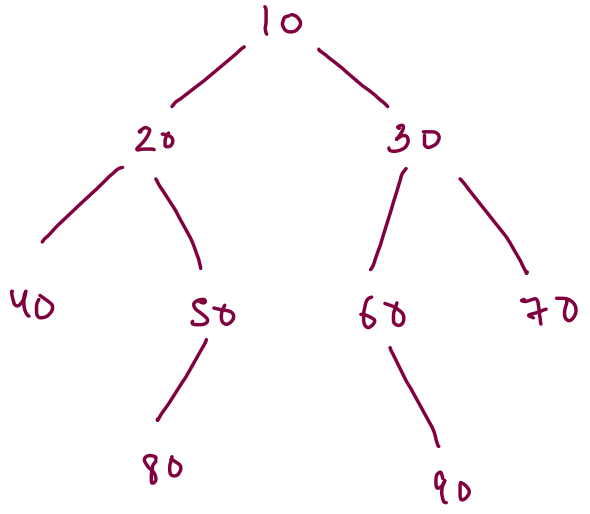
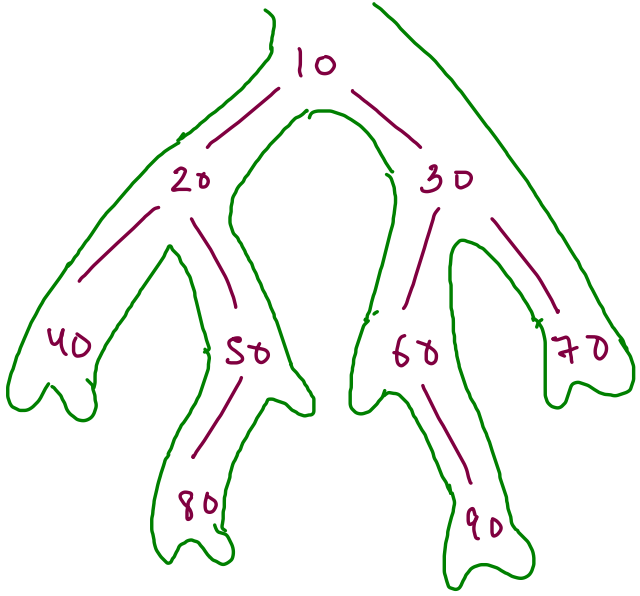
Binary Tree - Introduction And Data Members

↳ each node has atmost 2 child




```
Node {
    int data;
    Node left;
    Node right;
}
```

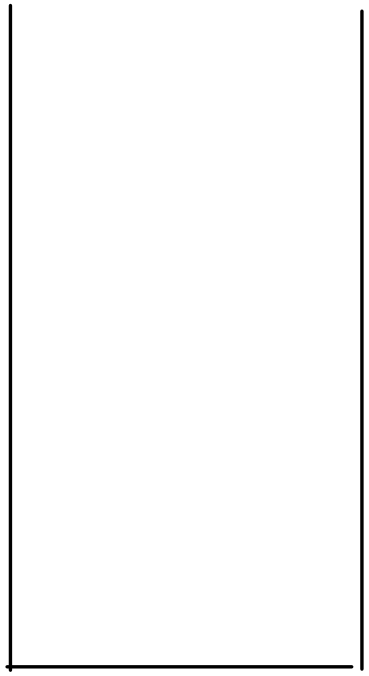
preorder



arr: 10 20 40 -1 -1 50 80 -1 -1

-1 30 60 -1 90 -1 -1 70 -1 -1

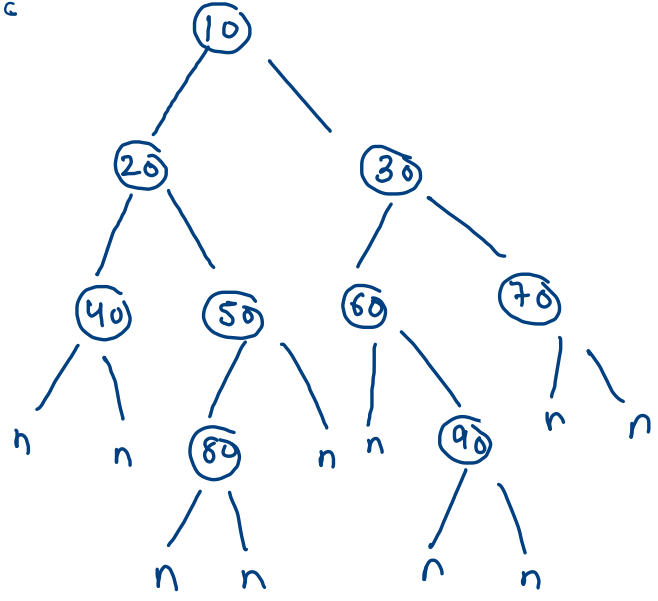
arr: 10 20 40 -1 -1 50 80 -1 -1 -1 30 60 -1 90 -1 -1 70 -1 -1



0 -> waiting for dc

1 -> waiting for rc

2 -> done, pop



root = 10

Pair {

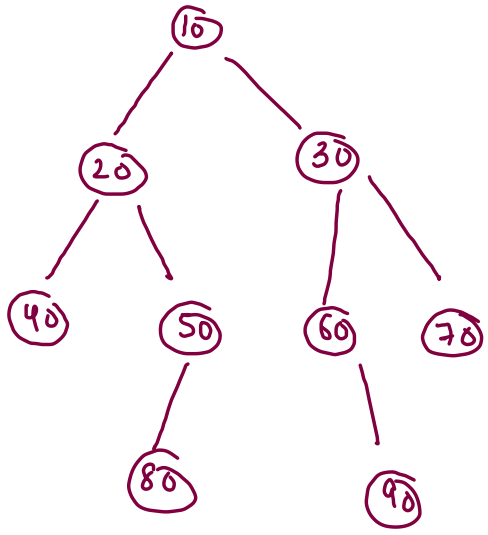
Node node;

int state;

