2XB3 Assignment 2 – Implementing Sorting Algorithms

Department of Computing and Software McMaster University
Instructor: Dr. Reza Samavi

January 27, 2015

1. Assignment Description

Handwriting is back with the world's obsession of smart phones and mobile devices. To recognize what a user writes on smart devices, the problem of Handwriting Recognition (HWR) needs to be addressed. HWR is the ability of a computer to receive and interpret intelligible handwritten input from sources such as touch screens and paper documents. A Handwriting Recognition System relies on a recognizer (e.g., Artificial Neural Network) to identify the handwritten image from the source.

The output of the HWR system is a list of best-matching words with a ranking score associated with each word, the higher the score, the better the matching. However this list of words might not be arranged by their scores in descending order. Here, a sorting algorithm needs to be applied.

This assignment requires you to implement three sorting algorithms for the Handwriting Recognition System, and then explain in detail which one is the most preferable with respect to the running time. You are asked to compare the results for each implementation and for each size of the input array, and order the algorithms from best to worst. Your explanation is written in a text file named a2_out.txt.

2. Assignment Setup

To start with, you will need an input file of strings to sort. For this assignment you are asked to sort 3 arrays of size 2^4, 2^8 and 2^{12} elements respectively. You have to generate these arrays using the instructions below. This gives you a different input file every time you run it but you only need to do this once.

We use a *random number generator* to synthetically generate input data and simulate a real scenario for your algorithm.

1. Download the Eclipse project called 'cas2xb3_A2_lastname_initials' from the course website under the Assignments/A2 folder. Rename it using your last name and initials. Your entire implementation should be included in this project.

2. In this project, under the src folder you will see two packages, *gen* and *sort*. Also note the "data" folder created for you in the project. This is where your input file will appear after you have generated it.
3. Expand the Gen package under src and double-click the "Gen.java" file. Scroll to line 886 and replace the "[STUDENT#]" with your student number. If you are working on a Linux or OS X (Mac system) you might need to change the slashes in line 896 to suit your OS.
4. Run the file. Once it is done, right-click on the "data" folder and select "Refresh". You should now see a new file called "a2_in.txt".

This file contains three strings (separated by a newline character) that you are required to sort of size 2^4, 2^8 and 2^{12} elements. The file has the following format:

{chart,323},{chat,23},{cat,52},{create,3}, … ,{cater,652}

{baby,234},{bee,52},{dad,1},{body,65}, … ,{boil,123}

{date,43},{dad,623},{daily,3412},{delta,2346}, … ,{door,333}

Study this file carefully. Note that a word can actually be made up of some special characters like whitespace and hyphens. However there is no whitespace anywhere else, except a new-line character at the end of the array. You should store the word and the score together in an **abstract data type** called Word that implements the **Comparable** interface. Complete the implementation described in "Word.java" including the requirements for the Comparable interface. This is the ADT you will use throughout your implementation.

Now read the requirements and details that are given below carefully for each implementation. Then study the rest of the files given to you in the "*Sort*" package. They contain the skeleton of the code you are required to implement.

Before you start implementing the sorting algorithms, make sure you read Chapter 2 on sorting in your textbook. As a suggestion, you might also want to read the API for the **StringTokenizer** class, the **Comparable** interface as well as details on the **binary search algorithm**.

NOTE: *You cannot change the method signatures and file formats described in this document. Some of your test cases might take a few seconds to complete. You might need to implement more internal methods where necessary and also an abstract data type. Follow the output instructions exactly.*

3. Assignment Problems

3.1 Insertion Sort algorithm in Java

Insertion sort is a sorting algorithm that sorts one item at a time. Every repetition of Insertion Sort removes an element from the input data, and inserts it into the correct position in the already-sorted list, until no input elements remain. Refer to page 250 in your textbook for a step-by-step example.

In this assignment you are implementing three methods of Insertion Sort:

1. Regular insertion sort
2. Insertion sort using the Comparable interface (Refer to page *247* in your textbook)
3. Optimised insertion sort using binary search to find the insertion point

3.1.1 You are asked to use the Java class named "Insertion.java", where the three types of insertion sorts are to be implemented. Please use the signatures as "sortInsert, "sortComparable", and "sortBinary" respectively for each implementation. Your implementation should sort an array of ranking scores. When sorting words of the same score, the order of the words is not relevant.

Do not use the Comparable Interface method "compareTo" in your implementation of the simple Insertion sort algorithm. You might need to add a separate method to perform binary search for the third algorithm "sortBinary". This helps to modularise your code and should always be a design factor for you.

3.1.2 For testing your algorithm, this assignment requires you to create a JUnit class named "SortTest" and test methods named "testSortInsert", "testInsertComparable" and "testInsertBinary". You should test your sort algorithm by using three input arrays, which are located in the text file named a2_in.txt. You should test your sort algorithms thoroughly and include all necessary Assert statements. During testing**,** you also need to measure the running time of each of your sort algorithm implementations using the StopWatch library.

StopWatch provides a convenient API for timings. You can call start() to start a StopWatch before you invoke your insertion sort algorithm, and then call stop() when your sort method returns. The getTime() method gives you time measurement for the invocation. When using StopWatch you should consider careful placing of the start() and getTime() methods in your code such that you are measuring the correct elapsed execution time of the algorithm isolated from execution of your client code or test code.

For more information about StopWatch, please refer to https://commons.apache.org/proper/commons-lang/javadocs/api-

2.6/org/apache/commons/lang/time/StopWatch.html. Some information on how to import an external library into eclipse can be found here http://stackoverflow.com/a/2331818.

