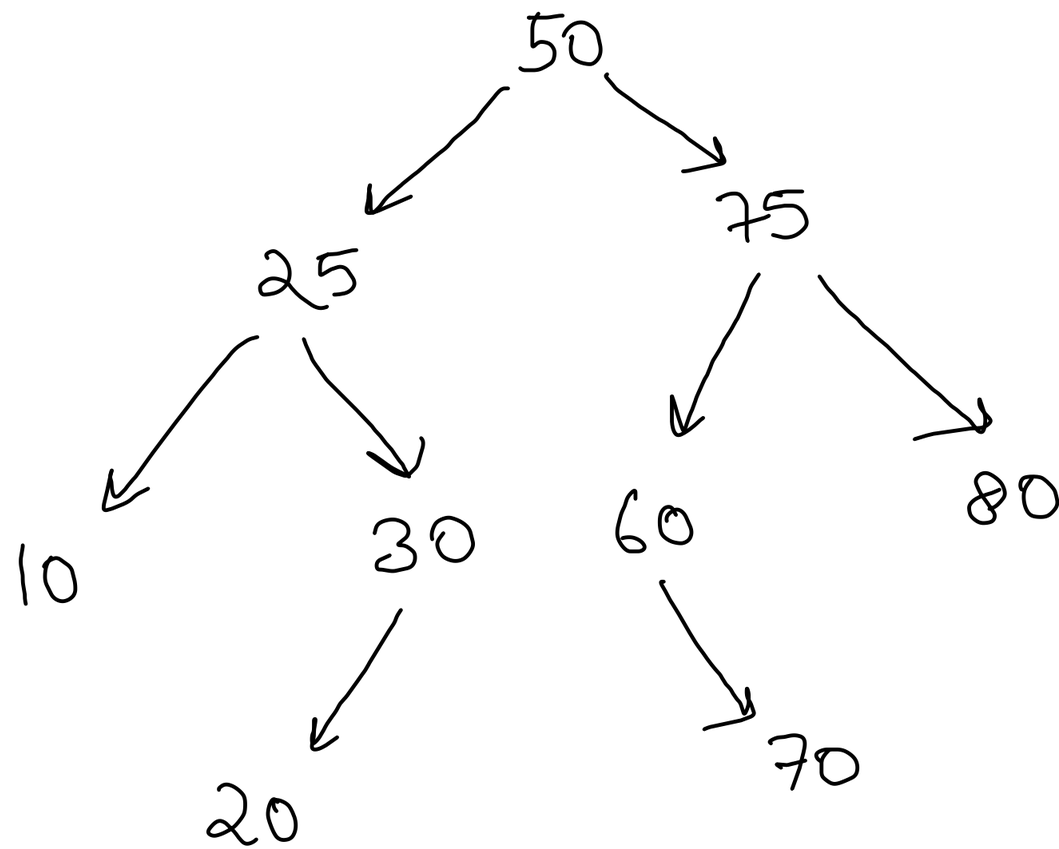
Binary Tree

Morning questions

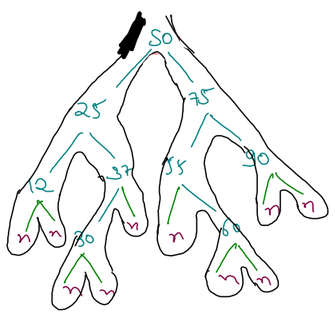
- Binary Tree - Introduction And Data Members
- Binary Tree - Constructor
- Display A Binary Tree
- Size, Sum, Maximum And Height Of A Binary Tree
- Traversals In A Binary Tree
- Levelorder Traversal Of Binary Tree
- Iterative Pre, Post And Inorder Traversals Of Binary Tree
- Find And Nodetorootpath In Binary Tree


```
class Node {

    int data;
    Node left;
    Node right;

    Node(int data)
      { this.data = data; }

}
```



Construction

50
25
12
null
null
37
30
null
null
75
55
null
60
null
null
90
null
null




```
Stack<Pair> stk = new Stack<>();
Node root = new Node(arr[0]);
stk.push(new Pair(root, -1));
int idx = 0;

while(!stk.isEmpty()){

    Pair par = stk.peek();

    if(par.state == -1){
        // preorder
        idx++;

        if(arr[idx] != null){
            Node child = new Node(arr[idx]);
            par.node.left = child;
            stk.push(new Pair(child, -1));
        }
        par.state++;

    } else if(par.state == 0){
        // inorder
        idx++;

        if(arr[idx] != null){
            Node child = new Node(arr[idx]);
            par.node.right = child;
            stk.push(new Pair(child, -1));
        }
        par.state++;

    } else if(par.state == 1){
        // postorder
        stk.pop();
    }
}
```

Display a Binary Tree

25 ← 50 → 75

Left

12 ← 25 → 37

. ← 12 → .

30 ← 37 → .

. ← 30 → .

right

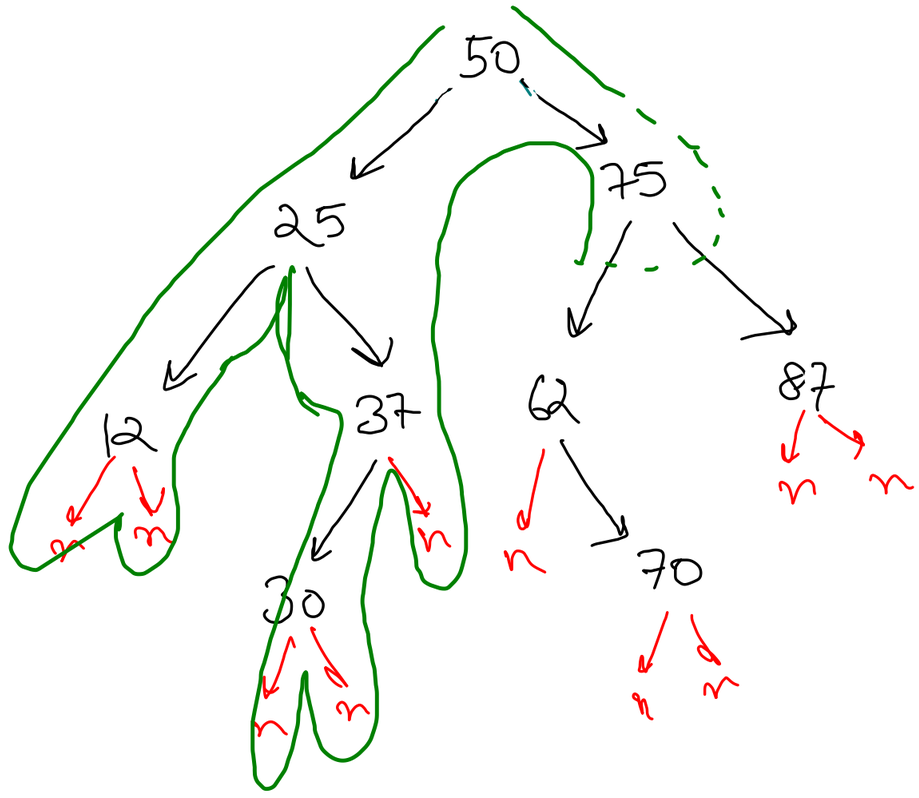
62 ← 75 → 87

. ← 62 → 70

. ← 70 → .

. ← 87 → .

```java
public static void print(Node node){

  if(node.left != null)
      System.out.print(node.left.data);
  else System.out.print(".");

  System.out.print(" <- " + node.data + " -> ");

  if(node.right != null)
      System.out.print(node.right.data);
  else System.out.print(".");

  System.out.println();
}

public static void display(Node node) {
    if(node == null) return;

  1 print(node);

  2 // preorder
    display(node.left);
    // inorder
  3 display(node.right);
    // postorder
}
```

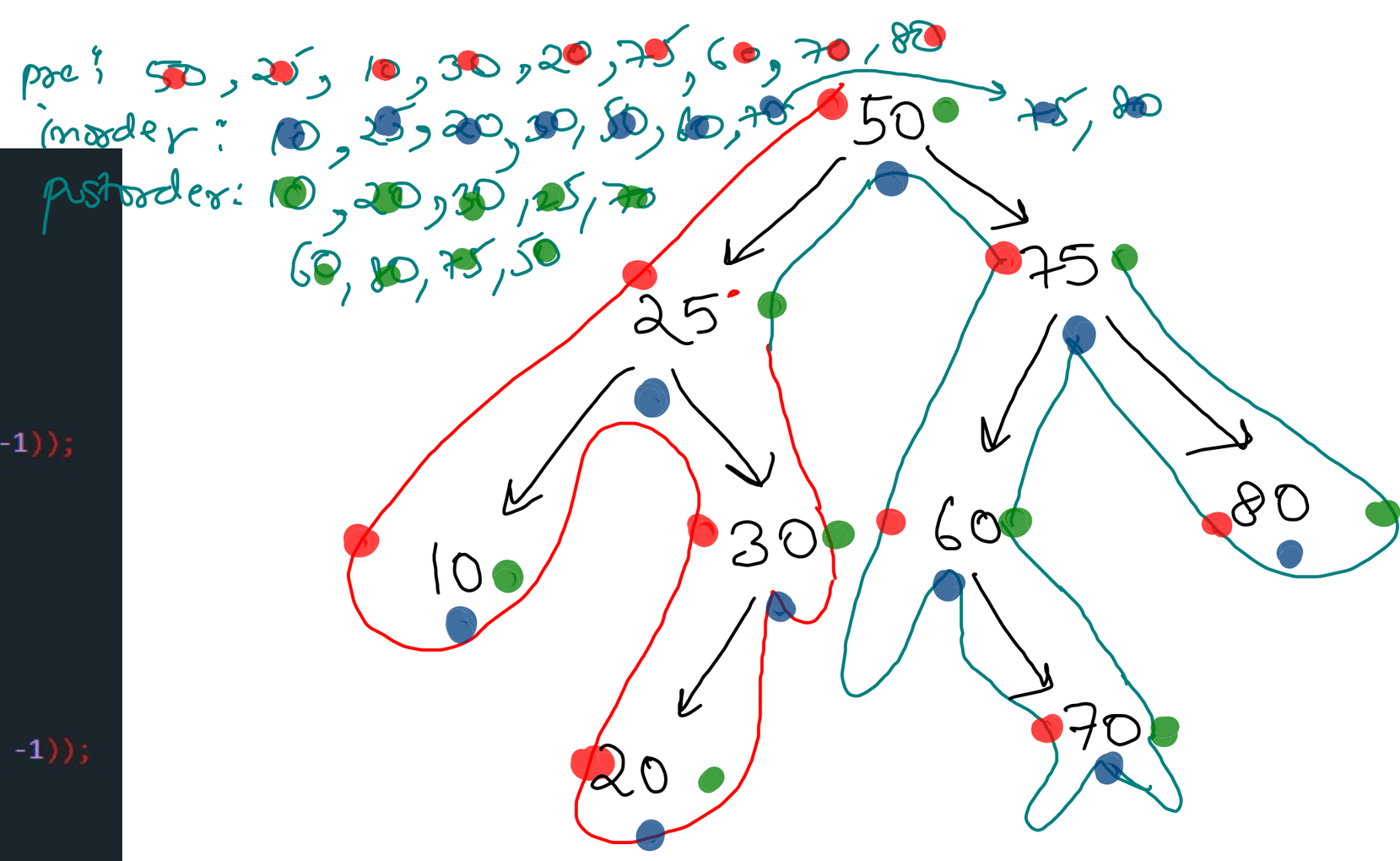



```
if (root == null) return;
print(root);
display(root.left);
display(root.right);
```

