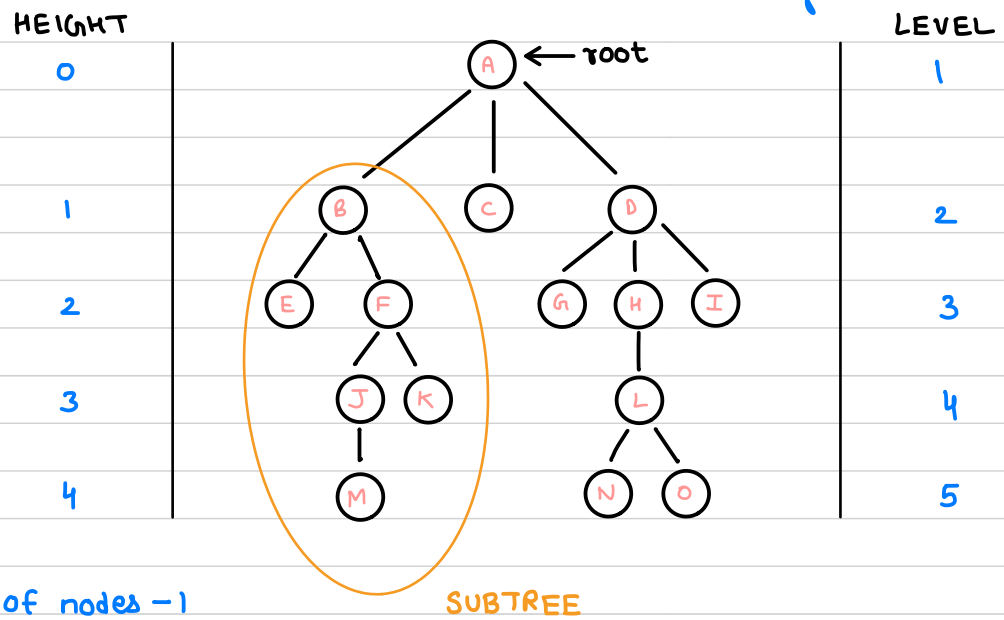
TREES → Tree is a collection of nodes and edges.



No of edges = No of nodes − 1

TERMINOLOGY

(1) **Root** : The first node on the top

(2) **Parent** : A node is a parent to its very next descendents.

(3) **Child**

(4) **Sibling** : Children of same parent G, H, I

(5) **Descendents** : Set of all those nodes which can be reached from a particular node.
E, F, J, K, M are descendents of B.

(6) **Ancestors** : For any node, all the nodes along the path from that node to root node
For M, J, F, B, A are ancestors.

(7) **Degree of a node** : No of children it is having L − 2

(8) **Internal Nodes / Non-Leaf Nodes / Non-Terminal Nodes** : Nodes with degree > 0

External Nodes / Leaf-Nodes / Terminal Nodes : Nodes with degree 0

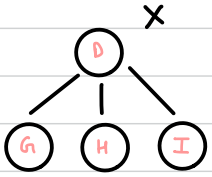
(9) **Levels** (10) **Height**

(10) **Forest** : Collection of trees.

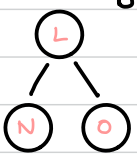
BINARY TREE

degree {2}

children – 0, 1, 2



X

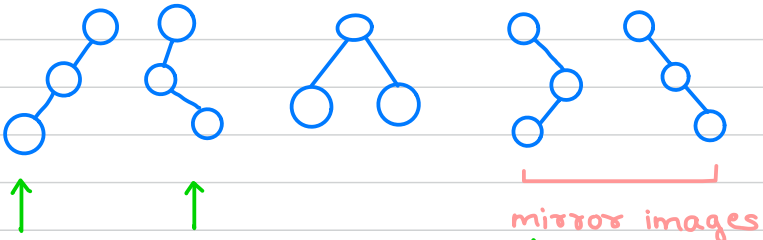


BINARY TREE ✓

NUMBER OF BINARY TREES USING N NODES

(1) Unlabelled Nodes
(2) Labelled Nodes

n = 3



height = 2

mirror images

T(3) = 5

No of trees with maximum height = $4 = 2^2$ (2)

