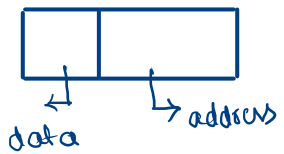
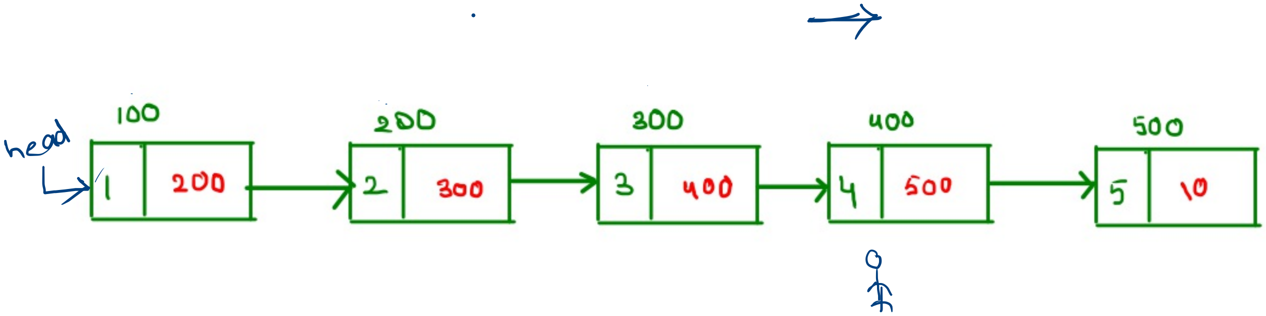
Trees

SLL [Singly Linked List]

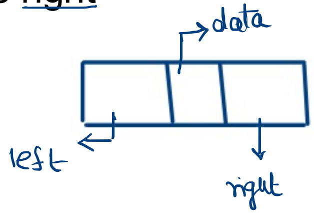
```
class Node
{
        data
        Node next
}
```

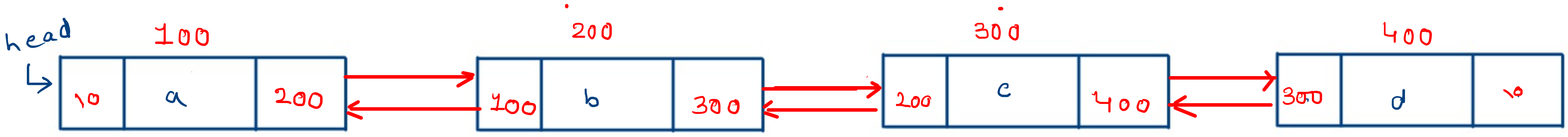




DLL [Doubly Linked List]

```
class Node
{
        data✓
        Node left
        Node right
}
```



