CS 340 Practice Problems

1. 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.

2. 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.
public class L1 {
    //implements a linked list of ints
    private class Node {
        private int data;
        private Node next;
        private Node(int d, Node n) {
            data = d;
            next = n;
        }
    }
    private Node head;

    public L1() {
        head = null; //no sentinel node
    }

    //assume insert has been implemented and zero or more values have
    //been inserted
    public int countNegI() {
        //return the number of ints in the list that are less than 0
        return countNegR(head);
    }

    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
        return countNegR(head);
    }
}
3. 2. Implement the method sum in the Q4 class.
public class Q4 {

    private class Node {
        private int num;
        private Node next;

        private Node(int i, Node n) {
            num = i;
            next = n;
        }
    }

    public int sum(Node x[]) {
        // Each x[i] is a linked list of Nodes
        // the lists do not contain a sentinel Node
        // return the sum of the ints in the lists of x
        // for example if x.length == 3 (do not assume this in
        // your code) and the list beginning with x[0] contains 2, 7
        // and the list beginning with x[1] contains 1, 5, 2, 1
        // and the list beginning with x[2] contains 11, 9, 3
        // then the function should return 41
    }
}

4. Implement the private method countGreater in the BinaryTree class.

5. Implement the private method countLeaves in the BinaryTree class.
import java.util.*;
import java.io.*;

public class BinaryTree {
    //Implements a binary tree of ints
    private class Node {
        private Node left;
        private int data;
        private Node right;
        private Node(Node L, int d, Node R) {
            left = L;
            data = d;
            right = R;
        }
    }
    private Node root; //In an empty tree the root is null

    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
    }

    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
    }
}
6. Suppose the array shown below represents a binary heap. Show the contents of the array after the minimum value (highest priority) is removed. The value in position 0 of the array is not shown (i.e. 9 is in position 1). Your answer should be an array of values.

9, 22, 25, 60, 31, 40, 45, 65, 70, 47, 52, 43

7. Implement the private method toPostorder in the CompleteBinaryTree class. The implementation must be recursive. Remember for a complete binary tree the tree is stored in an array. Your code must move through the array in the pattern of a binary tree. For example if the array contains the values 30, 20, 40, 10, 15, 7, 45, 90, 4 (position 0 is not used or shown) the value returned by the public method should be "90 4 10 15 20 7 45 40 30". Note the values in the array are a complete binary tree but they are not necessarily a binary heap. This does not matter in your answer.
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
        //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.
    }
}
8. Implement the private method leaves in the BinaryTree class. Your implementation must be recursive. See the example on the accompany page.
For the tree shown below the function leaves should return the string
MNO IJ UVW PQ
public class BinaryTree {
    //Implements a binary tree of strings
    private class Node {
        private Node left;
        private String data;
        private Node right;
        private Node(Node L, String d, Node R) {
            left = L;
            data = d;
            right = R;
        }
    }
    private Node root; //In a empty tree root is null

    //ASSUME VALUES HAVE BEEN PUT IN THE TREE

    public String leaves() {
        //if the tree is empty return the empty string otherwise
        //return a string of the data in the leaves in the tree
        return leaves(root);
    }

    private String leaves(Node r) {
        //if the r is null return the empty string
        //otherwise return a string of the data in the leaves of
        //the subtree rooted at r
    }
}
9. Implement the outDegree function in the DirectedGraph class.
public class DirectedGraph {

    private class VertexNode {
        private String name;
        private VertexNode nextV;
        private EdgeNode edges;

        private VertexNode(String n, VertexNode v) {
            name = n;
            nextV = v;
            edges = null;
        }
    }

    private class EdgeNode {
        private VertexNode vertex1;
        private VertexNode vertex2;
        private EdgeNode nextE;

        private EdgeNode(VertexNode v1, VertexNode v2, EdgeNode e) {
            vertex1 = v1;
            vertex2 = v2;
            nextE = e;
        }
    }

    private VertexNode vertices; //head of the linked list of vertices
    private int numVertices;

    public DirectedGraph() {
        vertices = null;  //no sentinel value
    }

    private int outDegree(VertexNode v) {
        //return the out degree of vertex v
        return 0;
    }

}
10. Implement the method `largestWeight` in the `Graph` class. The class implements a weighted undirected graph. Unlike in class the vertices are stored in an array. A weight is associated with each edge.
public class Graph {
    private class Vertex {
        private EdgeNode edges1;
        private EdgeNode edges2;

        private Vertex() {
            edges1 = null;
            edges2 = null;
        }
    }

    private class EdgeNode {
        private int vertex1;
        private int vertex2;
        private EdgeNode next1;
        private EdgeNode next2;
        private int weight;

        private EdgeNode(int v1, int v2, EdgeNode e1, EdgeNode e2, int w) {
            //PRE: v1 < v2
            vertex1 = v1;
            vertex2 = v2;
            next1 = e1;
            next2 = e2;
            weight = w;
        }
    }

    private Vertex[] g; //array of vertices

    public Graph(int size) {
        g = new Vertex[size];
        for (int i = 0; i < size; i++) {
            g[i] = new Vertex();
        }
    }

    public int largestWeight(int i) {
        //PRE: i is legal vertex
        //return the largest weight on an edge incident
        //(connected) to vertex i
    }
}