Analysis of Common Sorts

Olivia Lara

CoSc 320, Data Structures

Pepperdine University

October 29, 2017

Abstract

There are three main purposes of this paper, and they are listed as follows: analyze the theoretical Θ of the insertion sort, selection sort, heap sort, and merge sort, and quick sort; determine if the graphs formed from this experiment confirm to those theoretical Θ's of each of the sorts; and, conclude which of the five sort algorithms is the best in practice according to the data from this experiment. The theoretical Θ of the insertion sort, selection sort, heap sort, and merge sort, and quick sort, are as follows: insertion and selection have $\Theta(n^2)$, heap and merge have $\Theta(n \lg n)$, and quick sort has $\Theta(n \lg n)$ in the best case and $\Theta(n^2)$ in the worst case. Moreover, upon close examination of the graphs and curve fits of the data, it appears the experimental Θ's do, indeed, match the theoretical Θ's of each of the sorts. In conclusion, the data from this experiment prove that the merge sort is the best of the five algorithms.

1 Introduction

This assignment is for a computer science course at Pepperdine University called Data Structures. The objectives of the course are outlined as follows: to teach the data structures central to computer science, to teach object-oriented design patterns, and to teach C++ programming. The class is taught by the well-known and highly accredited J. Stanley Warford, who is the author of *Computer Systems* and *Computing Fundamentals*.

The following sections give an in-depth analysis of each of the following sorts and the methods used to analyze the various sorts. Section 2 explains the methods used to analyze the various sorts, specifically the characteristics of the various sorts (Section 2.1), the descriptions of the object-oriented design pattern and computer runs that took the data (Section 2.2), and the definition and use of the Residual Standard Error (Section 2.3). Additionally, Section 3 demonstrates the conclusions of the analysis of each of the sorts in the form of plots and tables. The first subsection gives the raw data in table form and graphically, the following subsections analyze each of the sorts in detail, and the last subsection explains which sort is the best. Lastly, Section 4 provides the conclusion of the entire experiment. The conclusion provides a summary of the results section with specific and detailed conclusions.

2 Method

2.1 Sort Algorithms

This paper analyzes the efficiency of five sorting algorithms: insertion sort, selection sort, merge sort, quick sort, and heap sort. A sorting algorithm is an algorithm that is composed of a sequence of instructions that takes an array as input, implements detailed procedures on the array, and returns a sorted array. Although each algorithm will return the same result, how each algorithm does such a performance varies greatly.

Insertion sort is an algorithm that creates a final sorted array one item at a time. It begins at the first index of the array and inserts each value in the correct slot. The left side of the list is sorted while the right side is unsorted until the loop reachers the last element. Thus, it starts with a sorted list of one element on the left and an unsorted list of n-1 elements on the right. It then takes the first unsorted element and inserts it into the correct position of the sorted list. By inserting the element in the correct location, the rest of the elements move as necessary. There are many approaches to moving the element over to the left (sorted) list: copy each element in the list to its neighbor or swap the elements on each iteration.

Selection sort considers every number unsorted until the algorithm sorts it itself. It starts by selecting the largest value in the array and then placing it in the last index of the array. Then, it selects the second largest value and places it in the second to last index of the array. This process repeats until each element is sorted. Or, another way to implement this is to select the smallest value in the array and then placing it in the first index of the array. However, in this implementation of the algorithm select sort searches for the largest value in the array.

Merge sort concentrates on how to merge together two initially sorted arrays in order for the resulting array to be also sorted. First, the array is split into two segments, which is done by a single line of code that initializes the value middle to the addition of the lowest index of the array, highest index of the array, and 1 all divided by 2. The next step is to examine the first element of left segment, the first element of the right segment, and the first element of the temporary array. By doing this, the temporary array will eventually be sorted and can then be copied back into the original array.

Quick sort utilizes a divide-and-conquer approach to sort an array. It selects a median and rearranges the items into two arrays where one array contains the elements that are less than the value and the other contains the elements greater than the value. The split operation functions as follows: choose a median, if the value is less than the median it is moved to the left side and if it is greater than the median it is moved to the right side. The challenge with this process is determining the median, as the only way to measure the median is to actually sort the list. Because of this, there are various strategies to determine the median; however, in this implementation of split used in our analysis, the median is the highest element of the array. The other approaches to estimate median include randomly selecting an integer index in the list or selecting three random values from the list. The next step is to recursively sort the values less than the median and then recursively sort the values greater than the median.

These characteristics allow all the algorithms discussed above to fall into two categories: easy to split and hard to join OR hard to split and easy to join. Merge sort is easy to split and hard to join, whereas quick sort is hard to split and easy to join. It is important to note that insertion sort is a member of the merge sort family and selection sort is a member of quick sort family. Therefore, insertion sort is easy to split and hard to join and selection sort is hard to split and easy to join.

In contrast, heap sort does not fall into one of the two categories and differs greatly from the algorithms described above. The reason it differs from the other algorithms is because it requires a preprocessing step that the other sort algorithms do not require, which is organizing the elements of the list into a heap. A heap is a binary tree that has a max-heap shape and a max-heap order. The author [1] defines max-heap as "for every node other than the root, the value of the node is at most the value of its parent." In other words, the first value of the heap is the largest value in the list. Implementing this strategy is very efficient when selecting the largest value, as it is the value at the top. Moreover, heap sort adds all the elements of the list into a heap, pops the largest element from the heap, and inserts it at the end of the list. It continuously does this until all elements from the list are popped out of the heap and into the sorted list.

Because of the different characteristics, some algorithms may be more efficient than others; thus, its execution time is shorter. One way of determining the performance of the sorts is to look at the theoretical asymptotic bounds of the sort algorithms. The author [1] explains that the theoretical Θ of the insertion sort, selection sort, heap sort, and merge sort, and quick sort, are as follows: insertion and selection have $\Theta(n^2)$, heap and merge have $\Theta(n \lg n)$, and quick sort has $\Theta(n \lg n)$ in the best case and $\Theta(n^2)$ in the worst case. A closer analyzation of these Θ's and if they are consistent with the experimental Θ's are discussed in following sections.

2.2 Data Collection

There are various methods we can use in order to measure the performance and determine the most efficient algorithm. The author [1] lists the three possible standards of measurements as follows:

- The number of array element comparisons
- The number of array element assignments
- The number of array probes

In this experiment, the number of array element assignments and array element comparisons are the metric used to determine the most efficient algorithm. Assignment is the operation that places an element in its ordered position and comparison is the operation that looks at two elements and determines the larger one. Furthermore, the use of decorator design patters are needed to count the executions for the various algorithms. The two decorator patterns the author [1] explains that can be used to collect data are listed below:

- Parametric decoration: used for the assignment and comparison standard of measurements
- Decorator design pattern: used for the probe standard of measurement

Because the number of array element assignments and array elements comparisons are the metric used to determine the most efficient algorithm, this experiment utilizes the parametric decoration. A C++ class called CAMetrics was written in NetBeans in order to find the count of assignments and comparisons for each sort. Additionally, this class is an object oriented program that is a decorator pattern.

