Brian Faure

RUID:150003563

NetID:bdf39

Programming Methodology II

Homework #3

PseudoCode & Explanations

**Problem #1**

Q1. Write code for selection sort, insertion sort, and bubble sort. Be sure you can read input from a file which has an integer on every line.

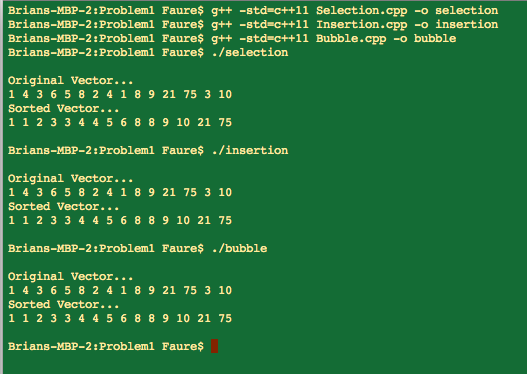
🡪 Within the Problem1 folder I have included four files, test.txt is the file that the Sorting Algorithms read their data in from and the other three files (Selection.cpp, Bubble.cpp, & Insertion.cpp) are the three sorting algorithms required by the problem.

🡪 All three of the sorting algorithms have a function called read\_file which takes the string version of the name of a file in the same folder as the code itself and returns a vector containing the integer data from the file.

🡪 Each sorting algorithm also contains a main function that calls the read\_file function for a specified file name then calls the respective sorting function.

🡪 The main function also controls the terminal output (outputs original vector from file then the sorted version).

🡪 The following is the terminal output after running all three algorithms consecutively…



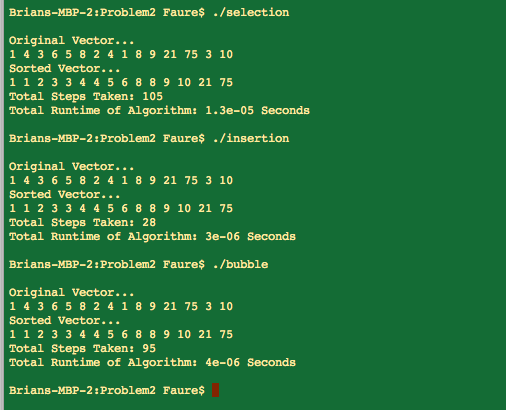
**Problem #2**

Q2. Using your implementation of selection, bubble and insertion sort, insert a counter in the appropriate place so as to measure the runtime of your code (e.g., this could be a counter to track the number of compare operations, or move/shift operations in the innermost loop).

🡪 Within the Problem2 folder I have included four files, test.txt is the same as it was in Problem1 and serves the same function.

🡪 The three sorting algorithms are identical to the ones in Problem1 except they include step counters and run time counters.

🡪 The following is the terminal output after running all three algorithms consecutively…



**Problem #3**

Q3. Plot the runtime as measured by the counter inserted in (2) for each of the three implementations selection, bubble and insertion sort for the data-set provided.

🡪 Within the Problem3 folder are the data sets provided in Sakai as well as my program Algorithms.cpp and the sorted data-set outputs.

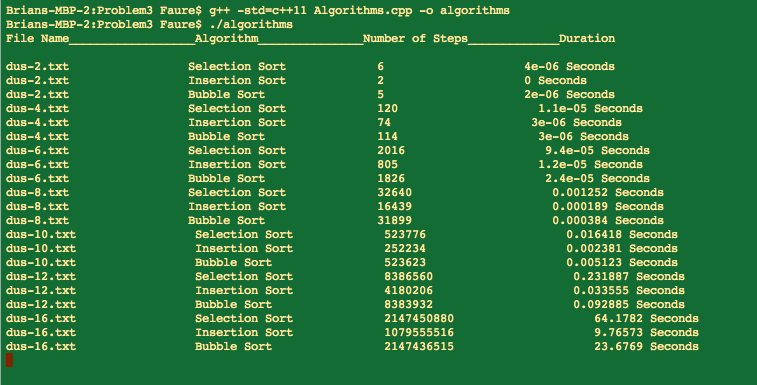
🡪 Rather than having Algorithms output the sorted vectors to the terminal, I used fstream to save them to their own respective files.

🡪 The Algorithms.cpp program itself runs all 3 of the sorting algorithms on all of the datasets and outputs the number of steps taken and the run time for each iteration.

