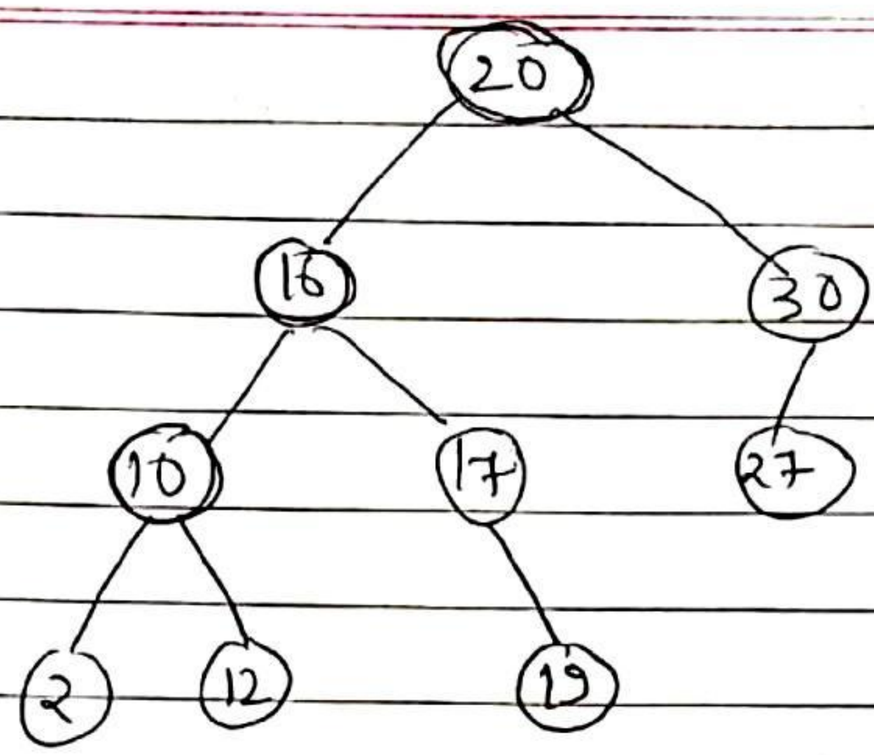
→ Whenever a node have one child and you are going to delete the node then make that child point to the grand-parent.

(ii) having two child : (15)

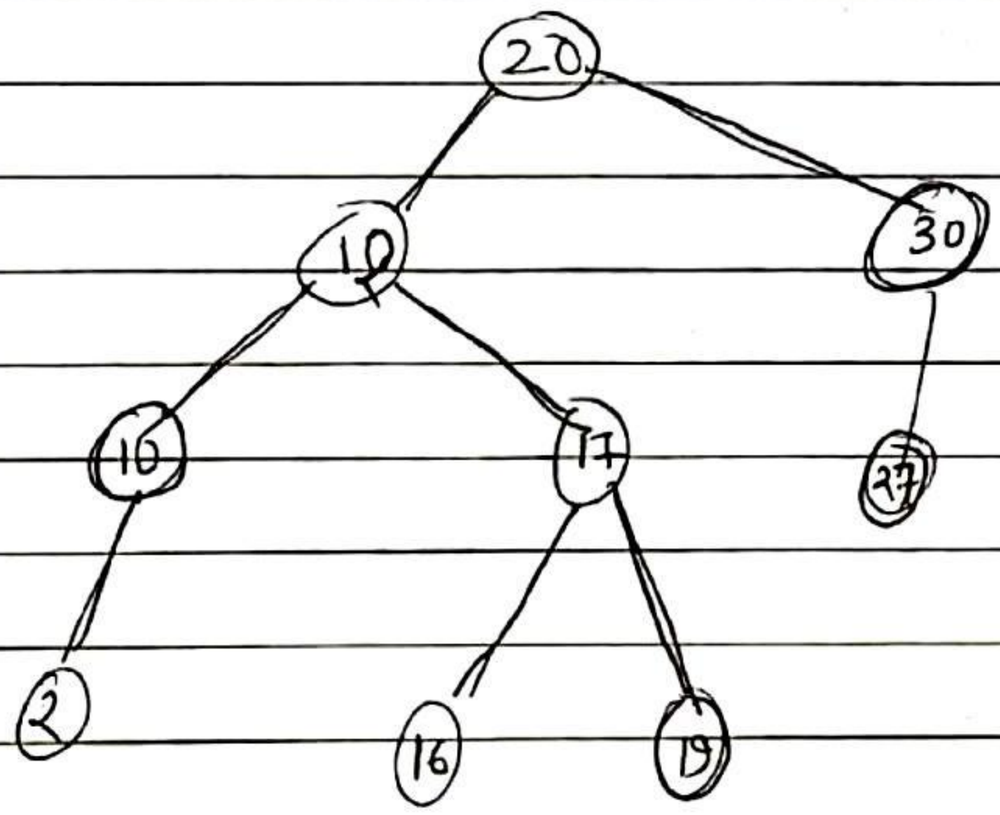


here 2-method :

→ go to the Right sub tree take the ~~list~~ least element and then put in place of deleted node.



→ other method is, go to the Left Subtree then take the largest ~~node~~ element and the put it in place of deleted node.



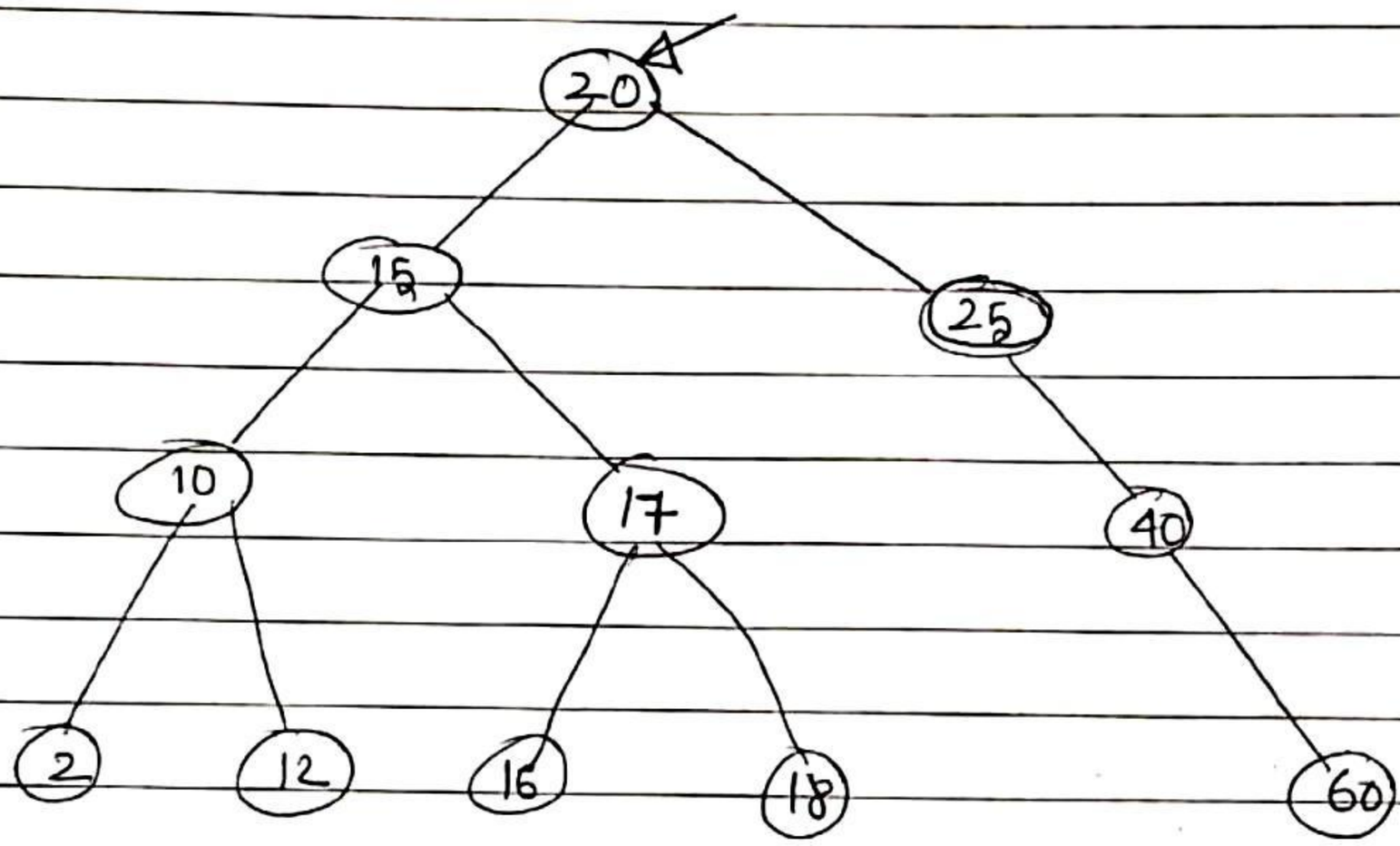
→ By doing deletion never increase the hight of the tree.

→ By doing ~~deletion~~ insertion hight of the tree might increase.

→ Inorder Sucessor: ~~Greatest~~ least ~~Grates~~ element in Right-sub-tree

→ Inorder predecessor: ~~least~~ grates element in Left-sub-tree.

• Finding minimum and maximum element in a BST =



• find out least Element of the BST :-
(min / inorder predecessor)

```
find-inord-pre (struct node *t)
 - min
{
    if(t==NULL) return 0;

    while(t->left)
    {
      t = t->left;
    }
```

• find out greates element of the BST :-

```
find-max (struct node *t).
{
    if (!t) return 0
    while (t->right)
    {
      t = t->right;
    }
}
```

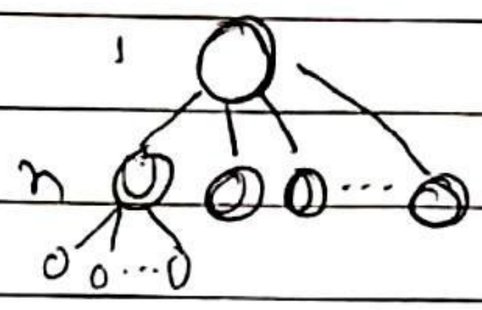
g-1998

Question -①

A complete n-ary tree is one in which every node has '0' or 'n' sons. If 'x' is the number of internal nodes of complete n-ary tree, the number of leaves in it is given by —

a) $x(n-1)+1$ b) $xn-1$ c) $xn+1$ d) $x(n+1)$.

→



Internal
1^node — n leaves.

2 node — $(n-1)+n$

Internal.
3^node — $(n-2)+n+n = 3n-2$

4 node — $(n-3)+n+n+n = 4n-3$

⋮

x node — $xn-(x-1)$

$= xn - x + 1$

$= x(n-1)+1$ (leaves)

g-2002

Question -②

The number of leaf nodes in a rooted tree of 'n' nodes, with each node having 0 or 3 children is —

a) $n/2$ b) $(n-1)/3$ c) $(n-1)/2$ d) $(2n+1)/3$.

→ I → internal node.
n → total node.
l → leaf.

$l+I = n$ ... (i) $l = I(3-1)+1$ ... (ii)

$$2I - L = -1$$

$$I + L = n$$

$$3I = n - 1$$

$$I = (n-1)/3$$

$$L + I = n$$

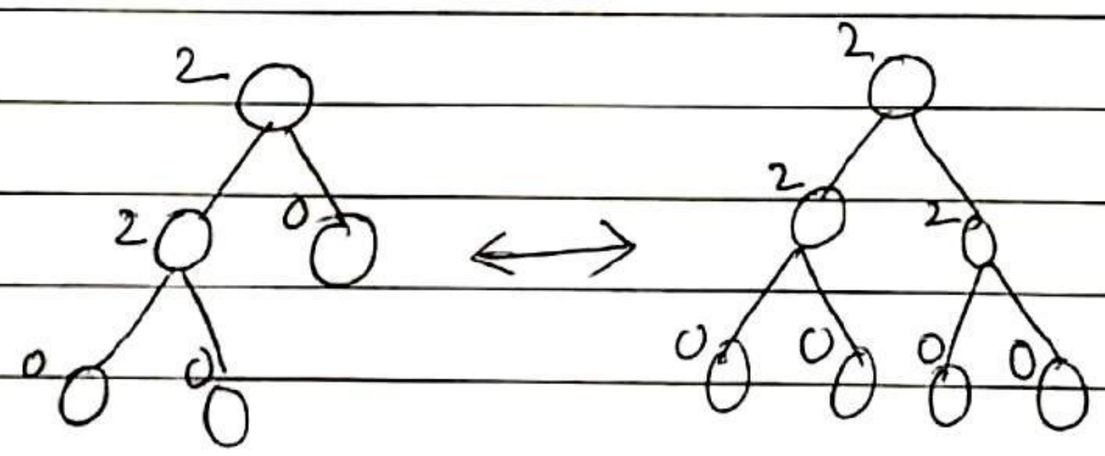
$$L + (n-1)/3 = n$$

$$L = n - \frac{n-1}{3}$$

$$= \frac{3n - n + 1}{3} \Rightarrow \left(\frac{2n+1}{3}\right)$$

- Recursive program on testing whether a tree is complete binary tree.

