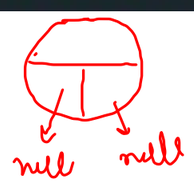
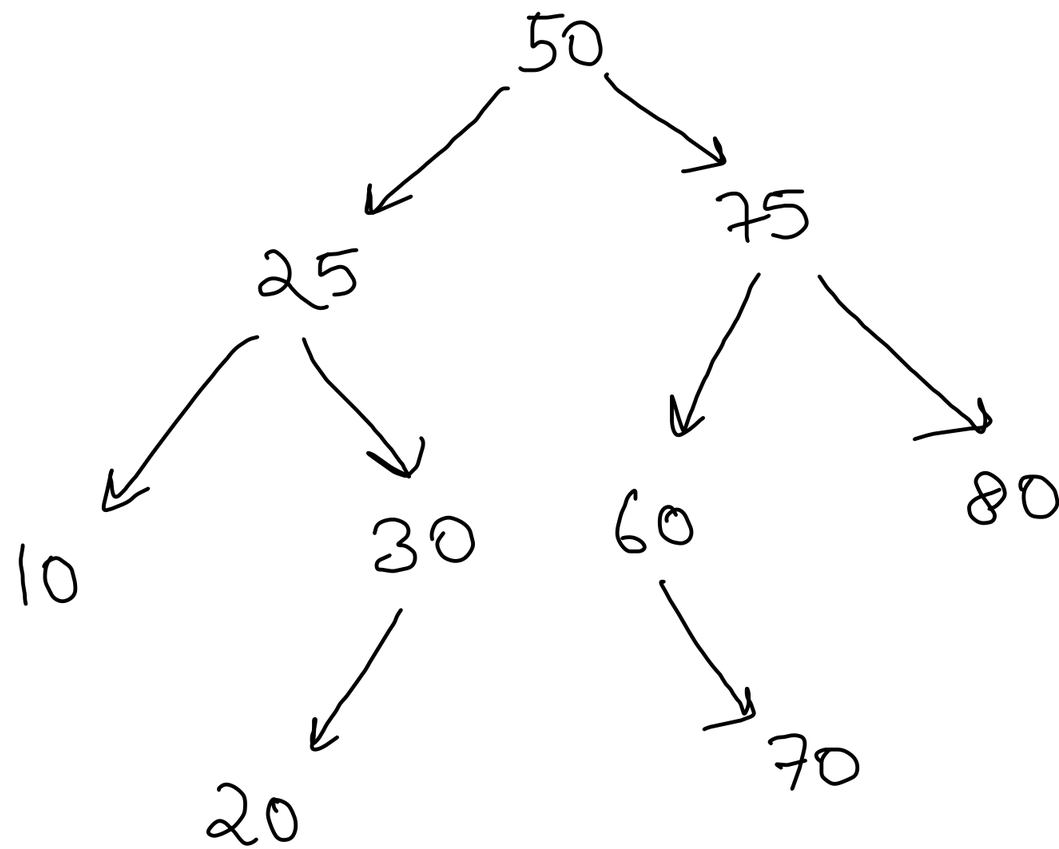
Binary Tree

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



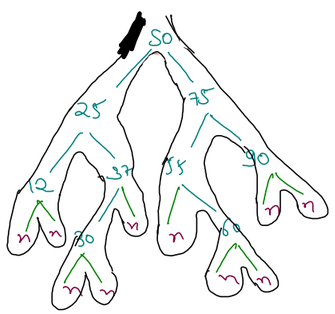

```
class Node {

    int data;
    Node left;
    Node right;

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

}
```



Construction

50
25
12
null
null
37
30
null
null
null

75
55
null
50
null
null
90
null
null




```java
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

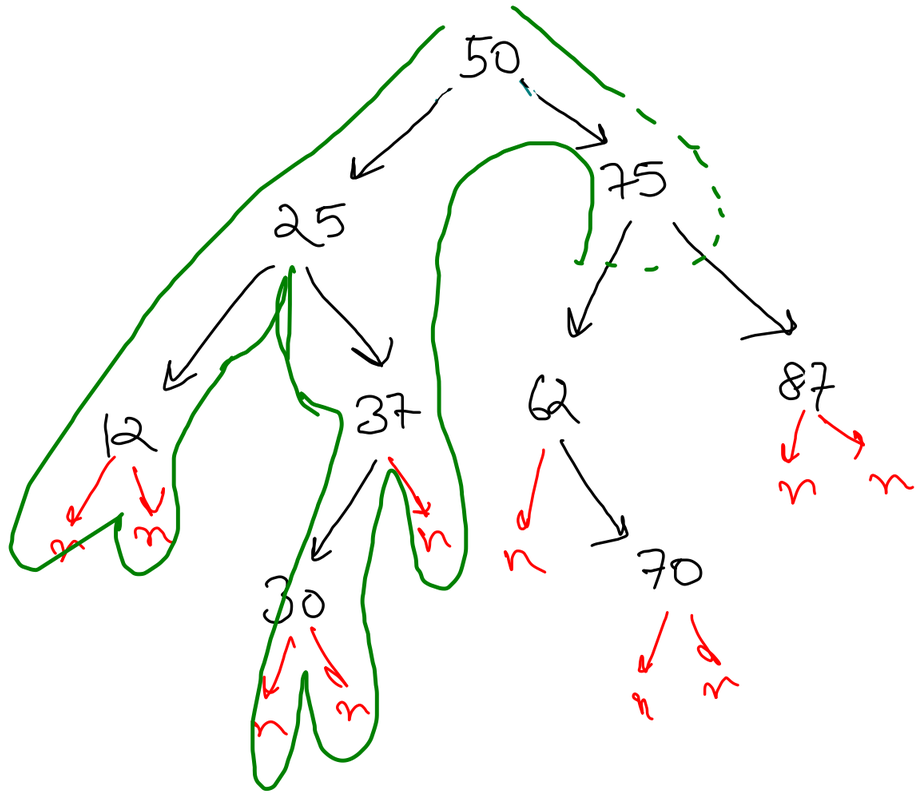
62 ← 75 → 87

. ← 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
}
```

