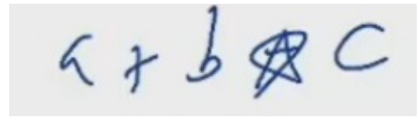
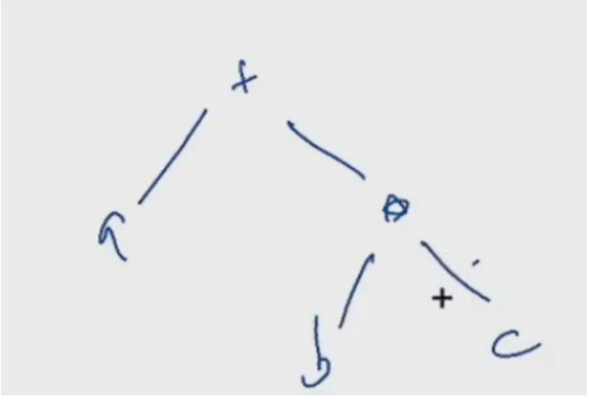
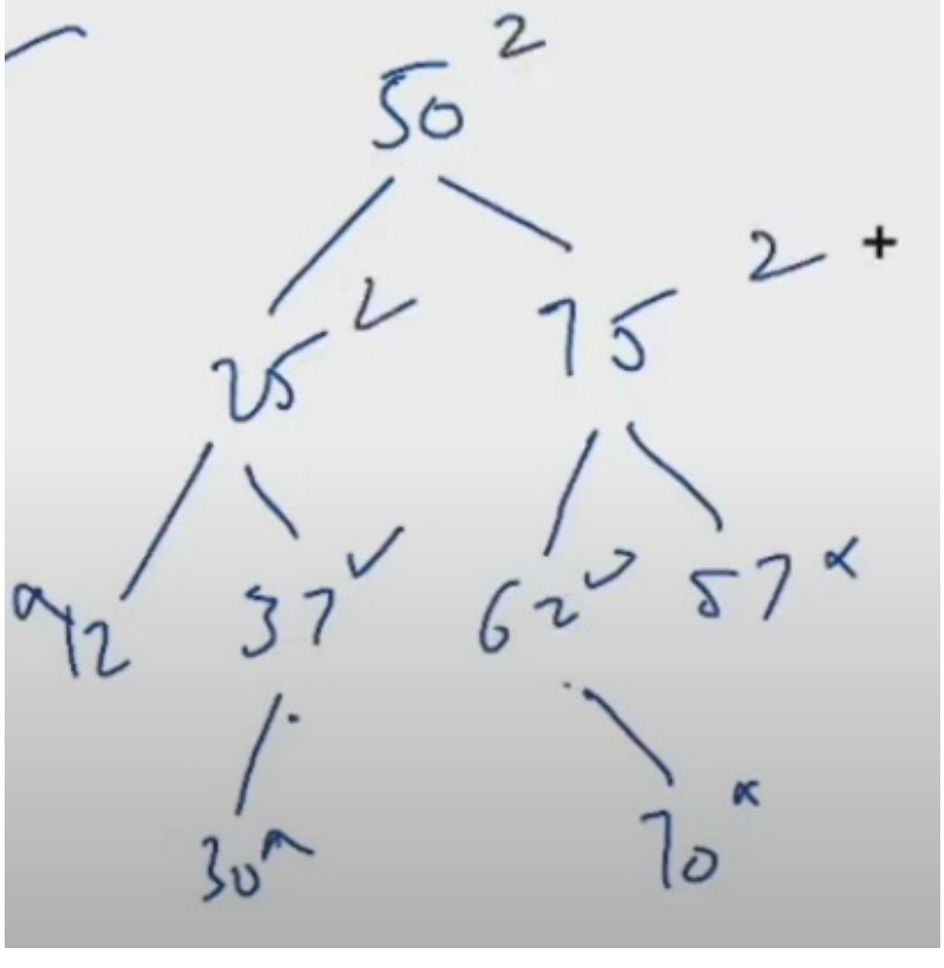


used in mathematical exp

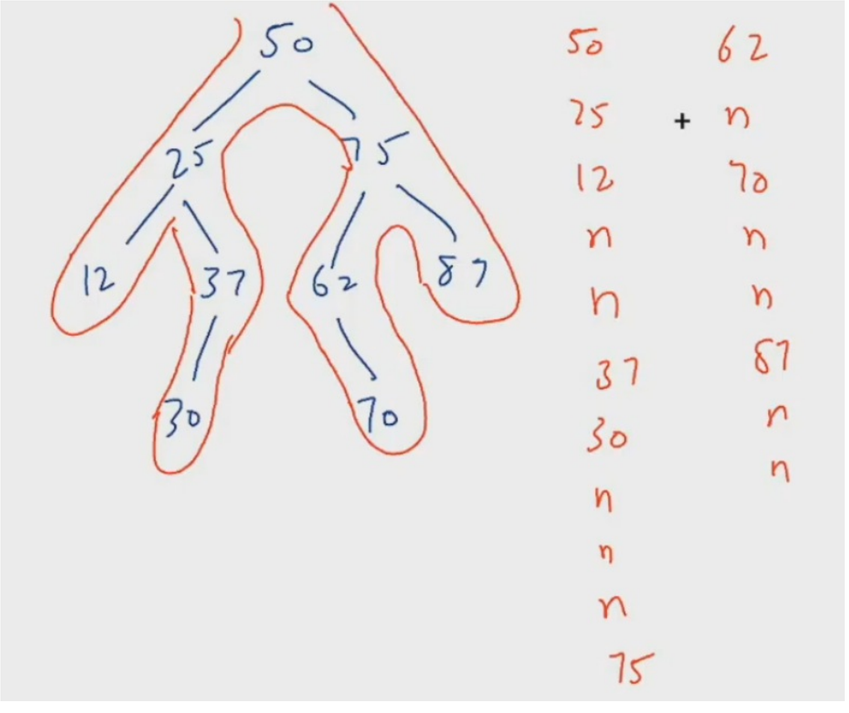
either no child
or
1 child
or
2 child





```
4  public class Main {
5      public static class Node {
6          int data;
7          Node left;
8          Node right;
9
10         Node(int data, Node left, Node right){
11             this.data = data;
12             this.left = left;
13             this.right = right;
14         }
15     }
16
17     public static void main(String[] args) throws
18     }
19
20 }
```

