Brian Faure

RUID:150003563

NetID:bdf39

Programming Methodology II

Homework #4

Analysis

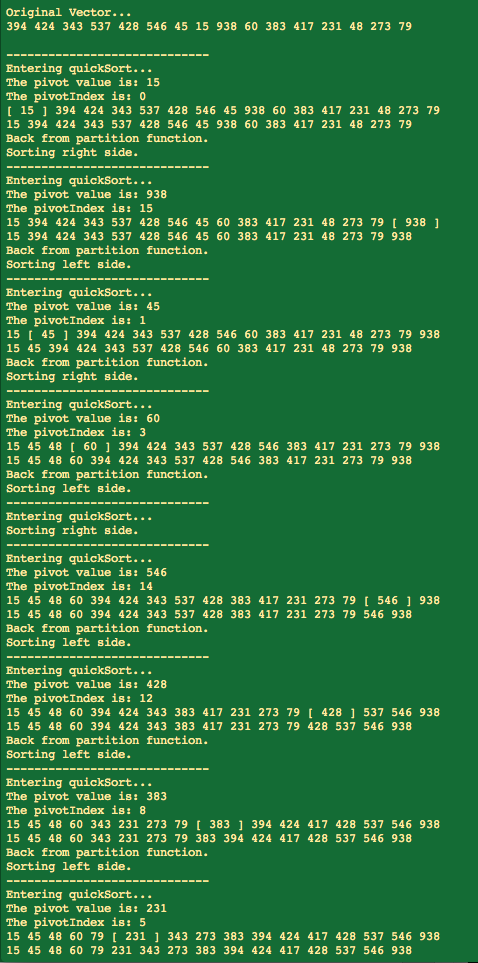
**Question #1**

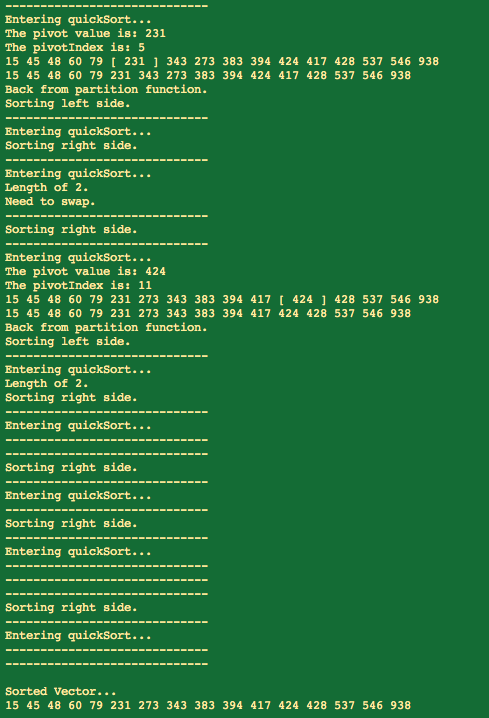
🡪 Folder: Question1

🡪 File: Question1.cpp, test.txt

🡪 Code Explanations present in Question1.cpp code comments

The next two pages are the terminal output running Question1.cpp on the text.txt text file included in the Question1 folder. The first and last outputs are the Original input and Sorted output, respectively.





**Question #2**

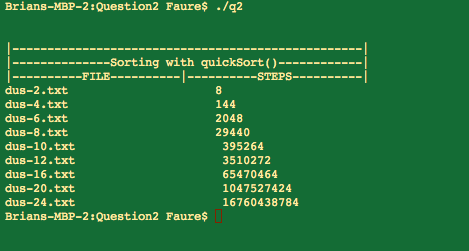
🡪 Folder: Question2

🡪 Files: Question2.cpp and all provided data-sets

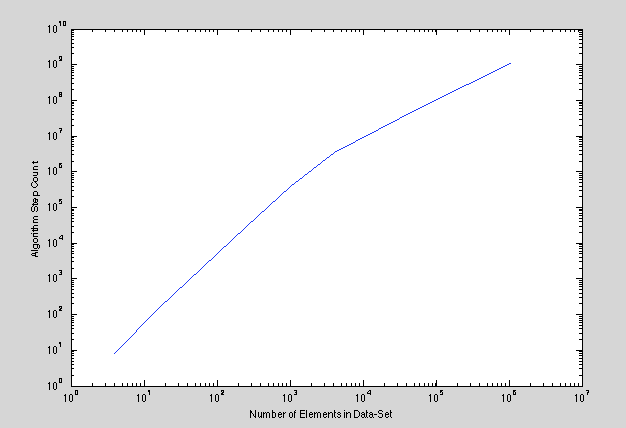
🡪 Code Explanations present in Question2.cpp code comments

