Implementing Merge Sort
Lab #3

By
Joshua Matthews

CS 303 Algorithms and Data Structures
September 15, 2015

1. Problem Specification

The goal of this Lab assignment was to implement the Merge Sort and test the difference in time it takes to sort the input from various sized files passed in to the program and also compare it to the Insertion Sort. The last task was to make the Merge sort decide to use the Merge sort or switch to Insertion sort depending on the size of the array.

2. Program Design

This program required two functions besides the main function to implement the Merge sort. It had to accept and extract values from a file, test the Merge sort, and display the amount of time it took to complete the insertion sort. When turning the sort to incorporate the Insertion sort to be used on smaller sized arrays this program required three functions.

The following steps were needed to develop this program:

A. I needed to setup the environment with the correct c++ libraries included and main function to accept arguments that would be passed into the program.
B. Next I setup the code to take the file passed in and extract the values from the file and push them on to a vector to be used for later.
C. I then created a separate vector to copy the initial vector to have a copy for the Merge sort to use.
D. I then created a function which implemented the Merge sort which has two vectors passed in as reference, a size parameter for the size of the array, starting point integer, and a ending point integer. This function handled calling the recursive calls to itself to break down the array and then send it to the merge function.
E. I then create the function for merge which took in both vectors passed in the merge sort and takes in the starting integer, ending integer, and an integer for pointing to the middle of the array. The function then goes through the section that it needs to and sorts the elements in order.
F. The Insertion sort part was implemented by me just copying over the code from the last assignment and setup an if condition in the Merge sort function to invoke Insertion instead if the array is small enough.
G. I then setup code around where I am invoking the Merge sort function on the data from the file to calculate how long the Merge sort took to complete.

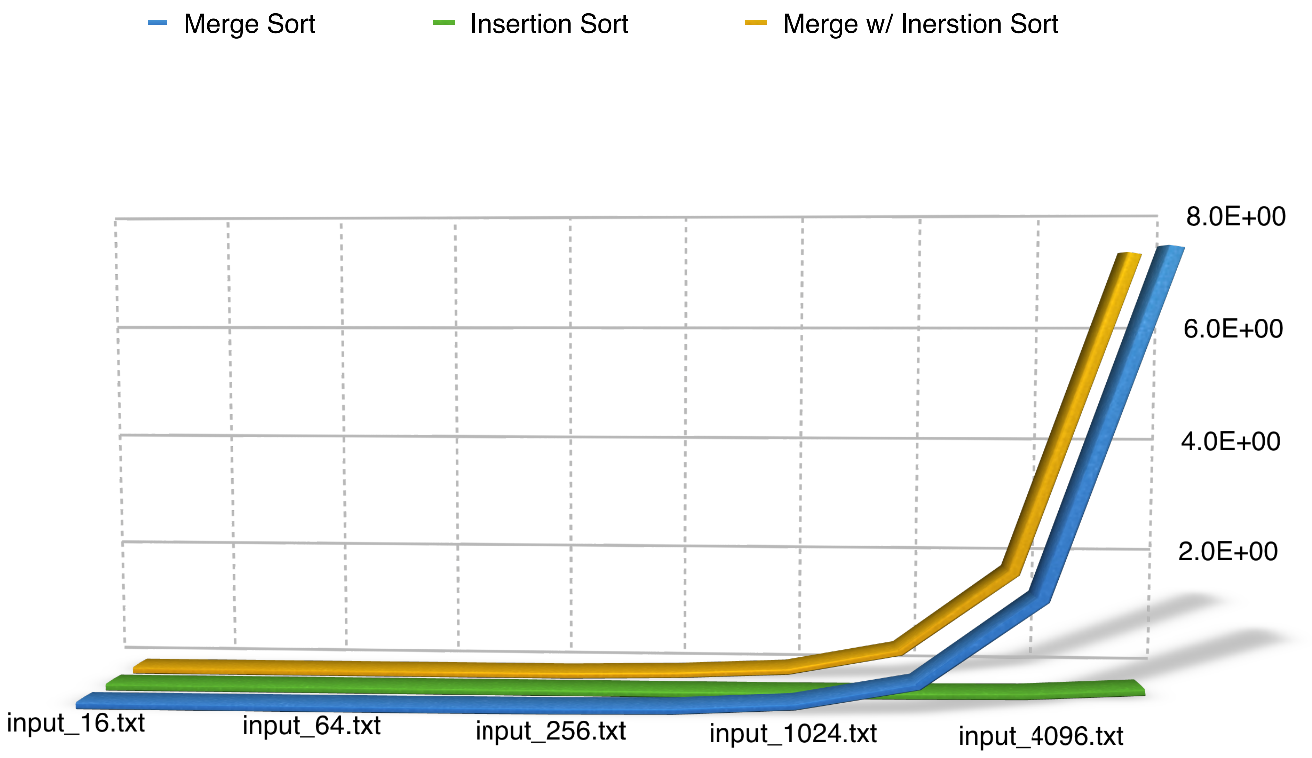
The included libraries were: iostream(for printing out to the screen), fstream(for reading in data from a file), vector(for having a dynamically sizing array), ctime(for calculating program running time in seconds).

3. Testing Plan

The Lab03.cpp file tests the Merge sort on the values from whatever file is passed in. The program accepts a file and parsers out each number in the file and pushes them back onto the vector object that was created to hold all of the values. The initial values we retrieve from the files are random numbers between 1 and the number of elements that are in the file inclusively. Once the vector is filled with every value from the file the current time is capture in a variable then the Merge sort is invoked. The vectors is passed in by reference so there is no time being wasted on copying the values over to the function and allows the function to have direct manipulation over the object. Once the Merge sort is done we capture the current time minus the start time to calculate how long the function took in seconds, then display it to the screen for the user to see.

4. Test Cases

The test cases are shown below. The graph shows how long the sorts took from input_16.txt to input_8192.txt.



5. Analysis and Conclusion

The results show that the Merge sort has a time complexity of O(nlogn) in both best case and worst case scenarios. The Merge sort seems to be slower than the Insertion sort at first but because Insertion sort can grow in O(n^2) time complexity the Merge sort will be faster with larger sized arrays in theory.

6. References

The www.cplusplus.com has references and documentation on all of the standard libraries that were used in this program. Input files were supplied by Abhishek Anand.