

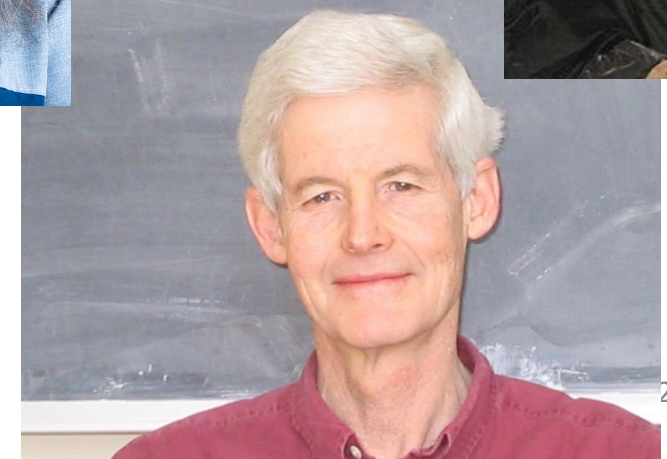
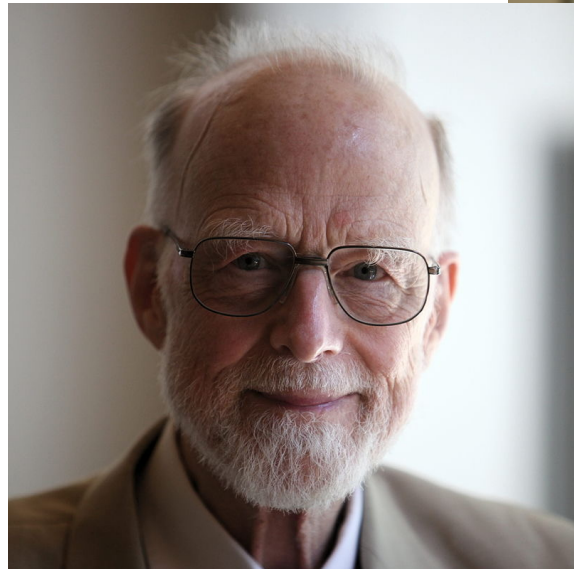
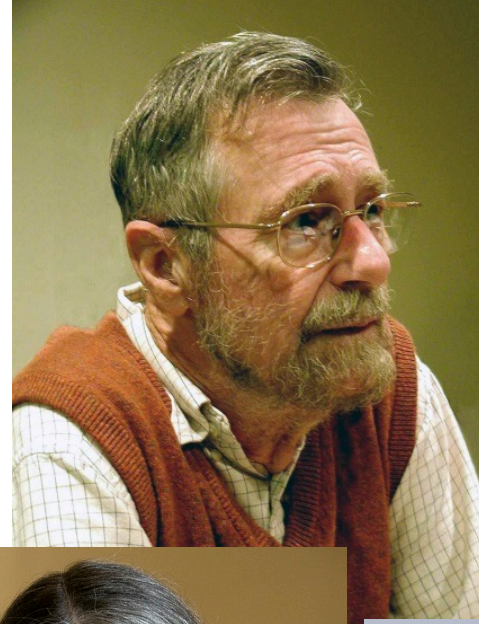
Data Structures and Algorithms 2

Lecture 1: Introduction and Logistics

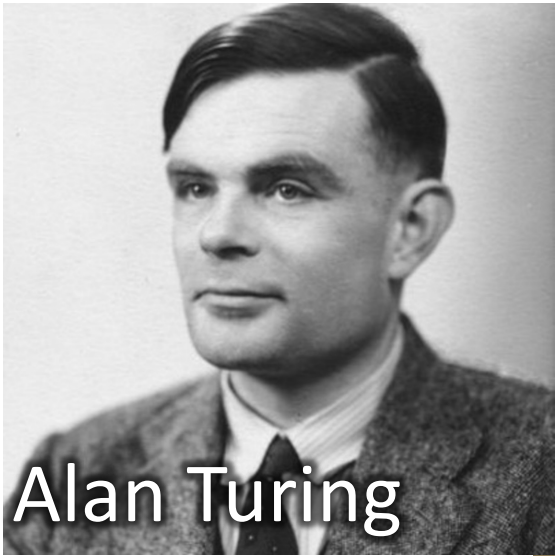
Co-instructors: Robbie Hott and Tom Horton

Fall 2023

Who's Who in Algorithms?



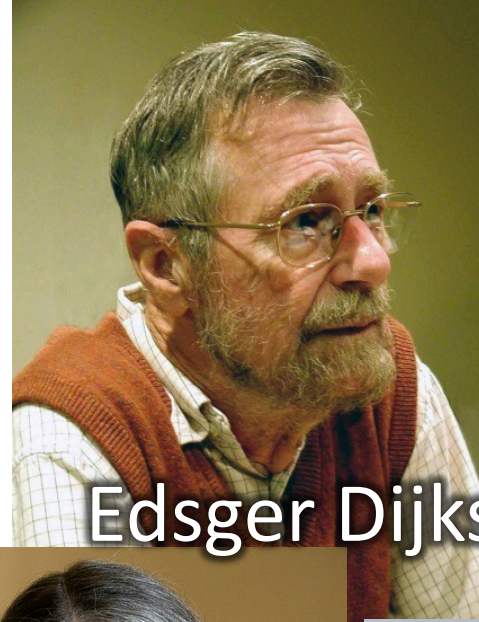
Who's Who in Algorithms?



