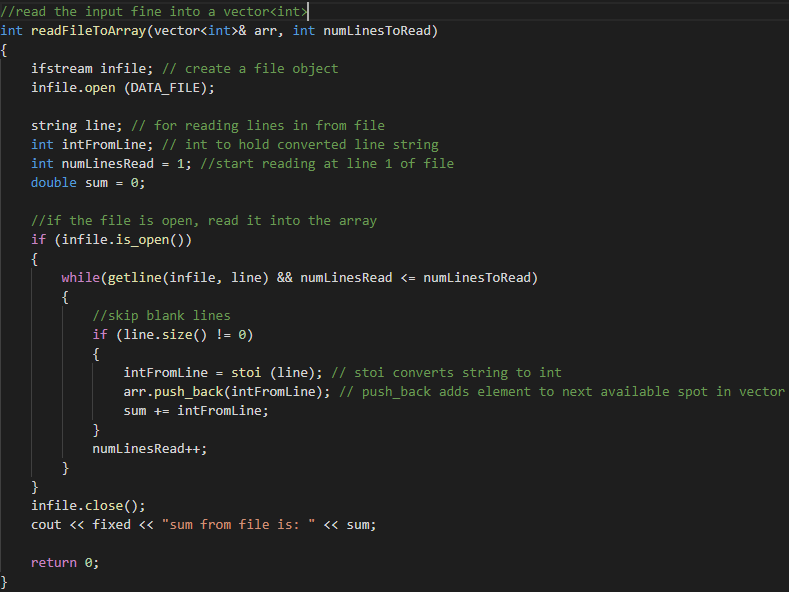
Tyler Bartlett   
01/27/2019  
Dr. Breeann Flesch  
CS361 – Algorithms

Lab 1 Report

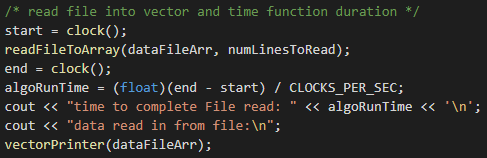
This was the first assignment I have ever done in C++ and I wanted to push myself to learn the language while doing this first lab. I have written ANSI C before though, so it is a little familiar already, and is the reason why many of my functions return an int, namely zero, to indicate function success.

I started by wring the code to read in a text file. I made a simple test file to use in place of the data file provided for this lab while writing and testing all my functions. It is eleven lines long and included the values: 12, 15, 123, 20, 100, 200, 50, and 10. I had a couple empty lines in between a few data points so I could prepare my file reader function to skip them. I originally had written the function to read in the entire file and later modified it to read in a variable amount of lines by adding an additional parameter to the function and adding an additional check in the while loop to read n lines.

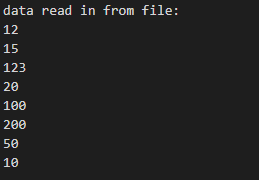
This is my function to read in data from a text file.  


I checked the sum of all the data in the file provided, and it was 49,999,995,000,000, which matched the provided sum in the lab instructions.

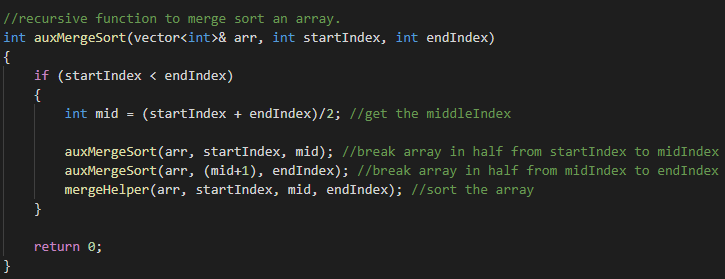
This is where I call the readFile function in main, and how I timed the output. numLinesToRead is declared in main prior to this function call and was changed from 1,000 by 10x until it was equal to 10 million lines to read in.



I verified that the function worked by printing the contents of the read-in array, and as we see below, the printed values matched the values of my test file, in order they were read, without including the empty lines.

