2XB3 Assignment 1 – Implementing Sorting Algorithms

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

Assignment due February 17, 2019, 11:59pm

1. Assignment Description

At the end of the year, a sales company would like to evaluate the sales amount of their different productions, which can be used to develop sales plan for the next year. Consider the company has *N* products, the total sales amount of a product *p* is p_a. The sales amount of a product reflects how customers like it. A sorting algorithm can be applied to help you order all the products with respect to their sales amount, which can help you understand customers' preferences.

This assignment requires you to implement an abstract data type (ADT) for products and four sorting algorithms for sales amount of various sizes, and then explain in detail which one is the most preferable with respect to the running time. The entire process must be done as a software engineering experiment with the proper use of analysis to compare the results for each implementation.

The analysis has a number of required steps:

- Recording your observations through the use of tables and plots
- Making hypotheses regarding the complexity of your algorithms
- Using your hypotheses to make a prediction
- Validating your hypotheses with a mathematical model

Upon completion of the analysis, you are then able to order the algorithms from best to worst. Your explanation must be written in a text file named a1_out.txt.

2. Assignment Setup

To start with, you will need an input file of products to sort. For this assignment, you are asked to sort 5 arrays of size 2^4, 2^6, 2^8, 2^{10} and 2^{12} elements respectively: each element representing a product with a unique ID and an integer sales amount. 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_A1_lastname_initials' from the course website under the Assignments/A1 folder. Rename it using your last name and initials. Your entire implementation should be included in this project.

NOTE: *You cannot change the method signatures and file formats in this project. You might need to implement more internal methods where necessary and also an abstract data type. Follow the instructions exactly. You are not allowed to use any Java built-in sorting methods.*

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 14 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 23 to suit your OS (change the directory from "\\data\\a1_in.txt" to "data/a1_in.txt").
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 "a1_in.txt".

This file contains five strings (separated by a newline character) that you are required to sort of size 2^4, 2^6, 2^8, 2^{10} and 2^{12} elements. The file has the format {e, e_a}, for example:

This file contains five lines where each line is a string representation of an array. The elements of the arrays are "products" with their "sales amount" represented as {e, e_a }. The first to fifth array contain 2^4, 2^6, 2^8, 2^{10} and 2^{12} elements, respectively.

{f90f934d-5168-4cfb-bb40-bdd493909b49, 64}, {bf9d9384-dd50-4680-9bcc-8d777c1be6ee, 85}

2. Creating Product ADT (20%)

You should store each product's ID (as String) and the sales amount (as int) together in an **abstract data type** called Product that implements the **Comparable** interface. Complete the implementation in "Product.java". In addition to including the requirements for the Comparable interface, your design should meet the following minimum requirements:

1. Construct a product
2. Retrieve the sales amount of a product
3. Set the sales amount of a product
4. Retrieve a product ID
5. String representation of an product
6. Ensure that the methods and the Product data type conserve encapsulation and immutability

This is the ADT you will use throughout your implementation for the rest of this assignment. You should submitted a pdf file called a1_API.pdf with the same format describe in the lecture documenting your ADT APIs. This file should be included in the data directory of your project.

The remaining files in the Sort package provide you a skeleton of the code that you are required to implement.

3. Implementing Sort Algorithms (35%)

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**.

3.1 Merge Sort algorithm in Java

Merge sort is an algorithm that 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.

In this assignment you are implementing two methods of Merge Sort: (1) Top-down merge sort, (2) Bottom-up merge sort. The merge sort algorithm should be in a Java class named "Merge", and you should use the method names "sortMergeTD", and "sortMergeBU" respectively for each implementation. When sorting products that have the same sales amount, products should be sorted lexicographically (e.g. alphabetically) by their ID.

3.2 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) and (3) Optimized insertion sort using binary search to find the insertion point.

Use the Java class named "Insertion.java" to implement the three types of insertion sorts. Use the method names "sortInsert", "sortComparable", and "sortBinary" respectively for each implementation. Your implementation should sort an array of products by ranking the sales amount. When sorting products have the same sales amount, products should be sorted lexicographically (e.g. alphabetically) by their ID.

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 modularize your code and should always be a design factor for you.

3.3 Quick Sort algorithm in Java

Quicksort is a divide-and-conquer method for sorting. It works by partitioning an array into two (or more) subarrays, then sorting the subarrays independently.

Please implement the Quick Sort algorithm in a Java class named "Quick" with two methods: (1) named "sortBasicQuick" which partitions the array into two subarrays, and (2) named "sortThreePartition" which uses use the median of a small sample of items taken from the subarray as the partitioning item (Refer to page *296*: Median-of-three partitioning, in your textbook). When sorting products have the same sales amount, products should be sorted lexicographically (e.g. alphabetically) by their ID.

3.4 Heap Sort algorithm in Java

Implement the Heap Sort algorithm (Refer to Section *2.4* in your textbook) in a Java class named "Heap" and a method named "sortHeap" using the Comparable interface. When sorting products have the same sales amount, products should be sorted lexicographically (e.g. alphabetically) by their ID.

4. Test your implementation using JUnit (12%)

For testing your algorithms, this assignment requires you to create a JUnit class named "SortTest" and multiple test methods named "testMergeTD", "testMergeBU", "testSortInsert", "testInsertComparable", "testInsertBinary", "testBasicQuick", "testThreePartition", and "testHeap". You should test all your sort algorithms by using the five input arrays that are located in the text file named a1_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.

5. Document your implementation using JavaDoc (8%)

You must generate appropriate Javadocs for all classes and methods that you have implemented as shown during the lecture (lecture note week 4). You should use as many tags as necessary to sufficiently document your code.

6. Experimental analysis of your implementations (25%)

You must perform proper analysis on experimental analysis of your implementations. The result of each step in this section should be recorded and finally submitted as a pdf file. This document should be called a1_analysis.pdf and should be included in the data directory of your project.

Create a table that has the following headers: the size of the dataset being sorted, the execution time of each type of Insertion Sort, the execution time of Merge Sort, and the execution time of Heap Sort. This should be included in your output file.

3.1.1 Plot the results in both a normal and a log-log scale with the problem size as the x-axis, and the running time on the y-axis (using graphs).
3.1.2 Formulate hypotheses based upon your observations to determine the running time (in Big O notation) of your implementation of the four types of sorting algorithms.
3.1.3 Use your hypotheses to predict the execution time of your implementation of the five sorting algorithms for an array size of 2^{14} and 2^{16}.
3.1.4 Verify your prediction by running your program on an array size of 2^{14} and 2^{16}.

Assignment 1 Submission

You should include a txt file named 2xb3_A1_*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 own 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 name it **cas2xb3_A1_lastname_initials.zip** to export the project. 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 1 Submission' Dropbox.

Assignment 1 Marking

Assignment 1 is worth 15% 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 (from the Internet or any other sources) 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.

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