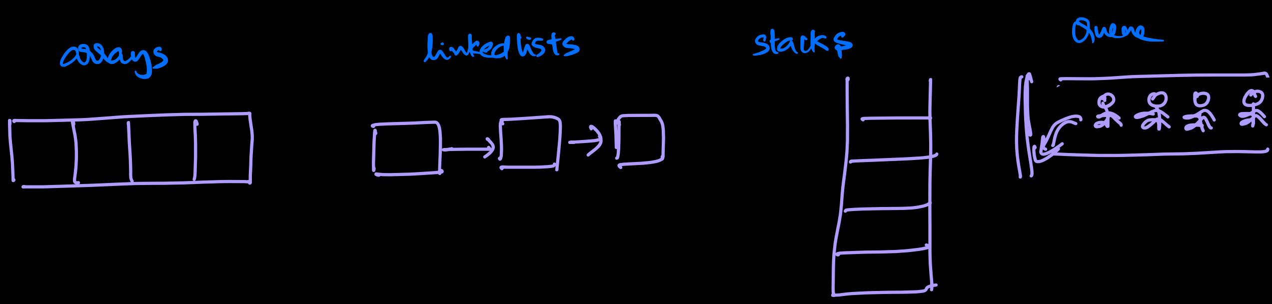
Today's content.

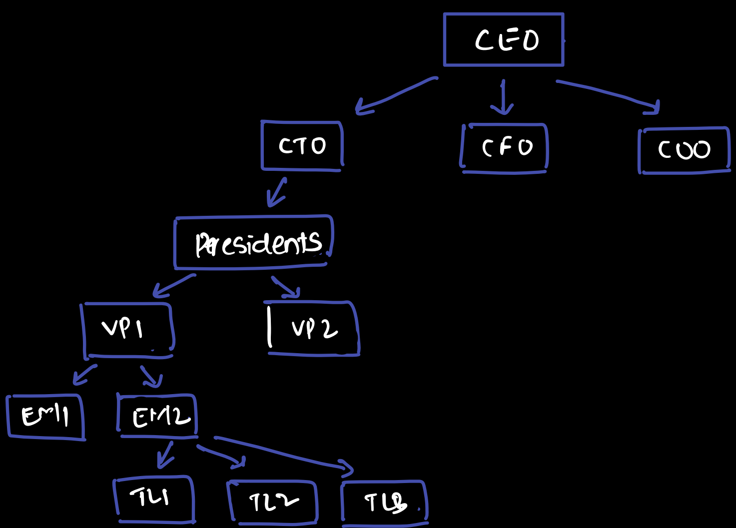
- Trees introduction
- Naming convention
- Tree traversals
- Basic tree problems.

linear:

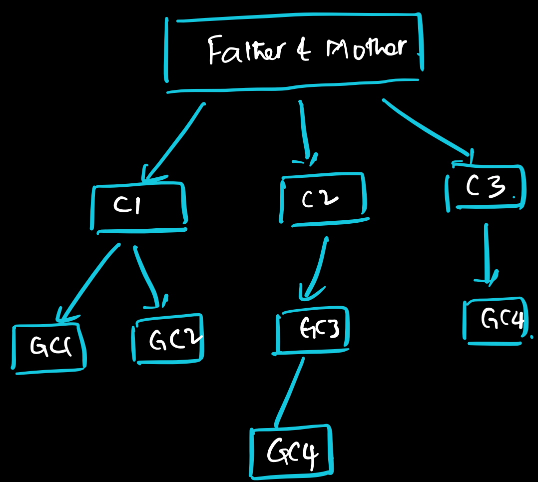


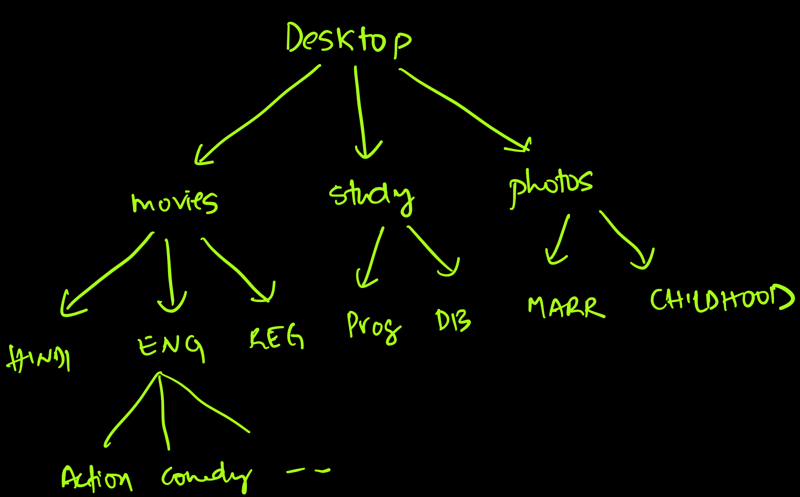
Heirarchical data.