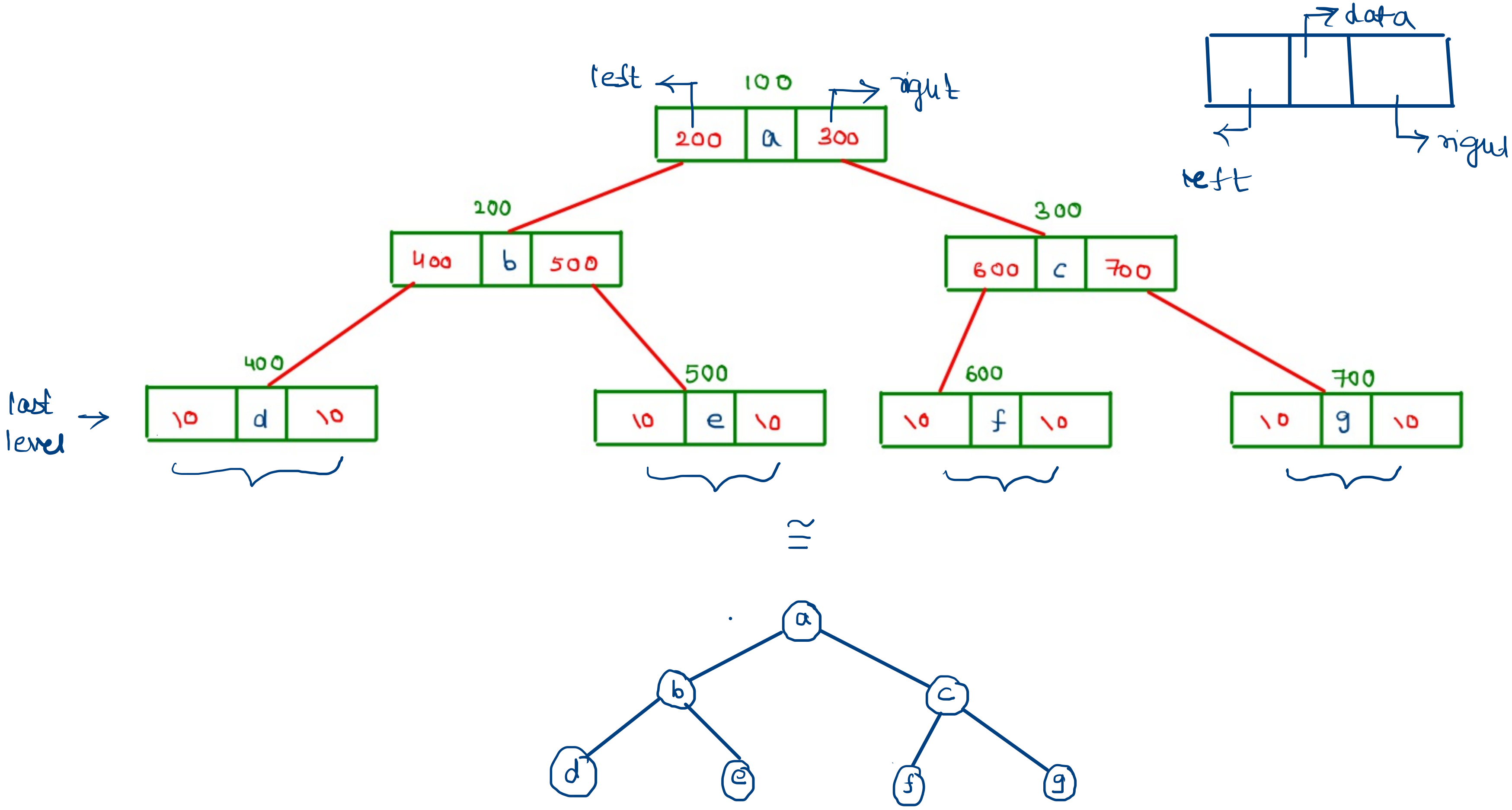


Trees

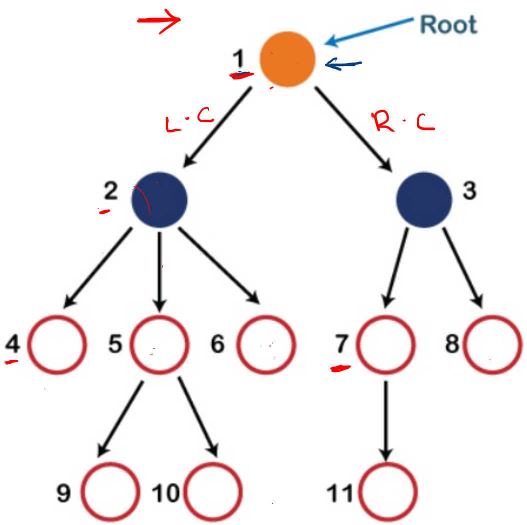
SLL
DLL } Linear DS

Trees → Non-linear DS

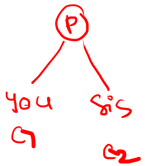
```
class Node
{
→  data
→  Node left
→  Node right
}
```


data
left
right
left
100
right
200
a
300
200
400
b
500
300
600
c
700
400
\0
d
\0
500
\0
e
\0
600
\0
f
\0
700
\0
g
\0
last level →
≅
a
b
c
d
e
f
g

Ternary Tree



Binary tree → [node, contains at most 2 childrens]



a) Root Node → 1

b) Child Node → (5) : 9, 10

c) Parent Node → (7, 8) : 3

d) Siblings → (4) : 5, 6

e) Leaf Node → node having 0 children : 4, 6, 8, 9, 10, 11

f) Internal Node → a node with at least 1 children

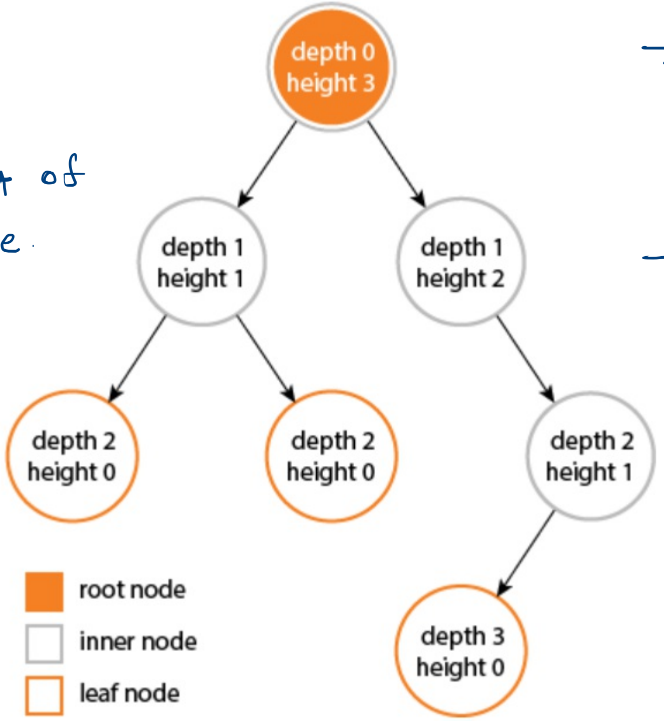
g) Ancestor Node → predecessor : (9) : 5, 2, 1

h) Descendant → successors : (3) : 7, 8, 11

total nodes = Internal nodes + leaf nodes

Depth v/s height :-

① Heigut of root = Heigut of tree.



