Name: _____

Please refer to the previous assignments for important instructions on the allowable use of resources and the requirements for electronic submission.

1. (1 pt.) Write your name in the space provided above.

2. (1 pt.) Package your solutions to the programming questions as in previous assignments. Submit your work via `http://cs.umaine.edu/~chaw/u/`. *After* submitting your work, complete the following:
   File name: _____ Size, in bytes: _____
   MD5 checksum: _____
   Timestamp: _____

3. (10 pts.) The textbook describes a simple method for detecting duplicates in an input array.[1] Modify that method, as little as possible, to count the number of occurrences of each array element. The modified method should return an integer array `b` that has the same size as the input array `a`, with `b[i]` being the number of occurrences of `a[i]` in the array `a`. For example, given $\texttt{a} = (3, 1, 4, 1, 5, 3, 1, 2)$, the desired result is $\texttt{b} = (2, 3, 1, 3, 1, 2, 3, 1)$.

   Present the modified method as Java code, or as pseudocode at a similar level of detail. Describe its asymptotic running time as a function of the size of the input array.

[1] Mark Allen Weiss, *Data Structures and Problem Solving Using Java*, 3rd edition (Addison-Wesley, 2006), Figure 8.1, p. 305.

[additional space for answering the earlier question]

4. (25 pts.) Read the *Sun Java 6* documentation[2] of the method `Collections.sort` and answer the following briefly:

   (a) Describe, as precisely as possible, how the usability of the method would change if we were to modify the signature

   ```
   public static <T extends Comparable<? super T>>
      void sort(List<T> list)
   ```

   by replacing `<? super T>` with `<T>`.

   (b) Why does the documentation also include the following?

   ```
   public static <T> void sort(List<T> list,
                               Comparator<? super T> c)
   ```

   Provide a concrete example of a situation in which the variant of Question 4a (unmodified) cannot be easily used but this one can.

[2] `http://java.sun.com/javase/reference/`.

(c) Which sorting algorithm does `Collections.sort` use, and what performance guarantee does it offer?

(d) What does it mean for a sorting method to be *stable*? Is `Collections.sort` stable?

(e) Does the implementation sort the input list in place? Why?

(f) The documentation refers to a task that requires $n^2 \log n$ time. What is it? Explicitly describe a method for that task and prove that the running time of your method is $\Theta(n^2 \log n)$.

5. (8 pts.) Describe an alternate method for computing the number of occurrences of each array element, as described in Question 3, based on sorting the input. You may use the Java library method `Collections.sort`. As in Question 3, present your method as Java code or detailed pseudocode, and describe its asymptotic running time.

6. (30 pts.) Consider a function *boosort* defined on lists as follows, where $\min(l)$ returns the minimum value in l and where $\cdot$ denotes list concatenation.

$$\text{boosort}(l) \quad = \quad \begin{cases} l & \text{if } l = () \\ (\min(l)) \cdot \text{boo}(l) & \text{otherwise} \end{cases}$$

   (a) If boosort is to live up to its name and return the elements of its input l in sorted order, how must the function boo be defined? Define boo by providing Java code or detailed pseudocode. Explain why your definition results in boosort having the desired property. [Hint: Be sure to properly handle duplicates in l.]

(b) Trace the evaluation of boosort on the list $(19, 87, 52, 76, 17, 35, 20, 1, 91, 5)$. For each recursive invocation boosort(x), clearly indicate the argument x, the result of boo(x), and the result of boosort(x).

(c) Is boosort, with your definition of boo, stable? Justify your answer.

(d) Describe the asymptotic running time of boosort as a function of the number of elements in the input list.

(e) Provide Java code or detailed pseudocode for a *nonrecursive* variant of boosort, say *noosort*, that is otherwise very similar to boosort.

(f) Trace the evaluation of your definition of noosort from Question 6e on the input list of Question 6b. Depict the states of the main data structures used by your method (array, list, etc.) at each iteration, in a manner similar to that used by the textbook to trace insertion sort.[3]

[3]Weiss, *op. cit.*, Figure 8.3, p. 306.

(g) Is noosort stable? Explain your answer.

7. (25 pts.) Refer to the description of the *marked simple digital trie* in the previous assignment.

   (a) Define a traversal on marked simple digital tries that visits the marked nodes in ascending order of the keys they represent. You may use formal notation (say, similar to what we have used for other traversals) or plain English for your definition. In either case, the description must be precise enough to enable others to implement the traversal.

(b) Provide Java code, or detailed pseudocode, for the traversal of Question 7a.

(c) Devise an algorithm, *digitriesort*, to sort a collection of nonnegative integers using a marked simple digital trie and the traversal of Question 7a. Ensure that your method works correctly when there are duplicates in the input. You may use pseudocode or plain English to describe your algorithm. In either case, the description must be precise enough to enable others to implement the algorithm.

(d) Provide Java code, or detailed pseudocode, for the method of Question 7c.

(e) Characterize, as precisely as possible, the running time of your method for Question 7c as a function of the number and length of the integers in the input.

8. (20 pts.) Implement the methods you describe for Questions 3 and 5. Your implementation should read the elements of the input array `a` from standard input (one element per line). The array `b` should be computed using the each of the two methods, and your implementation must check that the result is identical. The output of your implementation should be the elements of `b`, one per line, followed by the running times for computing `b` using each method (in milliseconds, also one per line). For this question, and all that follow, make sure the README file in your submission includes the appropriate instructions for testing your work.

9. (40 pts.) Implement *boosort* and *noosort* from Question 6. Your implementations should be completely interchangeable with the standard Java methods *Collections.sort*. That is, replacing `Collections.sort` with `boosort` or `noosort` should not change program behavior, except for possible performance differences. Write a program that reads a nonnegative integer n from standard input, generates a list of n random integers, sorts them using each of `Collections.sort`, boosort, and noosort, and writes to standard output the time (in milliseconds, one per line) required for each of the three sorting implementations. Your source code should make it obvious that your boosort and noosort implementations are interchangeable with `Collections.sort`.

10. (40 pts.) Extend your implementation of Question 9 to include the *digitriesort* method of Question 7 in addition to the three methods used earlier. The output should be augmented with the running time of *digitriesort* in the natural way.