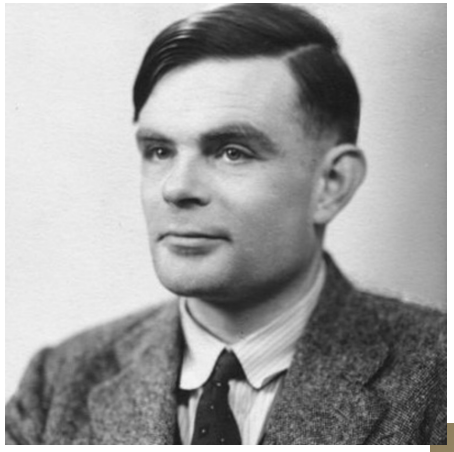
CS 4102: Algorithms

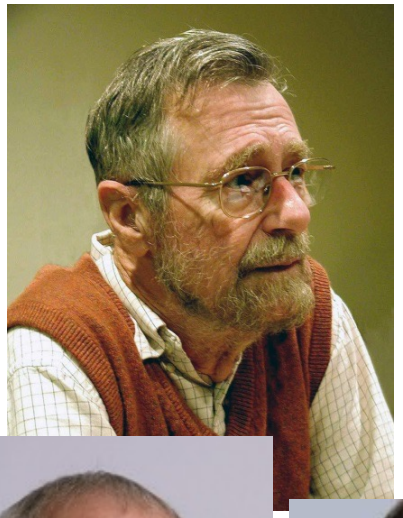
Lecture 1: Introduction and Logistics

Co-instructors: Robbie Hott and Tom Horton

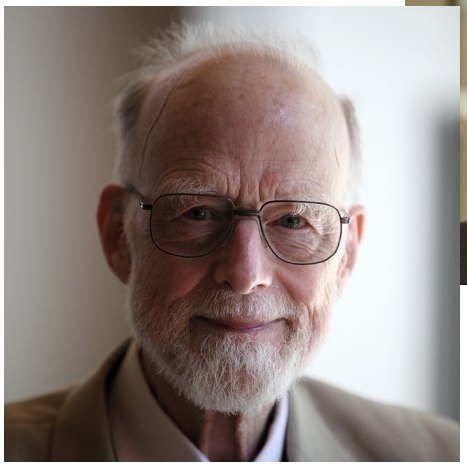
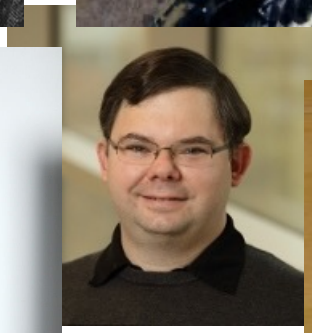
Spring 2022

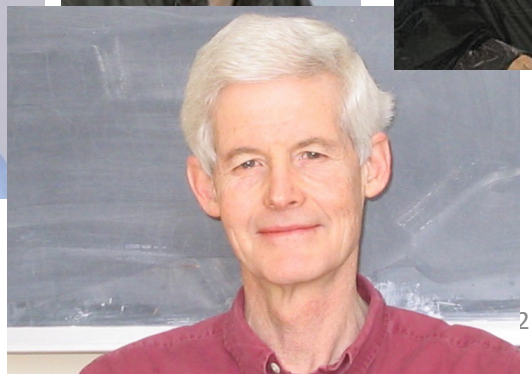
Who's Who in Algorithms?



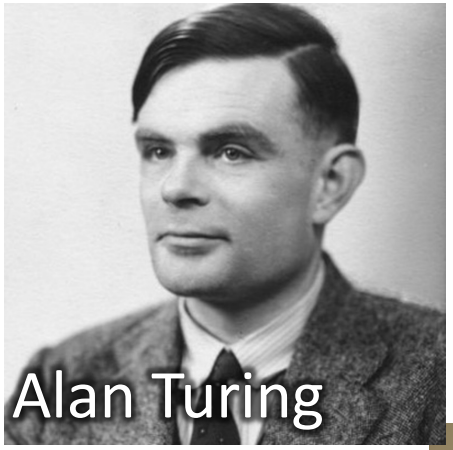




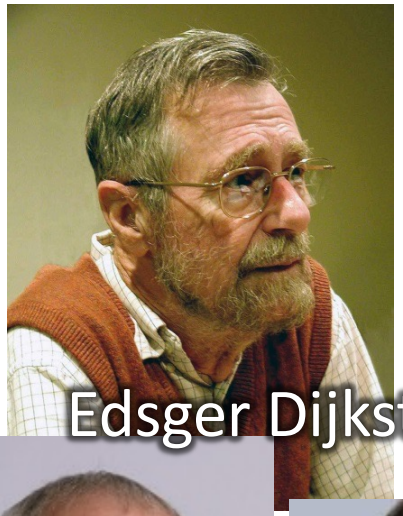






Who's Who in Algorithms?


