# 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:

This experiment will test the following hypotheses:

- Randomization can be used to generate data for testing an algorithm and determining performance.
- Two algorithms of the same efficiency class can have different average running times and different ranges of performance.

## Testing:

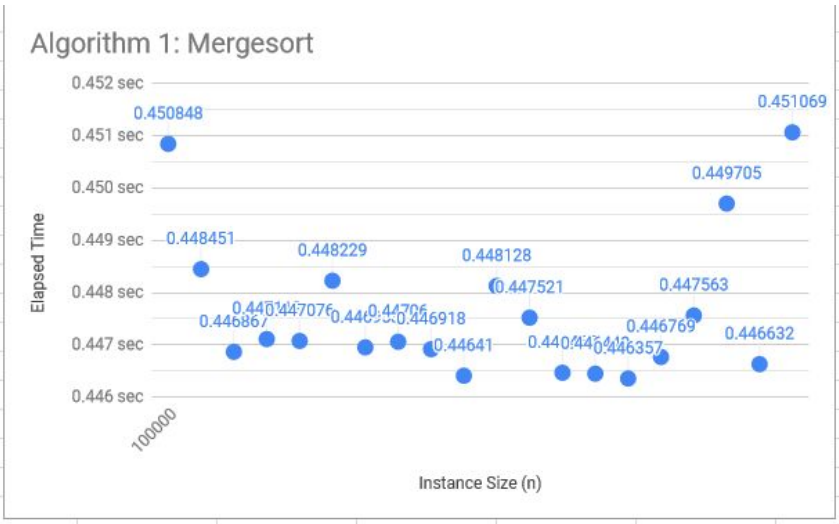
- In order to test the performance of an algorithm it is necessary to run it on a variety of different inputs (instances of the problem).
- Since there are an “infinite” number of possible inputs to an algorithm we need to generate random data sets in order to test a wide range of inputs.

Plots:

## Repeated Size (n = 100000)

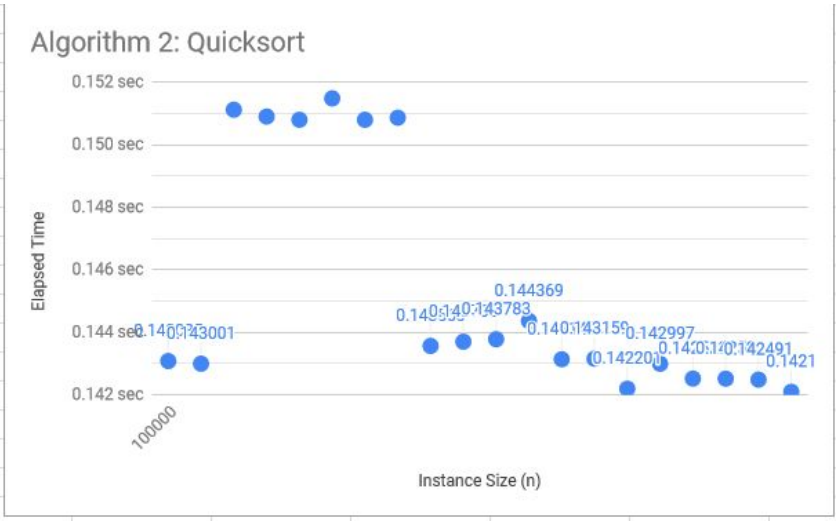
| Instance Size (n) | Elapsed Time (sec) |
|---|---|
| 100000 | 0.450848 |
| 100000 | 0.448451 |
| 100000 | 0.446867 |
| 100000 | 0.447112 |
| 100000 | 0.447076 |
| 100000 | 0.448229 |
| 100000 | 0.446953 |
| 100000 | 0.44706 |
| 100000 | 0.446918 |
| 100000 | 0.44641 |
| 100000 | 0.448128 |
| 100000 | 0.447521 |
| 100000 | 0.446467 |
| 100000 | 0.446449 |
| 100000 | 0.446357 |
| 100000 | 0.446769 |
| 100000 | 0.447563 |
| 100000 | 0.449705 |
| 100000 | 0.446632 |
| 100000 | 0.451069 |

