Agenda

i) Why trees ?

ii) Terms related to trees

iii) structure of trees

iv) Traversal (In, pre, post)

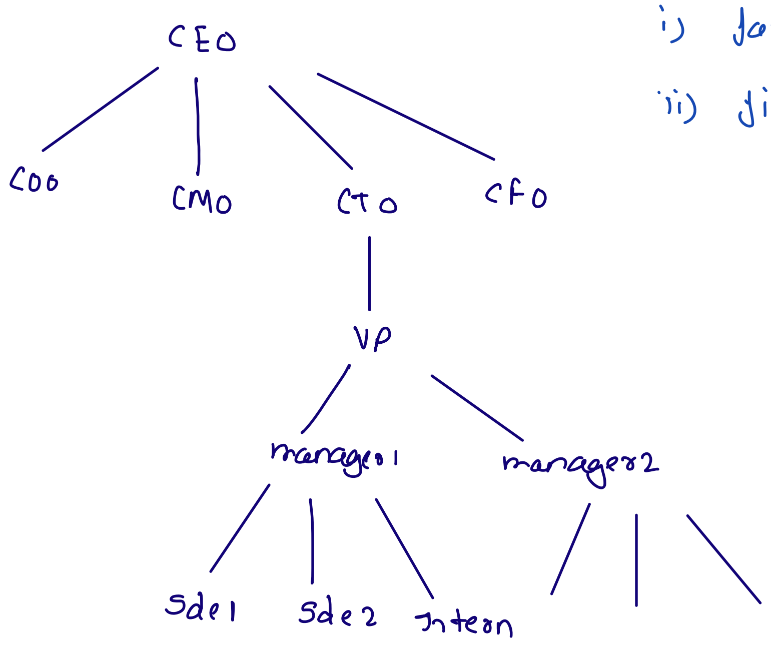
v) Questions
  → size, sum, height

Linear DS

{ Arrays , AL, Strings, Stacks , queue {continous memory allocation}

Linked List {discontinous memory allocation}

non-linear / hierarchical DS

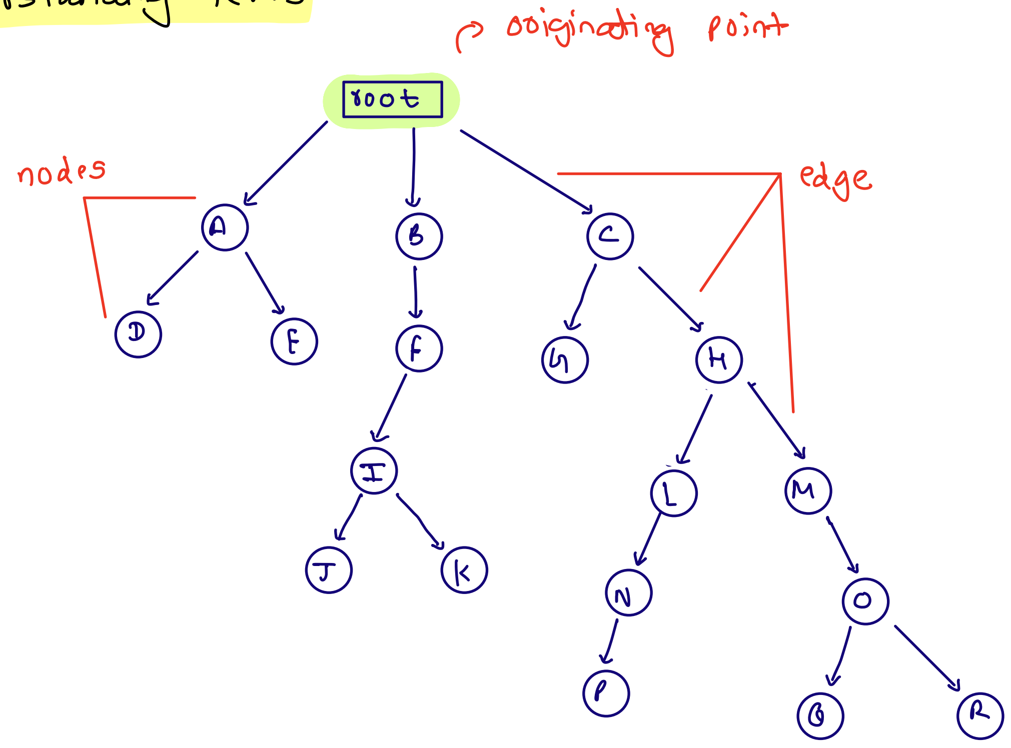


i) family tree

ii) file system

Tree

Understanding terms



i) parent – child : L is parent of H (false)

O is child of M (true)

each node has a single parent only (Except root)

ii) siblings : nodes having same parents.