Up to dus-16.txt, the program runs fairly quickly (dus-16.txt only takes about a minute for selection sort and even less for insertion and bubble) but past that it takes an incredibly long time to make all the required comparisons. I ran the program for over 28 hours throughout Saturday and Sunday and was able to finish dus-20.txt as well but from my estimates, dus-24.txt will take about 27 days to complete (and that’s only for Selection Sort). Due to time constraints I used the built in C++ std::sort function to get the dus-24\_SORTED.txt file that is in the Problem3 folder. The rest of the \_SORTED.txt files were output by the Algorithms.cpp program. The following table summarizes my data gathered up to dus-20.txt.

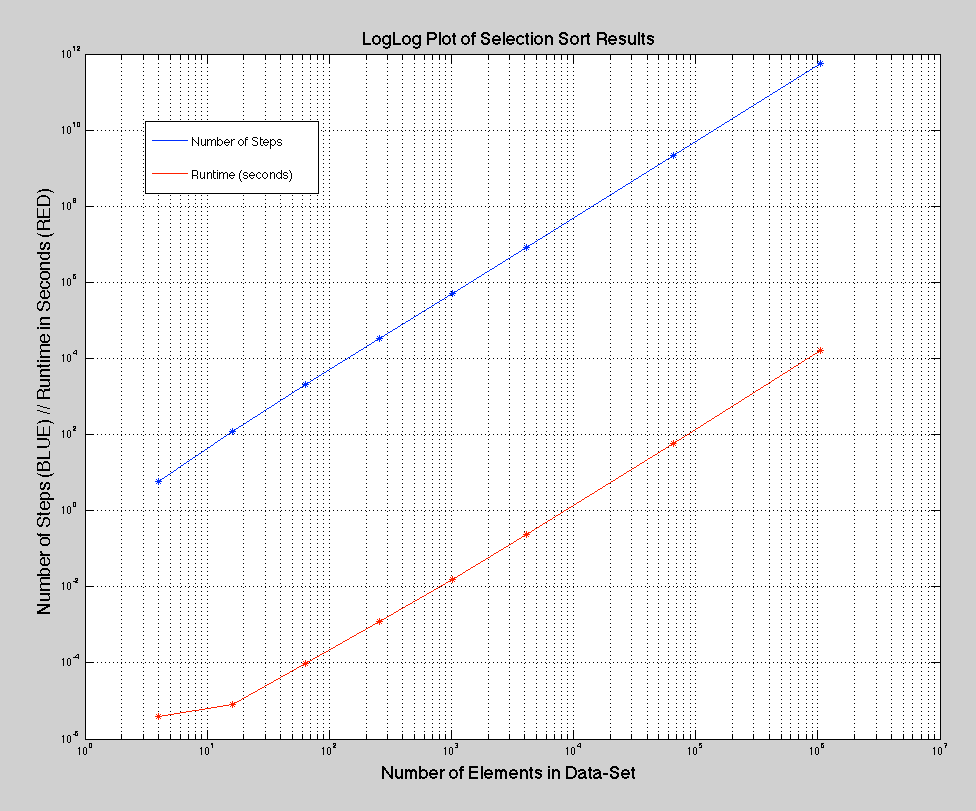
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Selection Sort** | **Insertion Sort** | **Bubble Sort** |
| dus-2.txt | Steps: | 6 | 2 | 5 |
| Runtime: | 0.000004 Seconds | 0.000001 Seconds | 0.000001 Seconds |
|  | | | | |
| dus-4.txt | Steps: | 120 | 74 | 114 |
| Runtime: | 0.000008 Seconds | 0.000002 Seconds | 0.000002 Seconds |
|  | | | | |
| dus-6.txt | Steps: | 2,016 | 805 | 1,826 |
| Runtime: | 0.000094 Seconds | 0.000011 Seconds | 0.000023 Seconds |
|  | | | | |
| dus-8.txt | Steps: | 32,640 | 16,439 | 31,889 |
| Runtime: | 0.001248 Seconds | 0.000185 Seconds | 0.000347 Seconds |
|  | | | | |
| dus-10.txt | Steps: | 523,776 | 252,234 | 523,623 |
| Runtime: | 0.015721 Seconds | 0.002083 Seconds | 0.003802 Seconds |
|  | | | | |
| dus-12.txt | Steps: | 8,386,560 | 4,180,206 | 8,383,932 |
| Runtime: | 0.237962 Seconds | 0.03377 Seconds | 0.068143 Seconds |
|  | | | | |
| dus-16.txt | Steps: | 2,147,450,880 | 1,079,555,516 | 2,147,436,515 |
| Runtime: | 59.7559 Seconds | 8.74367 Seconds | 20.9085 Seconds |
|  | | | | |
| dus-20.txt | Steps: | 549,755,289,600 | 274,363,900,787 | 549,754,508,975 |
| Runtime: | 16426.9 Seconds | 2321.46 Seconds | 5719.72 Seconds |
|  | | | | |
| dus-24.txt | Steps: | ? | ? | ? |
| Runtime: | ? | ? | ? |

