CS & IT ENGINEERING

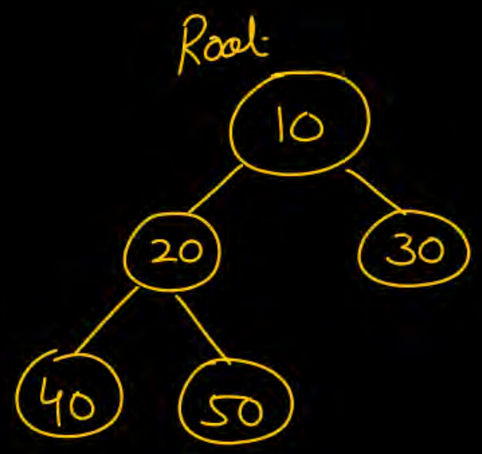
Data Structure & Programming

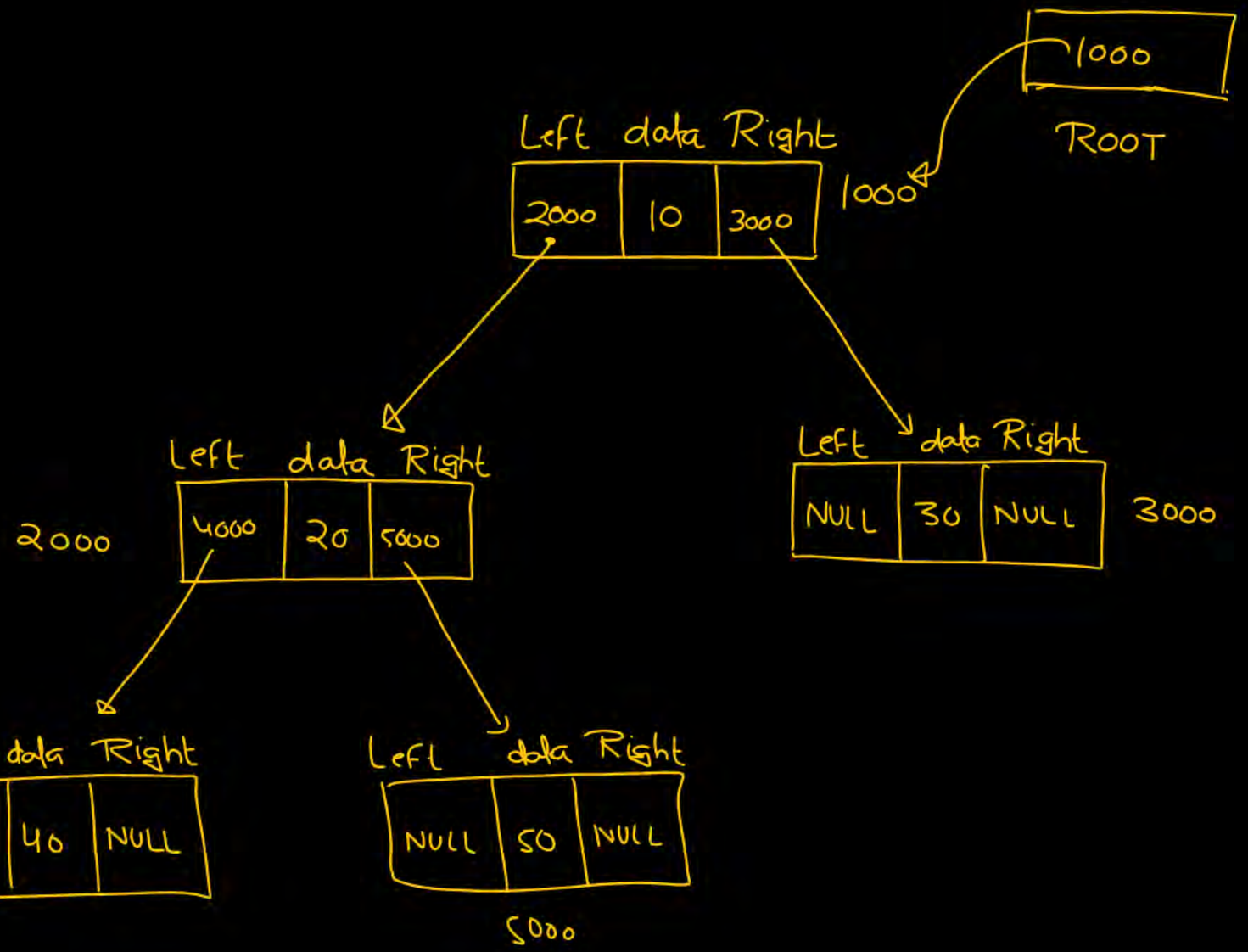
Tree

Lec - 02

By- Pankaj Sharma sir

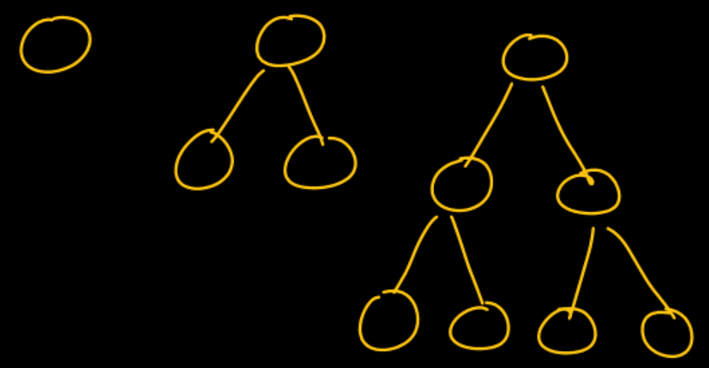
Tree-2





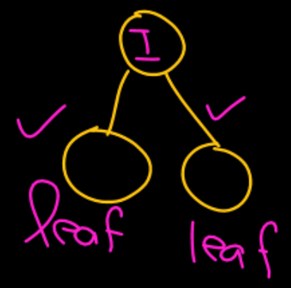

```
struct Node {
    struct Node *Left;
    int data;
    struct Node *Right;
};
struct Node * ROOT;
```

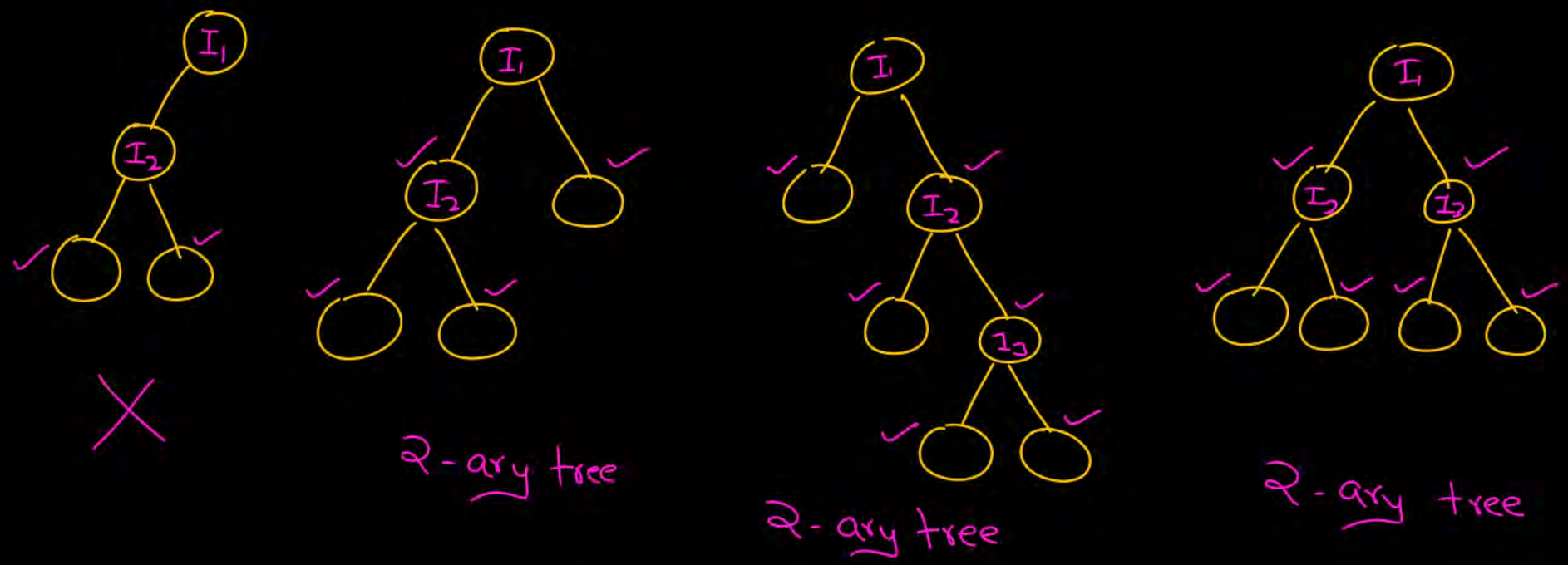
Full Binary Tree :



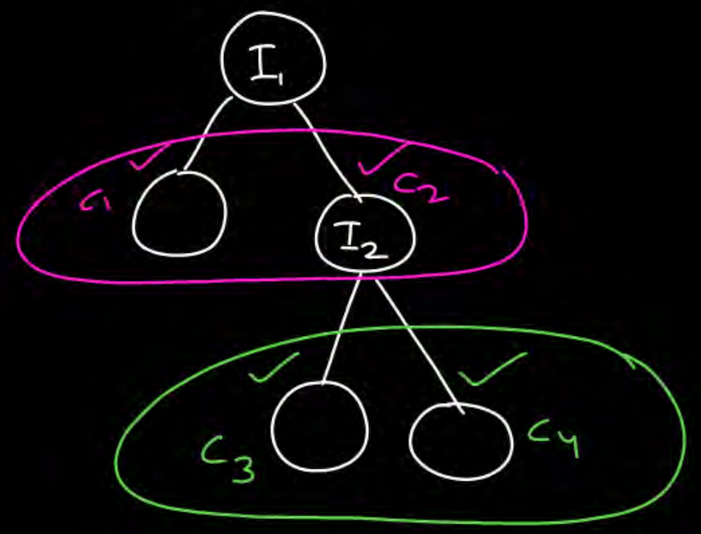
2-ary tree :

A tree in which every internal node has exactly 2-childs.




I_1
I_2
I_1
I_2
2-ary tree
I_1
I_2
I_3
2-ary tree
I_1
I_2
I_3
2-ary tree

2-ary tree



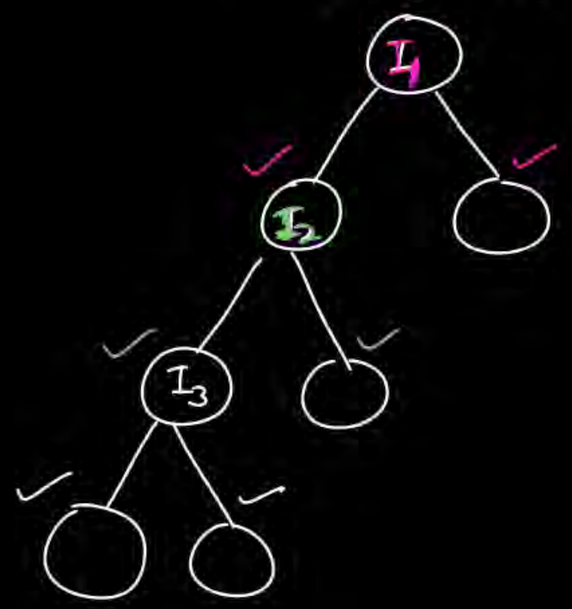
Each internal node has 2 childs

Total no. of node $\Rightarrow$ $2 \times 2 + 1$

Root

No. of internal node

2-ary tree



Total nodes = $3 \times 2 + 1$ (root)

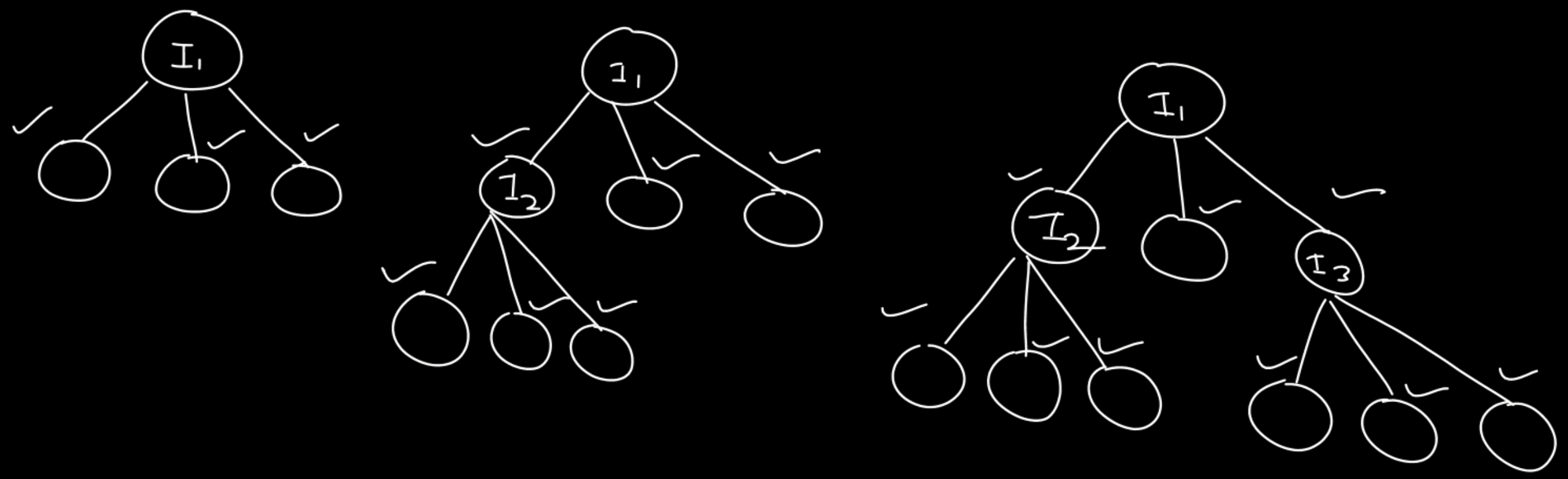
$$\overset{I_1}{2} + \overset{I_2}{2} + \overset{I_3}{2}$$

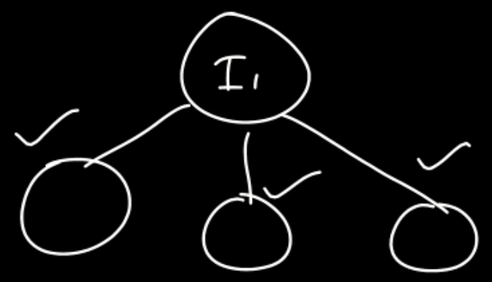
$$= 3 \times 2$$

3 internal nodes

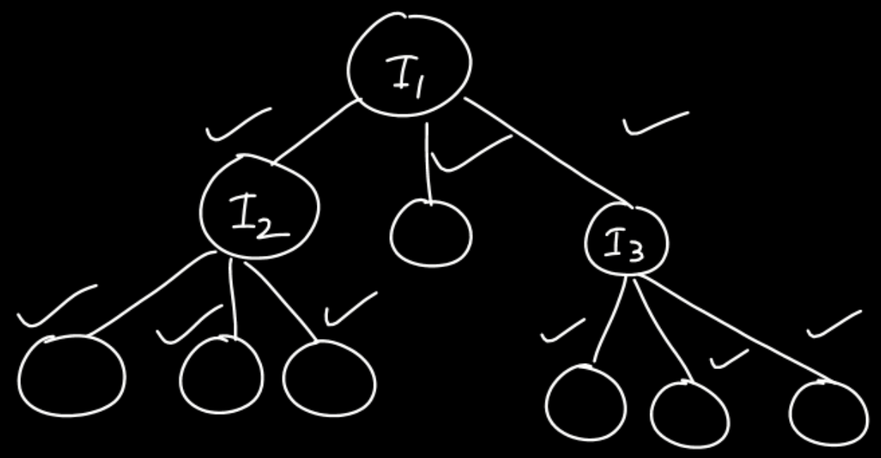
Every internal node has 2 childs

3-ary tree: A tree in which every internal node has exactly 3 childs





$$\text{Total nodes} = \underbrace{1}_{\text{no. of internal nodes}} \times \underbrace{3}_{\text{Each internal node is having 3-childs}} + \overset{\text{Root}}{1}$$



no. of internal node = 3

nodes (childs) b/c of these internal nodes = 3×3

Total nodes = $3 \times 3 + 1$

K-ary tree : A tree in which every internal node has K-childs.

let I is the no. of internal nodes in a K-ary tree.

$$\text{Total nodes} = \underbrace{I}_{\text{no. of internal nodes}} \times \underbrace{K}_{\text{Each internal node has K-childs}} + \overset{\text{(root)}}{1}$$

$$\Rightarrow \boxed{n = K \cdot I + 1}$$

$$n = K \cdot I + 1$$

# of Leaf nodes + # of internal node $= K \cdot I + 1$

$$L + I = K \cdot I + 1$$

$$L = K \cdot I - I + 1$$

$$\boxed{L = (K-1)I + 1}$$

$$n = k \cdot I + 1 \quad \text{— (1)}$$

$$L = (k-1)I + 1 \quad \text{— (2)}$$

$$\Downarrow$$

$$L - 1 = (k-1)I$$

$$\boxed{I = \frac{L-1}{k-1}}$$

$$n = k\left(\frac{L-1}{k-1}\right) + 1$$

$$n = \frac{kL - \cancel{k} + \cancel{k} - 1}{k-1}$$

$$\boxed{n = \frac{k \cdot L - 1}{k-1}}$$

Don't try to by-heart

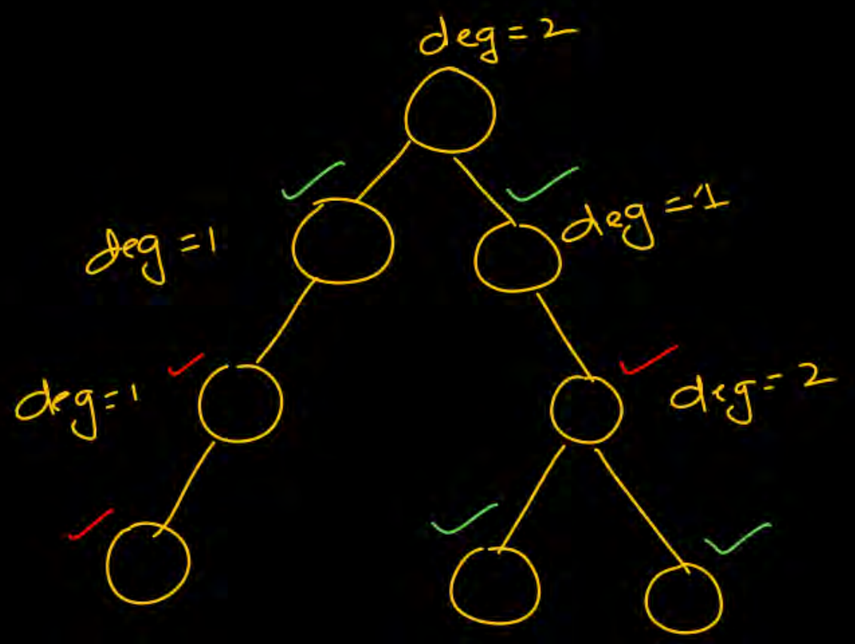
रटना नहीं है

A binary tree with

2 nodes of degree 2,

3 nodes of degree 1

find the no. of leaf node.



$$n = 2 \times 2 + 3 \times 1 + \overset{\text{root}}{1}$$

$$n = 8$$

$$S + L = 8$$

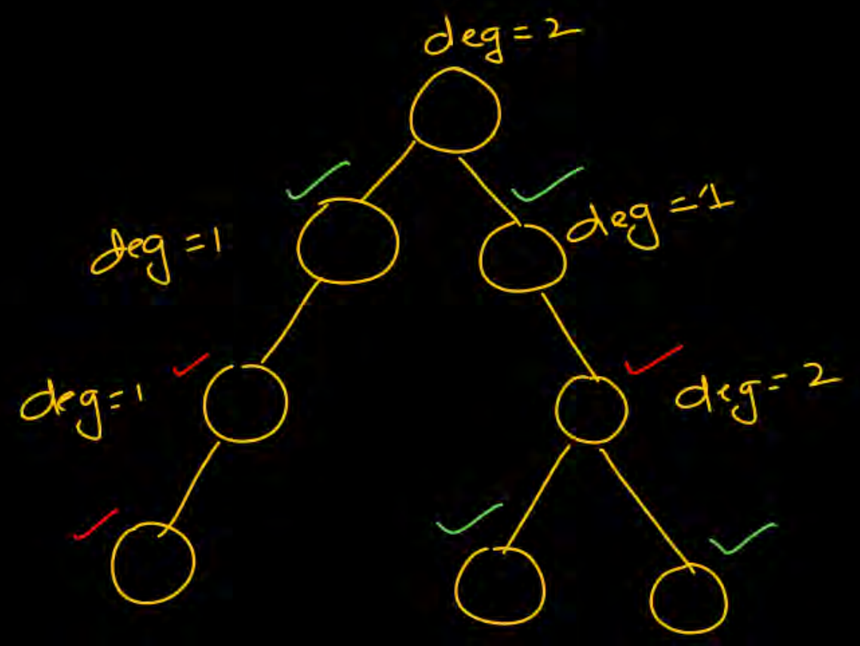
$$\boxed{L = 3}$$

A binary tree with

2 nodes of degree 2,

3 nodes of degree 1

find the no. of leaf node.