2.3 Analysis

One way of determining the performance of the comparison sorts is to look at the theoretical asymptotic bounds of the sort algorithms, which describes the running time of a specified algorithm. The following is a list of Θ values that are ordered from least to greatest and explain regular algorithm performance:

$$\Theta(1) < \Theta(lgn) < \Theta(n) < \Theta(nlgn) < \Theta(n^2) < \Theta(n^3) < \Theta(2^n) < \Theta(n!)$$

An algorithm with the shortest execution time is the algorithm that has the smallest Θ. The execution time describes how many operations—or specifically assignments and comparisons in this experiment—an algorithm must carry out before it terminates and returns a sorted list. There is performance data consisting of the sum of assignment counts and comparison counts as a function of data counts (seen in the tables of the next section). The author [1] explains that computer science theory predicts the performance data of each of the comparison sorts should follow a quadratic curve or *nlgn* curve. In other words, all of the sort's execution time is T(n) = $\Theta(n^2)$ or T(n) = $\Theta(n\lg n)$. Below are the equations of the curves that match these execution times, where the execution counts are a function of data counts. Here is quadratic curve fit equation.

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

Here is $n\lg n$ curve fit equation.

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

And, statistical computation called a curve fit to the data is used to determine whether a set of data more closely matches the quadratic curve or *nlgn* curve. In order to use such a computation two things need to be provided: an equation and a set of data. The equations used in this computation are the quadratic curve and the $n\lg n$ curve show above. Additionally, the data used is the execution counts from this experiment's algorithms. This statistical computation changes the values of the coefficients so the curve matches the data as much as possible and computes a residual standard error. Here is the definition of RSE.

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

The curve fit that has the smaller RSE is the one that more accurately matches the data.

3 Results

3.1 Raw Data

Each of the various algorithms sort numbers differently. Therefore, the total number of assignment and comparisons vary for each algorithm. The raw data for the execution counts for each algorithm are displayed in table format on the next page.

	Number of data points									
Algorithm	200	400	800	1600	2400	3200	4000	4800	5600	6400
Insert	10130	40023	166851	638438	1448491	2548181	4024049	5760169	7789330	10231222
Select	597	1197	2397	4797	7197	9597	11997	14397	16797	19197
Heap	2012	4424	9635	20866	32711	44905	57347	70124	83178	96278
Merge	3088	6976	15552	34304	54208	75008	95808	118016	140416	162816
Quick	3343	7884	15877	34836	58070	83401	102021	120374	146053	170811

Figure 1: A pdf figure to illustrate the number of assignments in table form.

	Number of data points									
Algorithm	200	400	800	1600	2400	3200	4000	4800	5600	6400
Insert	10323	40419	167645	640025	1450879	2551370	4028037	5764961	7794921	10237610
Select	20099	80199	320399	1280799	2881199	5121599	8001999	11522399	15682799	20483199
Heap	2284	5365	12304	27826	44517	62013	79882	98545	117604	136868
Merge	1470	3366	7511	16671	26381	36477	46787	57523	68411	79401
Quick	2183	5300	10970	23634	36826	52105	66028	78555	94341	109832

Figure 2: A pdf figure to illustrate the number of comparisons in table form.

Figure 1 organizes the assignment data for each of the different sorts. There is one independent variable called numSorted and five dependent variables called InsertAsgn, SelectAsgn, HeapAsgn, MergeAsgn, and QuickAsgn. The first column of the table gives the specific amount of numbers that are sorted. The following columns are separated by the different sort's and their corresponding assignment counts.

Figure 2 organizes the comparison data for each of the different sorts. There is one independent variable called numSorted and five dependent variables called Insert-Comp, SelectComp, HeapComp, MergeComp, and QuickComp. The first column of the table gives the specific amount of numbers that are sorted. The following columns are separated by the different sort's and their corresponding comparison counts.

There are various ways to interpret the graphs displayed above. One way is to see how the value of numbers sorted correspond to a specific algorithm. As you move across one row there is a different count for each algorithm because each algorithm functions in a different way. Additionally, the table exhibits how the value of numbers sorted affects the execution counts. To exemplify, as you move down the InsertAsgn row in Figure 1 the counts increase. This pattern is the same for each of the executions of an algorithm, which makes sense because the more numbers to be sorted the more the algorithm will need to assign and/or compare.

Another way of examining the data other than in a table format is to create a plot of the data. It is possible to plot a family of raw data in a line plot for a given data frame. In other words, it is possible to plot the counts seen in the tables below graphically.

Figure 3 displays the plot of the assignments data. The independent variable is the value of numbers sorted and the dependent variable is the total number of assignments for each of the different sorts. This graph highlights that insert sort has a significantly higher total number of assignments, as the line for insert sort is far above the other lines. The other sorts vary slightly with the total number of assignments, each one differing more as the value of numbers sorted increases. Nonetheless, the following sorts are ordered from the highest total number of assignments to the lowest total number of assignments: quick sort, merge sort, heap sort, select sort.

Figure 4 displays the plot of the comparisons data. The independent variable is the value of numbers sorted and the dependent variable is the total number of comparisons for each of the different sorts. This graph highlights that select sort has a significantly higher total number of comparisons. Although not as large as select sort, insert sort also has a noticeably high total number of comparisons. One can determine the total number or comparisons for select sort and insert sort is far greater than the remaining sorts because their lines are way above the other lines. The other sorts vary slightly with the total number of comparisons, each one differing more as the value of numbers sorted increases. In fact, the other sorts variation is so small that it is hard to determine by simply looking at the graph, as the lines are almost on top of each other.

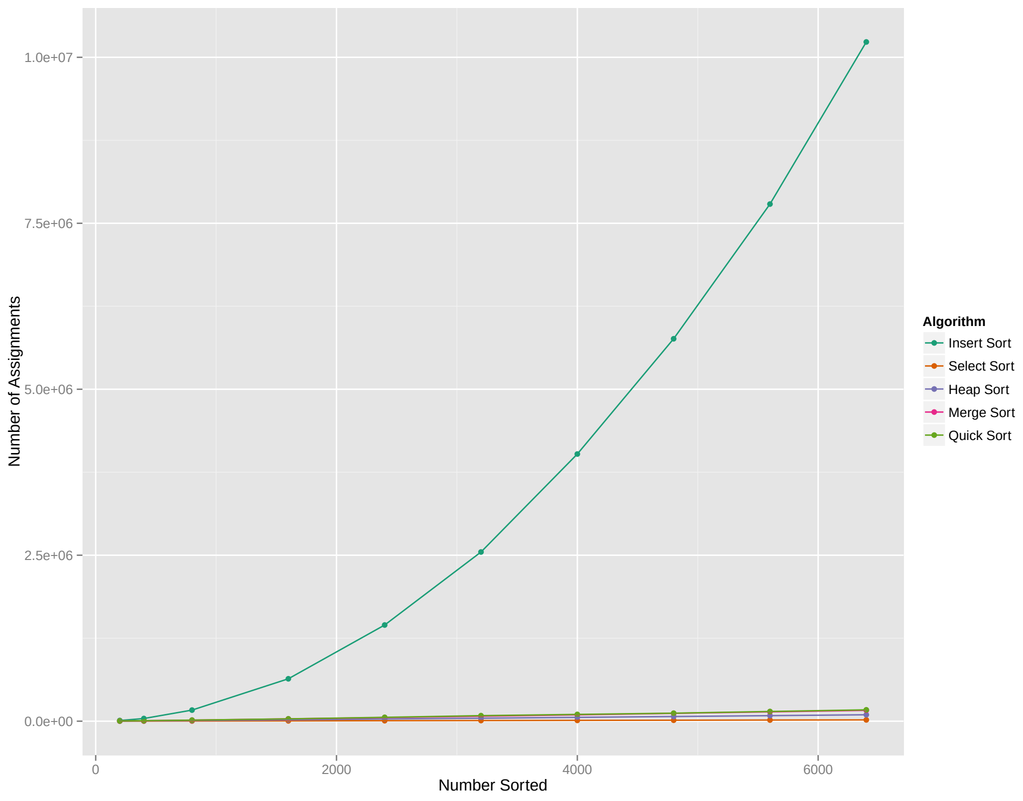

Figure 3: A pdf figure to illustrate the assignments data graphically.

