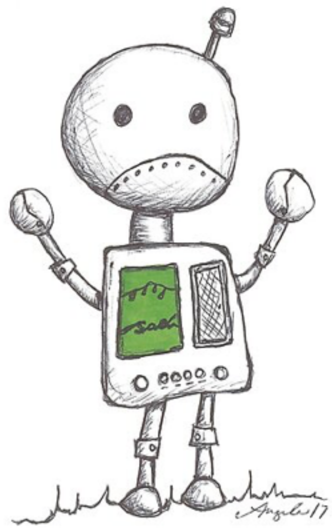
Limits of Computing, Course Wrap-Up



Take a look at the worksheet and start trying to fill it out!

Sam Wolfson

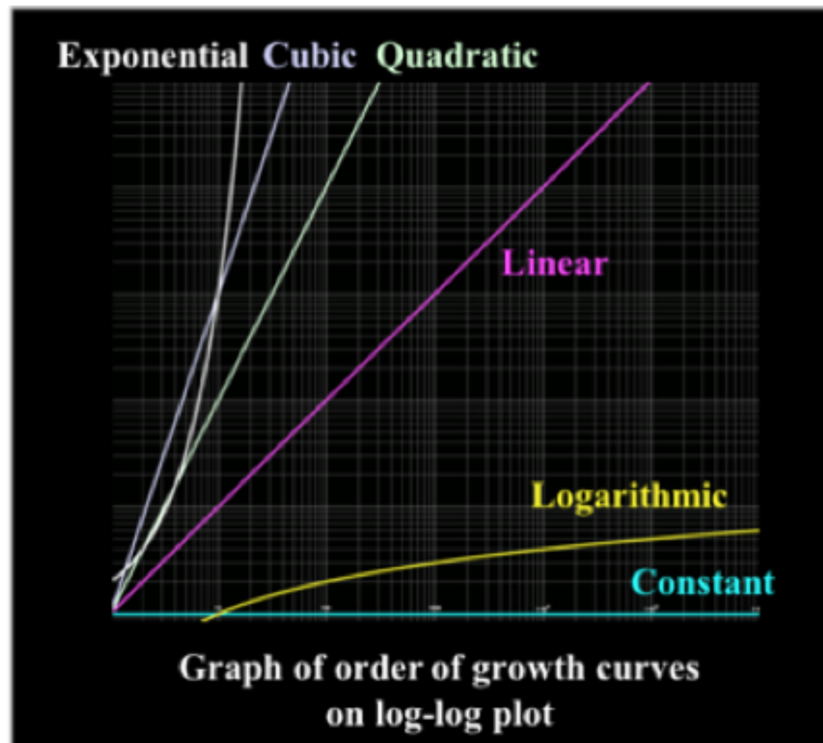
CSE 120, Winter 2020

Administrivia

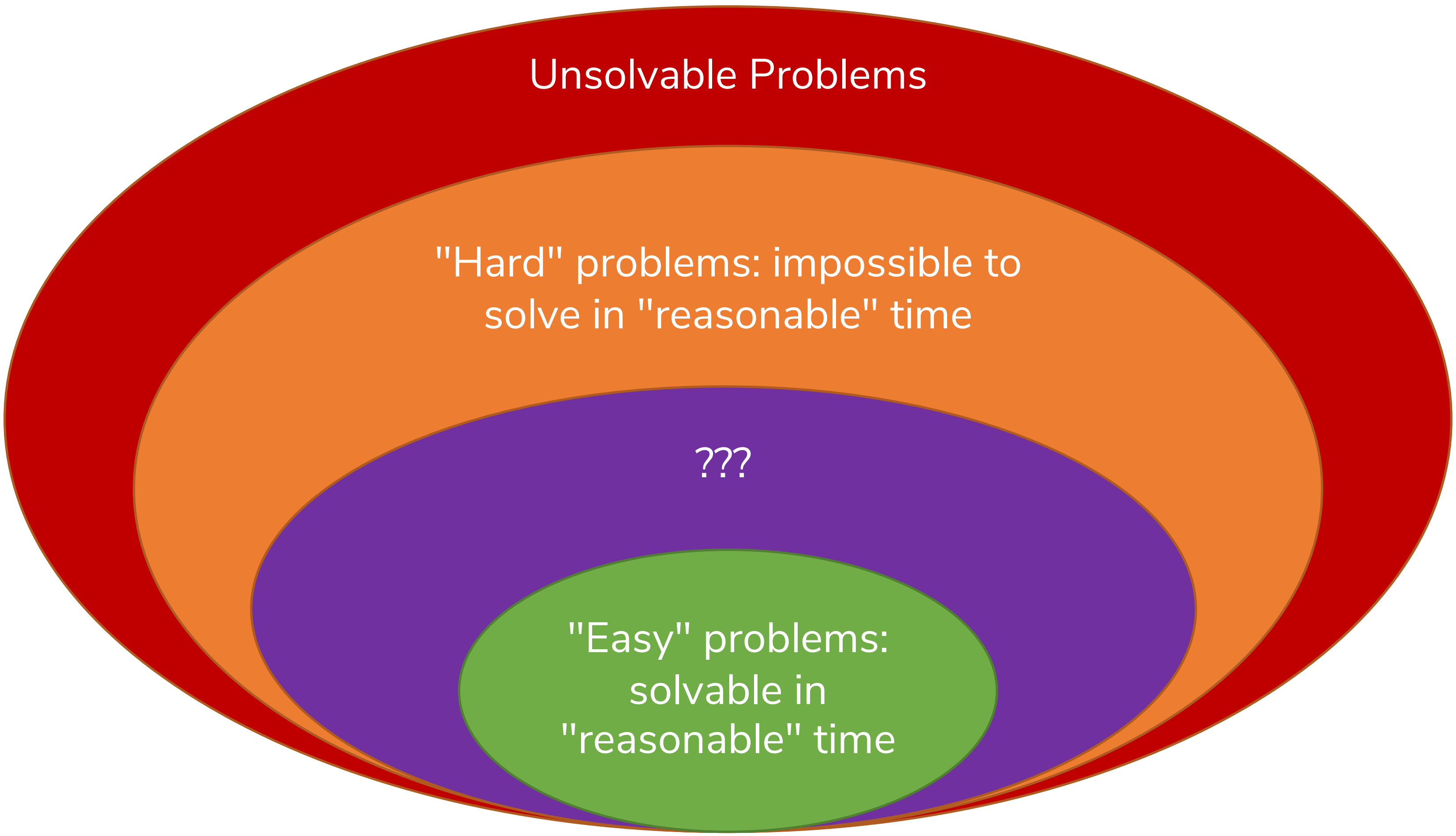
- Assignments: just the final project!
  - Instead of giving a presentation to the class, we ask that you record a video of yourself presenting.
    - If necessary, you and your partner can each present half separately then cut the videos together afterwards.
    - It's OK to make a recording of your screen and talk over it.
  - The assignment page and rubrics have been updated – please read through them and let us know if you have questions or concerns.
- Next week:
  - We will plan to hold office hours remotely (details to come)
  - The TAs and I will continue to monitor Piazza
  - Remote lecture on Computing for Good
- Course evaluation: https://uw.iasystem.org/survey/221453

Revisiting Algorithm Complexity

- Recall: the runtime of an algorithm is based on the size of its input (e.g., the size of an array)
  - The “order of growth”
- Different algorithms have different orders of growth:
  - Constant
  - Logarithmic
  - Linear
  - Quadratic
  - Cubic
  - Exponential



What problems do we encounter?



Problem Difficulty

- We call polynomial or faster problems “easy”
  - Runtime $\leq \mathcal{O}(n^b)$, where b is a constant ($\mathcal{O}(n), \mathcal{O}(n^2)$, …)
- Exponential, factorial, & slower problems are “hard”

	Easy easy				Hard		
	Linear	Logarithmic	Quadratic	Cubic	Exponential	Exponential	Factorial
	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