2 internal node $\Rightarrow$ degree 2 $\Rightarrow$ 2×2

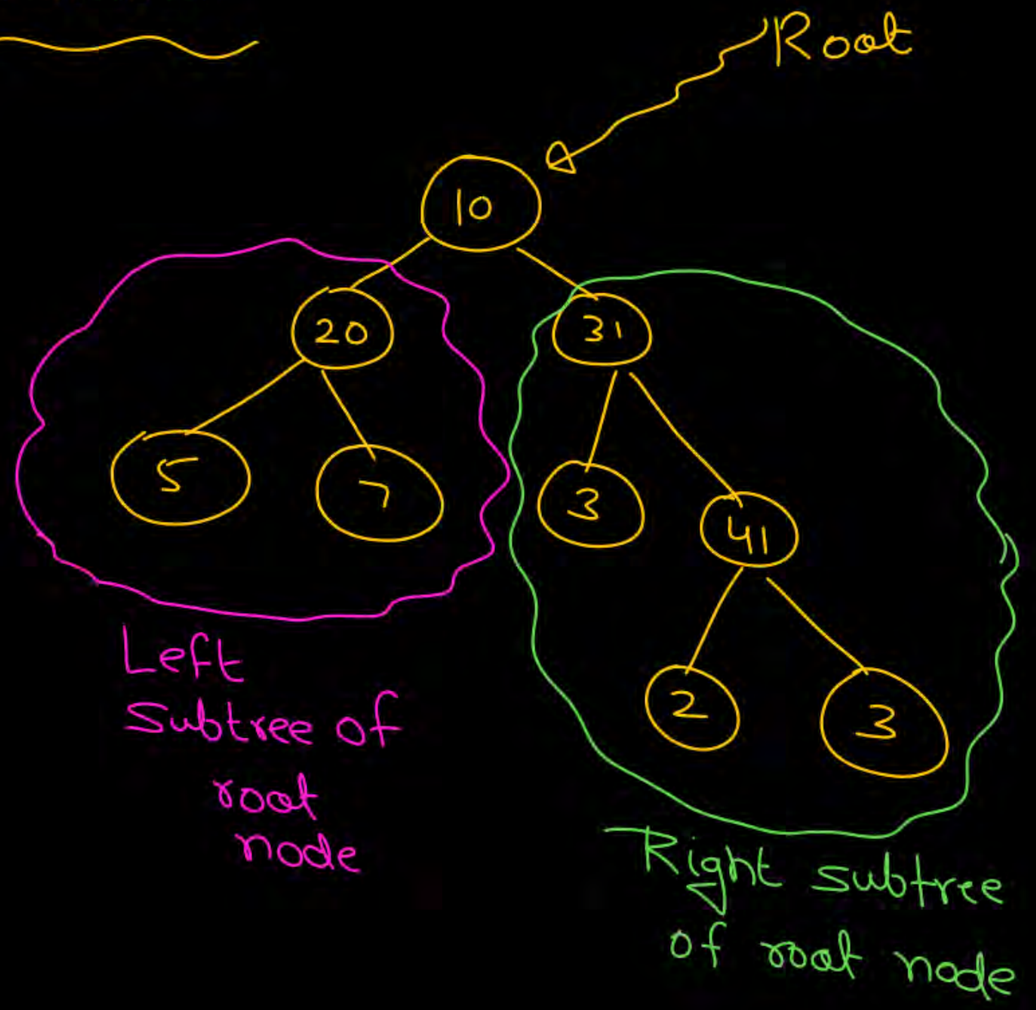
3 internal node $\Rightarrow$ degree 1 $\Rightarrow$ 3×1

$n = 8$

1 (root)

Tree Traversal

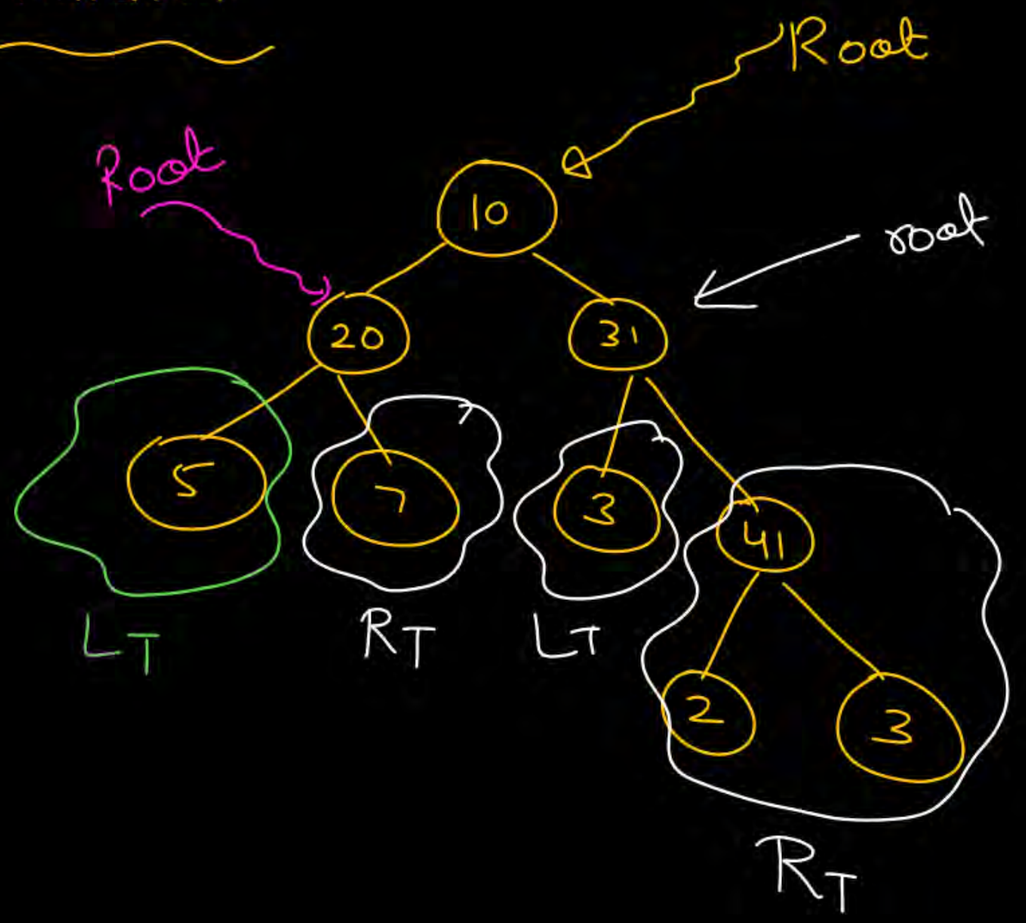
1) Root
2) L_T
3) R_T



Tree Traversal

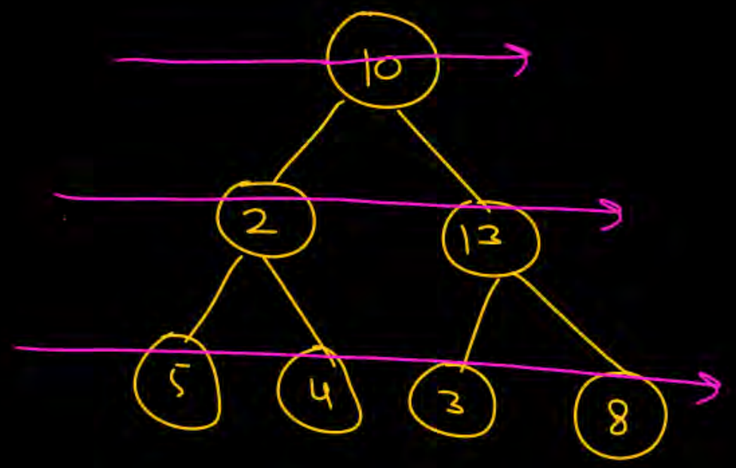
1) Root
2) L_T
3) R_T

Recursion



Tree Traversal

Level order



10, 2, 13, 5, 4, 3, 8

Depth order

Node, L_T, R_T

$3! = 6$ ways

(i) Root, L_T, R_T

(ii) L_T, Root, R_T

(iii) L_T, R_T, Root

(iv) Root, R_T, L_T

(v) R_T, Root, L_T

(vi) R_T, L_T, Root

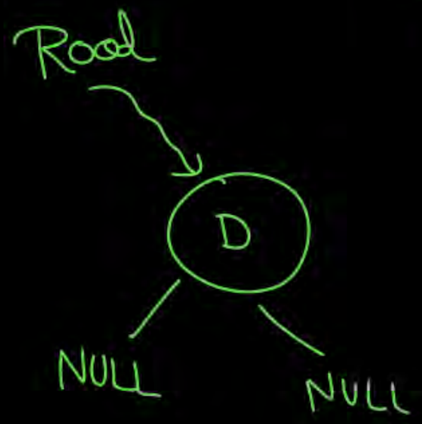
Pre-order Traversal

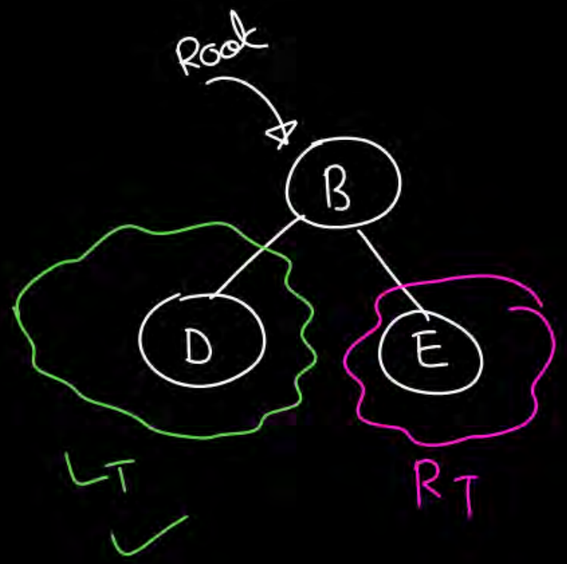
1.) Visit/Print/Process the root node.

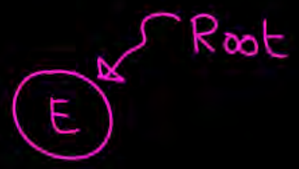
2.) Traverse L_T of root node in Pre-order.

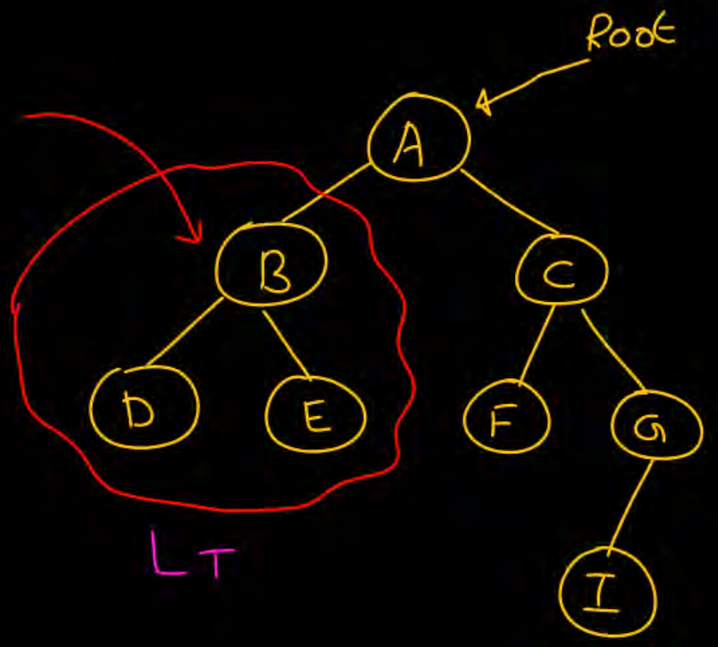
3.) Traverse R_T of root node in Pre-order

