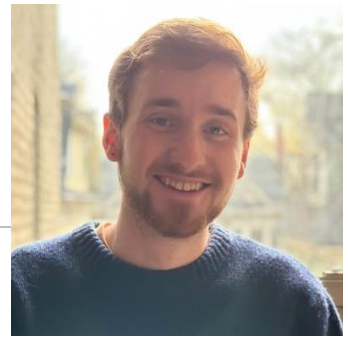
DS 320: Algorithms for Data Science

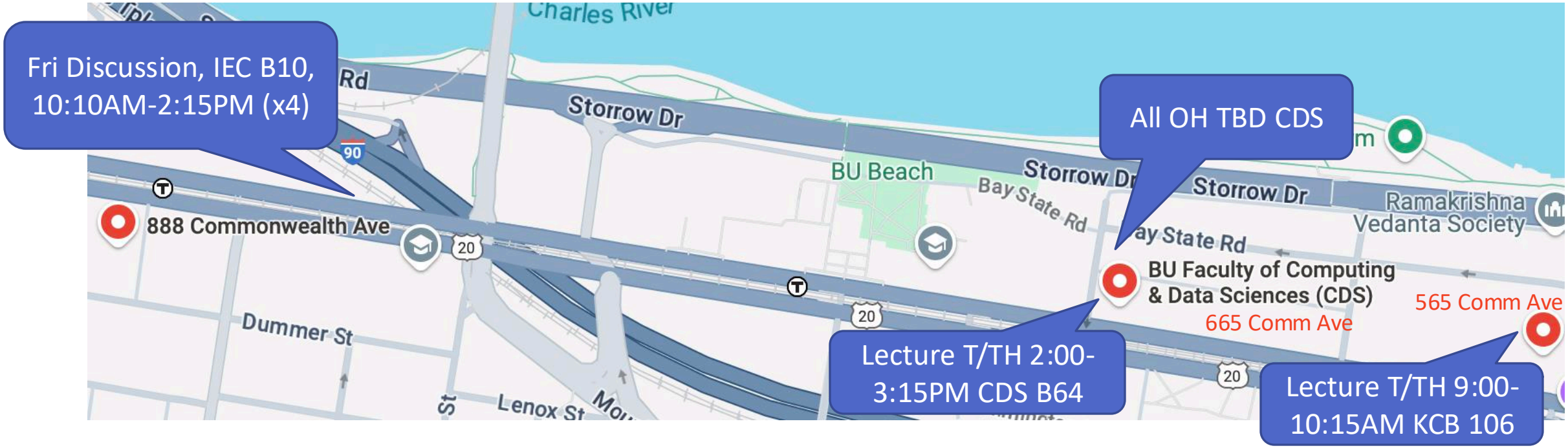
PROFESSORS KIRA GOLDNER AND JEFFREY CONSIDINE

Teaching Staff



Instructors: Prof. Kira Goldner
& Prof. Jeffrey Considine
OH: See online

TAs: Anup & Kevin
Email: Nope! Use Piazza!