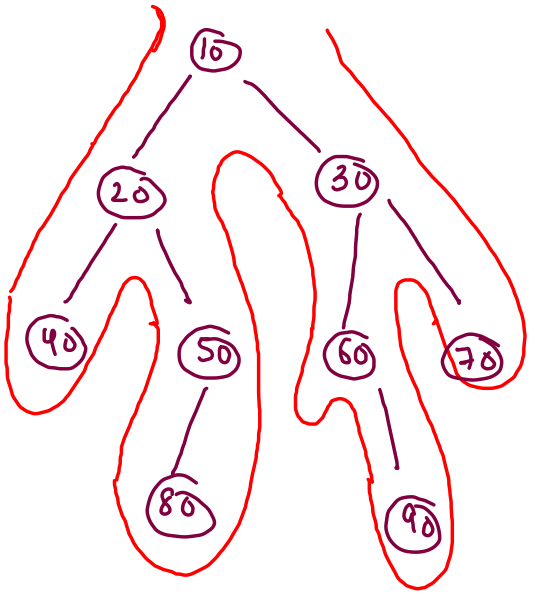
}

10 20 40 -1 -1 50 80 -1 -1 -1 30 60 -1 90 -1 -1 70 -1 -1

```
while(st.size() > 0) {
    Pair top = st.peek();
    if(top.st == 0) {
        //top's node is waiting lc
        if(arr[idx] == -1) {
            top.node.left = null;
        }
        else {
            Node node = new Node(arr[idx]);
            top.node.left = node;
            Pair lcp = new Pair(node,0);
            st.push(lcp);
        }
        top.st++;
        idx++;

    }
    else if(top.st == 1){
        //top's node is waiting rc
        if(arr[idx] == -1) {
            top.node.right = null;
        }
        else {
            Node node = new Node(arr[idx]);
            top.node.right = node;
            Pair rcp = new Pair(node,0);
            st.push(rcp);
        }
        top.st++;
        idx++;
    }
    else {
        //top's node is done
        st.pop();
    }
}
```



root = 10



display(20)

20 ← 10 → 30

40 ← 20 → 50

. ← 40 → .

80 ← 50 → .

. ← 80 → .

display(30)

60 ← 30 → 70

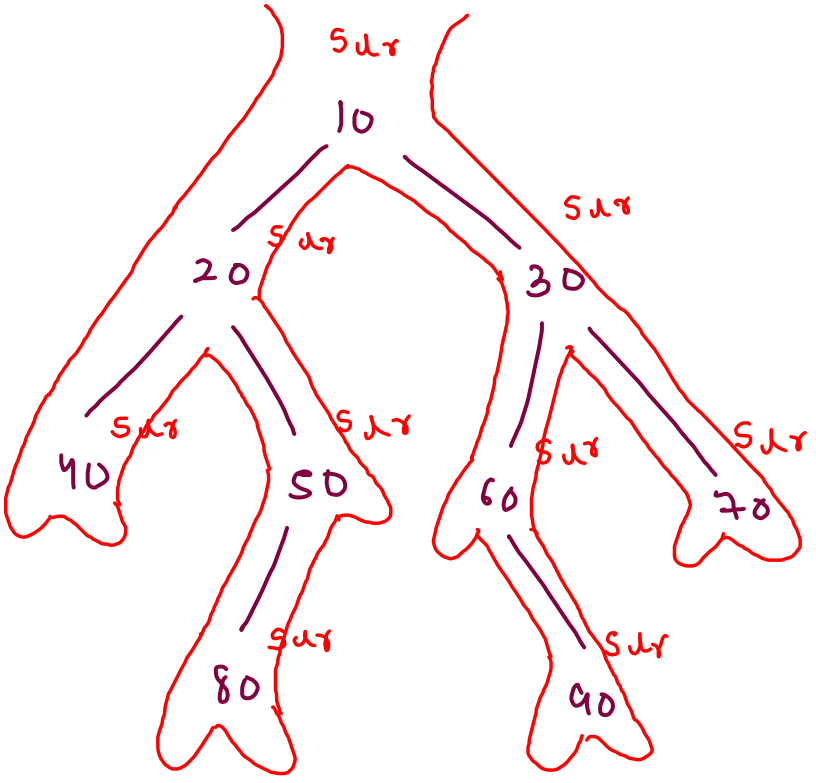
. ← 60 → 90

. ← 90 → .

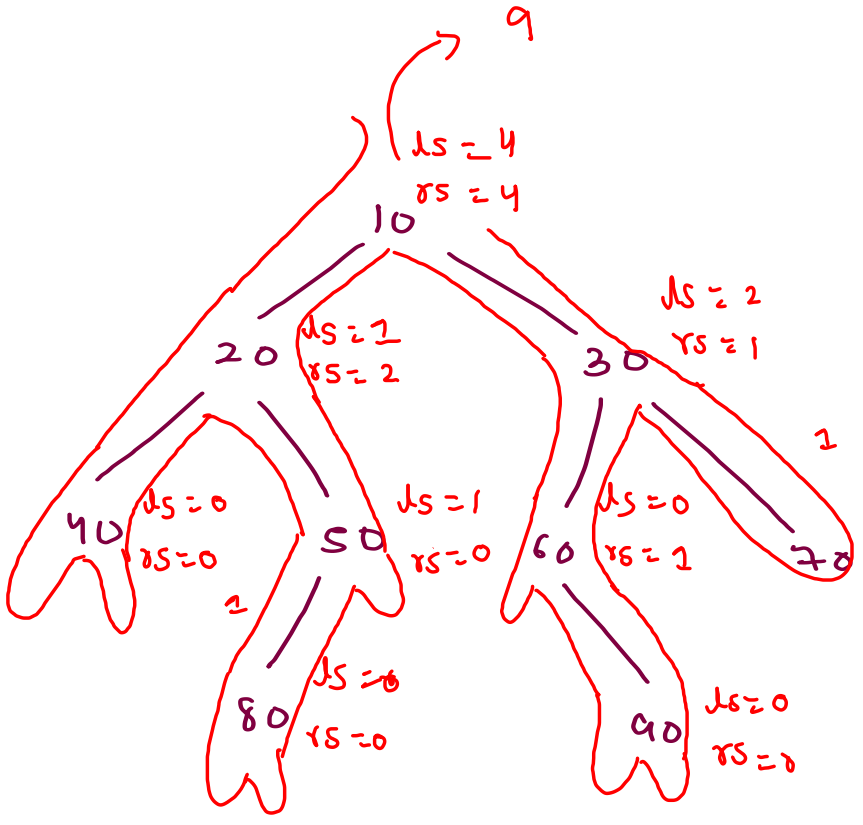
. ← 70 → .

