Program 2
Austin Dollar
CSCI 311
Professor Judy Challinger
2/20/2020

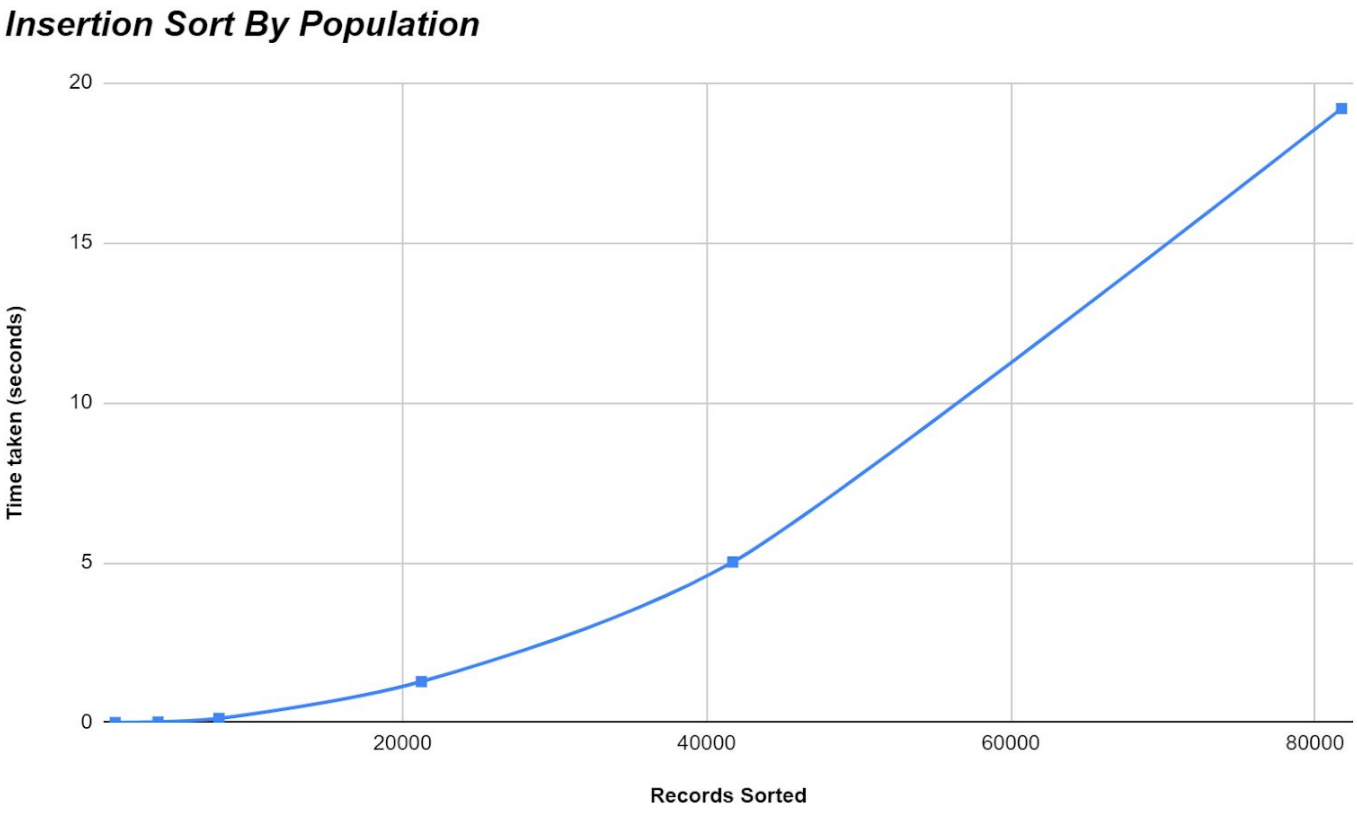
Insertion Sort

Sort by Population Tests:

Insertion Population		
Test Number	Runtime	Amount Sorted
1	0.00202092	1102
2	0.00204452	1102
3	0.00217697	1102
1	0.0209775	3920
2	0.0214929	3920
3	0.021725	3920
1	0.136527	7932
2	0.134425	7932
3	0.132984	7932
1	1.2831	21236
2	1.30737	21236
3	1.3404	21236
1	5.11864	41712
2	5.01814	41712
3	5.05922	41712
1	19.3647	81746
2	19.1938	81746
3	19.4889	81746

Graph of Insertion Sort by Population using best runtime from each input file:

Insertion Population		
	Runtime	Amount Sorted
	0.00202092	1102
	0.0209775	3920
	0.132984	7932
	1.2831	21236
	5.01814	41712
	19.1938	81746

