Class ,

Object → ,

↙ reference → Stack

instance → Heap

Constructor.

↳ parameterised

Trees Data Structure

→ Linear D.S.

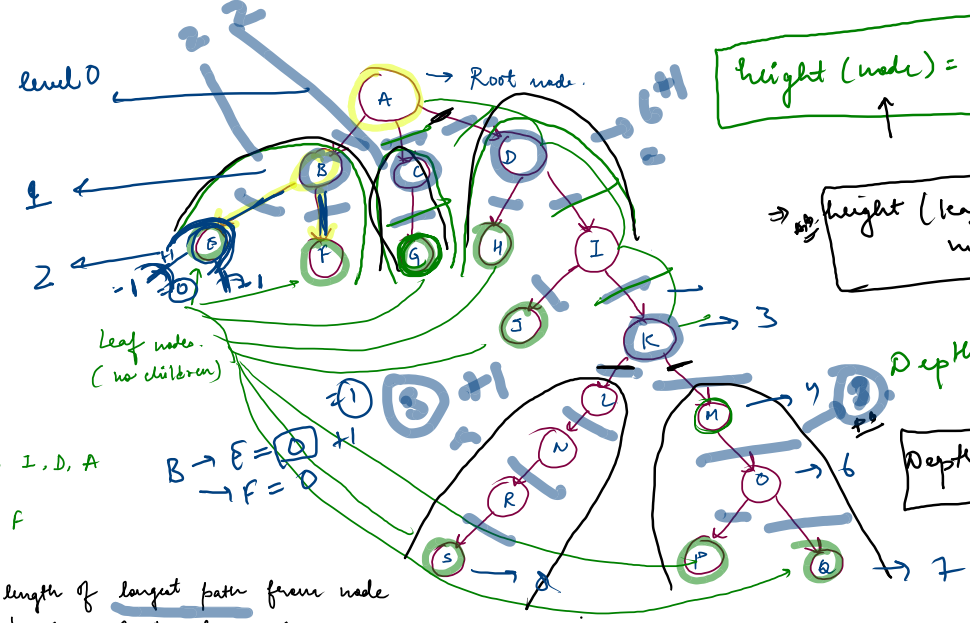
↳ Array, Stack, Queues, Arraylist

↳ Non-linear Data Structure

Tree Basics :

○ ⇒ node

→ ⇒ edge



Parent (G) ⇒ C

Parent (F) ⇒ B

Ancestor (M) ⇒ K, I, D, A

Children (B) ⇒ E, F
child node.

Height (Node) ⇒ length of longest path from node to any of its leaf node.

height (node) = 1 + max (height of its child nodes)

⇒ height (leaf node) = 0

Depth (K) = 3

Depth (Root node) = 0

B → E = 0 +1
→ F = 0

⇒ Depth :- length of path from root to given node.

Depth (G) = 2

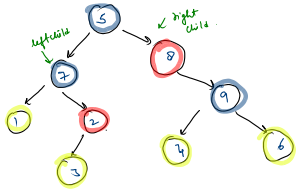
height (K) = 4

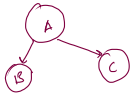
height (A) = ? 7 ✓

Binary Tree :-

Almost 2 child nodes.
↳ 0, 1, 2

2 nodes → 5, 7, 9
1 node → 8, 2
0 nodes → 1, 3, 4, 6

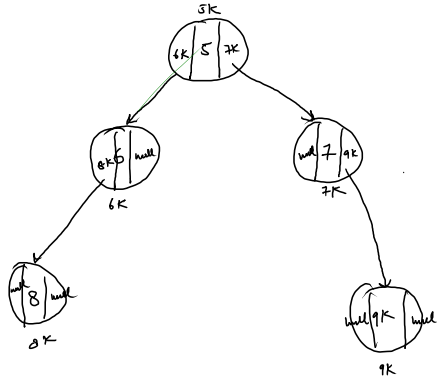




⇒ B is the left child of "A"

⇒ C is the right child of "A"