Alan Turing

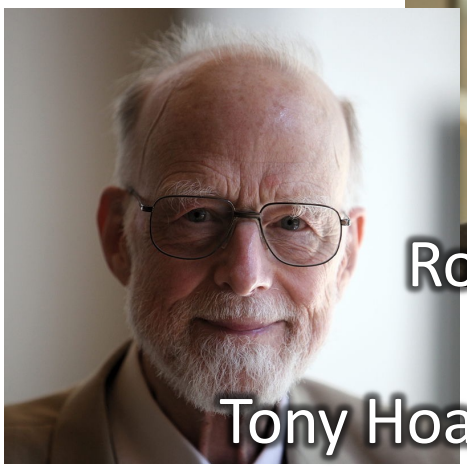

Ada Lovelace

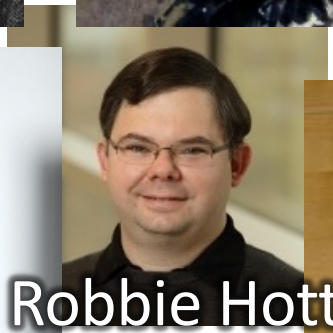

Edsger Dijkstra


Al-Khwarizmi


Tom Horton

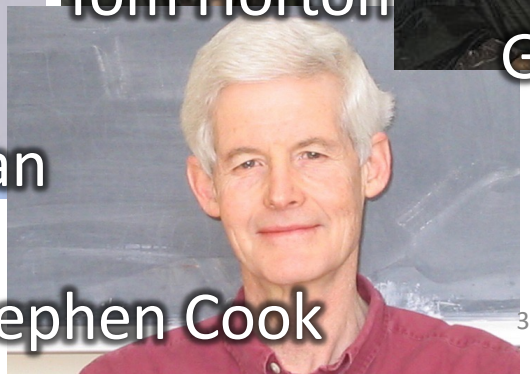

Gauss


Tony Hoare

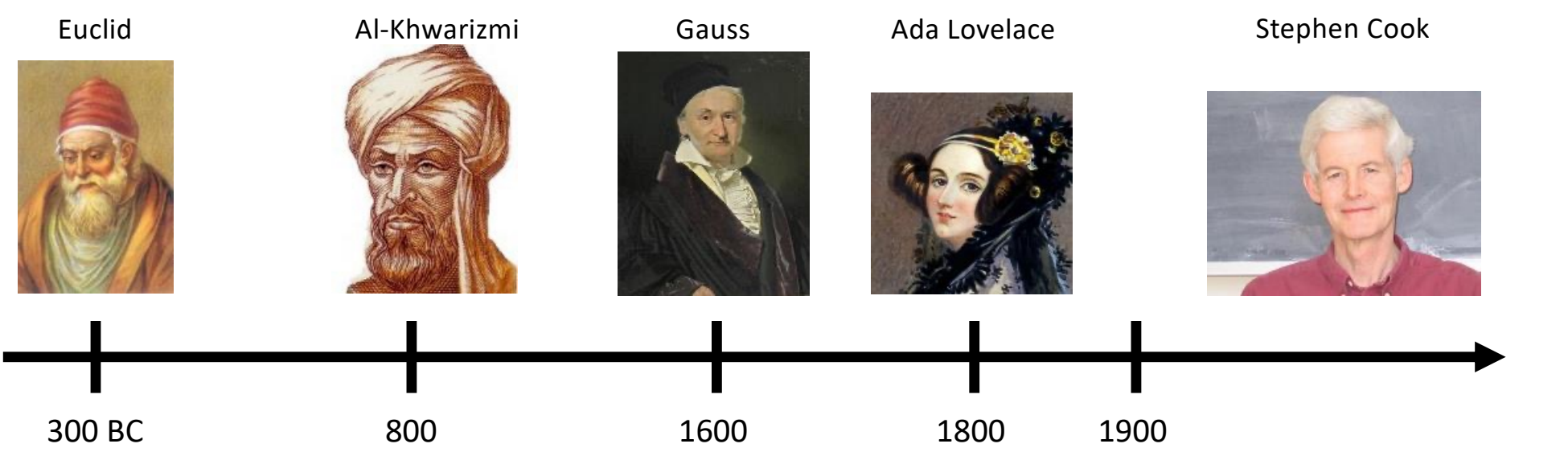

Robbie Hott


Donald Knuth

Robert Tarjan


Stephen Cook

A Historic Perspective



What Is an Algorithm?

- In mathematics and computer science, an algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation. Algorithms are unambiguous specifications for performing calculation, data processing, automated reasoning, and other tasks. [Wikipedia Jan 2020]
- An algorithm is a step by step procedure to solve logical and mathematical problems. [Simple English Wikipedia Aug 2019]

Motivating example

Takeaway: Being unambiguous is not always easy!

Goals

Create an awesome learning experience

Instill enthusiasm for problem solving

Give broad perspective on computer science

Have fun!

Co-Instructors

Prof. Hott

Rice 210

jrhott@virginia.edu

Email or Piazza
(no DMs in Discord)

Prof. Horton

Rice 401

horton@virginia.edu

Email or Piazza
(no DMs in Discord)

Office Hours TBD! Wait for announcement!

