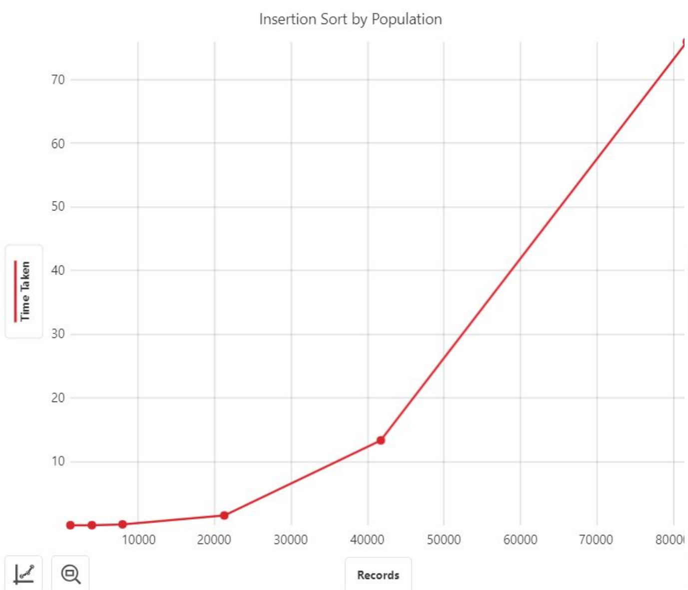
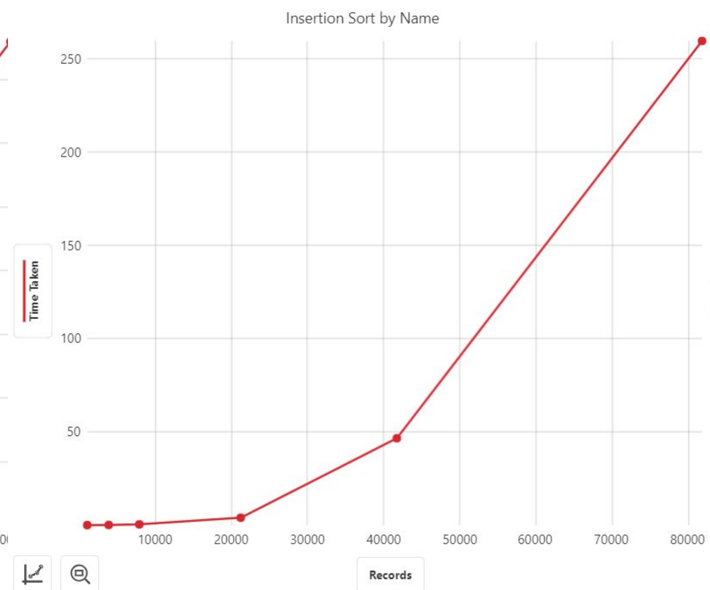
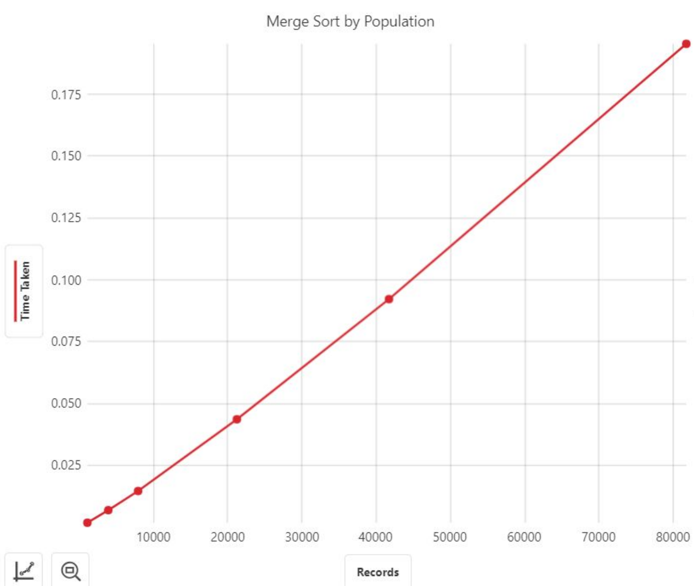
For this project, I implemented and tested three different sorting algorithms - insertion, merge, and quick sorts - on datasets of varying sizes. Data was pulled from the 2010 United States Census. Each algorithm sorted by population count, then by city name. In total, 18 runs were recorded.

