CS 137: Algorithms in the Real World — Spring 2022

Course Syllabus

Information At-A-Glance

Instructor	
Name:	**Adam Blank**
E-mail:	blank@caltech.edu
Office:	ANB 115
Office Hours:	TBD Or by private meeting.

Course Website
https://algos.world Visit early. Visit often.

Lecture
No lecture.

Course Overview

This course introduces algorithms in the context of their usage in the real world. The course covers compression, advanced data structures, semi-numerical algorithms, cryptography, computer algebra, and parallelism. The goal of the course is for students to see how to use theoretical algorithms in real-world contexts, focusing both on correctness and the nitty-gritty details and optimizations. Implementations focus on two orthogonal avenues: speed (for which C is used) and algorithmic thinking (for which Python is used).

Because this is a lot of material, we had to choose topics to *focus* on at the expense of the remaining topics. In particular, we have chosen lossless compression, multi-precision integer operations, RSA cryptography, parsing, and string matching.

Expectations

We expect that *coming into the course*, you...

- have prior exposure (but not necessarily proficiency) with the C programming language
- have taken CS 24 or equivalent
- are ready to work in a team
- will focus on learning and not grades

Alpha Build

This term is the *alpha build* of this course. Your instructor hopes that the materials are polished enough, but there will inevitably be issues with some of them. If you find a typo, mistake, or clarity issue, please bring it to Adam's attention as soon as possible. We will keep track of student contributions and this sort of help can contribute to a bump (up) in your final grade.

Assessment

There will be no exams in this course. The only assessments are "sprints" which are two week periods of time with a deliverable at the end. During this course, your group will participant in *four* two-week sprints and a poster presentation in week nine. There will be no work or content due in week 10 or finals week due to seniors not being available during that time.

Sprints

During each sprint, teams will choose between (1) completing a new "mini-project" and (2) "extending" a mini-project that they completed in a previous sprint. There is no limit on how many mini-projects or extensions each group can do beyond the fact that there are only four sprints during the quarter.

Mini-Projects

There are four potential mini-projects that each team can complete:

`git` In this mini-project, teams will write their own `git` utility capable of reading, writing, sending, and receiving "blobs" as per the `git` specification. This mini-project is large in scope and will have a significant starter codebase to compensate. This mini-project can be completed in any of the following programming languages: C, Rust.

`grep` In this mini-project, teams will write their own `grep` utility starting from minimal parsing code and explore string matching and regex matching algorithms currently in use. This mini-project can be completed in any of the following programming languages: Kotlin, Rust.

`ssh` In this mini-project, teams will write their own `ssh` key generation library which consists primarily of a big-number implementation. This project focuses nearly entirely on semi-numerical algorithms and has some but not a significant amount of starter code. This mini-project must be completed in C.

`zip` In this mini-project, teams will write their own `myzip` and `myunzip` utilities starting from no starter code and explore compression algorithms currently in use. This mini-project can be completed in any of the following programming languages: C++, Rust, Zig, or D.

Extensions

In lieu of going for breadth by completing every mini-project, a team can choose to extend a previously completed mini-project with a mini-research project which we call an "extension". Extensions should have (1) a non-code deliverable (readme, writeup, etc.), and (2) an artifact implemented, some sort of code or analysis that can be explained in the non-code deliverable. Beyond that, there are very few restrictions. Some possible directions to go in are: performance optmization, security audits, and comparisons to other real implementations.

Please note that this is a 12-unit course with no lecture; so, we expect these extensions to be interesting and quite a bit of work. That is, 12 units $\times$ 2 team members $\times$ 2 weeks ≈ 48 hours of effort on a single extension.

Weekly Check-ins

Every group must go to a weekly check-in with a member of course staff. At this check-in, teams will be presented with (1) a current grade for the week and (2) potentially test results run on our hidden tests that can be fixed to increase the current grade for the following week. More details about this later.

Getting Help

Please don't be afraid to ask for help if you don't understand something. Adam holds *at least three* office hours a week, and they get lonely and bored if you don't show up!

Here's some first steps on how to get help:

- Come to office hours
- Ask someone on course staff questions before/after lecture, etc.
- Post on Ed asking a question

Collaboration & Academic Integrity

Our collaboration policy boils down to "be reasonable". You may not read or discuss any code written by anyone except your partner. You may not write, copy, or modify code for any other group. You may not read, copy, or modify implementations (or pseudocode) of the projects (or parts thereof) found on the internet. You may discuss high-level design decisions with other groups. You may share tests with other groups provided that you share them with the course staff as well. Any level of collaboration between groups beyond the items discussed above is considered a violation of this policy. We reserve the right to modify or clarify this policy as needed.