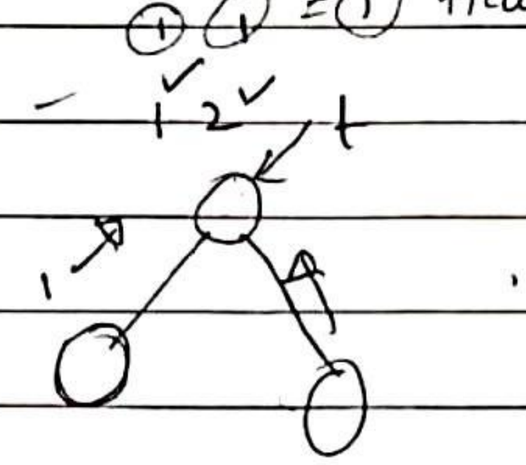
→

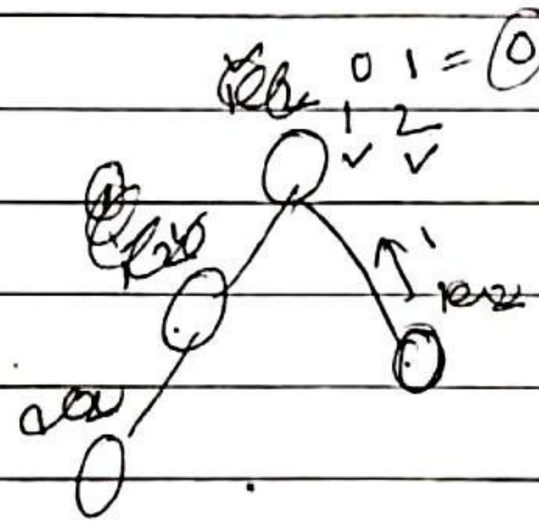


both are complete binary tree.

```
int iscomplete (struct node *t)
    {
        if (t == NULL) return 0;

    if ((!(t->left)) && (!(t->right)))
        return 1;

    else if (t->left && t->right)
        return (iscomplete (t->left) && iscomplete (t->right));
    }
```

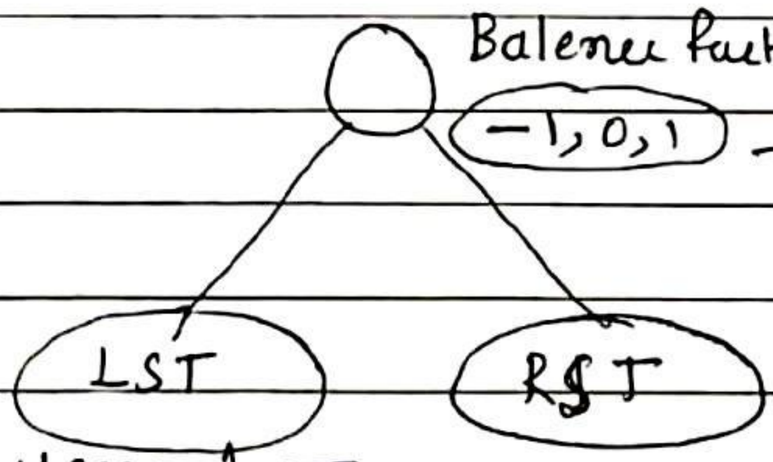
else
return 0;
}



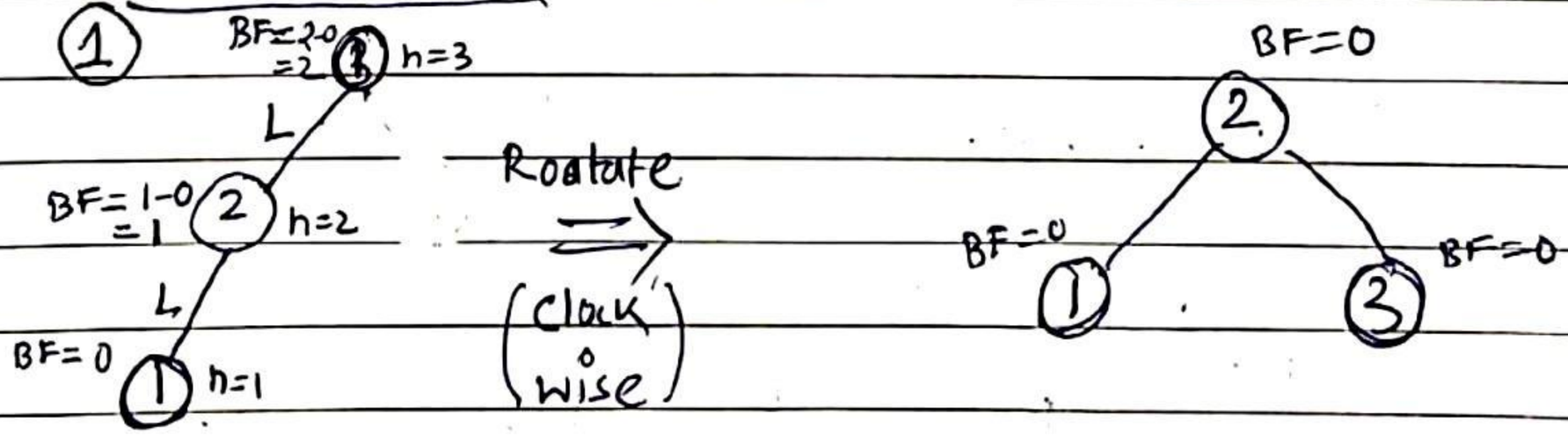


- Introduction to AVL tree =

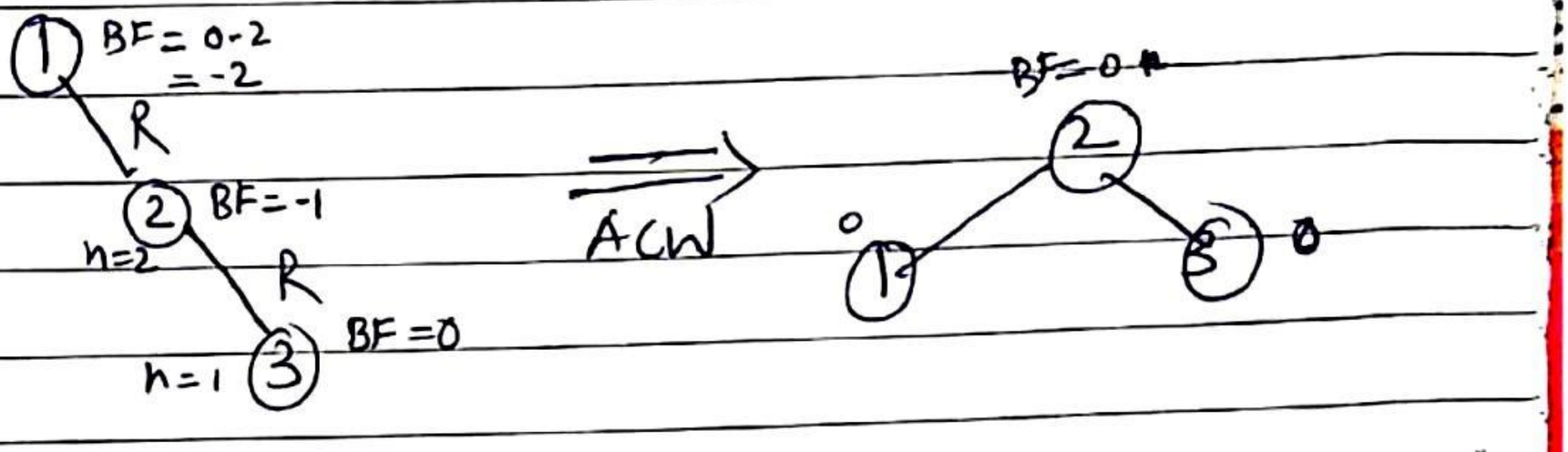
AVL is an ~~Blanding~~ balencing search tree.



LL - Imbalenced =

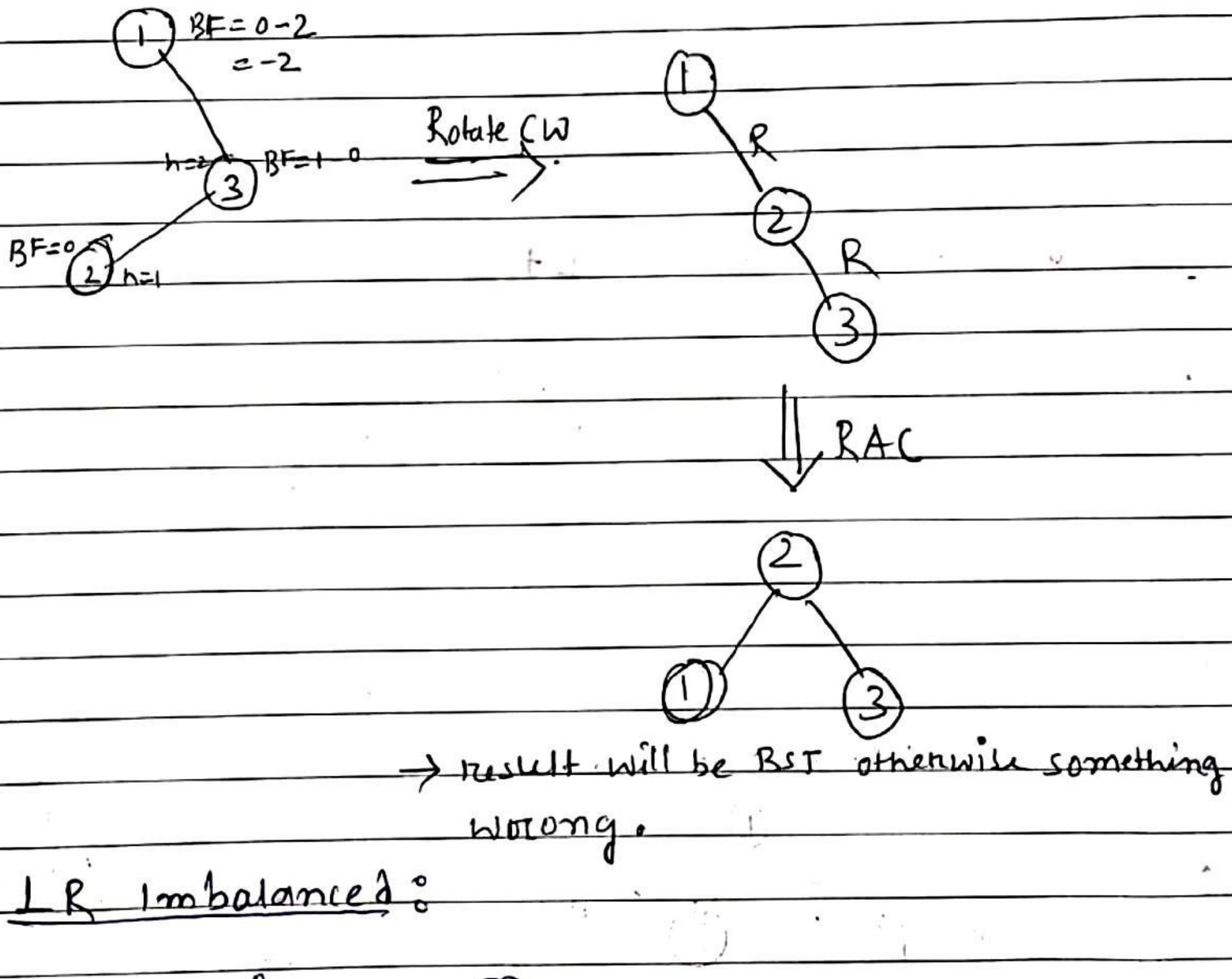


(2) RR - Imbalanced :



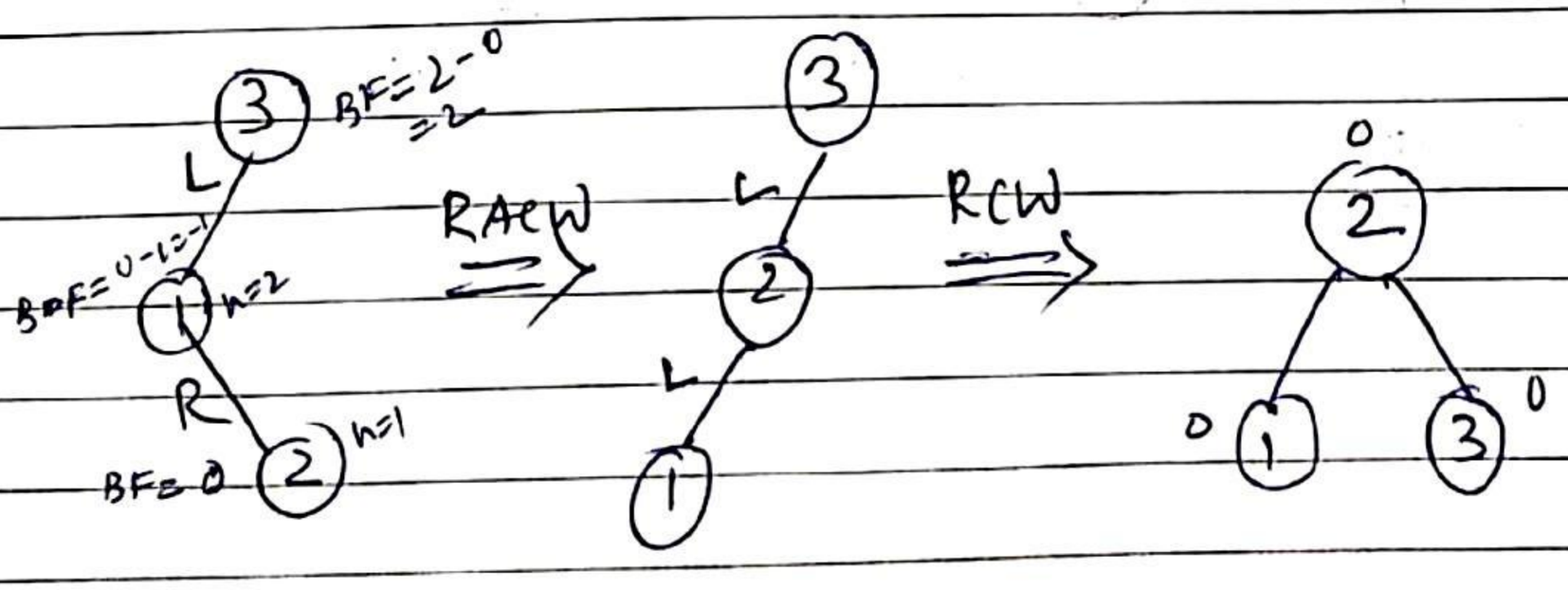
(3) Right-left (RL) Imbalanced :

BF = Balanced factor should (+,0,-)



→ result will be BST otherwise something wrong.