“Easy” Problems

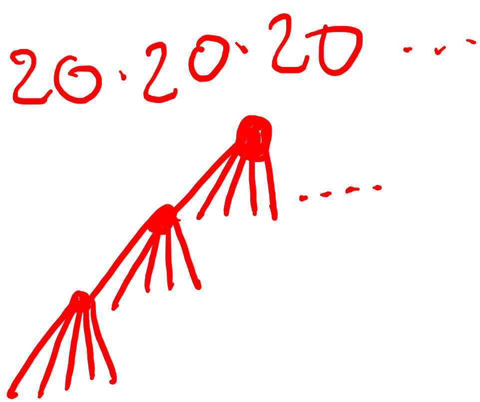
- Is this list sorted? $O(n)$
  - Look at each number and make sure it's greater than or equal to the one before it
- Do any two numbers in this array sum to 0? n^2
  - Check every number against every other number

- We say that these problems are "in P"
  - They can be solved in **polynomial time**

Notice: these are all yes/no problems (i.e., **decision problems**)

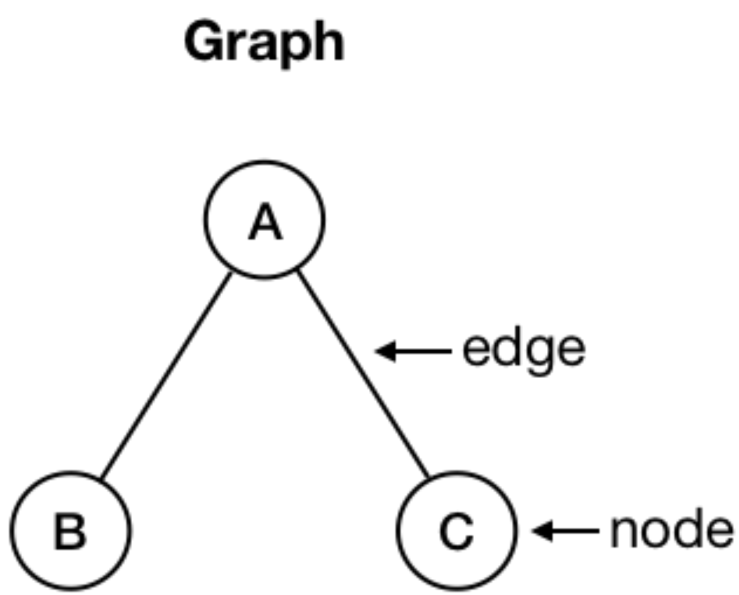
“Hard” Problems

- Given the chess board, is there a move that white can make that ensures white will **eventually** win?
    - There is an answer.
    - There's no way to compute it in any reasonable time.



This problem is **not** “in P.” It takes **exponential time.**

Let’s solve some problems!



Two nodes are **adjacent** if there is an edge directly connecting them.

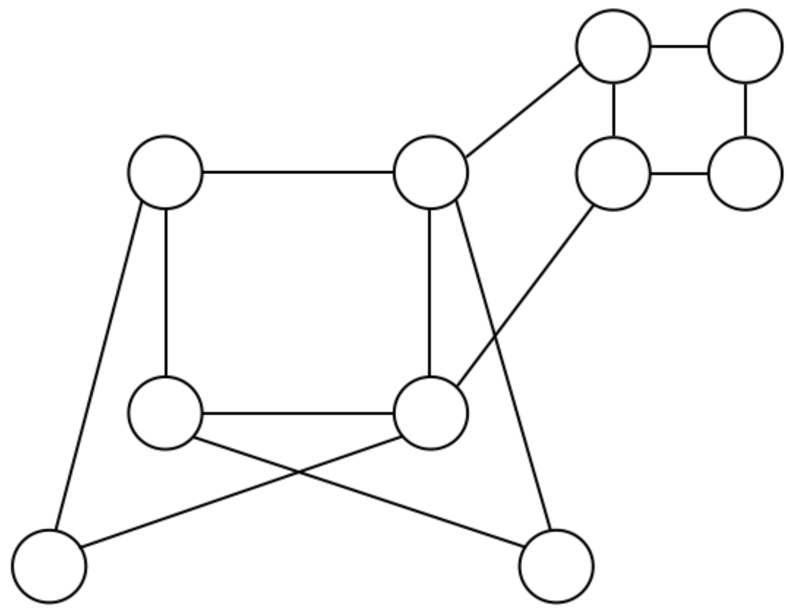
- A and B, B and C are adjacent.
- B and C are not.

