A Proper Look at the Efficiency of Common Sorting Algorithms

Frank Garcia

CoSc 320, Data Structures

Pepperdine University

November 2, 2015

Abstract

The purpose of this paper is to see if common conceptions about sorting algorithms and their theoretical efficiencies are correct when tested against real world conditions. The paper strives to answer the question of whether or not a variety of sorts are as efficient as they are regarded to be. While many of the sorts do in fact match their theoretical standards, a few return less than expected results, that stray away from common conceptions in regards to the sorts efficiency.

1 Introduction

This research was conducted by myself for an assignment in the Data Structures course at Pepperdine University. The task was to run tests on popular sorting algorithms to see if the theory of how efficient they are matches up against real, tested results. The goal was to see what sorts more closely matched a graph of n^2 and what graphs more closely match $nlgn$ as more data had to be sorted.

In terms of the rest of the paper, it will be split into three more sections, each of vital importance. The method section will look at how the data was gathered and analyzed in a meaningful way to check the effectiveness of each algorithm. The results section will use the data that was gathered in the method section and compare it assumed results to see if the results match up with popular theory, and explain why or why not the data does or does not support popular claims. Finally, the conclusion will look at all of the data acquired and draw important developments on the idea of sorting algorithms and their efficiencies.

Algorithm	Number of data points									
	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	2182	3799	10186	21776	33097	41572	56010	68293	78345	101197
Merge	2088	6976	15552	34304	54208	75008	95808	118016	140416	162816
Quick	20284	67809	265252	1046810	2276555	3987967	6185394	8909543	12016837	15612468

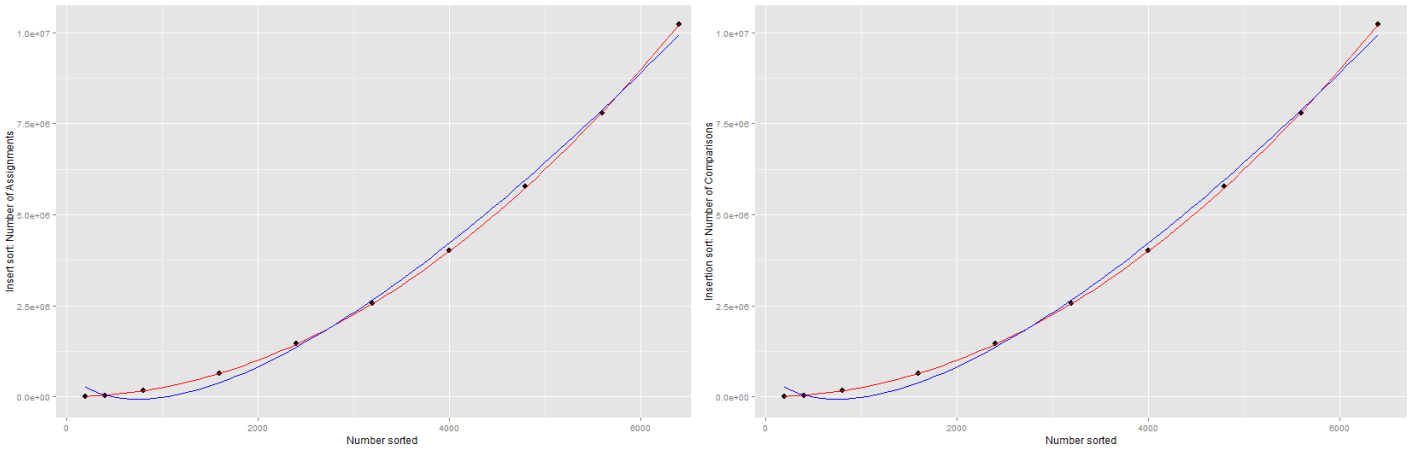
Figure 1: Number of Assignments.

