Algorithms Continued

So you've been challenged to come up with an algorithm!

Where do we start?

What kind of language do we use?

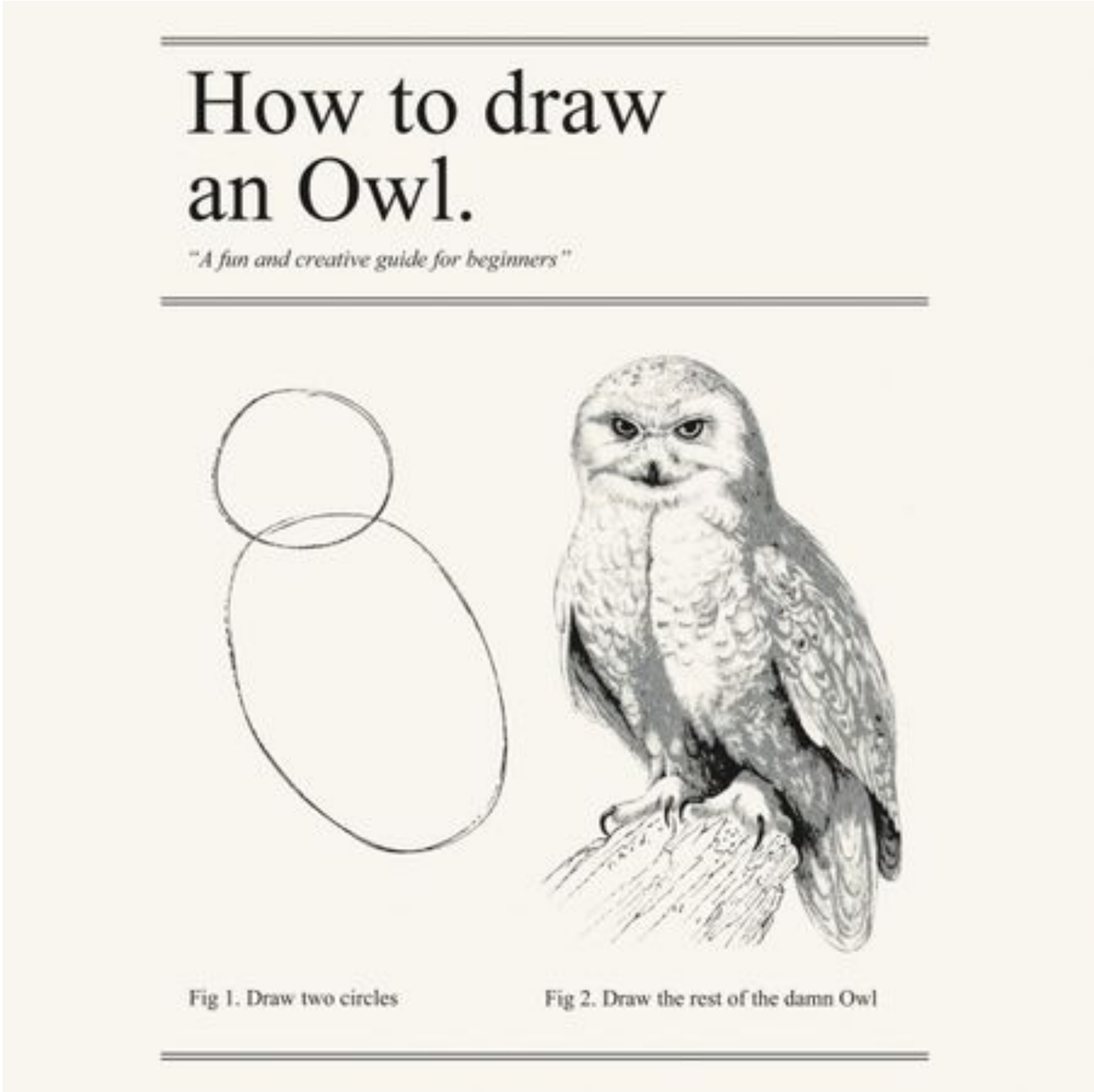
How detailed do we need to be?

Algorithm Detail

The more details, the better!

New developers/algorithm-creators don't specify enough detail.

Typical new SDG student generates about a **third** to a **fourth** the amount of detail required in an algorithm.



So, how do we learn and practice?

Use precise language

Break problems into smaller pieces

Learn common problem-solving approaches

Practice, practice, practice ... and more practice.

It takes **time**

Precise language

- We will be using programming languages like `C#` and `SQL` and `TypeScript` to write our algorithms.
- For now, we will be using **human** language, **English** specifically, which is an imprecise language.

Use your imagination

- When creating algorithms, pretend that you are giving instructions to:
  - A baby that just learned to speak
  - A robot that only knows a few specific words or phrases
  - The pickiest, pedantic person you know.
  - ... then **double** or **triple** the specificity and detail

Break a problem down

Zeno's Dichotomy Paradox

Who knows it?

Zeno's Dichotomy Paradox

That which is in locomotion must arrive
at the halfway stage before it arrives
at the goal.

