CS 375 – Analysis of Algorithms

Professor Eric Aaron

Lecture – M W 1:00pm

Lecture Meeting Location: Davis 117

Instructor Info

- **Professor Eric Aaron**

  Website: *https://cs.colby.edu/eaaron*

  My website is the most important info on this slide—the rest of it can be found there

  Office: Davis 113

  Office Hours: M 4:00-5:00pm, T 2:30-4:00pm, W 2:30-4pm, Th 11:00am-noon, and by email appointment

  Phone/Voicemail: (207) 859-5857

  E-mail: *eaaron@colby.edu*

  The above email address is the best way to contact me

  Course Website: ***https://cs.colby.edu/courses/F22/cs375***

A tiny bit about the course: Your textbook

- Required course textbook:
  - *Introduction to Algorithms, 3rd edition* by Cormen, Leiserson, Rivest, and Stein
  - **Online version is available FREE through the Colby library!**

See link from CS375 website "Additional Notes" page!

- "Recommended" textbook:
  - *Introduction to The Design and Analysis of Algorithms, 3rd edition* by Levitin
  - ***Not required***!
  - Good secondary source with different coverage; some images and exercises in the class may come from that book

A tiny bit more about the course: Components of an *algorithms* course

- Important elements for any course on algorithms:
  - Classic algorithms (which you might use or adapt for your work)
  - Algorithm design techniques and paradigms
    - Creating and working with algorithm specifications
  - Analyzing and explaining an algorithm's correctness
  - Analyzing and explaining an algorithm's complexity
- Communication is extremely important to our field
  - E.g., explanations of correctness and complexity

A tiny bit more about the course: Components of an *algorithms* course

- Important elements for any course on algorithms:
  - Classic algorithms (which you might use or adapt for your work)
  - Algorithm design techniques and paradigms
    - Creating and working with algorithm specifications
  - Analyzing and explaining an algorithm's correctness
  - Analyzing and explaining an algorithm's complexity
- Communication is extremely important to our field
  - E.g., explanations of correctness and complexity
- This course is not focused on programming, it's focused on *algorithms*
  - (More about that soon!)
  - Programming will not be required on any assignment (except possibly the last project assignment)

A tiny bit more about the course: Components of graded work in CS375

- Problem sets
  - Due at least one week after being assigned
- Smaller assignments
  - Typically due beginning of next class meeting
  - May be discussed in next class meeting
- Project assignments
  - *In general, no programming required…*
  - but please feel free to code stuff up, if you'd like!
  - Work in teams
  - Conceptually connected problems / applications, with team write-up
- Class participation
  - Anything that constructively contributes to class discussion: speaking in class, making points in office hours, making points by email, etc.

This course is about *algorithms*, not *programs*, although the two are certainly related! More about this soon, really!

Comparing CS375 / CS376

Courses about complexity and correctness of algorithms:

- **CS375 --** Math 274 [Mathematical Reasoning] not required
  - No experience needed with deep mathematics (not even calculus) or logical reasoning / proof techniques
  - Focused more on *what / how*, not deeply about *why*
- **CS376 --** *Prerequisite*: Math 274
  - Connects material to deeper mathematics and logical reasoning
  - Focused *rigorously* on all of *what*, *how*, and *why*
  - Recommended if interested in graduate study after Colby, or software engineering jobs that are more about design / architecting and less about programming

- **Basically the same material in both 375 and 376**
- **Covered differently in 375 than in 376**

Comparing CS375 / CS376

Courses about complexity and correctness of algorithms:

- **CS375 --** Math 274 [Mathematical Reasoning] not required
  - No experience needed with deep mathematics (not even calculus) or logical reasoning / proof techniques
  - Focused more on *what / how*, not deeply about *why*
- **CS376 --** *Prerequisite*: Math 274
  - Connects material to deeper mathematics and logical reasoning
  - Focused *rigorously* on all of *what, how*, and *why*
  - Recommended if interested in graduate study after Colby, or software engineering jobs that are more about design / architecting and less about programming

One way to think of it…

375: more "… for the rooms with the *keyboards*"

376: more "… for the rooms with the *whiteboards*"

Both classes discuss two key questions about problem solving:

- How do we get a good solution?
- How do we know we got it right?

…but CS376 is more mathematically rigorous, and as part of that, goes deeper than CS375 does into the "How do we know we got it right?" question

Comparing CS375 / CS376 / CS378

Courses about complexity and correctness of algorithms:

- **CS375 --** Math 274 [Mathematical Reasoning] not required
  - No experience needed with deep mathematics (not even calculus) or logical reasoning / proof techniques
  - Focused more on *what / how*, not deeply about *why*
- **CS376 --** *Prerequisite*: Math 274
  - Connects material to deeper mathematics and logical reasoning
  - Focused *rigorously* on all of *what, how*, and *why*
  - Recommended if interested in graduate study after Colby, or software engineering jobs that are more about design / architecting and less about programming

- Basically the same material in both 375 and 376
- Covered differently in 375 than in 376

Courses about models of computation: Very different material from that in 375 / 376

- **CS378** (cross-listed as Math 378) -- *Prerequisite*: Math 274
  - Covers mathematical models underlying computational tools, programming languages, and complexity
  - Abstract, broadly applicable and influential approach to system design and analysis
  - Emphasizes proofs and analytical methods
  - More conceptual, less application-oriented than CS375, CS376

("You think *you've* got problems....")

Problems and Solutions

- On a fundamental level, CS375 (and all of Computer Science!) is all about problems—and problem solving

  - What makes a solution to a problem good (or bad)?
  - *What makes one solution to a problem better than another?*

("You think *you've* got problems….")

