CS & IT ENGINEERING

Data Structures

Tree

Lecture No.- 02

By- Pankaj Sharma Sir

Recap of Previous Lecture

Topic Tree Part-01

Node, child, Parent,
sibling, height, level.
size, Ancestor, descendant,
binary tree, n_{max}, n_{min}

Topics to be Covered

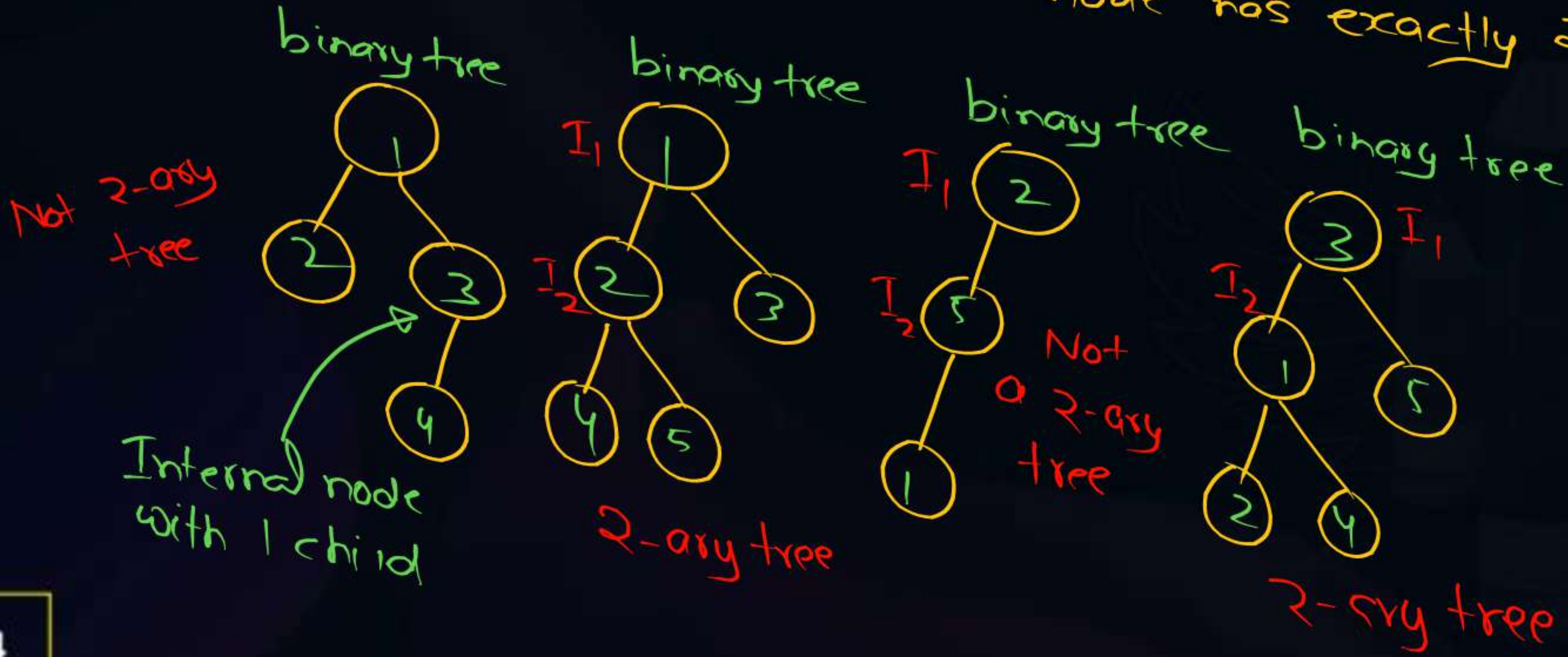
Topic Tree Part-02

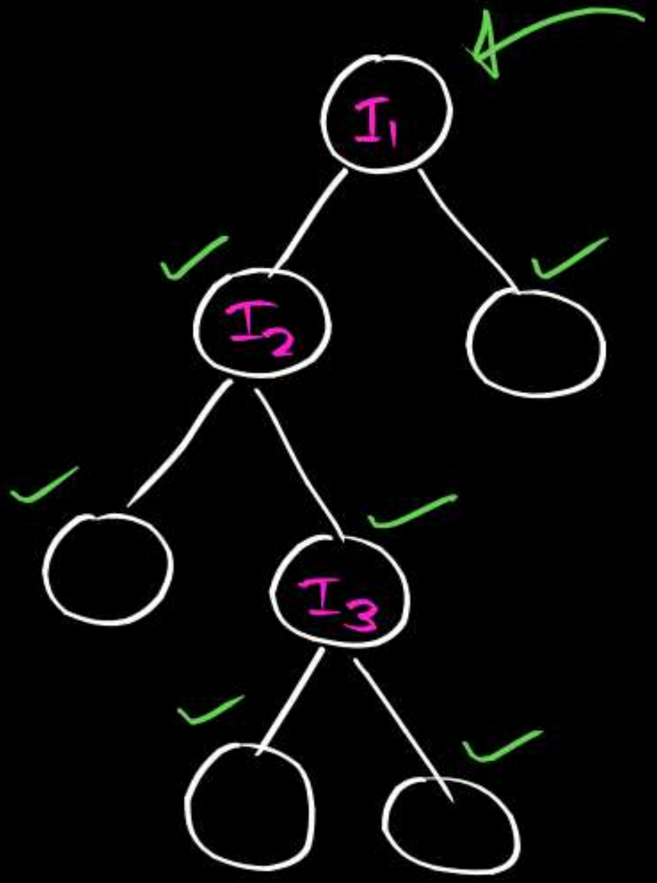
Topic : Tree

2-ary tree : Binary tree in which every node has either 0 child or 2 child.

OR

Binary tree in which every internal node has exactly 2 child.





3 internal nodes

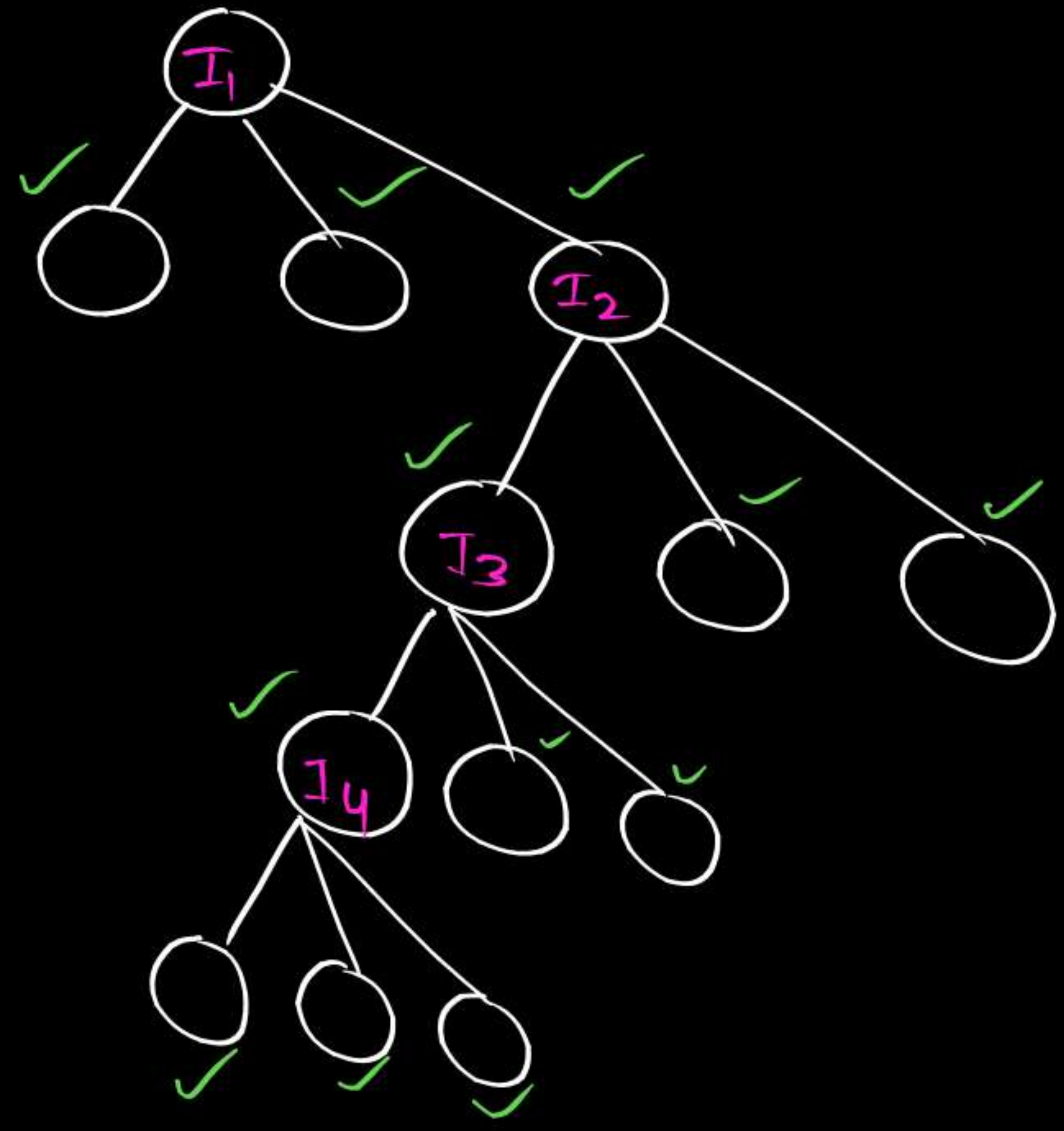
3 nodes of degree 2

3 internal node ⇒ Each with 2 childs

$$\underbrace{3 \times 2}$$

Total nodes = $3 \times 2 + \overset{\text{(root)}}{1}$

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



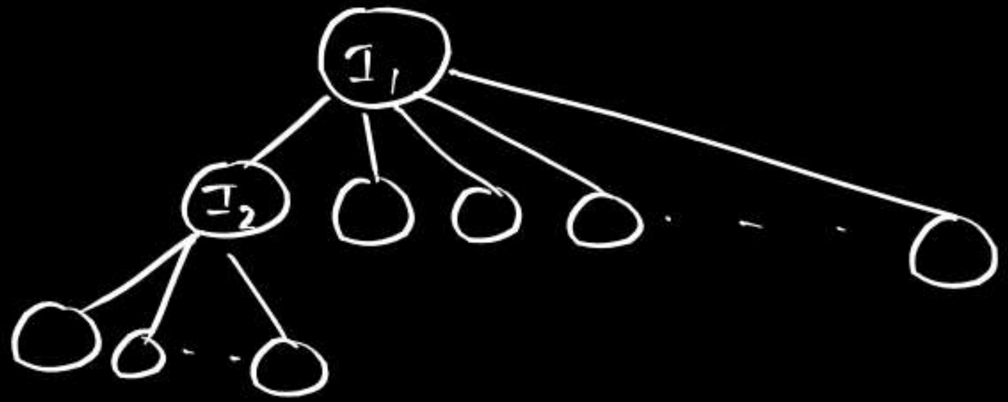
4 internal nodes ⇒ 3 child

$$\underset{\text{Internal nodes}}{4} \times \underset{\text{Every internal node has 3 child}}{3}$$

Total nodes $= 4 \times 3 + \overset{\text{(Root)}}{1}$

K-ary tree : Every internal node has exactly K childs.

let I : no. of internal nodes



Total nodes = $I \times K + \overset{\text{(root)}}{1}$

$$\boxed{N = K \cdot I + 1}$$

$$N = kI + 1$$

# of leaf nodes + # of nodes $= kI + 1$

$$L + I = kI + 1$$

$$L = KI - I + 1$$

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

1) $N = k \cdot I + 1$

2.) $L = (k-1)I + 1$

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

$$\Rightarrow I = \frac{(L-1)}{(k-1)}$$

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

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

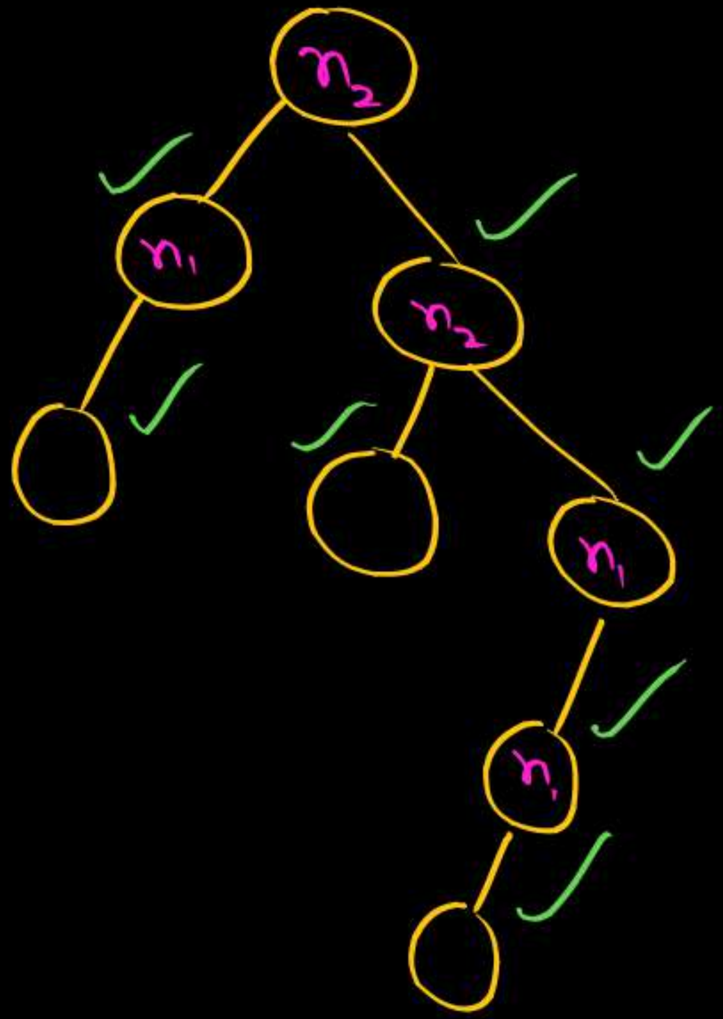
$$N = f(L)$$

Don't learn any of these

$$\boxed{N = \frac{k \cdot L - 1}{k-1}} \text{ —(3)}$$

Q A binary tree has 6 nodes of degree 1 , 12 nodes of degree 2 . find the no. of leaf nodes.

ex



$n_1 \longrightarrow 3 \longrightarrow 3 \times 1$

$n_2 \longrightarrow \textcircled{2} \longrightarrow 2 \times 2$

$+$

1 (root)

$N = 3 + 4 + 1 = 8$

Q A binary tree has 6 nodes of degree 1 , 12 nodes of degree 2 . find the no. of leaf nodes.

$$N = 6 \times 1 + 12 \times 2 + \overset{\text{root}}{1}$$

$$N = 31$$

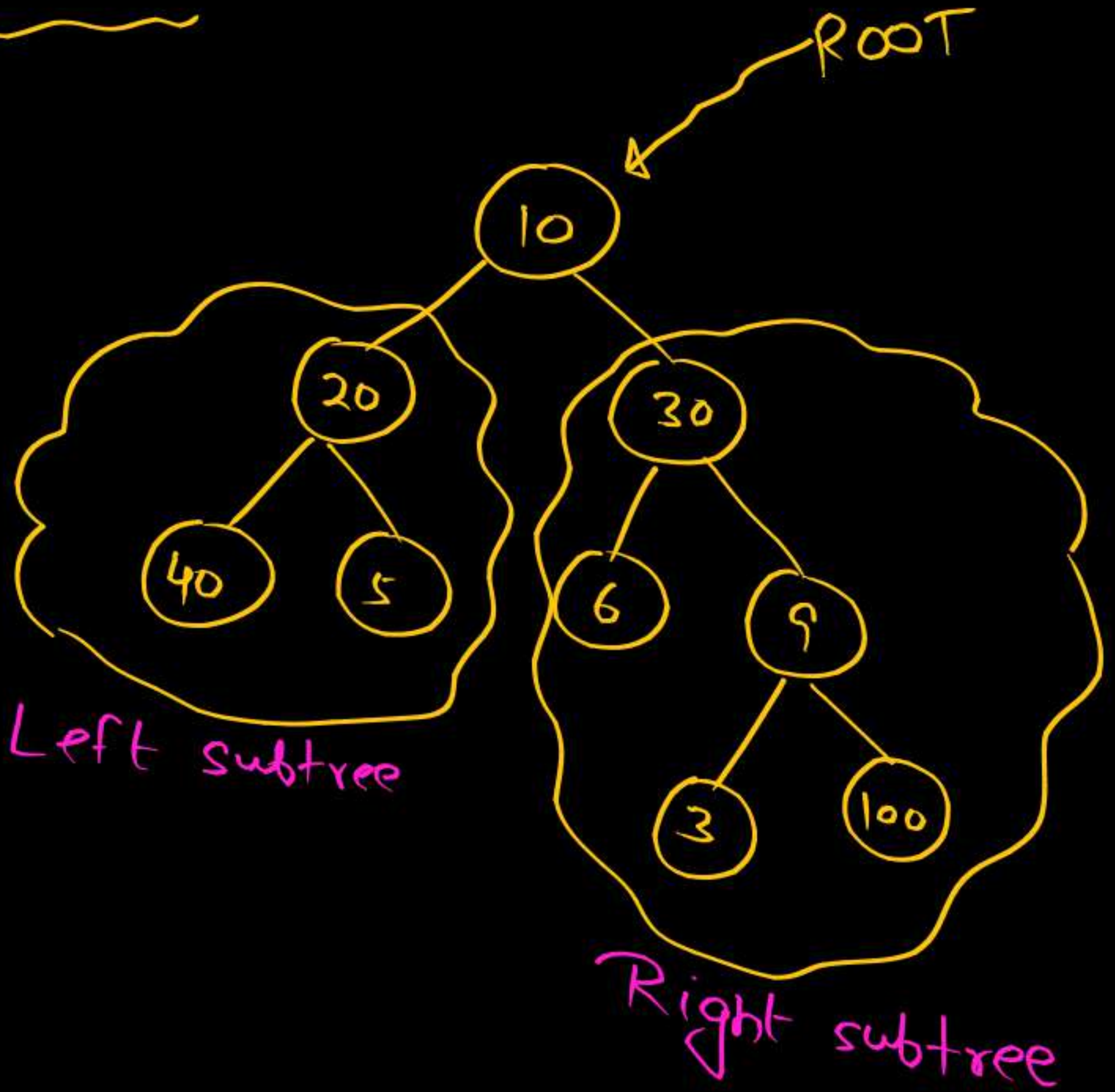
$$\underbrace{\text{\# node with 0 degree}} + \text{\# nodes with 1 degree} + \text{\# node with 2 degree} = 31$$