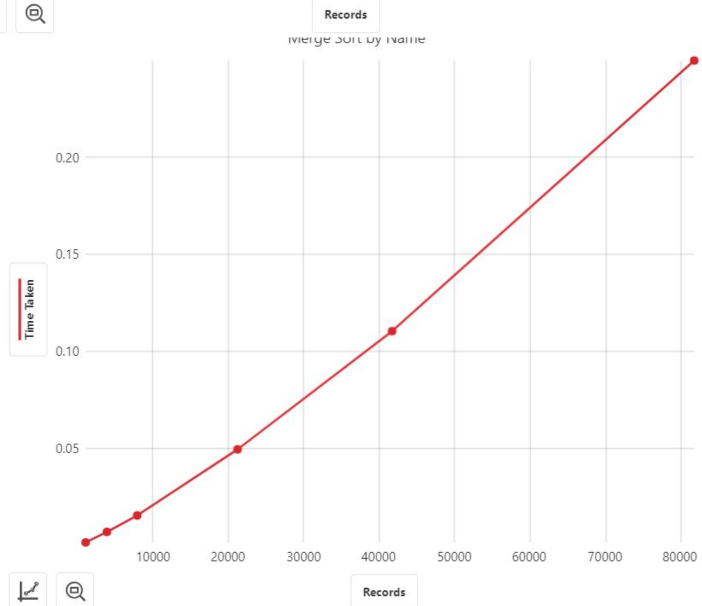
~	InsertionPOP	InsertionNAME	MergePOP	MergeNAME	QuickPOP	QuickNAME
1102	0.00294044	0.0077756	0.00186402	0.00191032	0.000679008	0.000886311
1102	0.00294624	0.007778	0.00184672	0.00192122	0.000698909	0.000930112
1102	0.00295584	0.0077848	0.00183652	0.00192252	0.000718809	0.000845811
BEST	**0.00294044**	**0.007756**	**0.00183652**	**0.00191032**	**0.000679008**	**0.000845811**
3920	0.0294972	0.0919463	0.00695719	0.00742239	0.00304934	0.00366995
3920	0.0305533	0.0932252	0.00693659	0.00732879	0.00310744	0.00374545
3920	0.0300497	0.0915376	0.00691348	0.00735598	0.00307894	0.00400165
BEST	**0.0294972**	**0.0915376**	**0.00691348**	**0.00732879**	**0.00304934**	**0.00366995**
7932	0.165358	0.399224	0.0146852	0.015788	0.00666729	0.00837881
7932	0.164203	0.40258	0.0147263	0.0159348	0.00654748	0.0082935
7932	0.163184	0.399989	0.0146518	0.0158031	0.00699099	0.00905751
BEST	**0.163184**	**0.399224**	**0.0146518**	**0.015788**	**0.00654748**	**0.0082935**
21236	1.58309	3.96882	0.0440514	0.0501428	0.0241481	0.0321503
21236	1.55718	3.96082	0.0436445	0.0498159	0.0232549	0.0281348
21236	1.55675	3.94892	0.0448149	0.050727	0.0216178	0.0332014
BEST	**1.55675**	**3.94892**	**0.0436445**	**0.0498159**	**0.0216178**	**0.0281348**
41712	13.3471	51.0163	0.178349	0.257382	0.0998316	0.202295
41712	19.5643	48.0478	0.273173	0.281231	0.0708421	0.0923041
41712	19.9388	46.5392	0.0922139	0.110645	0.0488115	0.0649654
BEST	**13.3471**	**46.5392**	**0.0922139**	**0.110645**	**0.0488115**	**0.0649654**
81746	75.9124	264.398	0.336439	0.572457	0.247339	0.346933
81746	80.1485	265.97	0.260035	0.320841	0.158277	0.224567
81746	83.9589	259.439	0.1954	0.249902	0.115406	0.18513
BEST	**75.9124**	**259.439**	**0.1954**	**0.249902**	**0.115406**	**0.18513**