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

    String mp = " <- " + node.data + " -> ";
    String lp = (node.left == null) ? "." : (node.left.data + "");
    String rp = (node.right == null) ? "." : (node.right.data + "");

    System.out.println(lp + mp + rp);

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

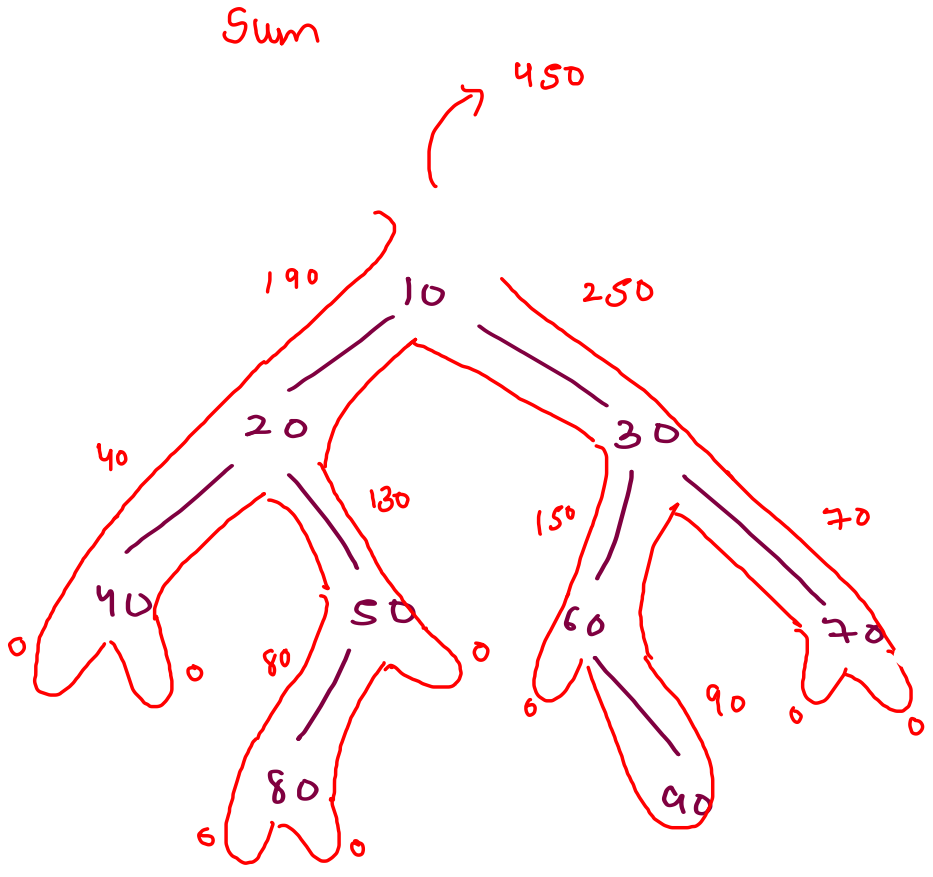
```
20 <- 10 -> 30
40 <- 20 -> 50
. <- 40 -> .
80 <- 50 -> .
. <- 80 -> .
60 <- 30 -> 70
. <- 60 -> 90
. <- 90 -> .
. <- 70 -> .
```



Size

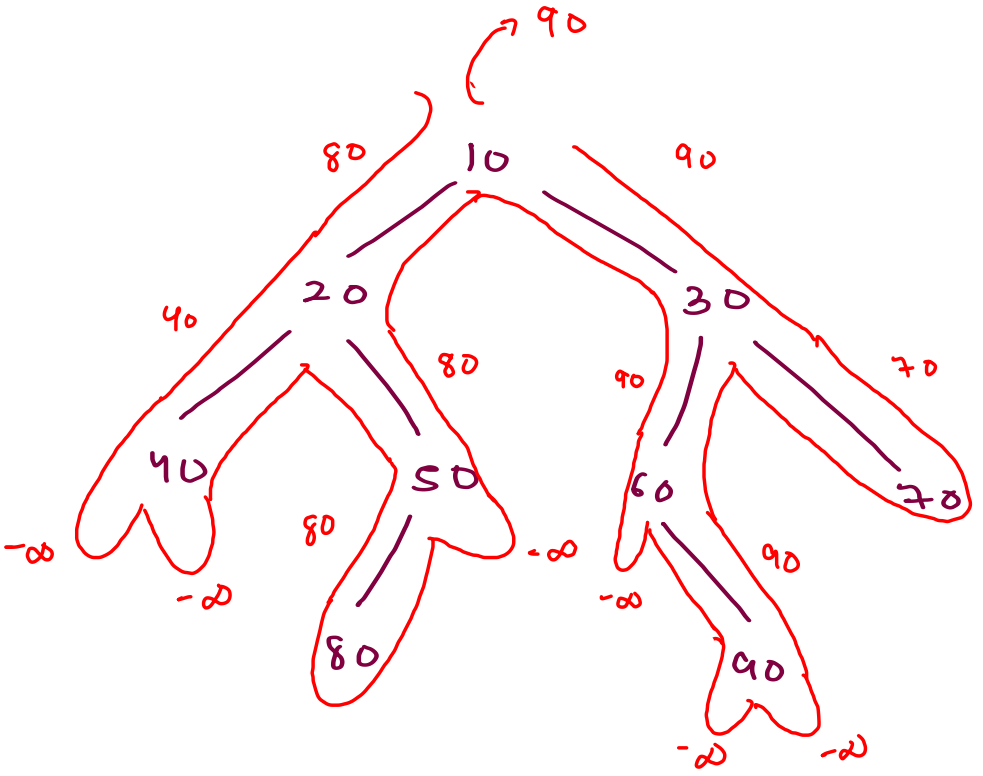



```java
public static int size(Node node) {
    if(node == null) {
        return 0;
    }

    int ls = size(node.left); //left subtree size
    int rs = size(node.right); //right subtree size

    return ls + rs + 1;
}
```