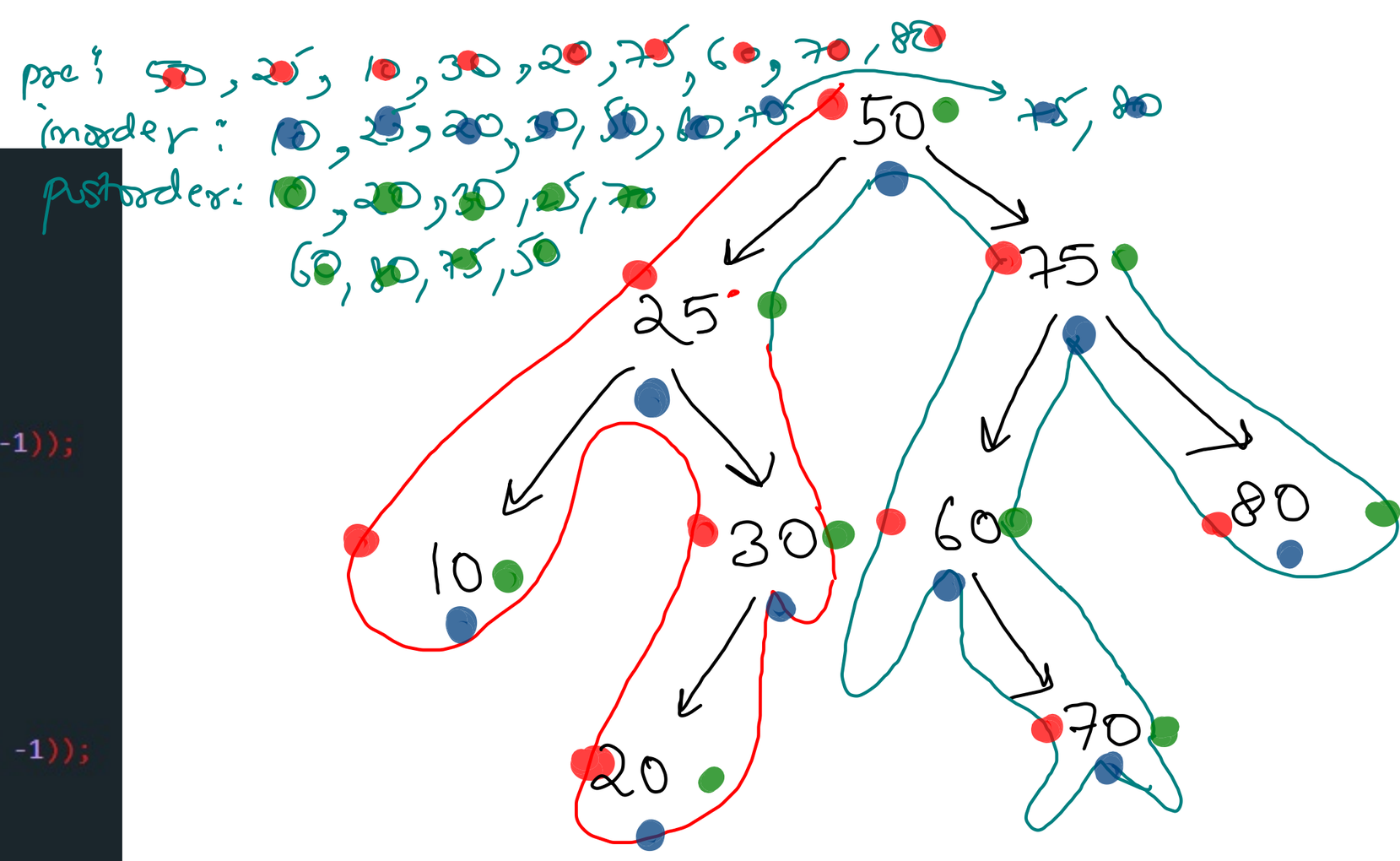



```
if (root == null) return;
print(root);
display(root.left);
display(root.right);
```

Iterative

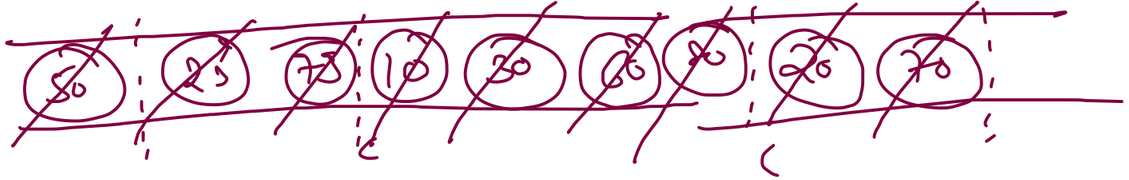
```
while(!stk.isEmpty()){
    Pair par = stk.peek();

    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();
    }
}
```