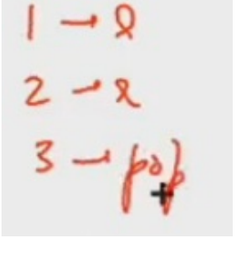
Binary Tree - Constructor | Data Structure and Algorithms in Java



this will be
given
you will need to
construct binary
tree using

null represent
no child

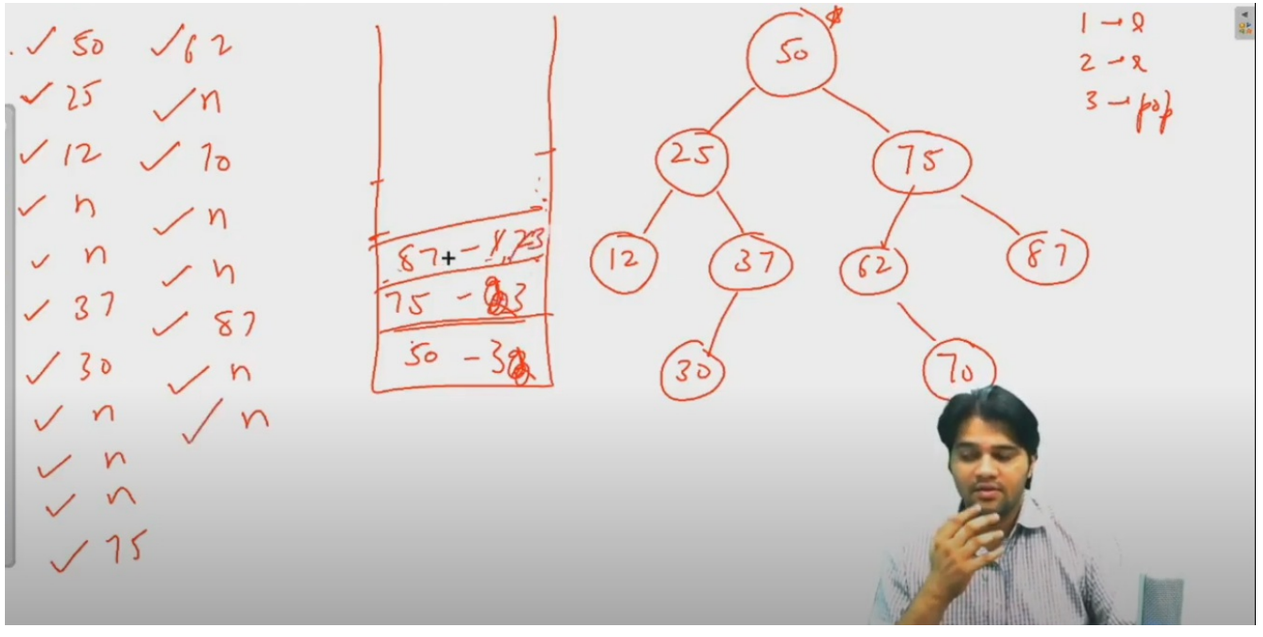
state



1->left pe lagana he
2->right pe lagana he

3->pop karna he

similary to generic tree pre orer and
post order iterative appraoch




```java
Stack<Pair> st = new Stack<>();
st.push(new Pair(root, 0));

int indx = 0;
while(st.size() > 0) {
    Pair p = st.peek();

    if(p.state == 0) {
        // left child processing
        indx++;
        if(arr[indx] != null) {
            Node nn = new Node(arr[indx]);
            p.node.left = nn;
            st.push(new Pair(nn, 0));
        }
        p.state++;
    } else if(p.state == 1) {
        // right child processing
        indx++;
        if(arr[indx] != null) {
            Node nn = new Node(arr[indx]);
            p.node.right = nn;
            st.push(new Pair(nn, 0));
        }
        p.state++;
    } else {
        // pop out node-pair from stack
        st.pop();
    }
}
return root;
}
```