A B D E



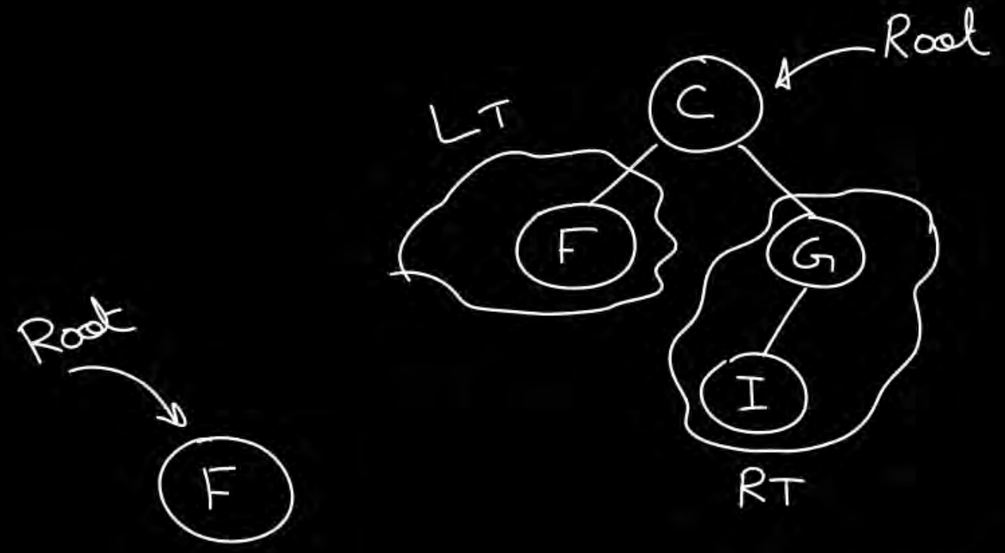


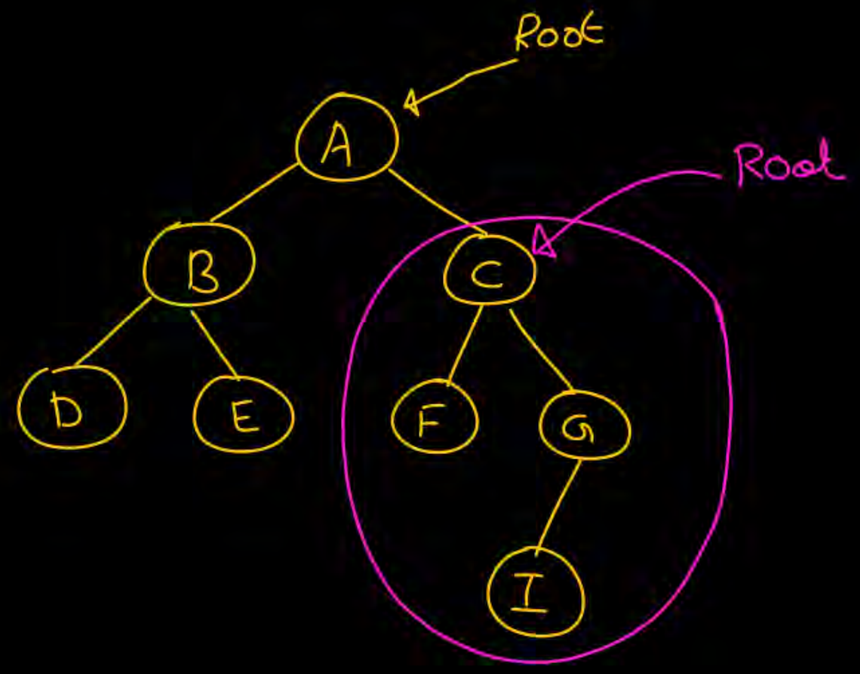


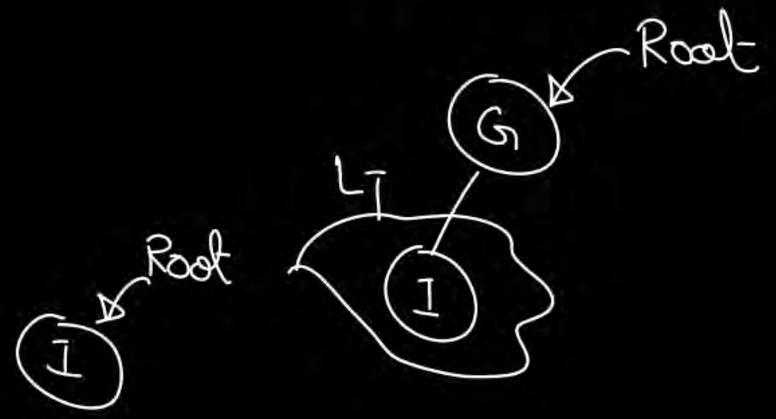


A B D E C F G I

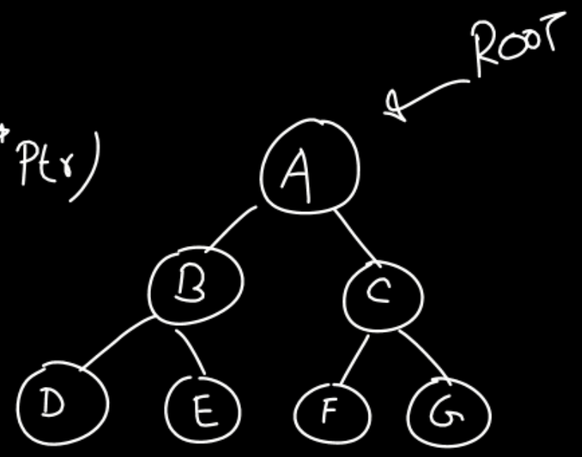
Root LT RT

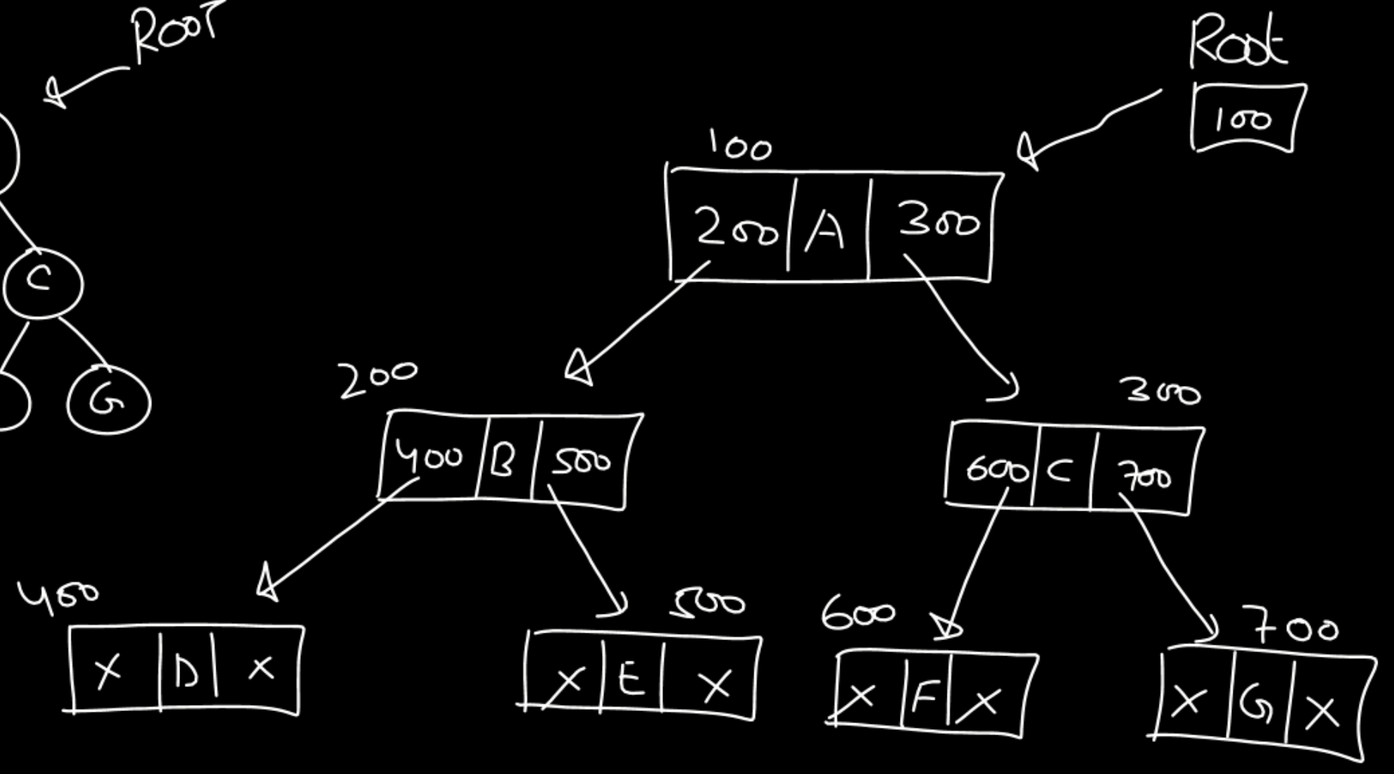






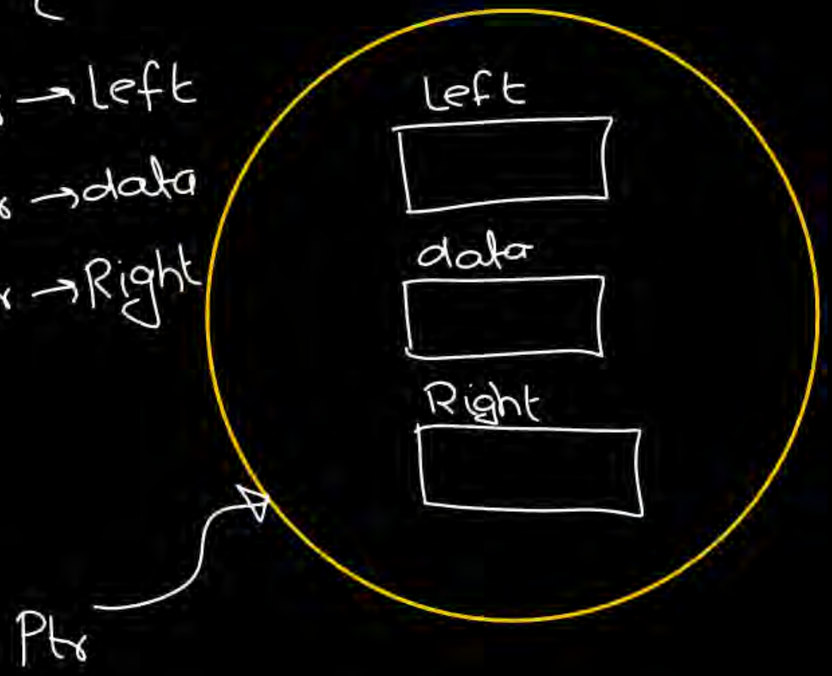
```
void Preorder(struct Node *Ptr)
{

}
```

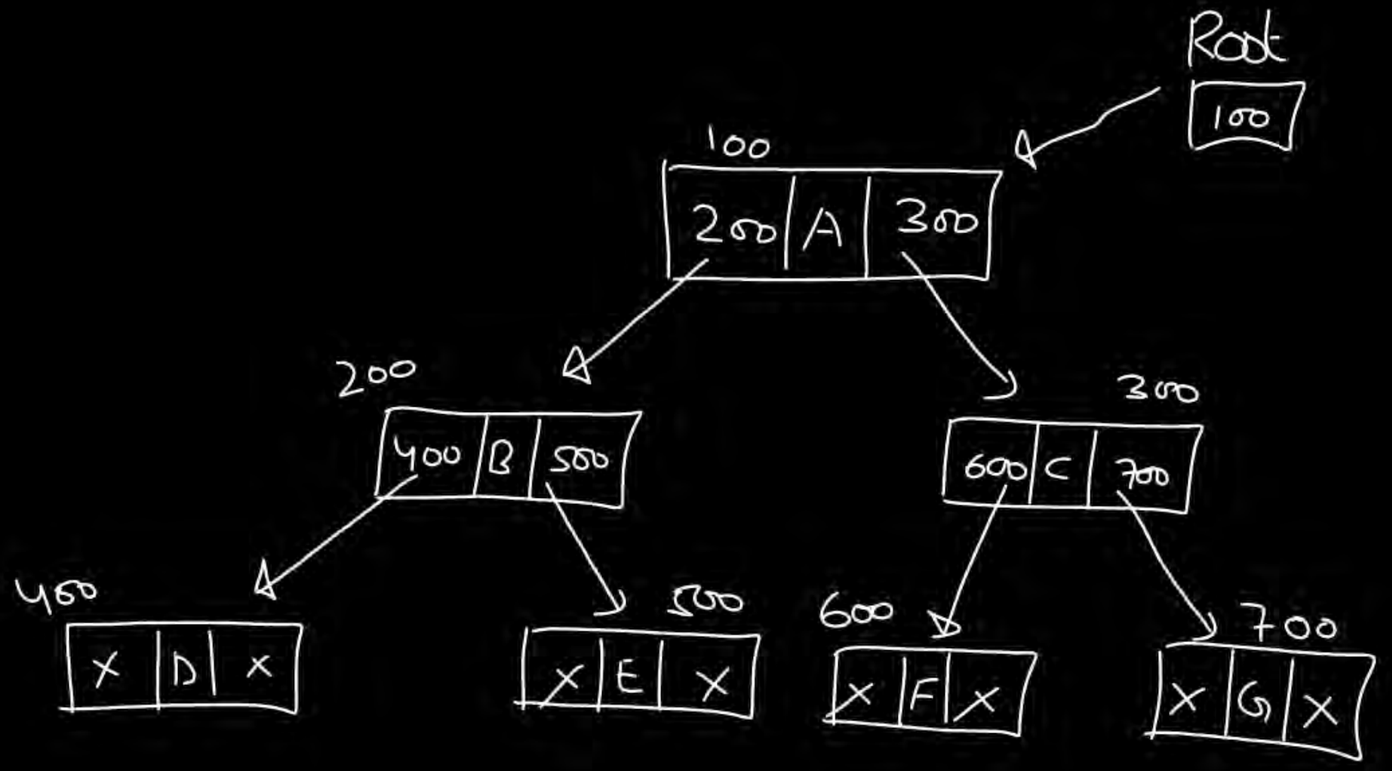





```
void main(){
    ≡
    Preorder( ROOT );
    ≡
}
```

```
void Preorder(struct Node *Ptr)
{
Ptr -> Left
Ptr -> data
Ptr -> Right

}
```






```
void main(){
    ≡
Preorder( ROOT );
    ≡
}
```

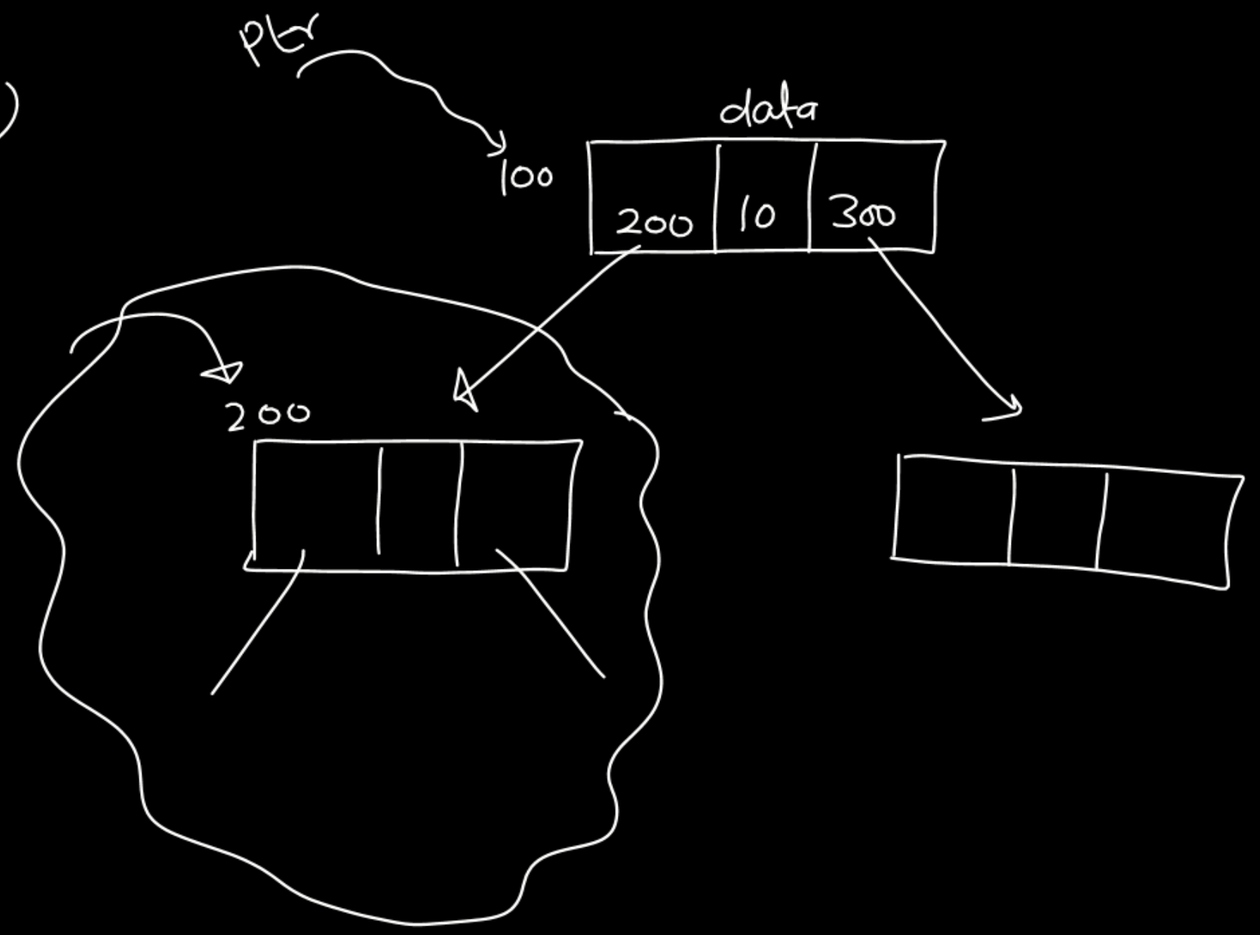
```
void Preorder( struct Node *Ptr)
{
        if (Ptr == NULL)
                return;

}
```

ROOT

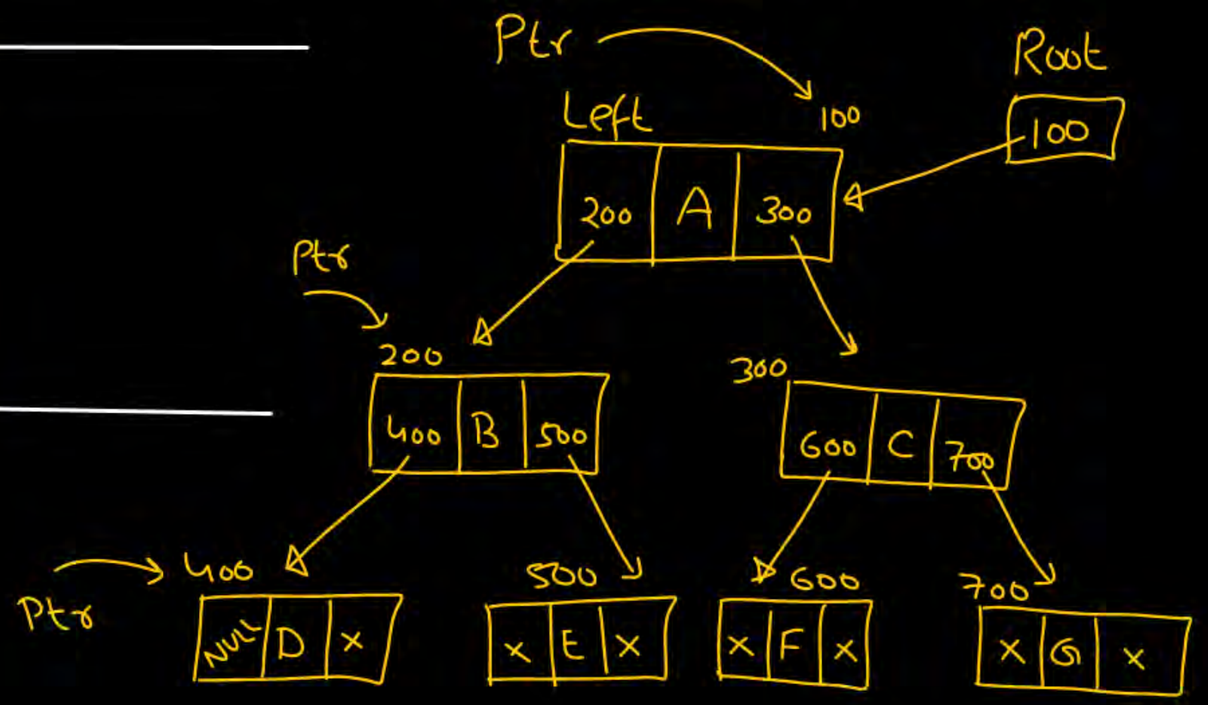
NULL

```
void main(){
=
Preorder(ROOT)
=

}
```

```
void Preorder( struct Node *Ptr)
{
        if (Ptr == NULL)
                return;

    printf("%d", Ptr -> data);

    Preorder( Ptr -> Left);

    Preorder(Ptr -> Right);

}
```



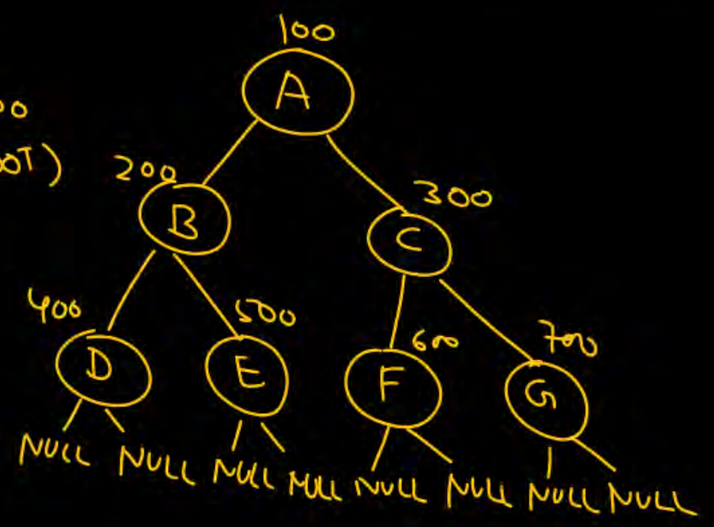
main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 4	Ptr=200 Preorder(200) 1 ✓ 2 ✓ 3 4	Ptr=400 Preorder(400) 1 ✓ 2 ✓ 3 4	Ptr=NULL Preorder(NULL)



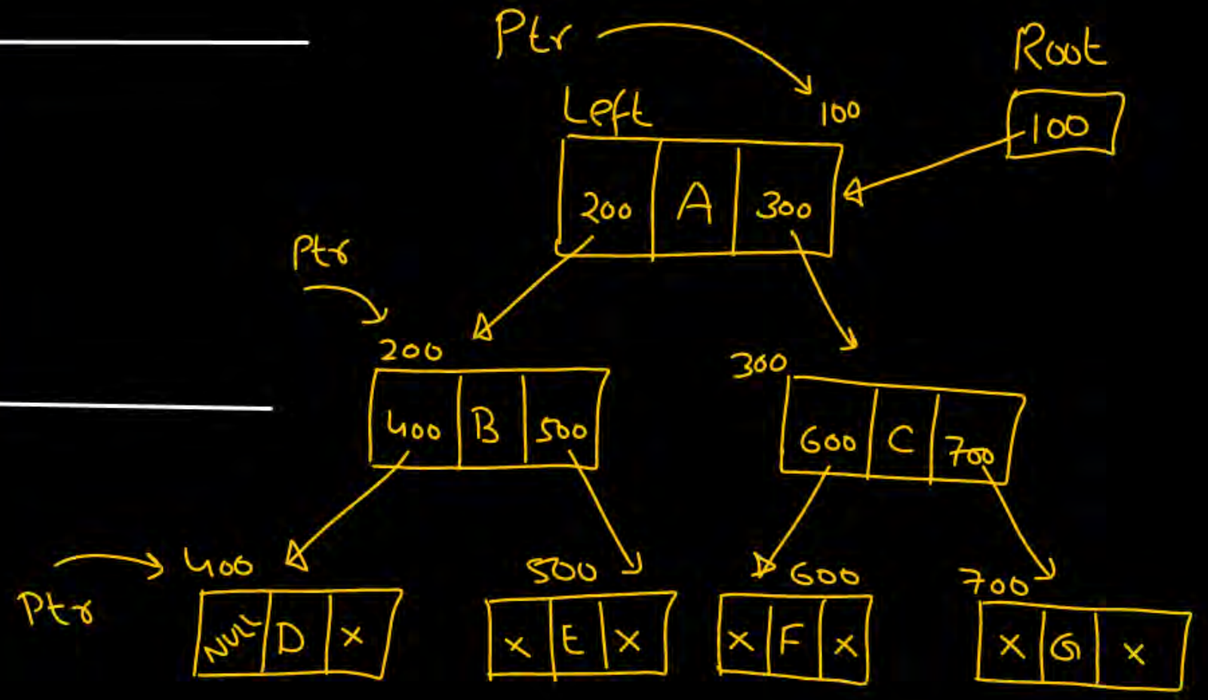

```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
    {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->left);
3.      Preorder(Ptr->Right);
4.  }
    }
```

