Lizzy Hanna
CS 3353 - Algorithms
9/12/19

Lab 1 Report

I) Tables for Different Data Set Sizes:

These numbers were recorded by running the algorithms on sets of 4 files at a time. For example, running all Random files of all sizes (10, 1000, 10000, 100000), then recompiling the program and running all Reversed Sorted Order files of all sizes, etc. to get the most accurate data. Since my Insertion sort did not perform as expected (more on this in Part III) when all the data sets were run back to back, I recorded the data in this way.

Data Set Size = 10 elements

Data Set Types:	Algorithms		
	Bubble	Insertion	Merge
Random	1	1	13
Reversed	2	3	19
Partial Sort	1	1	11
Unique	1	1	19

Data Set Size = 1000 elements

Data Set Types:	Algorithms		
	Bubble	Insertion	Merge
Random	11978	14070	1342
Reversed	9575	11938	955
Partial Sort	6417	7689	991
Unique	9620	9209	1019

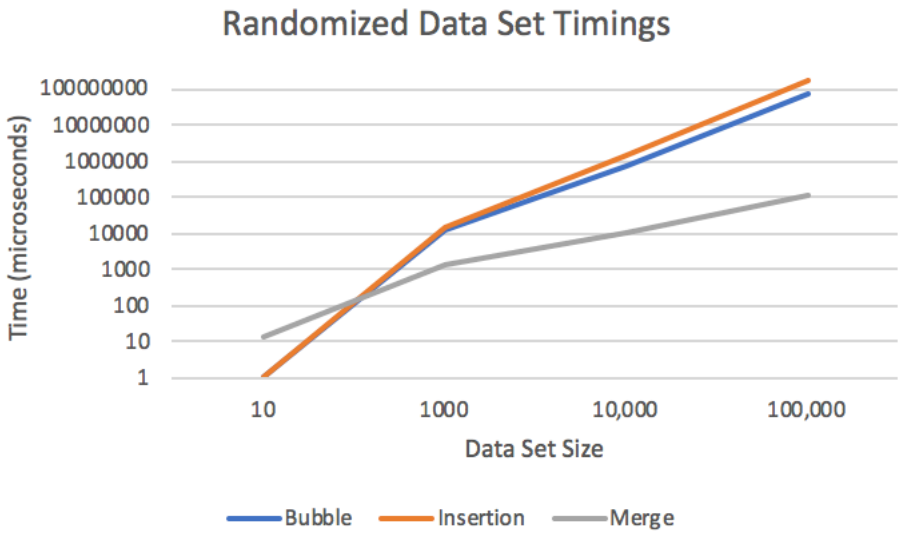
Data Set Size = 10,000 elements

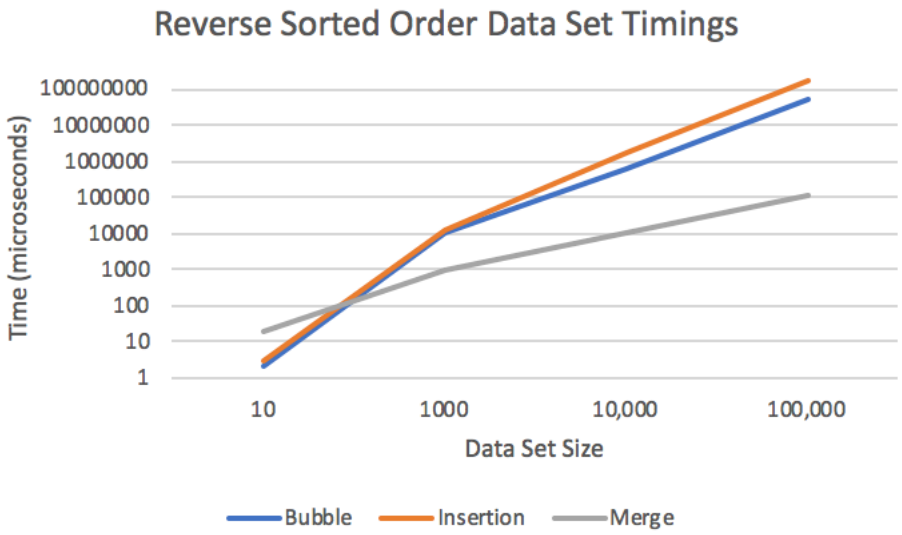
Data Set Types:	Algorithms		
	Bubble	Insertion	Merge
Random	779636	1.45E+06	10722
Reversed	570857	1.63E+06	10397
Partial Sort	523166	1.02E+06	10502
Unique	743004	1.44E+06	10574

