Trees Basics

Last class of Intermediate

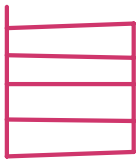
Today's content

- Trees intro
- Naming convention
- Tree traversal
  - Preorder
  - Inorder
  - Postorder
- Basic Tree problems

Linear Data Structure

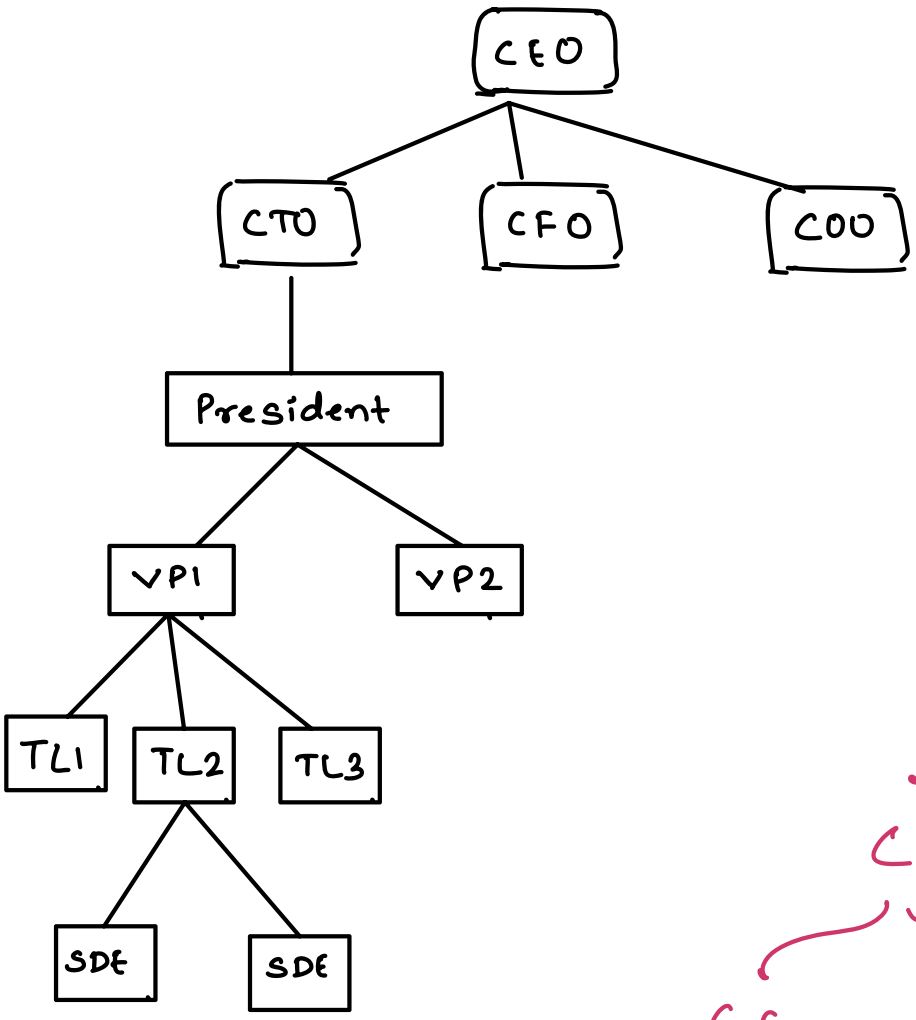
01. Array 

02 Linkedlist 

03. Stack 