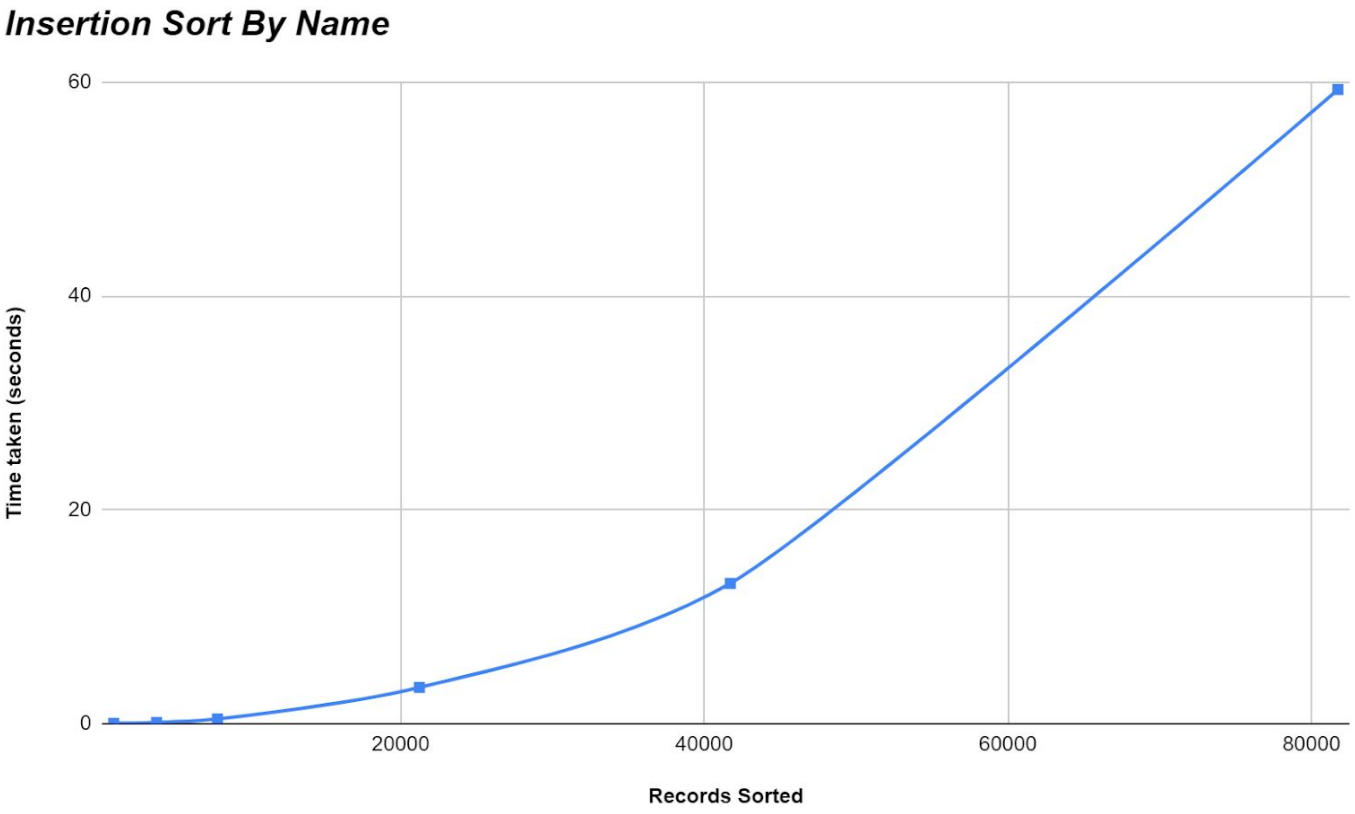


Sort by Name Tests:

Insertion Name		
Test Number	Runtime	Amount Sorted
1	0.00560084	1102
2	0.00580953	1102
3	0.00580201	1102
1	0.0864041	3920
2	0.0849099	3920
3	0.0850636	3920
1	0.420583	7932
2	0.42518	7932
3	0.413341	7932
1	3.47203	21236
2	3.39852	21236
3	3.36491	21236
1	13.0981	41712
2	13.2443	41712
3	13.1095	41712
1	59.3283	81746
2	76.4799	81746
3	65.7615	81746

Graph of Insertion Sort by Name using best runtime from each input file:

Insertion Name		
	Runtime	Amount Sorted
	0.00560084	1102
	0.0849099	3920
	0.413341	7932
	3.36491	21236
	13.0981	41712
	59.3283	81746



