CS 441
Final Project, Spring 2021

This project is intended to give you some exposure and introductory experience with functional programming, and to re-familiarize yourself with the effects of concurrency. The final project will be written in either Clojure or Racket; both are LISP descendants and have similar programming styles. See the notes below for some thoughts about pros and cons of each language.

The program itself is quite simple, merely sorting a large list of unsigned 32-bit integers. Your program will read the data from a large file, then measure the time needed to sort them using a standard algorithm (quicksort and mergesort are both simple to code in functional languages). The important thing is that the algorithm uses some form of divide-and-conquer strategy, in which there is more than one recursive call made. You will then compare how long it takes to carry out the same task using various multithreading options. Finally, you will prepare a short report summarizing your findings and explaining why you saw the results you did.

Program input is a text file with one million integer values, 1 per line. During development, you may want to start with a smaller subset of items—100, then 1000, etc.--for faster turnaround. (Likewise, if the running time is getting inconveniently long with less than the whole file, it's OK to use a subset; just discuss that in your report.)

Choice of programming language

Clojure runs on the Java Virtual Machine, and so has access to the full Java API. This makes managing multiple threads fairly easy, setting up a thread pool of specified size to handle the recursive calls. As the JVM allows system-level programming, the programmer has fine control over the number of threads, and many of you are already familiar with the Java I/O and error-handling functions. *But*, Clojure is a mostly-volunteer project, and has fallen in and out of careful maintenance. It appears to be on something of a down period lately, so there may not be much support out there.

Developing in Clojure: The easiest development environments for Clojure are Eclipse, which has a Clojure plug-in, or CounterClockwise, which is a standalone Eclipse editor/IDE. The default Clojure IDE, *Leinengen*, is favored by enthusiastic Clojure developers but isn't particularly beginner-friendly. The Eclipse plugin and CounterClockwise are both installed on Flarsheim labs and, like Clojure, are free downloads if you want to install on your own machine.

If you develop in Clojure: Time your sorting algorithm using 1, 2, 4, 8, 16, 32, and 64 threads. Run the program 3 times under each condition and report the average time.

Racket was developed as part of an NSF program, designed as a language optimized for developing domain-specific languages. (The standard starting point for that, if you're interested, is the book *Beautiful Racket*, available online.) It is not intended for system-level programming, but is designed as a relatively high-level functional language. Concurrency options are much more limited; `futures` allows running a function call in a separate thread on the same core, and `places` allows running it on a different core. Its documentation is professionally written and Racket is, in general, better maintained than Clojure.

Developing in Racket: Racket can run from the command line, so you can use the text editor of your choice, just as you could for Clojure. The standard development environment for Racket, *DrRacket*,

includes a syntax-highlighting editor and REPL (read/evaluate/print loop) console similar to Python. While Clojure can piggyback on the rich variety of Java IDEs, Racket has a smaller range of development tools to choose from.

If you develop in Racket: Compare timings of your program with no parallelism, with **futures** only, with **places** only, and with both. Run the program 3 times under each condition and report the average time.

Submission: Submit your source code and your report, either uploading the files to Canvas, or providing a link to a public repository on GitHub (or BitBucket, or whatever system you use). The report will probably be about 2-4 pages; include a graph of your running times as a function of the number of threads (or degree of parallelism, in Racket). Explain why you got the results you did; disciplines speculation is OK here. If something about your results surprised you, discuss that as well.