An Analysis of Sorts

Eric Njuku
CoSc 320, Data Structures
Pepperdine University

November 21, 2021.

Abstract

This sort paper assesses the performance metrics of five sorting algorithms: Insert Sort, Select Sort, Heap Sort, Merge Sort, and Quick Sort provided from the `dp4dsDistribution` software described in Design Patterns for Data Structures chapter 4 . The utilization of `RStudio` and `CLion` provided sufficient data to prove the theory that any sort of n elements that is based on element comparison and exchange is Ω(nlgn). This paper concludes with statements regarding the most effective sort algorithm and how the theory is proven.It is important to note that these metrics are only approximations.

1 Introduction

A sorting algorithm is used to rearrange a given array using a comparison operator in the most efficient way possible. This paper will show how and why some sorts are more efficient than others. For this paper, these sorts will be evaluated with raw data gathered from comparison and assignment measuring systems. Section 2 looks at the Methods and describes the performance metric issues. Section 3 reviews the results for each sorting algorithm and Section 4 is the conclusion.

2 Method

This section reviews how the sorting algorithms work to sort a list of numbers as well as how the data is collected and applied in the results section and explain the analysis methods.

2.1 Sort Algorithms

This paper analyzes five different sorts: Merge Sort, Quick Sort, Insertion Sort, Selection Sort, and Heap Sort.Below is a list that provides details on how each sort compares and organizes a list of numbers:

- Merge Sort: The merge sort works by performing a simple half split on the list.The sort takes L1 as the first half of the list and L2 as the second half of the list. The sort then recursively sorts L1 and L2 respectively and merges the 2 sub lists. It is easy to code the split of the list but harder to code the join which requires a loop to go through both sub lists L1 and L2 and select the smallest number to place at the right place in the merged list in order from the smallest to the largest. The theoretical asymptotic bound for the merge sort is $\Theta(n \lg n)$

- Quick sort: The quick sort works by splitting the original list into sub lists L1 and L2. The sort performs the split by selecting a ”key” value which is the median value of the list and placing the elements that are less than or equal to the key into sub list L1 and placing the values that are larger than the key into sub list L2. It is harder to code the split than the join of the quick sort because splitting requires the selection of the key value from the median and a loop that compares elements with the key value and placing them into the sub lists as per their values then recursively sorts the two sub lists while the join works by simply merging the two sub lists. The theoretical asymptotic bound for the merge sort is $\Theta(n \lg n)$ for the average case and $\Theta(n^2)$ for the worst case.

- Insertion Sort: The insertion sort is simply defined as a merge sort with a split of one element. The insertion sort performs a simple split that picks the rightmost element and places it into a sub list L2. L1 therefore contains the rest of elements in the list. The sort recursively sorts L1 and does not have to sort L2 because it is a single element. The sort then joins L1 and L2 by inserting L2 into L1 using a loop that shifts the elements less than the rightmost element down one slot to allow the element in L2 be placed in the right slot. Therefore we can see that the split for the insertion sort is easy while the join is harder to code just like the merge sort. The theoretical asymptotic bound for the insertion sort is $\Theta(n^2)$

- Selection Sort: The selection sort is simply defined as a quick sort with a split of one element. The sort selects the largest element and places it into sub list L2 using a loop while the rest of the elements are placed into sub list L1. The sort sorts the sub list L1 and then joins the two sub lists simply by placing the element in L2 at the end of the sorted list L1.The theoretical asymptotic bound for the selection sort is $\Theta(n^2)$

- Heap Sort: The Heap sort, like the selection sort is a specialization of the quick sort with a split of one element. The heap sort has a processing step called Build Heap that arranges the original list of values into a special order that that makes them satisfy the max heap property where the heap’s largest value is placed at the top of the tree. After the max heap is constructed, the greatest value in the list is considered sorted, removed from the tree, and placed at the end of the list. As the Heap Sort goes on, it continues to make max heaps where the greatest value is placed at the top of the tree, considered sorted, removed, and placed at the end of the list. This continues until the entire list is sorted.The theoretical asymptotic bound for the heap sort is $\Theta(n \lg n)$