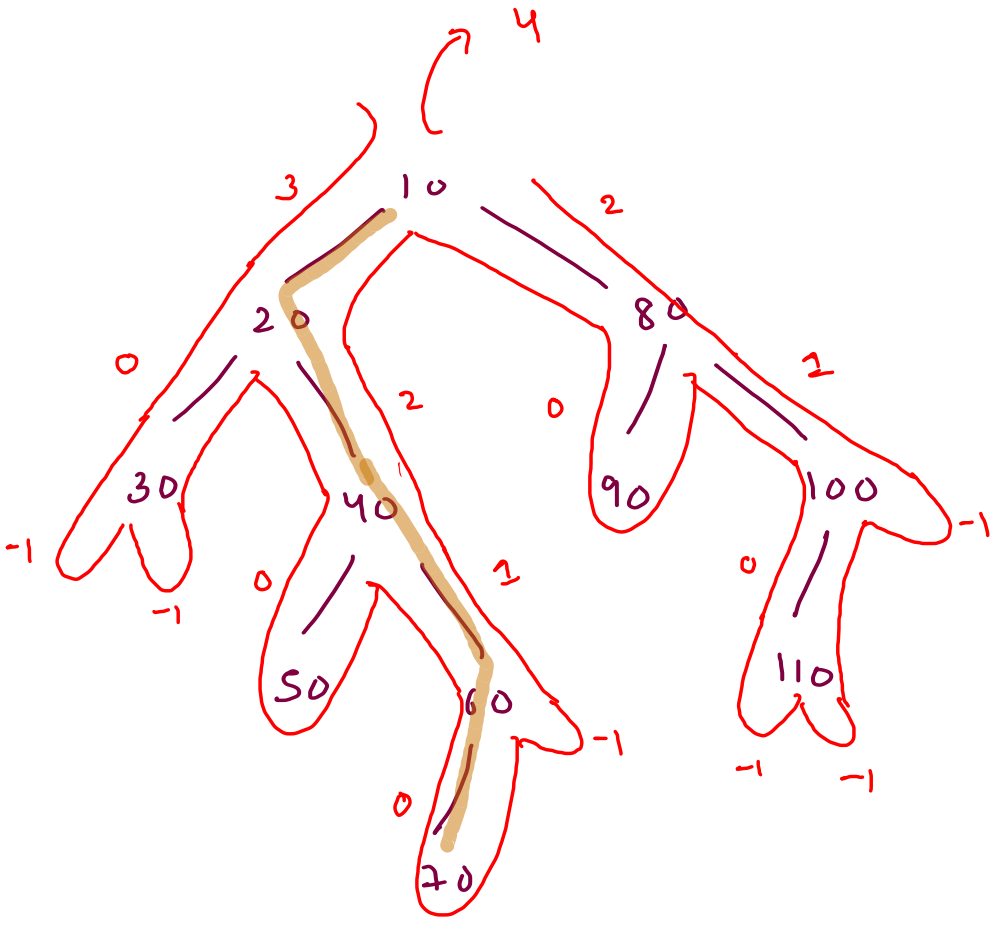



```java
public static int sum(Node node) {
    if(node == null) {
        return 0;
    }

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

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

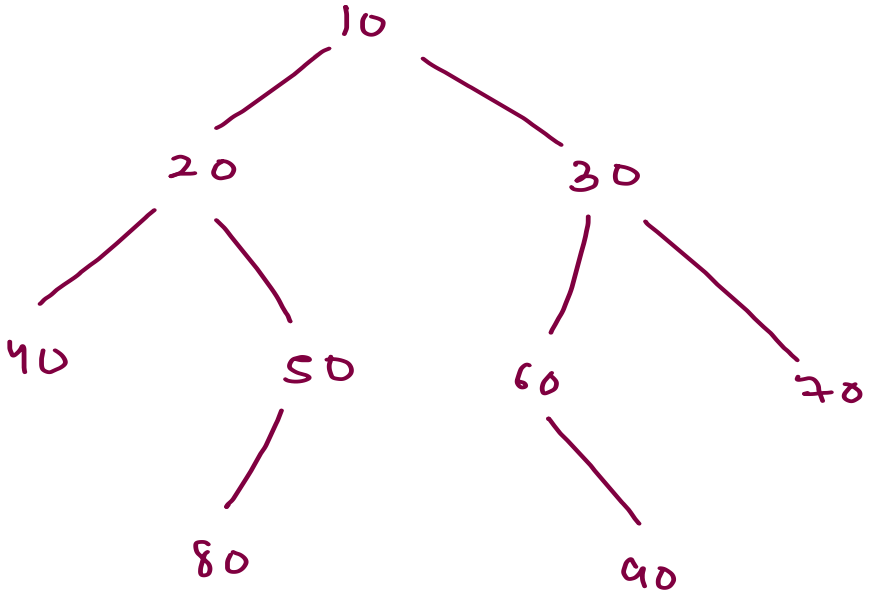
```java
public static int max(Node node) {
    if(node == null) {
        return Integer.MIN_VALUE;
    }

    int lmax = max(node.left); //left subtree max
    int rmax = max(node.right); //right subtree max

    return Math.max(Math.max(lmax,rmax),node.data));
}
```

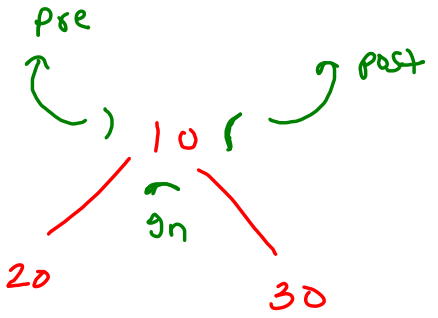