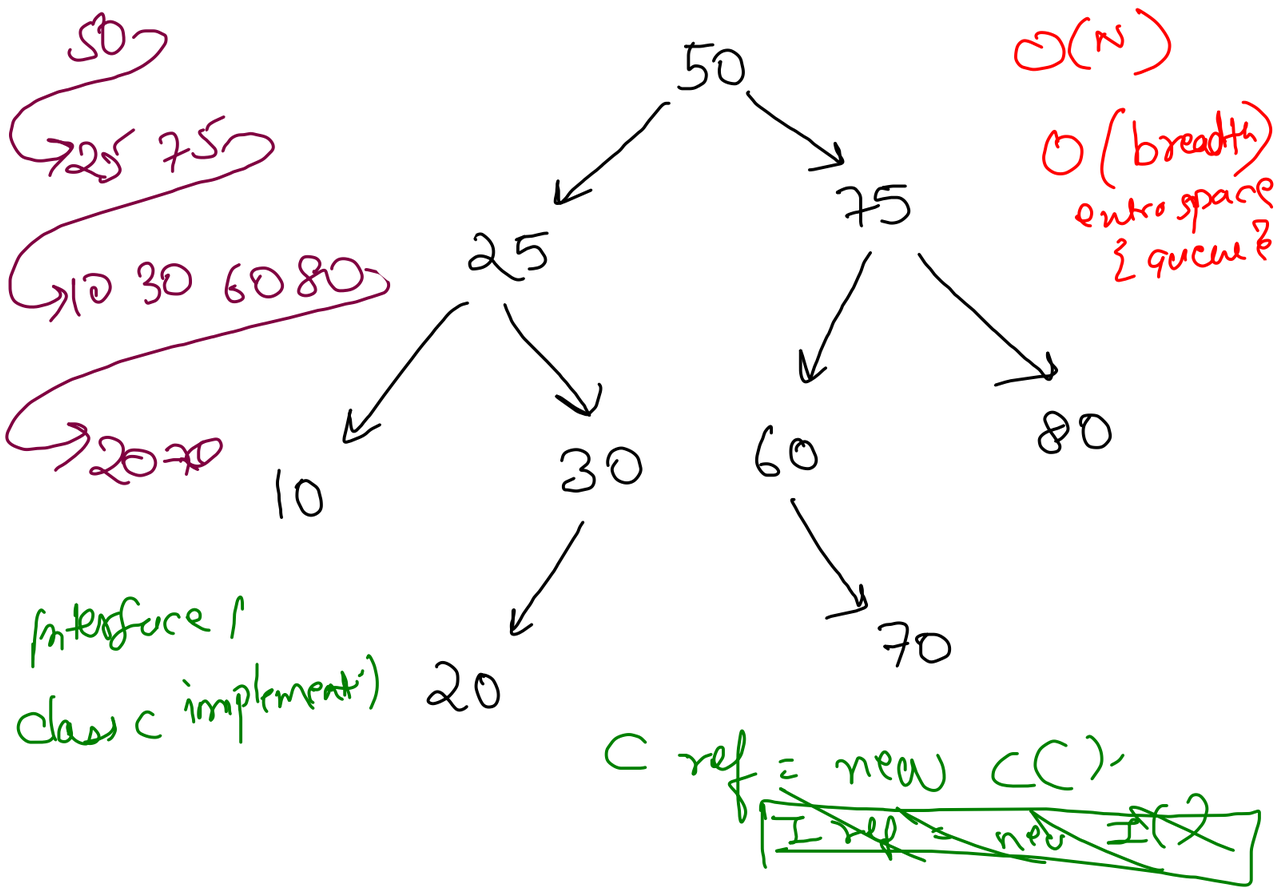
You are screen shari



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

LL

removeFirst, addLast

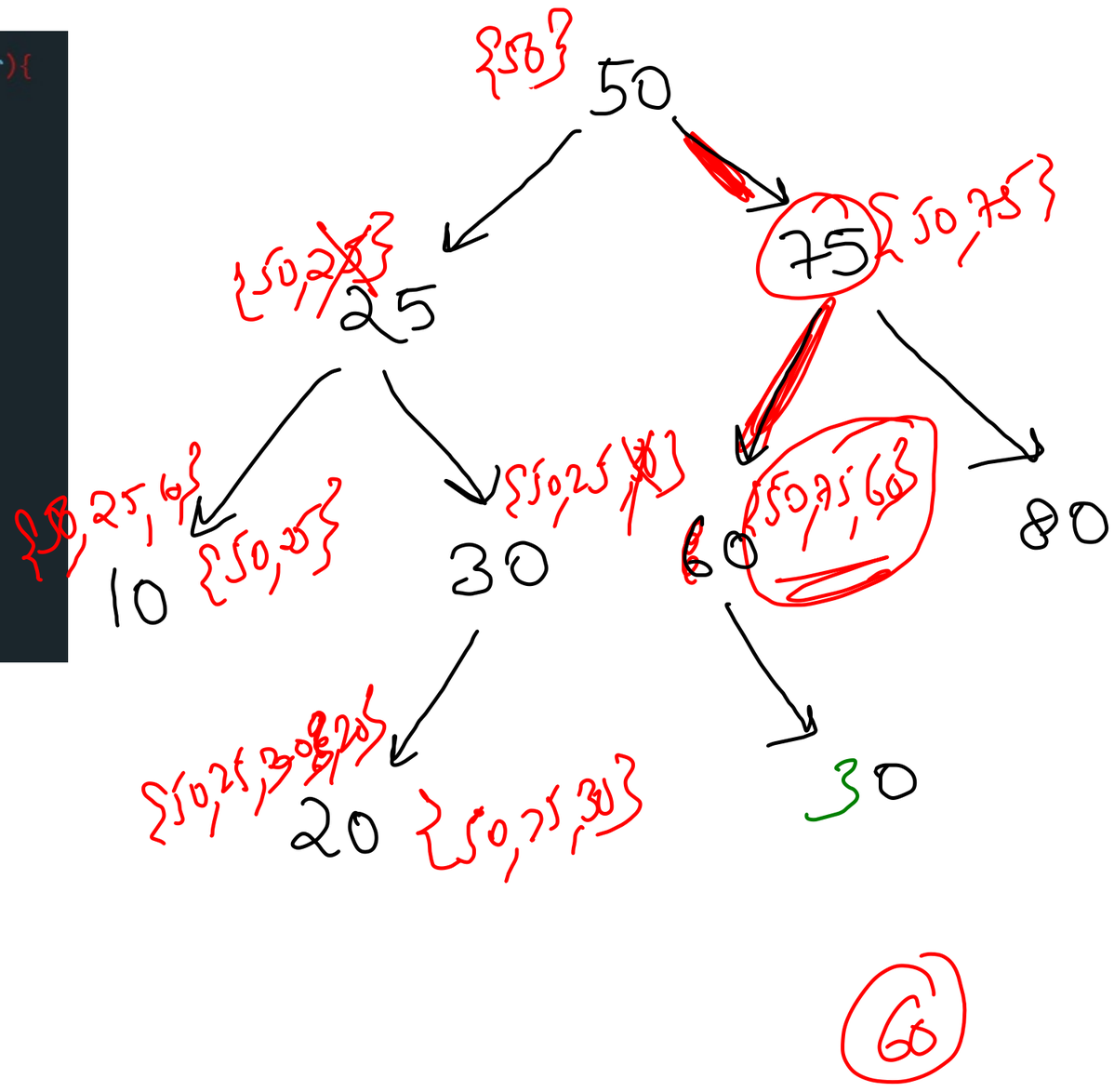
rF, aF