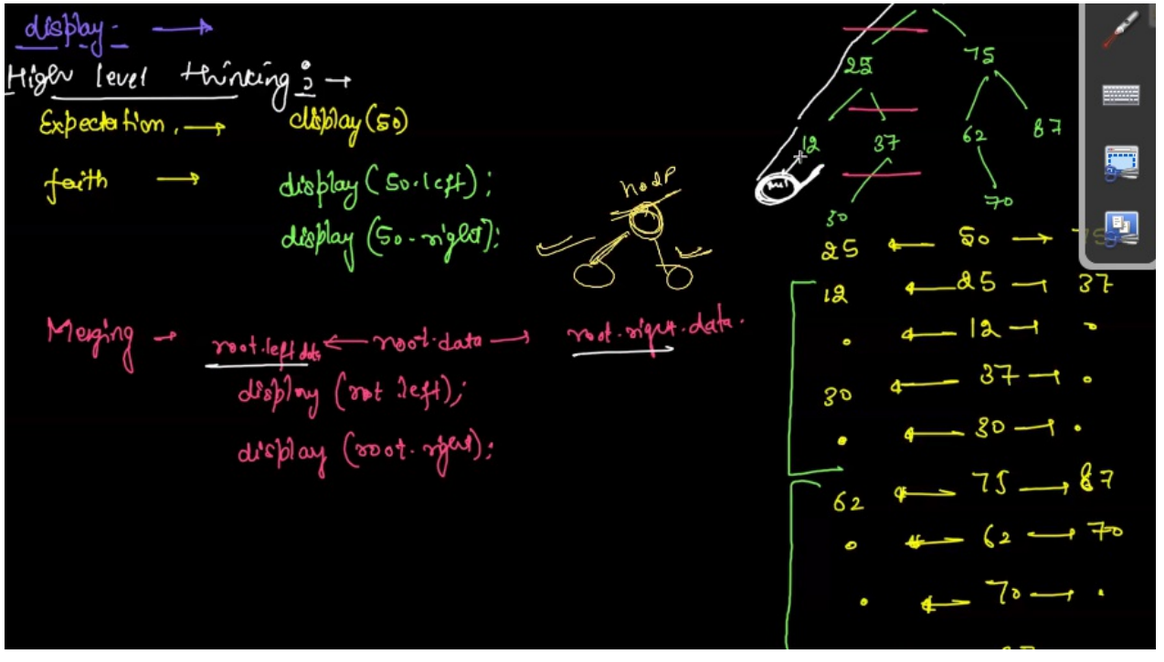
```java
public static void main(String[] args) throws Exception {
    Integer[] arr = {50, 25, 12, null, null, 37, 30, null, null, null, 75,

    Node root = new Node(arr[0], null, null);
    Pair rtp = new Pair(root, 1);

    Stack<Pair> st = new Stack<>();
    st.push(rtp);
```

```java
public static class Pair {
    Node node;
    int state;

    Pair(Node node, int state){
        this.node = node;
        this.state = state;
    }
}
```

```java
public static class Node {
    int data;
    Node left;
    Node right;

    Node(int data, Node left, Node right){
        this.data = data;
        this.left = left;
        this.right = right;
    }
}
```

```java
int idx = 0;
while(st.size() > 0){
    Pair top = st.peek();
    if(top.state == 1){
        idx++;
        if(arr[idx] != null){
            top.node.left = new Node(arr[idx], null, null);
            Pair lp = new Pair(top.node.left, 1);
            st.push(lp);
        } else {
            top.node.left = null;
        }

        top.state++;
    } else if(top.state == 2){
        idx++;
        if(arr[idx] != null){
            top.node.right = new Node(arr[idx], null, null);
            Pair rp = new Pair(top.node.right, 1);
            st.push(rp);
        } else {
            top.node.right = null;
        }

        top.state++;
    } else {
        st.pop();
    }
}
```

Display a Binary Tree | Data Structures and Algorithms in JAVA




```java
/* Given a binary tree, print its nodes in preorder */
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);

    /* first print data of node */

    /* then recur on left sutree */
    display(node.left);

    /* now recur on right subtree */
    display(node.right);
}
```

Size, Sum, Maximum And Height Of A Binary Tree

```java
public static int sum1(Node node) {
    // if(node == null) return 0; // root == null

    if(node.left != null && node.right != null) {
        int lsum = sum1(node.left);
        int rsum = sum1(node.right);
        return lsum + rsum + node.data;
    } else if(node.left != null) {
        int lsum = sum1(node.left);
        return lsum + node.data;
    } else if(node.right != null) {
        int rsum = sum1(node.right);
        return rsum + node.data;
    } else {
        return node.data;
    }
}
```

```java
public static int sum2(Node node) {

    int sum = 0;
    if(node.left != null) {
        sum += sum2(node.left);
    }

    if(node.right != null) {
        sum += sum2(node.right);
    }
    return sum + node.data;
}
```

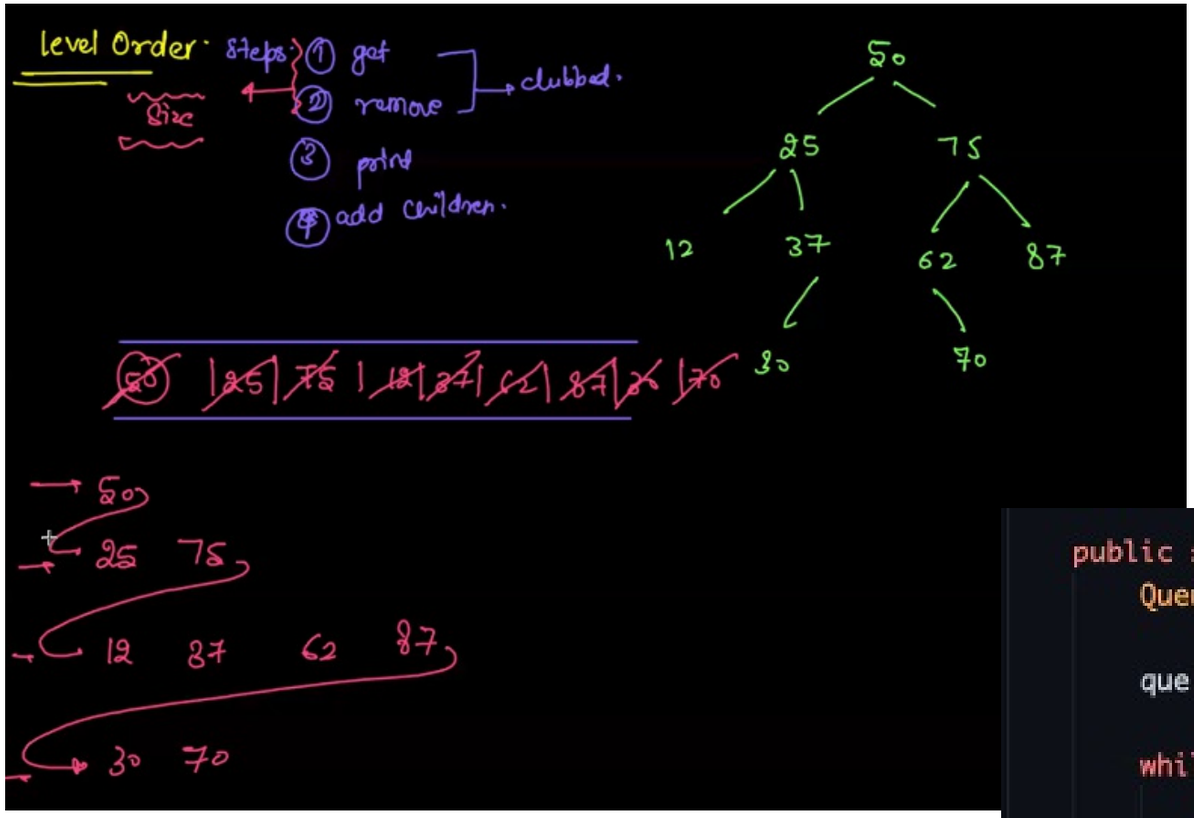
```java
public static int size(Node node) {
    // write your code here

    if (node == null)
        return 0;
    int ls = size(node.left);
    int rs = size(node.right);

    return ls + rs + 1;
}

public static int sum(Node node) {
    // write your code here

    if (node == null)
        return 0;
    int ls = sum(node.left);
    int rs = sum(node.right);

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