Algorithm	Number of data points									
	200	400	800	1600	2400	3200	4000	4800	5600	6400
Insert	10124	40020	166846	638426	1448480	2548171	4024038	5760162	7789322	10231211
Select	20099	80199	320399	1280799	2881199	5121599	8001999	11522399	15682799	20483199
Heap	2240	3799	11914	26707	41055	50773	70653	86955	99487	134798
Merge	1470	3366	7511	16671	26381	36477	46787	57523	68411	79401
Quick	7855	24667	93591	360256	776071	1352686	2085777	3003572	4043429	5243034

Figure 2: Number of Comparisons.

2 Method

To approach the task of measuring all of the data, multiple sorts were set up in a standard recursive fashion and there was an overloading of the operators of $<$ and $<=$ so that a there would be a count of +1 for each time an comparison was made. Only these specific operators had to be overloaded because each program was set up to structure each comparison so that only less than and less than or equals to signs were used. On top of this assignments were counted every time there was an assignment (=) call. After this, tests were ran using increasingly big data sets, with the number of assignments and comparisons measured on each run (see figure 1 and figure 2). This new found data was then inputed into an R function that produces an RSE graph based on the graph of each sort as the data sets increase compared against the graphs of n^2 and $nlgn$. The RSE, or Residual Standard Error ($RSE = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{d.f.}}$), is a measure of how closely one graph



(a) Assignments (b) Comparisons

Figure 3: Insertion Sort.

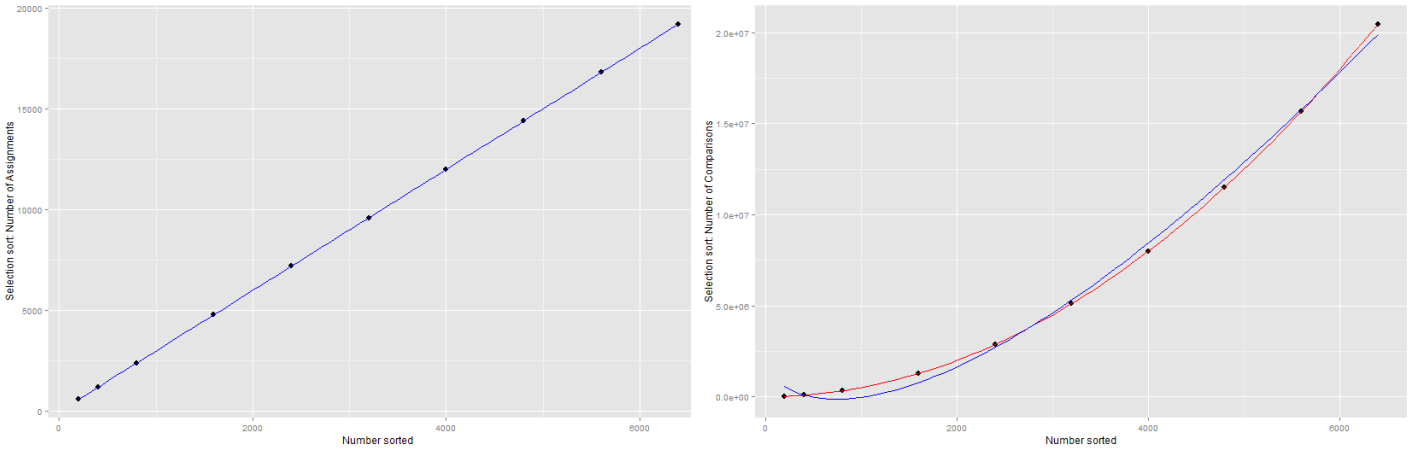
maps onto the other, or in other words how many units off one graph is from another at each deviation. The R function would then also output the RSE value up to seven standard deviations with n^2 and $nlgn$ so that it was obvious which one the graph fit to better (the one with the lower RSE). With this data, comparisons were between the RSE outputted by each sort, and the theory of what should have been outputed.