rL, aL

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;
}
```

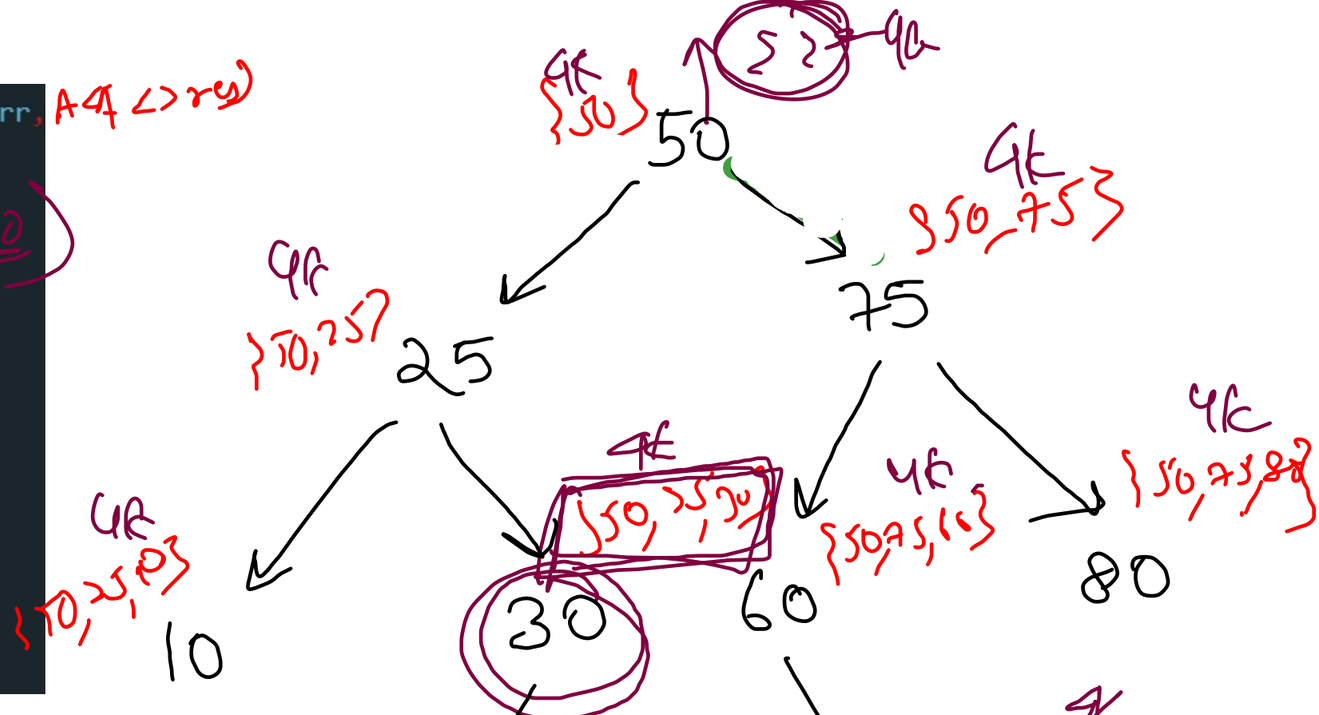
Root to Node Path
{ Backtracking }

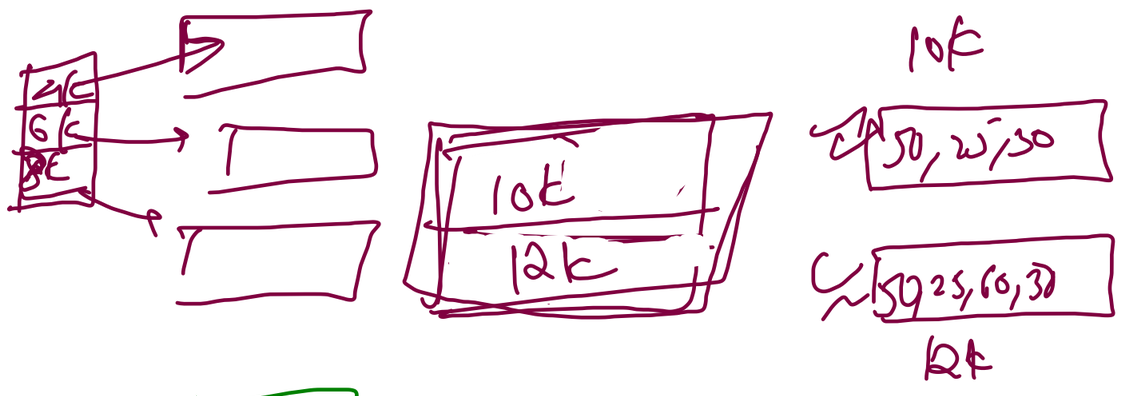