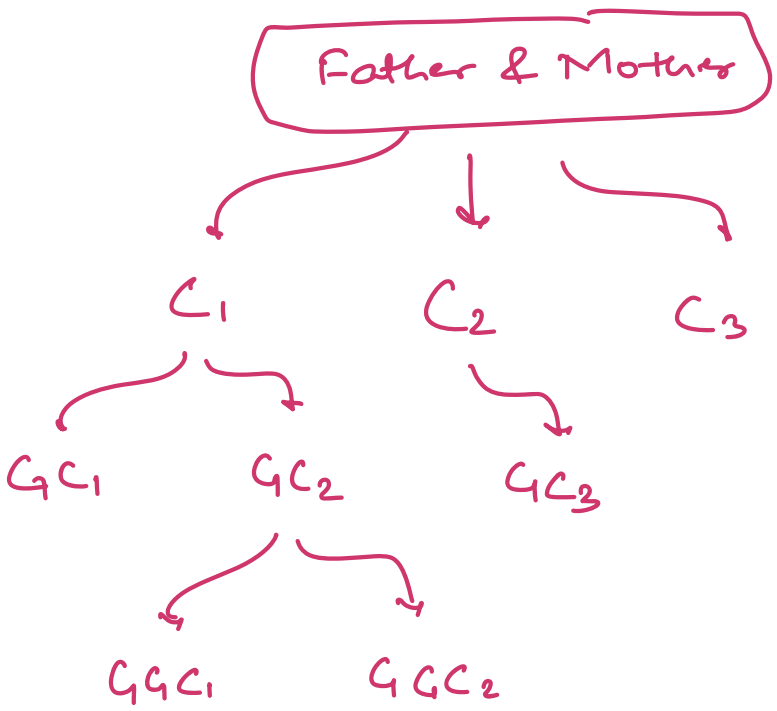
LIFO

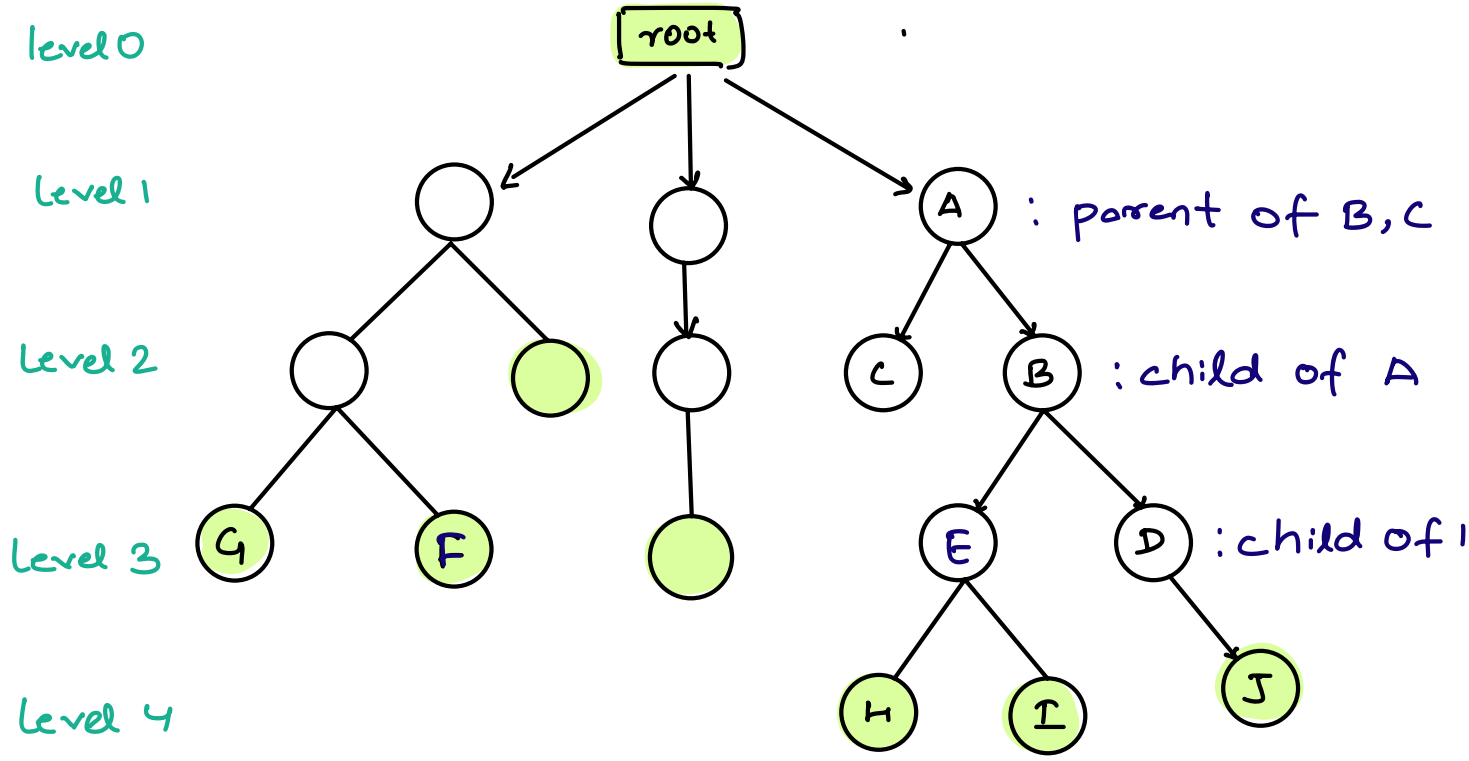
Hierarichal Data Structure



File system

Family tree





A → D A is ancestor of D

D is descendant of A

B ↔ C , siblings , since same parent

D, E, F → Nodes at same level

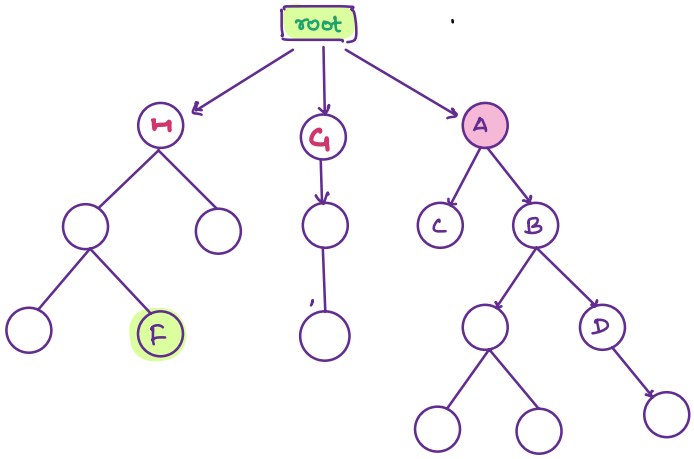
leaf nodes → Nodes with no child

