Stephen Emmons
CptS 233
Prof. Guizani
9/18/2020

A - Problem Statement

The idea of this project is to utilize a linked list and insert the values from a given document into the list and sort it as the values are added, then recording the minimum, maximum, and median values within the list while giving time benchmarks for each of those tasks.

B - Algorithm Design

My most significant design choice in regards to sorting/inserting values is my use of the Collections.sort interface which has an O(n log n) complexity while sorting a linked list. This means the actual code used to sort the list is only a single line long and has a superior time complexity compared to simply iterating through the list with nested for loops and comparing each integer added to every other integer in the list, which would have an $O(n^2)$ complexity. As for benchmarking, I made sure to record the time each task started on the line directly before the first line of code for that task is executed and the time each task ended on the line directly after so as to avoid unrelated commands such as the creation of a buffered reader or print statements factoring into the time elapsed. I also used System.nanoTime() for the linked list creation instead of System.currentTimeMillis() even though the time elapsed was in milliseconds to make sure the level of precision was high.

To actually obtain the minimum, maximum, and median I simply used the linked list interface to utilize the methods get(), getFirst() and getLast() with the median being found by using the command get(list.size()/2) to find the middle index in the list. To ensure the median is accurate I added a check to see if the list has an even size, and if so the median will be the two middle indexes added and then divided by two. I could have alternatively iterated through the list and compared values to find the min, max, and median, but this likely would have taken longer especially with a dataset much larger than the one I was using, so I stuck with the methods available to me in the interface.

C - Experimental Setup

Machine:

- CPU: AMD Athlon x4 860k quad core
- RAM: 16 GB
- Storage: 1TB Seagate HDD
- Windows 10

Experiment repetitions: 3
Compiler: VS Code Native Java Compiler

D - Experimental Results

input1.txt:

- insert/sort time: 31 ms
- get minimum time: 45 μs
- get maximum time: 12 μs
- get median time: 54 μs

input2.txt: Could not use, file too large

Discussion:

Above are the best times I can achieve with the knowledge and equipment I have on hand. I think there may be more time to be found if I were to find a way to insert the values in a sorted fashion rather than running the Collections.sort command after every insertion but the approach I took seems to do the job in a reasonable amount of time so I didn't feel that extra optimization was necessary for this particular assignment. Everything lines up with about what I expected; I knew the creation of the final list would take much longer than finding any particular value in it just because of the amount of sorting being done, and I knew the median would take longer to find than the minimum or maximum because of the extra arithmetic I added to get the best level of accuracy for that value. I am a little confused why the minimum value takes longer to find than the maximum value considering the min is the first value in the list, but it's difficult to make a guess without knowing how the methods I used to find them are coded and considering the difference is a couple dozen microseconds I don't think it's too significant.