→ level 0

→ level 1

→ level 2

→ level 3

2 3

Traversal

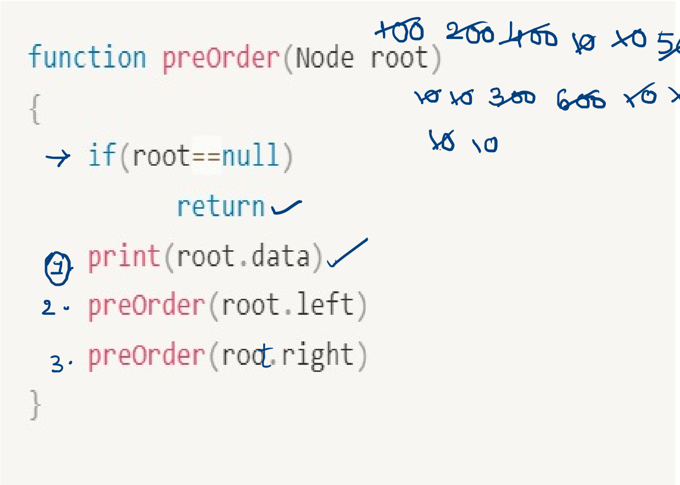
→ ① pre-order

② In-order

③ post-order

④ level-order

Pre-Order

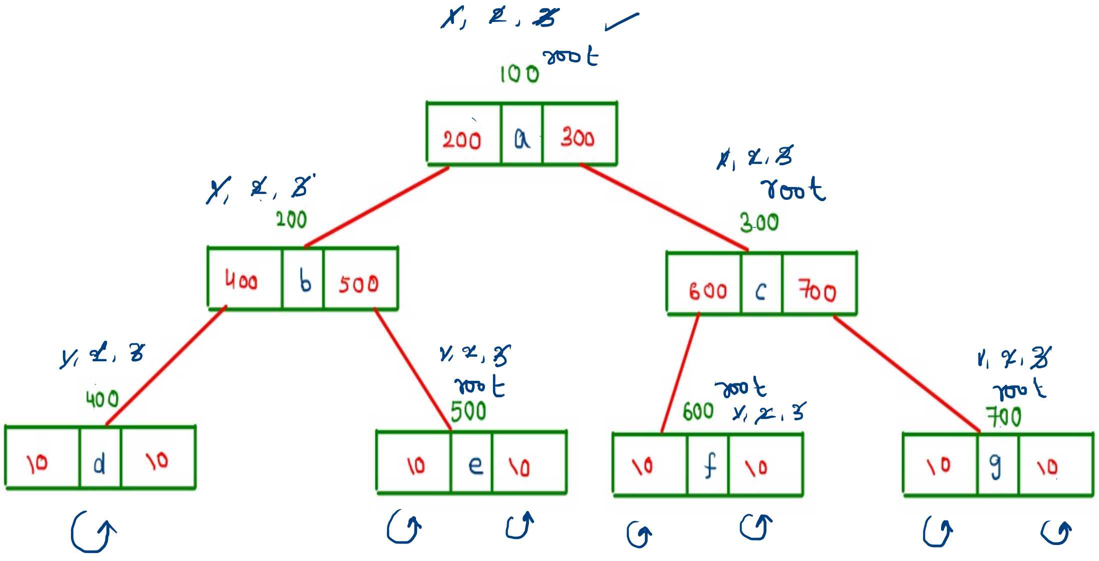


```
function preOrder(Node root)
{
 → if(root==null)
          return ✓
 1. print(root.data) ✓
 2. preOrder(root.left)
 3. preOrder(root.right)
}
```

~~100~~ ~~200~~ ~~400~~ ~~\0~~ ~~\0~~ ~~500~~

~~\0~~ ~~\0~~ ~~300~~ ~~600~~ ~~\0~~ ~~\0~~ ~~700~~

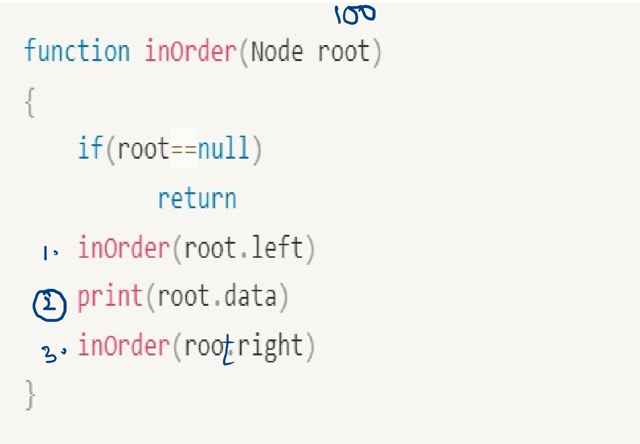
~~\0~~ \0



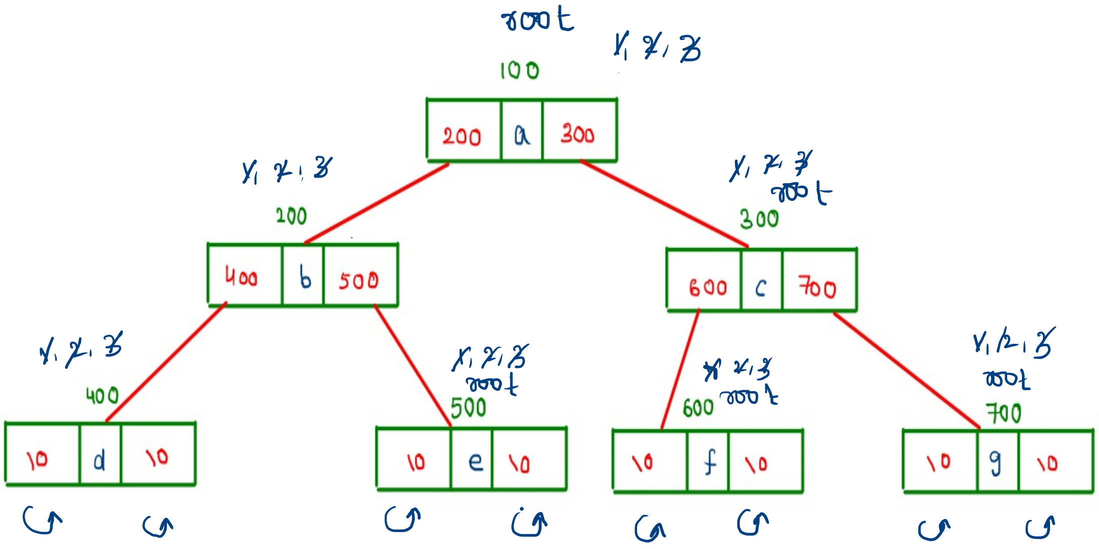
o/p

⇒ a, b, d, e, c, f, g, (pre-order)

In-Order



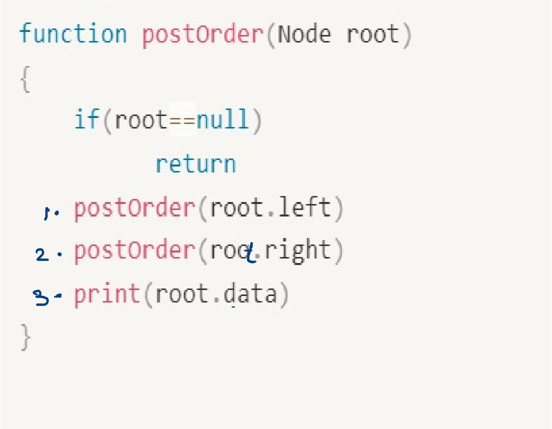
```
                                  100
function inOrder(Node root)
{
    if(root==null)
          return
 1. inOrder(root.left)
 ② print(root.data)
 3. inOrder(root.right)
}
```



