# Lab 3

Concurrent Programming

Due: 11:59pm, November 9, 2020

This lab should familiarize you with using OpenMP to parallelize code.

**Summary:** In this lab you'll parallelize a sorting algorithm from Lab 0 using OpenMP, either MergeSort or QuickSort. You are welcome to either reuse or rewrite your implementation.

**Code Requirements / Restrictions:**
Your code should leverage the OpenMP library to parallelize code. You should not use pthreads or C/C++ atomics to manage or synchronize threads. As with Lab 0, you need to write the sorting code — you can't use a prewritten sorted data structure.

**Lab write-up:** Your lab write-up will be relatively short. It should include:

- A comparison between the ease of parallelization between pthreads and OpenMP.
- A description of your parallelization strategies
- 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. Your makefile will generate one executable with effectively the same syntax as Lab 0, that is:

```
mysort [--name] [sourcefile.txt] [-o outfile.txt]
```

Using the `--name` option should print your name. Otherwise, the program should sort the source file. 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).

**Testing:** Testing can be done using the same methodology as Lab 0 (e.g. `shuf`, `sort`, and `cmp`).

**Grading:** Your assignment will be graded as follows:

**Unit tests (80%)** We will check your code using sixteen randomly generated input files. Correctly sorting a file is worth five points.

**Lab write-up and code readability (20%)** 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 10% per day late, and will only be accepted for three days after the due date. 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
cat out.txt
# prints:
1
2
3
```

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