- Problem: is it possible to color each node so that no two directly connected node are the same color?
  - On the first page of your worksheet, try to do this using only **two colors.**
  - Can you think of an *efficient* algorithm for this?

2-Colored Graph

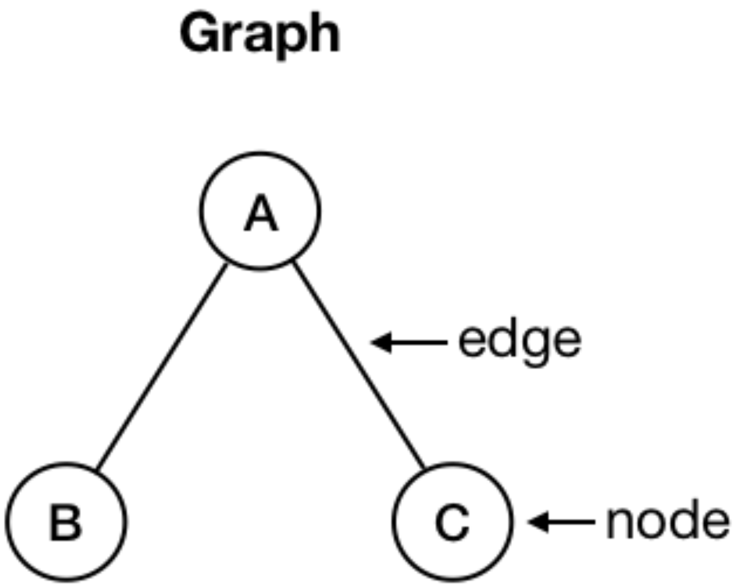
$O(n)$

- Is it possible to color each node so that no two directly connected nodes are the same color?
  - Start with a random node, and fill with a color
  - Fill every adjacent node with the other color
  - Do the same thing with every node adjacent to that node
  - Repeat until:
    - We find a node that can't be either color → answer is **NO**
    - The graph is completely filled in → answer is **YES**



This problem is "in P" (only need to look at each node once)

Let’s solve some problems!



Two nodes are **adjacent** if there is an edge directly connecting them.

- A and B, B and C are adjacent.
- B and C are not.

- Problem: is it possible to color each node so that no two directly connected node are the same color?
  - On the second page, try to do the same thing, but now you have **three colors** to work with.
  - Can you think of an *efficient* algorithm for this?

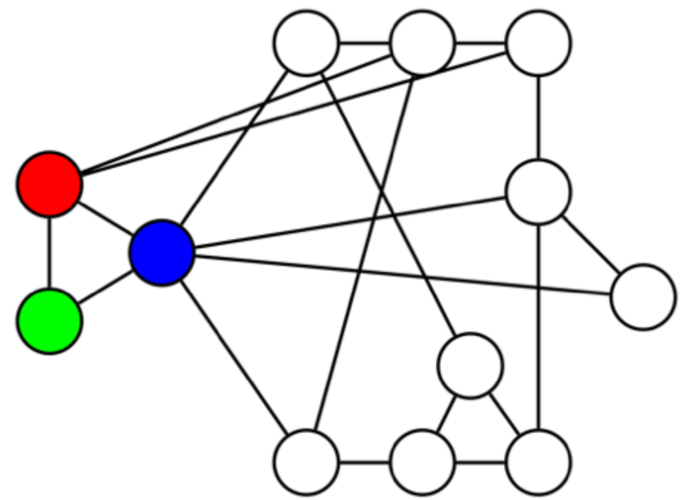
3-Colored Graph

$O(3^n)$

$3 \cdot 3 \cdot 3 \cdot \ldots$

- Is it possible to color each node so that no two directly connected nodes are the same color?
  - Start with a ran… a color
  - Fill all the adja… t color
  - What if we re… s will work??
    - **Backtrack** and try different colors
  - Repeat until:
    - We find a node where no combination of other colors in the graph will make it work → answer is NO
    - The graph is completely filled in → answer is YES

May need to try **every possible combination** of colors



This problem is **not** "in P" (why?)

