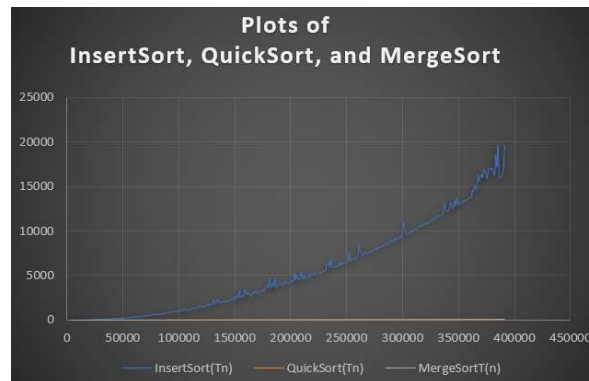
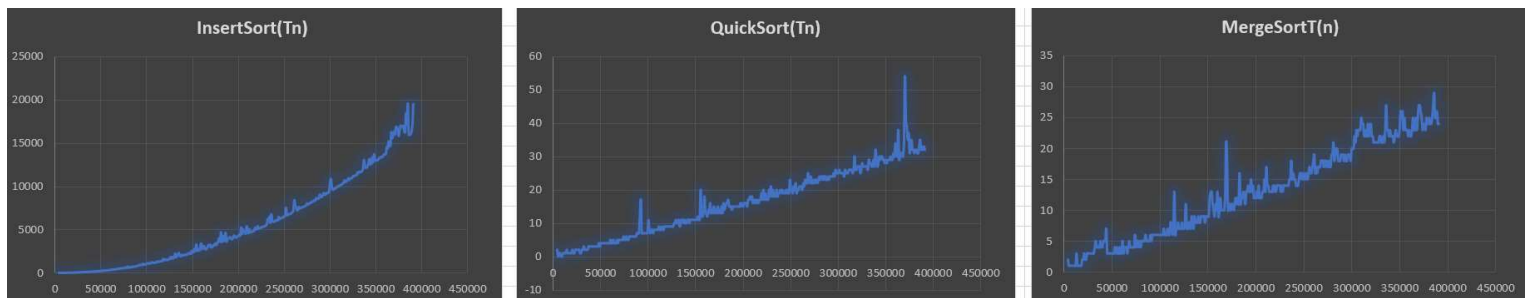


Submitted by: Wilbur Rtoni and Genji Nakano
CPSC-3273-A01
Module 5 Programming Assignment Report

Note: The assignment requires that the $T(n)$ of *InsertSort*, *QuickSort*, and *MergeSort* be on the same graph. Following this instruction will result in the following:



Notice that due to the very wide differences in the values of *InsertSort* versus *QuickSort* and *MergeSort*, they can't be all seen on the same graph clearly. Therefore, separate plots were generated for each sorting technique as seen below. This will allow us to visualize each value more clearly:





Notice each plot's Y-axis values (i.e. $T(n)$). At $n \sim 400,000$, *InsertSort* has reached $\sim 25,000$ ms while *QuickSort* and *MergeSort* are only 60ms and 35ms, respectively. (Note that the sample size limit was set to 400,000 because the PC was taking a very long time (i.e. many hours) to finish the execution).

These plots, including the wide differences in the $T(n)$ values of *InsertSort* versus *QuickSort* and *MergeSort*, are expected because: As we have learned in Modules 4 and 5, *InsertSort* has an average time complexity of $\theta(n^2)$ while *QuickSort* and *MergeSort* both have an average time complexity of $\theta(n \log(n))$. Thus, as we can see from the plots of the real-time (T_n) above, the plot of *InsertSort* resembles a polynomial (i.e. quadratic) graph, most especially, as the sample size, n , becomes larger. This is due to the fact that, and as we have learned in Module 5, the *InsertSort* algorithm requires many element swaps to shift all of the proceeding elements when inserting the $(k+1)$ -st element into the sorted portion of the array (so imagine doing this to an unsorted array of size 1,00,000 or more!!!). On the other hand, *QuickSort* and *MergeSort* algorithms both use the divide-and-conquer approach in which the very large sample size is recursively broken down into sub-problems until they become simple enough to be solved, then they are combined as the final solution to the original problem; hence, regardless of the sample size, their graphs, **on average**, resemble logarithmic graphs.

We can notice, however, (graphs not shown in this report due to page limitation) that: When the sample size is small enough (e.g. $< \sim 50$), *InsertionSort* works faster ($T_n = n$). Therefore, and in conclusion, *InsertionSort* works well when we are dealing with just a very small sample size. On the other hand, *QuickSort* and *MergeSort* are more practical to use when we're dealing with medium-to-large sample size.

In addition:

- Yes, the program works.

- The program is written in Java using Auburn University's jGrasp IDE. Therefore, simply open the ***M5Project.java*** file in jGrasp -> then Compile  -> and then Run .

The file F is a *.txt file but has been formatted like a *.csv file. This way, it can be conveniently opened (and plotted) in excel.

Notes:

1. In **line # 20** of the code (***M5Project.java***), the user can change the value of L to any desired HUGE value. Just note that depending on the user's computer speed, it may take a while (i.e. many hours) for the program to finish generating the file F.
2. In **line # 38**, the user should change the location where the log file (F) should be saved. Currently, it is set to the programmer's local disk path/folder. **It is very strongly recommended that the user selects the same folder where he/she saved the source code *M5Project.java* file.**
3. In **line # 312**, the user should make sure that **the file name in line # 38 and line # 312 should exactly match.**