iii) leaf nodes : nodes with 0 child

iv) Ancestor : nodes coming in the path from root to this node.

v) Descendant : all the nodes coming under this node.

1) Parent of F ⇒ B

2) Are N and O siblings ⇒ false

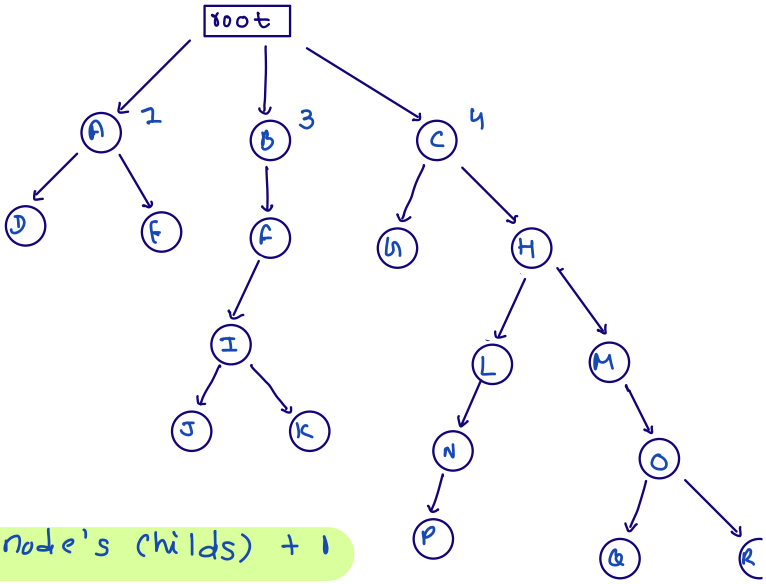
3) Are A, B, C siblings ⇒ yes

4) Ancestors of K ⇒ root, B, F, I

5) Descendants of B ⇒ F, I, J, K

i) height (node)

the max distance b/w node to any of its descendant leaf node.

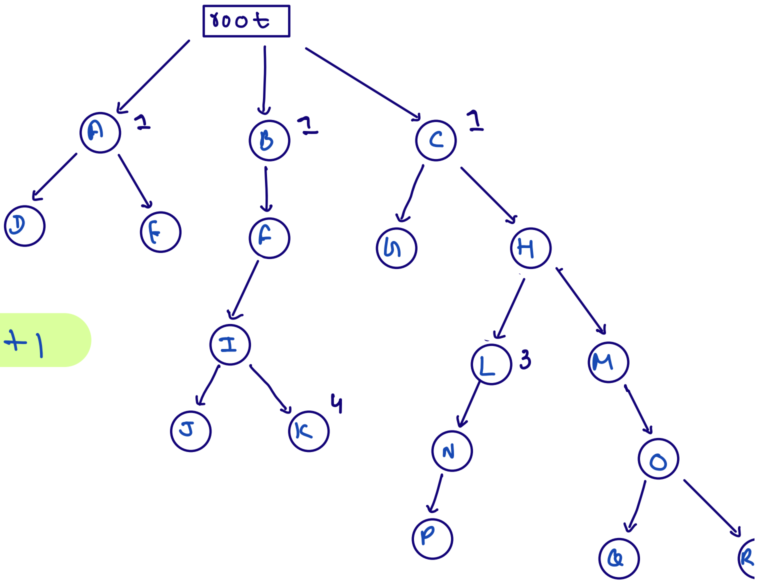


height (leaf node) = 0

height (node) = max (height of node's childs) + 1

ii) Depth (node)

distance of path from node to root.