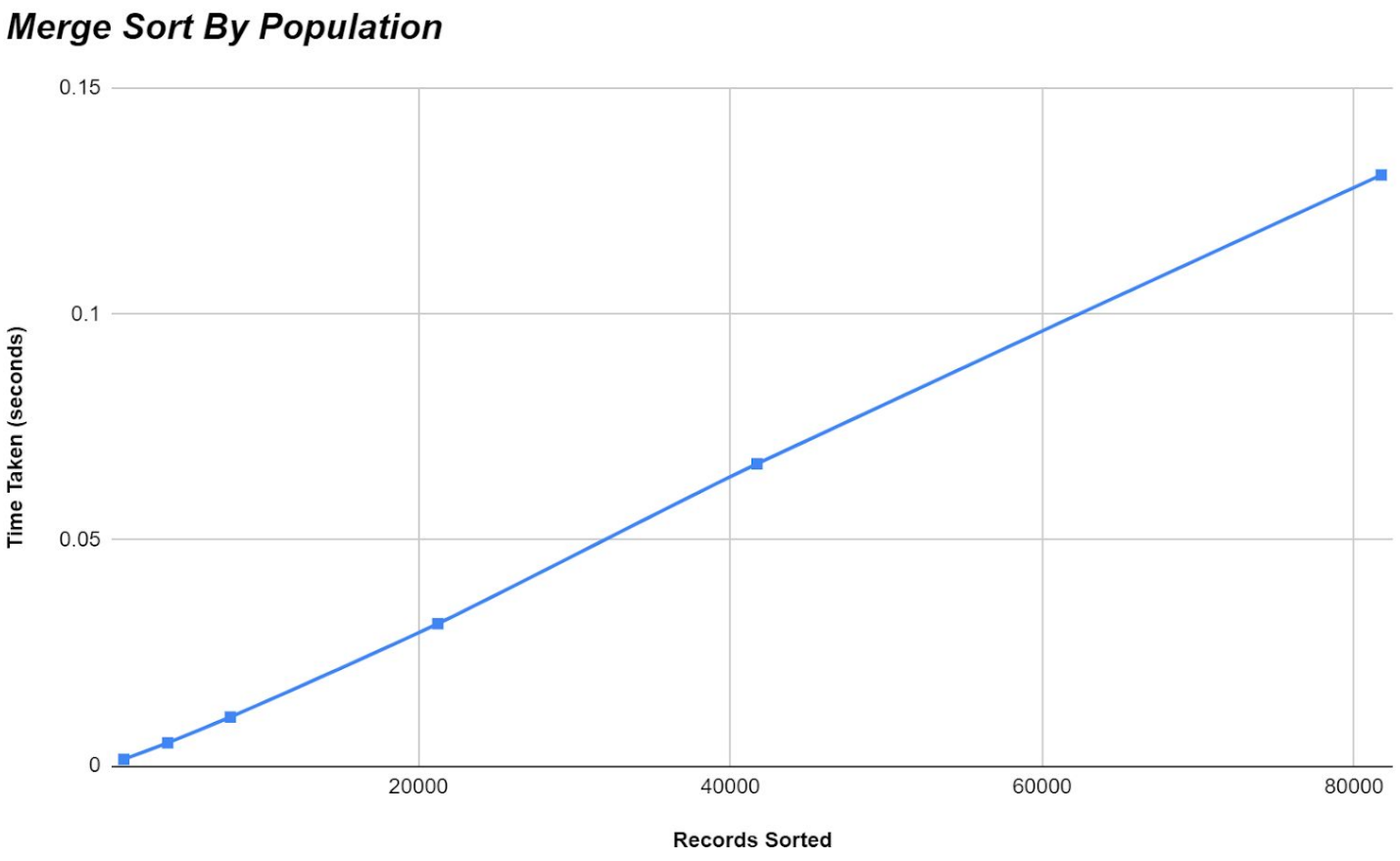
Merge Sort

Sort by Population Tests:

Merge Population		
Test Number	Runtime (s)	Amount Sorted
1	0.00148177	1102
2	0.00156457	1102
3	0.00156512	1102
1	0.00556499	3920
2	0.00514164	3920
3	0.00530456	3920
1	0.0113211	7932
2	0.0109	7932
3	0.0108366	7932
1	0.0384042	21236
2	0.0327392	21236
3	0.0314974	21236
1	0.0722735	41712
2	0.0668847	41712
3	0.0680524	41712
1	0.130788	81746
2	0.144576	81746
3	0.152395	81746

Graph of Merge Sort by Population using best runtime from each input file:

Merge Population		
	Runtime (s)	Amount Sorted
	0.00148177	1102
	0.00514164	3920
	0.0108366	7932
	0.0314974	21236
	0.0668847	41712
	0.130788	81746