3 Results

3.1 Insertion Sort

The insertion sort matched popular theory perfectly and more closely fit to a curve of n^2. as seen in Figure 3 This can be seen by the results because the RSE of n^2 for both assignments and comparisons (18920) is significantly smaller than the RSE of $nlgn$ for both assignments and comparisons (233300). This means that the graph more accurately maps onto the graph of n^2 than it does the graph of $nlgn$. We know from this that its efficiency is closer to $O(n^2)$ than $O(nlgn)$, since the graph shows how efficient a sort is based on the number of data points.

The only interesting thing to note about the sort was that it had the same RSE for assignments and comparisons. The reason for this is not readily apparent, but actually has to do with the fact that the number of comparisons and assignments executed was always so close. Because of this, an RSE measured at seven standard deviations does not catch the difference between the two.



(a) Assignments (b) Comparisons

Figure 4: Selection Sort.

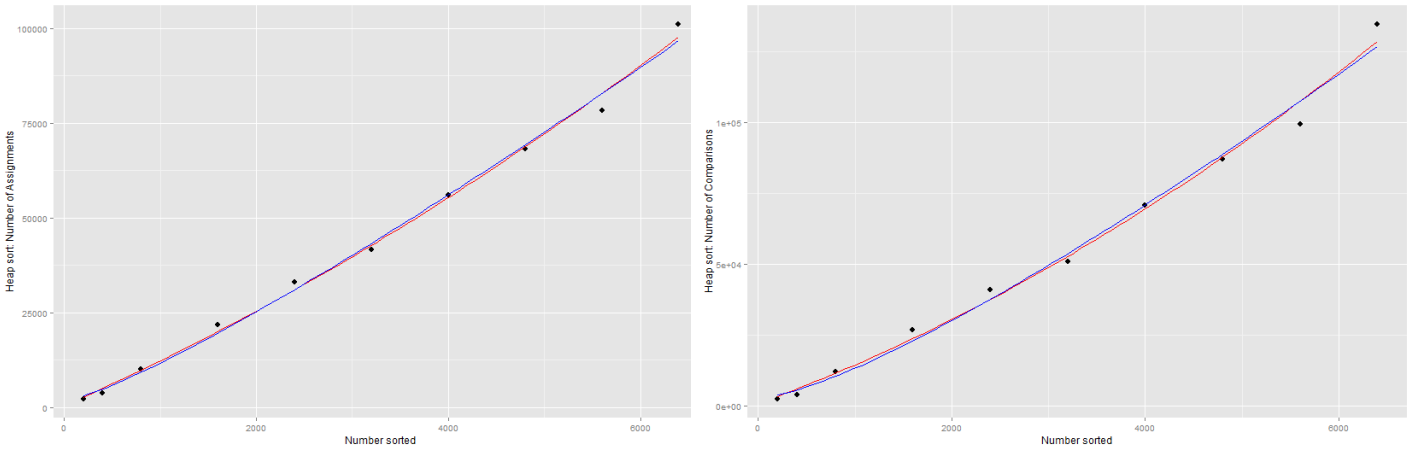
3.2 Selection Sort

The selection sort at first glance seems to go against theory in some ways, yet when all the data is observed together we see it does in fact conform to popular theory and more closely fit to a curve of n^2 as seen in Figure 4. The disconnect occurs because while The RSE for assignments more closely matches $nlgn$, the RSE for comparisons more closely matches n^2. This occurs because the RSE for assignments of n^2 is 2.333e-12 and of $nlgn$ is 1.295e-12 meaning it maps on to a graph of nlgn better since the RSE is smaller. Yet, the RSE for comparisons of n^2 is 3.56e-09 while for $nlgn$ it is 467400. This means in terms of comparisons the graph more closely matches the graph of n^2

This conflicts seems to be a problem at first, but it becomes apparent that the graph more closely fits n^2 when you look at the difference between assignments and comparisons. The comparisons graph fits with $nlgn$ better, but only by a very small margin, while the comparisons graph fits the graph of n^2 better by a very significant amount. Because of this it is seen that selection sort does in fact map better to n^2.