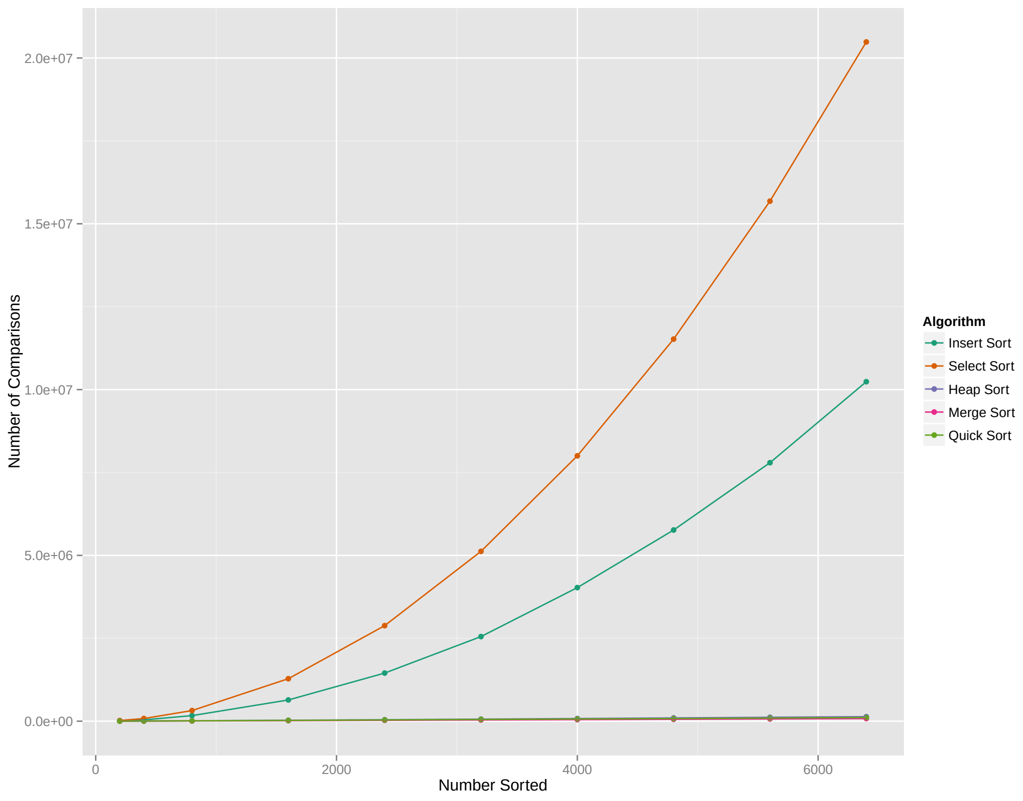

Figure 4: A pdf figure to illustrate the comparisons data graphically.

3.2 Insert Sort

Figure 5 shows the difference in number of comparisons and number of assignments for insert sort. There is approximately the same counts for both executions.

Figure 6 shows the two curve fits for quadratic and $n\lg n$ curve for insert sort according to the summation of executions for the sort. The red line represents the quadratic curve and the blue line represents the $n\lg n$ curve.

Here are the coefficients for the quadratic curve.

$y-intercept : 6534307$

$n : 20926242$

$n^2 : 5548183$

Therefore, here is the quadratic curve fit equation.

$y = 5548183n^2 + 20926242n + 6534307$

Lastly, here is the residual standard error on 7 degrees of freedom for this equation.

$RSE : 37830$

Here are the coefficients for the $n\lg n$ curve.

$y-intercept : 1673654.1$

$n : -18672.7$

$nlgn : 2455.2$

Therefore, here is the $n\lg n$ curve fit equation.

$y = 2455.2n\lg n + -18672.7n + 1673654.1$

Lastly, here is the residual standard error on 7 degrees of freedom for this equation.

$RSE : 466700$

The quadratic curve creates the best fit curve because it has a smaller residual error than the $n\lg n$ curve.

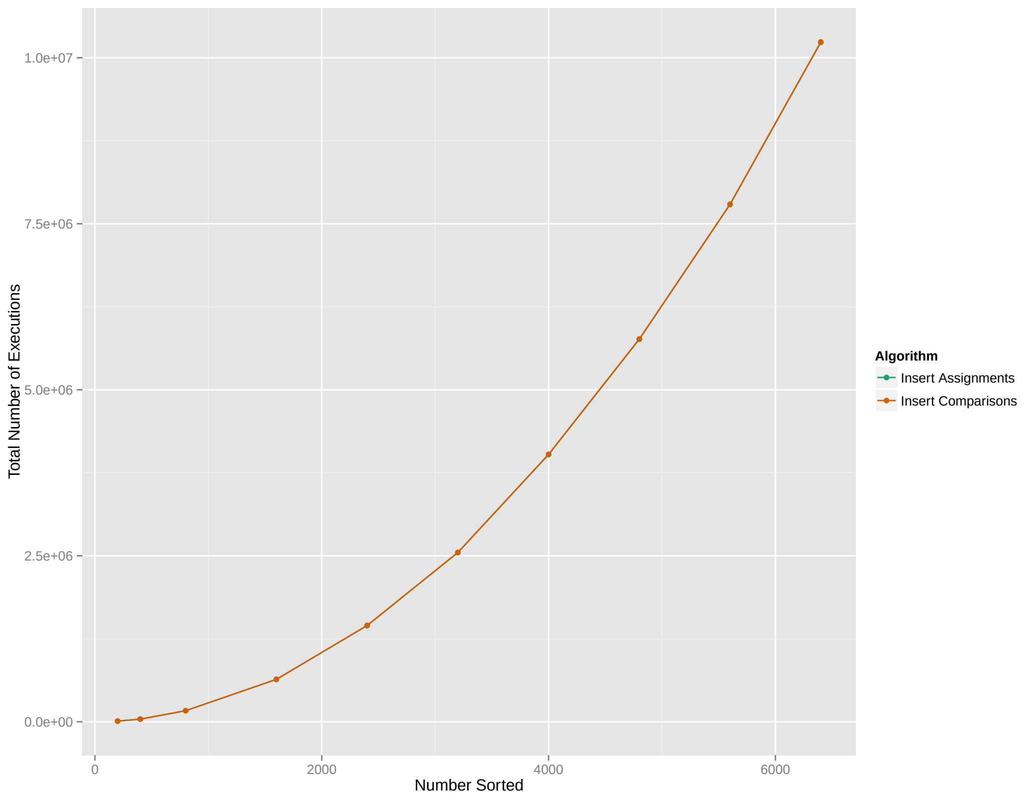

Figure 5: A pdf figure to illustrate the difference in number of comparisons and number of assignments for insert sort.