```java
public static int max(Node node) {
    // write your code here

    if (node == null)
        return Integer.MIN_VALUE;
    int ls = max(node.left);
    int rs = max(node.right);

    return Math.max(node.data, Math.max(ls, rs));
}

public static int height(Node node) {
    // write your code here

    if (node == null)
        return -1;
    int ls = height(node.left);
    int rs = height(node.right);

    return Math.max(ls, rs) + 1;
}
```

Levelorder Traversal Of Binary Tree

refer generic tree for 3 approach

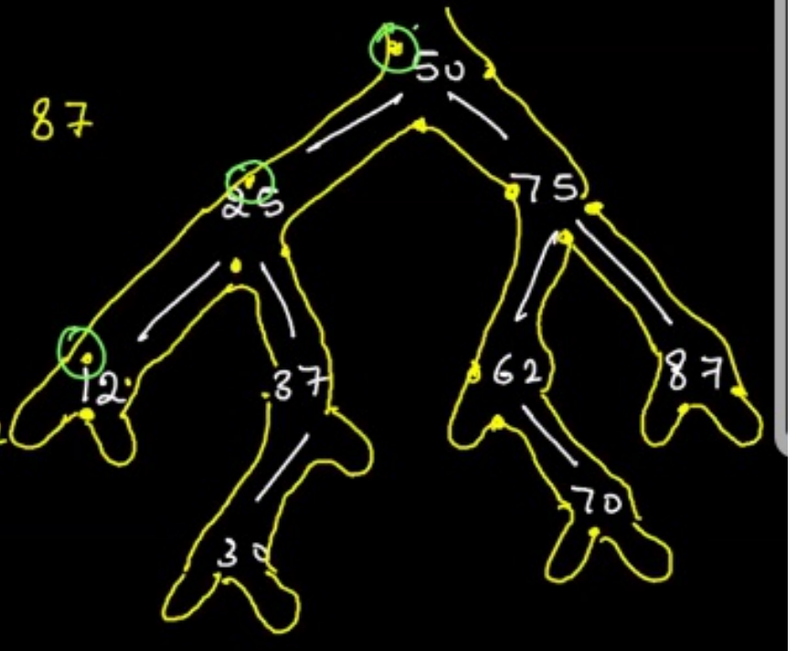
rpa




```java
public static void levelOrder(Node node) {
    Queue<Node> que = new ArrayDeque<>();

    que.add(node);

    while(que.size() > 0) {
        int sz = que.size();

        while(sz-- > 0) {
            // 1. get + remove
            Node rem = que.remove();
            // 2. print
            System.out.print(rem.data + " ");
            // 3. add children
            if(rem.left != null)
                que.add(rem.left);