$$L + 6 + 12 = 31$$

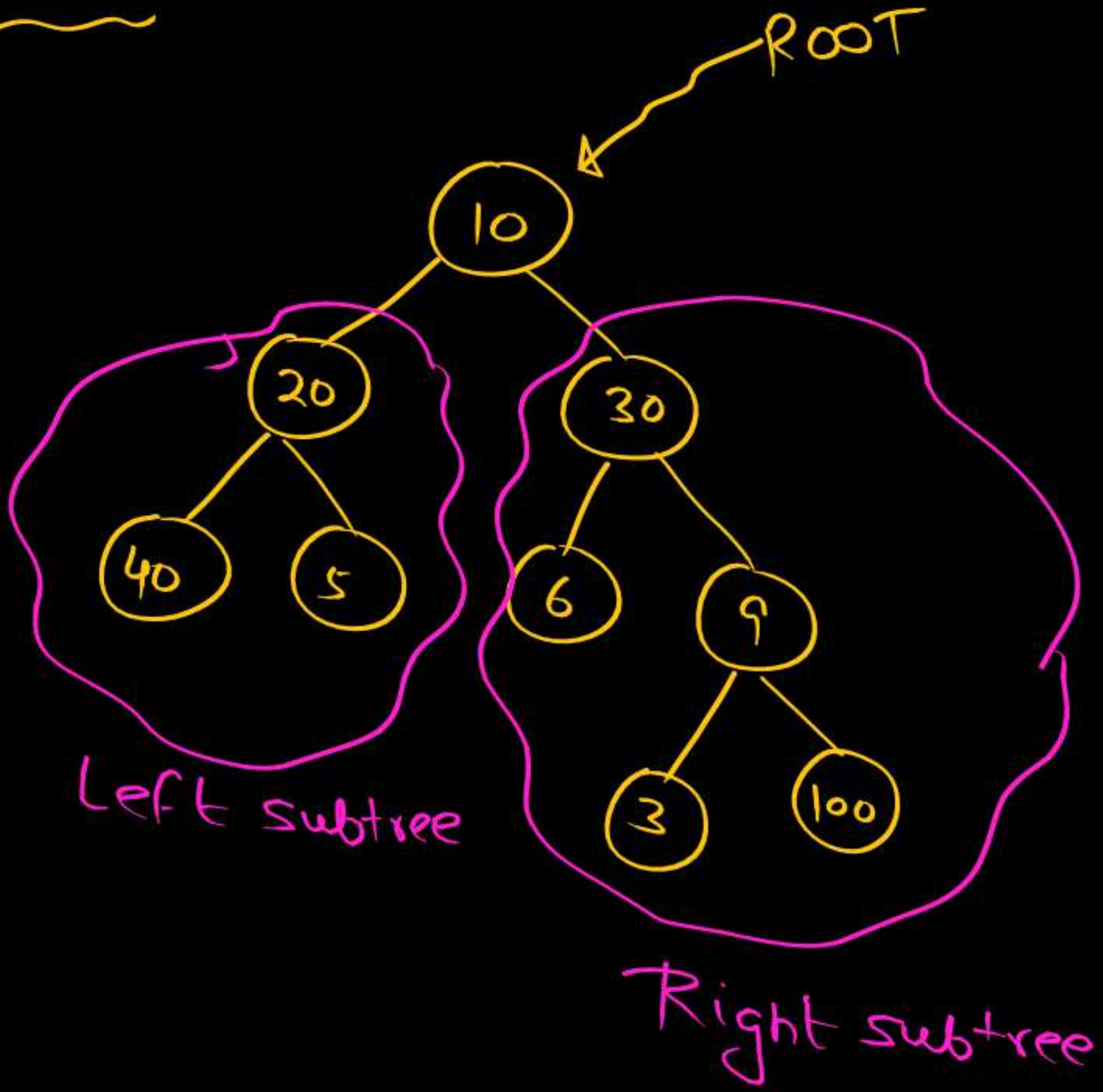
$$L + 18 = 31$$

$$\boxed{L = 13}$$

Tree Traversal

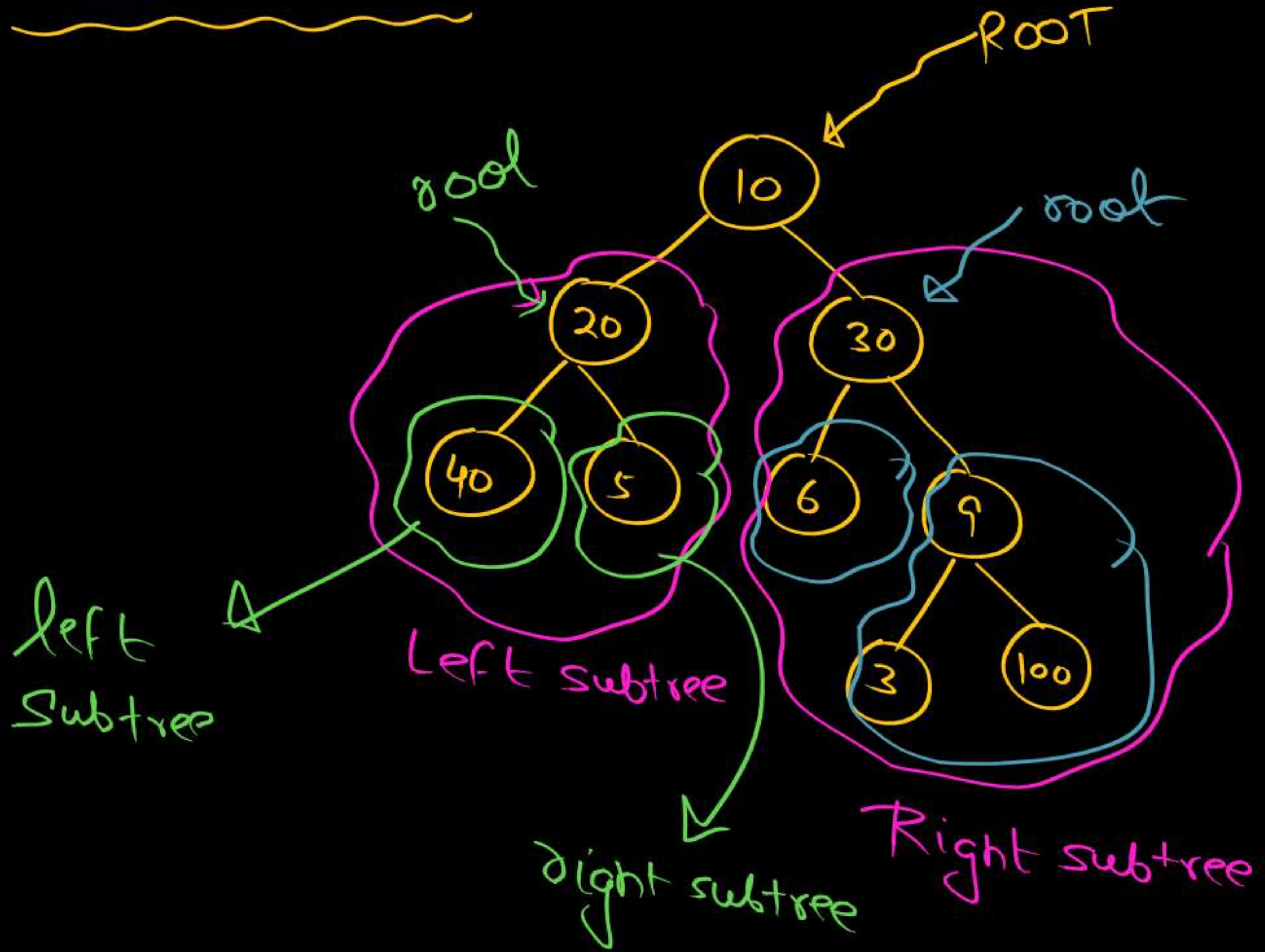


Tree Traversal



Tree Traversal

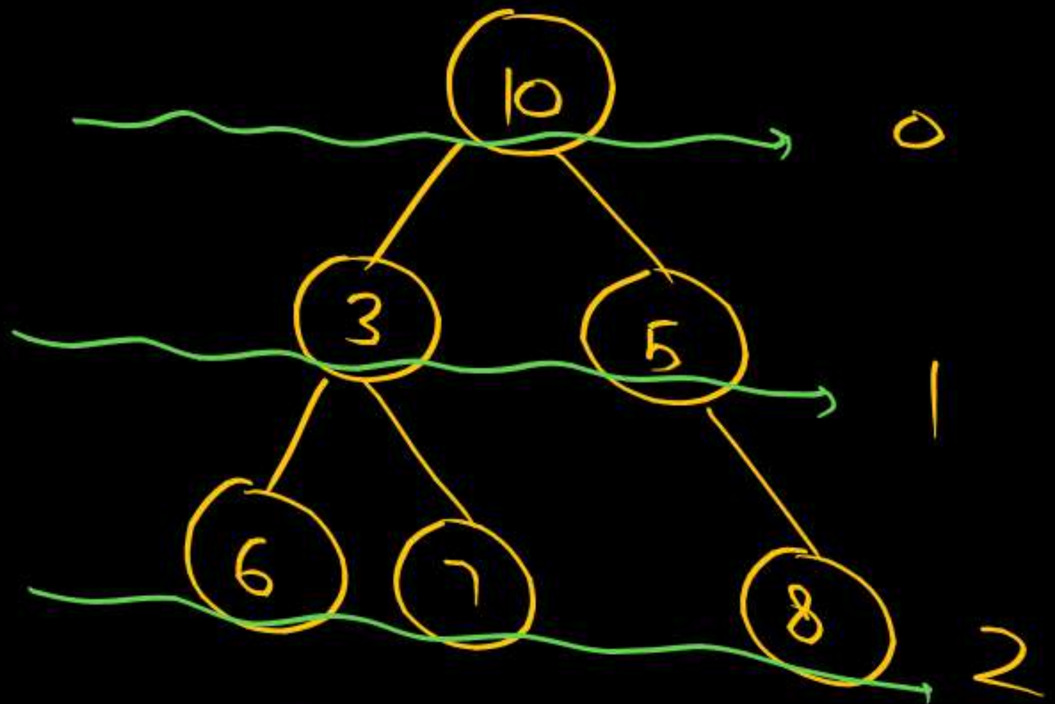
Root, L_T, R_T



Tree Traversal

Level order traversal

level



10 3 5 6 7 8

Depth order

Root , L_T, R_T

3! = 6 ways

✓

1.) Root, L_T, R_T

2.) L_T, Root, R_T

3.) L_T , R_T , Root

L_T is traversed before R_T

4.) Root, R_T , L_T

5.) R_T , Root, L_T

6.) R_T, L_T , Root

1.) ROOT, L_T, R_T Preorder traversal

2.) L_T, ROOT, R_T Inorder traversal

3.) L_T, R_T, ROOT Postorder traversal

Pre-order Traversal

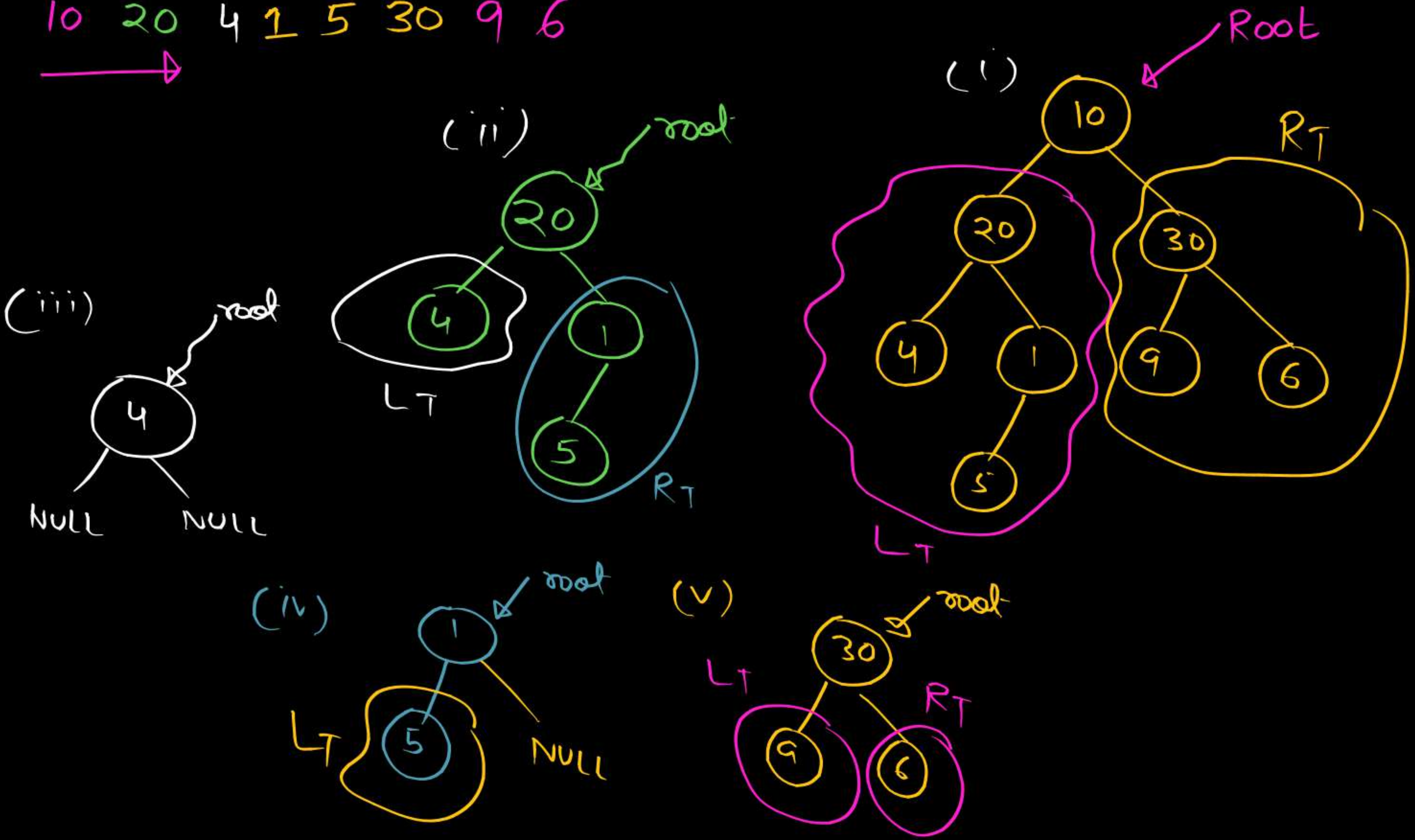
Preorder Traversal

1.) visit/print/process the Root node.

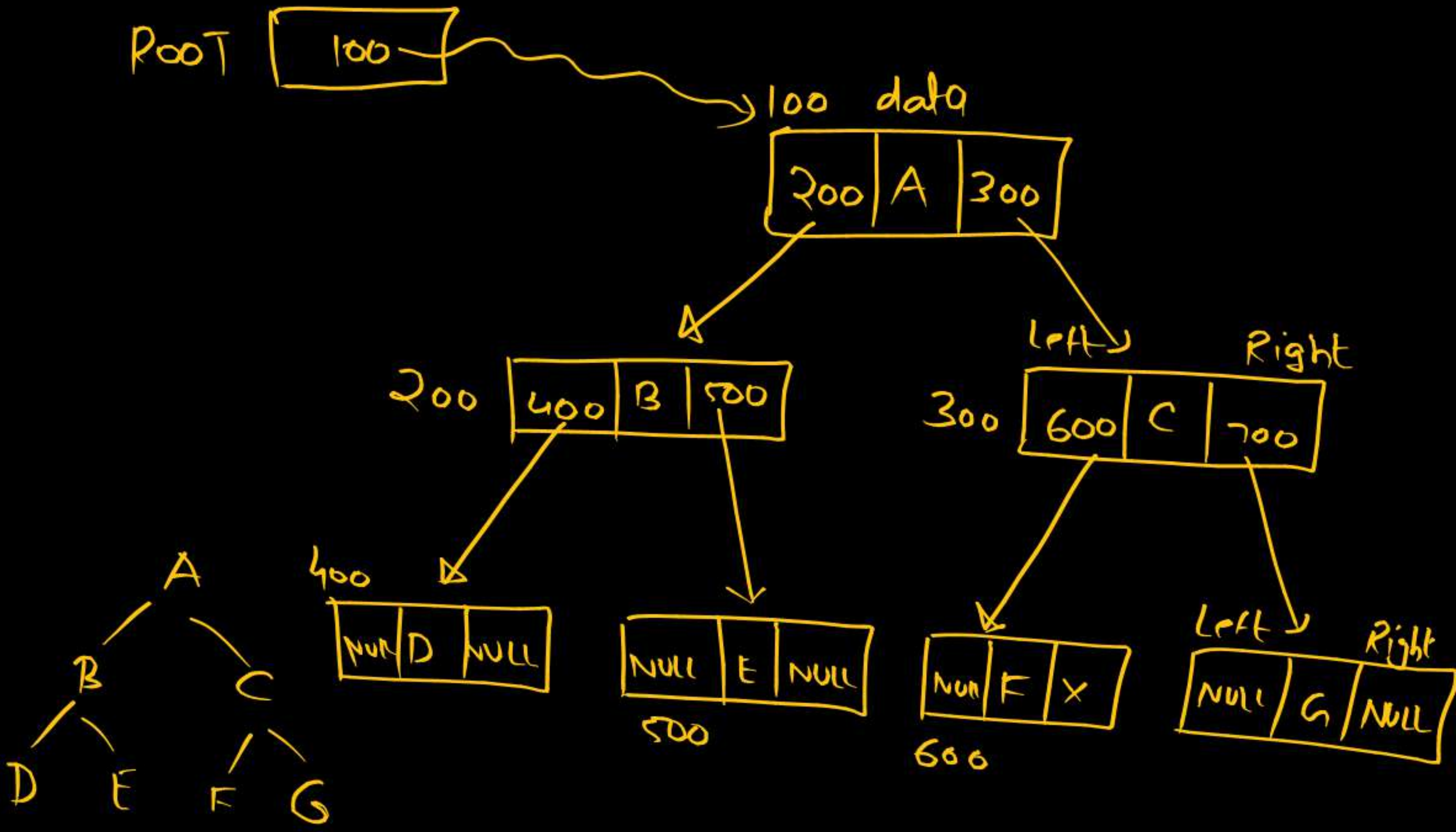
2.) Troverse the left subtree(L_T) in Preorder.

