CS2XB3 Assignment 2 – Implementing Sorting Algorithms

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

Assignment due: March 5th 2017, at 22:00.

1. Assignment Description

Scheduling is an important concept which makes it possible for a single CPU to multitask. The CPU scheduler can aim for one of many different goals such as, ensuring the highest number of jobs are being processed during a time period, minimizing the amount of time each job has to wait, or ensuring that each job gets an equal amount of processing time.

One common scheduling policy is **shortest job first**. If we have *N* jobs to complete at time *t*, each job *j* requiring j_t seconds of processing time, then the job with the lowest processing time would be scheduled first. This is where a sorting algorithm needs to be applied. For the purpose of this assignment, you can assume that the scheduler puts all available jobs that arrive at the same time into a queue based upon the shortest job first policy. Once jobs are placed into the queue, they are not modified anymore. For example, job A arrives at t = 1 and has a processing time of 3, job B arrives at t = 3 and has a processing time of 4, job C arrives at t = 1 and has a processing time of 2, and job D arrives at t = 4 and has a processing time of 1. The resulting job schedule would be C, A, B, D.

This assignment requires you to implement three sorting algorithms for a number of scheduling problems 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:

1. Recording your observations through the use of tables and plots
2. Making hypotheses regarding the complexity of your algorithms
3. Using your hypotheses to make a prediction
4. Verifying your prediction
5. Validating your hypotheses
6. Build a mathematical model for the running time of your algorithm (Tilde approximation)

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 a2_out.txt.

2. Assignment Setup

To start with, you will need an input file of jobs 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

job with a unique ID, an integer execution time (in seconds), and an arrival time. 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. ***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.*
3. 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.
4. 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 24 to suit your OS.
5. 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 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. Each element is of format $\{j, j_t, t\}$, for example:

{f90f934d-5168-4cfb-bb40-bdd493909b4,2,5},… ,{bf9d9384-dd50-4680-9bcc-8d777c1be6e,85,3}

2. Creating Job ADT (10%)

You should store each job's ID, execution time, and arrival time together in an **abstract data type** called Job that implements the **Comparable** interface. Complete the implementation described in "Job.java" including the requirements for the Comparable interface. This is the ADT you will use throughout your implementation. 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 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) Optimised 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 jobs by ranking the execution time. When sorting jobs of the same execution time, the order of the jobs 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.

3.2 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.

Your implementation of the mergesort algorithm should be in a Java class named "Merge" and your method is named "sortMerge" using the Comparable interface. When sorting that arrive at the same time and have the same processing time, the order of the jobs is not relevant.

3.3 Heap Sort algorithm in Java

Implement the Heap Sort algorithm in a Java class named "Heap" and a method named "sortHeap" using the Comparable interface. When sorting that arrive at the same time and have the same processing time, the order of the jobs is not relevant.

4. Test your implementation (15%)

For testing your algorithms, this assignment requires you to create a JUnit class named "SortTest" and multiple test methods named "testInsertSort", "testInsertComparable", "testInsertBinary", "testMergeSort", and "testHeapSort". You should test all your sort algorithms by using the five input arrays that 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.

5. Document your implementation (10%)

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 a2_analysis.pdf and should be included in the data directory of your Eclipse 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.

6.1 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.

6.2 Formulate hypotheses based upon your observations to determine the running time of your implementation of your algorithms (as described in P. 176 of your book) .

6.3 Use your hypotheses to predict the execution time of your implementation of the five sorting algorithms for an array size of 2^{14}.

6.4 Verify your prediction by running your program on an array size of $2^{14.}$

6.5 Validate if your prediction and the result converge by running your program for array size of 2^{16} and 2^{18}

Hint: You need to modify the given Gen.java class to generate additional arrays.

6.6 Report the mathematical models (Tilde approximation) of the five implementations with a short description on justification of each model.

7 . **Comments and Formatting** (5%)

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 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_A2_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 2 Submission' Dropbox.

Assignment 2 Marking

Assignment 2 is worth 10% 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.