Itrative

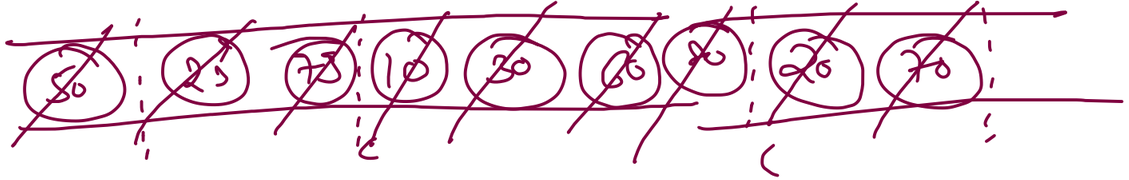
```
while(!stk.isEmpty()){
    Pair par = stk.peek();

    if(par.state == -1){
        // preorder
        preorder.add(par.node.data);

        if(par.node.left != null){
            stk.push(new Pair(par.node.left, -1));
        }
        par.state++;

    } else if(par.state == 0){
        // inorder
        inorder.add(par.node.data);

        if(par.node.right != null){
            stk.push(new Pair(par.node.right, -1));
        }
        par.state++;

    } else if(par.state == 1){
        // postorder
        postorder.add(par.node.data);
        stk.pop();
    }
}
```

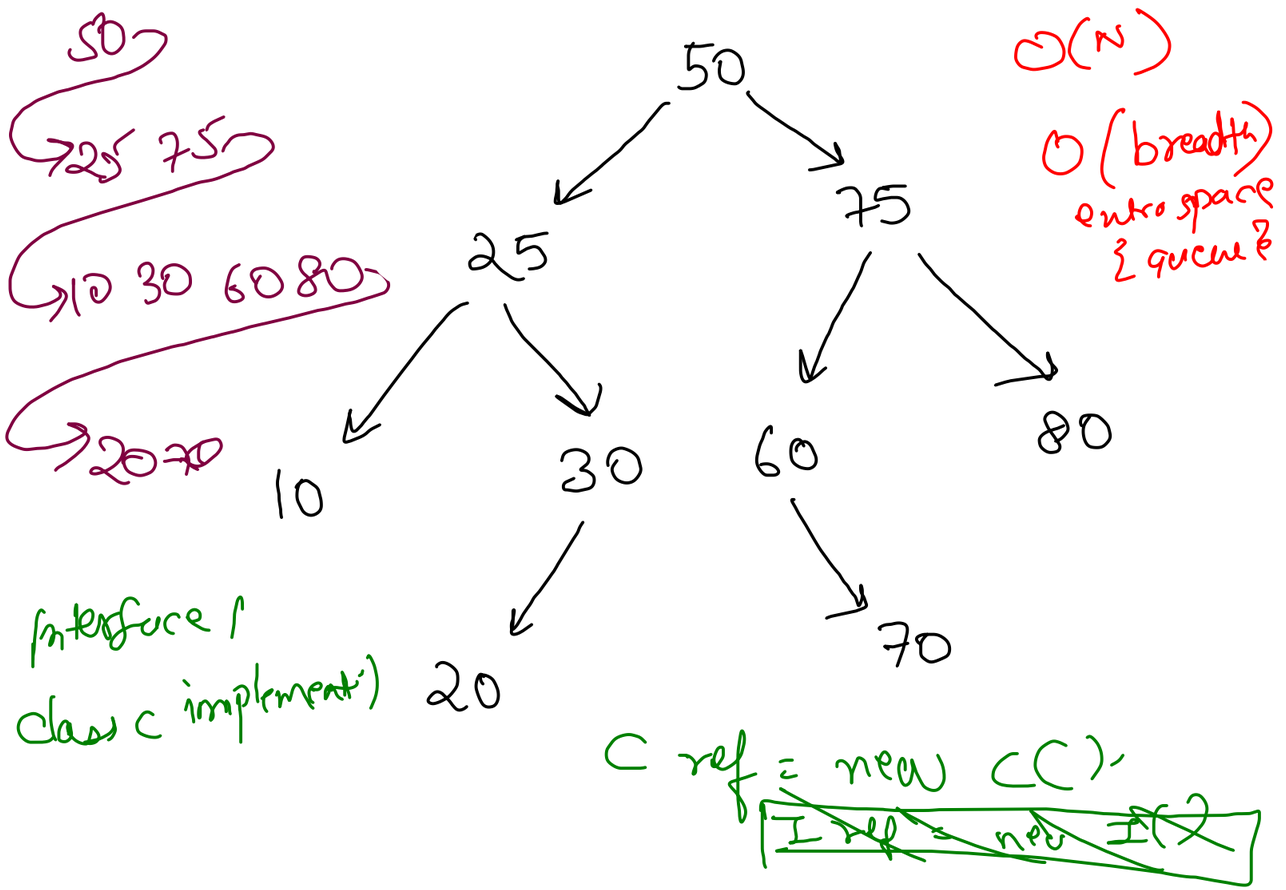
You are screen shari



level Order linewise

```java
Queue<Node> q = new ArrayDeque<>();
q.add(node);

while(q.size() > 0){

   int counter = q.size();
   for(int i=0; i<counter; i++){

       Node par = q.remove();
       System.out.print(par.data + " ");

       if(par.left != null)
           q.add(par.left);

       if(par.right != null)
           q.add(par.right);

   }
  System.out.println();
}
```





interface I
class C implements I

C ref = new C();
~~I ref = new I();~~

Queue

remove first, add last

LL

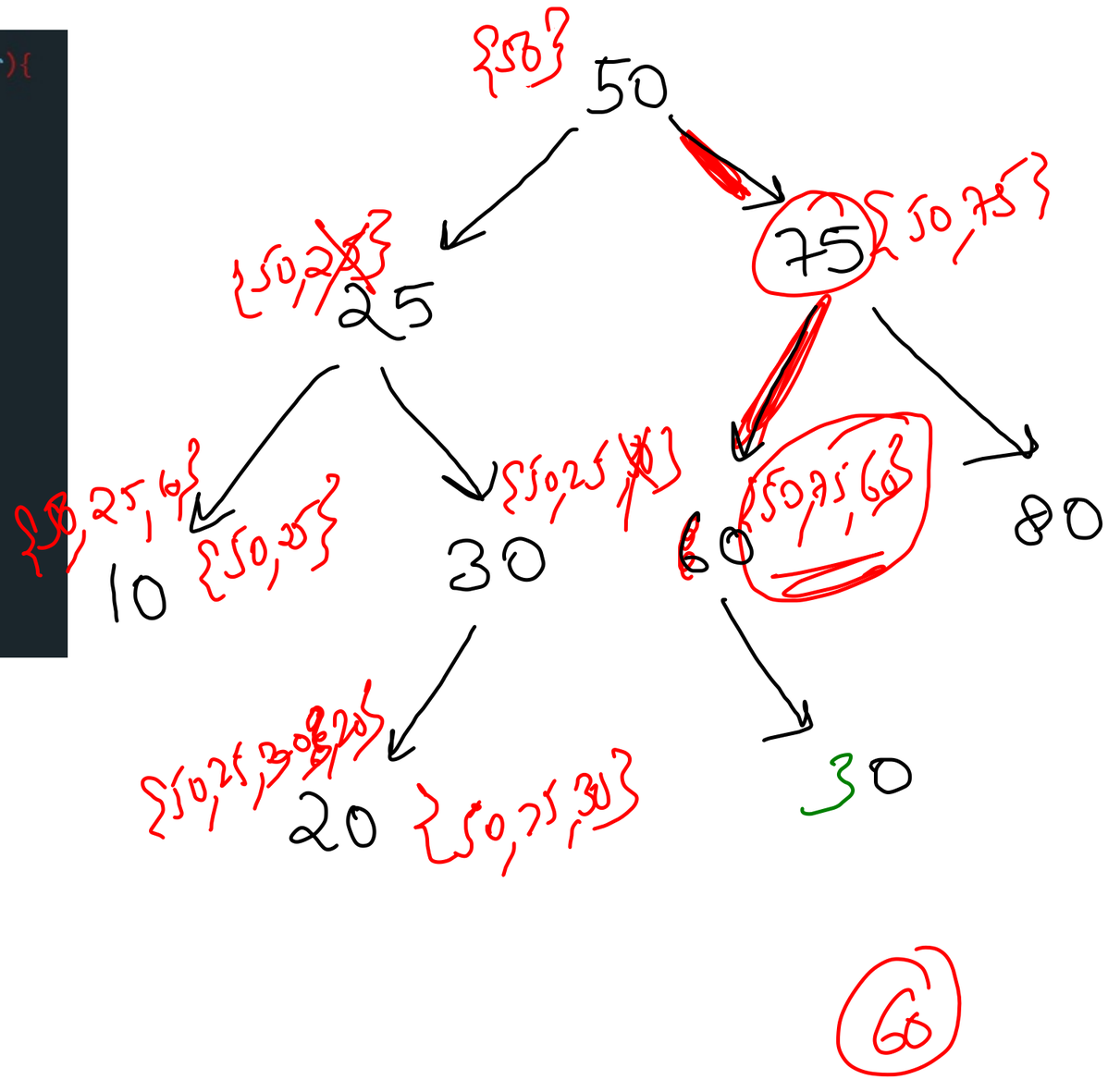
rF, aF
rL, aL

Queue ref = new LL();
LL ref2 = new LL();

2

```java
public static boolean nodeToRootPath(Node node, int data, ArrayList<Integer> curr){
    if(node == null) // negative base case
        return false;

    if(node.data == data){ // positive base case
        curr.add(node.data);
        return true;
    }

    curr.add(node.data);
    boolean left = nodeToRootPath(node.left, data, curr);
    if(left == true) return true;

    boolean right = nodeToRootPath(node.right, data, curr);
    if(right == true) return true;

    curr.remove(curr.size() - 1);
    return false;
}
```