🡪 The following picture is a terminal sample output running the Algorithms.cpp program…



*\*\*\*Because I never got an output for dus-24.txt, the following plots only include up to dus-20.txt\*\*\**

🡪 Selection Sort Data Plot:



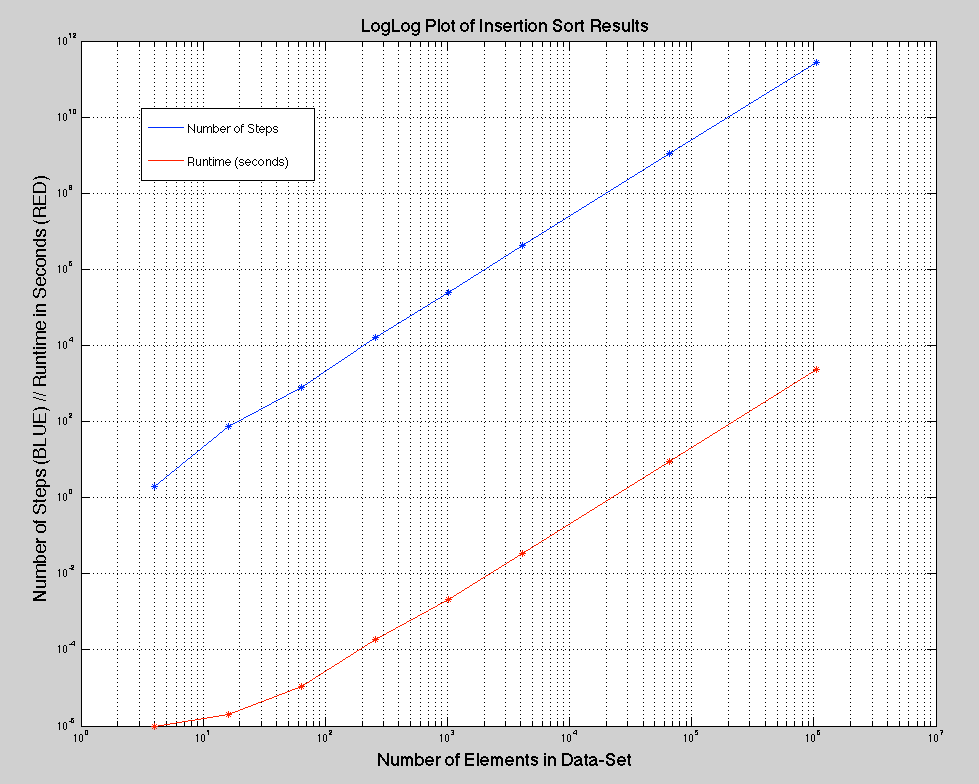
Explanation:

The red line in the above plot represents the runtime of the selection sort algorithm plotted against the data-set size on the x-axis. Similarly, the blue line represents the number of steps of the selection sort algorithm plotted against the data-set size on the x-axis. I chose to use log-log plots for this and the following plots because the true shape of the lines is lost on the fact that the inputs and outputs are just so enormous in range. When I plotted this line on a regular x/y axis, the only thing you are able to make out is a very tiny portion in the bottom left corner where most of the data is, then one long straight line connecting to the last pair of points at dus-20.txt because it is so large compared to the other data points.

Analysis:

As can be expected, the step count for Selection Sort is linear when plotted on a logarithmic plot because the complexity of Selection Sort is **always** exponential, O(n^2), no matter whether or not the components are already somewhat in order or not. This means that later in problem 4 we should get a similar result even through all of the inputs will be previously sorted.