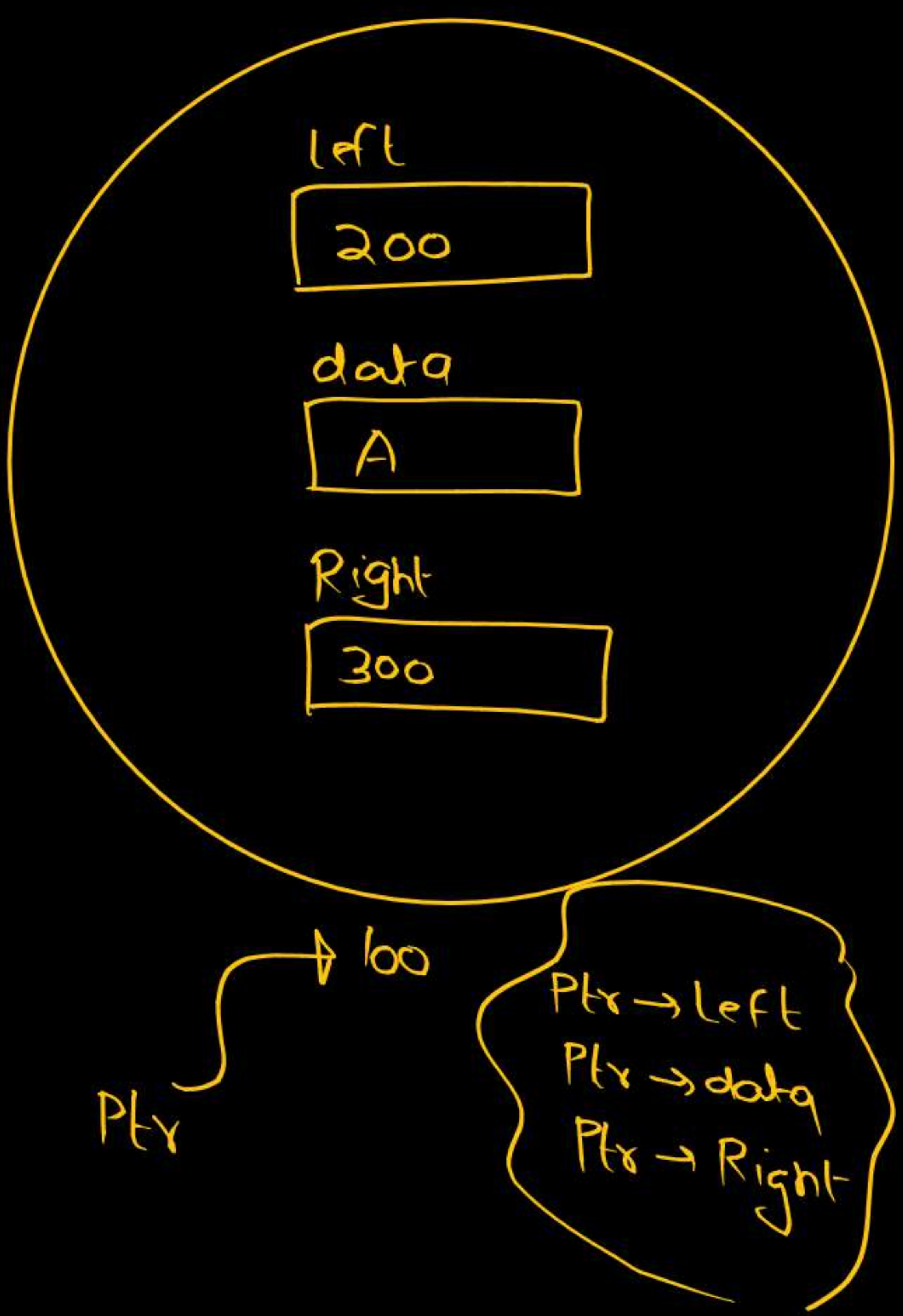
3.) Traverse the Right subtree(R_T) of root node in Preorder.

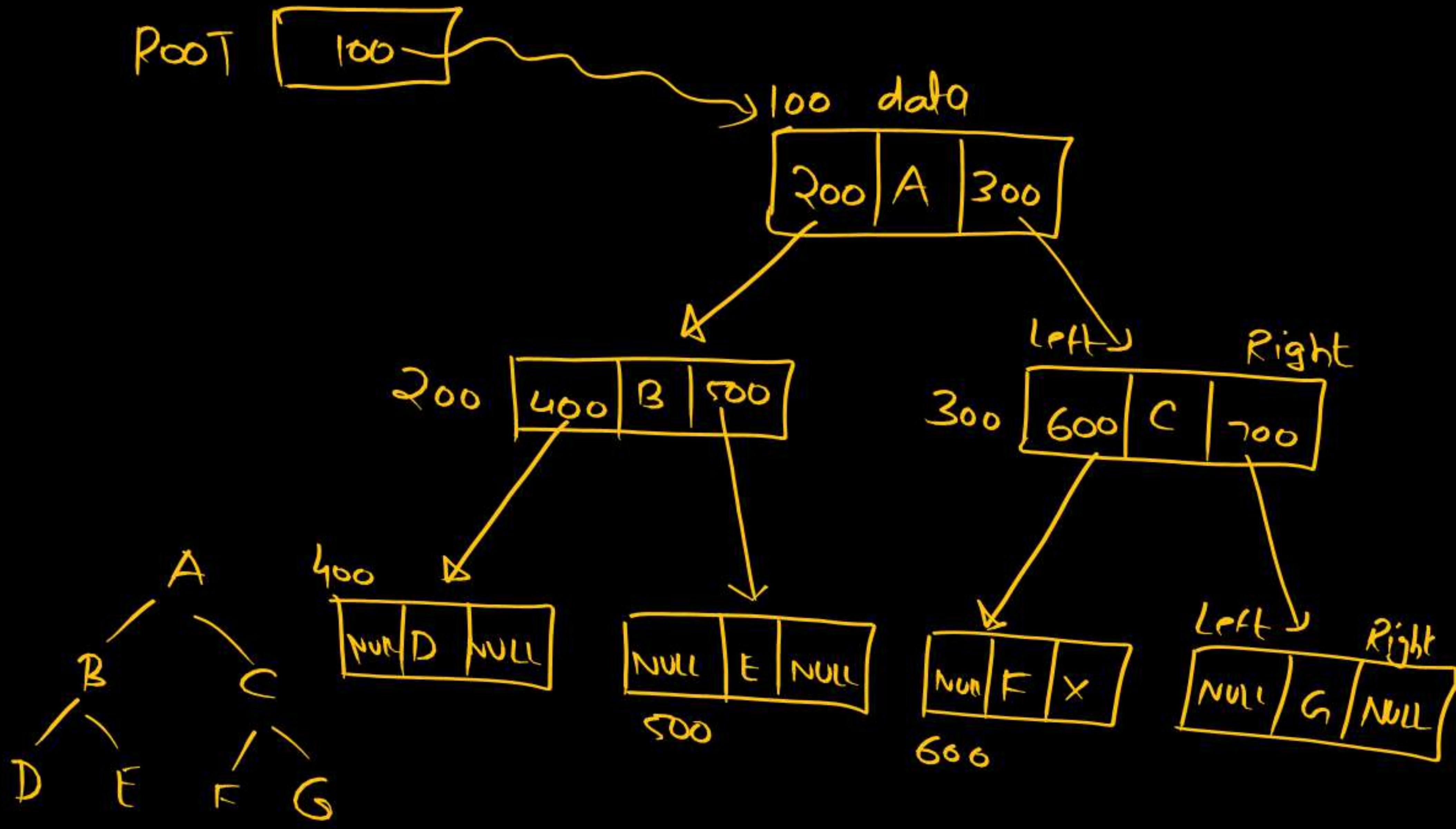
Preorder: 10 20 4 1 5 30 9 6



```
struct Node{
  struct Node *left;
      char  data;
  struct Node *Right;
          } *ROOT=NULL;
```

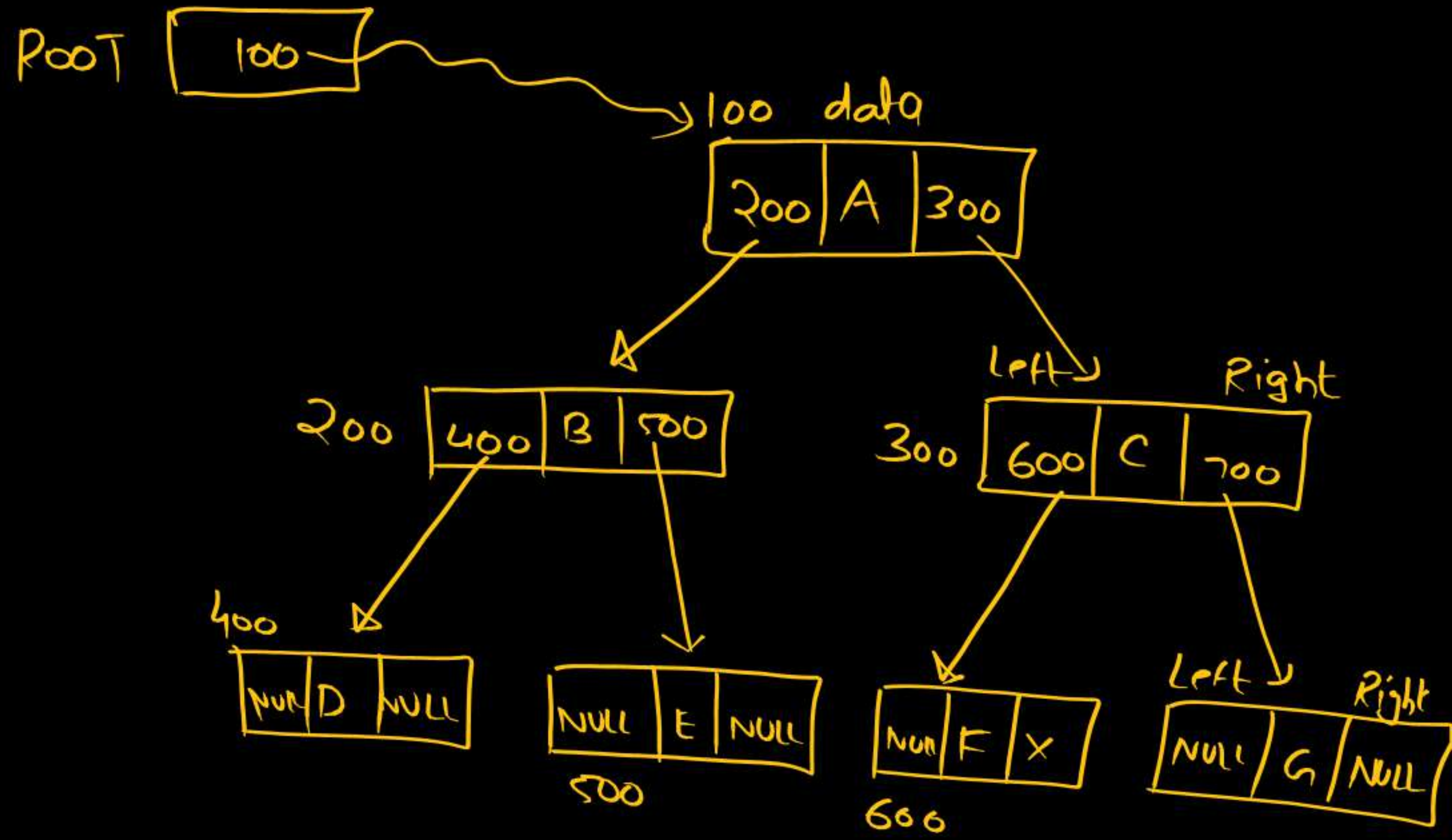


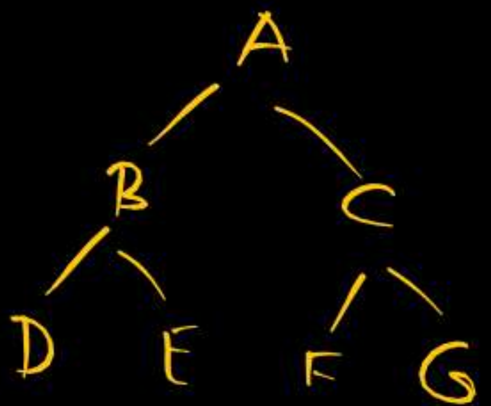




```
void main() {
    ≡
    Preorder( ROOT );
    ≡
}
```

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



    }
```



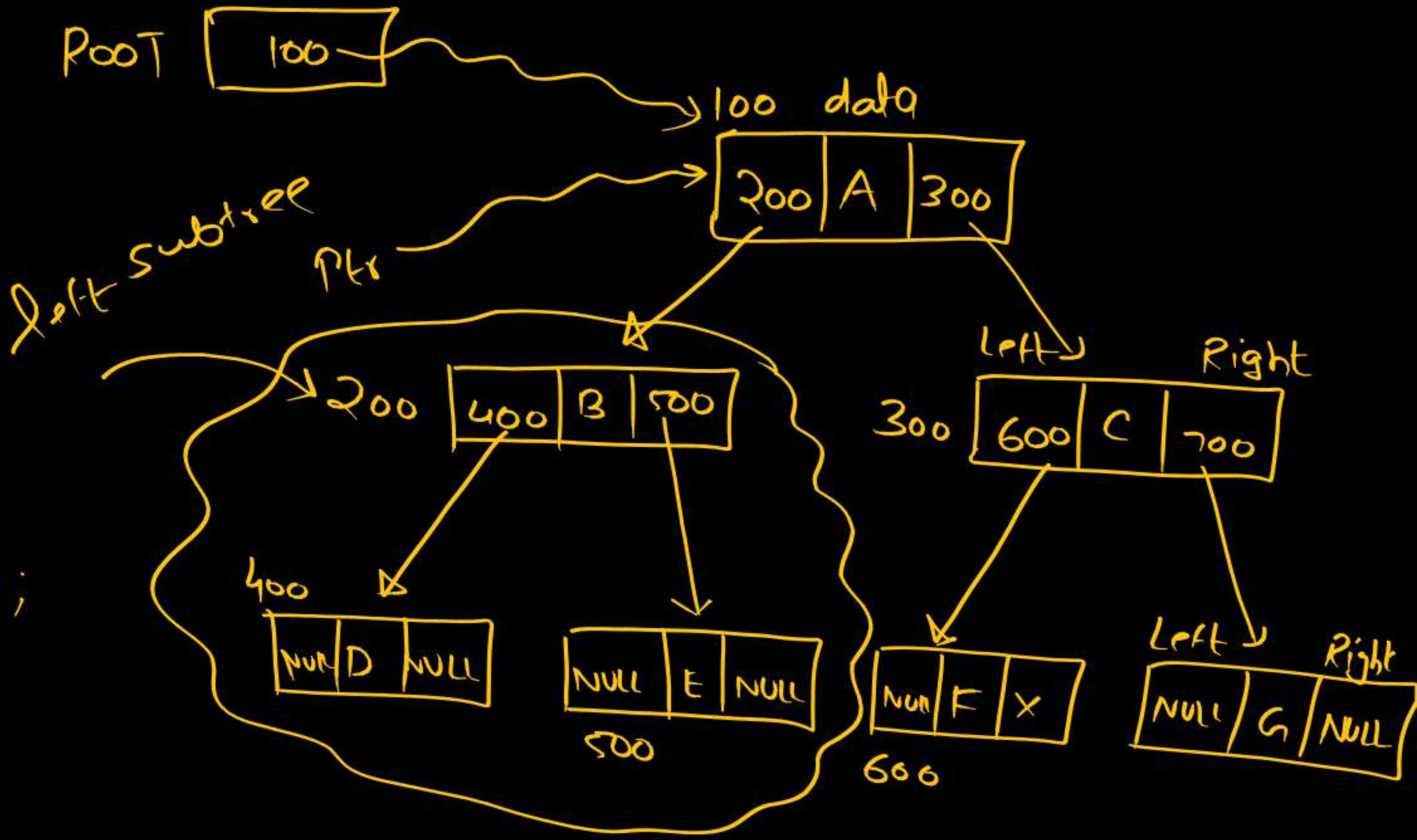


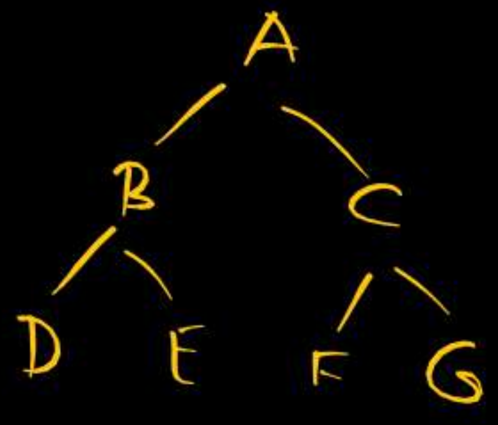
```
void main(){
        ≡
  Preorder( ROOT );
        ≡
}
```

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



  }
```