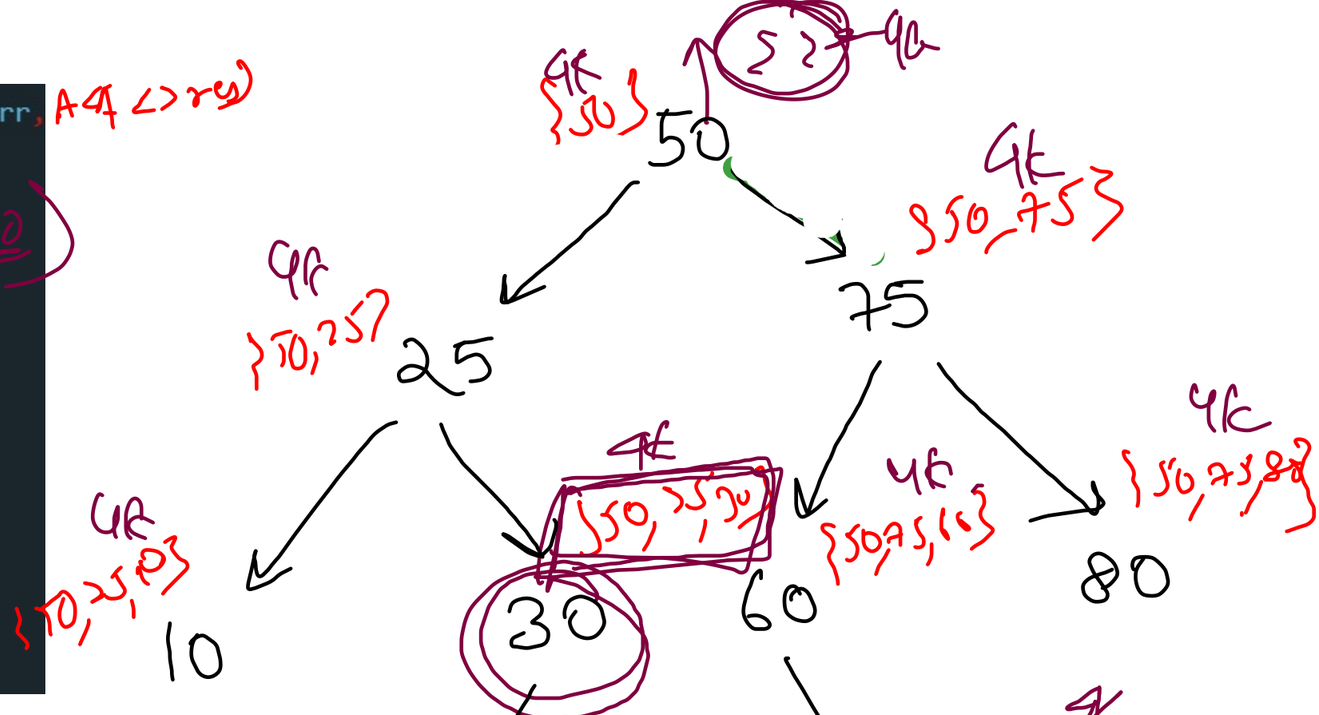
Root to Node Path
{ Backtracking }

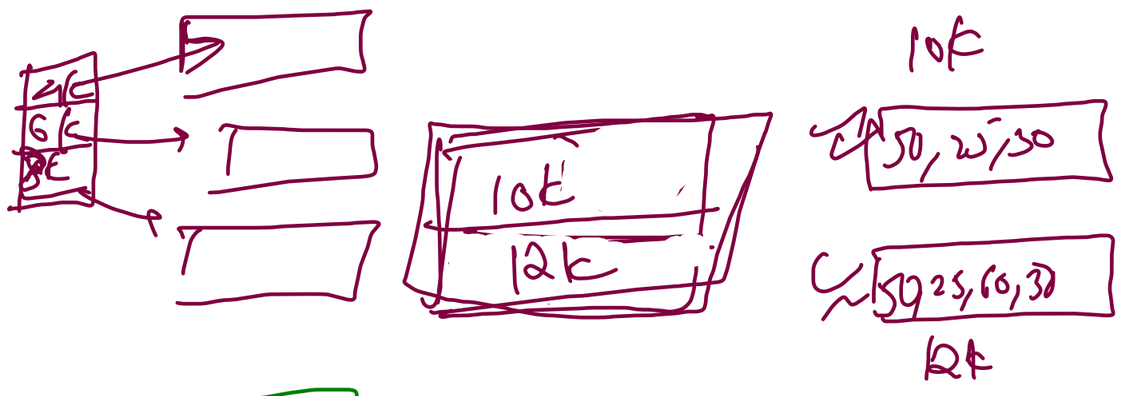


```java
public static void rootToNodePath(Node node, int data, ArrayList<Integer> curr, ArrayList<ArrayList<Integer>> res){
    if(node == null) // negative base case
        return;

    curr.add(node.data);

    if(node.data == data){ // positive base case
        ArrayList<Integer> temp = new ArrayList<>();
        for(Integer i: curr)
            temp.add(i);
        res.add(temp);
    }

    rootToNodePath(node.left, data, curr, res);
    rootToNodePath(node.right, data, curr, res);
    curr.remove(curr.size() - 1);
}
```

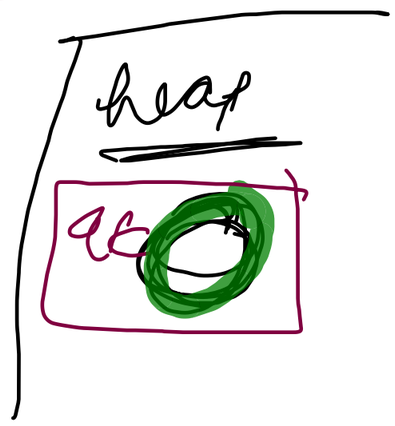
res.add(curr)

O(h)