2.2 Data Collection

The `SortCompAsgn.cpp` in the `dp4dsDistribution` software is the main method of collecting data of the sorts within this paper. This program runs using each sort `.hpp` file from `ASorter` and `SortCompAsgn`. The assignment and comparison counts for each sort method was collected using lists of random numbers that contained 50 numbers to 6000 numbers, with each list increasing by 500 numbers (for example, `SortCAd0500.txt` contains 500 numbers and the next list, `SortCAd1000.txt`, contains 1000 numbers). The Assignment and Comparison counts print within the `CLion` console and this data then transferred over to create tables within LaTeX and graphs within `RStudio`. A curve fit for each sort was performed and the program `RStudio` gave the Residual Standard Error for each sort as well as graphical analysis. Each graph and table created a visual representation of the data used to analyze individual sorts.

2.3 Analysis

The five sorting algorithms were analyzed through the use of the `dp4dsDistribution` software on `CLion` and `RStudio`. `RStudio` was used to organize data, such as assignment and comparison counts and determine the residual standard error (RSE) of each curve fit. In order to determine the best sort algorithm, the RSE of the graph should be low, seeing that the RSE is what determines how well the data collected fits curve. The residual standard error can be represented mathematically as:

$$RSE = \sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{d.f.}}$$

where the sum is over all the data points, y_i is the y value of an individual data point, $\hat{y}_i$ is the y value of the point on the curve whose x value is the same as the x value of y_i, and $d.f.$ is the degrees of freedom.The quadratic curve fit equation can be represented mathematically as:

$$y = An^2 + Bn + C$$

The $n \lg n$ curve fit equation can be represented mathematically as:

$$y = An \lg n + Bn + C$$

3 Results

This Section shows the raw data in table form, as well as the plots for the raw data for the number of assignment statements executed and the number of comparison statements executed for the 5 sorts; Insert sort, Selection sort, Heap sort, Merge sort, Quick sort. Furthermore, each sort is analysed in terms of curve fit and residual standard error to find the theoretical in execution time.

3.1 Raw Data

Number of	**Algorithm**				
data points	Insert	Select	Heap	Merge	Quick
500	63597	125249	7000	4344	6409
1000	254362	500499	5967	9700	13961
1500	554589	1125749	25813	15467	21921
2000	1003667	2000999	35962	21400	29254
2500	1606133	3126249	46662	27595	38687
3000	2272433	4501499	57608	33950	47903
3500	3083553	6126749	68686	40320	56134
4000	3997696	8001999	79945	46820	64984
4500	5066599	10127249	91547	53511	74613
5000	6349081	12502499	103334	60253	82888
5500	7670586	15127749	115147	66972	93824
6000	9080159	18002999	127042	73910	110112

Figure 1: A table that shows the number of array element comparisons.

Number of	**Algorithm**				
data points	Insert	Select	Heap	Merge	Quick
500	63604	1497	5667	8976	9115
1000	254367	2997	12335	19952	21966
1500	554593	4497	19415	31904	33734
2000	1003673	5997	26684	43904	45364
2500	1606137	7497	34207	56808	59646
3000	2272438	8997	41845	69808	79535
3500	3083561	10497	49556	82808	84349
4000	3997707	11997	57388	95808	94245
4500	5066605	13497	65354	109616	120458
5000	6349094	14997	73410	123616	125185
5500	7670593	16497	81442	137616	143187
6000	9080166	17997	89631	151616	162536

Figure 2: A table that shows the number of array element assignments.

