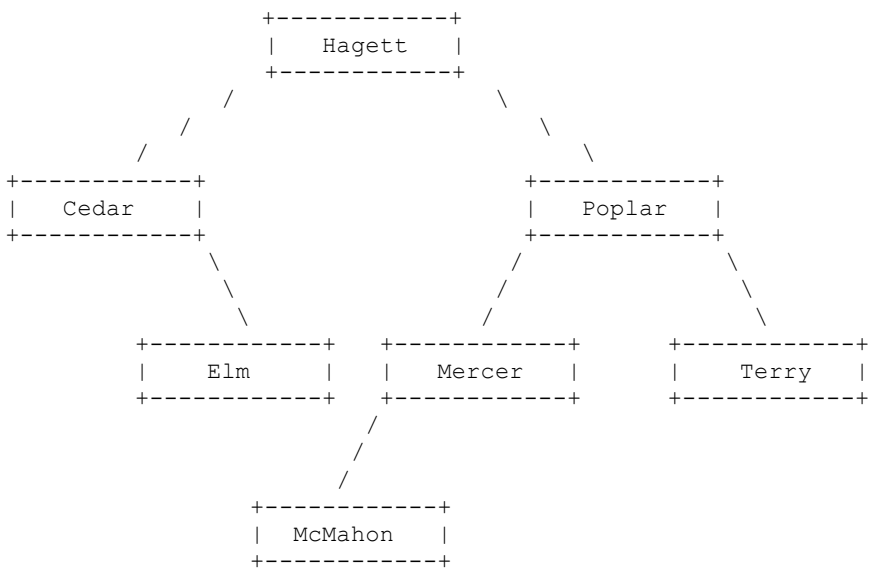
CSE 143 Final Key
Summer 2013

1.
```
Preorder traversal  3, 7, 9, 5, 2, 4, 0, 1, 8, 6
Inorder traversal   9, 7, 2, 5, 4, 3, 1, 8, 0, 6
Postorder traversal 9, 2, 4, 5, 7, 8, 1, 6, 0, 3
```

2.



3.

Statement	Output
var2.method3();	Ace3/Ace2
var3.method3();	Queen3
var5.method3();	error
var6.method3();	Ace3/King2
var1.method2();	error
var2.method2();	Ace2
var3.method2();	Ace2
var4.method2();	King2
((Queen)var6).method1();	error
((Ace)var1).method1();	error
((Ace)var1).method3();	Ace3/Jack2/Ace2
((Jack)var1).method2();	Jack2/Ace2
((Jack)var6).method1();	Jack1
((King)var1).method1();	error
((King)var4).method3();	Ace3/King2

4. One possible solution appears below.

```
public class Beverage implements Comparable<Beverage> {
    private double sugarContent;
    private String name;
    private int numDrinkers;
    private int totalAge;

    public Beverage(String name, double sugarContent) {
        this.name = name;
        this.sugarContent = sugarContent;
    }

    public String getName() {
        return name;
    }
```

```
    public double getSugarContent() {
        return sugarContent;
    }

    public void drink(int age) {
        totalAge += age;
        numDrinkers++;
    }

    public int compareTo(Beverage other) {
        if (sugarContent < other.sugarContent) {
            return -1;
        } else if (sugarContent > other.sugarContent) {
            return 1;
        } else {
            return name.compareTo(other.name);
        }
    }

    public String toString() {
        String result = name + "(" + sugarContent + "g sugar) - ";
        if (numDrinkers == 0) {
            result += "no drinkers";
        } else {
            result += "average drinker is " +
                      ((double)totalAge / numDrinkers);
        }
        return result;
    }
}
```

5. One possible solution appears below.

```
    public int countOddBranches() {
        return countOddBranches(overallRoot);
    }

    private int countOddBranches(IntTreeNode root) {
        if (root == null || (root.left == null && root.right == null)) {
            return 0;
        } else {
            return root.data % 2 + countOddBranches(root.left) +
                         countOddBranches(root.right);
        }
    }
```

6. One possible solution appears below.

```
public static Map<Point, Set<Integer>> rateCoffee(List<Point> list) {
    Map<Point, Set<Integer>> coffee = new HashMap<Point, Set<Integer>>();
    int index = 0;
    for (Point p : list) {
        if (!coffee.containsKey(p)) {
            coffee.put(p, new TreeSet<Integer>());
        }
        coffee.get(p).add(index);
        index++;
    }
    return coffee;
}
```

7. One possible solution appears below.

```
public void stretch() {
    overallRoot = stretch(overallRoot);
}

private IntTreeNode stretch(IntTreeNode root) {
    if (root != null) {
        root.left = stretch(root.left);
        root.right = stretch(root.right);
        if (root.left != null && root.right == null)
            root = new IntTreeNode(-1, root, null);
        else if (root.left == null && root.right != null)
            root = new IntTreeNode(-1, null, root);
    }
    return root;
}
```

8. Two possible solutions appear below. The second is shorter because it uses recursion in an x=change(x) manner.

```
public void rotate3() {
    if (front != null && front.next != null && front.next.next != null) {
        ListNode temp = front;
        front = front.next;
        temp.next = front.next.next;
        front.next.next = temp;
        while (temp.next != null && temp.next.next != null &&
               temp.next.next.next != null) {
            ListNode temp2 = temp.next;
            temp.next = temp.next.next;
            temp2.next = temp.next.next.next;
            temp.next.next.next = temp2;
            temp = temp2;
        }
    }
}
```

```
public void rotate3() {
    front = rotate3(front);
}

public ListNode rotate3(ListNode lst) {
    if (lst != null && lst.next != null && lst.next.next != null) {
        ListNode temp2 = lst;
        lst = lst.next;
        temp2.next = lst.next.next;
        lst.next.next = temp2;
        temp2.next = rotate3(temp2.next);
    }
    return lst;
}
```