n = 4

T(4) = 14

+ other 7 mirror images

No of trees with maximum height = $8 = 2^3$

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

Catalan Number

No of trees with maximum height = 2^{n-1}

2nd Method for finding Catalan Number

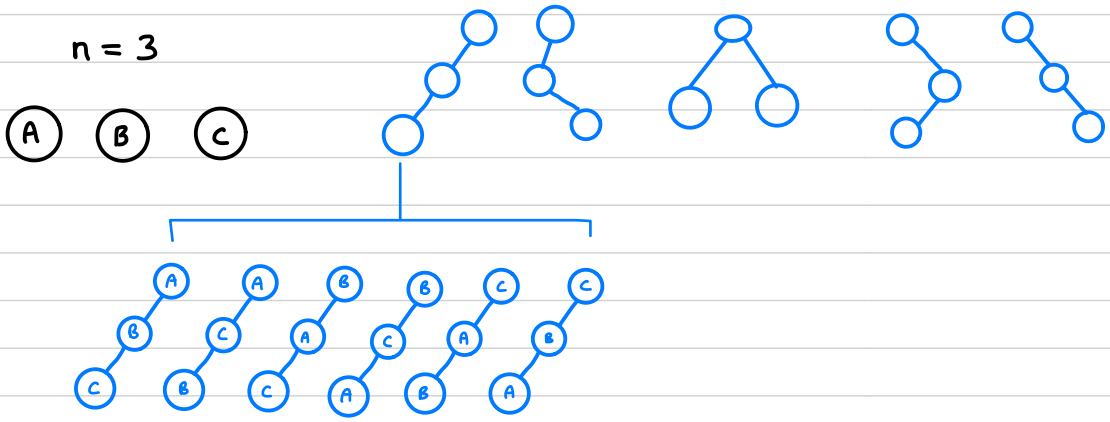
n	0	1	2	3	4	5	6
$T(n) = \frac{^{2n}C_n}{n+1}$	1	1	2	5	14	42	

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

$$T(6) = T(0) \times T(5) + T(1) \times T(4) + T(2) \times T(3) + T(3) \times T(2) + T(4) \times T(1) + T(5) \times T(0) = 132$$

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

(2) Labelled Nodes



$6 = 3!$

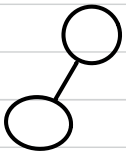
$$T(n) = \frac{^{2n}C_n}{n+1} * n!$$

Shapes — Filling

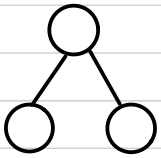
Height vs Node

Height
h = 1

min

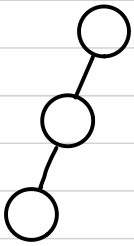


max

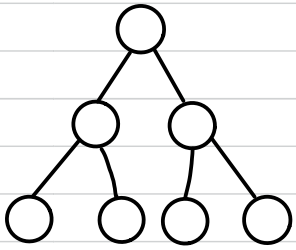


Height
h = 2

min

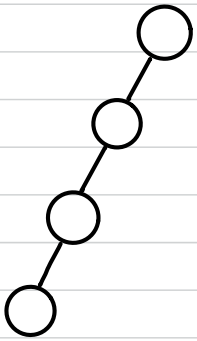


max

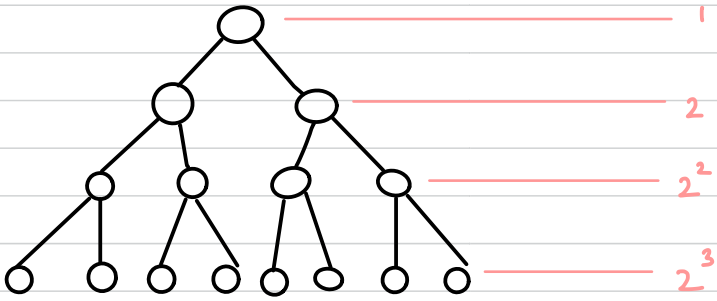


Height h = 3

min



max



$1 + 2 + 2^2 + 2^3 = 15$

Minimum Number of Nodes

min nodes = h+1

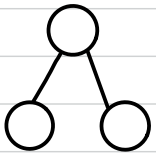
Maximum Number of Nodes

$$a + ar + ar^2 + ar^3 + ar^k = \frac{a(r^{k+1}-1)}{r-1}$$

$$1 + 2 + 2^2 + 2^3 + \ldots\ldots 2^h = \frac{1 \cdot (2^{h+1}-1)}{2-1} = 2^{h+1}-1$$