Ex: Company organization.

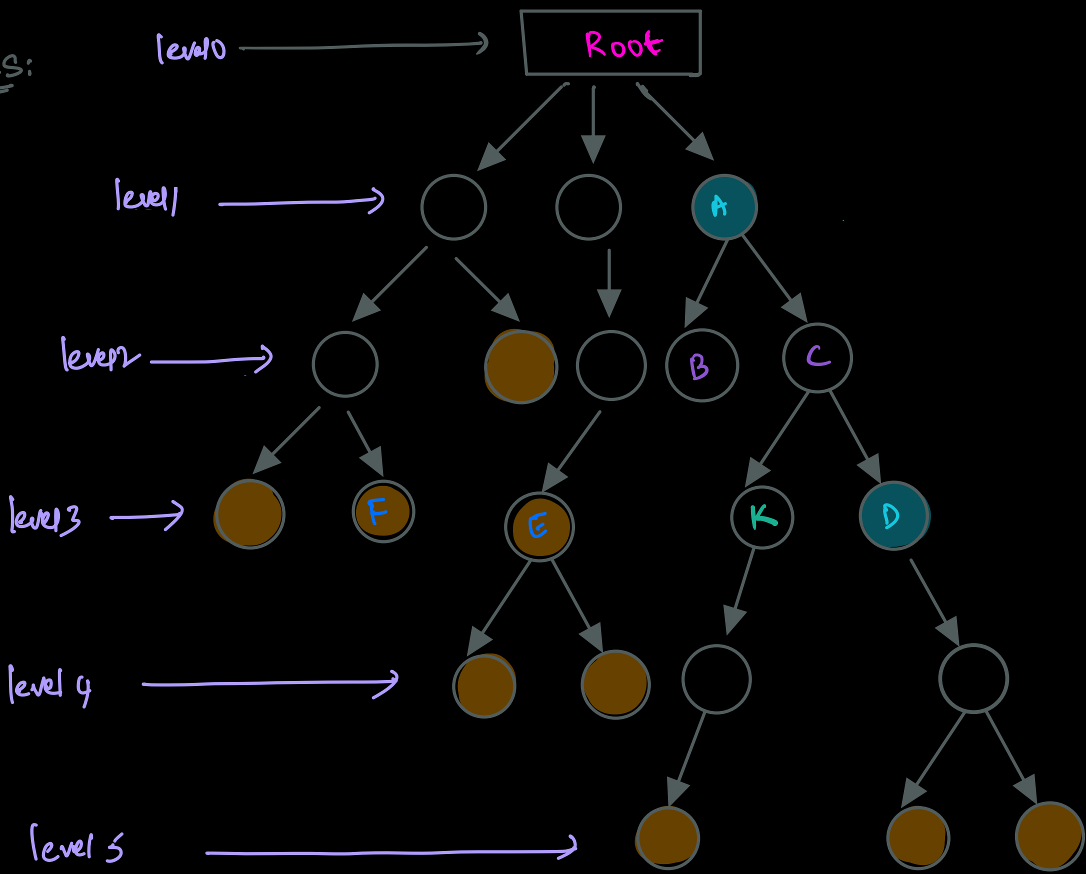


Ex: Family Tree.





Trees:



Relations: Naming conventions.

A & D → A is ancestor of D or D is descendent of A.

B & C → Sibling nodes, share same parent.

F, E, D → Nodes at same level.

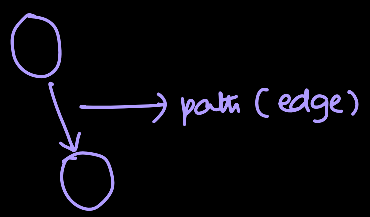
Root → Node with no parent.

