Introduction:

For this lab, I implemented and examined the following sorting algorithms for analysis of runtime complexity: Selection Sort, Merge Sort, and Quick Sort. In terms of implementation, Selection Sort finds the smallest element using a linear scan and moves it to the front (swapping it with the front element). Then, it finds the second smallest element and moves it, again doing a linear scan. Selection Sort continues the aforementioned routine until all the elements are in place. Selection Sort’s worst case complexity turns out to be O(n2), simple but inefficient. The second algorithm, Merge Sort, divides the array in half, sorts each of those halves (in which Merge Sort is applied recursively), and then merges them back together. Merge Sort has a O(n Log(n)) average and worst case runtime. In Quick Sort, we pick a random element and partition the array, such that all numbers that are less than the partitioning element come before all elements that are greater than it. The partitioning can be performed efficiently through a series of swaps. Since the partitioned element is not guaranteed to be the median, our sorting could be very slow which lends Quick Sort’s worst case runtime of O(n2). However, it does employ a O(n log(n)) average case runtime. Understanding the common sorting algorithms is incredibly valuable, as many of the sorting and searching problems are tweaks of the well-known algorithms.