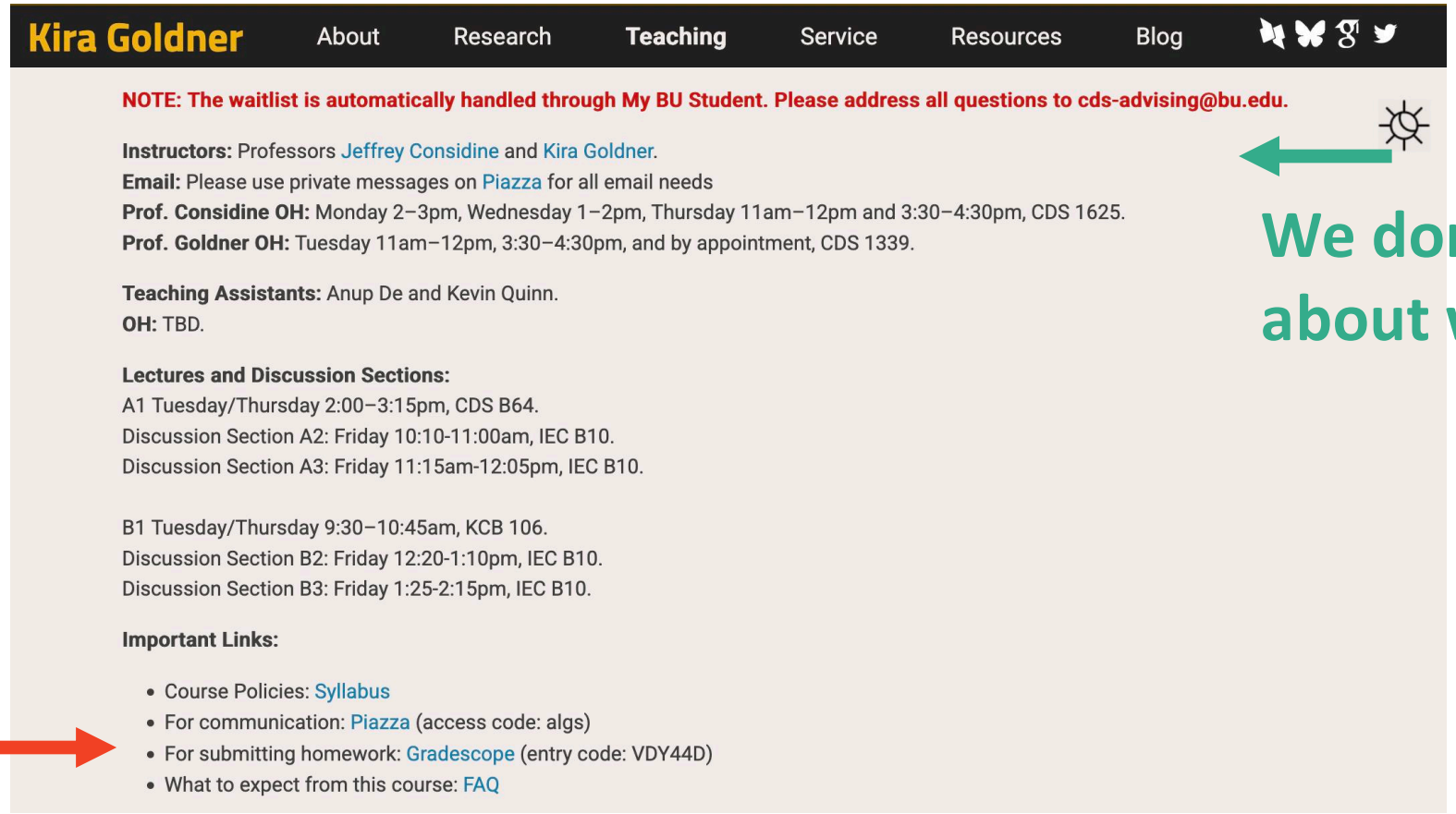


Class Resources

Website

Course website: https://www.kiragoldner.com/teaching/DS320/

- Lecture notes, links to everything



We don't know anything about waitlists.

Today! →

Class Resources

Course website: https://www.kiragoldner.com/teaching/DS320/
- Lecture notes, links to everything

Website

Piazza (code “algs”):
- Class announcements, Q+A, **assignments + solutions**, use **instead of email (won’t respond to email)**
- I am a human who does not live inside the computer!

Piazza

Gradescope (code “VDY44D”):
- Turn in assignments and view grades

Gradescope

Sign up for these if you have not already!

This is a theoretical problem-solving class

No programming assignments! Evaluation based on problem sets and exams.

Prerequisites:

- Intro programming (DS 110, CS 111, …)
- **A first proofs class** that's Discrete-Math-esque (DS 121, CS 131, MA 293, …)

Not required but might make you more comfortable:

- Data structures and algorithms (DS 210, CS 112, …)
- More proof classes

Homework 0: Things you should already know! **Due Thursday 11:59pm.**

- Answer 100% of the questions, get 100% toward participation.
- Helps us and you know where you need to fill in preparation.

Evaluation

Homework (45%)
- ~Weekly problem sets

Midterm Exams (30%)
- Two midterm exams, worth 15% each. (Approx Feb 25 and April 1)

Final Exam (20%)
- During our scheduled time, closed-book.

Class participation (5%)
- In class (participation cards every two weeks) and via Piazza (asking and answering questions—Piazza stats).