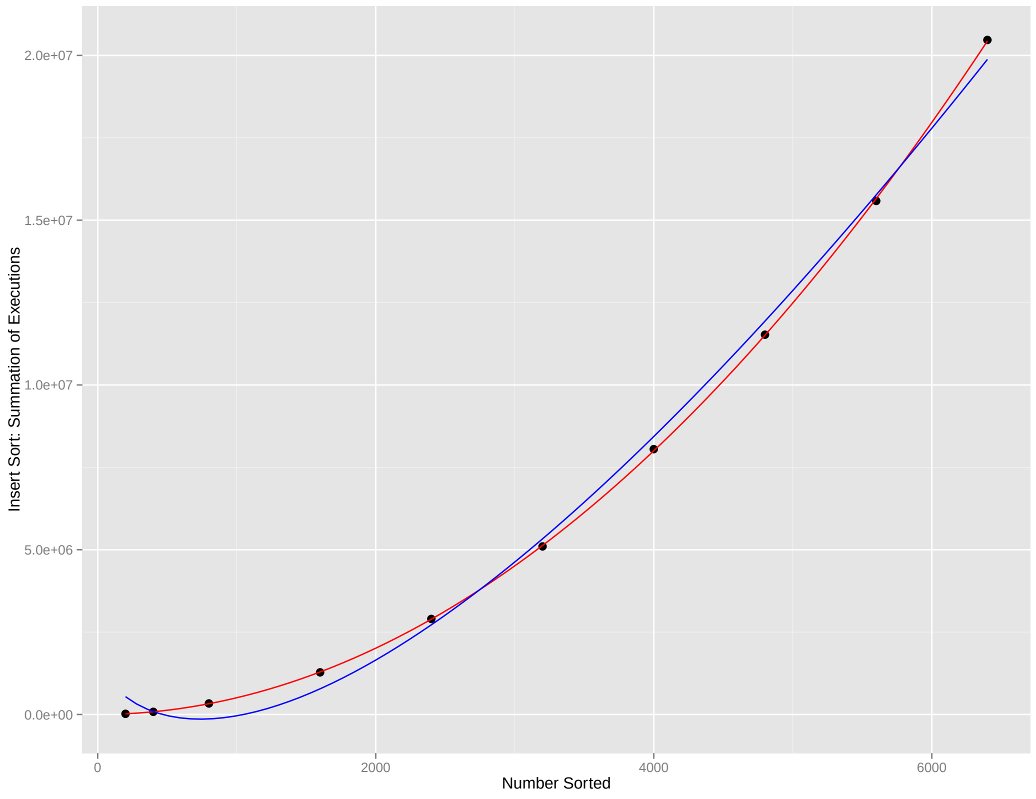

Figure 6: A pdf figure to illustrate the two curve fits for quadratic and $n \lg n$ for select sort according to the summation of executions for the insert sort.

3.3 Select Sort

Figure 7 shows the difference in number of comparisons and number of assignments for select sort. There are far more assignment operations than comparison operations.

Figure 8 shows the two curve fits for quadratic and $n \lg n$ for select sort according to the summation of executions for the sort. The red line represents the quadratic curve and the blue line represents the $n \lg n$ curve.

Here are the coefficients for the quadratic curve.

$$y - intercept : 6.548e + 06$$

$$n : 2.099e + 07$$

$$n^2 : 5.582e + 06$$

Therefore, here is the quadratic curve fit equation.

$$y = 5.582e + 06n^2 + 2.099e + 07n + 6.548e + 06$$

Lastly, here is the residual standard error on 7 degrees of freedom for this equation.

$$RSE : 2.523e - 09$$

Here are the coefficients for the $n \lg n$ curve.

$$y - intercept : 1686354.6$$

$$n : -18798.9$$

$$nlgn : 2470.5$$

Therefore, here is the $n \lg n$ curve fit equation.

$$y = 2470.5n \lg n + -18798.9n + 1686354.6$$

Lastly, here is the residual standard error on 7 degrees of freedom for this equation.

$$RSE : 467400$$

The quadratic curve creates the best fit curve because it has a smaller residual error than the $n \lg n$ curve.

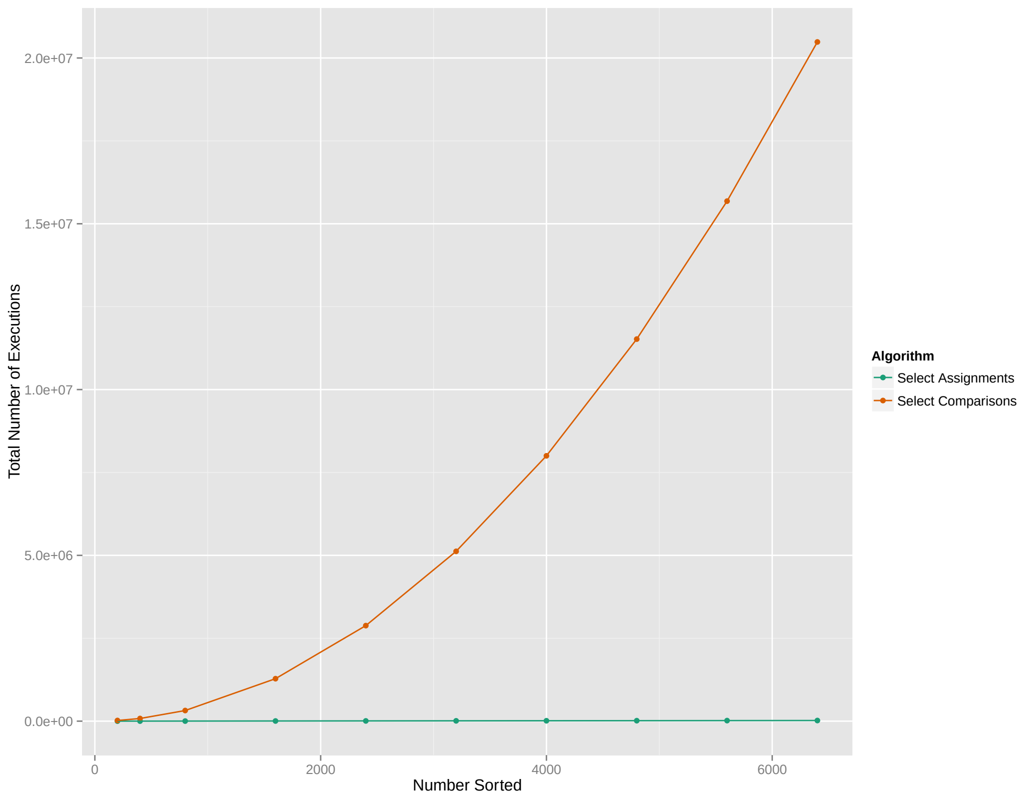

Figure 7: A pdf figure to illustrate the difference in number of comparisons and number of assignments for insert sort.

