Programming Assignment: Sorting and Binary File I/O in Java

This part of the assignment involves implementing a smallish Java program that performs some basic input/output involving binary files and implements a radix sort algorithm.

The program will deal with a binary input file, named Unsorted.bin, that contains a collection of data records of the following form:

```
[ 8-byte non-negative integer | 4-byte integer ]
```

The first part may be treated as a Java `long` and the second as a Java `int` (or their wrapped equivalents `Long` and `Integer`). The number of records is unspecified, but will never be more than 100,000.

Your program will read the records from the input file and create an appropriate in-memory linear data structure to hold them. You may use a Java library type for this, or implement your own container.

Next, your program will employ a radix-sort algorithm to sort the records into ascending order, using the 8-byte integer field as the sort key. For efficiency, the radix-sort should avoid base-10 operations and take advantage of the underlying bit-level representation of the values, and therefore employ bit-wise operations supported in Java.

Finally, your program will write the records, in sorted order, into a binary file named Sorted.bin.

Supplied Tools

For your convenience we will supply two tools:

`MakeDataFile.exe <number of records> <binary file name>`

> This creates a sequence of randomly-ordered records of the form described above, and writes them in binary format to a file with the specified name.

`CheckDataFile.exe <binary file name>`

> This parses a binary file, as described above, and determines whether the records are in ascending order, by the sort key. If not, a descriptive error message is written to standard output.

The tools were written in C, compiled with gcc, and are provided as Windows executables. Linux versions may be supplied upon request.

An Implementation Hint

When implementing radix sort, you will want to create arrays of linked lists to serve as the bins. When doing so, you will encounter an annoying limitation of Java generics. For instance, if you attempt to instantiate an array as follows:

```
private LinkedList<Integer>[] Bins = new LinkedList<Integer>[10];
```

you will receive an error message:

```
RadixSorter.java:9: generic array creation
      . . . Bins = new LinkedList<Integer>[10];
                   ^
```

If your first thought is that you are not instantiating an array of generic objects, since `LinkedList<Integer>` does not involve a generic type specifier, look up *type erasure* and read up on it. Fortunately, there is a relatively trivial fix.

If you declare a trivial class to serve as a front for `LinkedList<Integer>`:

```
public class IntegerList extends LinkedList<Integer> { }
```

you can then instantiate an array of `IntegerList` objects as you would expect:

```
private IntegerList[] Bins = new IntegerList[10];
```

Note that you probably do not want an array of `Integer` objects for this project, but you will want something similar.

Testing:

It is your responsibility to design and conduct thorough and sensible tests of your implementation before submitting it. For that purpose, you may share test input and output files (but absolutely no solution code!!) via the Piazzza board. You should **not** use the Curator for your own testing and debugging purposes. The curator will only accept a limited number of submissions by each student, so use them wisely for locking in your grades.

Evaluation:

You should document your implementation in accordance with the *Programming Standards* page on the course website. It is possible that your implementation will be evaluated for documentation and design, as well as for correctness of results. If so, your submission that achieved the highest score will be evaluated by one of the TAs, who will assess a deduction (ideally zero) against your score from the Curator.

Your score on the program will count as 50% of your score for this assignment.

Experimental Assignment: Measuring and Comparing Performance

Note: you are allowed to work in pairs for this part of the assignment (but NOT for the first part!).

This part of the assignment requires that you design and perform some experiments that measure the performance of some common sorting algorithms:

- Insertion Sort (as a baseline versus the other algorithms)
- Quicksort
- Heapsort

Now, implementations of all the common sorting algorithms are readily available online (and even in data structures textbooks), so we do not assume or require that you implement these yourself. But, be sure the implementations you use do actually sort correctly.

For each algorithm, you will instrument an implementation to count the number of data object comparisons and the number of data object copy operations that are performed. (Since in most situations a Java implementation would actually copy object references rather than the objects themselves, you will count the number of times that references to data objects are copied.) Count these operations only in the portions of the code related to the sorting algorithms, not in the portions related to data input or output.

You will measure performance by sorting arrays of Java `Integer` objects. You will examine the performance of each algorithm on a variety of input sizes; it is up to you to decide what range of sizes and how many cases to consider, but you should use the same data sets for all the algorithms so you can sensibly compare the results.

After performing the testing for each algorithm, you will write a report that summarizes your experimental results in detail, and compares your measurements to what you would expect, given the theoretical complexity results we have for each of these algorithms. A good report will, of course, include a tabular and graphical presentation of results for each algorithm, and a description of the data used and how it was chosen. But an acceptable report must include a written discussion of the results as well.

Include the source code for each of the sorting algorithms at the end of your report. If you obtained source code from the internet or a book, include a proper citation of the source in the header comment for the algorithm code.

Your report will be evaluated according to the following criteria:

- the quality of your experimental design (number of cases, variation of data sets, etc.)
- the credibility of your reported results
- the quality of your discussion of your reported results in comparison to the expected theoretical performance

Your score on the report will count as 50% of your grade on this assignment.

What to turn in and how:

The first part of the assignment will be auto-graded on the Curator system. The testing will be done under Windows (which should not matter at all) using Java version 1.6.21 or later.

Submit a single `zip` file (not an RAR or other compressed format) containing the source code for your solution to the Curator System. Submit nothing else. Your solution should not write anything to standard output (i.e., `System.out` in Java).

Your source code must be "flat". That is, you must not place code in subdirectories or use Java packages. Doing so will ensure that your submitted code does not compile.

Your main class (the one that implements `public static void main()`) must be named `MinorP4`. If not, your submitted code will fail to execute properly.

The second part of the assignment will also be submitted to the Curator, as a single file that can be edited in MS Word. The file should list the name(s) and PID(s) of the student(s) who contributed to the report.

Instructions, and the appropriate link, for submitting to the Curator are given in the *Student Guide* at the Curator website:

`http://www.cs.vt.edu/curator/`.

You will be allowed to submit your solution multiple times, up to a limited number as indicated in the Curator system; the highest score will be counted.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement provided on the Programming Standards page in one of your submitted files.