ABD

```
main()
{
    Preorder(ROOT)
    =
}
```



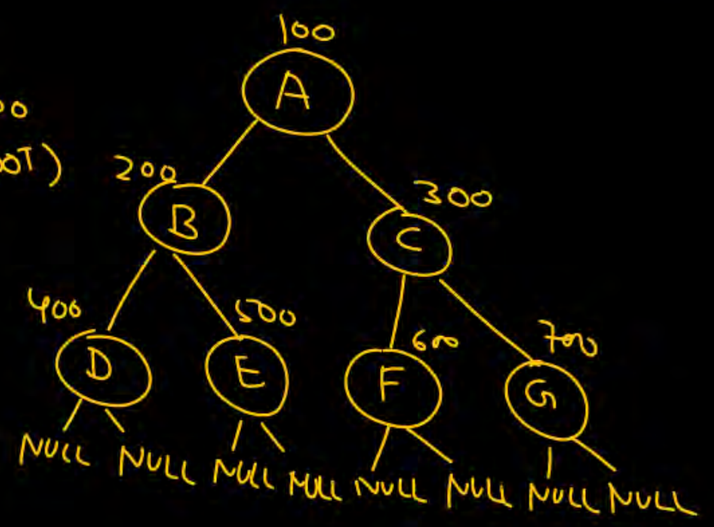
main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 4	Ptr=200 Preorder(200) 1 ✓ 2 ✓ 3 4	Ptr=400 Preorder(400) 1 ✓ 2 ✓ 3 4	Ptr=NULL Preorder(NULL)



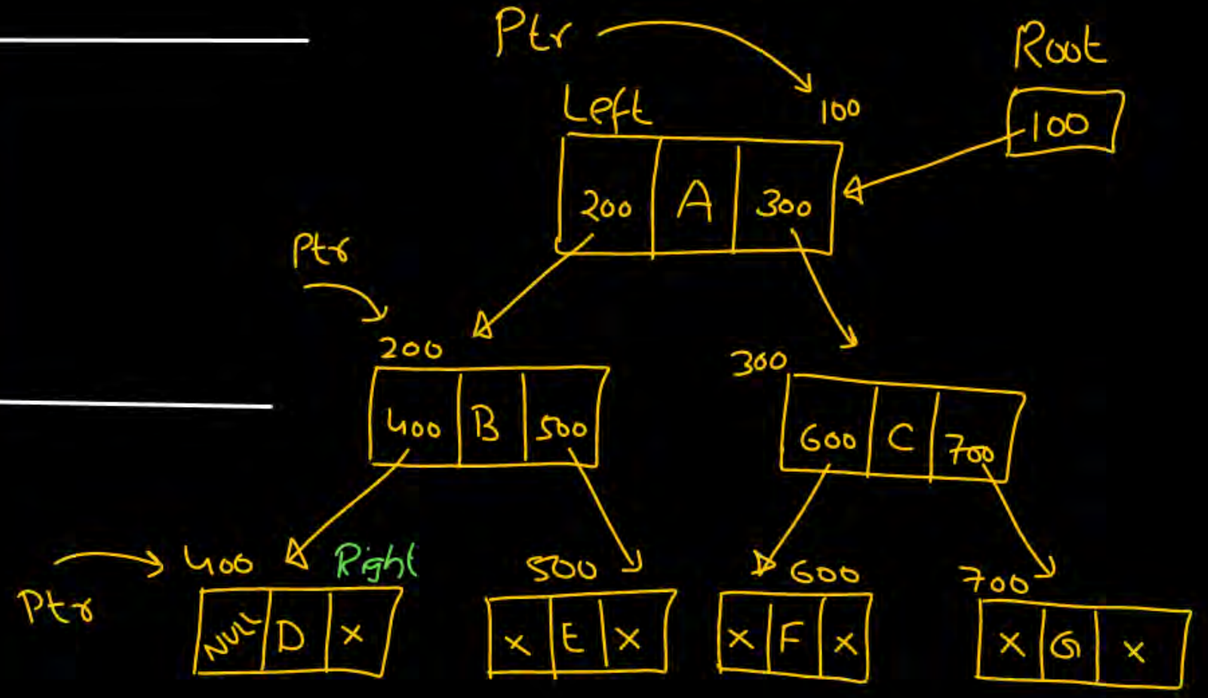

```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
        {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->left);
3.      Preorder(Ptr->Right);
4.      }
        }
```

ABD

```
main()
{
    Preorder(ROOT)
    }
```



main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 4	Ptr=200 Preorder(200) 1 ✓ 2 ✓ 3 4	Ptr=400 Preorder(400) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=NULL Preorder(NULL)

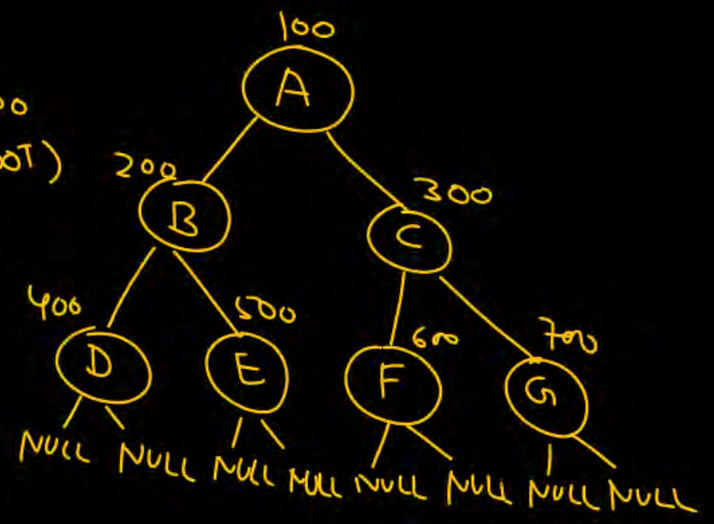


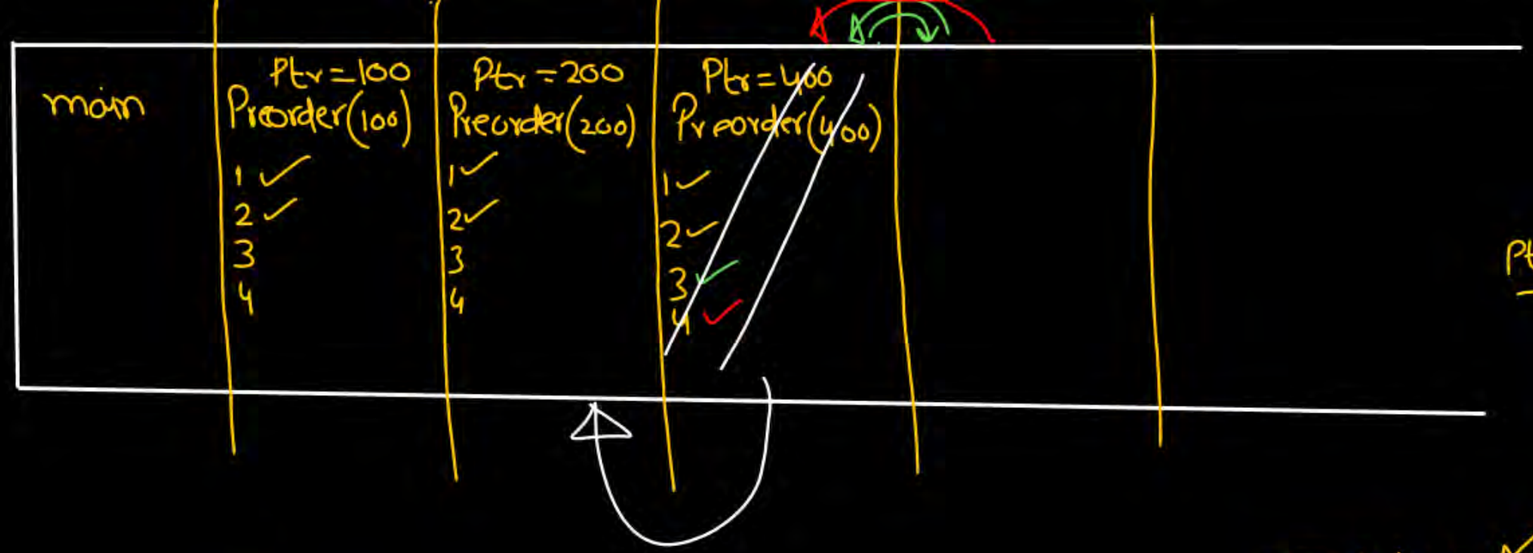

```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
        {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->Left);
3.      Preorder(Ptr->Right);
4.      }
        }
```

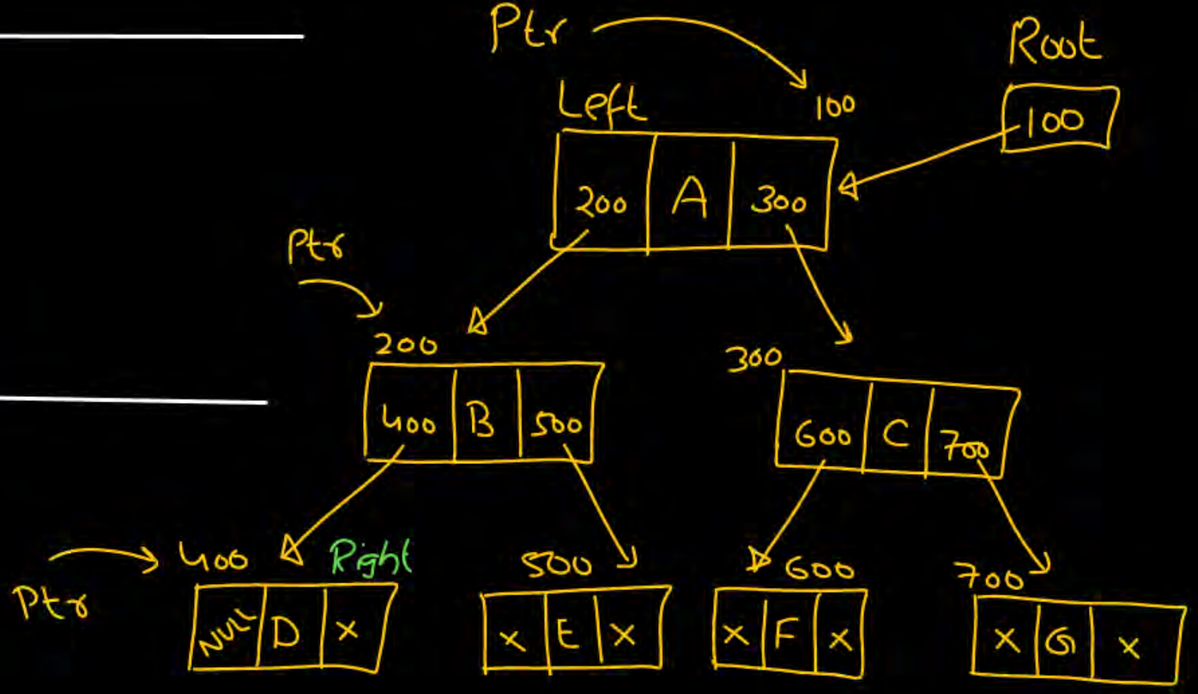
ABD

```
main()
{
    Preorder(ROOT)
    }
```

100





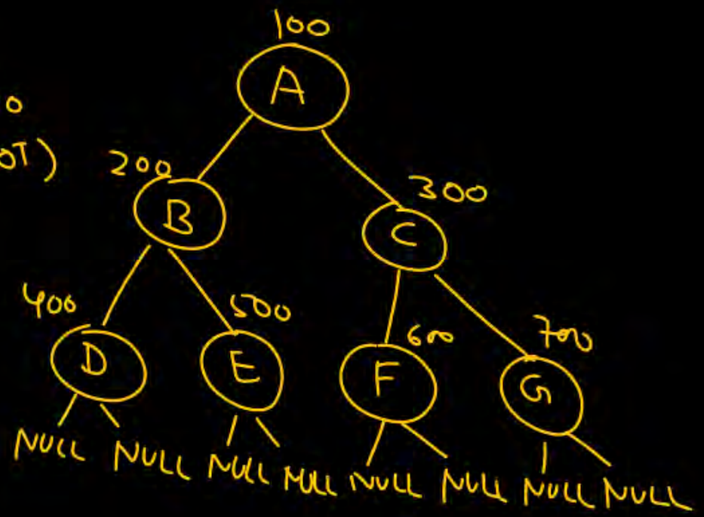



```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
        {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->Left);
3.      Preorder(Ptr->Right);
4.      }
        }
```

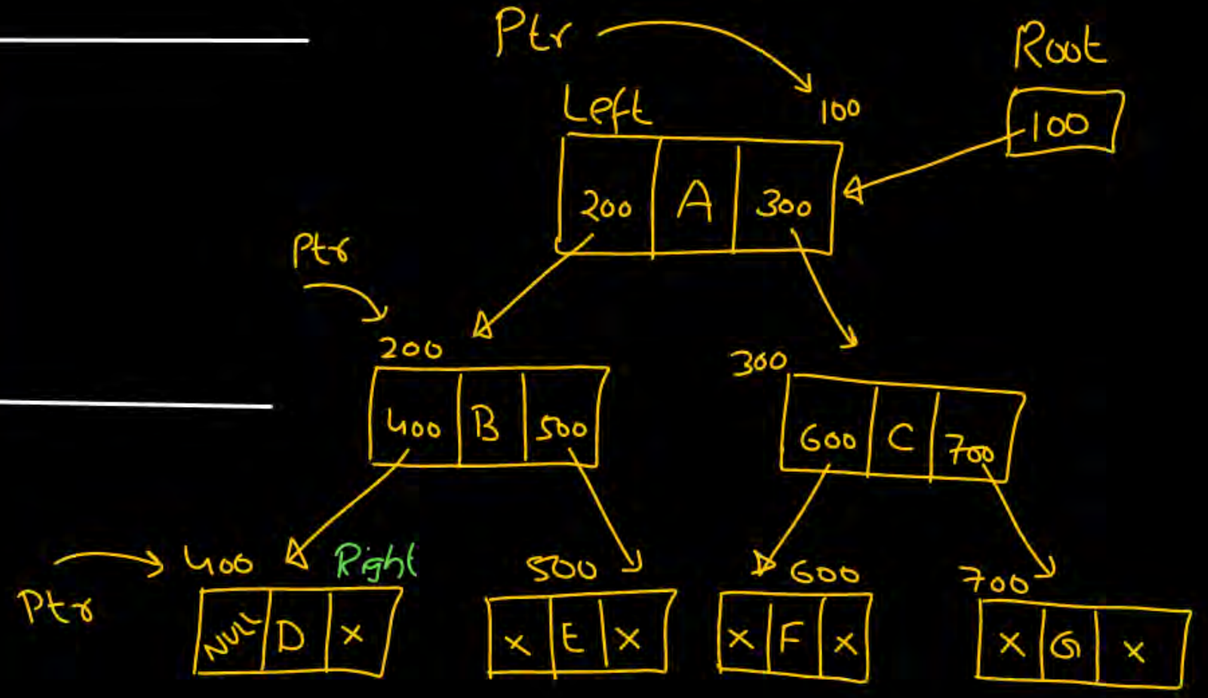
ABD

```
main()
{
    Preorder(ROOT)
    }
```

100



main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 4	Ptr=200 Preorder(200) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=500 Preorder(500) 1 ✓ 2 ✓ 3 4	Ptr=NULL Preorder(NULL)

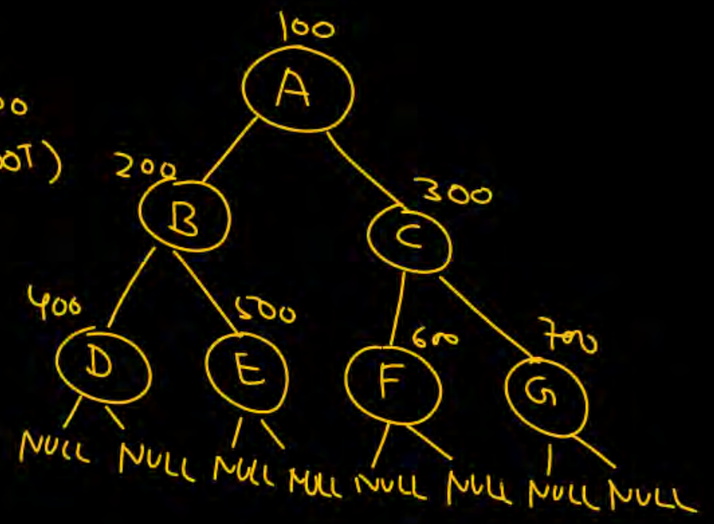



```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
        {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->Left);
3.      Preorder(Ptr->Right);
4.      }
    }
```

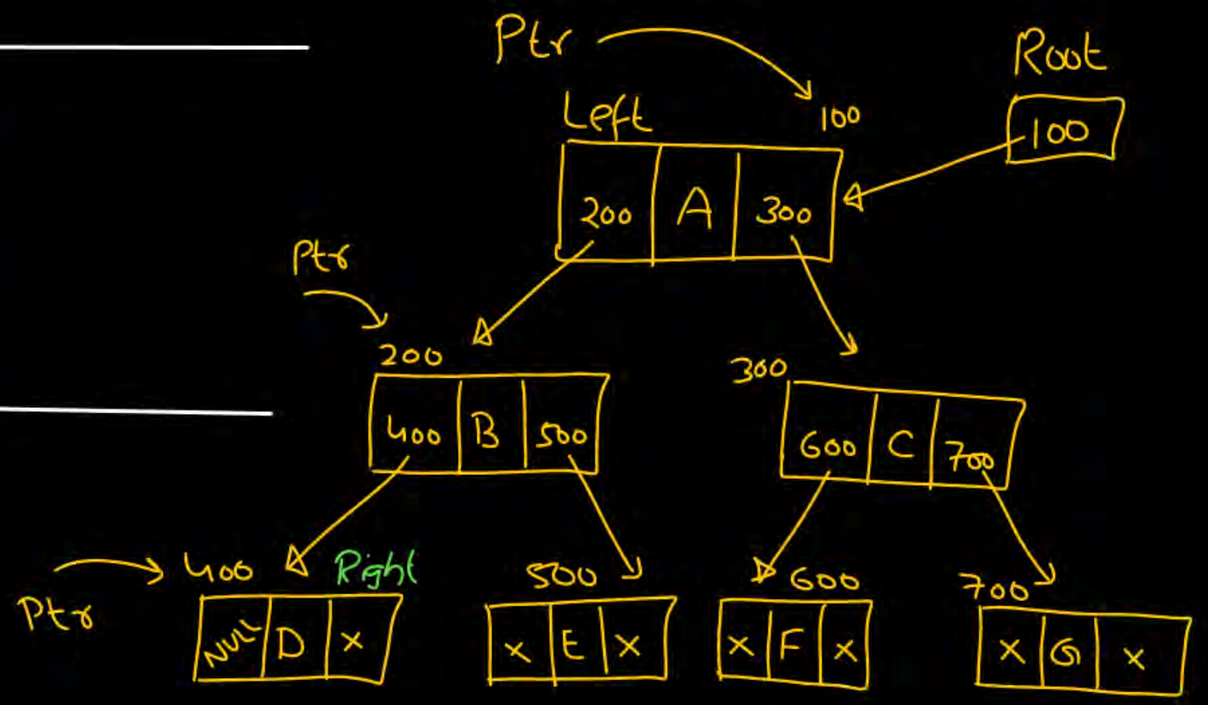
ABDE

```
main()
{
Preorder(ROOT)
}
```

100



main	Ptr=100 Preorder(100)	Ptr=200 Preorder(200)	Ptr=500 Preorder(500)	Ptr=NULL Preorder(NULL)
	1 ✓	1 ✓	1 ✓	
	2 ✓	2 ✓	2 ✓	
	3	3 ✓	3	
	4	4	4	