See course website for TA office hours

Logistics

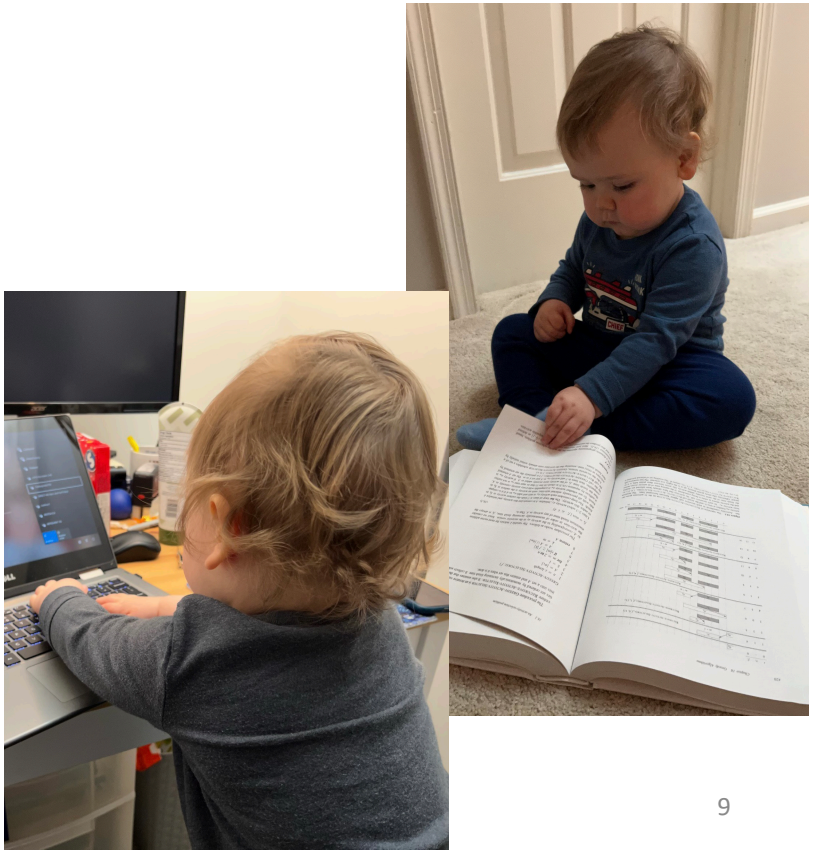
Course website: https://uva-cs.github.io/cs4102-s22
Including syllabus, resources, slides, HWs, etc.

Also **Collab** for some things (e.g. lecture recordings)

Piazza for student questions? Yes, through Collab
Online office hours through Discord
Anonymous feedback? No….

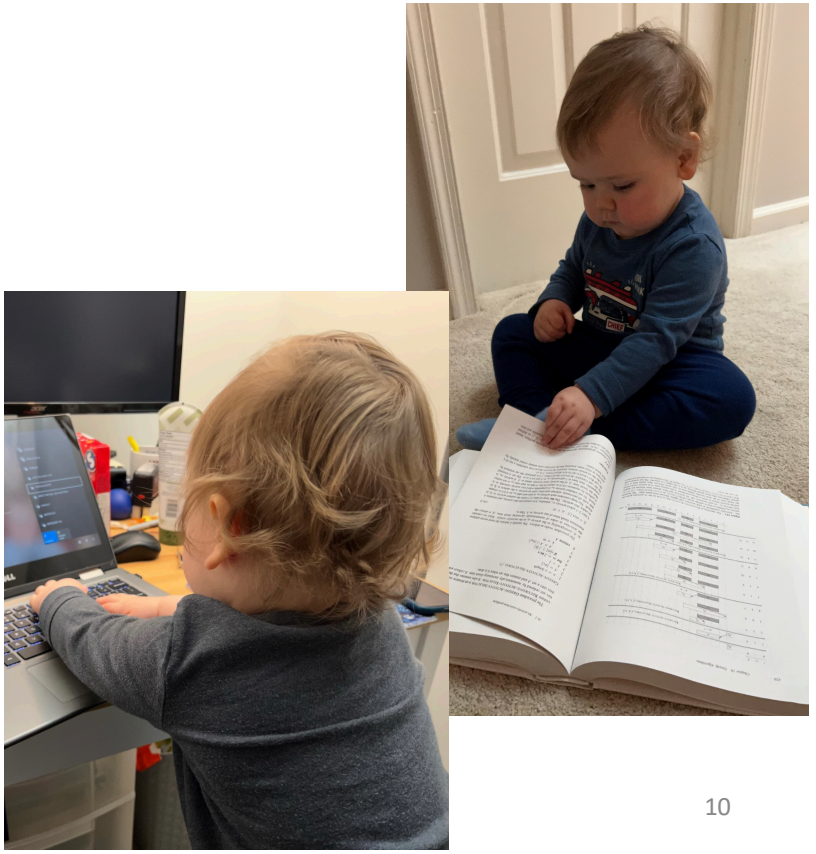
Covid-19 Course Modifications (Hott)

- Modified Meeting Format (Hott's section)
- Online Synchronous on Zoom
  - Recordings posted to Panopto
- Resume in-person: Feb 8
  - MEC 205