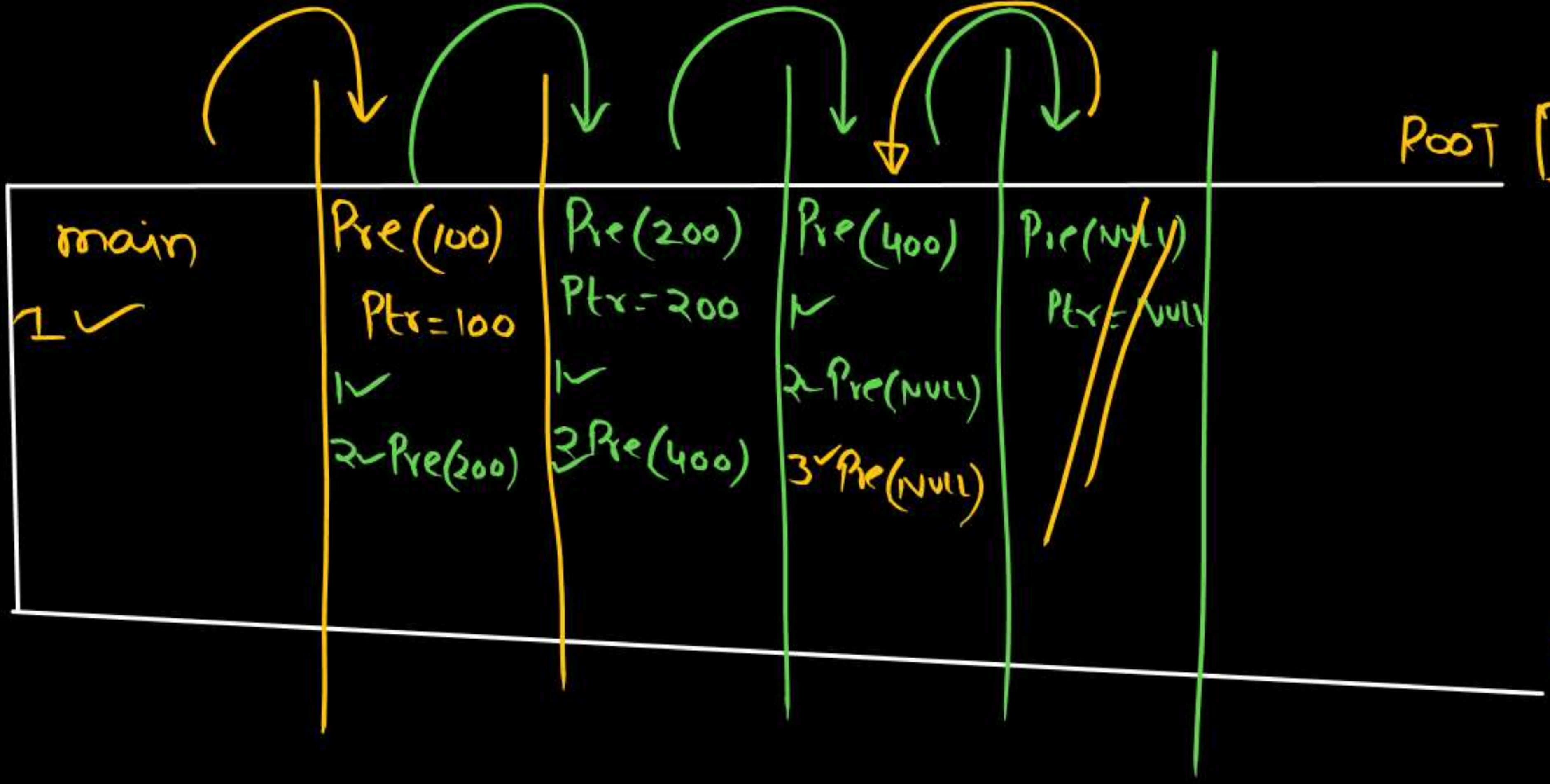
```
void main( ) {
      ≡
    Preorder(ROOT);
        ≡
        }
```

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

1.  printf("%c", Ptr→data);
2.  Preorder( Ptr→Left);
3.  Preorder(Ptr→Right);

4. }
```

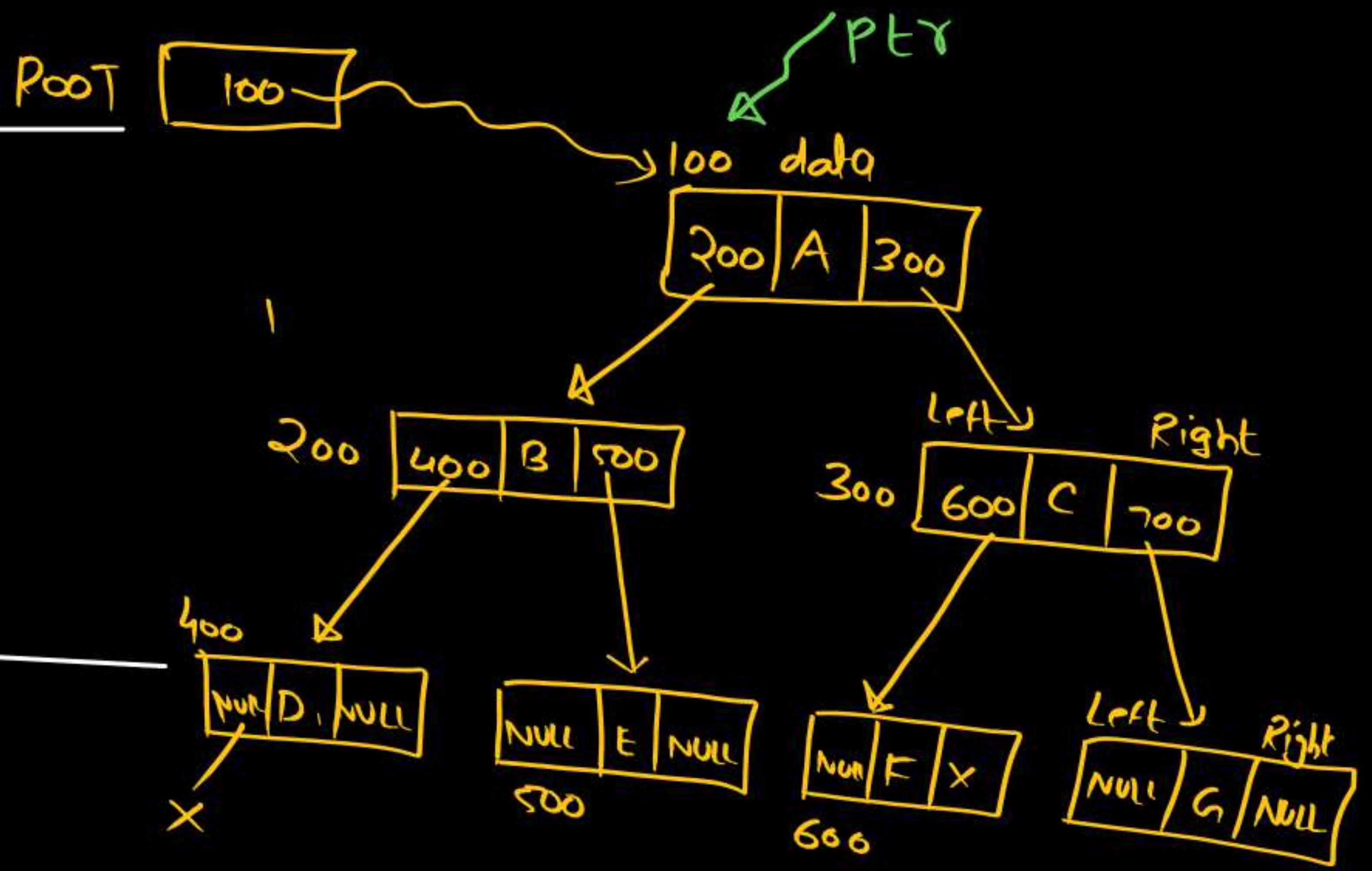


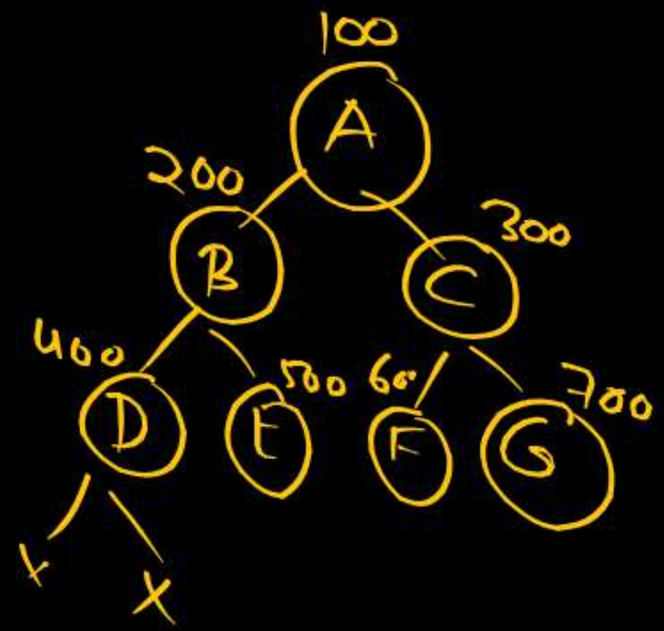


```
void main(){
        ≡
   Preorder( ROOT         );
        ≡
        }
```

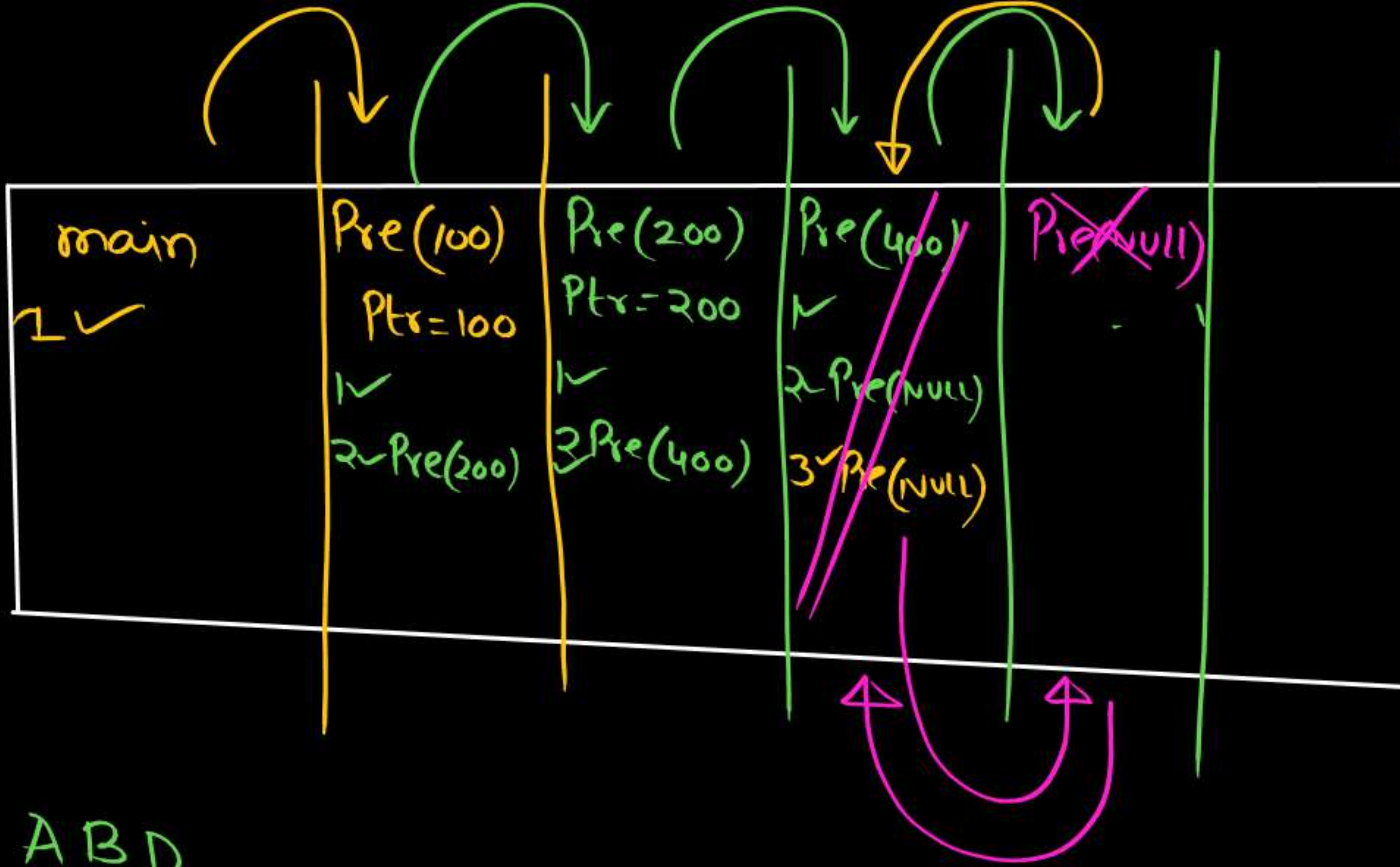


A B D

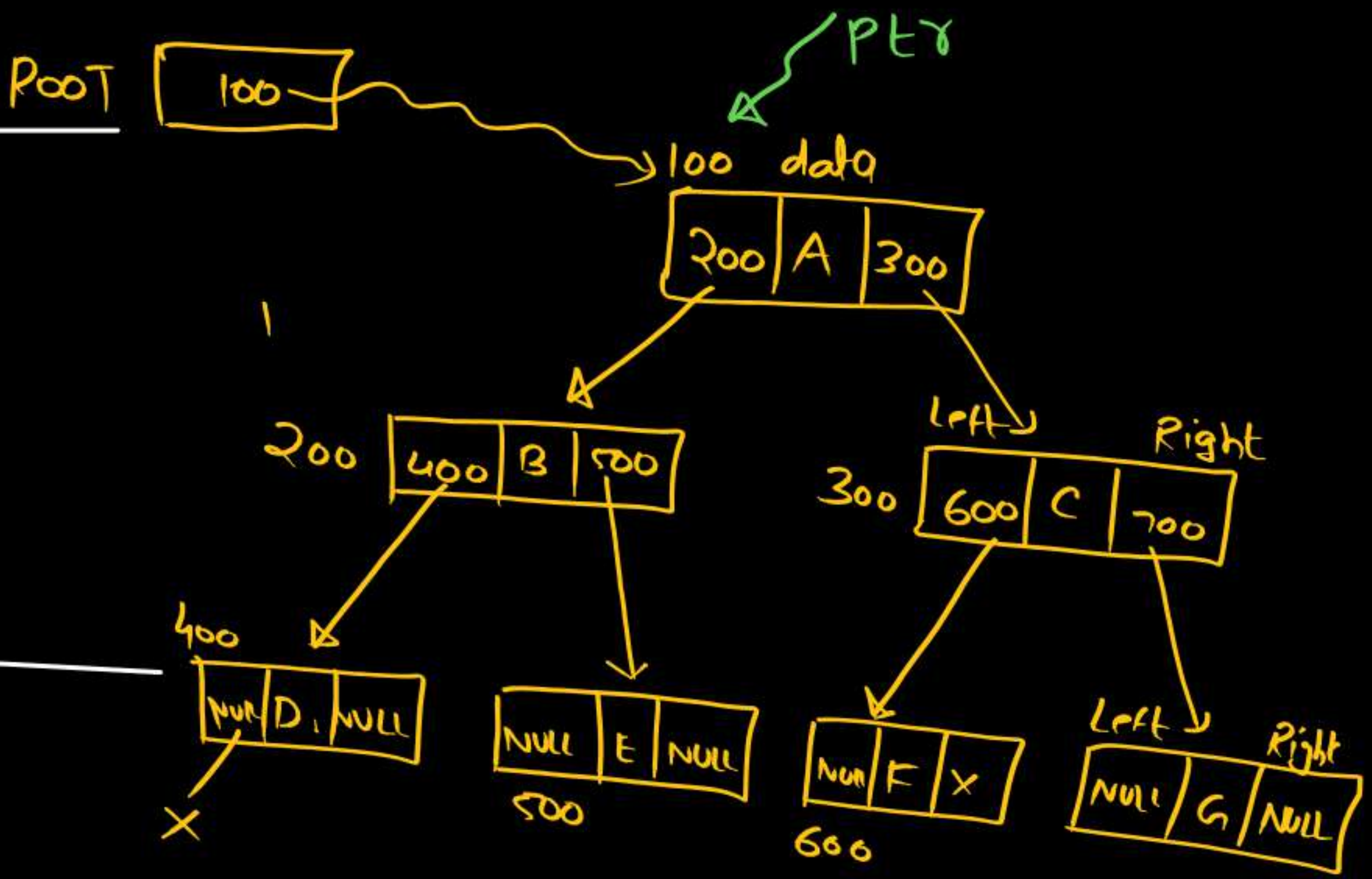


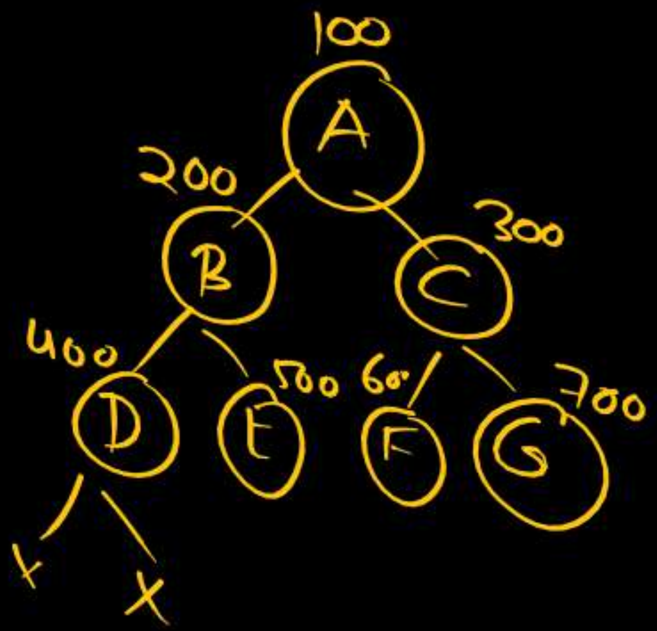


```
void main(){

1) Preorder( ROOT );