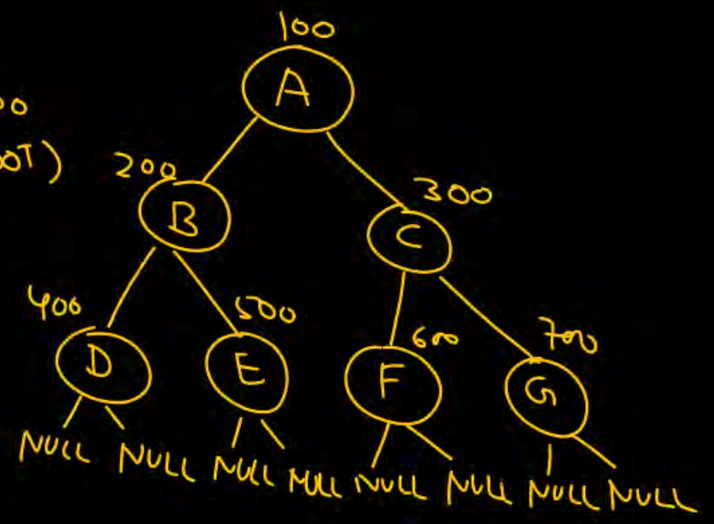



```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
        {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->left);
3.      Preorder(Ptr->Right);
4.      }
    }
```

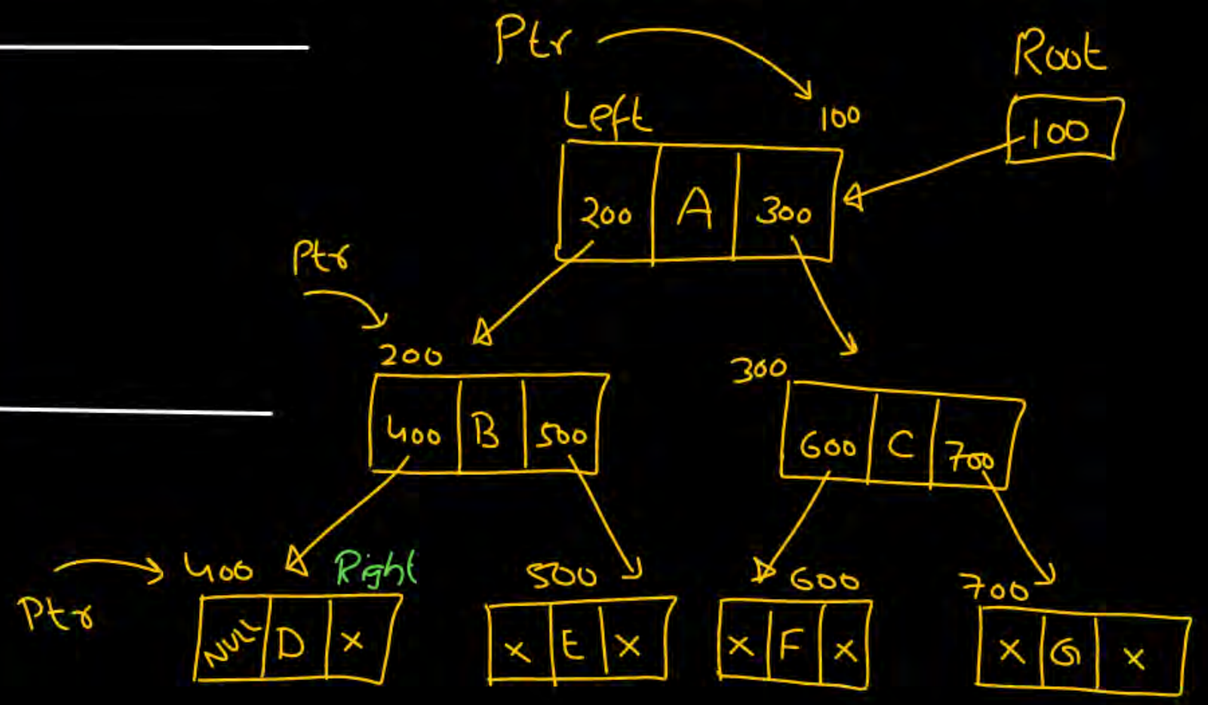
ABDE

```
main()
{
    Preorder(ROOT)
    }
```

100



main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 4	Ptr=200 Preorder(200) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=500 Preorder(500) 1 ✓ 2 ✓ 3 ✓ 4	~~Ptr≠NULL~~ ~~Preorder(NULL)~~

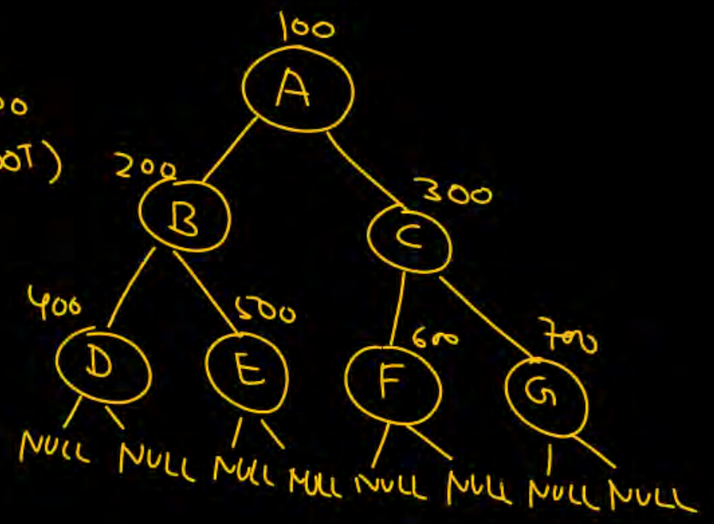



```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
        {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->Left);
3.      Preorder(Ptr->Right);
4.      }
    }
```

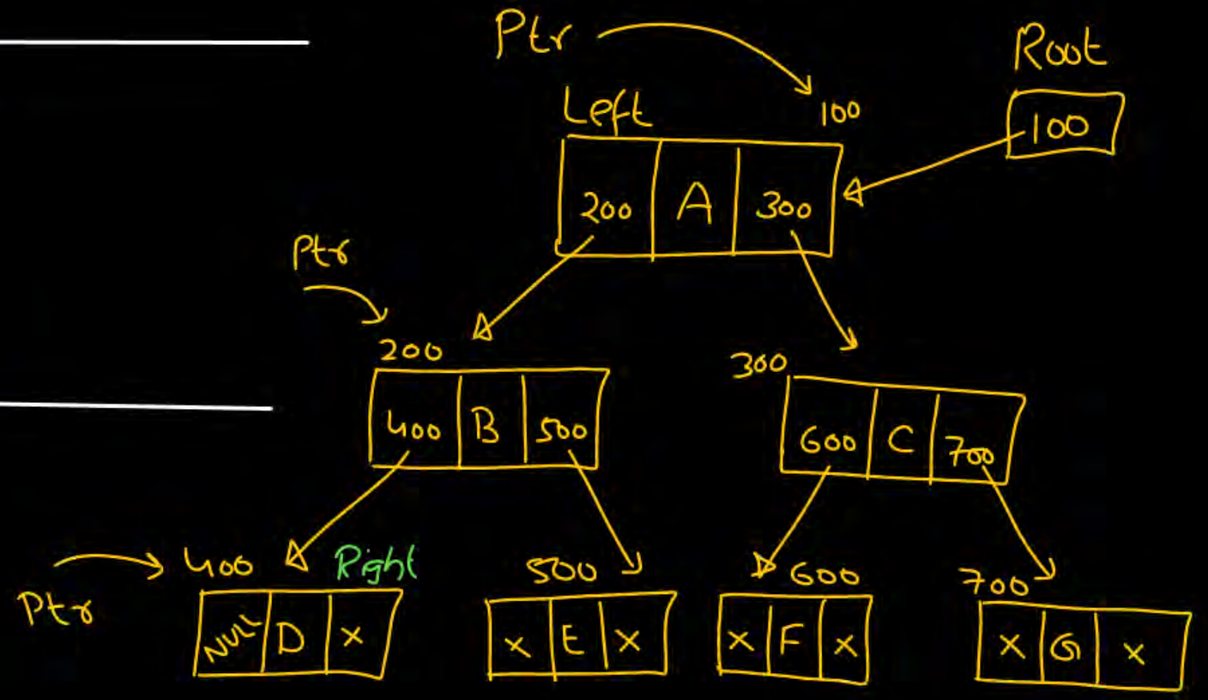
ABDE

```
main()
{
    Preorder(ROOT)
    =
}
```

100



main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 4	Ptr=200 Preorder(200) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=500 Preorder(500) 1 ✓ 2 ✓ 3 ✓ 4 ✓		

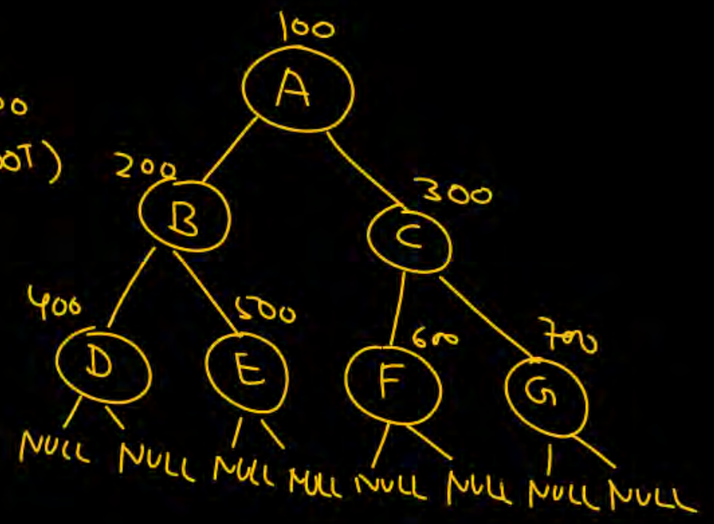



```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
        {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->left);
3.      Preorder(Ptr->Right);
4.      }
    }
```

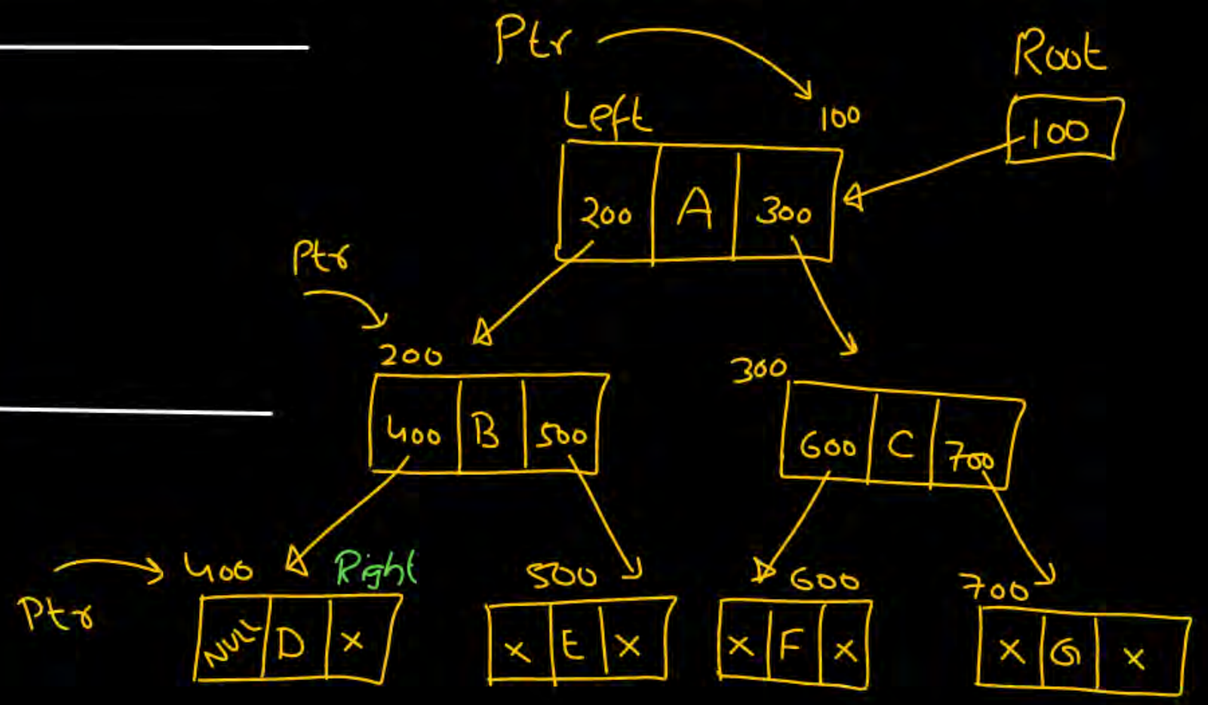
ABDE

```
main()
{
    Preorder(ROOT)
    =
}
```

100



main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 4	Ptr=200 Preorder(200) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=500 Preorder(500) 1 ✓ 2 ✓ 3 ✓ 4 ✓		

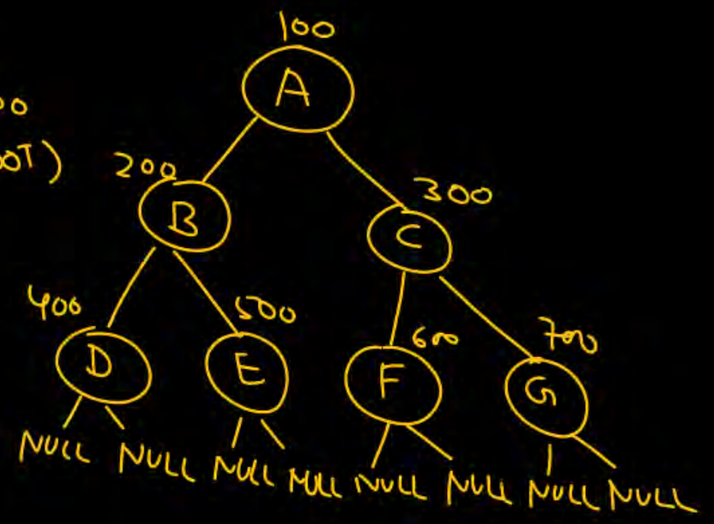



```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
        {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->left);
3.      Preorder(Ptr->Right);
4.      }
    }
```

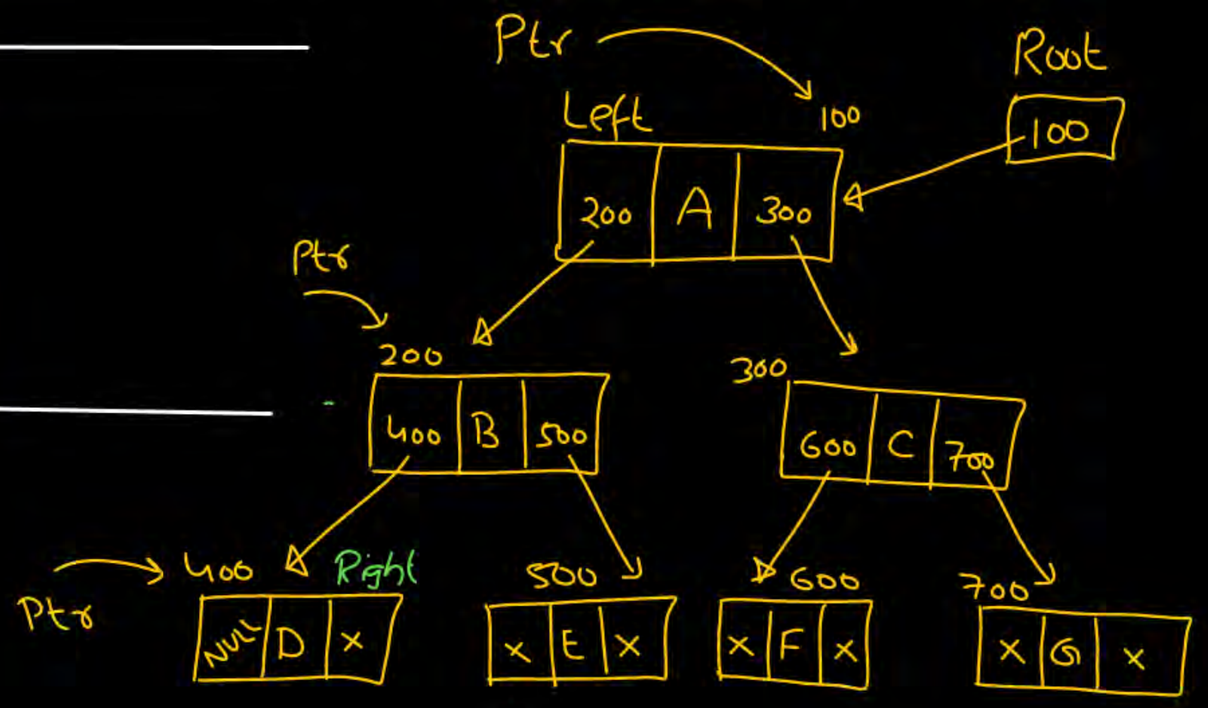
ABDE

```
main()
{
    Preorder(ROOT)
    }
```

100



main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 4	Ptr=200 Preorder(200) 1 ✓ 2 ✓ 3 ✓ 4 ✓			

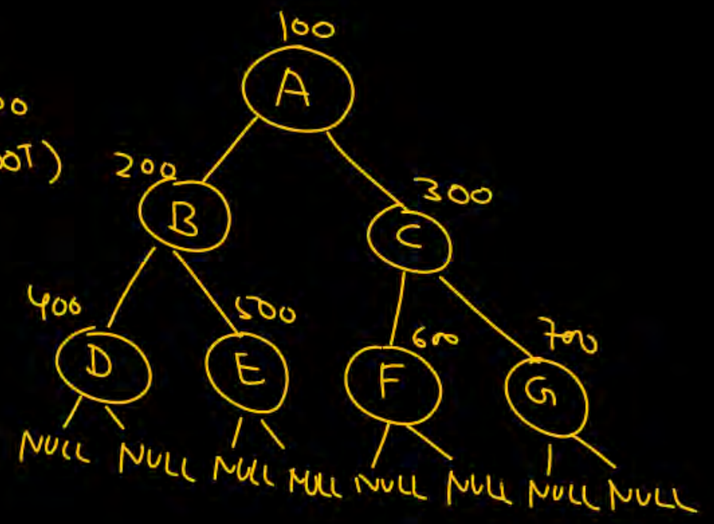



```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
        {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->left);
3.      Preorder(Ptr->Right);
4.      }
    }
```