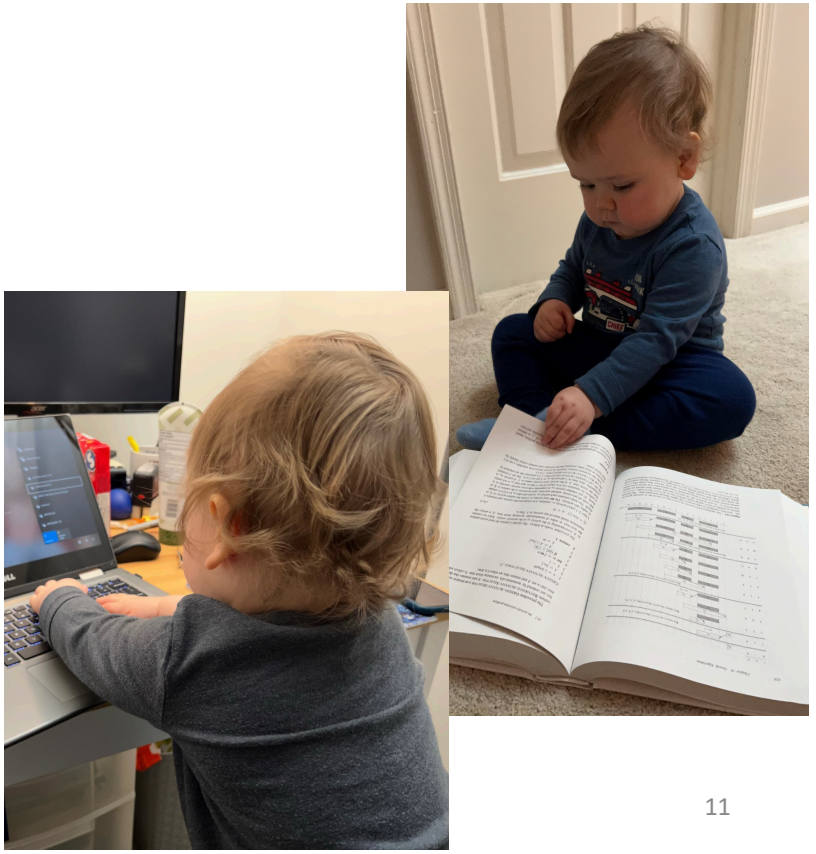
Covid-19 Course Modifications (Hott)

- Following University Guidance in this course when meeting in person
  - Masks are *REQUIRED* in lecture
  - Masks will always be encouraged
- All lectures recorded and posted
- Hott's office hours: on Zoom only (for now)



Covid-19 Course Modifications (Hott)

- If you feel ill, please stay home
  - I will **never** take attendance
  - I will work with you—if you stay home—to ensure it does not affect your grade
- If you are uncomfortable, you are welcome to stay home
  - I will work with you to ensure it does not affect your grade



Requirements

Discrete Math (CS 2102)

Data Structures (CS 2150) with C- or higher

Derivatives, series (Calc I)

Tenacity

Inquisitiveness

Creativity

Note: CS2150 pre-req taken seriously. Don’t meet it? Need approval or you will be dropped (after add deadline). ☹

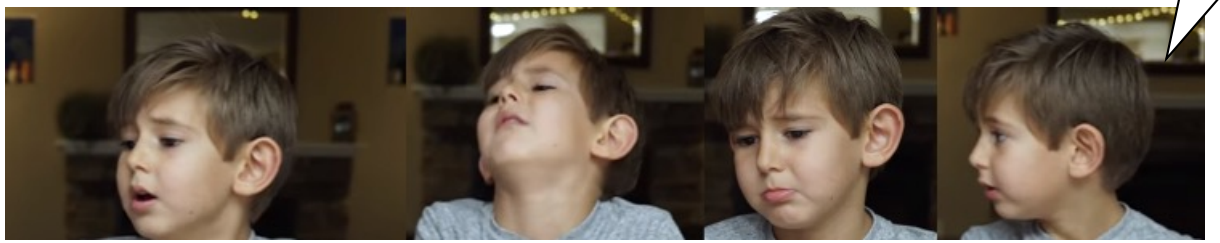
Warning

This will be a very difficult class

- Hard material
- “Holy grail” of computer science
- Useful in practice
- Job interviews

Lots of opportunities to succeed!

Hopefully not you…



“Learning Sources”

From what sources will you learn?

What I say in Lectures
What you get from the slides
Explanations you read in CLRS
Activities you do in/out of class
Assignments

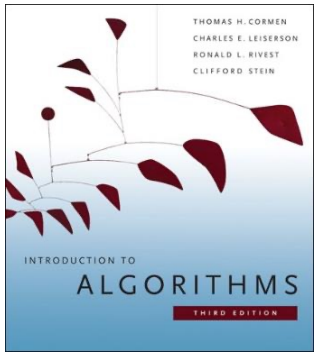
- All of these are important.
- Realistically, IMHO it’s impossible to get all the “book knowledge” from lectures and slides!

Textbook

You really need to read and study material other than the slides.