3.2 Merge Sort algorithm in Java

Merge sort is an algorithm which divides an unsorted list and combines sub-lists into a new list; For more details on Merge sort, see section 2.2 in your textbook.

3.2.1 You are asked to implement the merge sort algorithm. Your Java class should be named "Merge" and your method is named "sortMerge" using the Comparable interface. When sorting words of the same score, the order of the words is not relevant.

3.2.2 Test your merge sorts in the same JUnit class "SortTest" and name your test method "testMerge". Again, you should use the three arrays as input from a2_in.txt, sort them and measure the time taken via StopWatch. Remember to test your sort algorithms thoroughly.

3.3 Heap Sort algorithm in Java

3.3.1 You are asked to implement the Heap Sort algorithm in a Java class named "Heap" and a method named "sortHeap" using the Comparable interface. When sorting words of the same score, the order of the words is not relevant.

3.3.2 You are asked to test your heap sort in the JUnit class "SortTest" and name your test method "testHeapSort". Again, you should use the three arrays as input from a2_in.txt, sort them, and measure time via StopWatch. Remember to test your sort algorithms thoroughly.

4. Output Specifications

Your test code at the end should generate an output file called "a2_out.txt" and store it in the "data" folder. This file contains the results of running time comparison of the five algorithms you have just implemented. The format of this file should be as follows:

You should have three lists of 15 elements each, separated by newline characters. The first element is the size of the array (16, 256 and 4096 respectively). The consecutive elements are pairs of the names of the algorithms followed by the time taken by each algorithm. Each of the three lists should be in order of ascending time. This is what it should look like:

```
16,heap sort,0.1245s,insertion sort,0.31001s,…
```

256,comparable binary sort,3.333,heap sort, 3.4123,...

4096, binary insertion sort,4.3098s,merge sort,4.9.009,...

NOTE: The times and orders of the algorithms shown above is not realistic, it is just an example of what your text file should look like.

In your submission, you should include your entire source code, test case class as well as your generated input file, output file, and *2xb3_A2_lastname_initials.txt* file that you usually include in your submission (as detailed below under the **Assignment 2 Submission** section). To do this in Eclipse, export your entire project as a zip file and name it **cas2xb3_A2_lastname_initials.zip**.

NOTE*: You cannot change the method signatures and file formats described in this document. Some of your test cases might take a few seconds to complete. You might need to implement more internal methods where necessary and also an abstract data type. Follow the output instructions exactly.*

Assignment 2 Due Date

Friday, February 13th 2015, at 23:59.

Assignment 2 Marking

Assignment 2 is worth 5% of the course marks. Your grade for this assignment will be determined based on the following rules:

- A submitted solution that does not compile or run gets 0 credit.
- A solution that runs but is partially correct gets partial credit (depending on the progress towards a full solution).
- Providing adequate, concise, and meaningful comments throughout your code is part of the solution grade (i.e., a piece of code that correctly solves a problem without (or with inadequate) comments will score less than a well-commented piece of code that does the same).
- Your implementation should not only be correct but also concise and efficient. Quality aspects of your implementation and programming style particularly preservation of encapsulation and modular programming will be evaluated (refer to pages 96-108 of your textbook).
- Not following the assignment instructions properly for the requested formatting will cost you marks. You may even get 0 credit if the improper formatting will prevent your program from running.
- Every hour after an assignment deadline 2% will be deducted from the assignment mark. After 48 hours the assignment will no longer be accepted and the student will get 0 credit.
- This assignment is individual work. The work you submit must be your own. Both copying assignments and allowing others to copy your assignment are strictly forbidden

and will be treated as an academic offence. All assignments deemed to be substantially similar to each other will get 0 credit.

- If you include libraries from any sources other than your own or from the course material (course lecture notes and lab notes/instructions) you must acknowledge them and explicitly give proper credit with meaningful comments inside your code (when using methods from the external libraries). Properly cited external codes can only be included as Java libraries, i.e. you are not allowed to copy full or partial codes from other resources and include them inside your code. The included libraries should not be a substantial part of your assignment. Your work will be checked for plagiarism to account for this.

Assignment 2 Submission

You should include a txt file named 2xb3_A2_*lastname_initials*.txt resided in the “data” folder containing the following information (each item in a separate line):

- The course code (COMP SCI 2XB3 or SFWR ENG 2XB3)
- Your full name
- Your student number
- A dated statement that attests to "the fact that the work being submitted by you is your individual work."
- Any design decisions you feel need explanation or attention by the marker (extra methods etc.).

In order to submit your Eclipse project to the relevant lab assignment dropbox in Avenue, you first need to save everything and then export your project.

1. In Eclipse, right-click on the name of the project, select Export->General->Archive File.
2. Ensure that just your project has a check-mark beside it, and select a path and filename to export the project to. Ensure that your export project has a file extension of '.zip'. **IMPORTANT:** You MUST export the FULL Eclipse project. Submitting individual files (e.g. java/class files) will NOT be counted towards your submission. Click 'Finish' to export.
3. Verify the zip file by opening it and ensuring that it has the same folder structure as in Eclipse (it may have some extra files or folder such as 'bin', which is okay).
4. Go to Avenue and upload your zipped project to ' Assignment 2 Submission’ Dropbox.

VERY IMPORTANT: YOU CAN SUBMIT MULTIPLE TIMES, HOWEVER ONLY THE LAST SUBMITTED FILE WILL BE CONSIDERED FOR MARKING AND ANY PREVIOUS SUBMISSION WILL BE AUTOAMTICALLY REMOVED FROM THE COURSE WEBSITE - IT IS YOUR RESPONSIBILITY TO CHECK YOUR ZIP FILE BEFORE SUBMITTING