(4) LR Imbalanced :

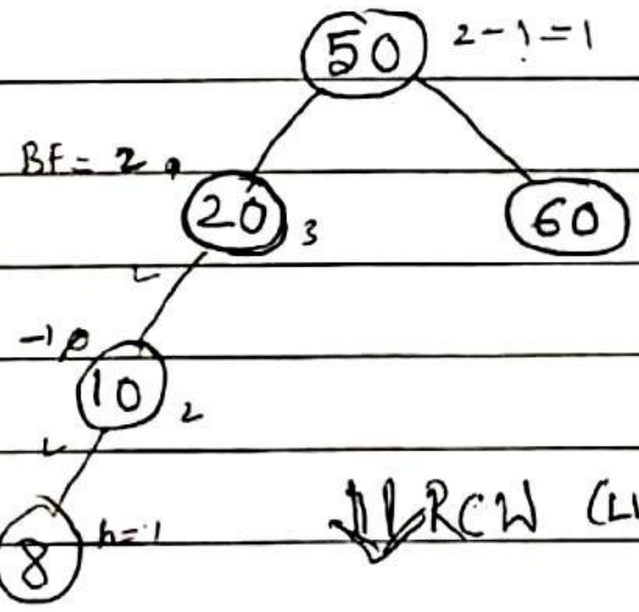


→ After inserting a newnode you should travel from newly inserted node toward root to see on the path, if there is any imbalenced.

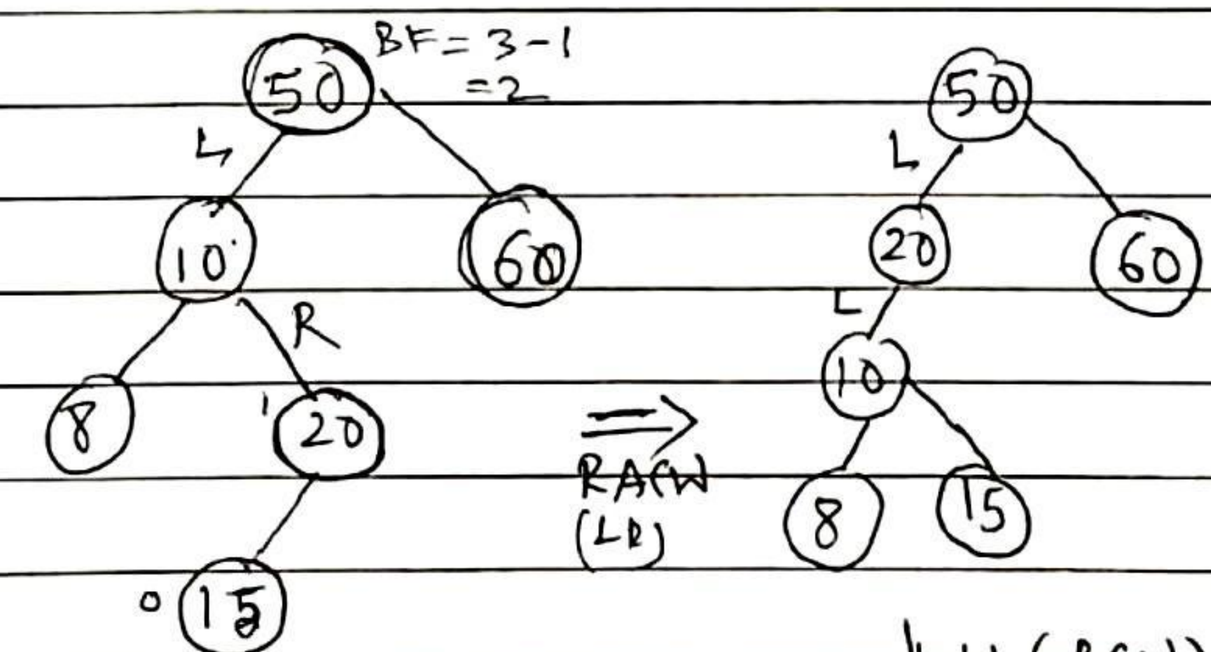
if there is any imbalenced node then start from imbalanced node and start going toward added newnode to then you understand what type of imbalenced (LL, RR, LR, RL). then rotate them accordingly.

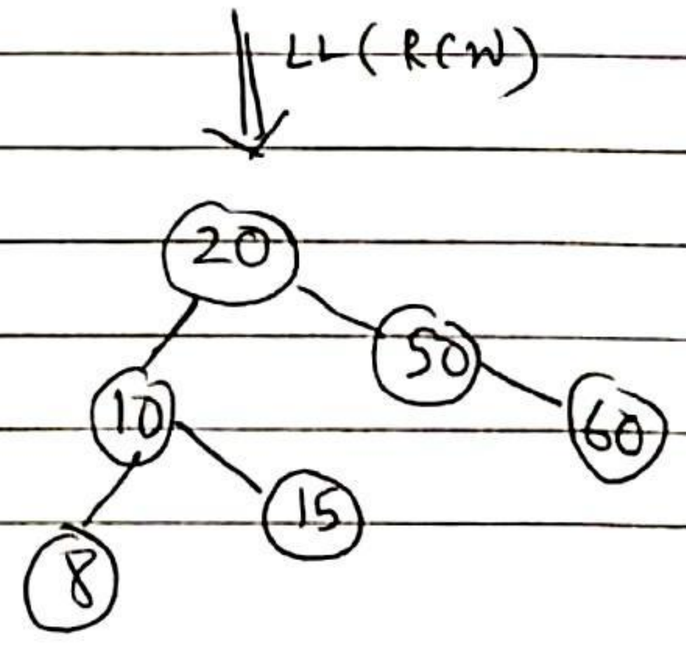
- Constructing AVL trees and time complexity analysis =
- example - (1)

50, 20, 60, 10, 8, 15, 32, 46, 11, 98.

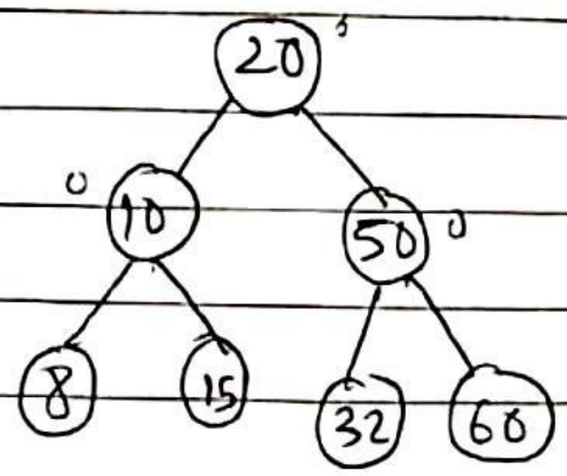


RCW (LL)

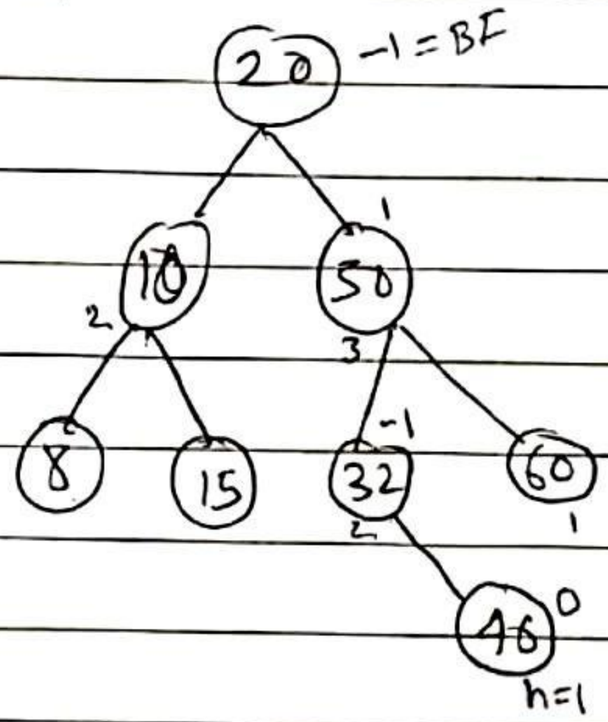




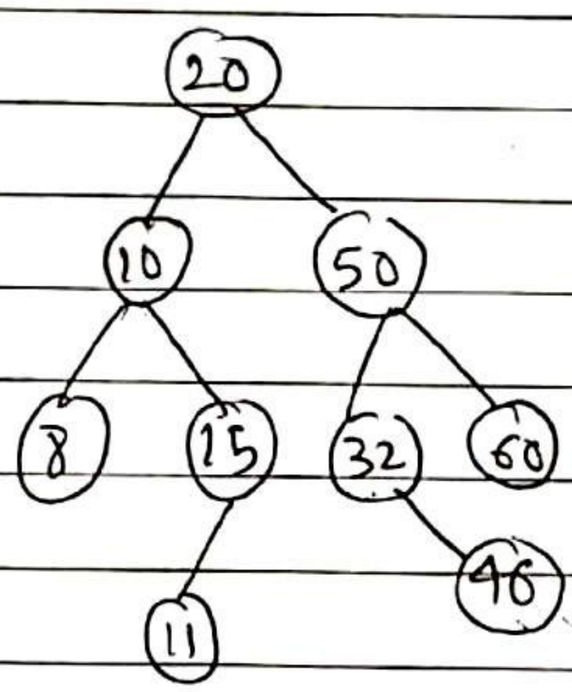
then insert 32.



Insert -46

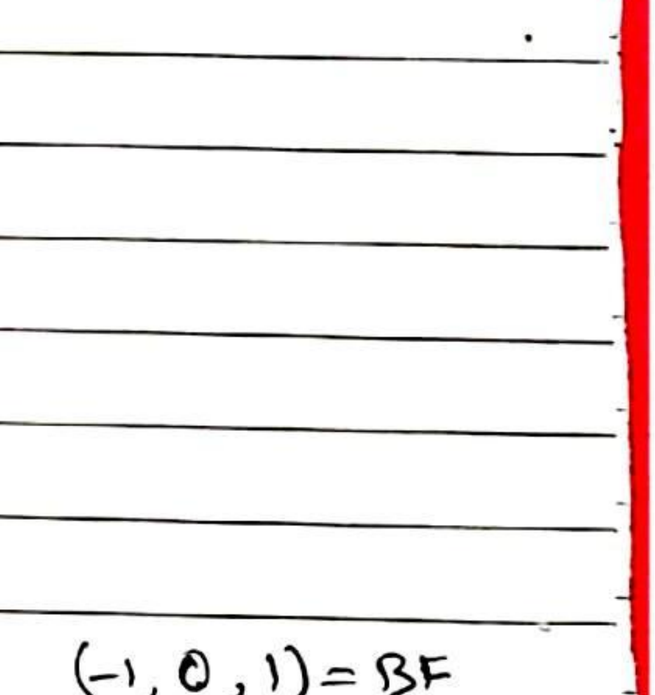


insert - 11



(-1, 0, 1) = BF

Insert - 48



↳ Balanced Binary Search tree

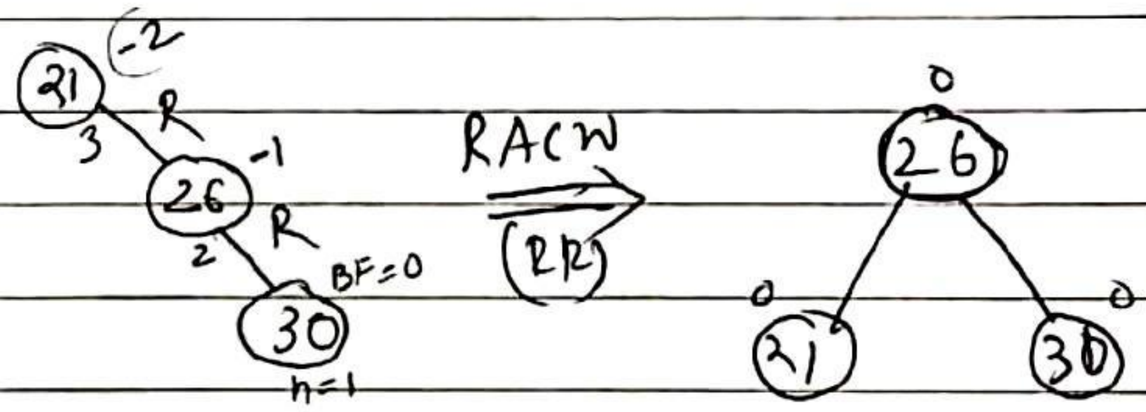
- Complexity analysis =

	BST	BBST (AVL) (balanced)
search -	O(n)	O(log n)
hight -	O(n)	O(log n)
Insertion -	O(n)	O(log n) + O(log n) + C => O(log n)

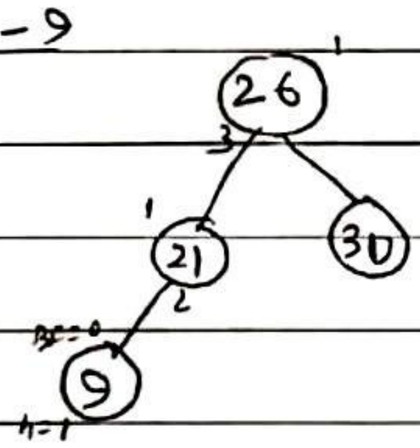
- Construction of AVL tree =
  - Example-②

21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7.

