BT ⇒ G.T. + at most 2 child

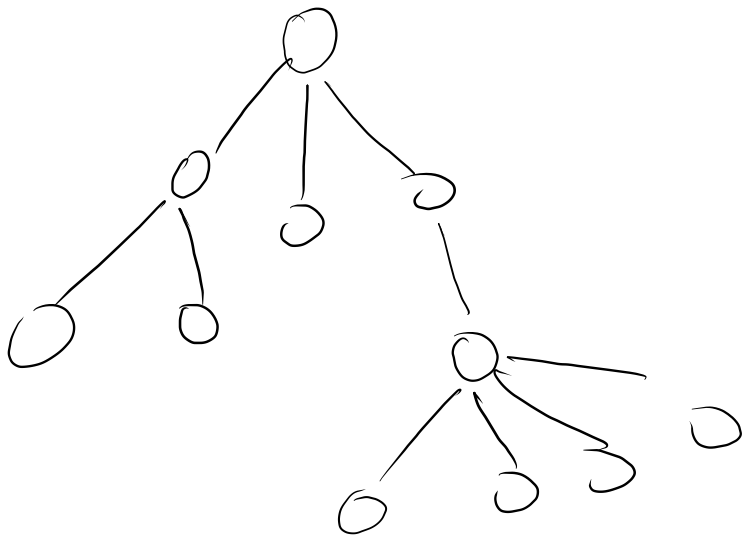
Node → Input → Construct → Display

Node

Type : Node

data

children ⇒ AL

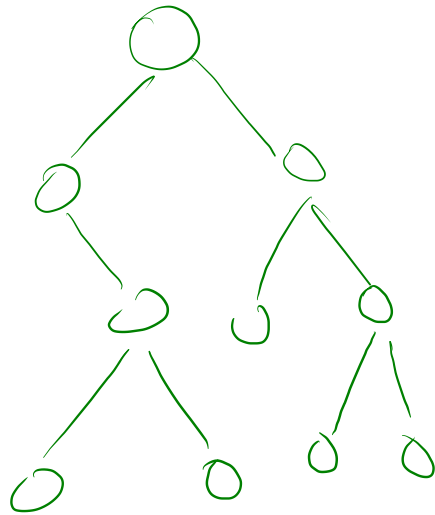


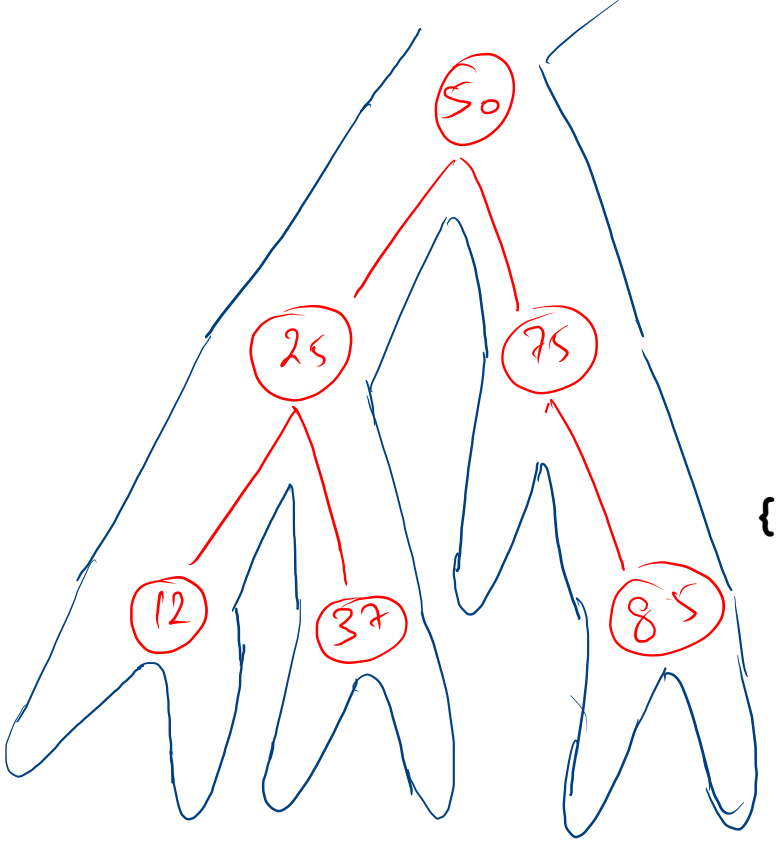
Node

data

Left right

Type : Node



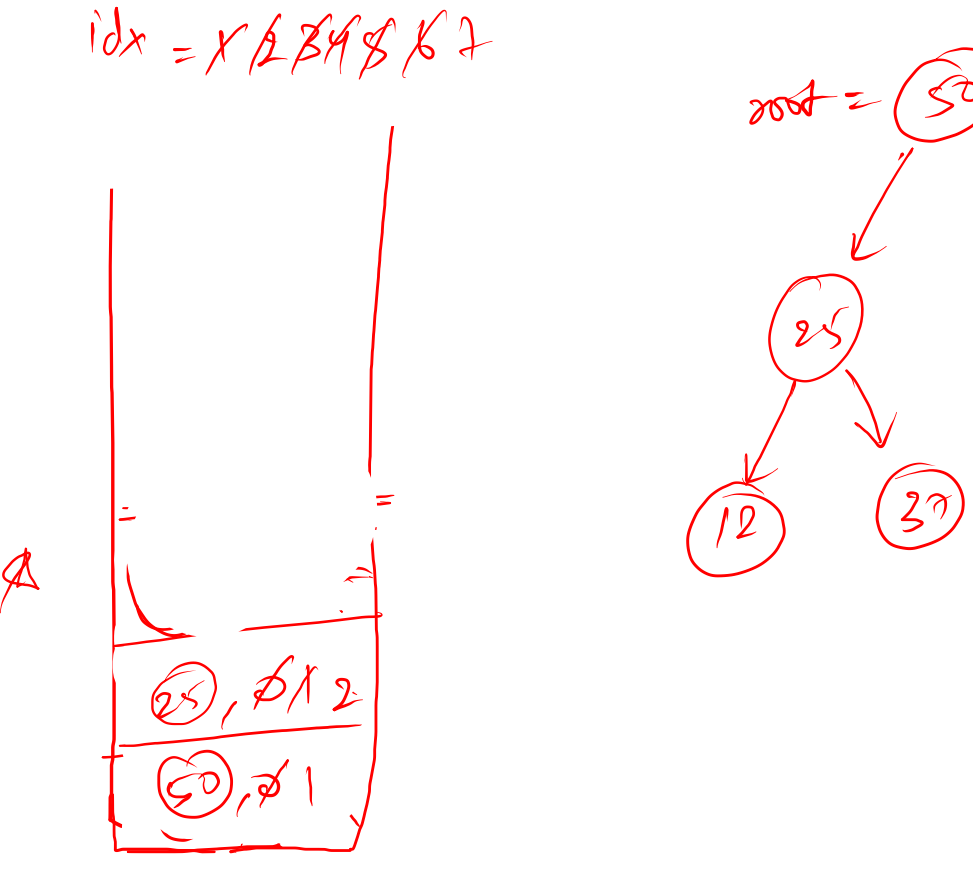


50 25 12 null null 37 null null

75 null 85 null null

{ 50 , 25 , 12 , null , null , 37 , null , null , 75 , null , 85 , null , null }

{ 50 , 25 , 12 , null , null , 37 , null , null , 75 , null , 85 , null , null }



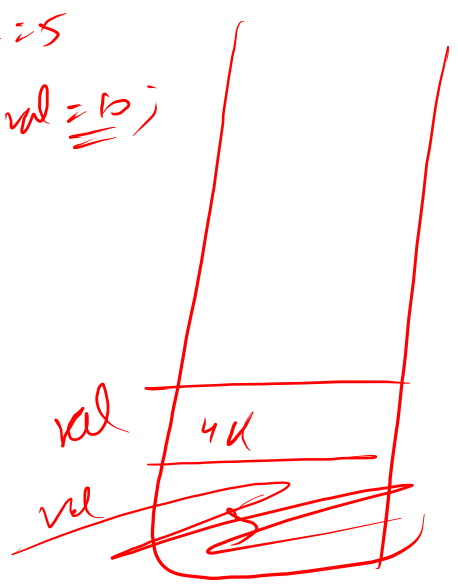

```
public static Node construct(Integer inp[]){
    Node root = new Node(inp[0],null,null);
    int idx = 1;
    Stack<> st = new Stack<>();
    st.push(new Pair(root,0));

    // 0 -> no child processed , 1 -> left child processed , 2 -> both child processed
    while(st.size() > 0){
        Pair top = st.peek();

        if(top.state == 0){
            Integer val = inp[idx++];
            if(val != null){
                Node lchild = new Node(val, null, null);
                top.node.left = lchild;
                st.push(new Pair(lchild,0));
            }
            top.state++;
        }else if(top.state == 1){
            Integer val = inp[idx++];
            if(val != null){
                Node rchild = new Node(val, null, null);
                top.node.right = rchild;
                st.push(new Pair(rchild,0));
            }
            top.state++;
        }else {
            st.pop();
        }
    }

    return root;
}
```

int → Integer (null)

