All exercises must be done by yourself. You may discuss questions and potential solutions with your classmates, but you may not look at their code. If in doubt, ask the instructor.

Acknowledge all sources you found useful. Your code should compute the correct results.

Partial credit is available, so attempt all exercises.

Submit your code and answers in a single archive file (ZIP/TGZ/TBZ2, etc.) that contains your name in the filename (e.g. `JRandom_A1.zip`). Your answers should be a PDF file in the archive.

In this assignment, you will implement two concurrent queues, one blocking and another non-blocking, and use them in an asynchronous parallel version of SSSP.

Your programs will be tested on `node2x14a`, `node2x18a` and `node-ibm-822.csug`. Modify the makefile so that `make` *`binaryname`* makes the binary.

When reporting times below for each question, report times for 1, 2, 4, 8, and N_{max} threads along with which machine you used to obtain the time. As in Assignment 2, repeat each experiment and report averages and standard deviation.

Here, N_{max} is the maximum number of hardware thread contexts supported on a machine. $N_{max} = 56$ for `node2x14a`, $N_{max} = 72$ for `node2x18a` and $N_{max} = 160$ for `node-ibm-822.csug`.

Check for correctness against the serial version using `test.sh` command.

Exercise 1

Implement a blocking concurrent queue that has the same interface as serial queue using a single lock per queue.

Write a parallel test harness for this queue. The harness is not graded, but is useful to quickly identify issues. Submit the implementation of this queue as `blocking_queue.h`.

Exercise 2

Implement a *lock-free* concurrent queue. You can use any of Michael and Scott queue (from MLS), the Herlihy and Wing queue (on page 475 of their paper), or the variant of the M&S queue described in Herlihy and Shavit (the HS textbook).

Note: this exercise involves significant effort, start *now*.

Use the harness from E1 to test this queue. Submit the implementation of this queue as `lockfree_queue.h`.

All code you submit here must be your own.

Exercise 3

Parallelize the `sssp` function using the two queue implementations above to obtain asynchronous and (in theory) much faster versions of SSSP than the one you wrote in A2. Use lock-free atomic read/modify/writes to handle data races within the SSSP algorithm. Note that you cannot exit a thread until the queue is empty *and all* threads are done (i.e. no new work is guaranteed to appear on the queue). Do not use barriers (i.e. do not implement a synchronous version).

1. Call the binary for sssp using the concurrent queue from E1 as `sssp_blocking`. Report the performance for all inputs for varying number of threads.
2. Call the binary for sssp using the lock-free queue from E2 as `sssp_lockfree`. Report the performance for all inputs for varying number of threads.
3. How does your implementation of the lock-free queue compare with the blocking queue at N_{max} threads?

END.