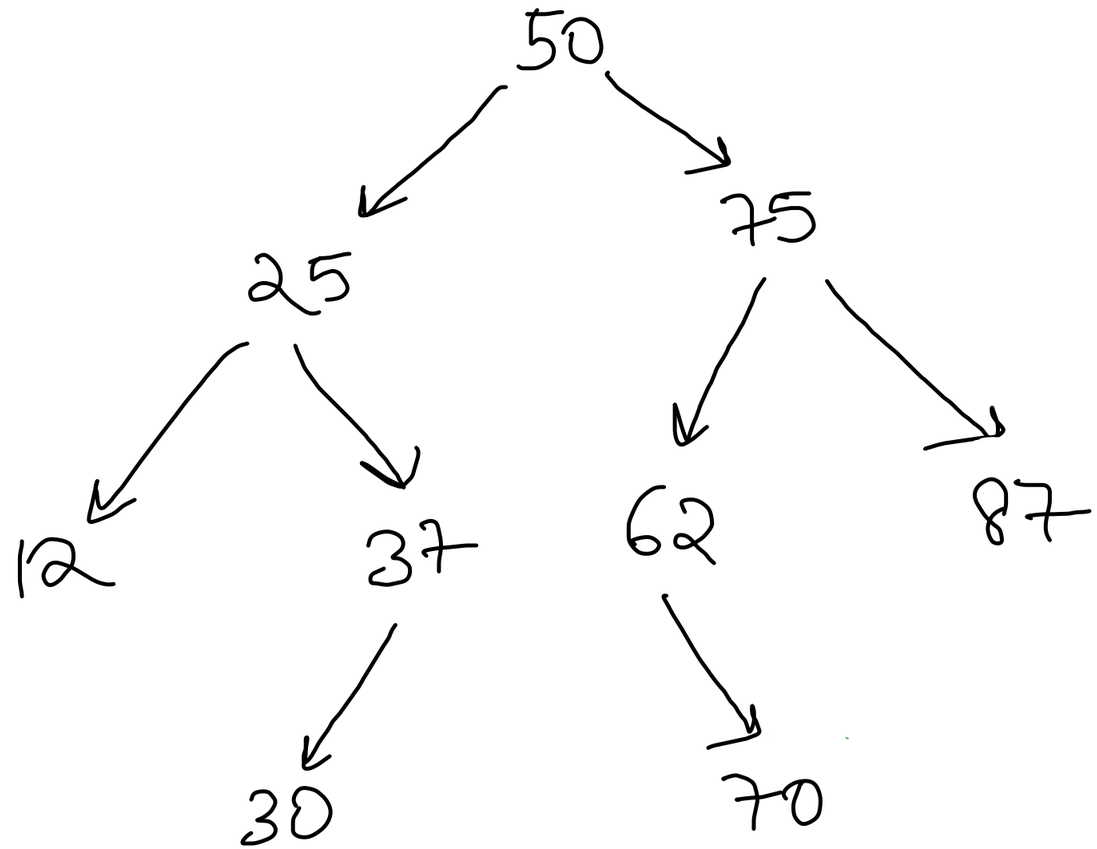


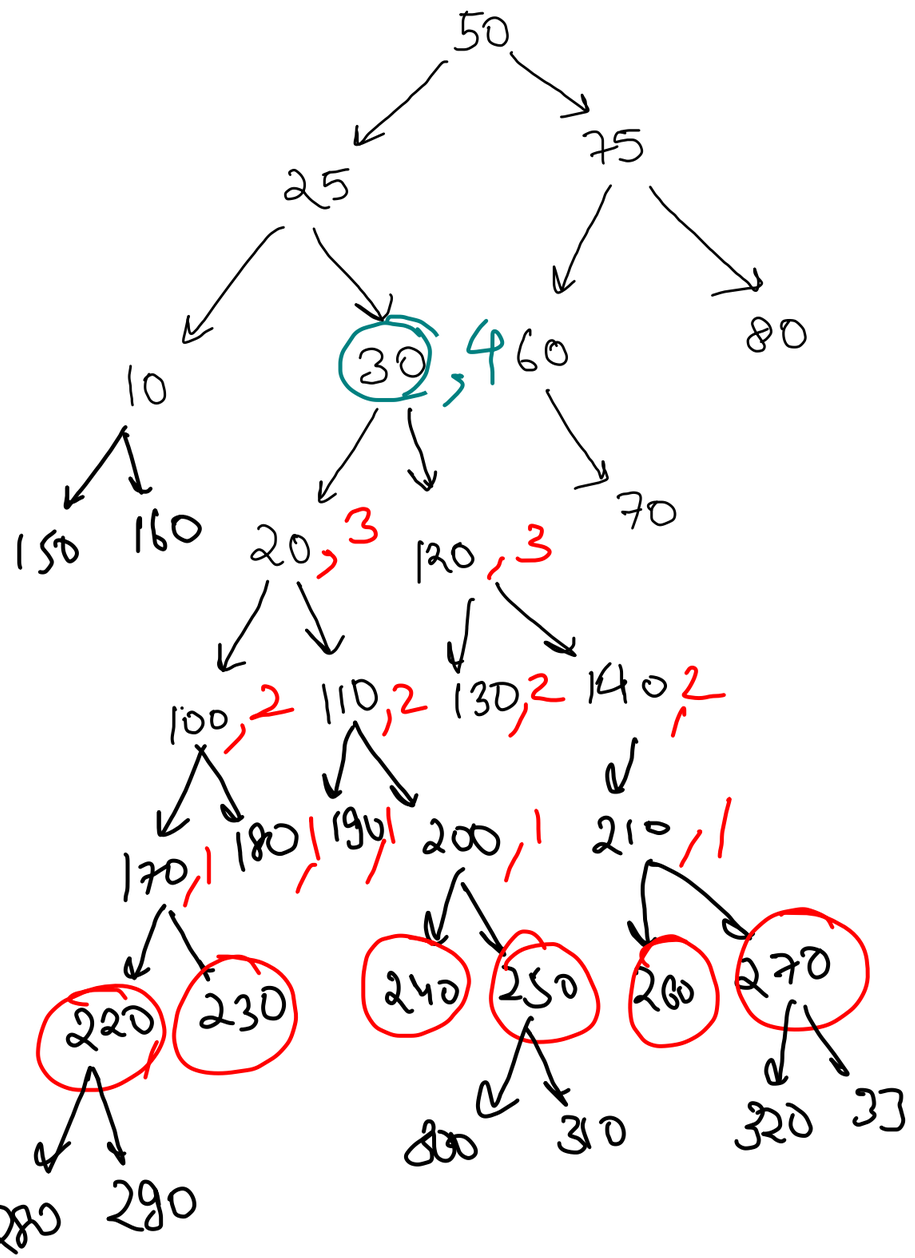
All root to node paths



Binary Tree Evening class

1. Print K Levels Down
2. Print Nodes K Distance Away
3. Path To Leaf From Root In Range
4. Transform To Left-cloned Tree
5. Transform To Normal From Left-cloned Tree
6. Print Single Child Nodes
7. Remove Leaves In Binary Tree





k levels down

30, k = 4

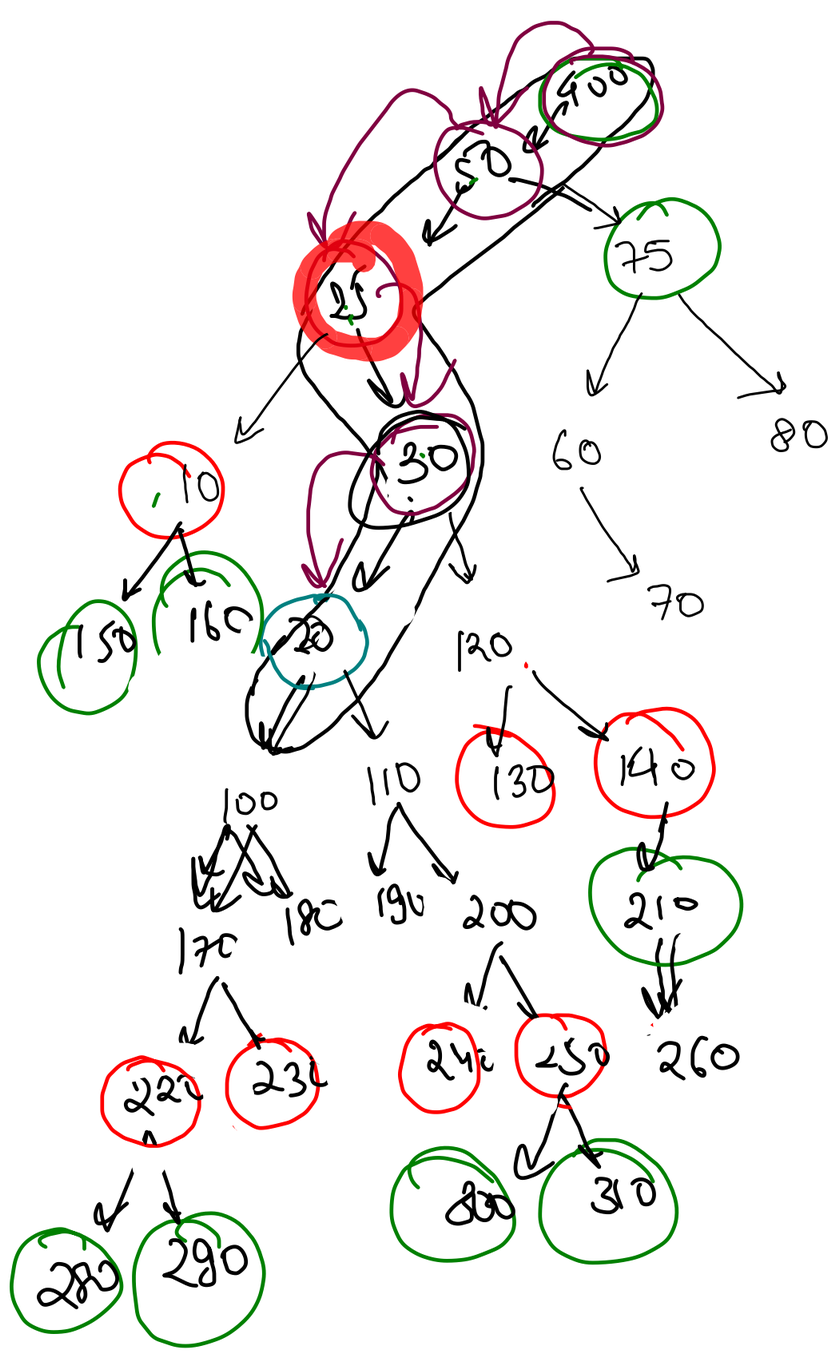
```
public static void printKLevelsDown(Node node, int k){
  if(node == null) return;
  if(k == 0){
      System.out.println(node.data);
      return;
  }

  printKLevelsDown(node.left, k - 1);
  printKLevelsDown(node.right, k - 1);
}
```

SC = I/P + Aux + O/P

I/P → O(N)

Aux → DS → O(1), Recursion → O(k)

O/P → O(1)



Amazon

Nodes k Distance Away

20, 3

Nodetorootpath(root, node)

{20, 30, 25, 50, 400}

3 2 1 0 -1

Kleveldown(20, 3, null)

Klevel down(30, 2, 20)

Klevel down(25, 1, 30)

Klevel down(50, 0, 25)

Klevel down(400, -1, 50)

```java
public static void printKNodesFar(Node node, int data, int k) {
  ArrayList<Node> n2rpath = nodeToRootPath(node, data); // o(n)

  int distance = k;
  for(int i=0; i<n2rpath.size(); i++){ // O(k)
     if(distance < 0) break;
     Node blockage = (i == 0) ? null : n2rpath.get(i - 1);
     printKLevelsDown(n2rpath.get(i), distance, blockage); // O(n)
     distance--;
  }

  // O(n + k*n) = O(n*k)
}
```

Path to leaf from root in range

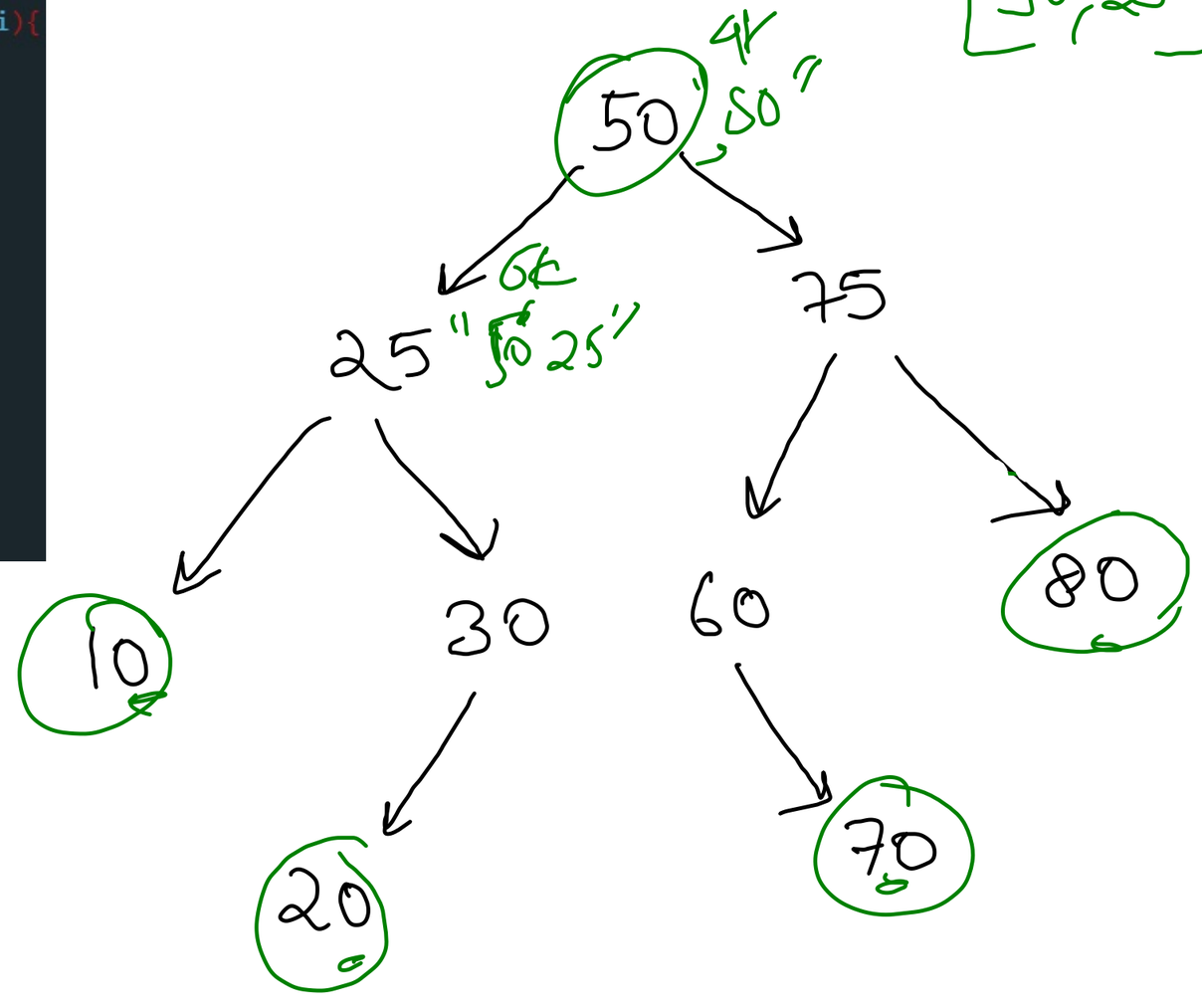
[low, high]

```java
public static void pathToLeafFromRoot(Node node, String path, int sum, int lo, int hi){
    if(node == null) return;

    sum += node.data;
    path = path + node.data + " ";

    if(node.left == null && node.right == null){
        // leaf node
        if(sum >= lo && sum <= hi)
          System.out.println(path);

        return;
    }

    pathToLeafFromRoot(node.left, path, sum, lo, hi);
    pathToLeafFromRoot(node.right, path, sum, lo, hi);
}
```

path -= " "