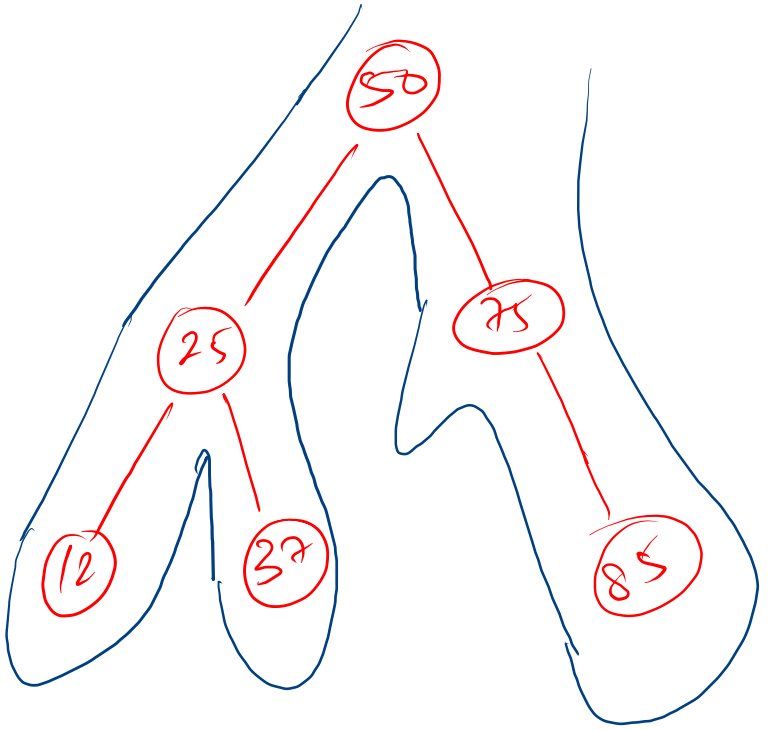
char → Character

int val = 5;

Integer val = 10;



Display



25 ← 50 → 75

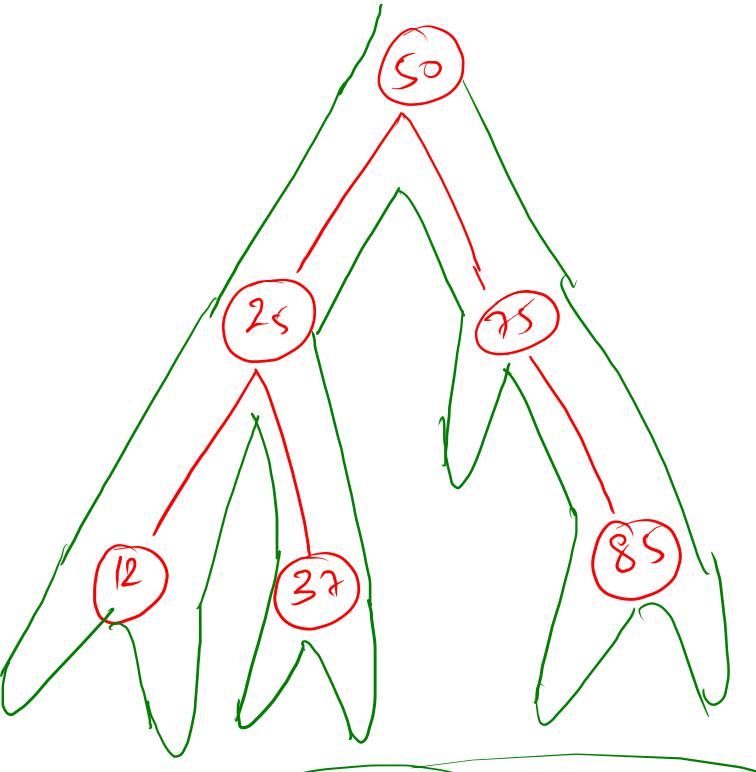
12 ← 25 → 37

. ← 12 → .

. ← 37 → .

. ← 75 → 85

. ← 85 → .

