Algorithms, not Al Gore's Rhythm

About Me


0 1 0 1 0 1 0
SUNCOAST
DEVELOPERS GUILD


Express Lane
ABOUT 12 ITEMS
HAVE CHECKS READY
9
9
Express Lane
10
10
Express Lane
11
12
12
13
14
15
INFORMATION
Family
Fred Meye
JEWELERS
REPAIR
PROTECT

retail

telephony


space


ADD A FLAVOR
TO YOUR DRINK
WITH
FlavorRage
CHERRY
OR
VANILLA
DISCOUNT COMBOS
1 LARGE POPCORN
2 LARGE DRINKS
PLUS CHOICE OF SELECT CANDY
1 MEDIUM POPCORN
2 MEDIUM DRINKS
PLUS CHOICE OF SELECT CANDY
1 MEDIUM POPCORN
1 MEDIUM DRINK
PLUS CHOICE OF SELECT CANDY
LARGE NACHOS
HOT DOG
MOVIE MEAL
LARGE POPCORN
POPCORN
CANDY
DRINKS
SNACKS
rave cinemas
POP CORN

signage

Algorithms

Like any good presentation, let us give the boring definition.

Definition (1/3)

Self-contained step-by-step set of operations to be performed.

Definition (2/3)

Algorithms perform calculations,
data processing, and/or
automated reasoning tasks.

Definition (3/3)

An algorithm is an effective method that can be expressed within a finite amount of space and time and in a well-defined formal language for calculating a function.


COMEDY
CENTRAL

We use them in life all the time

Examples

from **Tampa International Airport, 4100 George J Bea...**
to **The Iron Yard - Tampa/St. Petersburg, 260 1st Aven...**

25 min (21.1 miles)

via I-275 S

23 min without traffic

Tampa International Airport
4100 George J Bean Parkway, Tampa, FL 33607

- Take George J Bean Outbound Pkwy/George J Bean Pkwy to I-275 S

  7 min (3.6 mi)

- Follow I-275 S to 4th Ave N in Saint Petersburg

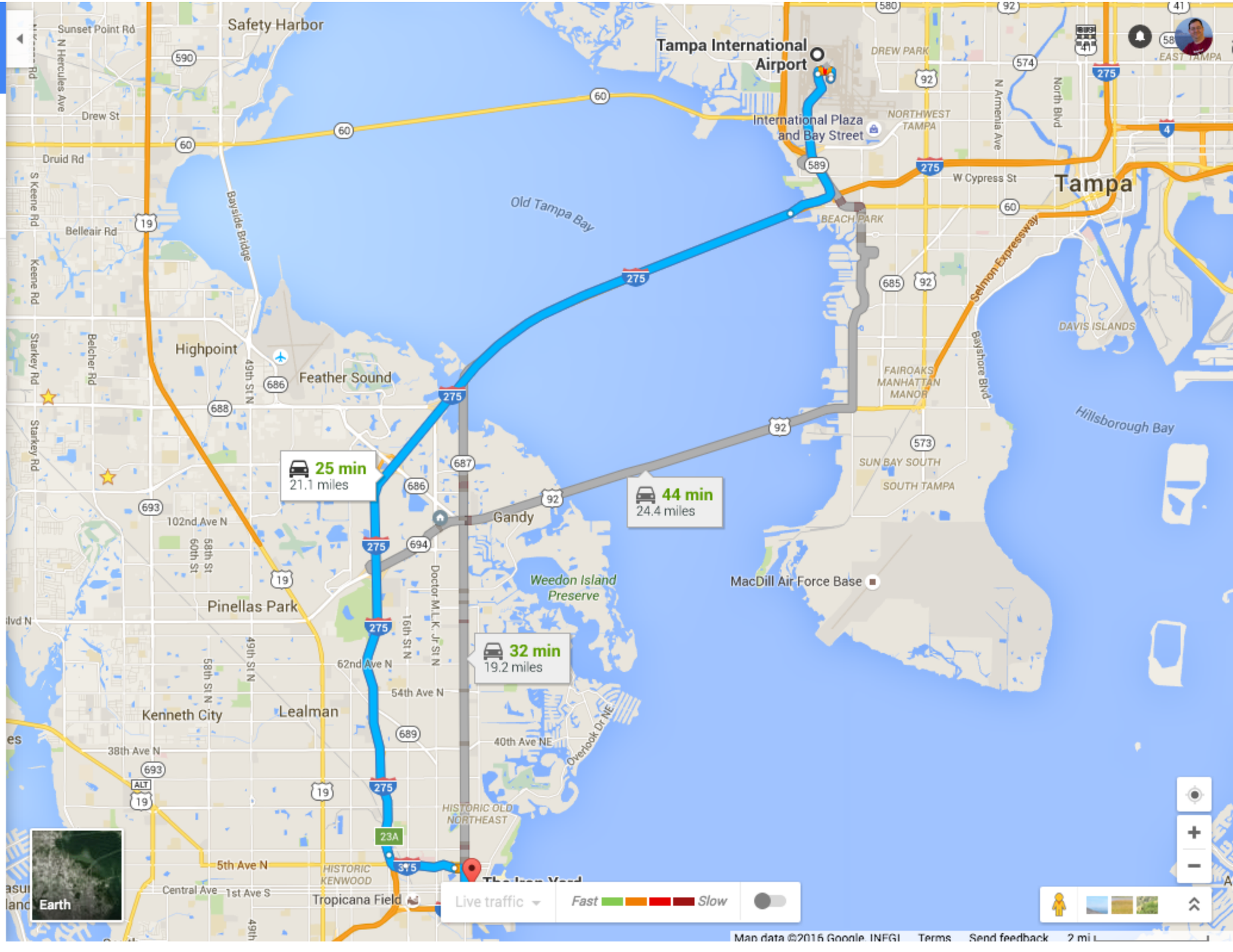
  16 min (16.9 mi)