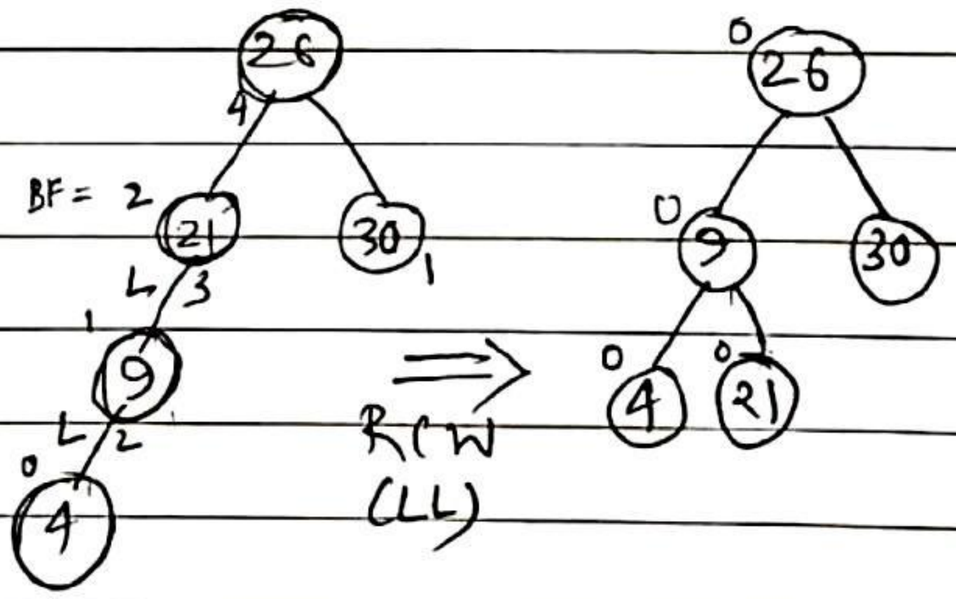
→



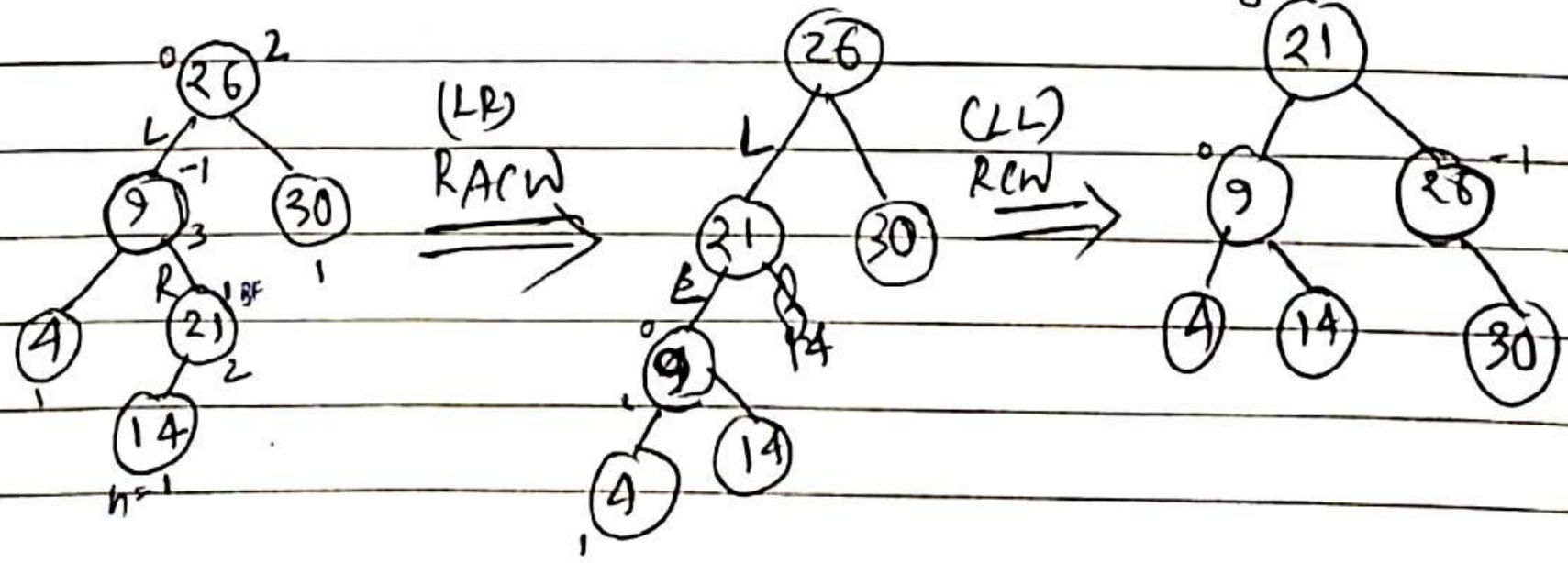
Insert - 9



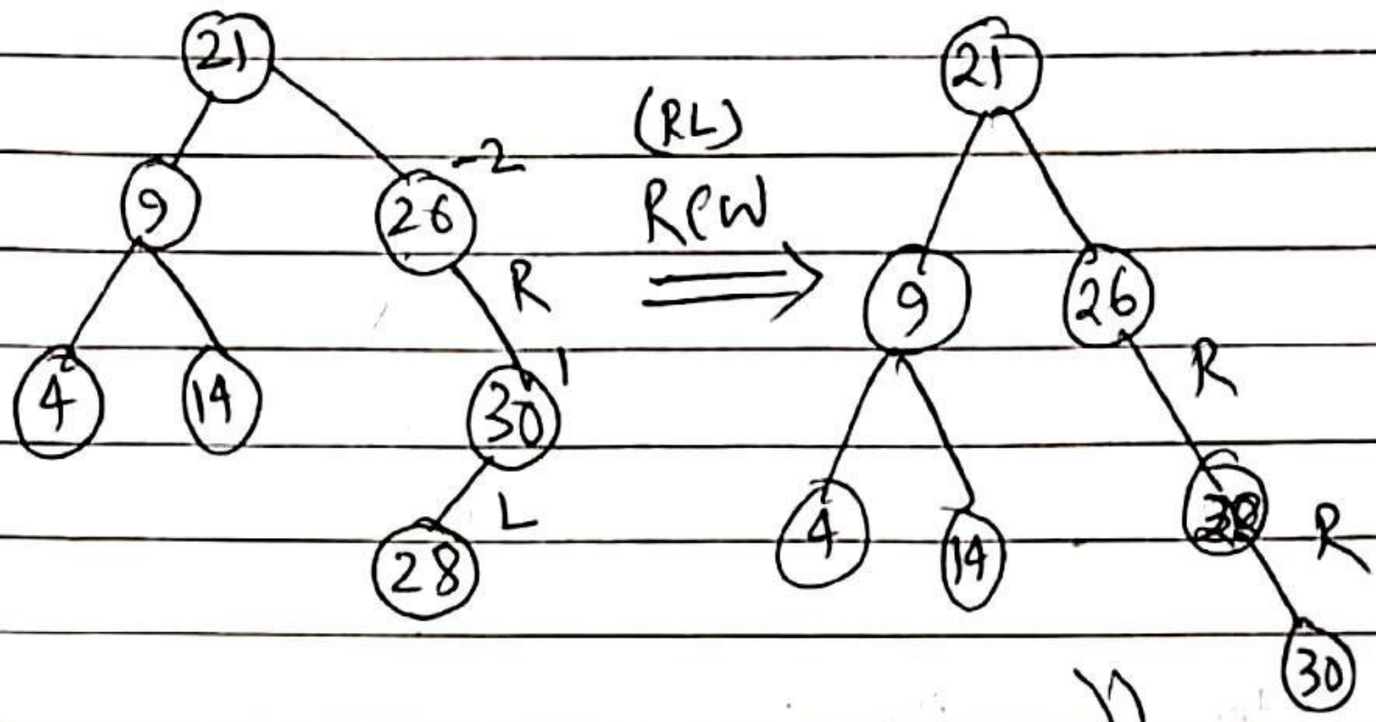
Insert - 4



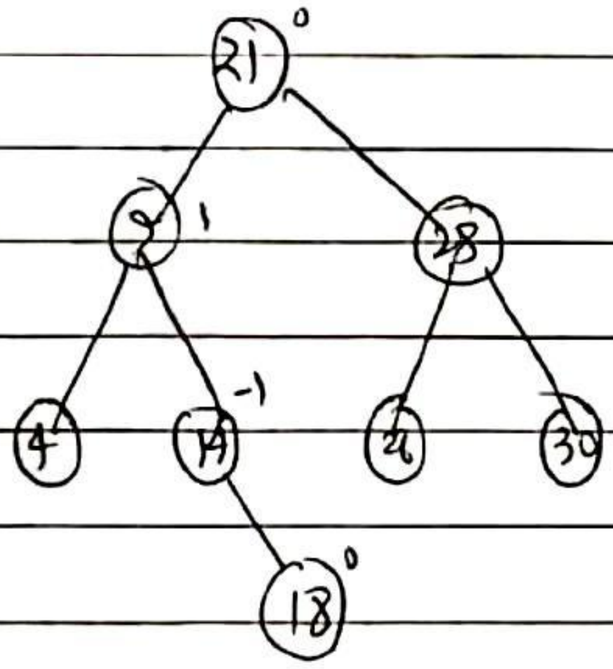
Insert - 14



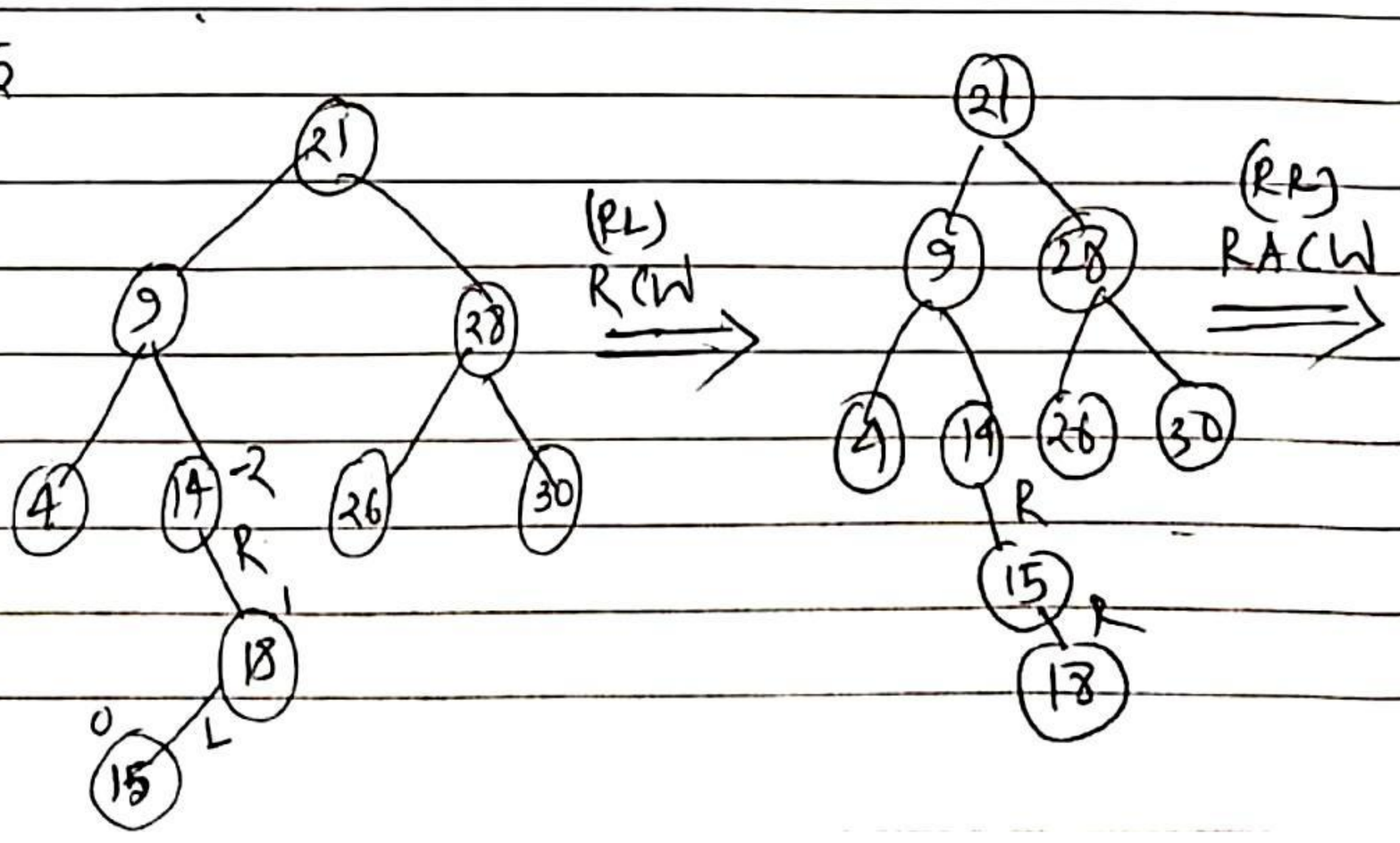
Insert - 28