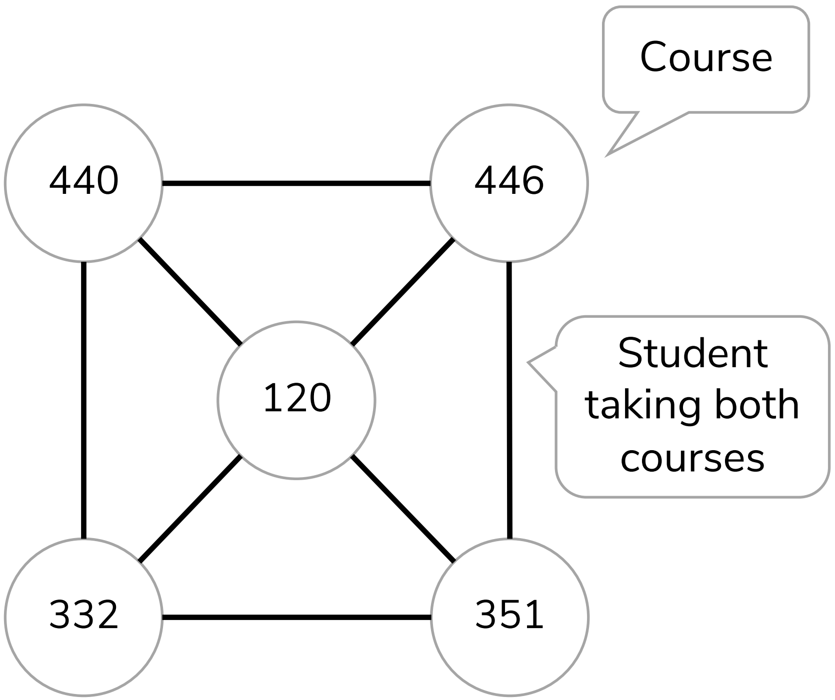
3-Colored Graph

- Runtime for our algorithm: $\mathcal{O}(3^n)$
  - In the worse case, we'd need to try every possible combination of colors!
- Does a polynomial algorithm exist?
  - This is an **unsolved problem**

This problem is "in **NP**"
We don't know of any polynomial time solutions
But we haven't been able to prove they don't exist
We can easily **check** whether a solution is correct

Example: Exam Planning

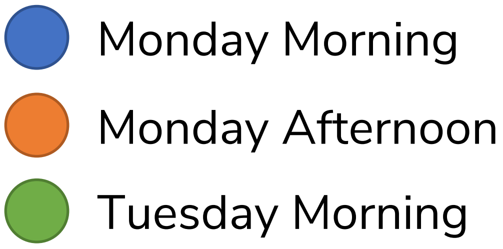
Student	CSE Courses
Sam	120, 440, 446
Eunia	120, 351, 446
Yae	120, 332, 440
Erika	120, 332, 351
Justin	120, 332, 351



We have three available exam slots.
How to schedule exams without conflicts?

Called a **reduction**

Equivalent to 3-coloring!



P vs. NP

- P problems have polynomial time algorithms
- NP problems have solutions that can be efficiently **verified**, but no known efficient way to **find them**

$1 million reward!

Big open problem in computer science: is P = NP?

What if P=NP?

"If P=NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found."

-Scott Aaronson

Beyond NP: Unsolvable Problems

- Are there problems that we can't solve, no matter how much time you're given?
  - Remember: an algorithm is only a **solution** if is produces the correct answer in a finite amount of time
  - A problem is **decidable** if it has a solution
- Some problems are **undecidable**

The Halting Problem

Is there a way to tell whether **`mystery`** will ever finish?

```
void mystery(String input) {
    /* here be dragons */
    .
    .
    .
}
```

Assume the function **`doesItHalt(String code, String args)`** exists:

- **Input**: a function's code **`code`** and the function's arguments **`args`**
- **Output**: **`true`** if that function halts, **`false`** if it doesn't

The Halting Problem

- Assume the function **doesItHalt(String code, String args)** exists:
  - **Input**: a function's code **code** and the function's arguments **args**
  - **Output**: **true** if that function halts, **false** if it doesn't
- Can we fool this function?

```
void trickster(String code) {
  if (doesItHalt(code, code)) {
    while(true == true) { };  // run forever
  } else {
    return;  // halt
  }
}
```

We're checking if **code** halts when given its own code as input!