Root → Node without parent

Obs 1 → there is only 1 root node

→ Root node is the ancestor/parent of all the nodes.

Height (node) → length of longest path from node to any of its descendant leaf child



Height (A) = 3

Height (G) = 2

Height (H) = 2

Height (F) = 0

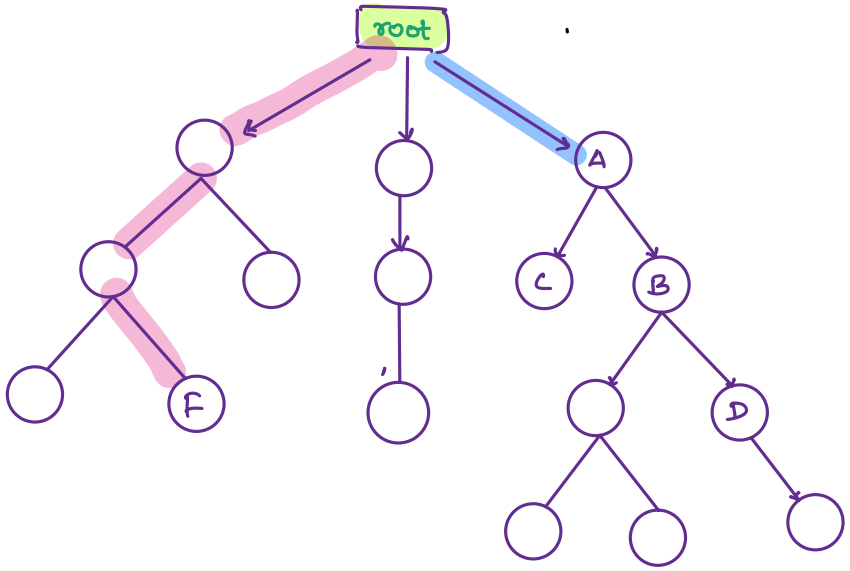
Height (root)

