**Lab 2 Lab Report**

**Compiling Instructions:**

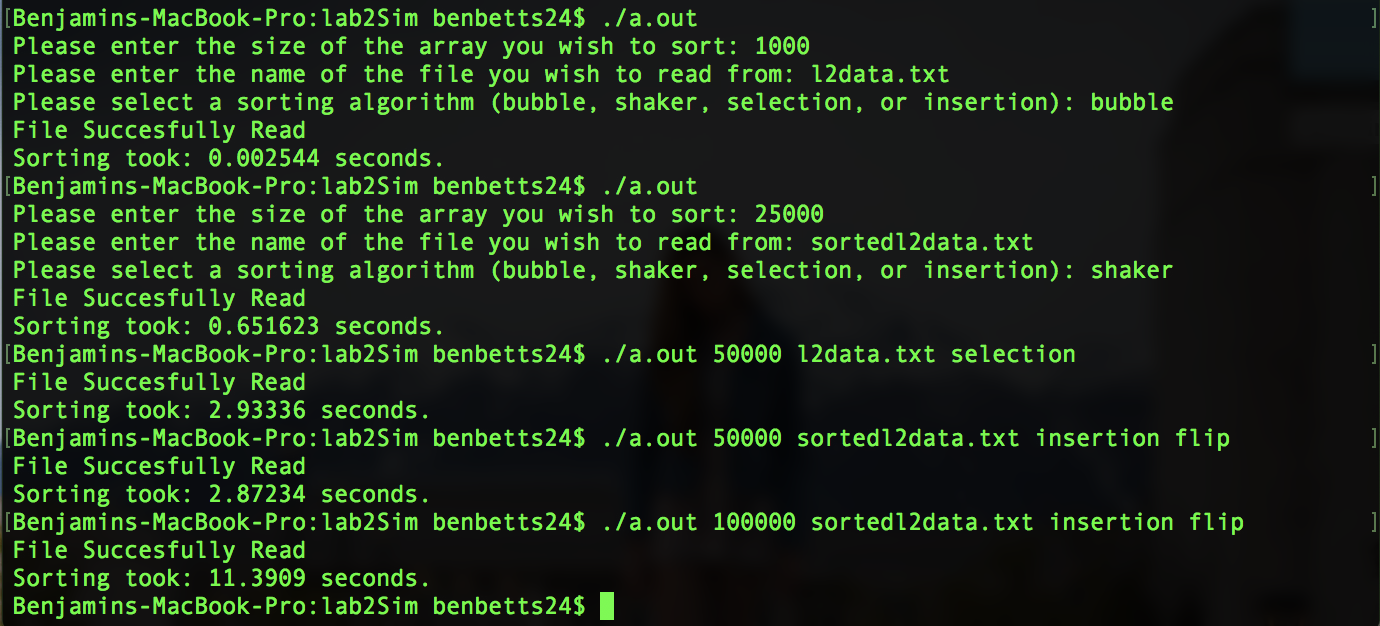
This code will compile very easily. Make sure you have the following files together in the same directory:

* main.cpp
* support.h
* support.cpp
* a list of integers to sort (not required to compile)

With the files listed together, type “g++ main.cpp support.cpp”. This will compile into a.out, the new executable file.