— as recounted by Aristotle, Physics VI:9, 239b10

Mathematically, you do arrive at your goal

Curious students can see me after class to see how

Take a significant problem and break it into (two?) smaller problems

We might know this as "delegation"

We may not know how to solve an entire problem.

We can look at parts that we *can* solve

Example: Spaghetti and Sauce

Major Steps

- Make Sauce
- Make Spaghetti

Break this down

- Sauce:
  - Garlic, Onion, Basil, Tomatoes
- Spaghetti
  - Eggs, flour, semolina, salt
  - Make dough
  - Rise
  - Roll
  - Cut

Even this is too simplistic

Anyone who cooks knows I've summarized many of these steps.

Breaking down problems into manageable steps is an art

Transform a problem

Take a problem in an area we don't know how to solve and turn it into one that we do!

Here is a simple to state problem.
See if you can solve it (no computer usage)
I'll give you thirty seconds

Ready, set, go ...

Add up the first 1000 numbers

```
1 + 2 + 3 + ... + 998 + 999 + 1000
```

Did you do it?

Transforming the problem

$$1 + 2 + 3 + \ldots + 998 + 999 + 1000$$

Transforming the problem

```
1 + 2 + 3 + ... + 998 + 999 + 1000
```

1	1000
2	999
3	998

Transforming the problem

$$1 + 2 + 3 + \ldots + 998 + 999 + 1000$$

1	+	1000
2	+	999
3	+	998

Transforming the problem

1 + 2 + 3 + ... + 998 + 999 + 1000

1	+	1000	= 1001
2	+	999	= 1001
3	+	998	= 1001

Transforming the problem

1 + 2 + 3 + ... + 998 + 999 + 1000

There are 500 of these pairs.

500 * 1001 = 500,500

1	+	1000	= 1001
2	+	999	= 1001
3	+	998	= 1001

General formula

Breaking problem down: Mail

As another example ... Donald Knuth gives the method a post office typically uses to route mail: letters are sorted into separate bags for different geographical areas, each of these bags is sorted into batches for smaller sub-regions, and so on until they are delivered.

Breaking problem down, one bit at a time.

Find the largest number in a list.

Reduce the problem by ONE

The largest number in a list is either:

- The first number in a list
- **or**
- If other numbers are remaining in the list ... the largest number is in the remainder of the list

Example:

Find largest number in:

`43,71,75,66,48`

Example

- Largest number of $43, 71, 75, 66, 48$ is either 43
- **or**
- Largest number in $71, 75, 66, 48$

Example

- Largest number of $71, 75, 66, 48$ is either 71
- **or**
- Largest number in $75, 66, 48$

Example

- Largest number of $75, 66, 48$ is either 75
- **or**
- Largest number in $66, 48$

Example

- Largest number of $66, 48$ is either 66
- **or**
- Largest number in 48

Example

- Largest number of 48 is 48

- So largest number of $66, 48$ is 68

- So largest number of $75, 66, 48$ is 75

- So largest number of $71, 75, 66, 48$ is 75

- So largest number of $43, 71, 75, 66, 48$ is 75

Example

- Median number in a list
  - First sort
  - Then find middle number
  - If there are an odd count of numbers, take the middle
  - If there are even count of numbers, everage the two middle numbers.

Example

43,71,75,66,48

sorted

43,48,66,71,75

Middle is 66

Example

4,56,97,25,21,2

sorted

2,4,21,25,56,97

Middle is 21 and 25, so median is: 23

0, 1, few,

0, 1, few,

- Solving a problem with *zero* inputs is typically easy

0, 1, few,

- Solving a problem with *zero* inputs is typically easy
- Solving a problem with *one* input is typically easy

0, 1, few,

- Solving a problem with *zero* inputs is typically easy
- Solving a problem with *one* input is typically easy
- Solving a problem with a *few* inputs is typically easy due to our natural intuition

0, 1, few,

- Solving a problem with *zero* inputs is typically easy
- Solving a problem with *one* input is typically easy
- Solving a problem with a *few* inputs is typically easy due to our natural intuition
- Solving a problem with a *huge* amount of input challenges our intuition and assumptions

0, 1, few,

- Solving a problem with *zero* inputs is typically easy
- Solving a problem with *one* input is typically easy
- Solving a problem with a *few* inputs is typically easy due to our natural intuition
- Solving a problem with a *huge* amount of input challenges our intuition and assumptions
- See an example of adding up the first N numbers.

Use "0, 1, few, " in your thinking process

- Perhaps start with an input size of "a few."
- Use your intuition to understand the problem
- Don't forget about `0` and `1` inputs
  - (sometimes the problem has a curveball here)
- See if your intuition still works with

Idea of conditions and loops in algorithms

- When thinking of an algorithm, it often helps to solve a smaller version of the problem
- Then, see if applying that process to each element of the input works.
- This is "looping."
- We'll work with looping **often** in this class

Idea of conditions and loops in algorithms

"If something is true" or *"If something is false"* help determine if or when we should do some smaller steps

Loop with counting

- "Do the following X times"
- "Do the following until X is less-than/more-than/equal to Y"

Computers are *very* good at conditions and loops.

Computers are primarily machines that work with conditions and loops.