Lecture :- Trees

Agenda :-

- Introduction. ✓
- Naming conventions. ✓
- Traversals
- Assignment problems.

Class starts at 8:35 PM

Linear data structure

arr[] = [1 (0) | 2 (1) | 3 (2) | 4 (3) | 5 (4) | 6 (5)]

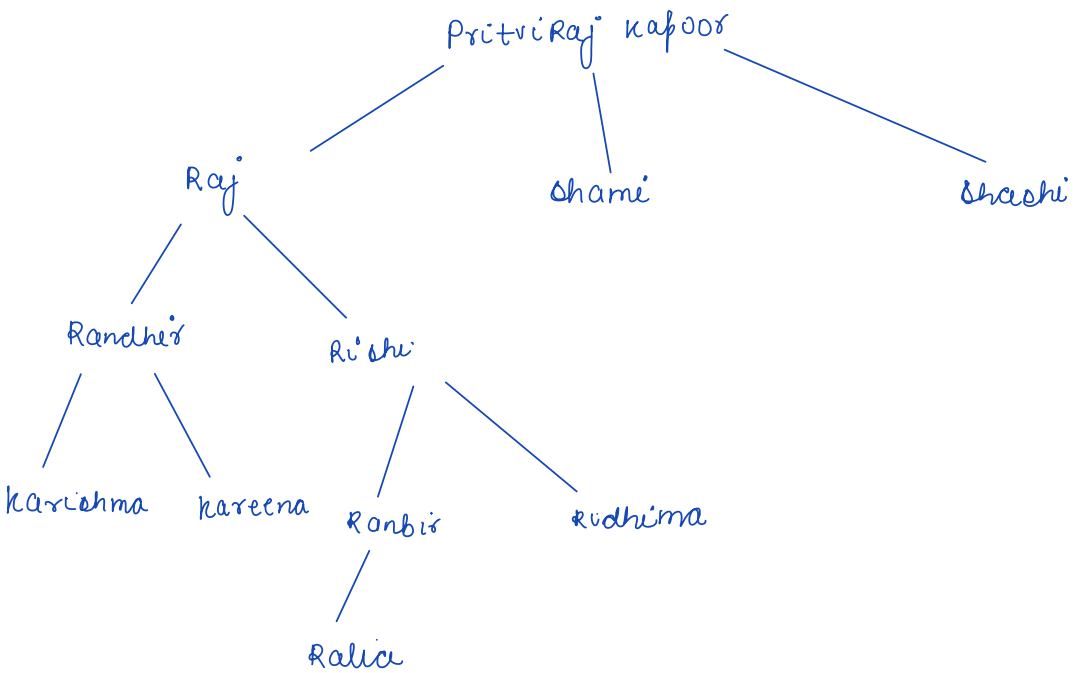
&100 &104 &108 &112 &116 &120

Linked list :- (&100) 1 → (&200) 2 → (&104) 3 → (&219) 4 → null

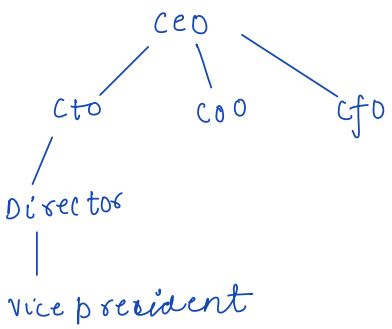
val = 1 next = &200 | val = 2 next = &104 | ....

Heirarchial data structures [Trees]