            if(rem.right != null)
                que.add(rem.right);
        }
        System.out.println();
    }
}
```

Pre order 50 25 12 37 30 75 62 70 87

Pre Area



Inorder: 12 25 30 37 50 62 70 75 87

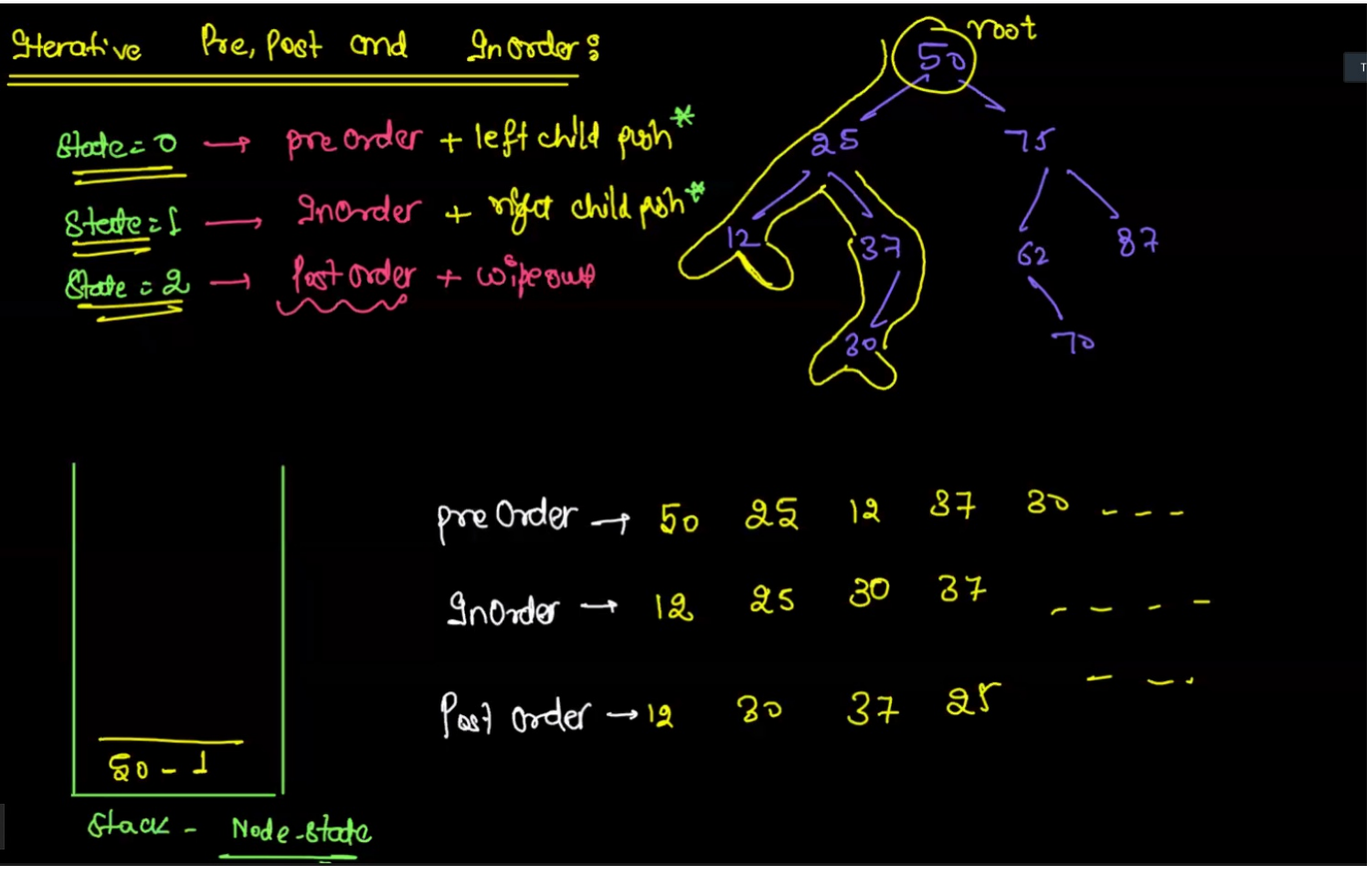
In Area } Area between left and right call

PostOrder 12 30 37 25 70 62 87 75 50

Area after both calls.

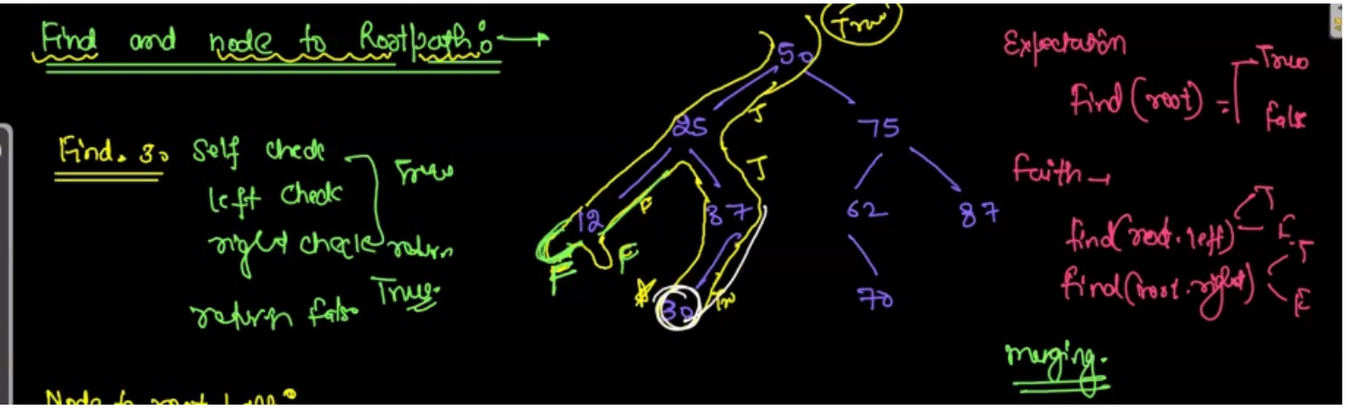
```java
// preAre -> area before all calls
public static void preOrder(Node root) {
    if(root == null) return;
    System.out.print(root.data + " ");
    preOrder(root.left);
    preOrder(root.right);
}

// inAre -> area between the calls i.e. left and right calls
public static void inOrder(Node root) {
    if(root == null) return;
    inOrder(root.left);
    System.out.print(root.data + " ");
    inOrder(root.right);
}

// postArea -> area after all calls
public static void postOrder(Node root) {
    if(root == null) return;
    postOrder(root.left);
    postOrder(root.right);
    System.out.print(root.data + " ");
}
```

Iterative Pre, Post And Inorder Traversals Of Binary Tree




```java
ArrayList<Integer> pre = new ArrayList<>();
ArrayList<Integer> in = new ArrayList<>();
ArrayList<Integer> post = new ArrayList<>();

st.push(new Pair(node, 0));

while(st.size() > 0) {
    Pair p = st.peek();
    if(p.state == 0) {
        pre.add(p.node.data);
        p.state++;
        if(p.node.left != null) {
            st.push(new Pair(p.node.left, 0));
        }
    } else if(p.state == 1) {
        in.add(p.node.data);
        p.state++;
        if(p.node.right != null) {
            st.push(new Pair(p.node.right, 0));
        }
    } else {
        post.add(p.node.data);
        st.pop();
    }
}
```

Find And Nodetorootpath In Binary Tree

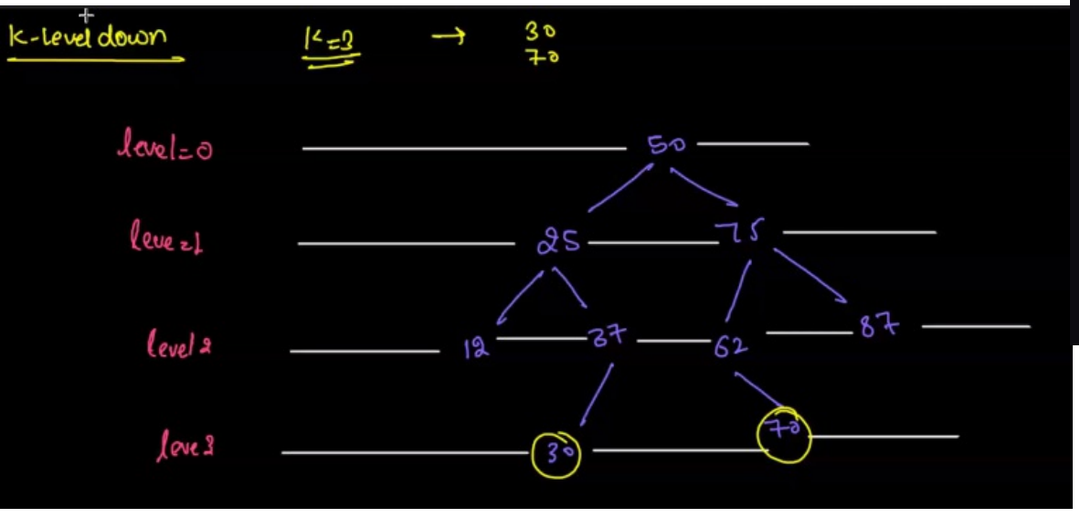