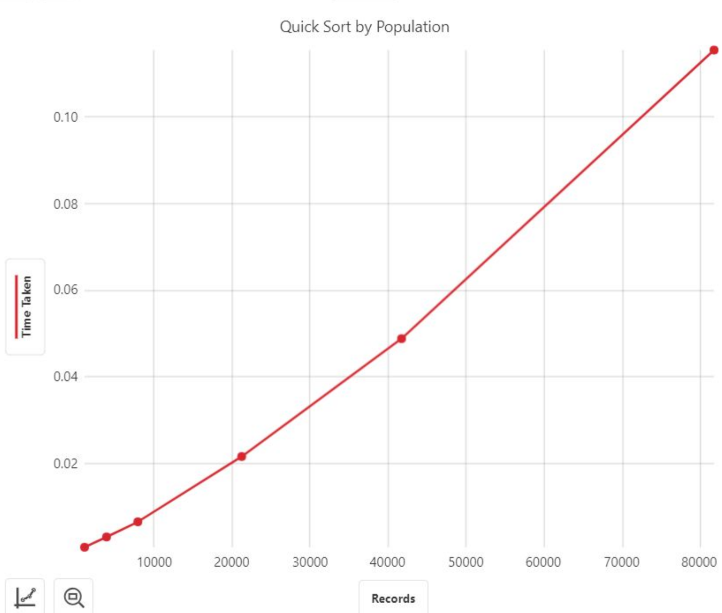
Above is the full table of data from each run. Each white row represents a run, each column relates to one of the 6 sorting operations performed. The green rows are the best runtime for each operation. Below are graphs showing each algorithm's runtime as the sample size increases.

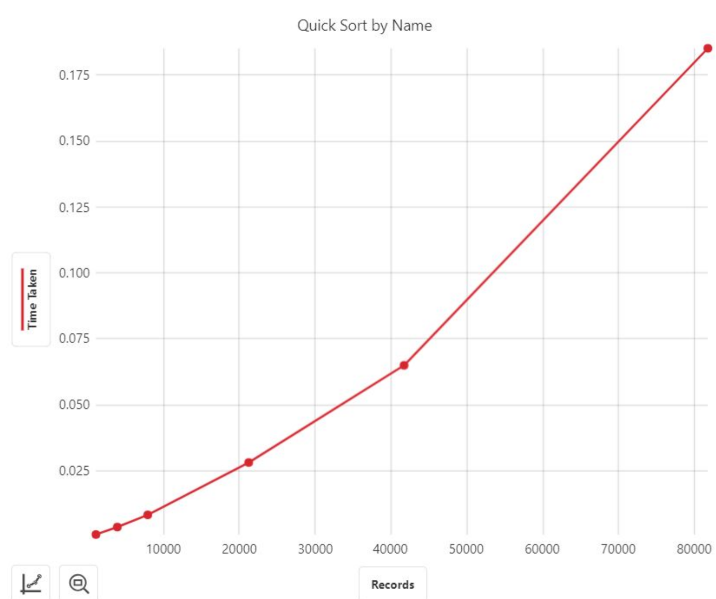

Insertion Sort by Population
70
60
50
40
30
20
10
Time Taken
10000
20000
30000
40000
50000
60000
70000
Records


Insertion Sort by Name
250
200
150
100
50
Time Taken
10000
20000
30000
40000
50000
60000
70000
80000
Records

