CS3114 (Fall 2011)
PROGRAMMING ASSIGNMENT #3

Due Thursday, November 3 @ 11:00 PM for 100 points
Due Wednesday, November 2 @ 11:00 PM for 10 point bonus
Initial Schedule due Tuesday, October 18 @ 11:00 PM
Second Schedule due Thursday, October 27 @ 11:00 PM

Assignment:

You will implement an external sorting algorithm for binary data. 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 the Heapsort. The modification comes in the interaction between the Heapsort algorithm and the file storing the data. The heap array 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 Section 8.3 in the book for more information about buffer pools.

Design Considerations:

The primary design concern for this project will be the interaction between the logical heap as viewed by the Heapsort algorithm, and the physical representation of the heap 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 heap array, and all accesses to the heap from the Heapsort algorithm will be in the form of requests to the buffer pool for specific blocks of the file.

Invocation and I/O Files:

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

```
heapsort <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 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 8 records to a line (showing both the key value and the data 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 deduction in points, since it will be assumed that your program did not work correctly.

2. You will generate and output some statistics about the execution of your program. Formatting does not matter so long as we can tell what each statistic is. 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 cache misses, or times your program did not find the data it needed in a buffer and had to go to the disk.

   (d) The number of disk reads, or times your program had to read a block of data from disk into a buffer.

   (e) The number of disk writes, or times your program had to write a block of data to disk from a buffer.

   (f) 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 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. Our most important rule of thumb is that the program must be easy to understand (which makes it easier for the TAs to grade). While we do not require a specific standard, here is a set of reasonable guidelines that are good to follow.

- You should always include a header comment, preceding main(), specifying things like the author(s), compiler and operating system used, and date completed.
- Your header comment should describe what your program does; don't just plagiarize language from this spec.
- You should include a comment explaining the purpose of every variable or named constant you use in your program. This is not just for the TAs – if you ever plan to look at the code again for a future project, you need this for your own sake.

- You should use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc. This is one of the most important ways to create readable code.
- Always use named constants or enumerated types instead of literal constants in the code. Doing so will save you from making many errors.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe in detail how it works unless you do something so sneaky it deserves special recognition.
- Proper indentation and blank lines 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 required pre-conditions that it assumes.

Do not expect the GTAs or instructors to help debug an implementation unless it is properly documented and exhibits good programming style. Its hard enough to debug good code. Be sure to begin your internal documentation right from the start, since that will save you time in the long run.

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.

Deliverables:

When structuring the source files of your project (be it in Eclipse as a "Managed Java Project," or in another environment), use a flat directory structure; that is, your source files will all be contained in the project root. Any subdirectories in the project will be ignored. If you used a makefile to compile your code, or otherwise did something that won't automatically compile in Eclipse, be sure to include any necessary files or instructions so that the TAs can compile it.

If submitting through Eclipse, the format of the submitted archive will be managed for you. If you choose not to develop in Eclipse, you will submit either a ZIP-compressed archive (compatible with Windows ZIP tools or the Unix `zip` command) or else a tar'ed and gzip'ed archive. Either way, your archive should contain all of the source code for the project, along with any files or instructions necessary to compile the code. 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.

You will submit your project through the automated Web-CAT server. Links to the Web-CAT client and instructions for those students who are not developing in Eclipse are posted at the class website. If you make multiple submissions, only your last submission will be evaluated.

You are permitted (and ecouraged) 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. 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.

Scheduling:

In addition to the project submission, you are also required to submit an initial project schedule, an intermediate schedule, and a final schedule with your project submission. The schedule sheet template for this project is posted at the course website. You won't receive direct credit for submitting the schedule as required, but each instance of failing to submit scheduling information as required will lose 10 points from the project grade.

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 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.