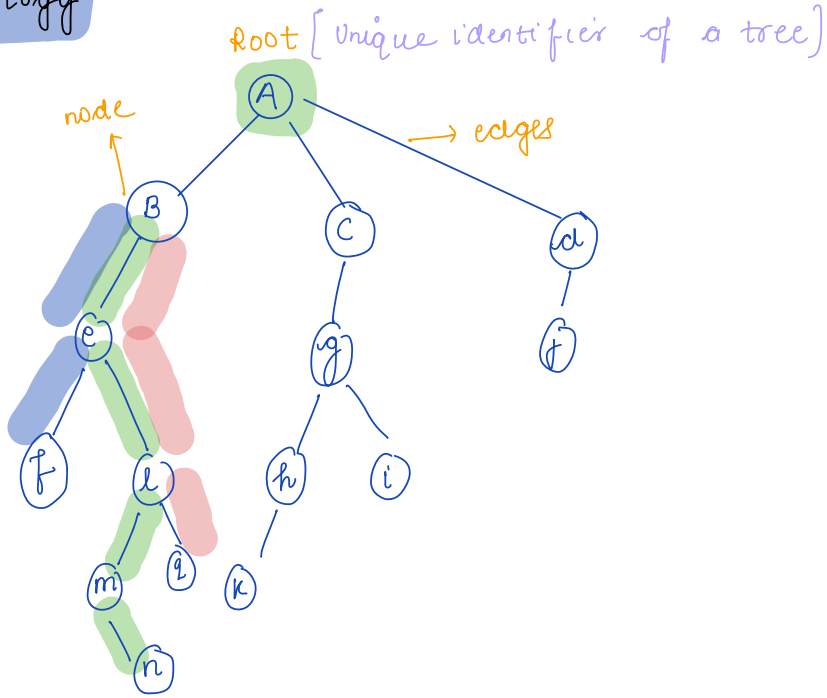
1. family tree.



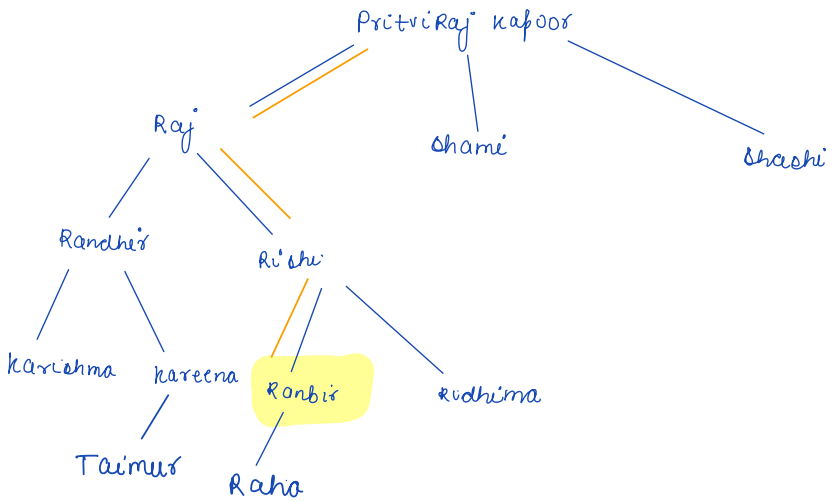
2. Coorporate world



Terminology



height (b) = 4



Parent & child: Raha is a child of Ranbir

Ranbir is a parent of Raha.

Ancestors: ancestors (Ranbir) = Prithvi, Raj and Rishi

Descendents: descendents (Raj): Randhir, Ranbir, Rishi, Raha, Karishma and Kareena

All the nodes below given node

Siblings: Nodes with same parent

Eg: Ranbir and Ridhima

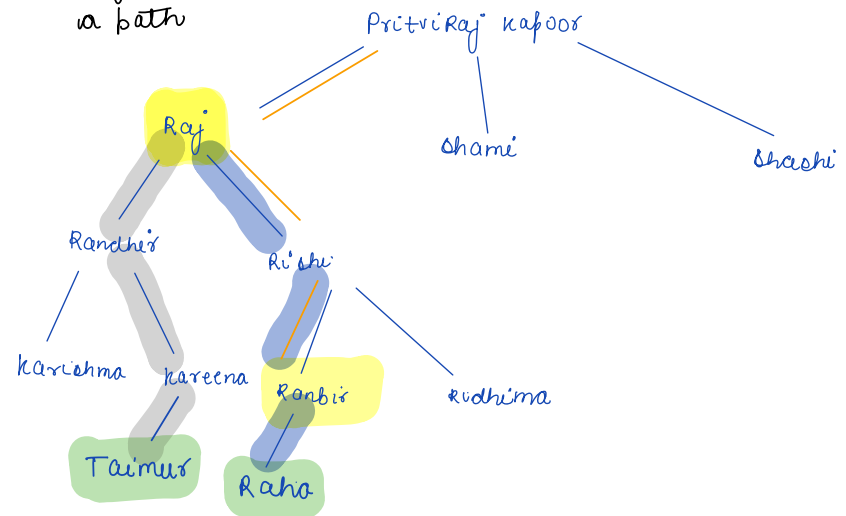
Raj, Shami and Shashi

Leaf nodes: Nodes with no children.