- Continue on 4th Ave N. Drive to 1st Ave S

  3 min (0.7 mi)

The Iron Yard - Tampa/St. Petersburg
260 1st Avenue South #300, Saint Petersburg, FL 33701

These directions are for planning purposes only. You may find that construction projects, traffic, weather, or other events may cause conditions to differ from the map results, and you should plan your route accordingly. You must obey all signs or notices regarding your route.

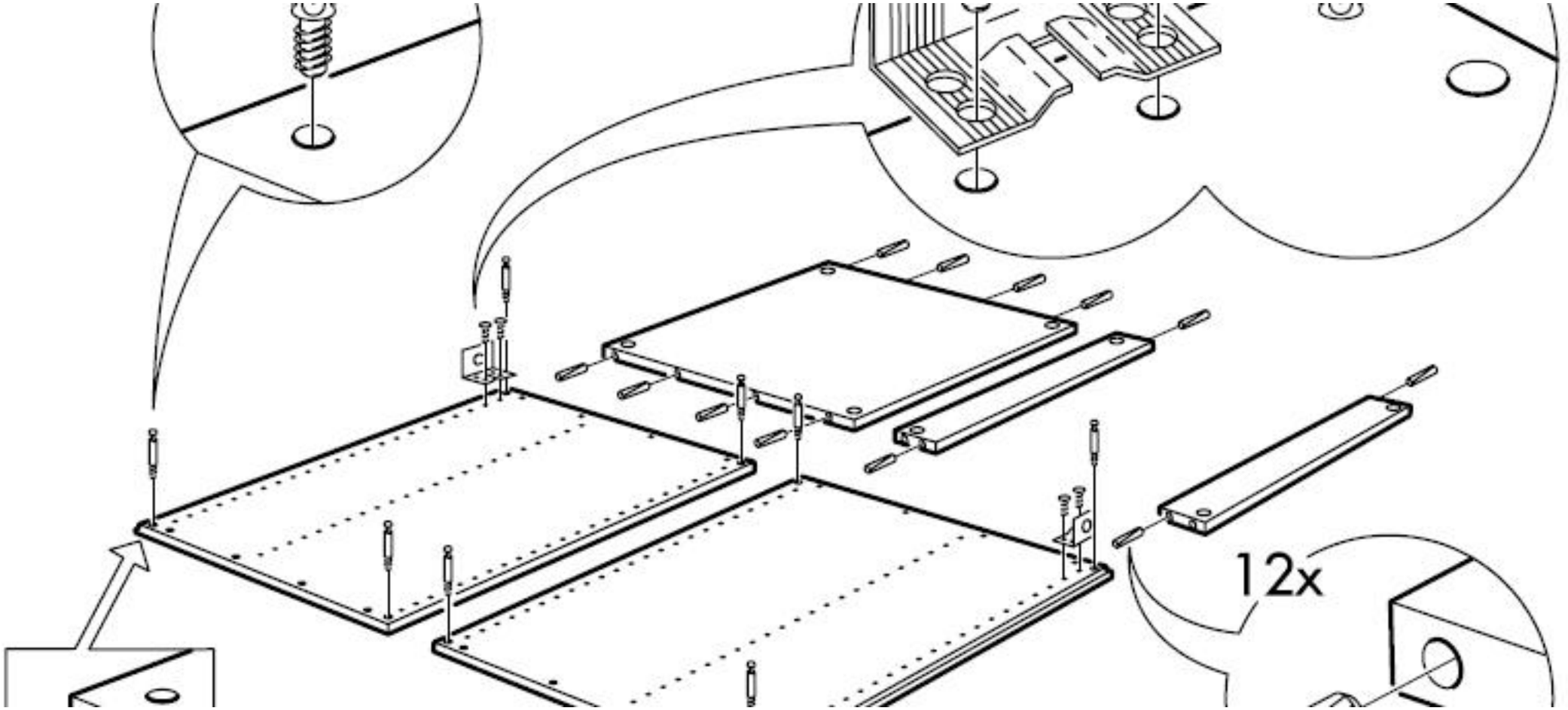


http://bit.ly/1tsqopl

½ lb butter or margarine
1 cup icing sugar
1 cup cornflour
2 cups plain flour

Rub icing sugar into butter. Rub in cornflour then the 2 cups of flour. Mix thoroughly. Grease flat tin, dust with flour. Roll