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

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

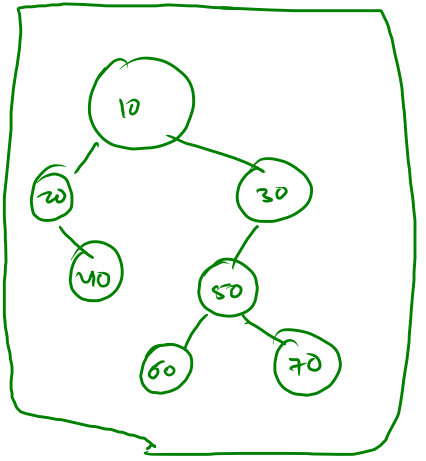


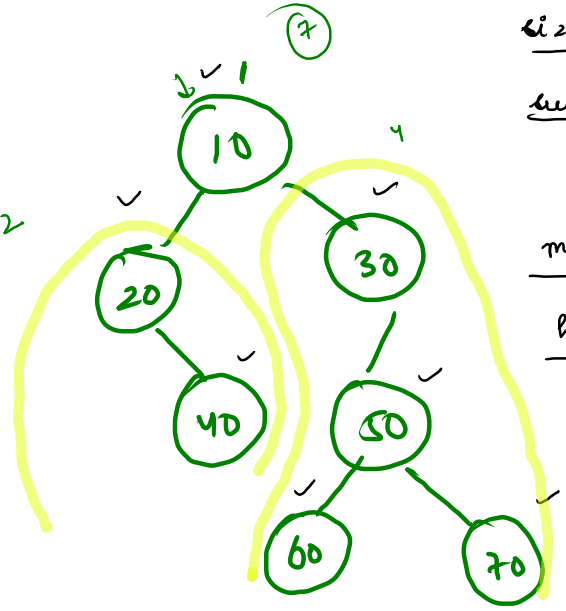
```java
public class Main {
    public static class Node {
        int data;
        Node left;
        Node right;

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

    public static void main(String [] args){
        Node root = new Node(10);
        Node rootLeft = new Node(20);
        Node rootRight = new Node(30);
        Node rootLeftRight = new Node(40);
        Node rootRightLeft = new Node(50);
        Node rootRightLeftLeft = new Node(60);
        Node rootRightLeftRight = new Node(70);

        // Linking Part

        root.left = rootLeft;
        root.right = rootRight;

        rootLeft.right = rootLeftRight;

        rootRight.left = rootRightLeft;

        rootRightLeft.left = rootRightLeftLeft;
        rootRightLeft.right = rootRightLeftRight;

        display(root);
    }
```

⇒ Creating node.

⇒ Linking node.

