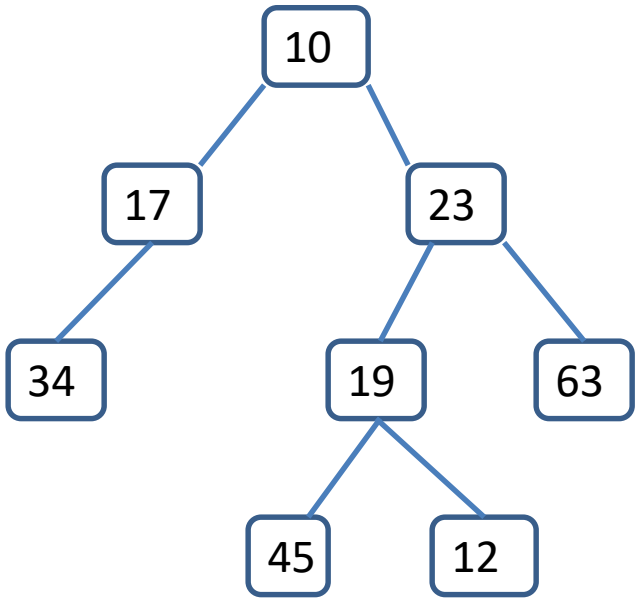
Tree Traversals

See Section 8.4 of the text.

We might create Binary Tree structures with the following Node class

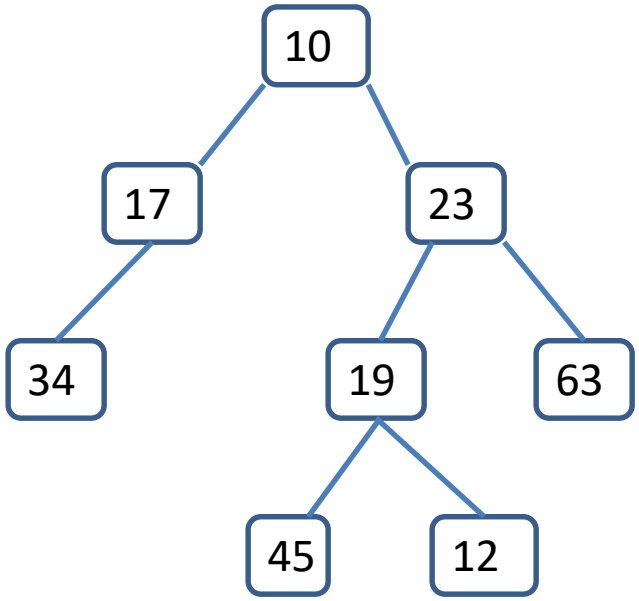
```
class Tree <T> {
        T data;
        Tree<T> left;
        Tree<T> right;
        ....
```

Here is a picture of a Binary Tree with Integer values at each node:



There are three standard ways to iterate through the nodes of a tree:

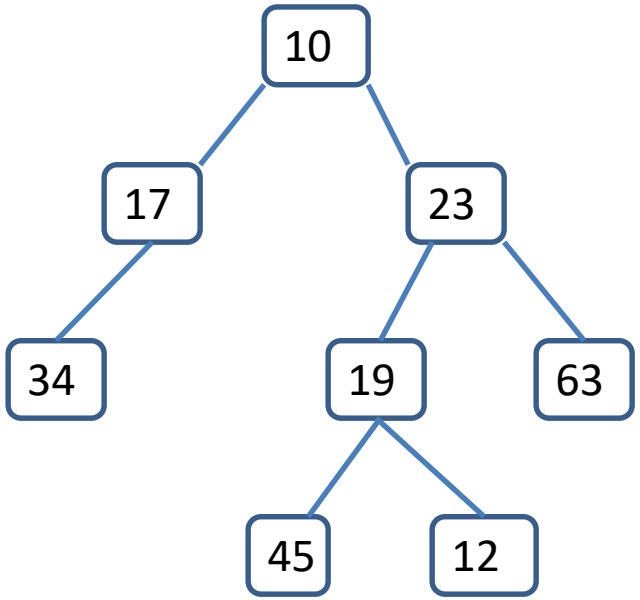
A *preorder traversal* of a tree lists the root, then its left subtree, then its right subtree.



