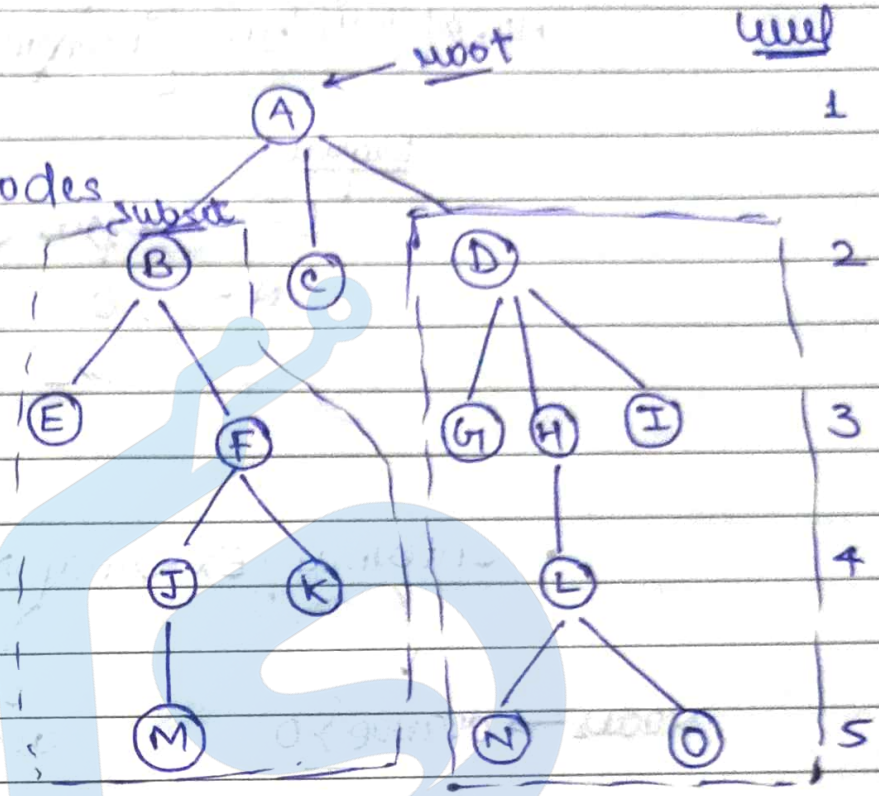
# Section-15

## Trees

# Terminology :-

Tree is a collection of nodes, vertices, and edges.

N Nodes → (N-1) Edges




Tree is a collection of nodes where one node is taken as a root node and rest of the node is divided into disjoined subsets and each subset is a tree or subtree.

1. Root {A}
2. Parent {B}
3. Child {E, F}
4. Siblings {G, H, I}, {J, K}
5. Descendents {G, H, I, L, N, O} → All the nodes reachable from D.
6. Ancestors - For M {J, F, B, A}, For N - {L, H, D, A}

- Degree of Node
  - No. of children
  - For
    - Indegree + outdegree
    - incoming edge + outcoming edge

Degree

D → Degree 3

N → 0

L → 2

- Internal / External Nodes
  - Nodes → degree > 0
  - degree = 0 → leaf nodes
  - Terminal nodes

- Levels

Root – 1

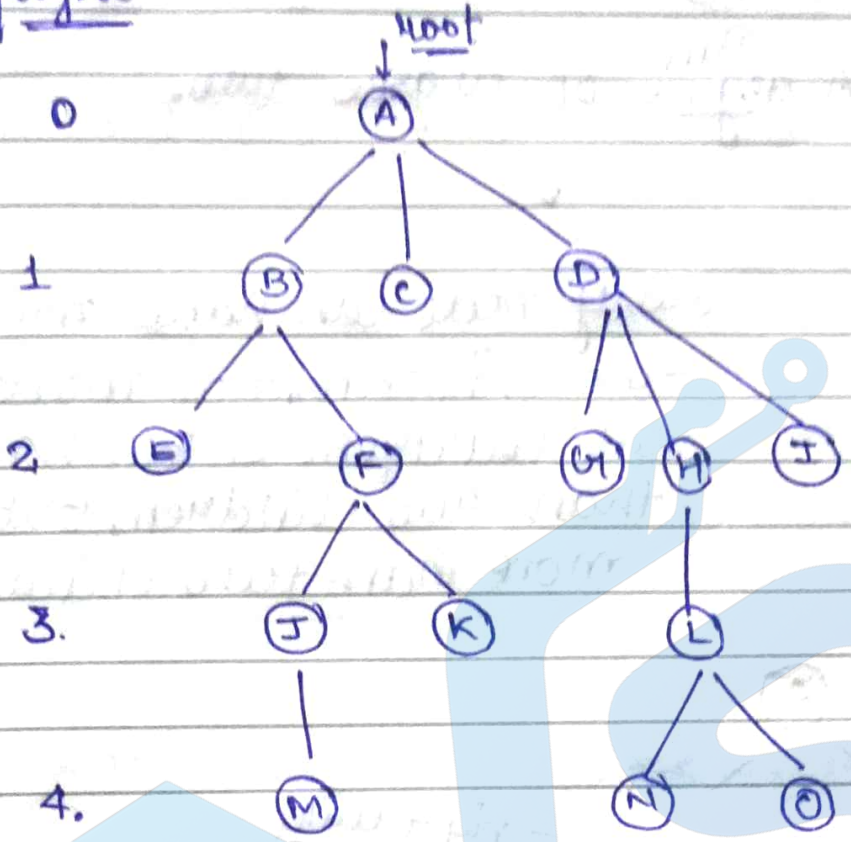
Children – 2

3

4

5

10. Height




Height and level are used for analysis

11. Forest

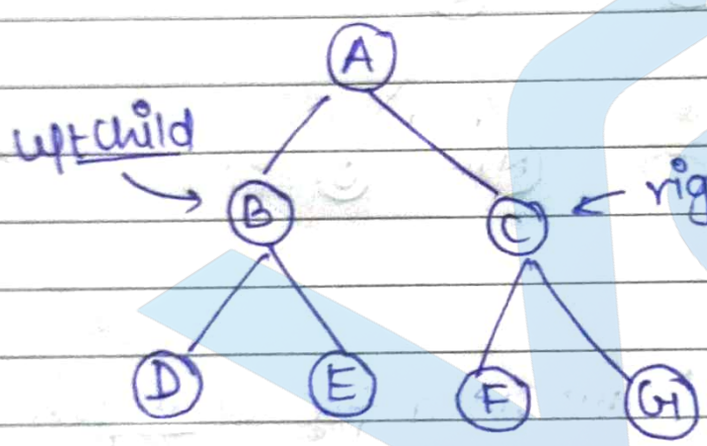
↳ collection of Tree is forest

Binary Tree

↳ A ~~degree~~ Tree of degree two.

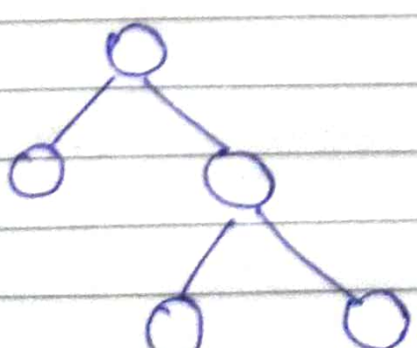
↓

Every node can have maximum two children, not more than two children. It can have less than two children, but not more than two children.



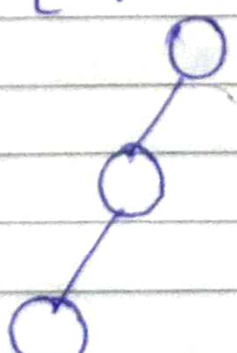

deg(T) = 2

children → {0, 1, 2}



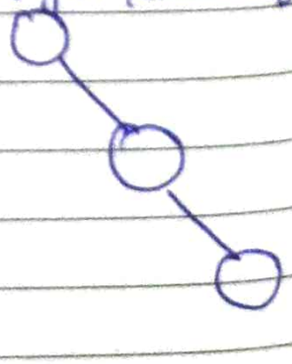
Yes, binary tree

[Left Skewed]

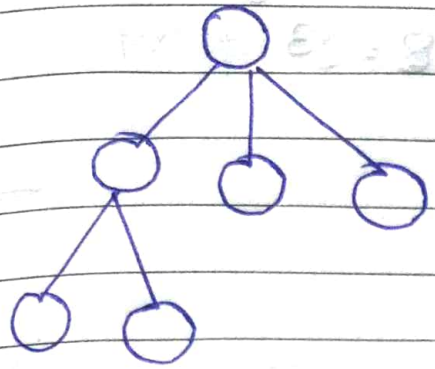


Yes, binary tree

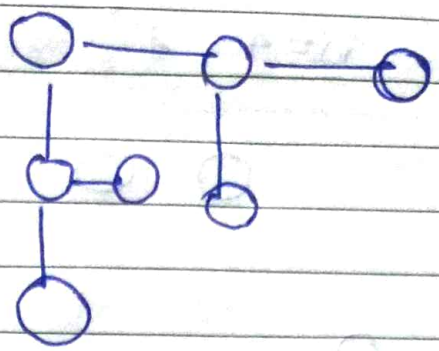
[Right Skewed]



Yes, binary tree



No, binary Tree



Yes, binary Tree

---

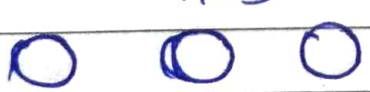
## Number of binary Trees using N Nodes

$n=3$

1. Unlabelled Nodes
2. Labelled Nodes

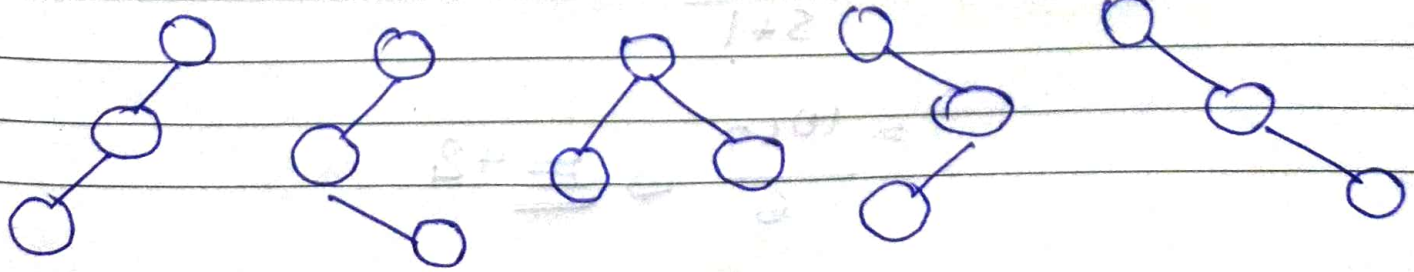
- Unlabelled Nodes [No of tree with max height $= 4 = 2^2$]

$n=3$

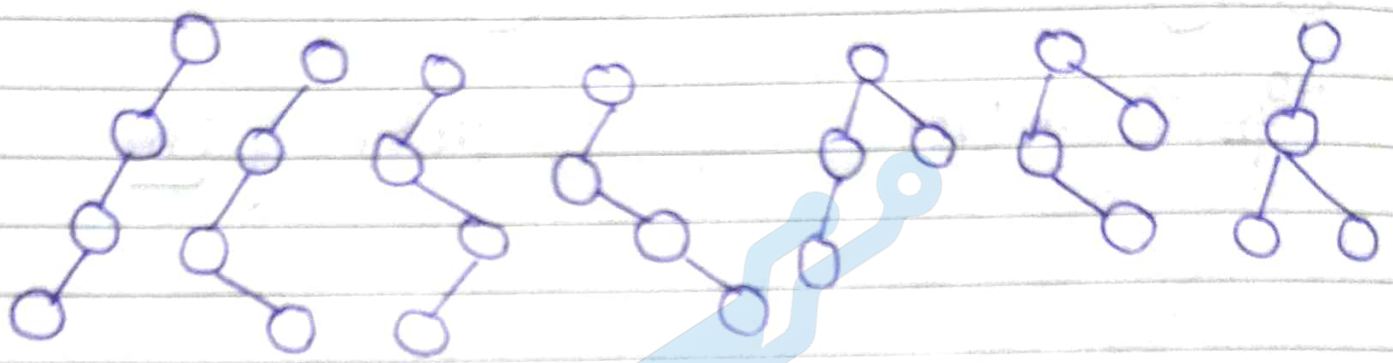


$T(3) = 5$

5 Number of binary tree



$n = 4$ → No. of Tree with Max height $= 8 = 2^3$



T(4) = 14

$T(4) = 14$

$$T(n) = \frac{2nC_n}{n+1}$$ → Catalan number

combination formula

$$T(5) = \frac{2 \times 5 C_5}{5+1}$$

$$= \frac{10 C_5}{6} \Rightarrow 42$$

Number of Tree with Max Height = $2^n - 1$

| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $T(n) = \frac{2nCn}{n+1}$ | 1 | 1 | 2 | 5 | 14 | 42 | 132 |

