Project 2

October 21, 2018

Scott Wolfe

1 Time Complexity and Analysis of Sorting Algorithms

1.0.1 Introduction:

With thoughtful considerations, the average case run-time of complex sorting algorithms can be brought to a minimum for many if not most inputs. However, even the most complex algorithms with the most efficient average case run-times may not be the best tool for every input. This paper is an analysis of the time complexities of common sorting algorithms for varying inputs. Below we begin with a table outlining the theoretical best, average and worst case time complexities for 6 sorting algorithms. The run-times of these algorithms is then calculated and plotted for arrays of sorted, semi-sorted and randomly generated integers. The run-times for the algorithms below were captured for each array sorted 10 times to get consistent values. Actual run-times are irrelevant as they depend on the test machine hardware and other system variables. Therefore, it is the run-time comparisons we will be discussing.

1.0.2 Table of Theoretical Time Complexities of Sorting Algorithms:

Table 1:

Algorithm	In-Place	Stable	Adaptive	Best Case	Average Case	Worst Case
Insertion Sort	yes	yes	yes	Ω(n)	$\Theta(n^2)$	O(n^2)
Bubble Sort	yes	yes	no	$\Theta(n^2)$	N/A	N/A
Bubble Sort 2	yes	yes	yes	Ω(n)	$\Theta(n^2)$	O(n^2)
Selection Sort	yes	no	no	$\Omega(n^2)$	N/A	N/A
Merge Sort	no	yes	no	Θ(nlgn)	N/A	N/A
Quicksort	yes	no	yes	Ω(nlgn)	Θ(nlgn)	O(n^2)

1.0.3 Experimental Data for the Sorting of Sorted Arrays:

In [96]:

Figure 1:

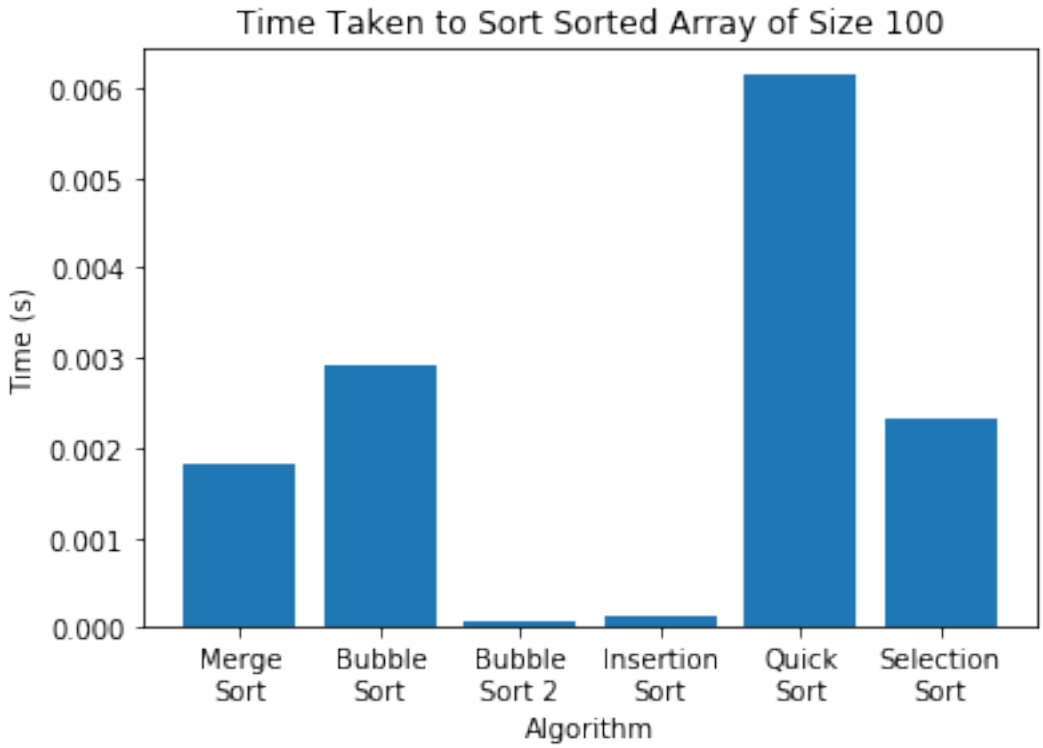


Figure 2:

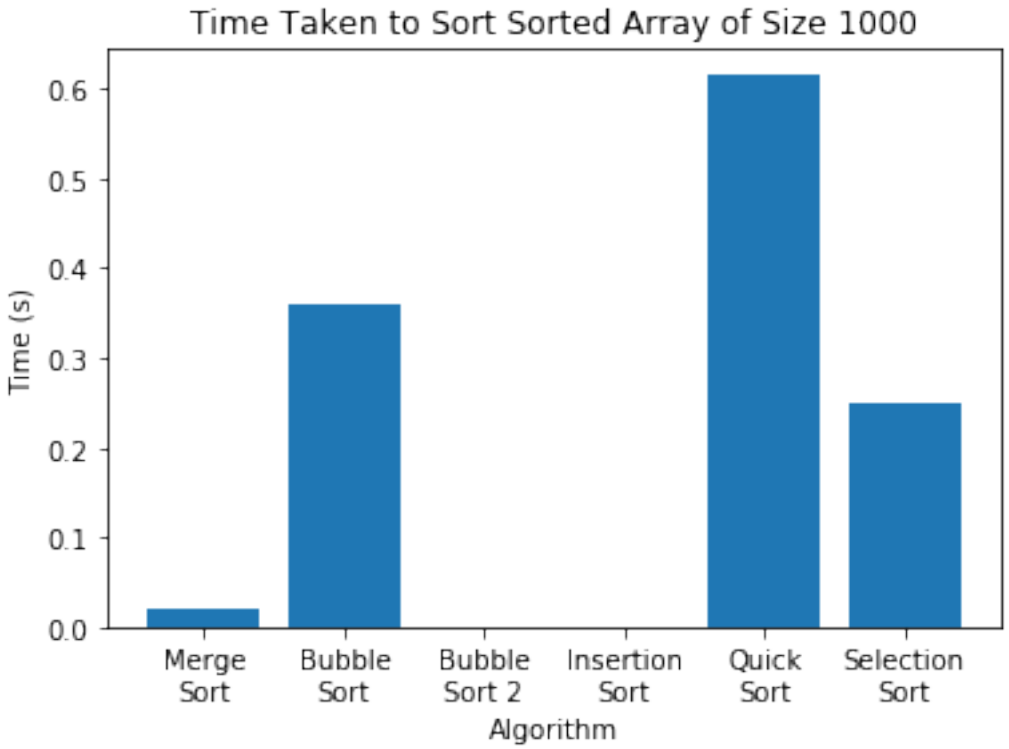
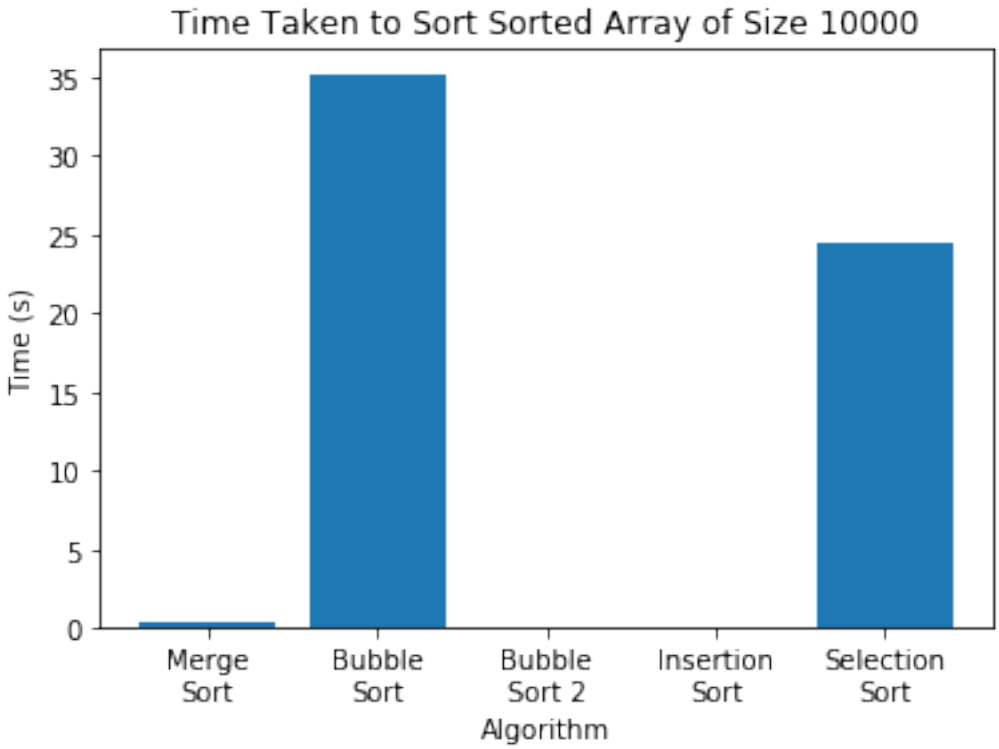

