Eshaan Vora
2318955
Evora@chapman.edu
CPSC 350-02
Assignment 6 Write-Up

To compare the run time performance of the 5 sorting algorithms (Quick sort, Merge sort, Selection sort, Insertion Sort and Bubble Sort) I measured each algorithm's total run time in milliseconds when running cases of 100, 1,000, 10,000, and 160,000 element lists. When working with a small list (100 elements), each of the sorting algorithms have comparable performances with quick sort, merge sort and insertion sort being the fastest. However, as we increased the number of elements, it was clear that certain algorithms were much more efficient.

I expected merge sort to be slightly more efficient than quick sort in the case of 160,000 elements, but quick sort seemed to run faster in all of my cases. Both would be good choices to use for larger datasets where merge sort would require additional memory space to store the auxiliary array. On paper, both have a Big-O runtime of O(N log N).

Insertion sort was by far the most efficient across the different cases and did not increase much in run time (from .217 milliseconds to .478 milliseconds) even as we increased the number of elements, N, from 10,000 to 160,000 whereas selection sort and bubble sort's run time increased dramatically (from 249.04 milliseconds to 67,228 milliseconds and from 395.401 milliseconds to 96624 milliseconds, respectively).

Although bubble sort may be the easiest to implement, I would only use it for sorting small lists because of how drastically run-time was affected when more elements were introduced. Bubble sort is usually the slowest because it has a runtime of O(N^2) in every case, whereas insertion sort and selection sort could perform more efficiently in certain cases but have an average-case runtime of O(N^2) as well.

Implementing sort algorithms in C++ as is beneficial because we can manage memory allocation better by choosing the most efficient algorithm for our specific data.

There could be inconsistent performance among cases due to the different resource constraints in the user's hardware. Another shortcoming of this analysis is having to implement each algorithm to test performance as opposed to calculating the efficiency mathematically.