CS 201, Summer 2014 — Aug 11th — Exam 2 Name: Solution

Note: In questions where you are asked about 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 code:

```
int count = 0;
for (int i = 0; i < n * n; i++) {
    for (int j = 0; j < n; j++) {
        count++;
    }
}
```

n^2 times

n times

$O(1)$

State a big-O upper bound for this code, using n (the value of the variable n) as the problem size. Briefly explain your answer.

$$n^2 \cdot n \cdot C \quad \text{is} \quad O(n^3)$$

Question 2. [10 points] Consider the following code:

```
int count = 0;
for (int i = 0; i < n * n; i++) {
    for (int j = 0; j <= i; j++) {
        count++;
    }
}
```

dependent inner loop $\longrightarrow$

$1, 2, 3, 4, \dots, n^2$ times

$O(1)$

State a big-O upper bound for this code, using n (the value of the variable n) as the problem size. Briefly explain your answer.

$$\sum_{i=1}^{n^2} i = \underbrace{\overbrace{1 + \overbrace{2 + 3 + \dots + (n^2-1)}^{n^2+1} + n^2}^{n^2+1}}_{n^2/2 \text{ pairs of terms}}$$

$$= \frac{n^2}{2}(n^2+1) = \frac{n^4}{2} + \frac{n^2}{2} \quad \text{is} \quad O(n^4)$$

Question 3. [10 points] Complete the definition of the `CaseInsensitiveCharacterComparator` class. Its behavior is shown by the following JUnit test code:

```
Character[] letters = { ’H’, ’a’, ’E’, ’k’, ’D’ };
Arrays.sort(letters, new CaseInsensitiveCharacterComparator());

assertEquals((Character)’a’, letters[0]);
assertEquals((Character)’D’, letters[1]);
assertEquals((Character)’E’, letters[2]);
assertEquals((Character)’H’, letters[3]);
assertEquals((Character)’k’, letters[4]);
```

Hint: You can use the `Character.toLowerCase` method to convert a character value to an equivalent lower case character value. E.g., `Character.toLowerCase(’A’)` would return `’a’`.

```
public class CaseInsensitiveCharacterComparator implements Comparator<Character> {
    public int compare(Character left, Character right) {
            char l = Character.toLowerCase(left);
            char r = Character.toLowerCase(right);

            if (l < r) {
                return -1;
            } else if (l == r) {
                return 0;
            } else {
                return 1;
            }

    }
```

Question 4. [5 points] Consider the following method:

```
public static int countEvens(LinkedList<Integer> list) {
    int count = 0;
    for (int i = 0; i < list.size(); i++) {     — N times
        if (list.get(i) % 2 == 0) { count++; }
    }
    return count;   O(N) on average
}
```

State a big-O upper bound for this method, where the problem size N is the number of elements in the `list` parameter. Briefly explain your bound.

$O(N^2)$ because loop executes N times and get(int) is $O(N)$ on average for LinkedList (since time is proportional to i, which is about $N/2$ on average),

Question 5. [5 points] Consider the following method:

```
public static int countEvens(LinkedList<Integer> list) {
    int count = 0;
    for (Iterator<Integer> i = list.iterator(); i.hasNext(); ) {   — N times
        Integer value = i.next();   O(1)
        if (value % 2 == 0) { count++; }
    }
    return count;
}
```

State a big-O upper bound for this method, where the problem size N is the number of elements in the `list` parameter. Briefly explain your bound.

$O(N)$, because all Iterator operations (including next()) are $O(1)$ for LinkedLists

Question 6. [10 points] Consider the following static method:

```
public static<E extends Comparable<E>> List<E> mystery(List<E> src) {
    Set<E> set = new TreeSet<E>();
    for (E elt : src) { set.add(elt); }

    List<E> result = new ArrayList<E>();
    for (E elt : set) { result.add(elt); }

    return result;
}
```

N times

O(log N)

O(N) times

O(1)

(a) State a big-O upper bound on the running time of this method. Assume that the problem size N is the number of elements in the list parameter src. Briefly explain your answer.

O(N log N) because of the first loop, which ~~[illegible]~~ executes N times and incurs O(log N) for each element added to the TreeSet. (The second loop is O(N), so doesn't affect the big-O bound overall.)

(b) What output is printed by the following code?

```
List<Integer> myList = new ArrayList<Integer>();
myList.add(9);
myList.add(0);
myList.add(1);
myList.add(2);
myList.add(5);

List<Integer> result = Q6.mystery(myList);
for (Integer x : result) { System.out.print(x + " "); }
```

0 1 2 5 9