Overall Summary: I have chosen to use the “middle” value as my partition point rather than the first or last element because it should provide slightly better efficiency results. The program outputs to the terminal as well as outputting the sorted data-sets to their own respective files with the –SORTED extension but this feature has been commented out because it was only used to provide the needed data-sets for Question3.

🡪 The following is the terminal output running Question2.cpp



Even thought the step count is fairly high for the bigger data-sets, the algorithm only took about 10 minutes to completely finish. From this data we can also see that the complexity increases in an almost exponential form. I have taken these step-counts and plotted them against the number of elements in each data-set in a log-log MATLAB plot to get the following plot…



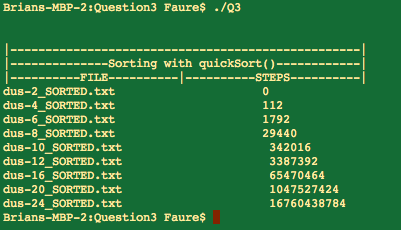
Because this plot is Log-Log, if the slope had been constant it would have meant that the increase in complexity was in fact exponential, because it is not perfectly straight and actually slopes downward, it can be inferred that the quickSort algorithm actually performs fairly well as the input vector is increased in size.

**Question #3**

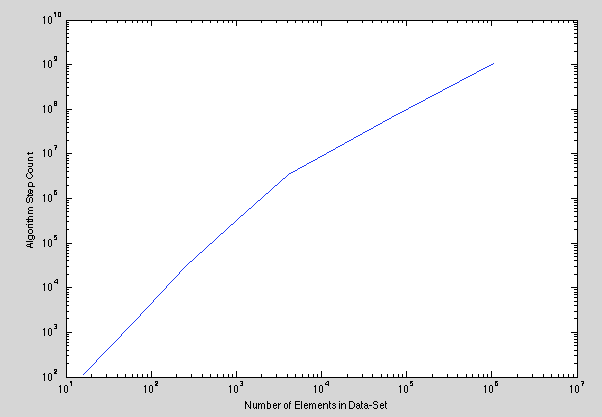
🡪 Folder: Question3

🡪 Files: Question3.cpp and all sorted data-sets

Question3.cpp terminal output running on sorted data-sets:



MATLAB Log-Log plot of algorithm complexity:

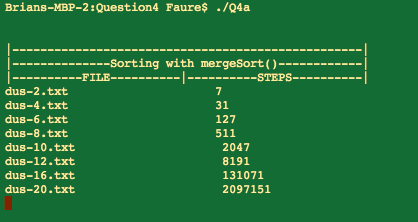


**Question #4**

1. mergeSort with temporary Array (extra memory):

🡪 File: Question4a.cpp, all provided data-sets

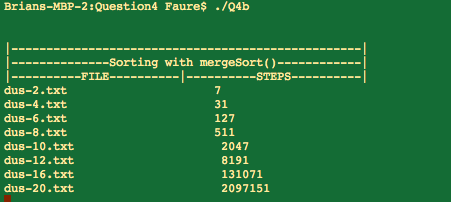
Question4a.cpp terminal output:



1. mergeSort using in-place array/vector management:

🡪 File: Question4b.cpp, all provided data-sets

Question4b.cpp terminal output:



Analysis: In the two screenshots above, the counter was placed right inside the mergeSort() function. It makes sense that the count would be the same on both the temp-array and in-place implementations because the only difference between the two of them is the way they handle the merge() function. Even through the number of times mergeSort() was called was the same for the two implementations, I noticed a significant difference in the amount of time the two algorithms took. On my computer, the in-place sorting was taking more time to complete than the temp-array sorting. I believe this is the case because the vector member function .emplace has to move every element to the right of the emplace position, to the right by 1 and this is a time-consuming process for the function to have to deal with on every iteration of the while-loop.