out shortbread on the tin then fork mark all over. Put in middle shelf of pre-heated oven 350 F for ½ hour, then ¼ hour on top shelf or until brown.

http://bit.ly/1Pp7bZY



How to draw an Owl.

"A fun and creative guide for beginners"

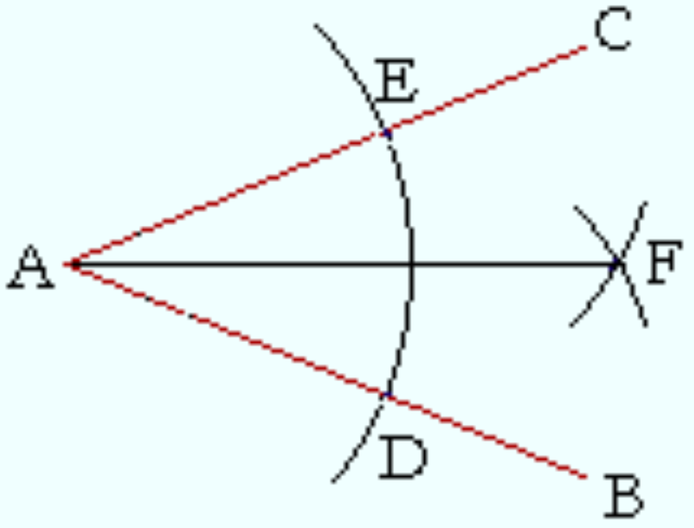
Fig 1. Draw two circles

Fig 2. Draw the rest of the damn Owl

More formal

Bisecting an angle

Here is how to bisect the angle BAC:



Place the point of the compass on A, and swing an arc ED.

Then, with D as center and DE as radius, draw an arc.

Keeping the same radius and with E as center, draw an arc that will intersect the first; call that point of intersection F, and draw AF.

Then AF bisects angle BAC.