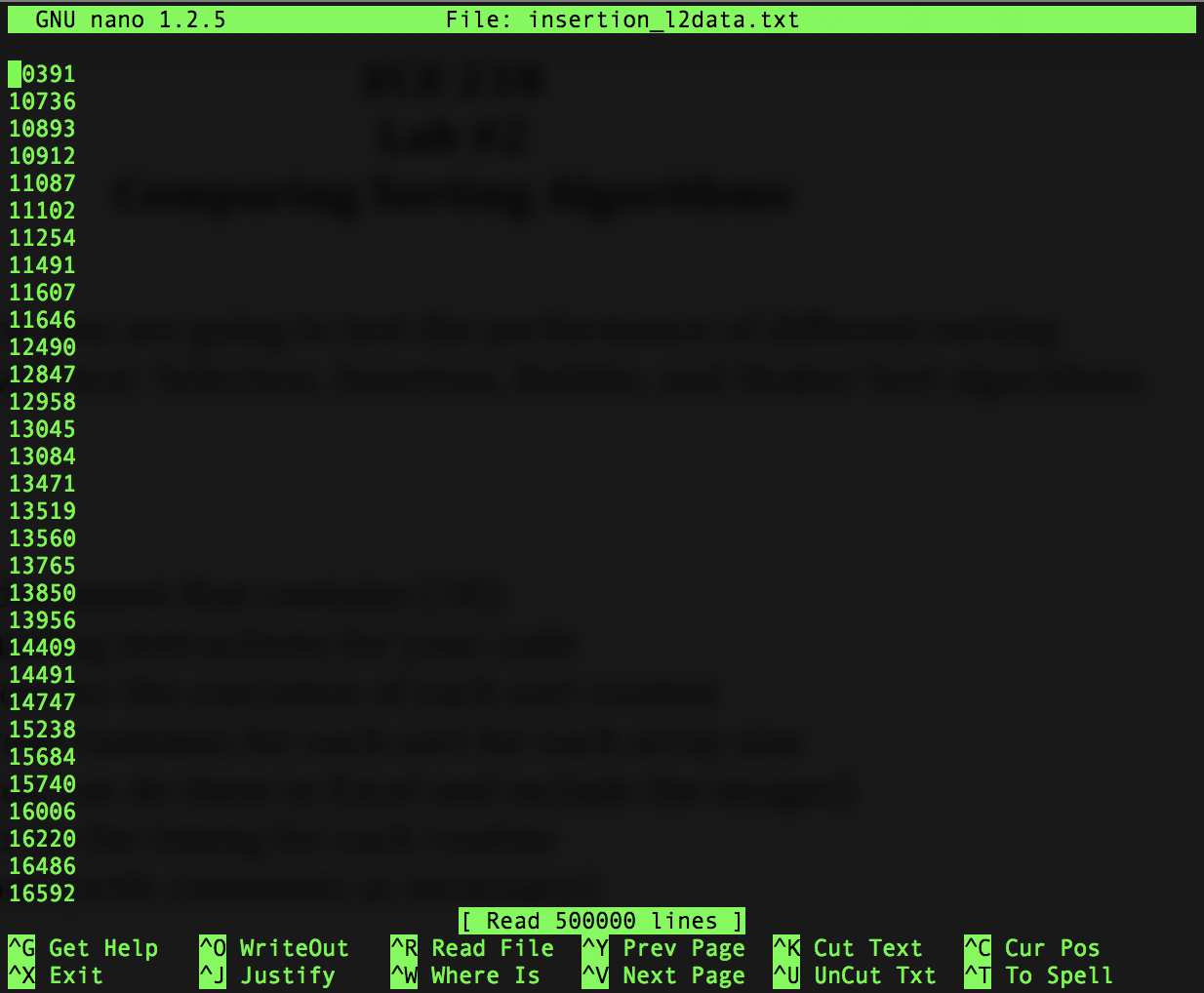
In this program I implemented command line input to save myself time while testing the algorithms. Using it is completely optional. Running without cause the program to ask for your array size, the name of the filename to be sorted, and the algorithm to use. If you wish to use command line input, simply type those values in in the same order. For example: “a.out 10000 l2data.txt insertion” would sort an array of 10,000 integers from the file l2data.txt with the insertion algorithm. An additional feature of the command line input is telling the algorithms to sort biggest to smallest instead of smallest to biggest. To do this, simply type anything at all after the algorithm. I usually type flip.

**Screenshot of Implementations:**



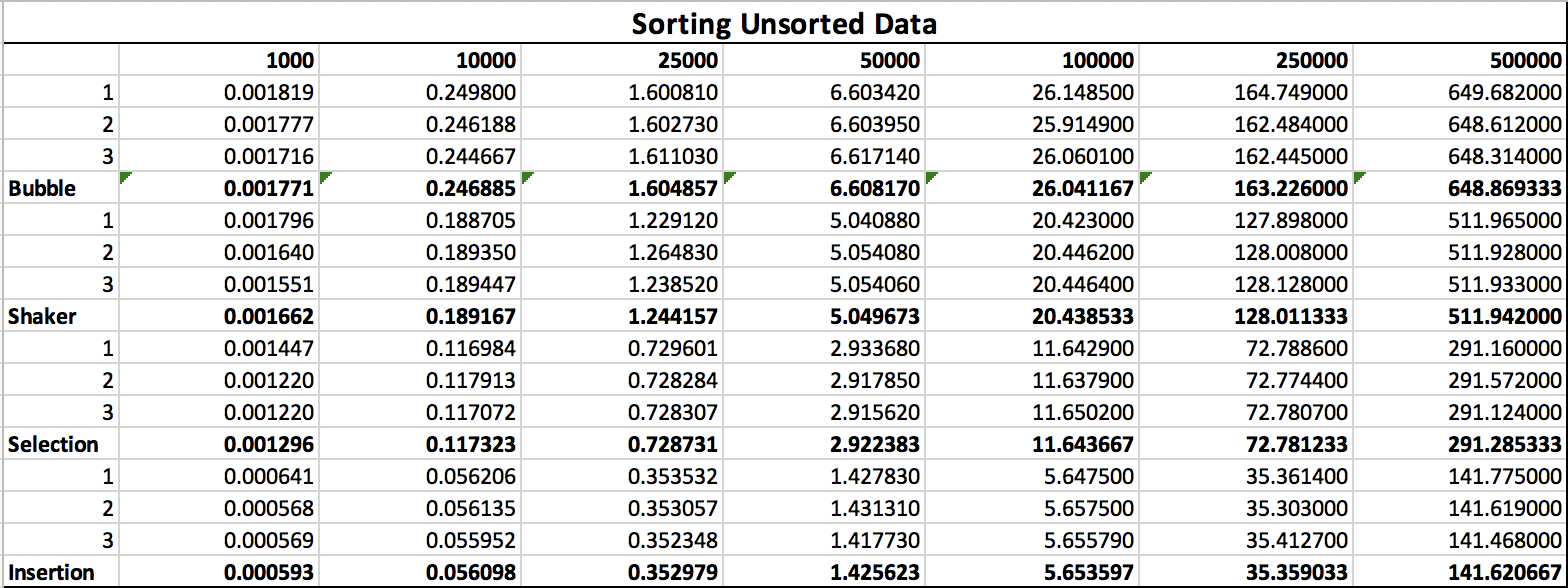
This screenshot demonstrates every feature of the code (except for file writing):