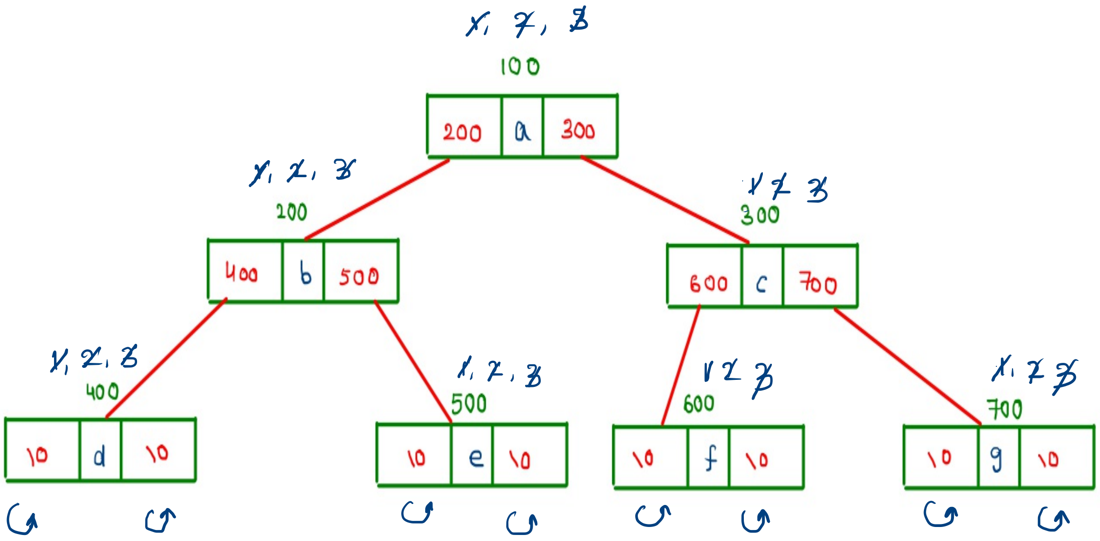
o/p

d, b, e, a, f, c, g, [In-order]

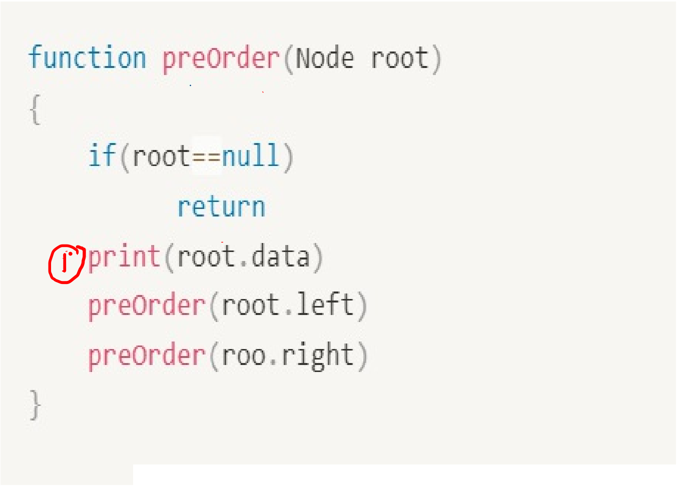
post order



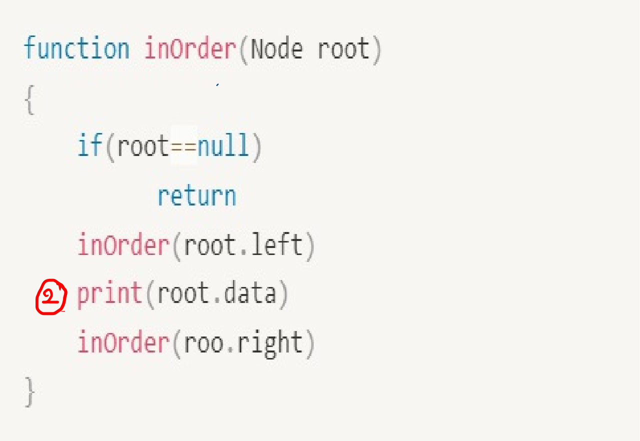
```
function postOrder(Node root)
{
    if(root==null)
          return
 1. postOrder(root.left)
 2. postOrder(root.right)
 3. print(root.data)
}
```



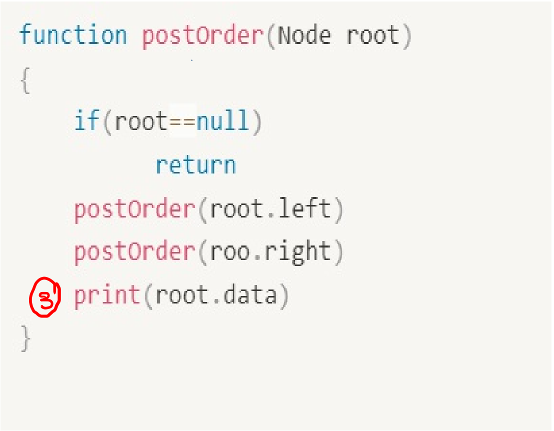
o/p :- d, e, b, f, g, c, a [post-order]