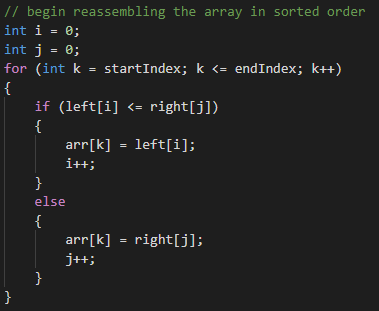
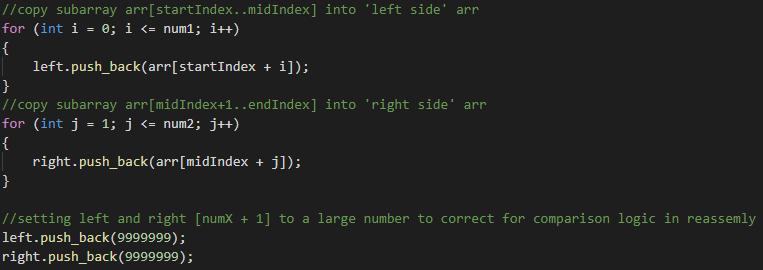


After my readFile function was done, I started working on the merge sort. I used pseudo code from the text book to write the merge sort function and the merge sort helper function, which has code snippets on the following page.

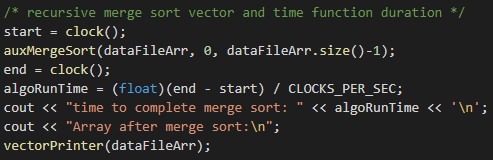


The two following code snippets are from my merge sort helper function, which receives an address to a vector, a startIndex, a midIndex and an endIndex values. I copy the contents of the passed in vector into two sub-vectors and set the sub-vectors next available space to a very large number, which is necessary in the logic of reassembling the pieces into a sorted vector.

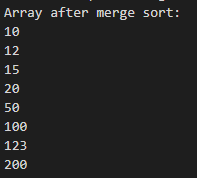
I was having some trouble with this function initially because I did not include the step of adding a large number to next available spots in the two sub-vectors, which Breeann helped my fix after class.



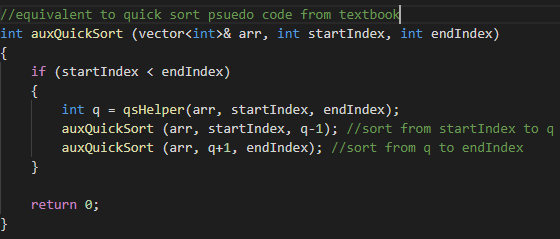
This is where I call the merge sort function in main, and how I timed the output.

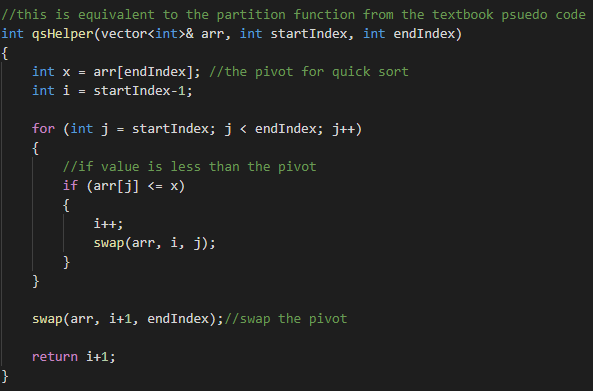


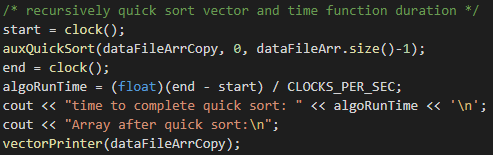
I verified that the function worked by printing the contents of the sorted array, and as we see in the photo below, the array is sorted properly.



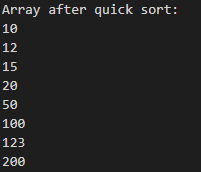
The quick sort algorithm was quick to setup and get working. I based my function off the pseudo code in the book.



