CptS 223 PA #1 - Benchmarking a LinkedList

In this first Programming Assignment you're being tasked with benchmarking the performance of a linked list. We'll be using the Java library LinkedList class so you don't need to build the list object yourself, though you can construct your own linked list implementation if you like.

The assignment has two parts:

1) Coding your solution to generate statistics
2) Writing a short report about your environment, the benchmark results, and other details

The final material for submission should be entirely written by you. If you decide to consult with others or refer materials online, you MUST give due credits to these sources (people, books, webpages, etc.) by listing them in the report that accompanies your submission. Note that no points will be deducted for referencing these sources. However, your discussion/consultation should be limited to the initial design level (if any). Sharing or even showing your source code/assignment solution verbiage in totality to anyone else in the class, or direct reproduction of source code/verbiage from online resources, will all be considered plagiarism, and will therefore be awarded ZERO points and subject to the WSU Academic Dishonesty policy. (Reproducing from the Weiss textbook is an exception to this rule, and such reproduction is encouraged wherever possible.)

Project Concept

The main purpose of this very first programming assignment is to become familiar with the programming and submission environments new in this course. To this end, this assignment has only small emphasis on algorithm design choices, though it will show the process of how you can benchmark programs instead of running algorithm analysis. You will be reporting on your observations about the experiments that will be carried out in this assignment.

Project Description

You are asked to implement the code that:

- Reads an input file containing integer values;
- Inserts the values into a sorted singly linked list. You can either create a linked list from scratch by yourselves or use the Java standard library LinkedList class libraries. Both approaches are acceptable.
- Reports minimum (referred to as 'min'), maximum ('max'), and median ('med') of the list
- Reports the time that it takes to:
    - insert all the values into the list ('time_insert')

- find minimum of the list after all values are inserted into the list (‘time_min’)
- find maximum of the list after all the numbers are inserted into the list (‘time_max’)
- find median of the list after all the numbers are inserted into the list (‘time_med’)

Your program should get through filename through one of two places:

1) The command line: java Benchmarking myFilename.txt
2) By prompting the user to enter a filename if no filename is provided on command line

It should open the file, read in the values to store in sorted order into the list, and should print out the output numbers (min, max, med, time_insert, time_min, time_max, time_med) with appropriate messages.

Note that the linked list should contain ‘sorted’ values at any point in time. Thus, you cannot insert all the values into the list first and then try to sort the numbers. Instead, the list should remain a sorted one after every insertion.

The timing statistics should be in seconds or milliseconds or microseconds (whichever gives the closest precision to measure the actual time of the event). Now, it may so happen that if a particular timed event's time is less than a second, your timer function will show 0 seconds. Obviously this does not mean 0.0 seconds. It just means you need to measure the time at a lower/finer resolution and use milliseconds or microseconds. If it turns out that the timed event is smaller than a microsecond, then you can consider that time to mean really "0" seconds - i.e., nothing to add to your timer variable.

Starting materials include this description, and a directory for your project to live with the test files of numbers to process. I have also included the “main” java file as Benchmarking.java. It’s empty now, but please keep the file name so graders can easily find where to build and run your code from.

The Report

In a separate written document (PDF format for submission), compile and include sections A through D as follows:

A: *Problem statement*. In one or two sentences, summarize the goal of the problem.

B: *Algorithm design*. This project does not have a lot of algorithm design decisions. However, there are few areas that you will need to make a decision about how to compute the output values such as ‘min’, ‘max’, and ‘med’. Within one page, provide a brief description of the main ideas behind your algorithms for finding ‘min’, ‘max’, and ‘med’ within the list. Discuss any potential alternative approaches that could be used to find these values which could have resulted in a poor timing performance.

C: *Experimental setup*. In this section, you should provide a description of your experiment setup, which includes but is not limited to:

- Machine specification - CPU maker, CPU speed, RAM available, hard drive type
- How many times did you repeat each experiment before reporting the final timing statistics?
- O/S and environment used during testing: Windows or Unix? Also mention which compiler environment (e.g., javac version). This information will help the TAs determine where to run your programs during grading.

D: *Experimental Results & Discussion*. In this section, you should report the performance (running time) and outputs based on your observations, and provide justification for your observations. For your testing and reporting, conduct the following two experiments using the two provided input files, namely 'input1.txt' and 'input2.txt'.

Submission

Once you've finished your code in the project src directory and written your report (and saved it in PDF format), put the report in the root of the project tree (under PA1-Benchmarking) and upload them into your git repo.

Grading

Assume that the whole assignment is worth 100 points:

Coding (50 pts):

- (15 pts): Is the code implemented in an efficient way? i.e., are there parts in the code that appear redundant, or implemented in ways that can be easily improved? Does the code conform to good coding practices of Object Oriented programming?
- (15 pts): Does the code compile and run successfully on a couple of test cases?
- (20 pts): Is the code documented well and generally easy to read (with helpful comments and pointers)?

Report (50 pts):

- (20 pts): Is the algorithm (for searching min, max, and med) designed efficiently?
- (10 pts): Experimental setup specified.
- Experimental Results & Discussion. In this section, you should report the performance (running time) and outputs based on your observations, and provide justification for your observations. For your testing and reporting, conduct the following two experiments using the two provided input files, namely 'input1.txt' and 'input2.txt'.

Obviously to come up with the above evaluation, graders are going to both read and run your code.

Java code / tech suggestions:

To get the number of nanoseconds since the UNIX epoch (when all UNIX clocks start counting from, or 00:00:00 UTC on 1 January 1970), you can use the interface:

```
System.nanoTime();
```

That gives you a way to time how long its been between two points in your program

To receive the filename on the command line, you should use the String args[] array in main:

```
public static void main(String args[])
```

I should be able to compile and run your java program by executing this on the command line:

```
javac Benchmarking.java
java Benchmarking input1.txt
```