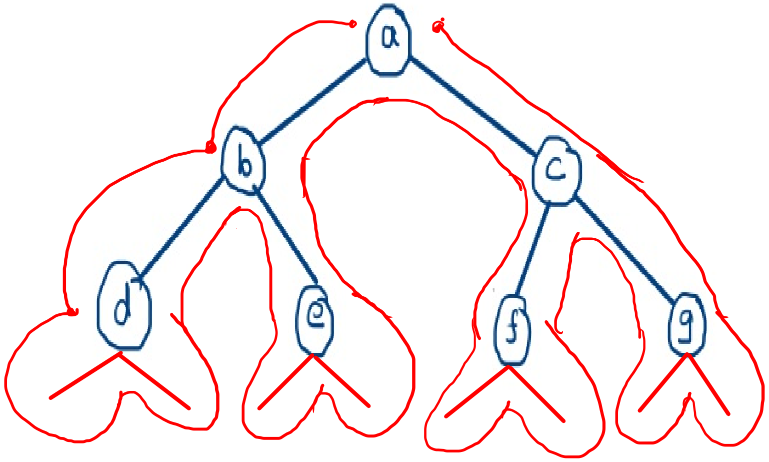
```
function preOrder(Node root)
{
    if(root==null)
          return
 ①print(root.data)
    preOrder(root.left)
    preOrder(roo.right)
}
```



```
function inOrder(Node root)
{
    if(root==null)
          return
    inOrder(root.left)
 ② print(root.data)
    inOrder(roo.right)
}
```



```
function postOrder(Node root)
{
    if(root==null)
          return
    postOrder(root.left)
    postOrder(roo.right)
 ③ print(root.data)
}
```



① Pre - order : a b d e c f g

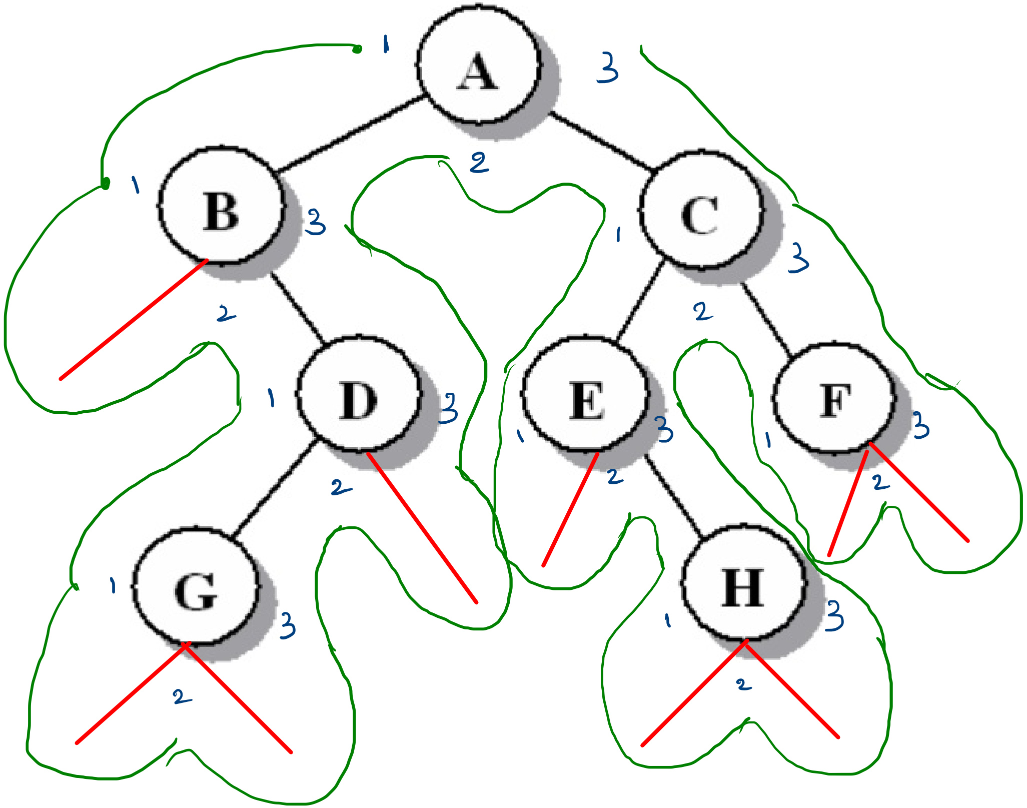
② In - order : d b e a f c g

③ Post - order : d e b f g c a

① Pre : A B D G C E H F

② In : B G D A E H C F

③ post : G D B H E F C A

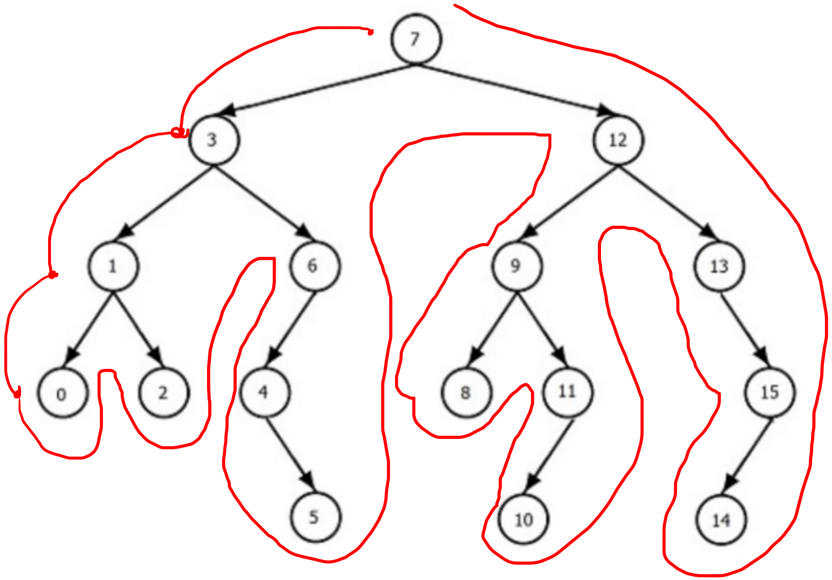


① pre-order:- [entry style]