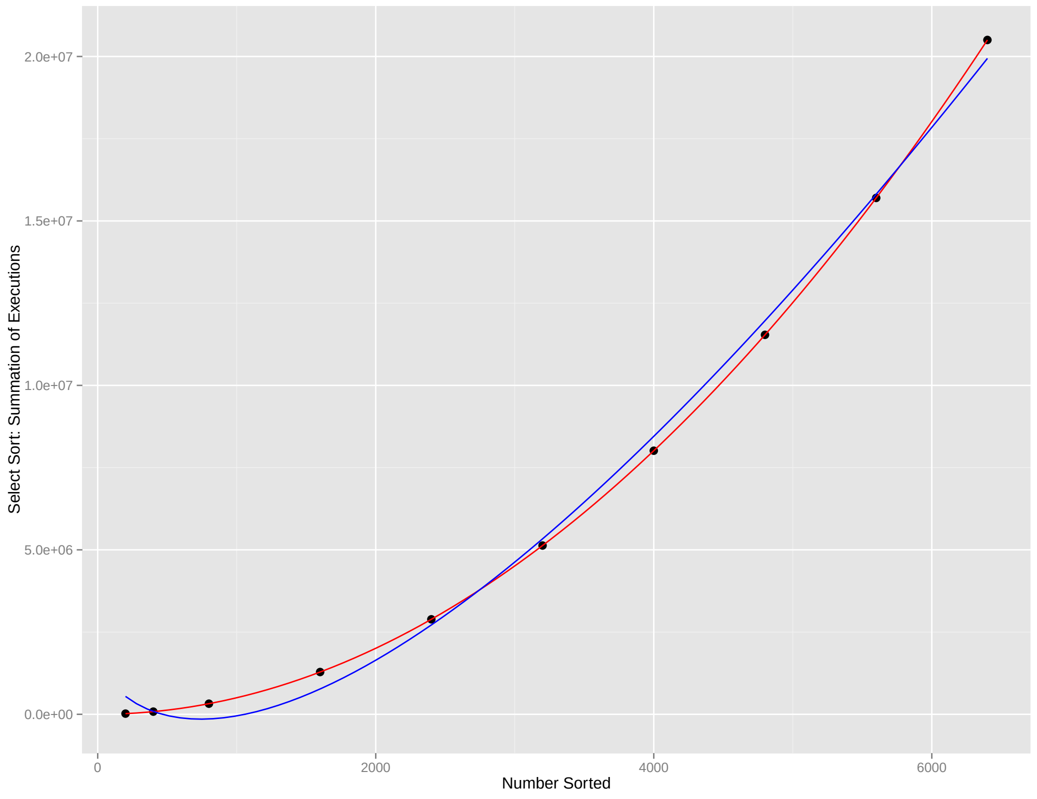

Figure 8: A pdf figure to illustrate the two curve fits for quadratic and nlgn for select sort according to the summation of executions for the insert sort.

3.4 Heap Sort

Figure 9 shows the difference in number of comparisons and number of assignments for heap sort. There are more assignment operations than comparison operations.

Figure 10 shows the two curve fits for quadratic and $n \lg n$ for heap sort according to the summation of executions for the sort. The red line represents the quadratic curve and the blue line represents the $n \lg n$ curve.

Here are the coefficients for the quadratic curve.

$y - intercept : 100868.8$

$n : 245915.9$

$n^2 : 9487.6$

Therefore, here is the quadratic curve fit equation.

$y = 9487.6n^2 + 245915.9n + 100868.8$

Lastly, here is the residual standard error on 7 degrees of freedom for this equation.

$RSE : 742.9$

Here are the coefficients for the $n \lg n$ curve.

$y - intercept : 113.4081$

$n : -2.1126$

$nlgn : 4.3947$

Therefore, here is the $n \lg n$ curve fit equation.

$y = 4.3947n \lg n + -2.1126n + 113.4081$

Lastly, here is the residual standard error on 7 degrees of freedom for this equation.

$RSE : 120.5$

The logarithmic curve creates the best fit curve because it has a smaller residual error than the quadratic curve.

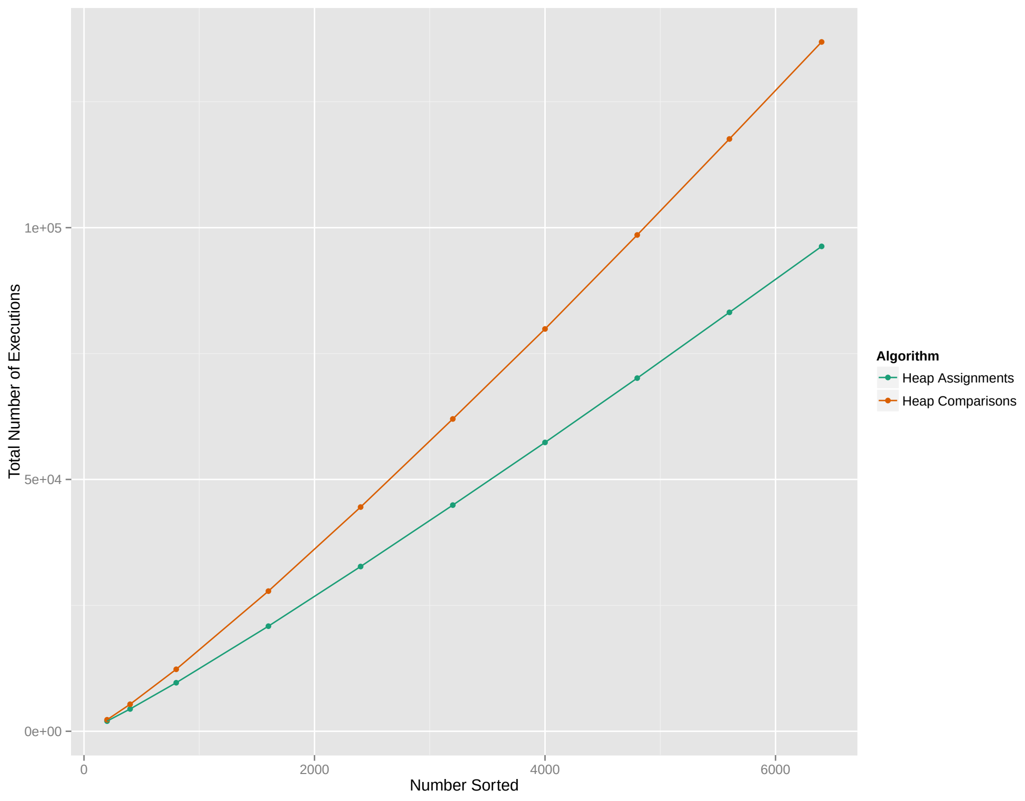

Figure 9: A pdf figure to illustrate the difference in number of comparisons and number of assignments for insert sort.