```java
public static void rootToNodePath(Node node, int data, ArrayList<Integer> curr, ArrayList<ArrayList<Integer>> res){
    if(node == null) // negative base case
        return;

    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)

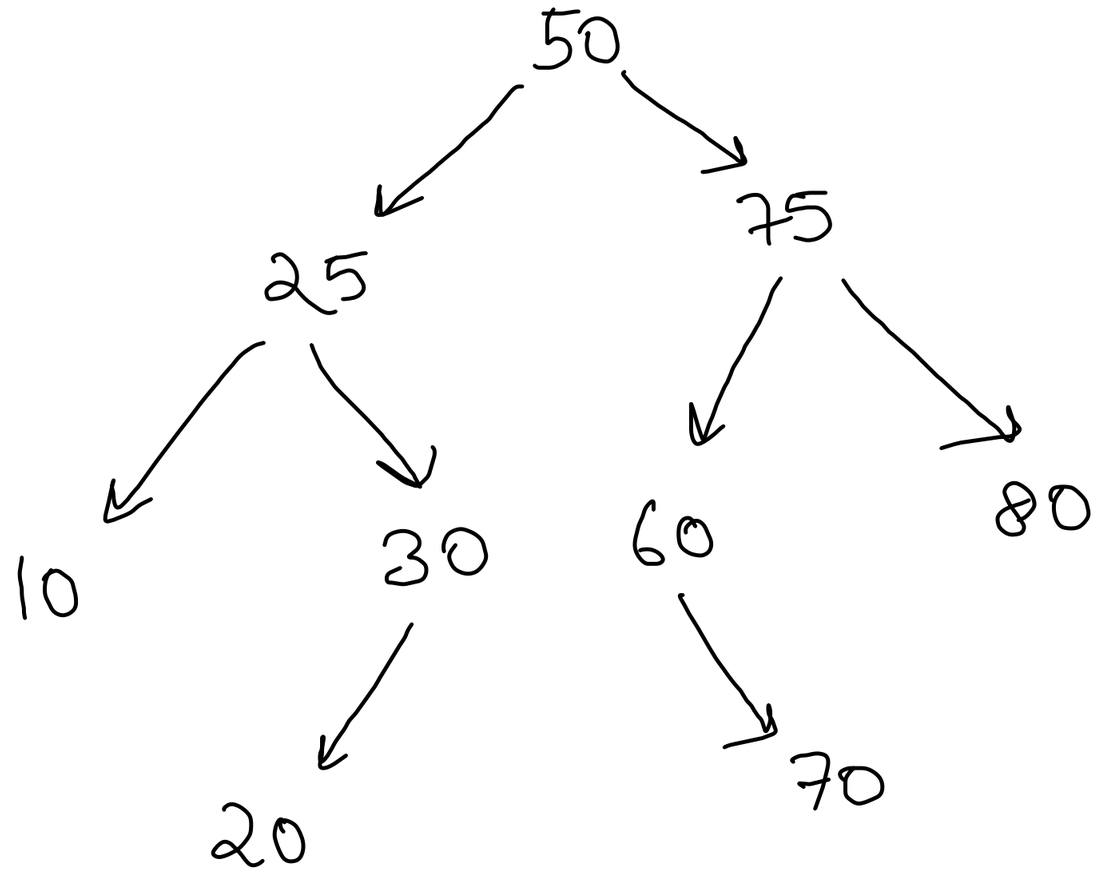


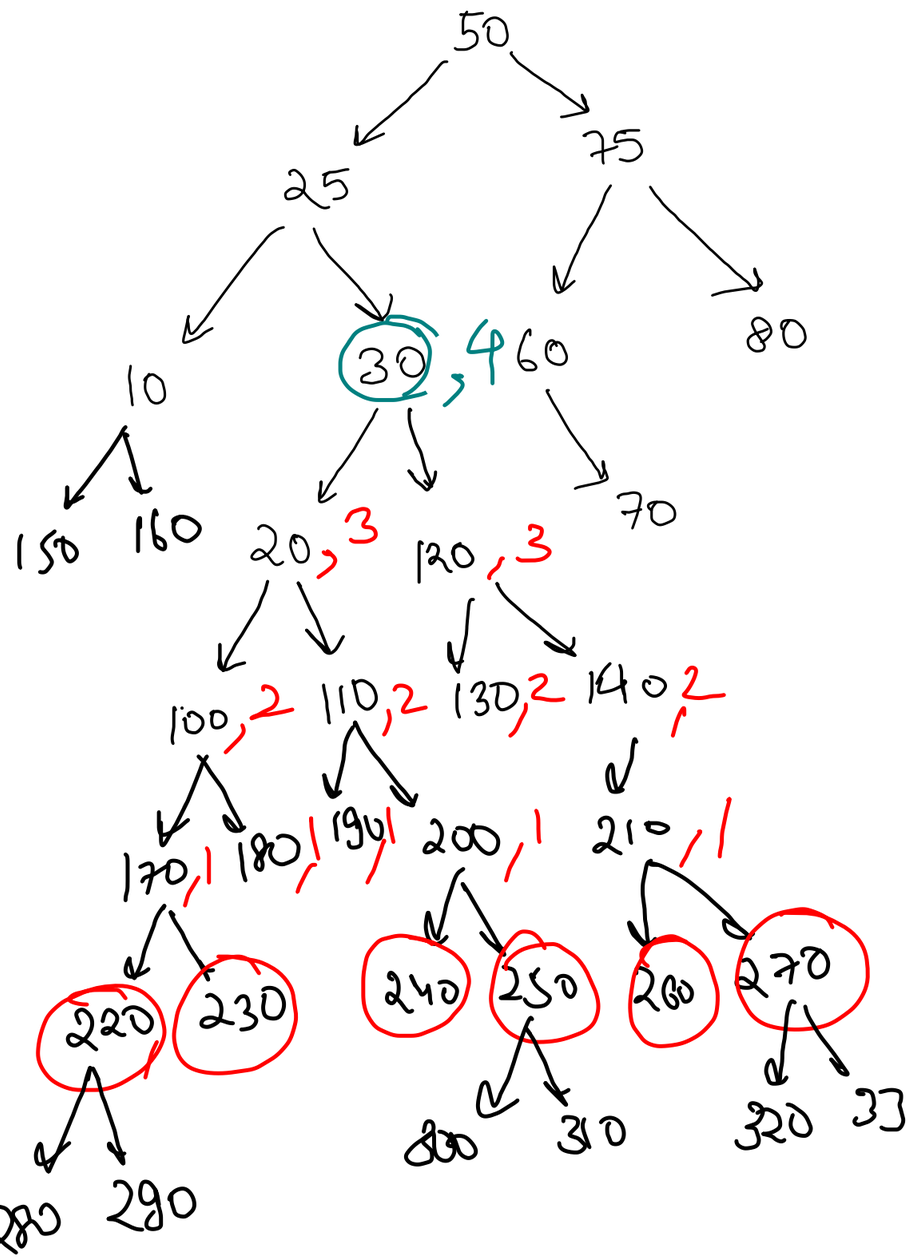


All root to node paths