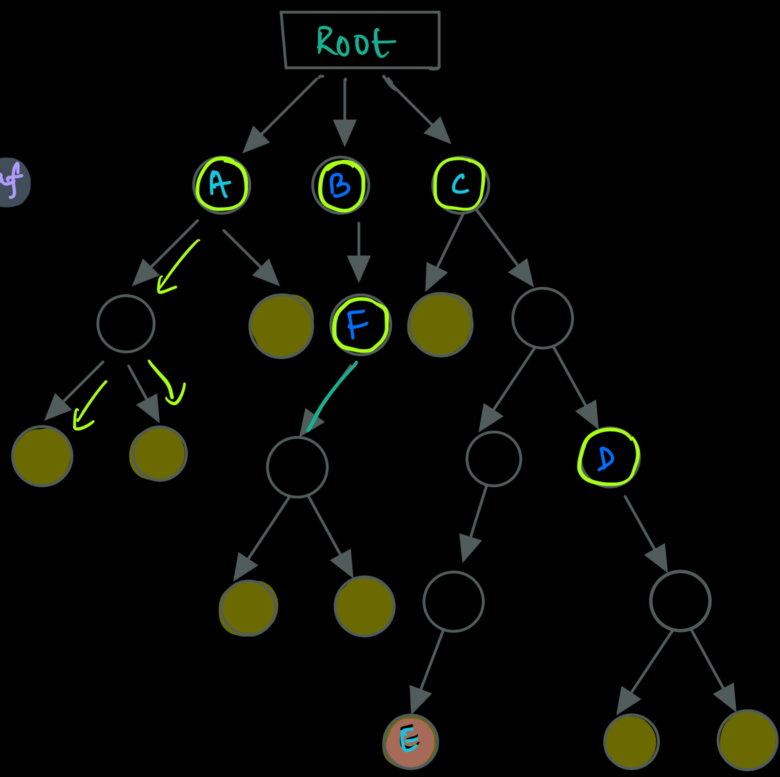
leaf → Node with no children.

Tree → It should have only 1 root node
very node must have single parent

Height (node).

length of the longest path from the node to any of its descendent leaf nodes.





Ex:

$H(A) = 2$

$H(B) = 3$

$H(C) = 4$

$H(E) = 0$

$H(\text{leafnode}) = 0$.

$H(\text{node}) = 1 + \max(\text{Height of its child nodes})$

$H(\text{root}) = \text{Height of tree}$

depth of a node.

length of path from root to the node.

$d(A) = 1$

$d(F) = 2$

$d(E) = 5$

$d(D) = 3$

Depth (root) = 0.

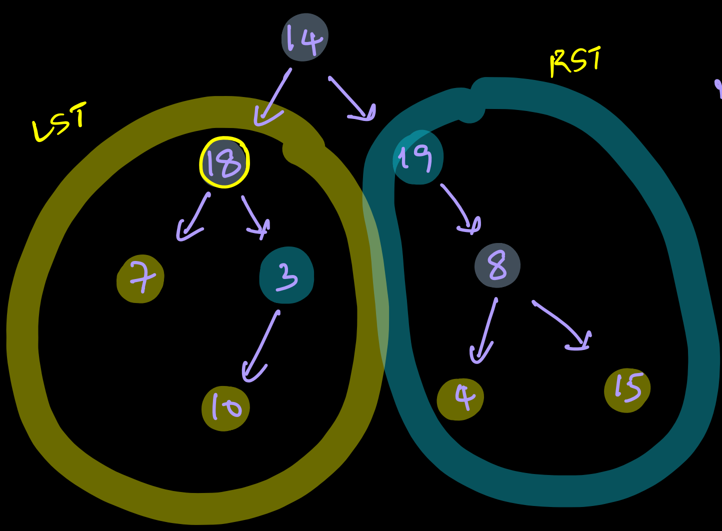
If depth of a node is d,

Then depth of child nodes = d+1.

Our learning is limited to binary trees.

Binary trees. : Every node must have at the max 2 children

0, 1, 2, 3, 4, 5 ..