$T(6) = 1 \times 42 + 1 \times 14 + 2 \times 5 + 5 \times 2 + 14 \times 1 + 42 \times 1$

$= 132$

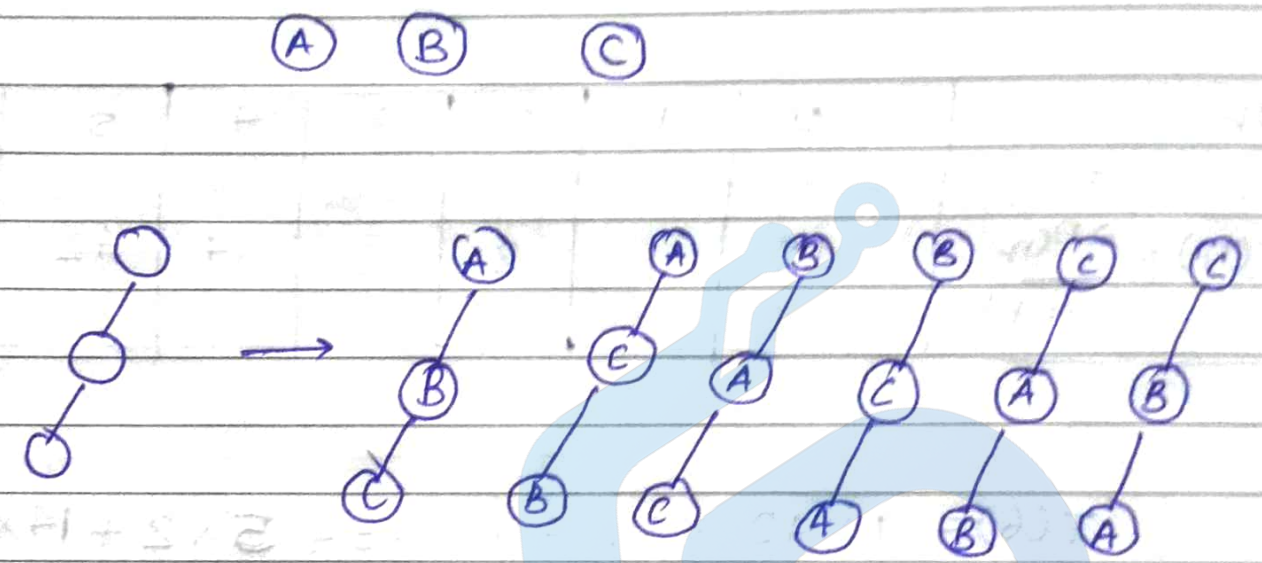
$T(6) = T0 \times T5 + T1 \times T4 + T2 \times T3 + T3 \times T2 \times T4 \times T1 + T5 \times T0$

$$T(n) = \sum_{i=1}^{n} T(i-1) * T(n-i)$$

Recursive formula

- Labelled Nodes

- $n = 3$




$6 = 3!$

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

shapes — filling

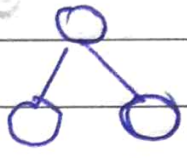
# Height vs Nodes in Binary tree

Height
h=1

min

n=2

max



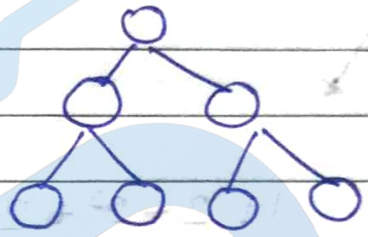
n=3

Height
h=2

min

n=3

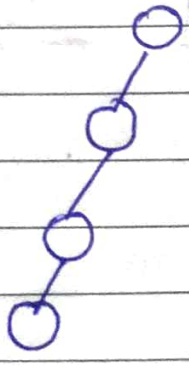
max



n=7

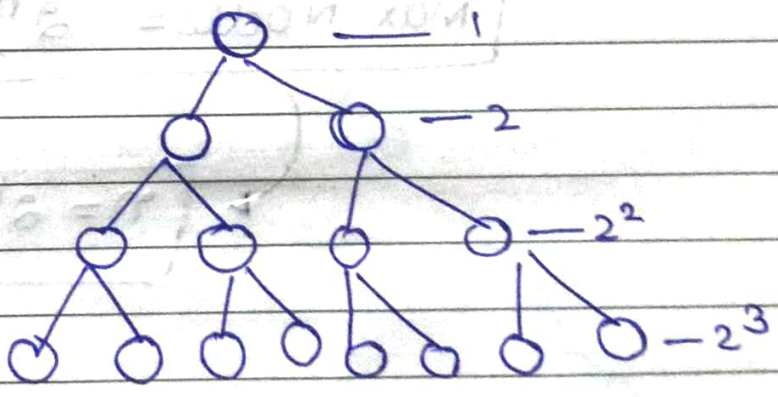
Height
h=3

min



n=4

max




n=15

Min Nodes $n = h+1$

Max Nodes =

↳ G.P series $[\ a + ar^2 + ar^3 + \ldots\ldots + ar^k\ ] \Rightarrow$

$$\frac{a(r^{k+1}-1)}{r-1}$$

$1 + 2 + 2^2 + 2^3 + \ldots + 2^h$

$a = 1$ $r = 2$

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

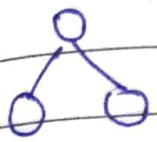
Max Nodes $= 2^{h+1} - 1$

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

When number of nodes are given:

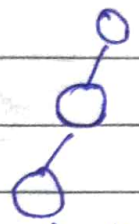
Nodes
$n=3$

min



$h=1$

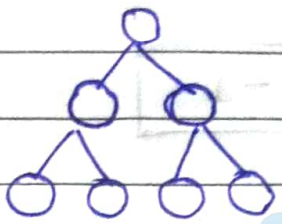
max



$h=2$

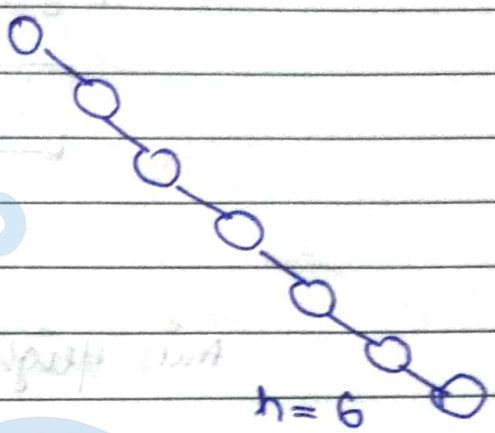
Nodes
$n=7$

min



$h=2$

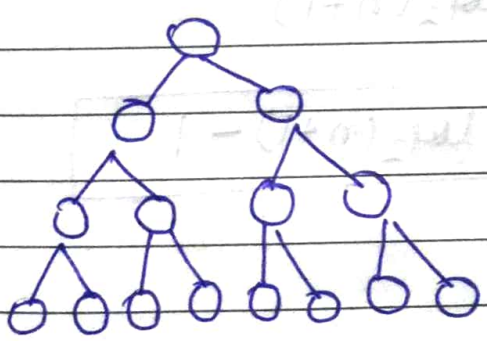
max



$h=6$

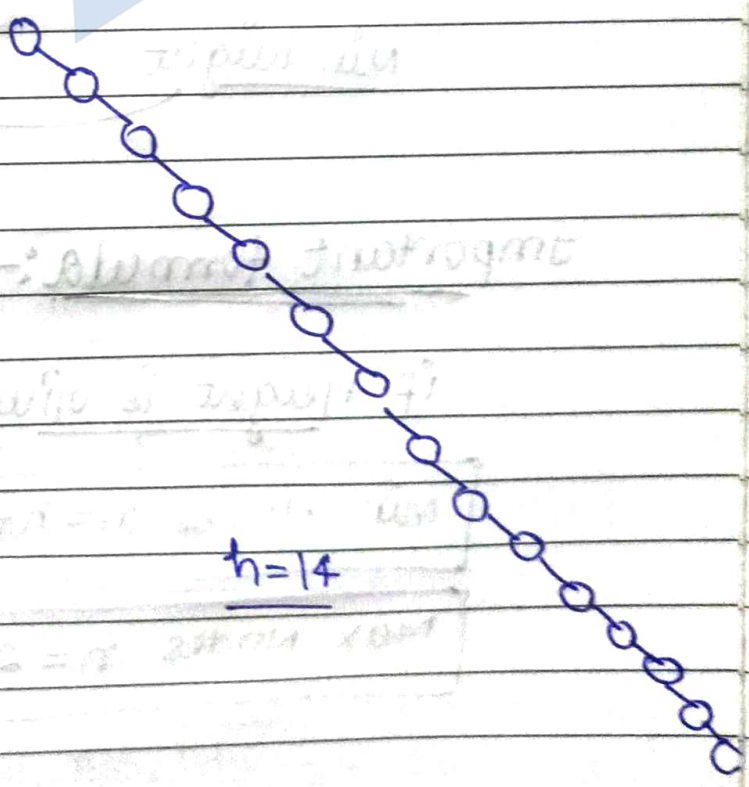
Nodes
$n=15$

min



$h=3$

max



$h=14$

Height of Binary Tree

O(logn) O(n)

$\log_2(n+1) - 1 \leq h \leq n-1$

Number of Nodes in Binary Tree

$h+1 \leq n \leq 2^{h+1} - 1$

Min Height ↓

$h =$

Max Height ↓

$H = n - 1$

Min Height ↓

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

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

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

$h+1 = \log_2(n+1)$

Min height → $h = \log_2(n+1) - 1$

Important formula:-

if Height is given

Min Nodes $n = h+1$

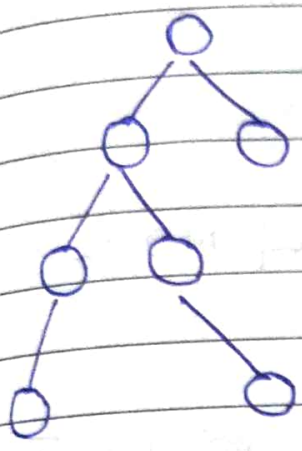
Max Nodes $n = 2^{h+1} - 1$

if Nodes are given

Min Height $h = \log_2(n+1) - 1$

Max height $h = n - 1$

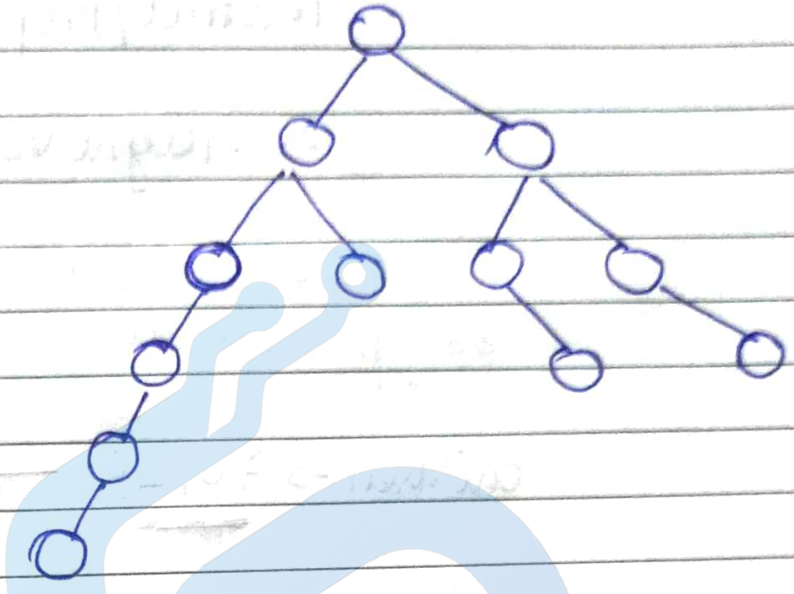
# Internal Nodes vs External Nodes in Binary Tree



deg(2) = 2

deg(1) = 3

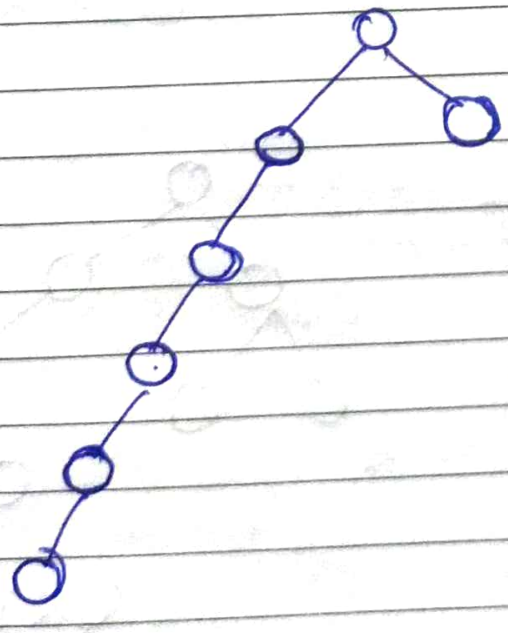
deg(0) = 3



deg(2) = 3

deg(1) = 5

deg(0) = 4



deg(2) = 1

deg(1) = 4

deg(0) = 2

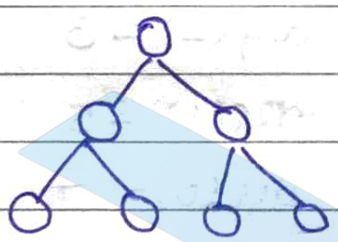
* $deg(0) = deg(2) + 1$

Valid for Binary tree

# Strict Binary Tree

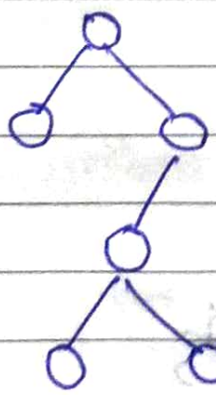
1. Strict / Proper / Complete
2. Height vs Nodes
3. Internal vs External Node.

