Pod 9

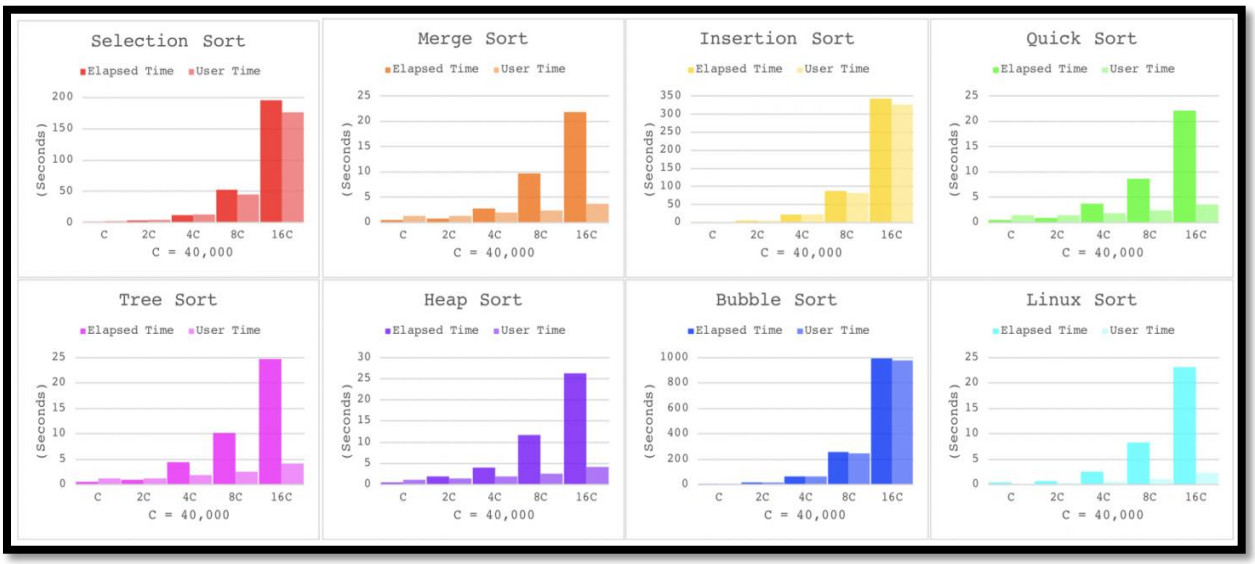
Dr. Binkley

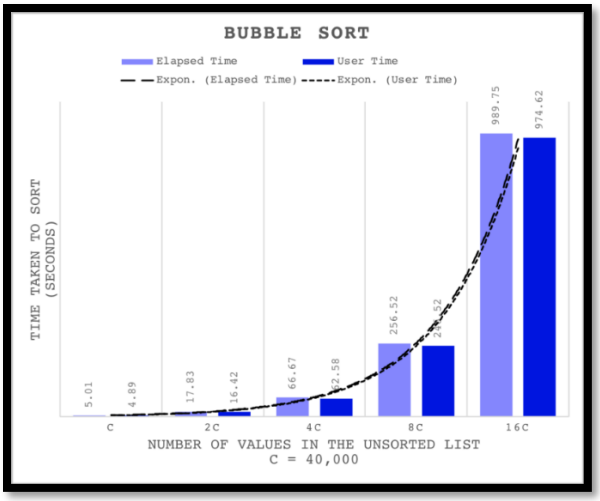
CS 312 – Assignment 7

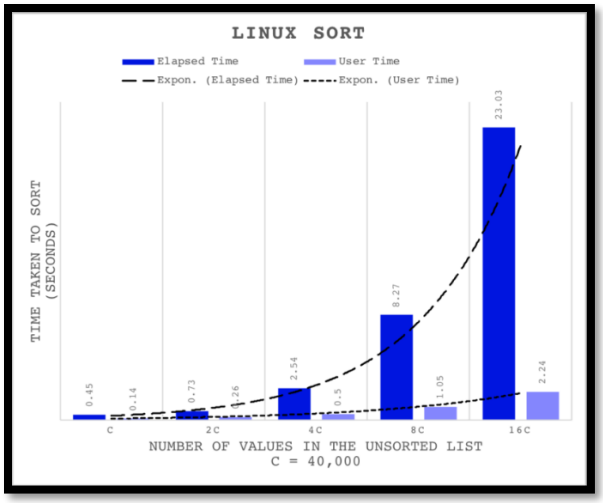
13 November 2020

Empirical Analysis

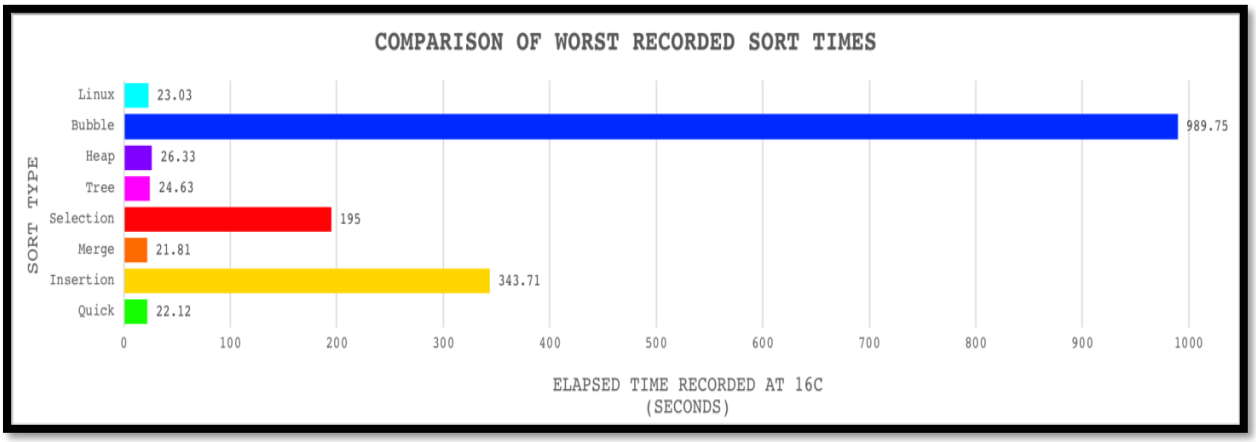
In this assignment, our goal was to empirically study the many sorting algorithms. This included a selection, merge, insertion, quick, tree, heap, and bubble sort. The idea was to see which sorting method was consistently the fastest. We tracked the data of this study on a table that matched each sort with their time taken, given a certain *C*. This *C* is the input amount of values and the base case was established at the maximum *C* below 5 seconds; then the sorts were tested at 2C, 4C, 8C, and 16C. Our *C* value was 40,000 and the random seed we ran the study on was consistently 42. This allowed for our random values to be randomly selected within our unique seed.







There were two main trends that were observed during the study. The first is seen in the Bubble, Insertion, and Selection sort graphs. In these sorts, the user and elapsed times follow similar, steep exponential curves. This indicates that the elapsed time (real time) was in close proximity to the user time (time charged to the CPU). In contrast, the remaining sorts shared a similar steep curve in their elapsed times however, the user times showcased a low, gradual slope. This indicates that the Linux sort and the other sorts that followed the second trend are more efficient than those that followed the first. If the amount of time taken to run a sort is relative, then a more sophisticated sort uses a significantly smaller portion of that time to compute a sorted list rather than one of the more basic sorts.



In regard to challenges (bugs, time taken to code, code length, etc.), the quick, heap, and tree sorts had the three longest code lengths but, logged some of the fastest times. This indicates that code length does not have a direct or consistent relationship with run times. Interestingly, Bubble sort's code is remarkably short but had the longest run time by a large margin.