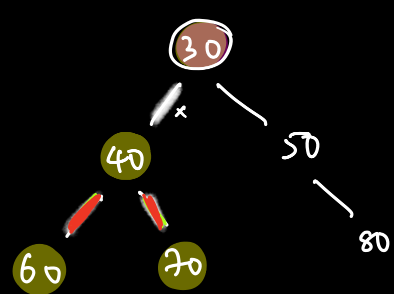


nodes with 1 child : 19, 3

nodes with 0 child : 7, 10, 4, 15

nodes with 2 child : 14, 18, 8

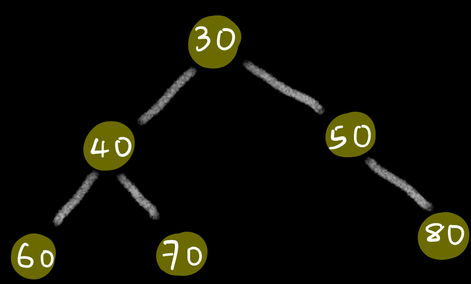
Tree.



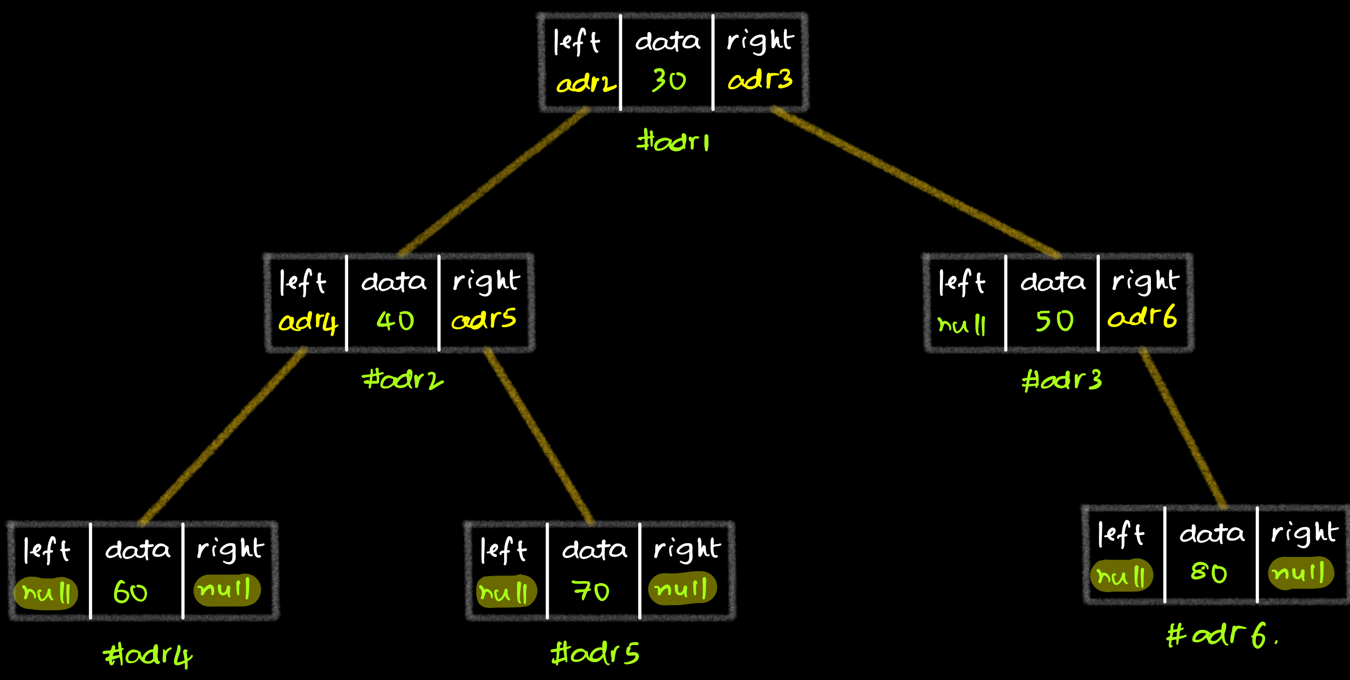
Structure of binary tree nodes.

```
class Node
{
    int data;
    Node left;
    Node right;

    Node(int x)
        data = x
        left = null
        right = null
}
```



root [only this is given in all q's]



Tree traversals

- Inorder
- Preorder
- Postorder.