Homework Policies

- Expect to spend at least 10 hours per week on homework.
- **Late policy:** You have 4 late days, max 2 per assignment (integer numbers used only). No exceptions. You don't get extra later if you're sick!
- Lowest homework grade will be down-weighted to 30% at the end of the semester.
- Type up homework with **LaTeX**.
- Turn in via **gradescope**. Due at 11:59pm on Wednesdays (typically).

- **Regrades:** Requests within 7 days, only via gradescope, with explanation/argument. Only for **incorrect** grading (not insufficient credit). If you request a regrade, the whole assignment/exam may be regraded, and your score may go up or down. **Do not use these to ask for feedback.**

Type up homework with LaTeX

- Slight learning curve! May want to use Overleaf (overleaf.com). See template on the course website. **DO NOT** use ChatGPT or a friend's template.

Asymptotic Notation

Definition 1 (Upper bound $O(\cdot)$)**.** For a pair of functions $f, g : \mathbb{N} \to \mathbb{R}$, we write $f \in O(g(n))$ if there exist ($\exists$) constants c_1, c_2 such that for all (s.t. $\forall$) $n \geq c_1$,

$$f(n) \leq c_2 g(n).$$

We'll often write $f(n) = O(g(n))$ because we are sloppy.

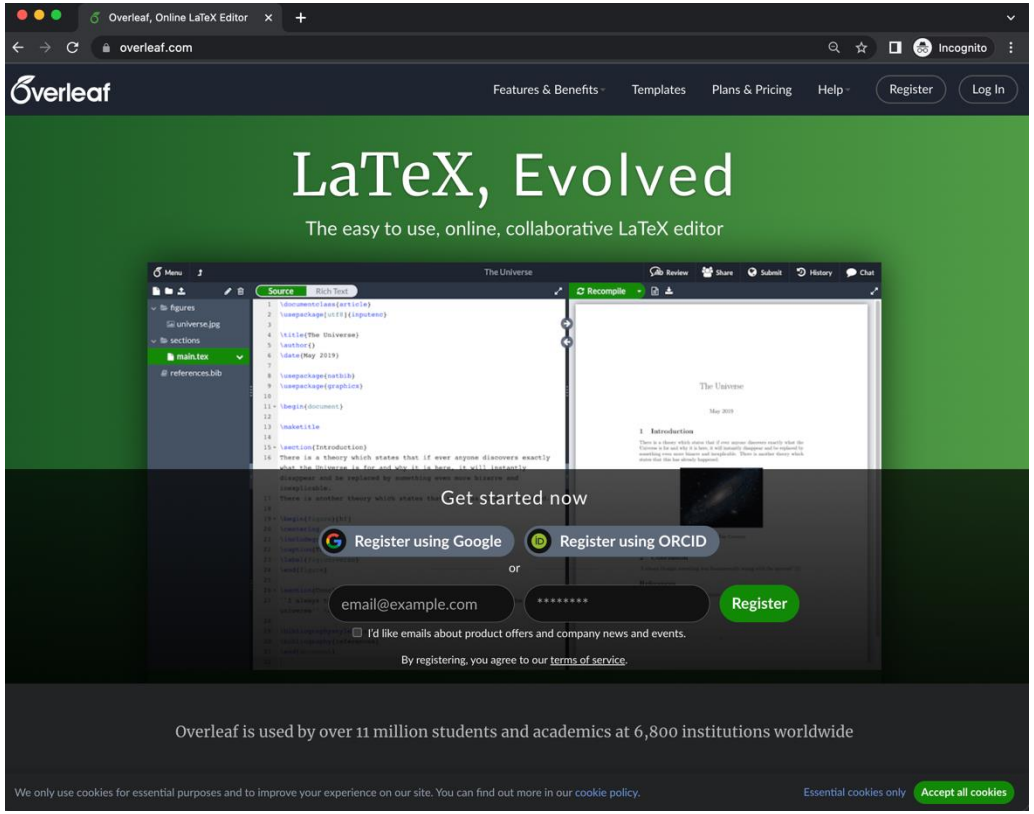
Translation: For large n (at least some c_1), the function $g(n)$ dominates $f(n)$ up to a constant factor.