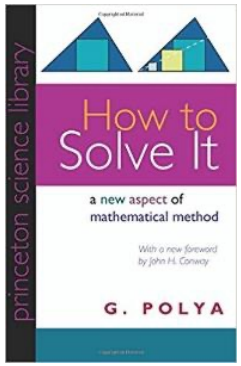
There are options, but a textbook is the easiest option.

I'll post readings from CLRS, urge you to read them or get that info from another source. **Note:** We will also have some resources posted on Collab site.



Cormen et al. (CLRS) *Introduction to Algorithms*. 3rd Edition.

Also recommended (but not a primary source)



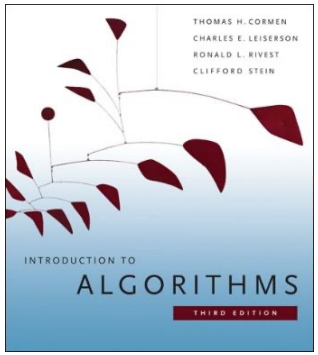
Polya. *How to Solve It.*

Textbook

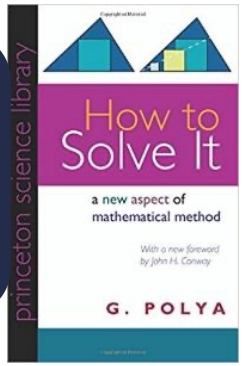
You <u>really need to</u> read and study material other than the slides.

There are options, but a textbook is the easiest option.

I'll post readings from CLRS, urge you to read them or get that info from another source. **Note:** We will also have some resources posted on Collab site.



Freely accessible online via the <u>UVA library</u>



Cormen et al. (CLRS) *Introduction to Algorithms*. <u>3rd</u> Edition.

Polya. *How to Solve It.*

https://search.lib.virginia.edu/catalog/u6757775

Units and Assignments

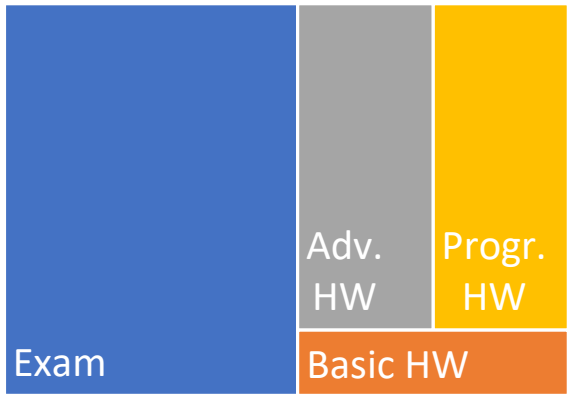
Four Units:

A. Sorting, Recurrence Relations, Divide and Conquer
B. Graphs: Search, Minimum Spanning Trees, Shortest Paths
C. Dynamic Programming and Greedy Algorithms
D. Network Flow, Reductions, NP-Completeness

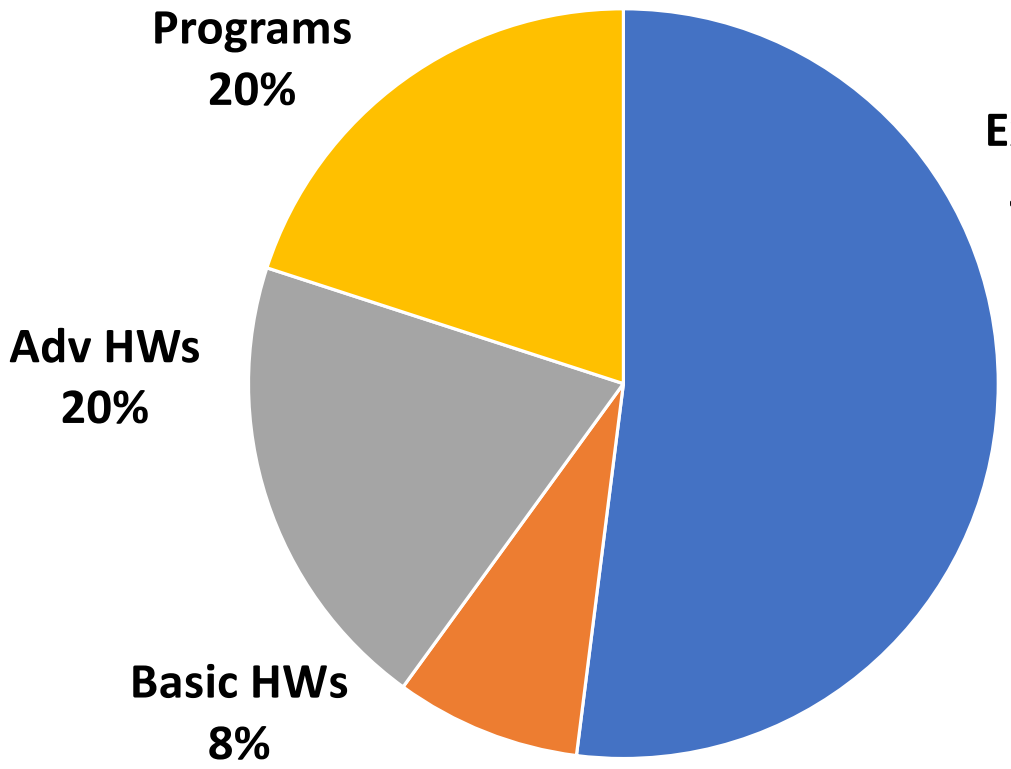
Each unit is 25% of your grade, broken down as follows:

- Exam (13%)
- Basic HW (2%) and Advanced Homework (5%)
- Programming Assignment (5%)

Note: unlike last two semesters, each assignment gets a numeric score, and they're combined to get an overall course score from 0-100.



