Binary Tree

- Size, Sum, Max & Height B.T.
- Creation of B.T.

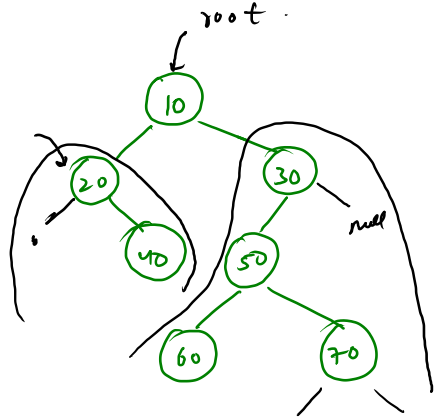
Traversals in B.T.

1. Pre-order Traversal
2. Inorder Traversal
3. Postorder Traversal
4. Level order Traversal

① Pre order Traversal

↳ Node left right

10 , 20 , 40, 30, 50, 60 , 70

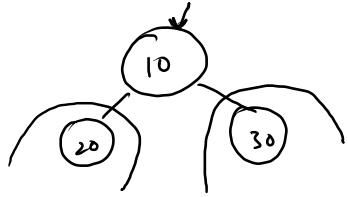



```
if ( node == null)
    return;
```

→ System.out.println (node.data);

→ preorder (node.left)

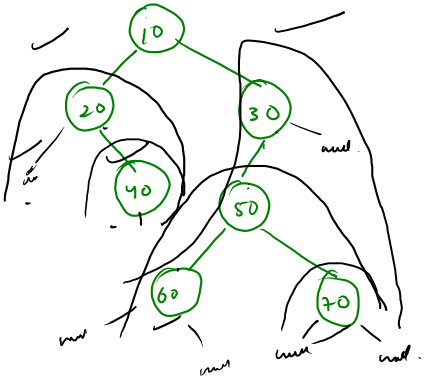
→ preorder (node.right)



10 20 30

② Post order Traversal

↳ left, right, node.

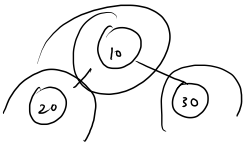


40, 20, 60, 70, 50, 30, 10

```
if (node == null)
    return;
```

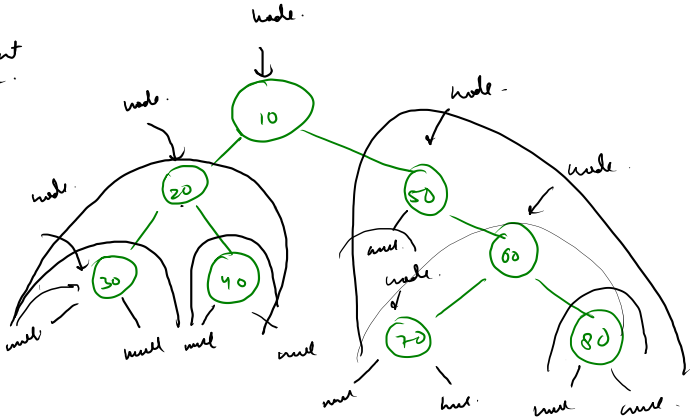
⇒

```
postorder (node.left);
postorder (node.right);
System.out.println(node.data);
```



20, 30, 10

left, right, current node.