Definition 2 (Lower bound $\Omega(\cdot)$)**.** For a pair of functions $f, g : \mathbb{N} \to \mathbb{R}$, we write $f \in \Omega(g(n))$ if there exist constants c_1, c_2 such that for all $n \geq c_1$,

$$f(n) \geq c_2 g(n).$$

Definition 3 (Tight bound $\Theta(\cdot)$)**.** For a pair of functions $f, g : \mathbb{N} \to \mathbb{R}$, we write $f \in \Theta(g(n))$ if $f \in O(g(n))$ and $f \in \Omega(g(n))$.

Exercise: True or False?



Collaboration Policy

Collaboration is encouraged!!!

- You may work with up to **three** classmates on an assignment. **List your collaborators' names on your assignment.** (E.g., Collaborators: None.)
- Good rule: **Nobody should leave the room with anything written down.** If you really understand, you should be able to reconstruct it on your own.
- You may **not** use the internet or ChatGPT on homework problems. You may use course materials and the recommended readings from textbooks.

We believe **strongly** in learning over evaluation, learning via collaboration, and academic integrity. **Please adhere to BU's academic conduct policy.**

Midterms

Two midterm exams, worth 15% each.
Tentative dates: Feb 25 and April 1

Essentially: **problems will be the same format as homework, but cumulative, and designed to take the period of one class.**

Think of them as short solo problem sets to prove you can do them by yourself.

Class Etiquette

We strive toward an accessible and equitable classroom for all students.

- Raise your hand.
- Be conscious of how often you participate (in class and in collaboration).
  - Don’t talk over others, leave room for other voices if you speak up a lot, and speak up more if you do not.
- Use your participation card to estimate.

But also

- Ask questions!!!

Best advice I ever got was to just ask and not wait to fill in gaps myself later.

Class Time

Date	Topic	Resources
Sep 6	Overview and Policies, Intro to AGT	Slides, Worksheet, Notes, R1.1-2
Sep 8	Incentive Compatibility	Worksheet, Notes, R1.3
Sep 13	The Revelation Principle	Worksheet, Notes, R1.4, H2

DS 320 Algorithms for Data Science
Spring 2023

Lecture #1 Worksheet
Prof. Kira Goldner

Covered in introduction slides:

- Course policies (also in syllabus).
- What to expect in this class (also in FAQ).
- Sample of content we'll cover.

Runtime Review

In runtime analysis we do an informal accounting. We count basic operations (algebra, array assignment, etc) as constant time.[1]

Analyze the runtime of the following algorithm:

Algorithm 1 FindMinIndex($B[t+1, n]$).

```
Let MinIndex = t + 1.
for i = t + 1 to n do
    if B[i] < B[MinIndex] then
        MinIndex = i.
    end if
end for
return MinIndex.
```

Which operations are constant-time?

Are there any loops? How many times do they run?

How does everything come together?

Which factors dominate asymptotically?

[1] This isn't quite right—for example, multiplication of large numbers should scale with the bit complexity—but is a good approximation for us.

- Worksheet listed in advance on website
- **Bring worksheet to class** (on iPad, printed, etc)
- Lecture + exercises
- Notes posted after class

DS 320 Algorithms for Data Science
Spring 2023

Lecture #1
Prof. Kira Goldner

Covered in introduction slides:

- Course policies (also in syllabus).
- What to expect in this class (also in FAQ).
- Sample of content we'll cover.

Runtime Review

When we analyze runtime, we'll do an informal accounting. We'll count basic operations (algebra, array assignment, etc) as constant time.[1]

We will analyze the runtime of the following algorithm:

Algorithm 1 FindMinIndex($B[t+1, n]$).

```
Let MinIndex = t + 1.
for i = t + 1 to n do
    if B[i] < B[MinIndex] then
        MinIndex = i.
    end if
end for
return MinIndex.
```

Each of the following lines is a unit (constant-time) operation:

- **Let** MinIndex $= t + 1$.
- **if** $B[i] <$ B[MinIndex] **then**
- MinIndex $= i$.

The for-loop runs $n - t$ times (notice that both n and t are variables as they are in our input). Thus the runtime of this algorithm is $O(n - t)$.