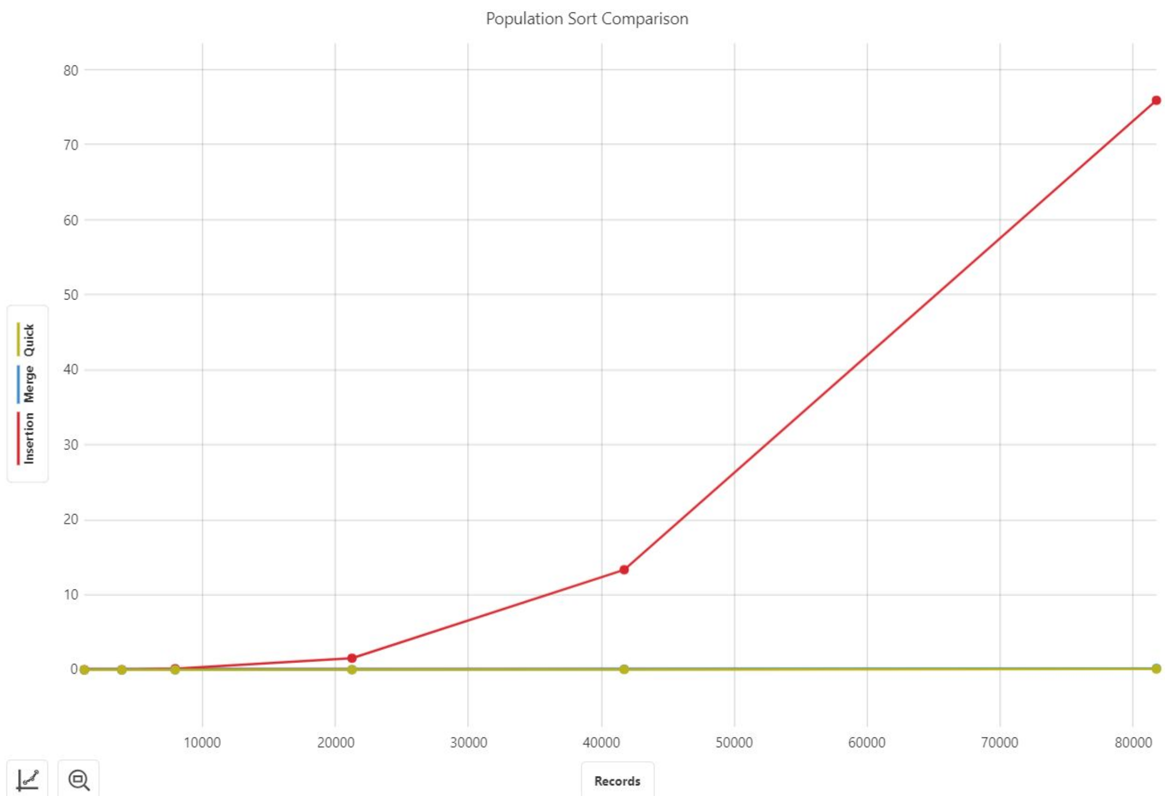

Merge Sort by Population
0.175
0.150
0.125
0.100
0.075
0.050
0.025
Time Taken
10000
20000
30000
40000
50000
60000
70000
80000
Records