children → {0, 2} → For complete Binary tree

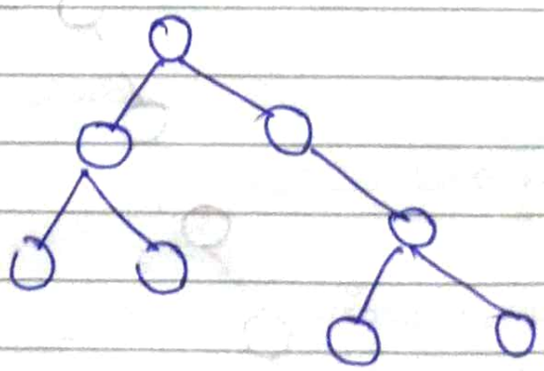


Yes, strict binary tree

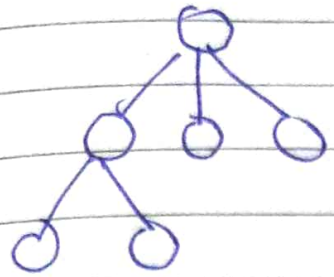
(✓)



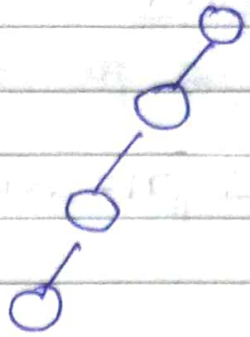
(X)



(X)

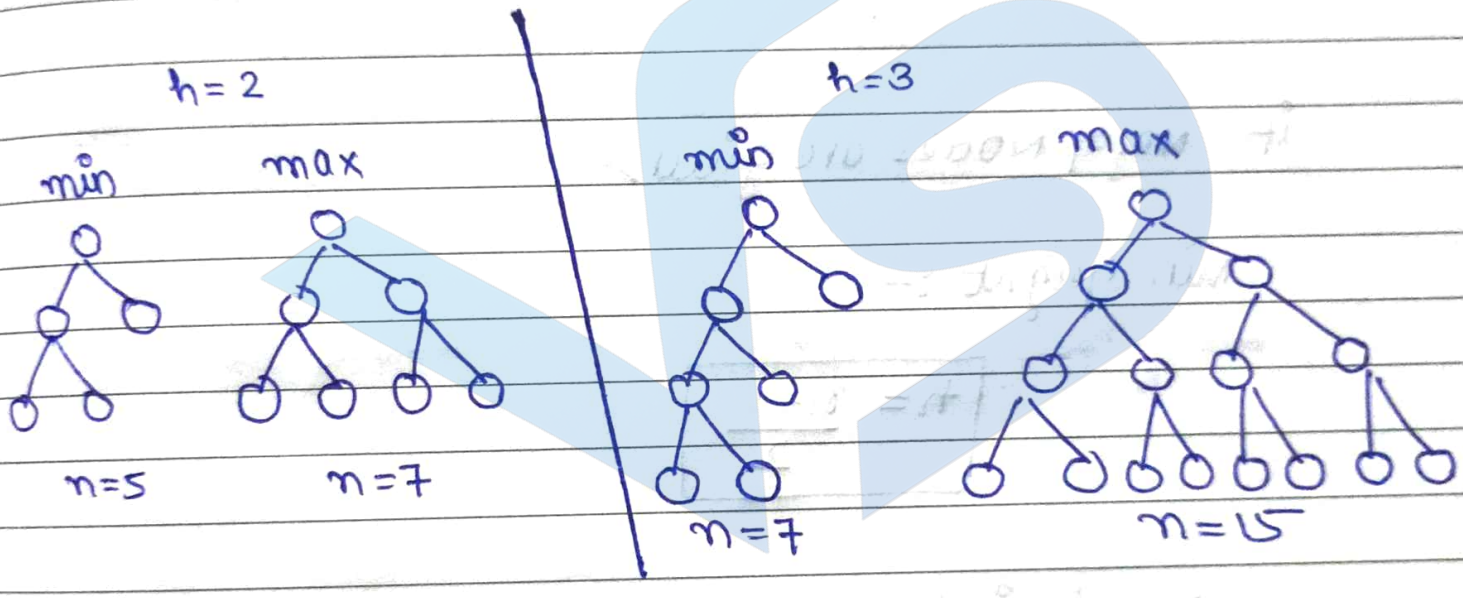


(X)

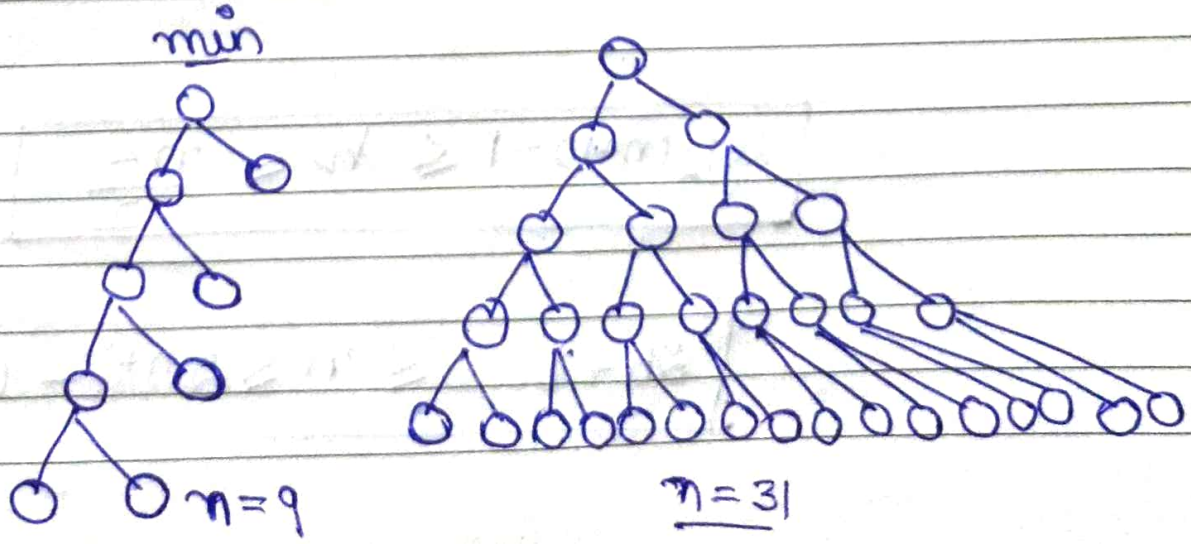


(X)

- Height vs Node of strict Binary tree




h=4



if Height is given :-

Min Nodes

$h \Longrightarrow \boxed{n = 2 \times h + 1}$

Max Nodes

$$\boxed{n = 2^{h+1} - 1}$$

if No. of Nodes are given

Min Height :-

$$\boxed{h = \frac{n-1}{2}}$$

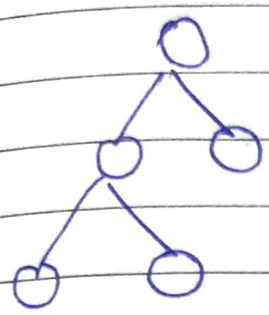
Max height :-

$$\boxed{h = \log_2(n+1) - 1}$$

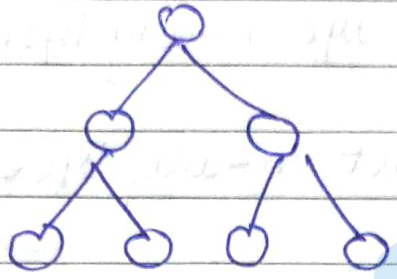
$$\boxed{\log_2(n+1) - 1 \leq h \leq \frac{n-1}{2}}$$

$$\boxed{2h+1 \leq n \leq 2^{h+1} - 1}$$

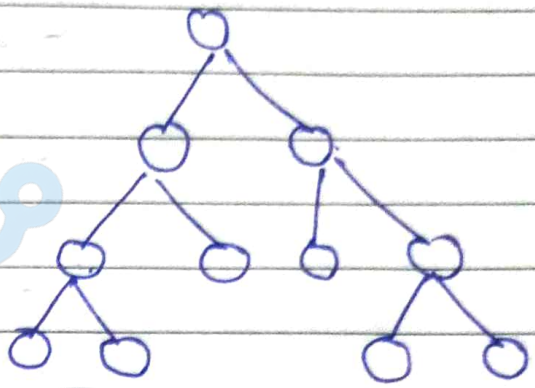
# Internal vs External Nodes of strict Binary Trees



i = 2
e = 3



i = 3
e = 4



i = 5
e = 6

~~e = i * 1~~

$e = i + 1$

↓

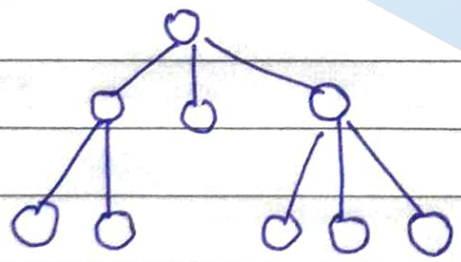
valid strict binary Tree.

# n-ary Trees :-

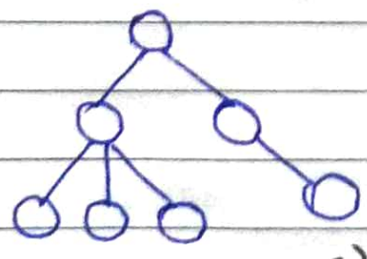
- What are n-ary Trees
- Strict n-ary Tree
- Hight vs Node
- ~~Hight~~ Internal vs External Nodes

n → degree of tree

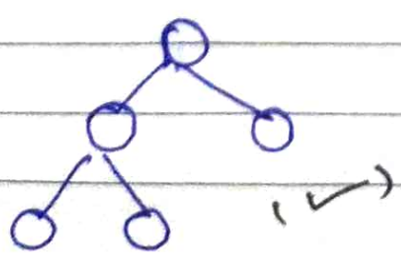
3-ary Tree {0,1,2,3}



(✓)

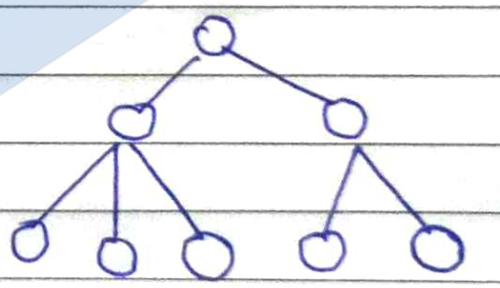


(✓)

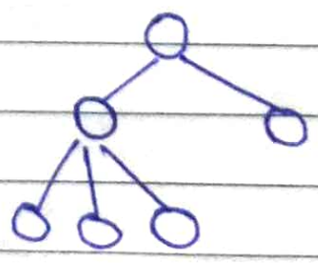


(✓)

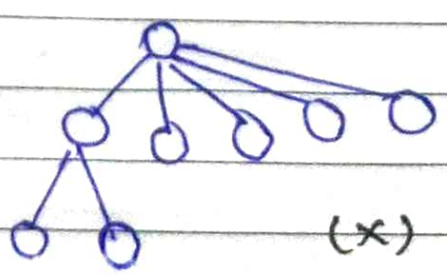
4-ary tree {0,1,2,3,4}



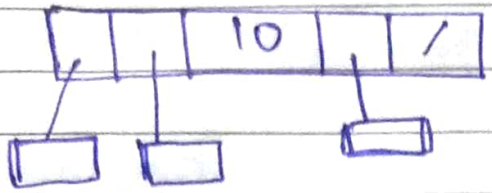
(✓)



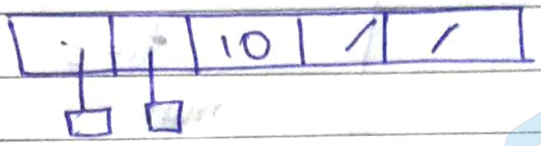
(✓)



(x)




→ 4-ary

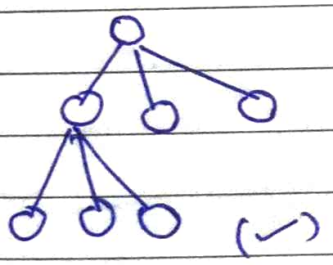



Strict n-ary tree is one in which every node can have either a zero children or exactly n children.