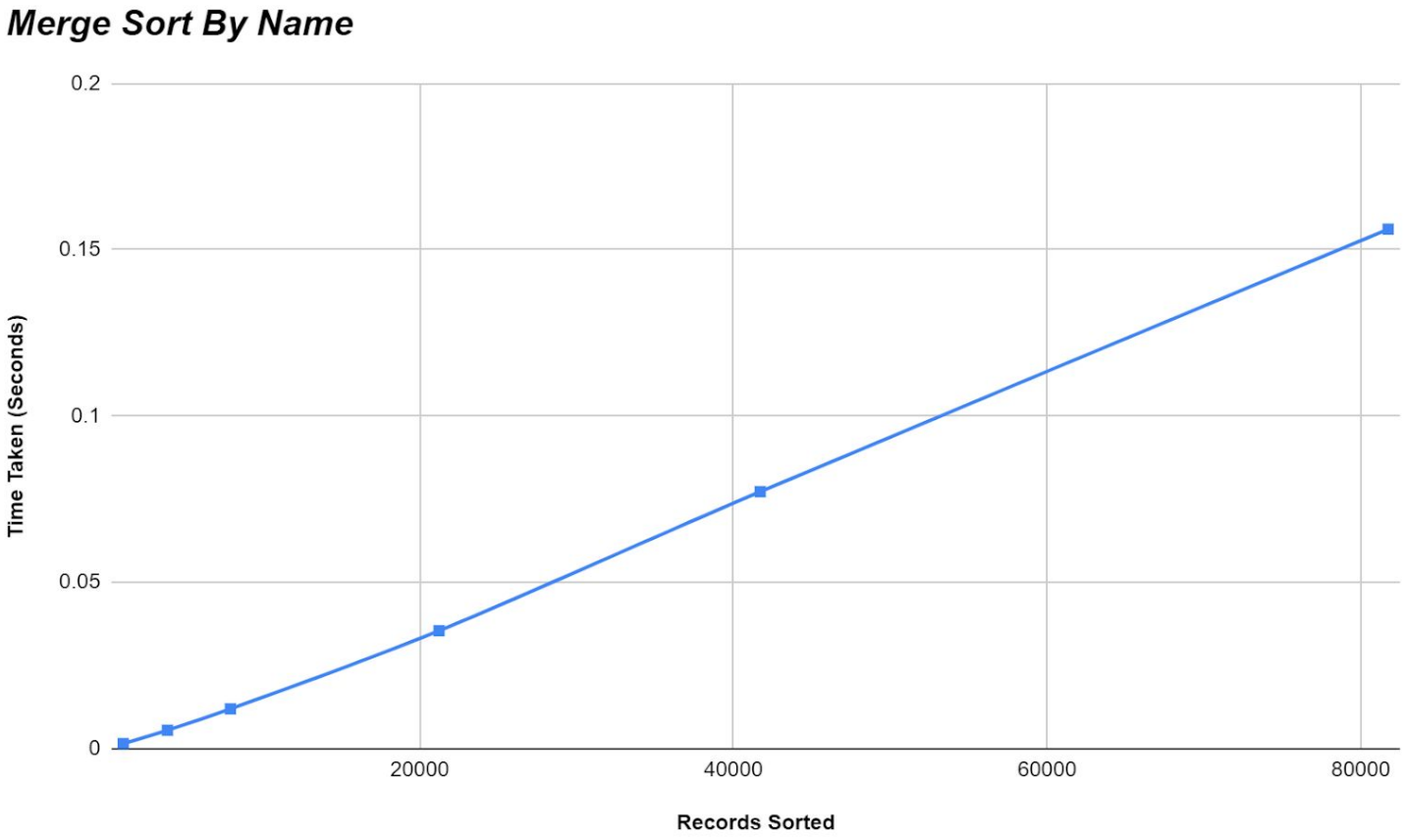


Sort by Name Tests

Merge Name		
Test Number	Runtime (s)	Amount Sorted
1	0.00151519	1102
2	0.00150372	1102
3	0.00152456	1102
1	0.00576089	3920
2	0.00552798	3920
3	0.00560542	3920
1	0.0122713	7932
2	0.0119533	7932
3	0.0127905	7932
1	0.0418396	21236
2	0.0377392	21236
3	0.0354513	21236
1	0.0842953	41712
2	0.0801667	41712
3	0.0772598	41712
1	0.156177	81746
2	0.172968	81746
3	0.168613	81746

Graph of Merge Sort by Name using best runtime from each input file:

Merge Name		
	Runtime (s)	Amount Sorted
	0.00150372	1102
	0.00552798	3920
	0.0119533	7932
	0.0354513	21236
	0.0772598	41712
	0.156177	81746



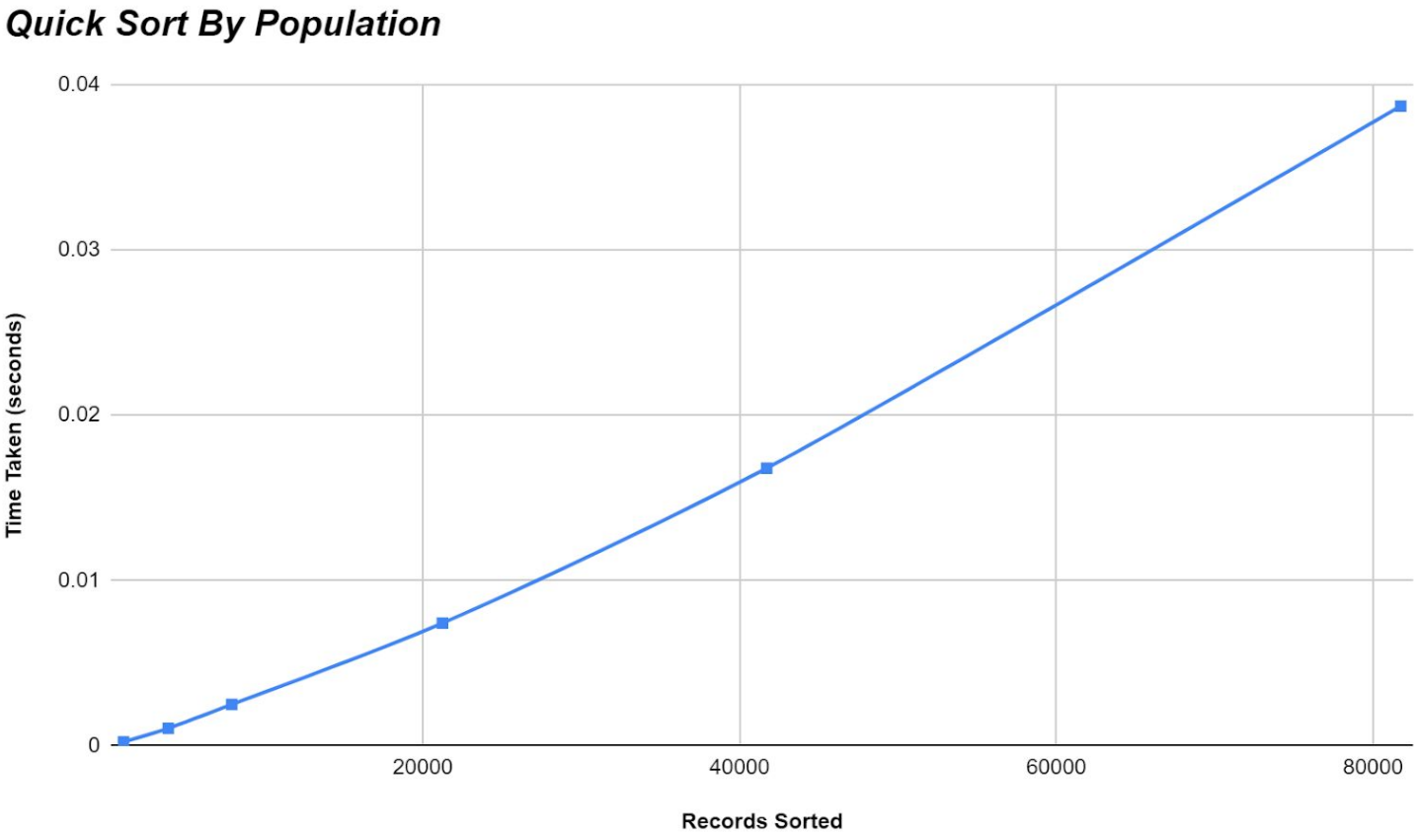
Quick Sort

Sort by Population Tests:

Quick Population		
Test Number	Runtime	Amount Sorted
1	0.00024896	1102
2	0.000309956	1102
3	0.000250926	1102
1	0.001093	3920
2	0.00108351	3920
3	0.00106576	3920
1	0.00251029	7932
2	0.00269633	7932
3	0.00239787	7932
1	0.00760701	21236
2	0.0074235	21236
3	0.00749339	21236
1	0.0181649	41712
2	0.0170659	41712
3	0.0167905	41712
1	0.0386718	81746
2	0.0450033	81746
3	0.0401152	81746

Graph of Quick Sort by Population using best runtime from each input file:

Quick Population		
	Runtime	Amount Sorted
	0.00024896	1102
	0.00106576	3920
	0.00251029	7932
	0.0074235	21236
	0.0167905	41712
	0.0386718	81746

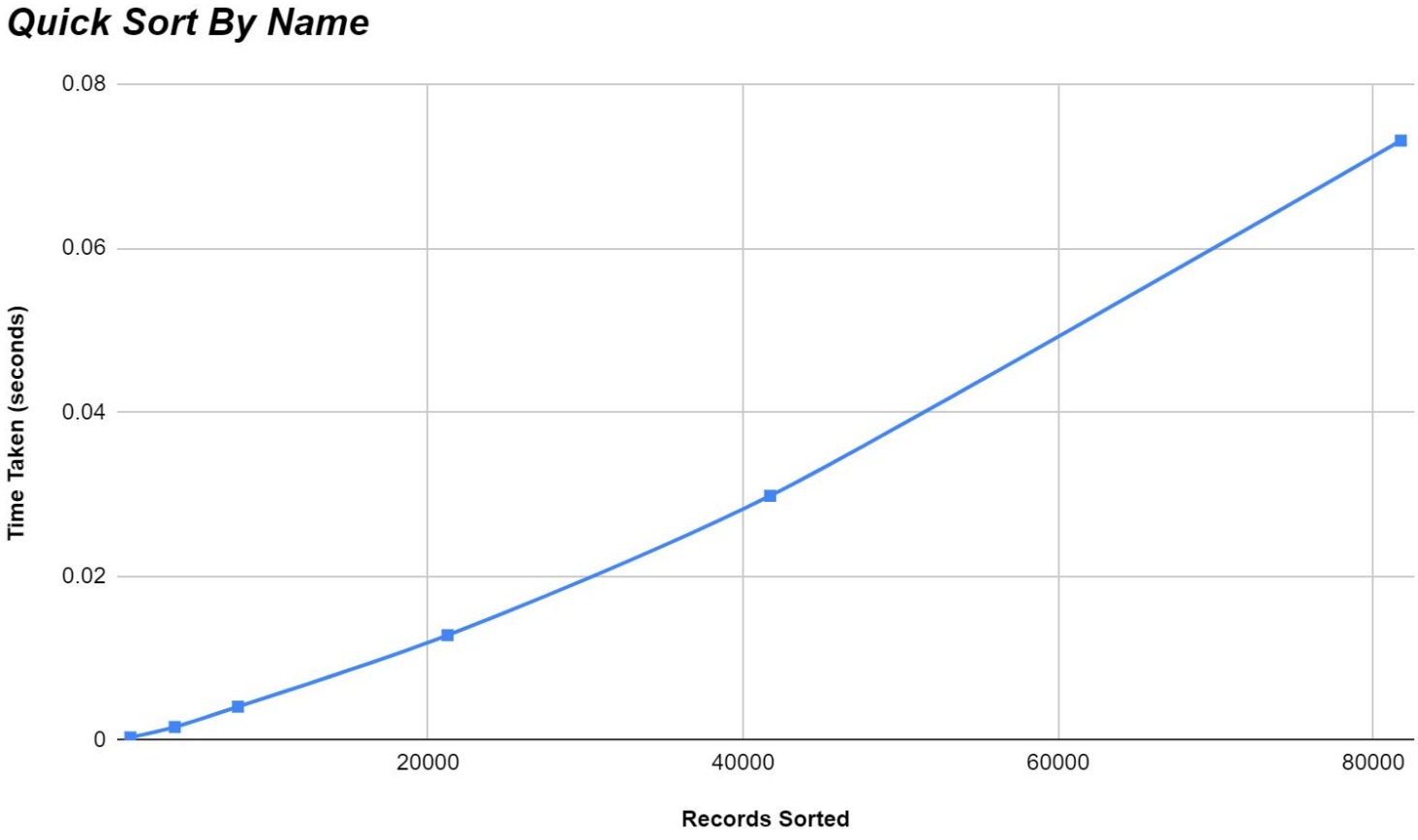


Sort by Name Tests:

Quick Name		
Test Number	Runtime	Amount Sorted
1	0.000376141	1102
2	0.000389338	1102
3	0.000370185	1102
1	0.00183283	3920
2	0.00163694	3920
3	0.00185567	3920
1	0.00441539	7932
2	0.00472121	7932
3	0.00412059	7932
1	0.0135085	21236
2	0.0128318	21236
3	0.0139639	21236
1	0.0320902	41712
2	0.0303209	41712
3	0.0298166	41712
1	0.073118	81746
2	0.085536	81746
3	0.0886013	81746