Karishma, Taimur, Raha and Ridhima, Shami, Shashi

Height of a node — length of longest path from given node to a leaf node

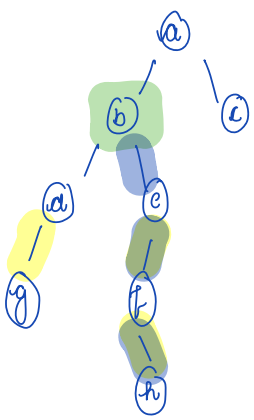
- nodes in a path
- edges in a path



height (Raj) = 3 — counting edges

height (Raj) = 4 — Counting nodes

Observation: 1) height (leaf node) = 0



2) height (b) = max (height (d), height (e)) + 1 (children)

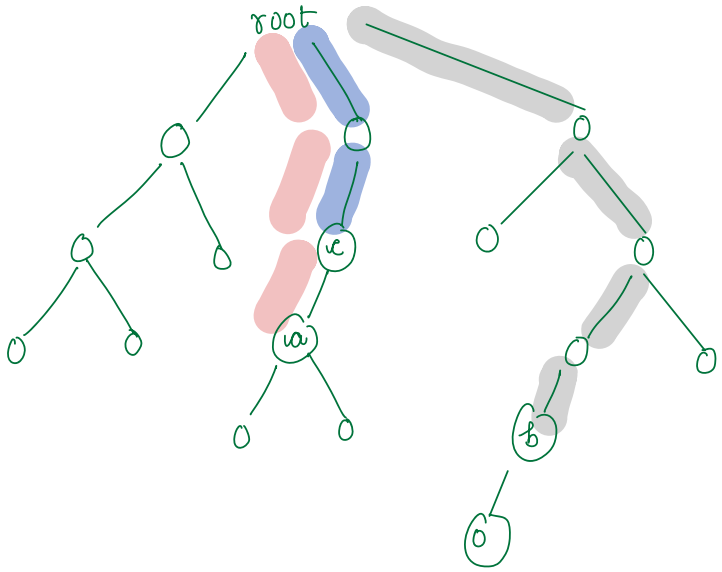
height (d) = 1

height (e) = 2

3) height (tree) = height (root)

Depth of a node

length of path from root to given node



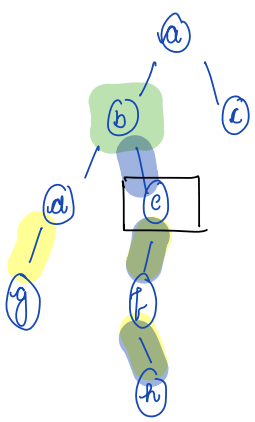
depth (c) = 2

depth (b) = 4

depth (a) = 3

1.) depth (root node) = 0

2.) depth (node) = depth (parent) + 1



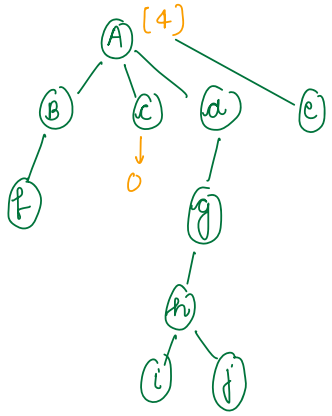
depth (f) = depth (c) + 1.

depth (e) = 2

Type of trees

n-ary tree [Advanced]

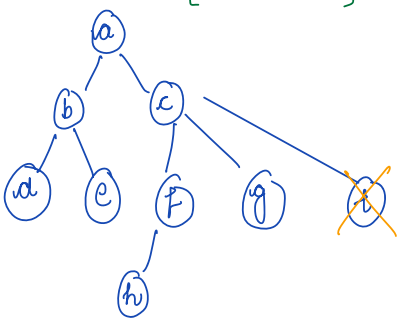
[Any no of children]