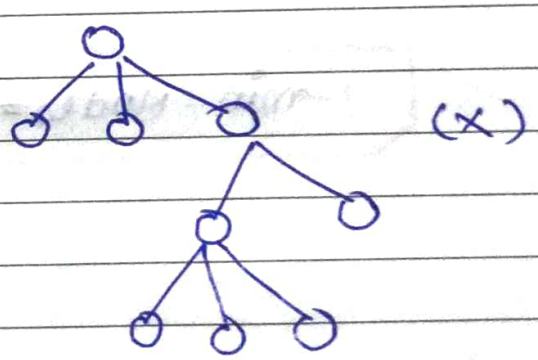
Strict 3-ary tree

{0,3}

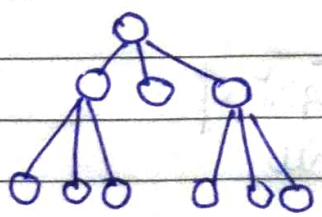
↳ every node have either zero children or exactly 3 children.



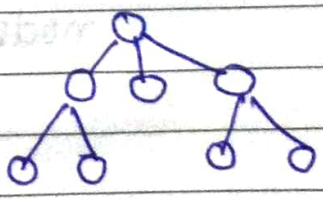
(✓)



(X)



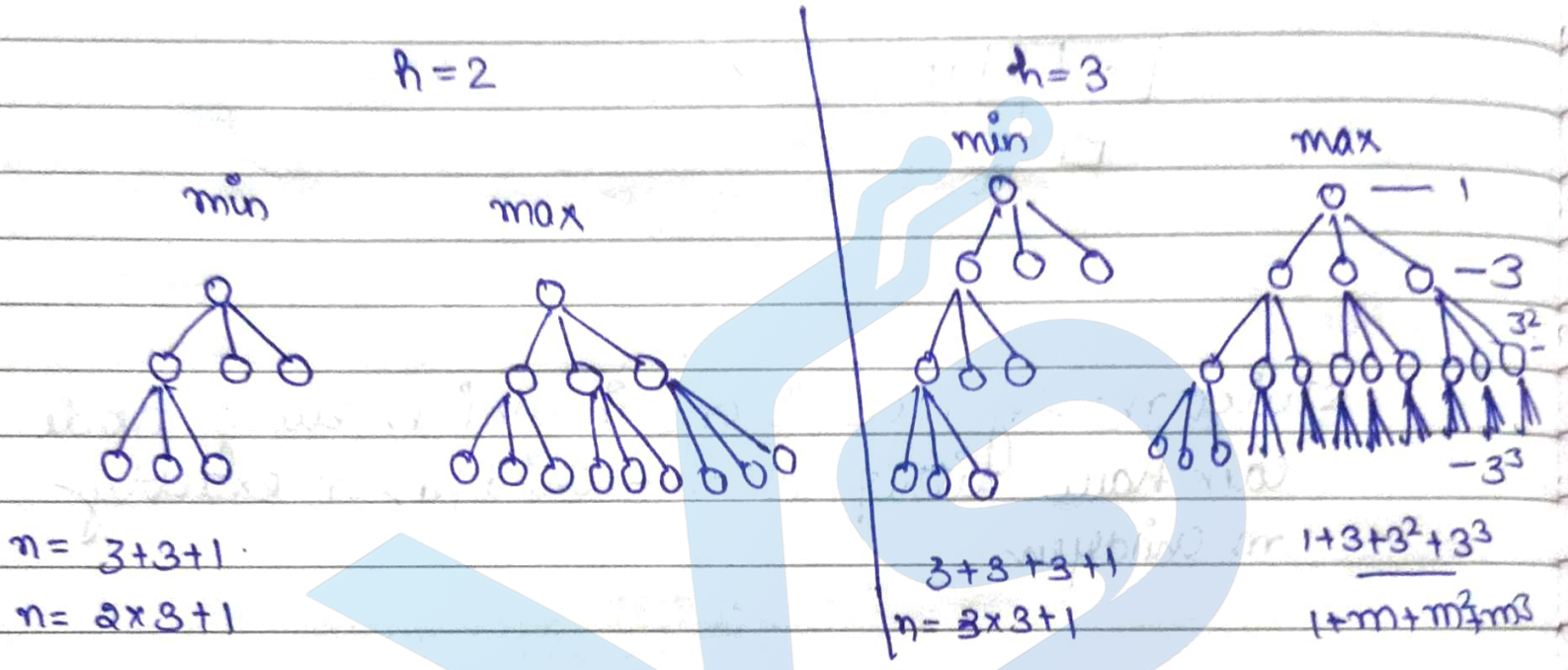
(✓)



(X)

# Analysis of n-Ary Tree :-

## Strict 3-ary trees




if Height is given

min Nodes = $n = 3h + 1$

$n = mh + 1$

- Number of nodes
- degree of tree
- height of tree

## max Nodes

max Nodes $n = 1 + m + m^2 + m^3 + \cdots + m^h$

$$= \frac{(m^{h+1} - 1)}{m - 1}$$

$$\boxed{n = \frac{m^{h+1} - 1}{m - 1}}$$

if 'n' nodes are given

Max ~~Min~~ Height

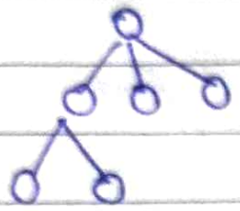
$$\boxed{h = \frac{n - 1}{m}}$$

Min ~~Max~~ Height

$$\boxed{h = \log_m [n(m-1) + 1] - 1}$$

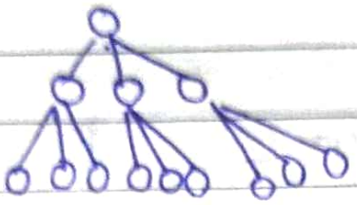
Internal vs External Nodes

Strict 3-ary tree:



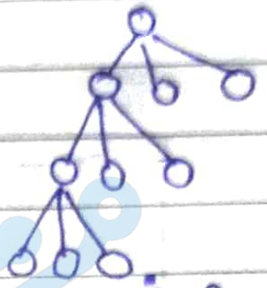
i = 2
e = 5

2×2+1 ⇒ 5



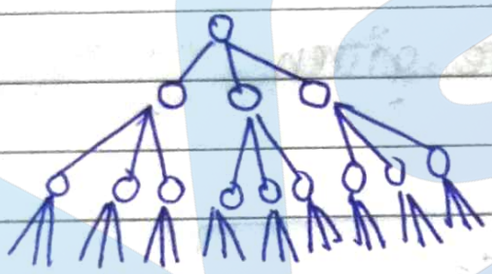
i = 4
e = 9

2×4+1 = 9



i = 3
e = 7

7 = 2×3+1



i = 13
e = 27

27 = 2×13+1

$$e = 2i + 1$$

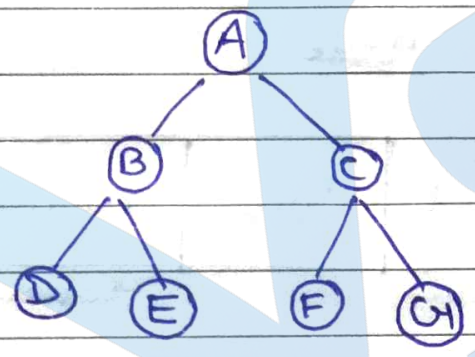
$$e = (m-1)i + 1$$

# Representation of Binary Tree

→ Array Representation

→ Linked Representation

- **Array Representation :-**




Store
→ All the element
→ Relationship between these element

| T | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

use of relationship

element → i

left child → 2i

Right child → 2*i+1

Parent → i/2

↓

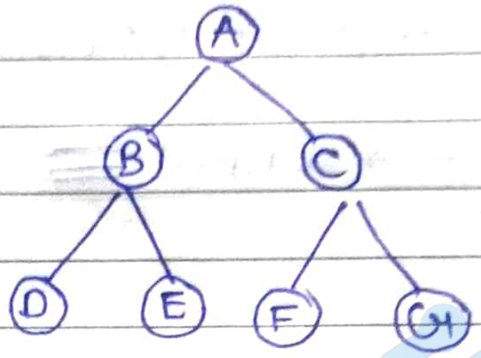
floor(i/2)

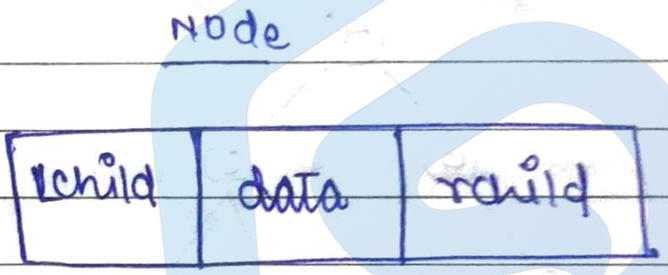
| element | index | L child | R child |
|---|---|---|---|
| A | 1 | 2 | 3 |
| B | 2 | 4 | 5 |
| C | 3 | 6 | 7 |
| | i | 2*i | 2*i+1 |

~~Linked List~~ Representation :-

- Linked ~~List~~ Representation :-

mostly used for implementation of a Tree

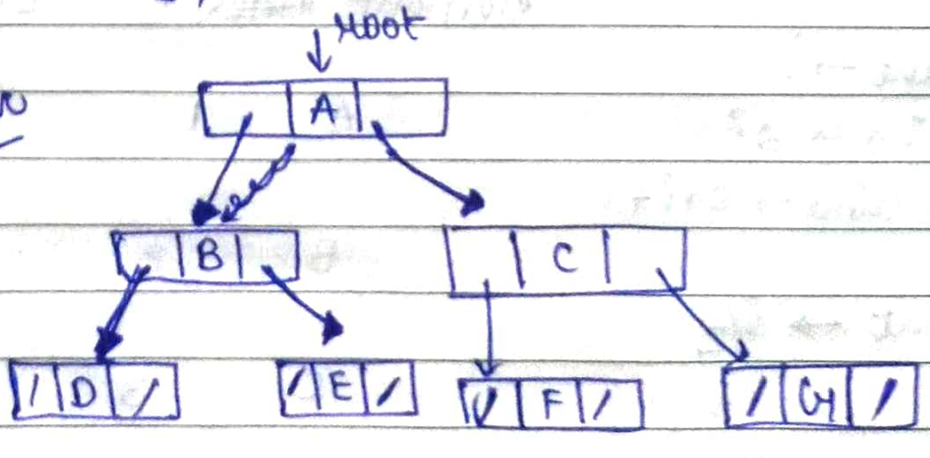



```
struct Node
{
    struct Node * lchild;
    int data;
    struct Node * rchild;
};
```

e = i + 1

dynamic

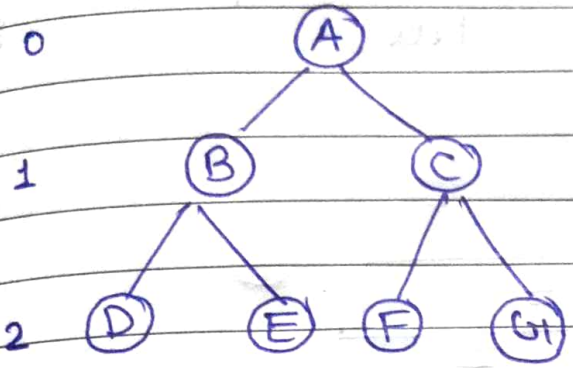



n = 7

Null pointer → n+1

# Full vs Complete Binary Tree

Full ↓




→ A Binary tree of Height h having maximum Number of Node is called Full Binary Tree

Max Node →

$h \rightarrow$

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

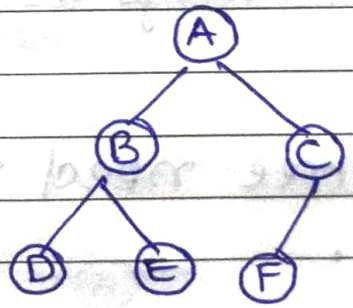
$n = 2^{2+1} - 1$

Max Nodes → $n = 7$

T

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

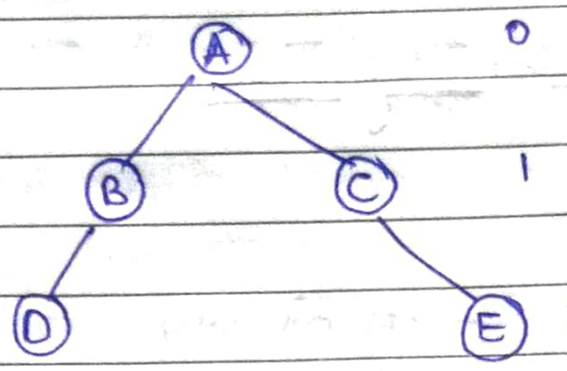
Complete ↓




T

| A | B | C | D | E | F |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |

A binary tree is represented in an array, then they should not be any blank spaces in betweeen them. If there are blank spaces, it's not accompanied by and there are no blank space, it is a complete binary Tree.