Alan Turing



Ada Lovelace



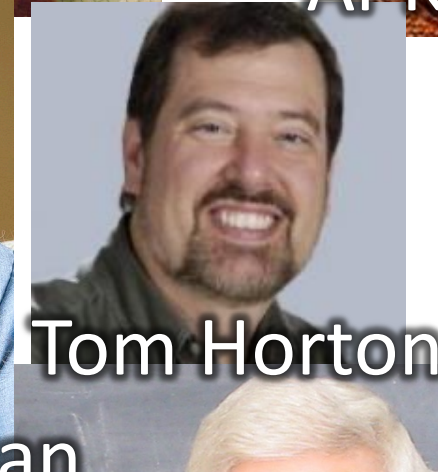
Edsger Dijkstra



Al-Khwarizmi



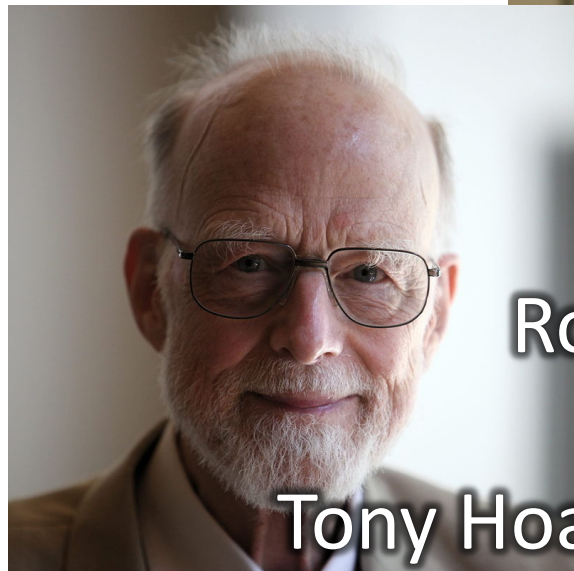
Radia Perlman



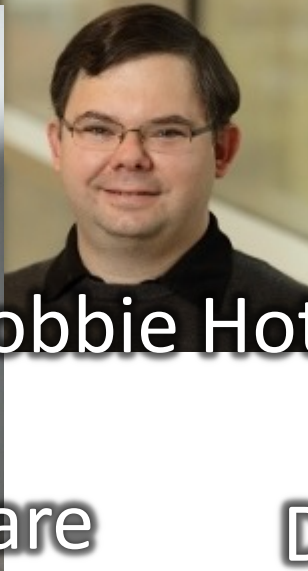
Tom Horton



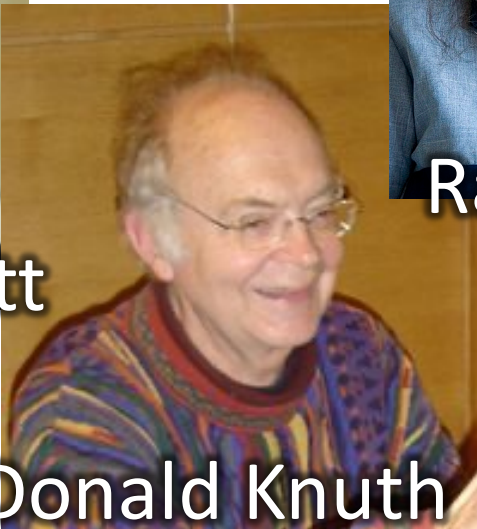
Gauss



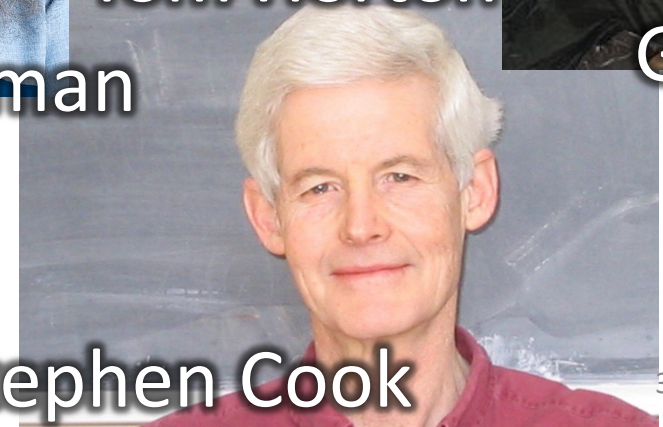
Tony Hoare



Robbie Hott



Donald Knuth



Stephen Cook

A Historic Perspective

Euclid



Al-Khwarizmi



Gauss



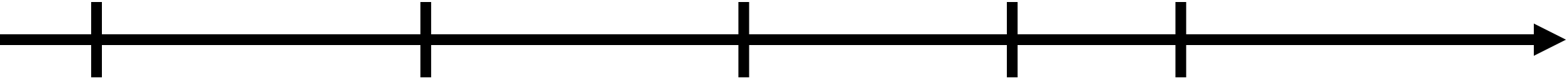
Ada Lovelace



Stephen Cook



Radia Perlman



300 BC

800

1600

1800

1900



Alan Turing



Edsger Dijkstra



Don Knuth

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]

Being unambiguous is not always easy!