2)

}
```

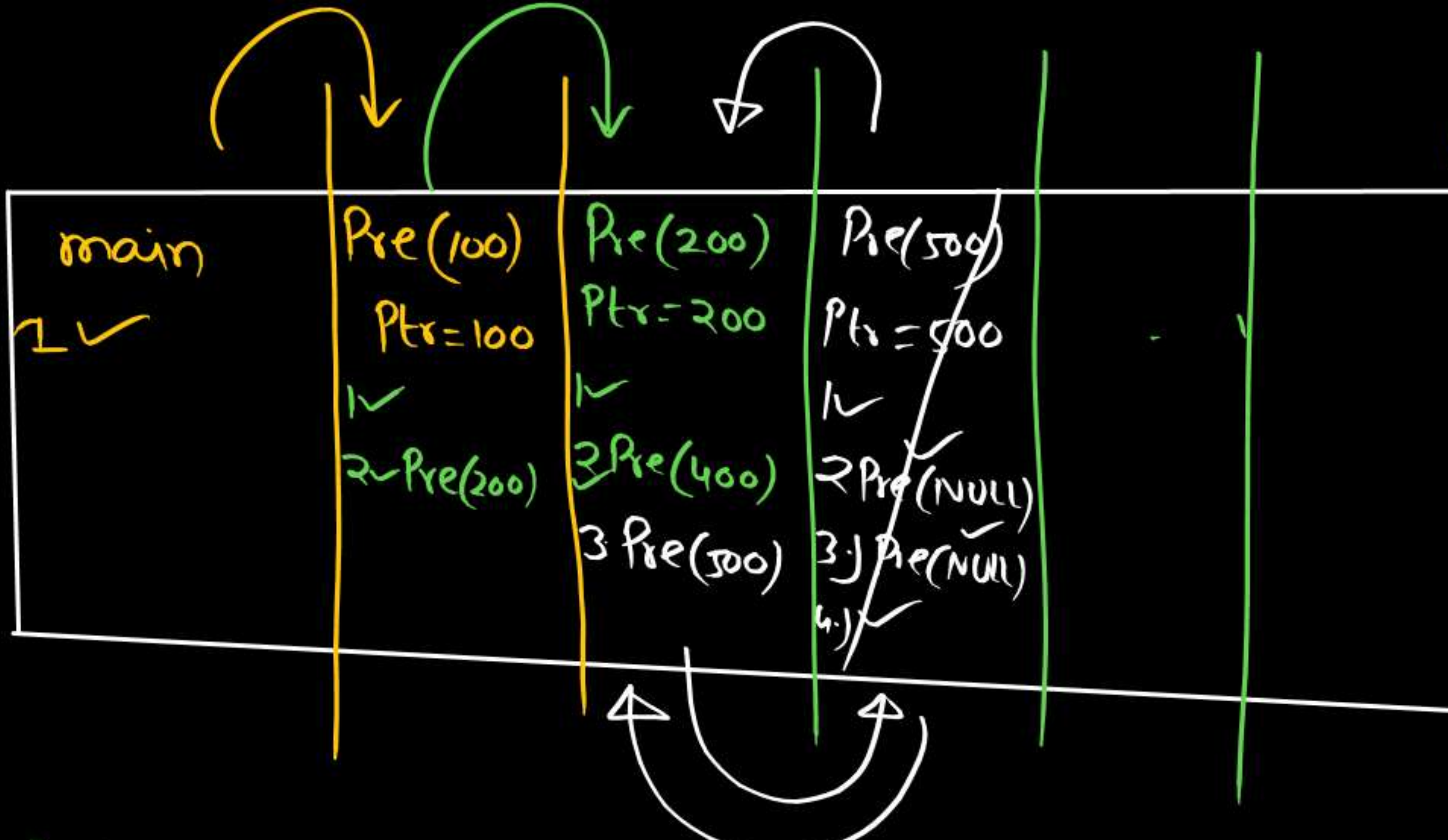


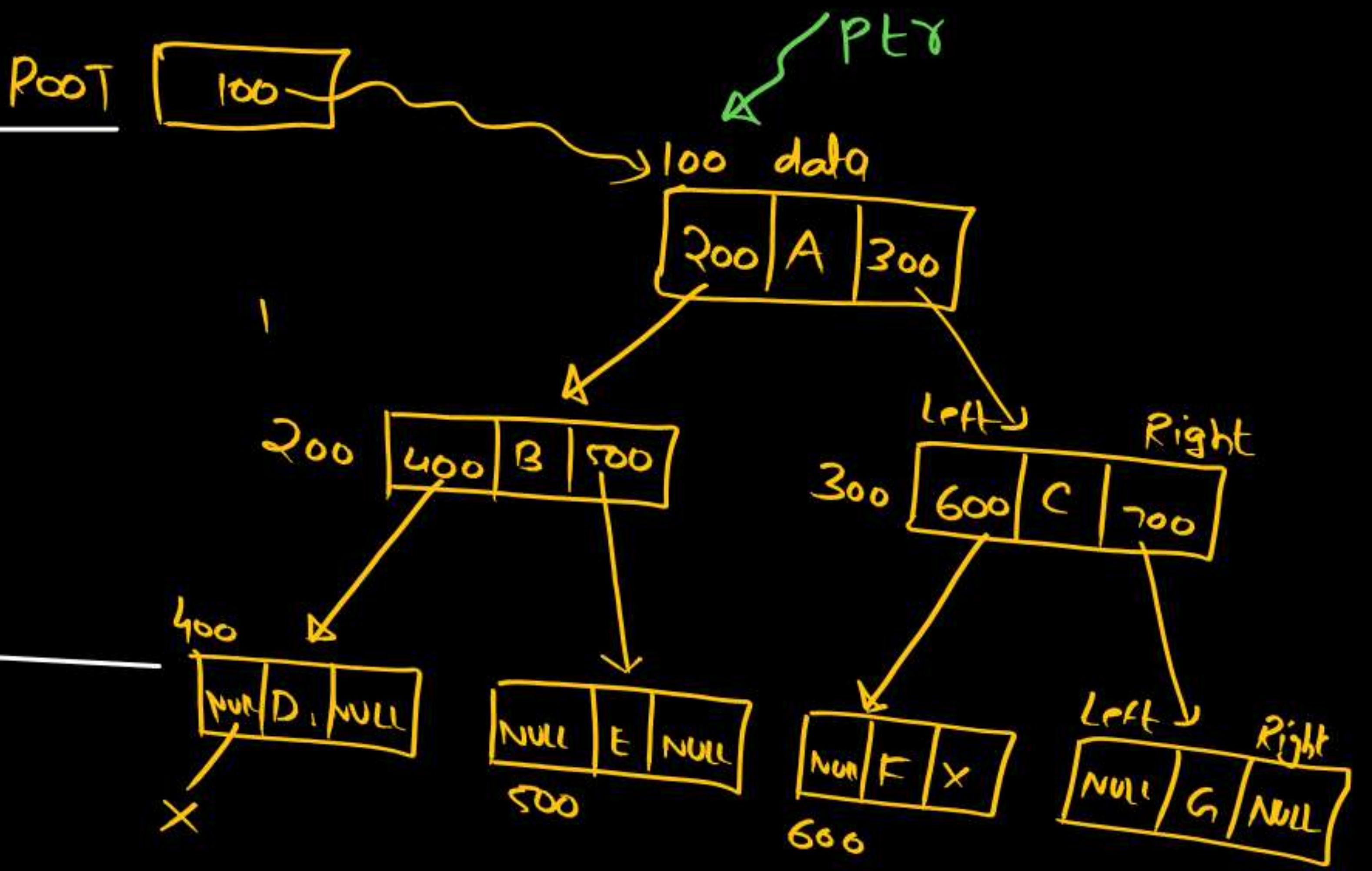
A B D



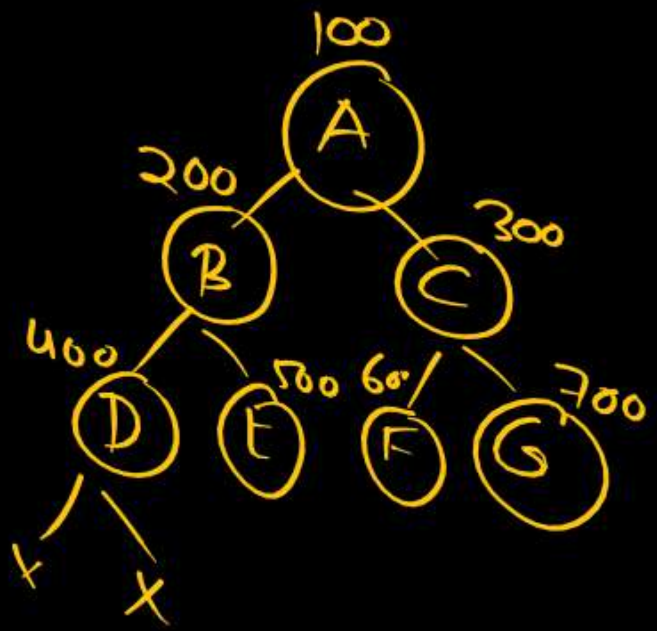


```
void main() {
1) Preorder( ROOT );
2.)
}
```

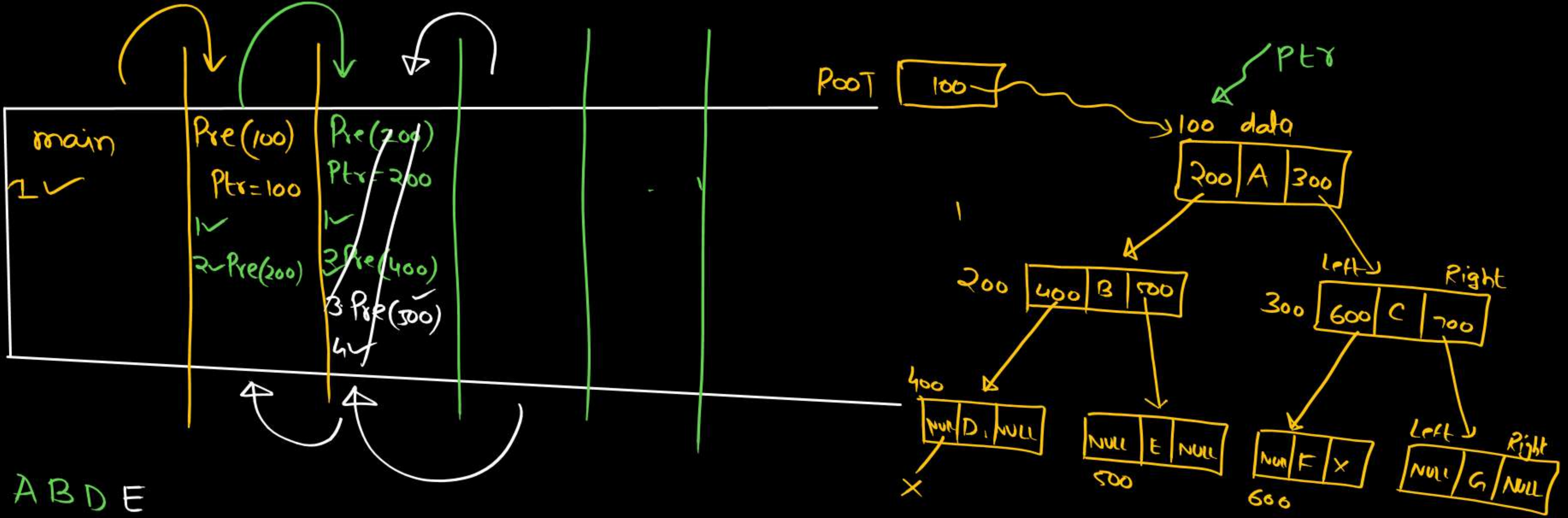




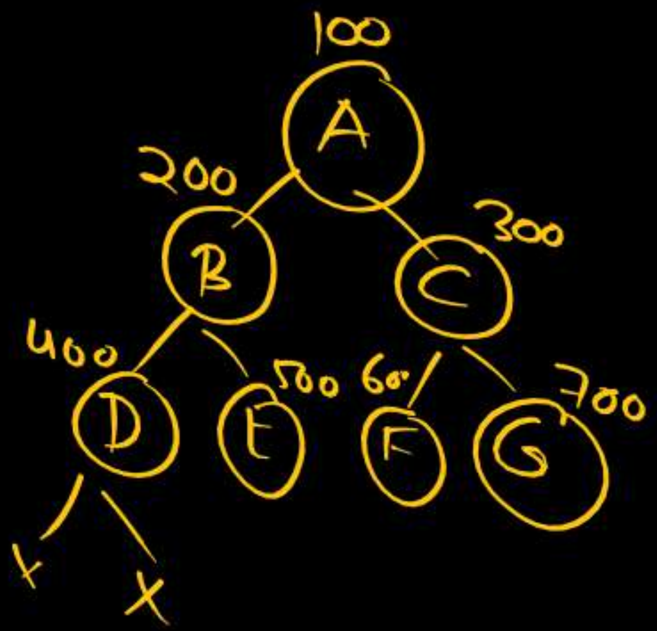
A B D E



```
void main() {

1.) Preorder( ROOT );

2.)

}
```



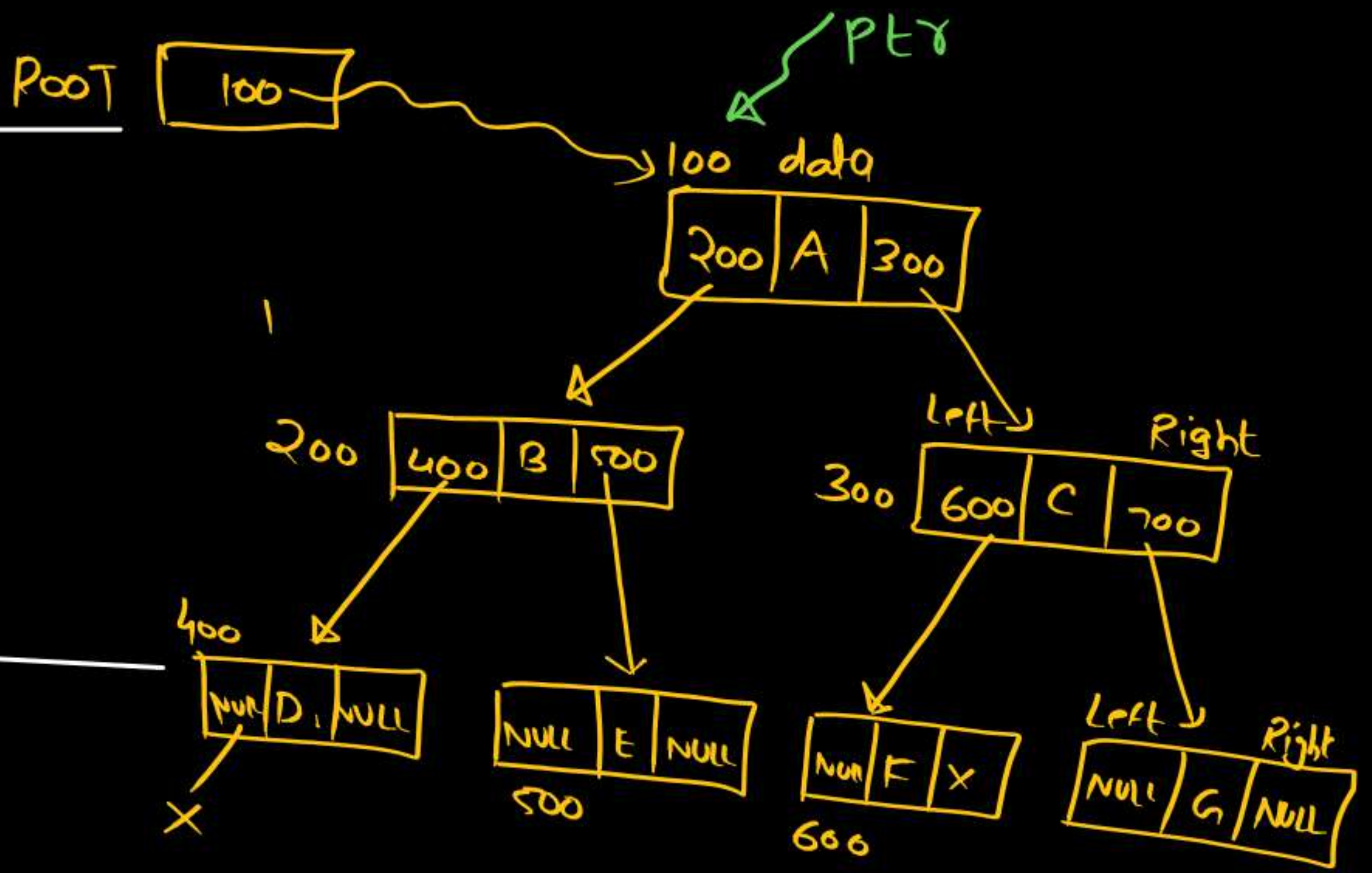
A B D E

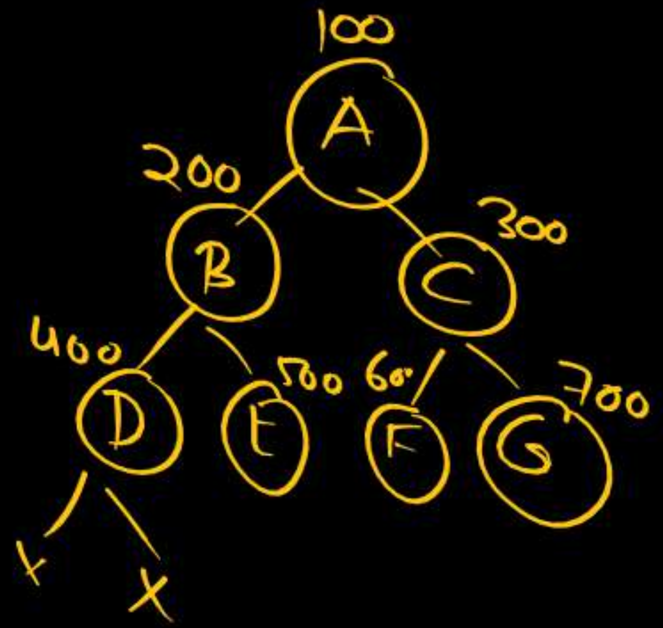


```
void main() {

1) Preorder( ROOT );

2.)