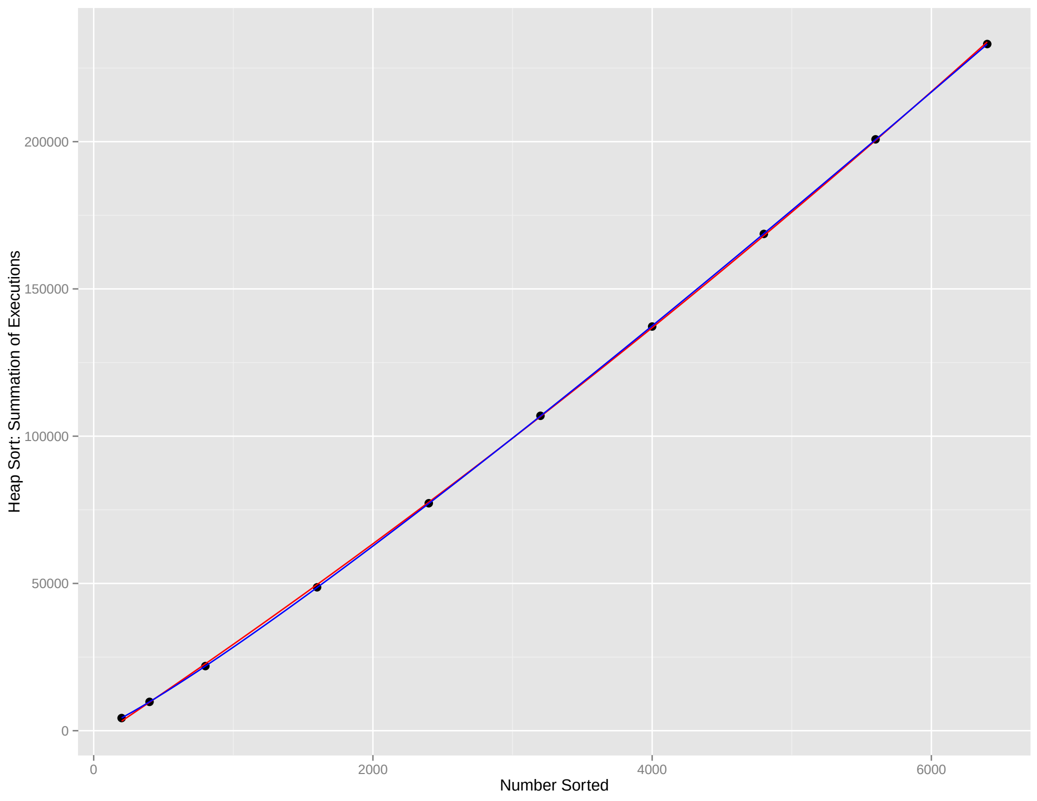

Figure 10: A pdf figure to illustrate the two curve fits for quadratic and nlgn for select sort according to the summation of executions for the insert sort.

3.5 Merge Sort

Figure 11 shows the difference in number of comparisons and number of assignments for merge sort. There are more assignment operations than comparison operations.

Figure 12 shows the two curve fits for quadratic and $n \lg n$ for merge sort according to the summation of executions for the sort. The red line represents the quadratic curve and the blue line represents the $n \lg n$ curve.

Here are the coefficients for the quadratic curve.

$y - intercept : 105019.0$

$n : 255384.2$

$n^2 : 9579.7$

Therefore, here is the quadratic curve fit equation.

$y = 9579.7n^2 + 255384.2n + 105019.0$

Lastly, here is the residual standard error on 7 degrees of freedom for this equation.

$RSE : 742.9$

Here are the coefficients for the $n \lg n$ curve.

$y - intercept : 145.0701$

$n : -1.002$

$nlgn : 4.4298$

Therefore, here is the $n \lg n$ curve fit equation.

$y = 4.4298n \lg n + -1.0026n + 145.0701$

Lastly, here is the residual standard error on 7 degrees of freedom for this equation.

$RSE : 224.6$

The logarithmic curve creates the best fit curve because it has a smaller residual error than the quadratic curve

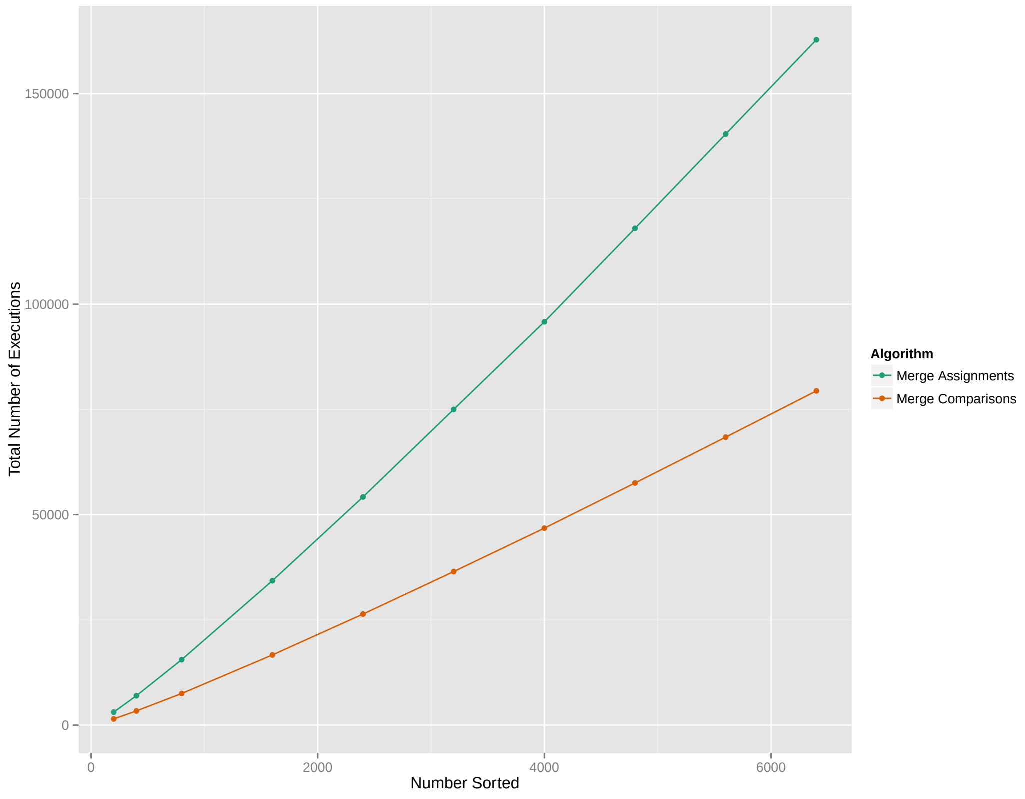

Figure 11: A pdf figure to illustrate the difference in number of comparisons and number of assignments for insert sort.