* The first command is a test of the bubble algorithm sorting 1000 integers from l2data.txt
* Next is the shaker command sorting 25,000 integers from sortedl2data.txt
* Next is a demonstration of the command line input, using the selection sort algorithm to sort 50,000 integers from the file l2data.txt
* Next is another use of command line input, using the insertion algorithm to reverse the order of the sortedl2data.txt file for the first 50,000 integers
* Last is a repeat of the previous, except for 100,000 integers



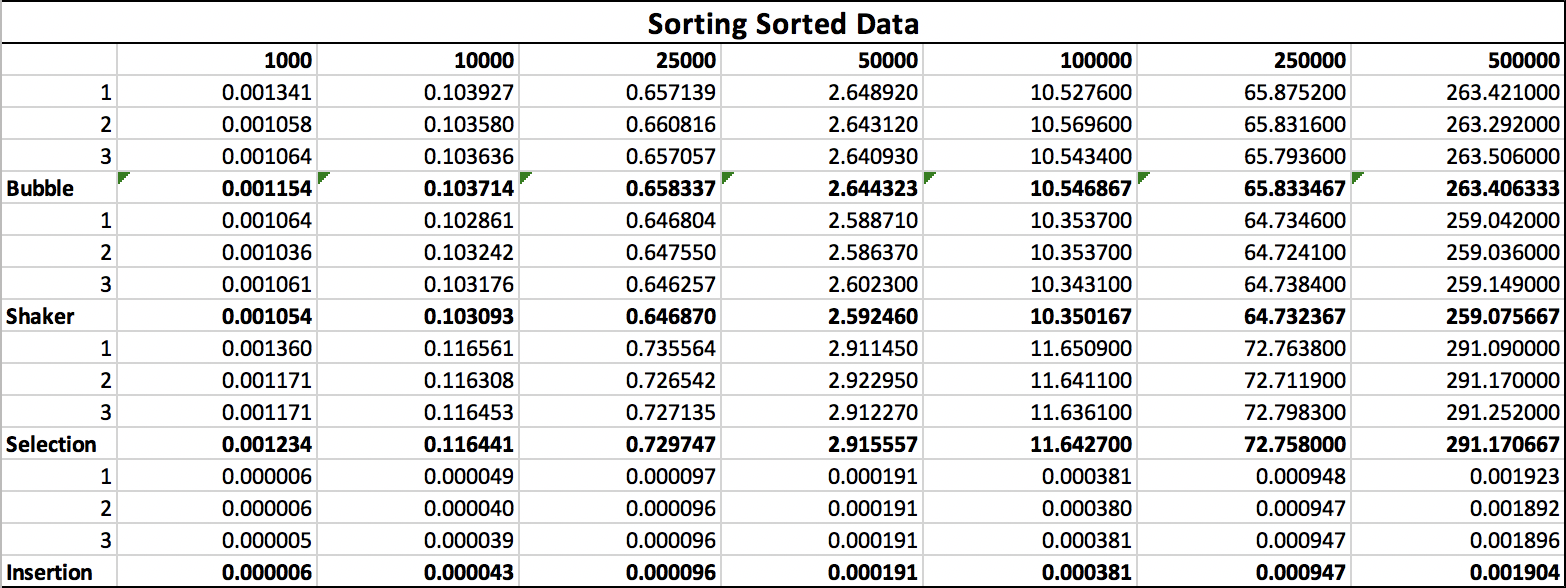
This screenshot shows one of the files written by running the algorithms. This is the list of 500,000 integers sorted from l2data.txt by the insertion algorithm, as you can see in the filename listed at the top.

