Exam prep – *ON PAPER*

Don't forget that we have an exam on Friday! You should be expecting some questions requiring you to write methods for linked lists and/or trees. Some of the questions might specifically say "you *must* use recursion" and some questions might specifically say "you may *not* use recursion". If you don't follow these instructions you are likely to get no points, so read the instructions carefully!

Please practice solving the problems below **on paper** (no computer), just as you would during the exam.

1. Assume we are writing a traditional linked list (terminating with a null-reference), partially defined below:

```
public class LinkedList<T> {
     private class Node {
          private T data;
          private Node next;

          private Node(T data) {
               this.data = data;
               next = null;
          }
     }
     private Node head;
}
```

a. Write a method called `insertValueAfter`, with the prototype you see below. The `value` must be inserted immediately after the first occurrence of the `target`. If the `target` is not in the list, then your method should do nothing. **You must use recursion for this question. You may use an auxiliary helper method if you find it useful.**

```
public void insertValueAfter(T value, T target)
```

b. Write a method called `insertValueBefore`, with the prototype you see below. The `value` must be inserted immediately before the first occurrence of the `target`. If the `target` is not in the list, then your method should do nothing. **You may not use recursion for this question.**

```
public void insertValueBefore(T value, T target)
```

[More questions on the back page….]

2. Assume we are writing a traditional binary search tree (with null references to represent empty child branches), partially defined below. **You should assume that this tree contains no duplicate keys.**

```
public class BinarySearchTree<K extends Comparable<K>, V> {
     private class Node {
          private K key;
          private Node left, right;

          private Node(K key) {
               this.key = key;
               left = right = null;
          }
     }
     private Node root;

}
```

a. Write a method called `contains`, which accepts a parameter of type K, and returns true if the parameter is found in the tree, false otherwise. Your code must be efficient (do not traverse any subtrees unnecessarily) or you will lose most of the points. **You must use recursion for this question. You may use an auxiliary helper method if you find it useful.**

b. Write a method called `countRange` with the prototype shown below. You may assume that the parameter `low` is "smaller than" the parameter `high` when they are compared. The method should return the number of elements found in the tree with values that are between `low` and `high` (inclusive). Your code must be efficient (do not traverse any subtrees unnecessarily) or you will lost most of the points. **You must use recursion for this question. You may use an auxiliary helper method if you find it useful.**

```
public int countRange(K low, K high)
```