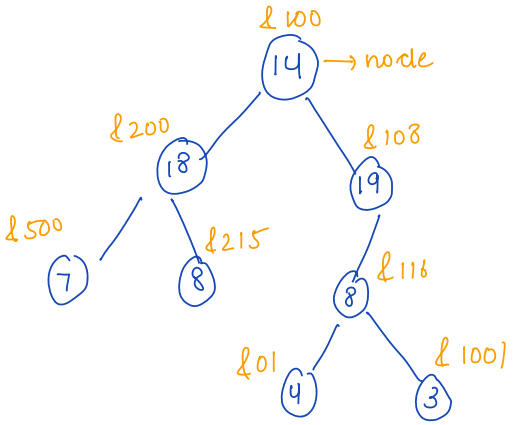
Binary tree [Intermediate]

Max of only 2 children

[0, 1, 2]



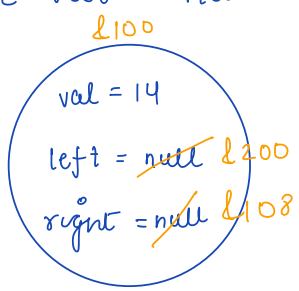
Structure of Binary tree



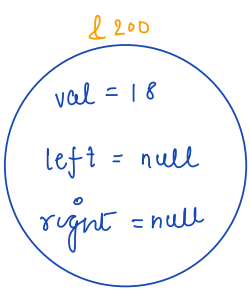

```
class Node {
    int val;
    Node left;
    Node right;
    Node(int x) {
        val = x;
    }
}
```

Detailed:

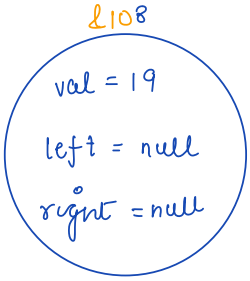
Node root = new Node(14)



Node n1 = new Node(18);

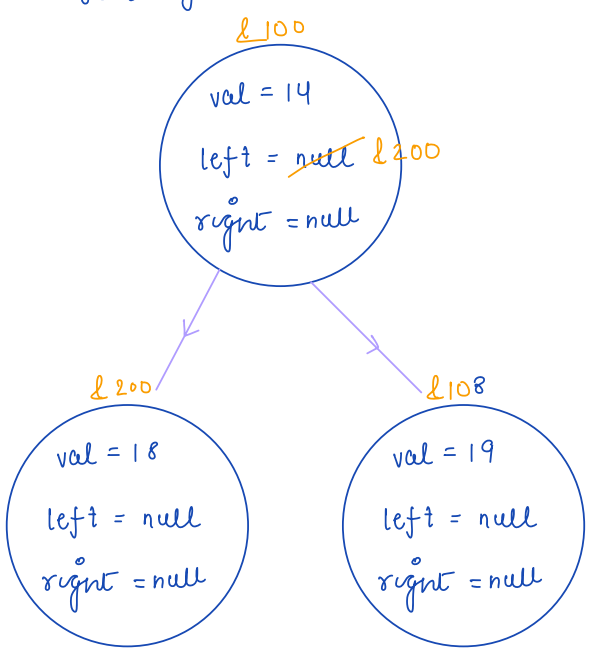


Node n2 = new Node(19);



root.left = n1.

root.right = n2



Break: 9:44 PM

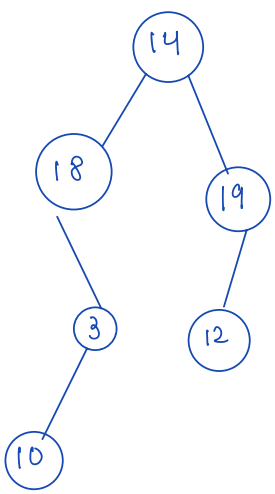
Note:- i/p will already be given, root will be given

Traversals

Recursive traversals.

- Pre-order [Root left right]
- Inorder [left root right]
- Postorder [left right root]

Qu: Print all the nodes of trees in any manner you want.



Opt: 14 18 19 12 3 10

14 18 12 19 10 3

18 14 3 10 19 12

} valid

Post order traversal

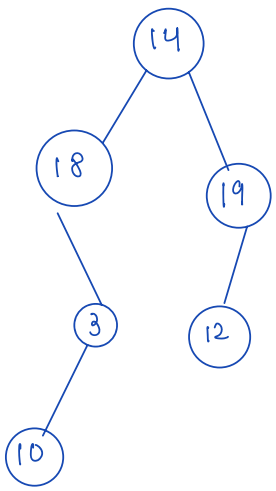
```
void traversal(Node root) {
    if (root == null) {
        return;
    }
    traversal(root.left)  // 18 10 3
    traversal(root.right) // 19 12
    print(root.data)      // 14
}
```

