Liz Lyon
CPSC 350-03

Sorting Report

Overall, my experience using the 6 different sorting algorithms was rather straightforward and informational. This assignment made me realize how valuable time is from a performance perspective mainly because I was able to experience the resulting durations of a quadratic vs log linear algorithm. My findings include that a bigger input size correlates to a bigger time difference. For example, sorting 300,000 doubles on an O(n^2) algorithm such as bubble sort took 513.2 seconds whereas sorting on an O(NlogN) algorithm such as merge sort took 0.06 seconds. I honestly didn't know if bubble sort was going to be able to complete sorting given the amount of time it took, but to my surprise it indeed sorted eventually.

There are clear trade-offs when invoking an O(n^2) versus an O(NlogN) algorithm. Runtime performance for O(NlogN) is significantly faster in comparison to O(n^2). However, I think every sorting algorithm has its own tradeoff. For example implementing **bubble** sort is fairly naive and manageable for an average programmer, but the tradeoff of bubble sort is its Big O runtime which is on the order of O(n^2) meaning that it is slow, especially on large inputs because it requires O(n^2) swaps and comparisons. An advantage of **insertion** sort is its in-place nature and that no swaps are necessary rather it copies the data over. In the best case, if the input data is sorted then the runtime is O(n). Insertion's sort tradeoff is its worst case runtime of O(n^2) when the data is unordered. Concluding that for optimal use insertion sort is best for small input that is pre ordered. An advantage of **selection sort, other than being an in-place algorithm,** is the number of swaps is O(n), but the tradeoff is in the number of comparisons which is O(n^2); however it is better than bubble sort! An advantage of **merge sort** is its ability to quickly sort large amounts of data in O(NlogN) due to the divide and conquer method it utilizes, but the tradeoff is the requirement for extra space given that it is an out-of-place sorting algorithm. An advantage of **quicksort** is that it is an in-place sorting algorithm that requires no additional space and is extremely fast at sorting random input given any size because it is on the order of O(NlogN) by using the divide and conquer method. A tradeoff of quick sort is that its worst case runtime is no better, performance wise, than insertion or selection sort if the data is presorted which regresses its runtime to O(n^2). An advantage of using **Radix** sort is its ability to sort integers in O(n) time which is by far the most efficient, however a tradeoff is that it is far less flexible and only sorts integers.

Using C++ affected the results because it is optimizable for recursive calls and passing arrays as pointers. In terms of empirical analysis, a shortcoming includes its ambiguous interpretation. In addition, empirical analysis is not time efficient. In order to deem the individual algorithms as being "fast" or "slow" it required multiple testing attempts on a range of small to large input sizes which, in itself, was expensive.