| Average | Standard Deviation |
|---|---|
| 0.4476292 | 0.00137524 |

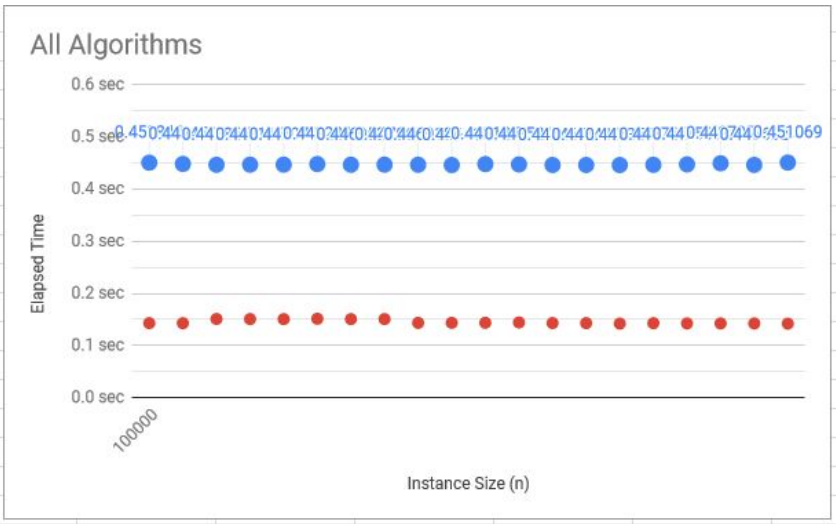



| Instance Size (n) | Elapsed Time (sec) |
|---|---|
| 100000 | 0.143085 |
| 100000 | 0.143001 |
| 100000 | 0.151128 |
| 100000 | 0.150909 |
| 100000 | 0.150806 |
| 100000 | 0.151491 |
| 100000 | 0.150806 |
| 100000 | 0.150873 |
| 100000 | 0.143568 |
| 100000 | 0.143705 |
| 100000 | 0.143783 |
| 100000 | 0.144369 |
| 100000 | 0.143142 |
| 100000 | 0.143159 |
| 100000 | 0.142201 |
| 100000 | 0.142997 |
| 100000 | 0.142523 |
| 100000 | 0.14252 |
| 100000 | 0.142491 |
| 100000 | 0.1421 |

