Coding Assignment 3

CSE3318

Your program will run and time Merge Sort and Insertion Sort. You will create multiple input file of different sizes using the file generator. You will record these run times in a provided Excel spreadsheet and graph the outputs to look for patterns. You will be asked to write a conclusion about the run times of Merge Sort and Insertion Sort.

Name your program `Code3_xxxxxxxxxx.c` where xxxxxxxxxx is your student id (not netid). My file would be named `Code3_1000074079.c`.

Please place the following files in a zip file with a name of `Code3_xxxxxxxxxx.zip` and submit the zip file.

```
Code3_xxxxxxxxxx.c

Output_xxxxxxxxxx.xls

TestFile.txt
```

A zip file is used to avoid Canvas's file renaming convention.

Reminder – ANY compiler warning(s) or error(s) when compiled on gcc 9.3.4 or Omega will result in an automatic 0. This applies no matter what the warning/error is. You will need to code so that your final program will compile cleanly and run on both gcc 9.3.4 and Omega.

Coding Assignment 3 will use a lot of code from Coding Assignment 2. You should copy Coding Assignment 2 to Coding Assignment 3.

Step 1

Add Merge Sort code to your code (remember, you copied Coding Assignment 2 to Coding Assignment 3 so Coding Assignment 3 already has Insertion Sort in it and the ability to get a file from the command line and the ability to read that file into an array – keep all of that code). You can copy the merge sort code from the lecture slides or copy it from an Internet source. The code **MUST BE** in C. If you are copying from an Internet source, make sure you are getting MERGE SORT and not some other version of sort. Your assignment will not be graded and you will be assigned a grade of 0 if the code you submit is not in C and is not merge sort.

NOTE : At this point, you can take a copy of this working code and add some print statements and practice for OLQ5. Create your own input file and see if your printouts match the values you manually came up with – start with the array explained in lecture and see if your program and your manual calculations match the OLQ5 Review and Practice document.

You should already have the Insertion Sort coding from Coding Assignment 2 – keep it in Coding Assignment 3.

The merge sort code will run and then the insertion sort code will run. They are both in the same program to give both processes approximately the same run environment (rather than put them in separate programs).

Step 2

Run your 7 files from Coding Assignment 2 on your VM/gcc 9.3.4 and on Omega. Record the number of tics output by the program for each file of *n* records in the "Output_xxxxxxxxxx.xls" file you downloaded from Canvas for Coding Assignment 3 (do not use the spreadsheet from Coding Assignment 2 – download a new one from Coding Assignment 3 – it has changed). Once you have recorded all of those values in your spreadsheet, create a chart of the "n and VM tics" and a chart of the "n and Omega tics" using the same technique used with the Big O values in the video. All 3 graphs will be in the "Results" tab of the spreadsheet.

Make sure you use the numbers from Coding Assignment 3 for the Insertion Sort tics. DO NOT reuse the tics from Coding Assignment 2. Your values will be checked and you will be given a 0 on the whole assignment if you reuse the values.

Step 3

Write a paragraph and answer the following questions

1. What do the 3 charts show you?
2. How similar did your graphs for each sort come out on the VM vs Omega vs Big O Notation?
3. Were you surprised by the differences between the run times of Merge Sort vs Insertion Sort?
4. How did your tics on Omega compare to the VM?

Write your paragraph in the box in the "Conclusions" tab of the spreadsheet.

Notes

Run Merge Sort on the array and then `free()` it and `malloc()` it again and fill it again and then run Insertion Sort. Do not run both sorts on the same copy of the array. Do not try to make a copy of the array – this assignment uses VERY large arrays and you will not have enough memory to hold 2 copies.

Using `TestFile.txt`, run your new version and confirm that it sorts correctly and uses the file as input by compiling your code with the conditional compile.

```
gcc Code3_xxxxxxxxxx.c -D PRINTARRAY
```

It will be harder and harder to see the proper sorting as your file grows so confirm your coding with smaller tests. The conditional compile will allow the GTA to test your program as well. Compile without the conditional compile when you start running the tests so that you do not print the larger arrays.

Psuedocode for main()

```
main
     declare array pointer
     Call function to read file into array

     #ifdef PRINTARRAY
     Print the array (should be unsorted)
     #endif

     Start the clock for Merge Sort
     Call Merge Sort
     Stop the clock for Merge Sort

     #ifdef PRINTARRAY
     Print the array (should be sorted)
     #endif

     Free the malloced pointer

     Call function to read file into array

     #ifdef PRINTARRAY
     Print the array (should be unsorted)
     #endif

     Start the clock for Insertion Sort
     Call Insertion Sort
     Stop the clock for Insertion Sort

     #ifdef PRINTARRAY
     Print the array (should be sorted)
     #endif

     Free the malloced pointer

     print how many records were in the file/array

     print how many tics were used by Merge Sort
     print how many tics were used by Insertion Sort
```

Without the conditional compile (sample data – your data and results will be different)

```
./a.out TestFile.txt
Processed 10 records
Merge Sort     = 3
Insertion Sort = 1
```

With the conditional compile (sample data – your data and results will be different)

```
./a.out TestFile.txt
74079
21259
59758
25398
4381
62060
61981
59743
21162
32541


4381
21162
21259
25398
32541
59743
59758
61981
62060
74079


74079
21259
59758
25398
4381
62060
61981
59743
21162
32541
```

```
4381
21162
21259
25398
32541
59743
59758
61981
62060
74079


Processed 10 records
Merge Sort     = 3
Insertion Sort = 1
```