

8 July 2023

Handwritten notes and diagram illustrating the recursive calculation of the sum of nodes in a binary tree.

Handwritten Diagram:

```
graph TD
    node1((1)) --- node2((2))
    node1 --- node3((3))
    node2 --- node4((4))
    node2 --- node5((5))
```

The diagram shows a binary tree structure with nodes labeled 1, 2, 3, 4, and 5. Node 1 is the root, with left child 2 and right child 3. Node 2 has left child 4 and right child 5. The nodes are circled in pink, and the edges are labeled with 'node'.

Handwritten Calculation:

$$1 + \underbrace{\text{sum}(\text{left})}_{1 + \text{sum}(\text{left}) + \text{sum}(\text{right})} + \underbrace{\text{sum}(\text{right})}_{3 + \text{sum}(\text{null}) + \text{sum}(\text{null})}$$

The calculation shows the recursive steps for summing the nodes. The first line represents the root node (1) plus the sum of its left subtree (2, 4, 5) and right subtree (3). The second line shows the expansion of the left subtree sum, which is 1 (node 2) plus the sum of its left subtree (4) and right subtree (5). The third line shows the expansion of the right subtree sum, which is 3 (node 3) plus the sum of its left subtree (null) and right subtree (null).

Code Snippets:

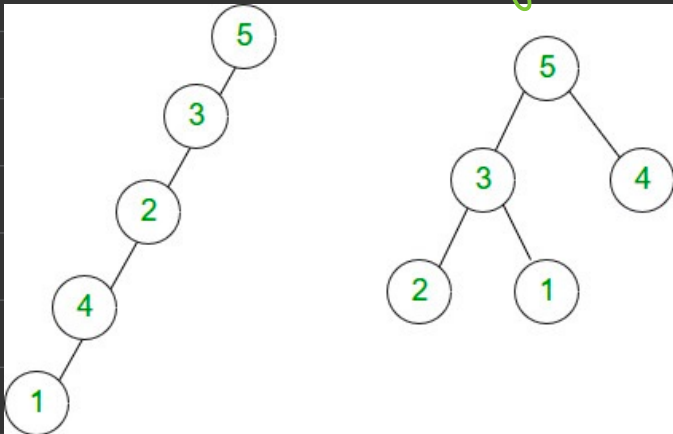
```
if (node != null) {
    postorderTraversal(node.left); // L
    postorderTraversal(node.right); // R
    System.out.print(node.data + " "); // V
}
```

```
int sumOfNodes(Node node)
{
    if (node == null)
        return 0;
    return node.data + sumOfNodes(node.left) + sumOfNodes(node.right);
}
```

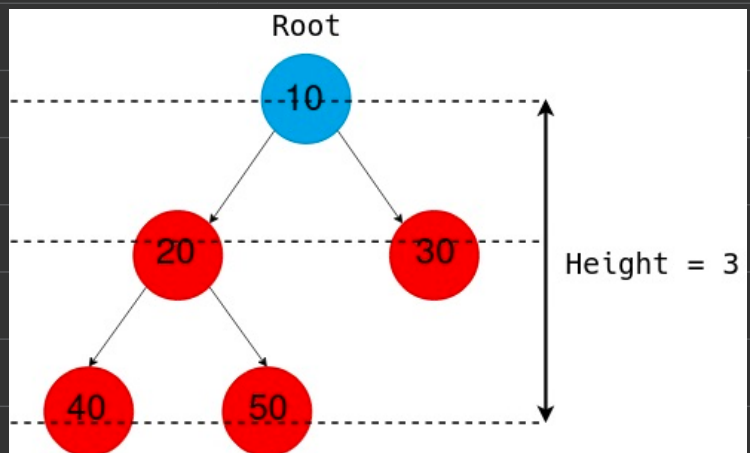
```
public static void main(String[] args) {
    BinaryTreeNode binaryTree = new BinaryTreeNode();
    binaryTree.inorderTraversal(binaryTree.root);
    System.out.println();
    binaryTree.preorderTraversal(binaryTree.root);
    System.out.println();
    ✓ binaryTree.postorderTraversal(binaryTree.root);
    System.out.println("\nSum of Nodes: " + binaryTree.sumOfNodes(binaryTree.root));
}
```

Height of Tree'

Height = 5



Height = 3



```
return 1 + Math.max(hOT(root.left), hOT(root.right))
```



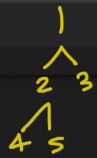
J Main.java 1 •

J Main.java > Main > main(String[])

```
59 {
60     if(node == null)
61         return 0;
62     return 1 + Math.max( heightOfTree(node.left) , heightOfTree(node.right));
63 }
64
65 int maxElement(Node node)
66 {
67     if(node == null)
68         return -1;
69     return Math.max( node.data, Math.max(maxElement(node.left), maxElement(node.right)));
70 }
71 }
72 }
```

Run | Debug

```
73 public static void main(String[] args) {
74     BinaryTreeNode binaryTree = new BinaryTreeNode();
75     binaryTree.inorderTraversal(binaryTree.root);
76     System.out.println();
77     binaryTree.preorderTraversal(binaryTree.root);
78     System.out.println();
79     binaryTree.postorderTraversal(binaryTree.root);
80     System.out.println("\nSum of Nodes: "+binaryTree.sumOfNodes(binaryTree.root));
81     System.out.println("\nHeight of Tree: "+binaryTree.heightOfTree(binaryTree.root));
82     System.out.println("\nMaximum Element of Tree: "+binaryTree.maxElement(binaryTree.root));
83 }
```



Handwritten recursive calculations for height and max element:

$$5 \leftarrow \max(1, \max[2, 3])$$

$$5 \leftarrow \max(2, \max[4, 5])$$

$$3 \leftarrow \max(3, \max(\text{null}, \text{null}))$$

$$4 \leftarrow \max(4, \max(\text{null}, \text{null}))$$

$$5 \leftarrow \max(5, \max(\text{null}, \text{null}))$$

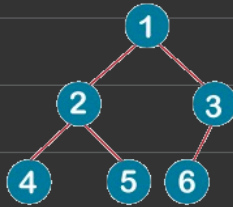
Full / Strict Binary Tree: It has 0/2 children

Types of Binary Trees

Strict



Complete



Degenerate



Perfect