Time Taken to Sort Sorted Array of Size 1000
Time (s)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Merge Sort
Bubble Sort
Bubble Sort 2
Insertion Sort
Quick Sort
Selection Sort
Algorithm

Figure 3:


Time Taken to Sort Sorted Array of Size 10000
Time (s)
0
5
10
15
20
25
30
35
Merge Sort
Bubble Sort
Bubble Sort 2
Insertion Sort
Selection Sort
Algorithm

1.0.4 Experimental Data for the Sorting of Random Arrays:

In [97]:

Figure 4:

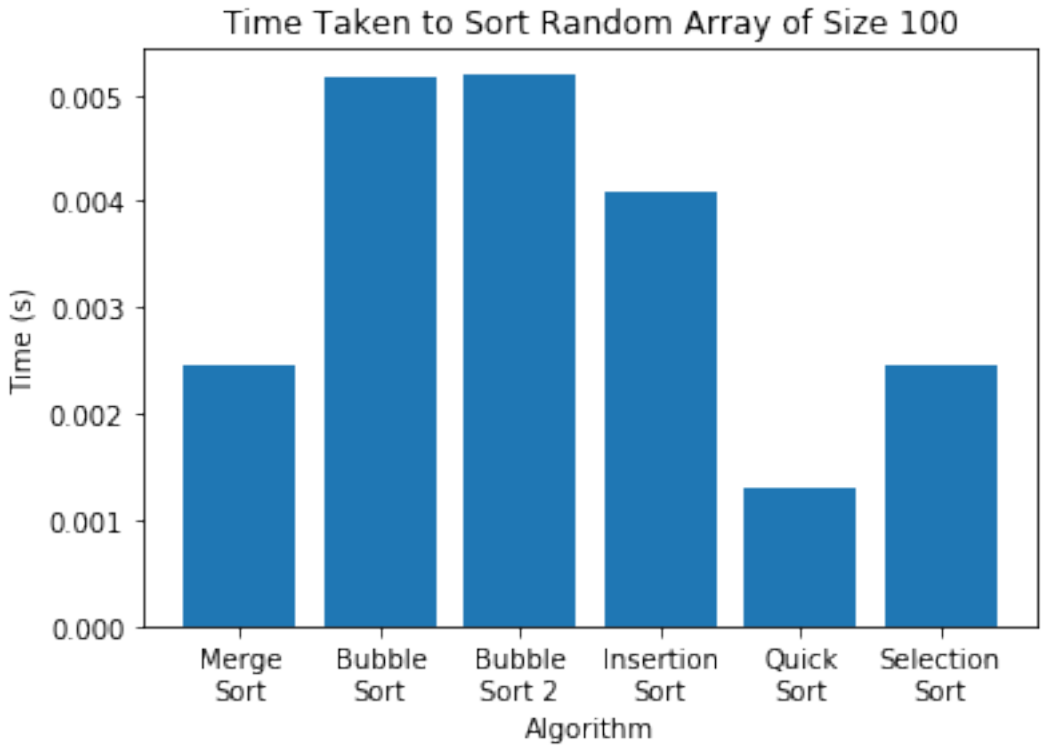


Figure 5:

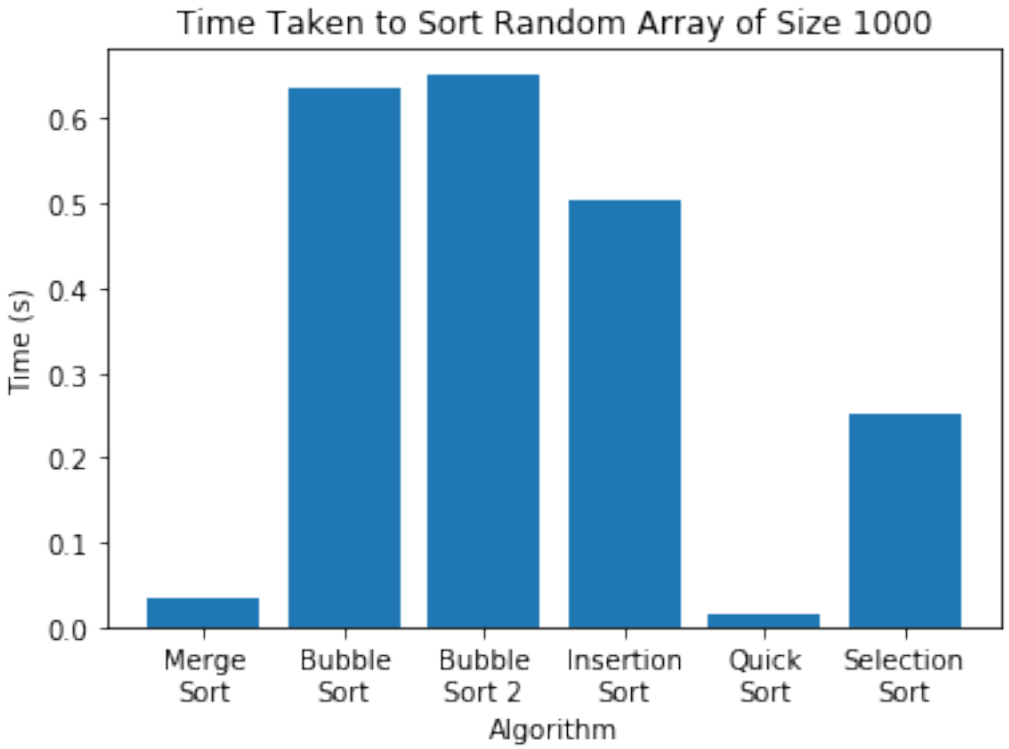
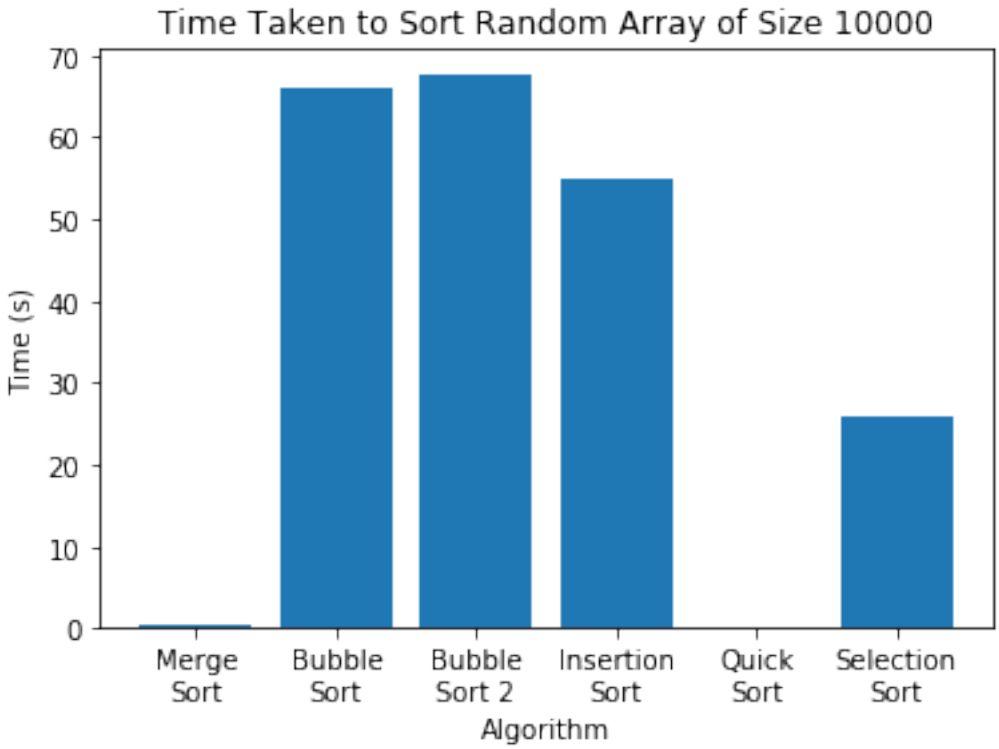

