Pedro Valdivia
CSCI 311
Judy Challinger
October 4, 2016

Comparison Between 3 Different Sorting Algorithms

Insert Sort, Merge Sort, and Quick Sort are three different sorting algorithms that can be used to sort an n (total number of elements) amount of elements in any order. By analyzing the algorithms for each sorting method, we are able to derive the run time complexities for each algorithm. Through this analysis we know that Insert Sort has a worst case and a best case, Merge sort has a best and worst case that are exactly the same, and Quick Sort has a best and worst case. Insert has a runtime complexity of $T(n) = O(n^2)$ and $T(n) = \Omega(n)$, Merge has a runtime complexity of $T(n) = \Theta(nlgn)$, and Quick has a runtime complexity of $T(n) = O(n^2)$ and $T(n) = \Omega(nlgn)$.

I was given a program that was nearly completed by Judy Challinger. I only had to implement each sorting algorithm in order to complete the program. The program itself took in a file with an n amount of Records and stored them in a vector. Each Record had a city, state and a population. For each sorting algorithm, the program then prints out the original data, the data sorted by the Record's population, and the data sorted by the Record's city name in ascending order. The program also gives us the CPU time taken to complete each individual sorting algorithm which we used to create our table below. The table contains the total CPU time for each individual sorting algorithm with different n amount of Records. Through Table 1 we created 6 graphs that display a visual representation of the runtime complexity of each sorting algorithm as n increases. By analyzing each individual graph, we can see that when n is small insert is faster than both quick and merge sort, but as n (total Records) increases, Insert's CPU time seems to be increasing in a quadratic relationship validating its worst and best runtime complexity. We also see that Merge and Quick Sort's CPU Time seem to be steadily increases in a straight line (nlgn) validating their runtime complexities as well.

It's very clear to say that Merge and Quick sort are more efficient algorithms when it comes to sorting a very large amount of elements when compared to insert sort. In graph 7 we can see that merge and quick sort overlap with each other but if we zoom in closely enough we can see their differences and compare. When we compare, we can see that quick sort is faster than merge sort but not by much. Therefore, we can say that they are very similar and once again validating their runtime complexities.

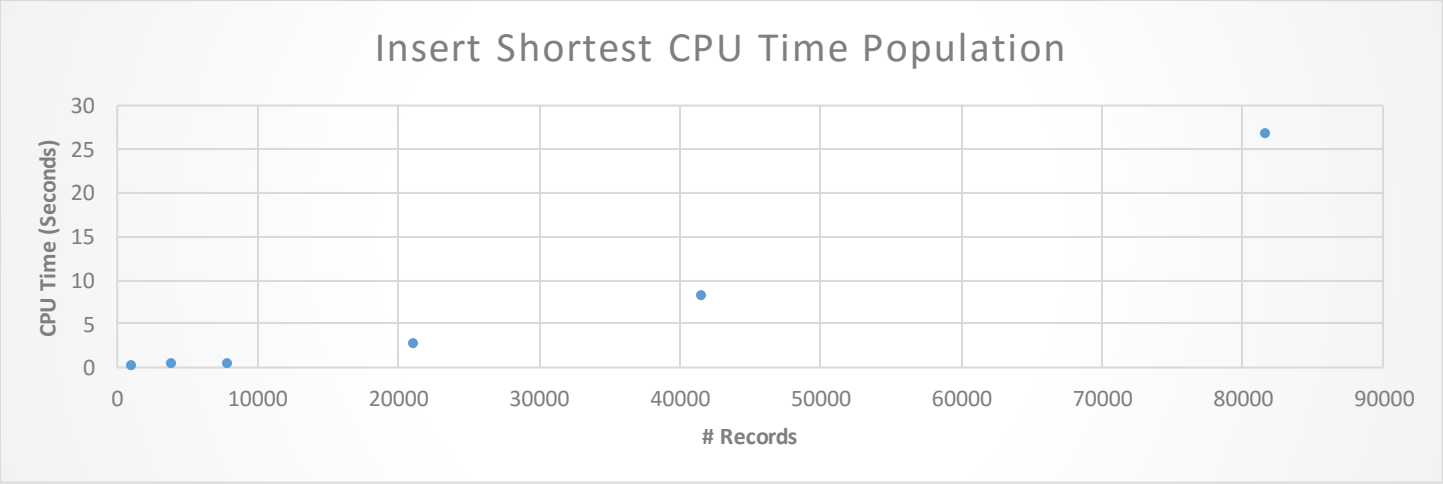
Table 1: CPU Time For Different Sorts at different amounts of Records