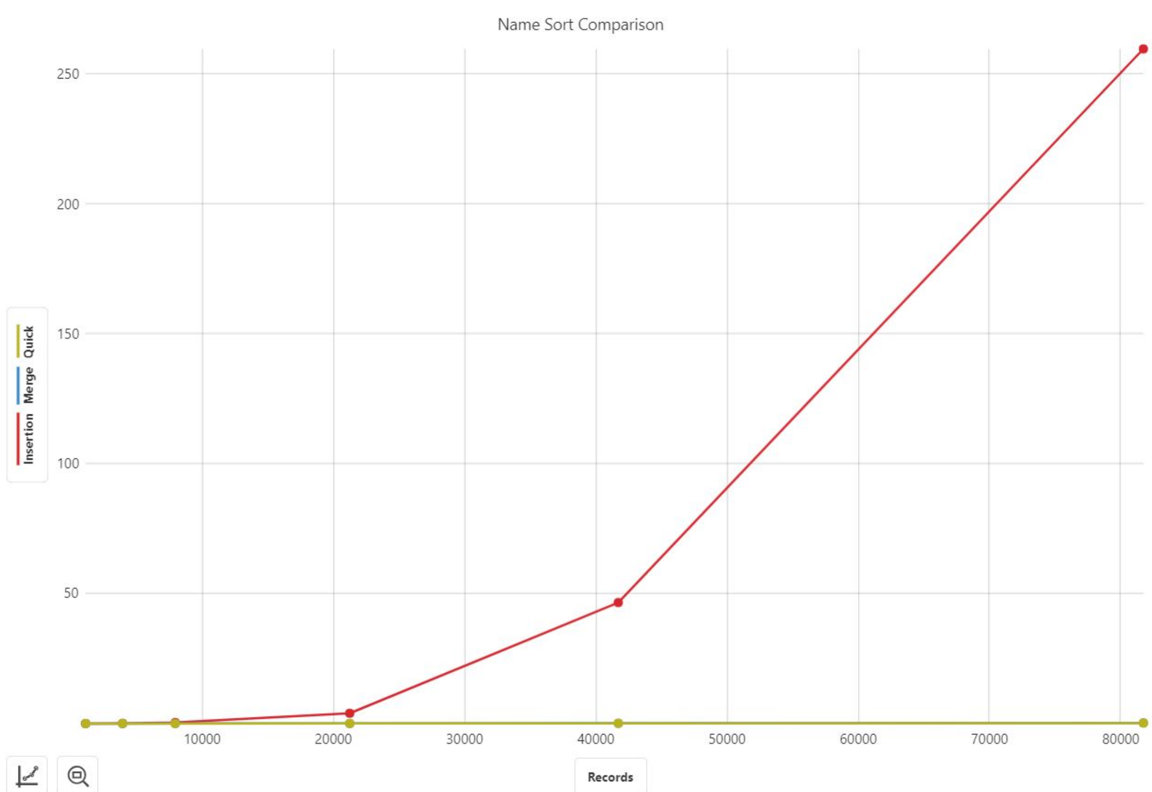

0.20
0.15
0.10
0.05
Time Taken
10000
20000
30000
40000
50000
60000
70000
80000
Records


Quick Sort by Population
0.10
0.08
0.06
0.04
0.02
Time Taken
10000
20000
30000
40000
50000
60000
70000
80000
Records

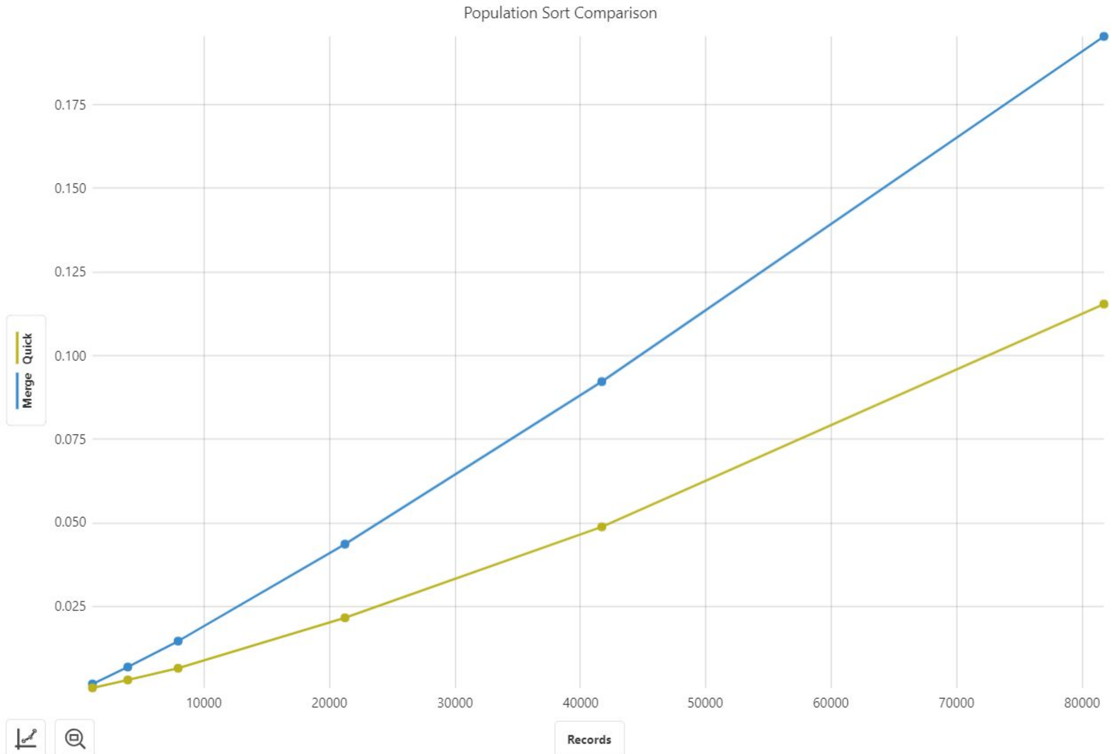

Quick Sort by Name
0.175
0.150
0.125
0.100
0.075
0.050
0.025
Time Taken
10000
20000
30000
40000
50000
60000
70000
80000
Records