↳ 1 + max height among the root children

Height (node) = 1 + max height among node children

Height (leaf) = 0

Depth (node) → length of the path from root node to given node



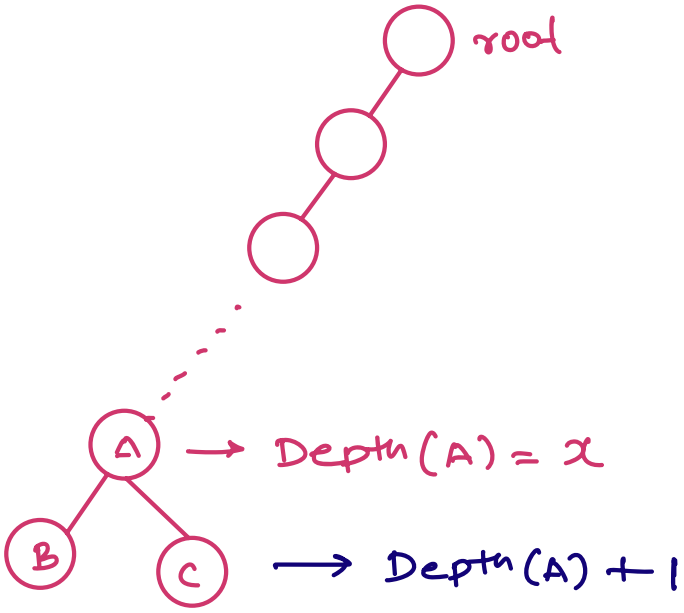
Depth (A) = 1

Depth (F) = 3

Depth (D) = 3

Height = Node to leaf

Depth = Node to root



A → Depth (A) = x

B, C → Depth (A) + 1

Obs 1 → If depth of a node = d

Child (node) = d + 1

Obs 2 → depth of root = 0

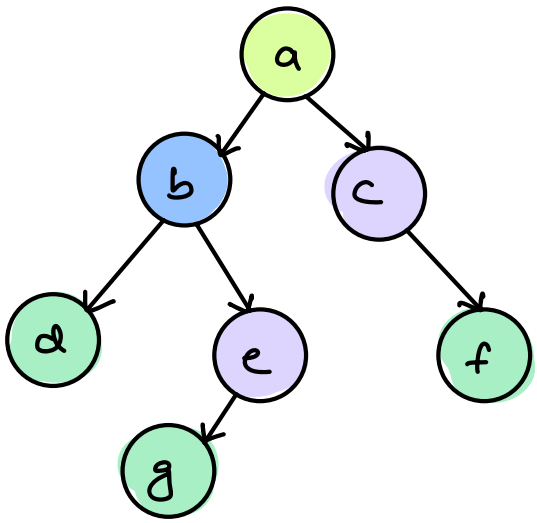
depth(tree) = height(Tree) = Max level

Binary Tree

↳ Every node is going to have at max 2 children

0, 1, 2, 3, 4

✓ x x



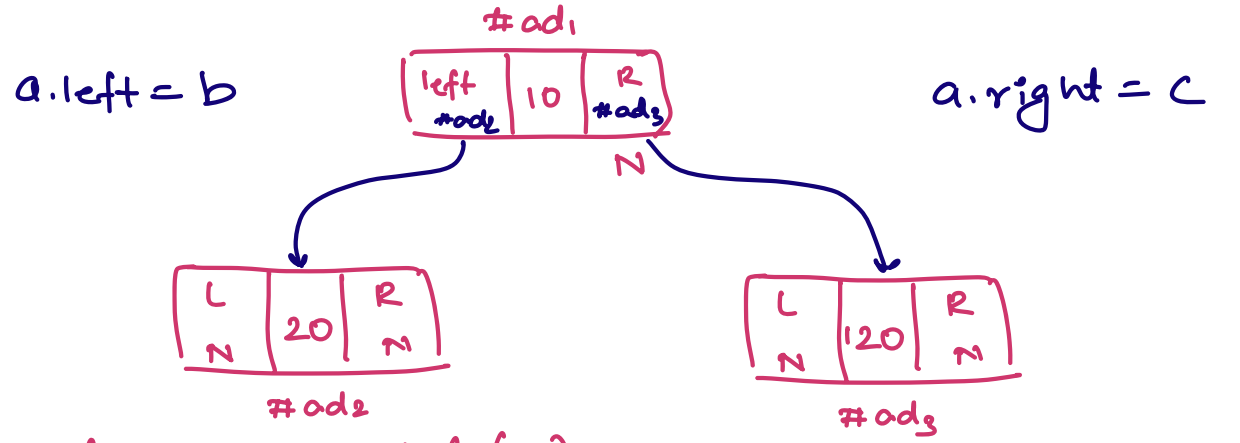
- root node
- node with 2 children
- node with 1 child
- leaf nodes