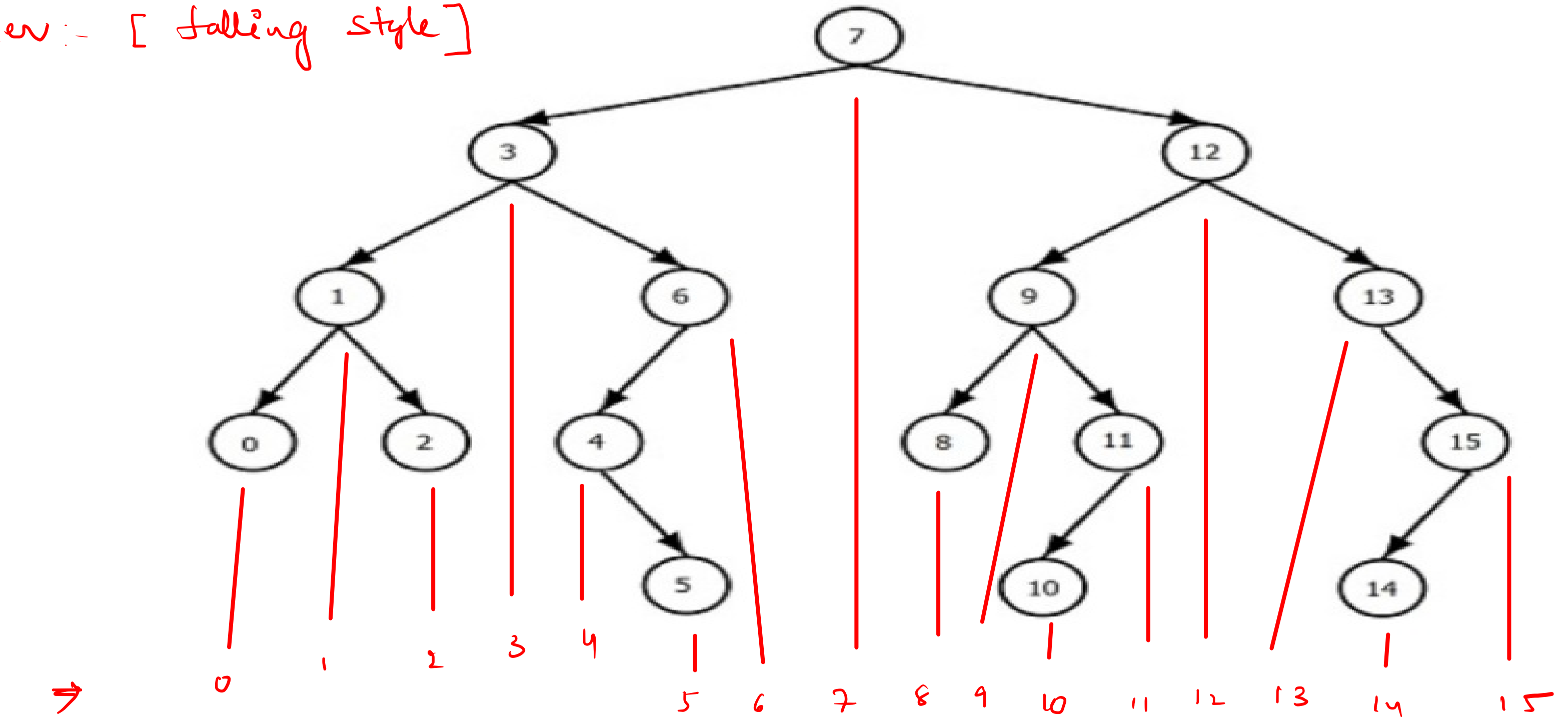
7 3 1 0 2 6 4

5 12 9 8 11 10

13 15 14



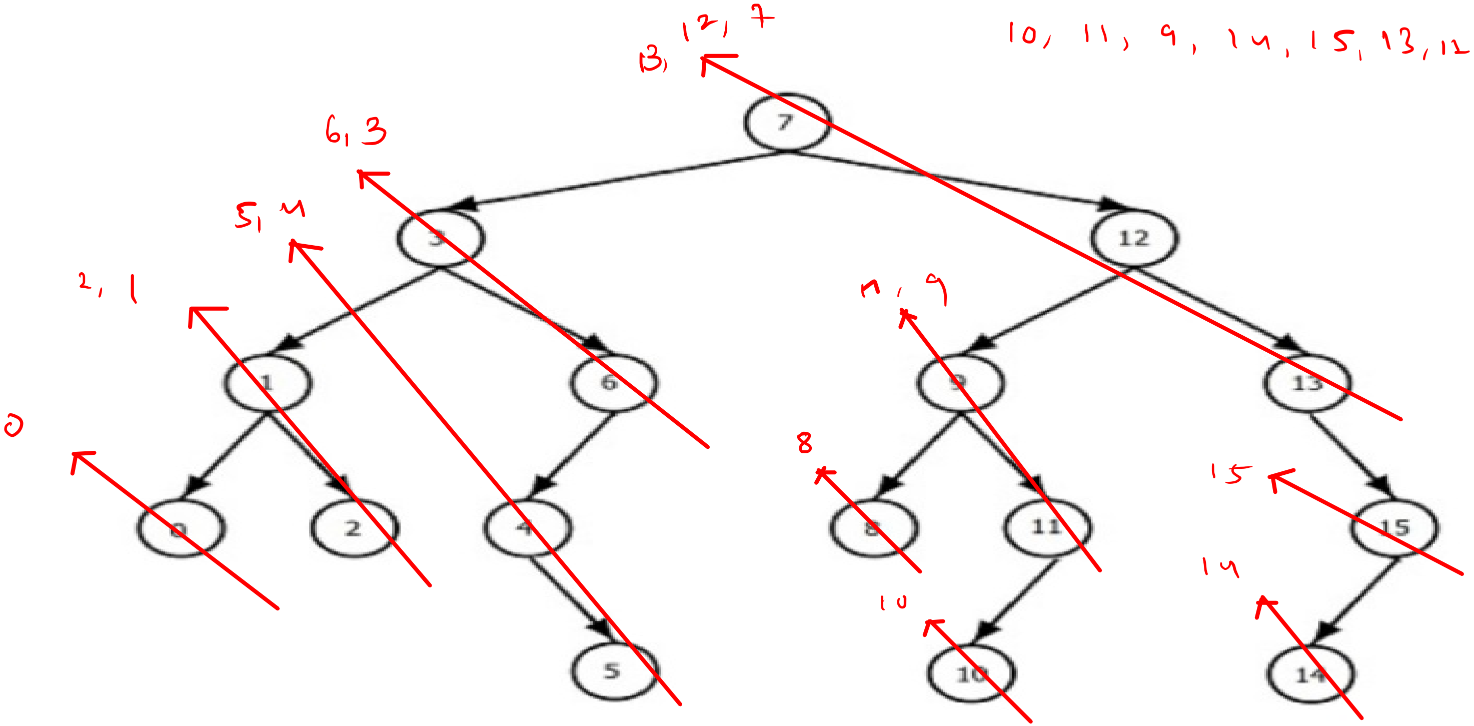
In-order :- [falling style]