[An example](#)

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
jrhatt@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

Overview of Platforms

Canvas

- Mostly for linking to other things

Course Webpage

- <https://uva-cs.github.io/cs3100-f23>
- Contains syllabus, coursework instructions, lecture slides, lecture recordings

Gradescope (linked through Canvas)

- For submitting all assignments, viewing grades, requesting regrades

Discord

- Link will be announced soon!
- For all communication, including: announcements, (some) office hours, memes, collaboration, finding teams, etc.

Piazza (available through Canvas)

- For Q&A about content and assignments
- Instructors, TAs and other students read and answer, so ask here rather than email instructors

Requirements

Discrete Math and Theory 1 (CS 2120 or CS 2102) with C- or higher

Data Structures and Algorithms 1 (CS 2100) with C- or higher

Logarithms, exponents, summations, derivatives, limits & series (Calc I)

Tenacity

Inquisitiveness

Creativity

Note: CS2100 pre-req taken seriously. Don't meet it? Need approval or you will be dropped (after add deadline). ☹️

Warning

This can be a very challenging class

- Hard material that combines problem-solving, logic, math and programming
- “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



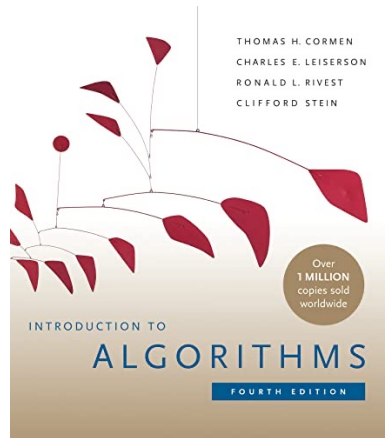
- All of these are important.
- Realistically, IMHO it's impossible to get all the “book knowledge” from lectures and slides!

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.



Freely accessible online via
the UVA library
[Follow this link!](#)

Cormen et al. (CLRS) *Introduction to Algorithms*
4th Edition. (3rd edition OK, but...)

We may also ask you to
read other online materials!

Units and Assignments

Five Units:

1. Graph Algorithms
2. Divide and Conquer
3. Greedy Algorithms
4. Dynamic Programming
5. Network Flow, Reductions, Machine Learning

Course work for each unit will include:

- Quiz (in-class, closed book)
- A Programming Assignment (PA)
- A Problem Set (PS)
- One or more In-class Activities (ICs)

Additional coursework:

- “PA0”, a warm-up exercise on using LaTeX typesetting
- Several surveys

Course Learning Objectives

By the end of this course, students will:

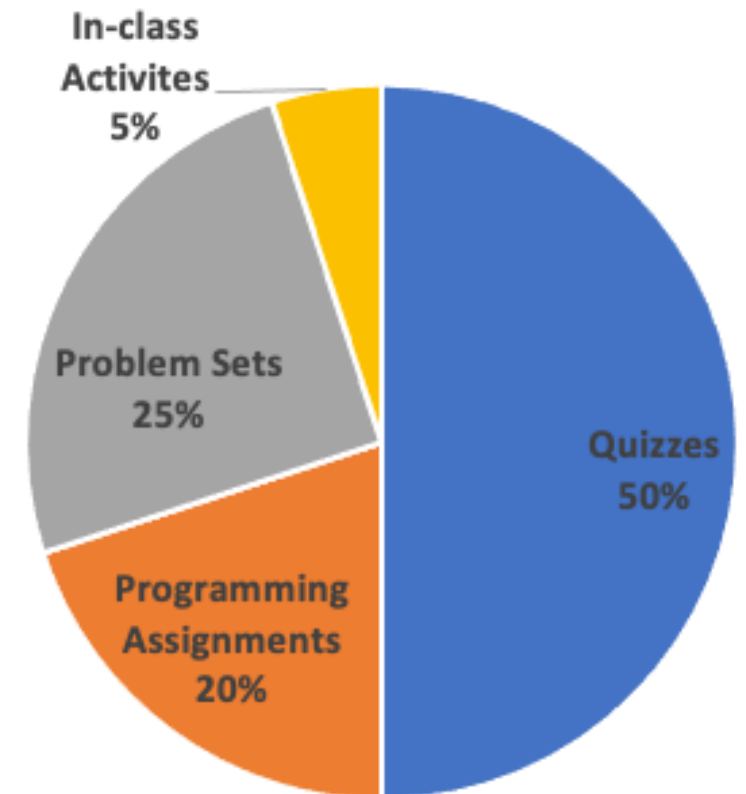
- **Understand and implement more advanced data structures**, including (but not limited to) graphs and sets (e.g., find-union). Be able to analyze the asymptotic complexity of the operations of these structures and use them to solve problems.
- **Understand a variety of problem solving paradigms such as dynamic programming, divide-and-conquer, and greedy algorithms**. Be able to implement a variety of algorithms, including Breadth-First (and Depth-First) Search, Dijkstra's algorithm, Ford-Fulkerson, and others.
- **Analyze a new problem and select an appropriate problem-solving technique**. Be able to construct an algorithm to solve the problem, prove the correctness of their algorithm, and analyze the asymptotic behavior of the algorithm.
- **Analyze behavior of algorithms with recursive, and other more advanced properties**. Solve recurrence relations for their closed form either manually or via the Master Theorem. Prove both upper and lower bounds on the behavior of such algorithms.