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

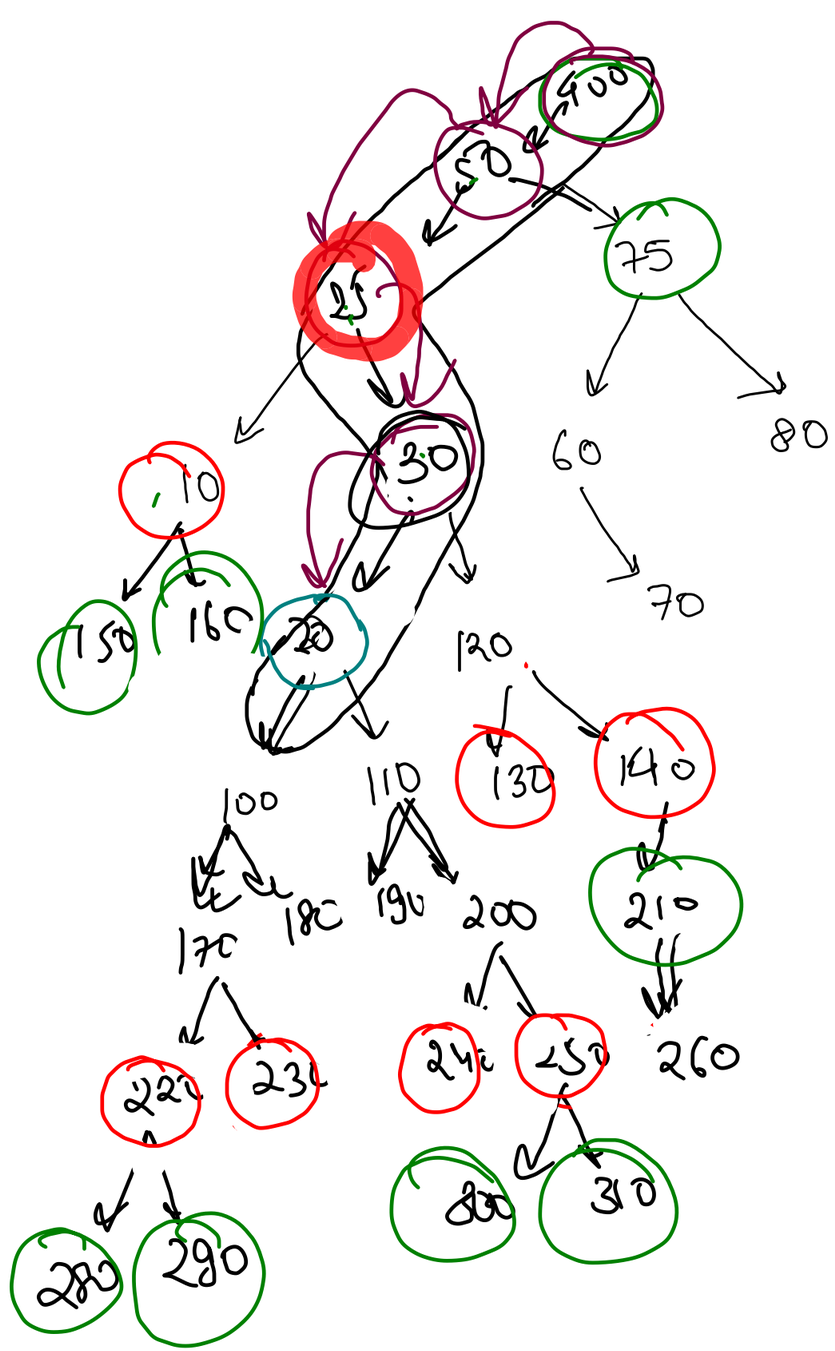
```java
public static void printKLevelsDown(Node node, int k){
  if(node == null) return;
  if(k == 0){
      System.out.println(node.data);
      return;
  }

  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)



Nodes k Distance Away

Amazon

20, 3

Nodetorootpath(root, node)