depth (root) = 0

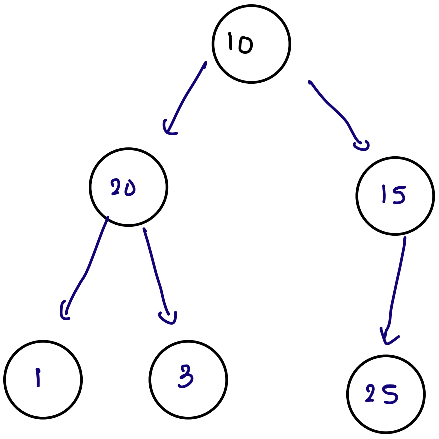
depth (node) = depth (parent) + 1

Structure of tree

Generic tree (N-ary tree)

→ structure of Binary tree

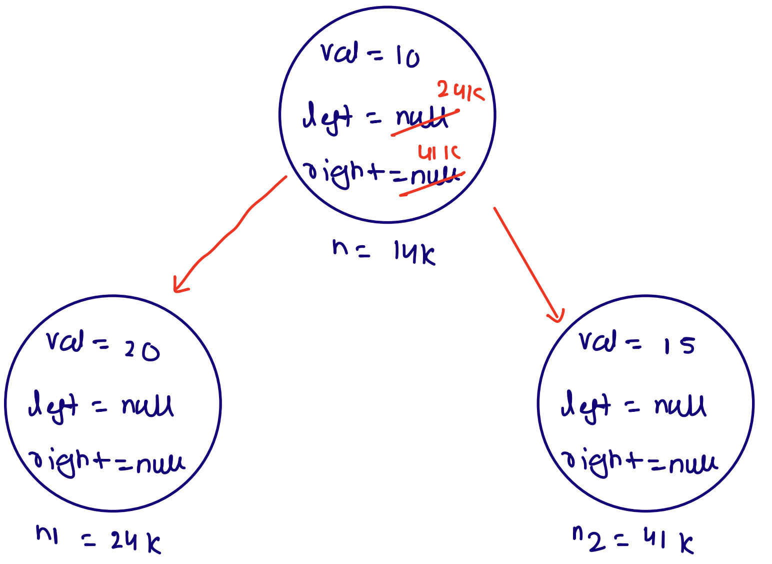
↳ In binary each node can have atmax 2 childs (0, 1, 2)



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

        Node (int val) {
              this.val = val;
        }
}
```

```
Node n = new Node(10);
Node n1 = new Node(20);
Node n2 = new Node(15);
```

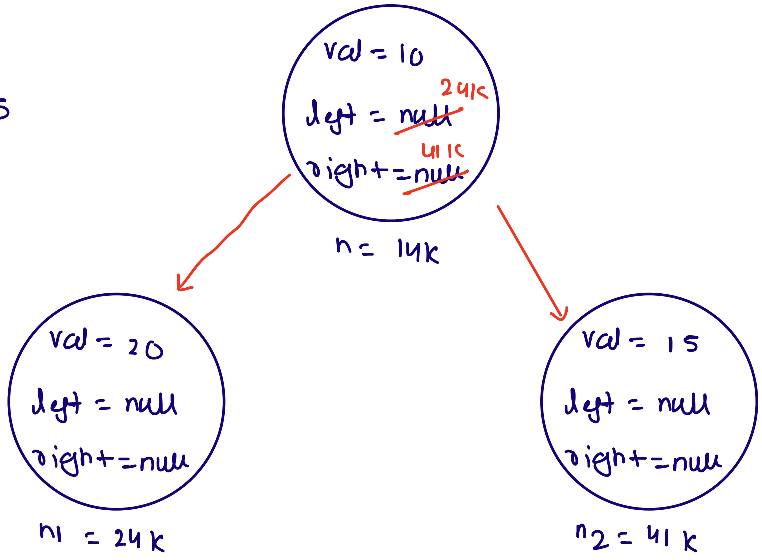
```
n.left = n1;
n.right = n2;
```



note: don't worry about creation of tree, you will be given the root of already created binary tree to solve question.