Note: department course specification also includes an introduction to machine learning and topics on ethics and machine learning.

Grading Breakdown

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

- Quizzes: 50%
- Programming Assignments (PAs): 20%
 - Reminder: one per unit, plus PA0 (LaTeX warmup)
- Problem Sets (PSs): 25%
- In-Class Activities (5%)
- Surveys: missing a required one is a 0.5% deduction on final grade



Quizzes

In-person, in your assigned section's classroom

- Oct. 5: Quiz 1 (Graphs), Quiz 2 (Divide and Conquer)
- Nov. 9: Quiz 3 (Greedy), Quiz 4 (Dynamic Programming)
- Dec. 12 (7-10 pm): Quiz 5 (Network Flow, Machine Learning), Re-take Questions

“Re-take” Opportunity for Quizzes 1-4

- During final exam period (along with Quiz 5)
- Available only if you made a reasonable first attempt on quiz
- Answer 2-3 questions on that unit
- Grade on those questions will replace up to 20 missing points on score from first attempt of that unit's quiz
 - Example: First attempt score is a 65/100. You get 15 of 20 on re-take. Final score: 80/100.

Problem Sets (PSs)

Goal: Think carefully and practice problem solving with techniques discussed during the unit

Grading: We will assign points for each question based on a limited set of thresholds

What are they like? A set of approximately 5-6 questions

- Concept-level questions, details of what we've studied
- Design an algorithm for a given problem (pseudocode, words)
- Analyze an algorithm (mathematically), provide a proof of correctness

Collaborate? Yes, in groups of up to 5

- But everyone must prepare and submit an individual write-up
- Submissions must be formatted in LaTeX! (more later)

Programming Assignment (PAs)

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

Grading: Primarily based on passing test-cases on Gradescope

- Include comments: any sources, collaborators
- We may ask you to answer short discussion questions about your solution

Languages accepted: Python (3.10.6), Java (19.0.2)

Collaborate? Not in writing or debugging the code, but...

- Can discuss **the problem** and **the overall strategy** with other students
- Must list these people in your submission
- Cannot share code, look at each other's code! (Read syllabus carefully!)

In-Class Activities (ICs)

Goal: Learn collaboratively and problem solve in the classroom (with the professor, TAs, each other)

- We believe students learn better when coming to class
- ICs will **not be recorded or announced** in advance.

Grading: Primarily based on making an effort (all ICs worth 5% total)

- Write solutions on a worksheet, turned in at the end of the lecture
- Discuss and work in groups, but individual write-up and submission

What if I miss? No penalty for missing an IC as long as you:

- Work on the problems **by yourself**
- Submit the worksheet **by 9am on the 2nd day after the missed class**
- One IC score will be dropped

PA0 – LaTeX Warmup

PA0 is out this week!

- Learning LaTeX
 - LaTeX (pronounced “LAH-tech”) is a markup language where you mix text and commands in a .tex file, run process the file to get a publication-quality PDF output file
 - Created to format mathematical symbols and texts
 - You can create .tex files and run LaTeX in the cloud using a site called Overleaf
- PA0 is due right after the Add deadline (but don’t wait that long!)

Academic Integrity

Collaboration Encouraged!

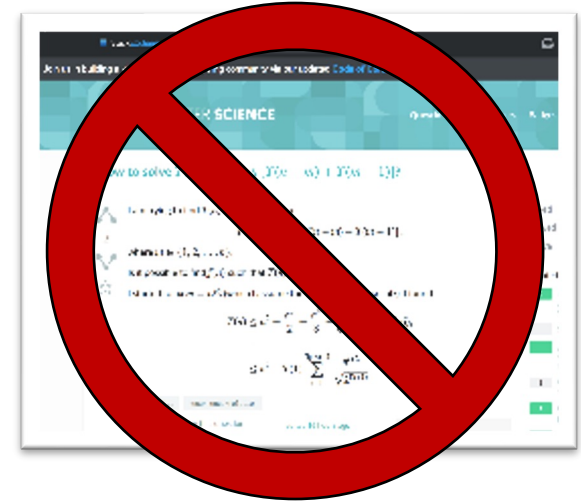
For PSs: groups of up to 5 per assignment (you + 4)

- List your collaborators (by name and UVA computing ID)
- OK to discuss problem, approach to solution, even details about solution, but... collaboration is "whiteboard only"
- Write-ups (.tex files) must be created **independently**
- **DO NOT** share written notes / pictures / code / etc
- **DO NOT** share documents (ex: Overleaf)

Be able to explain any solution you submit!

DO NOT seek published solutions online

See syllabus about online code examples



Academic Integrity

Collaboration Encouraged!