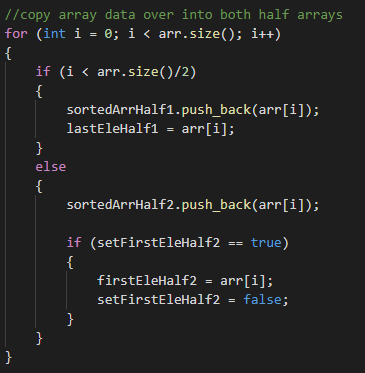


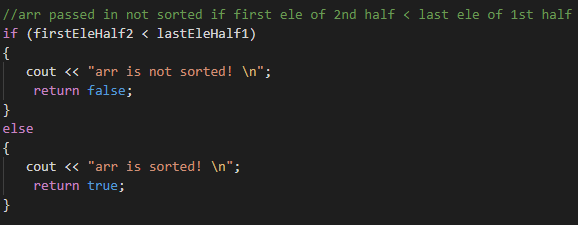
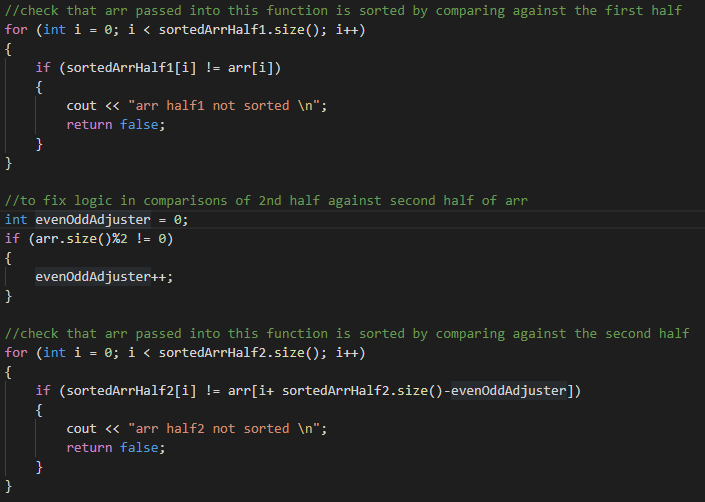
The following snippet is where I call the quick sort function in main. 

I verified that the function worked by printing the contents of the sorted array, and as we see in the photo below, the array is sorted properly.

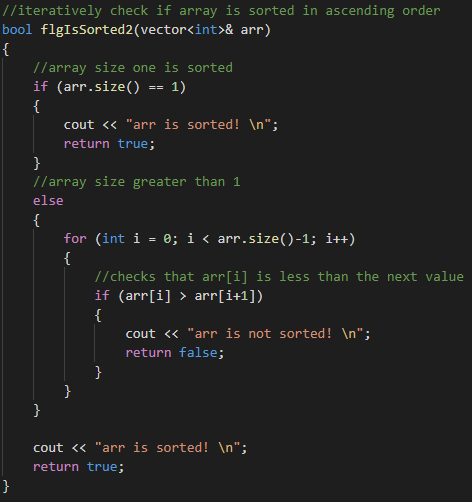


I was having quite a bit of trouble with the flgIsSorted function. I set it up to take only one parameter, the vector, and run through it to confirm data is sorted. I set it up to copy the contents of the passed in vector into a first half, second half vectors, ran quick sort on those two halves, then compared the vector passed into the function against the two halves. I then checked if the last element of the first half was larger than the first element of the 2nd half to ensure the array was sorted in ascending order. I had some difficulty with a bug not making this work with my test data, so I had talked to Mike online and got some help from him debugging. This function was not recursive as I did not see that it needed to be in the instructions, and when testing the function, it caused my program to crash when trying to verify the 100,000 data values were sorted properly. Bellow are some snippets of the broken function.