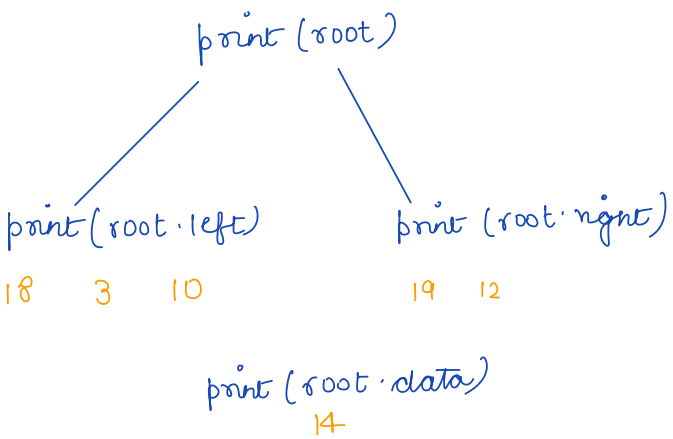
Assumption

Given a root, print whole tree

Given a node, print all the nodes of tree starting from that node

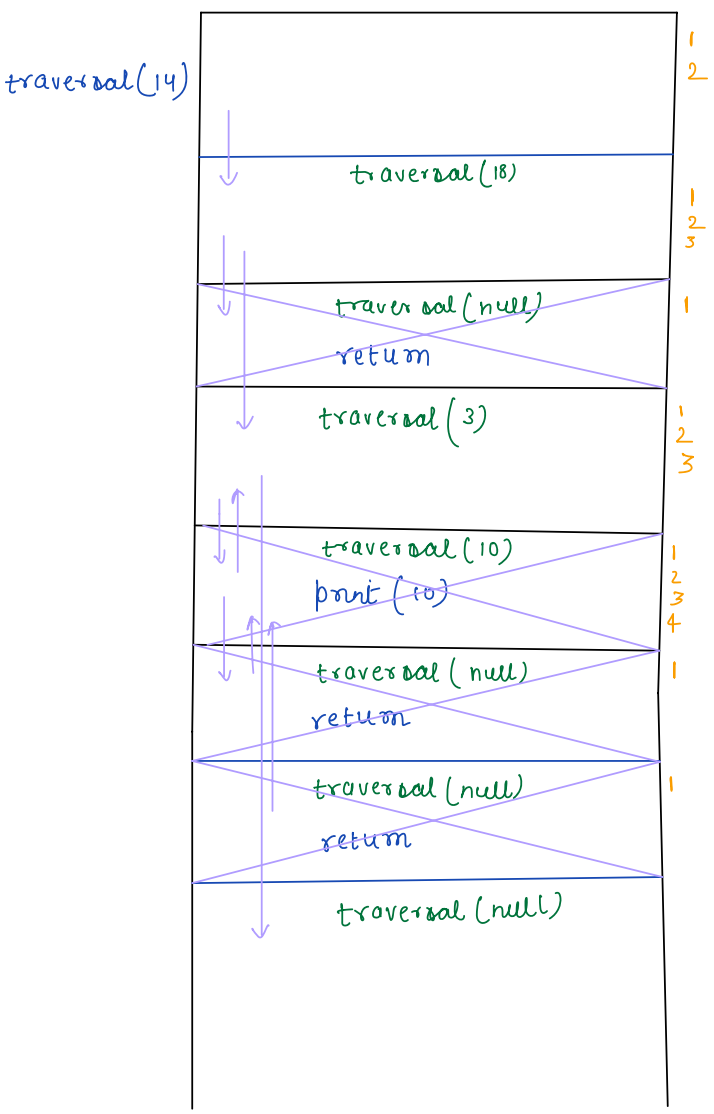
Main logic



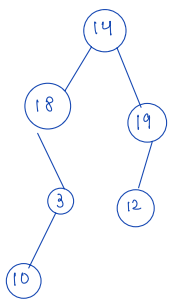


Base condition

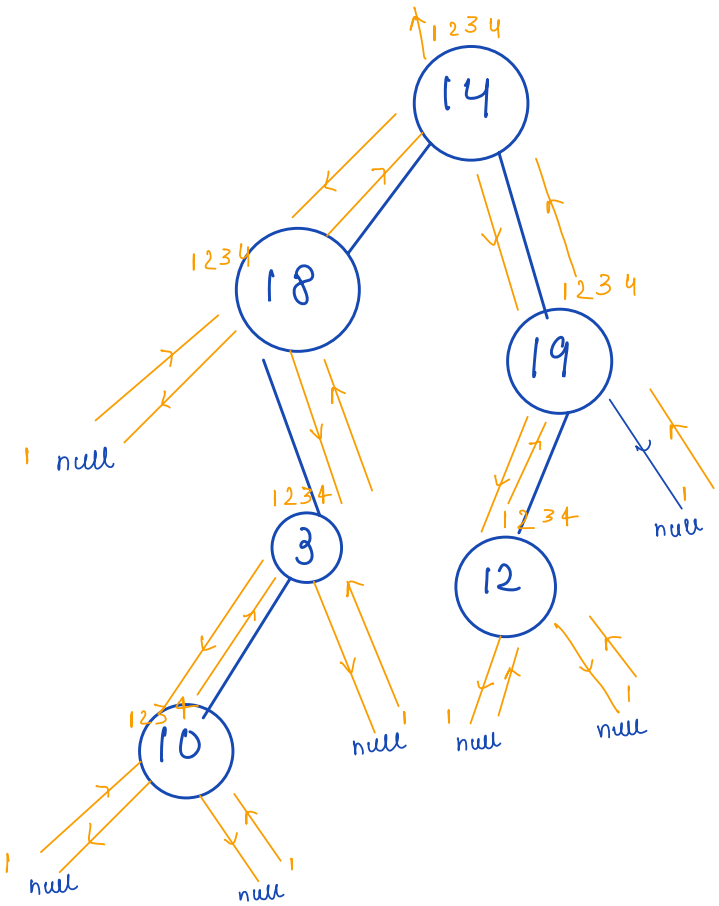
```
if (root == null) return;
```




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



op: 10



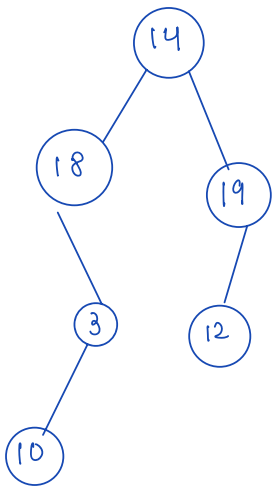

```
void traversal(Node root) {
    1 if (root == null) {
          return;
      }
    2 traversal(root.left)
    3 traversal(root.right)
    4 print(root.data)
}
```