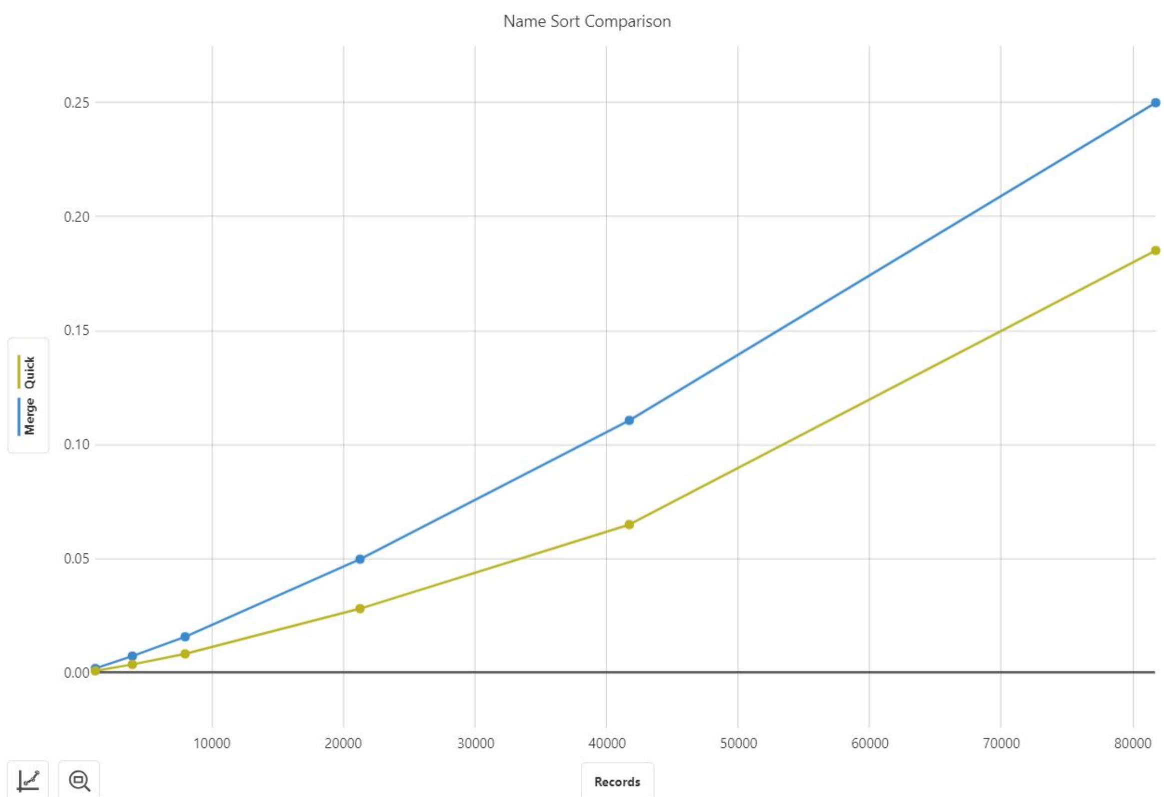
And here we compare the runtimes of each algorithm to each other





If we ignore the data from insertion sort, we can see the comparative runtimes of merge and quick sorts





Clearly, insertion sort is far less efficient than the other two algorithms. Based on the slope of the graph, insertion sort seems to be operating at a runtime complexity of n^2, while merge and quick sorts are close to linear. But that doesn't tell the whole story, as insertion sort's runtime is larger by several orders of magnitude. It appears that the runtime of a single operation on *n* is greater than a comparable operation in one of the other sorting algorithms.

In comparing merge sort and quick sort, it is clear that quick sort performed better. But it should be noted that the growth of merge sort as *n* increases is slightly smaller than the growth of quick sort. Also, analysis of the algorithms tells us that merge sort will always perform the same amount of operations, while quick sort is less consistent.

The one advantage of insertion sort is that it does not require recursion to implement. Recursion, while a powerful tool, places a high demand on memory.

In conclusion, insertion sort is very inefficient, but merge and quick sorts are both fast and reliable. Choosing between the two comes down to any expectation of niche situations in which one sort may be less efficient.