🡪 Insertion Sort Data Plot:



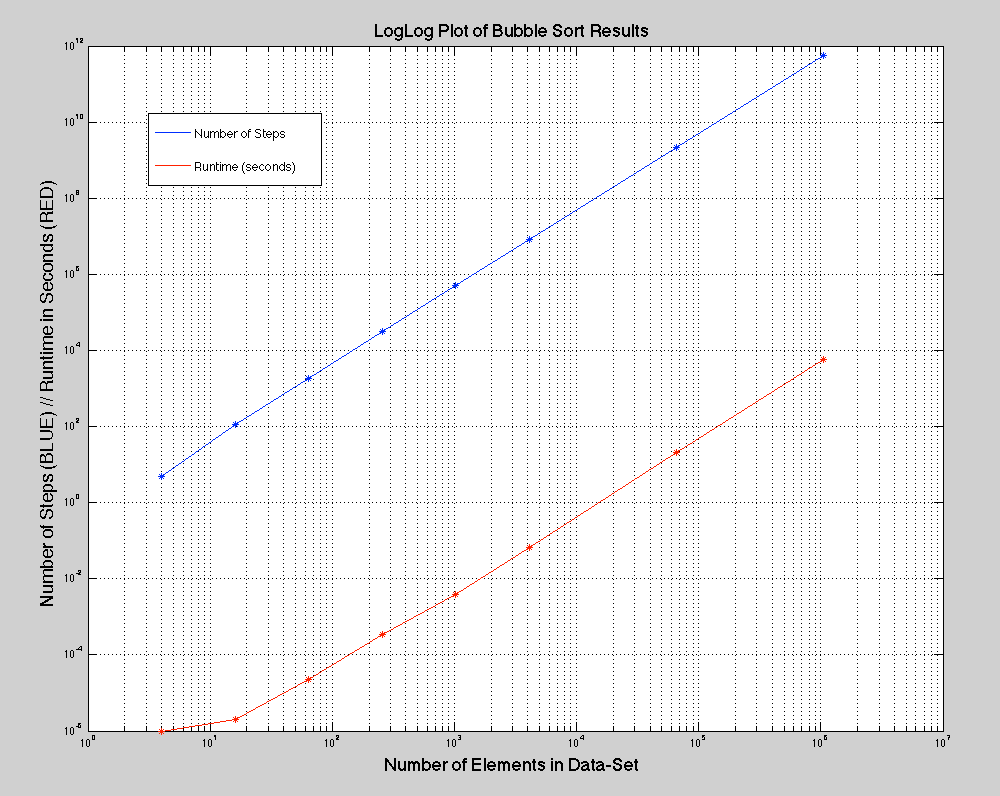
Explanation:

Once again the blue line corresponds to the step count for the algorithm, in this case Insertion Sort, and the red line corresponds to the runtime, in seconds, for the algorithm.

Analysis:

The average complexity for Insertion sort is O(n^2) while its best case is O(n). This is reflected in the plot because, as opposed to the constant linear slope on the Selection Sort plot, the Insertion Sort plot does not have a constant slope. In the data sets where the data is already partially sorted, the Insertion Sort algorithm takes significantly less time and runs on less steps.

🡪 Bubble Sort Data Plot:



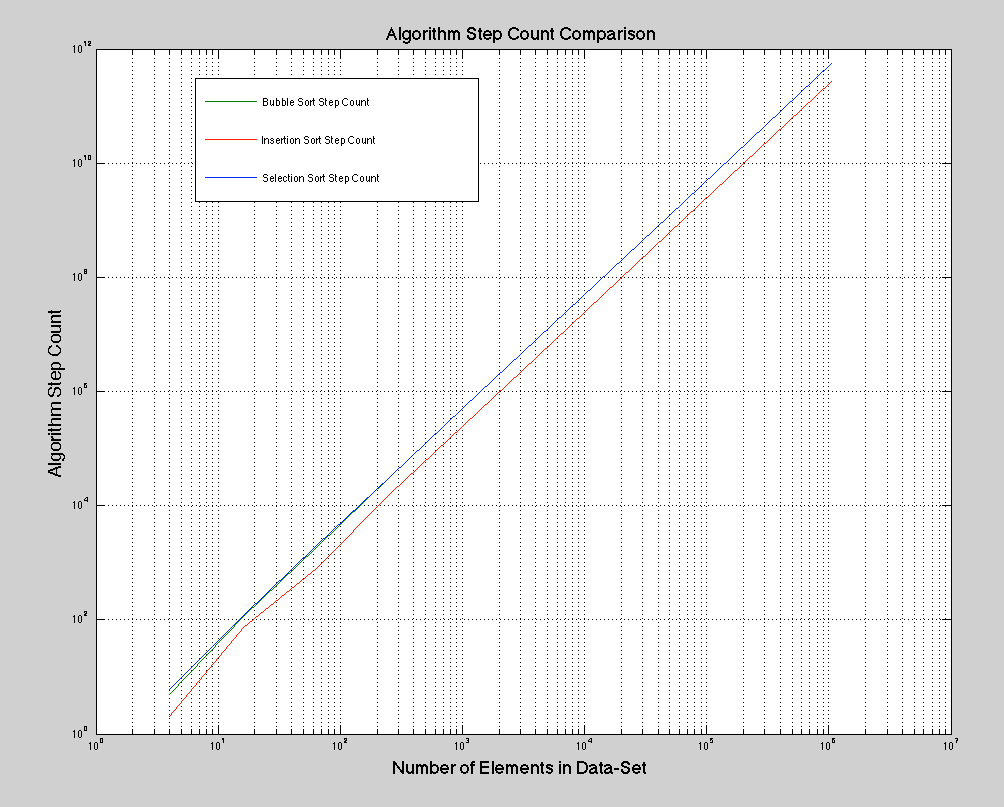
Explanation:

Same layout as the previous 2 plots.

Analysis:

Similar to the Insertion Sort, Bubble sort has a best case of O(n) and an average case of O(n^2). This is also presented in the graph as a non-perfect linear line. If the line were perfectly straight, it would mean that the sorting algorithm had a constant exponential gain in complexity when in reality the data sets we ran are not all perfectly unsorted.

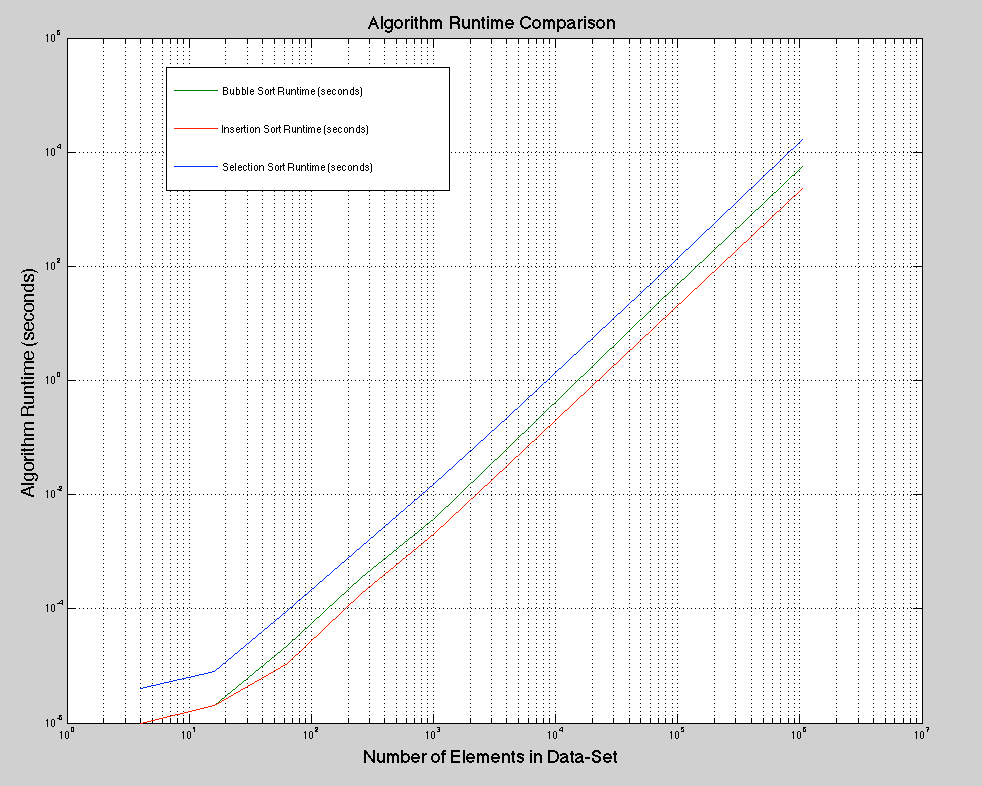
🡪 Algorithm Step Count Comparison Plot:



Analysis:

As we can expect based on the fact that Selection Sort will always have O(n^2) complexity, it has more steps per data set than both the Insertion Sort and Bubble Sort. The Bubble Sort at first is slightly more efficient than the Selection Sort but as the number of elements in the Data-Set Increases, this slight advantage in efficiency is washed out by the sheet size of the step count. The Insertion Count is the clear winner when it comes to minimizing step count as it always is well below both the Selection Sort and Bubble Sort.

