CS 3310: Data and File Structures

Instructor: Dr. Ajay K. Gupta, Western Michigan University

Lab TA: Yu Guo

Matt Peter

**Software Life Cycle Report – Assignment 5**

**Phase 1: Specification**

Objectives:

1. Gain experience in manipulating binary files and using buffer pools
2. Compare the performance of the program with and without buffer pools

Write a program that:

1. Sorts a binary file that consists of many 4-byte records, each consisting of two 2-byte integer values, using a quicksort algorithm
2. Sorts using block sizes of:
   1. 4 bytes (1 record)
   2. 4096 bytes (1024 records)
3. Writes statistics about the execution of the program to a text file

**Phase 2: Design**

2.1 Modules and their Basic Structure

1. Module 1: Class Hw5Main contains:
   1. public static void main (String[] args) throws IOException – Calls methods from the UserInput class to decide what will be executed next
2. Module 2: Class UserInput contains:
   1. public char mainMenu () – Prints options to the user and reads selection
   2. public void generateFile () throws IOException – Collects name and number of records and generates binary file
   3. public void sortFile (FileWriter fw) throws IOException – Collects name and sorts binary file
   4. public void resizeBufferPool () – Changes number of buffers in buffer pool
3. Module 3: Class QuickSort contains:
   1. public QuickSort (RandomAccessFile f, int n, int s) throws IOExcpetion – Assigns values to attributes
   2. public void quickSortRecords (RandomAccessFile f, int first, int last) throws IOException – Quicksorts binary file by accessing only individual records
   3. public void quickSortBlocks (RandomAccessFile f, int first, int last, int level) throws IOException – Quicksorts binary file by accessing blocks of records
   4. private void getBlock (RandomAccessFile f, int blockNum) throws IOException – Reads data from binary file and stores it in the buffer pool
   5. public void getReadsWrites (FileWriter fw, String type) throws IOException – Writes statistics from sorting binary file to text file

2.2 Pseudocode for the Modules

2.2.1 Pseudocode for Hw5Main

1a. Hw5Main Pseudocode Refinement #1:

// Method main – Calls methods from the UserInput class to decide what will be executed next

2a. Hw5Main Pseudocode Refinement #2:

// Method main – Calls methods from the UserInput class to decide what will be executed next

// Declare and Instantiate Scanner and UserInput objects

// Declare and Instantiate File and FileWriter

// Enter loop that will display menu and continue taking commands until the user decides to quit

2.2.2 Pseudocode for UserInput

2a. UserInput Pseudocode Refinement #1:

// Method mainMenu – Prints options to the user and reads selection

// Method generateFile – Collects name and number of records and generates binary file

// Method sortFile – Collects name and sorts binary file

// Method resizeBufferPool – Changes number of buffers in buffer pool

2b. UserInput Pseudocode Refinement #2:

// Method mainMenu – Prints options to the user and reads selection

// Print out menu

// Get user input and check if it’s valid

// Return valid input

// Method generateFile – Collects name and number of records and generates binary file

// Get name of file and number of records to put in it from user

// Randomly generate number of specified values for file

// Method sortFile – Collects name and sorts binary file

// Get name of file from user

// Count the number of records in the file

// Create an array to store these records in their original order

// Read and store all of the records

// Create QuickSort objects

// Time the quicksorting by record method

// Reset the file

// Time the quicksorting by block method

// Print appropriate results to console and text file

// Method resizeBufferPool – Changes number of buffers in buffer pool

// Get a size for the buffer pool from the user

// If it’s not valid, repeat until a valid one is entered

2.2.3 Pseudocode for QuickSort

3a. QuickSort Pseudocode Refinement #1:

// Method QuickSort – Assigns values to attributes

// Method quickSortRecords – Quicksorts binary file by accessing only individual records

// Method quickSortBlocks – Quicksorts binary file by accessing blocks of records

// Method getBlock – Reads data from binary file and stores it in the buffer pool

// Method getReadsWrites – Writes statistics from sorting binary file to text file

3b. QuickSort Pseudocode Refinement #2:

// Method QuickSort – Assigns values to attributes

// Assign passed values to attributes

// Set up the buffer pool

// Method quickSortRecords – Quicksorts binary file by accessing only individual records

// Find the lower, upper, and pivot keys

// While the position of lower is less than upper…

// Find a lower key that is on the wrong side of the pivot

// Find an upper key that is on the wrong side of the pivot

// Swap the two keys and their values

// Recursively call quickSortRecords for the lower and upper partially sorted sections of the file

// Method quickSortBlocks – Quicksorts binary file by accessing blocks of records

// Find the lower, upper, and pivot keys

// While the position of lower is less than upper…

// Find a lower key that is on the wrong side of the pivot

// Find an upper key that is on the wrong side of the pivot

// Swap the two keys and their values

// Recursively call quickSortBlocks for the lower and upper partially sorted sections of the file

// When the file is done being sorted, write all blocks in the buffer pool back to the file

// Method getBlock – Reads data from binary file and stores it in the buffer pool

// Check if the buffer is already in the buffer pool

// If it is already in the buffer pool…

// Move that buffer to the top and move all the ones previously above it down one spot

// Increment the number of cache hits

// If it is not already in the buffer pool…

// Temporarily store the information for the buffer at the bottom of the buffer pool

// Place the new buffer at the top and move all the other buffers down one spot

// If the buffer at the bottom contained data…

// Write that data to the file

// Read data from the binary file into the new buffer

// Method getReadsWrites – Writes statistics from sorting binary file to text file

// Check what type of statistics is wanted

// Write the corresponding statistics to the text file

**Phase 3: Risk Analysis**

There are no known risks associated with this application.

**Phase 4: Verification**

All the steps of the algorithm were checked to ensure correct results in all circumstances. A variety of tests with a wide range of values were also done to check for correct output.

**Phase 5: Coding**

Program code can be found in the .java files and contains in-line comments as well as a Javadoc.

**Phase 6: Testing**

All program statistics are printed to hw5stat.txt

**Phase 7: Refining the Program**

All required features are included in the program so no refinements are needed.

**Phase 8: Production**

A zip file containing source files, a Javadoc, and test data have been submitted.

**Phase 9: Maintenance**

Any maintenance can be performed once feedback has been obtained.