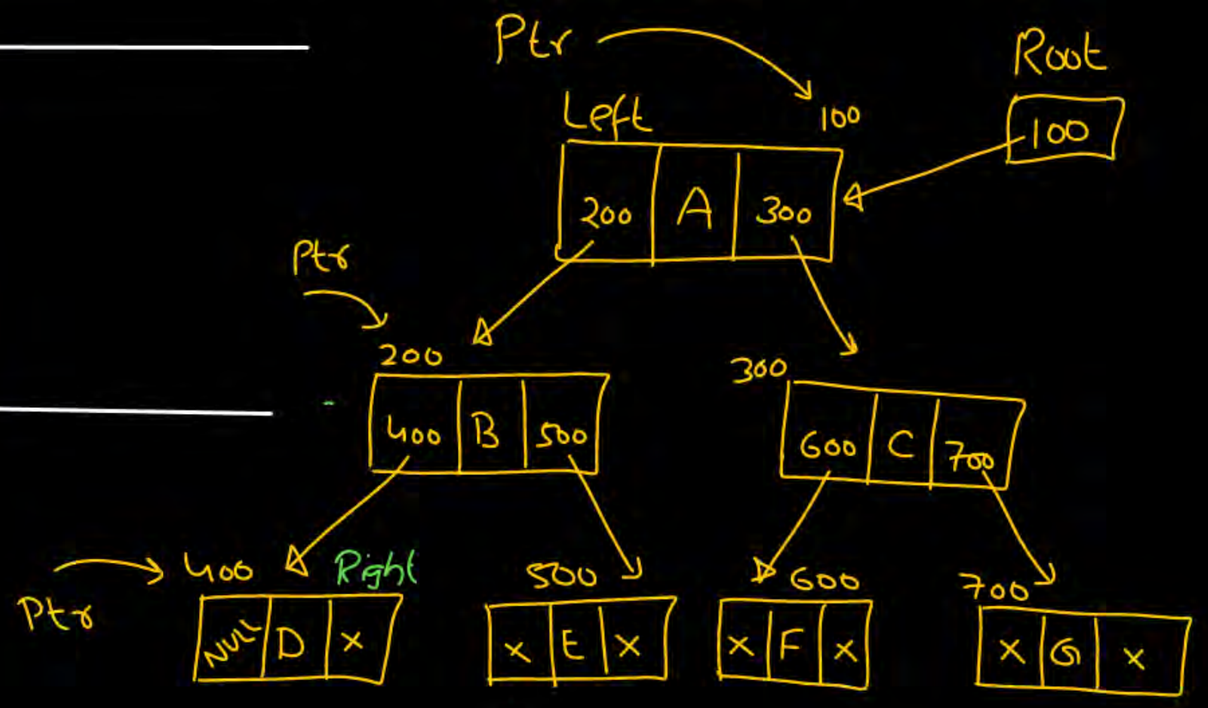
ABDE

```
main()
{
    Preorder(ROOT)
    }
```

100



main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 4	Ptr=200 Preorder(200) 1 ✓ 2 ✓ 3 ✓ 4 ✓			

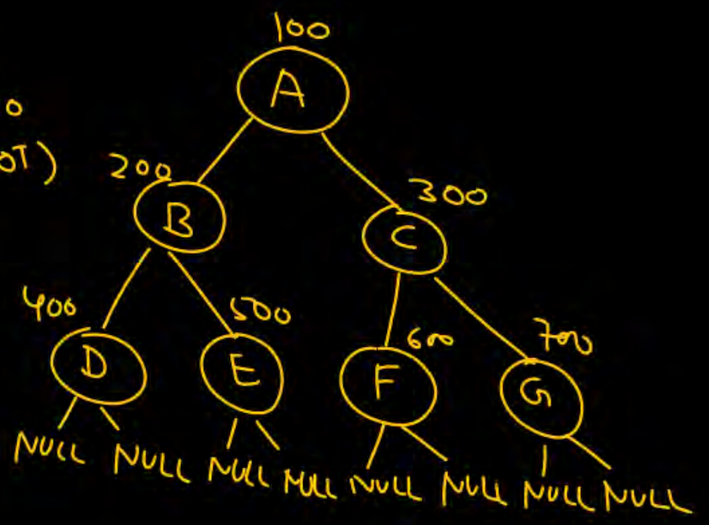


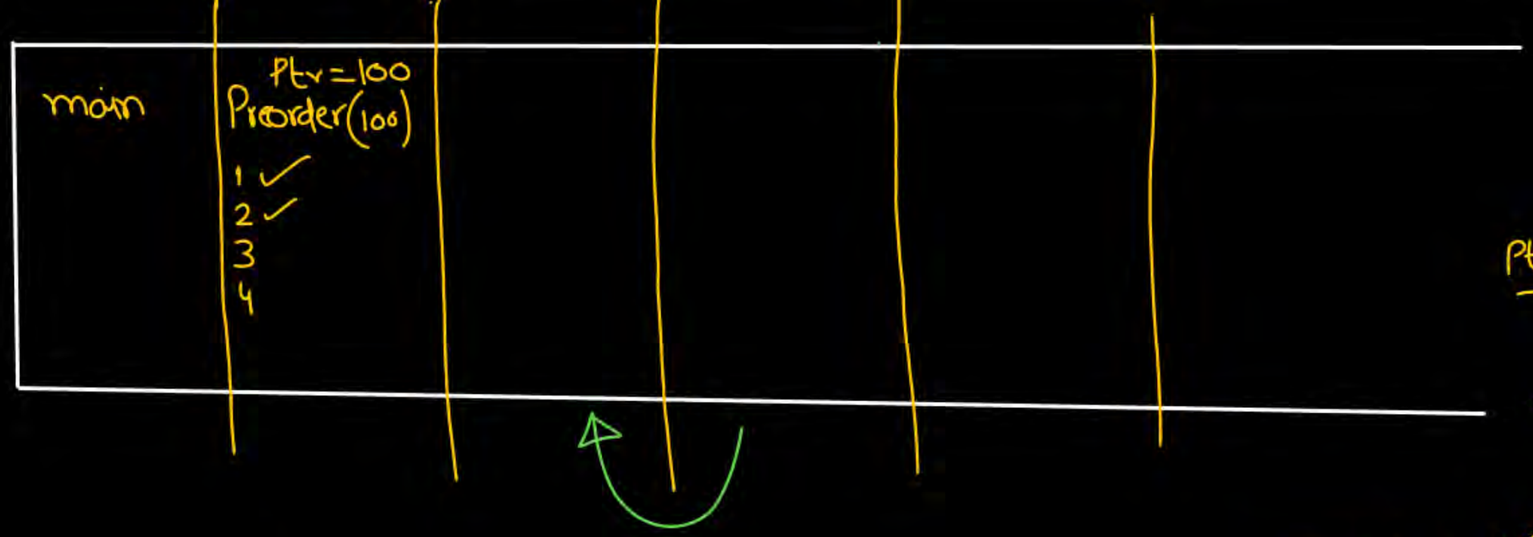

```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
        {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->left);
3.      Preorder(Ptr->Right);
4.      }
    }
```

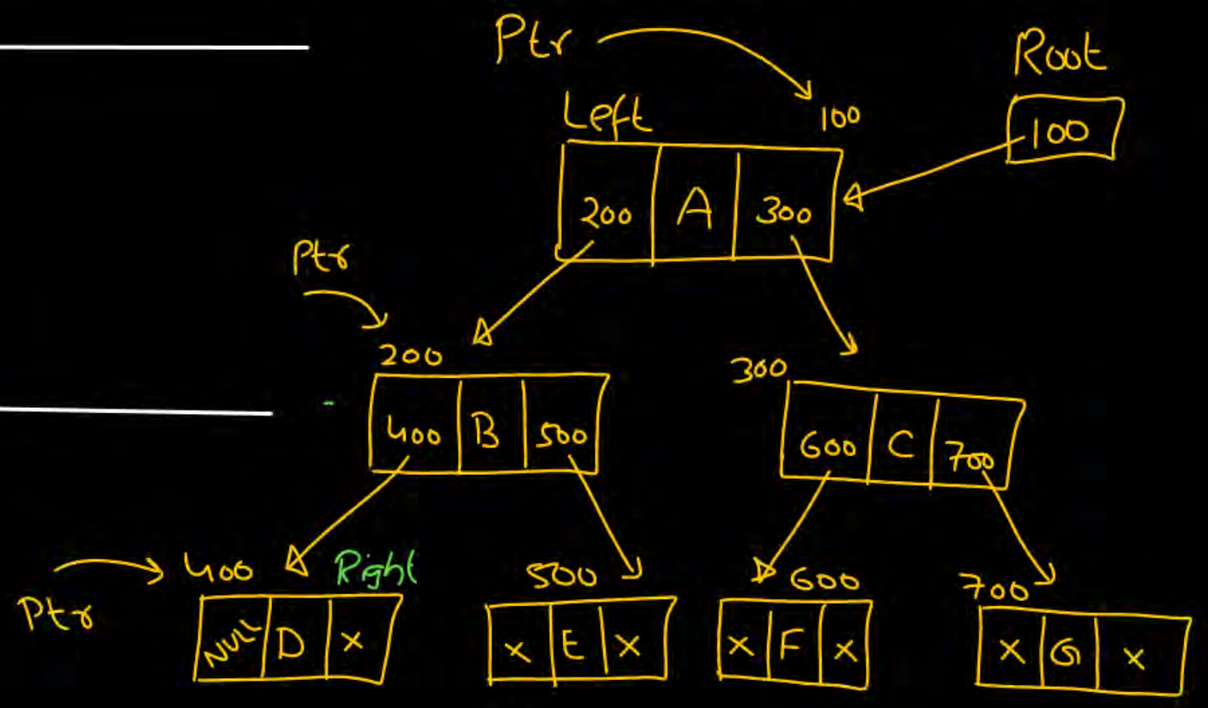
ABDE

```
main()
{
    Preorder(ROOT)
    }
```

100





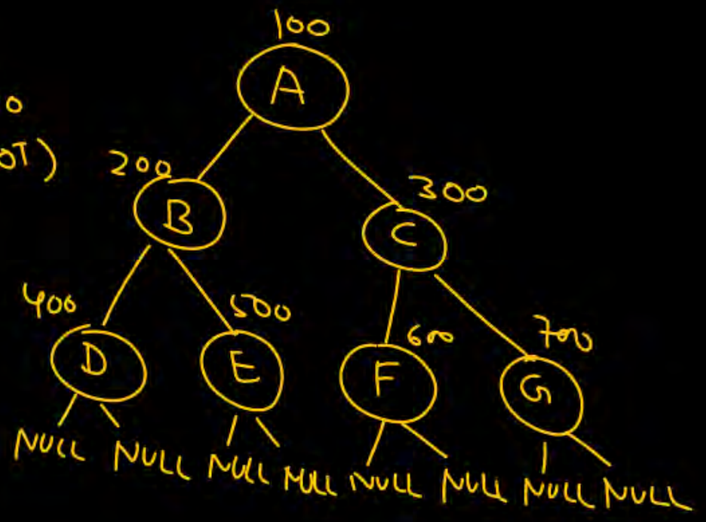



```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
        {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->left);
3.      Preorder(Ptr->Right);
4.      }
    }
```

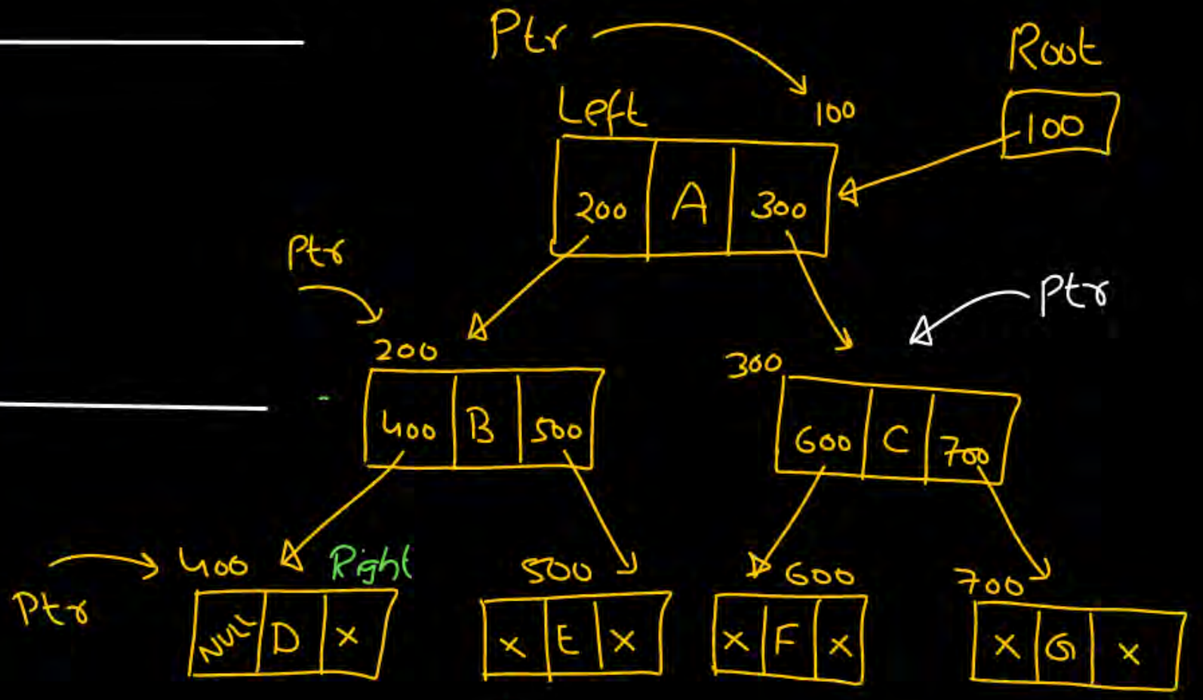
ABDE

```
main()
{
    Preorder(ROOT)
    }
```

100



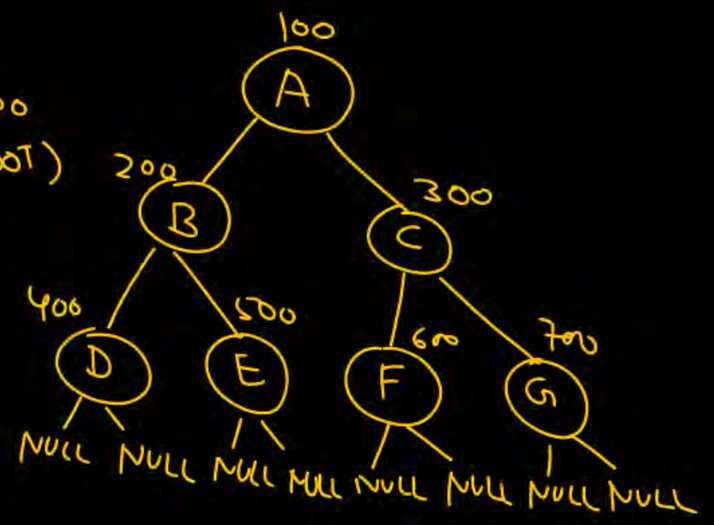
main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=300 Preorder(300) 1 ✓ 2 ✓ 3 4	Ptr=600 Preorder(600) 1 ✓ 2 ✓ 3 ✓ 4 ✓	



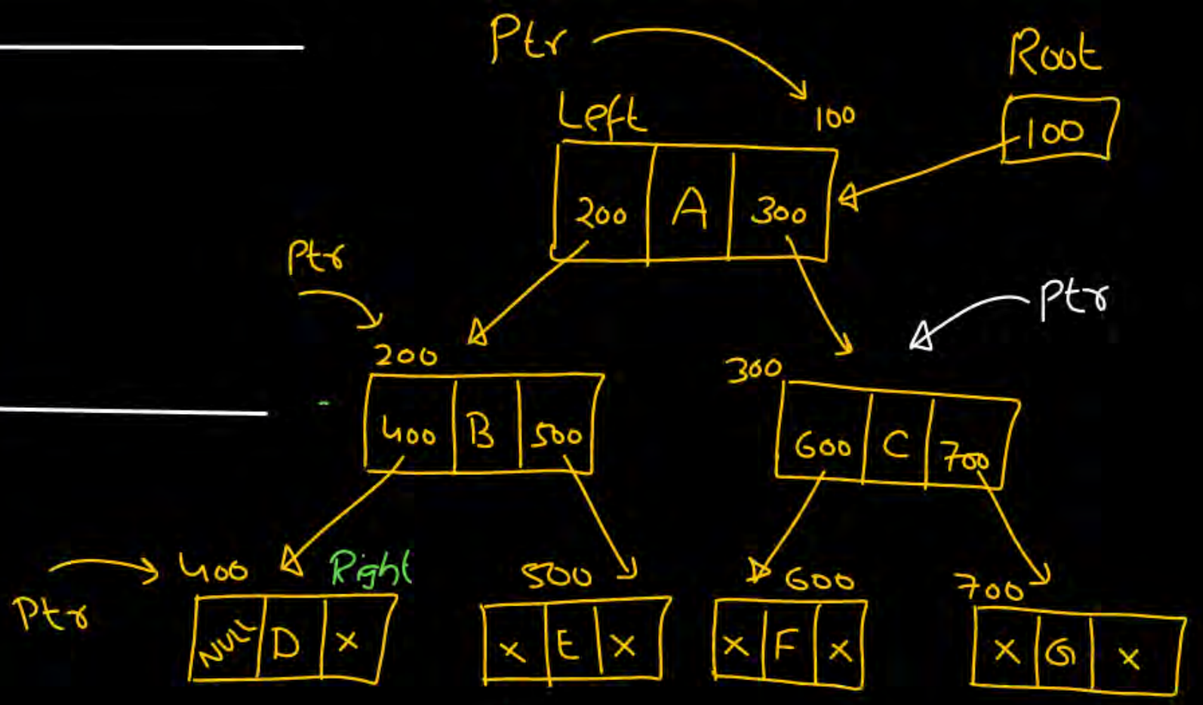

```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
    {
1.  pf("%d", Ptr->data);
2.  Preorder(Ptr->Left);
3.  Preorder(Ptr->Right);
4.  }
}
```

ABDECF

```
main()
{
    Preorder(ROOT)
    =
}
```



main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=300 Preorder(300) 1 ✓ 2 ✓ 3 4	Ptr=600 Preorder(600) 1 ✓ 2 ✓ 3 ✓ 4 ✓		

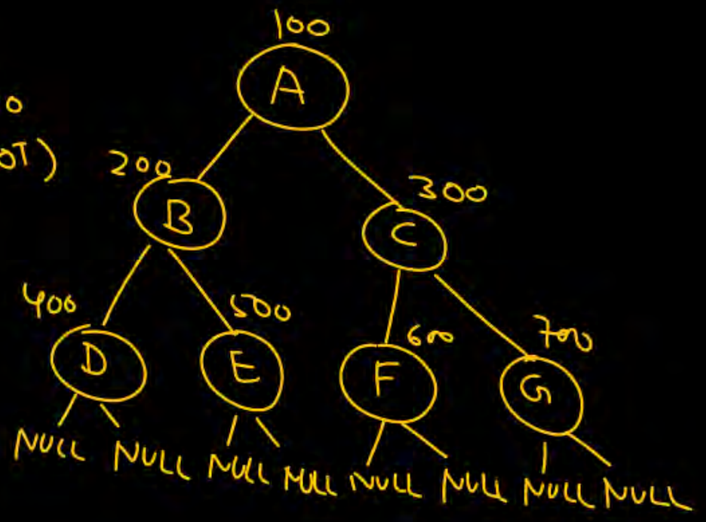



```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
    {
1.  pf("%d", Ptr->data);
2.  Preorder(Ptr->Left);
3.  Preorder(Ptr->Right);
4.  }
}
```