Grade Components



Overview of these in the next slides. But see syllabus for full details!

Exams

In-person, in your assigned section's classroom

Exams for Units A, B, and C can be re-taken! (Highest score counts)

See schedule on website for dates

- Exam 1 is Feb. 22. (Re-take is two weeks later, after spring break)
- Re-takes for Units B and C during Final Exam period
- Only one chance for Unit D exam, during Final Exam period

In order to re-take an exam, you must have made a reasonable attempt on it the first time!

Basic Homeworks

Goal: Reinforce basic concepts and prepare you for exam

Due: Shortly before the exam (no extensions)

Format: Relatively short questions, nothing harder/longer than what's on an exam

Grading: Three possible scores (100, 83, 0)

Feedback: Not much, but can submit up to two times before deadline.
Solutions will be posted right after the deadline.

Collaborate? Yes, in groups (more later)

Submissions must be formatted in LaTeX! (more later)

Advanced Homeworks

Goal: A small number of challenging problems, requiring more time and deeper thought

Due: After the unit's exam (details TBD)

Grading: Graded more carefully than Basics HWs.
Possible scores (100, 93, 83, 30, 0)

Feedback: Yes. Details on deadlines and possible resubmissions TBD.

Collaborate? Yes, in groups (more later)

Submissions must be formatted in LaTeX! (more later)

Programs

Goal: Explore one or more topics from a unit by applying and implementing it

Due: After the unit's exam (details TBD)

Grading: Primarily based on passing test-cases on GradeScope. Possible scores (100, 93, 83, 30, 0), same as Advanced HWs.

Feedback: From GradeScope. Details on deadlines and possible resubmissions TBD.

Collaborate? Yes, in groups (more later)

Submissions can be in Python, Java, or C++

Basic Homework 1

Basic Homework 1A is out!

- Learning LaTeX
- Counts as one of the Basic HWs
- Due right after the Add deadline (but don't wait that long!)

Academic Integrity

Collaboration Encouraged!

- Groups of up to 4 per assignment (you + 3)
- List your collaborators (by UVA computing ID)
- OK to discuss problem, approach to solution, even details about solution, but…

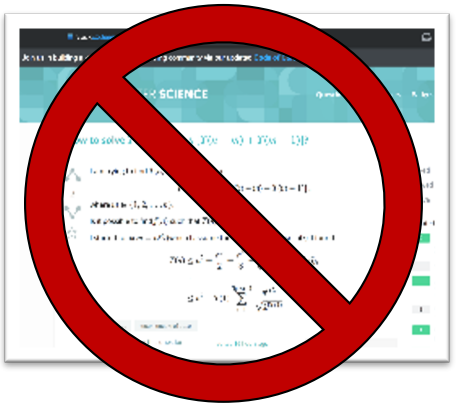
Write-ups/code must be written independently

- DO NOT share written notes / pictures / code
- DO NOT share documents (ex: Overleaf)
- DO NOT share debugging of code

Be able to explain any solution you submit!

DO NOT seek published solutions online

See syllabus about online code examples



Feedback

We professors are not course dictators, more like civil servants.

We're open to any suggestion to help you learn.

Let us know!

- In person
- Piazza
- Email: horton@virginia.edu or jrhott@virginia.edu
  - PLEASE: put CS4102 in subject line of all emails
- (No Discord DMs, please)

A first algorithm: making change

OK… But What's It Really All About?

Let's illustrate some ideas you'll see throughout the course

- Using one example

Concepts:

- Describing an algorithm
- Measuring algorithm efficiency
- Families or types of problems
- Algorithm design strategies
  - Alternative strategies
- Lower bounds and optimal algorithms
- Problems that seem very hard

Everyone Already Knows Many Algorithms!

Worked retail? You know how to make change!

Example:

- My item costs $4.37. I give you a five dollar bill. What do you give me in change?
- Answer: two quarters, a dime, three pennies
- Why? How do we figure that out?

Making Change

Problem:

- Give back the right amount of change, and…
- Return the fewest number of coins!

Inputs: the dollar-amount to return

- Also, the set of possible coins.
  (Do we have half-dollars? That affects the answer we give.)

Output: a set of coins

Note this problem statement is simply a transformation

- Given input, generate output with certain properties
- No statement about how to do it.

Can you describe the algorithm you use?

A Change Algorithm

1. Consider the largest valued coin
2. How many go into the amount left?
3. Add that many of that coin to the output
4. Subtract the amount for those coins from the amount left to return
5. If the amount left is zero, done!
6. If not, consider next largest valued coin, and go back to Step 2

Is this a “good” algorithm?