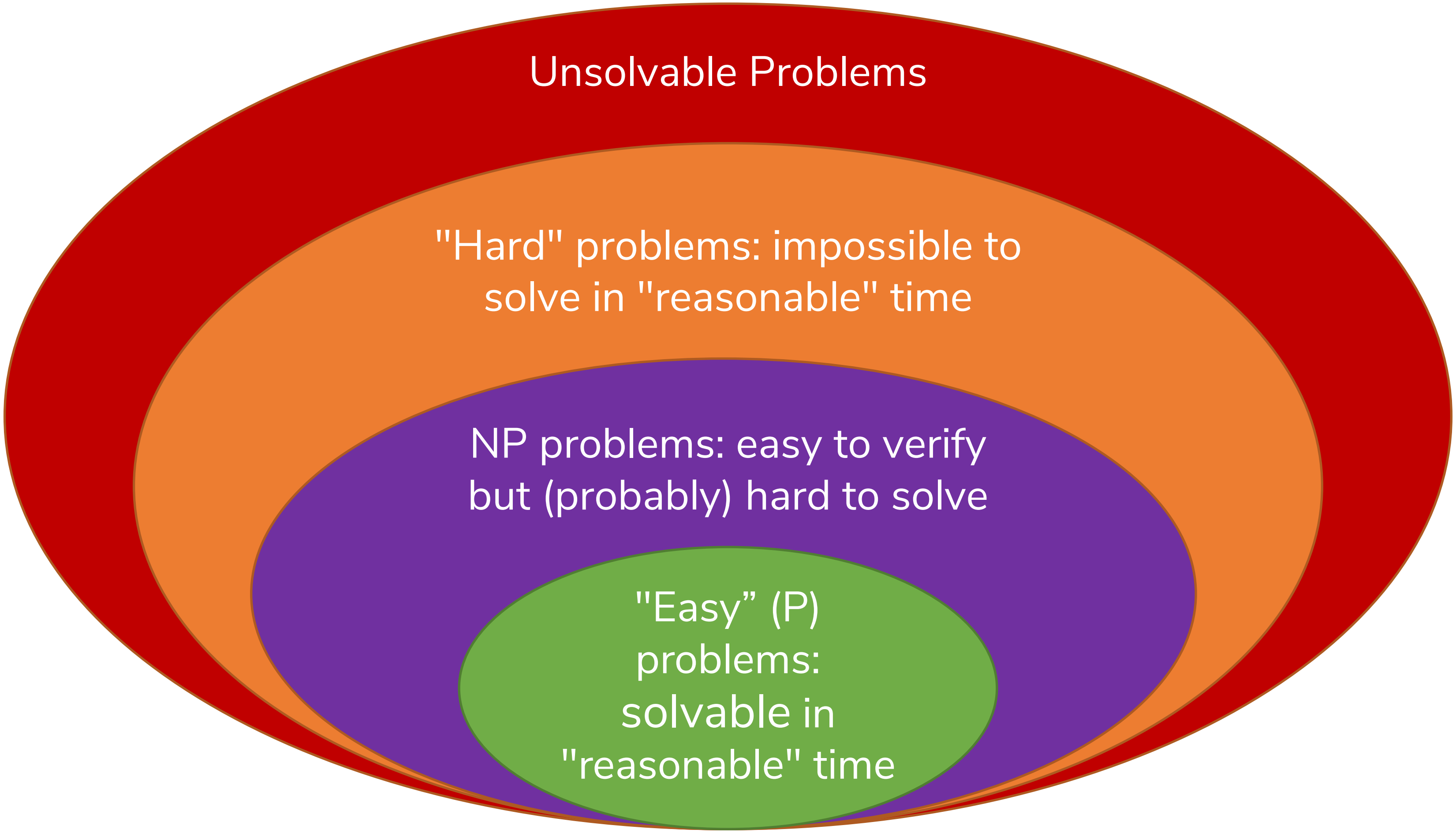
code(f) returns the code for **f** as a **String**

What happens if we run **trickster(code(trickster))**?
(i.e., give it its own code as input)

- If **doesItHalt** returns true, then **trickster** will run forever.
- If **doesItHalt** returns false, then **trickster** will halt.