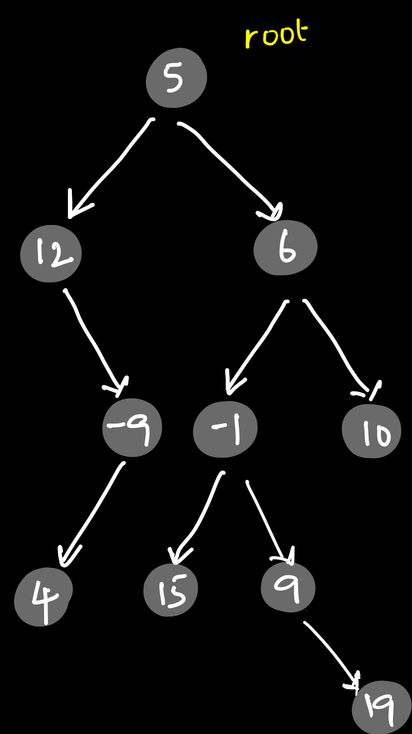
- level order
- vertical level order

Pre-order traversal [Data][LST][RST]

Step 1: print (root.data)

Step 2: Goto left subtree, and print entire left subtree using pre-order traversal

Step 3: Goto right subtree, and print entire right subtree using pre-order traversal



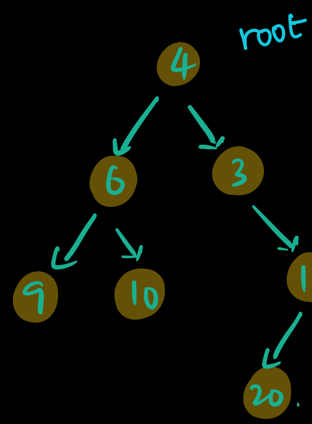
[5, 12, -9, 4, 6, -1, 15, 9, 19, 10]

DLR Pre-order traversal : [4 6 9 10 3 14 20]

LDR In-order traversal : [9 6 10 4 3 20 14]

LRD Post-order traversal : TO-DO.

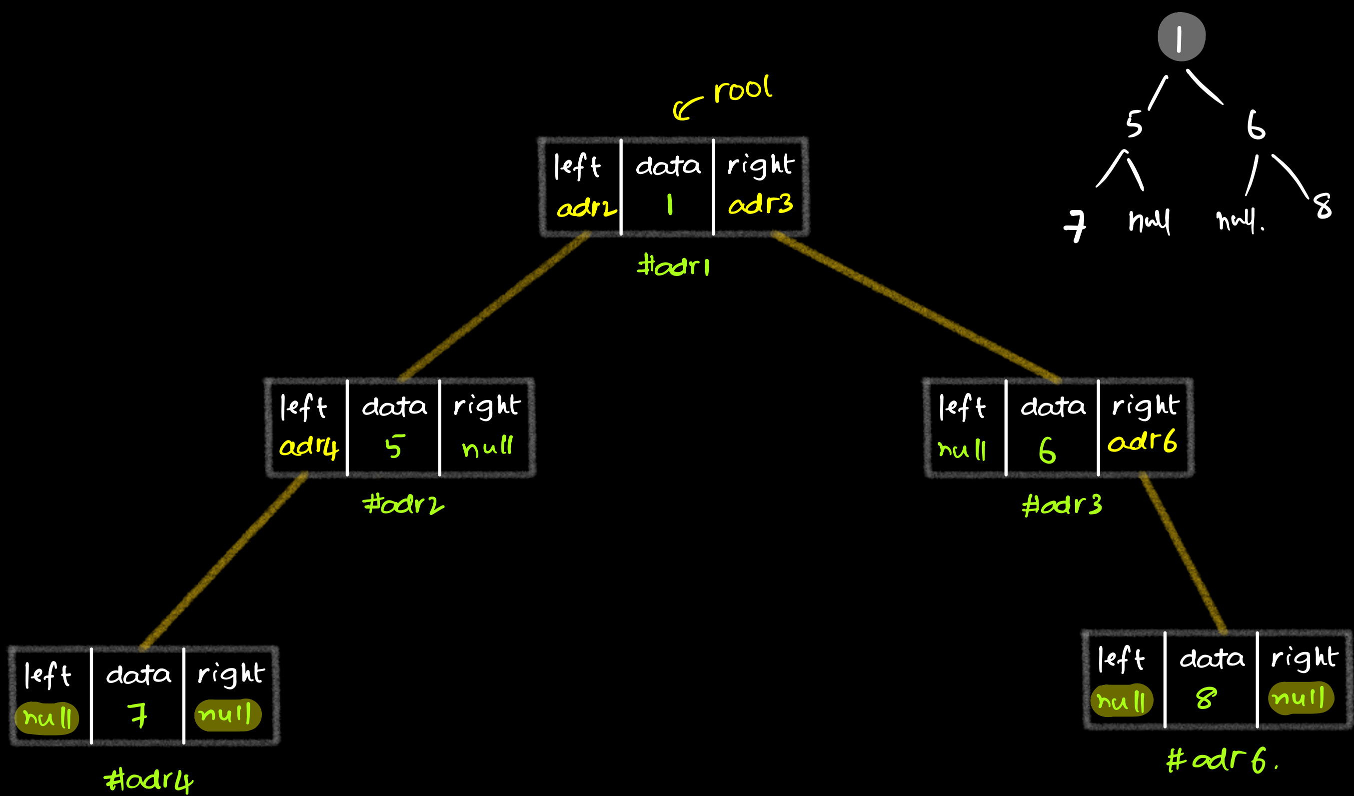
Always 'L' befor 'R', 'D' comes based on traversal.