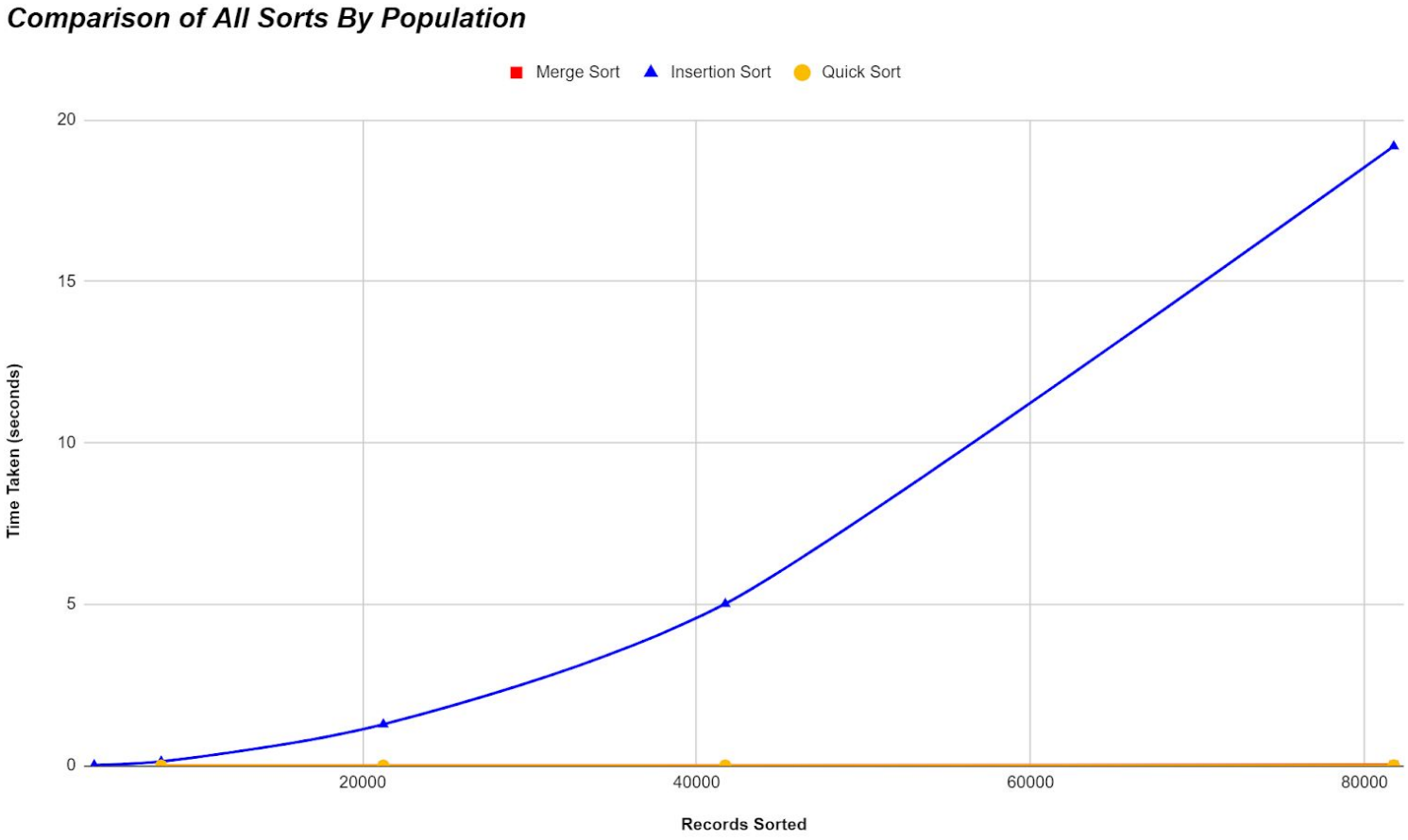
Graph of Quick Sort by Name using best runtime from each input file:

Quick Name		
	Runtime	Amount Sorted
	0.000370185	1102
	0.00163694	3920
	0.00412059	7932
	0.0128318	21236
	0.0298166	41712
	0.073118	81746

