External Sorting

For this project, you will implement an external sorting algorithm for binary data. The input data file will consist of 8N blocks of data, where a block is 4096 bytes. Each block will contain a series of records, where each record has 8 bytes. The first 4-byte field is a non-negative integer value for the record ID and the second 4-byte field is a `float` value for the key, which will be used for sorting. Thus each block contains 512 records.

Your job is to sort the file (in ascending order of the key values), as follows:

Using replacement selection (as described in the class notes), you will sort sections of the file in a working memory that is 8 blocks long. To be precise, the heap will be 8 blocks in size; in addition you will also have a one block input buffer, a one block output buffer and any additional working variables that you need. To process, read the first 8 blocks of the input file into memory and use replacement selection to create the longest possible run. As it is being created, the run is output to the one block output buffer. Whenever this output buffer becomes full, it is written to an output file called the run file. When the first run is complete, continue on to the next section of the input file, adding the second run to the end of the run file. When the process of creating runs is complete, the run file will contain some number of runs, each run being at least 8 blocks long, with the data sorted within each run. For convenience, you will probably want to begin each run in a new block.

You will then use a multi-way merge to combine the runs into a single sorted file. You must also use 8 blocks of memory used for the heap in the run-building step to store working data from the runs during the merge step. Multi-way merging is done by reading the first block from each of the runs currently being merged into your working area, and merging these runs into the one block output buffer. When the output buffer fills up, it is written to another output file. Whenever one of the input blocks is exhausted, read in the next block for that particular run. This step requires random access (using seek) to the run file, and a sequential write of the output file. Depending on the size of all records, you may need multiple passes of multiway-merging to sort the whole record.

Program Invocation and Operation:

The program will take the names of two files from the command line, like this:

```
java extsort <record file name> <statistics file name>
```

If the specified record file does not exist, output a suitable error message and exit. Your program will create the statistics file if it does not already exist, and append to it if it does.

The record file is the file to be sorted. At the end of your program, the record file (on disk) should be sorted. So this program does modify the input data file. Be careful to keep a copy of the original when you do your testing.
In addition to sorting the data file, you must report some information about the execution of your program.

1. You will need to report part of the sorted data file to standard output.\

   Specifically, your program will print the first record from each 4096- byte block, in order, from the sorted data file. The records are to be printed 5 records to a line (showing both the key value and the id value for each record), the values separated by whitespace and formatted into columns. This program output must appear EXACTLY as described; ANY deviation from this requirement will result in a significant deduc- tion in points.

2. You will generate and output some statistics about the execution of your program. Formatting does not matter as long as we can tell what each statistic is. Write these statistics to the statistics file. Your code will be tested with different sample files. Make sure your program DOES NOT overwrite the statistics file each time it is run; instead, have it append new statistics to the end of the file. The information to write is as follows.

   (a) The name of the data file being sorted.
   (b) The time that your program took to execute the heapsort. Put two calls to the standard Java timing method "System.currentTimeMillis()" in your program, one at the beginning and another at the end. This method returns a long value. The difference between the two values will be the total runtime in milliseconds. You should ONLY time the sort, and not the program output as described above.

Programming Standards

You must conform to good programming/documentation standards. Some specifics:

- Your header comment must describe what your program does; don't just plagiarize language from this spec.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Always use named constants or enumerated types instead of literal constants in the code.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe how it works unless you do something so sneaky it deserves special recognition.
- You must use indentation and blank lines to make control structures more readable.
- Precede each function and/or class method with a header comment describing what the function does, the logical significance of each parameter (if any), and pre- and post-conditions.
- Decompose your design logically, identifying which components should be objects and what operations should be encapsulated for each.

Neither the GTAs nor the instructors will help any student debug an implementation unless it is properly documented and exhibits good programming style. Be sure to begin your internal documentation right from the start. You may only use code you have written, either specifically for this project or for earlier programs, or code taken from the textbook. Note that the textbook code is not designed for the specific purpose of this assignment, and is therefore likely to require modification. It may, however, provide a useful starting point.

Testing

A sample data file will be posted to the website to help you test your program. This is not the data file that will be used in grading your program. The test data provided to you will attempt to exercise the various syntactic elements of the command specifications. It makes no effort to be comprehensive in terms of testing the data structures required by the program. Thus, while the test data provided should be useful, you should also do testing on your own test data to ensure that your program works correctly.

Administrative Issues

Project Schedule:

You will find a template for planning a development schedule on the course website. You are required to submit three versions by the due date for the project solution:

Deadline	Description
Oct 14	Initial plan
Oct 21	Revised plan
Final project submission	Summary of the actual schedule you followed, up to the point you completed the project. This should be submitted shortly after you have submitted your final solution for the assignment.

Failure to submit each of these by the specified deadline will result in a deduction of 5% from your final score on the project.

Submission:

You will submit this assignment to the Curator System (read the *Student Guide*), where it will be archived for grading at a demo with a TA.

For this assignment, you must submit a zip (or jar) file containing all the source code files for your implementation (i.e., the `java` files). If you need to explain any pertinent information to aid the TA in the grading of your project, you may include an optional "README" file in your submitted archive. Submit nothing else.

In order to correct submission errors and late-breaking implementation errors, you will be allowed up to five submissions for this assignment. Unless you notify us otherwise, the TAs will evaluate the last version you submit.

The Student Guide and link to the submission client can be found at: `http://www.cs.vt.edu/curator/`

Evaluation:

The TAs will evaluate the correctness of your results. In addition, the TAs will evaluate your project for good internal documentation and software engineering practice.

Remember that your implementation will be tested using version 1.6.0_16 of `javac`. If you use a different development platform, it is entirely your responsibility to make sure your implementation works correctly in the lab.

Note that the evaluation of your project will depend substantially on the quality of your code and documentation. See the Programming Standards page on the course website for specific requirements that should be observed in this course.

Pedagogic points:

The goals of this assignment include, but are not limited to:

- documented planning of a formal timeline for completing a development project
- understanding how to navigate and parse a binary file in Java
- creation of a sensible OO design for the overall system, including the identification of a number of useful classes not explicitly named in this specification
- implementation of such an OO design into a working system
- incremental testing of the basic components of the system in isolation
- satisfaction when the entire system comes together in good working order

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 Submitting Assignments page of the course website.