🡪 Algorithm Runtime Comparison Plot:



Analysis:

I expected this plot to very closely mimic the step count comparison but it actually is very different. From this comparison we can see that despite the fact that Bubble Sort has almost the same number of steps as Selection Sort, it takes significantly less time for it to finish sorting. Once again the Insertion Sort algorithm is the winner when it comes to runtime efficiency.

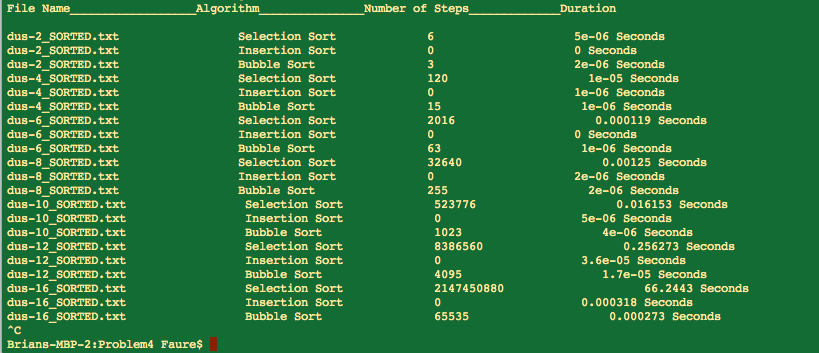
**Problem #4**

Q4. Take the sorted output from 3, and use that as the sorted input for your three implementations. Do the runtimes change? Explain.

🡪 Within the Problem4 folder I have included the sorted data sets as well as the program Algorithms\_SORTED.cpp which I used to run the Selection Sort, Insertion Sort and Bubble Sort algorithms on the sorted data sets.

🡪 I also have included another cpp file called noSelection.cpp that I will explain below.

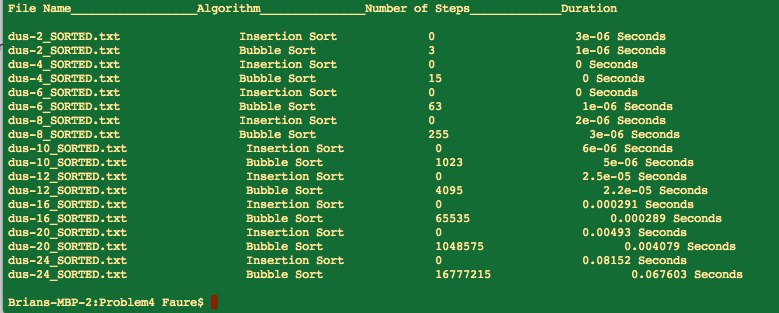
The following is the terminal output after running Algorithms\_SORTED.cpp up to the dus-16\_SORTED.txt data set:



Explanation:

I only ran this program up to the dus-16\_SORTED.txt data set because I ran into the same problem I had been having with the unsorted data sets; it was taking forever. The only difference with the sorted data sets was that the only sorting algorithm that was taking a long time was the Selection Sort. This is because, even in it’s best case, Selection Sort still will have O(n^2) complexity while both Insertion Sort and Bubble Sort have a best case complexity of O(n).

Instead of just leaving the program the way it was, I decided to create another C++ file called noSelection.cpp, which is also in the Problem4 folder that only includes Insertion Sort and Bubble Sort. This algorithm runs ***significantly*** faster. The terminal output for noSelection.cpp is below…

