Practice Problems Solutions
Implement the method countNegI in the L1 class. The implementation must be iterative. Do not add any parameters to the method or instance variables to the class.

```java
public int countNegI() {
    //return the number of ints in the list that are less than 0
    Node temp = head;
    while (temp != null) {
        temp = temp.next;
    }
}
```
```java
public int countNegI() {
    // return the number of ints in the list that are less than 0

    Node temp = head;
    int count = 0;
    while (temp != null) {
        if (temp.data < 0) count++;
        temp = temp.next;
    }

    return count;
}
```
Implement the private method countNegR in the L1 class. The implementation must be recursive. Do not add any parameters to the method or instance variables to the class.

```java
public int countNegR() {
    //return the number of ints in the list that are less than 0
    return countNegR(head);
}
```
private int countNegR(Node h) {
    // return the number of ints in the list beginning at h that are less than 0
    if (h == null) return 0;
    if (h.data < 0) return countNegR(h.next) + 1;
    return countNegR(h.next);
}
Binary Tree Algorithms

data

Left Subtree
Right Subtree
public int countGreater(int d) {
    // if the tree is empty return 0
    // otherwise return the number of values in the tree greater than d
    return countGreater(root, d);
}

private int countGreater(Node r, int d) {
    // if r is null return 0
    // otherwise return the number of values in the subtree rooted at r
    // that are greater than d
    if (r == null) return 0;
    if (r.data > d) k = 1 else k = 0
    i = Find the number greater than d in the left tree
    j = Find the number greater that d in the right tree
    return i+j+k
}
public int countGreater(int d) {
    //if the tree is empty return 0
    //otherwise return the number of values in the tree greater than d
    return countGreater(root, d);
}

private int countGreater(Node r, int d) {
    //if r is null return 0
    //otherwise return the number of values in the subtree rooted at r
    //that are greater than d
    if (r == null) return 0;
    int k = r.data > d ? 1 : 0;
    return k + countGreater(r.left, d) + countGreater(r.right, d);
public int countLeaves() {
    //if the tree is empty return 0
    //otherwise return the number leaves in the tree
    return countLeaves(root);
}

private int countLeaves(Node r) {
    //if r is null return 0
    //otherwise return the number of leaves in the subtree rooted at r
    if r is null return 0
    if r is a leaf return 1
    i = Find the number of leaves in the left tree
    j = Find the number of leaves in the right tree
    return i+j;
}
public int countLeaves() {
    //if the tree is empty return 0
    //otherwise return the number leaves in the tree
    return countLeaves(root);
}

private int countLeaves(Node r) {
    //if r is null return 0
    //otherwise return the number of leaves in the subtree rooted at r
    if (r == null) return 0;
    if (r.left == null && r.right == null) return 1;
    return countLeaves(r.left) + countLeaves(r.right);
}
RemoveMin From a Binary Heap

0 9
1 22
2 25
3 60
4 31
5 40
6 45
7 65
8 70
9 47
10 52
11 43
12
Implement the private method toPostorder in the CompleteBinaryTree class. The implementation must be recursive.

```java
public class CompleteBinaryTree {
    // implements a complete binary tree stored in an array

    int[] values; // stores the tree
    // values are stored in positions 1 through currentSize;

    int currentSize; // the current number of items in the tree

    public CompleteBinaryTree(int s) {
        values = new int[s + 1];
        currentSize = 0;
    }
}
```
public String toPostorder() {
    //Returns a string of the ints in the tree in post order
    return toPostorder(1);
}

private String toPostorder(int r) {
    //Returns a string of the ints in the subtree rooted at r
    //in post order. A space should follow each int.
}
Postorder traversal

If the tree is not empty
  postOrder(left subtree)
  postOrder(right subtree)
  visit(Root)
public String toPostorder() {
    //Returns a string of the ints in the tree in post order
    //a space should follow each int
    return toPostorder(1);
}

private String toPostorder(int r) {
    //Returns a string of the ints in the subtree rooted at r
    //in post order. A space should follow each int.
    if (r > currentSize) return "";
    return toPostorder(2*r) + toPostorder(2*r+1) + values[r] + " ";
}