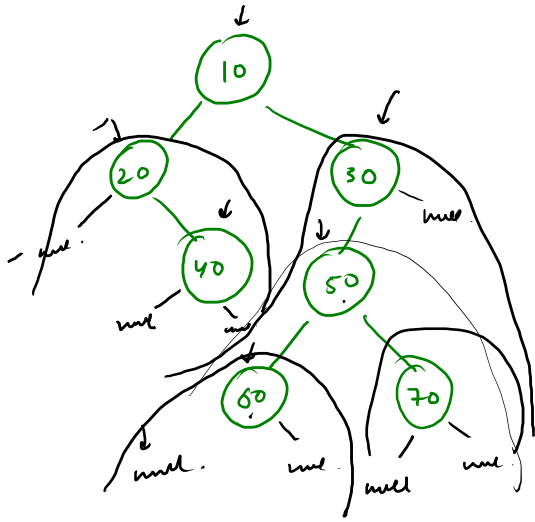
30, 40, 20, 70, 80, 60, 50, 10

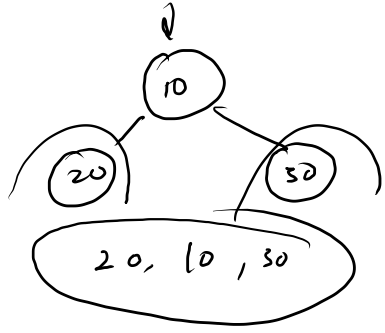
③ Inorder Traversal

↳ left, node, right.

20, 40, 10, 60, 50, 70, 30

```
if (node == null)
    return;
inorder (node.left);
System.out.println (node.data);
inorder (node.right);
```

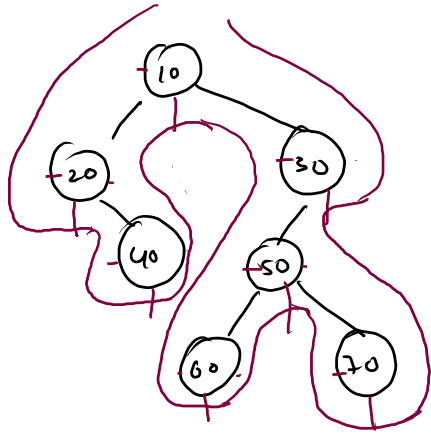




⇒ Preorder ⇒ node, left, right

⇒ Postorder ⇒ left, right, node.

⇒ Inorder ⇒ left, node, right.



⇒ Preorder: 10, 20, 40, 30, 50, 60, 70

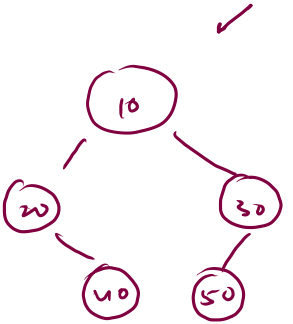
⇒ Postorder: 40, 20, 60, 70, 50, 30, 10

⇒ Inorder: 20, 40, 10, 60, 50, 70, 30

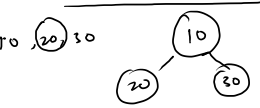
```java
public static void preOrder(Node node){
    if(node == null) {
        return;
    }

    System.out.print(node.data + " ");
    preOrder(node.left);
    preOrder(node.right);
}

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

    inOrder(node.left);
    System.out.print(node.data + " ");
    inOrder(node.right);
}

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

    postOrder(node.left);
    postOrder(node.right);
    System.out.print(node.data + " ");
}
```

→ Code + Diagram.

5 mins + 10 mins = 15 mins.

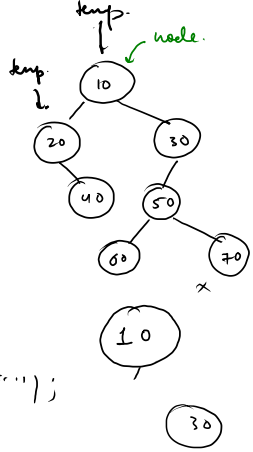


Preorder Traversal Iterative way

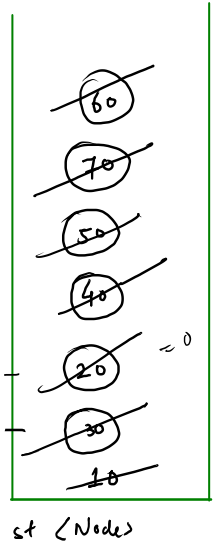


node = null.

```
Stack<Node> st = new stack<>();
st.push(node);
while( st.size()>0)
{
    Node temp = st.pop();
    System.out.print(temp.data + " ");
    if(temp.right != null)
        st.push(temp.right);
    if(temp.left != null)
        st.push(temp.left);
}
```