**Tables of Runtimes:**

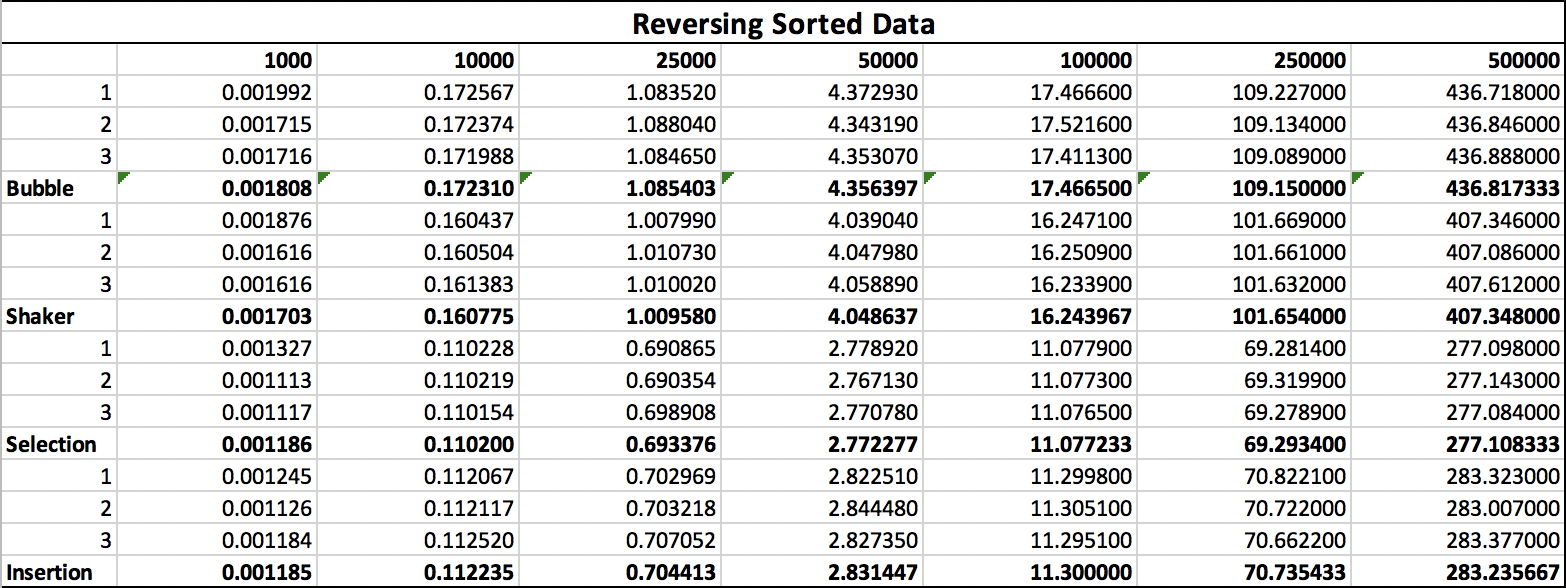


This table lists the data from every test that sorted unsorted data. Each algorithm is listed on the left, and the size of the array sorted is on the top. Each algorithm was tested three times for each size (shown as rows 1, 2, and 3), and then the average was taken (shown in bold).

It is important to note that all of these tests were run on my personal MacBook pro. I started on rabbit but discovered that my computer completed these sorts in often less than half of the time rabbit took.

