**CS3114 (Summer II 2017)**
**PROGRAMMING ASSIGNMENT #3**
**Due August 11 at 11:59 PM for 100 points**
**Due August 9 at 11:59 PM for 15 points bonus**

## Background

You will implement an external sorting algorithm for binary data. This project is a little different from previous projects in that you will be doing disk I/O on a random access 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).
<u>Warning: The data file is a binary file, not a text file.</u>
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 Module 9.4 in the OpenDSA textbook for more information about buffer pools.

## Design Considerations

The primary design concern for this project will be the interaction between the heap as viewed by the Heapsort algorithm, and the logical representation of the heap as implemented by the disk file mediated by the buffer pool. 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 buffer pool 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.

- 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.
- 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.
  - The name of the data file being s orted.
  - 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.
  - 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.
  - The number of disk reads, or times your program had to read a block of data from disk into a buffer.
  - The number of disk writes, or times your program had to write a block of data to disk from a buffer.
  - 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.

**Java Code**
For this project, you may only use standard Java classes, Java code that you have written yourself, and Java code supplied by the CS3114 instructor (see the class website for the distribution). You may not use other third-party Java code.

**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 check style 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.
- 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.

**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 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. 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 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.