What makes an algorithm “good”?

- Good time *complexity*. (Maybe space complexity.)
- Better than any other algorithm
- Easy to understand

How could we measure how much work an algorithm does?

- Code it and time it. Issues?
- Count how many “instructions” it does before implementing it
- Computer scientists count basic operations, and use a rough measure of this: order class, e.g. O(n lg n)

Evaluating Our Greedy Algorithm

How much work does it do?

- Say C is the amount of change, and N is the number of coins in our coin-set
- Loop at most N times, and inside the loop we do:
  - A division
  - Add something to the output list
  - A subtraction, and a test
- We say this is O(N), or linear in terms of the size of the coin-set

Could we do better?

- Is this an *optimal algorithm*?
- We need to do a proof somehow to show this

You're Being Greedy!

This algorithm is an example of a family of algorithms called *greedy algorithms*

Suitable for **optimization problems**

- There are many *feasible answers* that add up to the right amount, but one answer is **optimal** or best (fewest coins)

Immediately greedy: at each step, choose what looks best now.
No "look-ahead" into the future!

What's an **optimization problem**?

- Some subset or combination of values satisfies problem *constraints* (*feasible solutions*)
- We have a rule to judge feasible solutions. One is best: the *optimal solution*

Does Greed Pay Off?

Greedy algorithms are often efficient.

Are they always right? Always find the optimal answer?

- For some problems.
- Not for checkers or chess!
- Always for coin-changing problem? Depends on coin values
  - Say we had a 11-cent coin
  - What happens if we need to return 15 cents?
- So how do we know?

In the real world:

- Many optimization problems
- Many good greedy solutions to some of these

Formal algorithmic description

A full, formal description of an algorithm would have the following components:

- Problem description (could be brief)
- Inputs
- Outputs
- Assumptions
- Strategy overview
  - Perhaps just 1 or 2 sentences outlining the basic strategy, including the name of the method you are going to use for the algorithm
- Algorithm description
  - If listed in English (as opposed to pseudo-code), then it should be listed in steps

Change solution (greedy)

Problem description: providing coin change of a given amount in the fewest number of coins

Inputs: the dollar-amount to return. Perhaps the possible set of coins, if it is non-obvious.

Output: a set of coins that obtains the desired amount of change in the fewest number of coins

Assumptions: If the coins are not stated, then they are the standard quarter, dime, nickel, and penny. All inputs are non-negative, and dollar amounts are ignored.

Strategy: a greedy algorithm that uses the largest coins first

Description: Issue the largest coin (quarters) until the amount left is less than the amount of a quarter ($0.25). Repeat with decreasing coin sizes (dimes, nickels, pennies).

Another Change Algorithm

Give me another way to do this?

Brute force:

- Generate all possible combinations of coins that add up to the required amount
- From these, choose the one with smallest number

What would you say about this approach?

There are other ways to solve this problem

- *Dynamic programming*: build a table of solutions to small subproblems, work your way up

Change solution (brute-force)

Problem description: providing coin change of a given amount in the fewest number of coins

Inputs: the dollar-amount to return. Perhaps the possible set of coins, if it is non-obvious.

Output: a set of coins that obtains the desired amount of change in the fewest number of coins

Assumptions: If the coins are not stated, then they are the standard quarter, dime, nickel, and penny. All inputs are non-negative, and dollar amounts are ignored.

Strategy: a brute-force algorithm that considers every possibility and picks the one with the fewest number of coins

Description: Consider every possible combination of coins that add to the given amount (done via a depth-first search). Return the one with the fewest number of coins.