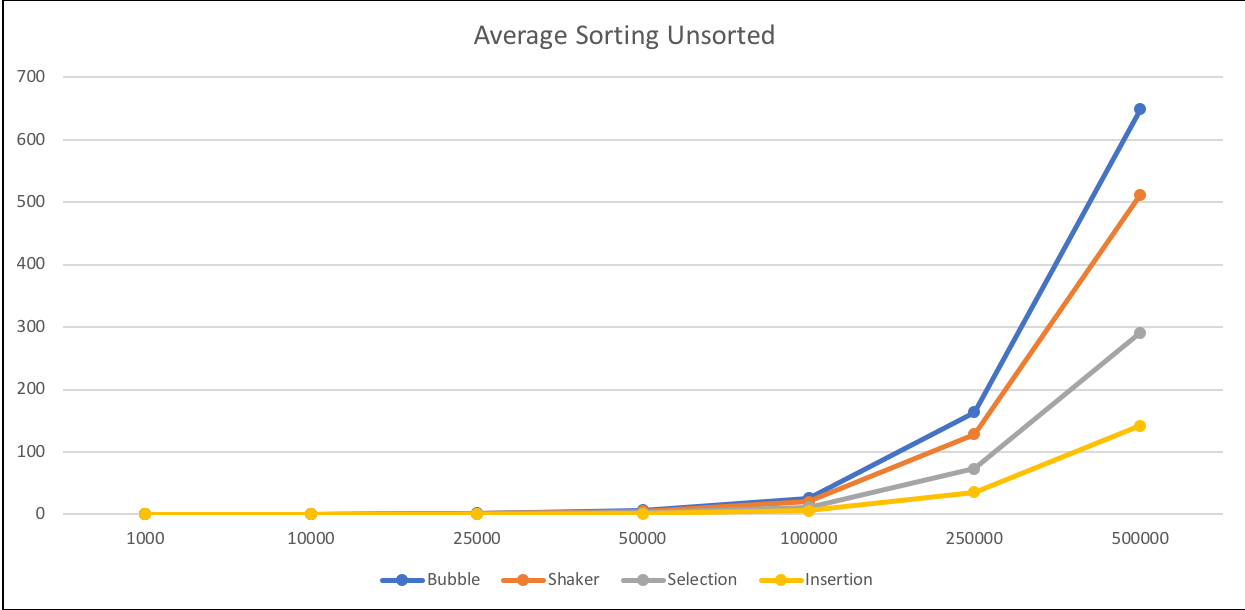


This table lists the data from every test that sorted already sorted data. The layout is exactly the same as above.

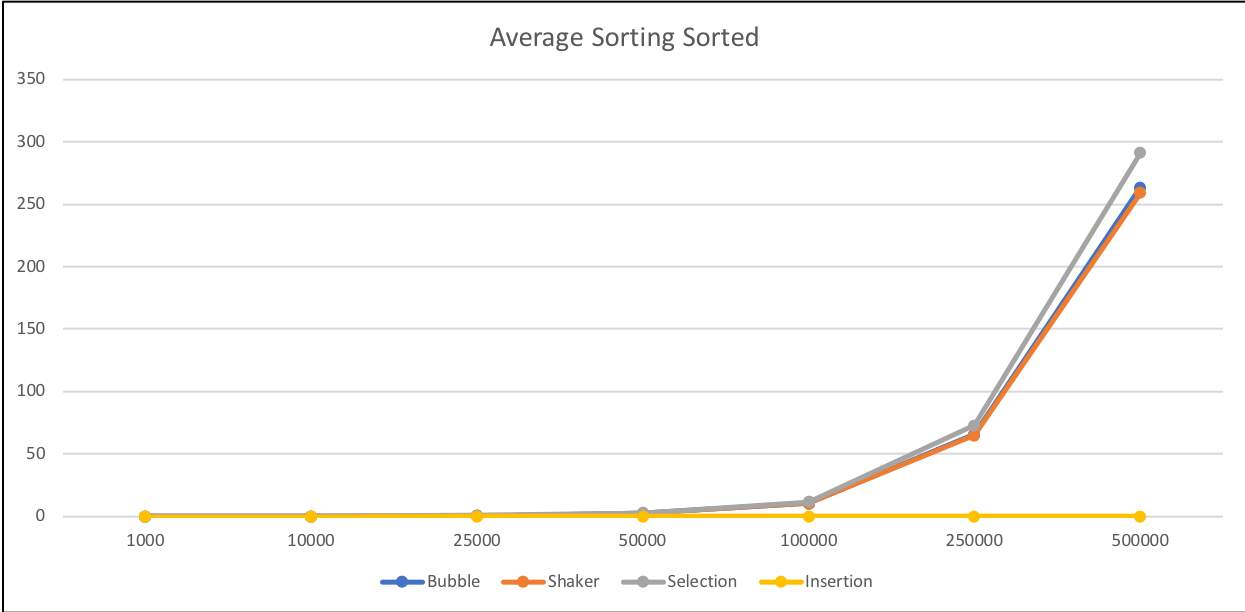


This table lists the data from every test that reversed already sorted data. The layout is once again the same as the previous two tables.