Contradiction! This means that **doesItHalt** cannot exist.

The Problem Complexity Zoo



For a more in-depth version of this lecture:
https://uw.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=245f2b87-13d4-42f8-a967-ab3c016ce30b

Course Wrap-Up

- **What We've Learned**
- Lecture 1 Revisited
- Your Future Beyond CSE120

Source: DragoArt.com

Source: Project Gutenberg

Computational Thinking

- It's all about problem solving
  - How to attack your problem in a way that a computer can help
- Most important idea: abstraction!
  - Detail removal and generalization help us decompose complex problems
  - Use bits to represent *everything* (*i.e.* digitization)
  - Reuse and combine building blocks (algorithms) in ways that hopefully scale well

Building Blocks of Algorithms

- **Sequencing**
  - The application/execution of each step of an algorithm in the order given

```
fill(255);
rectMode(CORNERS);
rect(-r, -r, 0, r);
ellipse(0, -r/2, r, r);
```

- **Iteration**
  - Repeat part of algorithm a specified number of times

```
int i=20;
while(i<40) {
  line(i,40,i+60,80);
  i=i+60;
}
```

- **Selection**
  - Use of conditionals to select which instruction to execute next

```
if(mousePressed) {
  fill(0,0,255);
}
```

- **Functional Abstraction**
  - Break larger problem into smaller, reusable parts

Programming

- Learned our first programming language
  - Processing (Java syntax)
- Iterative design cycle:
  - The value of a precise specification
  - Design, prototype, implement, and evaluate
  - Testing and debugging
- Coding style and documentation
  - Proper commenting and formatting are essential for maintenance and collaboration

Some Big Ideas

- Computers can only do a small number of things
  - Execute *exactly* what you tell it to
- Computing has physical and theoretical limits
- The Internet is a physical realm
- Data is constantly generated, stored, and analyzed
  - And can be copied and distributed
- Machines can "think" and "learn"?
  - AI & the importance of probability and training sets

Social Context and Impact

- Impacts of computing:
  - Algorithms can have unintended consequences
  - Privacy and security (or lack thereof)
  - Social media influences the way we access and share information
  - Can improve the lives of those with disabilities (and everyone else) as well as those in developing countries

- Design matters!
  - Must keep in mind users and user interface
  - What are the ethical implications and whose values are we promoting?

Course Wrap-Up

- What We've Learned
- **Lecture 1 Revisited**
- Your Future Beyond CSE120

Why Study Computer Science?

- Massive impact on our lives and society as a whole

- Increasingly useful for *all* fields of study and areas of employment
  - Farmer – machines to help farm, drones for pesticides
  - Chef – analyze customer data: popular dishes, hours, etc.
  - Street Performers
  - Dancer, Gymnast – monitor and evaluate performance, diet, etc.

Computing in Your Future

- Computing and its data are inescapable
  - You generate “digital footprints” all the time
- Computing is a regular part of every job
  - Use computers and computational tools
  - Generate and process data
  - Dealing with IT people
  - Understanding the computation portion of projects
- Our goal is to help you make sense of the “Digital Age” that we now all live in

About Programming

- programming ≠ computational thinking
  - *Computational thinking* is knowing how to break down and solve a problem in a way that a computer can do it
  - *Programming* is the tool you use to execute your solution
  - We use programming in this course as a way of teaching computational thinking

- Can be learned, just like any other skill
  - It's not black magic; no such thing as a "coding gene"
  - Yes, at first it may be challenging and mind-bending – just like learning your first non-native language
  - My hope is that you will think differently after this course

Big Ideas of Computing

- Exposure to a broad range of topics in computer science
  - Not going to dive into the details
  - These are the motivations & the applications for programming (the tool)
  - Focus on what to be aware of to navigate the digital world

- **Goal:** become "literate" in computing
  - As new innovations arise, can you read about it, understand its consequences, and form your own opinion?
  - This course will ask you to *read*, *discuss*, and ~~*write*~~ present about computing

Course Wrap-Up

- What We’ve Learned
- Lecture 1 Revisited
- **Your Future Beyond CSE120**

Keep What You've Made!