```java
public static int height(Node node) {
    if(node == null) {
        return -1;
    }

    int lht = height(node.left); //left subtree height
    int rht = height(node.right); //right subtree height

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

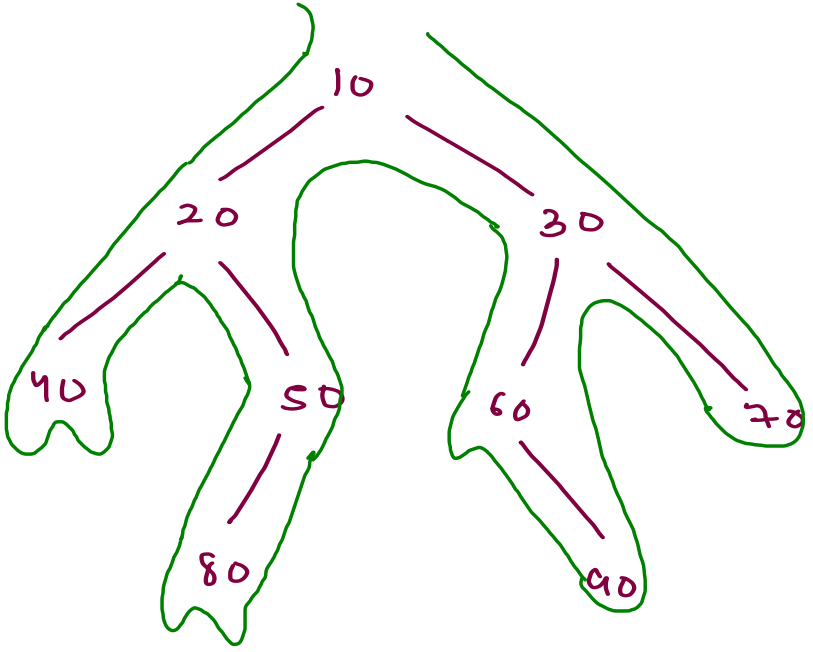


Traversals In A Binary Tree





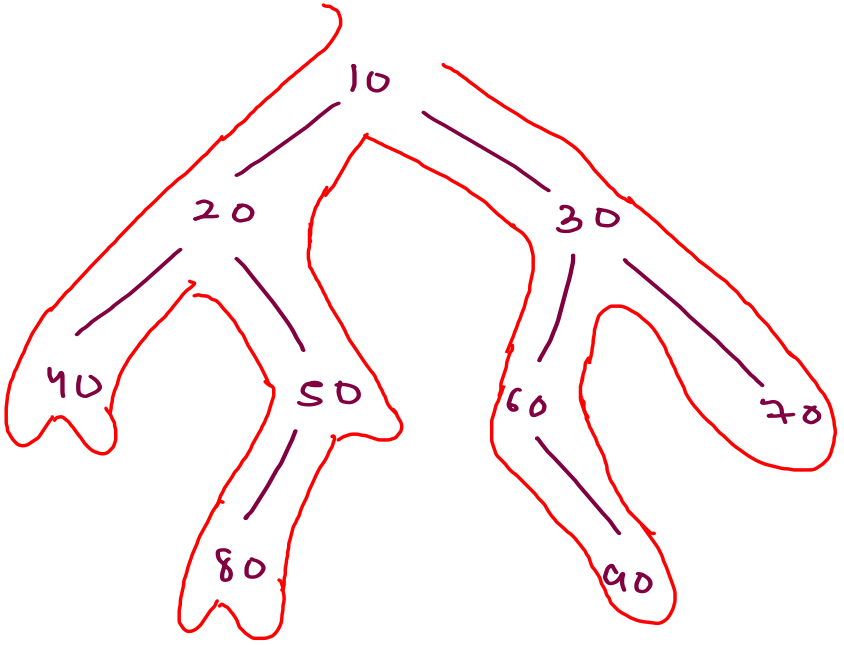

```
fun (node) {
    if (node == null) return;
    // pre
    fun(node.left);
    // in
    fun(node.right);
    // post
}
```