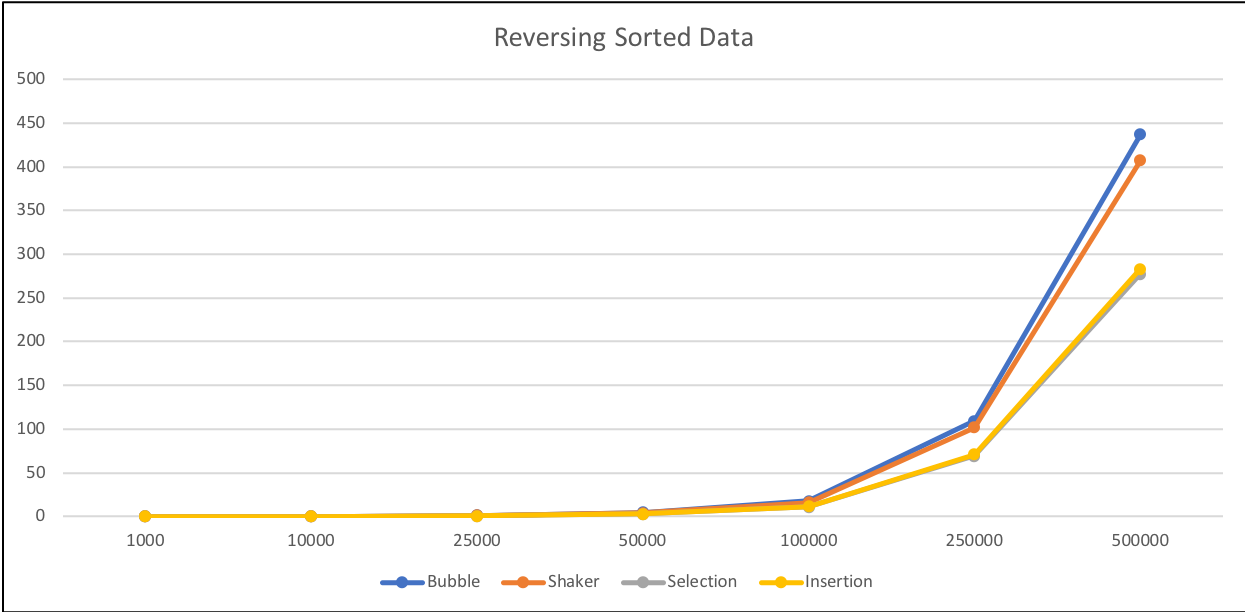
**Graphs:**



This graph shows the results of sorting unsorted data with each algorithm. The layout is fairly self explanatory. The y axis is the number of seconds taken to sort, and the x axis is the size of the array sorted. Each line represents a different algorithm as labeled at the bottom. Note that the values used for each array size was the average of the three sorts recorded above.



This graph shows the results of sorting already sorted data with each algorithm, with the same layout as above.



This graph shows the results of reversing already sorted data with each algorithm, with the same layout as above.

**Comments:**

There were several surprises once I had finished tabulating the results of the sorting. In fact, there were several things that I double and triple checked just to make sure they were correct.

Let’s start with the obvious:

The bubble sort is almost always the slowest, followed by the shaker sort. These two hang near each other most of the time with marginal differences in their sort times. Next is the selection sort, which is significantly faster than both bubble and shaker sort. Lastly is the insertion sort (meaning it is the fastest). This sort shone above the others as incredibly quick, consistently the fastest in almost every scenario.

But things started to get a little weird when you mess with already sorted data though.

Although I expected the bubble sort to be faster while sorting sorted data, I did not expect the difference to be so significant. Even weirder though is the fact that it reverses sorted data faster than it sorts unsorted data. Something about the data being sorted to begin with speeds things up, even if it has to completely reverse the data. At first, I thought that this was an error with my testing script, but after manual testing several times I found this to stay true. Sorting an unsorted file in either direction was consistently slower than sorting a sorted file in either direction.

The Shaker sort acted pretty much the same as the bubble sort. Same surprises. The only new thing that I noticed is that the difference in sorting time is greatest when the data starts unsorted. The times are much closer when the data is already sorted.

The Selection sort gave me a lot of trouble. The results confused me for a long time. It took me a while to realize that the way the selection sort works means it’s generally going to always take roughly the same amount of time. The results were almost exactly the same regardless of whether the data was sorted or not. The only offset of this observation is when reversing sorted data. In this case, the selection sort actually *speeds up* slightly, catching me completely by surprise. Lots of checking and testing made sure that this is accurate, and this leads me to believe that potentially having the data already sorted in reverse makes it easier to find the number it is selecting quickly. Lastly, it’s worth mentioning that because of the consistent speed of the selection sort, when sorting sorted data, it is actually *slower* than the bubble and shaker sorts.

Lastly, the insertion sort. This sort was obviously fast in the beginning, but the difference between the best case and the worst case was by far larger than any other algorithm. When sorting the sorted data, it was blisteringly fast, never taking over 2 one thousandths of a second to sort even 500,000 integers. This speed was difficult to believe, but more testing showed that it was accurate. When sorting the reversed data, it was actually slower than the selection sort and took *twice* the time it took to sort unsorted data.

**Source Code:**

To get all of the data from the algorithms, I originally was hand typing array sizes and algorithm names for each individual test. This quickly ramped up in time taken and was incredibly frustrating when running on rabbit. At 100,000 and above the bubble sorts were taking several minutes, and at 500,00 the bubble sorts took roughly 30 minutes to run. Even the insertions took 5-10 minutes. This was very difficult to test, as if I left my computer for even a few minutes and it went to sleep, the connection to rabbit was lost, and I had to start the test over again.

To remedy this, I wrote a script that would automatically test each algorithm 3 times in each situation for each array size, find the average of the three times, then put all three times and the average into a nice and organized file. I’ll include the script here just to help show my work process. It should work on rabbit, but I haven’t tested it there.

Note that the properly view the output of the script, you should use the command “less -r datafile.txt”. This lets less show color and makes it easy to scroll through the data. If you want to test the script with different combos of algorithms and sizes, simply modify the arrays established at the start of the file.