Motivating problems

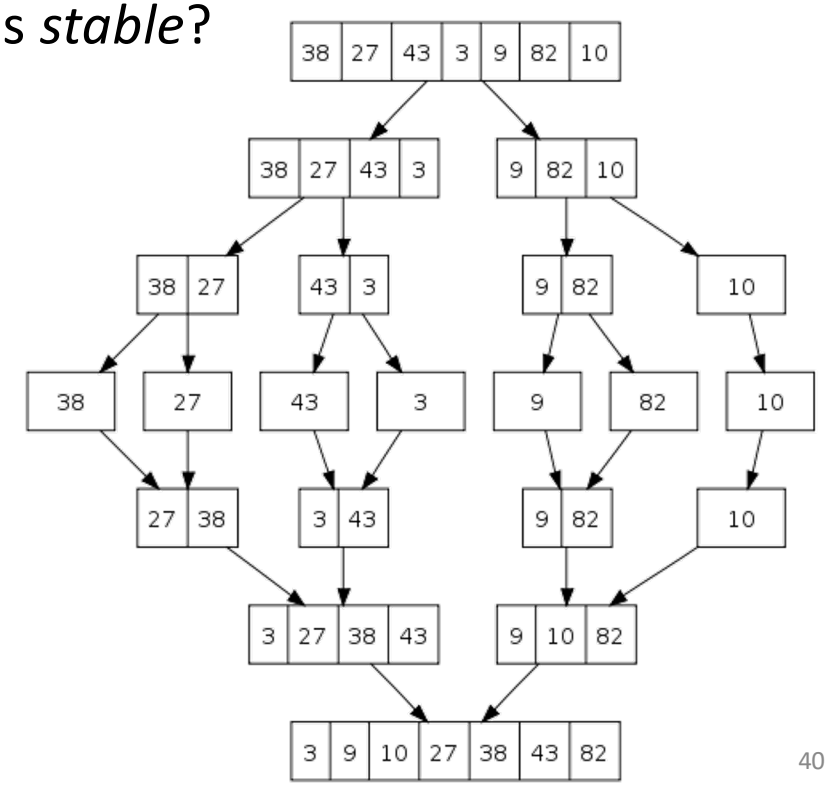
Motivating problem: Sorting

How do you implement a general-purpose sort that is as efficient as possible in both space and time, and is *stable*?

One solution is mergesort

- We'll see later why quicksort, heapsort, and radix sort are not sufficient

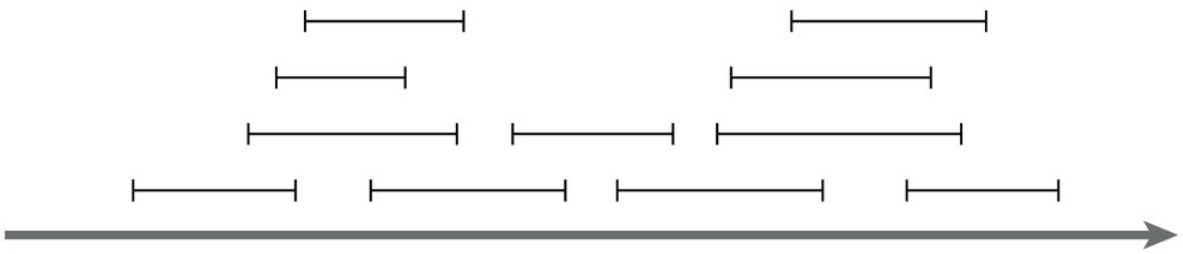
This is an application of both *sorting* and *divide and conquer*



Motivating problem: Interval scheduling

Interval scheduling

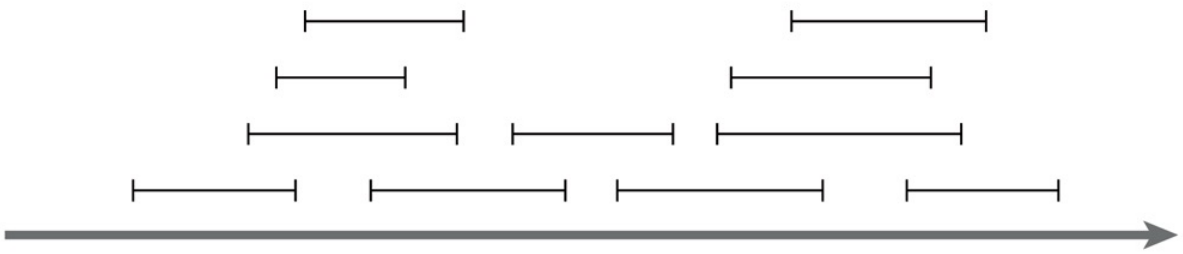
- Given a series of requests, each with a start time and end time, maximize the number of requests scheduled
- This is solved by a *greedy* algorithm
- Most of the CS 2150 algorithms you've seen are greedy algorithms: Dijkstra's shortest path, both MST algorithms, etc.



Motivating problem: Weighted interval scheduling

Weighted interval scheduling

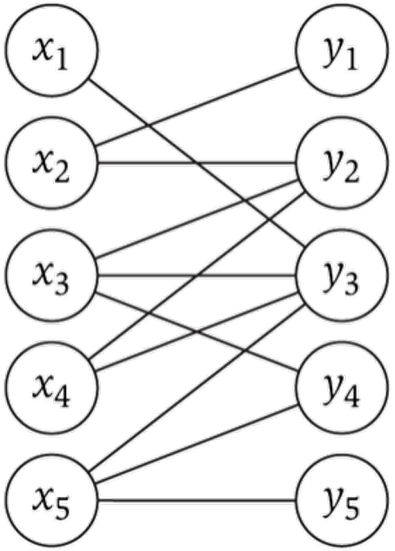
- Same as the regular interval scheduling, but in addition each request has a cost associated with it
- The goal is to maximize the cost from scheduling the items
- This is solved by *dynamic programming*



Motivating problem: Bipartite matching

Bipartite matching

- Given a graph *G*, find the maximum sub-graph of *G* that partitions *G* into sets *X* and *Y* such that no node from *X* is connected to a node in *Y*, and vise-versa
- Example: given a series of requests, and entities that can handle each request (such people, computers, etc.), find the optimal matching of requests to entities
- This is a *network flow* problem



Motivating problem: Independent set

Independent set

- Given a graph *G*, find the maximum size subset *X* of *G* such that no two nodes in *X* are connected to each other
- This is a *NP-complete* problem
- You've seen TSP (travelling salesperson problem) in CS 2150, which is a NP-complete problem