0

1 (Not a complete binary tree)

2

| T | A | B | C | D | - | - | E |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

A complete binary tree of Height H, will be full binary upto ~~h~~ h-1 hight and on the last level the element will be filled from left to right without skipping any elements.

Note :-

→ Full binary tree is always a complete binary tree.

→ complete binary tree need not be a full binary tree.

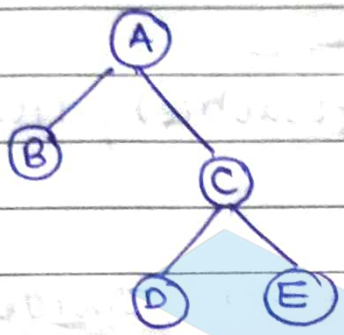
A complete binary tree is a type of binary tree in which all the levels are completely filled, except the possibly the deepest level.

Strict vs complete

Every node have either zero children or two children.

complete

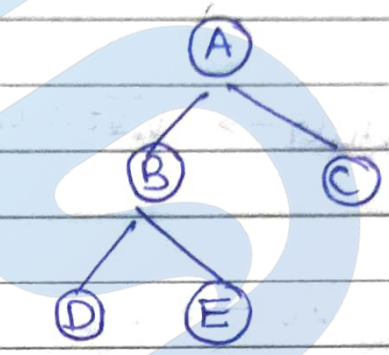
atmost complete




T | A | B | C | - | - | D | E

strict ✓

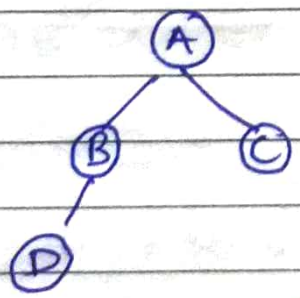
complete ×




T | A | B | C | D | E

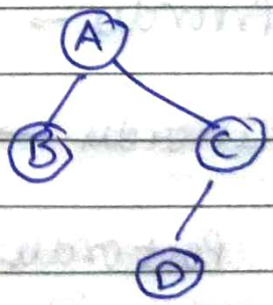
✓ strict

✓ complete




× strict

✓ complete

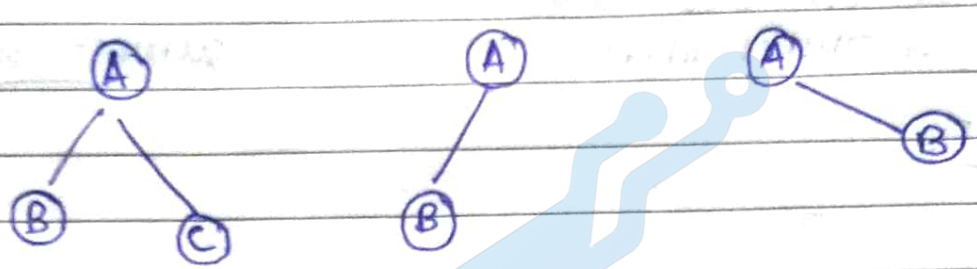



(x) strict

(x) complete

## Binary Tree traversals :-

### Tree traversals




Preorder → visit(node), Preorder(left subtree), Preorder(right subtree)

Inorder → Inorder(left), visit(node), Inorder(right)

Postorder → Postorder(left), Postorder(right), visit(node)