```
sopln (n.right.val);  → 15

sopln (n.right.left.val);
```

↳ null ptr Exception



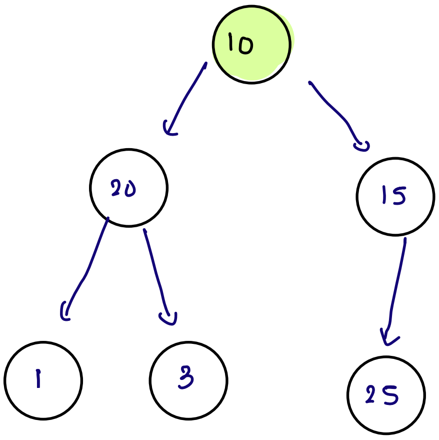
Traversal in Binary tree

traversal of BT

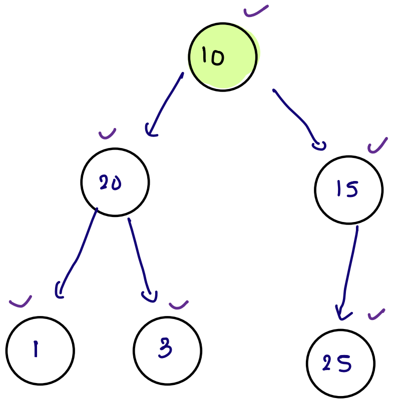
i) Iterative (tricky) { next class }

ii) Recursive (easy)

```
void traversal (Node node) {
      if (node = = null) {
            return;
      }

      traversal (node.left);

      traversal (node.right);
}
```

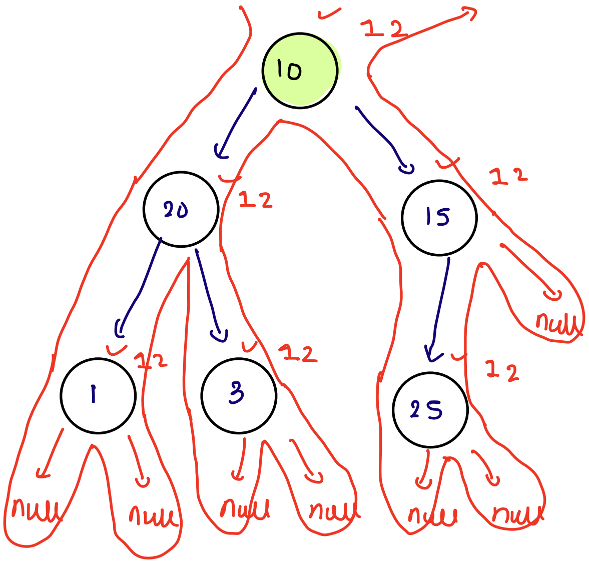


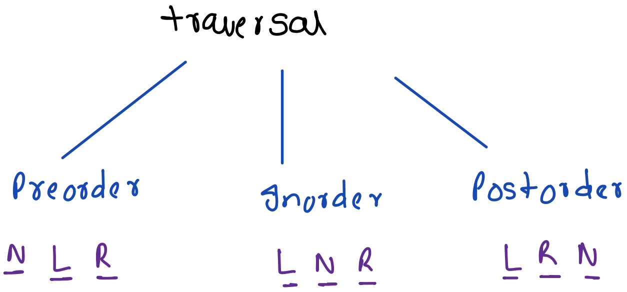
```
void traversal (Node node) {
      if (node = = null) {
            return;
      }
   1. traversal (node.left);

