Final Key

1. Pre: 7 2 0 6 1 4 3 9 8 5
   In: 6 0 2 1 4 7 3 8 9 5
   Post: 6 0 4 1 2 8 5 9 3 7

2.
```
                        Moist
                   /              \
            Gaspode                 Vimes
            /      \               /
      Angua         Magrat   Rincewind
          \                        \
           Cohen                    Vetinari
```

3.
```
{lime=green, orange=blue, purple=grape, strawberry=red, yellow=lemon}
{may=green, rain=february, sunny=july}
{dog=cat, ham=eggs, milk=cookies, super=hero, watson=holmes}
{mints=chocolate, toffee=crisp, vanilla=cream}
```

4.

| **Statement** | **Output** |
|---|---|
| `var1.a();` | `Grape 1 / Lime 3 / Grape 3` |
| `var1.c();` | `Lime 3 / Grape 3` |
| `var2.a();` | `Grape 1 / Melon 3` |
| `var2.b();` | `Melon 2 / Lime 3 / Grape 3` |
| `var3.a();` | `Grape 1 / Apple 3` |
| `var3.b();` | `error` |
| `var4.a();` | `Grape 1 / Melon 3` |
| `var5.a();` | `error` |
| `((Melon) var5).a();` | `error` |
| `((Lime) var1).a();` | `Grape 1 / Lime 3 / Grape 3` |
| `((Lime) var4).a();` | `Grape 1 / Melon 3` |
| `((Melon) var3).b();` | `error` |
| `((Apple) var3).b();` | `Grape 1 / Apple 3 / Apple 2` |
| `((Lime) var4).c();` | `Melon 3` |
| `((Grape) var5).c();` | `Lime 3 / Grape 3` |

5.
```
 public class Course implements Comparable<Course> {
   private String department;
   private int number;
   private String instructor;
   private int size;
   private List<String> students;

   public Course(String department, int number, String instructor, int size) {
      if(number < 0 || size < 0 || department == null || instructor == null) {
         throw new IllegalArgumentException();
      }
      this.department = department;
      this.number = number;
      this.instructor = instructor;
      this.size = size;
      students = new ArrayList<String>();
   }

   public void register(String student) {
      if(!students.contains(student) && students.size() < size) {
         students.add(student);
      }
   }

   public String getName() {
      return department + " " + number;
   }

   public String getInstructor() {
      return instructor;
   }

   public boolean isSpace() {
      return students.size() < size;
   }

   public String toString() {
      double percent = ((int)(students.size() * 1000.0 / size))/10.0;
      return department + " " + number + " taught by " + instructor + " - " + percent + "% full";
   }

   public int compareTo(Course other) {
      double myPercent = ((double) students.size()) / size;
      double otherPercent = ((double) other.students.size()) / size;
      if (myPercent < otherPercent) {
         return -1;
      } else if (otherPercent < myPercent) {
         return 1;
      } else {
         if (size != other.size) {
            return other.size - size;
         } else {
            return getName().compareTo(other.getName());
         }
      }
   }
}
```

**6.** One possible solution appears below.

```
public int printLevel(int target) {
      if (target < 1) {
          throw new IllegalArgumentException();
      }
      return printLevel(overallRoot, target, 1);
  }

  private int printLevel(IntTreeNode root, int target, int level) {
      if (root != null) {
          if (level == target)
              return root.data;
          else {
              return printLevel(root.left, target, level + 1) +
                     printLevel(root.right, target, level + 1);
          }
      }
      return 0;
  }
```

**7.** One possible solution appears below.

```
public void stretch() {
    overallRoot = stretch(overallRoot, 1);
}

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

**8.** One possible solution appears below.

```
public void mergeFrom(LinkedIntList other) {
    if (other.front != null) {
        if (front == null)
            front = other.front;
        else {
            ListNode current1 = front;
            ListNode current2 = other.front;

            if (front.data <= other.front.data) {
                current1 = current1.next;
            } else {
                front = other.front;
                current2 = current2.next;
            }
            ListNode current3 = front;
            while (current1 != null && current2 != null) {
                if (current1.data <= current2.data) {
                    current3.next = current1;
                    current1 = current1.next;
                } else {
                    current3.next = current2;
                    current2 = current2.next;
                }
                current3 = current3.next;
            }
            if (current1 != null) {
                current3.next = current1;
            } else {
                current3.next = current2;
            }
        }
        other.front = null;
    }
}
```