Euclid's Algorithm for Computing the Greatest Common Divisor (GCD)

English: Given two numbers, A and B, what is the largest number that evenly divides **both** numbers

```
while (A != B) {

 if (A > B)
   A = A - B
 else
   B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 210, B = 45

```
while (A != B) {

 if (A > B)
   A = A - B
 else
   B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 210, B = 45

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 210, B = 45

A > B (210 > 45)

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

```
A = 210, B = 45
 A > B (210 > 45)
 A = 210 - 45
 A = 165
```

```
while (A != B) {

  if (A > B)
     A = A - B
  else
     B = B - A

}

return A
```

Euclid's Algorithm for GCD

```
A = 210, B = 45
 A > B (210 > 45)
 A = 210 - 45
 A = 165
```

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 165, B = 45

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 165, B = 45

A > B (165 > 45)

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

```
A = 165, B = 45
 A > B (165 > 45)
 A = 165 - 45
 A = 120
```

```
while (A != B) {

  if (A > B)
     A = A - B
  else
     B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 120, B = 45

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 120, B = 45

A > B (120 > 45)

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

```
A = 120, B = 45
 A > B (120 > 45)
 A = 120 - 45
 A = 75
```

```
while (A != B) {

  if (A > B)
     A = A - B
  else
     B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 75, B = 45

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 75, B = 45

A > B (75 > 45)

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

```
A = 75, B = 45
 A > B (75 > 45)
 A = 75 - 45
 A = 30
```

```
while (A != B) {

  if (A > B)
     A = A - B
  else
     B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 30, B = 45

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 30, B = 45

else ... since it isn't true that (30 > 45)

B = 45 - 30

B = 15

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

```
A = 30, B = 45
 else ... since it isn't
true that (30 > 45)
 B = 45 - 30
 B = 15
```

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 30, B = 15

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 30, B = 15

A > B (30 > 15)

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

```
A = 30, B = 15
 A > B (30 > 15)
 A = 30 - 15
 A = 15
```

```
while (A != B) {

  if (A > B)
     A = A - B
  else
     B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 15, B = 15

```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Euclid's Algorithm for GCD

A = 15, B = 15

done with *while*

since A == B

GCD = 15

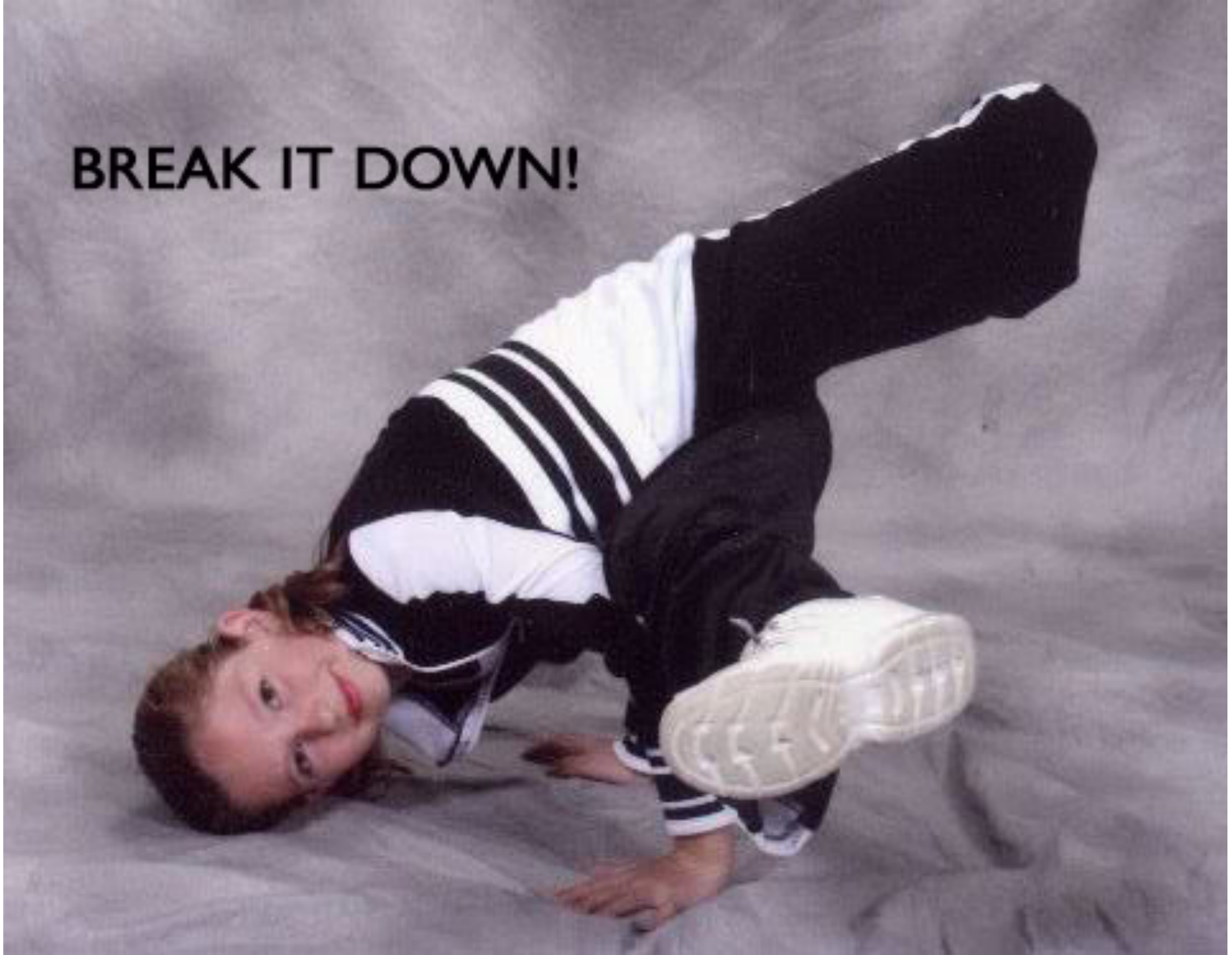
```
while (A != B) {