pre: N L R

10 20 40 50 80 30 60 90 70

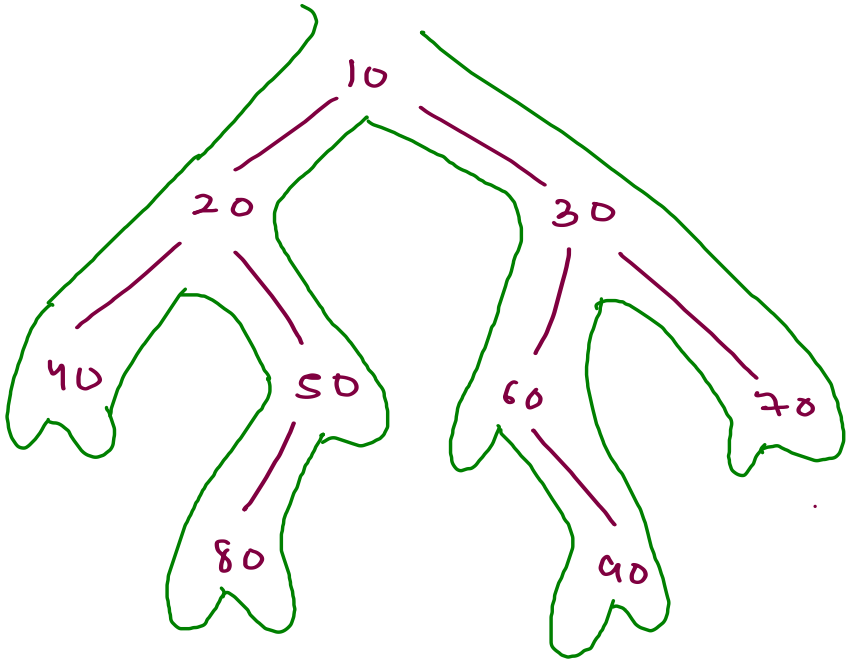
N L R



post : L R N

40 80 50 20 90 60 70 30 10

L R N



Inorder : LNR

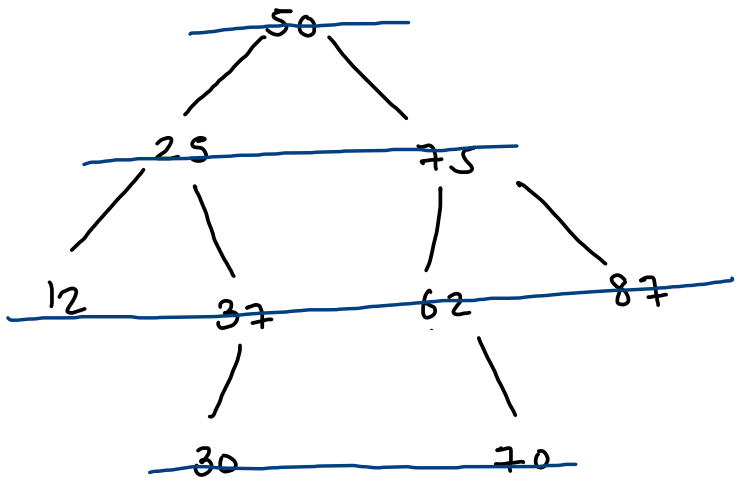
40 20 80 50 10 60 90 30 70

L N R

Levelorder Traversal Of Binary Tree

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

50
25 75
12 37 62 87
30 70

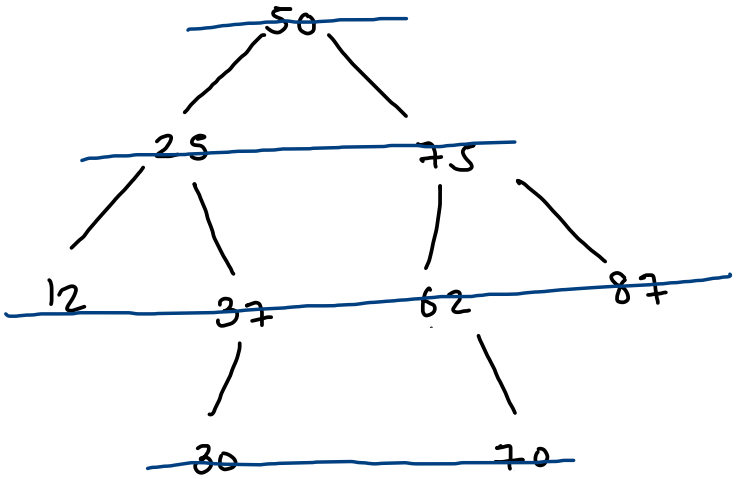


50

25 75

12 37 62 87

30 70



level order line-wise

c = 2

50

25 75

12 37 62 87

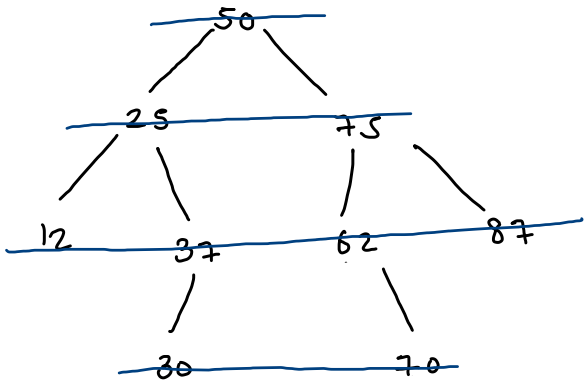
30 70

50	25	75	12	37	62	87	30	70

count times

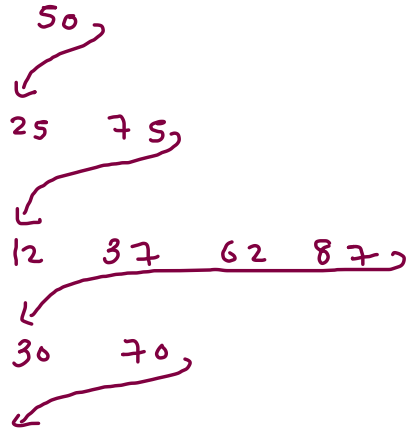
- remove
- work
- add children

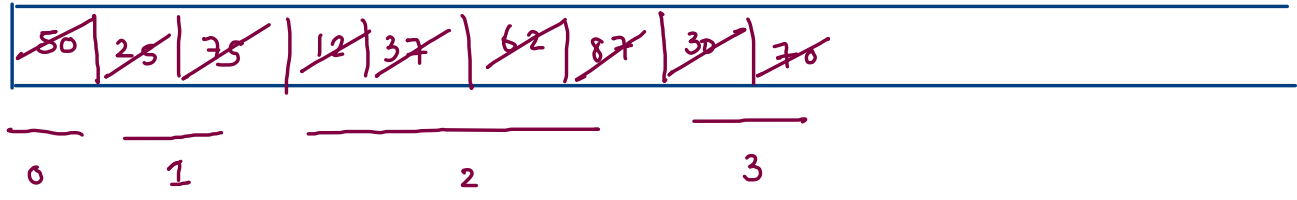
Syso line




```
while(q.size() > 0) {
    int count = q.size();

    //----------------------k--------------------

    for(int i=0; i < count;i++) {
        //remove
        Node rem = q.remove();

        //work
        System.out.print(rem.data + " ");

        //add children
        if(rem.left != null) {
            q.add(rem.left);
        }
        if(rem.right != null) {
            q.add(rem.right);
        }
    }

    //---------------------(k+1)------------------
    System.out.println();
}
```