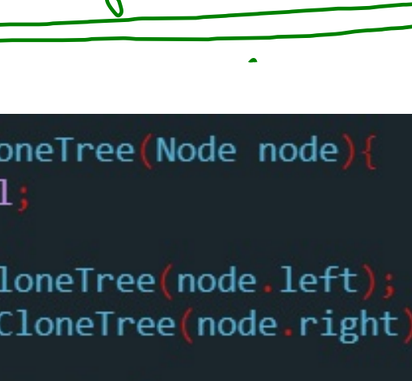
sum -= node.data;

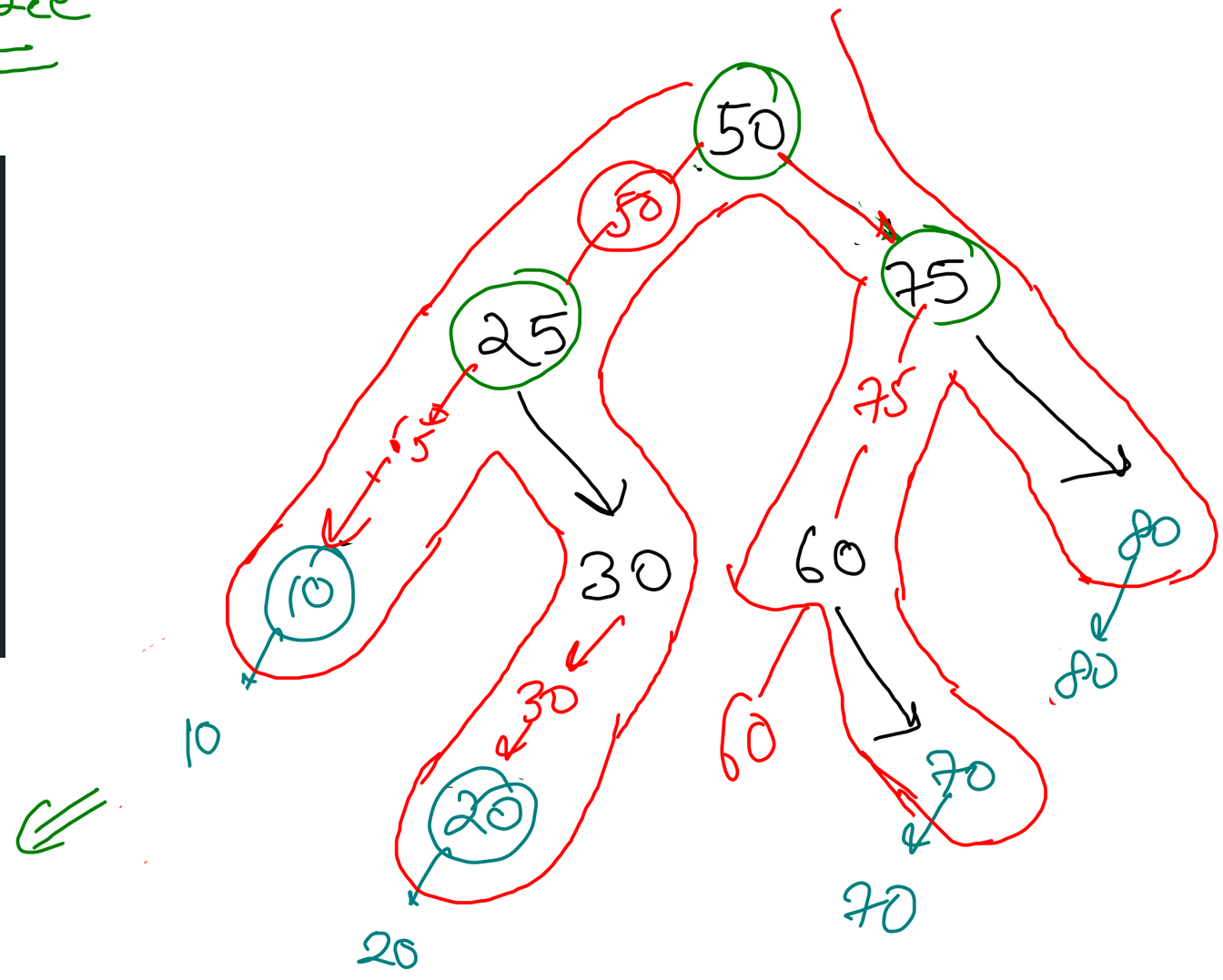
[90, 230]



Transform to Left-cloned Tree

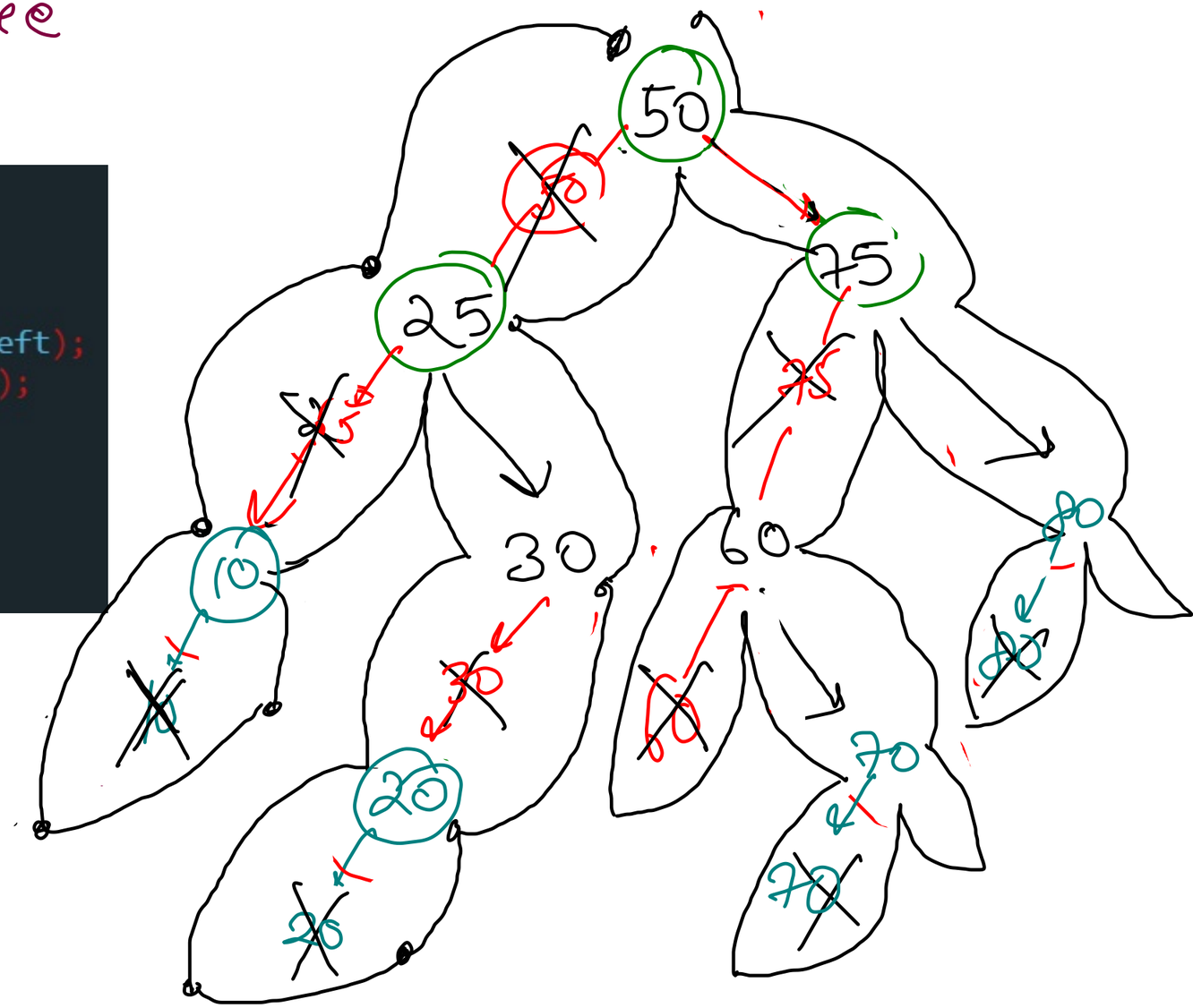
```java
public static Node createLeftCloneTree(Node node){
    if(node == null) return null;

    Node leftRoot = createLeftCloneTree(node.left);
    Node rightRoot = createLeftCloneTree(node.right);

    Node copyNode = new Node(node.data);
    copyNode.left = leftRoot;
    node.left = copyNode;

    return node;
}
```





Transform back to Normal tree

```java
public static Node transBackFromLeftClonedTree(Node node){
    if(node == null) return null;

    // faith
    Node leftRoot = transBackFromLeftClonedTree(node.left.left);
    Node rightRoot = transBackFromLeftClonedTree(node.right);

    // meeting expectation -> delete our duplicate
    node.left = leftRoot;
    return node;
}
```