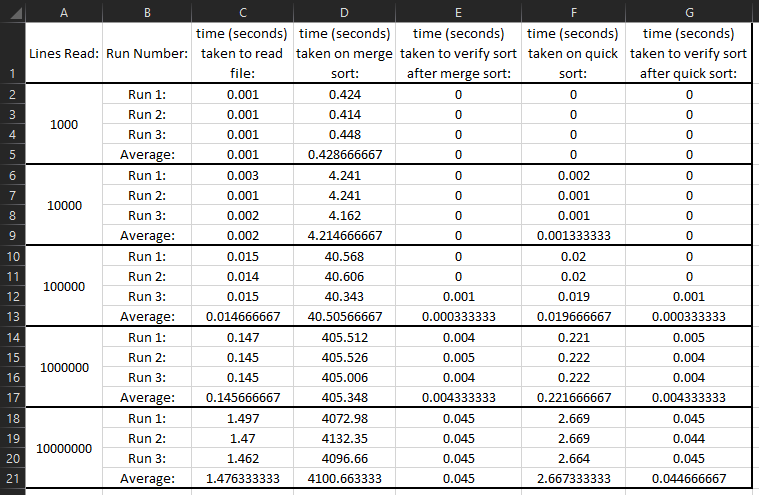


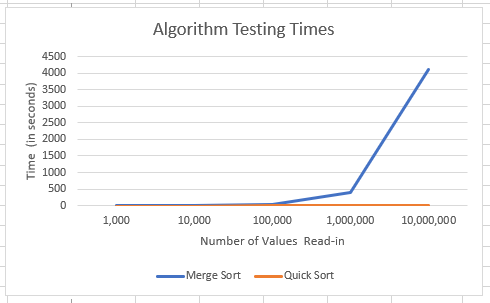
After I had found that this function breaks at 100,000 data values, I quickly rewrote another function to verify if the data was sorted in ascending order. I was struggling to figure out how to do this recursively with only one parameter on the flgIsSorted function and while working with vectors, so for the sake of time, it too is an iterative function. Below is a snippet of the new flgIsSorted function.



Now onto the data recording:

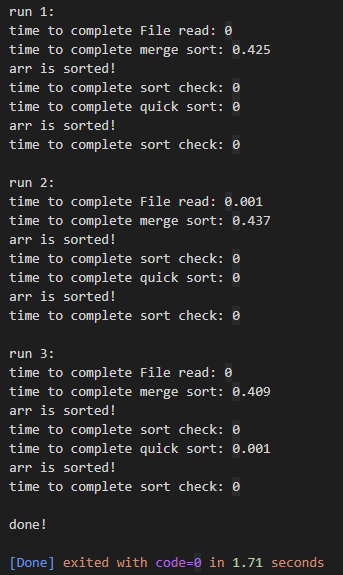
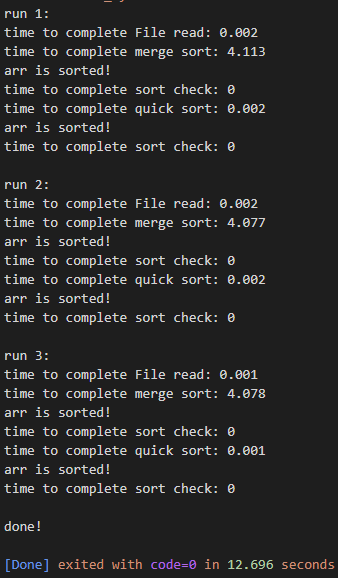
The time taken for the quick sort algorithm is drastically less at every interval of amount of lines read in.  
even at 10 million records, my average quick sort run time is 2.6 seconds in length, which is drastically better than the aver of 4,100 seconds to complete a merge sort. This is why the line for quick sort on the graph appears flat.



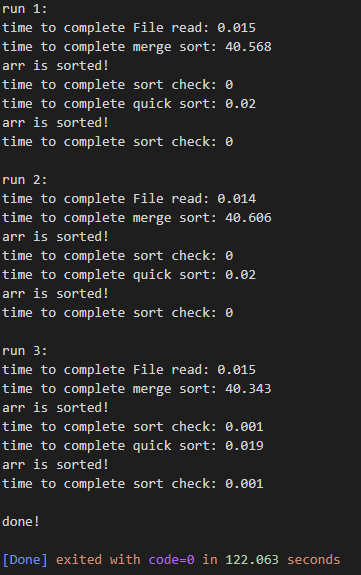
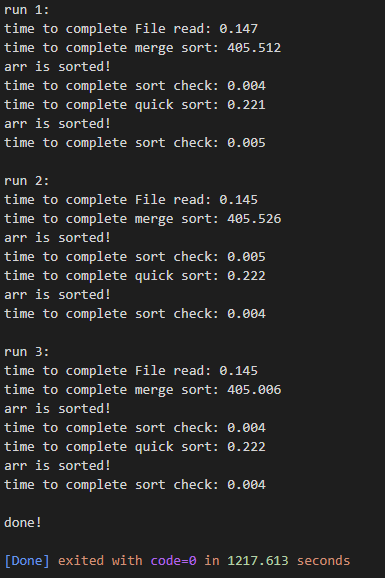


The following series of screen dups shows a printout that the array was verified to be in ascending order.

1,000: 10,000:

100,000: 1,000,000:

10,000,000:

