CS 201, Fall 2012 — Nov 16th — Exam 2 Name: _____

**Note:** In questions where you are asked to complete the implementation of a static method, assume that the method is in a class called `Q`$n$ where $n$ is the question number, e.g., `Q1` for Question 1.

**Question 1**. [10 points] Consider the following JUnit test, which uses the `Arrays.sort` static method to sort an array of `Integer` values:

---

```
Integer[] values = new Integer[]{4, 3, 8, 9, 1};
Comparator<Integer> comp = new MysteryComparator();
Arrays.sort(values, comp);

assertEquals((Integer) 9, values[0]);
assertEquals((Integer) 8, values[1]);
assertEquals((Integer) 4, values[2]);
assertEquals((Integer) 3, values[3]);
assertEquals((Integer) 1, values[4]);
```

---

Complete the definition of the `MysteryComparator` class below so that the assertions in the JUnit test all pass. Hint: think about the order of the values in the array following the call to `Arrays.sort`.

```
public class MysteryComparator implements Comparator<Integer> {
  public int compare(Integer left, Integer right) {
```

**Question 2**. [10 points] Complete the definition of the `countLessThan` method below. The method takes two parameters: `arr`, an array of values of type `E`, and `value`, a value of type `E`. The method returns an integer which is the number of elements of `arr` that compare as less than `value`. Use the `compareTo` method to compare elements of type `E`.

Example JUnit test:

---

```
Integer[] values = new Integer[]{4, 3, 8, 9, 1};

assertEquals(3, Q2.countLessThan(values, (Integer)5));
assertEquals(4, Q2.countLessThan(values, (Integer)9));
```

---

```
public static<E extends Comparable<E>> int countLessThan(E[] arr, E value) {
```

**Question 3**. [10 points] Complete the implementation of the `difference` method below. Given collections `a` and `b`, both of which are collections of elements of type `E`, the method should return a collection which has all of the elements of `a` that are not elements of `b`.

**Important**: the method must complete in $O(n)$ or $O(n \log n)$ time, where $n$ is the total number of elements in `a` and `b`.

Example JUnit test (which begins on the left and continues on the right):

```
Collection<String> people =
  new ArrayList<String>();
people.add("Alice");
people.add("Bob");
people.add("Cormac");
people.add("Delores");
people.add("Edith");

Collection<String> men =
  new ArrayList<String>();
men.add("Bob");
men.add("Cormac");
men.add("Zebulon");
```

```
Collection<String> diff =
  Q3.difference(people, men);

assertTrue(diff.contains("Alice"));
assertTrue(diff.contains("Delores"));
assertTrue(diff.contains("Edith"));
assertFalse(diff.contains("Bob"));
assertFalse(diff.contains("Cormac"));
```

```
public static<E extends Comparable<E>>
Collection<E> difference(Collection<E> a, Collection<E> b) {
```

**Question 4**. [10 points] Consider the following method:

```
public static int countStartingWith(List<String> list, char first) {
  int count = 0;
  for (int i = 0; i < list.size(); i++) {
    String s = list.get(i);
    if (s.length() > 0 && s.charAt(0) == first) {
      count++;
    }
  }
  return count;
}
```

(a) State a big-O upper bound on the running time of the method if `list` is an `ArrayList`, where the problem size $N$ is the number of elements in the list. Explain briefly.

(a) State a big-O upper bound on the running time of the method if `list` is an `LinkedList`, where the problem size $N$ is the number of elements in the list. Explain briefly.

**Question 5**. [10 points] Complete the definition of the `removeDups` static method below. It takes a `List` of elements of type `E` as a parameter, and returns a `List` which contains the same elements in the same order, except that only the first occurrence of any value is preserved.

**Important**: the method must complete in $O(n)$ or $O(n \log n)$ time, where $n$ is the number of elements in the list.

**Hint**: iterate through the list using a `TreeSet` to keep track of which elements have been encountered previously.

Example JUnit test:

---

```
List<String> list = new ArrayList<String>();
list.add("A");
list.add("B");
list.add("B");
list.add("D");
list.add("E");
list.add("D");

List<String> nodups = Q5.removeDups(list);
assertEquals(4, nodups.size());
assertEquals("A", nodups.get(0));
assertEquals("B", nodups.get(1));
assertEquals("D", nodups.get(2));
assertEquals("E", nodups.get(3));
```

---

```
public static<E extends Comparable<E>> List<E> removeDups(List<E> list) {
```

**Question 6**. [5 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 a parameter. Briefly explain your answer.

```
public static<E> void reverse(E[] arr) {
  for (int i = 0; i < arr.length / 2; i++) {
    int j = arr.length - i - 1;
    E tmp = arr[i];
    arr[i] = arr[j];
    arr[j] = tmp;
  }
}
```

**Question 7**. [5 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 a parameter. Briefly explain your answer.

```
public static int mystery(int[] arr) {
  int count = 0;
  for (int i = 0; i < arr.length; i++) {
    for (int j = i+1; j < arr.length; j++) {
      if (arr[i] == arr[j]) {
        count++;
      }
    }
  }
  return count;
}
```

**Question 8**. [10 points] Complete the following recursive method. Given a string `s`, it should return a string containing all of the characters of the original string, doubled.

Example test:

```
String str = "Recursion is fun";
assertEquals("RReeccuurrssiioonn  iiss  ffuunn", doubleCharacters(str));
```

Note: your method *must* use recursion. Do not use a loop.

Make sure you have an appropriate base case. Hint: call the `length()` method on the string to see how many characters it contains.

```
public static String doubleCharacters(String s) {
```

**Question 9**. [30 points] Programming problem — see instructions in class.