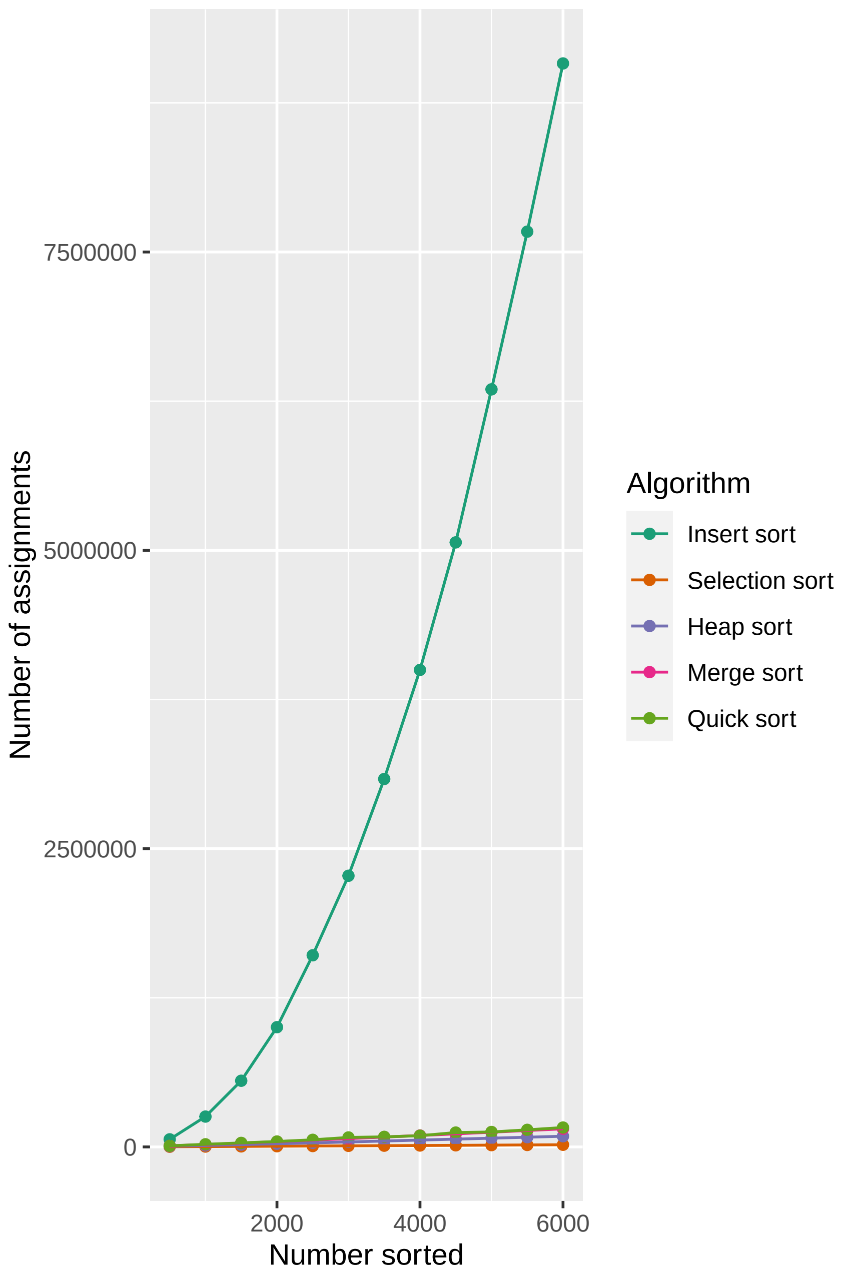

Figure 3: A figure that illustrates a plot of raw data of the number of assignment statements executed as a function of the number of data values sorted for 5 sorts; Insert sort, Selection sort, Heap sort, Merge sort, Quick sort.