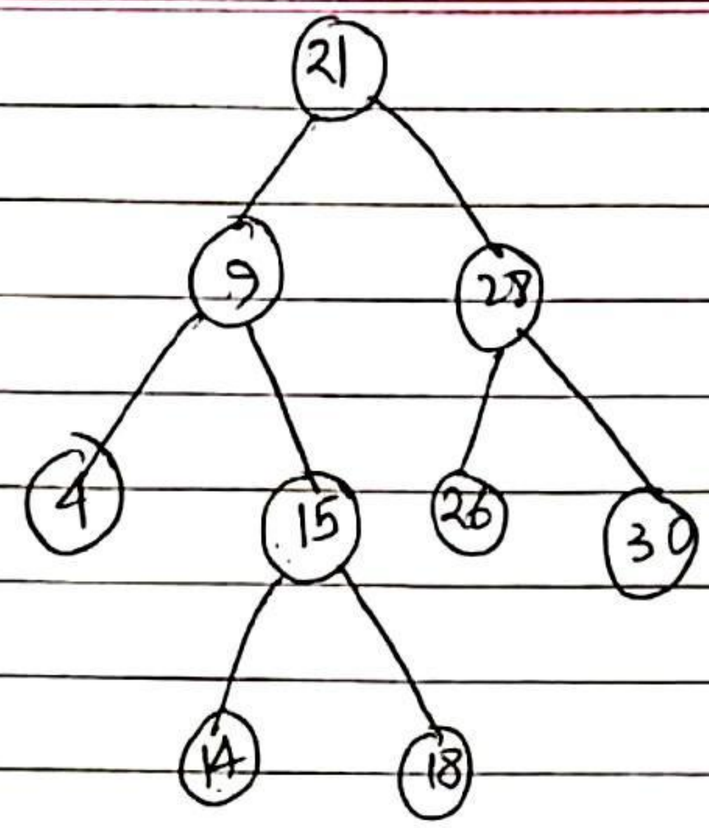


then, Insert - 18

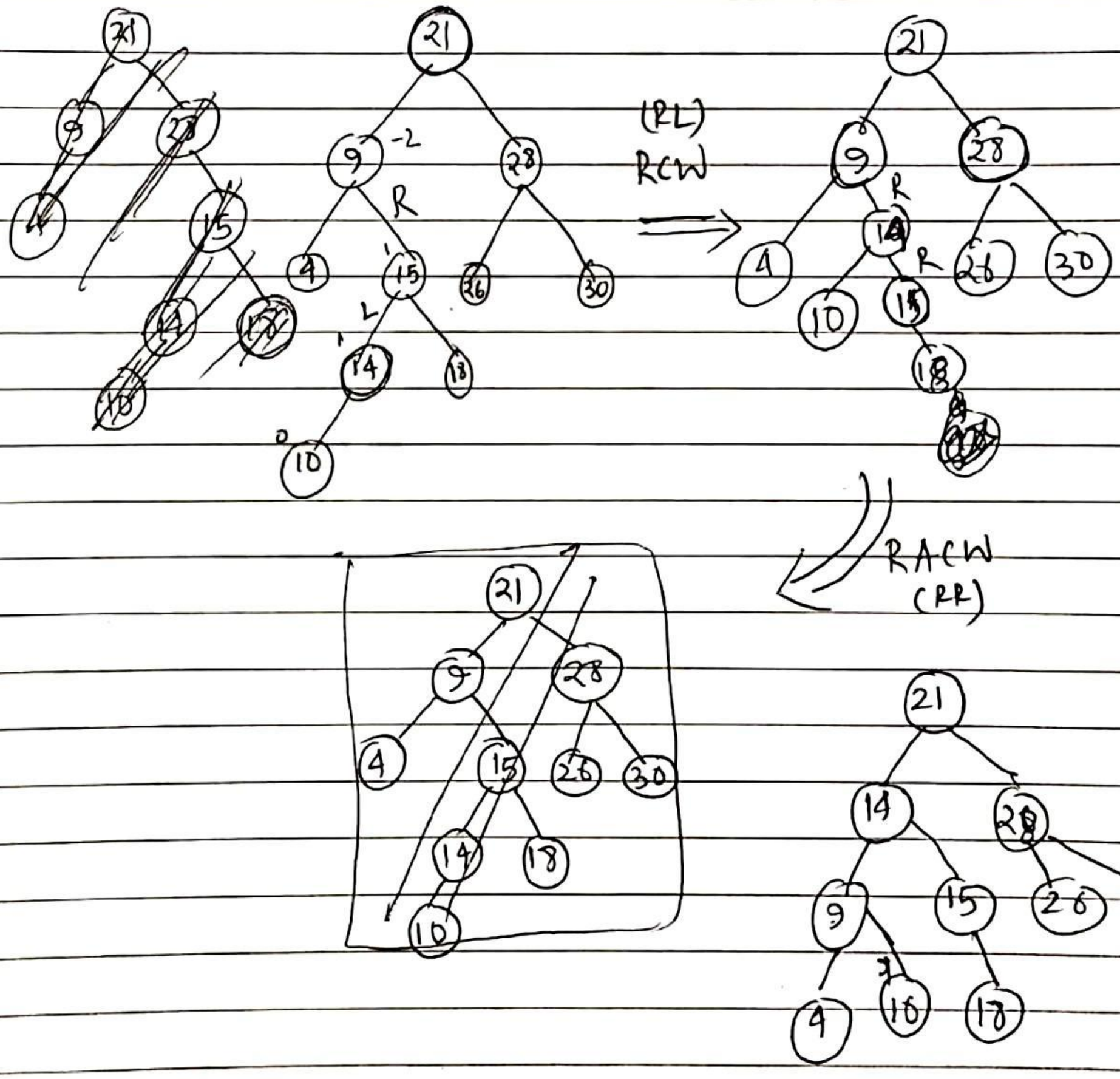


Insert - 15

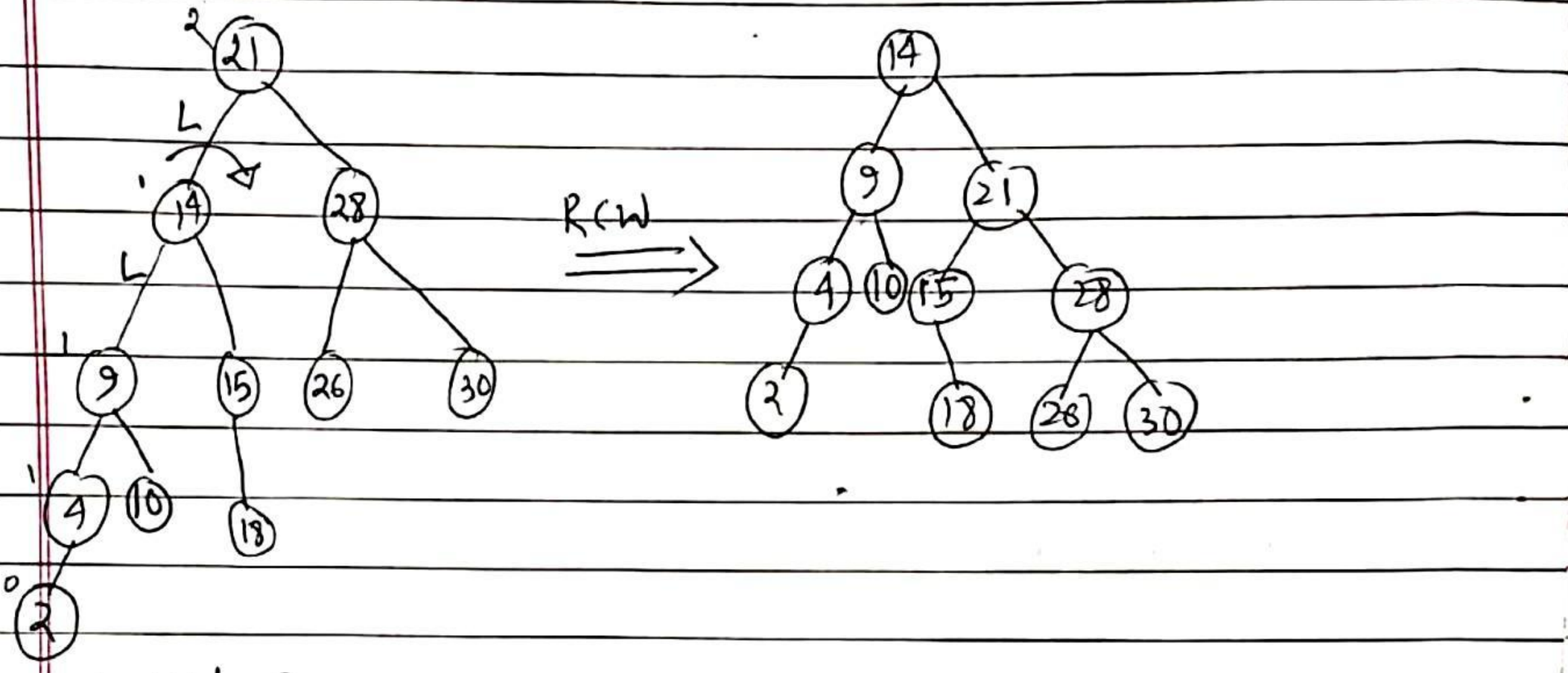



21
9
28
4
15
26
30
14
18

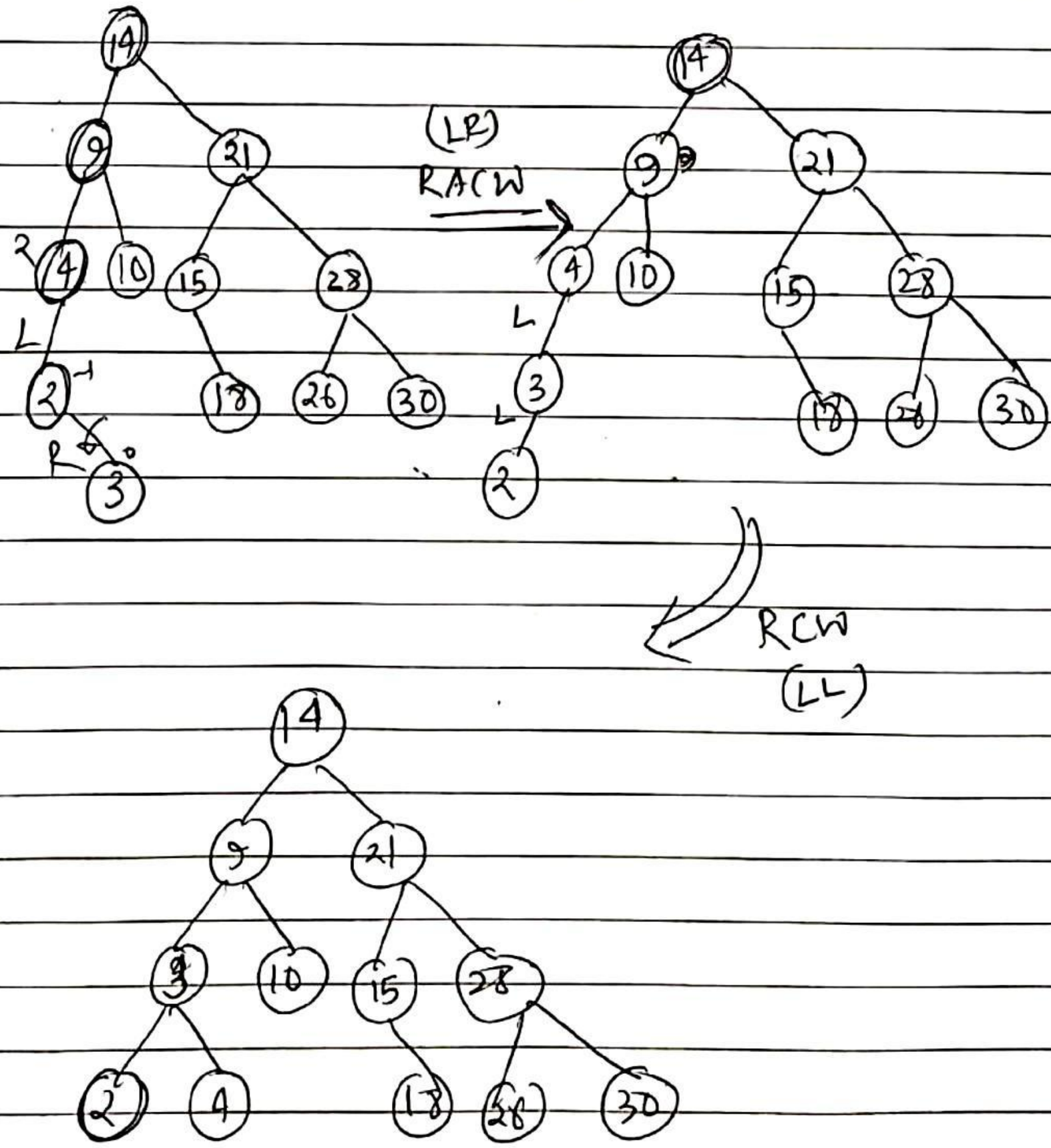
Insert - 10


21
9
28
4
15
26
30
14
18
10
-2
R
L
(RL)
RCW
R
R
RACW
(RR)