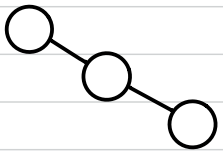
Nodes $n = 3$

min



$h = 1$

max



$h = 2$

If nodes are given

(1) Max height $h = n-1$

↳ Can be obtained by previous formula

(2) For minimum height

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

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

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

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

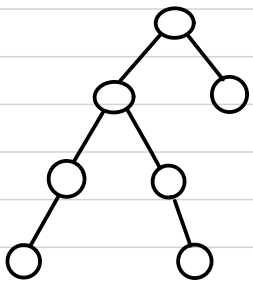
Height of a binary tree

$$\underset{O(\log n)}{\log_2(n+1) - 1} \leq h \leq \underset{O(n)}{n-1}$$

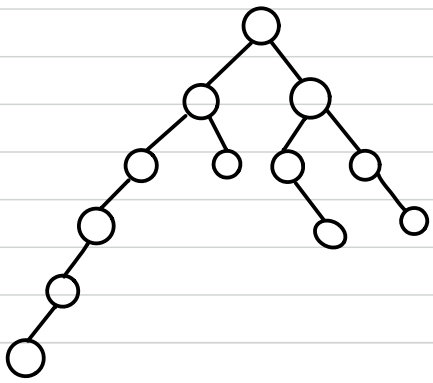
Number of Nodes in a binary tree

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

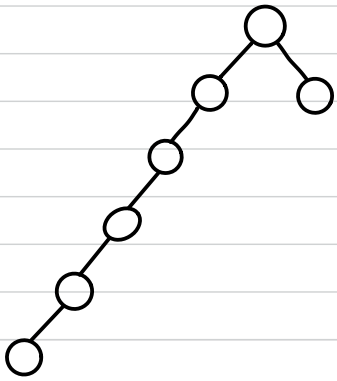
INTERNAL NODES VS EXTERNAL NODES IN A BINARY TREE



deg(2) = 2
deg(1) = 2
deg(0) = 3



deg(2) = 3
deg(1) = 5
deg(0) = 4

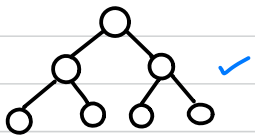


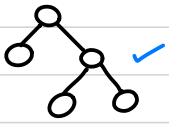
deg(2) = 1
deg(1) = 4
deg(0) = 2

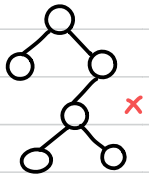
$deg(0) = deg(2) + 1$ True in binary tree

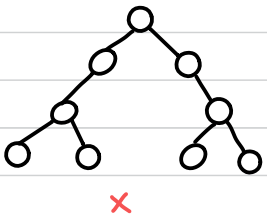
STRICT / PROPER / COMPLETE BINARY TREES

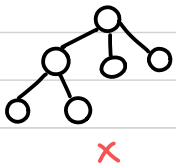
deg = { ✓ 0, ✗ 1, ✓ 2 }

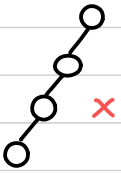








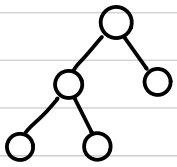




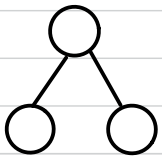
Height vs Node (Strict Binary Tree)

