1. Binary Tree Traversals, 6 points. Consider the following tree.

```
                             +---+
                             | 8 |
                             +---+
                           /       \
                         /           \
                   +---+               +---+
                   | 7 |               | 5 |
                   +---+               +---+
                  /                   /     \
                 /                   /       \
              +---+               +---+     +---+
              | 2 |               | 1 |     | 0 |
              +---+               +---+     +---+
             /     \             /     \
            /       \           /       \
         +---+     +---+     +---+     +---+
         | 4 |     | 9 |     | 3 |     | 6 |
         +---+     +---+     +---+     +---+
```

Fill in each of the traversals below:

Preorder traversal: ____________________________________

Inorder traversal: ____________________________________

Postorder traversal: ____________________________________

2. Binaray Search Tree, 4 points. Draw a picture below of the binary search tree that would result
   from inserting the following words into an empty binary search tree in the following order:

   Hagett, Cedar, Poplar, Elm, Mercer, Terry, McMahon

   Assume the search tree uses alphabetical ordering to compare words.

3. Details of inhehritance, 10 points. [Pepsi/Soda/DietCoke/Coke]

Assuming that the following classes have been defined:

```
public class Pepsi extends Soda {
    public void m2() {
        System.out.println("Pepsi2");
        m1();
    }
}

public class Soda {
    public void m1() {
        System.out.println("Soda1");
    }
}

public class DietCoke extends Coke {
    public void m1() {
        System.out.println("DietCoke1");
    }
}

public class Coke extends Soda {
    public void m2() {
        System.out.println("Coke2");
        super.m1();
    }

    public void m3() {
        System.out.println("Coke3");
        m1();
    }
}
```

Now assume that the following variables have been declared and initialized.

```
Soda v1 = new Coke();
Soda v2 = new Pepsi();
Soda v3 = new DietCoke();
Object v4 = new Soda();
Coke v5 = new DietCoke();
Coke v6 = new Coke();
```

```
Statement                           Output
----------------------------------------------------------------
v1.m1();
----------------------------------------------------------------
v2.m1();
----------------------------------------------------------------
v3.m1();
----------------------------------------------------------------
v4.m1();
----------------------------------------------------------------
v5.m1();
----------------------------------------------------------------
v6.m1();
----------------------------------------------------------------
v1.m2();
----------------------------------------------------------------
v2.m2();
----------------------------------------------------------------
v3.m2();
----------------------------------------------------------------
v4.m2();
----------------------------------------------------------------
v5.m3();
----------------------------------------------------------------
v6.m3();
----------------------------------------------------------------
((Object)v2).m1();
----------------------------------------------------------------
((Soda)v5).m1();
----------------------------------------------------------------
((Pepsi)v4).m2();
----------------------------------------------------------------
((Soda)v4).m2();
----------------------------------------------------------------
((Soda)v4).m1();
----------------------------------------------------------------
((DietCoke)v5).m3();
----------------------------------------------------------------
((Soda)v6).m1();
```

```
4. Binary Trees, 10 points. [sameStructure]

Write a method sameStructure that compares two binary trees of integers to
see if they have the same structure.
This method will be added to the IntTree class from lecture and section (see
cheat sheet).
Your method should accept a reference to another IntTree as a parameter and
return true if each node in one tree has
a corresponding node in the same location relative to the root in the other
tree. You may assume that the tree passed in is not null.

For example, suppose IntTree variables tree1, tree2 and tree3 refer to the
following trees:

            tree1

            +---+
            | 1 |
            +---+
           /     \
          /       \
     +---+         +---+
     | 9 |         | 6 |
     +---+         +---+
    /     \
   /       \
+---+     +---+
| 4 |     | 3 |
+---+     +---+


            tree2

            +---+
            | 7 |
            +---+
           /     \
          /       \
     +---+         +---+
     | 3 |         | 5 |
     +---+         +---+
    /     \
   /       \
+---+     +---+
| 6 |     | 0 |
+---+     +---+
            tree3

            +---+
            | 7 |
            +---+
           /     \
          /       \
     +---+         +---+
     | 3 |         | 5 |
     +---+         +---+
    /
   /
+---+
| 6 |
+---+
```

The call on tree1.sameStructure(tree2) should return true because the two
trees have matching nodes in all the same relative positions.
However, the call on tree2.sameStructure(tree3) should return false because
the leaf node with value 0 in tree2 has no corresponding node in tree3.
Note that since this method is in the IntTree class, it has access to the
private fields of IntTree objects, including overallRoot.