}
```

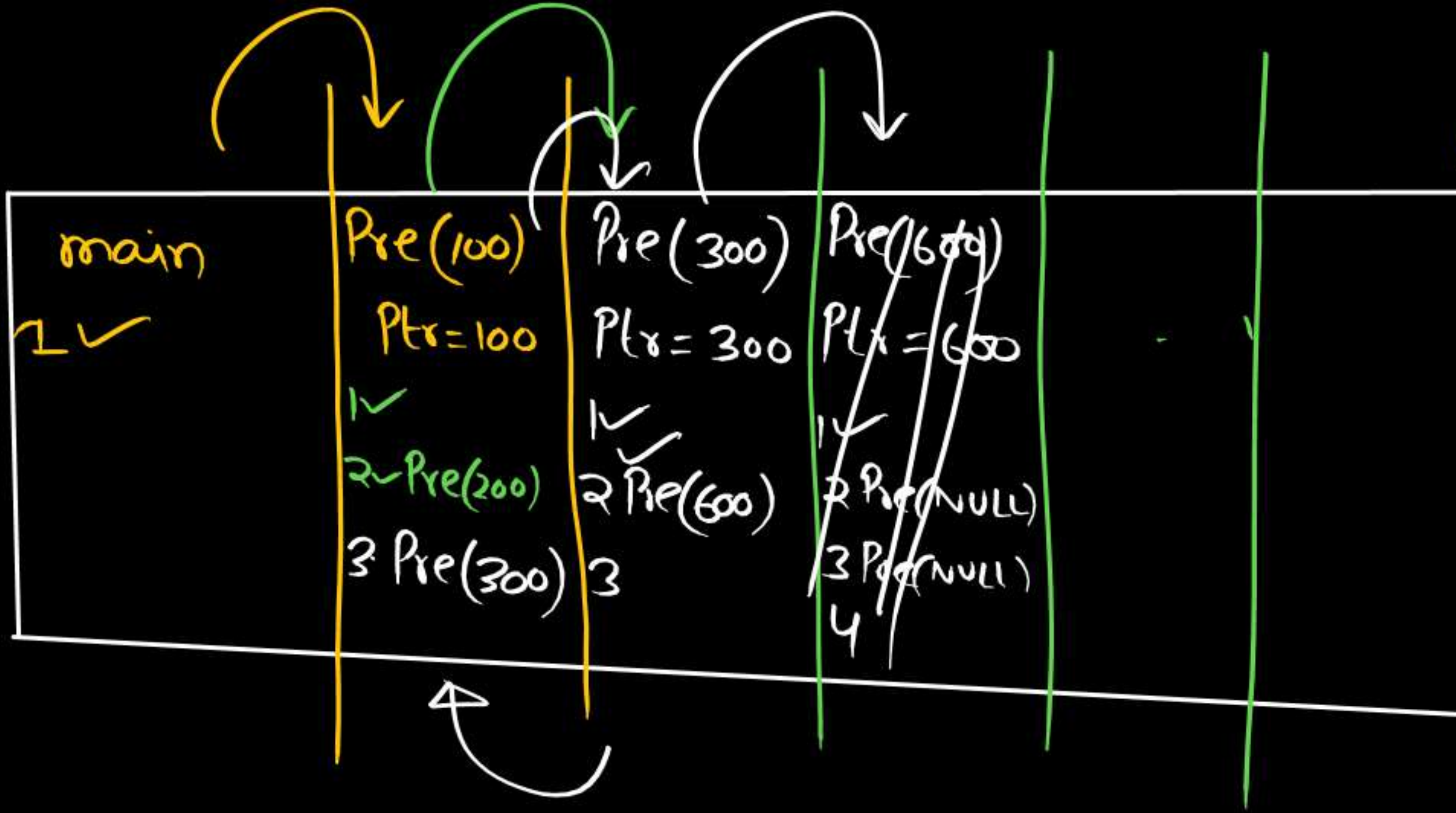
main	Pre(100)	Pre(300)	Pre(600)		
1 ✓	Ptr=100	Ptr=300	Ptr=600		
	1✓	1✓	1✓		
	2 Pre(200)	2 Pre(600)	2 Pre(NULL)		
	3 Pre(300)	3	3 Poe(NULL)		
			4		

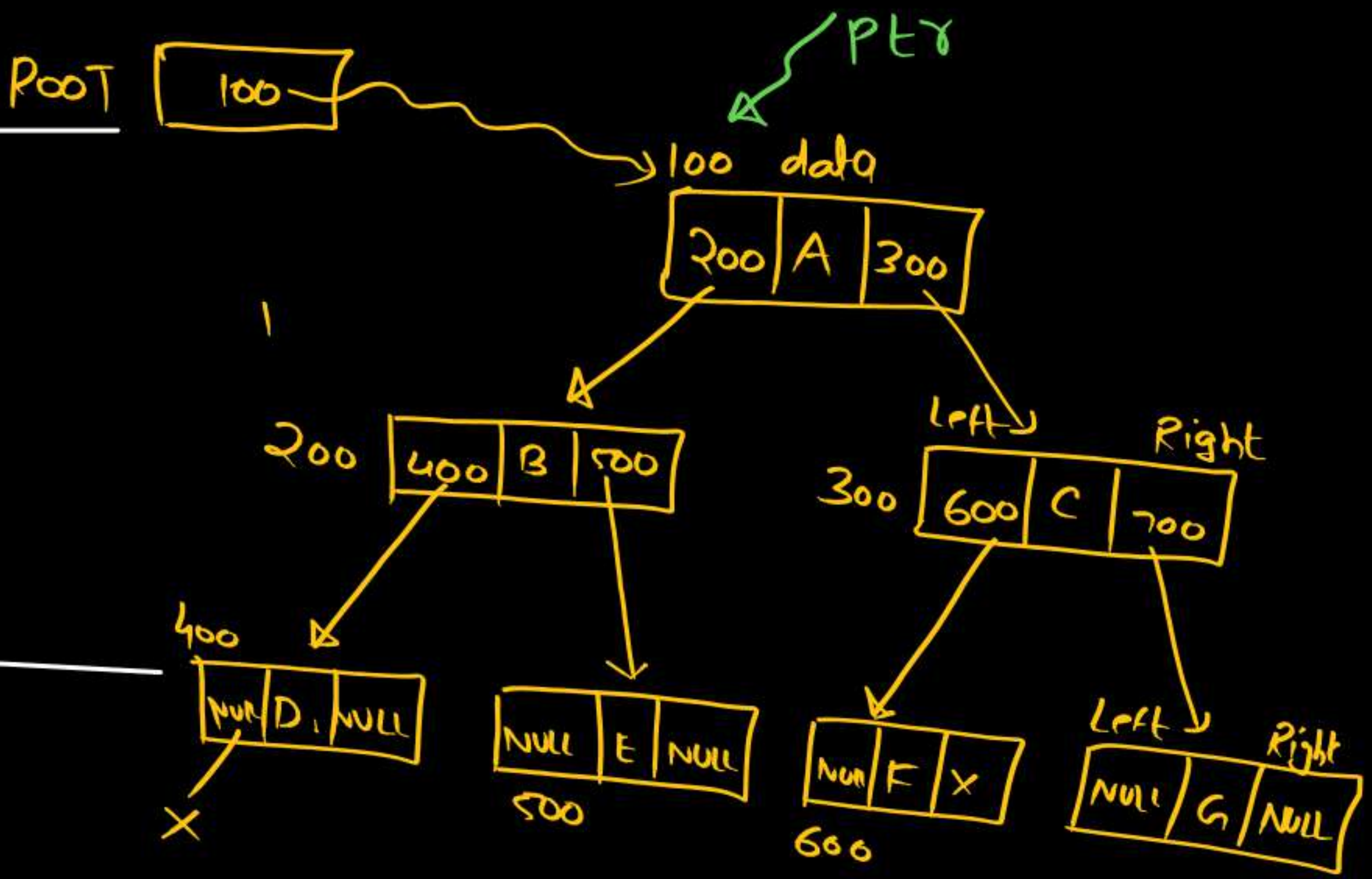
A B D E C F



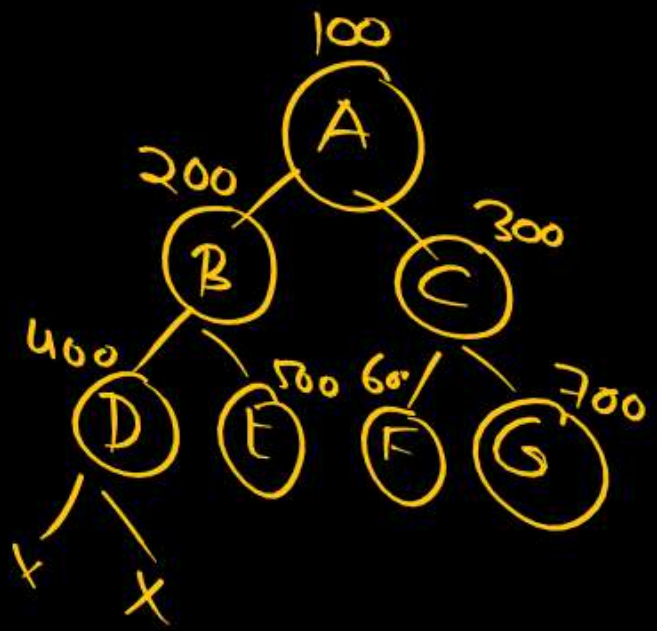


```
void main() {

1) Preorder( ROOT );

2)

}
```





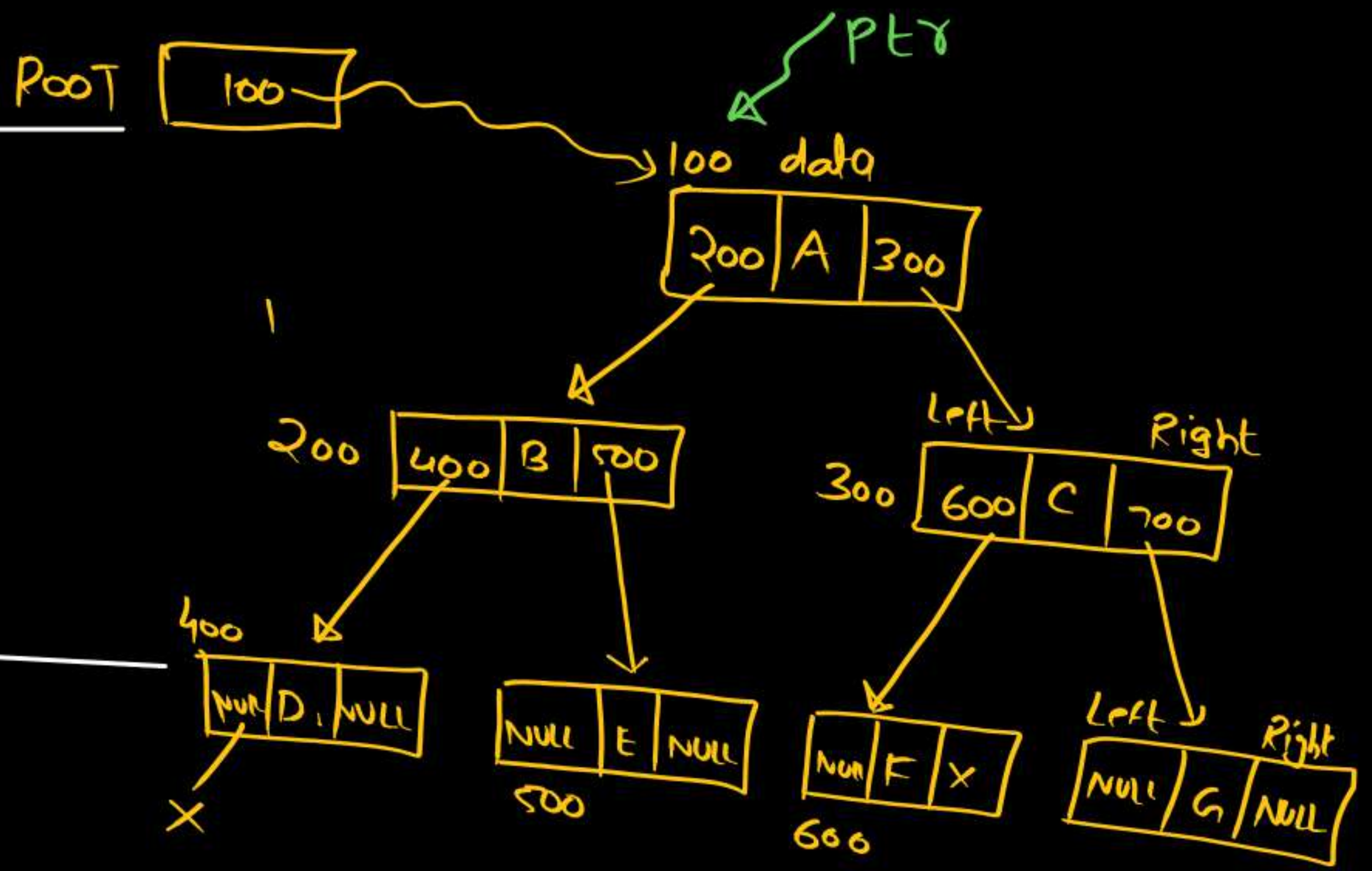
A B D E C F

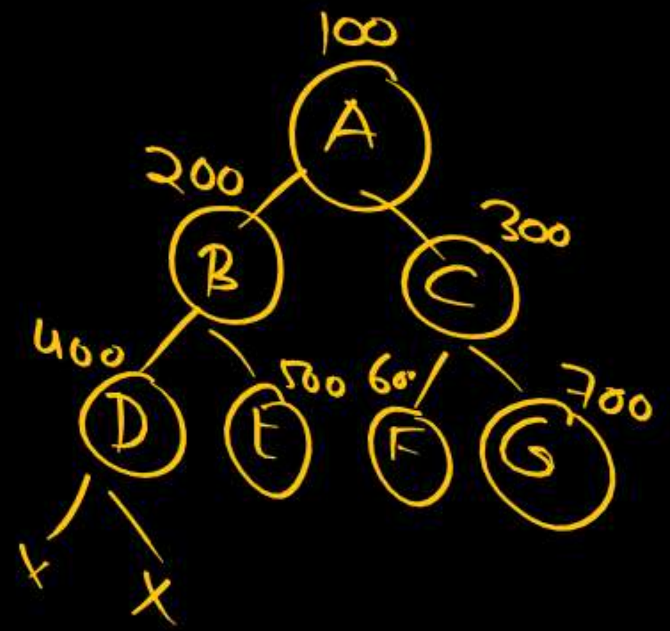


```
void main() {

1) Preorder( ROOT );

2.)