   2. traversal (node.right);
}
```



travel	~~node = null~~	~~if~~
travel	~~node = null~~	~~if~~
travel	~~node = null~~	~~if~~
travel	~~node = 25~~	~~1 2~~
travel	~~node = 15~~	~~1 2~~
travel	~~node = null~~	~~if~~
travel	~~node = null~~	~~if~~
travel	~~node = 3~~	~~1 2~~
travel	~~node = null~~	~~if~~
travel	~~node = null~~	~~if~~
travel	~~node = 1~~	~~1 2~~
travel	~~node = 20~~	~~1 2~~
~~travel~~	~~node = 10~~	~~1 2~~

```
void  traversal (Node  node) {
      if (node = = null) {
            return;
      }

   1. traversal (node.left);

   2. traversal (node.right);

}
```




traversal

Preorder

N L R

Inorder

L N R

Postorder

L R N

```
void traversal (Node node) {
    if (node = = null) {
        return;
    }
    sopln(node.val);
    traversal (node.left);
    traversal(node.right);
}
```

→ preorder

```
void traversal (Node node) {
    if (node = = null) {
        return;
    }
    traversal (node.left);
    sopln(node.val);
    traversal(node.right);
}
```

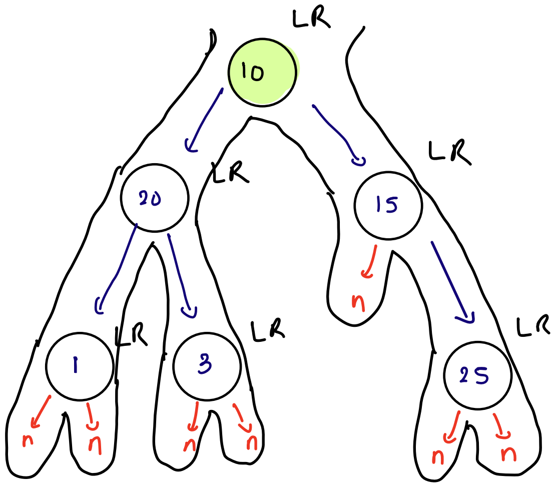
→ inorder

```
void traversal (Node node) {
    if (node = = null) {
        return;
    }
    traversal (node.left);
    traversal(node.right);
    sopln(node.val);
}
```

Postorder

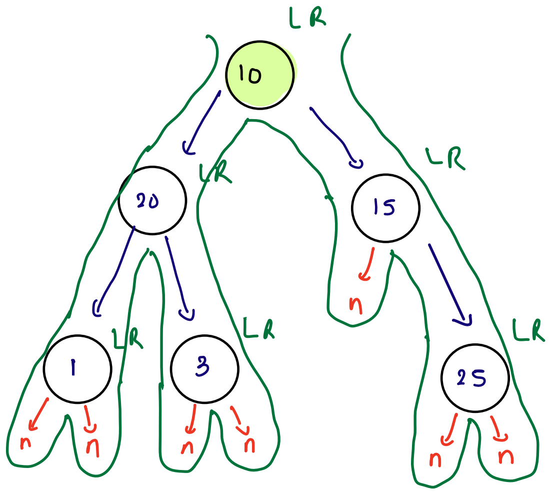
Dry run