Problems and Solutions

- On a fundamental level, CS375 (and all of Computer Science!) is all about problems—and problem solving
- Throughout the course (and throughout all of Computer Science!), definitions are very important, so let's start here!
  - What is a *problem*, in a useful, computational sense?
  - What does it mean to solve a problem?

    (Note: This isn't the question of "*How* does one solve a problem?" This is about what it means *to have a solution*, no matter how that solution is found.)
  - What makes a solution to a problem good (or bad)?
  - *What makes one solution to a problem better than another?*

Problems, Algorithms, and Code, pt. 1: Problems

- What is a *problem*, in a useful, computational sense?
  - Informal definition: In a relevant sense, a problem is an *input/output relationship*

From our textbook: "The statement of a problem specifies, in general terms, the desired input / output relationship"

Problems, Algorithms, and Code, pt. 1: Problems

- What is a *problem*, in a useful, computational sense?
  - Informal definition: In a relevant sense, a problem is an *input/output relationship*

From our textbook: "The statement of a problem specifies, in general terms, the desired input / output relationship"

  - For example, you may have heard of *the sorting problem*

Input: A sequence L of n numbers (a_0, …, a_{n-1})

Output: A sequence L' of n numbers (b_0, …, b_{n-1}) that re-orders the input sequence (perhaps leaving them unchanged) such that $b_0 \leq b_1 \leq \ldots \leq b_{n-1}$

It's often a good idea to have a specification like the one in yellow above as part of the comments for functions you write in your programs—it makes understanding the code and ensuring its correctness much, much easier!

Problems, Algorithms, and Code, pt. 1: Problems

- What is a *problem*, in a useful, computational sense?
  - Informal definition: In a relevant sense, a problem is an *input/output relationship*

From our textbook: "The statement of a problem specifies, in general terms, the desired input / output relationship"

- What does it mean to solve a problem?

Now that we have a definition of *problem*, we can come up with a cleaner answer to this!

- What makes a solution to a problem good (or bad)?
  - *What makes one solution to a problem better than another?*

Problems, Algorithms, and Code, pt. 2: Algorithms

- What is a *problem*, in a useful, computational sense?
  - Informal definition: In a relevant sense, a problem is an *input/output relationship*
- What's an algorithm? Informal definition, from CLRS:

An *algorithm* is a well-defined computational procedure that takes input and produces output.

- What does it mean to solve a problem?
  - Informal definition: In this computational sense, a solution to a problem is an algorithm…
  - We say an algorithm *correctly solves a problem* when *it transforms every input to its related, correct output*

Problems, Algorithms, and Code, pt. 2: Algorithms

- What is a *problem*, in a useful, computational sense?
  - Informal definition: In a relevant sense, a problem is an *input/output relationship*
- An algorithm *correctly solves a problem* when *it transforms every input to its related, correct output*

This top part is just review, all repeated from before

- *What makes a solution to a problem good (or bad)?*
- For example, consider the sorting problem (below). It has many solutions! What makes one solution better than another?

Input: A sequence L of n numbers (a_0, …, a_{n-1})

Output: A sequence L' of n numbers (b_0, …, b_{n-1}) that re-orders the input sequence (perhaps leaving them unchanged) such that $b_0 \leq b_1 \leq \ldots \leq b_{n-1}$

Problems, Algorithms, and Code, pt. 2: Algorithms

- *What makes a solution to a problem good (or bad)?*
- (For instance, consider the sorting problem.) What makes one solution to a problem better than another?
  - *Correctness* (i.e., does it work?)
  - *Time complexity* (i.e., how fast is it?)
  - *Space complexity* (i.e., how much memory does it use?)
  - (Other?)

Input: A sequence L of n numbers (a_0, …, a_{n-1})

Output: A sequence L' of n numbers (b_0, …, b_{n-1}) that re-orders the input sequence (perhaps leaving them unchanged) such that $b_0 \leq b_1 \leq \ldots \leq b_{n-1}$

Problems, Algorithms, and Code, pt.3: Code

- Just as a problem can have many algorithms that solve it (e.g., sorting, searching problems)…
  (There are multiple sort and search algos. Which ones do you know?)
- … An algorithm can have many possible implementations in code
  - For example, every programming language would lead to a different implementation
- For CS375, we'll consider problems and algos more than code
  - You already know enough to implement algos in at least one language!

Determining the best solutions for a problem is often better at the algorithm level than the code level—all implementations of the same algorithm will have the same time / space complexity!

Business

- **Optional, but appreciated**: Please email me from the account at which you'd want me to contact you
  - Include a sentence on what you'd like to get out of the course
  - … plus anything else you might like to tell me!
  - Also, in your email, let me know if you were able to access the course website and lecture notes without any difficulties
    - Recall: website is at ***https://cs.colby.edu/courses/F22/cs375***

- **Lecture notes are a primary means of communication with you in CS375. You may well end up looking back at them regularly.**
- **Please be sure you can access / read them without difficulties!**

(I'll post lecture notes by the end of the day—i.e., midnight—today)

Business, pt. 2

- Smaller Assignment 0 out today
  - Due in 1 week, Sept. 14
    - As always in this class, unless otherwise specified, deadline is at 1pm on the due date, before the beginning of class
    - If a time isn't given for a deadline, assume it is 1pm, all semester long
  - Please be sure to do the reading on it!
    - If you have questions about the math pre-requisites in the Appendices, please let me know—otherwise, I will assume you are comfortable with them
  - Note: Unusual deadline because it's the first day of class
    - (Most Smaller Assignments will be due by the next class meeting, not in 1 week)
- Find it at the *Smaller Assignments* page at the course website