# Records	Test #	Insert CPU Time Population	Insert CPU Time City	Merge CPU Time Population	Merge CPU Time City	Quick CPU Time Population	Quick CPU Time City
	1	0.007908	0.027366	**0.00449525**	0.00567683	0.00326482	0.00511478
	2	**0.00760241**	**0.0266602**	0.00457878	**0.00518751**	0.003088325	0.00383325
1102	3	0.00774986	0.0270333	0.00455327	0.00540203	**0.00301914**	**0.00379184**
	4	**0.078457**	**0.377415**	**0.00677629**	**0.00822145**	**0.00464937**	0.00635248
	5	0.0786306	0.393933	0.0169344	0.0206746	0.0115854	0.0149941
3920	6	0.0844199	0.393107	0.0168166	0.0187952	0.00489629	**0.00630317**
	7	0.473823	0.956966	0.0142843	0.0180254	0.0102442	0.013817
	8	**0.194701**	0.709909	0.0146736	0.0185731	**0.00993126**	**0.0137974**
7932	9	0.463365	**0.692298**	**0.0142321**	**0.0176962**	0.00993845	0.0146085
	10	3.19279	**5.45639**	0.0456893	0.0632248	0.0332154	0.0442302
	11	**2.36433**	6.31048	**0.0428371**	**0.0553116**	0.0337749	**0.0424991**
21236	12	2.85606	5.7403	0.0429001	0.0566986	**0.0325308**	0.0437408
	13	8.26559	24.1609	**0.0890918**	0.125417	**0.0750799**	**0.0914715**
	14	10.0385	24.4502	0.100674	0.134671	0.184966	0.23155
41712	15	**8.03295**	**22.9295**	0.0927839	**0.125208**	0.0759421	0.0960111
	16	33.1512	109.05	**0.185069**	**0.263584**	**0.157094**	0.278976
	17	32.2927	105.837	0.27028	0.289685	0.212837	0.2196
81746	18	**26.5499**	**93.8414**	0.201141	0.349481	0.163007	**0.219038**

