Anthony Matter
5/10/2022

Assignment 8 Report

The three sorting algorithms that I implemented into my program were the Merge Sort, Selection Sort, and the Insertion Sort. After I was done implementing all algorithms, I was convinced that the Merge Sort would take the most time to execute because it had much more involved implementation than the other two. When I initially tested my code with a file that had 10 elements, I felt that my theory was justified because the runtime for Merge Sort was higher than the other two. However, that completely changed when I started running my code against files with 1,000 to 100,000 elements. The runtime had drastically shifted to a degree that I did not expect because the runtime for the Merge Sort was exponentially lower than the runtimes for Selection Sort and Insertion Sort. Once I saw this I knew then when you are picking each of these algorithms, there will be tradeoffs. If you want to sort a list with a small amount of values, it is more efficient to utilize Insertion Sort and Selection Sort because they have a space complexity of O(1) and they don't have to run through many values. It is less efficient to use Merge Sort in this scenario because it has a space complexity O(n) and even if the array is sorted, it still goes through the entire process. While I do think that the choice of using C++ made the runtime results for each sorting algorithm as big as they were, I believe that I could've used any programming language and against 1,000 elements, Merge Sort still would have been the most efficient. There would have been some small variations when running the program against smaller lists of values because in the case of Python, it's a simpler language and so you achieve the same results with less code. This might've made the results between the three sorting algorithms a little bit closer than they would've been in C++, but in the long run, Merge Sort would still be the most efficient with larger lists of values. However, making claims off of simply the results of the program can have its shortcomings. I think one of the biggest shortcomings is that it limits our ability to project how big a certain runtime will be in comparison to another algorithm. We'd place more emphasis on the algorithm to determine how the runtime for one algorithm will compare to others in every scenario and that in itself could take a lot of time. The other shortcoming is what happens if out of nowhere we compare Merge Sort and Selection Sort and for the longest time Merge Sort has the better runtime, but let's say we make one comparison and suddenly Selection Sort has the better runtime. If we solely rely on the evidence presented without knowing what is happening behind the scenes, how are we supposed to know what could have happened that sparked the Selection Sort to become the better runtime. I believe that evidence should not be what you rely on to interpret the efficiency of certain algorithms, but it should simply be used to ensure that your understanding of the algorithms being tested is either correct or incorrect.