C = 2



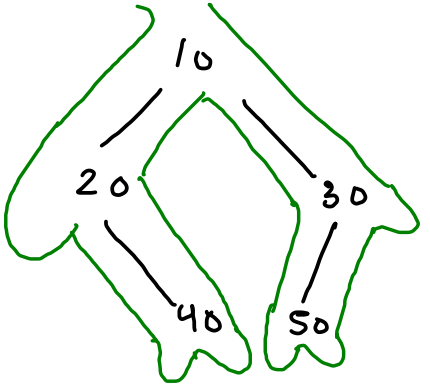


Iterative Pre, Post And Inorder Traversals Of Binary Tree

0 : pre, st++, go to left

1 : in, st++, go to right

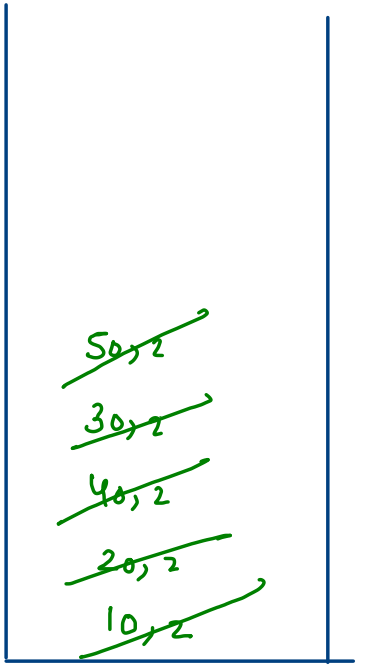
2 : post, pop



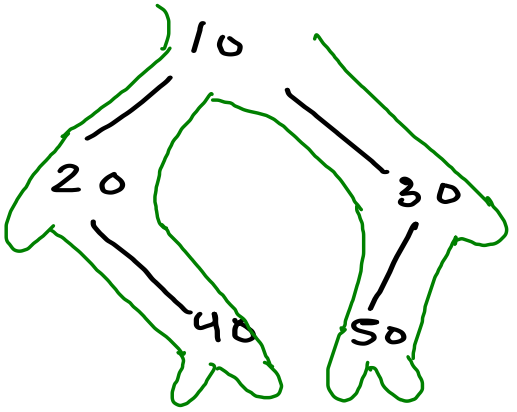
pre : 10 20 40 30 50

in : 20 40 10 50 30

post : 40 20 50 30 10



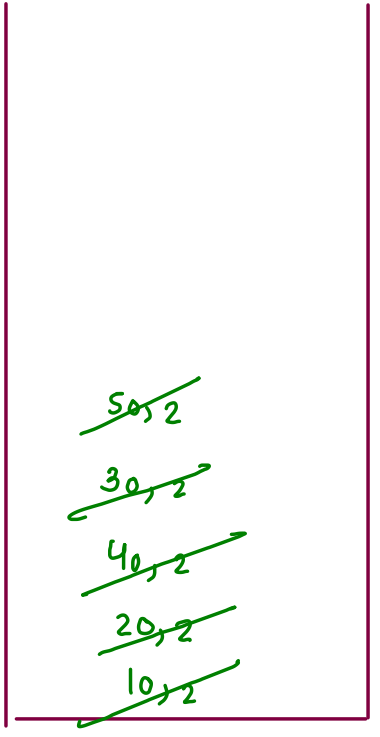
```
while(st.size() > 0) {
    Pair top = st.peek();

    if(top.state == 0) {
        //pre
        pre += (top.node.data + " ");
        top.state++;

        if(top.node.left != null) {
            Pair lcp = new Pair(top.node.left,0);
            st.push(lcp);
        }
    }
    else if(top.state == 1) {
        //in
        in += (top.node.data + " ");
        top.state++;

        if(top.node.right != null) {
            Pair rcp = new Pair(top.node.right,0);
            st.push(rcp);
        }
    }
    else if(top.state == 2) {
        //post
        post += (top.node.data + " ");
        st.pop();
    }
}
```



