Applied Algorithms Lec 1: C Review; Analyzing Efficiency

Sam McCauley

September 5, 2025

Williams College

Welcome!

- Welcome back to campus.

- Can everyone see me and the projector?

Admin

- Colloquium Fridays at 2:30
- Some attendance required for majors
- Welcome colloquium today

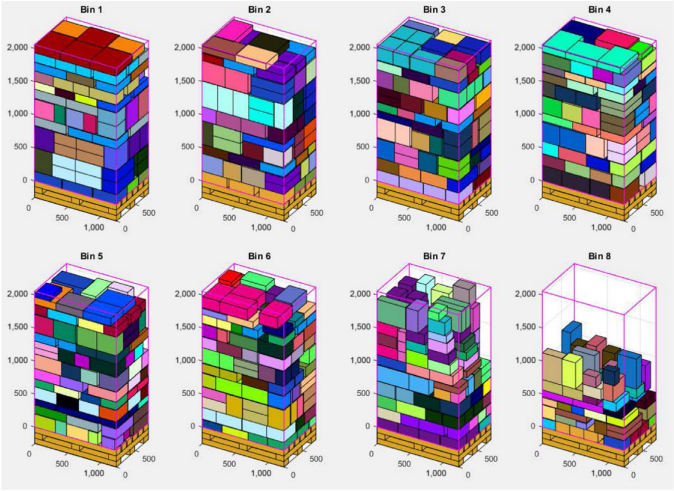
About the Class

- Goal: bridge the gap between theory and practice
- How can theoretical models better predict practice?
- Useful algorithms you may not have seen
- Using algorithmic principles to become better coders!

Pantry Algorithms

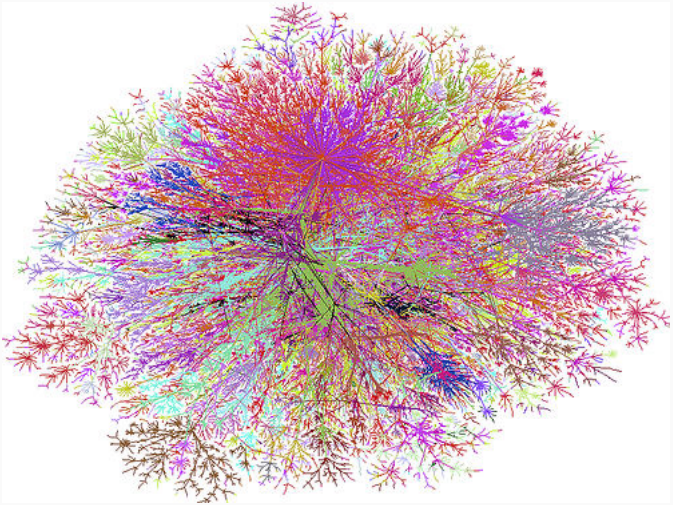
- Algorithms that you should always have handy because they are incredibly useful
- Bloom filters, linear programming, suffix trees
- What drives the course
- Algorithmic understanding of these ideas!

Power of Modern Algorithms



A shipping company needs to efficiently pack items into its truck. How can we use algorithms to find a good, or even the best, solution?

Power of Modern Algorithms



How far away are two average people in the Facebook graph? $O(n^2)$ doesn't work when n is in the billions!

Coding

- We’ll be doing some coding practice each week
- Code review from time to time
- Collaboration highly encouraged

About Me

- Call me Sam (he pronouns)
- Research is in algorithms
    - Some experimental algorithms
- Office is TCL 306 for now; moving to TPL 304 soon
- (Extra) office hours Monday 12–1:30

About the Course

- No course textbook
  - We will use Kleinberg-Tardos once; I'll give you access to relevant pages in case you don't already have it
- Questions *particularly* welcome!
- From last time this course was taught:
  - Some grading changes
  - Focus more on algorithmic and implementation ideas, less time on C debugging

Help, Questions, Comments, Etc.

- Email `sam@cs.williams.edu`
- During or after class
- Stop by the lab during (or not during) office hours
- Stop by my office (no promises!)
- Ask our TA, Alex Root
- Help each other!