{20, 30, 25, 50, 400}

3 2 1 0 -1

Klevedown(20, 3, null)

Klevedown(30, 2, 20)

Klevedown(25, 1, 30)

Klevedown(50, 0, 25)

Klevedown(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

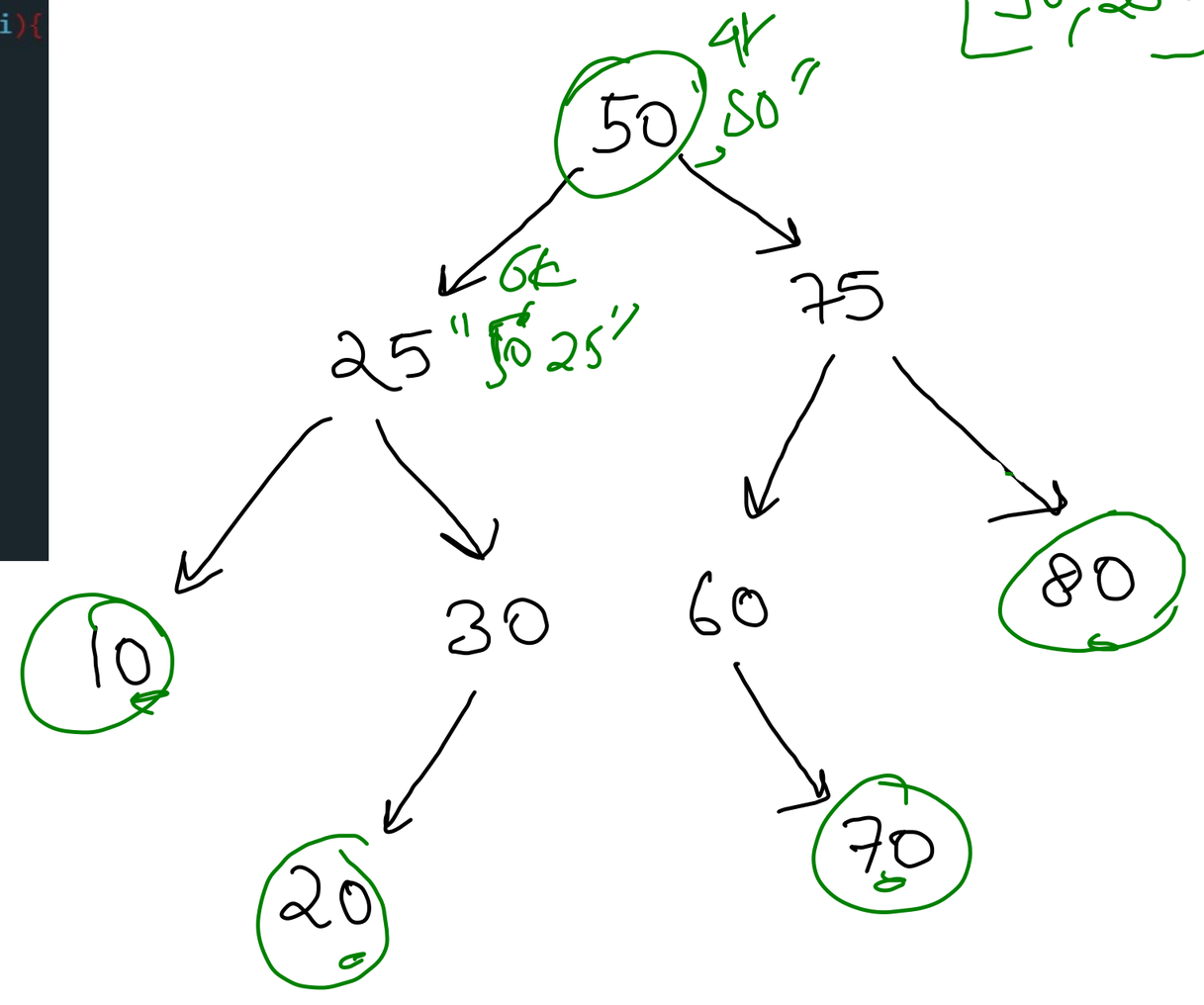
```java
public static void pathToLeafFromRoot(Node node, String path, int sum, int lo, int hi){
    if(node == null) return;