op: 10 3 18 12 19 14

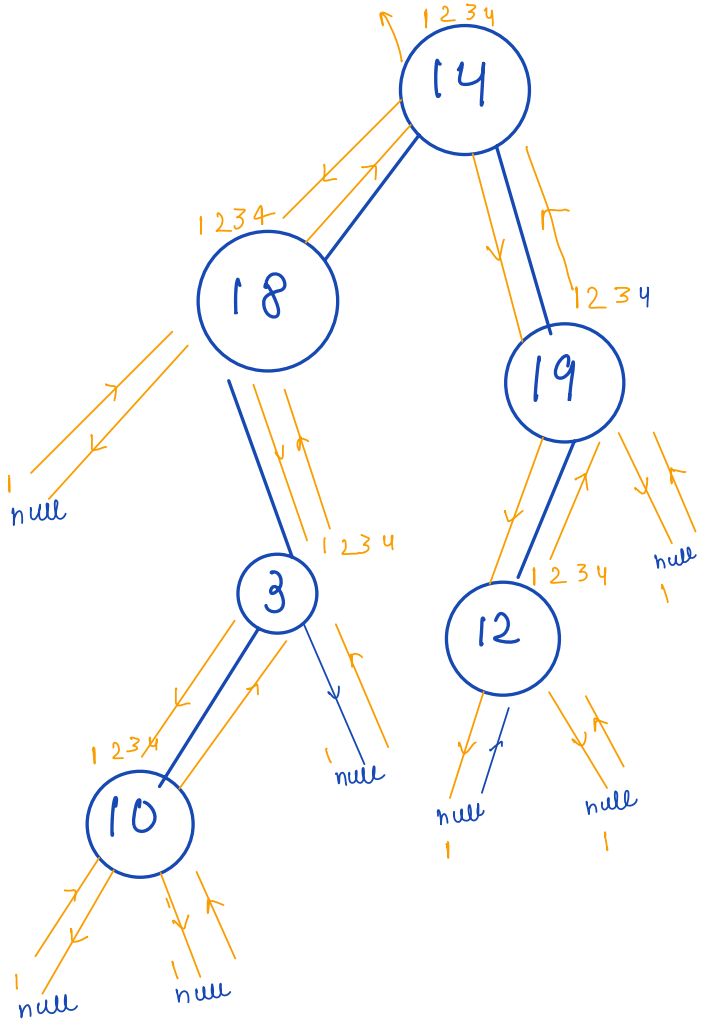
pre-order traversal

[Root left right]

subproblem subproblem




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



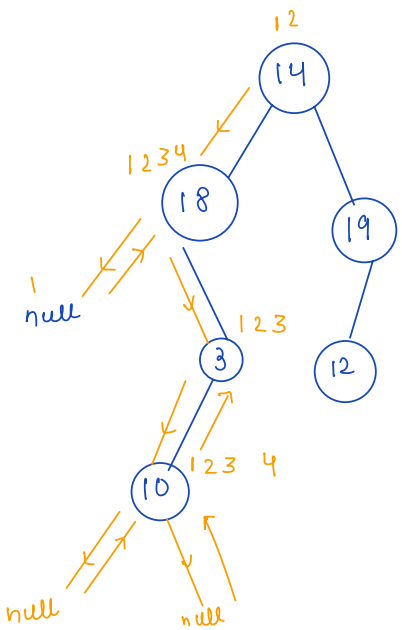

```
void preorderTraversal(Node root) {
1  if (root == null) {
       return;
   }
2  print(root.data)
3  preorderTraversal(root.left);
4  preorderTraversal(root.right);
}
```

op: 14 18 3 10 19 12

Inorder traversal

Left Subproblem Root Right Subproblem

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

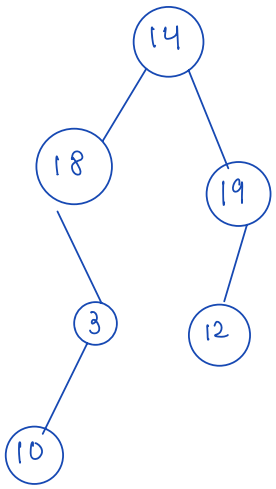



```
void inorder(Node root) {
1   if(root == null) {
        return;
    }
2   inorder(root.left);
3   print(root.data);
4   inorder(root.right);
}
```

op: 18 10 3

Assignment: complete dry run

Q. Count no. of nodes in a tree



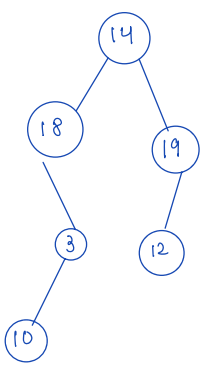
op: 6

```
int count( Node root) {
    if (root == null) {
        return 0;
    }
    int la = count(root.left);
    int ra = count(root.right);
    return la + ra + 1;
}
```