- You've all done some amazing work!
  - We will make a "Student Showcase" Piazza post
- Make sure your files and programs are saved somewhere so you can access them later
  - Can re-download from Canvas submissions if necessary
  - Could be helpful as examples for future projects
- Portfolio
  - Your website will remain live until you disable it or graduate
  - You can download a copy of your website files using Cyberduck

Giving Back to CSE 120

- Enjoyed the class? Lots of ways to help out!
  - Feedback: course evaluations, feedback on final "quiz", send me an email with your thoughts 😊
  - Examples: Permission to show your work to future classes?
    - I'll make a "Student Showcase" Piazza post later
  - Recommendations: CSE120 will hopefully be offered in winter 2021 – tell your friends!

Future Courses

- Intro CS courses descriptions:
  - https://www.cs.washington.edu/academics/ugrad/overview/intro-courses

- Staff recommendations and descriptions:
  - https://docs.google.com/presentation/d/1JP0tMaCgTWYWM7ALeSNHWOqjv3iporODr1tB0rg8NvM/edit?usp=sharing

More CS at UW

- CSE 142 + CSE 143: Computer Programming I/II
  - Needed for declaring CSE major
- CSE 160 + CSE 163: Data Programming I/II
  - Recommended to take 142 first
- CSE/STAT/INFO 180: Intro to Data Science
  - A basic math prerequisite
- CSE 154: Web Programming
  - Must have taken 142, 143, or 160

Social Implications Courses

- Informatics
  - INFO 101: Social Networking Technologies
  - INFO 102: Gender and Information Technology
  - INFO 200: Intellectual Foundations of Informatics
  - INFO 270: Calling Bullshit: Data Reasoning in a Digital World

- Human Centered Design & Engineering
  - HCDE 210: Explorations in Human Centered Design

No More CS at UW or Break

- You are now somewhat programming-literate
  - Can automate tasks to make your life easier
  - More aware of possibilities of computing
  - Easier to interact with IT/CS staff at work
- Figure out what will be most useful to you
  - Some languages specific to type of work (e.g. R, MATLAB, Ruby on Rails, SQL)
  - Learn on your own via the Internet:

Google CS Education

CODE

KHAN ACADEMY

edX

coursera

Making the Most of College

- Seek out experiences that lead to new experiences
  - Build skills, interests, relationships
  - Meet new people, join interesting clubs, go on adventures

- Don't go it alone – find a friend group for classes

- Take advantage of educational opportunities
  - **Research:** https://www.washington.edu/undergradresearch/students/find/
  - **Student Groups:** ACM, Animation Research Labs, Husky Robotics, WOOF3D, etc.
  - **Classes:** non-major courses, P.E., languages, anything of interest

- Take care of yourself ☺️

Making the Most of Our Future

- Computing is resurfacing our world
  - Now almost everyone has access to everything, always
  - New technology affects privacy, jobs, safety, beliefs, etc.

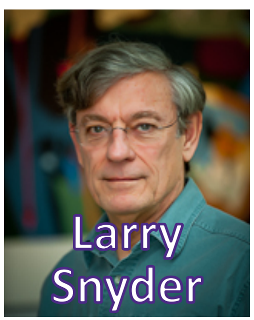
- You now know the most important parts of how it all works!
  - Can bring computing to new fields/jobs/areas
  - Keep these considerations in mind as you use and/or build things

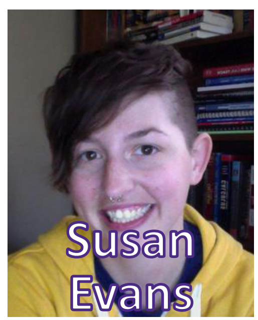
Thanks for a great quarter!

- Huge thanks to your awesome TAs!



- Thanks to course content creators:





- Best of luck in the future! I am happy to chat more if you have questions about CSE, college, etc.