```
void traversal (Node node) {
    if (node = = null) {
        return;
    }
    sopln(node.val);
    traversal (node.left);
    traversal(node.right);
}
```



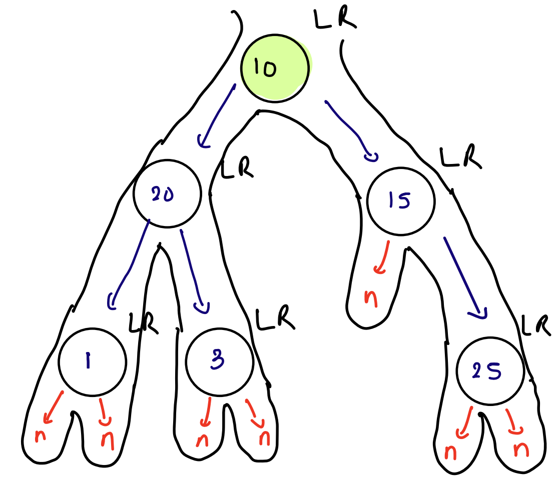
preorder : 10 20 1 3 15 25

```
void traversal (Node node) {
    if (node = = null) {
        return;
    }
    traversal (node.left);
    sopln(node.val);
    traversal(node.right);
}
```

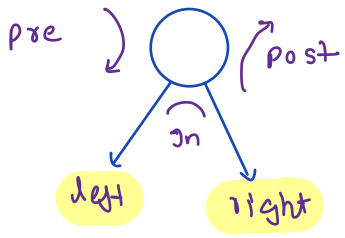


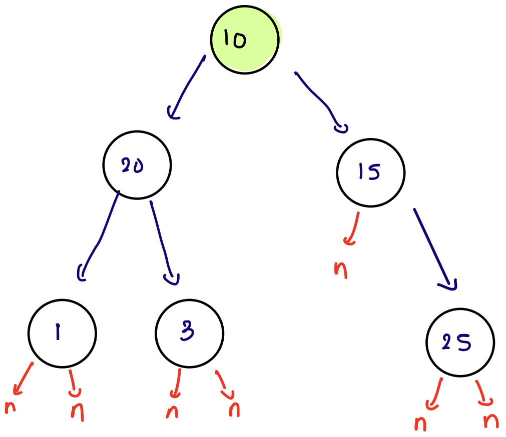
Inorder : 1 20 3 10 15 25

```
void  traversal (Node  node) {
      if (node = = null) {
            return;
      }
      traversal (node.left);
      traversal (node.right);
      sopln(node.val);
}
```



postorder : 1 3 20 25 15 10



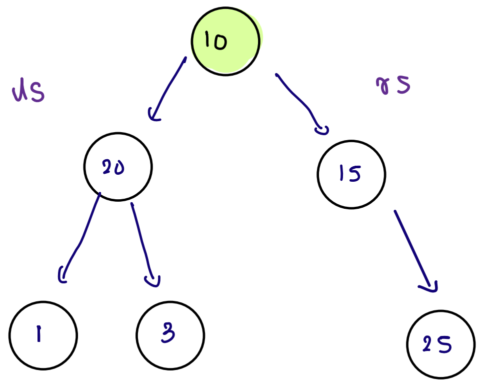


Preorder : 10 20 1 3 15 25

Inorder : 1 20 3 10 15 25

Postorder : 1 3 20 25 15 10

Q.1 Given root of binary tree, find its size (count of nodes)



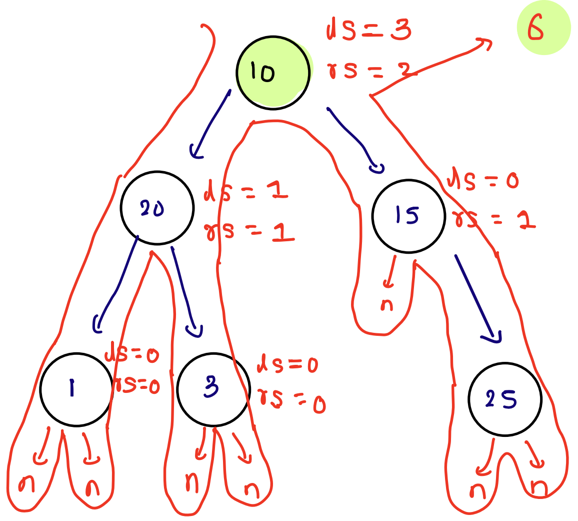
ans= 6

Idea1: create a global count variable and inc. it by one every time you hit a node in traversal function.

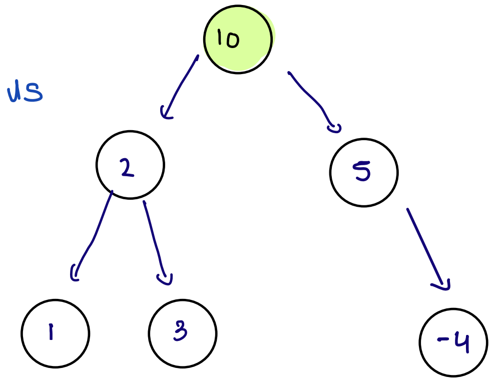
↳ (TODO)

Idea2: get left size, get right size and return your answer.