3.3 Heap Sort

The heap sort was interesting because it did not match popular theory in practice This is because the RSE for the assignments of n^2 was 2488 and for $nlgn$ was 2818, meaning the assignments graph more closely matched the graph of n^2 since its RSE was slightly smaller. On top of this, the RSE for the comparisons of n^2 was 4374 and for $nlgn$ was 5003, meaning the comparisons graph more closely matched the graph of n^2 as well as seen in Figure 5.



(a) Assignments (b) Comparisons

Figure 5: Heap Sort.

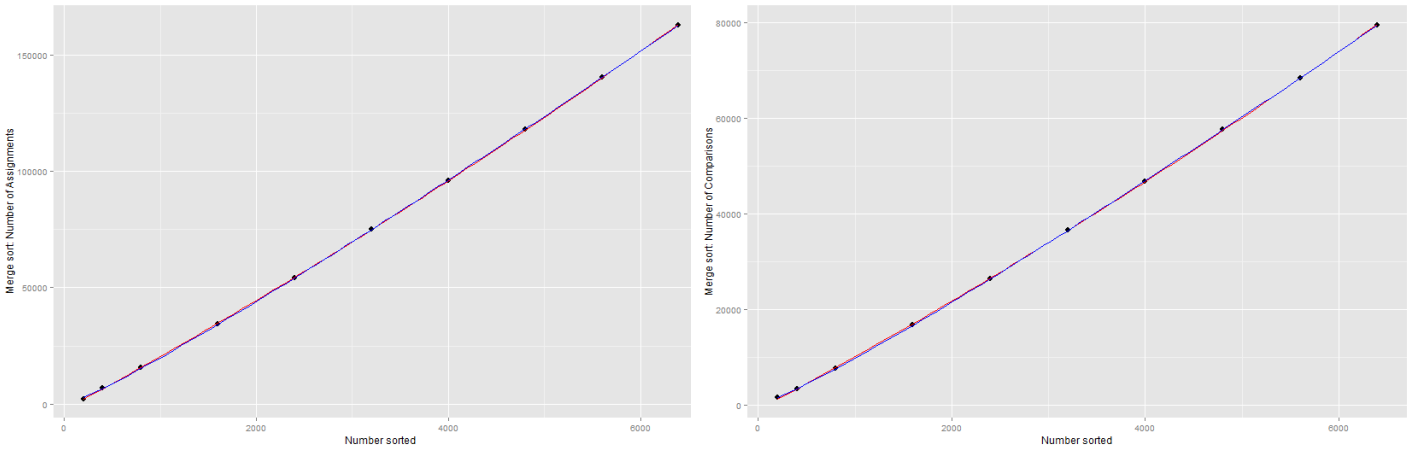
This is interesting because it goes against the popular notion that the heap sort has an effiuecenicy of $nlgn$. The reason for this is probably that the data points were not big enough to show the whole story. In other words, if larger data points were to be added then the trend would be n^2 would start to always be larger. In fact, this makes sense since the RSE is so small for both n^2 and $nlgn$ at the current data points, and also is very close in range, meaning it would be easy for one to surpass the other.

3.4 Merge Sort

Merge Sort was yet another sort that conformed to popular theory and fit best to a line of $nlgn$ as seen in Figure 6. This is because the RSE for the assignments of n^2 was 1.059e+05 and for $nlgn$ was 348, meaning the assignments graph more closely matched the graph of $nlgn$ since its RSE was way smaller. On top of this, the RSE for the comparisons of n^2 was 246.9 and for $nlgn$ was 44.79, meaning the comparisons graph more closely matched the graph of $nlgn$ as well, since its RSE was smaller. In essence, the merge sort acted exactly as expected based on the standards from popular theory.

