Assignment 6: Sorting Algorithms

Elise May
Student ID: 2271041
Student Email: may137@mail.chapman.edu
CPSC 350-02

I. Introduction

This project compares the space and time efficiencies of different sorting algorithms to gain a better understanding of best use cases amongst them. This analysis is driven by the multitude of factors that can affect performance. The algorithms being analyzed are as follows: Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, and Merge Sort.

II. Results

The above algorithms were each tested with lists of size 10, 100, 1,000, 10,000, 50,000, and 100,000. The differences in performance across the algorithms are accentuated as the size of input increases. The results below reflect the findings from testing all sizes, but are mostly focused on sorting a list of size 100,000 to regularize the results.

A. Bubble Sort

Average run-time:

$$O(N^2) \tag{1}$$

Bubble Sort performed sufficiently well with very small input sizes, however this algorithm breaks down quickly starting at inputs as little as 1000. For an input size of 100, Bubble Sort clocked in at 0.065ms but jumped up to 10.886ms for 1000. Considering its extremely simple implementation, Bubble Sort will suffice for such small datasets, however, it should not be used for larger inputs.

B. Selection Sort

Average run-time:

$$O(N^2) \tag{2}$$

Selection Sort performed nearly the same as Bubble Sort and did not differentiate itself until an input size of 100,000 was tested. Because of its near identical performance but more efficient process, it should be chosen over Bubble Sort for small inputs but should also not be used for large sets of data.

C. Insertion Sort

Average run-time:

$$O(N^2) \tag{3}$$

Insertion Sort was much faster than Bubble and Selection Sort at 31432.7ms for an input size of 100,000. Though it is much faster than the latter, it is still much slower than Merge or Quick Sort for large datasets. It will, however, outperform all other methods in the use case of partially sorted data.

D. Quick Sort

Average run-time:

$$O(NlogN) \tag{4}$$

Quick Sort had by far the quickest run-time of all the algorithms at every input size. Quick Sort is faster and more memory efficient than Merge Sort, making it the best sorting algorithm, especially for large inputs. For 100,000 values, Quick Sort only took 32.788ms.

E. Merge Sort

Average run-time:

$$O(NlogN) \tag{5}$$

Though Merge Sort performed exponentially better than Bubble, Selection, and Insertion, it still could not beat out Quick Sort. It is also less memory efficent than Quick Sort, therefore it is a viable option for sorting but not the algorithm of choice.

III. Language Choice

Choice of programming language can mostly affect sorting algorithms in terms of complexity. C++ works very well for testing algorithm performance as it is a compiled language and is very portable. Interpreted languages may subject our results to other lurking variables.

IV. Shortcomings

Some difficulties facing this empirical analysis are inconsistency of results given the randomness of the generated input lists. Partially sorted lists, generated by chance, can affect the performance of these algorithms relative to each other.

V. Conclusion

Overall, run-time is dependent on size of input as expected, however, different sorting algorithms still prove best in different use cases. If handling large amounts of data, one should leverage Quick Sort for the fastest run-time. Its divide and conquer recursive mechanism and lack of need for usage of auxiliary data structures allow it to iterate quickly and efficiently through data at high speeds. For very small sets, it may be more efficient to use Selection Sort due to its extremely simple implementation. In a case where one is dealing with partially sorted data, Insertion Sort would be the best performing method as it eliminates iterations and shifts its average run-time from quadratic to linear. Use case must be heavily taken into consideration when choosing a sorting algorithm.