```
int   size ( Node  node) {
   if (node==null) {
        return 0;
   }

   int ls  =  size(node.left);
   int rs  =  size (node. right);
   return ls+rs+1;

}
```



Q.2 Given root of binary tree, find sum of all nodes.

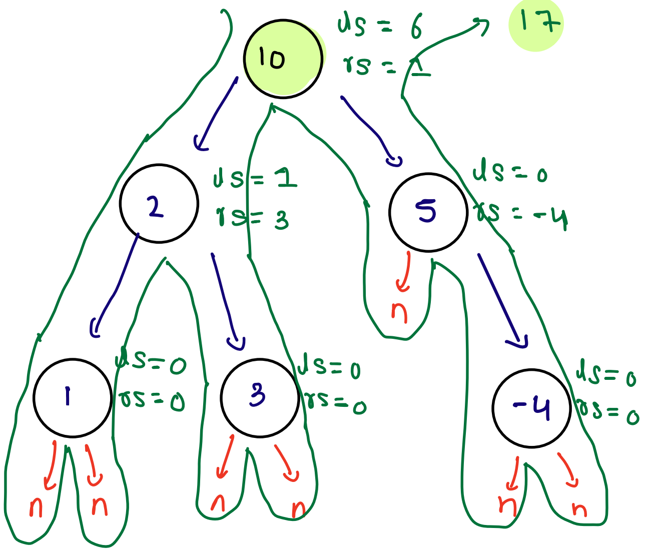


sum = 10 +2 + 5+ 1+3+ -4

= 17

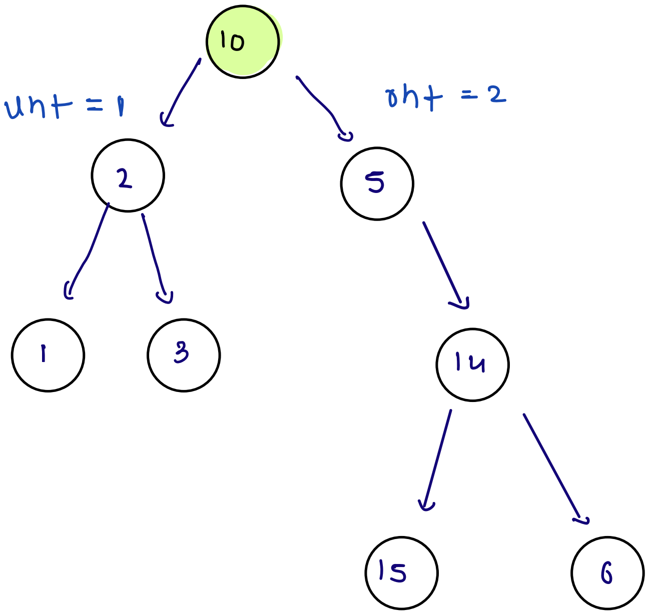
```
int  sum (Node node )  {
      if (node = =null) {
          return 0;
      }

      int ls = sum (node.left);
      int rs = sum (node. right);
      return ls + rs + node.val;
}
```