Files Included:

* main.cpp
* GetData.sh

**main.cpp**

#include <iostream>

#include <string>

#include <fstream>

#include "support.h"

using namespace std;

void bubble(int arr[], const int size, int argc) {

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - i - 1; j++) {

if (argc != 5) { //if argc == 5, you told it to reverse, Only usable with command line input

if (arr[j] > arr[j+1]) {

int temp = arr[j+1];

arr[j+1] = arr[j];

arr[j] = temp;

}

} else {

if (arr[j] < arr[j+1]) {

int temp = arr[j+1];

arr[j+1] = arr[j];

arr[j] = temp;

}

}

}

}

}

void shaker(int arr[], const int size, int argc) {

int start = 0;

int end = size - 1;

if (argc != 5) {

while (start < end) {

for (int i = start; i < end; i++) {

if (arr[i] > arr[i+1]) {

int temp = arr[i+1];

arr[i+1] = arr[i];

arr[i] = temp;

}

}

end--;

for (int i = end; i > start; i--) {

if (arr[i] < arr[i-1]) {

int temp = arr[i-1];

arr[i-1] = arr[i];

arr[i] = temp;

}

}

start++;

}

} else {

while (start < end) {

for (int i = start; i < end; i++) {

if (arr[i] < arr[i+1]) {

int temp = arr[i+1];

arr[i+1] = arr[i];

arr[i] = temp;

}

}

end--;

for (int i = end; i > start; i--) {

if (arr[i] > arr[i-1]) {

int temp = arr[i-1];

arr[i-1] = arr[i];

arr[i] = temp;

}

}

start++;

}

}

}

void selection(int arr[], const int size, int argc) {

int min;

for (int i = 0; i < size - 1; i++) {

min = i;

for (int j = i + 1; j < size; j++) {

if (argc != 5) {

if (arr[min] > arr[j]) {

min = j;

}

} else {

if (arr[min] < arr[j]) {

min = j;

}

}

}

int temp = arr[i];

arr[i] = arr[min];

arr[min] = temp;

}

}

void insertion(int arr[], const int size, int argc) {

for (int i = 1; i < size; i++) {

int temp = arr[i];

int j = i - 1;

if (argc != 5) {

while (j >= 0 && arr[j] > temp) {

arr[j+1] = arr[j];

j--;

}

} else {

while (j >= 0 && arr[j] < temp) {

arr[j+1] = arr[j];

j--;

}

}

arr[j+1] = temp;

}

}

int getNumber(string fileName, int arr[], const int size) {

ifstream source(fileName.c\_str());

int index = 0;

while (!source.eof() && (index < size)) {

source >> arr[index];

index++;

}

source.close();

return (index);

}

void menu(int &size, string &fileName, int &type, int argc, char\*\* argv) {

string typeString;

if (argc <= 1) { //checks if the number of arguments passed in included this information

cout << "Please enter the size of the array you wish to sort: ";

cin >> size;

cout << "Please enter the name of the file you wish to read from: ";

cin >> fileName;

cout << "Please select a sorting algorithm (bubble, shaker, selection, or insertion): ";

cin >> typeString;

type = 5;

} else {

size = atoi(argv[1]);

fileName = argv[2];

typeString = argv[3];

type = 5;

}

do {

if (type == 0) {

cout << "Please enter a listed algorithm: ";

cin >> typeString;

}

if (typeString == "Bubble" || typeString == "bubble" || typeString == "b")

type = 1;

else if (typeString == "Shaker" || typeString == "shaker" || typeString == "c")

type = 2;

else if (typeString == "Selection" || typeString == "selection" || typeString == "s")

type = 3;

else if (typeString == "Insertion" || typeString == "insertion" || typeString == "i")

type = 4;

else

type = 0;

} while (type == 0);

}

void writeFile(string fileName, int arr[], int size, int type) {

ofstream output;

string outName;

if (type == 1) {

outName = "bubble\_" + fileName;

output.open(outName.c\_str());

} else if (type == 2) {

outName = "shaker\_" + fileName;

output.open(outName.c\_str());

} else if (type == 3) {

outName = "selection\_" + fileName;

output.open(outName.c\_str());

} else if (type == 4) {

outName = "insertion\_" + fileName;

output.open(outName.c\_str());

}

for (int i = 0; i < size; i++) {

output << arr[i] << endl;

}

output.close();

}

void print(int arr[], const int size) {

for (int i = 0; i < size; i++)

cout << arr[i] << " ";

cout << endl;

}

int main(int argc, char\*\* argv) {

int size = 5;

string fileName;

int type;

menu(size, fileName, type, argc, argv);

int \* arr = new int[size];

int actual = getNumber(fileName, arr, size);

cout << "File Succesfully Read" << endl;

if (actual != size)

cout << "The requested size was " << size << " but the file only contained " << actual << " integers!" << endl;

double startTime = getCPUTime();

switch (type) {

case 1: bubble(arr, actual, argc); break;

case 2: shaker(arr, actual, argc); break;

case 3: selection(arr, actual, argc); break;

case 4: insertion(arr, actual, argc); break;