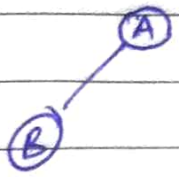
Level order → Level by level

Preorder → ABC

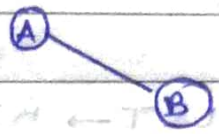
Inorder → BAC

Post order → BCA

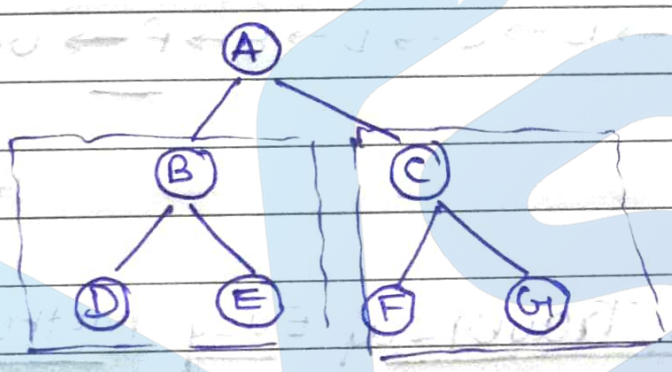
Level order → ABC




Pre → AB
In → BA
Post → BA
Level → AB




Pre → AB
In → ~~BA~~ AB
Post → BA
Level → AB




In ~~Preorder~~

LST → A → RST

LST → B → RST → A → RST

D → B → E → A → LST → C → RST

D → B → E → A → F → C → G

Preorder

A → LST → RST

A → ~~B~~ → LST → RST → RST

A → B → D → E → C → LST → RST

A → B → D → E → C → F → G

## Postorder

LST → RST → A

LST → RST → B → RST → A

D → E → B → LST → RST → C → A

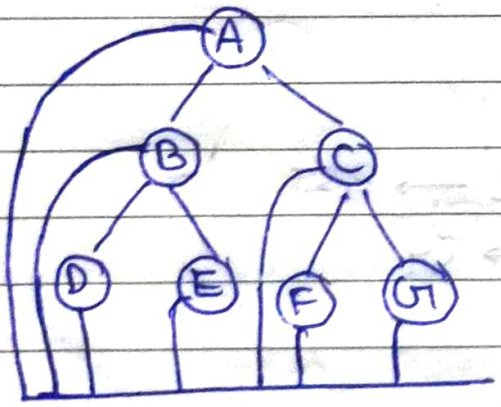
D → E → B → F → G → C → A

## Level order

A → B → C → D → E → F → G

## Binary Tree Traversal Easy Method 1 :-

Preorder

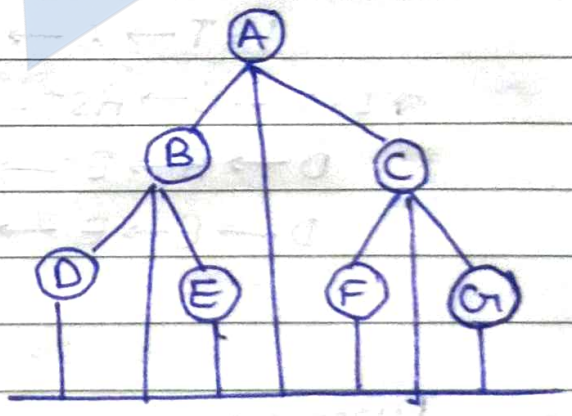



A → B → D → E → C → F → G

↓

Preorder

Inorder

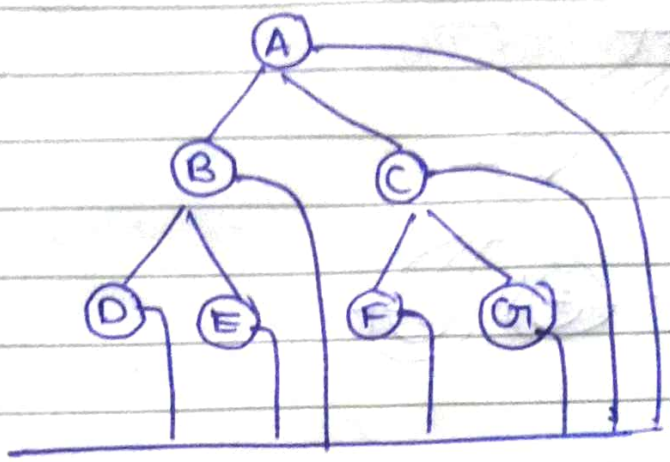



D → B → E → A → F → C → G

↓

Inorder

Postorder

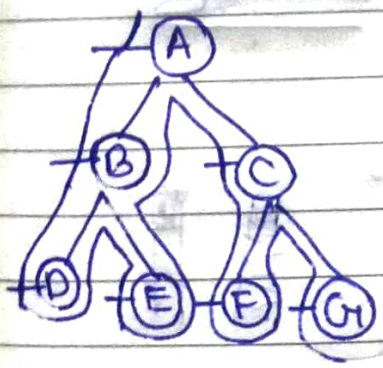



D → E → B → F → G → C → A

Postorder

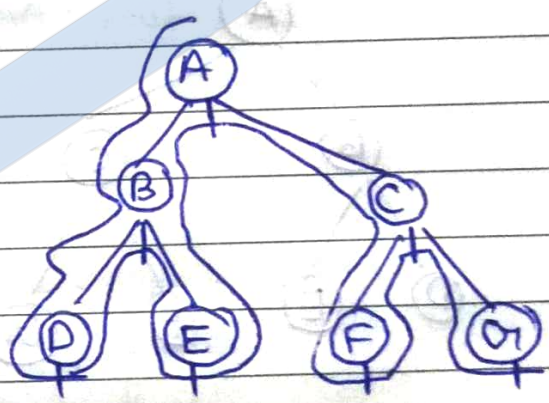
Binary Tree Traversal Easy method 2:-

Preorder



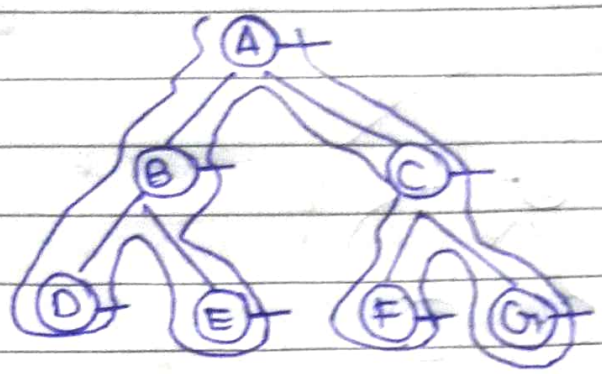

A → B → D → E → C → F → G

Preorder




D → B → E → A → C → F → G

~~Postorder~~ morder

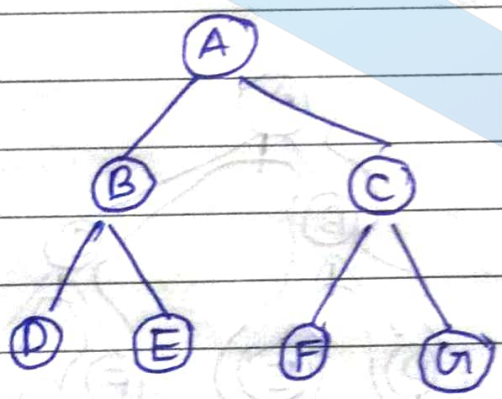



D → E → B → F → G → C → A

↑

Postorder

## Binary Tree Traversal Easy Method 3:-




Pre order →

morder

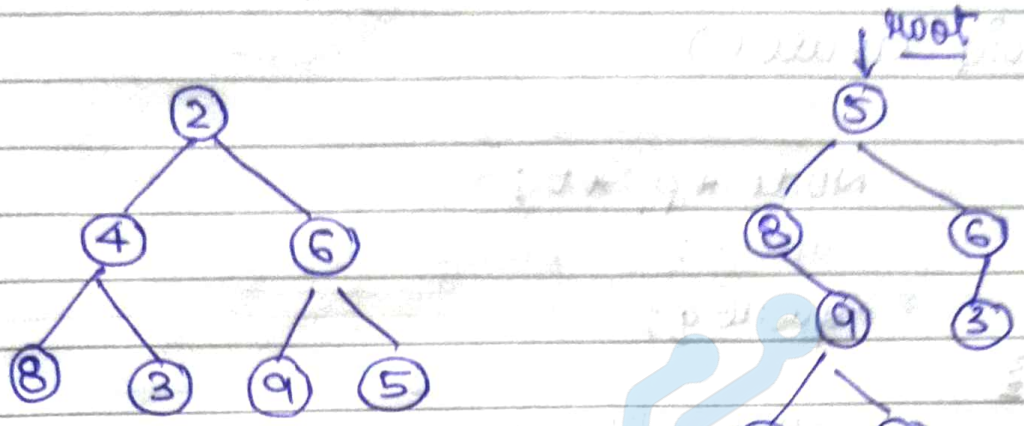
Pre → A → B → D → E → C → F → G

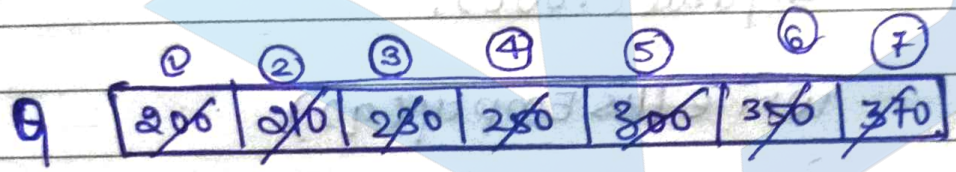
Postorder

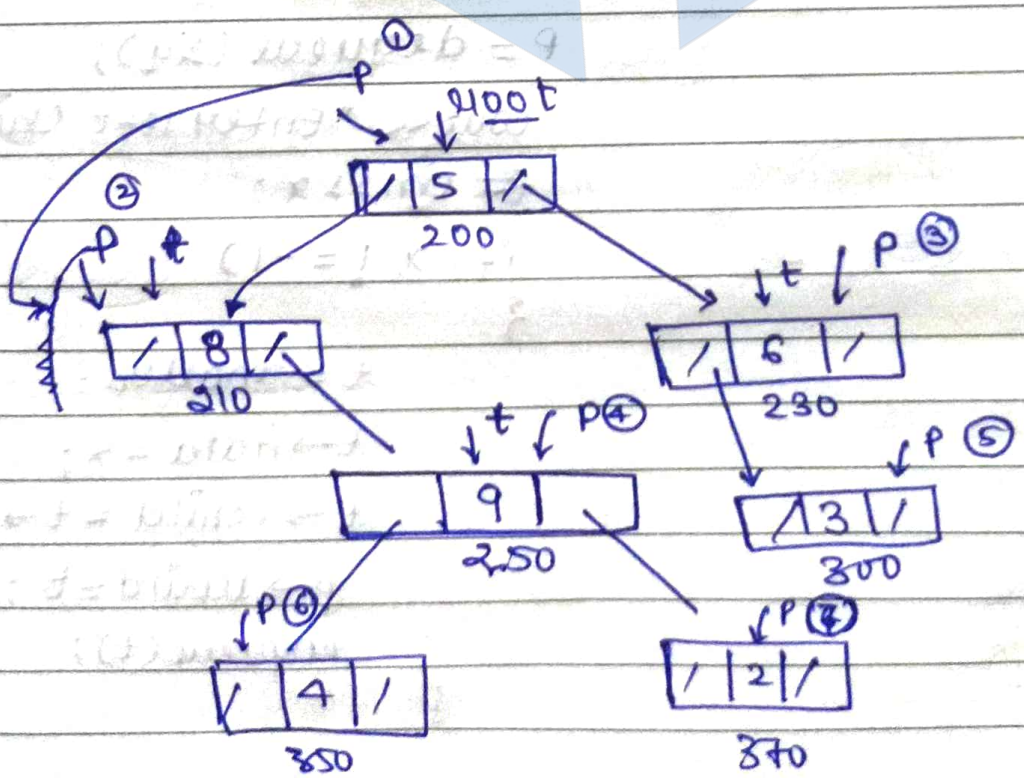
Inorder → D → B → E → A → F → C → G

Postorder → D → E → B → F → G → C → A

# Creating Binary Tree :-



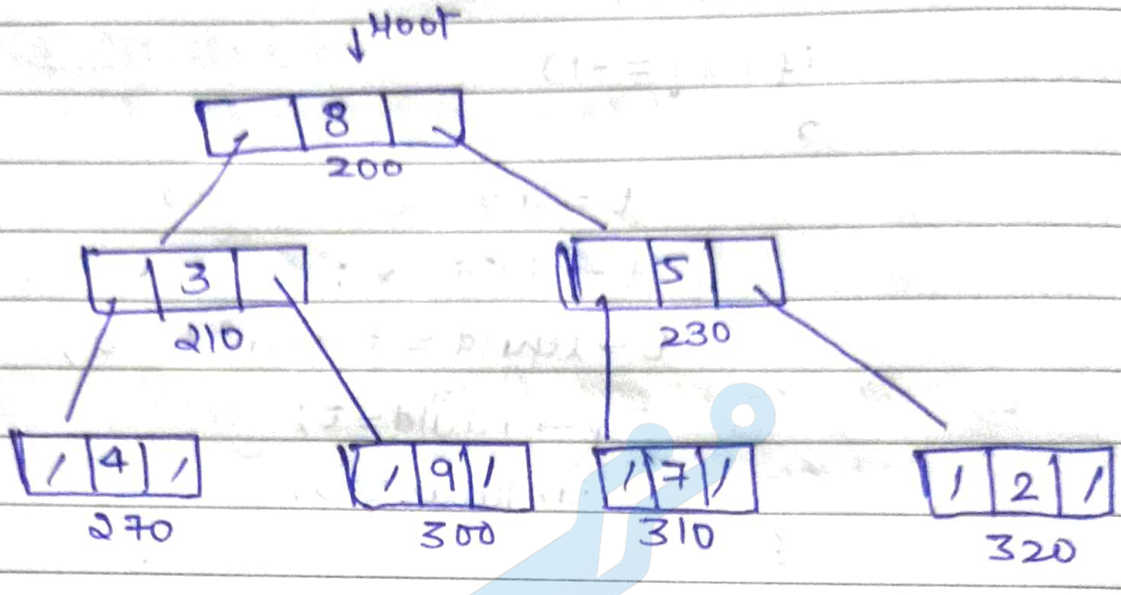



# Program to create Binary Tree

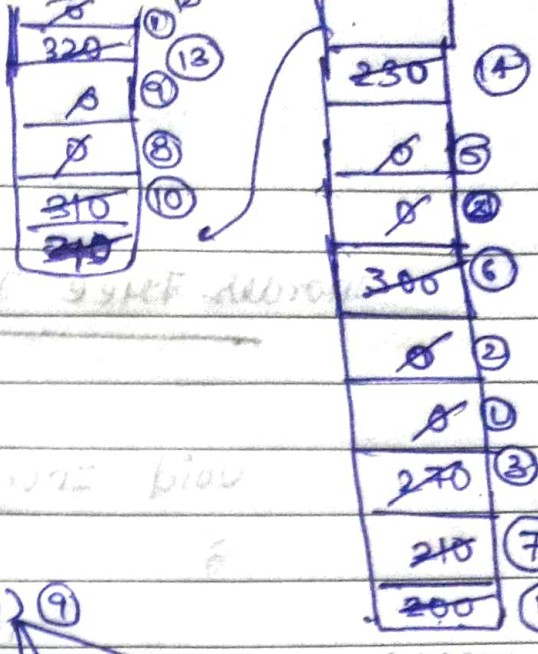
```
void create()
{
    Node *p, *t;
    int x;
    Queue q;
{
    cout<<"Enter root value";
    cin>>x;
    root = malloc(....)
    root->data = x;
    root->lchild = root->rchild = 0;
    enqueue(root);

    while(!isEmpty(q))
    {
        p = dequeue(&q);
        cout<<"Enter left child";
        cin>>x;
        if(x != -1)
        {
            t = malloc(...)
            t->data = x;
            t->lchild = t->rchild = 0;
            p->lchild = t;
            enqueue(t);
        }
```

```
cout << "Enter right child : ";
cin >> x;
if (x != -1)
{
        t = malloc ----- )
      t -> data = x;
      t -> lchild = t -> rchild = 0;
       p -> rchild = t;
       enqueue (t);
}

}

}
```

# Preorder Tree Traversal




```
void Preorder (Node *t)
{
    if (t != NULL)
    {
        cout << t->data << " ";
        preorder (t->lchild);
        preorder (t->rchild);
    }
}
```







8 → 3 → 4 → 9 → 5 → 7 → 2

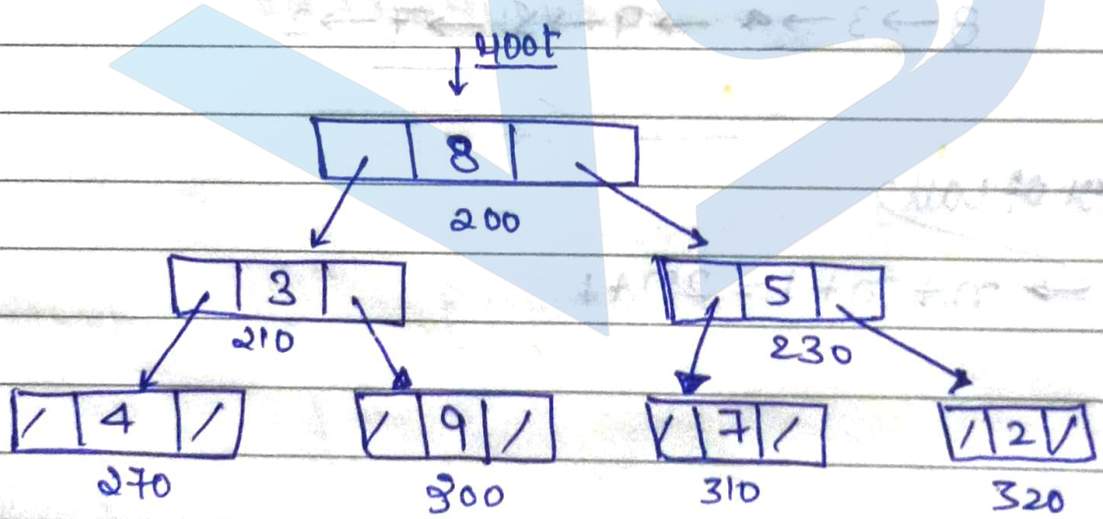
Number of call

→ n + n + 1 = 2n + 1

T.C → O(n)

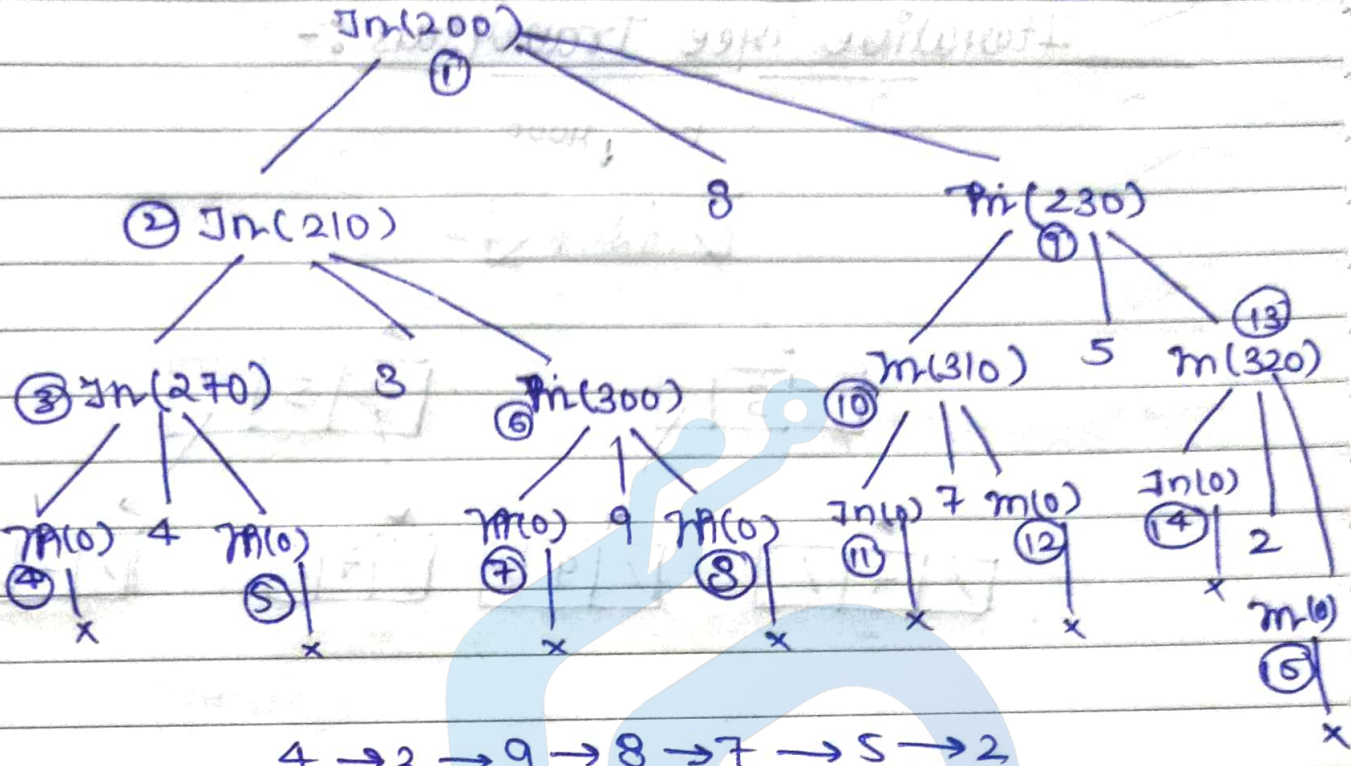
## Inorder Tree Traversals :-

```
void Inorder(Node *t)
{
    if(t!=NULL)
    {
        Inorder(t->lchild);
        cout<<t->data;
        Inorder(t->rchild);
    }
}
```