Height
h = 1

min

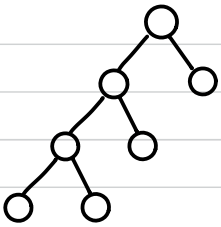


max

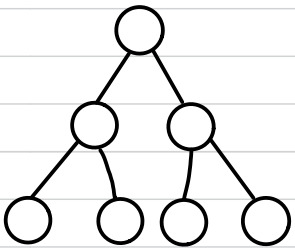


Height
h = 2

min

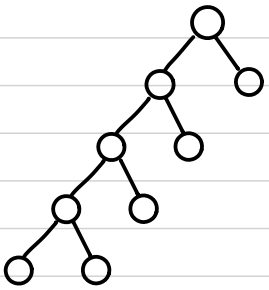


max

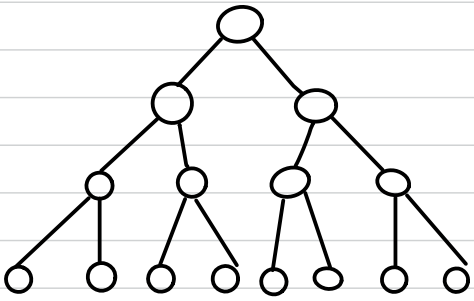


Height h = 3

min



max



Only minimum number changed

If height 'h' is given

Min Nodes $n = 2h+1$

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

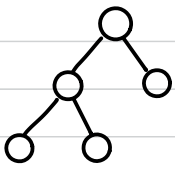
If 'n' nodes are given

Minimum height $h = \log_2(n+1)-1$

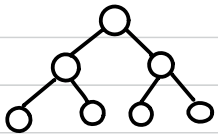
Max height $h = \frac{n-1}{2}$

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

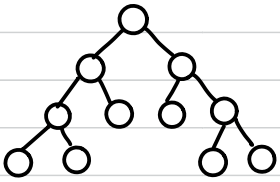
INTERNAL NODES VS EXTERNAL NODES IN A BINARY TREE (STRICT)