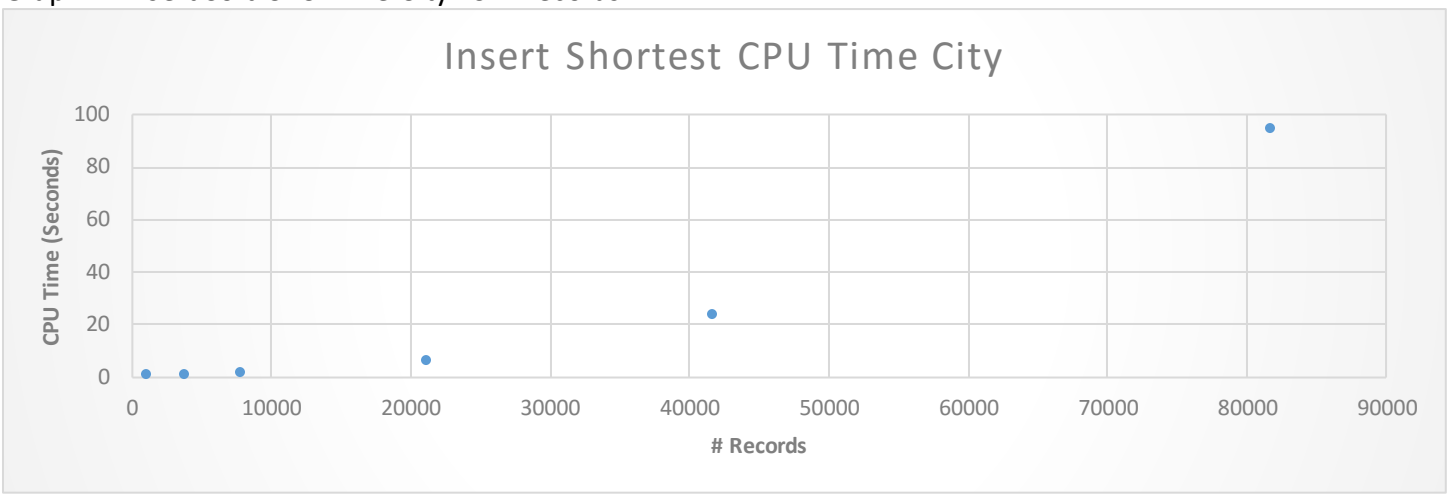
Description: This table shows the total CPU time (in seconds) it takes InsertSort, MergeSort, and QuickSort to finish sorting 6 different amount of elements (Records) inside a vector. Each Record contained a city, state, and population. For each sorting algorithm, I first sorted the vector by the Record's population and then by the Record's city in Ascending Order. The table shows 3 different tests taken for each total number of elements. The bolded time indicates the shortest time of the three tests. We will use these bolded times to create 6 different graphs to show how runtime complexity for each sorting algorithm.

I created two graphs for every sorting algorithm. The first graph is created by plotting the shortest CPU Times (bolded time) taken to sort the vector by the Records population for each different amount of elements. The second graph is created by plotting the shortest CPU Times (bolded Times) taken to sort the vector by the Record's city for each different amount of elements.

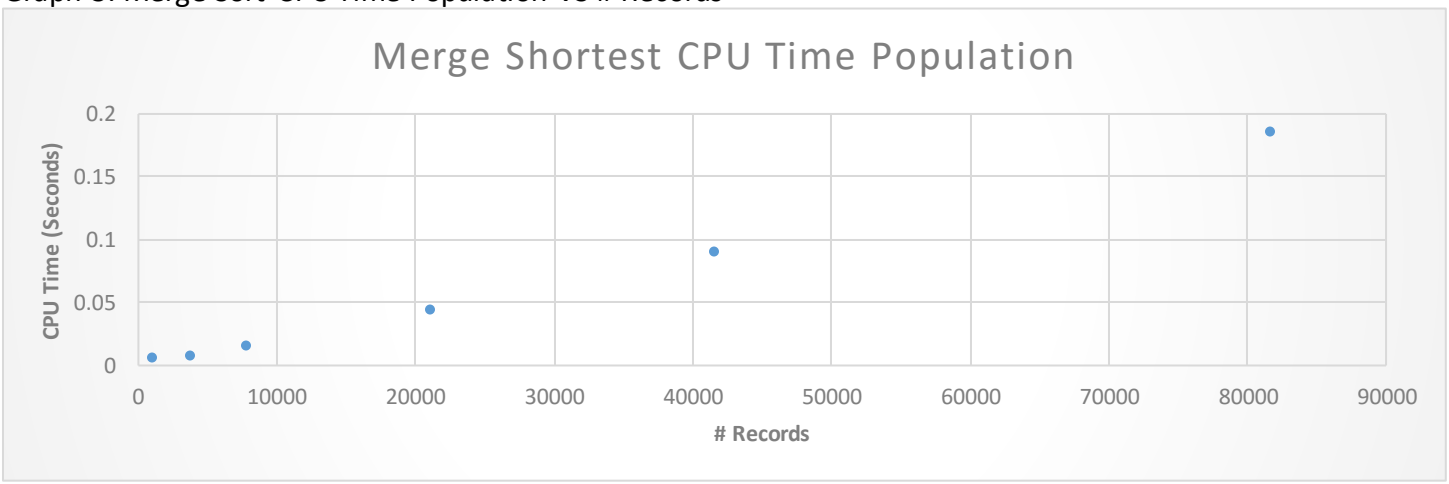
Graph 1: InsertSort CPU Time Population Vs # Records