No. of call → 2n+1

T.C → O(n)

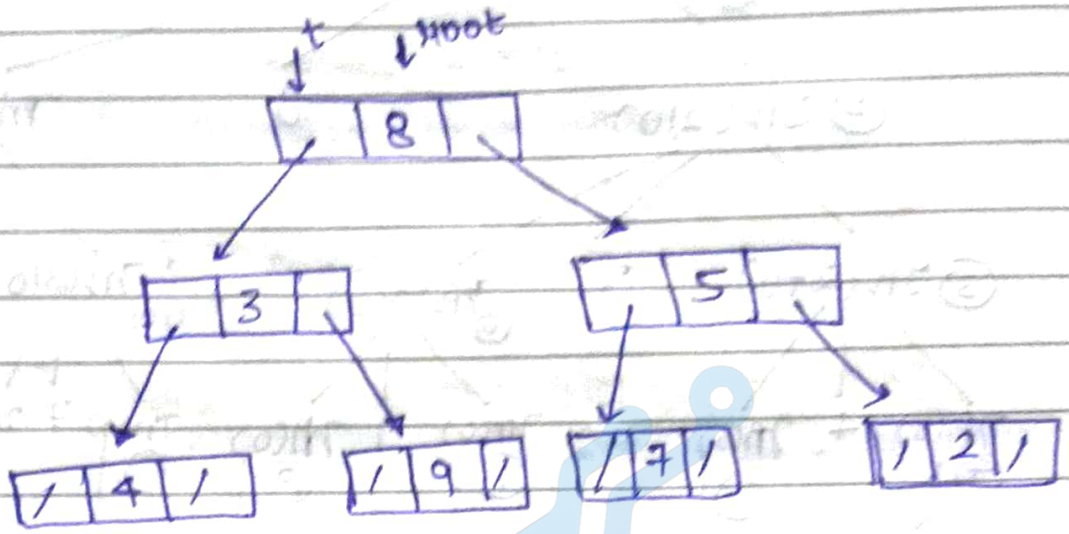



4 → 3 → 9 → 8 → 7 → 5 → 2

## Postorder Traversals :-

```
void Postorder(Node *t)
{
    if (t != NULL)
    {
        Postorder(t->Lchild);
        Postorder(t->rchild);
        cout<<t->data;
    }
}
```

# Iterative Tree Traversals :-




Iterative Tree Traversal

- Preorder
- Postorder
- Inorder

8, 3, 4, 9, 5, 7, 2

Iterative Preorder →

```
void Preorder(Node *t)
{
    struct stack st;

    while(t!=NULL || !isEmpty(st))
    {
        if(t!=NULL)
        {
            cout<<t->data;
            push(&st,t);
            t=t->lchild;
        }
```

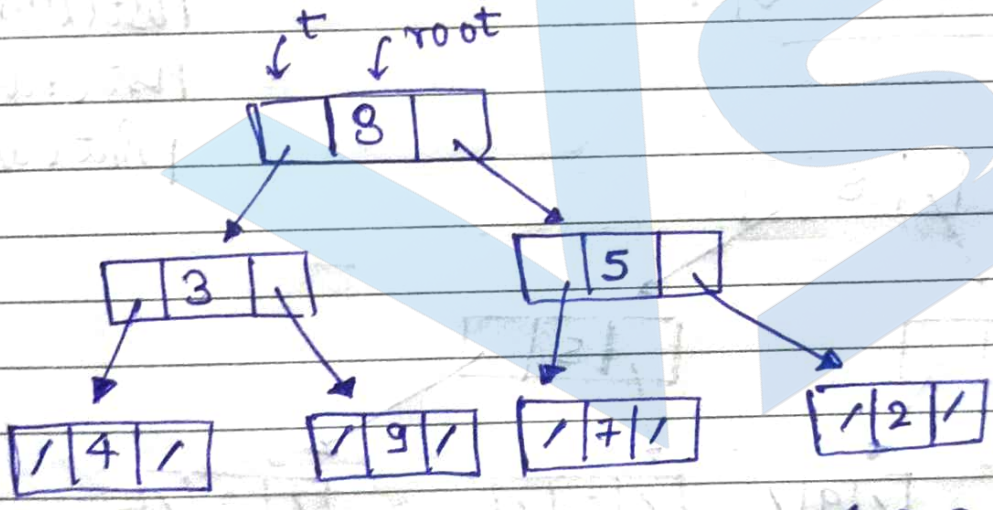
Add(2) ①
Add(7) ⑤
Add(5) ⑥
Add(9) ③
Add(4) ②
Add(3) ④
Add(8) ⑦

```
        else
        {
            t = pop(&st);
            t = t->rchild;
        }
    }
}
```

T.C → O(n)

S.C → O(Height of tree)

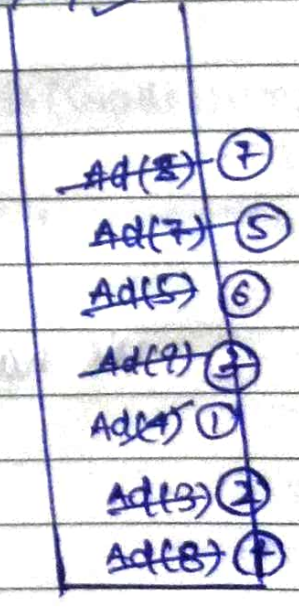
## Iterative Inorder :-




In(lchild)
print(data)
In(rchild)

Output → 4, 3, 9, 8, 7, 5, 2

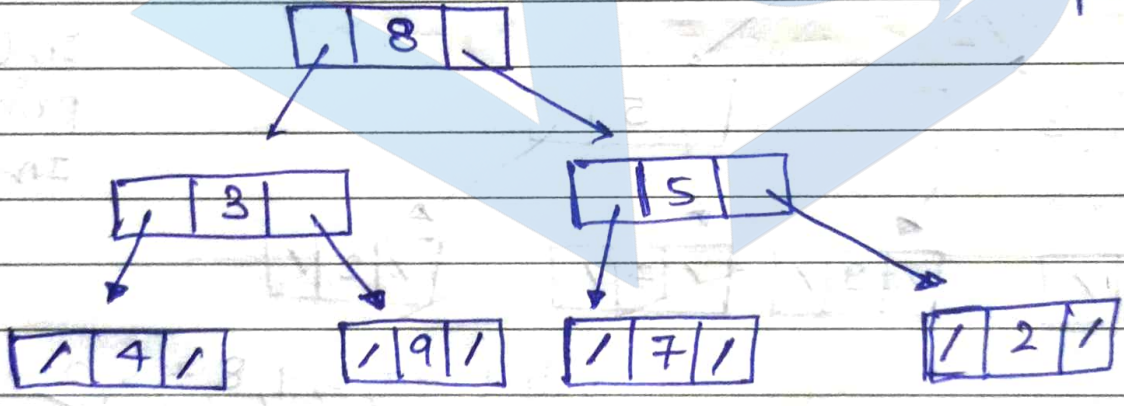
```
void Inorder(Node *t)
{
    struct stack st;
    while (t!=NULL || !isEmpty(st))
    {
        if (t!=NULL)
        {
            push(&st, t);
            t=t->lchild;
        }
```



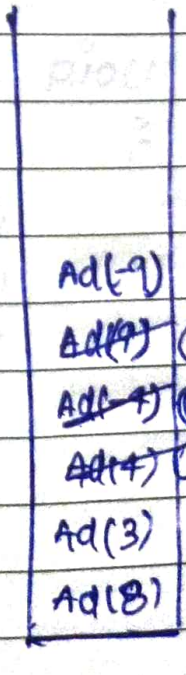
```
else
{
        t=pop(st);
        cout<<t->data;
        t=t->rchild;
    }
}
}
```

Iterative Postorder :-

```
Post(lchild)
Post(rchild)
print(data)
```




O/p :→ 4

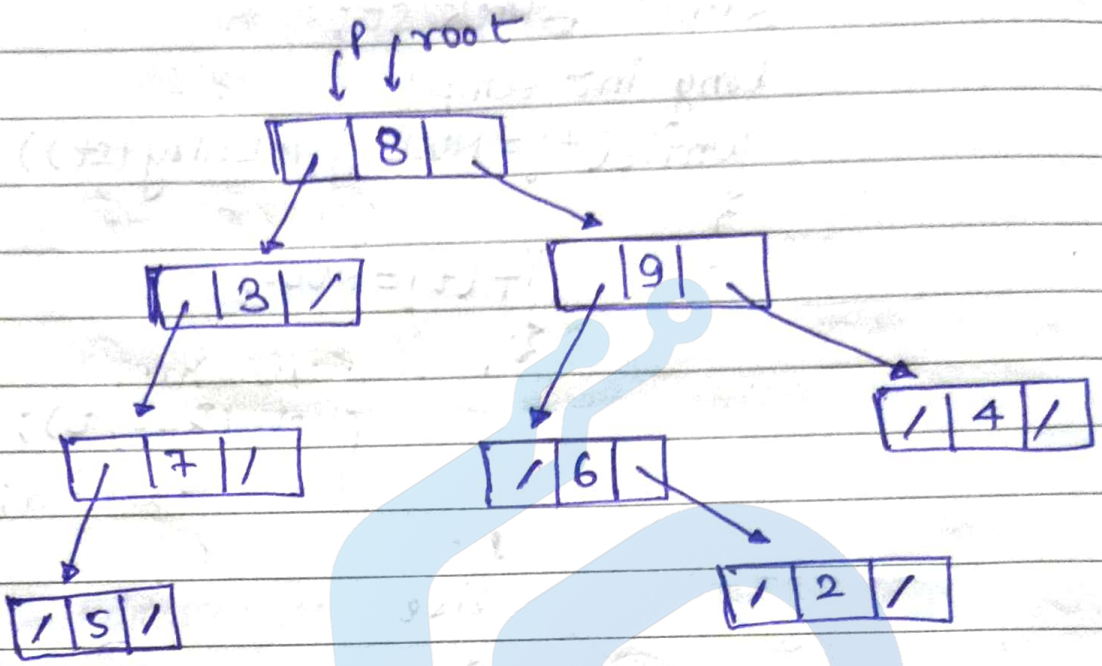



We have to push the address ~~of~~ two times,

↳

```
void Postorder(Node *t)
{
    Stack st;
    long int temp;
    while(t!=NULL || !isEmpty(st))
    {
        if(t!=NULL)
        {
            push(&st, t);
            t = t->lchild;
        }
        else
        {
            temp = pop(&st);
            if(temp>0)
            {
                push(&st, -temp);
                t=((Node *)temp)->rchild;
            }
            else
            {
                cout<<((Node *)temp)->data;
                t=NULL;
            }
        }
    }
}
```

## Level order Traversal :-




O/p :- 8, 3, 9, 7, 6, 4, 5, 2

Q | Add(8) | ~~Add(3)~~ | ~~Add(9)~~ | ~~Add(7)~~ | ~~Add(6)~~ | ~~Add(4)~~ | ~~Add(5)~~ | ~~Add~~

O/p → 8, 3, 9, 7, 6, 4, 5, 2

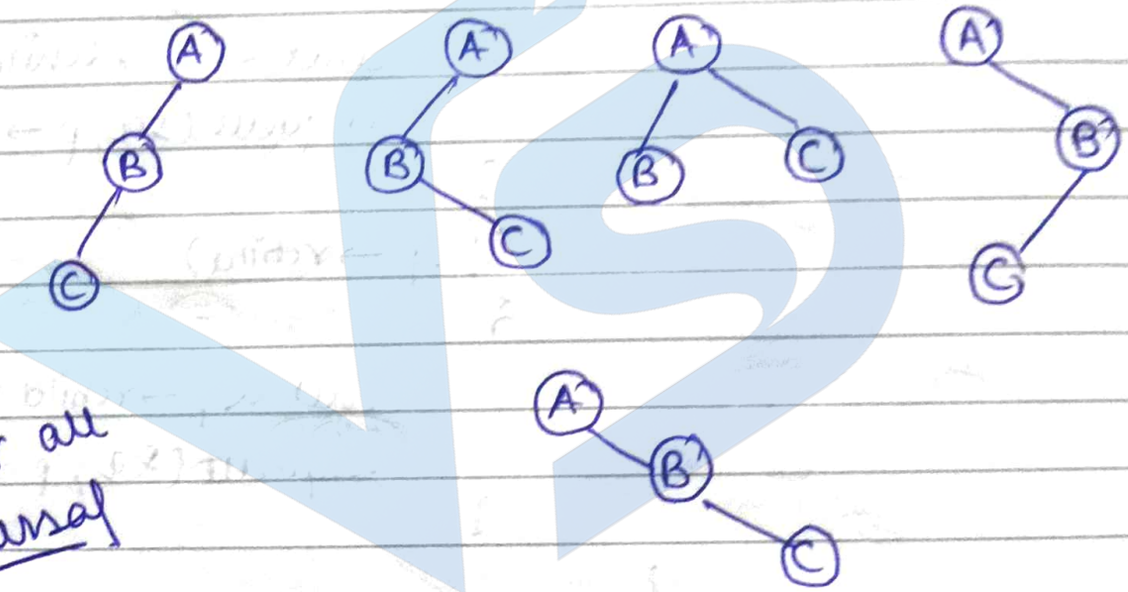
```
void Levelorder(Node *p)
{
    Queue q;
    cout<<p->data;
    enqueue(&q,p);
    while(!isEmpty(q))
    {
        p=dequeue(&q);
        if(p->lchild)
        {
            cout<<p->lchild->data;
            enqueue(&q,p->lchild);
        }
        if(p->rchild)
        {
            cout<<p->rchild->data;
            enqueue(&q,p->rchild);
        }
    }
}
```

Can we generate Tree from Traversals :-

$n = 3$

(A) (B) (C)

preorder → A, B, C — $\frac{2nC_n}{n+1}$



This is True for all Traversal

Conclusion, if only preorder is given, we can generate tree.

