

Coding Assignment 2

CSE2320

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 on Omega will result in an automatic 0. This apply no matter what the warning/error is. You will need to code so that your final program will compile cleanly and run on both the VM and Omega.

For example, if the GTA compiles your code on Omega and sees

```
[frenchdm@omega CA1]$ gcc Code2_1000074079.c
```

```
Code2_1000074079.c:44:2: warning: no newline at end of file
```

you will receive a 0 on the assignment.

Step 1

Do not remove any code from `Code2_XXXXXXXXXX.c` unless specifically told to do so. We are going to reuse most of the first assignment in this second assignment.

Replace the call to Merge Sort with a call to Quick Sort. 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 QUICK 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 or is not Quick Sort.

Remove the Insertion Sort functions. This program will only run Quick Sort.

Step 2

As in the previous assignment, open the file from the command line, read through the file to count the records, `malloc()` the array, read file into array, call Quick Sort and `free()` the array. Calculate the number of tics using the `clock()` as was done in the previous assignment.

Step 3

Using the conditional compile statement `PRINTARRAY` and `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 4

In the `partition()` function, add the conditional compile statements for running Quick Sort with different pivots. You will need to run Quick Sort with a right pivot, middle pivot and random pivot. Adding and using the conditional compile statements will be discussed in class.

```
#if QSM
int middle = (high+low)/2;
swap(&A[middle], &A[high]);
#elif QSRND
int random = (rand() % (high-low+1)) + low;
swap(&A[random], &A[high]);
#endif
int pivot = A[high];
```

Step 5

In order to get a good average run for each type of sort, you will add a for loop to your program. This for loop will run Quick Sort multiple times so the mallocing of the array, filling the array from the file, calling Quick Sort and freeing the memory should be inside the for loop. The counter for the for loop will be obtained from the command line arguments – `argv[2]`. Convert `argv[2]` to an `int` and use it in your for loop. Keep a running total of tics. Calculate the average tics after the for loop finishes.

Step 6

Download “Output_xxxxxxxx.xls” from Canvas. Change the x’s to your student id and save the spreadsheet. You will be submitting this spreadsheet with your assignment. You will be filling in the run times and charting them. You will create on chart that contains the run times for all 3 versions of Quick Sort on both Omega and the VM.

| n | VM tics | | | Omega tics | | |
|-----------|---------|-----|-------|------------|-----|-------|
| | QSR | QSM | QSRND | QSR | QSM | QSRND |
| 1,024 | | | | | | |
| 10,000 | | | | | | |
| 50,000 | | | | | | |
| 100,000 | | | | | | |
| 500,000 | | | | | | |
| 1,000,000 | | | | | | |
| 2,000,000 | | | | | | |

Step 7

Compile your program to run with a right pivot. If you don’t use any conditional compile directives, your executable will use the right pivot.

Run your 7 files on your VM and on Omega. Pass in the file name and 100 as the number of times to run Quick Sort on the file. Record the average tics output by the program for each file of *n* records.

Conditional compile your program to run with a middle pivot and repeat running the files and recording the results.

Conditional compile your program to run with a random pivot and repeat running the files and recording the results.

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 in the previous assignment. Both graphs will be in the “Results” tab of the spreadsheet. Each chart will compare the run time of the 3 sorts on each machine.

Step 8

Download the zip file named T100.zip from Canvas. When you unzip the file, you will have 5 files, T1.txt, T2.txt, T3.txt, T4.txt, and T5.txt. Copy all 5 files to both your VM and Omega. These are the files with 100,000 records each. You will use these files to fill in the second section of the spreadsheet.

| VM tics - 100,000 | | | | | Omega tics - 100,000 | | | | |
|-------------------|---------------------------|---------------------------|---------------------------|---------|----------------------|---------------------------|---------------------------|---------------------------|---------|
| T1 | T2 | T3 | T4 | T5 | T1 | T2 | T3 | T4 | T5 |
| Random | 1/4 ordered 3/4 random | 1/2 ordered 1/2 random | 3/4 ordered 1/4 random | Ordered | Random | 1/4 ordered 3/4 random | 1/2 ordered 1/2 random | 3/4 ordered 1/4 random | Ordered |
| QSR | | | | | | | | | |
| QSM | | | | | | | | | |
| QSRND | | | | | | | | | |

Using the right pivot version of your program, process all 5 files for 100 runs each and record the average run time for each file. Repeat this process for the middle pivot and random pivot versions.

Step 9

Write a paragraph (at least 3 sentences) about your conclusions. What do your graphs show you about the runtime of the various versions of Quick Sort? Is the overhead of calculating the pivot (middle pivot and random pivot) worth it? How does Quick Sort compare to Merge Sort and Insertion Sort? Insertion Sort and Quick Sort have the same worst run time of $O(n^2)$. In class, it was said that Quick Sort general does better in practice. Do you agree with this after running your tests? How did the “orderness” of the file effect the different forms of Quick Sort? Did certain versions do better or worse with the completely ordered/sorted file? Based on your tests, what version of Quick Sort would you recommend? Would your recommendation differ based on the data being sorted?

Write your paragraph in the box in the “Conclusions” tab of the spreadsheet.