i = 2
e = 3



i = 3
e = 4

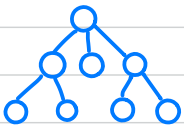


i = 5
e = 6

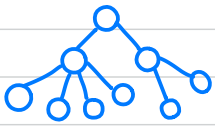
$e = i+1$

n-ary Trees

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



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

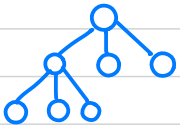


Strict 3-ary Tree {0,3}

Strict n-ary trees

h = 2

min



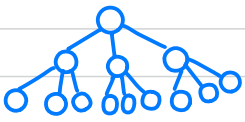
$n = 3+3+1$

$n = 2 \times 3 + 1$

$i = 2$

$e = 5$

max



$i = 5$

$e = 9$

If height 'h' is given

Min Nodes $n = nh+1$

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

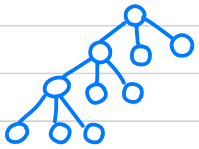
If 'n' nodes are given

Minimum height $h = \log_m[n(m-1)+1]-1$

Max height $h = \frac{n-1}{m}$

h = 3

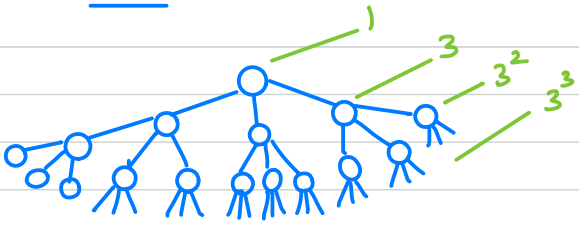
min



$i = 3$

$e = 7$

max



$\Rightarrow 1 + 3 + 3^2 + 3^3$

$\Rightarrow 1 + n + n^2 + n^3$

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

$i = 13$

$e = 27$

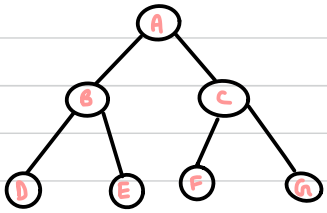
$e = 2i+1$

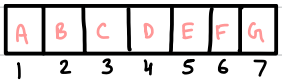
$\boxed{e = (n-1)i+1}$

REPRESENTATION OF BINARY TREE

1. Array Representation
2. Linked Representation

1. Array Representation





Element	Index	L. Child	R. Child
A	1	2	3
B	2	4	5
C	3	6	7
	i	2 * i	2 * i + 1

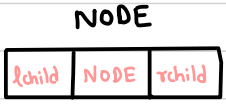
Element ⟶ i

L Child ⟶ 2 * i

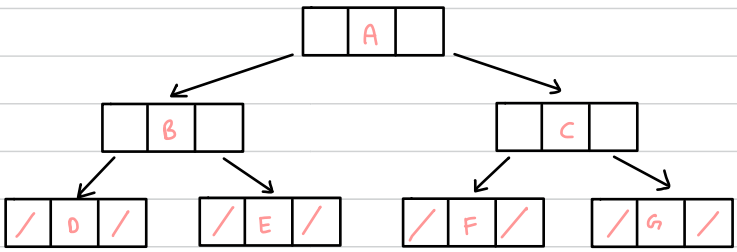
R Child ⟶ 2 * i + 1

Parent ⟶ $\left\lfloor \frac{i}{2} \right\rfloor$

2. Linked Representation




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

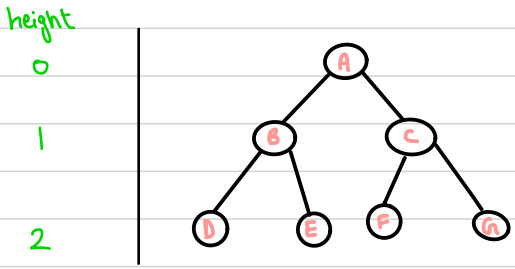


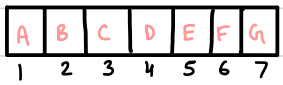
n = 7

nulls = n + 1

Full Binary Tree: A binary tree of height h having maximum number of nodes.

Eg

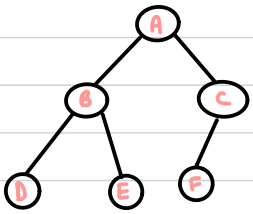


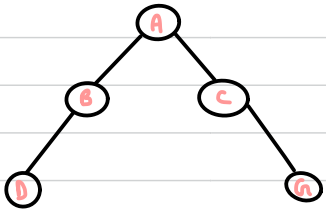


Complete Binary Tree: A complete binary tree of height h will be a full binary tree upto height h-1

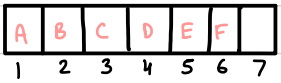
A full binary tree is always a complete binary tree.

Eg

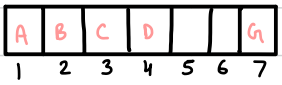




NOT A COMPLETE BINARY TREE



NO BLANK SPACES BETWEEN ELEMENTS

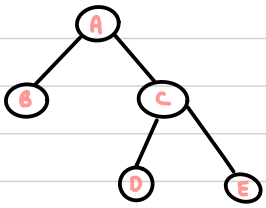


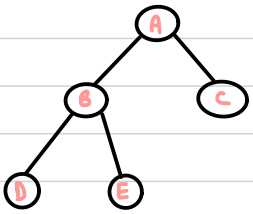
BLANK SPACES

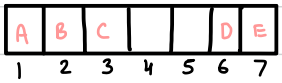
degree {0, 2}

STRICT VS COMPLETE

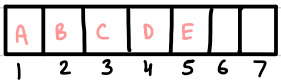
- Complete
- almost complete







Strict ✓
complete ✗



Strict ✓
complete ✓

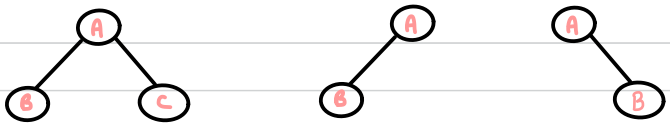
TREE TRAVERSALS

Preorder : NLR
Node , Left , Right

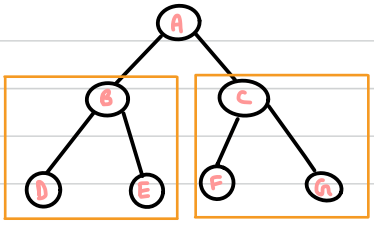
Inorder : LNR
Left , Node , Right

Postorder : LRN
Left , Right , Node

Level Order : Level by Level



Preorder	ABC	AB	AB
Inorder	BAC	BA	AB
Postorder	BCA	BA	BA
Level	ABC	AB	AB



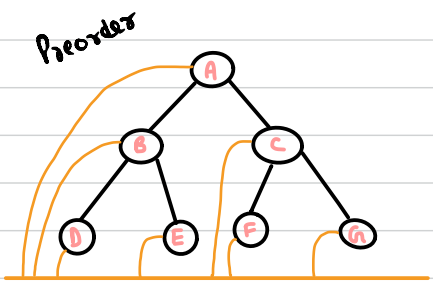
Preorder A , (B, D, E) , (C, F, G)
A, B, D, E , C , F, G