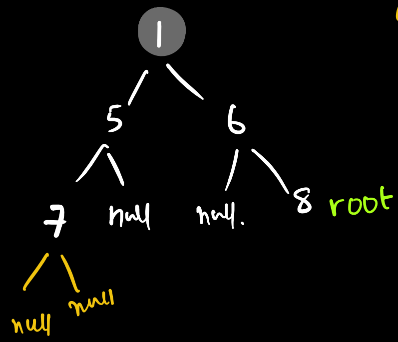
Code:

Preorder traversal.

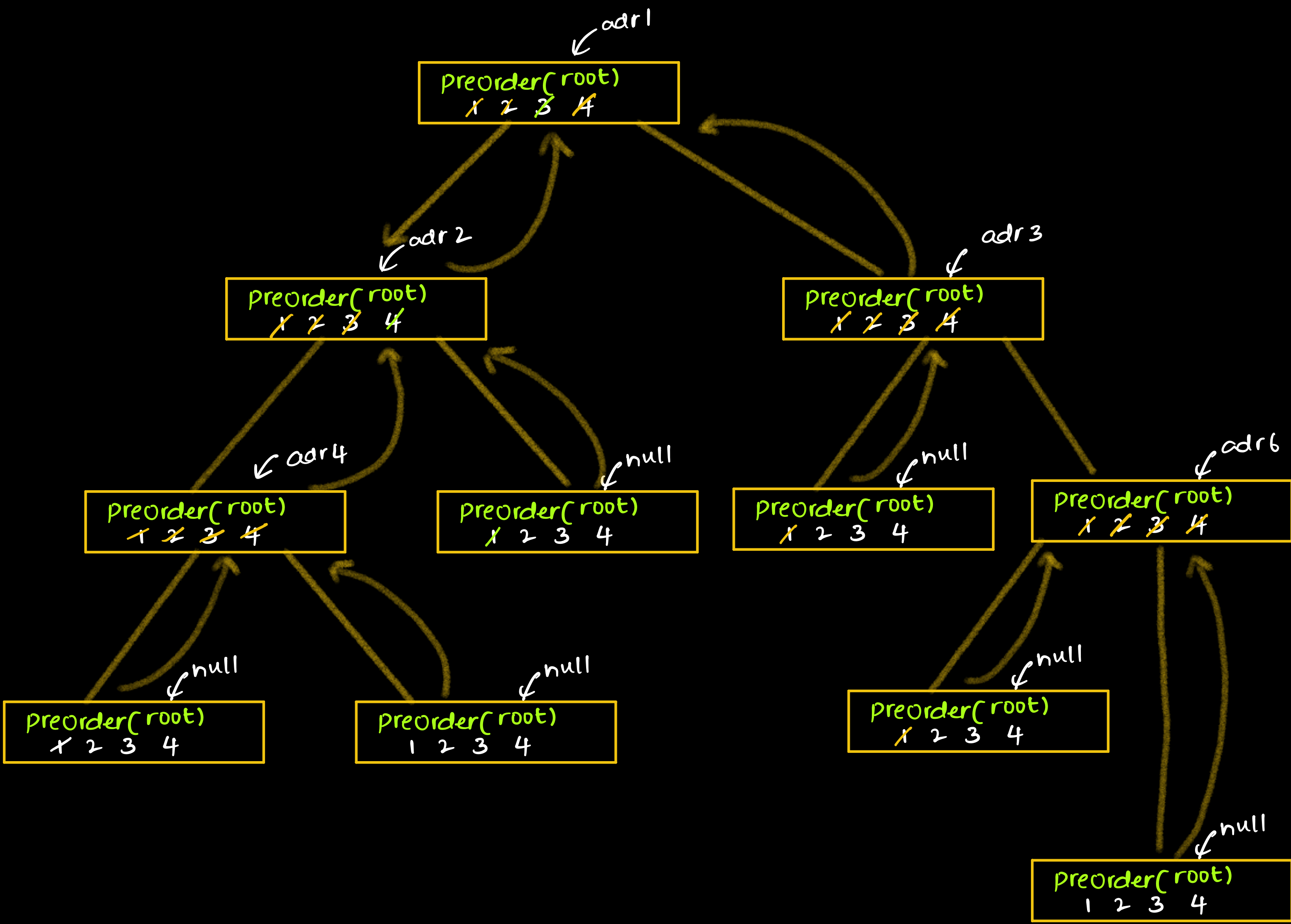
```
Void preorder(Node root)
  1. if(root==null) { return }
  2. print(root.data)
  3. preOrder(root.left)
  4. preOrder(root.right)
```