3.5 Quick Sort

Quick sort did not match up with popular theory in most cases, but this may have just been due to the data sets given. The data collected showed that the RSE for the assignments of n^2 was 20300 and for $nlgn$ was 338600, meaning the graph more closely matched the line n^2 since its RSE was smaller. On top of this, the RSE for the comparisons of n^2 was 7860 and for $nlgn$ was 112500, meaning the graph more closely fit to



(a) Assignments (b) Comparisons

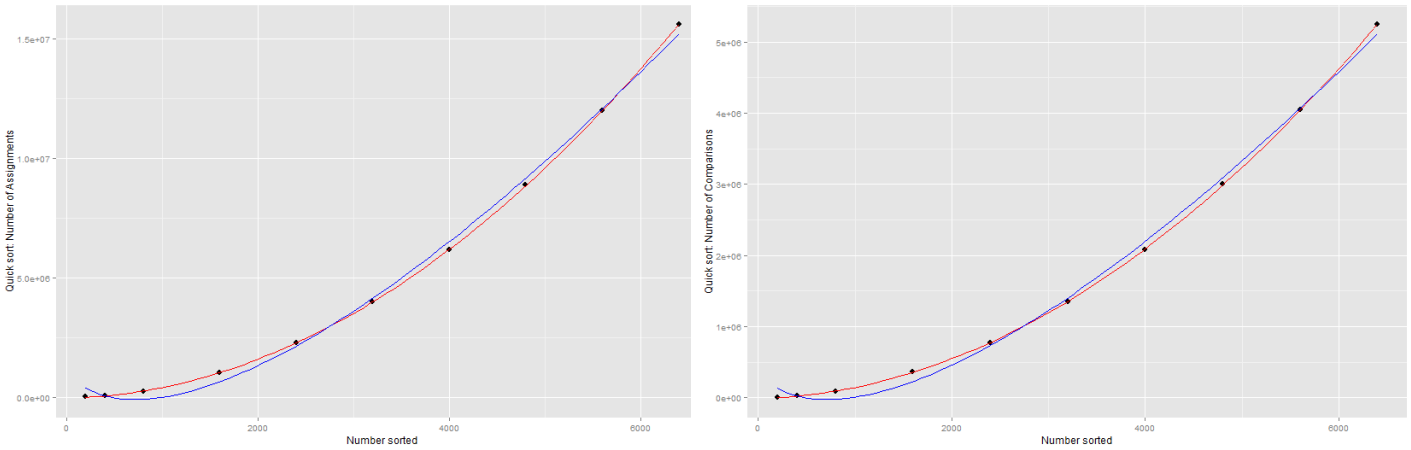
Figure 6: Merge Sort.

the line of n^2 as well since its RSE was smaller as seen in Figure 7.

The weird thing here is that Quick sort is not matching up to its theory based on this test, but that is not telling the whole story. The problem with quick sort is that it used random number generation to guess a median at the start of its execution, meaning each time you run the program you will get a different amount of assignments and comparisons based on how lucky of a guess the program made. Because of this there can be a fairly high variation on the number of assignments and comparisons per run and this skews the data a bit. Yet, on the average of all my runs quick sort was closer to n^2. This is puzzling since based on theory quick sort should average at $nlgn$ with a worst case of n^2. This problem with the data may have occurred because the specific data being fed to the program was good and making it guess a number closer to its worst case scenario. This would have happened because the median being guessed was always far off since the data had a big range of numbers.

4 Conclusion

The experiments conducted showed that for the most part the sorts efficiencies when tested with multiple data points did match up with popular theory about how these sorts should be behaving. Yet, some sorts did act strangely, but not in such absurd ways that trends could not be identified from the way the code was executed that explained why the sort was not conforming to its supposed efficiency. Overall, the experiment showed that while popular theory about sorts and their efficiencies is often correct and is a great baseline for how a sort will perform, in the end the sorts actual ability can



(a) Assignments (b) Comparisons

Figure 7: Quick Sort.

depend a lot on the data that is given and thus how much work the sort has to perform. Popular sorting efficiencies are a great general look at how sorts behave, but do not tell the whole story.