Question 1. [7 points] State a big-O upper bound on the running time of the following method, where the problem size N is the number of elements in the array passed as the method's parameter. Explain your answer briefly.

```
public static int q1(int[] arr) {
  int sum = 0;
  for (int i = 0; i < arr.length; i++) {
    for (int j = 0; j < arr.length * arr.length; j++) {
      sum += arr[(i+j) % arr.length];
    }
  }
  return sum;
}
```

Question 2. [7 points] State a big-O upper bound on the running time of the following method, where the problem size N is the number of elements in the array passed as the method's parameter. Explain your answer briefly.

```
public static int q2(int[] arr) {
  int sum = 0;
  for (int i = 0; i < arr.length; i++) {
    for (int j = 0; j <= i; j++) {
      sum += arr[(i+j) % arr.length];
    }
  }
  return sum;
}
```

Question 3. [7 points] State a big-O upper bound on the running time of the following method, where the problem size N is the number of elements in the `ArrayList` passed as the method's parameter. Explain your answer briefly.

```
public static int q3(ArrayList<Integer> list) {
  int sum = 0;
  while (list.size() > 0) {
    sum += list.get(0);
  }
  return sum;
}
```

Question 4. [7 points] State a big-O upper bound on the running time of the following method, where the problem size N is the number of elements in the `ArrayList` passed as the method's parameter. Explain your answer briefly.

```
public static int q4(ArrayList<Integer> list) {
  int sum = 0;
  while (list.size() > 0) {
    sum += list.get(0);
    list.remove(0);
  }
  return sum;
}
```

Question 5. [7 points] State a big-O upper bound on the running time of the following method, where the problem size N is the number of elements in the `LinkedList` passed as the method's parameter. Explain your answer briefly.

```
public static int q5(LinkedList<Integer> list) {
  int sum = 0;
  for (int i = 0; i < list.size(); i++) {
    sum += list.get(i);
  }
  return sum;
}
```

Question 6. [7 points] State a big-O upper bound on the running time of the following method, where the problem size N is the number of elements in the `LinkedList` passed as the method's parameter. Explain your answer briefly.

```
public static int q6(LinkedList<Integer> list) {
  int sum = 0;
  Iterator<Integer> i = list.iterator();
  while (i.hasNext()) {
    sum += i.next();
  }
  return sum;
}
```

Question 7. [8 points] Complete the generic `findMin` method below. The method should return the minimum element in the `Collection` parameter `c`. You can assume that `c` will contain at least one element. Use the `Comparator` parameter `comp` to compare elements to each other. Hint: use an iterator to access the elements in the collection.

```
public static<E> E findMin(Collection<E> c, Comparator<E> comp) {
```

Question 8. [8 points] Complete the generic `Box` class below. An instance of `Box` should store one value of type `E`, where `E` is the element type. Use the following JUnit tests (which begin on the left and continue on the right) specify the `Box` class's required methods and behavior:

```
Box<String> bs =
  new Box<String>("hello");
Box<Integer> is =
  new Box<Integer>(42);

assertEquals("hello", bs.get());
assertEquals((Integer)42, is.get());
```

```
bs.set("yeah");
is.set(17);

assertEquals("yeah", bs.get());
assertEquals((Integer)17, is.get());
```

```
public class Box<E> {
```

Programming Question

To get started, use a web browser to download the zipfile as specified by your instructor. Import it as an Eclipse project using File → Import... → General → Existing Projects into Workspace → Archive file.

Important: You may use the following resources:

- The textbook
- The lecture notes posted on the course web page
- Your previous labs and assignments

Do not open any other files, web pages, etc.

Question 9. [42 points] In the `Q9` class, complete the generic method called `isAscending`. It should return `true` if the elements in the `list` parameter are arranged in strictly ascending order, `false` otherwise. "Strictly ascending" means that for each adjacent pair of elements, the later element is strictly greater than the earlier element. The method must use the `Comparator` parameter `comp` to compare elements to each other.

Ideally, the method should execute in $O(N)$ time, where N is the number of elements in the list.

Hint: use an iterator to access the elements in the list:

```
Iterator<E> i = list.iterator();
```

Keep in mind that because the method is generic, the list's elements have type `E`. So, if you want to declare a variable for a list element, its type must be `E`. E.g.:

```
E elt = i.next();
```

JUnit tests are provided in the class `Q9Test` class. Make sure that all of the tests pass.

Bonus Question. [10 points] Implement the `isDescending` method in the `Q9` class. It should return `true` if the elements in the `list` parameter are strictly *descending*, false otherwise.

Requirement: the `isDescending` method must make a call to `isAscending`, using it to determine whether the elements are in strictly *descending* order.

Hint: You may define and additional class or classes as necessary.

JUnit tests are provided in the `Q9BonusTest` class.