CS3114 (Spring 2019)

PROGRAMMING ASSIGNMENT #3

Due Tuesday, Apr 11 @ 11:00 PM for 100 points

Early bonus date: Sun, Apr 9 @ 11:00 PM for a 10 point bonus

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 8,192 bytes. Each block will contain a series of records, where each record has **16** bytes. The first 8-byte field is a non-negative integer value (long) for the record ID and the second 8-byte field is a double 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 Section 9.6 in the OpenDSA in Canvas), 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 file.

Program Invocation and Operation:

The program will take the names of one file from the command line, like this:

java Externalsort <record file name>

You may assume that the specified record file does exist in our test cases. This 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. You will need to report part of the sorted data file to standard output. Specifically, your program will print (to standard output) the first record from each 8192-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 may result in a significant deduction in points.

Programming Standards:

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

- You must include a header comment, preceding main(), specifying the compiler and operating system used and the date completed.
- 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 codes 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:

Sample data files will be provided in class 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.

Deliverables:

You will implement your project using Eclipse, and you will submit your project using the Eclipse plugin to Web-CAT. Links to the Web-CAT client are posted at the class website. If you make multiple submissions, only your last submission will be evaluated. There is no limit to the number of submissions that you may make.

You are required to submit your own test cases (should cover at least 70% of your code) with your program. Of course, your program must pass your own test cases. Your grade will be fully determined by test cases that are provided by the graders (TAs). Web-CAT will report to you which test files have passed correctly, and which have not. Note that you will not be given a copy of these test files, only a brief description of what each accomplished in order to guide your own testing process in case you did not pass one of our tests.

To be able to write tests that can be run by webcat, you should go to the following link:

http://web-cat.org/junit-quickstart/
and download the student.jar file. You then need to reference this file as an external library. The library extends the existing JUnit library, so you can write tests using regular JUnit syntax. The webpage also includes some helpful hints to help you test better.

When structuring the source files of your project, use a directory structure; that is, your source files will all be contained in the project "src" directory. Any subdirectories in the project will be ignored.

You are permitted (and encouraged) to work with a partner on this project. When you work with a partner, then only one member of the pair will make a submission. Both names and emails **must** be included in the documentation and selected on any Web-CAT submission. The last submission from either of the pair members will be graded.

In this project, you should pay attention to the efficiency of your code. There will be test cases that even though your code is correct, its running time may exceed our limit and that will be considered a failed test. The time limit is set from TAs' running time with those test cases plus some extra time.

Pledge:
Your project submission must include a statement, pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the following pledge statement near the beginning of the file containing the function main() in your program.

```
// On my honor:
//
// - I have not used source code obtained from another student,
// or any other unauthorized source, either modified or
// unmodified.
//
// - All source code and documentation used in my program is
// either my original work, or was derived by me from the
// source code published in the textbook for this course.
//
// - I have not discussed coding details about this project with
// anyone other than my partner (in the case of a joint
// submission), instructor, ACM/UPE tutors or the TAs assigned
// to this course. I understand that I may discuss the concepts
// of this program with other students, and that another student
// may help me debug my program so long as neither of us writes
// anything during the discussion or modifies any computer file
// during the discussion. I have violated neither the spirit nor
// letter of this restriction.
```

Programs that do not contain this pledge will not be graded.