For PAs: you can talk, strategize, design,... together, but creating and debugging code must be done **alone**

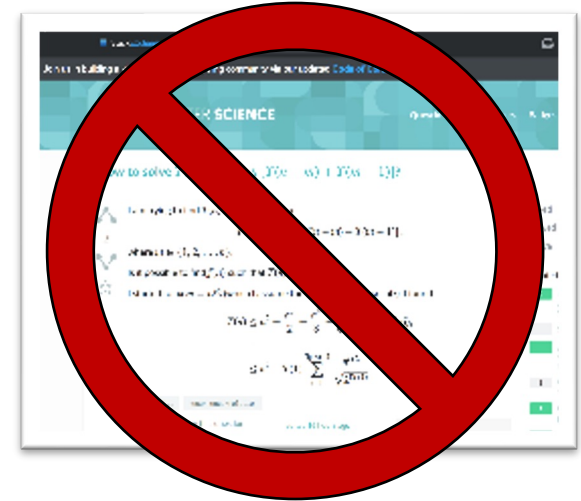
- List your collaborators (by name and UVA computing ID)
- **DO NOT** share pieces of code or programs or files
- **DO NOT** share debugging of code

Be able to explain any solution you submit!

DO NOT seek published solutions online

See syllabus about online code examples

See syllabus about generative AI



Late Policy

Deadlines are important in life! Plan for them, aim to meet them!

But, sometimes things happen in life, so....

- PAs and PSs may be submitted 48 hours after the deadline *with permission*
- Requests must be made using our online form (see course website)
 - Must include a valid reason
 - Must be made 24 hours before the deadline
 - If made less than 24 hours before the deadline, must include additional justification
- No penalty for late submissions within this window
- Significant work must be submitted before the posted deadline

Sometimes serious circumstances affect a student's ability to participate in their classes (e.g. severe illness, family situation, personal crisis).

If this happens to you, please contact your instructor as soon as you can!

Generative AI

For each assignment, we'll say if and how generative AI tools can be used

- When you use a tool, you must properly document and credit the tool (including the prompt)
 - Failure to do this is serious violation of academic integrity!
- Keep in mind tools may produce incorrect results, or may result in plagiarism or copyright violations
- It is your responsibility—not the tool's—to assure the quality, integrity, and accuracy of work you submit
- **Having said all this, in CS3100 we will examine how such tools can be useful in the study and practice of algorithm design and analysis!**

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
- Course staff email: cs3100@cshelpdesk.atlassian.net
- Email: horton@virginia.edu or jrhott@virginia.edu
 - PLEASE: put CS3100 in subject line of all emails
- (No Discord DMs, please)

Pre-Course Survey



Let's Get Acquainted

Would you like to know a bit about me? 😊

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

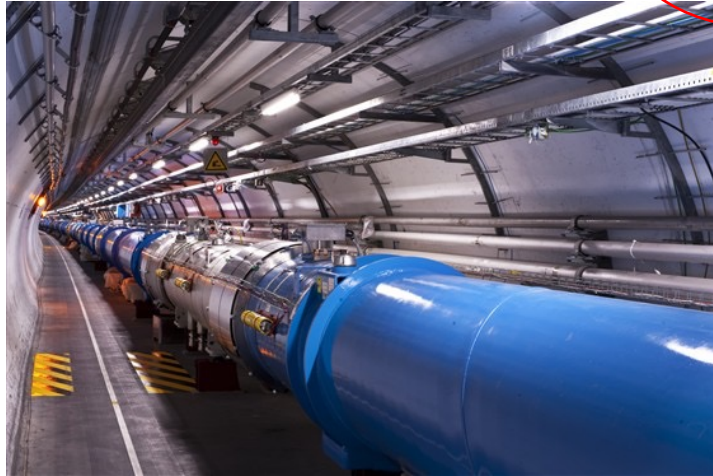
Assumptions: If the coins are not stated, then they are the standard quarter, dime, nickel, and penny. All inputs are non-negative, and dollar amounts are ignored.

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

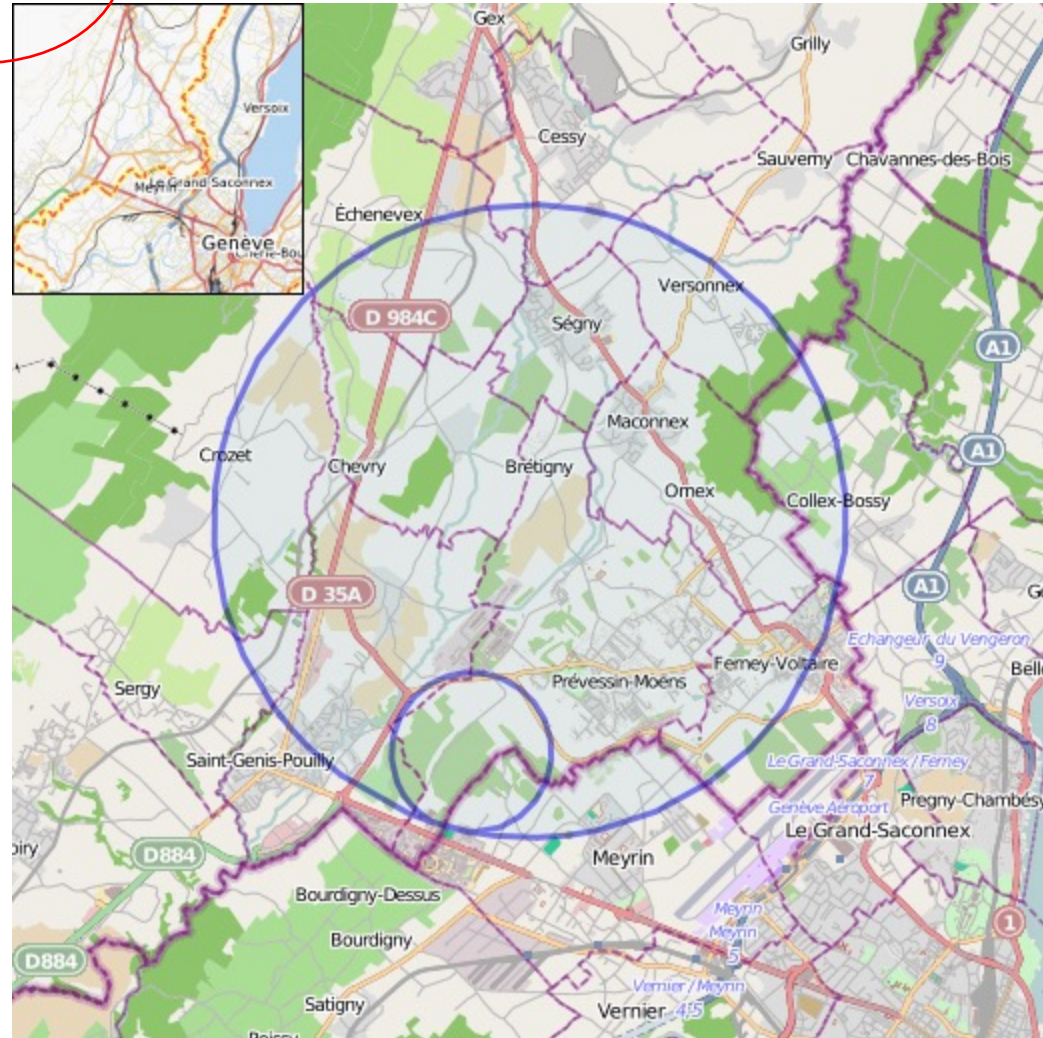
Description: Consider every possible combination of coins that add to the given amount (done via a depth-first search). Return the one with the fewest number of coins.