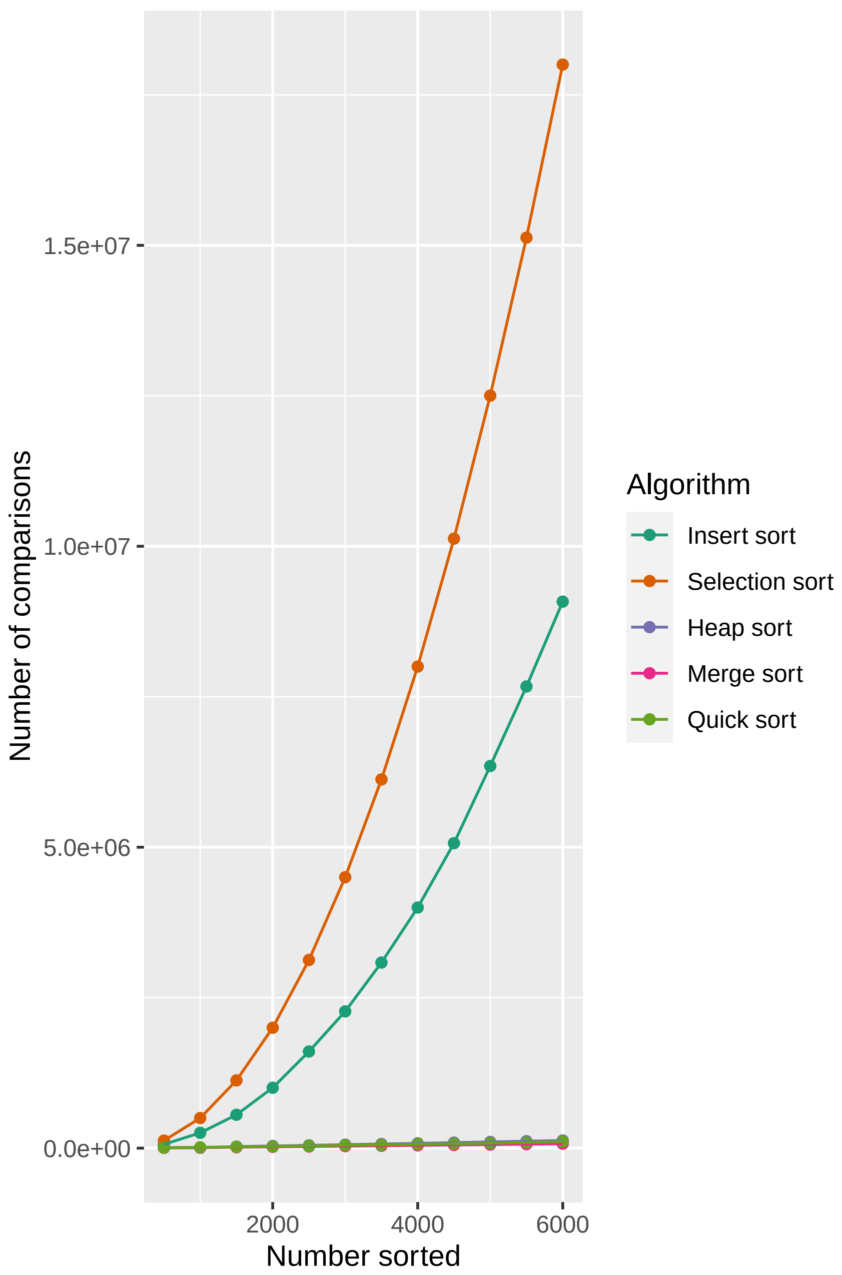

Figure 4: A figure that illustrates a plot of raw data of the number of comparisons statements executed as a function of the number of data values sorted for 5 sorts; Insert sort, Selection sort, Heap sort, Merge sort, Quick sort.