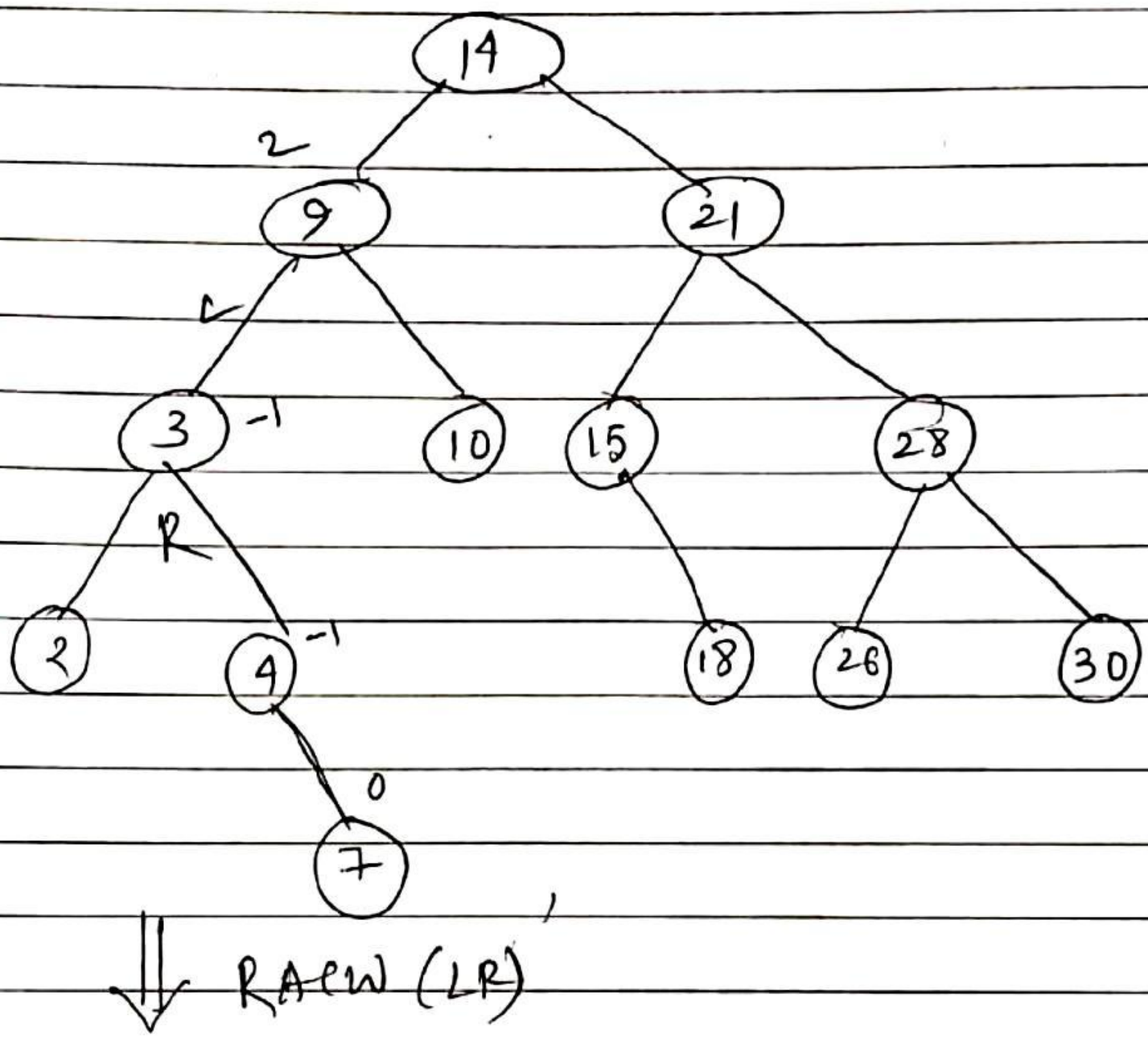
Insert -2



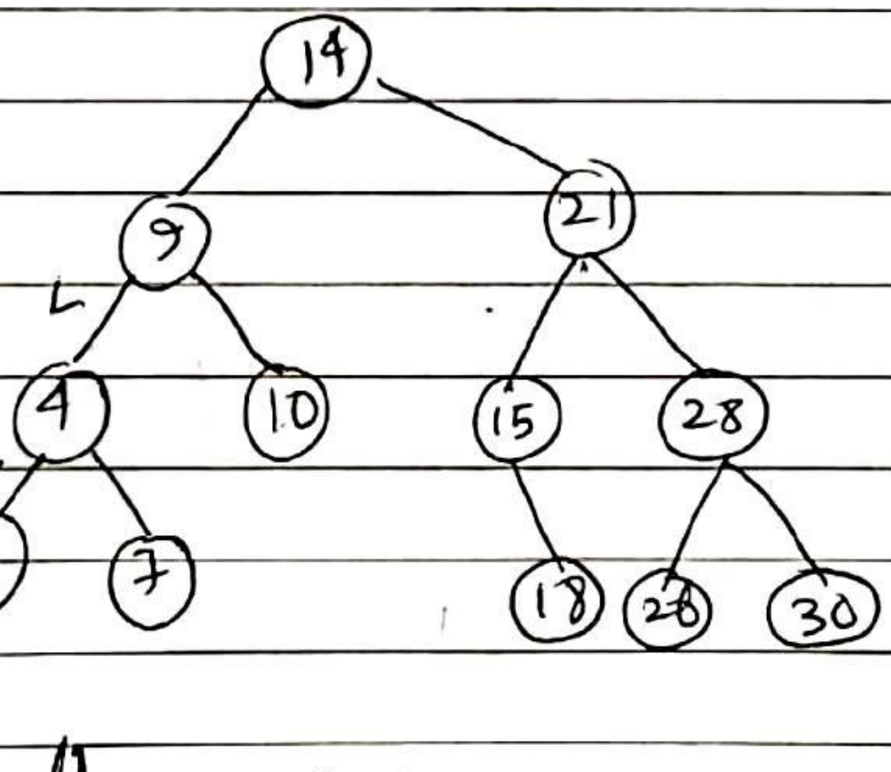
Insert -3



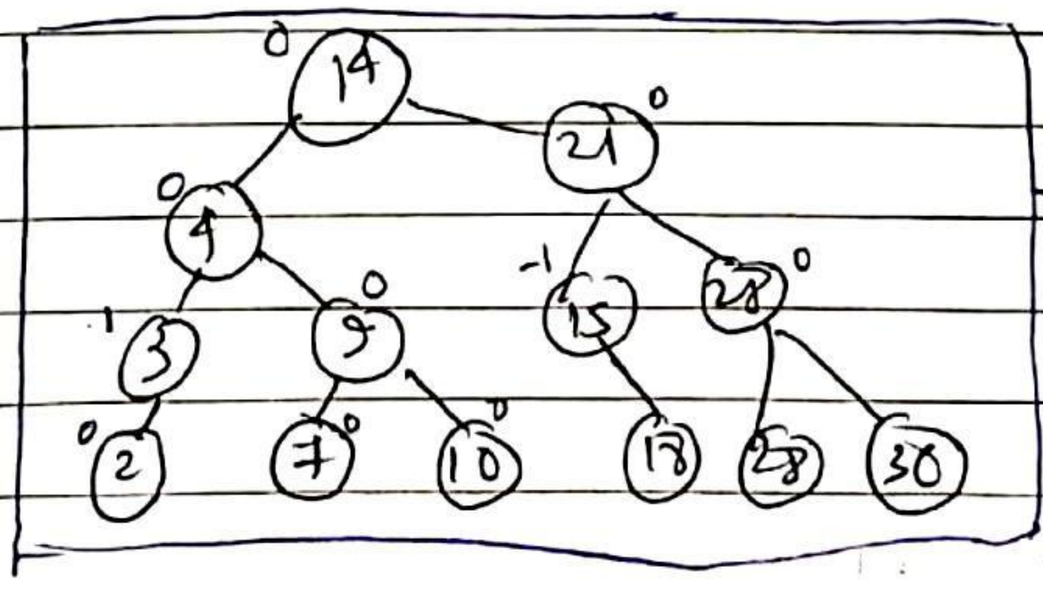
Insert ~~next~~ last ~~element~~ Input - 7



RACW (LR)



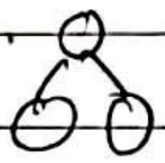
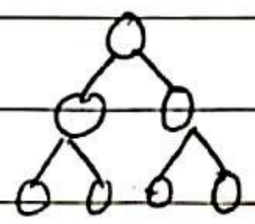
RCW (LL)



→ Final balanced tree

• Minimum and Maximum nodes in an AVL tree of hight h —

→ Maximum number of node present (in AVL tree or normal binary tree) is $= (2^{h+1} - 1)$ // h = high of the tree.

~~for hight~~ no. of node	~~node hight~~
1	
$1+2^1$	
$1+2^1+2^2$	
⋮	

$1+2^1+2^2+2^3+\cdots+2^h$ for hight h

$$\Rightarrow \frac{1(2^{h+1}-1)}{2-1}$$

$$= (2^{h+1}-1)$$

→ Minimum no. of node of AVL tree of hight-h is,

$$N(h) = N(h-1) + 1 + N(h-2)$$

$= 1$; hf $= 0$

$= 2$; h $= 1$

$N(2) = N(1) + 1 + N(0)$

$= 2 + 1 + 1$ (With hight 2 minimum 4 node possible)

$= 4$ node (min)

$N(3) = N(2) + 1 + N(1)$

$= 4 + 1 + 2 = 7$ (min node).

→ minimum no. of node can present in binary tree with hight-h $= (h+1)$

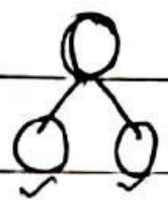
g-1995

Question - (1)

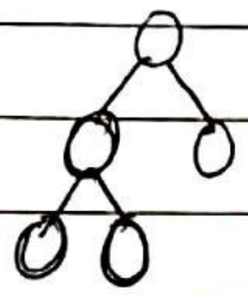
A binary tree 'T' has 'n' leaf nodes. The no. of nodes of degree '2' in 'T' is -

a) $\log_2 n$ b) $n-1$ c) n d) 2^n

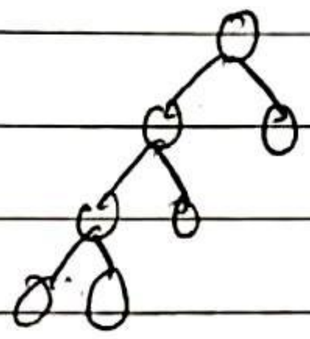
→



1 node of degree 2 = to 2 leaf.



2 " " = 3 "



3 " " " = 4 leaf.

n node with degree 2 = n+1 leaf.

node leaf

$n = (n+1)$

~~When 'n' leaf = (n-1) node of degree 2.~~

(n-1) node with deg 2 = n leaf.

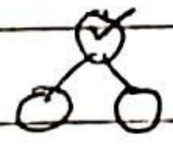
g-2001

Question - (2)

In a binary tree, the number of internal nodes of degree '1' is 5 and number of internal nodes of degree 2 is '10' the number of leaf nodes in the binary tree is =

a) 10 b) 11 c) 12 d) 15

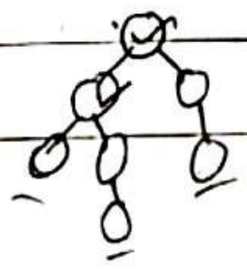
→



→ 2 leaf

10 node with deg 2 = 10+1

= 11



→ 3 leaf

gate-2008

Question (3)

Which of the following is true about search times —

AVL	BST
a) $\theta(\log n)$	$O(n)$
b) $\theta(\log n)$	$\theta(n \log n)$
c) $\theta(n)$	$\theta(\log n)$
d) $\theta(n \log n)$	$O(n)$

gate-1998

Question - (4)

Which of the following statement is false?

a) a tree with 'n' nodes has (n-1) edges.

b) a labeled rooted binary tree can be uniquely constructed given its post order and pre order traversal.

c) a complete binary tree with 'n' internal nodes has (n+1) leaves.

d) The max ~~node~~ no. of nodes in a binary tree of hight 'h' is $(2^{h+1}-1)$.

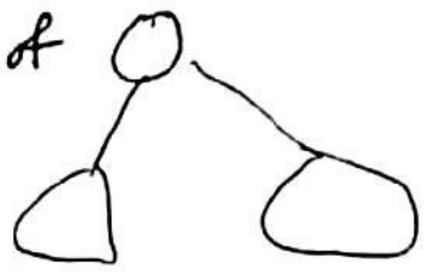
gate-2005 ~~1995~~

Question - (5)

In a binary tree, for every node the difference b/w the number of nodes in the left and right subtrees is at most 2. If the height of the tree is $h > 0$, then the minimum no. of nodes in the tree is —

a) 2^{h-1} b) $2^{h-1}+1$ c) 2^h-1 d) 2^h.

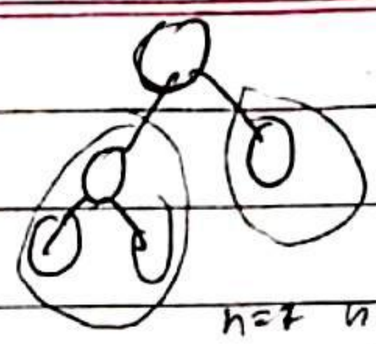
$N(h)$ → no. of node of hight h.

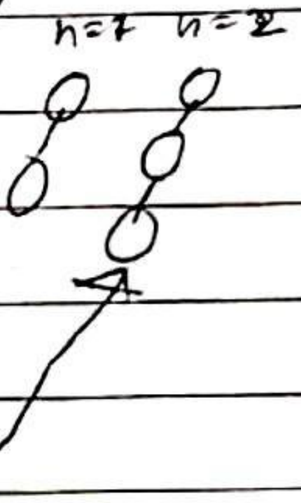


Recurrence equation —

→ $N(h) = N(h-1) + 1 + (N(h-1)-2)$

$$\boxed{N(h) = 2N(h-1) - 1} \quad \text{—— (1)}$$





$N(2) = 2N(2-1) - 1$

$= 2N(1) - 1$

$= 2(2) - = (3)$, $h=2$

$N(h) = 2N(h-1) - 1$ —— (I)

$N(h-1) = 2N(h-2) - 1$ —— (II)

$N(h-2) = 2N(h-3) - 1$ —— (III)

(II) and (III) put into (I)

$N(h) = 2^3 N(h-3) - 2^2 - 2 - 2^0$

$= 2^K N(h-K) - 2^{K-1} - 2^{K-2} + \cdots + 1$

$= 2^K N(h-K) - (2^{K-1} + 2^{K+1} + \cdots + 1)$

(G.P)

$= 2^K N(h-K) - (2^K - 1)$

$h-K=2$

$K = h-2$

$= 2^{h-2} N(2) - (2^{h-2} - 1)$

$= 2^{h-2} * 3 - (2^{h-2} - 1)$

$= 3 \cdot 2^{h-2} - 2^{h-2} + 1$

$= 2^1 \cdot 2^{h-2} + 1$

$= 2^{h-1} + 1$

→ First use substitution method by drawing the tree, according to question.

gate - 1997

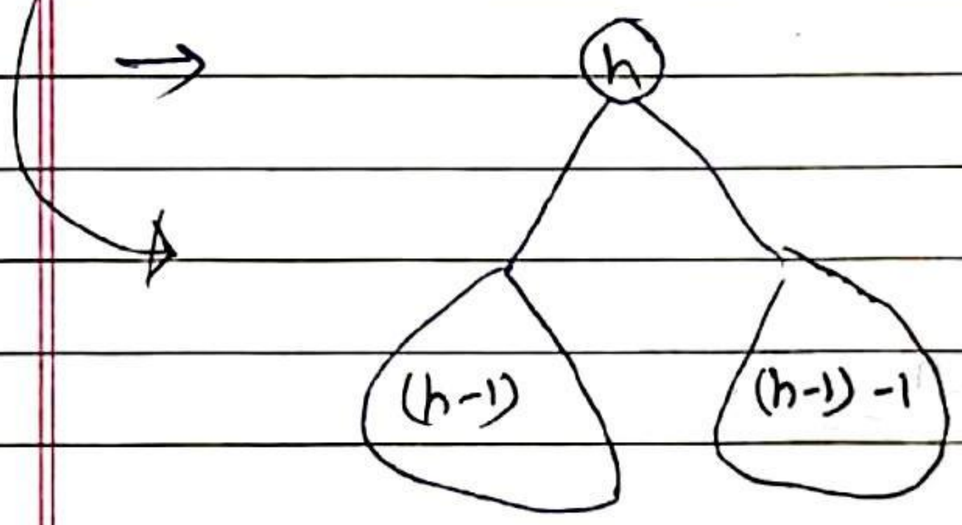
Question - (6)

A size balanced binary tree is a binary tree in which for every node, the difference b/w the no. of nodes in the left and right subtrees is at most 1.
(is the length of the path)

The ~~diff~~ distance of a node from the root to the node. (the root is the length of the path from)

The height of a binary tree is the maximum distance of a leaf node from the root.