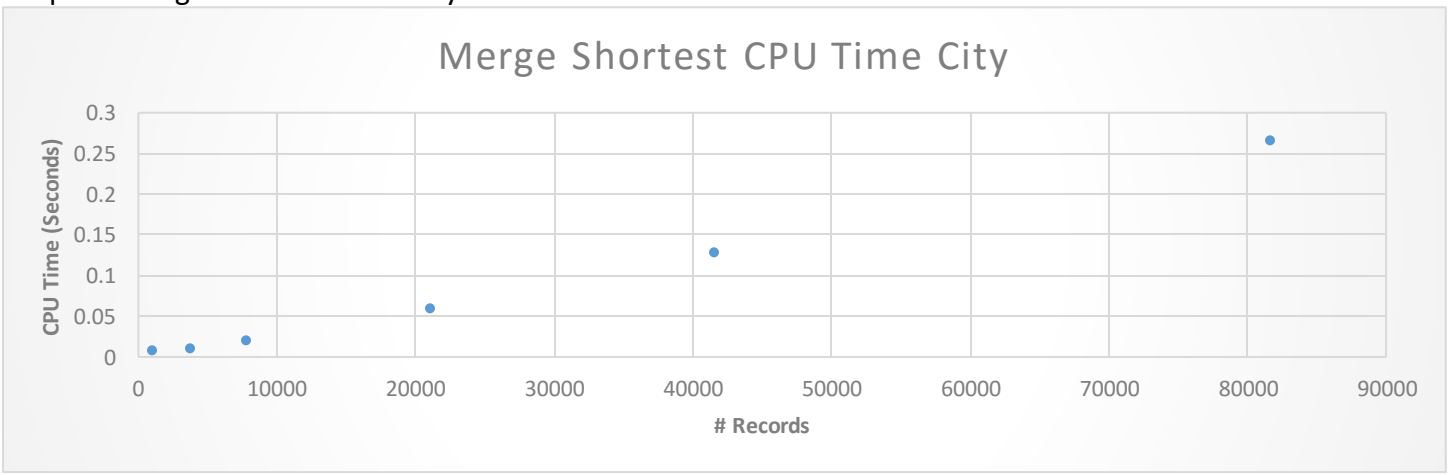
Graph 2: Insert Sort CPU Time City VS # Records



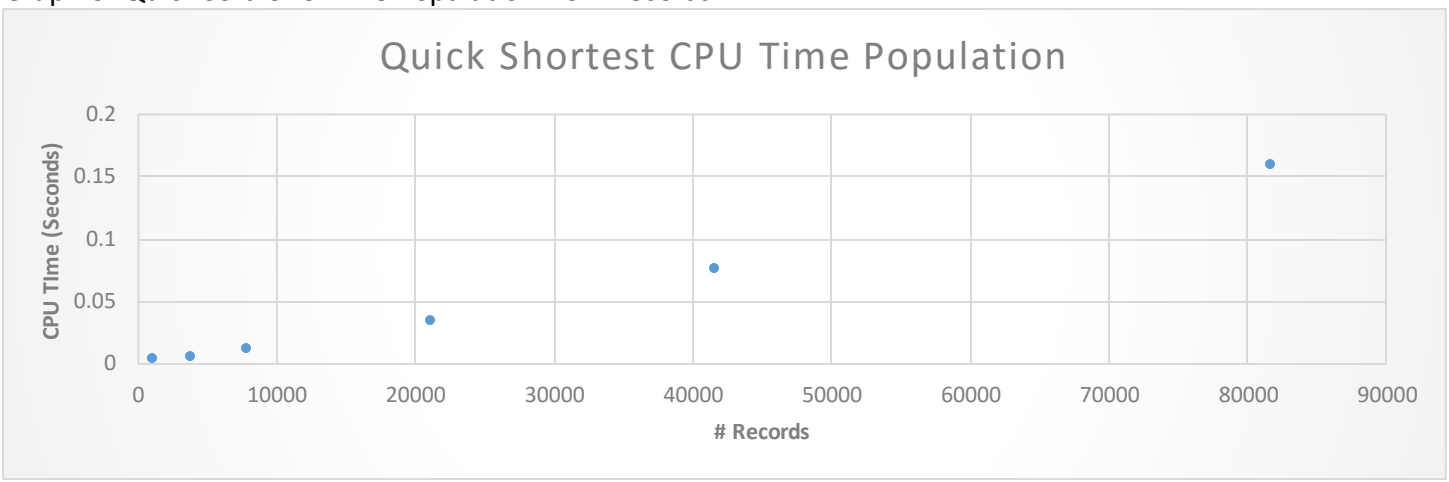
Graph 3: Merge Sort CPU Time Population VS # Records