3.2 Insertion sort

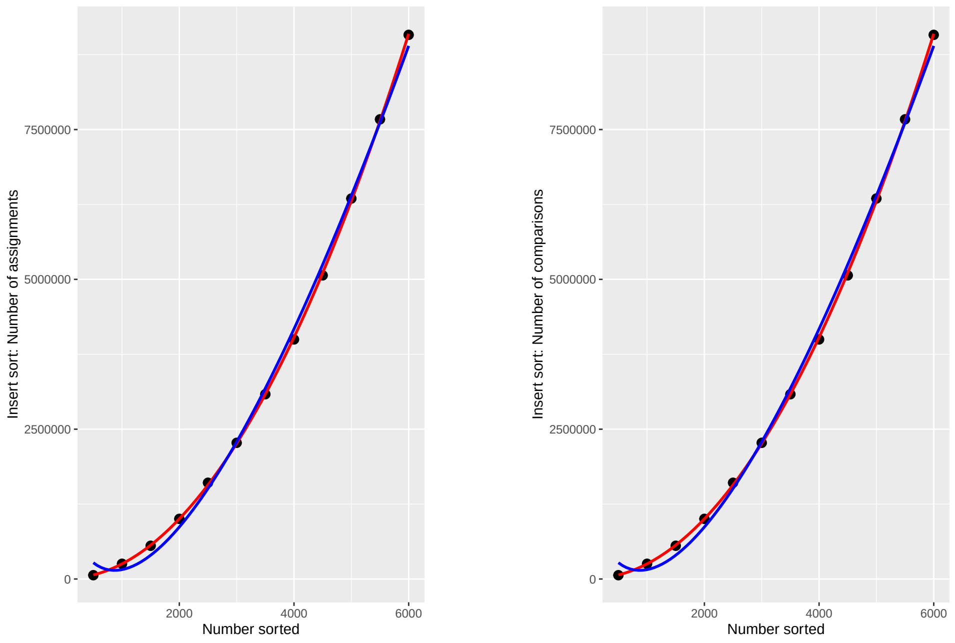

Figure 5: Side by side comparison of the plots of raw data for both the assignment statements executed (left) and comparison statements executed (right)as functions of the number of data values sorted for the insertion sort.

For the insertion sort, in terms of element comparisons, the residual standard error for the quadratic fit is 28130 on 9 degrees of freedom and for the $n\lg n$ fit is 155300 on 9 degrees of freedom. In terms of element assignments, the residual standard error for the quadratic fit is 28130 on 9 degrees of freedom and for the $n\lg n$ fit is 155300 on 9 degrees of freedom. Because for both element comparisons and element assignments the Residual Standard Error for the quadratic fit is less than that of the $n\lg n$ fit, the data shows that $\Theta(n^2)$ is the theoretical increase in execution time as a function of the number of values sorted. The insert sort results were different compared to other algorithms as the assignment and comparison had the same amount of residual standard error.This supports the theoretical asymptotic bound for the insertion sort is $\Theta(n^2)$

3.3 Selection Sort

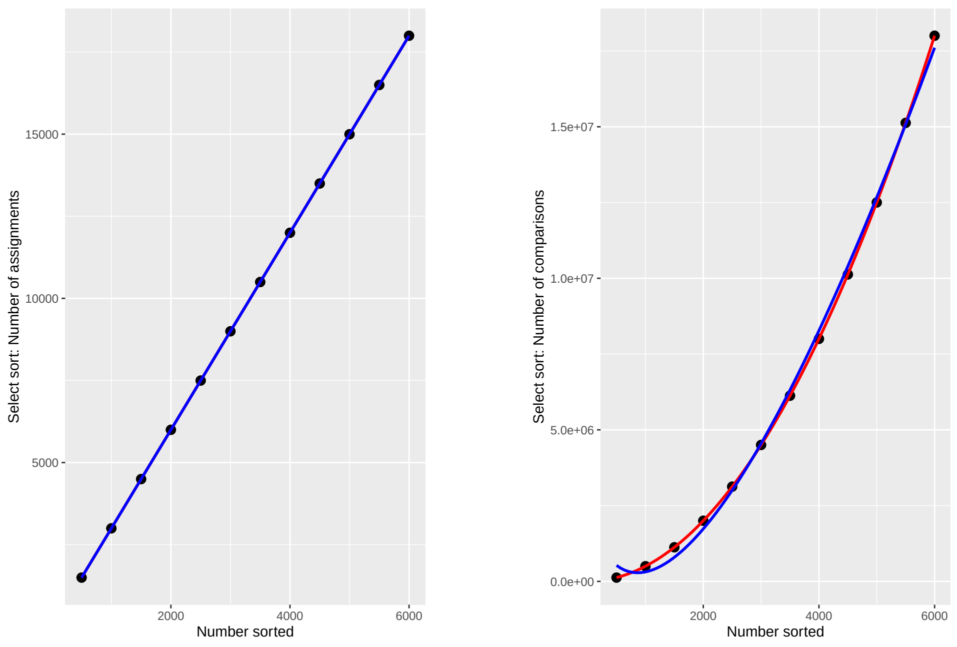

Figure 6: Side by side comparison of the plots of raw data for both the assignment statements executed (left) and comparison statements executed (right) as functions of the number of data values sorted for the selection sort.