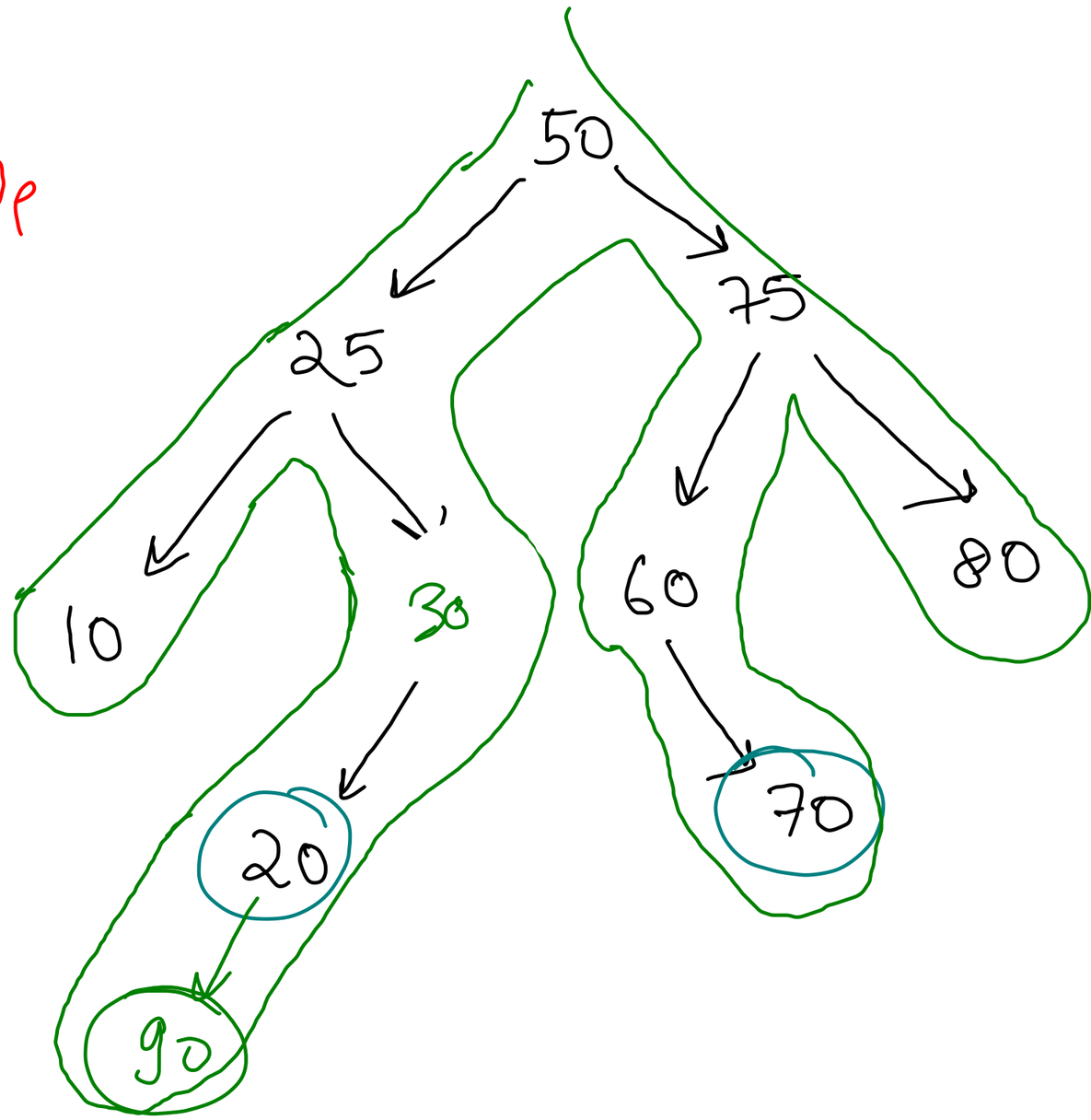
Print Single Child Nodes

```java
public static void printSingleChildNodes(Node node){
    if(node == null) return;
    if(node.left == null && node.right == null){
        // leaf node
        return;
    }

    if(node.left == null){
        System.out.println(node.right.data);
    }

    if(node.right == null){
        System.out.println(node.left.data);
    }

    printSingleChildNodes(node.left);
    printSingleChildNodes(node.right);
}
```

{ children wale node ke child ko print

```java
public static void printSingleChildNodes(Node node, Node parent){
    if(node == null) return;

    if(parent != null && parent.left == null){
        System.out.println(node.data);
    }

    if(parent != null && parent.right == null){
        System.out.println(node.data);
    }

    printSingleChildNodes(node.left, node);
    printSingleChildNodes(node.right, node);
}
```

{ children wale node ke child pe lai



Binary Tree lecture-3

(1) Remove leaves
- → without return type
- → with return type.

(2) Tilt of Binary Tree

(3) Diameter of Binay Tree
- → $O(N^2)$ approach
- → $O(N)$ approach
  - → static variable
  - → Pair class.

(4) Diameter of Generic Tree

N.C. (5) Is Binary Tree Balanced?
- → $O(N^2)$ approach
- → $O(N)$ approach.

N.C. (6) Theory
- maximum & Minimum nodes wrt h.
- Strict BT
- Full BT
- complete BT

Important Talk regarding JSP-4 Placement Preparation!
↳ starting at 8:10.

JSP-4 {+ foundation batch}

Level 1
- → BT - 3rd
- → BST - 2 classes
- → Hashmap & Heap - 4 classes
- → Graph → 5 classes
- → Remainig Qucs

- → Batch change (Health/Mech)
- → Further Schedule
- → Course Duration
- → have 2 Path
- → Spoonfeeding
- → Revision & notes
- → Doubt Support
- → level 1, level 2

JSP-4 {+ foundation batch}

→ Batch change (Health/Fresh)

- → Further Schedule
- → Course Duration
- → Level 2 Path
- → Spoonfeeding
- → Revision & notes
- → Doubt Support
- → level 1, level 2

Fin-tech

Arithmetic → Maths, GK, Puzzles

Sunnyajit

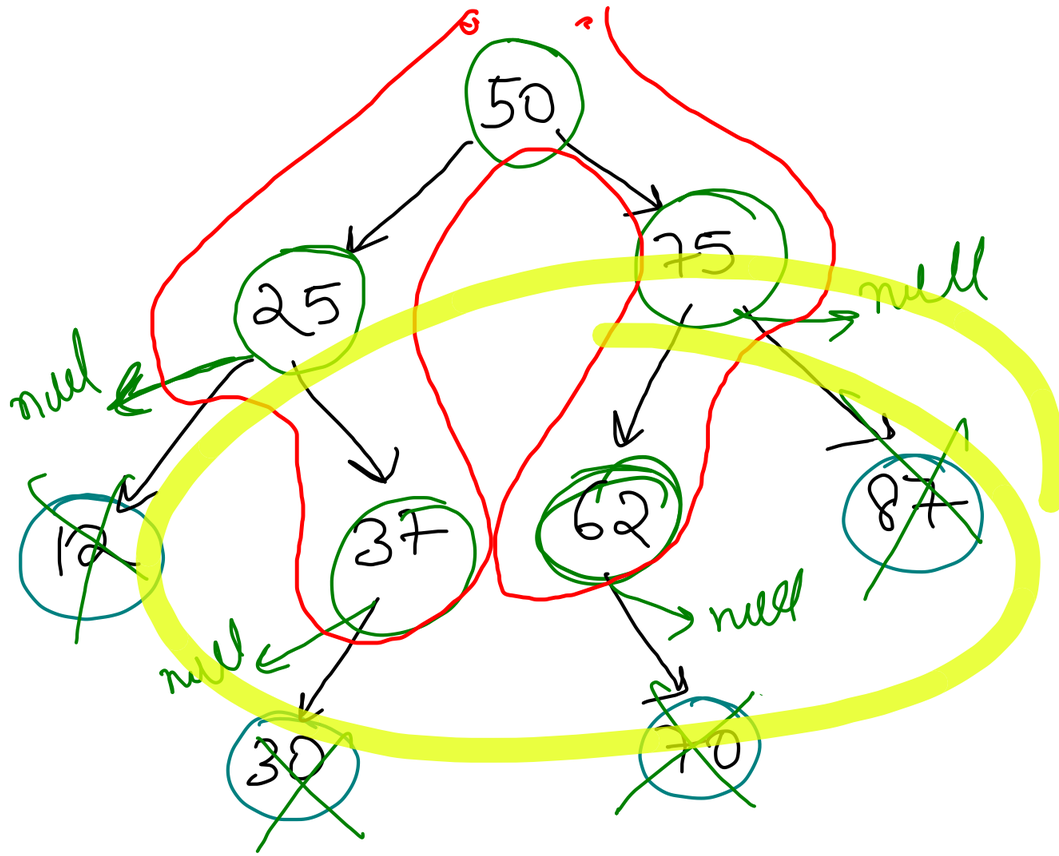
Level 1 :- Oracle, publicis sapient, paytm

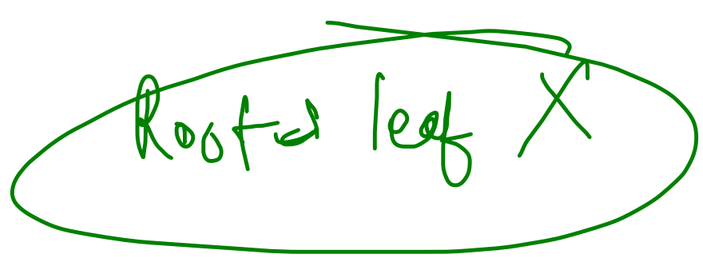
Level 2 :- Amazon, Micro, Intuit, Salesforce, Facebook, Adobe, ...so

Level 3 :- Uber, Google, Codenation, directi (media net), Tower Research

Remove leaves {void}

```java
// void return type : 1 edge case is not handled: root is leaf (1 node)
public static void removeLeaves(Node node){
    if(node == null) return; // root is null (0 nodes)

    if(node.left != null){
        // if left child is leaf node
        if(node.left.left == null && node.left.right == null){
            node.left = null;
        } else {
            removeLeaves(node.left);
        }
    }

    if(node.right != null){
        // if right child is leaf node
        if(node.right.left == null && node.right.right == null){
            node.right = null;
        } else {
            removeLeaves(node.right);
        }
    }
}
```





return type

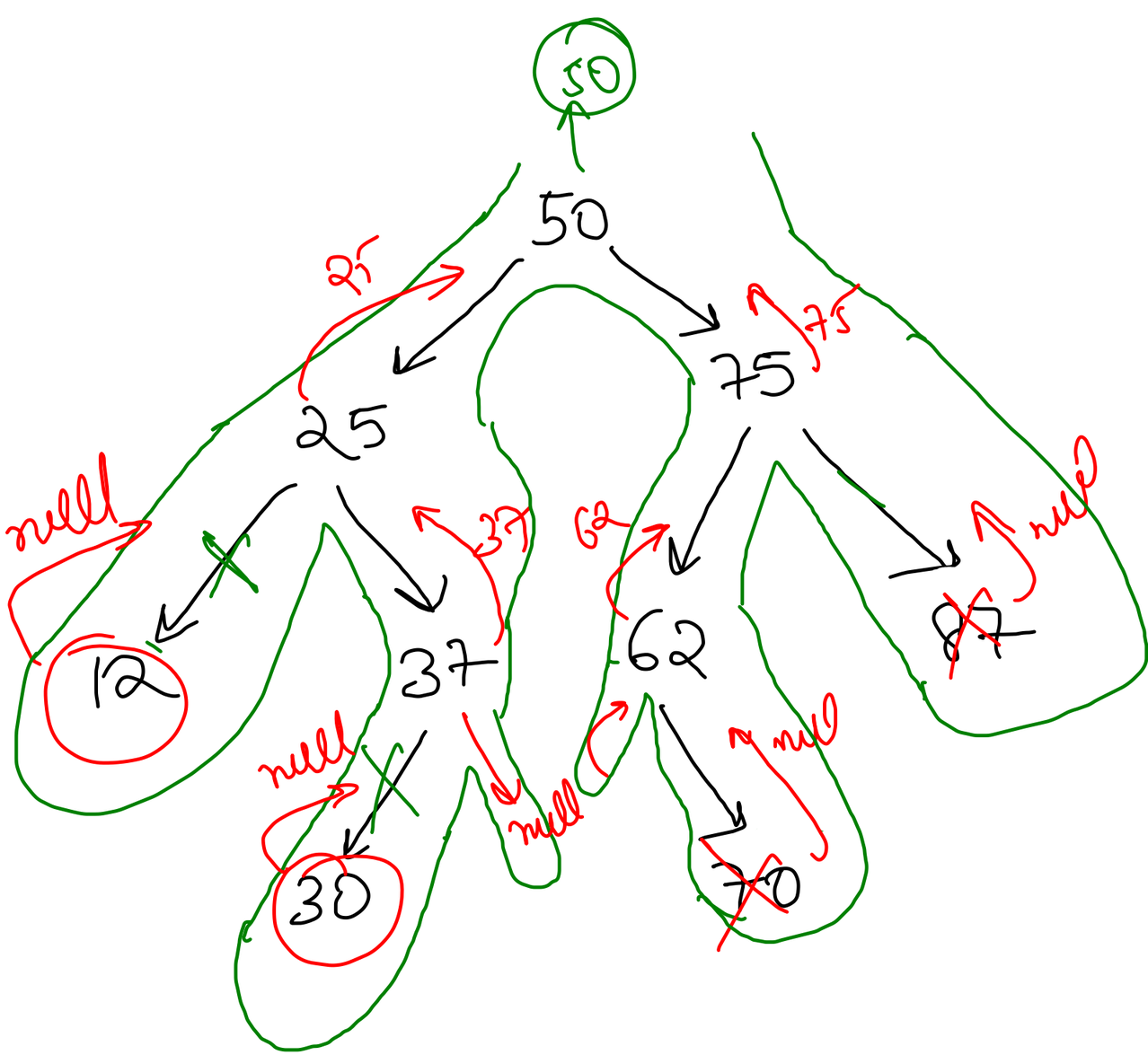
```
public static Node removeLeaves(Node node){
    if(node == null) return null;

    // leaf node -> delete node
    if(node.left == null && node.right == null)
      return null;

    Node leftChild = removeLeaves(node.left);
    Node rightChild = removeLeaves(node.right);

    node.left = leftChild;
    node.right = rightChild;

    // non-leaf node
    return node;
}
```