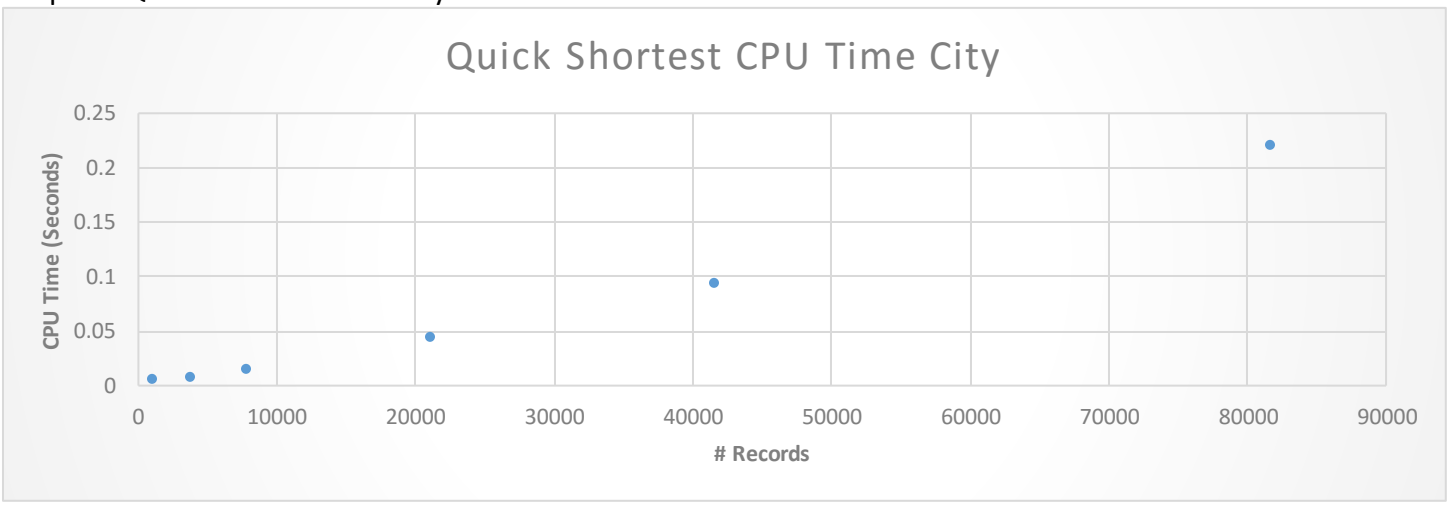
Graph 4: Merge Sort CPU Time City VS # Records



Graph 5: Quick Sort CPU Time Population VS # Records



Graph 6: Quick Sort CPU Time City VS # Records



Graph 7: All First 6 Graphs into 1 graph for Comparison