| Average | Standard Deviation |
|---|---|
| 0.14543285 | 0.00368588 |



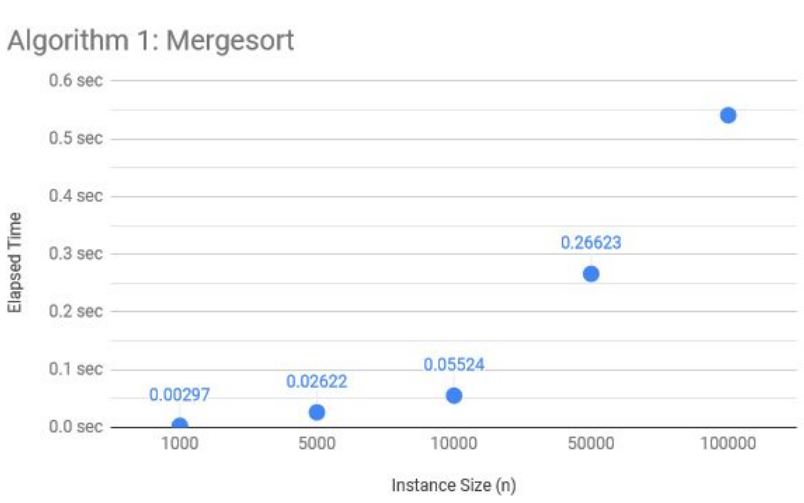
Blue = Mergesort
Red = Quicksort




# Multiple n

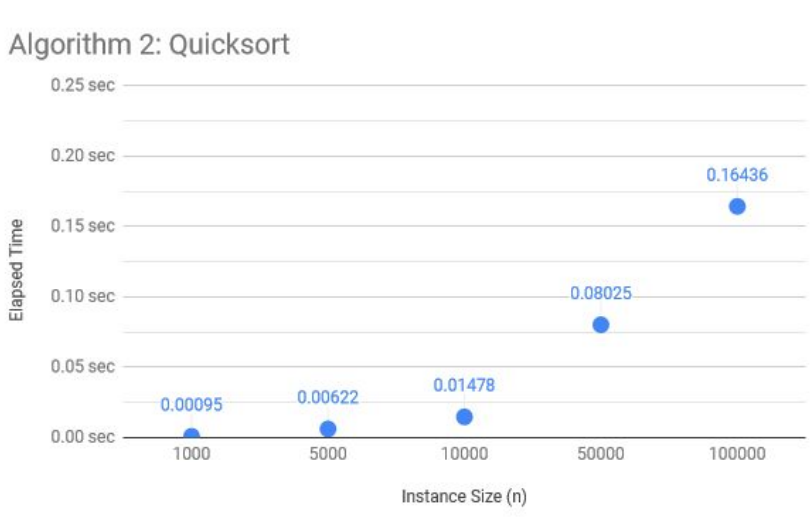
| Instance Size (n) | Elapsed Time (sec) |
|---|---|
| 1000 | 0.00297 |
| 5000 | 0.02622 |
| 10000 | 0.05524 |
| 50000 | 0.26623 |
| 100000 | 0.54108 |

| Average | Standard Deviation |
|---|---|
| 0.178348 | 0.20413412 |

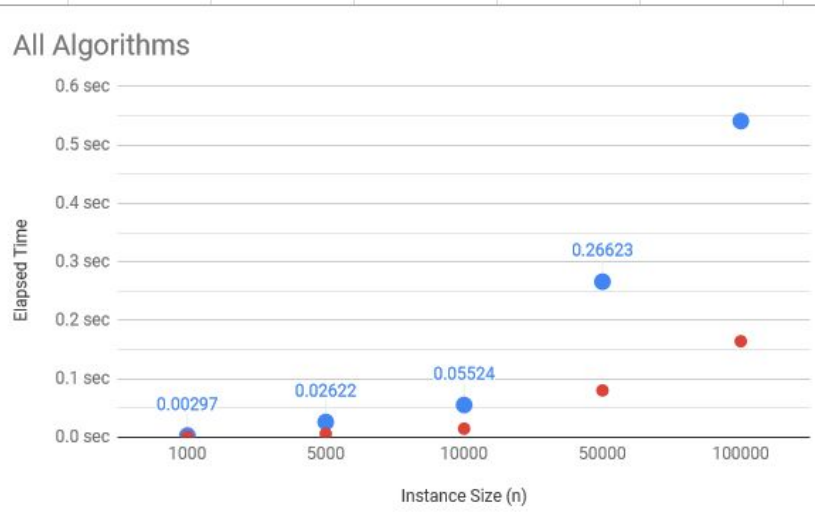


| Instance Size (n) | Elapsed Time (sec) |
|---|---|
| 1000 | 0.00095 |
| 5000 | 0.00622 |
| 10000 | 0.01478 |
| 50000 | 0.08025 |
| 100000 | 0.16436 |

| Average | Standard Deviation |
|---|---|
| 0.053312 | 0.06245232 |




Blue = Mergesort
Red = Quicksort




## Conclusion:

Our hypothesis was correct. The quicksort algorithm was indeed more efficient for larger data sets, but it also bested 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.