Ethan Clunie
Professor Rao Ali
Data Structures: CPSC-350-03
Monday, May 9, 2022

Assignment 8 Report

Getting the sorting algorithms working was actually pretty simple for this assignment as they were entirely on zyBooks, so I went there and reviewed the algorithms themselves in order to better understand what they are meant to do. I looked at the last section of the sorting algorithms on zyBooks where it touched on the speed of these algorithms, so I was not so surprised by the sheer differences in execution time. However, I was caught off guard when testing this program with a file of 100,000 randomly creating doubles between 0 and 10,000. Before running the algorithms, I noted their upper bound for time complexity and saw that the Merge, Selection, and Quick sorts were in: O(NlogN), O(N^2), and O(NlogN) respectively. When performing the test, I saw just how long it took for the selection sorting algorithm to run. As I am writing this report, I had a test going in the background and the times for the merge, selection, and quick sorts are respectively: 22580 microseconds, 9612928 microseconds, and 13075 microseconds. This just goes to show how, with large data sets, the merge and quick sorts are significantly more time efficient than the selection sort. Thus, the tradeoffs involved in selecting different algorithms comes down to time and space complexity. If you are choosing between two that take a similar amount of time with large data sets, you may select the algorithm with better space complexity. Similarly, you would choose the more time efficient algorithm if two algorithms took similar amounts of space. In this case, I would obviously use the quick sort as it had the smallest execution duration amongst the algorithms I tested. Using the C++ programming language allows us to directly create pointer objects which help to speed up algorithms by passing around pointers to the stored data rather than entire data sets themselves. Languages without this capability would most likely require the programmer to pass around the entire data set they would want sorted which would take significantly more time and space in execution. The shortcomings of this analysis are that I am only testing a few algorithms on what would probably be considered a small data set, so I don’t have an entirely accurate view on how algorithms perform with greater n elements. Furthermore, printing out the times just before and after the function calls to these algorithms is a relatively crude method of determining the runtime of the algorithms.