CPSC 335 - Project 2

Project Report

Team Members:

Adam Weesner - aweesner@csu.fullerton.edu
David Toledo - davidtv2008@csu.fullerton.edu
Antonio Lopez - antonio_lopez@csu.fullerton.edu

Hypothesis:

We hypothesis that the mergesort will be more efficient for lower input sizes and quicksort will be the most efficient for higher input sizes.

Average Performance:

Mergesort: 0.178348
Quicksort: 0.053312

Range:

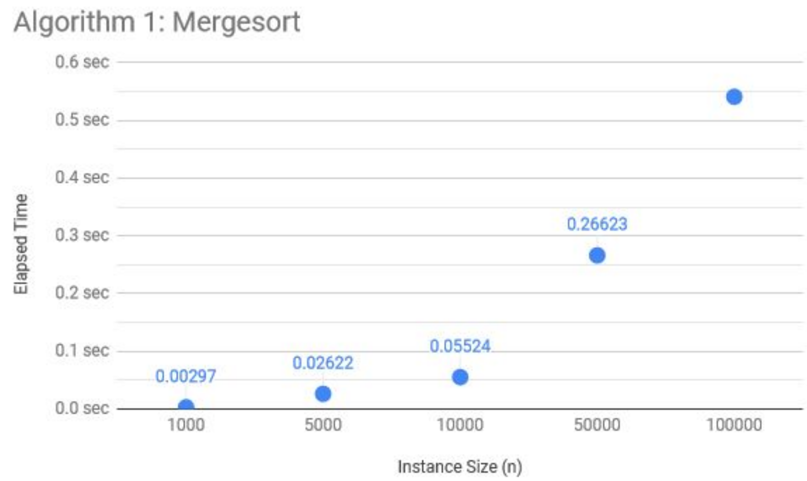
Mergesort: 0.53811
Quicksort: 0.16341

Standard Deviation:

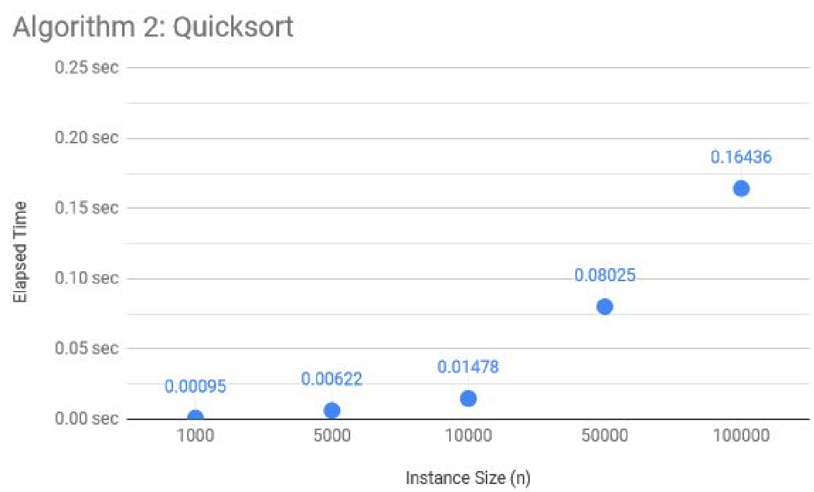
Mergesort: 0.20413412
Quicksort: 0.06245232

Plots:

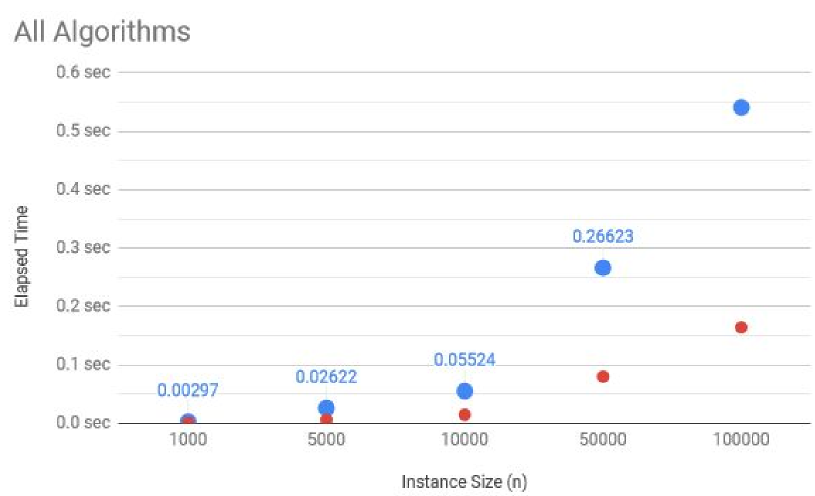
Instance Size (n)	Elapsed Time (sec)
1000	0.00297
5000	0.02622
10000	0.05524
50000	0.26623
100000	0.54108
Average	Standard Deviation
0.178348	0.20413412



Instance Size (n)	Elapsed Time (sec)
1000	0.00095
5000	0.00622
10000	0.01478
50000	0.08025
100000	0.16436
Average	Standard Deviation
0.053312	0.06245232



Blue = Mergesort
Red = Quicksort



Conclusion:

Our hypothesis was half-correct. The quicksort algorithm was indeed more efficient for larger data sets, but it also beast the mergesort algorithm in smaller data sets as well. The average Big O time for the quicksort algorithm is O(nlogn) while the worst-case scenario is $O(n^2)$. The worst case is when data is already sorted and the pivot is the first element. Meanwhile, the Big O for the mergesort algorithm is also O(nlogn) and its worst-case scenario is O(nlogn). Having said all this, we avoided the worst case for the quicksort algorithm by choosing a random pivot. Therefore, our quicksort performed slightly better than its mergesort counterpart in all cases.