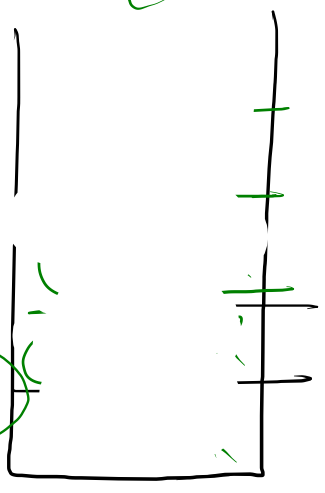



```
25 <- 50 -> 75
12 <- 25 -> 37
. <- 12 -> .
. <- 37 -> .
. <- 75 -> 85
. <- 85 -> .
```

Display

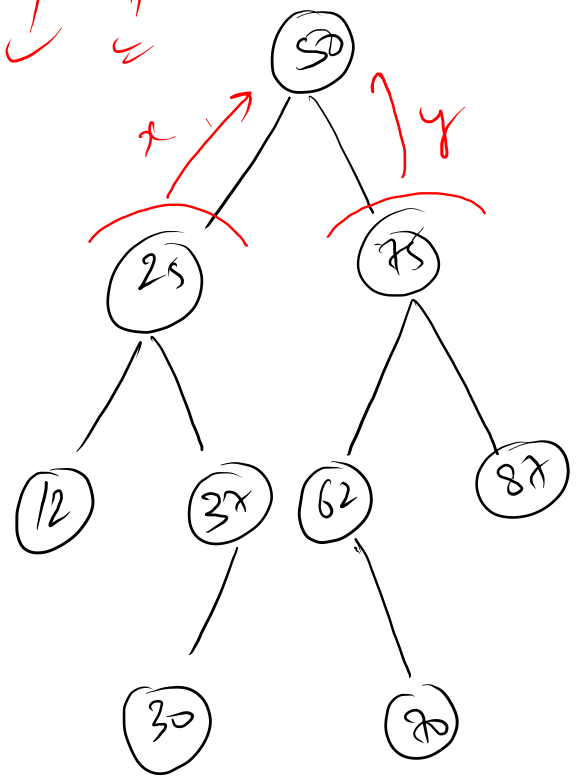
```
public static void display(Node node){
    if(node == null){
        return;
    }

    String str = "";
    str += (node.left == null) ? "." : node.left.data;
    str += " <- "+node.data+" -> ";
    str += (node.right == null) ? "." : node.right.data;
    System.out.println(str);

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

BT-, Insert, Construct, Display

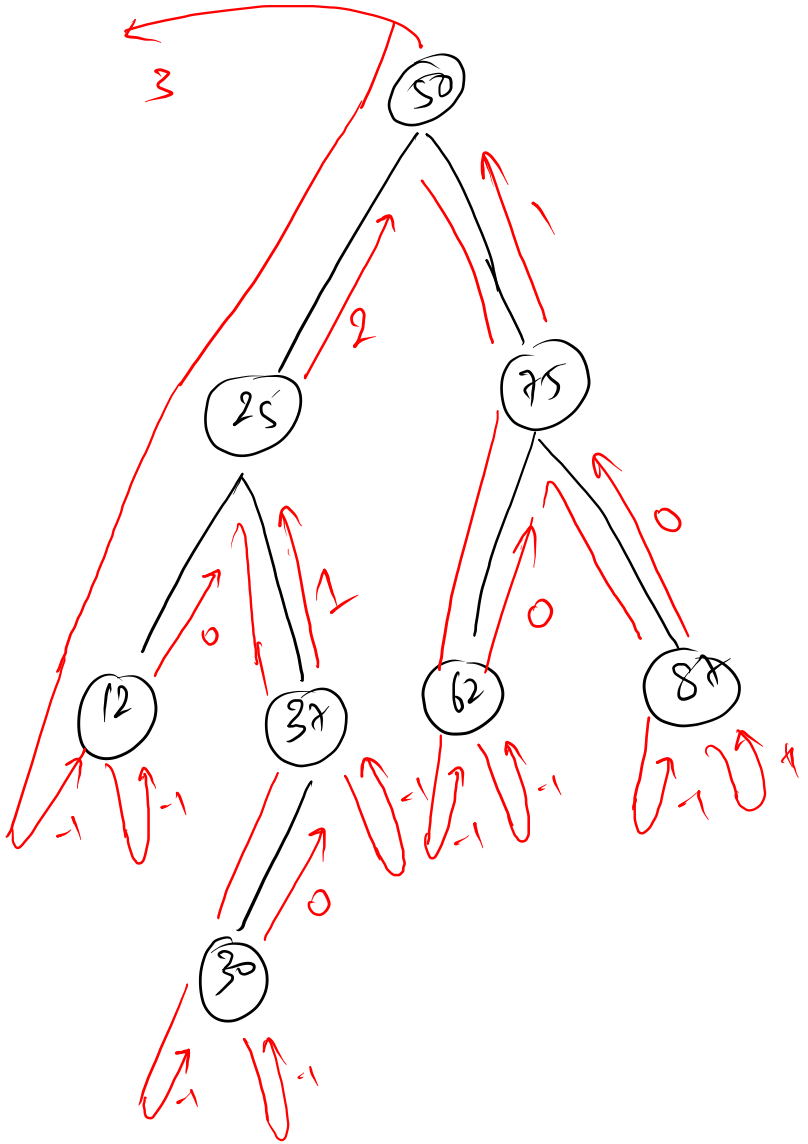


Size, Sum, Maximum And Height Of A Binary Tree

19
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n