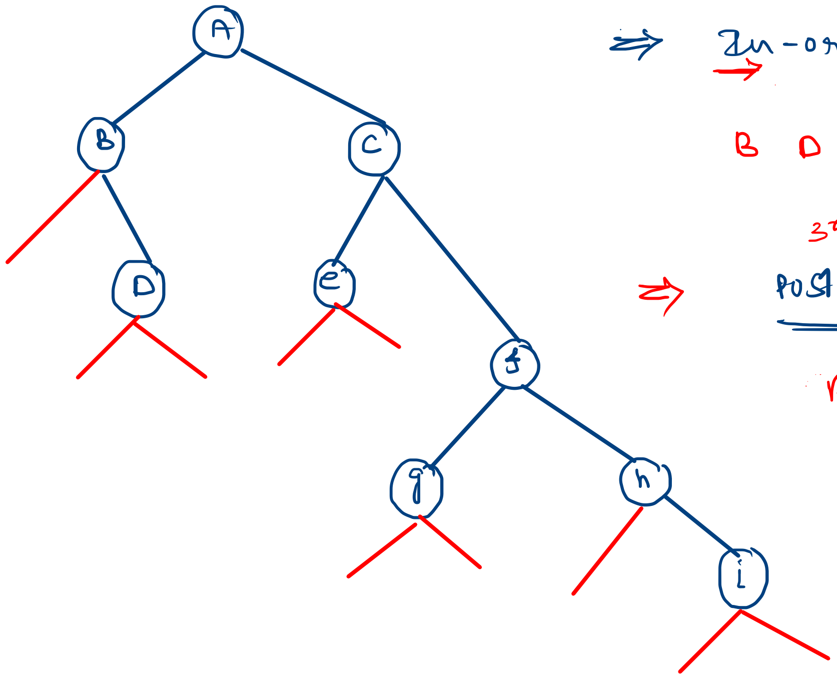


③ post-order [exit style]