main

```
root = removeLeaves(root);
```

null

root = null



Tilt of Binary Tree

tilt = ~~0~~ ~~8~~ ~~12~~ ~~16~~ ~~18~~ 32

Tilt of node = |leftsum - rightsum|

Tilt of tree = Σ tilt for each node

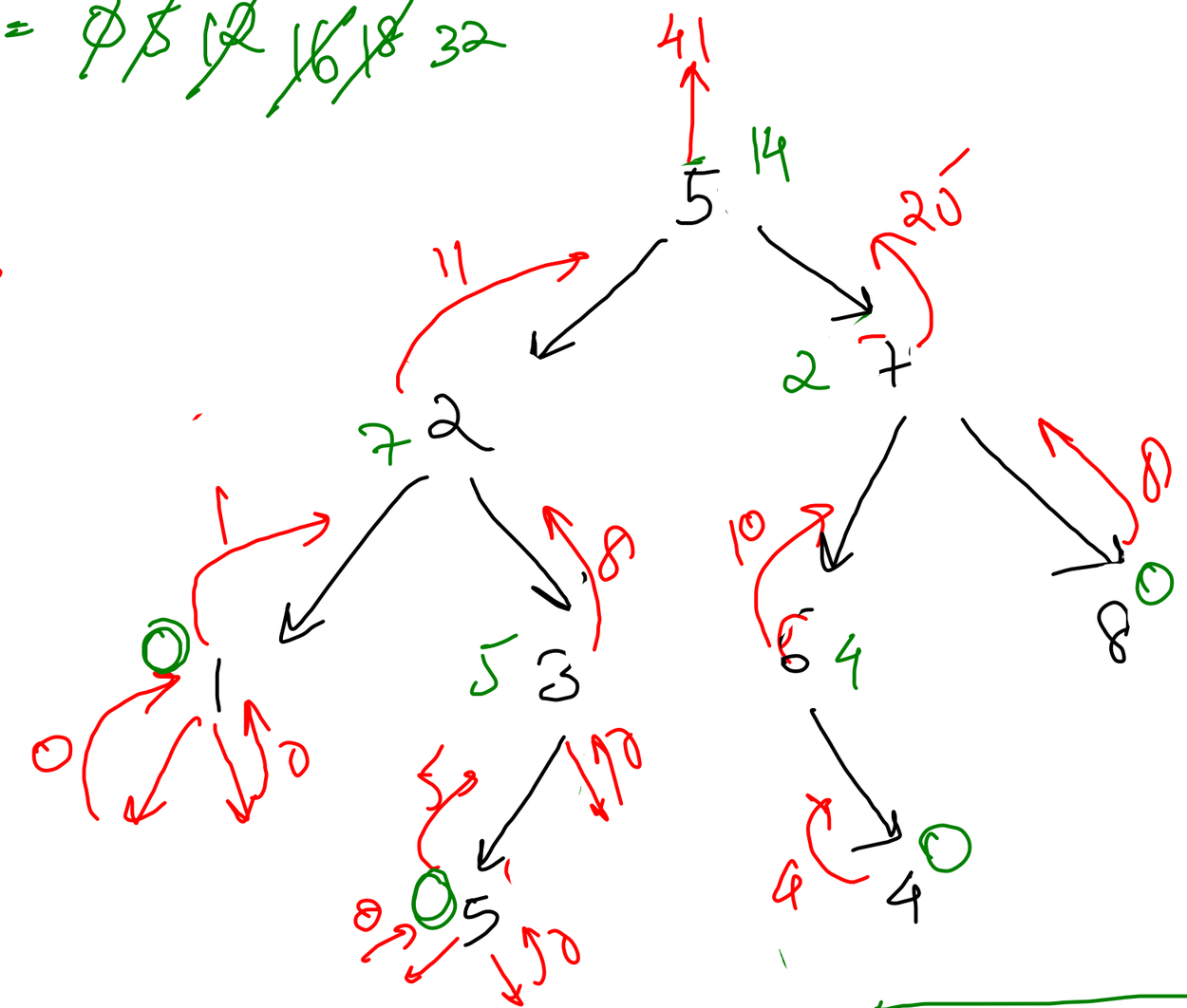
(Σ ← summation)

```java
static int tilt = 0;
public static int tilt(Node node){
  if(node == null) return 0;

  int lsum = tilt(node.left);
  int rsum = tilt(node.right);

  int tiltAtnode = Math.abs(lsum - rsum);
  tilt += tiltAtnode;

  return lsum + rsum + node.data;
}
```

return sum

calc → tilt

⇒ global variable strategy

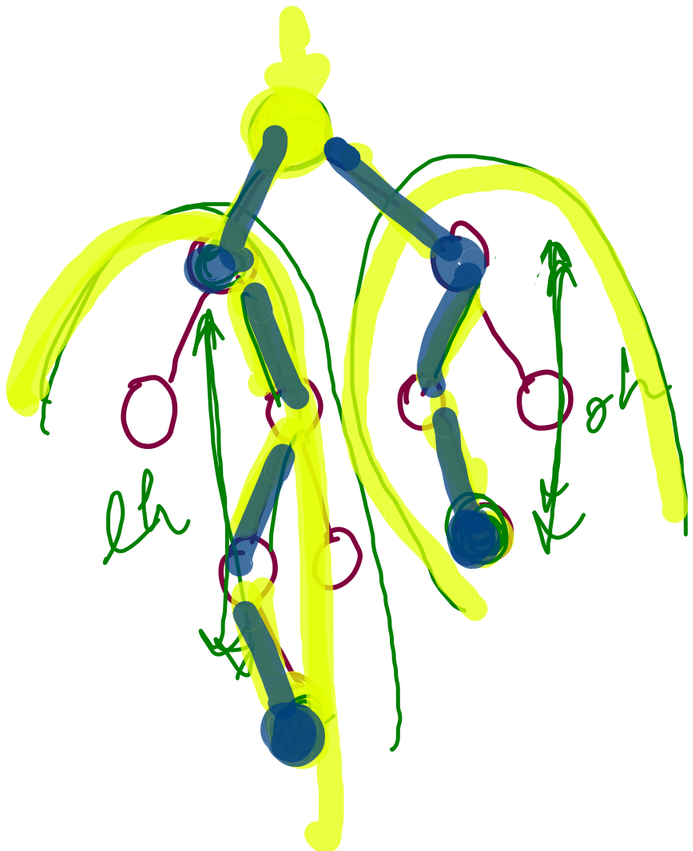


TC → O(n)