↓

Inorder

↓

Postorder

Preorder → A, B, C

Postorder → C, B, A

① Preorder
② Postorder
③ Inorder
] — $\frac{2nCn}{n+1}$

[ Preorder
Postorder ] → 1+

This traversal give unique tree using traversal.

① preorder + Inorder

② postorder + Inorder

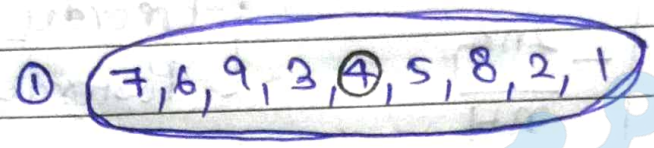
# Generating Tree from Traversals

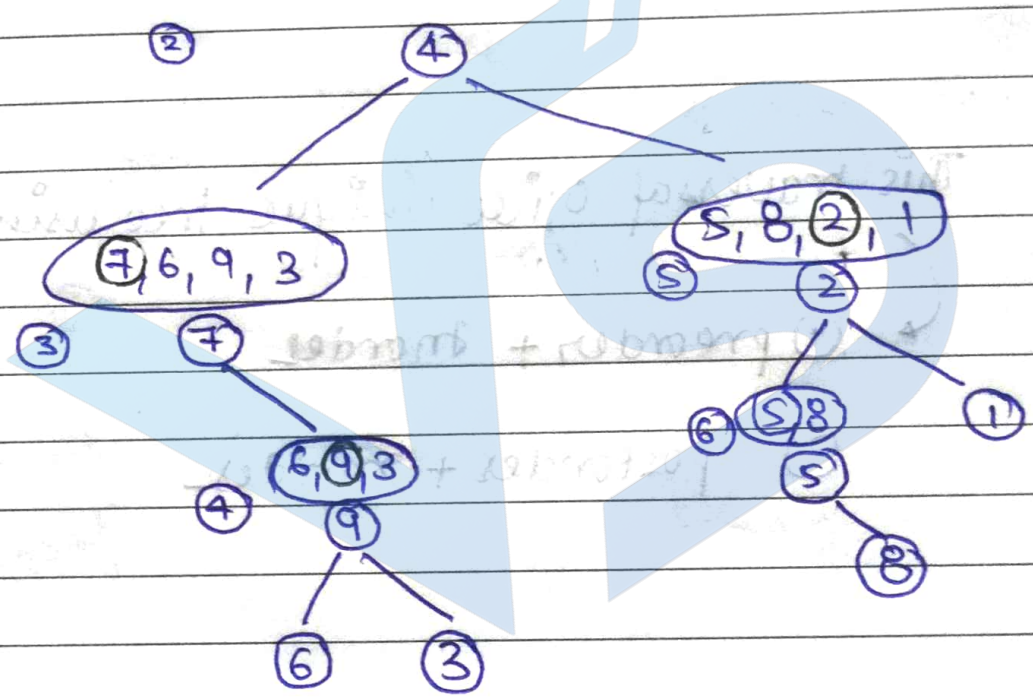
Preorder - 4, 7, 9, 6, 3, 2, 5, 8, 1 → n

Inorder - 7, 6, 9, 3, 4, 5, 8, 2, 1 → n × n = n²

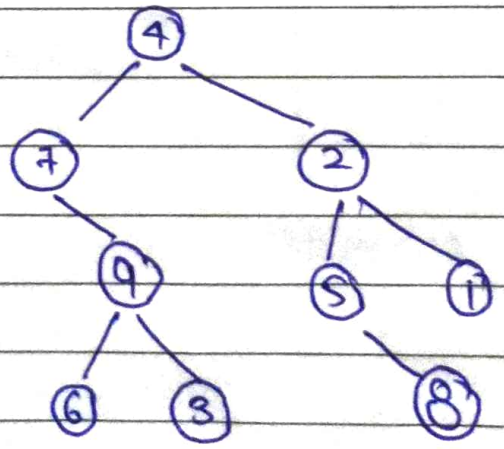
of Search

T.C → n²



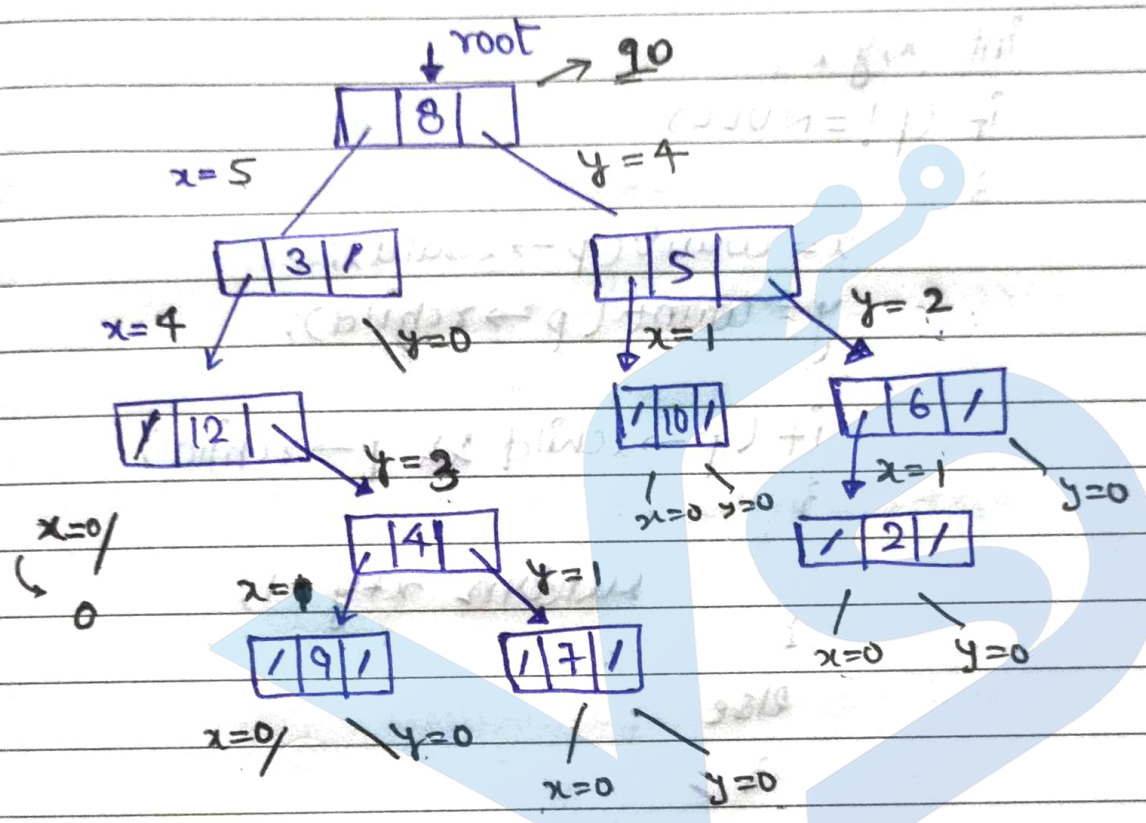




Tree ↓



# Height and count of Binary Tree :-

## Counting Nodes




```
int count(Node *p)
{
    int x, y;
    if (p != NULL)
    {
        x = count(p->lchild);
        y = count(p->rchild);
        return x+y+1;
    }
    return 0;
}
```

Counting the node with degree two

```
int count(NODE *p)
{
    int x,y;
    if(p!=NULL)
    {
        x=count(p->lchild);
        y=count(p->rchild);
        if(p->lchild && p->rchild)
        {
            return x+y+1;
        }
        else
        {
            return x+y;
        }
    }
    return 0;
}
```

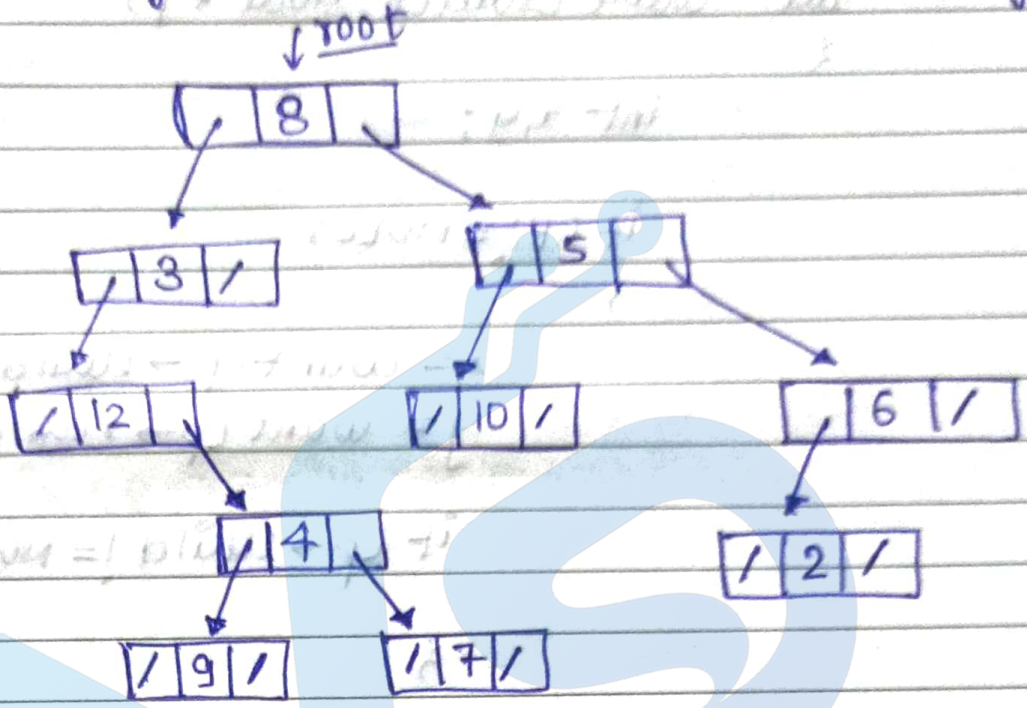
Sum of all the element of Tree :-

```
int sum(Node *p)
{
    int x,y;
    if (p!=NULL)
    {
        x = count(p->lchild);
        y = count(p->rchild);

        return x+y+p->data;
    }
    return 0;
}
```

## Height of the Tree :-

```
int fun(Node *p)
{
    int x,y;
    if(p!=NULL)
    {
        x=fun(p->lchild);
        y=fun(p->rchild);
        if(x>y)
            return x+1;
        else
            return y+1;
    }
    return 0;
}
```

# Student challenge :- count leaf Nodes of a Binary Tree




for degree 0 :-

```
int count (struct Node *p)
{
    int x,y;
    if(p!=NULL)
    {
        x=count(p->lchild);
        y=count(p->rchild);

        if(p->lchild==NULL && p->rchild
                                ==NULL)
        {
            return x+y+1;
        }
        else
            return x+y;
    }
    return 0;
}
```

for Degree 2 and 1 :-

```
int count(struct Node * p)
{
    int x,y;

    if(p!=NULL)
    {
        x= count(p->lchild);
        y= count(p->rchild);

        if(p->lchild != NULL && p->rchild
                                != NULL)
        {
            return x+y+1;
        }
        else
        {
            return x+y;
        }
    }
    return 0;
}
```

for degree :-

1 Leaf Nodes

if (!p → lchild && !p → rchild)

2. Node deg 2

if (p → lchild && p → rchild)

3. deg 1 or 2

if (p → lchild != NULL || p → rchild != NULL)

4. deg 1

if ((p → lchild != NULL && p → rchild == NULL)

|| (p → lchild == NULL && p → rchild
!= NULL)

$L \oplus R = LR' + L'R$

if (p → lchild ⊕ p → rchild)

if ( p → lchild != NULL ∧ p → rchild != NULL)

## Count Node in a given tree

```
int count (struct Node *p)
{
    if (p==NULL)
    {
        return 0;
    }

    return count(p->lchild)+ count(p->rchild)
                                        +1;
}
```