Lab 0

Concurrent Programming

Due: 11:59pm, September 4, 2019

This lab is a "warm-up" lab to make sure everyone's infrastructure is working and to get the easy part out of the way.

Summary: Write, in C or C++, a program that sorts a text file of unique integers and outputs the sorted list, and write-up a description of the project. In effect, you are emulating the performance of the UNIX `sort -n` command. This lab is an individual assignment.

Code Requirements / Restrictions You must implement both of the following sorting algorithms:

- Quicksort
- Mergesort

You are allowed to use pre-written unsorted data structures (e.g. STL's vector), but you may not use any pre-written sorted data structure or sorting algorithm (e.g. STL's map).

Lab write-up: Your lab write-up should include:

- A brief description of your code organization
- A description of every file submitted
- Compilation instructions
- Execution instructions
- Any extant bugs

I expect your lab write-up for this project will be around a page or two.

Code style: Your code should be readable and commented so that the grader can understand what's going on.

Submission: You will submit a zip file of your lab to canvas. When unpacked, the directory should contain all files required to build and run your program, along with a brief write-up. Pay particular attention to the requirements for compilation and execution, as some testing will be done using automatic scripts.

Compilation and Execution: Your submitted zip file should contain a Makefile and the project should build using a single `make` command. The generated executable should be called `mysort`. The `mysort` command should have the following syntax:

```
mysort [--name] [sourcefile.txt] [-o outfile.txt] [--alg=<merge,quick>]
```

Using the `--name` option should print your name. Otherwise, the program should sort the source file using the algorithm selected in the `alg` argument. The source file is a text file with a single integer on each line. The `mysort` command should then sort all integers in the source file and print them sorted one integer per line (as would be done by the `sort -n` command to an output file (specified by the `-o` option). See Figure 1 for `mysort` syntax examples. The `getopt` and `getopt_long` method calls are helpful for parsing the command line. You can assume that all input values are non-negative, less than or equal to `INT_MAX`, and that there are no duplicates.

Testing: You can generate test input files using the UNIX `shuf` (shuffle) command (see Figure 2 for examples using a bash shell). Using this input file you can compare your results with `sort` using the `cmp` command, which checks if files are equivalent.
I *highly* recommend you test your code using this methodology... it's how we'll grade you!

Grading: Your assignment will be graded as follows:

Unit tests (70%) We will check your code using fourteen randomly generated input files (generated by `shuf` and checked by `cmp`, see Figure 3). Correctly sorting a file is worth five points.

Lab write-up and code readability (30%) Lab write-ups and readable code that meet the requirements will get full marks. Incomplete write-ups or unreadable code will be docked points.

Recall that late submissions will be penalized 1% for every two hours late up to two days, and will be accepted up to four weeks late. Canvas submissions include the submission time.

```
### print your name
./mysort --name
# prints:
Your Full Name

### Consider an unsorted file
printf "3\n2\n1\n" > 321.txt
cat 321.txt
# prints
3
2
1

### Sort the text file and print to file
./mysort 321.txt -o out.txt --alg=quick
cat out.txt
# prints:
1
2
3
```

Figure 1: Examples of your `mysort` program's syntax

```
### To generate a random test file ###
# -i1-10 is the range (1 to 10)
# -n5 is the length (chose 5 numbers from the range)
# testcase.txt is the output shuffled file
shuf -i1-10 -n5 > testcase.txt

### To sort a text file with linux ###
# -n is cast each line to integers
# testcase.txt is the shuffled file
# testsoln.txt is the sorted file
sort -n testcase.txt > testsoln.txt

### To compare two text files ###
# e.g. to verify your program's correctness
# Note that line endings matter!
cmp --silent myoutput.txt testsoln.txt && echo "Same!" || echo "Different!"
```

Figure 2: Several useful shell commands for this assignment

```
### Generate a test file
### (of unspecified range and size)
shuf -i0-2147483643 -n382 > case1.txt

### Sort it using sort
sort -n case1.txt > soln1.txt

### Run your mysort program to also sort the test file
./mysort case1.txt -o your1.txt --alg=merge

### Compare test results
cmp --silent your1.txt soln1.txt && echo "Pass (5pts)" || echo "Fail (0pts)"
```

Figure 3: How we will grade your code