Labs

Labs

Labs

- TCL 312; card access
- Thursdays; you should have signed up for a slot. You do not need to stick to your slot.
- (Not open yet; should be by the time we have lab.)
- Lab is *optional* this coming week since there is no real assignment
- No food or drink in lab

Assignments: Pdf portion

- A small number of algorithmic problems and experiments each week
- Don’t fall behind! (Or get too distracted by coding)
- Goal: Understanding how the algorithms work
- Especially important on the midterm and final

Assignments: Coding portion

- (Almost) all in C
- Weekly assignments
- Assignment 1 is designed to give you an opportunity to catch up
- Coarse grading
- (Mostly) no parallelism in this course

Why C?

```
register short *to, *from;
register count;
{
    register n = (count + 7) / 8;
    switch (count % 8) {
    case 0: do { *to = *from++;
    case 7:      *to = *from++;
    case 6:      *to = *from++;
    case 5:      *to = *from++;
    case 4:      *to = *from++;
    case 3:      *to = *from++;
    case 2:      *to = *from++;
    case 1:      *to = *from++;
        } while (--n > 0);
    }
}
```

- Familiarity
- Low-level
  - Course is about how design decisions affect performance
- Fast, useful to know
- A couple specific features we'll be using

Summary of Policies and Assessments

Weekly Assignments

- Due Thursday 10pm
- Code component; latex component
- Resubmit lab with mistakes fixed for up to one letter grade improved
- Usually released one week before
- Late penalty 1 letter grade per day
  - Let me know if there is some reason why you cannot make it!
  - I have no problems giving late days if the need arises
  - (Seriously do this 🙂)
  - But please tell me before!
- Last part of course (after Nov. 13) will instead have a more open-ended project

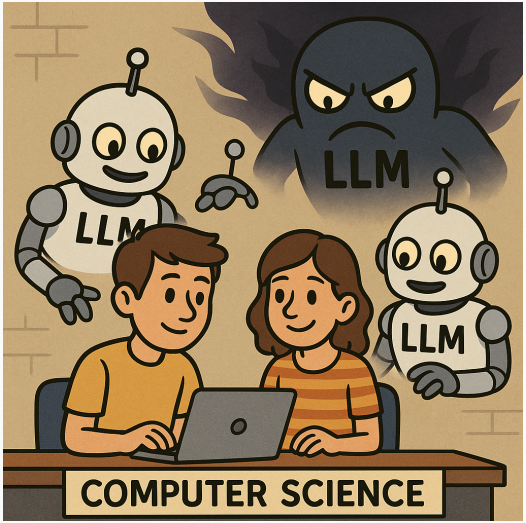
Assignment Logistics

- Code handed in via gitlab; pdfs via gradescope
- I'll give overleaf links for the latex
- For testing code, I will use the machines in the back of TCL 312
- It's a good idea to write and run your code on those machines (in-person or over SSH)

Exams

- Midterm Oct 24
- Final during finals period
- Goal: test your knowledge of experimental analysis, theory; some code
- I will not ask you to write code on the exams—instead, ask to explain code, fix it, analyze it, etc.
- Less focus on proofs (especially formal correctness) than in algorithms
- I'll give a practice midterm/final to help get an idea of what it will look like

LLMs



- No LLM usage in this class allowed for *anything*
    - Even debugging—you should use google instead
- Issues with incentives, fairness
- I will try to give you snippets of any small pieces of code

Let’s look over the syllabus quickly

Course Website

“Assignment” 0

- Due next Thursday
- I’ll post soon; release repos on Monday (can’t do it until then)
- Need to get your gitlab repo; answer a couple introductory questions, and commit (that’s it!)

Coding in C

Plan for this section

- Quick review of some key concepts
- Emphasize some particularly important areas for this course
- Use the first week as an opportunity to catch up!
- Instructor, other students, even stackexchange (etc.) are all good resources for questions you may have[1]

[1] Just remember to cite and be sure that you can explain anything you submit.

About C

- Lifetime of information to learn
- I am not an expert (though I’ve used it a lot)
- Many interesting features, many interesting behind-the-scenes effects
- Close connection between your code and the computer’s actions

Arrays

- Really just pointers
- No bounds checking
- Can use `sizeof` for fixed-size array (compiler replaces with size at compile time). Also works with variables

Structs

- What C has instead of classes
  - No member functions
  - Still uses `.` operator to access member variables
- Sequence of variables stored contiguously in memory
- Semicolon after declaration
- Need to use `struct` or `typedef` to refer to structs.