```
fun ( node ) {
if ( node == null ) {
    return;
}

fun ( node.left ) ;
fun ( node.right ) ;

}
```



★ Ht on basis of Edges

```java
public static int height(Node node) {
    if(node == null){
        return -1;
    }
    int lheight = height(node.left);
    int rheight = height(node.right);

    return Math.max(lheight,rheight)+1;
}
```

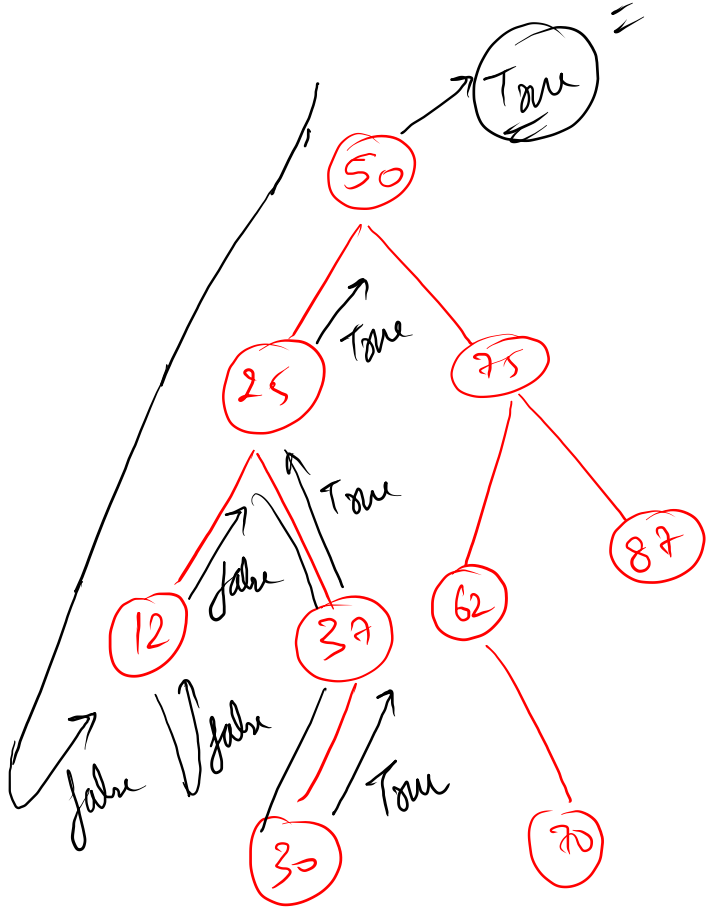
★ Ht on basis of nodes

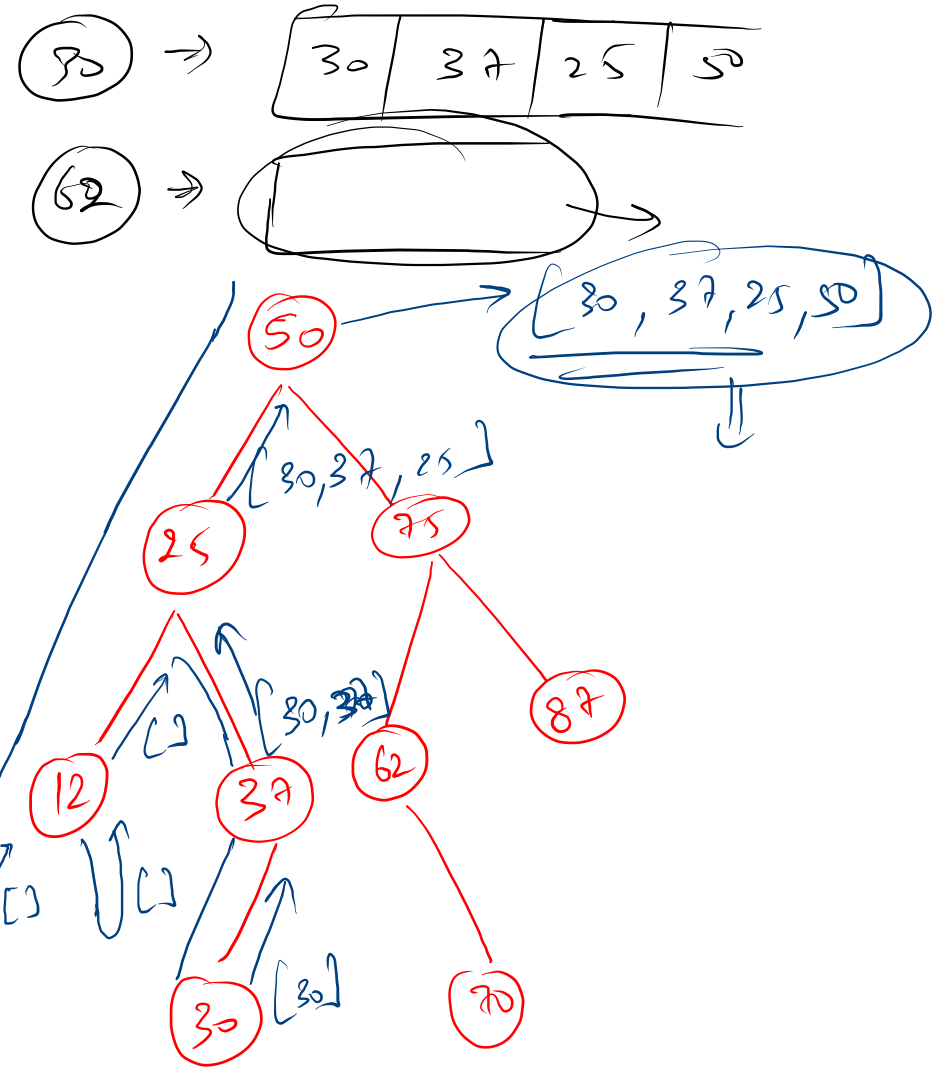
```java
public static int height(Node node) {
    if(node == null){
        return 0;
    }
    int lheight = height(node.left);
    int rheight = height(node.right);