For the selection sort, in terms of element comparisons, the residual standard error for the quadratic fit is 4.585e-09 on 9 degrees of freedom and for the $n\lg n$ fit is 291500 on 9 degrees of freedom. In terms of element assignments, the residual standard error for the quadratic fit is 3.832e-12 on 9 degrees of freedom and for the $n\lg n$ fit is 2.256e-12 on 9 degrees of freedom. Because for both element comparisons and element assignments the Residual Standard Error for the quadratic fit is less than that of the $n\lg n$ fit, the data shows that $\Theta(n^2)$ is the theoretical increase in execution time as a function of the number of values sorted. This supports the theoretical asymptotic bound for the selection sort is $\Theta(n^2)$ The select sort results were highly unusual and the algorithm used to provide the $n\lg n$ and quadratic fit data (RStudio) provided this statement in the console : 1: In summary.lm(dp4dsQuadraticFit) :essentially perfect fit: summary may be unreliable 2:In summary.lm(dp4dsNlogNFit) :essentially perfect fit: summary may be unreliable.

3.4 Heap Sort

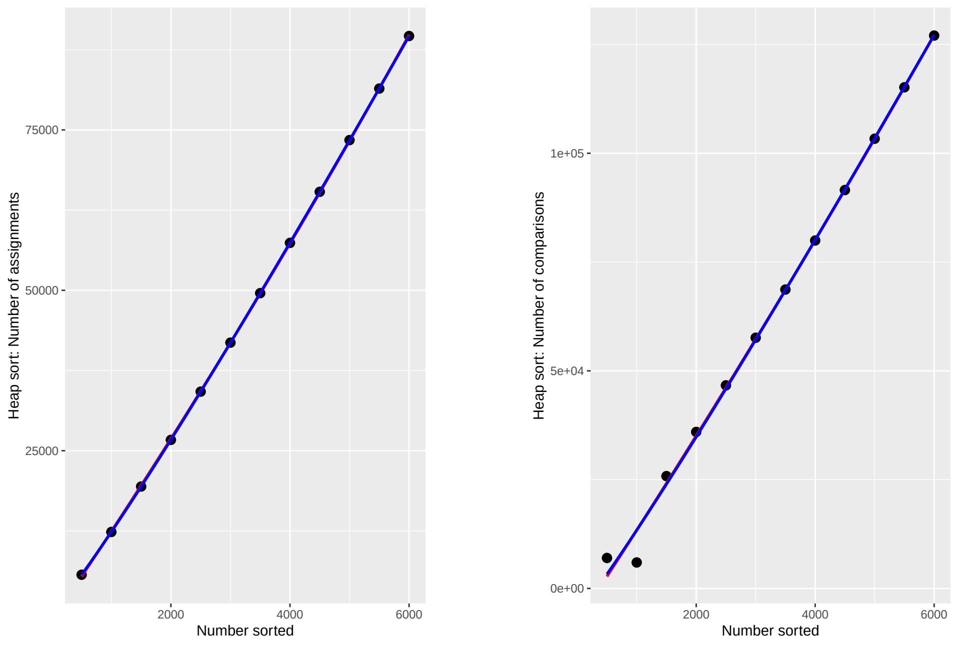

Figure 7: Side by side comparison of the plots of raw data for both the assignment statements executed (left) and comparison statements executed (right) as functions of the number of data values sorted for the heap sort.