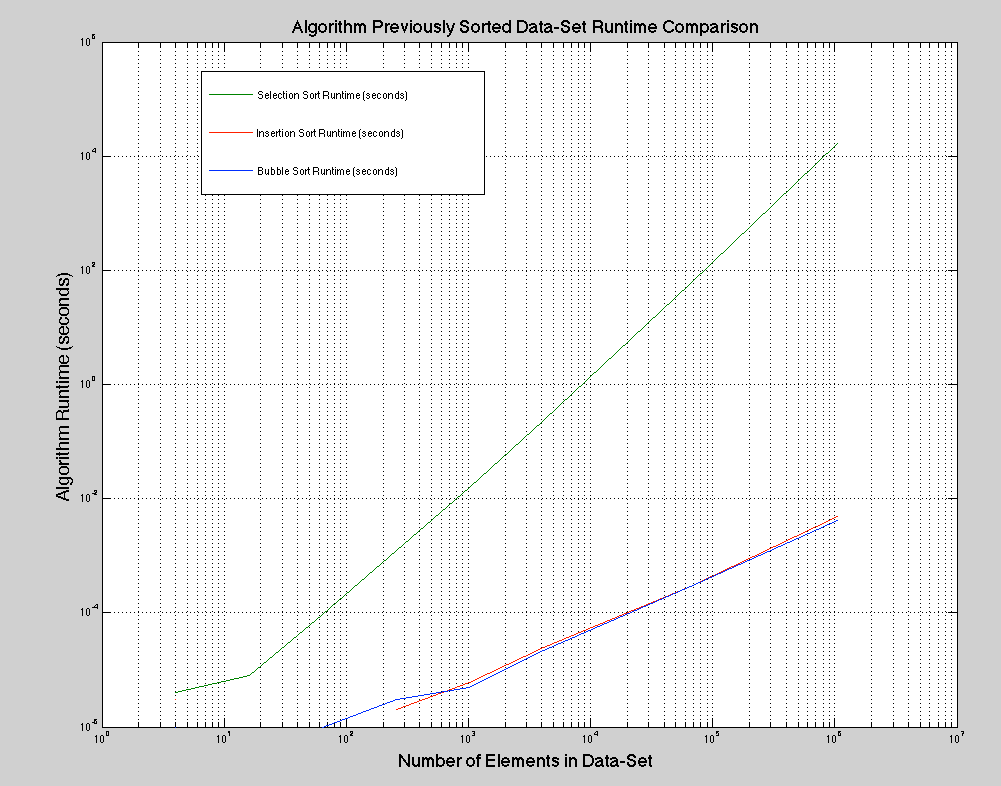


🡪 Putting this data together in a table we get the following:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Selection Sort** | **Insertion Sort** | **Bubble Sort** |
| dus-2\_SORTED.txt | Steps: | 6 | 0 | 3 |
| Runtime: | 0.000004 Seconds | 0.000003 Seconds | 0.000001 Seconds |
|  | | | | |
| dus-4\_SORTED.txt | Steps: | 120 | 0 | 15 |
| Runtime: | 0.000008 Seconds | ~0 Seconds | ~0 Seconds |
|  | | | | |
| dus-6\_SORTED.txt | Steps: | 2,016 | 0 | 63 |
| Runtime: | 0.000094 Seconds | ~0 Seconds | 0.000001 Seconds |
|  | | | | |
| dus-8\_SORTED.txt | Steps: | 32,640 | 0 | 255 |
| Runtime: | 0.001248 Seconds | 0.000002 Seconds | 0.000003 Seconds |
|  | | | | |
| dus-10\_SORTED.txt | Steps: | 523,776 | 0 | 1,023 |
| Runtime: | 0.015721 Seconds | 0.000006 Seconds | 0.000005 Seconds |
|  | | | | |
| dus-12\_SORTED.txt | Steps: | 8,386,560 | 0 | 4,095 |
| Runtime: | 0.237962 Seconds | 0.000025 Seconds | 0.000022 Seconds |
|  | | | | |
| dus-16\_SORTED.txt | Steps: | 2,147,450,880 | 0 | 65,535 |
| Runtime: | 59.7559 Seconds | 0.000291 Seconds | 0.000289 Seconds |
|  | | | | |
| dus-20\_SORTED.txt | Steps: | 549,755,289,600 | 0 | 1,048,575 |
| Runtime: | 16426.9 Seconds | 0.00493 Seconds | 0.004079 Seconds |
|  | | | | |
| dus-24\_SORTED.txt | Steps: | ? | 0 | 16,777,215 |
| Runtime: | ? | 0.08152 Seconds | 0.067603 Seconds |

\*\*\*I’ve left the values for Selection Sort the same\*\*\*

🡪 The following plot shows the relationship of all 3 sorting algorithms by their runtime for the sorted data sets:



Analysis:

As expected, the Insertion and Bubble Sort Algorithms performed much better than the Selection Sort Algorithm when subject to their best case scenarios.