* class Node

```
int data
Node left; // Obj reference
Node right; // Obj reference
Node (int d) {
    data = d
    left = null;
    right = null;
}
```

```
class Node
    int data;
    ArrayList<Node>
        children = new AL
    Node (int d) {
        data = d;
    }
}
```

Node a = new Node(10);



Node b = new Node(20);

Node c = new Node(30);

Obs → Given root node, we can traverse the complete tree.

Construction of trees → Serialization & Deserialization of trees

For today's class, Assume we are given with completely constructed Tree

Tree traversal

01. Preorder
02. Inorder
03 Postorder

→ Level order traversal
→ Vertical order traversal
→ Morris Inorder traversal

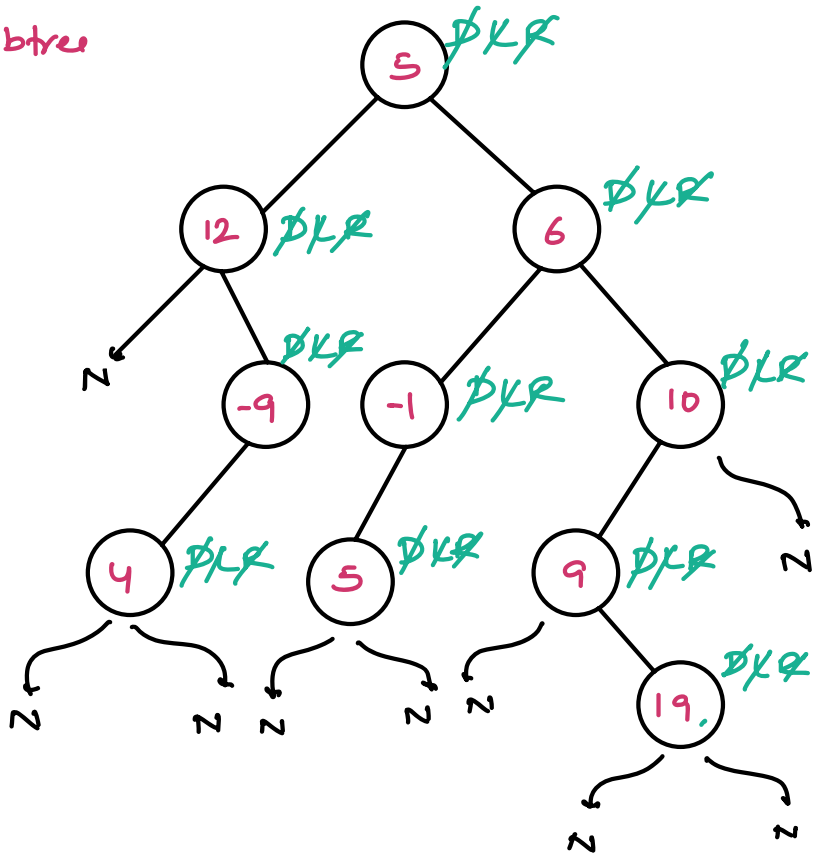
10:14 → 10:24 pm

Preorder : D L R → right subtree
data ↑ → left subtree

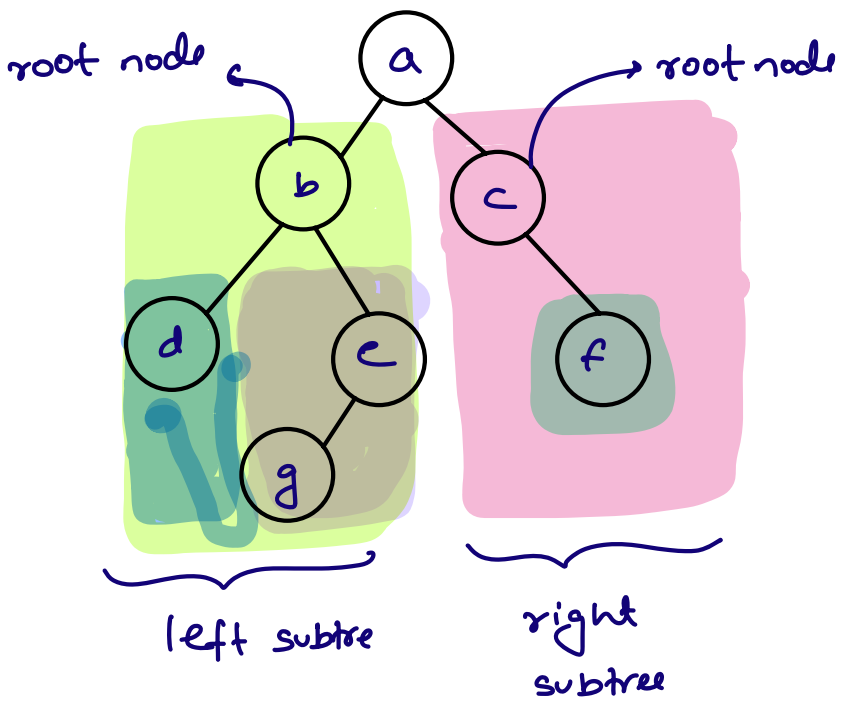
Steps 1 : print (root.data)

Steps 2 : Go to left subtree & print the entire subtree in preorder

Steps 3 : Go to right subtree & print the entire subtree in preorder



Preorder = 5 12 -9 4 6 -1 5 10 9 19



Preorder: a b d e g c f

```
void preorder (Node root)
    if(root == null) return;
    print(root.data)
    preorder(root.left)
    preorder (root.right)
```