For the heap sort, in terms of element comparisons, the residual standard error for the quadratic fit is 2927 on 9 degrees of freedom and for the $n \lg n$ fit is 2879 on 9 degrees of freedom. In terms of element assignments, the residual standard error for the quadratic fit is 155.4 on 9 degrees of freedom and for the $n \lg n$ fit is 22.41 on 9 degrees of freedom. Because for both element comparisons and element assignments the Residual Standard Error for the $n \lg n$ fit is less than that of the quadratic fit, the data shows that $\Theta(n \lg n)$ is the theoretical increase in execution time as a function of the number of values sorted. This supports the theoretical asymptotic bound for the heap sort is $\Theta(n \lg n)$

3.5 Merge Sort

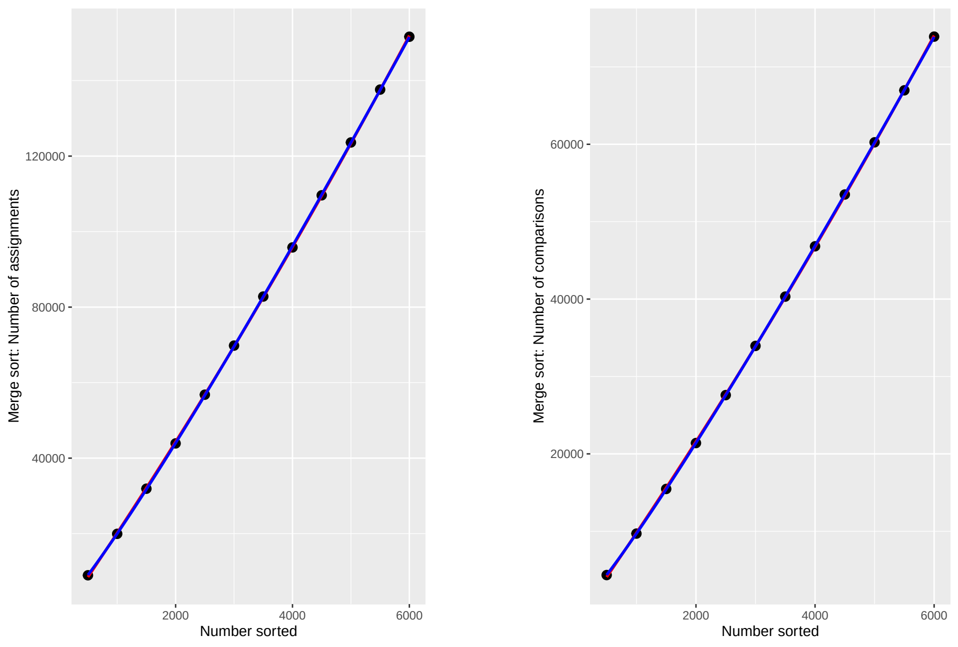

Figure 8: Side by side comparison of the plots of raw data for both the assignment statements executed (left) and comparison statements executed (right) as functions of the number of data values sorted for the merge sort.

For the merge sort, in terms of element comparisons, the residual standard error for the quadratic fit is 147.8 on 9 degrees of freedom and for the $n \lg n$ fit is 36.19 on 9 degrees of freedom. In terms of element assignments, the residual standard error for the quadratic fit is 291.8 on 9 degrees of freedom and for the $n \lg n$ fit is 190.1 on 9 degrees of freedom. Because for both element comparisons and element assignments the Residual Standard Error for the $n \lg n$ fit is less than that of the quadratic fit, the data shows that $\Theta(n \lg n)$ is the theoretical increase in execution time as a function of the number of values sorted.This supports the theoretical asymptotic bound for the merge sort is $\Theta(n \lg n)$