Prove by (without) induction on h, that a size-balanced binary tree of height 'h' contains at least 2^h nodes.



N(h) → min no. of node that can present in tree of hight h.

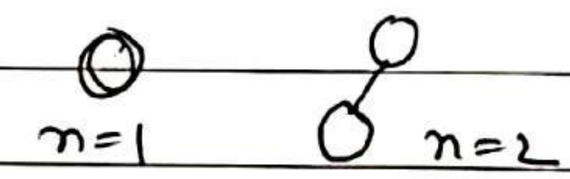
$N(h) = \underbrace{N(h-1)}_{LST} + \underbrace{1}_{root} + \underbrace{(N(h-1)-1)}_{RST}$

$= N(h-1) + 1 + N(h-1) - 1$

$N(h) = 2N(h-1)$ —(1)

(min n=0, node = 1)

h=0 h=1



$N(h-1) = 2N(h-2)$ —(2)

$N(h-2) = 2N(h-3)$ —(3)

Back substitution -

$N(h) = 2^3 N(h-3)$.

$N(h) = 2^K N(h-K)$.

$= 2^h N(0)$.

$= 2^h \cdot h$.

$N(h) = 2^h$

$h-K = 0$

$K = h$

• Can ask Question on this topic ⟹

(1) What is the min no. of node required ~~to make~~ in order to make hight ().

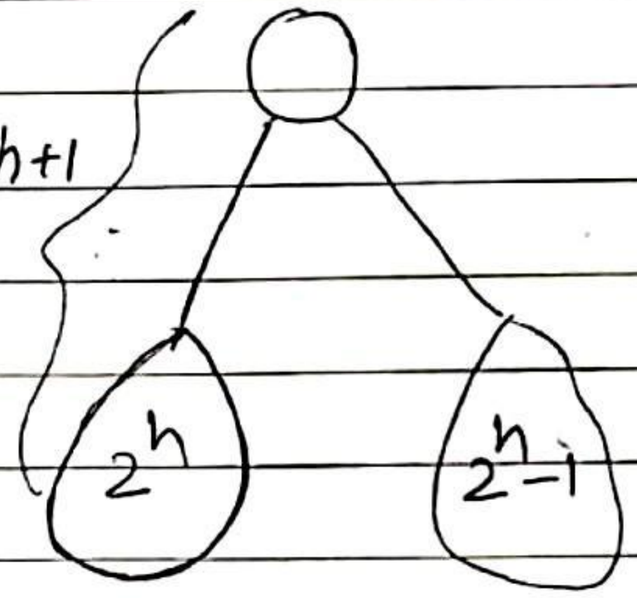
(2) given node then how much hight build.

~~$N(h) = 2^h$~~

By induction -

$N(n) = 2^h$

$N(h+1) = 2^{h+1}$



$N(h+1) = 2^h + 1 + 2^h - 1$

$N(h+1) = 2^{h+1}$

q-20 **Question** - (7) (Balanced binary search tree)

Suppose we have a balanced binary search tree 'T' holding 'n' numbers. We are ~~gi~~ given two numbers 'L' and 'H' and wish to sum up all the numbers in 'T' that lie b/w 'L' and 'H'. suppose there are 'm' such numbers T. If the highest upper bound on the time to complete the sum is $O(n^a \log^b n + m^c \log^d m)$, the value of $a + 10b + 100c + 1000d$.

→

$O(m \log n)$ ~~$O(n)$~~ $O(n^a \log^b n + m^c \log^d n)$

$a = 0, \quad b = 0$

$c = 1, \quad d = 1$

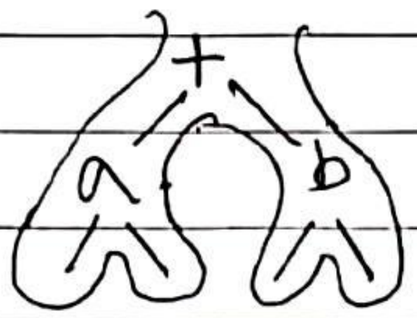
$a + 10b + 100c + 1000d$

$\Rightarrow 0 + 10*0 + 100*1 + 1000*1$

$\Rightarrow$ 1100.

- Expression trees =

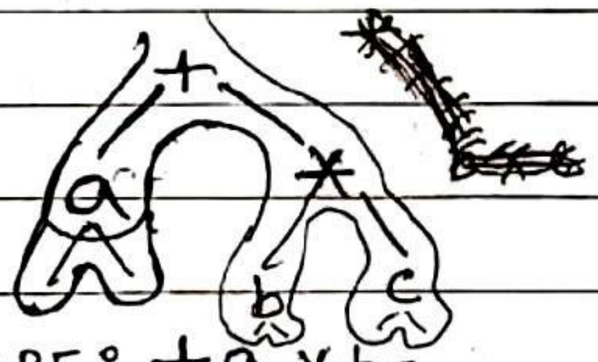
① a+b



PRE : +ab (prefix expression)
IN : a+b (infix expression)
POST : ab+. (postfix expression)

→ On the expression tree if you perform pre, IN, and post order traversal, ~~then~~ it will give you prefix, infix and postfix expression respectively.

② a+b*c



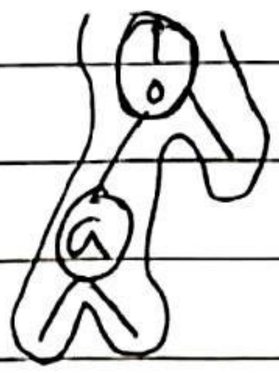
PRE : +a*bc
IN : a+b*C
POST : abC*+

③ -a ✓

IN : a-

IN : -a

④ a!

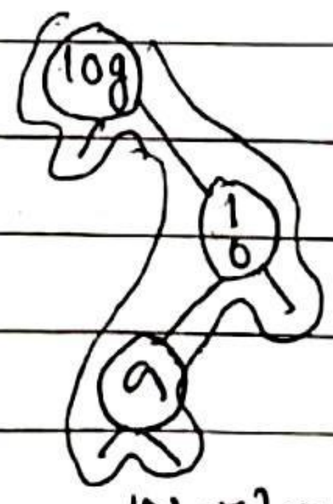


IN : a!

⑤ log a

IN : log a

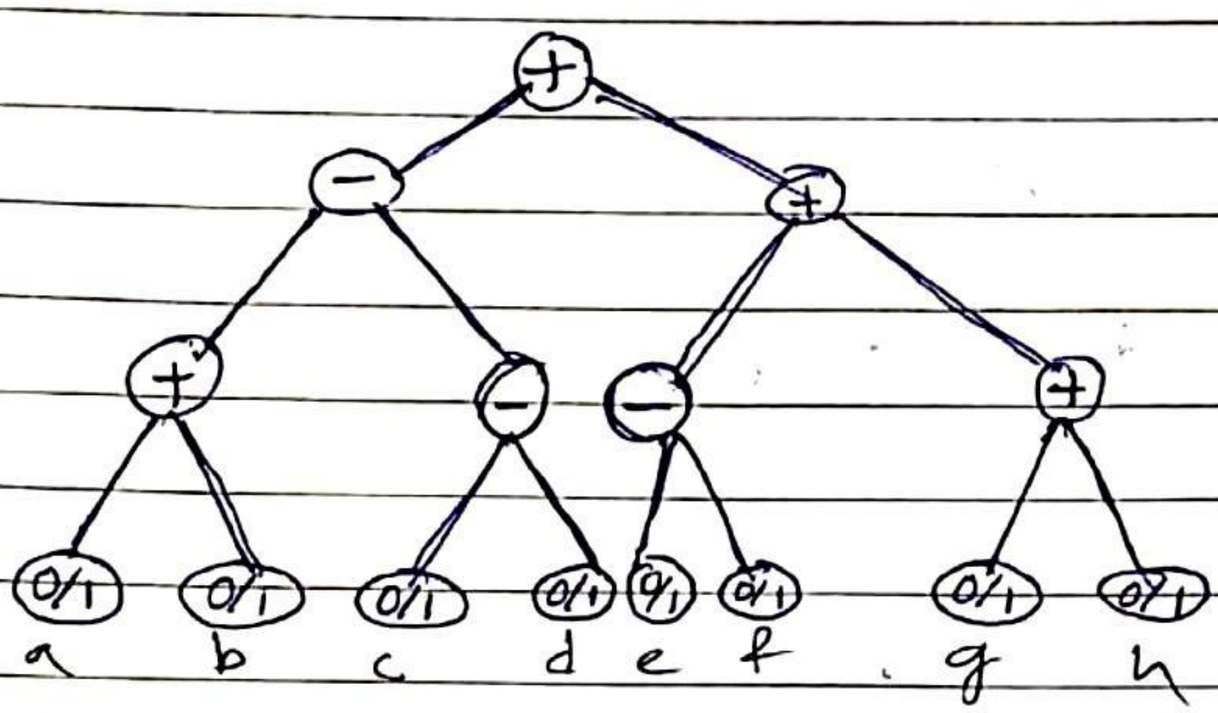
⑥ log(a!)



IN order : log(a!).

→ By traversing Inorder, find out the tree.
→ When combination of operators, then build tree step by step.

g-2014 **Question** -①

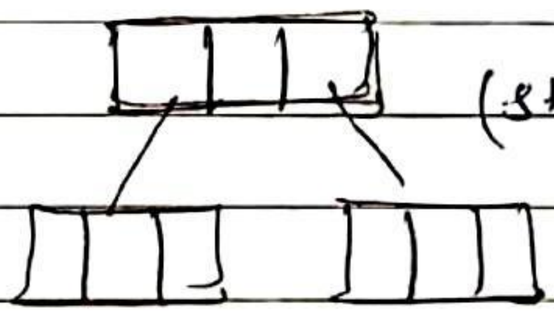


When every operand can take ~~0/1~~ '0' or '1' then what will be the max value of expression.

→ $\underset{1\ 1}{(a+b)} - \underset{0\ 1}{(c-d)} + \underset{1\ 0}{(e-f)} + \underset{1\ 1}{(g+h)}$

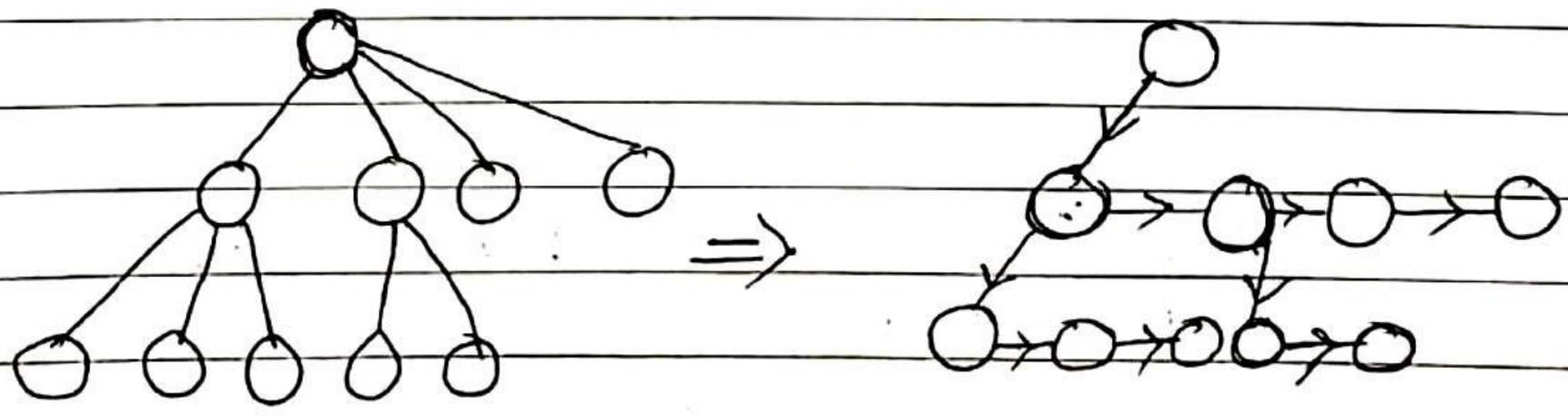
→ 3 + 3 → (6)

- Various tree representations ==

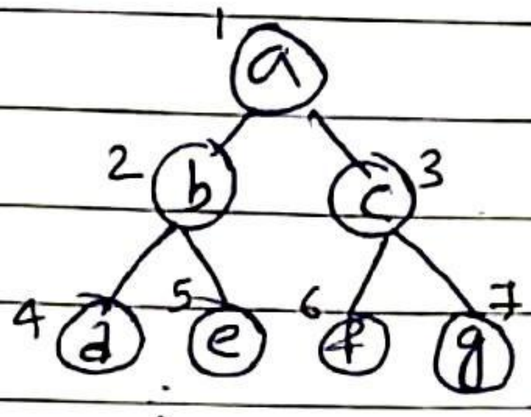


(structure representation)

- left child Right sibling representation ==
(LCRS)



- **array representation of Binary tree —**



i = index

$LC = 2i$

$RC = 2i + 1$

parent(x) = x/2.