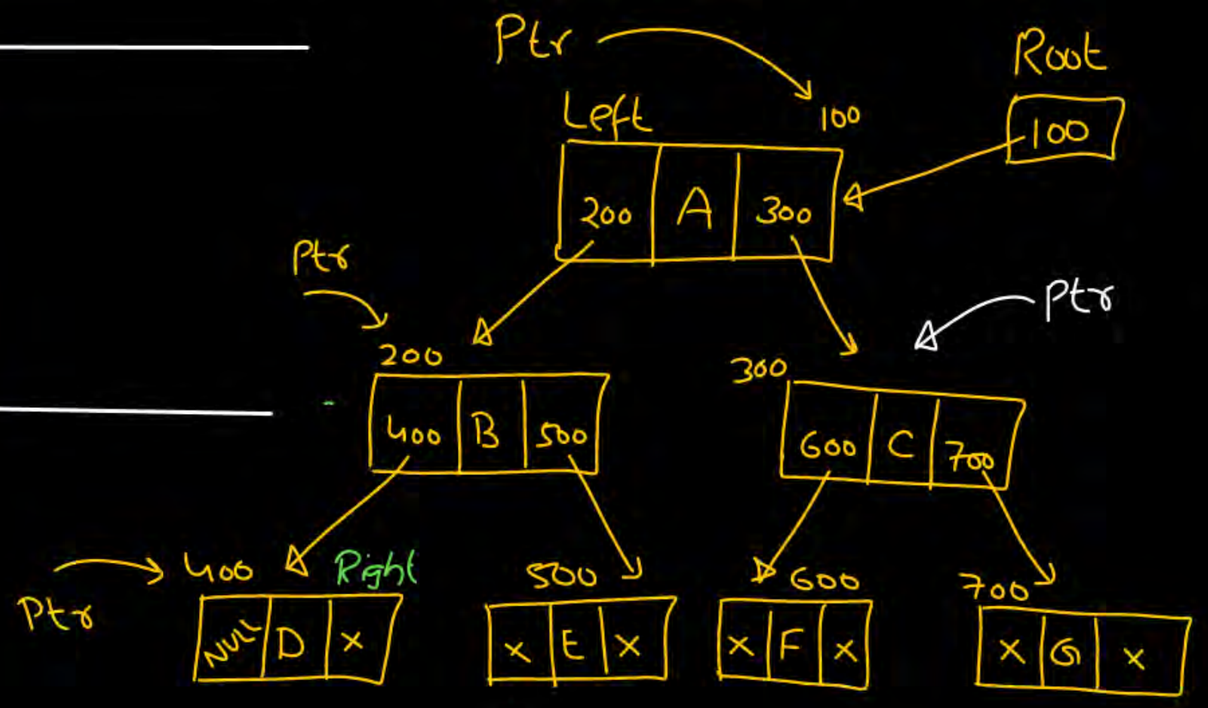
ABDECF

```
main()
{
    Preorder(ROOT)
    }
```

100



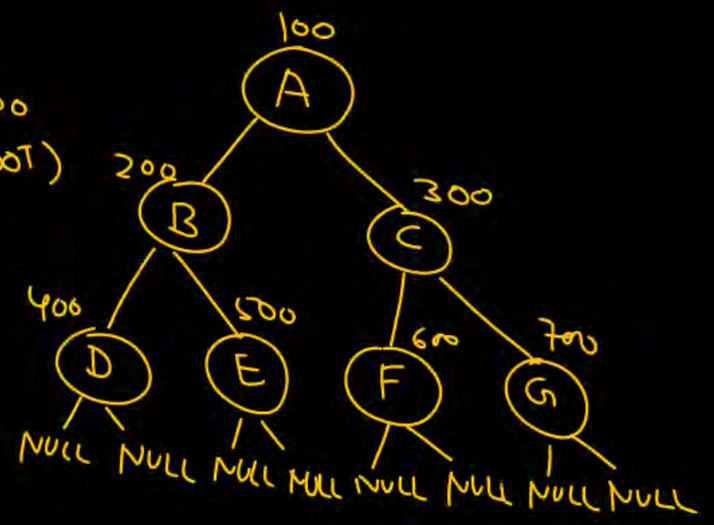
main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=300 Preorder(300) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=700 Preorder(700) 1 ✓ 2 ✓ 3 ✓ 4 ✓		



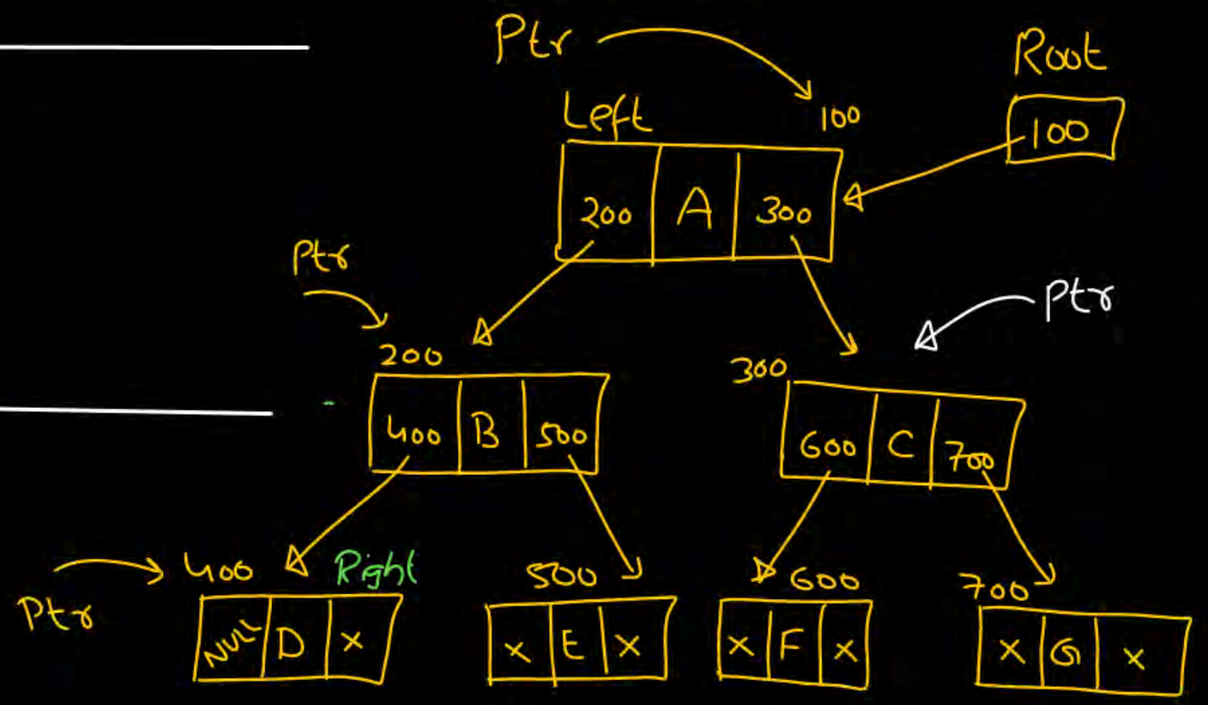

```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
    {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->left);
3.      Preorder(Ptr->Right);
4.  }
}
```

ABDECFG

```
main()
{
    Preorder(ROOT)
    =
}
```



main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=300 Preorder(300) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=700 Preorder(700) 1 ✓ 2 ✓ 3 ✓ 4 ✓	

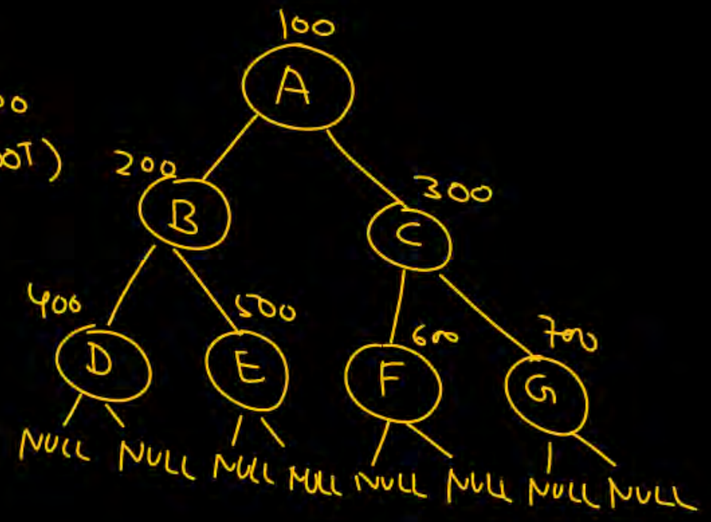



```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
    {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->left);
3.      Preorder(Ptr->Right);
4.  }
}
```

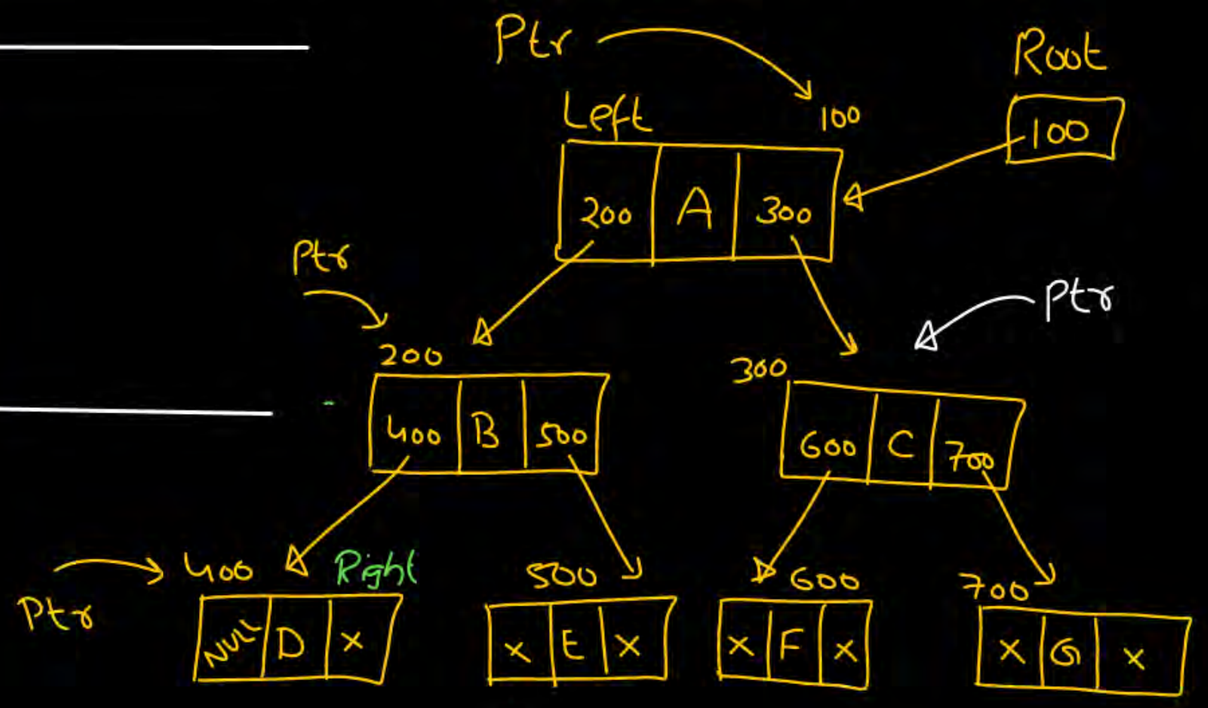
ABDECFG

```
main()
{
    Preorder(ROOT)
    =
}
```

100



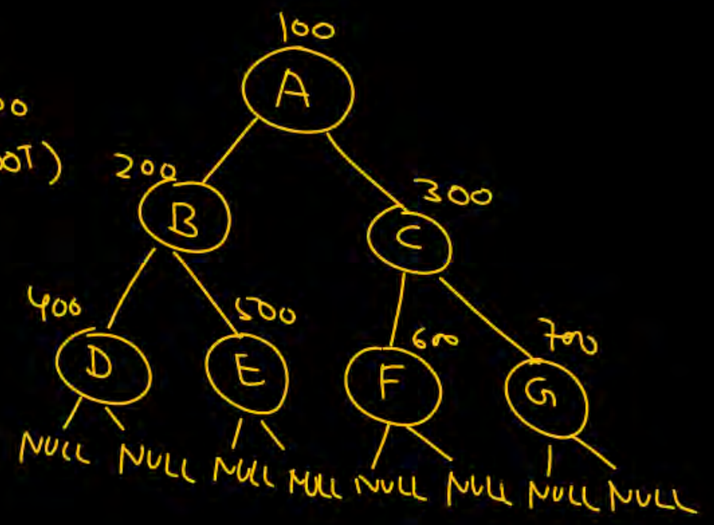
main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=300 Preorder(300) 1 ✓ 2 ✓ 3 ✓ 4 ✓			



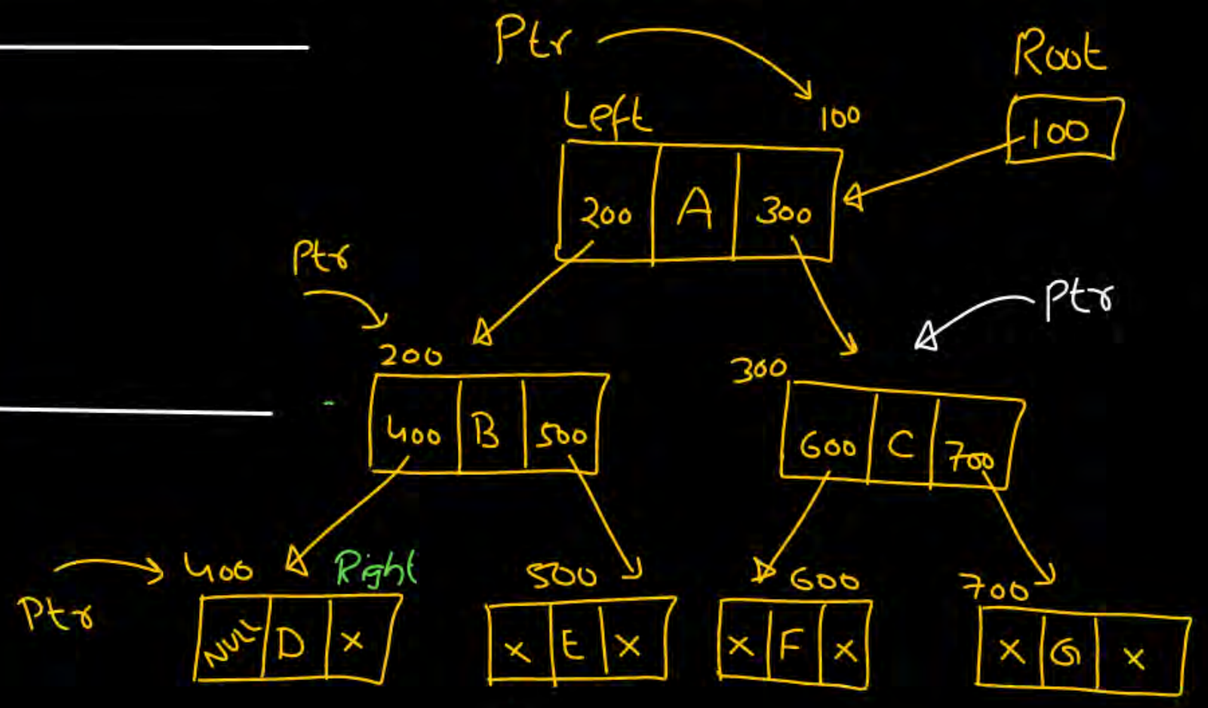

```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
    {
1.  pf("%d", Ptr->data);
2.  Preorder(Ptr->left);
3.  Preorder(Ptr->Right);
4.  }
}
```

ABDECFG

```
main()
{
    Preorder(ROOT)
    =
}
```



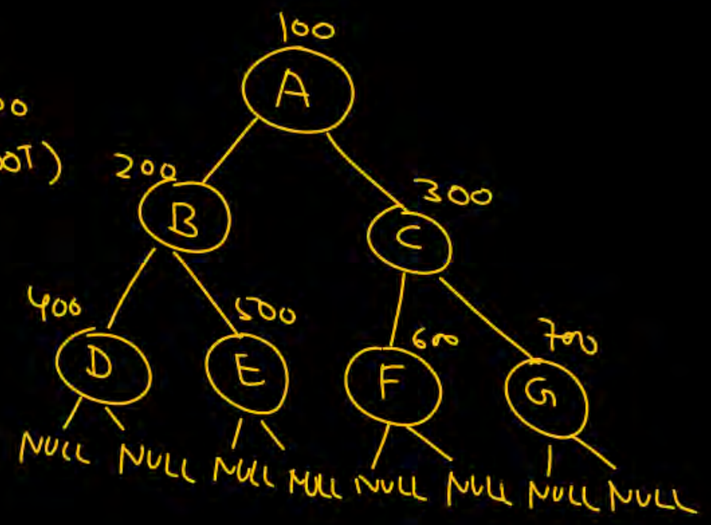
main	Ptr=100 Preorder(100) 1 ✓ 2 ✓ 3 ✓ 4	Ptr=300 Preorder(300) 1 ✓ 2 ✓ 3 ✓ 4 ✓			