O/P: 1 5 7 6 8




```
void preorder(Node root)
1. if(root==null) { return }
2. print(root.data)
3. preOrder(root.left)
4. preOrder(root.right)
```



HW.

Code & dry-run for In-order & post-order.

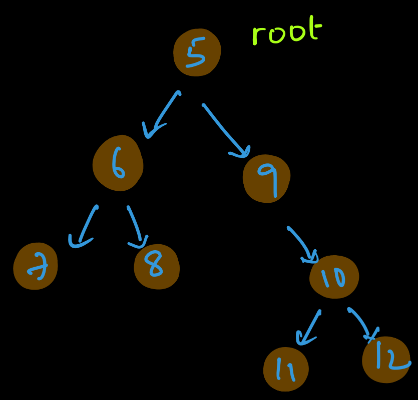
Trees Problems.

// All tree problems, solve them with recursion.

a) size of the tree: How many elements are present in tree.

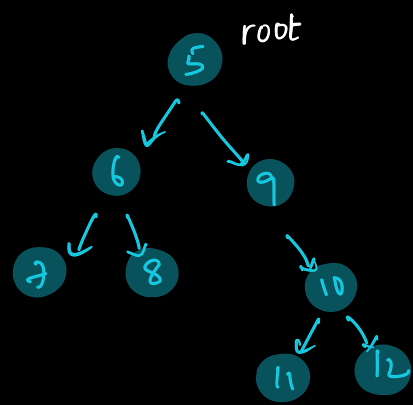
ans = 8

```
int size(root)
    if(root==null)
        return 0
    return 1+ size(root.left)
            +size (root.right)
```



b) Return sum of all nodes, ans = 68

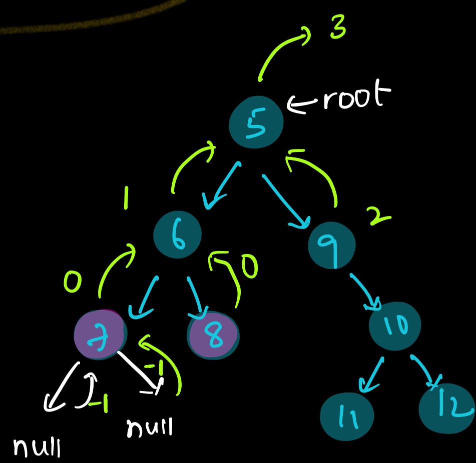
```
int sum(root)
    if(root==null)
        return 0
    return root.data + sum(root.left)
                     + sum(root.right)
```



H(node) = 1 + max(Height of its child nodes)

c) Height of tree.

```
int height(root)
    if(root==null)
        ~~return 0~~ return -1
    return 1 + max [height(root.left),
                    height(root.right)]
```



d) Invert binary tree. [Next class]

```
Node invert (Node root)
{

}
```