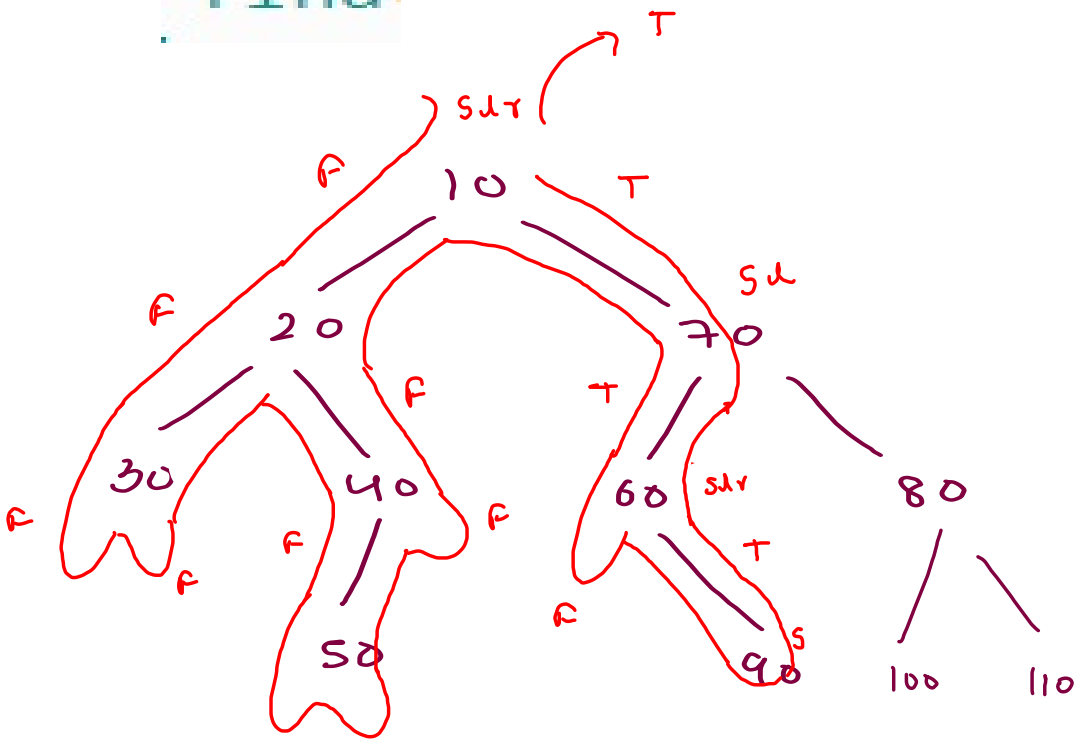
Pre : 10 20 40 30 50

In : 20 40 10 50 30

post : 40 20 50 30 10

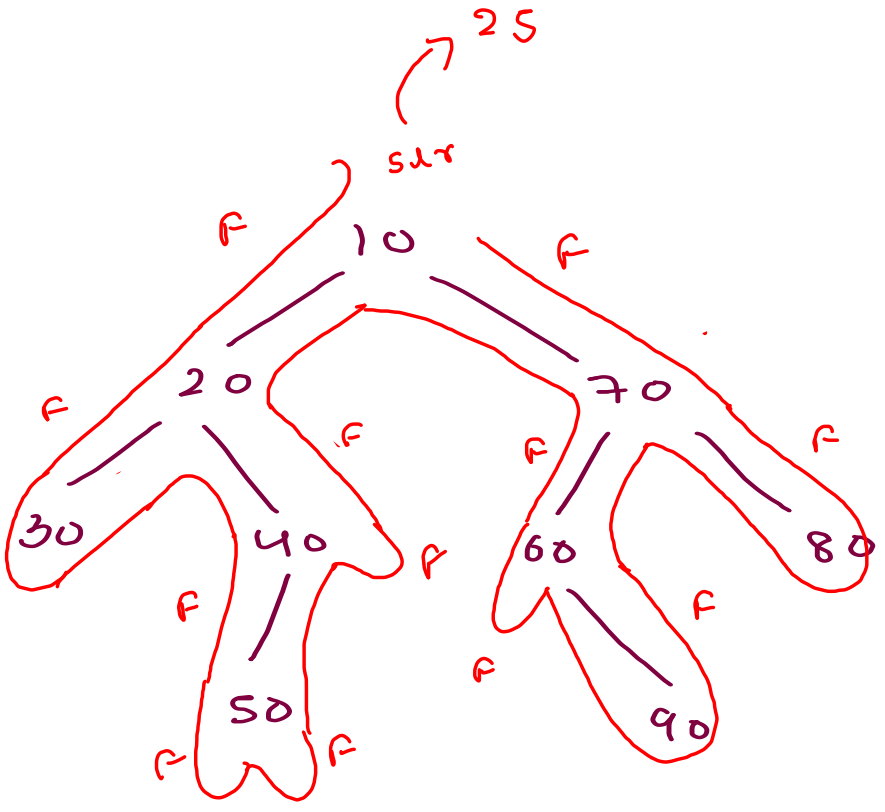


find



data = 90

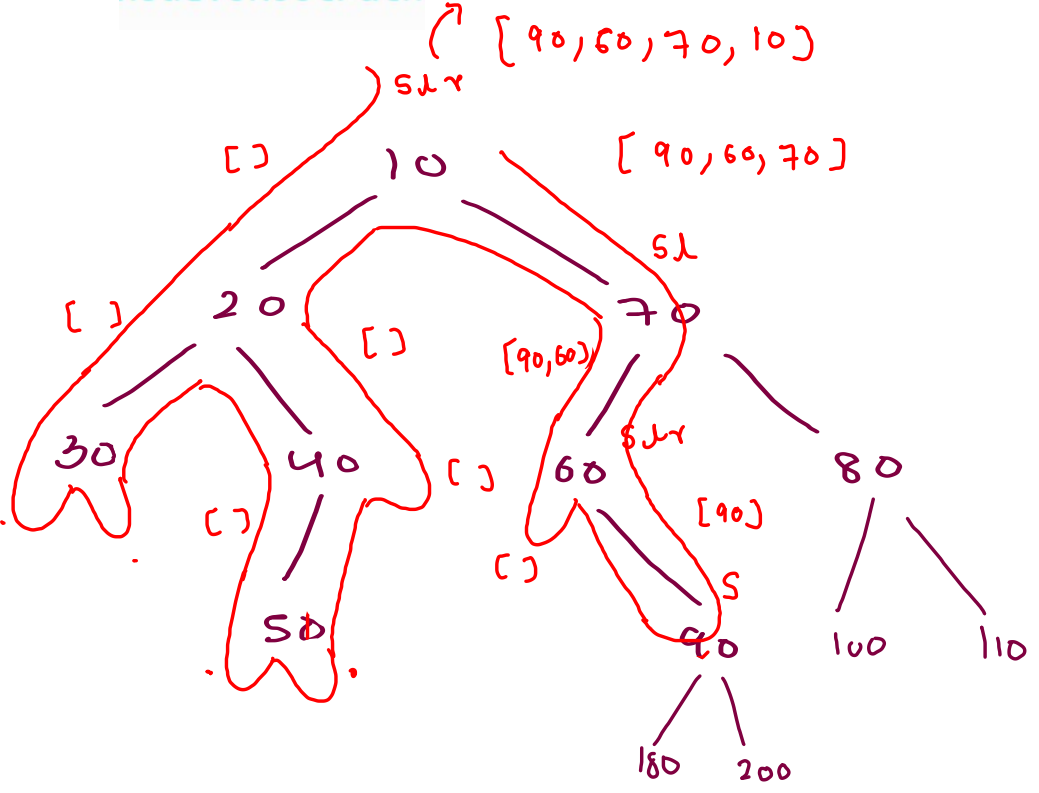
```java
public static boolean find(Node node, int data){
    if(node == null) {
        return false;
    }

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

    boolean fils = find(node.left,data); //found in left subtree

    if(fils == true) {
        return true;
    }

    boolean firs = find(node.right,data); //found in right subtree

    if(firs == true) {
        return true;
    }

    return false;
}
```



data = 25

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

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

    boolean fils = find(node.left,data); //found in left subtree

    if(fils == true) {
        return true;
    }

    boolean firs = find(node.right,data); //found in right subtree

    if(firs == true) {
        return true;
    }

    return false;
}
```

nodeToRootPath



data = 90

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

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

    ArrayList<Integer>n2lcp = nodeToRootPath(node.left,data);

    if(n2lcp.size() > 0) {
        n2lcp.add(node.data); //converting node to left child path --> node to root path
        return n2lcp;
    }

    ArrayList<Integer>n2rcp = nodeToRootPath(node.right,data);

    if(n2rcp.size() > 0) {
        n2rcp.add(node.data); //converting node to right child path --> node to root path
        return n2rcp;
    }

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