Diameter of Binary Tree

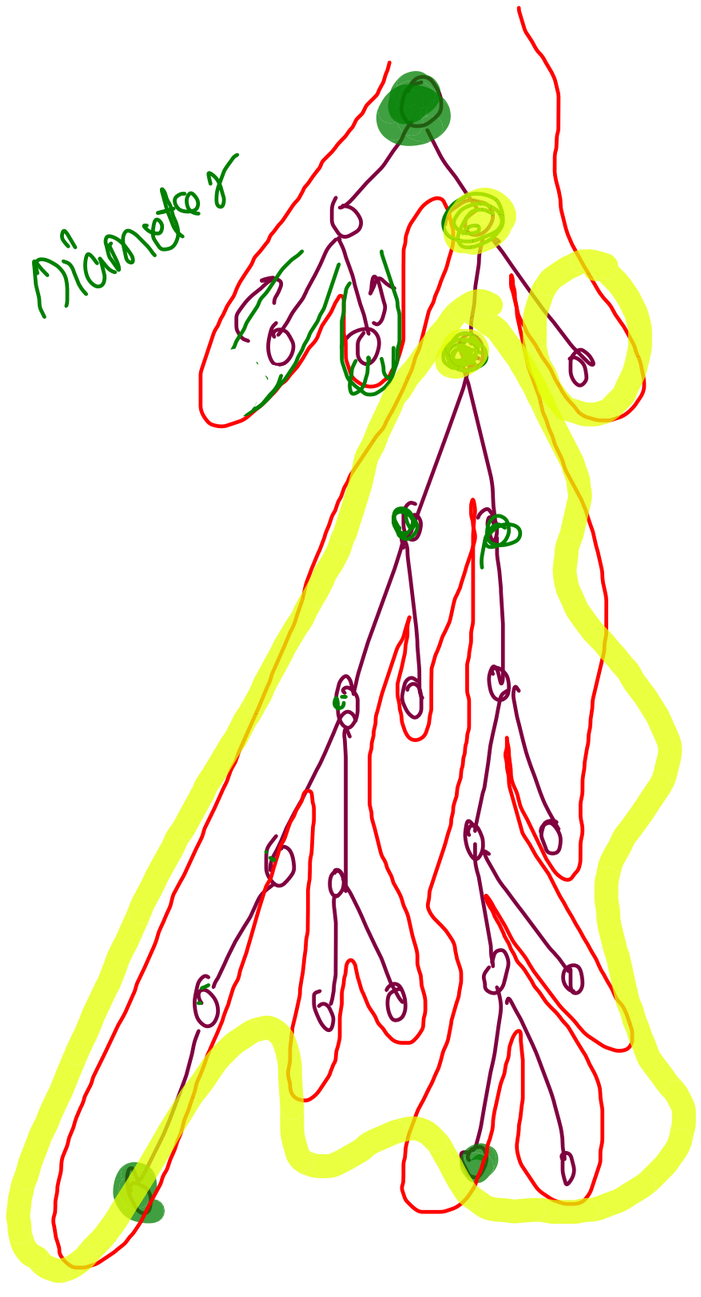
shortest path possible

Definition :→ Maximum distance b/w 2 ~~leaf~~ nodes {in terms of edges}



dia = lh + rh + 2

3 + 2 + 2 = 7

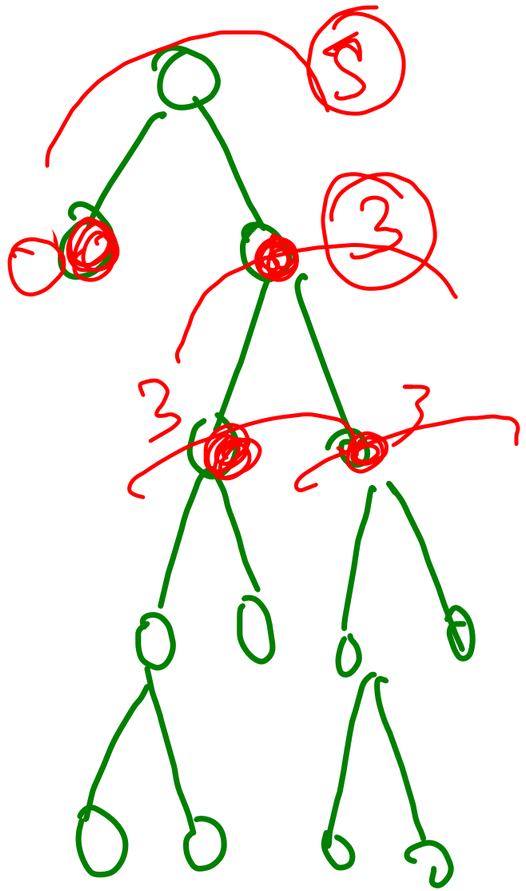


It is not necessary that diameter always passes through the root.

O(N)

```java
// we are returning diameter
public static int diameter1(Node node) {
    if(node == null) return 0;

    int lDia = diameter1(node.left);
    int rDia = diameter1(node.right);

    // meeting expectation
    int lh = height(node.left);
    int rh = height(node.right);
    int dia = lh + rh + 2;

    return Math.max(dia, Math.max(lDia, rDia));
}
```

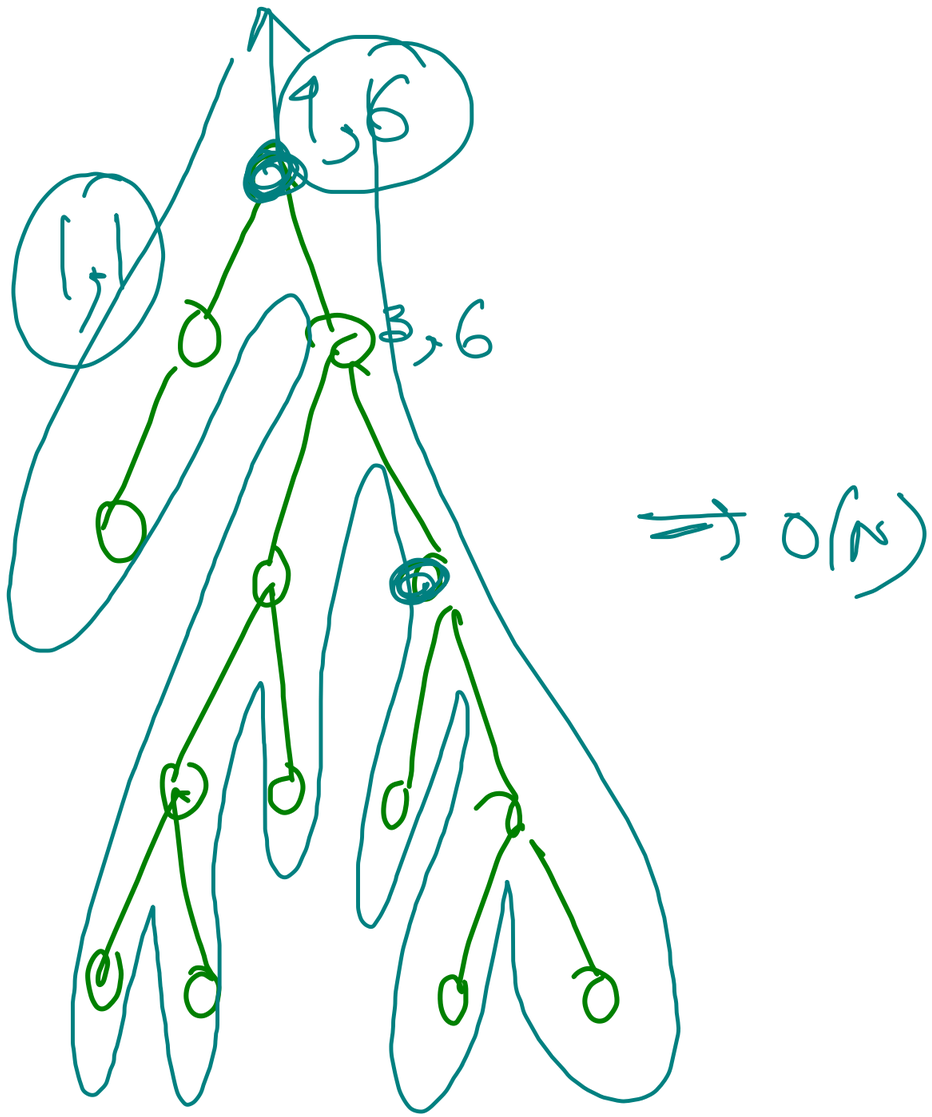
$n * n = O(n^2)$



TC: O(n)

② Global variable strategy!

```java
static int maxDiameter = 0;
public static int diameter2(Node node){
  if (node == null) {
    return -1;
  }

  int lh = diameter2(node.left);
  int rh = diameter2(node.right);

  int dia = lh + rh + 2;
  maxDiameter = Math.max(maxDiameter, dia);

  return Math.max(lh, rh) + 1;
}
```



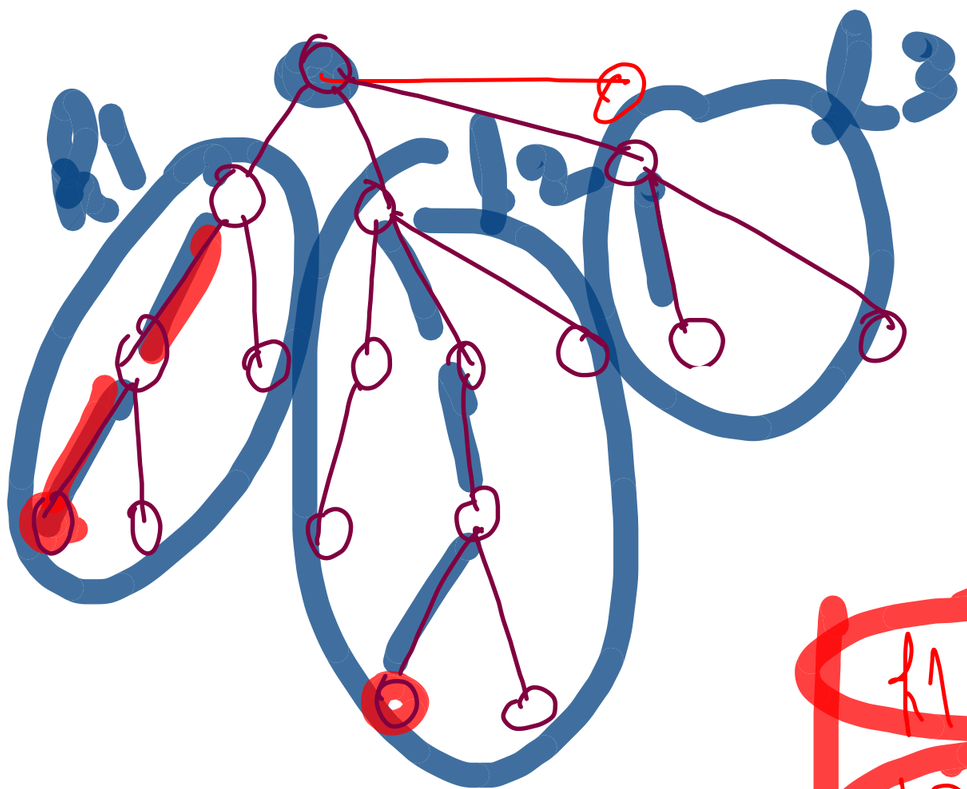
$\Rightarrow O(N)$

(3) Return both diameter & height!

```java
public static class diaPair{
    int height;
    int diameter;

    public diaPair(){
        height = -1;
        diameter = 0;
    }
}
```

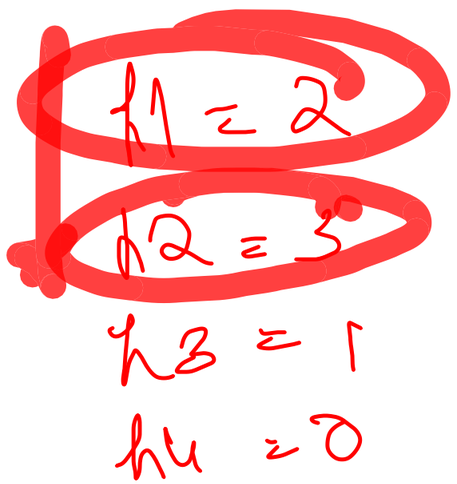
```java
// Return both diameter & height
public static diaPair diameter3(Node node){
    if(node == null)
        return new diaPair();

    diaPair l = diameter3(node.left);
    diaPair r = diameter3(node.right);

    diaPair curr = new diaPair();
    curr.height = Math.max(l.height, r.height) + 1;

    int dia = l.height + r.height + 2;
    curr.diameter = Math.max(dia, Math.max(l.diameter, r.diameter));

    return curr;
}
```

Diameter - Generic Tree



$lh + rh + 2$

$dia = max1 + max2 + 2$




```java
static int maxDiameter = 0;
public static int diameter(Node root){

    int max1 = -1, max2 = -1;
    for(Node child: root.children){
        int hc = diameter(child);

        if(hc >= max1){
            max2 = max1;
            max1 = hc;
        } else if(hc >= max2){
            max2 = hc;
        }
    }

    int dia = max1 + max2 + 2;
    maxDiameter = Math.max(dia, maxDiameter);

    return max1 + 1;
}
```