⇒ 0, 2, 1, 5, 4, 6, 3, 8,

10, 11, 9, 14, 15, 13, 12, 7





2nd time -

⇒ In-order :-

B D A e C g f h i

3rd

⇒ post-order :-

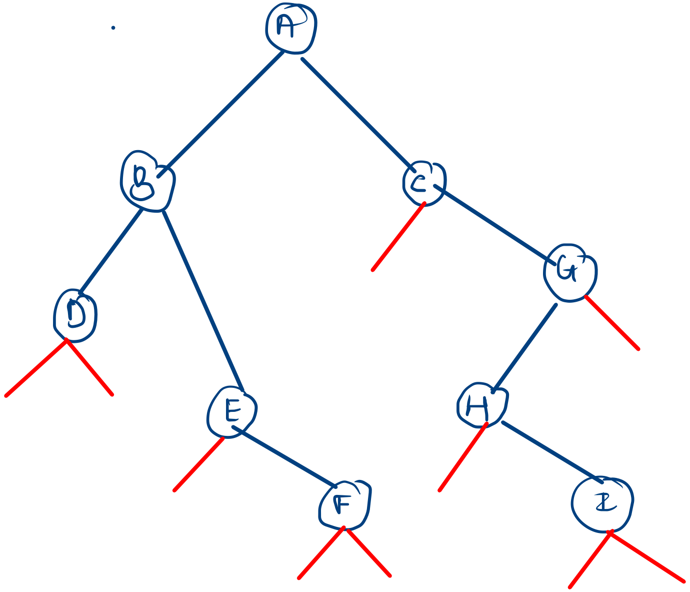
D B e g i h f

C A

⇒ pre-order :-

A B D C e f g

h i



Pre :- A B D E F C G H I

Post :- D F E B I H G C A

In :- D B E F A C H I G

④ level-order

entry

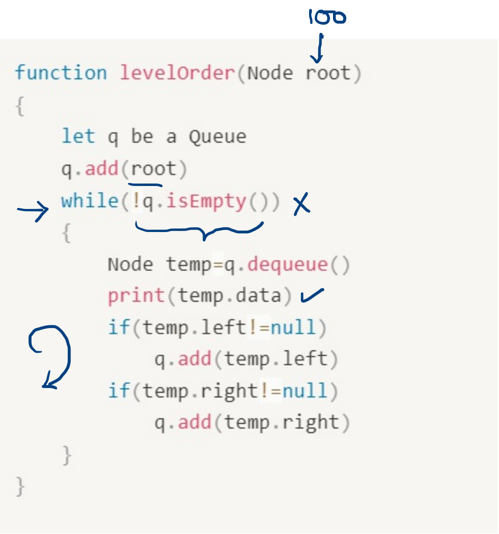
~~700~~ ~~600~~ 500 ~~400~~ ~~300~~

Queue

exit

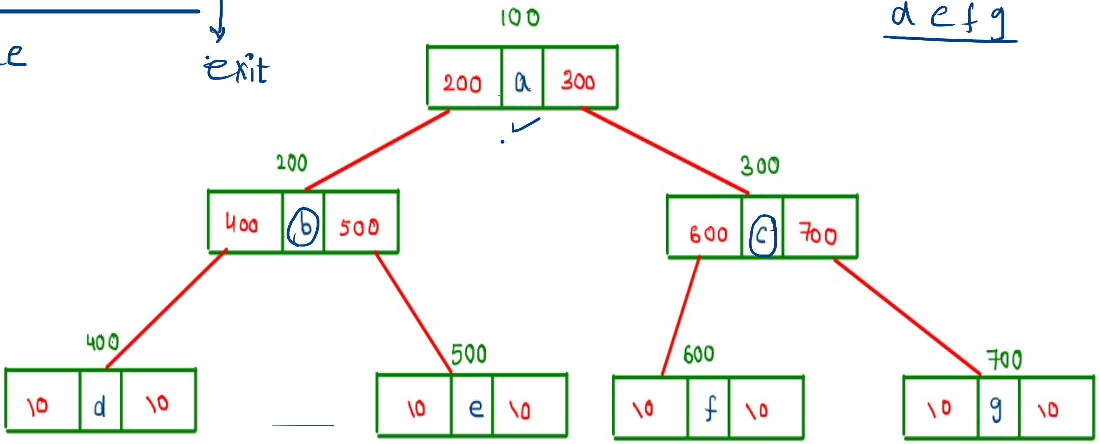
a b c

d e f g



100

```
function levelOrder(Node root)
{
    let q be a Queue
    q.add(root)
    while(!q.isEmpty()) X
    {
        Node temp=q.dequeue()
        print(temp.data) ✓
        if(temp.left!=null)
            q.add(temp.left)
        if(temp.right!=null)
            q.add(temp.right)
    }
}
```



temp = ~~200~~ ~~300~~ 400

~~500~~ 600 700

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

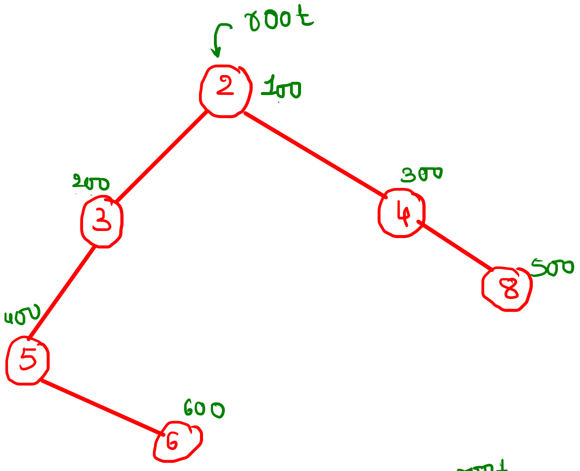
o/p

✓ 1) Find the sum of all nodes in Binary tree

```
                                                   ①
function sumOfNodes(Node root)
{

        if(root==null)
                return 0
        else
                return root.data+sumOfNodes(root.left)+
                                 sumOfNodes(root.right)

}
                                                   ②
```



⇒ 2+3+4+
5+8+6

28

28 ← S(100) (root)

28↑ ↓

ret 2 + S(200)↓ + S( ) (root)

↓↑14 ↑12 → 4 + 8

ret 3+S(400) + S(\0) (r)

↓↑11 ↑0

ret 5+S(\0)+S(600) (r) (r)

↑0 ↓↑6

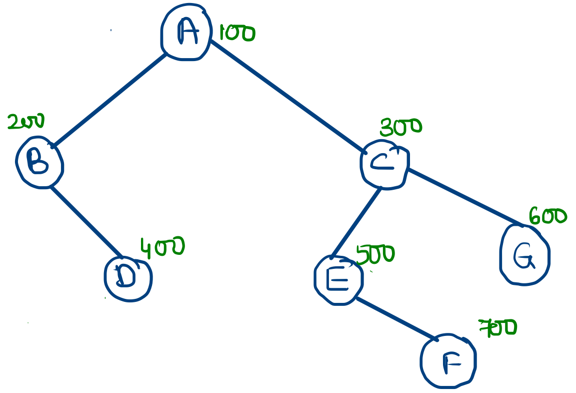
ret 6+S(\0)+S(\0)

↑0 ↑0

⇒ 2) count the total number of leafnodes

```
function leafCount(Node root)
{
        if(root==null)
                return 0 ✓
        if(root.left==null && root.right==null)
                return 1
        else
                return leafCount(root.left)+leafCount(root.right)
}
```

① ②



leaf_nodes = D, F, G ③

↳ Node with zero children

```
                       root
          ③ ⇐ LC( 100 )
                    ↓↑3
             root
      ret   LC(200)      +    LC( 300 )
              ↓↑1                ↓↑2
         root       root
ret  LC(10) + LC(400)     ret  LC(500) + LC( 600 )
      ↺0        ↓↑1              ↓↑1         ↑1
                          ret  LC(10) + LC(700)
                                 ↺0       ↓↑1
```

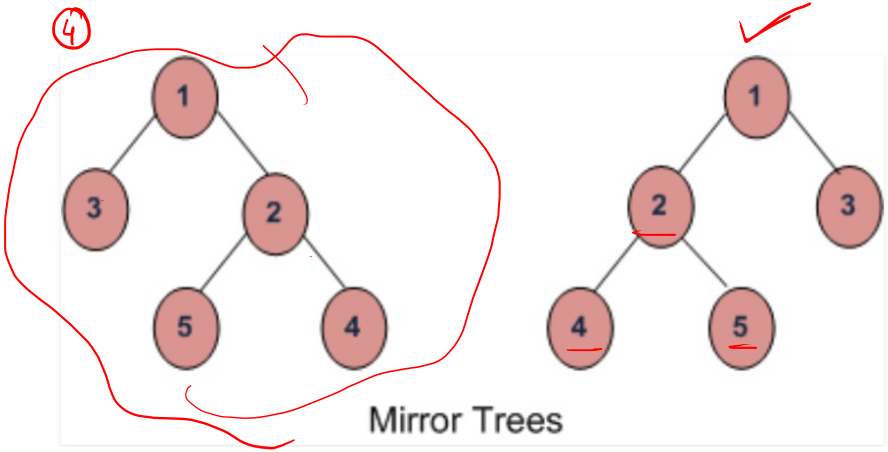
Assignment :-

✓ 1) Find the height of Binary tree

✓ 2) count the number of internal nodes

✓ 3) Sum of the leafnodes

✓ 4) Find the mirror image of trees



Mirror Trees