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

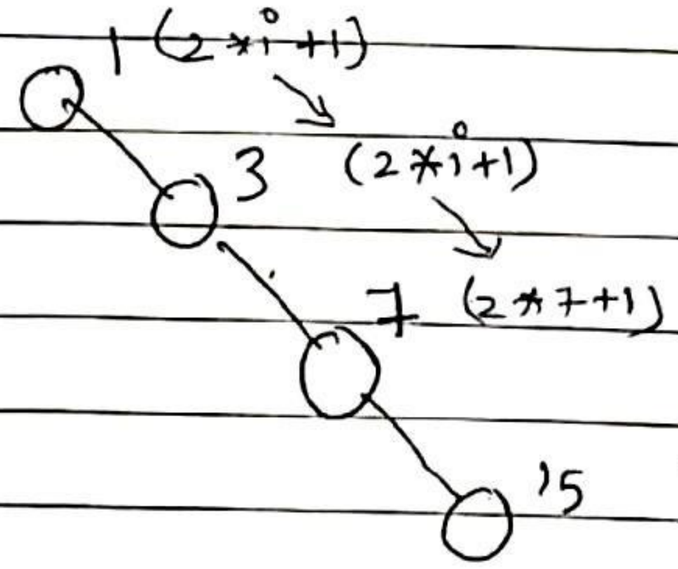
adv:

→ finding out parent easy, and finding out right child and left child is easy by array representation.

→ traversing easy.

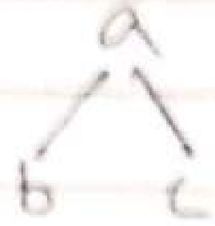
→ widely used in implementation of heaps.

disadv:



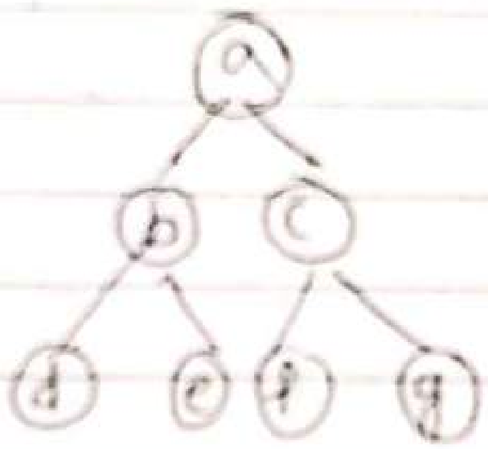
→ In worst case (if the tree is qued) and if has got 'n' elements then the size of array required is entire $(2^n - 1)$

- Nested representation =



(b a c) (a b c)
Left Root R Root L R

Ex:



= a (b d e) (c f g)
Root L R

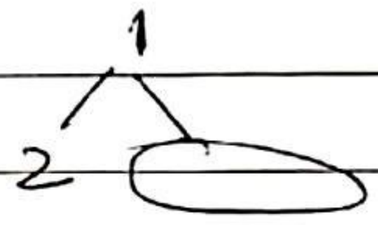
4 - representation =

(i) → Normal (pointer, struct)
(ii) → LCRS (n-ary, n>>2)
(iii) → Array (used in Heaps, all must complete BT)
(iv) → Nested.

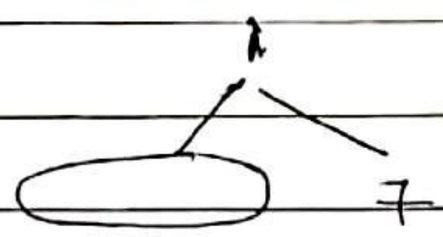
GATE-2000 Question - (2) Root L R
(x y z), y and z can be NULL,
which of the following does represent binary tree -

a) (12(4567))
b) (1((239)56)7)
✓ c) (1 (239) (567))
d) (1(23 NULL)(45))

(a) (12(4567)) ✗

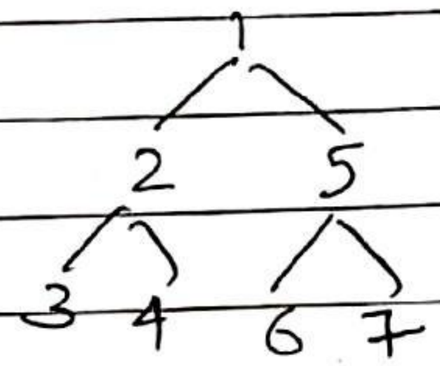


(b) (1(234)56)7) ✗

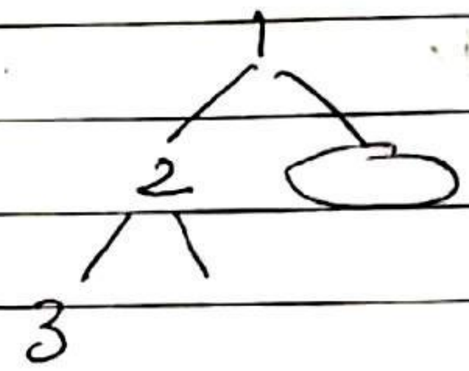


(c) ✗

(c) (1(234)(567)) ✓



(d) (1(23NULL)(45)) ✗

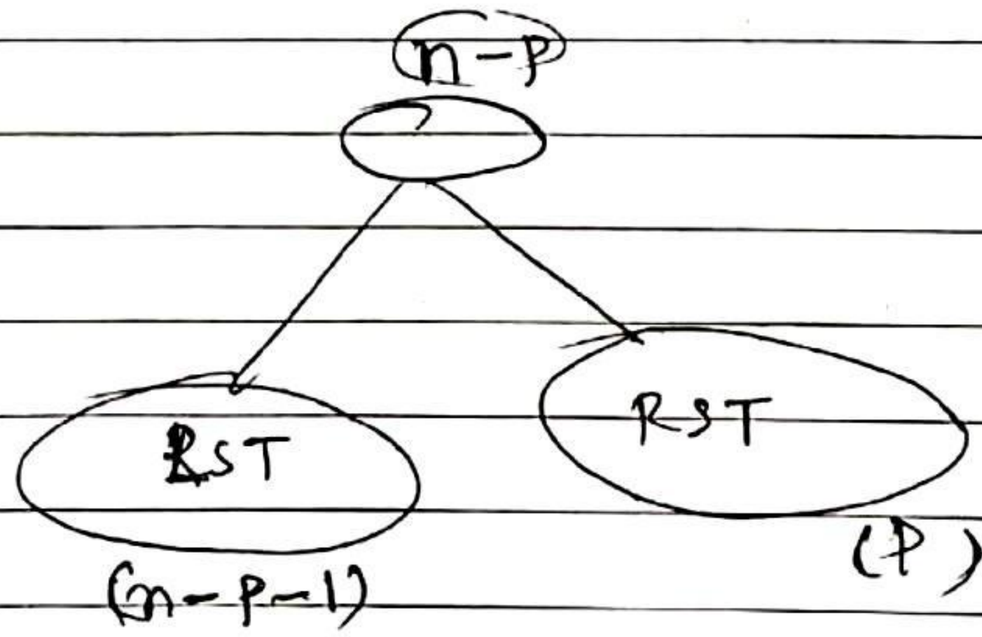


gate 2005 **Question - (3)**

The numbers 1, 2, ---n are inserted into a BST in some order. In the resulting tree, the right subtree of the root contains p nodes. The first numbers to be inserted in the tree must be -

(a) p (b) p+1 (c) n-p (d) n-p+1.

→



(1 2 3 ... (n-p-1)) ((n-p+1) ... n)
(P)

gate 2006

Question - (4)

An array 'X' of 'n' distinct integers is interpreted as a complete binary tree. The Index of the first element of the array is '0'.

(Q-1) The ~~Arr~~ index of the parent of element X[i], i≠0, is

(a) ⌊i/2⌋ ✓

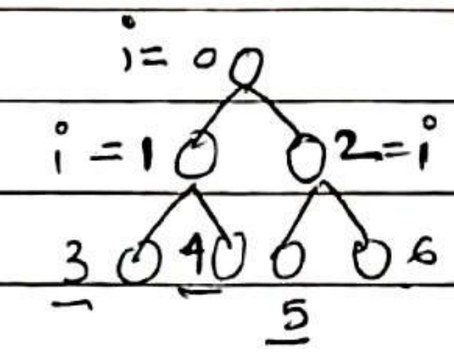
(b) ⌈i/2⌉ ✗

(c) ⌈(i-2)/2⌉ ✓

(d) ⌈i/2⌉ - 1 ✓

both correct.

→



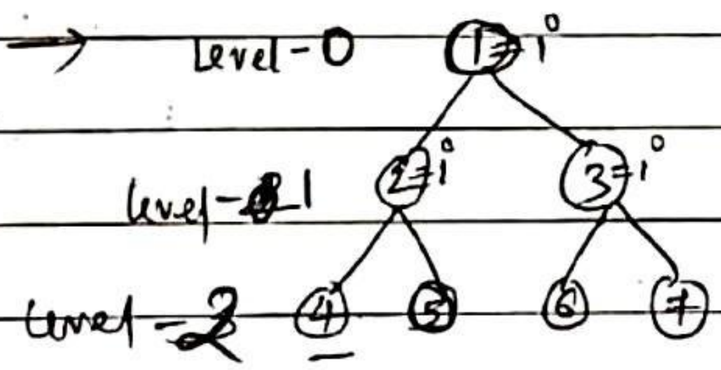
Value of i by substitute in option find correct option.

(Q-2)

If the root node is at level 0, the level of ~~node~~ element X[i], i≠0, is

(a) ⌊log₂ i⌋ ✗ (b) ⌈log₂(i+1)⌉ (c) ⌊log₂(i+1)⌋ ✓ (d) ⌈log₂ i⌉ ✗.

→



⌊2.58⌋

⌊2.58⌋ = 2

⌈2.58⌉ = 3

by using subtitution method ~~first~~ founded correct option.

q-2014

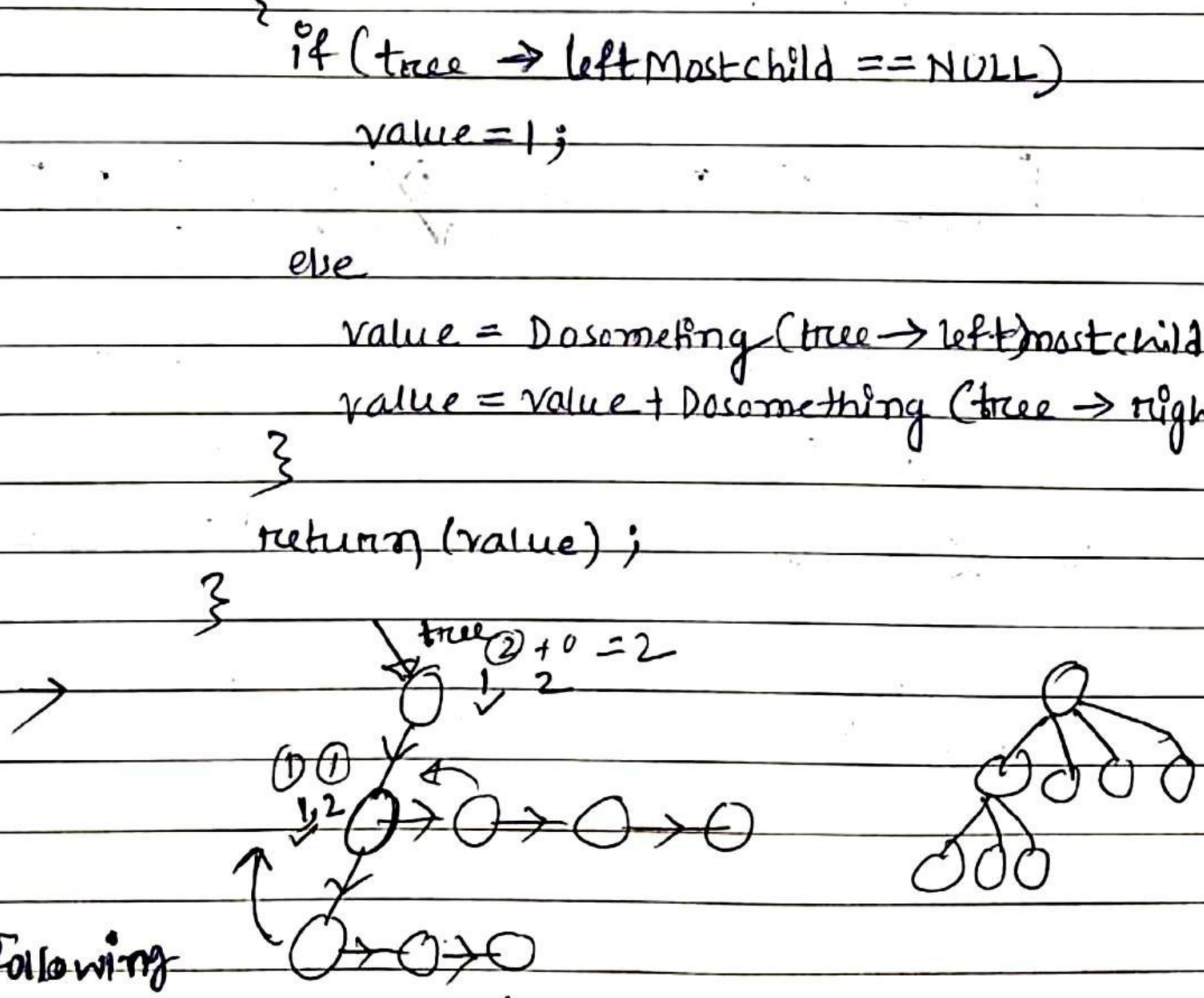
Question]-⑤ (LCRS)

leftmostchild – rightsibing rep is used -

Each node of tree is of type treeNode.

```
typedef struct treeNode * treeptr ;

struct treeNode
 {
    treeptr leftMostchild , rightSibling ;
 };

int DoSometing ( treeptr tree)
 {
   int value =0 ;
   if( tree != NULL)
    {
     if (tree → leftMostchild == NULL)
       value=1 ;

     else
       value = Dosometing (tree → leftmostchild );
       value = value + Dosomething (tree → rightSibling );
    }
   return (value) ;
 }
```

n



Following

∧ Which of option is false –

(a) no. of Internal node are counted.

(b) right of the tree counted.

(c) count no. of node without right sibling.

- no. of leaf node are counted.