Time Taken to Sort Random Array of Size 1000
Time (s)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Merge Sort
Bubble Sort
Bubble Sort 2
Insertion Sort
Quick Sort
Selection Sort
Algorithm

Figure 6:


Time Taken to Sort Random Array of Size 10000
Time (s)
0
10
20
30
40
50
60
70
Merge Sort
Bubble Sort
Bubble Sort 2
Insertion Sort
Quick Sort
Selection Sort
Algorithm

1.0.5 Experimental Data for the Sorting of Semi-Sorted Arrays:

In [98]:

Figure 7:

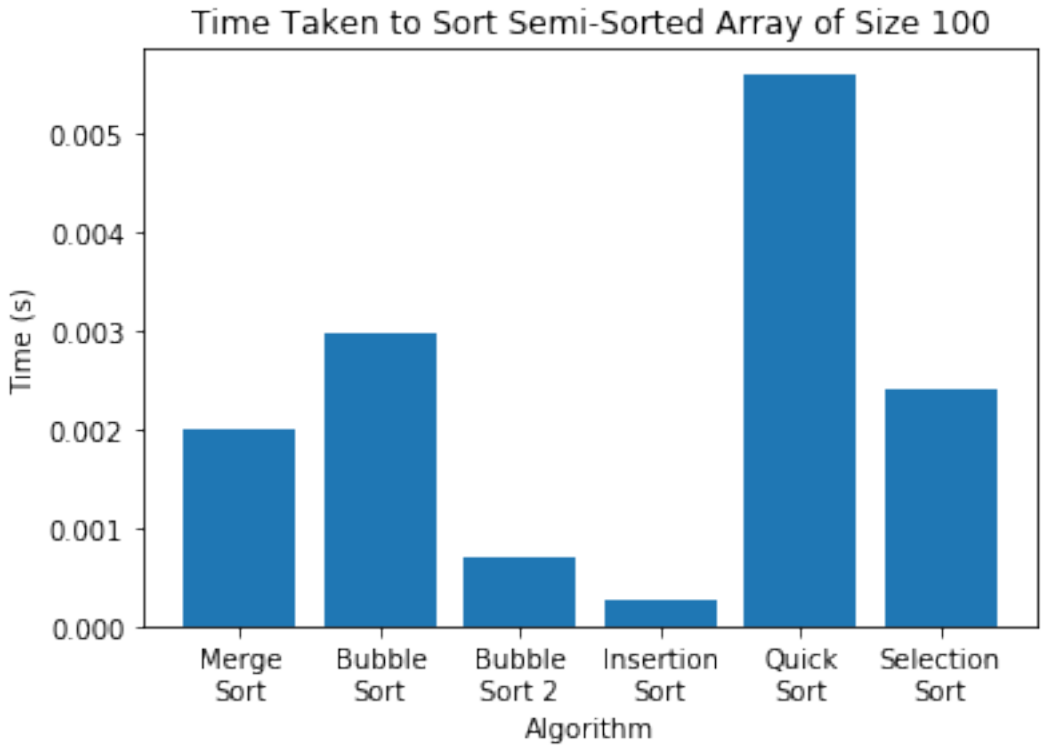


Figure 8:

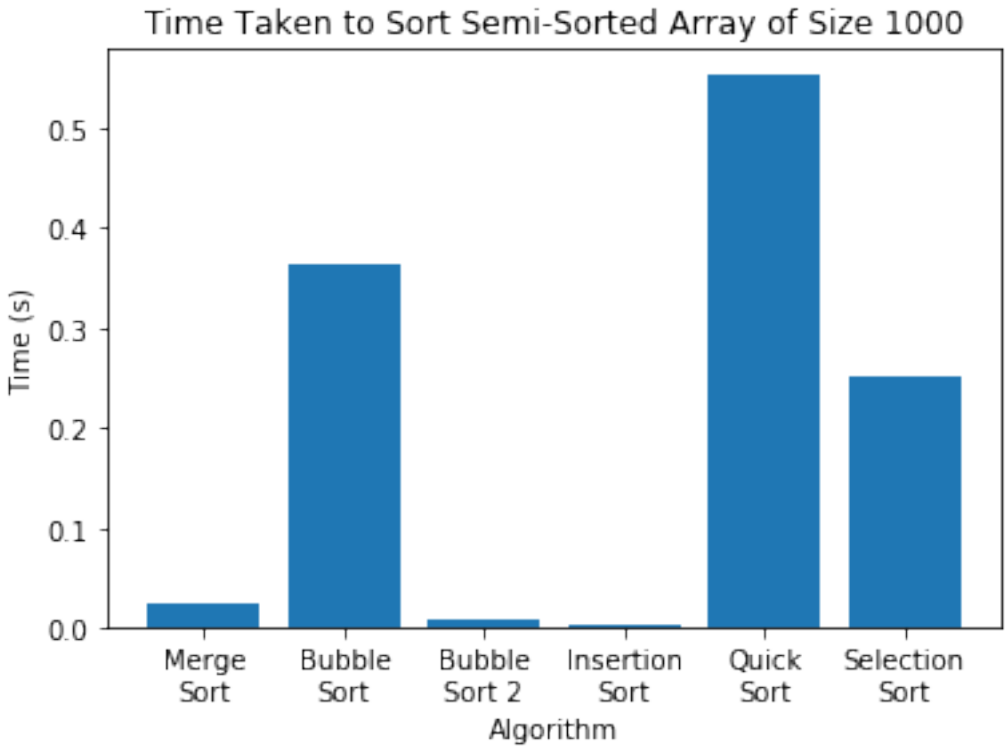
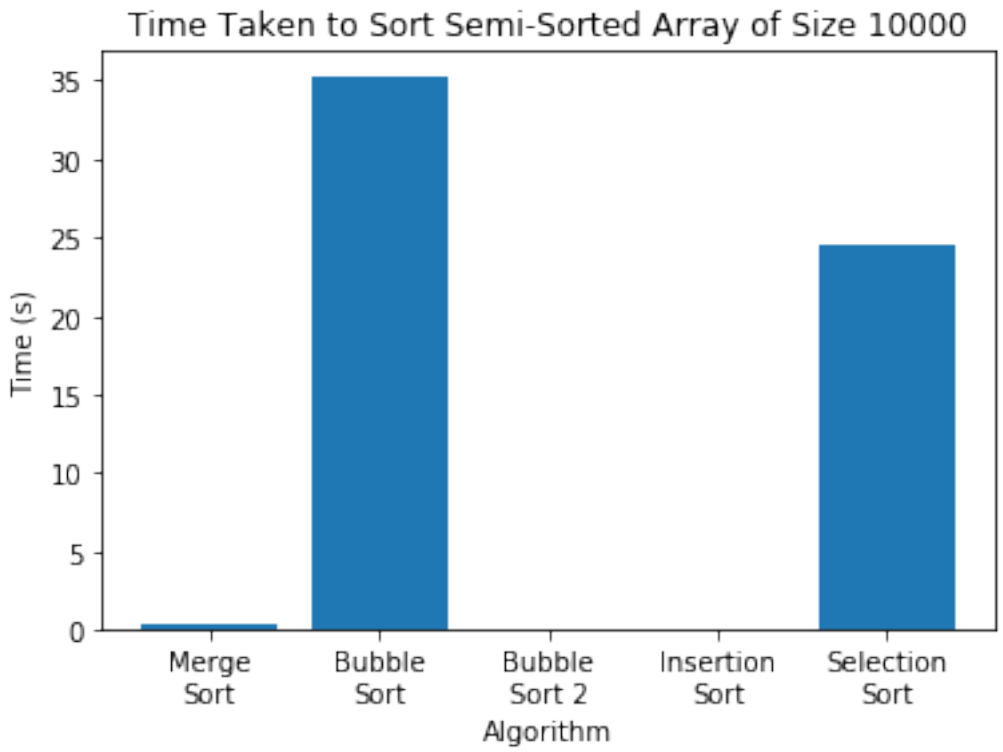