A motivating problem

Need an accurate approximation



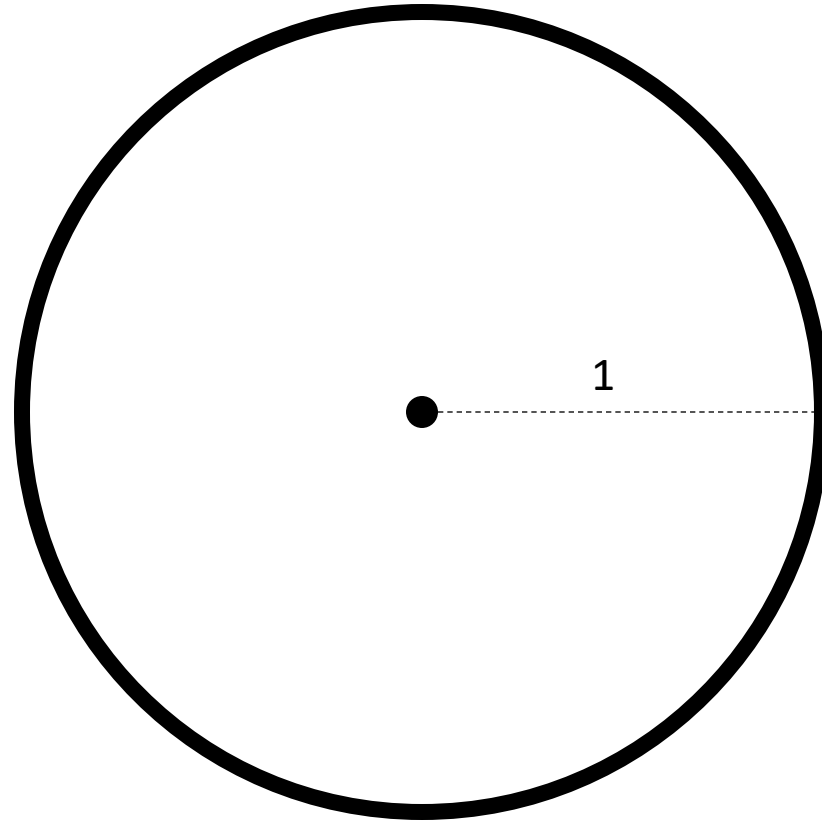
How much concrete do I need?



