

## CS3114/5040 (Fall 2023)

### PROGRAMMING ASSIGNMENT #3

Due Thursday, November 2 @ 11:00 PM for 100 points

Due Wednesday, November 1 @ 11:00 PM for 10 point bonus

Note: This project also has three intermediate milestones. See the Piazza forum for details.

Last update 10/12 @ 6:30pm

#### Assignment:

You will implement a sorting algorithm on a file. The input data file will consist of many 4-byte records, with each record consisting of two 2-byte (short) integer values in the range 1 to 30,000. The first 2-byte field is the key value (used for sorting) and the second 2-byte field contains a data value. The input file is guaranteed to be a multiple of 4096 bytes. All I/O operations will be done on blocks of size 4096 bytes (i.e., 1024 logical records).

**Warning:** The data file is a **binary** file, not a text file.

Your job is to sort the file (in ascending order), using a modified version of Quicksort. The modification comes in the interaction between the Quicksort algorithm and the file storing the data. The array being sorted will be the file itself, rather than an array stored in memory. All accesses to the file will be mediated by a buffer pool. The buffer pool will store 4096-byte blocks (1024 records). The buffer pool will be organized using the Least Recently Used (LRU) replacement scheme. See OpenDSA Module 9.4 for more information about buffer pools.

Note that you are **not** implementing an external sorting algorithm, as described in Module 9.6 of OpenDSA. What you are implementing is a standard in-memory Quicksort that has been modified to use a “virtual memory” in the form of a large array stored on disk. The primary modification to your Quicksort algorithm from the in-memory version is to change the in-memory array accesses to accesses to the disk file (these accesses actually come through the buffer pool).

#### Design Considerations:

The primary design concern for this project will be the interaction between the logical array as viewed by the Quicksort algorithm, and the physical representation of the array as implemented by the disk file mediated by the buffer pool. You should pay careful attention to the interface that you design for the buffer pool, since you will be using this again in Project 4. In essence, the disk file will be the array, and all accesses to the array from the Quicksort algorithm will be in the form of requests to the buffer pool for specific bytes from the file. You should implement the buffer pool interface using the “message passing” approach shown at the beginning of Module 9.4.1.3. The buffer pool interface should pass byte arrays back and forth, not “records” as the Quicksort algorithm understands them.

Be aware that performance does play an issue in the grading for this program—some of the tests will be timed. If your program takes significantly longer than it should, then it will be penalized. So another design consideration will be the efficiency of your program, because some of the test cases are timed. If you are careless about how you implement the buffer pool, or in how much unnecessary byte copying you do within your Quicksort implementation, then you will not be able to meet the time requirements. We are not going to check exactly what blocks are swapped in and out of the buffer pool. Therefore, there is room for different approaches to tuning your Quicksort and buffer pool implementations as needed to meet the efficiency requirements.

#### Invocation and I/O Files:

The program will be invoked from the command-line as:

```
java Quicksort <data-file-name> <numb-buffers> <stat-file-name>
```

The data file `<data-file-name>` is the file to be sorted. The sorting takes place in that file, so this program does modify the input data file. Be careful to keep a copy of the original when you do your testing. The parameter `<numb-buffers>` determines the number of buffers allocated for the buffer pool. This value will be in the range 1–20. The parameter `<stat-file-name>` is the name of a file that your program will generate to store runtime statistics; see below for more information.

At the end of your program, the data file (on disk) should be in a sorted state. Do not forget to flush buffers from your bufferpool as necessary at the end, or they will not update the file correctly.

In addition to sorting the data file, you will generate and output some statistics about the execution of your program. Write these statistics to `<stat-file-name>`. Make sure your program DOES NOT overwrite `<stat-file-name>` 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 number of cache hits, or times your program found the data it needed in a buffer and did not have to go to the disk.
- (c) The number of disk reads, or times your program had to read a block of data from disk into a buffer.
- (d) The number of disk writes, or times your program had to write a block of data to disk from a buffer.
- (e) The time that your program took to execute the Quicksort. Put two calls to the standard Java timing method `System.currentTimeMillis()` in your program, one when you call the sort function, and one when you return from the sort function. 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 time to write the program output as described above.

### Programming Standards:

You must conform to good programming/documentation standards. Web-CAT will provide feedback on its evaluation of your coding style, and be used for style grading. Beyond meeting Web-CAT's checkstyle requirements, here are some additional requirements regarding programming standards.

- You should include a comment explaining the purpose of every variable or named constant you use in your program.
- You should use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc. Use a consistent convention for how identifier names appear, such as “camel casing”.
- Always use named constants or enumerated types instead of literal constants in the code.
- Source files should be under 600 lines.
- There should be a single class in each source file. You can make an exception for small inner classes (less than 100 lines including comments) if the total file length is less than 600 lines.

We can't help you with your code unless we can understand it. Therefore, you should not bring your code to the GTAs or the instructors for debugging help 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 provided by the instructor. Note that the OpenDSA 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.

### Java Data Structures Classes:

You are not permitted to use Java classes that implement complex data structures. This includes `ArrayList`, `HashMap`, `Vector`, or any other classes that implement lists, hash tables, or extensible arrays. (You may of course use the standard array operators.) You may use typical classes for string processing, byte array manipulation, parsing, etc.

If in doubt about which classes are permitted and which are not, you should ask. There will be penalties for using classes that are considered off limits.

### 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 unless you arrange otherwise with the GTA. There is no limit to the number of submissions that you may make.

You are required to submit your own test cases with your program, and part of your grade will be determined by how well your test cases test your program, as defined by Web-CAT's evaluation of code coverage. Of course, your program must pass your own test cases. Part of your grade will also be determined by test cases that are provided by the graders. 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.

When structuring the source files of your project, use a flat 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 to work with a partner on this project. While the partner need not be the same as who you worked with on any other projects this semester, you may only work with a single partner during the course of one project unless you get special permission from the course instructor. When you work with a partner, then **only one member of the pair** will make a submission. Be sure both names are included in the documentation. Whatever is the final submission from either of the pair members is what we will grade unless you arrange otherwise with the GTA.

### 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. The text of the pledge will also be posted online.

```
// On my honor:
//
// - I have not used source code obtained from another current or
//   former 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.