    return Math.max(lheight,rheight)+1;
}
```

19

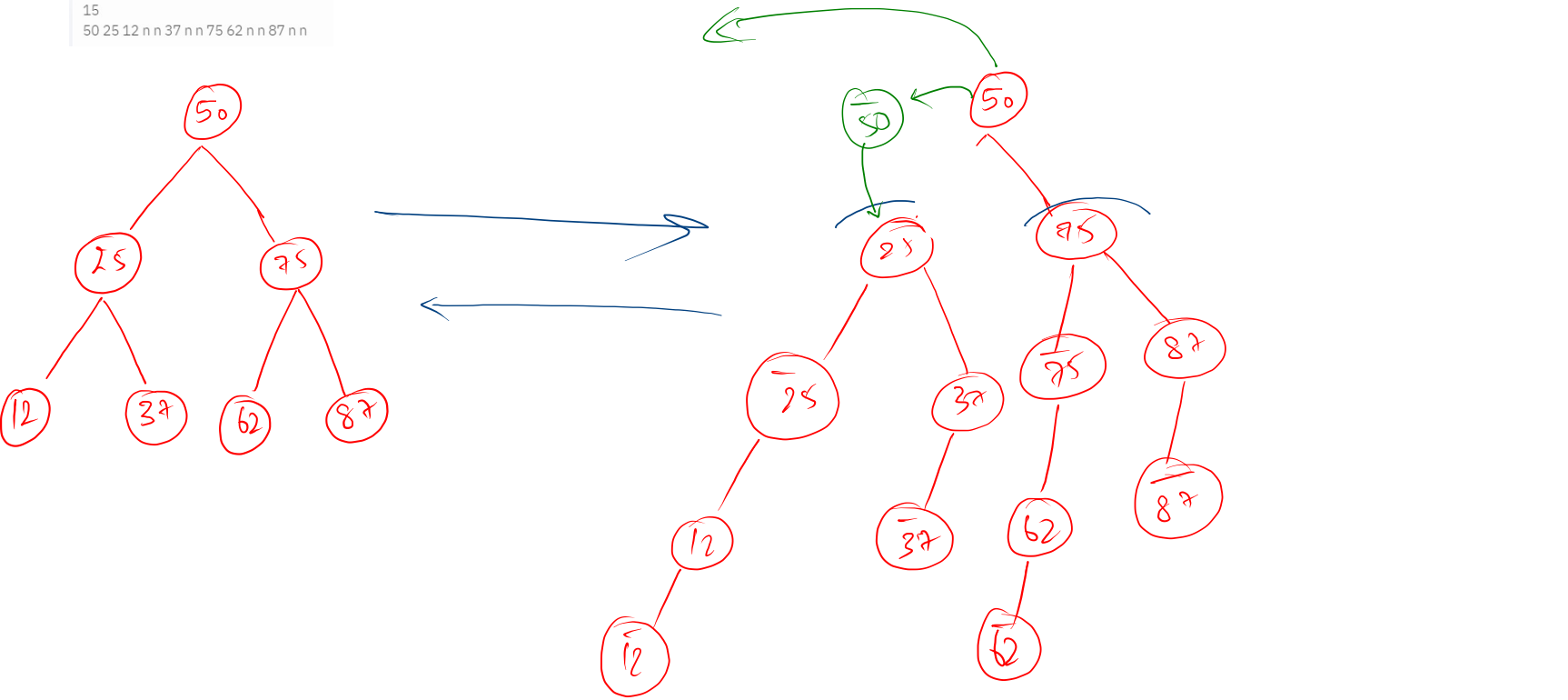
50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n

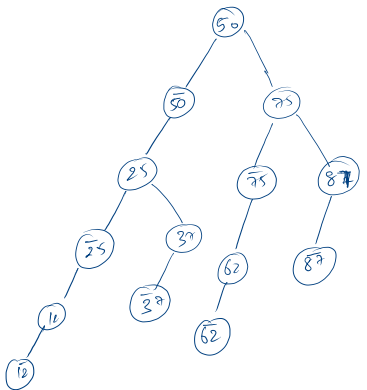
30 → data



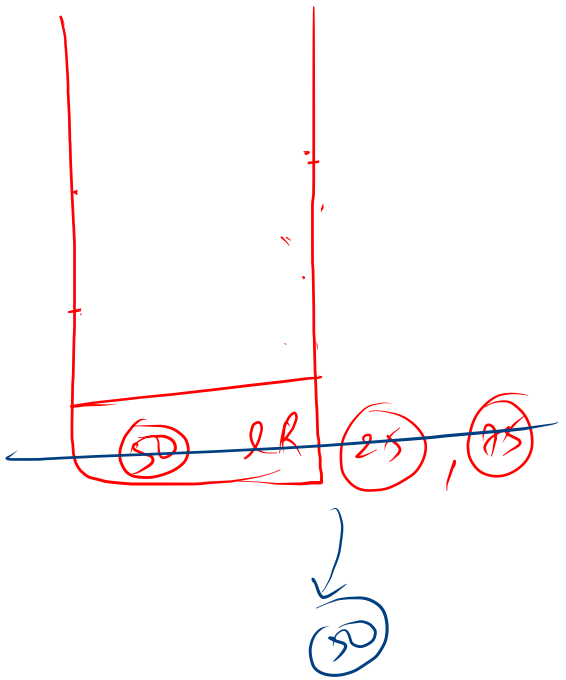


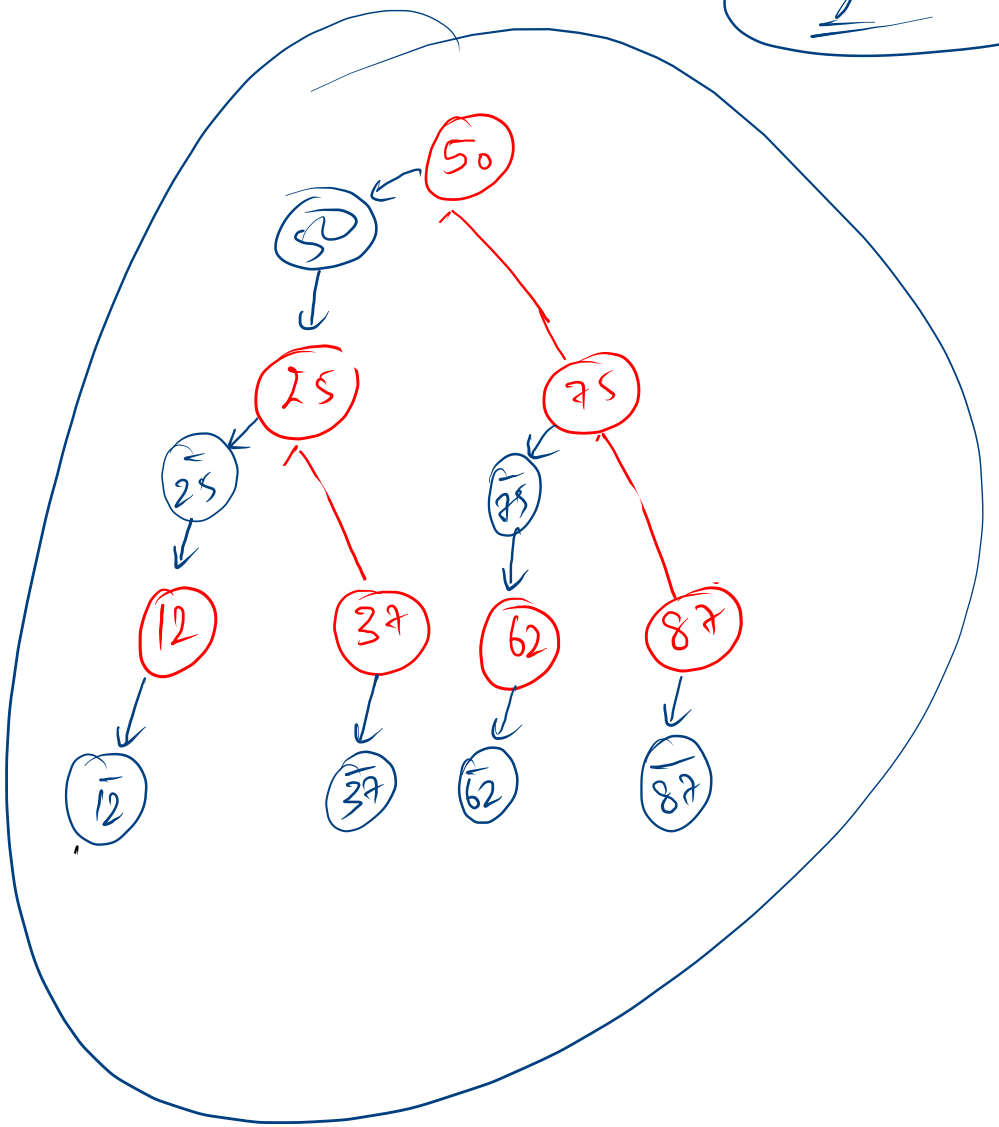
15
50 25 12 n n 37 n n 75 62 n n 87 n n

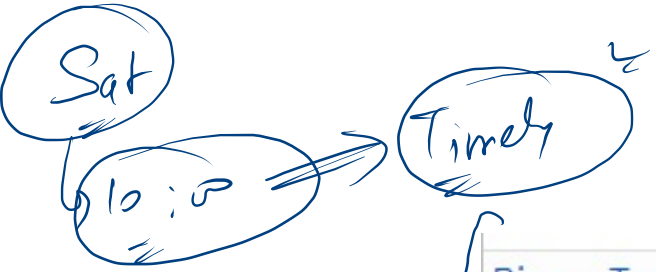




```java
public static Node createLeftCloneTree(Node node){
  if(node == null){
      return null;
  }
  Node lres = createLeftCloneTree(node.left);
  Node rres = createLeftCloneTree(node.right);

  Node clone = new Node(node.data,null,null);
  node.left = clone;
  clone.left = lres;

  return node;
}
```







Binary Tree - Introduction And Data Members	done
Binary Tree - Constructor	done
Display A Binary Tree	done
Size, Sum, Maximum And Height Of A Binary Tree	done
Find And Nodetorootpath In Binary Tree	done
Transform To Left-cloned Tree	done
Transform To Normal From Left-cloned Tree	logic
Traversals In A Binary Tree	HW
Levelorder Traversal Of Binary Tree	HW
Iterative Pre, Post And Inorder Traversals Of Binary Tree	HW