3.6 Quick Sort

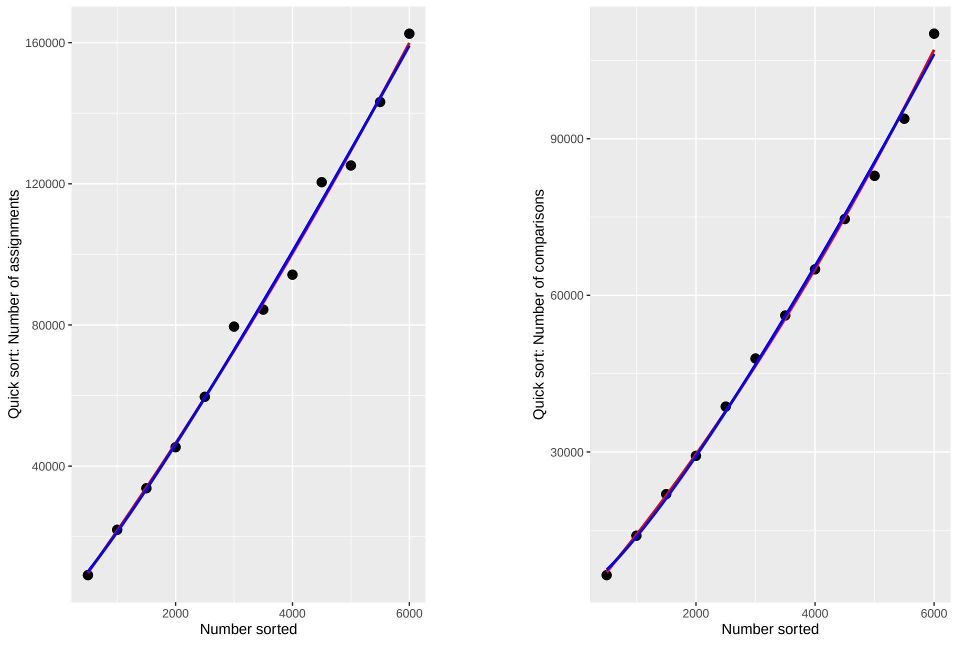

Figure 9: Side by side comparison of the plots of raw data for both the assignment statements executed (left) and comparison statements executed (right) as functions of the number of data values sorted for the quick sort.

For the quick sort, in terms of element comparisons, the residual standard error for the quadratic fit is 1611 on 9 degrees of freedom and for the $n \lg n$ fit is 1853 on 9 degrees of freedom. In terms of element assignments, the residual standard error for the quadratic fit is 4075 on 9 degrees of freedom and for the $n \lg n$ fit is 4164 on 9 degrees of freedom. Because for both element comparisons and element assignments the Residual Standard Error for the quadratic fit is less than that of the $n \lg n$ fit, the data shows that $\Theta(n^2)$ is the theoretical increase in execution time as a function of the number of values sorted.This supports the theoretical asymptotic bound for the average case of quick sort is $\Theta(n \lg n)$

3.7 Sort Comparisons

There are a few factors to consider when recommending the best sort out of the five. If a user would like to use a sorting algorithm with the least amount of assignment statements, the select sort is the best. However, for the least amount of comparisons merge sort is the best. For an overall good sorting algorithm the best is the heap sort. Not only does the raw data support this claim with the least amount of assignment and comparison statements. But also the residual standard error shows that it is much closer to a $\Theta(n \lg n)$ fit than a $\Theta(n^2)$ proving that its the fastest.

4 Conclusion

In this paper we have carefully analyzed different aspects of five sorting algorithms: Insert sort, Selection sort, Heap sort, Merge sort, Quick sort. The primary data collection method is from the `d4dpsDistribution` software on `CLion` then the data is transferred to `RStudio` that does a curve fit for the data as well as the residual standard error for each of the sorting algorithms. The residual standard error helps to determine the best sorting algorithm. The Raw data for the array element comparisons and array element assignments has been represented in a table form as well as a graphical plot of data for all five sorts. The following were the results on the theoretical increase in execution time of the number of values sorted: Insertion sort: $\Theta(n^2)$, Selection sort: $\Theta(n^2)$, Heap sort: $\Theta(n \lg n)$, Merge sort: $\Theta(n \lg n)$, Quick sort: $\Theta(n \lg n)$. Finally we have noted that the overall best sorting algorithm based on the data collected is the heap sort.

References