Applied Algorithms Lec 1: Welcome (and some C)

Sam McCauley

September 6, 2024

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

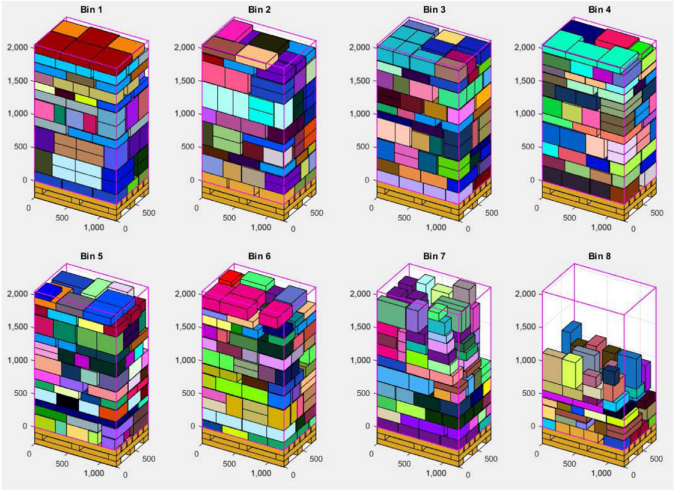
About the Class

- 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

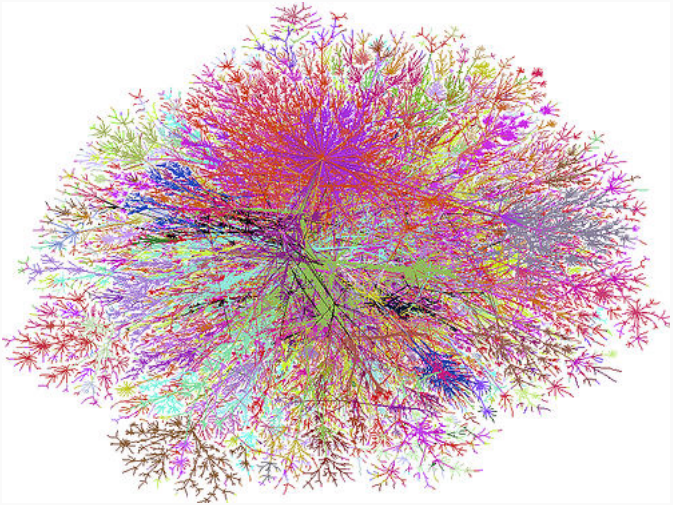
- Algorithms that you should always have handy because they are incredibly useful
- Bloom filters, linear programming, suffix trees
- What drives the course
- Algorithmic understanding of these ideas!

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
- Optional, friendly competition for those who want to optimize code (with some bonus points)

About Me

- Call me Sam
- Research is in algorithms
    - Some experimental algorithms
- Office is TCL 306
- Office Hours Wed Thu 3-5pm TCL 306 (?) Let me know if you can't make it!

About the Course

- No course textbook; some suggested readings
- Textbooks for background will be left in TCL 312.
- From last time it was taught: some grading changes; project added
    - Goal: focus more on learning the topics, less on grades. Also lower workload
- Questions *particularly* welcome!

Help, Questions, Comments, Etc.

- Slack; 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!)

Lab

- TCL 312
- Passcode (write it down)
- Office hours will generally be in TCL 312
- Feel free to stop by.
  - No one else has reserved it
  - But others use it—keep an eye out for occupancy
- No food or drink this semester!

Theory assessment

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

Coding assessment

- (Almost) all in C
- Weekly assignments
- Homework 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 Homeworks

- Due Thursday 10pm
- 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!

Assignments

- Used for assessment (as opposed to homeworks which are used for practice)
- 3 during the semester
- Look like homeworks, handed in like homeworks
- But all work must be *entirely* your own!
  - No instructor or TA help;
  - No help from other students; no online resources
  - Contact me with any questions or if issues come up

Final

- No final in the course

Final Project

- Idea: Pick a topic we went over, explore it in more depth
- Done in groups of up to 2
- I'll meet with you regularly to discuss directions to take the project, and to make sure that you have good content
- Start after third Assignment (Nov 14)
- Due: December 10th.

Homework Honor Code Policies: Problem Set Questions

- Normal CS department assignment rules
- You must do by yourself
- Instructor and TA can help
- Can discuss high-level strategies with other students (“hands-in-pockets” rule)
- Can ask other students about debugging and syntax issues

Homework Honor Code Policies: Code

- You can collaborate with other students and use online resources
- You may share code, use stackexchange, and use ChatGPT
- But you must cite what you use!!
- You have to understand anything you submit.
  - I may actually ask you about code you've written—possibly because what you've done is interesting (though it may also be to ensure you're keeping up)
- Details in syllabus; let me know if you have questions

“Leaderboard” extra credit

- On some homeworks we’ll have a fun competition to see who can write the fastest implementation
- Totally optional!
- First–third fastest will get 20, 15, 10 extra points
- +5 if you are faster than previous fastest
- Current 5 fastest times will be (anonymously) posted on website, along with last year’s and my lightly optimized implementation

CSCI 358 - Fall 2021

Applied Algorithms

Assignment8

Last Updated Dec 09 23:35

1	d46e	4.517875
2	8506	10.044391
3	8d4c	10.063052
4	005c	10.085786
5	590c	10.105472
6	Sam	10.288698

Grading

- Homeworks: 25%
- Assignments: 50%
- Final Project: 25%

Let’s look over the syllabus quickly

Course Website

“Assignment” 0

- Not worth points
- Due next Wednesday
- Just asks for your name and Github
- You can’t do Assignment 1 without 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
- Instructions to get this to work in handouts on the website (**strictly optional**)

Running Code

Accessing Lab Computers

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

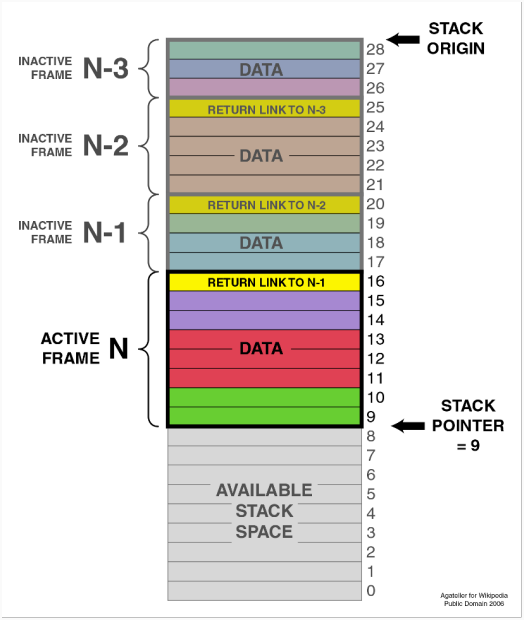
Notes on C and compilation

- 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, you should generally do it on lab computers. In any case you should write what you do it on.

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`