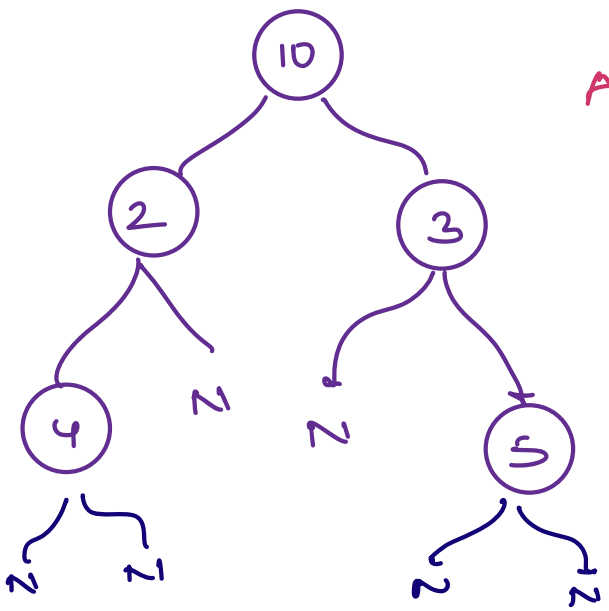
3

count of nodes

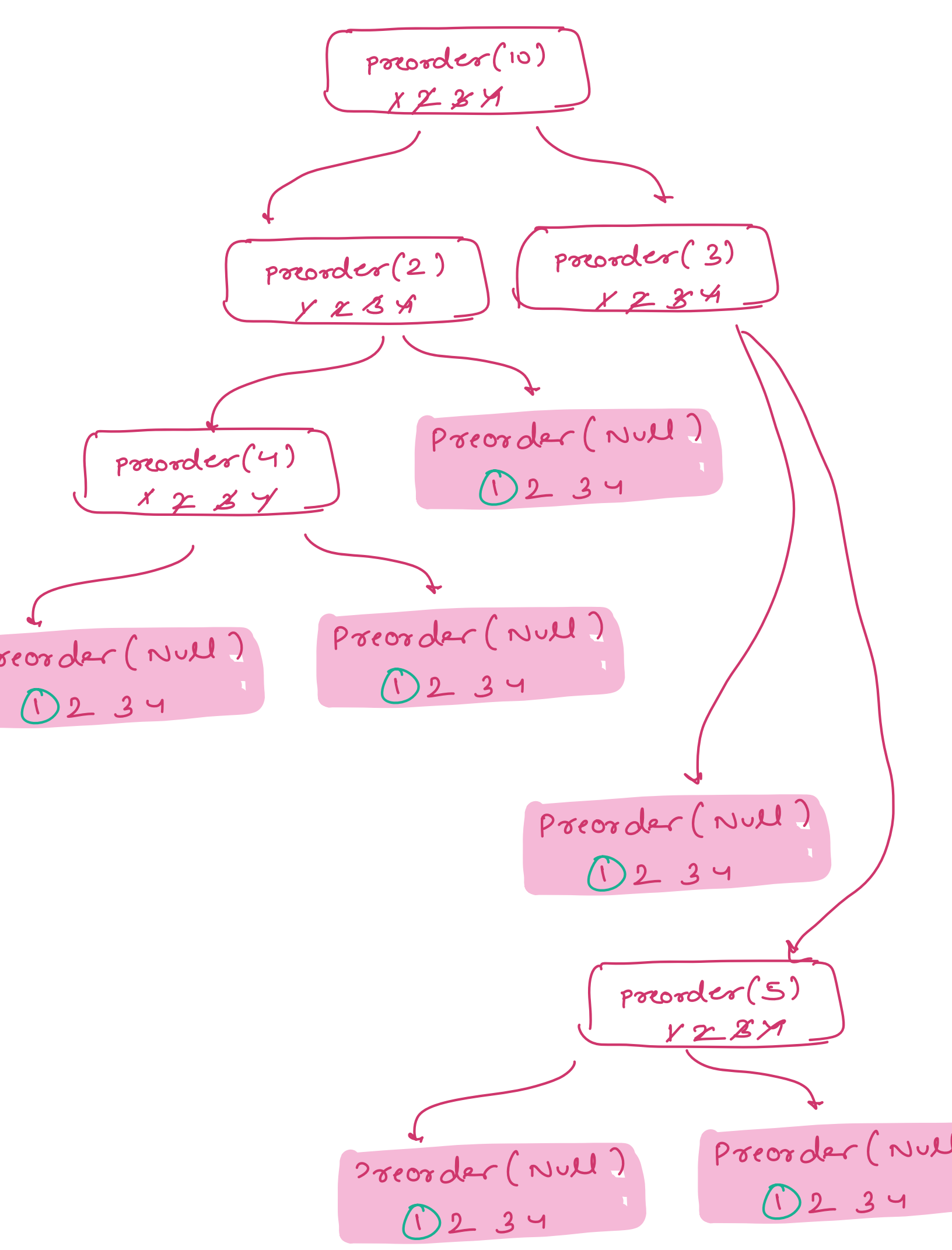
TC : O(n)

SC : O(h)

height of tree



Ans = 10 2 4 3 5


preorder(10)
preorder(2)
preorder(3)
preorder(4)
Preorder(Null)
Preorder(Null)
Preorder(Null)
Preorder(Null)
Preorder(5)
Preorder(Null)
Preorder(Null)

Preorder (D L R)

