PROBLEM 1:  (Family Trees)

In this problem assume all values in trees are unique, no value appears more than once in a tree. **In this problem the tree is not necessarily a search tree.**

The code in the function leastAncestor shown below returns a pointer to the least ancestor of two strings in a tree.

The least ancestor of two string values p and q is the node furthest from the root (deepest) which is an ancestor of both p and q (there is a path from the least ancestor to both p and q).

For example, the tree diagrammed below on the right yields the values shown in the table on the left.

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>least ancestor</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>E</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>K</td>
<td>H</td>
<td>C</td>
</tr>
<tr>
<td>H</td>
<td>J</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>G</td>
<td>A</td>
</tr>
<tr>
<td>G</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

Part A (8 points)

The complexity of leastAncestor shown on the next page is not O(n) for an n-node tree. What is the complexity and why? Justify using recurrence relations and an explanation of each part of the recurrence. Provide big-Oh complexities in the average case (assume trees are roughly balanced) and the worst case.
Here's code to find the least ancestor, the helper method \texttt{inTree} is called from method \texttt{leastAncestor}. Note (again) that in this problem trees are not search trees.

```java
public static boolean inTree(TreeNode root, String s){
    if (root == null) return false;
    if (root.info.equals(s)) return true;
    return inTree(root.left,s) || inTree(root.right,s);
}

public static TreeNode leastAncestor(TreeNode t, String p, String q){
    if (t == null) return null;

    // first check subtrees (lower than me) for ancestor

    TreeNode result = leastAncestor(t.left, p, q);
    if (result != null) return result;

    result = leastAncestor(t.right, p, q);
    if (result != null) return result;

    // didn't find in subtrees, am I the least ancestor? check
    // me and left/right subtrees for p/q (vice versa)

    if ( (t.info.equals(p) || inTree(t.left,p)) &&
         (t.info.equals(q) || inTree(t.right,q)) ) {
        return t;
    }

    if ( (t.info.equals(q) || inTree(t.left,q)) &&
         (t.info.equals(p) || inTree(t.right,p)) ) {
        return t;
    }

    return null;
}
```


Part B (6 points)
Write the function findPath whose header is given below. The function sets values in/returns an ArrayList representing the path from the root of t to the node containing target if there is a path, or returns an empty vector otherwise.

For example, given the tree on the previous page we have:

<table>
<thead>
<tr>
<th>call</th>
<th>ArrayList list</th>
</tr>
</thead>
<tbody>
<tr>
<td>findPath(t,&quot;F&quot;, list)</td>
<td>(A,B,C,F)</td>
</tr>
<tr>
<td>findPath(t,&quot;A&quot;, list)</td>
<td>(A)</td>
</tr>
<tr>
<td>findPath(t,&quot;J&quot;, list)</td>
<td>(A,D,J)</td>
</tr>
<tr>
<td>findPath(t,&quot;G&quot;, list)</td>
<td>(A,B,E,G)</td>
</tr>
<tr>
<td>findPath(t,&quot;B&quot;, list)</td>
<td>(A,B)</td>
</tr>
<tr>
<td>findPath(t,&quot;X&quot;, list)</td>
<td>()</td>
</tr>
</tbody>
</table>

/**
 * Add values in list so that they represent strings
 * on path from t to node containing target. If target
 * not in the tree then no values added to list
 */

public static void findPath(TreeNode t, String target, ArrayList<String> list) {
    if (t == null) return;
    if (t.info.equals(target)) {
        list.add(t.info);
        return;
    }
    // add code here
}
Part C (8 points)

Write a version of leastAncestor that runs in $O(n)$ time for an n-node tree. You can use any approach; one approach is to use the function findPath from part B, another is to write an auxiliary function that returns three values (e.g., in an array or list) an ancestor-pointer and a boolean that tells if p is in the tree and a boolean that tells if q is in the tree.

Write the code and justify that it runs in $O(n)$ time.