1.

```
        Preorder:  8 7 2 4 9 5 1 3 6 0
        Inorder:   4 2 9 7 8 3 1 6 5 0
        Postorder: 4 9 2 7 3 6 1 0 5 8
```

2.

```
                  +------------+
                  |   Hagett   |
                  +------------+
               /                  \
            /                        \
         /                              \
+------------+                      +------------+
|   Cedar    |                      |   Poplar   |
+------------+                      +------------+
              \                    /              \
               \                  /                \
                \                /                  \
         +------------+   +------------+      +------------+
         |    Elm     |   |   Mercer   |      |    Terry   |
         +------------+   +------------+      +------------+
                         /
                        /
                       /
                 +------------+
                 |  McMahon   |
                 +------------+
```

3.

Statement	Output
v1.m1();	Soda1
v2.m1();	Soda1
v3.m1();	DietCoke1
v4.m1();	Compiler Error
v5.m1();	DietCoke1
v6.m1();	Soda1
v1.m2();	Compiler Error
v2.m2();	Compiler Error
v3.m2();	Compiler Error
v4.m2();	Compiler Error
v5.m3();	Coke3 / DietCoke1
v6.m3();	Coke3 / Soda1
((Object)v2).m1();	Compiler Error
((Soda)v5).m1();	DietCoke1
((Pepsi)v4).m2();	Runtime Error
((Soda)v4).m2();	Compiler Error
((Soda)v4).m1();	Soda1
((DietCoke)v5).m3();	Coke3 / DietCoke1
((Soda)v6).m1();	Soda1

4.

```
public boolean sameStructure(IntTree other) {
   return sameStructureHelper(this.overallRoot, other.overallRoot);
}

private boolean sameStructureHelper(IntTreeNode root1, IntTreeNode root2) {
   if (root1 == null || root2 == null) {
      return root1 == root2;
   }

   return sameStructureHelper(root1.left, root2.left) &&
          sameStructureHelper(root1.right, root2.right);
}
```

5.

```
public Map<String, Set<String>> convertNames(List<String> names) {
        Map<String, Set<String>> firstToLast = new TreeMap<>();

        for (String name : names) {
                String[] parts = name.split(", ");
                String lastName = parts[0];
                String firstName = parts[1];

                if (!firstToLast.containsKey(firstName)) {
                firstToLast.put(firstName, new TreeSet<>());
                }

                firstToLast.get(firstName).add(lastName);
        }

        return firstToLast;
}
```

6.

```
public static class Produce implements Comparable<Produce> {
   private String name;
   private double weight;
   private boolean organic;
   private boolean isFruit;

   public Produce(String name, double weight, boolean organic, boolean isFruit) {
      this.name = name;
      this.weight = weight;
      this.organic = organic;
      this.isFruit = isFruit;
   }
```

```java
   public String toString() {
      String result = name;
      if (this.organic) {
         result = "*" + result;
      }

      if (this.isFruit) {
         result = result + " (F)";
      }

      result = result + " - " + this.weight + " lbs";
      return result;
   }

   public int compareTo(Produce other) {
      if (this.organic != other.organic) {
         if (this.organic) {
            return -1;
         }

         return 1;
      }

      if (this.isFruit != other.isFruit) {
         if (this.isFruit) {
            return -1;
         }

         return 1;
      }

      int cmp = this.name.compareTo(other.name);
      if (cmp != 0) {
         return cmp;
      }

      if (this.weight < other.weight) {
         return -1;
      } else if (this.weight > other.weight) {
         return 1;
      } else {
         return 0;
      }
   }
}
```

7.

One possible solutions appears below:

```
        public void trim(int min, int max) {
         this.overallRoot = trimHelper(min, max, this.overallRoot);
      }

      private IntTreeNode trimHelper(int min, int max, IntTreeNode root) {
         if (root != null) {
            root.left = trimHelper(min, max, root.left);
            root.right = trimHelper(min, max, root.right);

            if (root.data < min) {
               return root.right;
            }

            if (root.data > max) {
               return root.left;
            }
         }
         return root;
      }
```

An alternate solution appears beelow:

```
    public void trim(int min, int max) {
        overallRoot = trim(overallRoot, min, max);
    }

    private IntTreeNode trim(IntTreeNode root, int min, int max) {
        if (root != null) {
            if (root.data < min) {
                root = trim(root.right, min, max);
            } else if (root.data > max) {
                root = trim(root.left, min, max);
            } else {
                root.left = trim(root.left, min, max);
                root.right = trim(root.right, min, max);
            }
        }
        return root;
    }
```

8. One possible soltuion appears below.

```
  public boolean bubble() {
   boolean changed = false;
   // Check for list being at least length 2
   if (front != null && front.next != null) {

      // Handle Front Case
      if (front.next.data < front.data) {
         ListNode temp = front;
         front = front.next;
         temp.next = front.next;
         front.next = temp;
         changed = true;
      }

      // Hanlde Middle Case
      ListNode curr = front;
      ListNode first = curr.next;
      ListNode second = curr.next.next;

      while (first != null && second != null) {
         // Check for swap
         if (second.data < first.data) {
            // Swap
            first.next = second.next;
            second.next = first;
            curr.next = second;
            changed = true;
         }

         // Update curr, first, and second
         curr = curr.next;
         first = curr.next;
         second = curr.next.next;
      }

   }

   return changed;
}
```

An alternate solution appears below:

```
    public boolean bubble() {
        boolean swap = false;
        if (front != null && front.next != null) {
            if (front.data > front.next.data) {
                swap = true;
                ListNode temp = front;
                front = front.next;
                temp.next = front.next;
                front.next = temp;
            }
            ListNode current = front;
            while (current.next != null && current.next.next != null) {
                if (current.next.data > current.next.next.data) {
                    swap = true;
                    ListNode temp = current.next.next;
                    current.next.next = temp.next;
                    temp.next = current.next;
                    current.next = temp;
                }
                current = current.next;
            }
        }
        return swap;
    }
```