Inorder (D, B, E), A, (F , C , G)
D, B, E , A , F , C, G

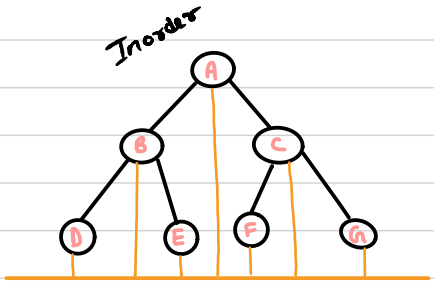
Postorder (D, E , B) , (F, G , C) , A
D , E , B, F , G , C, A

Level A , B, C , D, E, F, G

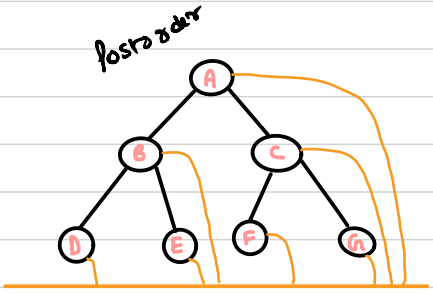
BINARY TREE TRAVERSAL EASY METHOD 1



A, B, D, E, C, F, G

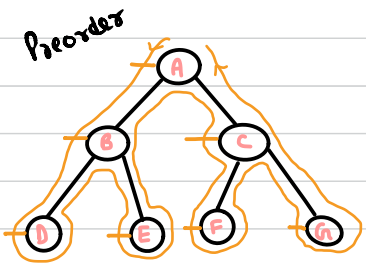


D, B, E, A, F, C, G

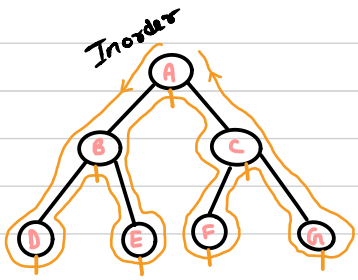


D, E, B, F, G, C, A

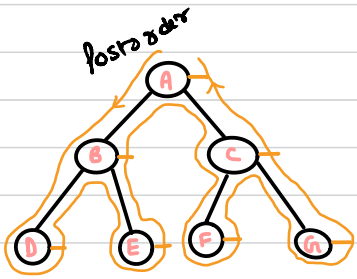
BINARY TREE TRAVERSAL EASY METHOD 2



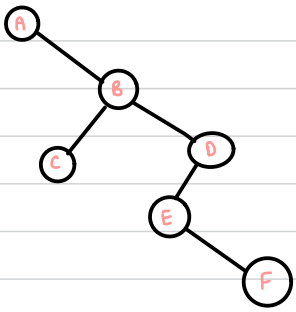
A, B, D, E, C, F, G



D, B, E, A, F, C, G



D, E, B, F, G, C, A



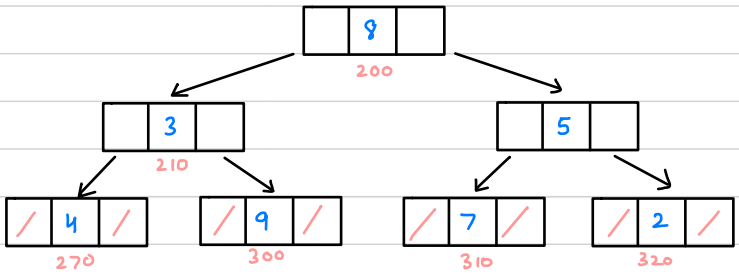
Preorder : A, B, C, D, E, F
Inorder : A, C, B, E, F, D
Postorder : C, F, E, D, B, A

PROGRAM TO CREATE BINARY TREE

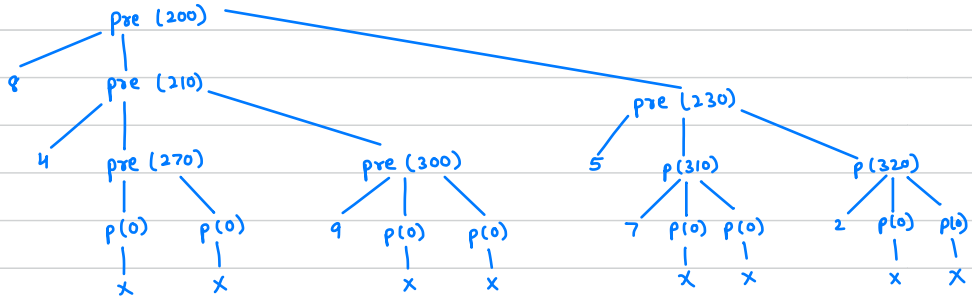
```
void create()
{
    Node *p, *t;
    int x;
    Queue q;        initializing a queue
    printf("Enter root value");
    scanf("%d", &x);
    root = malloc(.....);        initializing root
    root -> data = x;
    root -> lchild = root -> rchild = 0;
    enqueue(root);    Storing address of root in queue

    while(!isEmpty(q))
    {
        p = dequeue(&q);    take out a value from queue into p
        printf("Enter left child");
        scanf("%d", &x);

        if(x != -1)
        {
            t = malloc(....);
            t -> data = x; t -> lchild = t -> rchild = 0;
            p -> lchild = t;      p -> rchild = t      for right child everything
            enqueue(t);                                else will be same
        }
    }
}
```

RECURSIVE PREORDER TREE TRAVERSAL

```
void Preorder (Node *t)
{
    if (t != NULL)
    {
        printf ("%d", t -> data);
        Preorder (t -> lchild);
        Preorder (t -> rchild);
    }
}
```



TRACING TREE



ACTIVATION RECORDS ARE CREATED USING STACK

RECURSIVE INORDER TREE TRAVERSAL

```
void Inorder (Node *t)
{
    if (t != NULL)
    {
        Inorder (t -> lchild);
        printf ("%d", t -> data);
        Inorder (t -> rchild);
    }
}
```

RECURSIVE POSTORDER TREE TRAVERSAL

```
void Postorder (Node *t)
{
    if (t != NULL)
    {
        Postorder (t -> lchild);
        Postorder (t -> rchild);
        printf ("%d", t -> data);
    }
}
```

ITERATIVE PREORDER TRAVERSAL

```
void Preorder ( Node *t)
{
        Struct stack st;
                                  to check for empty stack
        while ( t! = NULL || isEmpty (st) )
        {
                if ( t! = NULL)
                {
                        printf(" %d ", t → data);
                        push ( &st , t ); push address of t in stack
                        t = t → lchild;
                }

                else
                {
                        t = pop ( &st ); pop out address
                        t = t → rchild; from stack
                }
        }
}
```

ITERATIVE INORDER TRAVERSAL

```
Inorder ( Node *t)
                        push ( &st , t );
                        t = t → lchild;

                        t = pop ( &st );
                        printf(" %d ", t → data);
                        t = t → rchild;
```

ITERATIVE POSTORDER TRAVERSAL*

```
void   Inorder ( Node *t)
{
        Struct 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
                        {
                                printf("%d",((Node *) temp)
                                                → data)
                                t= NULL;
                        }
                }
        }
}
```

LEVEL ORDER TRAVERSAL

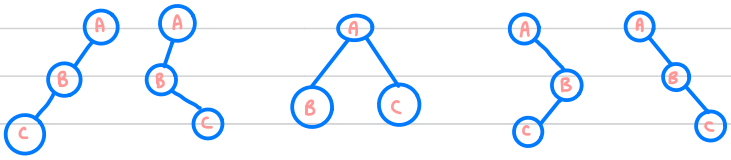
```
void Levelorder (Node *p)
{
        Queue q;
        printf("%d", p->data);
        enqueue (&q, p);

        while( ! isEmpty (q))
        {
                p = dequeue (&q);
                if (p->lchild)
                {
                        printf("%d", p->lchild->data);
                        enqueue (&q, p->lchild);
                }

                if (p->rchild)
                {
                        printf("%d", p->rchild->data);
                        enqueue (&q, p->rchild);
                }
        }
}
```

CAN WE GENERATE TREE FROM TRAVERSALS ?

n = 3

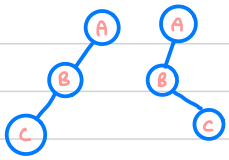
① Given preorder - A, B, C

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



② Given preorder A, B, C
posorder C, B, A

more than one



CONCLUSION

If we are given either preorder, inorder, postorder we cannot generate unique tree.

