**private** **int** recSize(BSTNode<T> node)

// Returns the number of elements in subtree rooted at node.

{

**if** (node == **null**)

**return** 0;

**else**

**return** 1 + recSize(node.getLeft()) + recSize(node.getRight());

}

**public** **int** size()

// Returns the number of elements in this BST.

{

**return** recSize(root);

}

**public** **int** size2()

// Returns the number of elements in this BST.

{

**int** count = 0;

**if** (root != **null**)

{

LinkedStack<BSTNode<T>> nodeStack = **new** LinkedStack<BSTNode<T>>();

BSTNode<T> currNode;

nodeStack.push(root);

**while** (!nodeStack.isEmpty())

{

currNode = nodeStack.top();

nodeStack.pop();

count++;

**if** (currNode.getLeft() != **null**)

nodeStack.push(currNode.getLeft());

**if** (currNode.getRight() != **null**)

nodeStack.push(currNode.getRight());

}

}

**return** count;

}

//private wrapper that calls the recursive method for leaf count

**public** **int** leafCount2()

{

**return** recCount(root);

}

**private** **int** recCount(BSTNode<T> node)

{

**if** (node == **null**)

**return** 0;

**if** (node.getLeft()==**null** && node.getRight()==**null**)

**return** 1;

**return** recCount(node.getLeft()) + recCount(node.getRight());

}

//method for iterative leaf count

**public** **int** leafCount()

{

**int** count = 0;

**if** (root != **null**)

{

LinkedStack<BSTNode<T>> nodeStack = **new** LinkedStack<BSTNode<T>>();

BSTNode<T> currNode;

nodeStack.push(root);

**while**(!nodeStack.isEmpty()) {

currNode = nodeStack.top();

nodeStack.pop();

**if**(currNode.getLeft()==**null** && currNode.getRight()==**null**)

count++;

**if** (currNode.getLeft() != **null**)

nodeStack.push(currNode.getLeft());

**if** (currNode.getRight() != **null**)

nodeStack.push(currNode.getRight());

}

}

**return** count;

}