```java
public static boolean find(Node node, int data) {
    if(node == null) return false;

    if(node.data == data) return true;

    boolean res = false;

    res = find(node.left, data);
    res = res || find(node.right, data);

    return res;
}
```

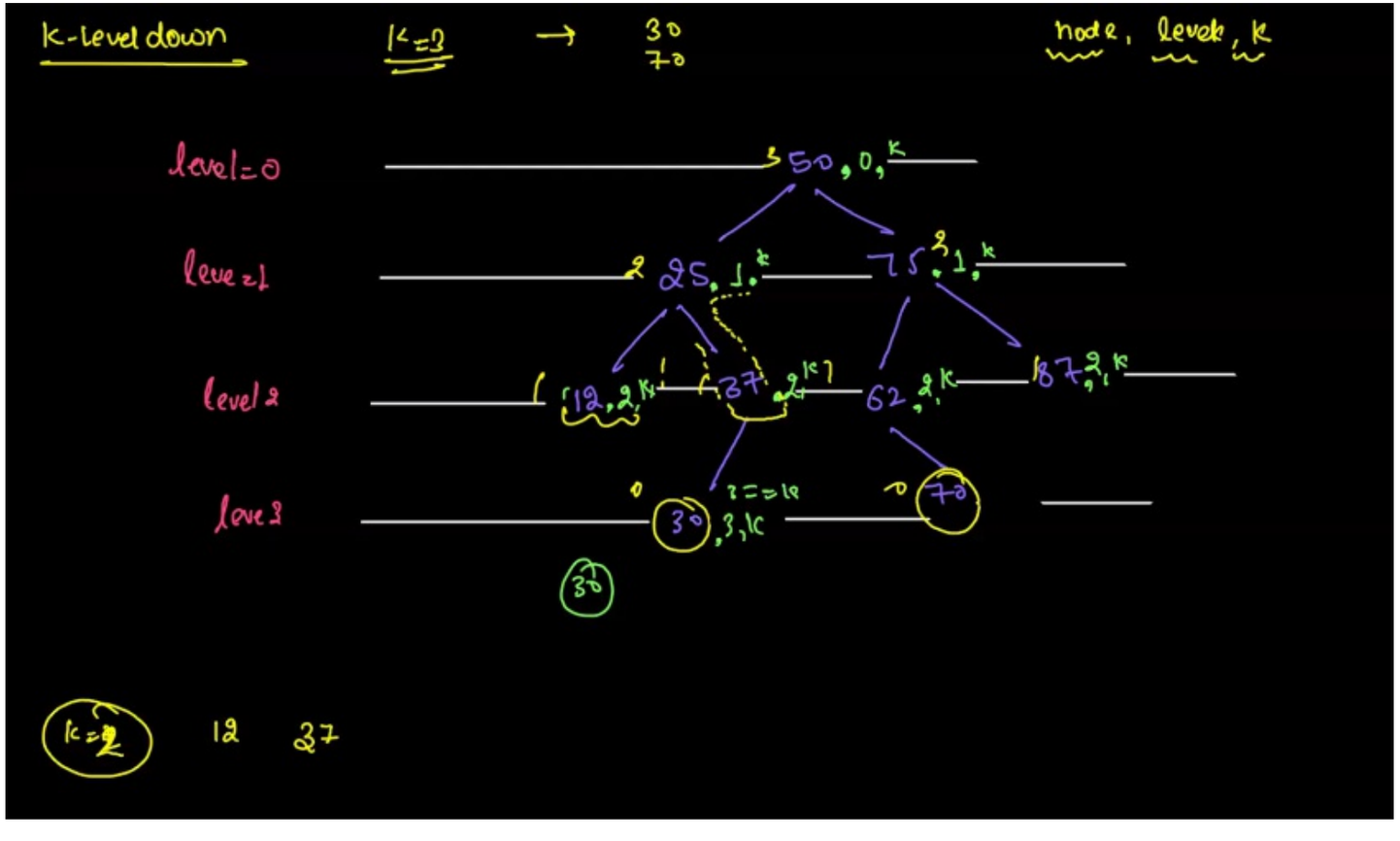
```java
public static ArrayList<Integer> nodeToRootPath(Node node, int data) {
    if(node == null) return new ArrayList<>();

    if(node.data == data) {
        ArrayList<Integer> bres = new ArrayList<>();
        bres.add(node.data);
        return bres;
    }

    ArrayList<Integer> lres = nodeToRootPath(node.left, data);
    if(lres.size() > 0) {
        lres.add(node.data);
        return lres;
    }

    ArrayList<Integer> rres = nodeToRootPath(node.right, data);
    if(rres.size() > 0) {
        rres.add(node.data);
        return rres;
    }

    return new ArrayList<>();
}
```

```java
public static boolean find(Node node, int data) {
    if(node == null) return false;

    if(node.data == data) return true;

    // boolean res = false;

    // res = find(node.left, data);
    // res = res || find(node.right, data);