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

    System.out.println(root.data);
    display(root.left);
    display(root.right);
}
```





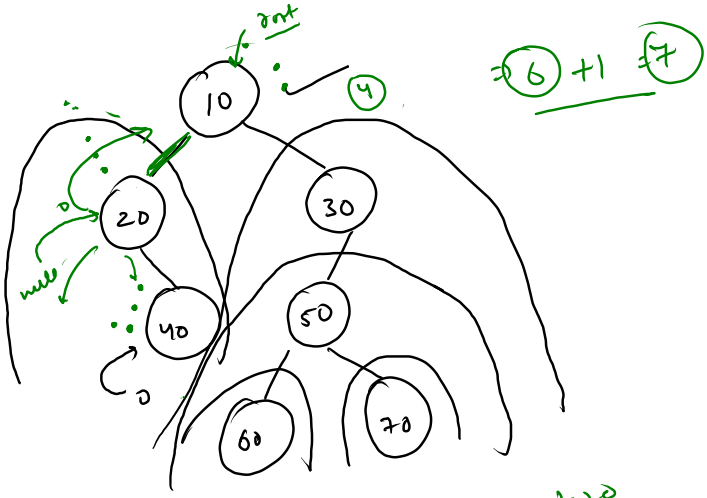
size ⇒ no of nodes present in the given B.T. ⇒ 7

sum ⇒ sum of data of all the nodes present in the given B.T.
⇒ 280

max ⇒ return the max value node ⇒ 70

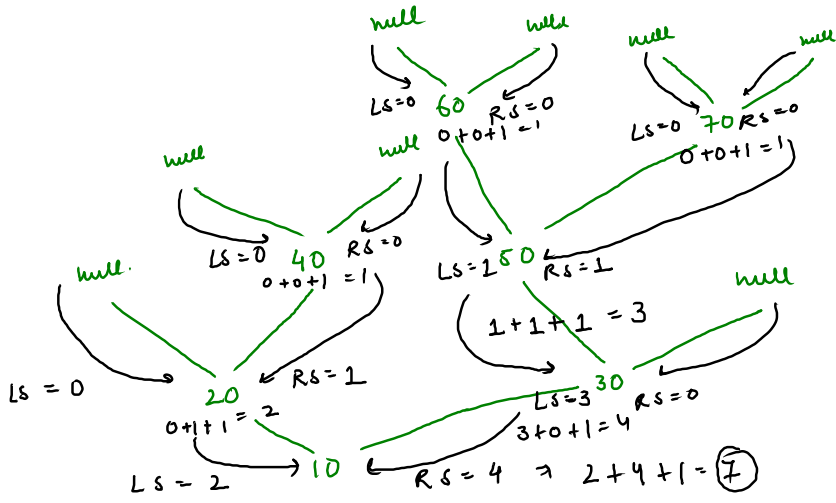
height ⇒ 1 + max (height of its child nodes)

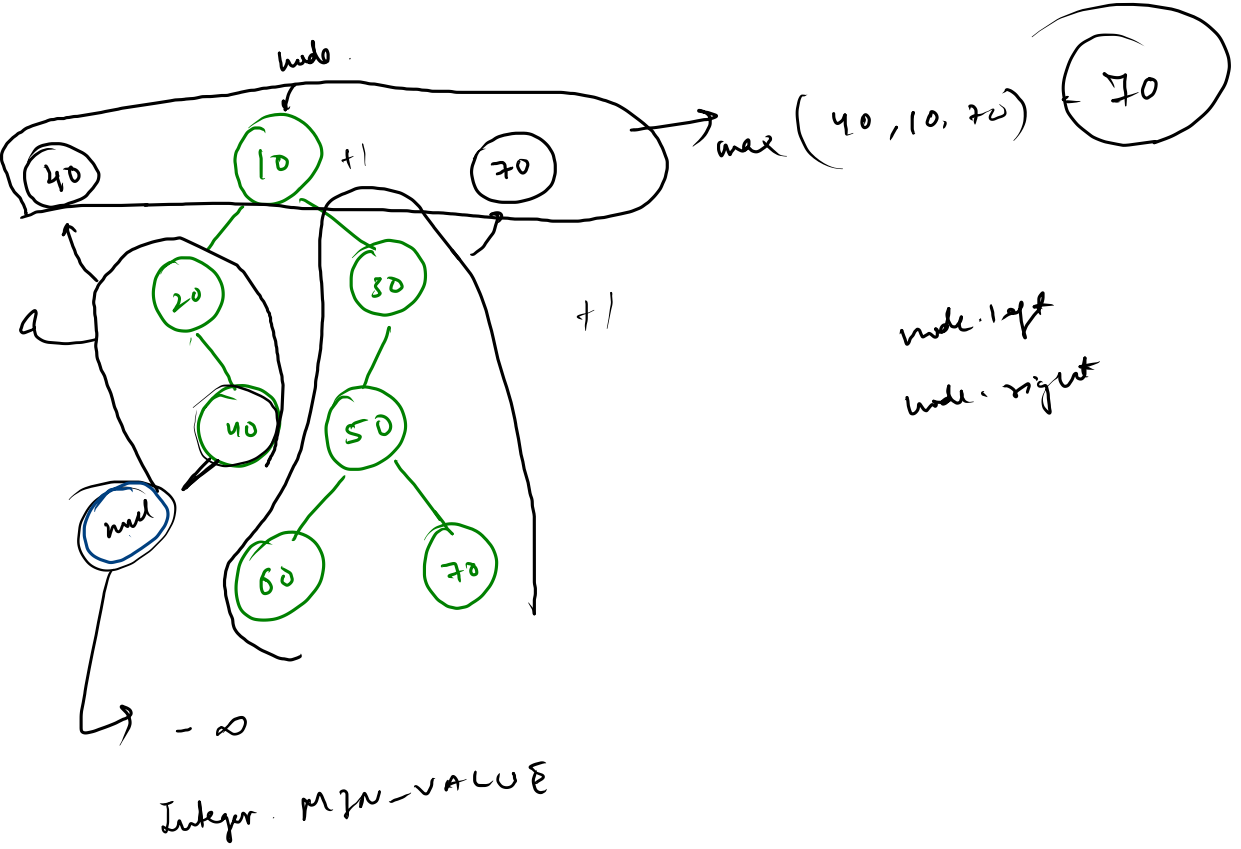
⇒ size ⇒ size(left child) + size(right child) + 1 (current node.)




```java
public static int size(Node node) {
  // write your code here
    if(root == null){
        return 0;
    }

    int leftSize = size(node.left);
    int rightSize = size(node.right);
    int totalSize = leftSize + rightSize + 1;
    return totalSize;
}
```

Ssum = ssum(left side) + ssum(right side) + node.data;




node
40
10
+1
70
20
30
40
50
null
60
70
-∞
max (40, 10, 70) = 70
+1
node.left
node.right
Integer MIN_VALUE

10 - 1

```java
public static int size(Node node) {
  // write your code here
    if(node == null){
        return 0;
    }

    int leftSize = size(node.left);
    int rightSize = size(node.right);
    int totalSize = leftSize + rightSize + 1;
    return totalSize;
}

public static int sum(Node node) {
  // write your code here
    if(node == null){
        return 0;
    }

    int leftSum = sum(node.left);
    int rightSum = sum(node.right);
    int totalSum = leftSum + rightSum + node.data;
    return totalSum;
}
```

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

    int leftMax = max(node.left);
    int rightMax = max(node.right);
    int overallMax = Math.max(node.data, Math.max(leftMax,rightMax));
    return overallMax;
}

public static int height(Node node) {
  // write your code here
    if(node == null){
        return -1;
    }

    int leftHeight = height(node.left);
    int rightHeight = height(node.right);
    int finalHeight = 1+Math.max(leftHeight, rightHeight);
    return finalHeight;
}
```

Complete the code + do the dry run