Inorder (L D R) → 4 2 10 3 5

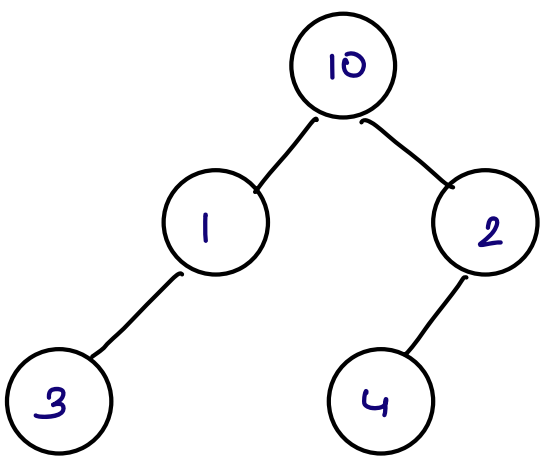
Postorder (L R D) → 4 2 5 3 10

Note :- Solve all these question using recursion without taking any global variable

(a) size (root) → No. of nodes in tree

(b) sum (root) → sum of nodes of tree

(c) Height (root) → Height of complete tree



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

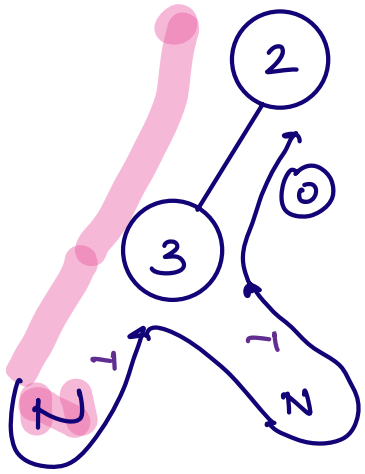
Ass- Given root, calculate & return the no. of nodes

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

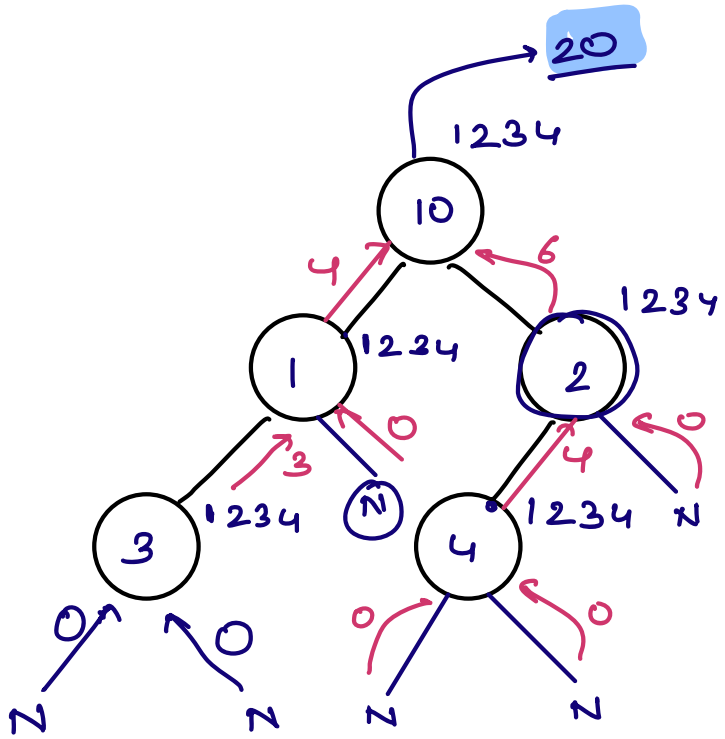
Calculating height in terms of edges
↳ Null return −1

Calculating height in terms of nodes
↳ Null return 0

```
int height(root)
    if (root==null) return -1
    int l= height(root.left)
    int r= height(root.right)
    return Math.max(l,r)+1
3
```



```
int sum (root)
|  if (root==null) return 0
|  int l= sum (root.left);
|  int r= sum (root.right)
|   return  l+r+ root.data;
3
```




```
if (n%2==0) return 0        (1)
int a = fun (n-1)           (2)
int b= func ( Math.floor(n/2));
return a+b
```

even no. → O(1)

N=13 → O(1)

N=14 → O(log N)