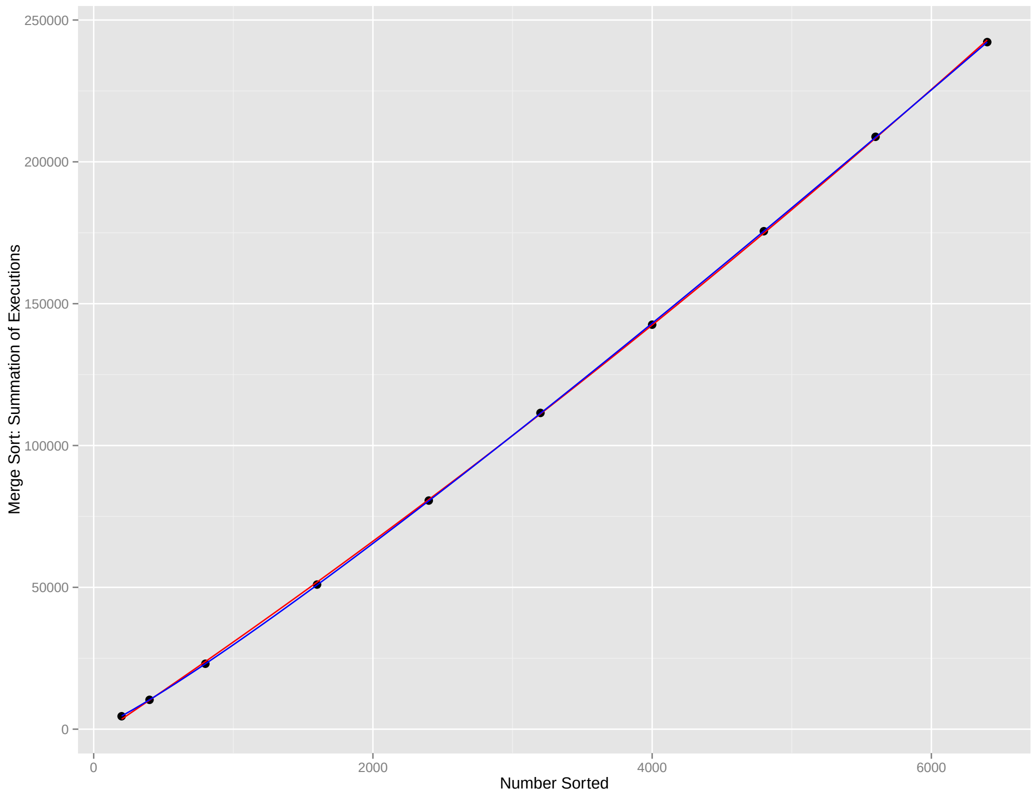

Figure 12: A pdf figure to illustrate the two curve fits for quadratic and nlgn for select sort according to the summation of executions for the insert sort.

3.6 Quick Sort

Figure 13 shows the difference in number of comparisons and number of assignments for quick sort. There are more assignment operations than comparison operations.

Figure 14 shows the two curve fits for quadratic and $n \lg n$ for quick sort according to the summation of executions for the sort. The red line represents the quadratic curve and the blue line represents the $n \lg n$ curve.

Here are the coefficients for the quadratic curve.

$$y - intercept : 122244$$

$$n : 294124$$

$$n^2 : 8107$$

Therefore, here is the quadratic curve fit equation.

$$y = 8107n^2 + 294124n + 122244$$

Lastly, here is the residual standard error on 7 degrees of freedom for this equation.

$$RSE : 2982$$

Here are the coefficients for the $n \lg n$ curve.

$$y - intercept : -885.268$$

$$n : 10.535$$

$$nlgn : 3.786$$

Therefore, here is the $n \lg n$ curve fit equation.

$$y = 3.786n \lg n + 10.535n + -885.268$$

Lastly, here is the residual standard error on 7 degrees of freedom for this equation.

$$RSE : 2888$$

The logarithmic curve creates the best fit curve because it has a smaller residual error than the quadratic curve.

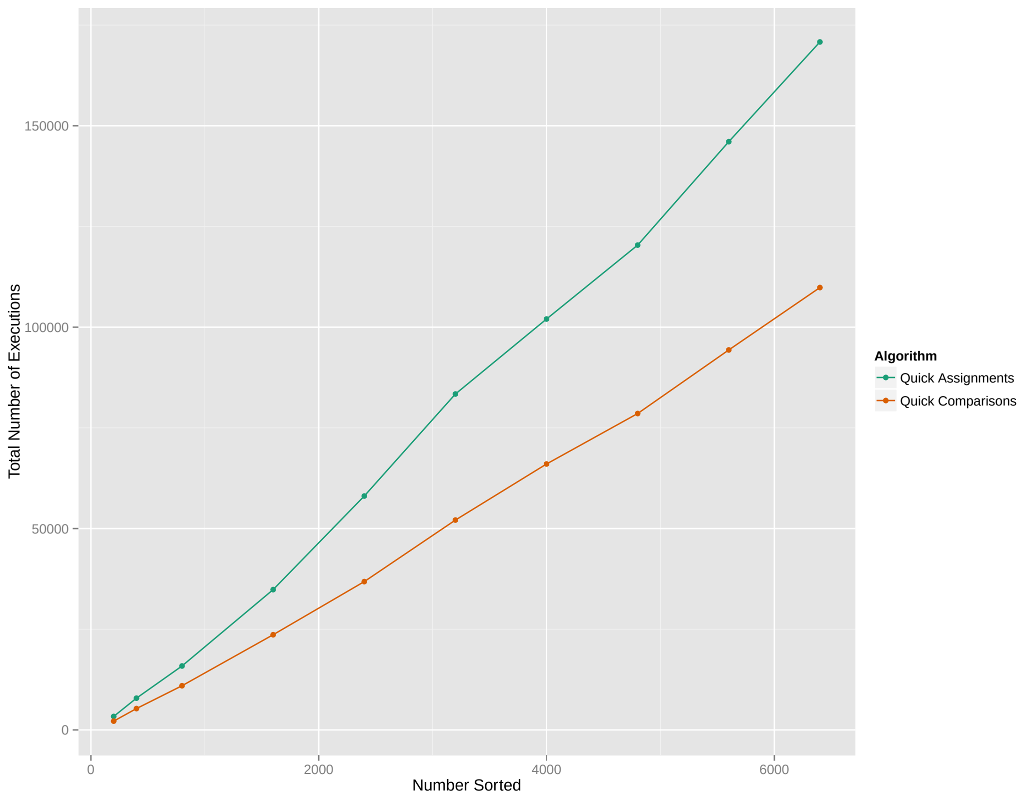

Figure 13: A pdf figure to illustrate the difference in number of comparisons and number of assignments for insert sort.