Assumption

Given a node, count all nodes starting from the given node

Main logic

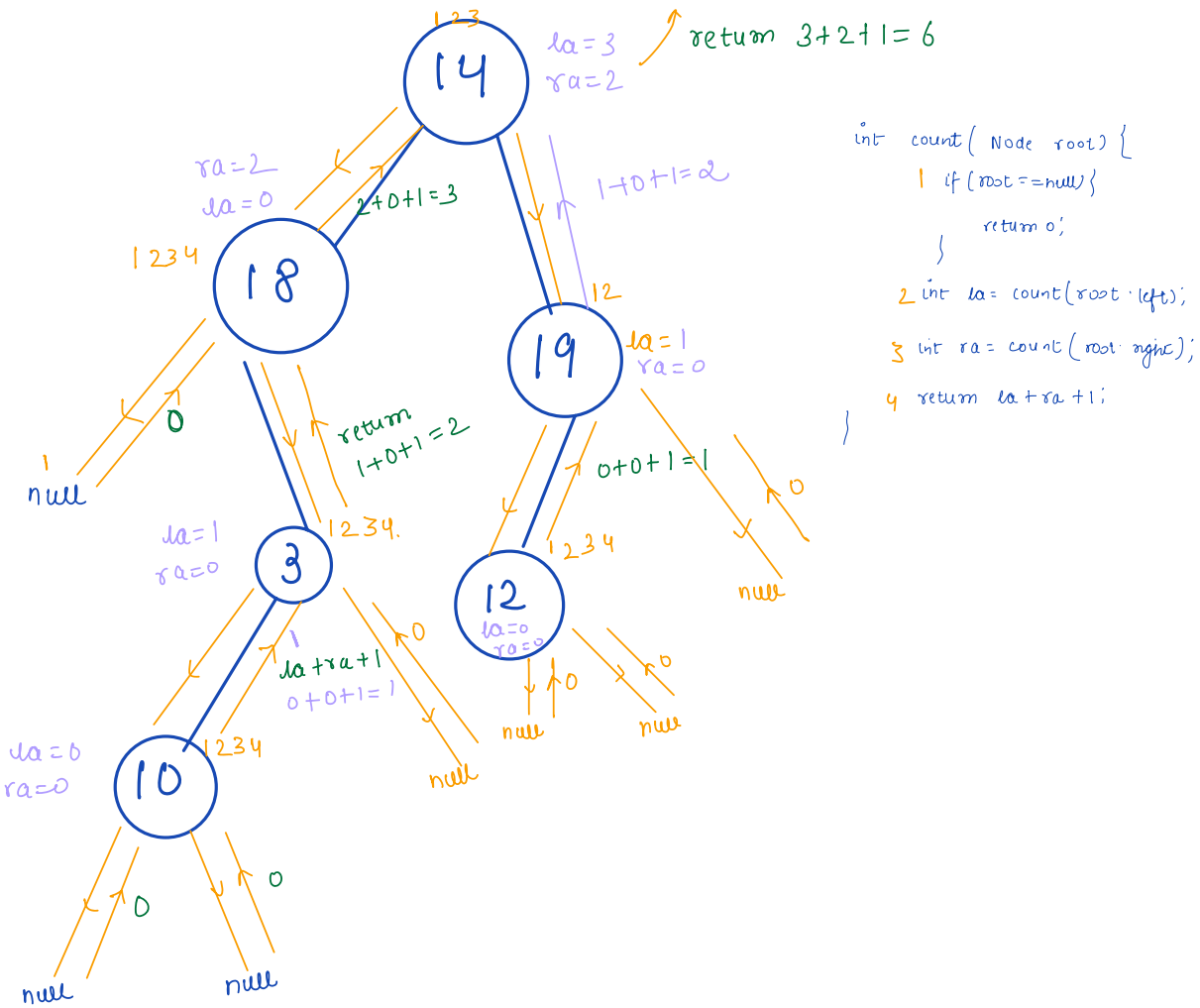


count (14)

18, 3, 10 — count (18) → la = 3

19, 12 — count (19) → ra = 2

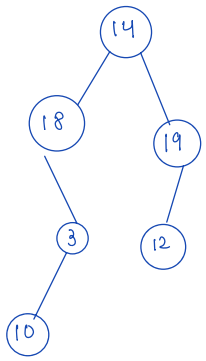
ans = la + ra + (1) → root data

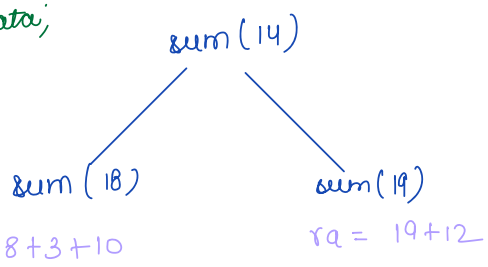



```
int count(Node root){
1  if(root==null){
       return 0;
   }
2  int la = count(root.left);
3  int ra = count(root.right);
4  return la+ra+1;
}
```

Qu find sum of all the nodes in a tree

```
int sum(Node root){
    if(root==null){
        return 0;
    }
    int la = sum(root.left);
    int ra = sum(root.right);
    return la+ra+root.data;
}
```





la = 18+3+10

ra = 19+12

ans = la+ra+14

Thankyou ☺

Practice Viva

- 30 mins
- 1 question to solve
  - Try to understand the problem
  - communicate
  - code (production level)