Recursive Programs

For the following methods, provide a recursive definition and then the corresponding recursive code. You may not assume that there exist protected leftTree() and rightTree() methods in the BinaryTreeInterface, but you may assume that the following methods are in fact defined in the BinaryTreeInterface (So you don’t have to worry about casting).

1) Find the number of nodes in a Binary Tree

The number of nodes in a Binary Tree is:
- 0 if the tree is Empty
- Otherwise, it is 1 + number of nodes in the Left Subtree + number of nodes in the Right Subtree

```
public int numNodes(BinaryTreeInterface tree)
//pre: tree!= null
//post: returns the number of nodes in tree
{  if (tree.isEmpty())
   { return 0;
   }
   BinaryTreeInterface left  = tree.detachLeftSubtree();
   BinaryTreeInterface right  = tree.detachRightSubtree();
   int answer = 1 + numNodes(left) + numNodes(right);
   this.attachLeftSubtree(left);
   this.attachRightSubtree(right);
   return answer;
}
```

2) Find out if a Binary Tree contains a particular Object (called key)

A tree contains an Object 'key' if:
- This is not empty AND the root item equals the key
- or this is not empty AND key is contained within the Left or Right Subtrees

```
public boolean contains(Object key)
//pre: key != null
//post: returns true iff this contains an item that 'equals' key
{   if (this.isEmpty())
    {    return false;
    }
   BinaryTreeInterface left  = tree.detachLeftSubtree();
   BinaryTreeInterface right  = tree.detachRightSubtree();
   boolean answer = this.getRootItem().equals(key) || left.contains(key) ||
right.contains(key);
   this.attachLeftSubtree(left);
   this.attachRightSubtree(right);
   return answer;
}
```

3) Find the rightmost TreeNode in a Binary Tree.

The rightmost TreeNode in a Binary Tree is:
- The root TreeNode if the rightsubtree is empty
- The rightmost TreeNode of the right subtree of this

```
public Object rightMost()
// pre: this is not empty
//post: returns the rightmost item in this
{  BinaryTreeInterface right = tree.detachRightSubtree();
    if (right.isEmpty())
    {  this.attachRightSubtree(right);
        return this.getRootItem();
    }
    else
    { Object ans = right.rightMost();
       this.attachRightSubtree(right);
       return ans;
    }
}
```

4) Find the number of leaves in a BinaryTree.

The number of leaves in a Binary Tree is:
- 0 if the tree is empty
- 1 if the tree is non-empty but has no Left or Right Subtrees
- Otherwise, it is the number of leaves in the Left Subtree + number of leaves in the Right Subtree.

```
public int numLeaves()
//post: returns the # of leaves in this.
{   if (this.isEmpty()
    {   return 0;
    }
   BinaryTreeInterface left  = tree.detachLeftSubtree();
   BinaryTreeInterface right  = tree.detachRightSubtree();
   int answer = 0;
   if (left.isEmpty() && right.isEmpty())
   {    answer = 1;
   }
   else
   {   answer = left.numLeaves() + right.numLeaves();
   }
   this.attachLeftSubtree(left);
   this.attachRightSubtree(right);
   return answer;
}
```

For your own practice:

5) Reverse a Queue
6) Print a Linked List Forward (Where each node's Item is a single char)
7) Print a Linked List Backward (Where each node's Item is a single char)
8) Find out if a given string is a Pallindrome (where spaces are counted too)
9) Copy a Stack (in the same order, without using a helper stack)
10) Copy all the objects of a Tree into another Tree