}
```

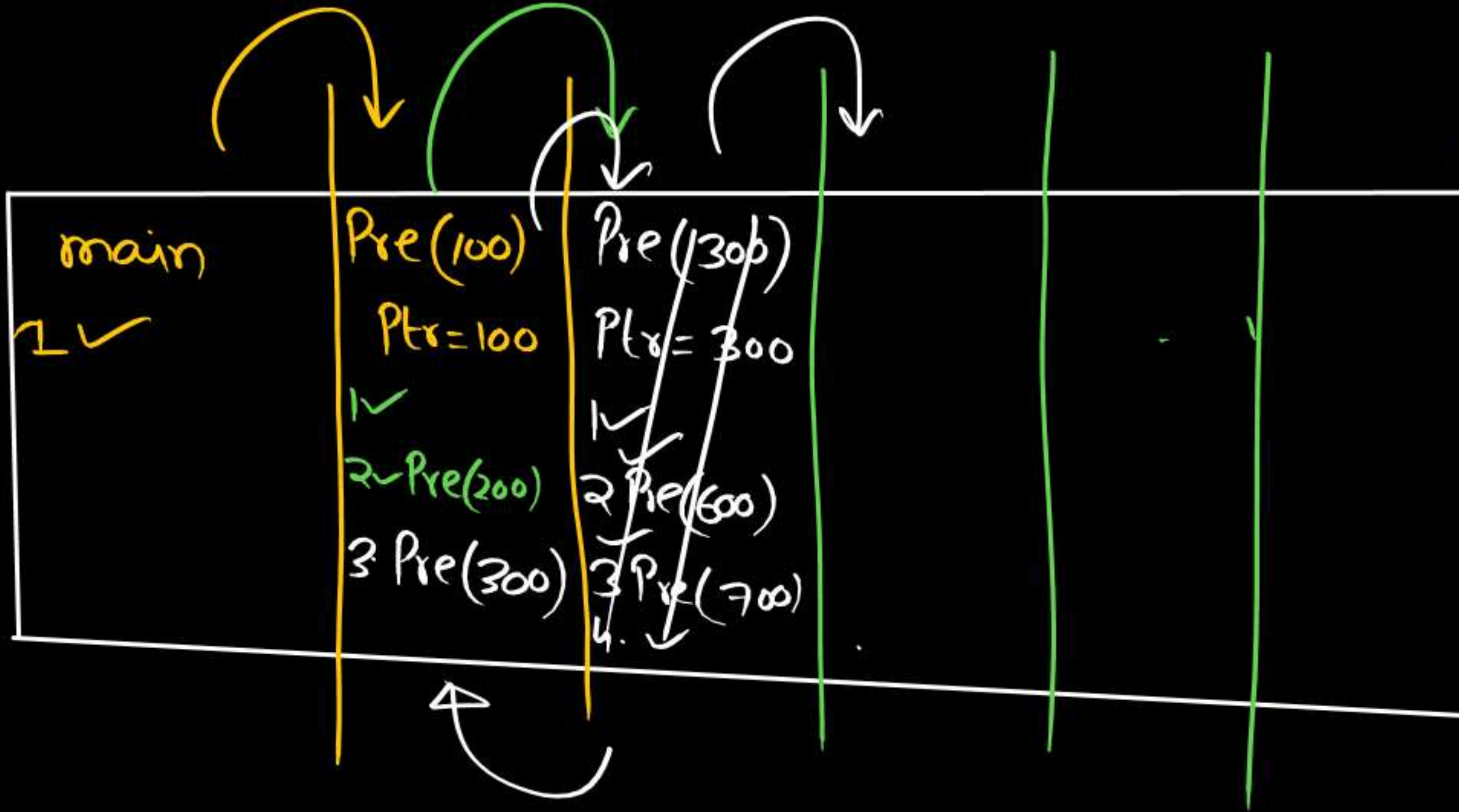
main	Pre(100)	Pre(300)	~~Pre(700)~~
	Ptr=100	Ptr = 300	~~Ptr = 700~~
1 ✓	1✓	1✓	~~1) ✓~~
	2✓ Pre(200)	2 Pre(600)	~~2. Pre(NULL)~~
	3 Pre(300)	3 Pre(700)	~~3. Pre(NULL)~~
			~~4.~~

A B D E C F G

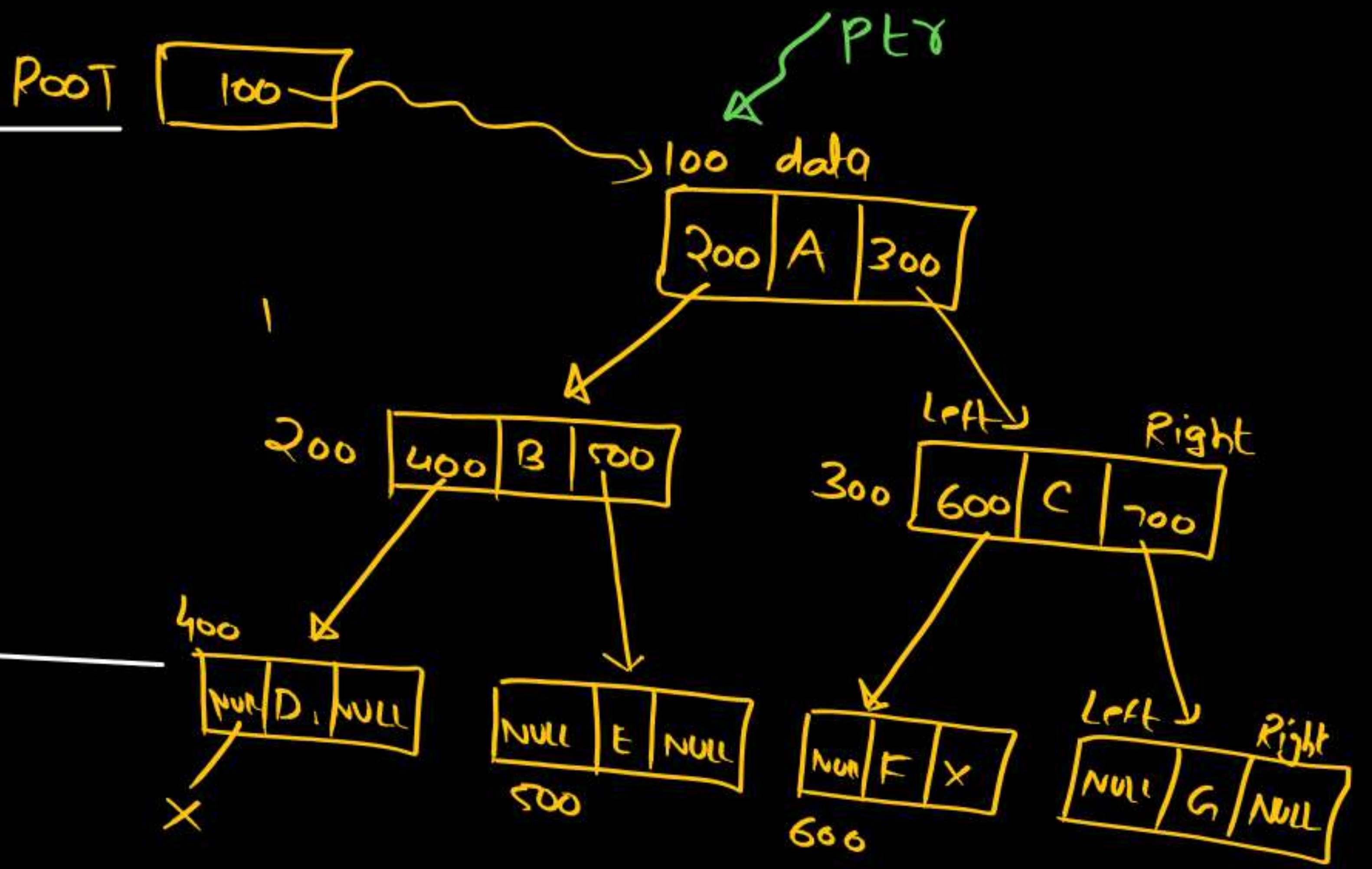


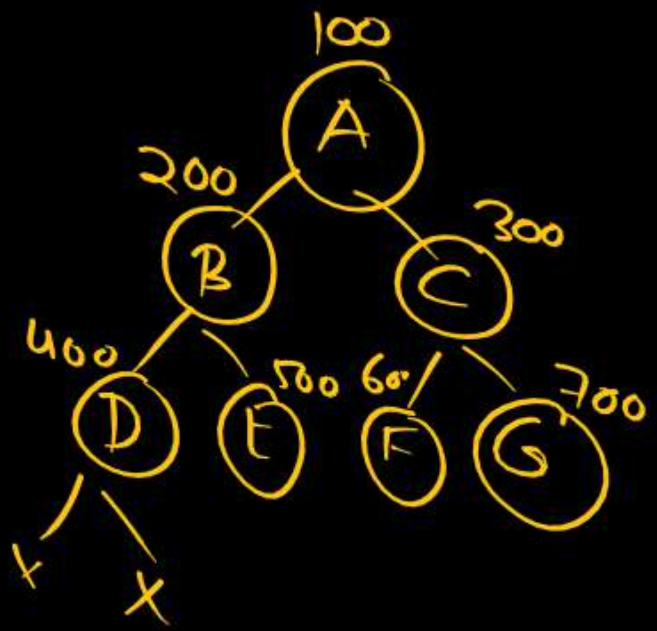


```
void main() {

1.) Preorder( ROOT );

2.)

}
```

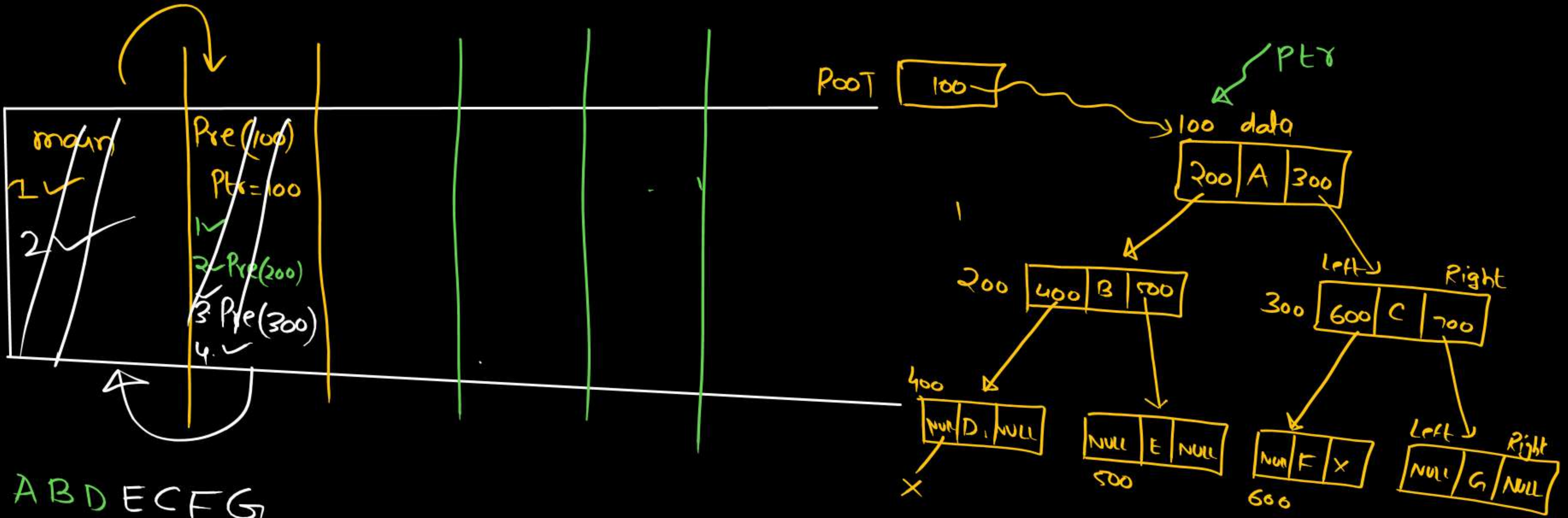


A B D E C F G

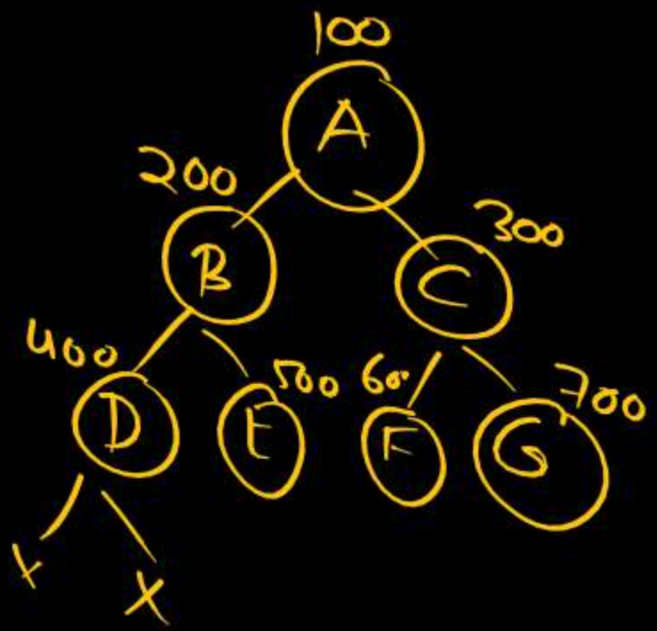




```
void main() {

1) Preorder( ROOT );

2.)

}
```

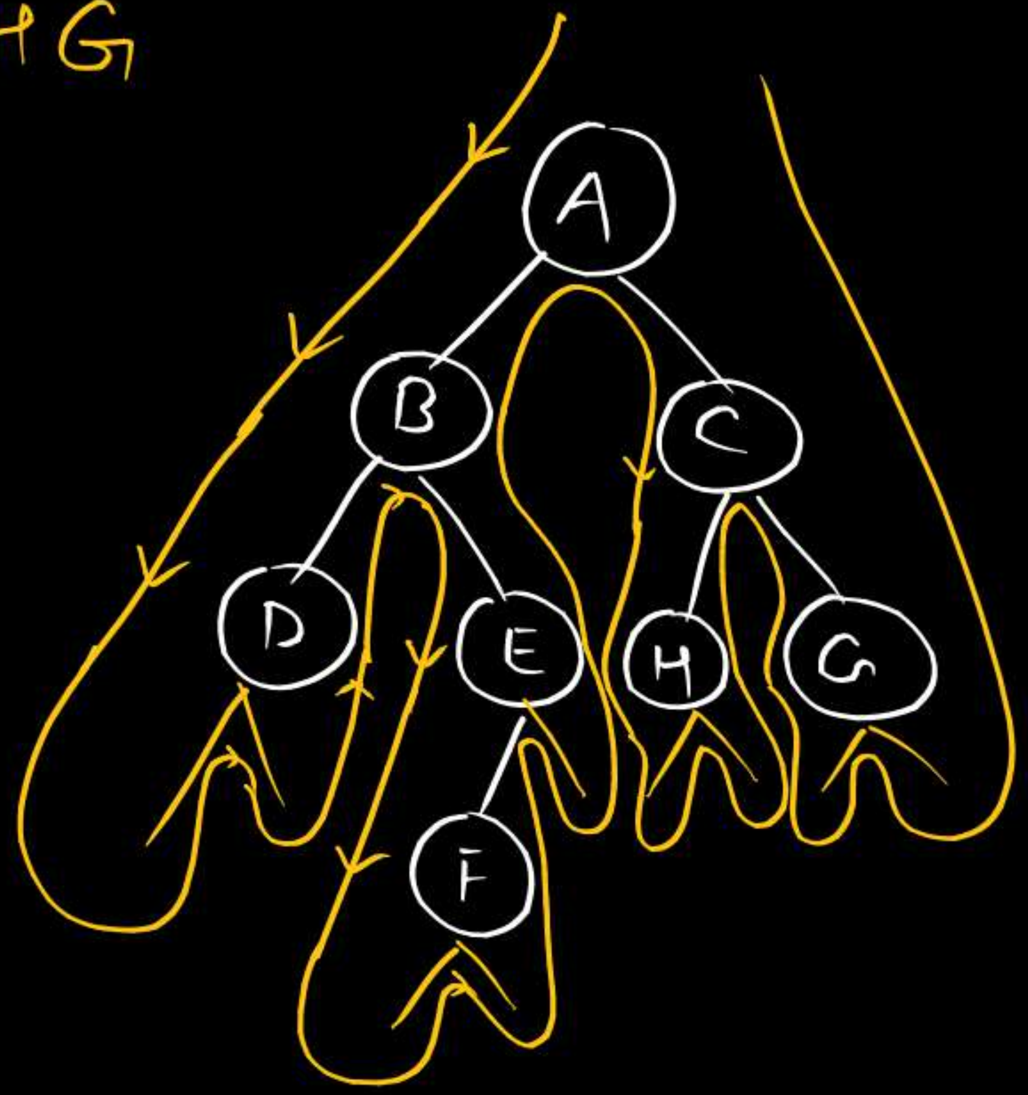


ABDECFG

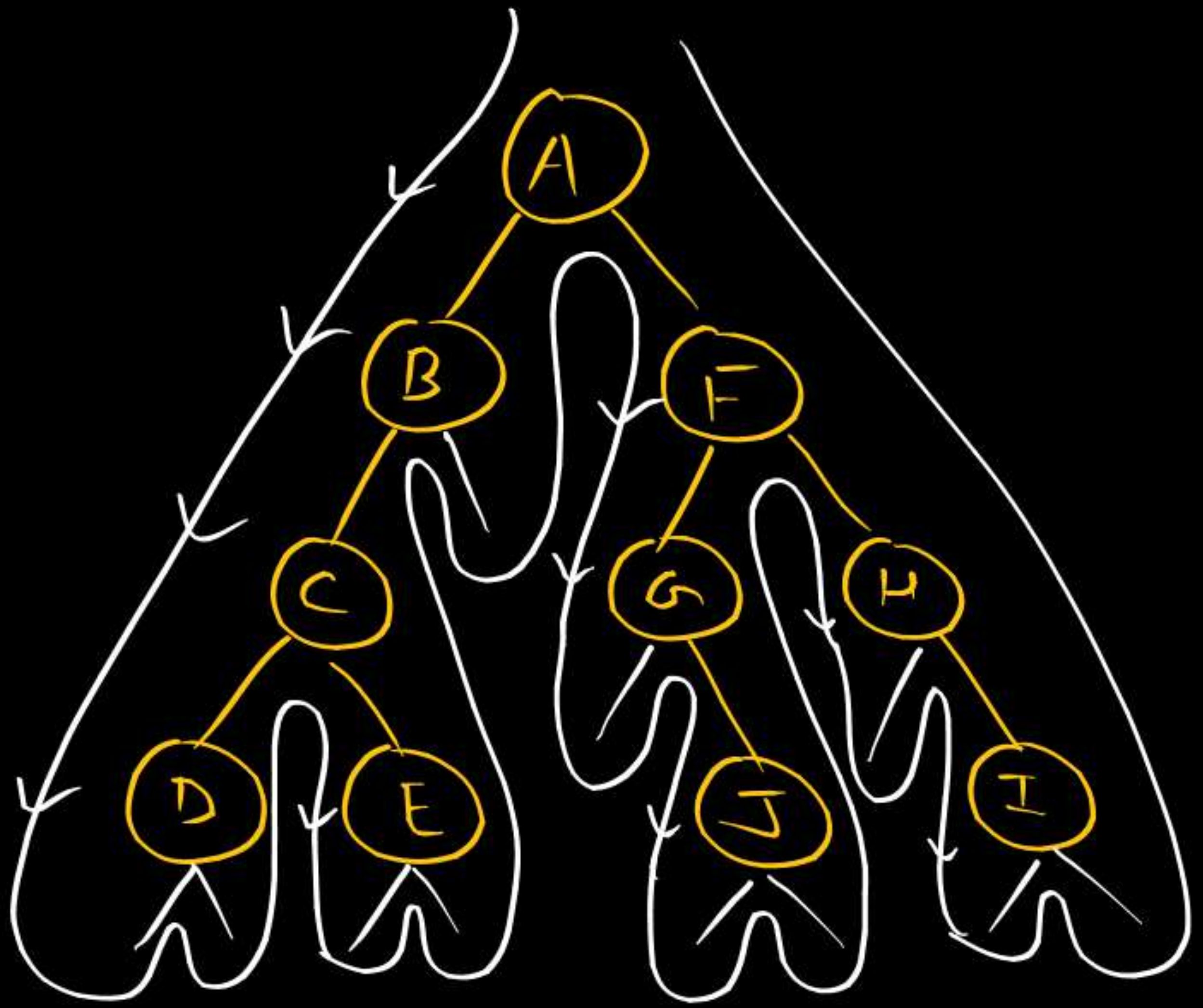


```
void main(){

1) Preorder( ROOT );

2.)