    // return res;

    boolean lres = find(node.left, data);
    if(lres == true) return true;

    boolean rres = find(node.right, data);
    if(rres == true) return true;

    return false;
}
```

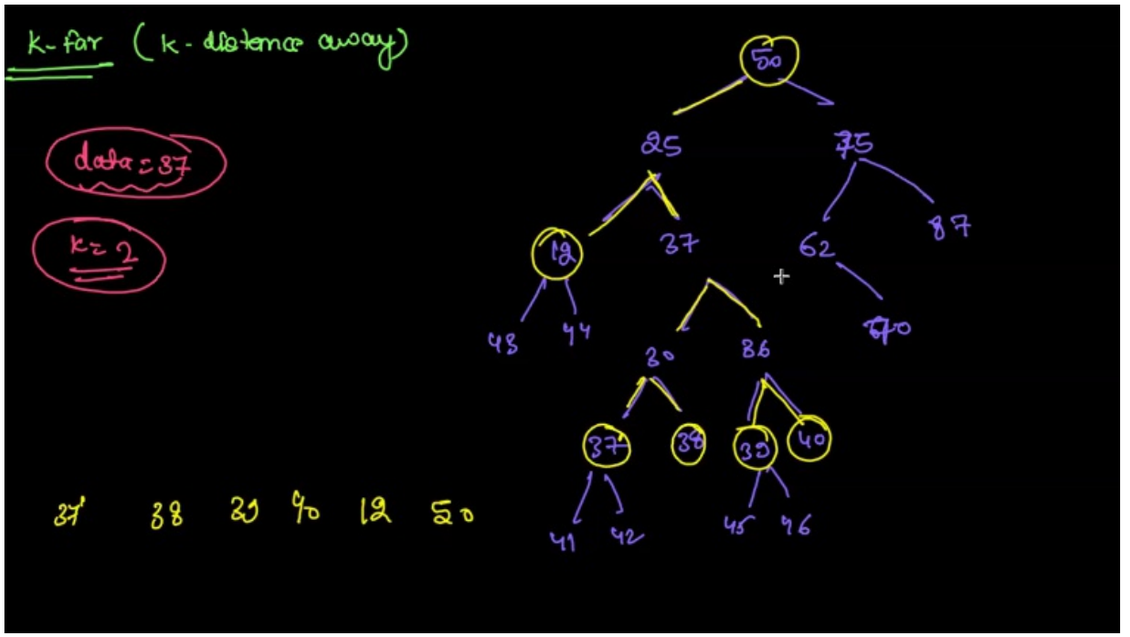
Print K Levels Down

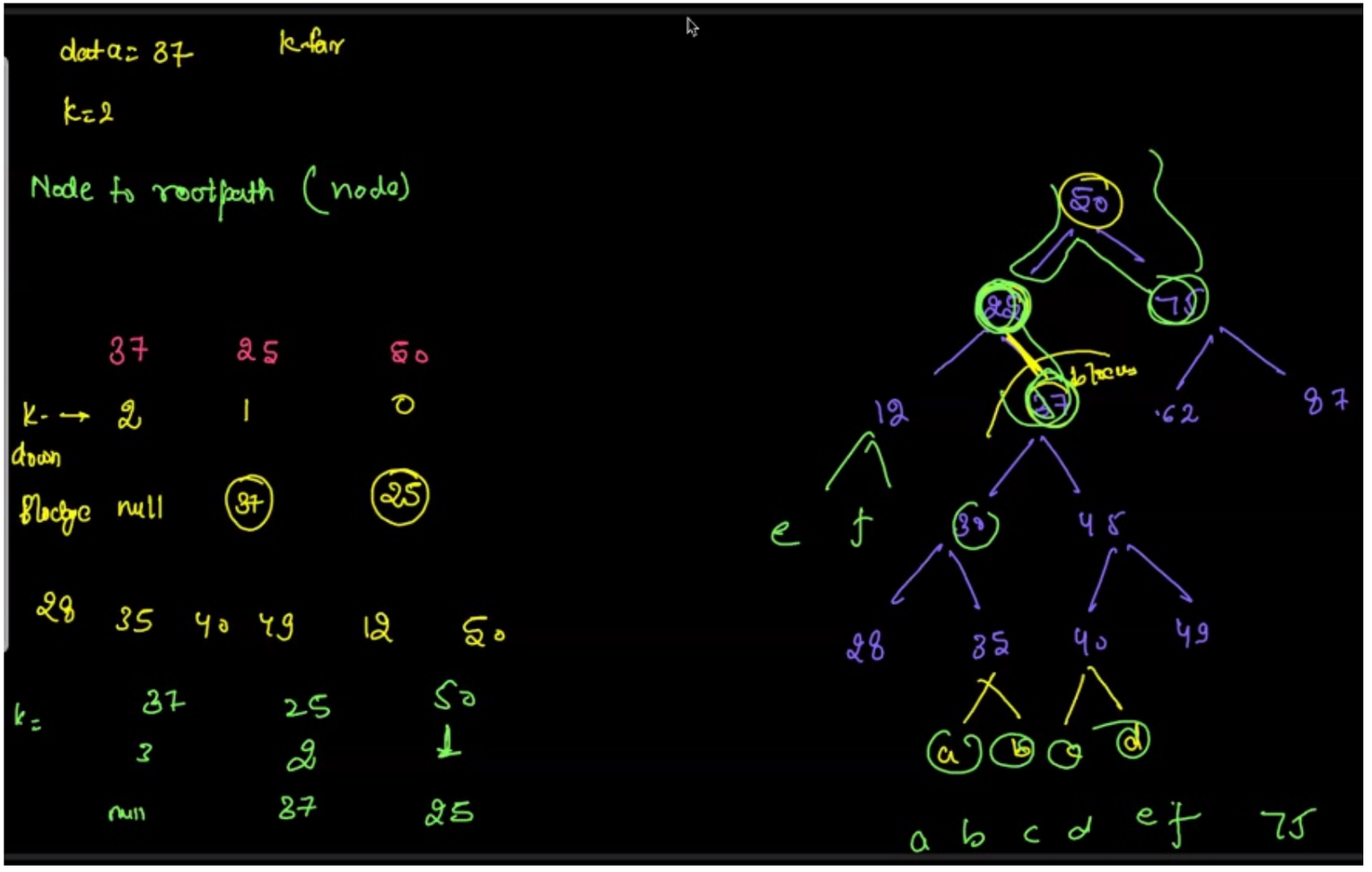