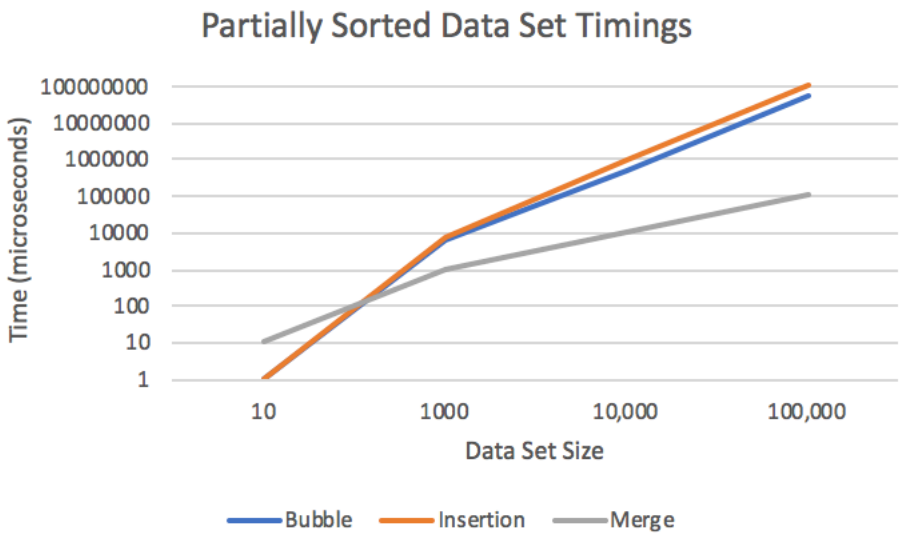
Data Set Size = 100,000 elements

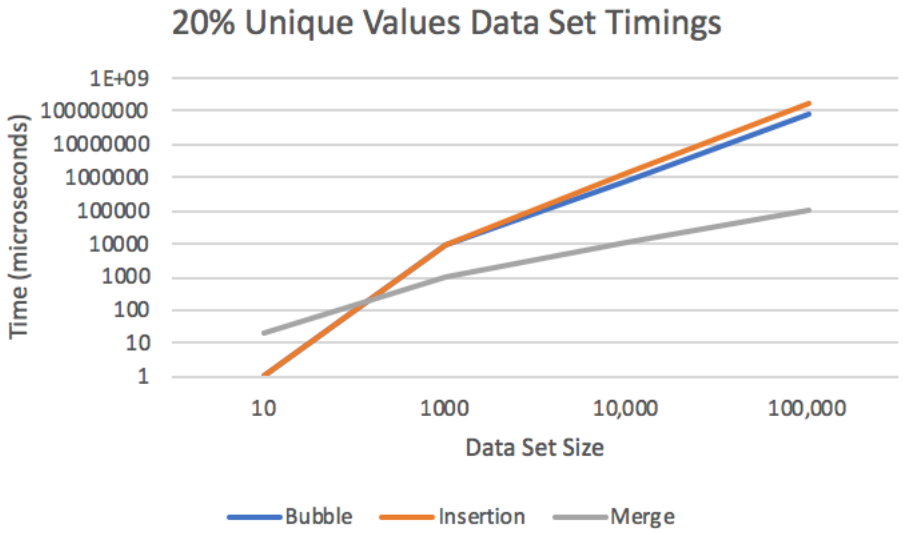
Data Set Types:	Algorithms		
	Bubble	Insertion	Merge
Random	7.54E+07	1.56E+08	111803
Reversed	5.39E+07	1.69E+08	111920
Partial Sort	5.20E+07	1.15E+08	109174
Unique	7.70E+07	1.60E+08	109572

II) Timing Performance Graphs:









III) Explanation of Results:

In smaller data sets, Bubble sort and Insertion sort prevailed as the faster sorting algorithms, but Merge sort quickly proved to be faster in larger data sets starting around 1000 elements or greater. This is because the time complexity of Merge sort is O(nlogn), whereas the time complexities of Bubble sort and Insertion sort are both $O(n^2)$. Taking this into consideration, sorting a data set of n = 100,000 elements would have a time complexity of $O(100{,}000^2) = O(1X10^{10})$ for Bubble and Insertion sort, whereas the time

complexity would be O(10log(100,000)) = O(50), since log(100,000) is only equal to 5. The drastic difference in time complexity explains why Merge sort is so much more efficient than Bubble and Insertion sort and why I had to use a logarithmic scale on the y axis in my graphs (see Part II) to accurately represent the differences in timings.

My version of an Insertion sort is not optimized because it uses a vector instead of a Linked List data structure, forcing the program to have to continually move elements down to make room for an insertion rather than just being able to switch pointers around in a Linked List. I was curious to see how this difference would affect the timing, since despite both having a time complexity of $O(n^2)$ Insertion sort is generally known to perform better than Bubble sort. Both sorting algorithms performed relatively the same, although my Insertion sort proved much slower than my Bubble sort in the largest data set. Some ideas I have to improve the performance of my Insertion sort include transforming the vector into a Linked List before performing the sort and using iterators instead of for loops to access data more efficiently. When I ran the files back to back, Insertion sort's performance was extremely slow on smaller data sets, but when I ran the groups of files by data set (not all back to back), Insertion sort performed as expected. I will need to look into this further to determine why Insertion sort's performance varies so drastically.

In nearly every trial, the data set type that was sorted the fastest by all three algorithms was the Partially Sorted data set. This was the data set that was semi-sorted with only 30% of the values out of order. I believe this was sorted fastest since there was overall less swapping and shifting needing to be done, therefore making the data

set closer to the best-case time complexity of each algorithm, meaning less steps would need to be done to sort the data. The data set type that was sorted the slowest tended to vary per algorithm. I think this is because their implementation varies; for example, the Reverse Sorted Order data set generally took longer for Insertion sort to sort than other data sets. I believe this is because Insertion sort works by using the first element as a “sorted” value and moving every other element based on its position. In a Reverse Sorted Order data set, the first element in the unordered list will become the last element in the ordered list, therefore Insertion sort will need to move every single element in the list. This is a worst-case time complexity for Insertion sort, especially for my implementation of Insertion sort which uses a vector instead of a Linked List.