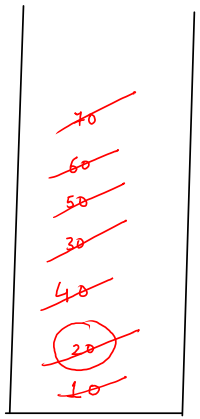
temp = 10 20 40 30 50 60 70

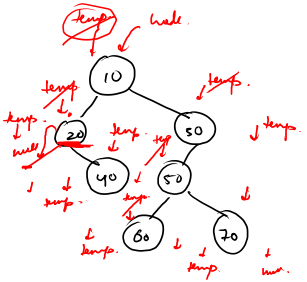


10 20 40 30 50 60 70

Inorder Traversal (Iterative way) — left, current, right

```
temp = node.
while (true) {
    if (temp != null) {
        st.push(temp);
        temp = temp.left;
    }
    else {
        if (st.size() == 0) break;
        temp = st.pop();
        print()
        temp = temp.right;
    }
}
```

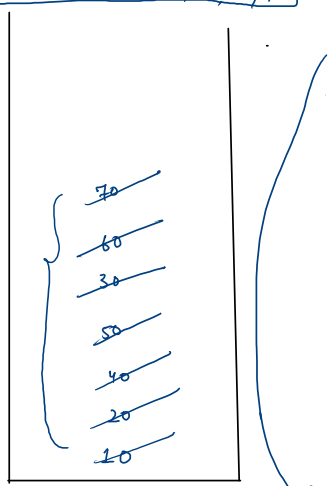


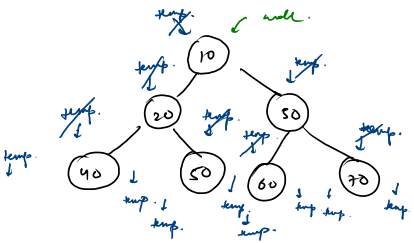


20, 40, 10, 60, 50, 70
30

40, 20, 50, 10, 60, 30, 70

```
Node temp = node.
while(true) {
    if (temp != null)
    {
        st.push(temp);
        temp = temp.left;
    }
    else {
        if(st.size() == 0)
            break;
        temp = st.pop();
        print()
        temp = temp.right;
    }
}
```

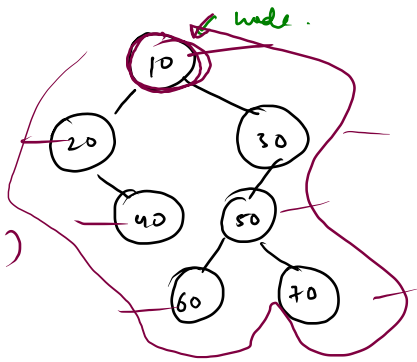


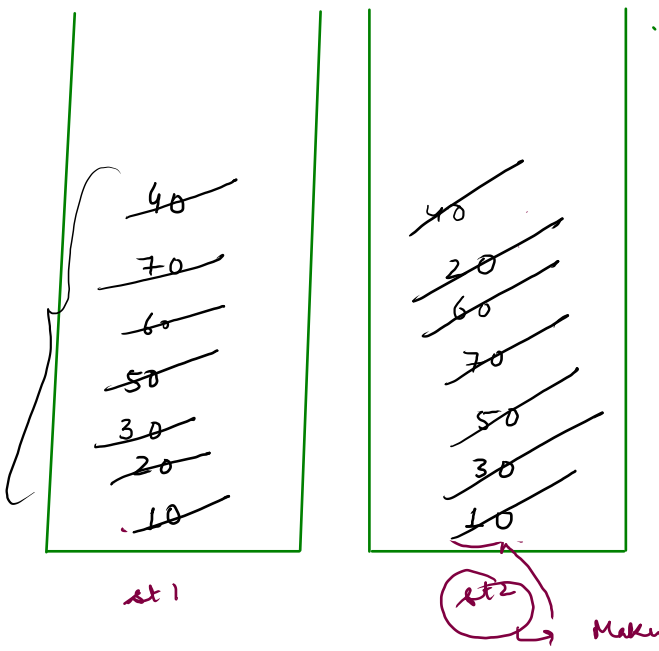


temp = 40 20 50 10 60 30 70

Postorder Traversal (Iterative way)

↳ left, right, node.