}

double endTime = getCPUTime();

double timeTaken = endTime - startTime;

cout << "Sorting took: " << timeTaken << " seconds." << endl;;

//print(arr, size);

writeFile(fileName, arr, size, type);

return 0;

}

**GetData.sh**

#!/bin/bash

# saves me a ton of time

#TEXT VARIABLES:

purpletext="\033[35m"

bluetext="\033[34m"

bold="\033[1m"

normal="\033[0m"

#HELPER VARIABLES:

sortTypes=(Bubble Shaker Selection Insertion)

arraySizes=(1000 10000 25000 50000 100000 250000 500000)

logfile=dataFile.txt

#STORAGE VARIABLES:

storage=0

data1=0

data2=0

data3=0

average=0

rm datafile.txt

printf "Script Started at " >> $logfile

date "+%H:%M:%S">> $logfile

echo >> $logfile

echo "-----------------------" >> $logfile

echo -e $bluetext"Sorting Unsorted Data"$normal >> $logfile

echo "-----------------------" >> $logfile

echo >> $logfile

for i in ${sortTypes[\*]}; do

echo -e $purpletext$i" Sort:"$normal >> $logfile #if you want to see color, you need to view the file with "less -r"

echo >> $logfile

for j in ${arraySizes[\*]}; do

echo "Array Size:" $j >> $logfile

echo "-----------------------" >> $logfile

for k in `seq 1 3`; do

storage=$(./a.out $j l2data.txt $i | grep Sorting | cut -f 3 -d " " | sed 's/e/\*10^/g')

if [ $k == 1 ]; then

data1=$storage

echo "First Sort:" $data1 >> $logfile

elif [ $k == 2 ]; then

data2=$storage

echo "Second Sort:" $data2 >> $logfile

elif [ $k == 3 ]; then

data3=$storage

echo "Third Sort:" $data3 >> $logfile

fi

done

echo "-----------------------" >> $logfile

average=$(bc -l <<< "scale=6; ($data1+$data2+$data3)/3")

echo "Average Sort Time:" $average >> $logfile

echo >> $logfile

done

done

echo "-----------------------" >> $logfile

echo -e $bluetext"Sorting Sorted Data"$normal >> $logfile

echo "-----------------------" >> $logfile

echo >> $logfile

for i in ${sortTypes[\*]}; do

echo -e $purpletext$i" Sort:"$normal >> $logfile #if you want to see color, you need to view the file with "less -r"

echo >> $logfile

for j in ${arraySizes[\*]}; do

echo "Array Size:" $j >> $logfile

echo "-----------------------" >> $logfile

for k in `seq 1 3`; do

storage=$(./a.out $j sortedl2data.txt $i | grep Sorting | cut -f 3 -d " " | sed 's/e/\*10^/g')

if [ $k == 1 ]; then

data1=$storage

echo "First Sort:" $data1 >> $logfile

elif [ $k == 2 ]; then

data2=$storage

echo "Second Sort:" $data2 >> $logfile

elif [ $k == 3 ]; then

data3=$storage

echo "Third Sort:" $data3 >> $logfile

fi

done

echo "-----------------------" >> $logfile

average=$(bc -l <<< "scale=6; ($data1+$data2+$data3)/3")

echo "Average Sort Time:" $average >> $logfile

echo >> $logfile

done

done

echo "-----------------------" >> $logfile

echo -e $bluetext"Reversing Sorted Data"$normal >> $logfile

echo "-----------------------" >> $logfile

echo >> $logfile

for i in ${sortTypes[\*]}; do

echo -e $purpletext$i" Sort:"$normal >> $logfile #if you want to see color, you need to view the file with "less -r"

echo >> $logfile

for j in ${arraySizes[\*]}; do

echo "Array Size:" $j >> $logfile

echo "-----------------------" >> $logfile

for k in `seq 1 3`; do

storage=$(./a.out $j sortedl2data.txt $i flip | grep Sorting | cut -f 3 -d " " | sed 's/e/\*10^/g')

if [ $k == 1 ]; then

data1=$storage

echo "First Sort:" $data1 >> $logfile

elif [ $k == 2 ]; then

data2=$storage

echo "Second Sort:" $data2 >> $logfile

elif [ $k == 3 ]; then

data3=$storage

echo "Third Sort:" $data3 >> $logfile

fi

done

echo "-----------------------" >> $logfile

average=$(bc -l <<< "scale=6; ($data1+$data2+$data3)/3")

echo "Average Sort Time:" $average >> $logfile

echo >> $logfile

done

done

printf "Script Finished at " >> $logfile

date "+%H:%M:%S">> $logfile

echo >> $logfile

echo All Algorithms Complete!