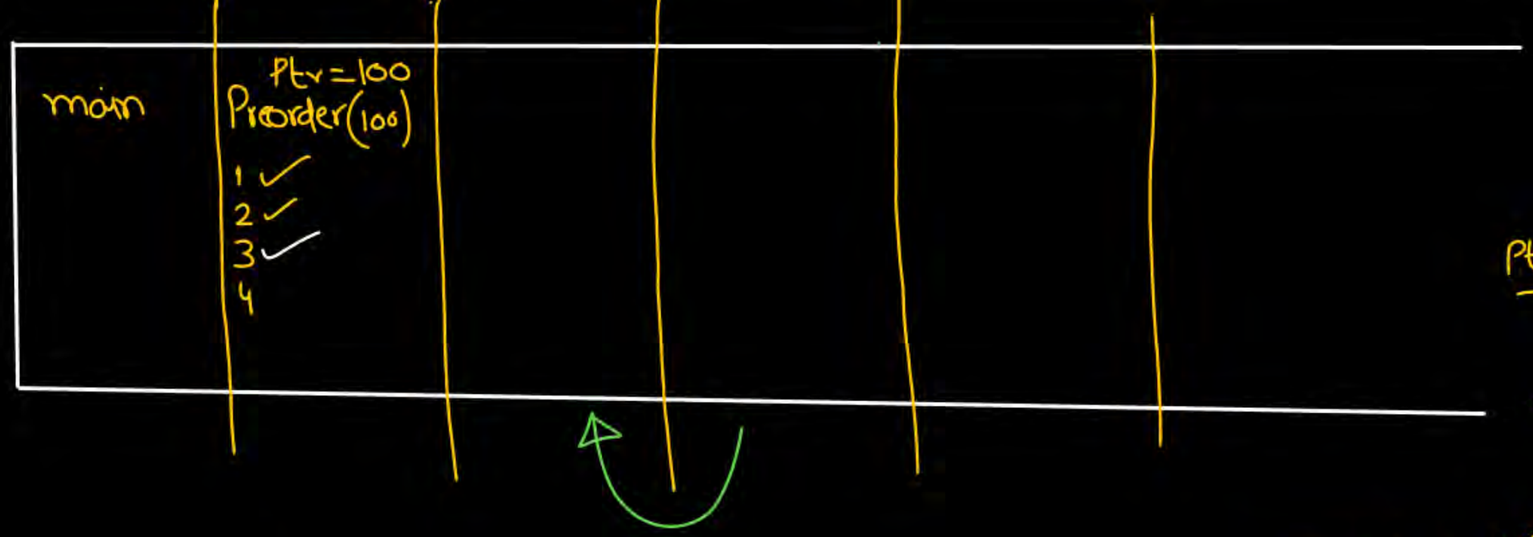


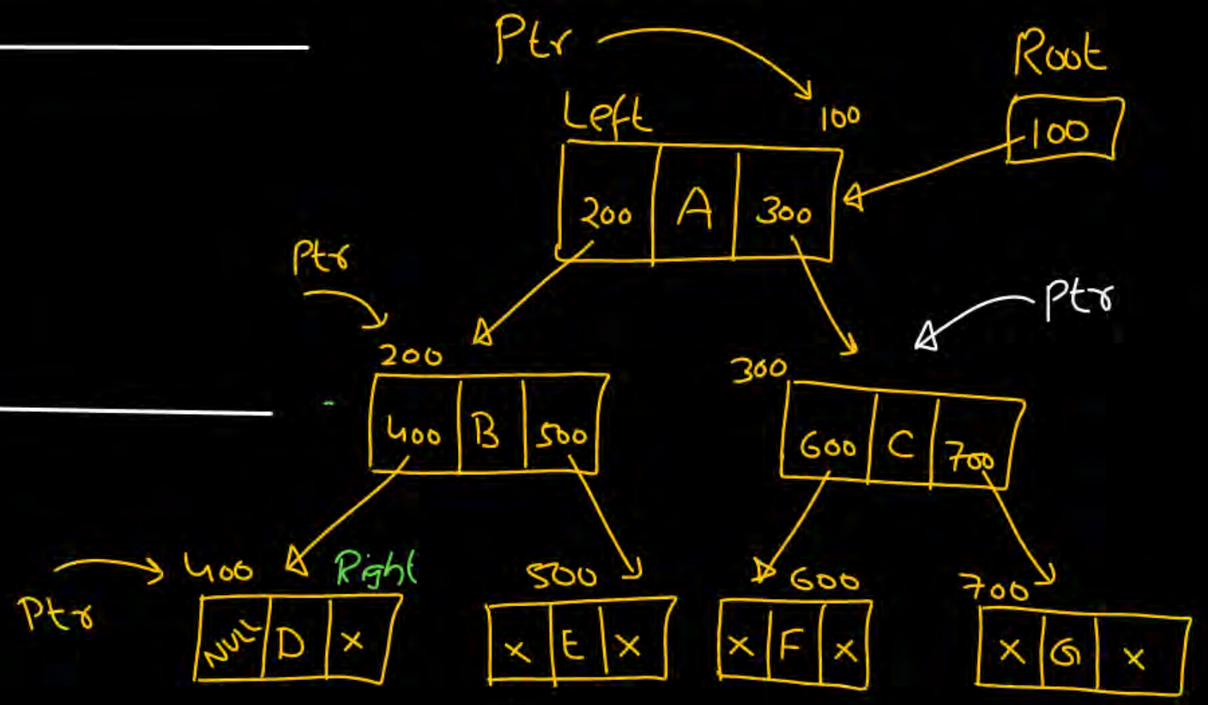

```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
    {
1.  pf("%d", Ptr->data);
2.  Preorder(Ptr->left);
3.  Preorder(Ptr->Right);
4.  }
}
```

ABDECFG

```
main()
{
    Preorder(ROOT)
    =
}
```





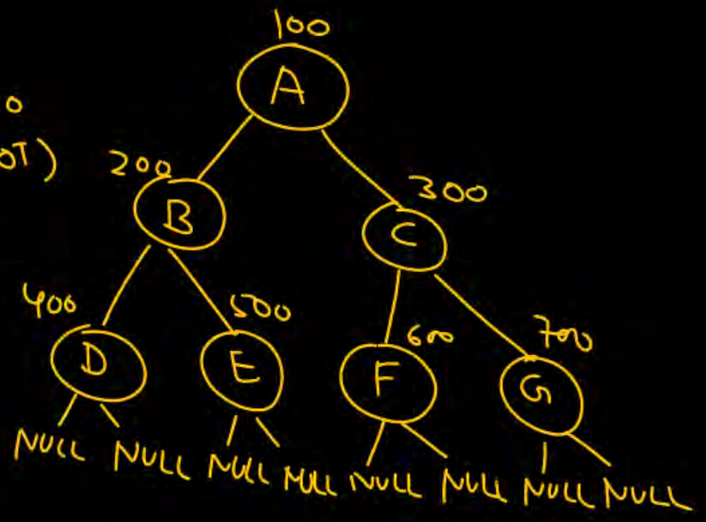


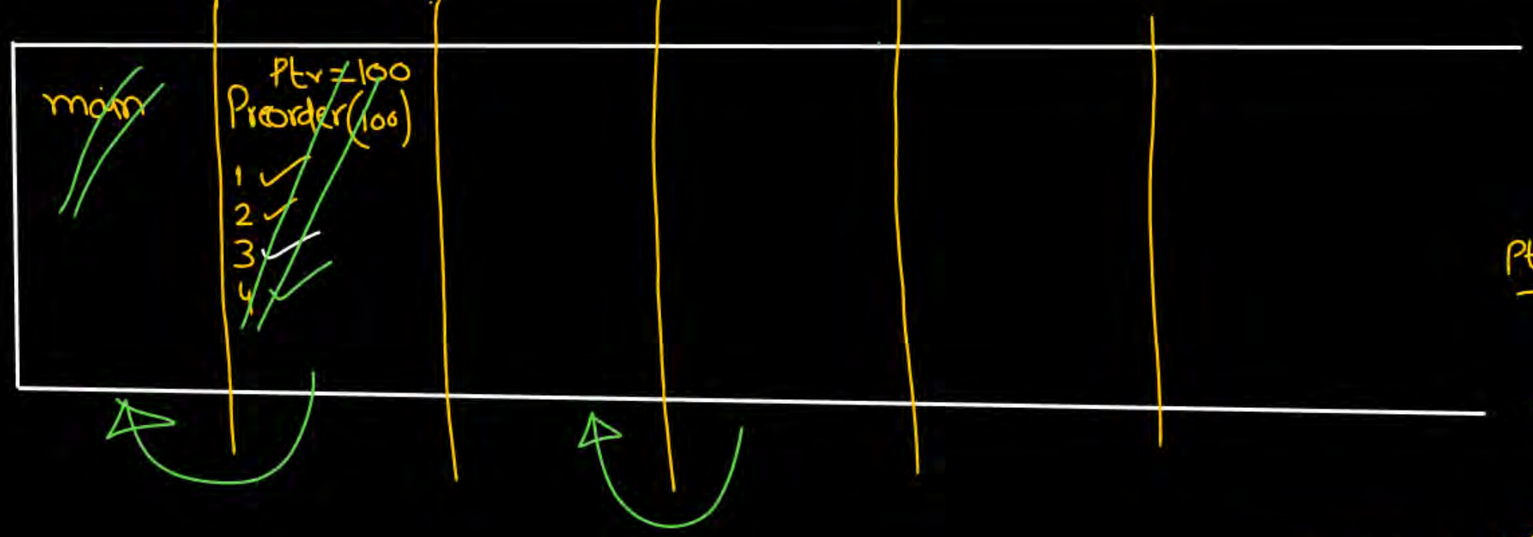

```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
    {
1.  pf("%d", Ptr->data);
2.  Preorder(Ptr->Left);
3.  Preorder(Ptr->Right);
4.  }
}
```

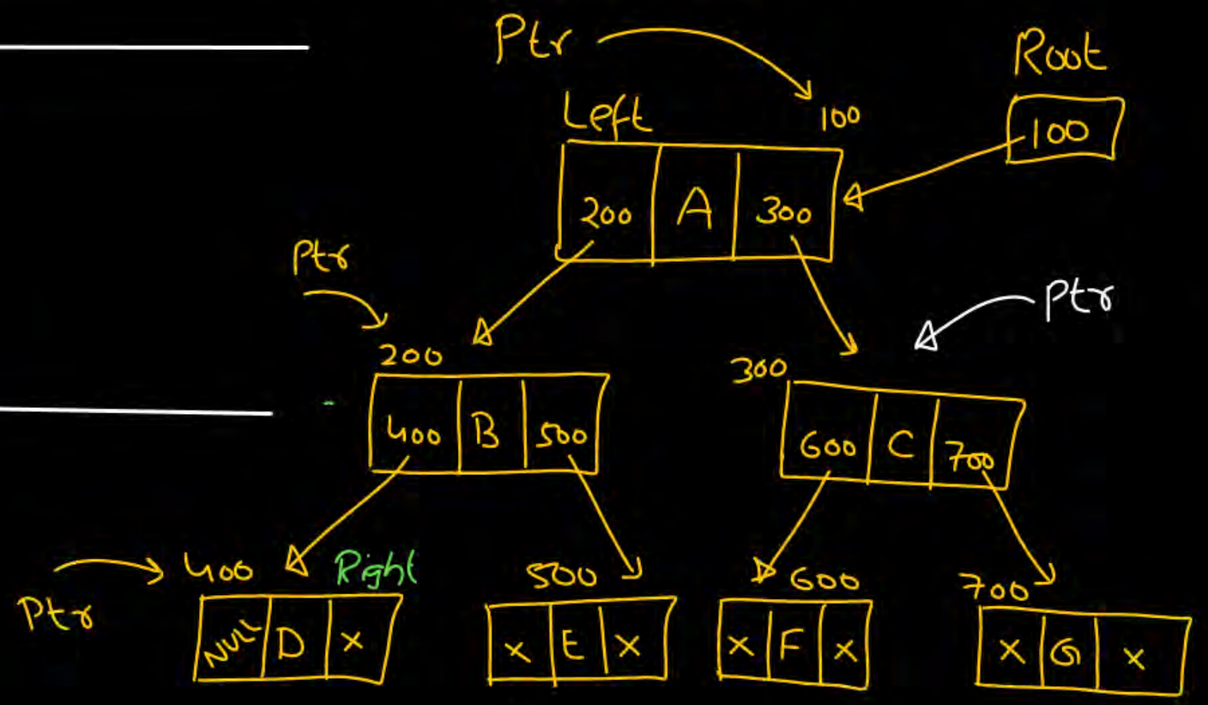
ABDECFG

```
main()
{
    Preorder(ROOT)
    =
}
```

100





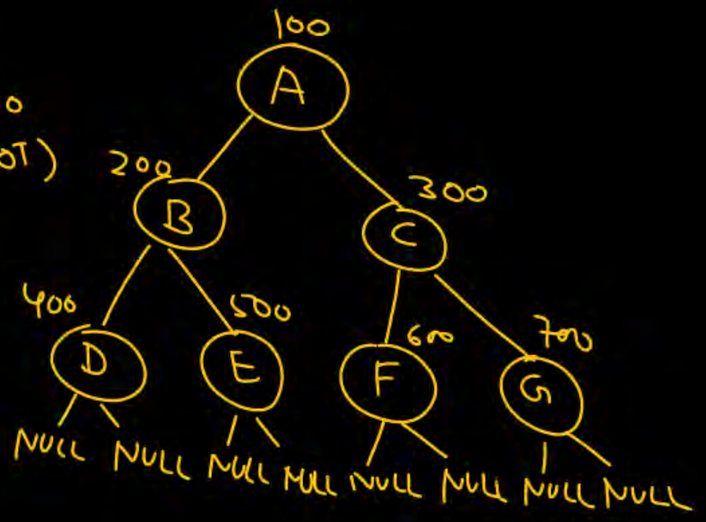


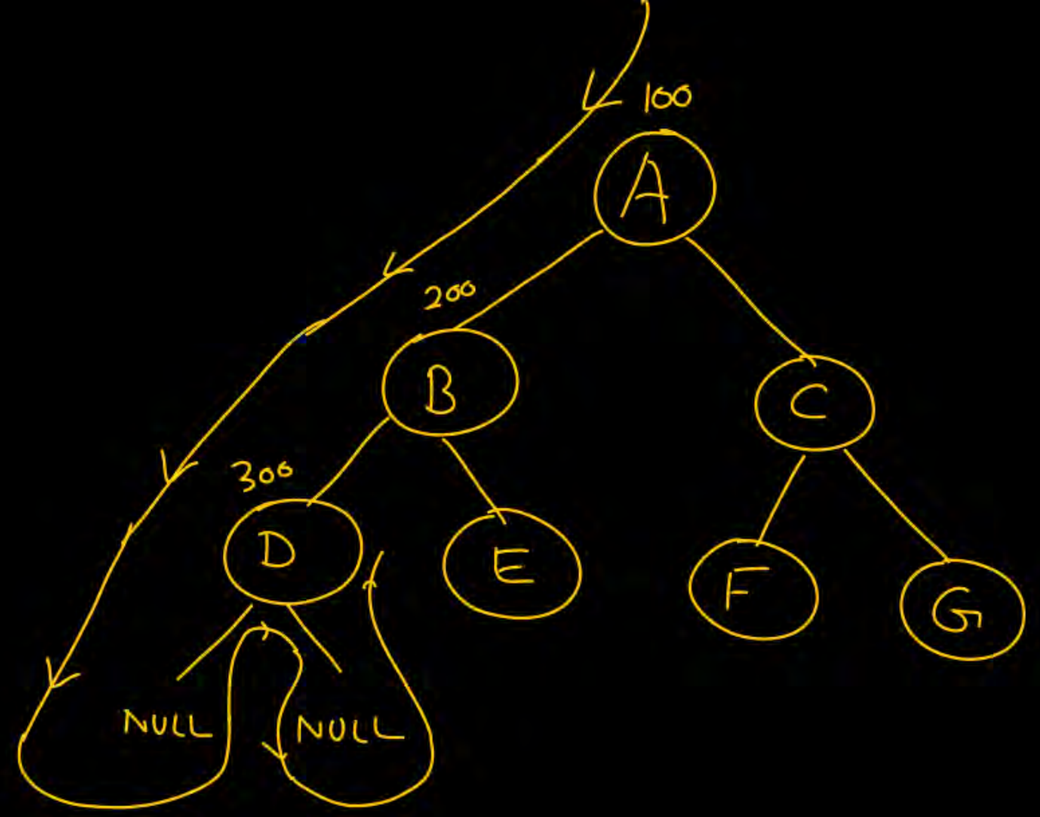

```
void Preorder(struct Node *Ptr){
    if (Ptr != NULL)
    {
1.      pf("%d", Ptr->data);
2.      Preorder(Ptr->Left);
3.      Preorder(Ptr->Right);
4.  }
}
```

ABDECFG

```
main()
{
    Preorder(ROOT)
    =
}
```

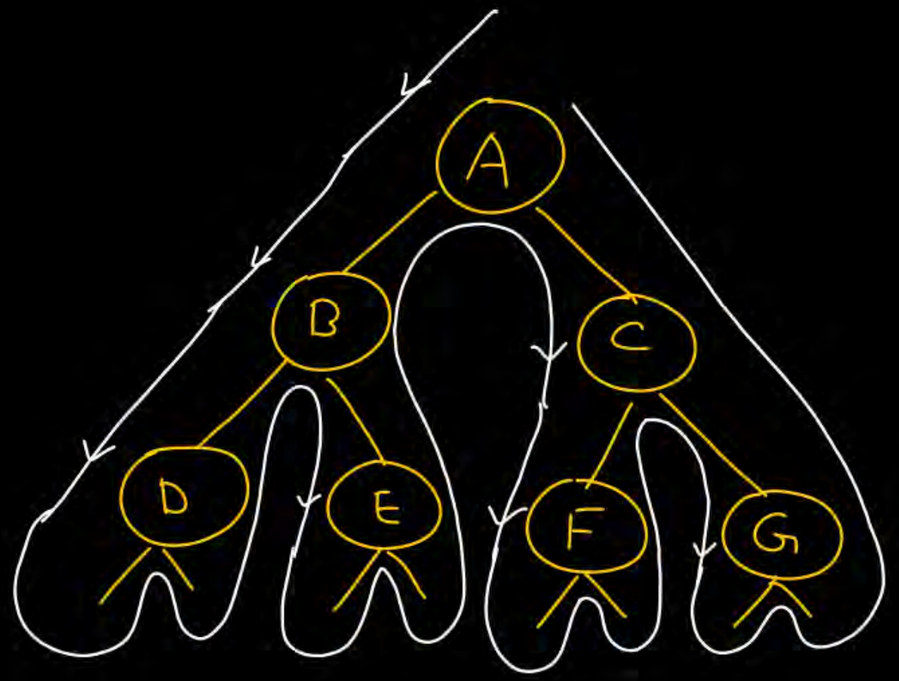
100

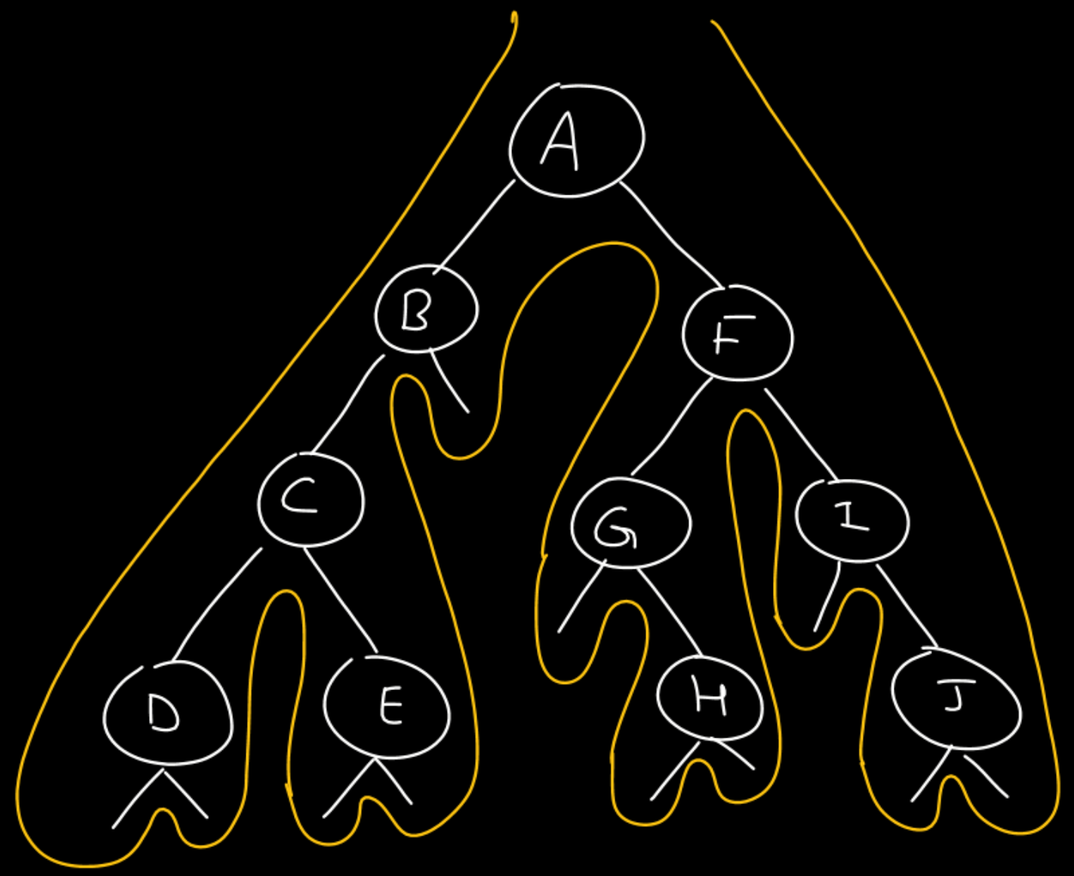




100 1 2 3	200 1 2	~~300~~ 1 2 3 4	Pre(~~NULL~~)

ABDECFG





ABCDEFGHIJ

log ⇒ unary operator

1 operand

Doubt ?

recursion in loop

bottom-up

C++/Java/DSA

standard Algo →

08 : 30 PM


THANK YOU GW
SOLDIERS !