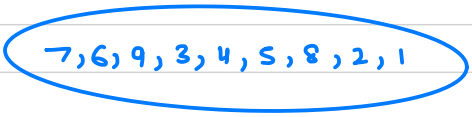
If we are given both preorder and postorder, then also unique tree cannot be generated

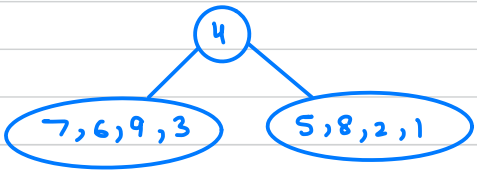
① preorder + inorder
② postorder + inorder

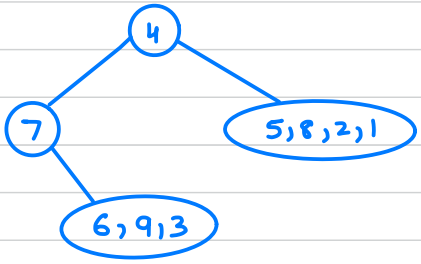
for generating unique tree

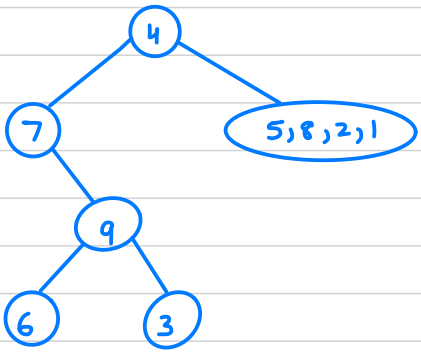
Q Preorder – 4,7,9,6,3,2,5,8,1
Inorder – 7,6,9,3,4,5,8,2,1

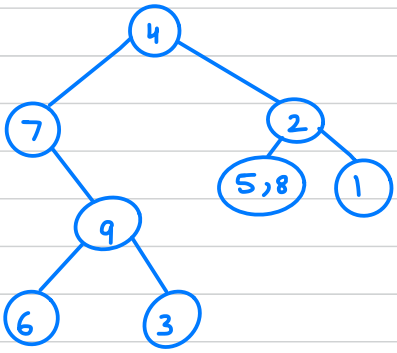
Sol



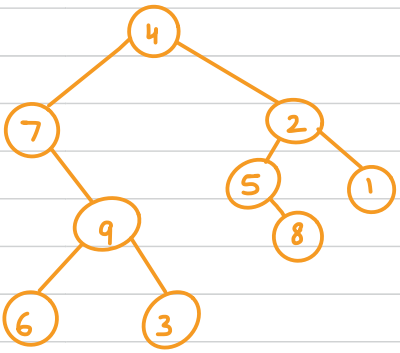






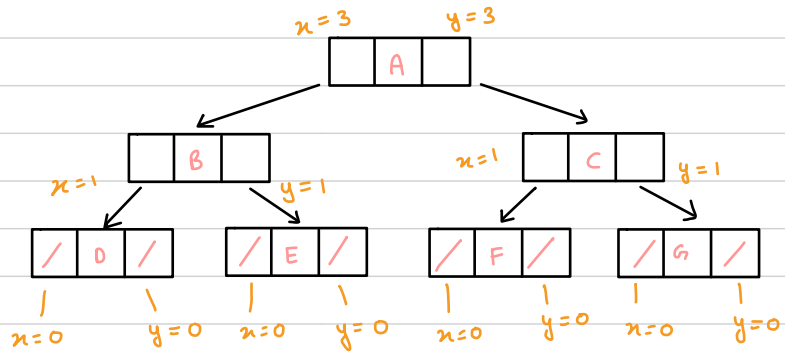


Unique Tree



COUNTING NO OF NODES IN A BINARY TREE

```
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 LEAF NODES

```
int count (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;
    }
    else
        return 0;
}
```

COUNTING NODES WITH DEGREE 2

COUNTING NODES WITH DEGREE 2 OR 1

COUNTING NODES WITH DEGREE 1

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

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

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

XOR

```
if (p → lchild != NULL ^ p→rchild != NULL)
```