Coding Assignment 2

CSE3318

Your program will run and time Insertion Sort. You will create multiple input files of different sizes using the file generator. You will record these run times in a provided Excel spreadsheet and graph the outputs to see how your actual run times compare to the Big O values.

This assignment is Part 1 of a 2 part assignment so it is important that you follow the directions as given so that you will not run into issues turning Coding Assignment 2 into Coding Assignment 3 later.

Name your program `Code2_xxxxxxxxxx.c` where xxxxxxxxxx is your student id (not netid). My file would be named `Code2_1000074079.c`.

Please place the following files in a zip file with a name of `Code2_xxxxxxxxxx.zip` and submit the zip file.

`Code2_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 using gcc 9.4.0 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 9.4.0 and Omega.

All references to "VM" in this assignment should be understood to be whatever environment you are using to compile your C code with gcc 9.4.0. I understand that not everyone is using the VM but your non-Omega environment is referenced that way throughout the document for clarity.

Be sure to reuse your code from Coding Assignment 1 – you should NOT be writing Coding Assignment 2 from scratch.

Step 1

Create a program than can run Insertion Sort on a hardcoded small array. You can copy the 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 INSERTION 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 insertion sort.

Step 2

Using FileGenerator.e that you used as part of Coding Assignment 1, generate the following 7 files. You can name them what you want but you need to create each file containing the required number of records per file. Create them on Omega and then copy them to whatever other system you are using to run your C programs. Do NOT create them on both systems – you want to use the exact same files on both systems.

Create 7 files where the first file contains 1024 records and the second contains 10,000 records, etc.

1,024
10,000
50,000
100,000
500,000
1,000,000
2,000,000

Run the file generator one more time and create a file of 10 records. Use this file for testing and submit in your zip as "`TestFile.txt`".

Step 3

Add the ability to accept command line parameters. **You already have this code from Coding Assignment 1.**

Step 4

Add the ability to read a file name from the command line. **You should already have this code from Coding Assignment 1.**

Using `fgets()`, loop through the file and count the number of lines in the file. Use `fseek()` to move the file pointer back to the beginning of the file (See "File Handling in C" in the "Review Materials" module of Canvas).

Step 5

Use `malloc()` with the number of lines you counted in the file to dynamically create an `int` array on the heap. Now read through the file again and put each line (each line is a number) in an array element. This process will allow you to create arrays of various sizes.

You will be running Insertion Sort on this array and then you will `free()` the memory since you used `malloc()` to reserve the memory (don't cause a memory leak).

Step 6

Add a function to print the array. Here is a suggested version – you are not required to use this exact code but your version must output the same – one number per line.

```
void PrintArray(int ArrayToPrint[], int SizeAP)
{
      int i;

      for (i = 0; i < SizeAP; i++)
            printf("%d\n", ArrayToPrint[i]);
}
```

Add conditional compile statements around the call to this function in `main()` where it is called before and after sorting the array.

```
      #ifdef PRINTARRAY

      PrintArray(AP, elements);

      #endif
```

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 Code2_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.

Step 7

Add the code to time your sort. **You should already have this code from Coding Assignment 1**. To do this, declare two variables of type `clock_t`

```
clock_t start, end;
```

You then call the `clock()` function to get the start time. You will call the `clock()` function again after the insertion sort completes to get the end time.

```
start = clock();
```

Call Insertion Sort

```
end = clock();
```

print the total tics by subtracting `start` from `end`

Be sure to only include the actual sort code in the time – do not include reading the file or setting up the array. We are just going to use the actual number of clock tics so do not divide your total ticks by anything. Print out the tics needed by insertion sort (see video for example).

Step 8

Download "Output_xxxxxxxxxx.xls" from Canvas. Change the x's to your student id and save the spreadsheet. You will be submitting this spreadsheet with your assignment. Watch the video "How to Chart" to see how the chart of the Big O values that is already in the spreadsheet was created. You will need to be able to repeat this process with the Omega and VM data you will be gathering. PLEASE NOTE THAT THE VIDEO SHOWS COLUMNS FOR BOTH MERGE SORT AND INSERTION SORT – THIS ASSIGNMENT IS ONLY FOR INSERTION SORT. Coding Assignment 3 will add Merge Sort.

Step 9

Run your 7 files on your VM and on Omega. Record the number of tics output by the program for each file of *n* records for your Insertion Sort. 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.

NOTE : it is highly likely that Omega will timeout and shut down your terminal before your larger files can finish running on Omega. If this happens, then you need to output your results to a file and run the process in the background. See video "Run background process" to see an example of how to do this.

Sample output

The numbers shown in this SAMPLE output are just examples – you should not worry about getting the exact same numbers.

```
gcc Code2_1000074079.c -D PRINTARRAY
./a.out TestFile.txt
74079
21259
59758
25398
4381
62060
```

```
61981
59743
21162
32541



4381
21162
21259
25398
32541
59743
59758
61981
62060
74079



Processed 10 records
Insertion Sort = 2 Tics
```

Notice that the array is printed BEFORE and AFTER the insertion sort runs.

Compile the program without the array printing turned on and running it with a file containing 1,000,000 records.

```
gcc Code2_1000074079.c
./a.out N1000000.txt
Insertion Sort = 694247174 Tics
```