Asymptotic Notation

Definition 1 (Upper bound $O(\cdot)$)**.** For a pair of functions $f, g : \mathbb{N} \to \mathbb{R}$, we write $f \in O(g(n))$ if there exist ($\exists$) constants c_1, c_2 such that for all (s.t. $\forall$) $n \geq c_1$,

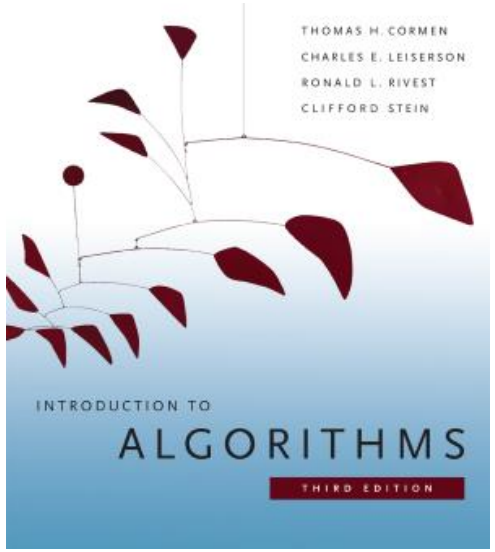
$$f(n) \leq c_2 g(n).$$

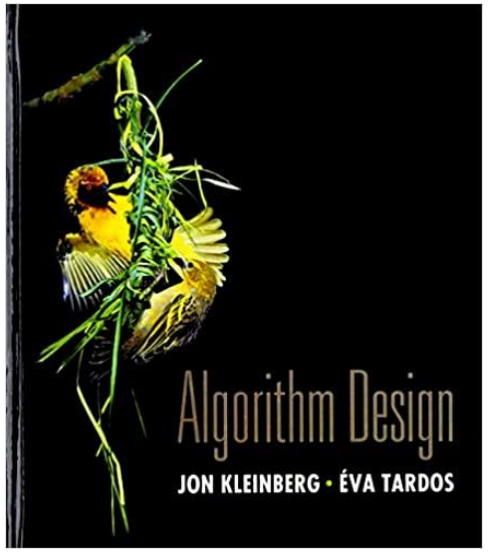
We'll often write $f(n) = O(g(n))$ because we are sloppy.

[1] This isn't quite right—for example, multiplication of large numbers should scale with the bit complexity—but is a good approximation for us. We will analyze runtime by counting these operations.

Book

There is no required textbook, and the lecture notes will be self contained. But many of the topics we are covering are well covered in standard algorithms textbooks; some lectures are adapted from Kleinberg and Tardos.





Course Learning Objectives

Learn the following skills:

- Get comfortable understanding and writing **formal definitions and statements**.
- Creative **problem solving** and **thinking algorithmically**.
- Write clear and convincing **arguments**.
- **Domain-specific skills:** Identify algorithmic problems within applications; determine when to apply which technique; analyze runtime.

IMO, **skills are more important** than course knowledge, so your **time is much better spent engaging with homework problems** than on reading additional material.

The Study of Efficient Algorithms

ALWAYS INCLUDE RUNTIME AND CORRECTNESS

Runtime Analysis

Analyze in the worst-case, for the biggest instances.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

An Arsenal of Algorithmic Techniques

Greedy Algorithms
- Make myopic choices. Very fast. Works when optimal solutions satisfy a certain "exchange" property.

Divide and Conquer
- Figure out how to quickly stitch together two (or more) optimal solutions to sub-problems. Recursively solve the sub-problems.

"Dynamic Programming" (actually Divide and Conquer++)
- The naïve recursion might have exponential size, but if we have only polynomially many *distinct* sub-problems, we can just cache the solutions to avoid wasted effort.

+ Continuous Optimization (“ML”)

Linear Programming

- Powerful framework for optimizing linear functions subject to linear constraints. Closely related to online optimization and zero sum games.

Multiplicative Weights

- For online optimization—obtains guarantees for adversarial sequences of loss functions.

Randomized Algorithms

- When and how randomization can improve upon deterministic guarantees.

Impossibilities & Approximation

Formal statements that you can do no better with a solution.

- E.g., the knapsack problem is NP-complete.
- If you could find a polynomial-time algorithm for it, then you could solve all these other algorithms in poly-time.

Approximation algorithms

- E.g. an algorithm that is fast and provably always get at least 1/2 as good as the optimal.

Where can you go after algorithms?

- Coding interviews
- Better problem solver in general, whether in code or puzzle hunts
  - which solution to apply when and ***why*** it's better
- Better formal thinking and writing
- More advanced toolkits (e.g., streaming, algorithmic game theory)