For this tree the preorder traversal is

10 17 34 23 19 45 12 63

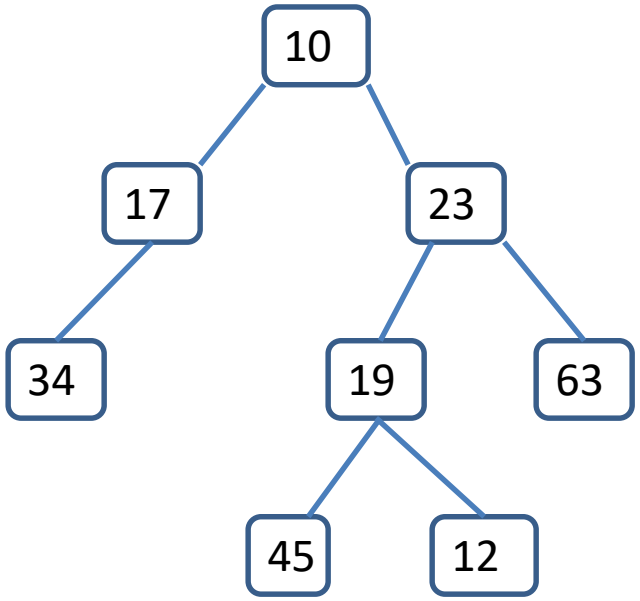
A *postorder traversal* of a tree lists the left subtree, the right subtree and then the root.



For this example the postorder traversal is

34 17 45 12 19 63 23 10

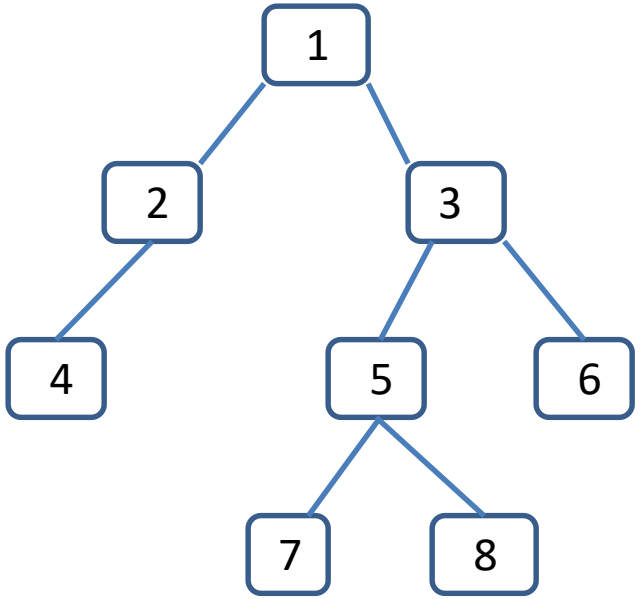
Finally, the *inorder traversal* lists the left subtree, the root, and then the right subtree.



For this example the inorder traversal is

34 17 10 45 19 12 23 63

Clicker Q: So what is the inorder (left child, node, right child) traversal of the following tree?



A. 1 2 3 4 5 6 7 8

B. 1 2 4 3 5 7 8 6

C. 2 4 1 3 7 5 8 6

D. 4 2 1 7 5 8 3 6

If all that you want to do is to print the data stored in a node in the order of one of these traversals, a simple recursion does the job. Here is the preorder traversal

```
public void PrintPreorder( TreeNode t) {
        if (t != null) {
                System.out.println(t.data);
                if (t.left != null)
                        PrintPreorder(t.left);
                if (t.right != null)
                        PrintPreorder(t.right);
        }
}
```

If you want to do more than just print the nodes, you can use a stack as a worklist. Consider first a preorder traversal. First push the root on the stack. Now repeatedly pop a node off the stack, push on its right child, then its left child. The nodes will pop off the stack in the correct sequence for a preorder traversal.

If you want an inorder traversal nodes appear twice -- first when we are about to push on their left child, then when we process the node itself and are ready to push the right child. So push the nodes with markers. First push the root with marker 1. When you pop a node with marker 1, push it back on with marker 2, then push its left child with marker 1. When you pop a node with marker 2, do whatever processing you need to do with the node, then push its right child with marker 1.

The postorder traversal takes 3 markers: we initially push nodes with marker 1. When we pop a node with marker 1, we push it back with marker 2 then push its left child with marker 1. When we pop a node with marker 2, we push it back with marker 3 and then push its right child with marker 1. Finally, when we pop a node with marker 3 we process it.