4.3km (2.7mi) diameter

π Approximation Algorithm

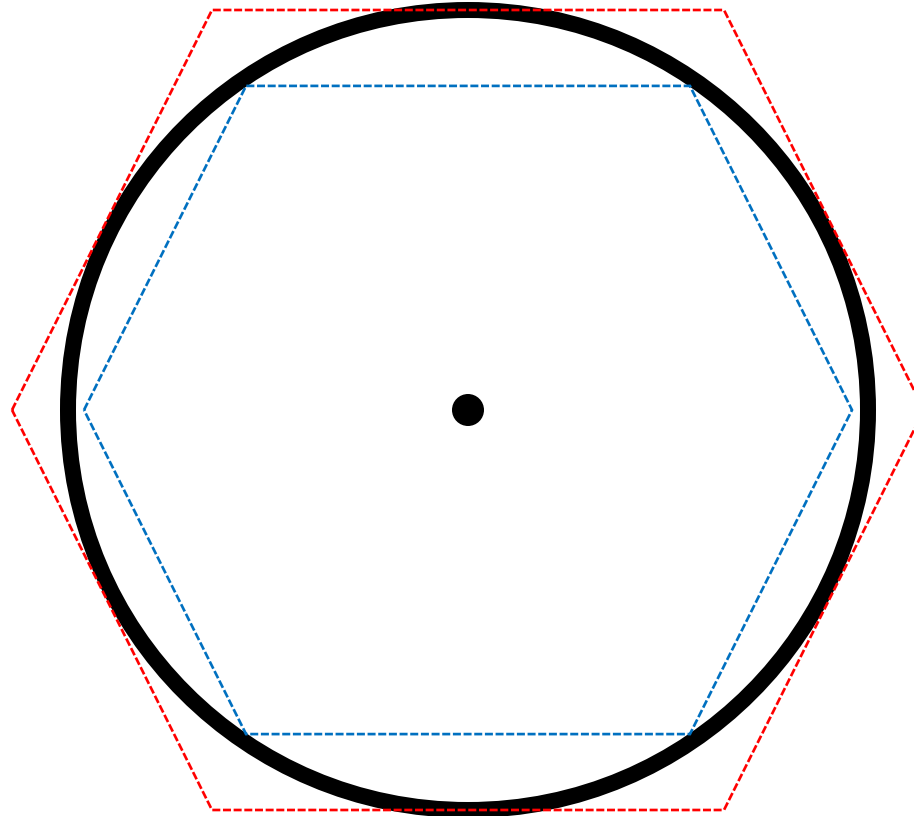
$$\pi = 3.14159265359\dots$$



$$\text{Circumference} = 2\pi$$

π Approximation Algorithm

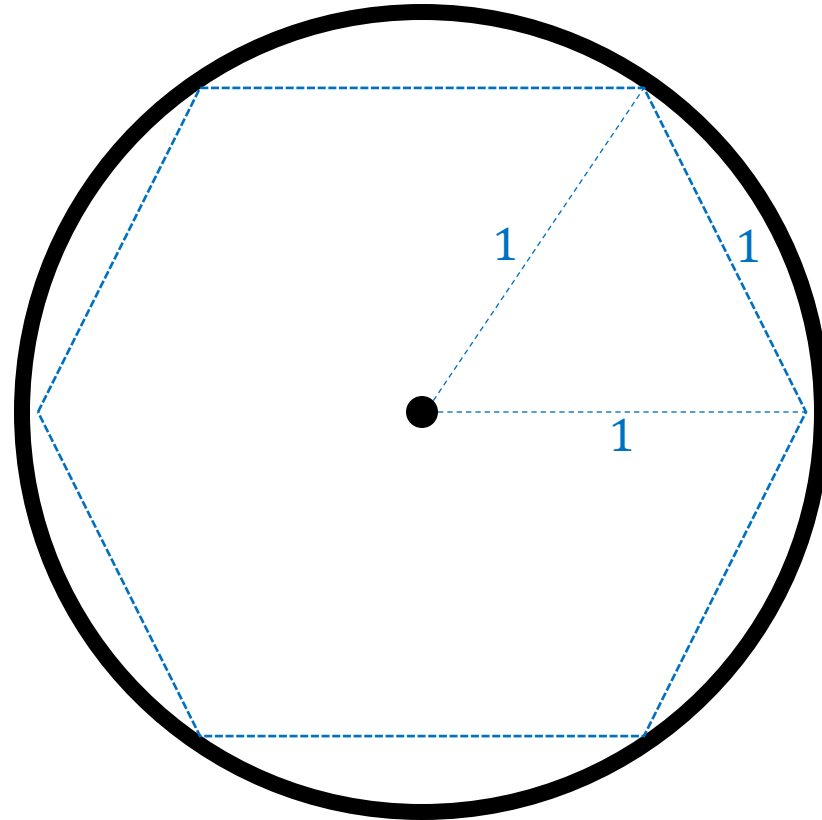
$$\pi = 3.14159265359\dots$$



$$\text{Perimeter} > 2\pi > \text{Perimeter}$$

π Approximation Algorithm

$$\pi = 3.14159265359\dots$$



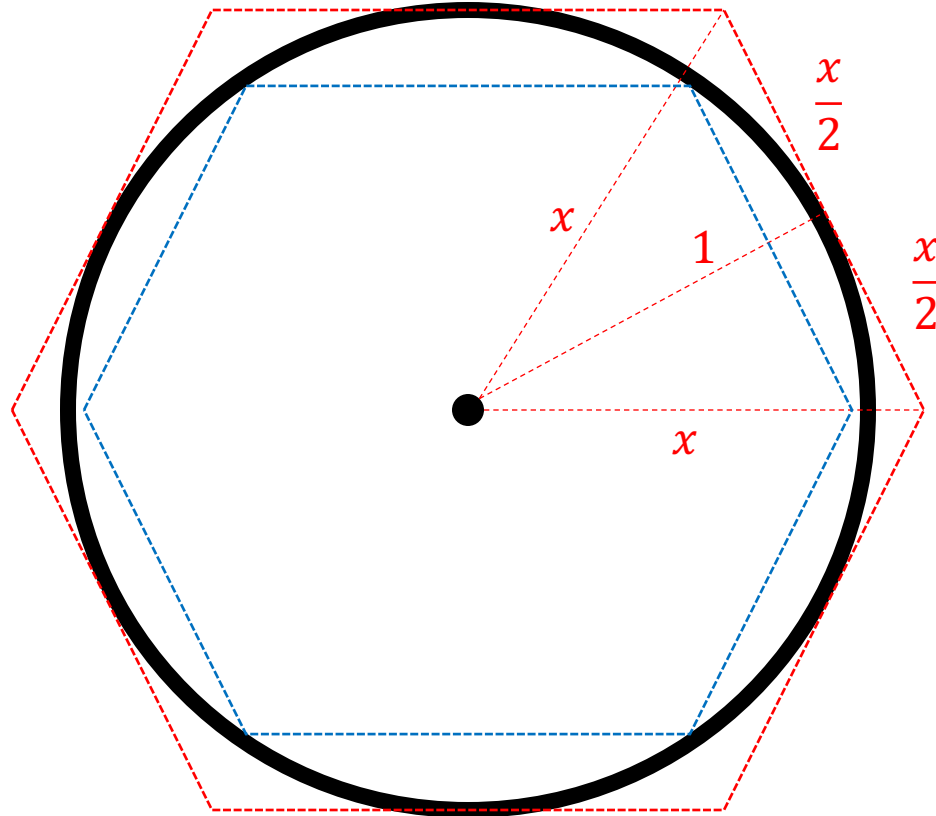
$$2\pi > \text{Perimeter} = 6$$

π Approximation Algorithm

$$\pi = 3.14159265359\dots \quad \text{1 digit correct}$$

Solve for x

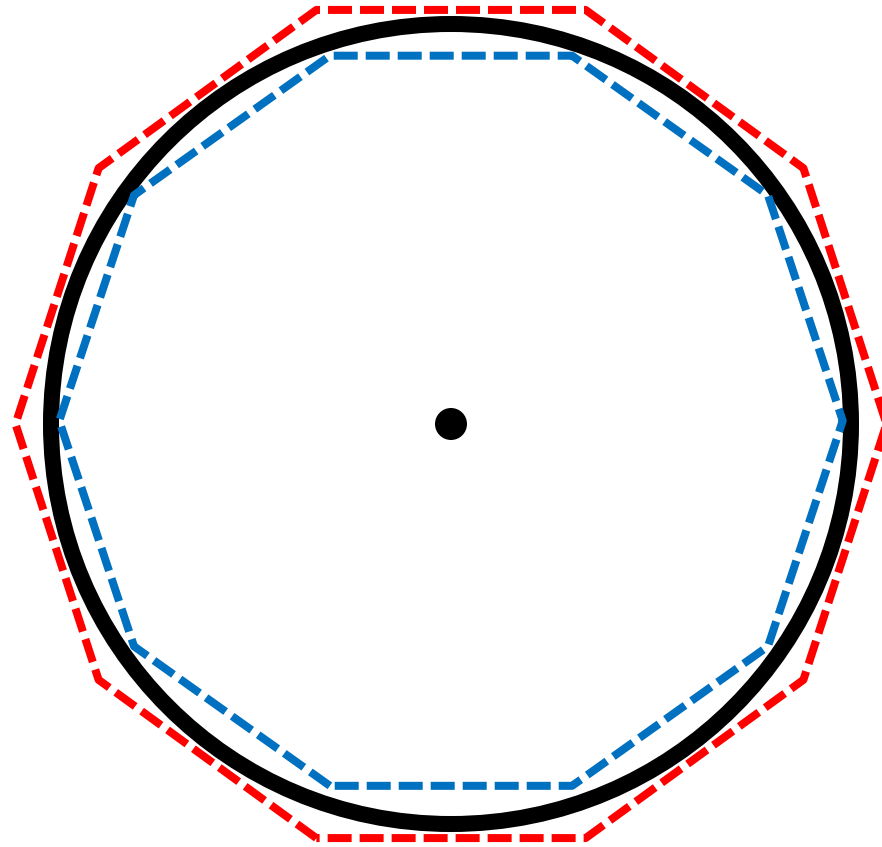
$$x = \frac{2}{\sqrt{3}}$$



$$\frac{12}{\sqrt{3}} = \text{Perimeter} > 2\pi > \text{Perimeter} = 6$$
$$3.46 > \pi > 3$$

π Approximation Algorithm

$\pi = 3.14159265359\dots$ 3 digits correct



$$6 + \frac{20}{70} = \text{Perimeter} > 2\pi > \text{Perimeter} = 6 + \frac{20}{71}$$
$$3.14285 > \pi > 3.14084$$

How to analyze this approach?

How fast do we “converge?”

How much work is needed to do better?



Better π Approximation (Ramanujan)

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)! (1103 + 26390k)}{(k!)^4 396^{4k}}$$

$\pi = 3.14159265358979323846264338327950288419716939937510582097494459$

$$k = 0$$

$$\pi \approx 3.1415927$$

8 digits per iteration!

$$k = 1$$

$$\pi \approx 3.1415926535897938$$

Next Steps

Review

Next class we will:

- Review how we analyze algorithms, including
 - Asymptotic notation, order classes, worst-case, etc.
- Remind you how all this works by applying/reviewing it with sorting and searching you did in previous course
 - Sequential search, binary search
 - Sorting: insertion, quicksort, maybe mergesort
- Review and/or introduce proofs from CS2120
 - Proof by induction
 - Introduce you to how this can be used for proving algorithm correctness