```
int  sum (Node node)  {
     if(node==null) {
          return 0;
     }

     int ls = sum(node.left);

     int rs = sum (node.right);

     return ls+rs+node.val;
}
```



Q.3 Given root of BT, find height of binary tree.
↳ (edge based distance)

height of tree = height (root)

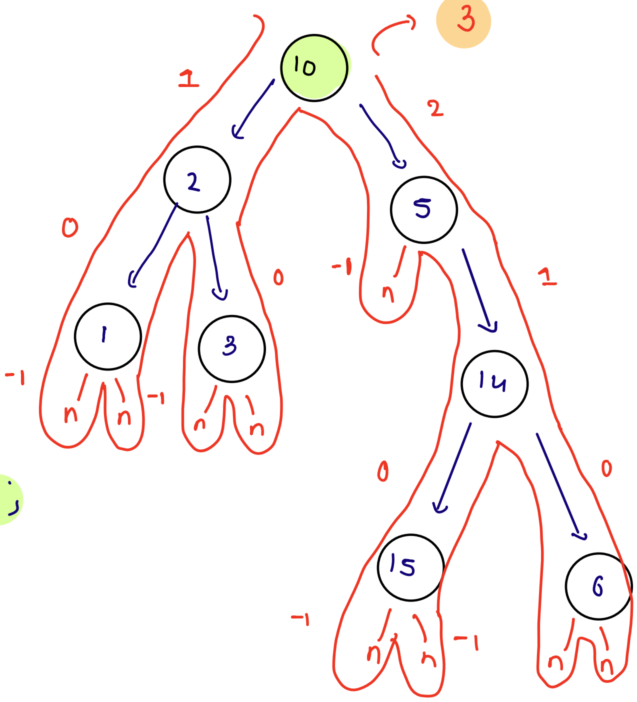


ht = 3

ht = max (lht, rht) +1

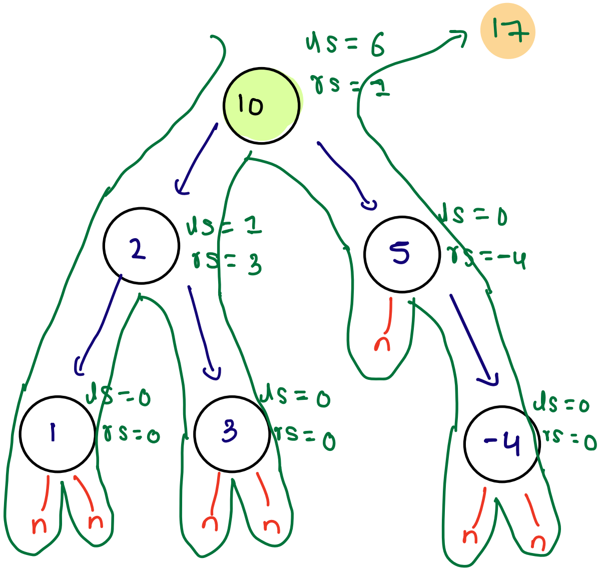
return ht;

```
int  height (Node root) {
    if(node = = null ) {
        return -1;
    }
    int lht = height (root.left);

    int rht = height (root.right);

    return Math.max (lht, rht) + 1;
}
```



Doubts

```
int  sum(Node node) {
    if(node= =null) {
        return 0;
    }

    int ls = sum(node.left);
    int rs = sum (node. right);

    return ls+rs+node.val;
}
```



10
ls = 6
rs = 1
17
2
ls = 1
rs = 3
5
ls = 0
rs = -4
1
ls = 0
rs = 0
3
ls = 0
rs = 0
-4
ls = 0
rs = 0
n
n
n
n
n
n
n