```java
public static void printKLevelsDown(Node node, int k){
    if(node == null) return;

    if(k == 0) {
        System.out.println(node.data);
        return;
    }

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



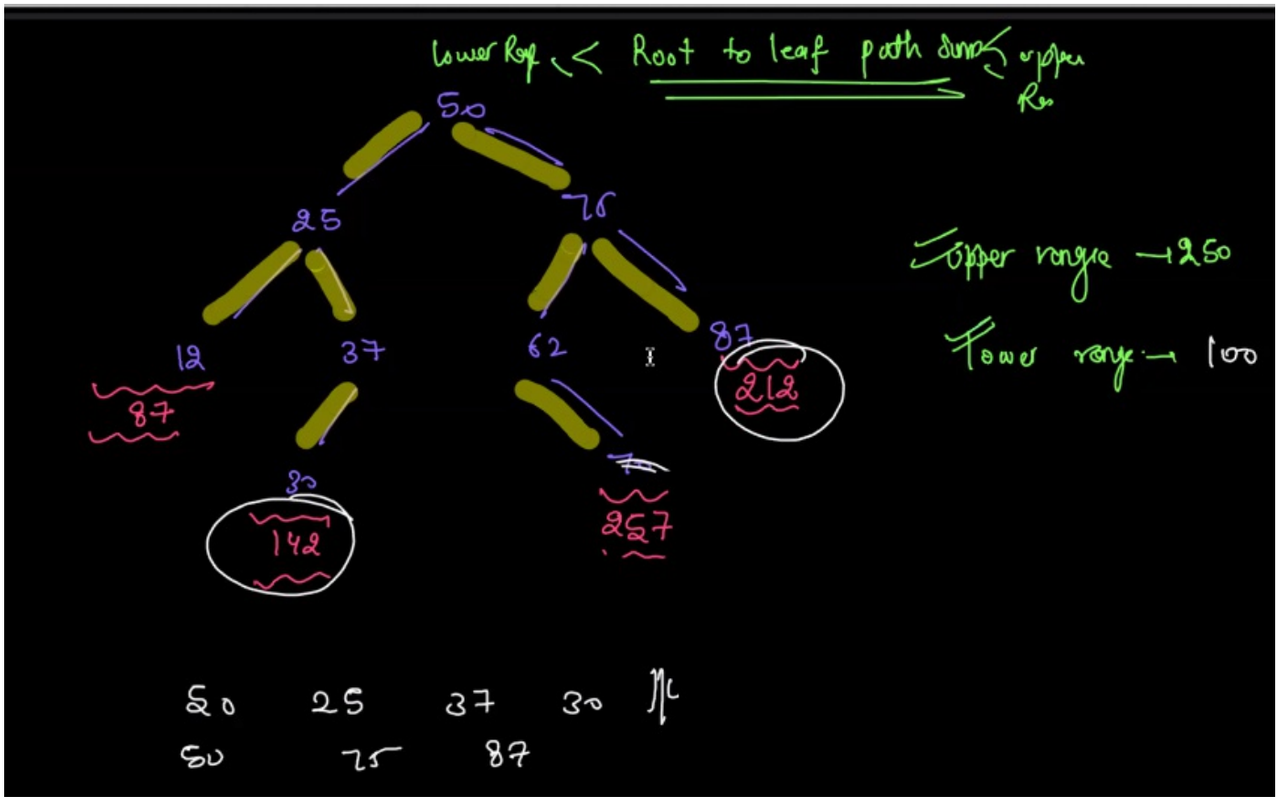
Print Nodes K Distance Away





```java
public static void printKDown(Node node, Node blockage, int k) {
    if(node == null || node == blockage) return;

    if(k == 0) {
        System.out.println(node.data);
        return;
    }

    printKDown(node.left, blockage, k - 1);
    printKDown(node.right, blockage, k - 1);
}

public static void printKNodesFar(Node root, int data, int k) {
    ArrayList<Node> n2rp = nodeToRoot(root, data);

    Node blockage = null;
    for(int i = 0; i < n2rp.size(); i++) {
        Node node = n2rp.get(i);
        printKDown(node, blockage, k);
        k--;
        blockage = node;
    }
}
```

Path To Leaf From Root In Range




```java
public static void pathToLeafFromRoot(Node node, String path, int sum, int lo, int hi) {
    // write your code here
    if (sum > hi) {
        return;
    }
    if (node.left == null && node.right == null) {
        sum = sum + node.data;
        if (sum <= hi && sum >= lo) {
            System.out.println(path + node.data);
            // System.out.println(sum);

        }
        return;
    }

    if (node.left != null) {

        pathToLeafFromRoot(node.left, path + node.data + " ", sum + node.data, lo, hi);
    }
    if (node.right != null) {

        pathToLeafFromRoot(node.right, path + node.data + " ", sum + node.data, lo, hi);
    }
}
```

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

    if(node.left != null && node.right != null) {
        pathToLeafFromRoot(node.left, path + node.data + " ", sum + node.data, lo, hi);
        pathToLeafFromRoot(node.right, path + node.data + " ", sum + node.data, lo, hi);
    } else if(node.left != null) {
        pathToLeafFromRoot(node.left, path + node.data + " ", sum + node.data, lo, hi);
    } else if(node.right != null) {
        pathToLeafFromRoot(node.right, path + node.data + " ", sum + node.data, lo, hi);
    } else {
        // leaf
        sum += node.data;
        path += node.data;
        if(lo <= sum && sum <= hi) {
            // print path
            System.out.println(path);
        }
    }
}
```