You may define private helper methods to solve this problem but you may not
call any other methods of the tree class nor create any data structures such
as arrays,
lists, etc. You should not construct any new node objects or change the data
of any nodes. For full credit, your solution must be recursive.

5. Collections Programming 15 Points.

Write a method called convertNames that takes as a parameter a list of strings representing names and that returns a map of strings to sets of strings that represent the same names split into first and last names. In particular, the names in the list passed to the method will each include a last name followed by a single comma and a space followed by the first name. You are guaranteed that there are no other commas in the string.

```
[“Monroe, James”, “Madison, James”, “Adams, John”, “Tyler, John”,
“Van Buren, "Martin”, “Jackson, Andrew”]
```

The method should split each string into the last name that comes before the comma and the space, and the first name that comes after.
The method should construct and return a map in which the keys are the first names and the values are the set of last names that match the first names.

For the list of names above, the following map would be constructed:

```
{“Andrew”=[“Jackson”], “James”=[“Madison”, “Monroe”],
“John”=[“Adams”, “Tyler”], “Martin”=[“Van Buren”]}
```

Notice, for example, that two of the names in the original list have “John” as the first name. In the map, the key "John" maps to a set of two elements (the two last names connected to “John”).

Your method should construct the new map and each of the sets contained in the map. Recall that the String class has a substring method that takes a starting index (inclusive) and a stopping index (exclusive). For example:

```
"Australia".substring(1, 5) returns "ustr"
```

The keys of the new map should be ordered alphabetically by the first names and each set should be ordered alphabetically by the last names contained in the set.

6. Comparable Prograaming 15 points. [Produce]

Define a class Produce that represents an item of produce sold at a grocery store. Each Produce object has a name, a weight,
a boolean to indicate whether the item is organic and a boolean to indicate whether the produce is a fruit. Your class must have the following public methods:

Constructor/Method	Description
public Produce(String name, double weight, boolean organic, boolean isFruit)	constructs a Produce object with the given name, weight, organic status and fruit status
public String toString()	returns a String representation of the Produce

The toString method returns a String composed of the name followed by the weight. If an item is organic,
the String should begin with an asterisk *. If the produce is a fruit, (F) is added after the name of the fruit.
You must exactly reproduce the format of the examples given below.

Make Produce objects comparable to each other using the Comparable<E> interface. Produce objects that are organic are
considered "less" than Produce that are not organic. In other words, all organic items should go before all non-organic items.
Within each organic and non-organic produce group, Produce that are fruits are considered "less" than Produce that are vegetables (non-fruit).
Then, they are sorted by name in ascending alphabetical order and ties are broken by weight in ascending order. For example, if the following objects are declared:

```
Produce simon = new Produce("Persimmon", 1.2, false, true);
Produce jack = new Produce("Jackfruit", 80.0, true, true);
Produce tom = new Produce("Tomato", 0.4, true, false); // tomatoes are
legally vegetables
Produce brock = new Produce("Broccoli", 2.3, false, false);
Produce lee = new Produce("Broccoli", 1.2, false, false);
Produce barry = new Produce("Raspberries", 0.6, true, true);
Produce larry = new Produce("Celery", 1.5, true, false);
```

Printing them in sorted order would result in the following output:

*Jackfruit (F) - 80.0 lbs
*Raspberries (F) - 0.6 lbs
*Celery – 1.5 lbs
*Tomato - 0.4 lbs
Persimmon (F) – 1.2 lbs
Broccoli - 1.2 lbs
Broccoli - 2.3 lbs

7. Binary Tree Programming 20 points. [trim]

Write a method trim that could be added to the IntTree class from lecture
and section. The method accepts minimum and maximum integers as parameters
and
removes from the tree any elements that are not within that range,
inclusive. For this method you should assume that your tree is a binary
search tree (BST)
and that its elements are in valid BST order. Your method should maintain
the BST ordering property of the tree.