Time Taken to Sort Semi-Sorted Array of Size 1000
Time (s)
0.0
0.1
0.2
0.3
0.4
0.5
Merge Sort
Bubble Sort
Bubble Sort 2
Insertion Sort
Quick Sort
Selection Sort
Algorithm

Figure 9:



1.0.6 Analysis and Discussion:

The worst case run-time of the Quicksort algorithm is on display in figures 1, 2, 7 and 8. At $O(n^2)$ the already sorted array is an example of the worst-case scenario for Quicksort. Without optimization, the Quicksort algorithm pivots away one array component for each iteration causing the maximinum number of recursions. However, the worst-case run-time is not the only important consideration for this recursive algorithm. The toll Quicksort takes on the memory in the worst-case scenario also plays a part on the efficiency, and eventually caused this algorithm to cause a kernal timeout on the test machine and the algorithms run-time was unable to be measured in figures 3 and 9 where the array size was 10,000. By comparison, the Bubble Sort 2 algorithm has linear performance in situtations where the array is sorted, similar to Insertion Sort as seen in the data. Bubble Sort 2 differs from Bubble Sort in that the swaps performed by the algorithm are tallied for each iteration and no swaps indicates an already sorted array and completion of the algorithm. Bubble Sort 2 is adaptive compared to the non-optimized version which always runs at $\Theta(n^2)$ as seen in table 1. The most consistently efficient algorithm according to table 1 also bears out in the data. Merge Sort is a consistently well performing algorithm with a time complexity of Θ(nlgn). By contrast, we can see that Selection Sort is consistently less efficient than Merge Sort for all arrays and maintains $\Omega(n^2)$ performance without memory complexity complications.