Two Examples

- `struct.c`
  - `typedef` to make things easier
- `pointers.c`
  - Local variables different local vs remote
  - Access out of bounds
  - Values change(?) with different optimizations
  - `valgrind` to catch these issues

Memory Allocation

- `malloc` and `free`
  - Also use `calloc` and `realloc`
  - Need `stdlib.h`
- If you call C++ code, be careful with mixing `new` and `malloc`
- Use useful library functions like `memset` and `memcpy`
- Example: `memory1.c`

Sorting in C

- `qsort()` from `stdlib.h`
- Takes as arguments array pointer, size of array, size of each element, and a comparison function. Let's look at `sort.c`
- What's a downside to this in terms of efficiency?
- Many ways to get better sorts in C:
  - Nicely-written "homemade" sort
  - `C++ boost` library
  - Third-party code

Running Code

Accessing Lab Computers

- Can access using `ssh`
- Use a text-based editor (like vim or emacs) on the lab machine through the terminal
- Can also use VSCode directly: run VSCode on your computer, modifying and running a remote file

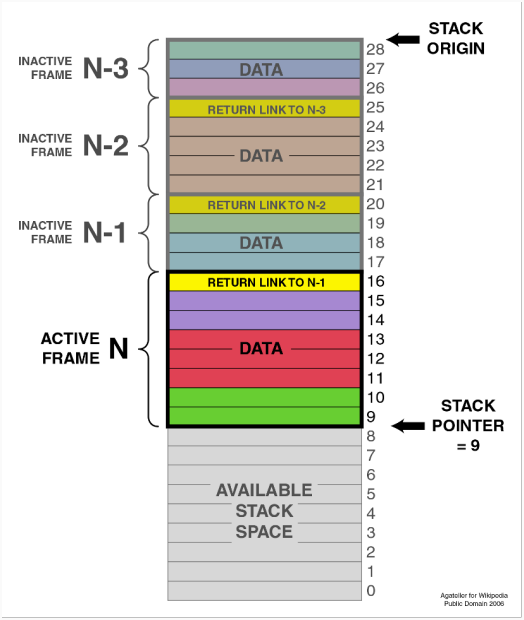
Notes on C and compilation

- We use `gcc` in this course
- Macs tell you they have `gcc` but it is not; it is actually `clang`
  - Can try to install `gcc` using `brew install gcc` (I just use lab computers...)
- Unlikely to make too much of a difference, but one reason to use lab computers if you're running into issues

Architecture

- x86 architecture (not AMD, not M1)
- Intel i7; run `lscpu` for details
- This *is* likely to have an effect on fine-grained performance in some cases
- Your home computers are fine for correctness and coarse optimization; use lab computers for fine-grained optimization
- If I ask you to do a performance comparison, or optimize to a certain wall clock running time, you should do it on lab computers.

Where are things stored?



- In CPU register (never touching memory)
  - Temporary variables like loop indices
  - Compiler decides this
- Call stack
  - Small amount of dedicated memory to keep track of current function and *local* variables
  - Pop back to last function when done
  - **temporary**

Other place to store things

- The heap!
- Very large amount of memory (basically all of RAM)
- Create space on heap using `malloc`
- Need `stdlib.h` to use `malloc`

How to decide stack vs heap?

- Java rules work out well:
  - “objects” and arrays on the heap
  - Anything that needs to be around after the function is over should be on the heap
  - Otherwise declare primitive types and let the compiler work it out
  - Keep scope in mind!

Makefile

- Each time we change a file, need to recompile that file
- Need to build output file (but don’t need to recompile other unchanged files)
- Makefile does this automatically

In this class

- I'll give you a makefile
- You don't need to change it unless you use multiple files or want to set compiler options
  - Probably don't need to use multiple files in this class
  - (Some exceptions for things like wrapper functions.)

Let’s look quickly at the default Makefile

- `make`, `make clean`, `make debug`

Compiler flags

- `-g` for debug, `-c` for compile without build (creates `.o` file)
- Different optimization flags:
  - `-O2` is the default level
  - `-O3`, `-Ofast` is more aggressive; doesn't promise correctness in some corner cases
  - `-O0` doesn't optimize; `-Og` is no optimization for debugging
  - Other flags to specifically take advantage of certain compiler features (we'll come back to this)
- `-S` (along with `-fverbose-asm` for helpful info) to get assembly
- Also: "Compiler Explorer" online

Variable types

- `int`, `long`, etc. not necessarily the same on different systems
  - On Windows `long` is probably 32 bits, on Mac and Unix it's probably 64 bits
  - `long long` is probably 64 bits
- Instead: include `stdint.h`, describe types explicitly
- Keep an eye out for unsigned vs signed.
- Quick example: `variabletypes.c`
- `printf` does expect primitive types

Variable types cont.

- `int` (etc.) is OK for things like small loops
- If you care *at all* about size you should use the type explicitly
- Up to you when and where you use unsigned
  - Controversial in terms of style

List of particularly useful integer variable types

- `int64_t, int32_t`: signed integers of given size
- `uint64_t, uint8_t`: unsigned integers of given size
- `INT64_MAX` (etc.): maximum value of an object of type `int64_t`

Board Discussion: What Makes Code Run Fast?