For example, suppose a variable of type IntTree called tree stores the
following elements:

```
                         +----+
                         | 50 |
                   _____ +----+ _____
                  /                  \
              +----+                   +----+
              | 38 |                   | 90 |
        _____ +----+               ___ +----+
       /           \              /
    +----+         +----+   +----+
    | 14 |         | 42 |   | 54 |
    +----+         +----+   +----+
   /      \                      \
+----+   +----+                  +----+
|  8 |   | 20 |                  | 72 |
+----+   +----+                  +----+
              \                  /    \
            +----+           +----+  +----+
            | 26 |           | 61 |  | 83 |
            +----+           +----+  +----+
```

The table below shows what the state of the tree would be if various trim
calls were made. The calls shown are separate; it's not a
chain of calls in a row. You may assume that the minimum is less than or
equal to the maximum.

```
       tree.trim(25, 72);           tree.trim(54, 80);
            +----+                        +----+
            | 50 |                        | 54 |
          _ +----+                        +----+
         /        \                            \
    +----+        +----+                       +----+
    | 38 |        | 54 |                       | 72 |
    +----+        +----+                       +----+
    /    \            \                        /

+----+  +----+       +----+                +----+
| 26 |  | 42 |       | 72 |                | 61 |
+----+  +----+       +----+                +----+
                      /
                   +----+
                   | 61 |
                   +----+


       tree.trim(18, 42);            tree.trim(-3, 6);
             +----+
             | 38 |
            _+----+
          /         \
       +----+      +----+
       | 20 |      | 42 |
       +----+      +----+
          \
          +----+
          | 26 |
          +----+
```

Hint: The BST ordering property is important for solving this problem. If a
node's data value is too large or too small to fit within the range,
this may also tell you something about whether that node's left or right
subtree elements can be within the range. Taking advantage of such
information
makes it more feasible to remove the correct nodes.

You may define private helper methods to solve this problem, but otherwise
you may not call any other methods of the class nor create any data
structures
such as arrays, lists, etc.

Recall the IntTree and IntTreeNode classes as shown in lecture and section:

```
public class IntTreeNode {
    public int data;          // data stored in this node
    public IntTreeNode left;  // reference to left subtree
    public IntTreeNode right; // reference to right subtree

    public IntTreeNode(int data) { ... }
    public IntTreeNode(int data, IntTreeNode left, IntTreeNode right) {...}
}

public class IntTree {
    private IntTreeNode overallRoot;

    methods
}
```

8. LinkedIntList Programming 20 points. [Bubble]

Write a method called bubble that performs one pass of the bubble sort algorithm on a list of integers, returning true if any changes were made and returning false otherwise. As Wikipedia describes, bubble sort "works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted."

For example, suppose that a variable called list stores the following:

[5, 1, 4, 2, 8]
There are four pairs of adjacent values to compare, which leads to the following changes:

Starting list	New list	explanation
[5, 1, 4, 2, 8]	[1, 5, 4, 2, 8]	1st pair is swapped
[1, 5, 4, 2, 8]	[1, 4, 5, 2, 8]	2nd pair is swapped
[1, 4, 5, 2, 8]	[1, 4, 2, 5, 8]	3rd pair is swapped
[1, 4, 2, 5, 8]	[1, 4, 2, 5, 8]	no change

All of these changes should be made in the first pass (i.e., on a single call to bubble). Notice that the list is not sorted because this represents just one pass of the algorithm. A second pass would swap the pair (4, 2), which would leave the list in sorted order. A third pass would not swap any values.

Remember that your method is performing just one pass, but it should return true if it swapped something and should return false if no swaps were performed.
Thus, a client can use your method to sort a list by saying:

```
boolean sorted = false;
while (!sorted) {
    System.out.println(list);
    sorted = !list.bubble();
}
```

You are writing a public method for a linked list class defined as follows:

```
public class ListNode {
    public int data;       // data stored in this node
    public ListNode next;  // link to next node in the list

    <constructors>
}

public class LinkedIntList {
    private ListNode front;

    <methods>
}
```

```
You are writing a method that will become part of the LinkedIntList class.
You may define private helper methods to solve this problem,
but otherwise you may not assume that any particular methods are available.
You are allowed to define your own variables of type ListNode,
but you may not construct any new nodes, and you may not use any auxiliary
data structure to solve this problem (no array, ArrayList, stack,
queue, String, etc). You also may not change any data fields of the nodes.
You MUST solve this problem by rearranging the links of the list.
```