}
```

ABDEFCHG



Preorder :



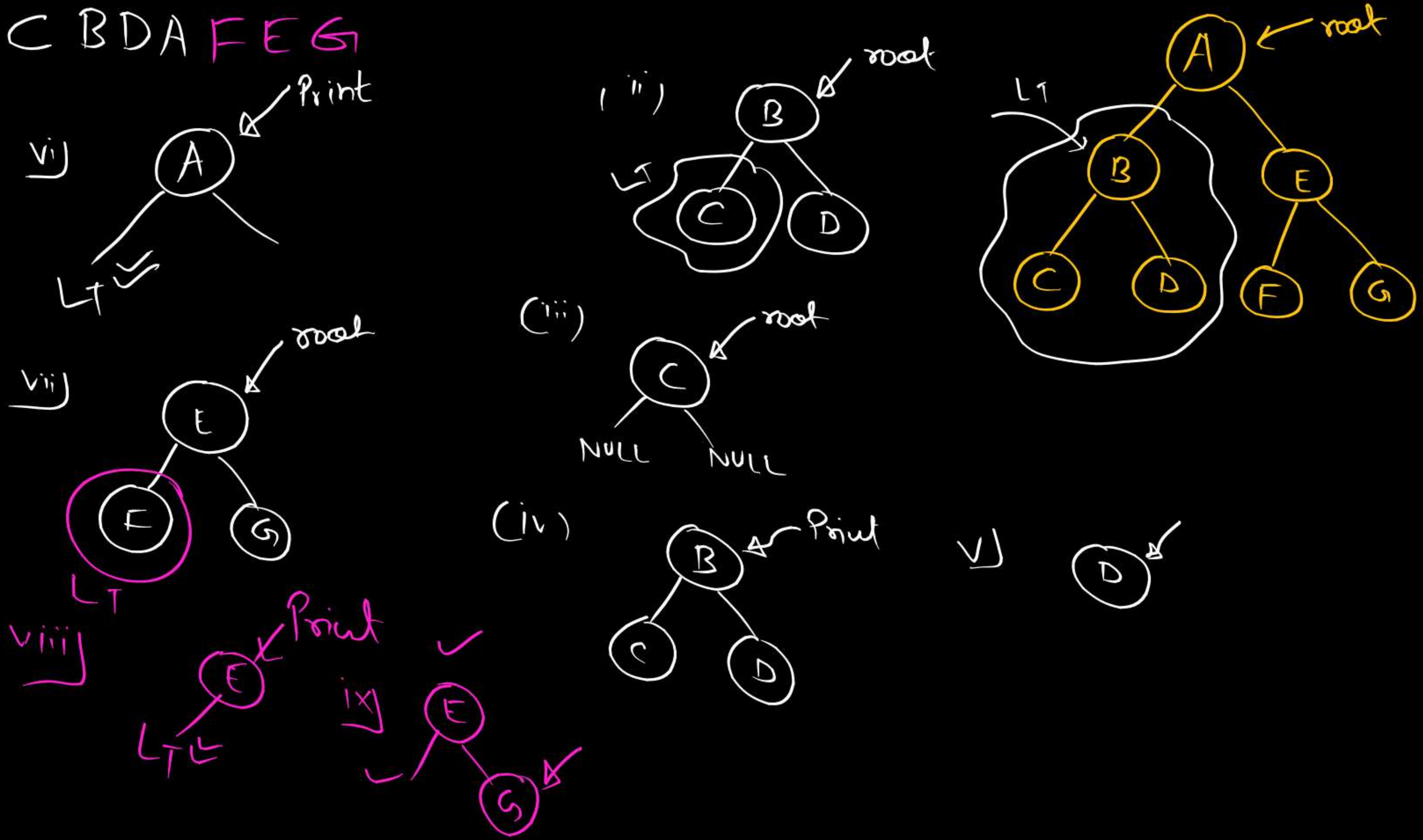
ABCDEFGJHI

Inorder Traversal

1.) Traverse L_T of root node in Inorder.

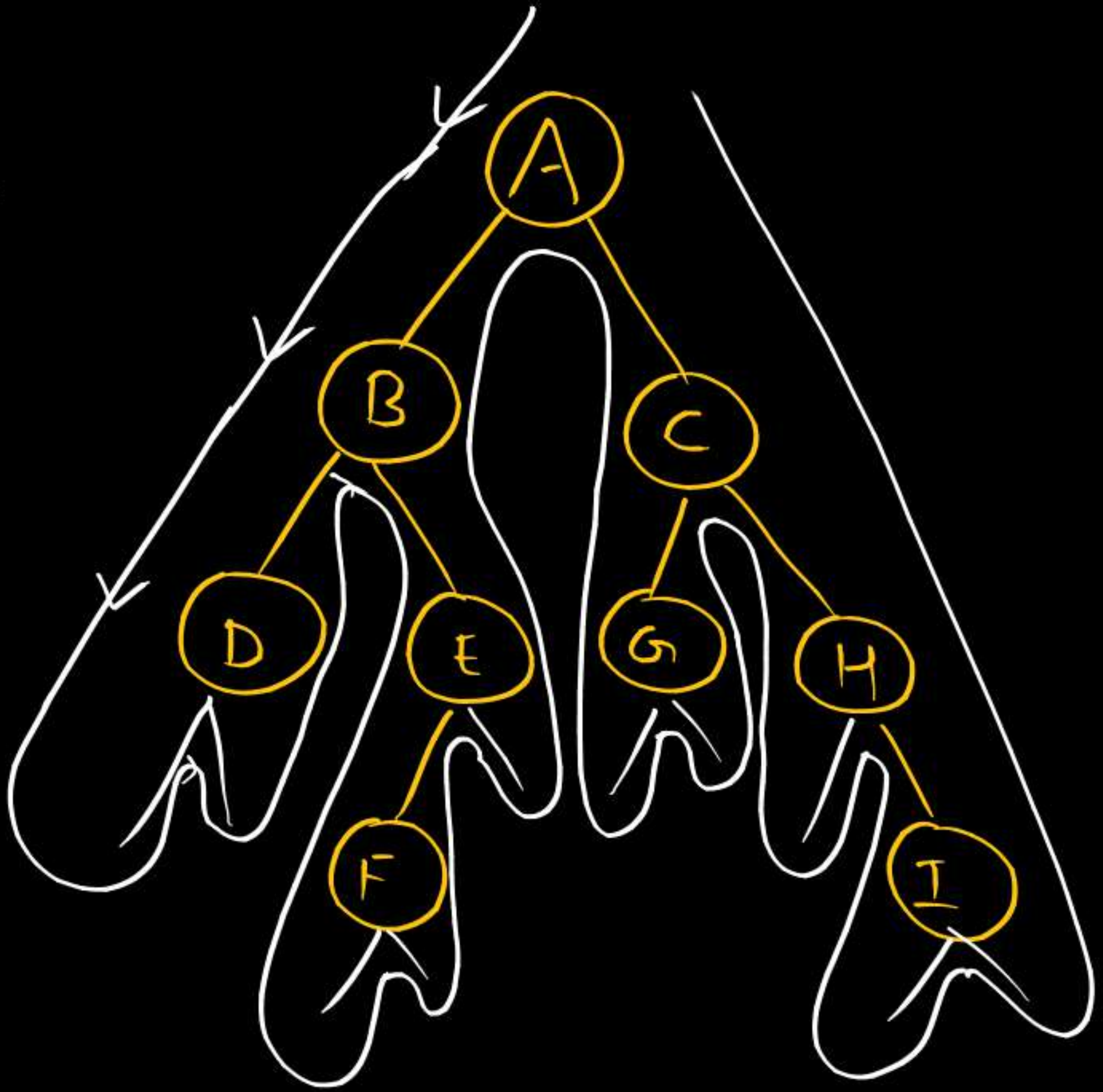
2.) Print/visit/Process root node.

3.) Traverse R_T of root node in Inorder.

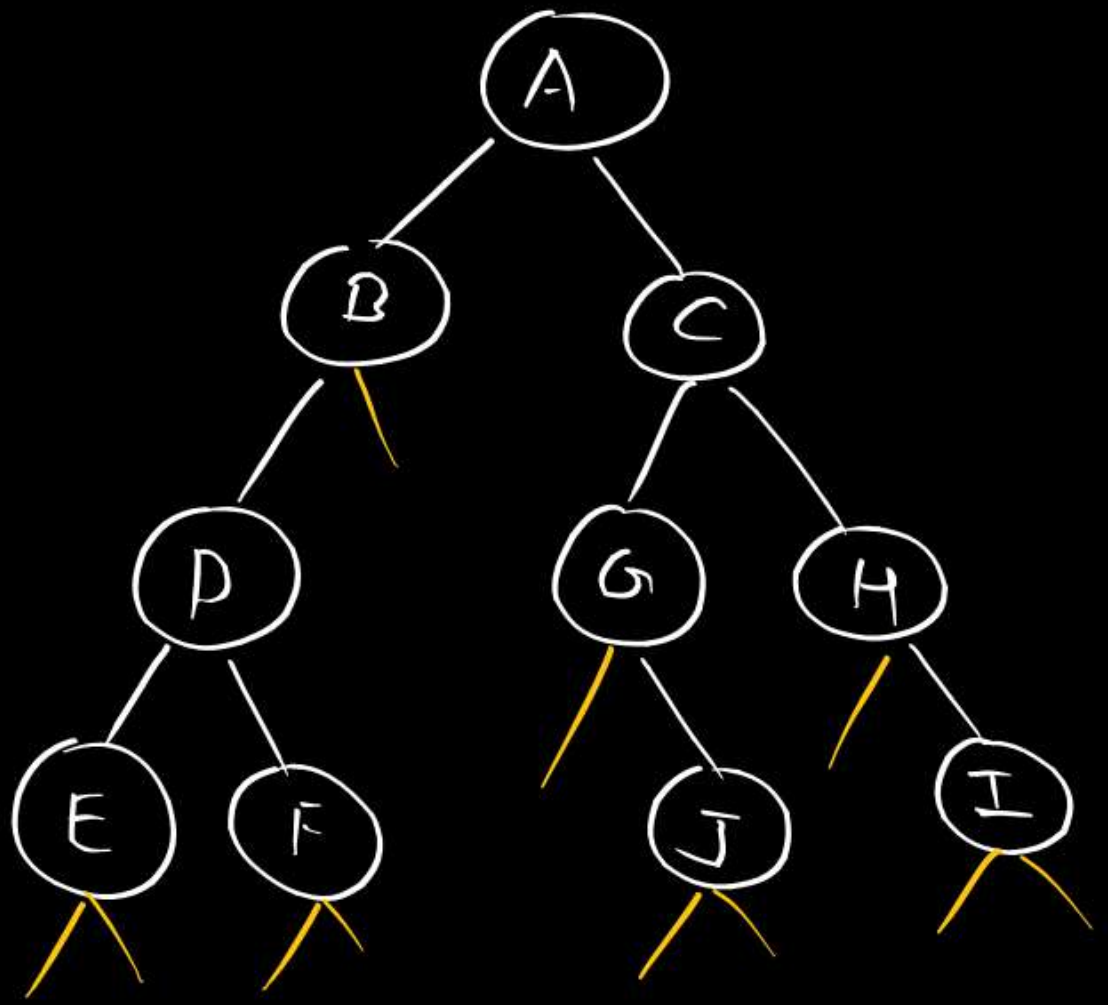

C B D A F E G
Print
vi)
A
LT
vii)
root
E
F
G
LT
viii)
Print
E
LT
ix)
E
G
(ii)
root
B
LT
C
D
(iii)
root
C
NULL
NULL
(iv)
Print
B
C
D
v)
D
root
LT
A
B
E
C
D
F
G

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


    Inorder( Ptr → Left);

    printf("%d", Ptr → data);
    Inorder( Ptr → Right);


  }
```

DBFEAGCHI



E D F B A G J C H I



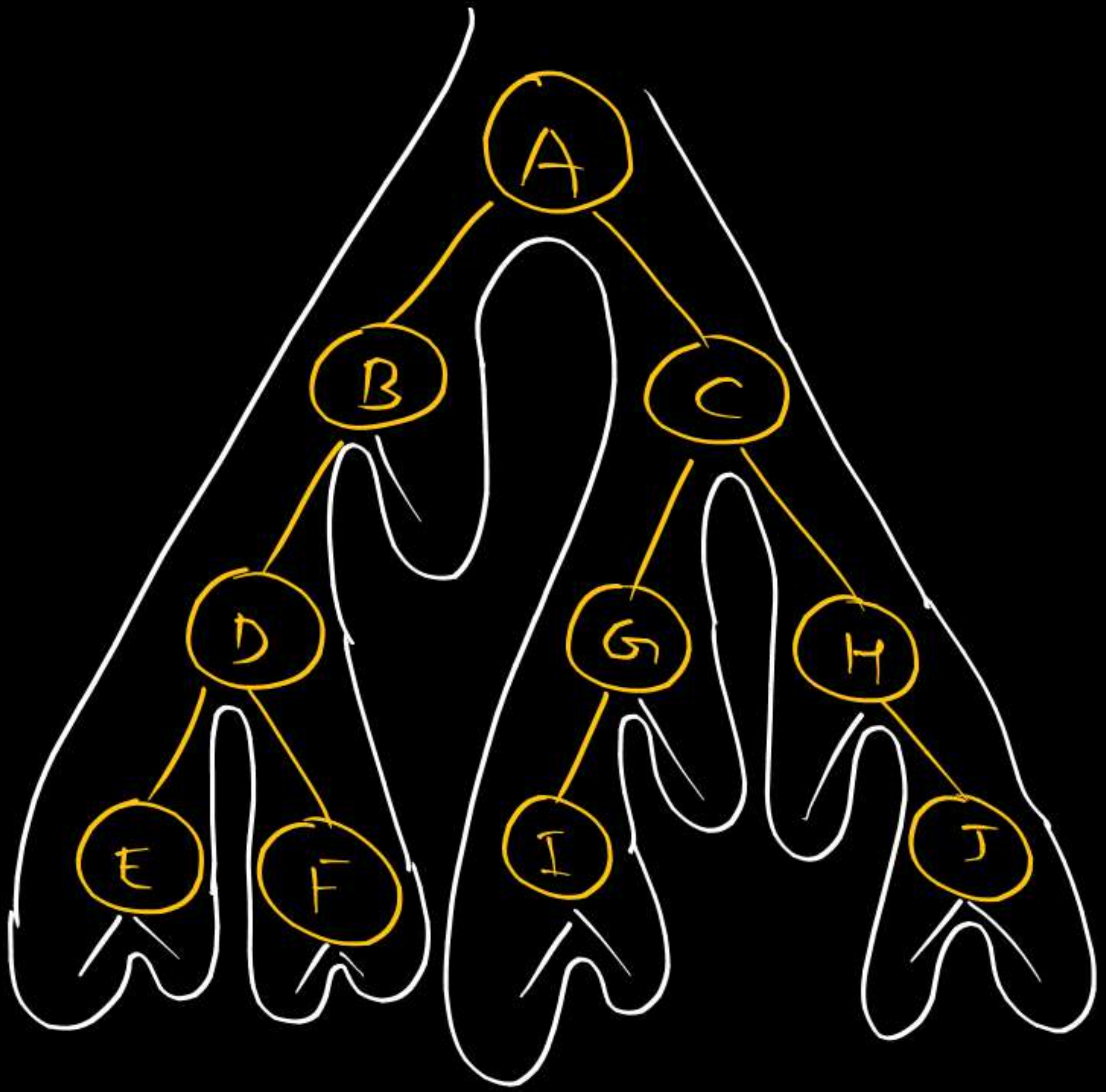
Post-order Traversal

1. Traverse L_T of root in Postorder.

2. Traverse R_T of root in Postorder.

3.) visit/Print root node

```
void Postorder(struct Node *Ptr)
  {
        if(Ptr == NULL)
              return;
Postorder(Ptr → Left);
Postorder(Ptr → Right);
printf("%d", Ptr → data);

  }
```

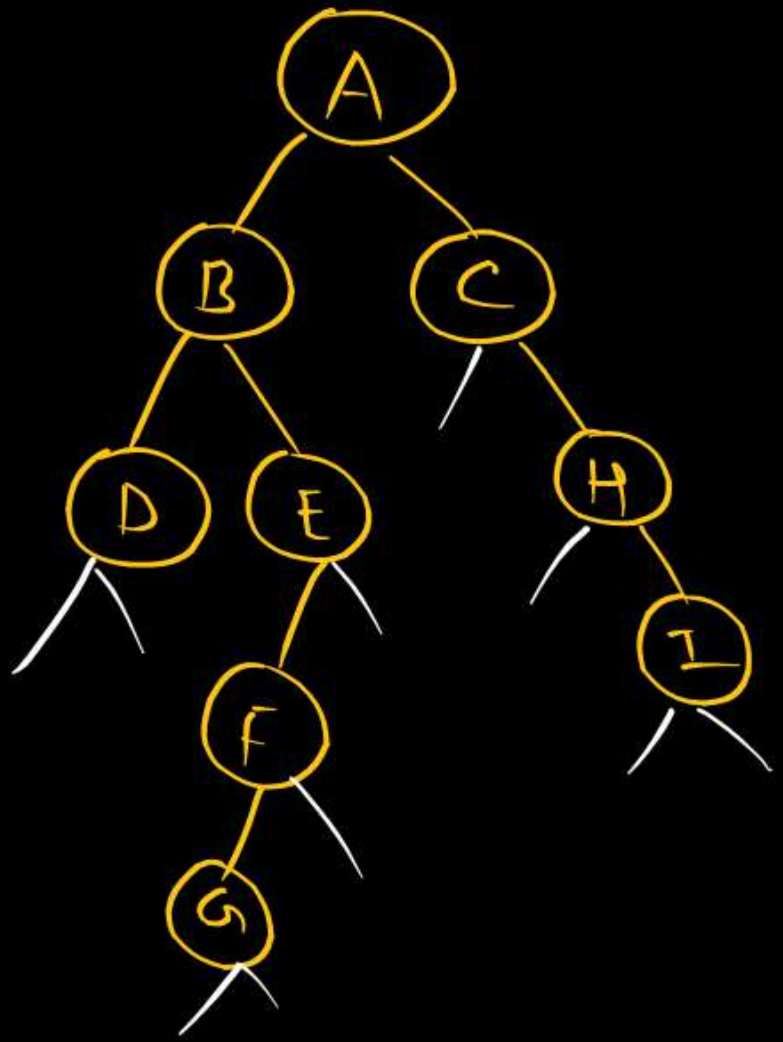


EFDBIGJHCA

Traversal ✓

DG FEBIHCA

✓



THANK - YOU