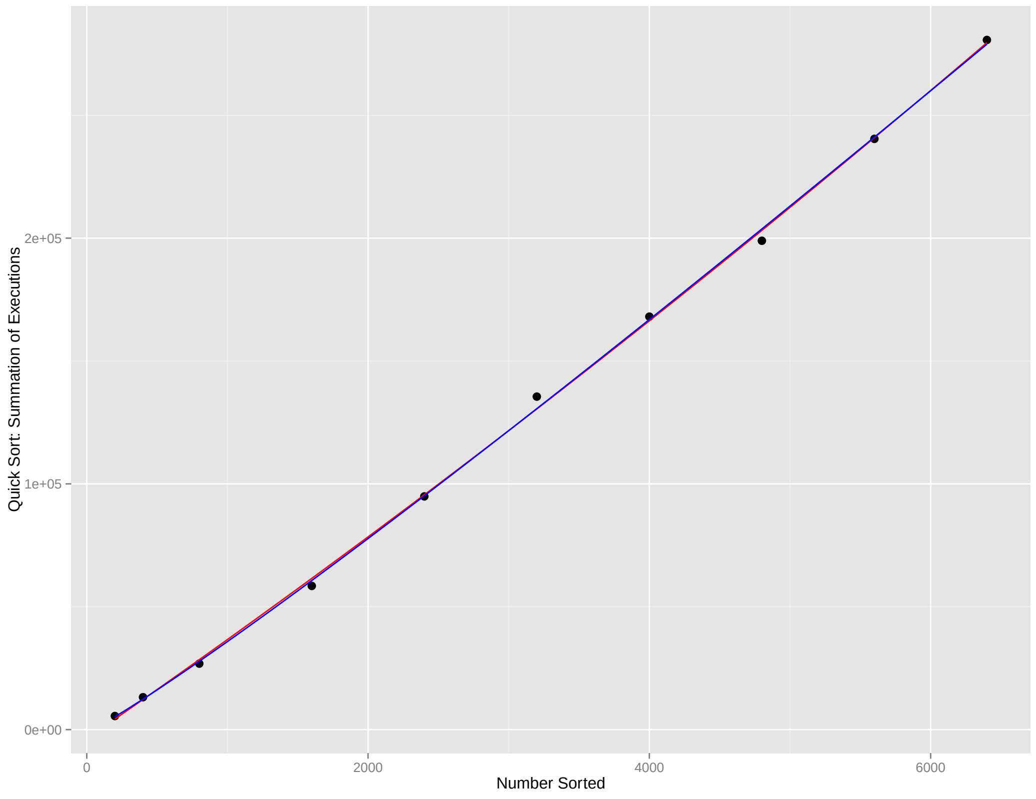

Figure 14: A pdf figure to illustrate the two curve fits for quadratic and nlgn for select sort according to the summation of executions for the insert sort.

3.7 Sort Comparisons

The results from this experiment prove that heap sort is the most efficient algorithm. There are two components that suggest heap sort is the best: the summation of its executions and its execution time, which is represented by the theta and found from the curve fits.

Figure 15 illustrates the summation of executions, which is the addition of assignment counts and comparison counts, for all of the algorithms. The algorithm that is able to sort the elements in the array with a fewer count of assignments and comparisons is the best algorithm. Selection and insertion are extremely inefficient in comparison to heap, merge, and quick because they require far more executions, as portrayed in both the algorithms lines in figure 15. Moreover, it appears heap, merge, and quick all have approximately the same counts. Upon closer examination of the graph, however, quick sort requires more executions followed by merge sort and then heap sort. In short, figure 15 illustrates heap sort is the best sort because it's line is the lowest one; therefore, it's execution counts are fewer than the other comparison sorts.

Another way the results from this experiment prove heap sort is the best is when examining the residual standard errors, or RSE of the curve fits. If a particular sorts' data matches the curve fit for the quadratic equation then this shows they are less efficient than the sort who data that matches

The sort whose data better matches the curve fit for the logarithmic equation is more efficient than the sort whose data better matches the curve fit for a quadratic equation. This is because a sort is matched with the $\Theta(n^2)$ equation if the run time for that particular equation is T(n) = $\Theta(n^2)$ and a sort is matched with the $\Theta(n \lg n)$ equation if the run time for that particular equation is T(n) = $\Theta(nlgn)$. Furthermore, an algorithm with the shortest execution time is the algorithm that has the smallest Θ. Because insert sort and select sort's data both match the curve fit for the quadratic equation—as portrayed in the section that analyses insert sort and select sort— they are immediately eliminated in the running for the most efficient algorithm. This leaves heap, merge, and quick sort for the running of the most efficient algorithm.

In order to further understand these sorts efficiency, the RSE is helpful. If the RSE is smaller for the $\Theta(nlgn)$ equation than this implies the algorithm matches the $\Theta(nlgn)$ better, and thus is the best algorithm. The following portrays the RSE of merge, heap, and quick sort (also seen in the analysis for each of these sorts) curve fits for the $\Theta(nlgn)$ equation: Merge RSE = 224.6, Heap RSE = 120.5, and Quick RSE= 2888. Because the RSE for heap sort is the smallest, it is the best. Additionally, heap appears to be the second best and quick appears to be the third best.

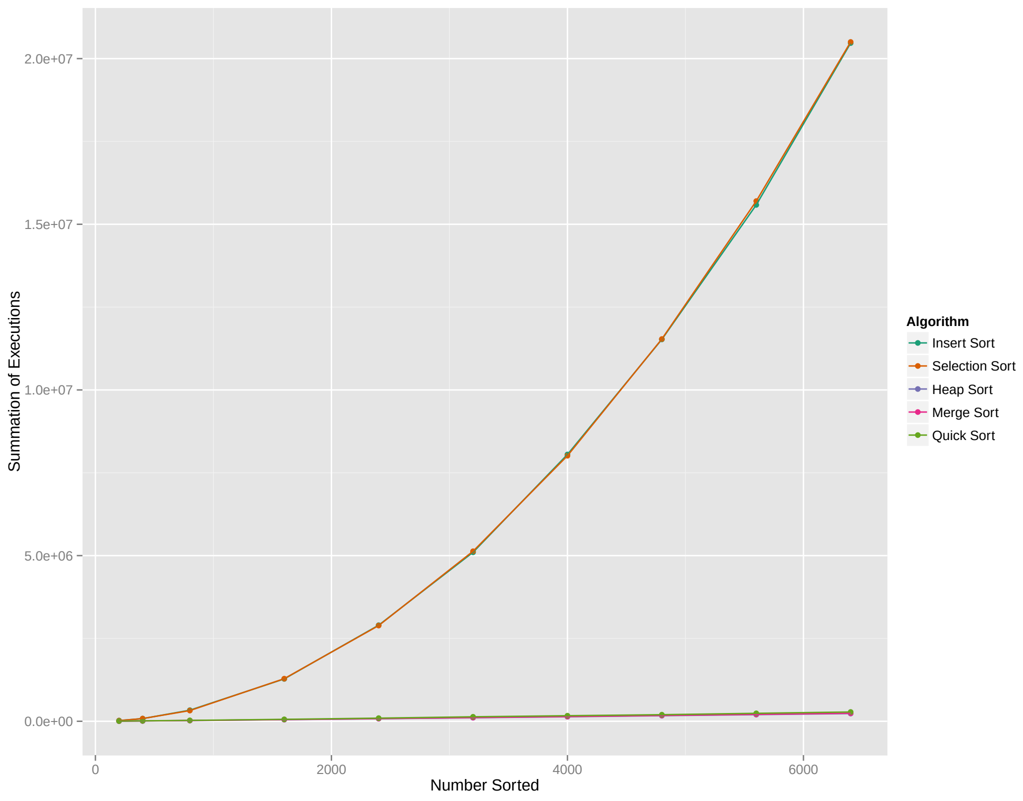

Figure 15: A figure to illustrate the summation of executions for all of the algorithms.

4 Conclusion

The theoretical Θ of the insertion sort, selection sort, heap sort, and merge sort, and quick sort, are as follows: insertion and selection have $\Theta(n^2)$, heap and merge have $\Theta(n \lg n)$, and quick sort has $\Theta(n \lg n)$ in the best case and $\Theta(n^2)$ in the worst case. The experimental Θ of each of the sorts match their theoretical Θ (match the best case for quick sort). Further specification of how this is determined is explained in the previous section. Moreover, the results also show that heap sort is the best algorithm because it requires the least amount of executions; thus, it requires the least amount execution time.

References

[1] Dung X. Nguyen and J. Stanley Warford. *Design Patterns for Data Structures*. Pepperdine, prepublication manuscript edition, 2016.