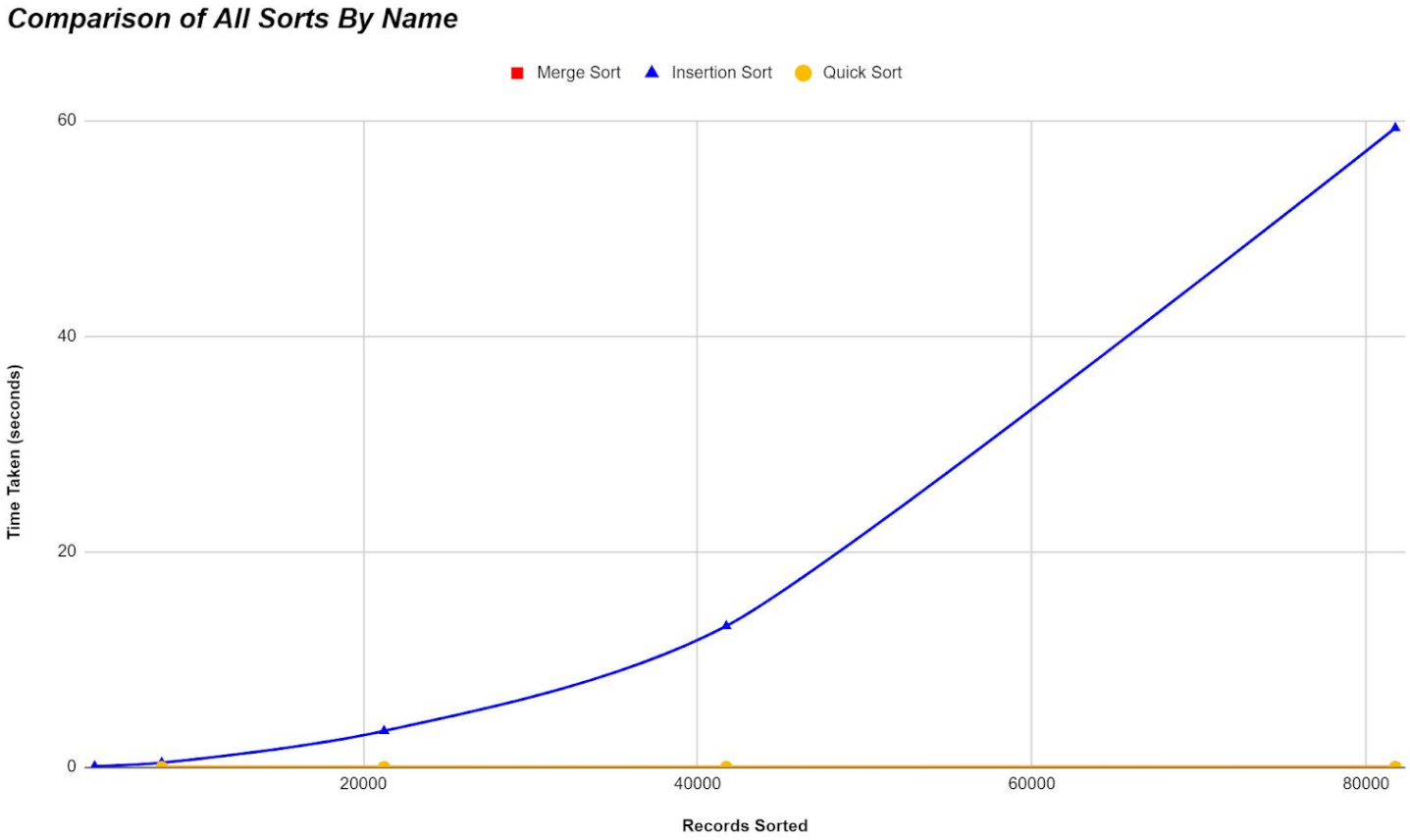


Conclusion

Graph Comparing sorts by population:



Graph Comparing Sorts By Name:



Written Analysis:

The comparison of these three sorting algorithms brings us to some incredible, yet simultaneously predictable conclusions. First of all, when looking at each sorting algorithm individually, they match almost exactly to their corresponding “Big O” notations. The insertion sort, for example, when observing the graph, it becomes obvious that the trend it is following is n^2 . This trend continues for all three sorting algorithms, each of the algorithms, while it is hard to tell when looking at the raw data, when observed graphically, they all match perfectly to their corresponding Big O notations. As for comparing all of these algorithms, it can actually become difficult to graphically, as insertion sort really is that bad. So bad, in fact, that by comparison, both merge sort and quicksort have what appears to be a constant runtime. We know that is not the case, however when they are observed individually. In conclusion, while insertion sort saves space, when dealing with such a large amount of data to sort, such as this, it is exponentially better in terms of runtime to use either quicksort or merge sort. Overall, due to the Big O notation of these sorting algorithms and the matching data gathered, it is obvious that insertion sort is only relatively effective for shorter lists such as the first few input files, meanwhile quicksort is the best overall in terms of runtime, however it does take more space. Merge Sort, meanwhile is a good in between, which would probably be most effective for medium size lists.