Making the answer.

```
st1.push(node)
while (st1.size()>0)
{
    node = st1.pop();
    st2.push(node);
    if(node.left != null)
        st1.push(node.left);

    if (node.right != null)
        st1.push(node.right);
}
while(st2.size()>0)
    syso(st2.pop() + " ");
```

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

    Stack<Node> st = new Stack<>();
    st.push(node);

    while(st.size() > 0){
        Node temp = st.pop();

        System.out.print(temp.data + " ");

        if(temp.right != null){
            st.push(temp.right);
        }

        if(temp.left != null){
            st.push(temp.left);
        }
    }
}
```

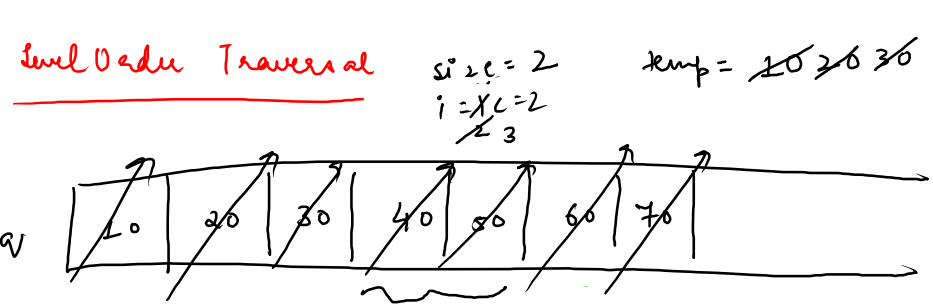
```java
public static void inOrder(Node node){
    Stack<Node> st = new Stack<>();
    Node temp = node;

    while(true){
        if(temp != null){
            st.push(temp);
            temp = temp.left;
        } else {
            if(st.size() == 0){
                break;
            }

            temp = st.pop();
            System.out.print(temp.data + " ");
            temp = temp.right;
        }
    }
}
```

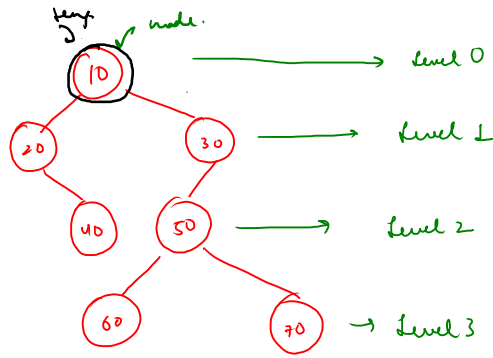
```java
public static void postOrder(Node node){
    if(node == null) {
        return;
    }

    Stack<Node> st1 = new Stack<>();
    Stack<Node> st2 = new Stack<>();

    st1.push(node);


    while(st1.size() > 0){
        node = st1.pop();
        st2.push(node);

        if(node.left != null){
            st1.push(node.left);
        }

        if(node.right != null){
            st1.push(node.right);
        }
    }


    while(st2.size() > 0){
        System.out.print(st2.pop().data + " ");
    }
}
```

Level Order Traversal

size = 2
i = 1 2 3

temp = ~~10~~ ~~20~~ ~~30~~ ~~40~~ ~~50~~ 60






```
while (q.size()>0)
{
    int size = q.size();          ⇒ (1) (2)
    for(int i=1; i<=size(); i++))   (1)
    {
        Node temp = q.remove();
        cout( temp.data + " ");   → printing current row.
        if (temp.left != null)    q.add(temp.left);
        if (temp.right != null)   q.add(temp.right);   → Elements of next row.
        cout();
    }
}
```

10
20 30
40 50
60 70

Level Order Traversal

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

    Queue<Node> q = new LinkedList<>();

    q.add(node);

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

        while(size>0){
            Node temp = q.remove();

            System.out.print(temp.data + " ");

            if(temp.left != null){
                q.add(temp.left);
            }

            if(temp.right != null){
                q.add(temp.right);
            }

            size--;
        }
        System.out.println();
    }

}
```