    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;

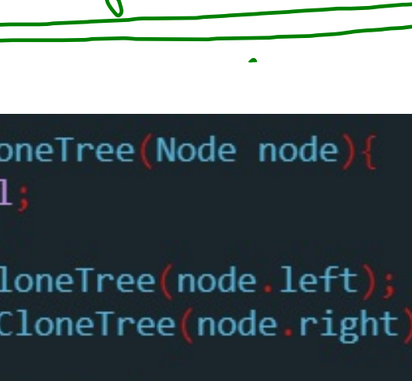
[low, high]

[90, 230]



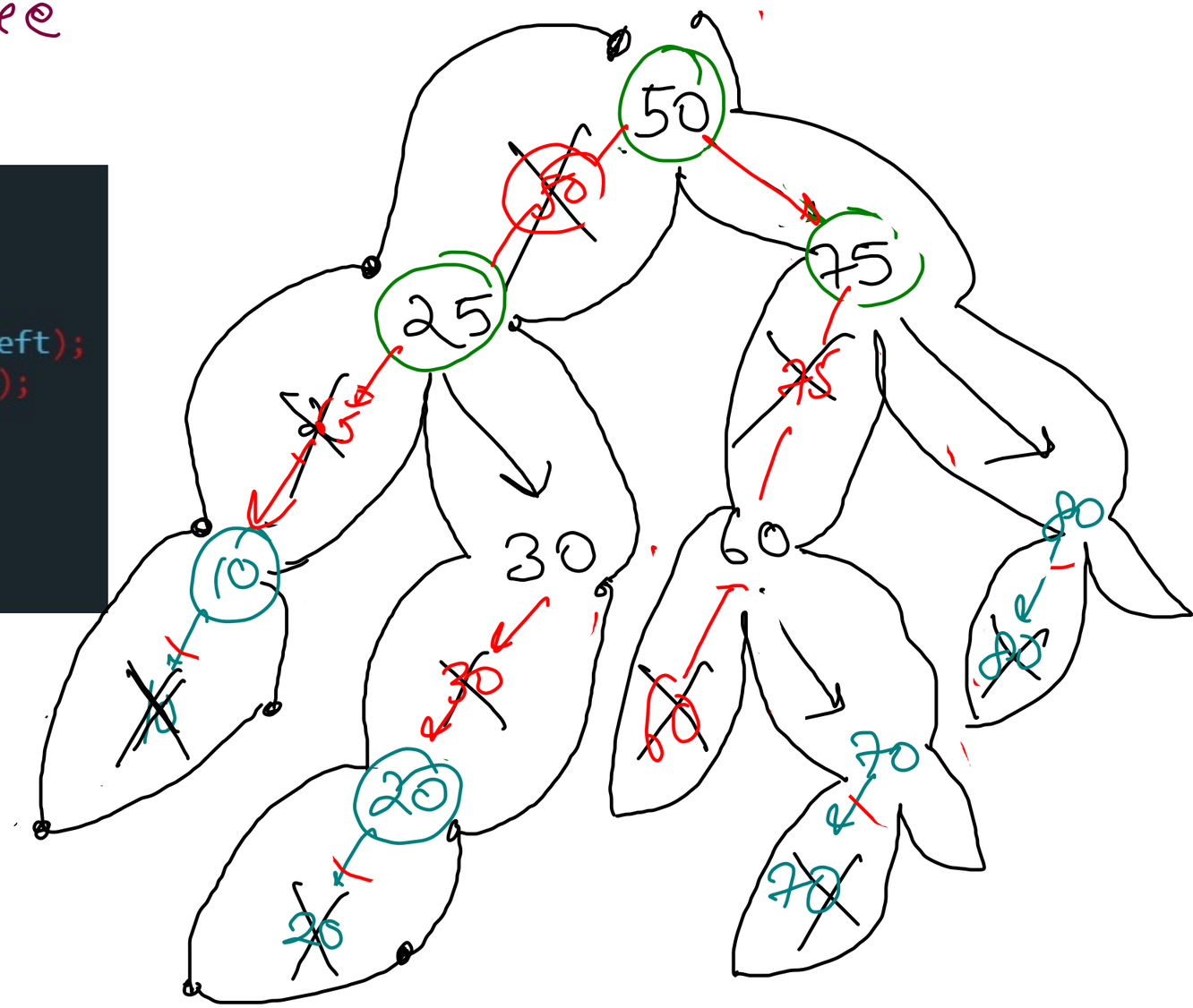
Transform to left-cloned Tree

```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 par lai