  if (A > B)
    A = A - B
  else
    B = B - A

}

return A
```

Must be precise and complete

"Make a PB&J Sandwich."

Example video of a family practicing this.

This is the 🔑 to mastering programming


BREAK IT DOWN!

1. Read the problem completely twice.
2. Solve the problem manually with several sets of sample data.
3. Optimize the manual steps.
4. Write the manual steps as comments or pseudo-code.
5. Replace the comments or pseudo-code with real code.
6. Optimize the real code.

Read the problem completely twice.

- Most important!
- Can you explain it (simply) to someone else?

Solve the problem manually

- Programming is automation
- Solve the problem manually
- Maybe even on pen and paper
- Or use physical objects
- Practice!

Optimize the manual solution

- Can you remove any steps?

Write pseudo-code or comments

- Open an editor and write the manual steps in English

Replace comments with real code

- Replace each individual step with code

PEDAC

- Created by Launch School
- Generalized process for creating algorithms:
  - **P** roblem
  - **E** xamples
  - **D** ata (structures)
  - **A** lgorithm
  - **C** ode

Example

Reverse a string (word)

Problem

Given a word, which is just a sequence of letters, make a new word with the same sequence of letters in reverse order.

Examples

zebra	arbez
word	dwow
rotator	rotator

Data (structures)

- `string`
- loops

A lgorithm ...

Start by using a specific example...

1. Write “Zebra” down.
2. Create a new empty word.
2. Start at the last letter in the word (the "a" from Zebra)
3. Put the current letter at the end of the new word
4. If there is a previous letter,
   make the previous letter the current letter
   letter and start back at 3.
5. When there are no more letters in the word, our
   new word is the answer.

Pseudo code

```
1. NewWord = ""
2. Loop backwards through word to reverse
3.   NewWord += CurrentLetter
4. Return NewWord
```

Code (Ruby)

```ruby
word = "Zebra"

new_word = ""
word.chars.reverse_each do |letter|
  new_word += letter
end

new_word # => "arbeZ"
```

Sorting!

Simplest Sorting EVAR

Bubble sort

For each two adjacent elements

Exchange them if out of order

Repeat until the array is sorted.

Break it down

1. Assume the array is sorted

2. Go through each pair of elements
   - If the first of the pair is larger than the second of the pair
     - Swap the two elements
     - Remember that the array isn't sorted

3. When done with all the elements, if we
   still believe the array is sorted, STOP

4. Otherwise, go back to step 1

Write out a psuedo-code algorithm

- Paste that in your code editor

```
# Assume the list is sorted
# Go through the indexes of the array in pairs
#   If the first is larger, swap, remember array isn't sorted
# If array is sorted, stop
# go back to the first step
```

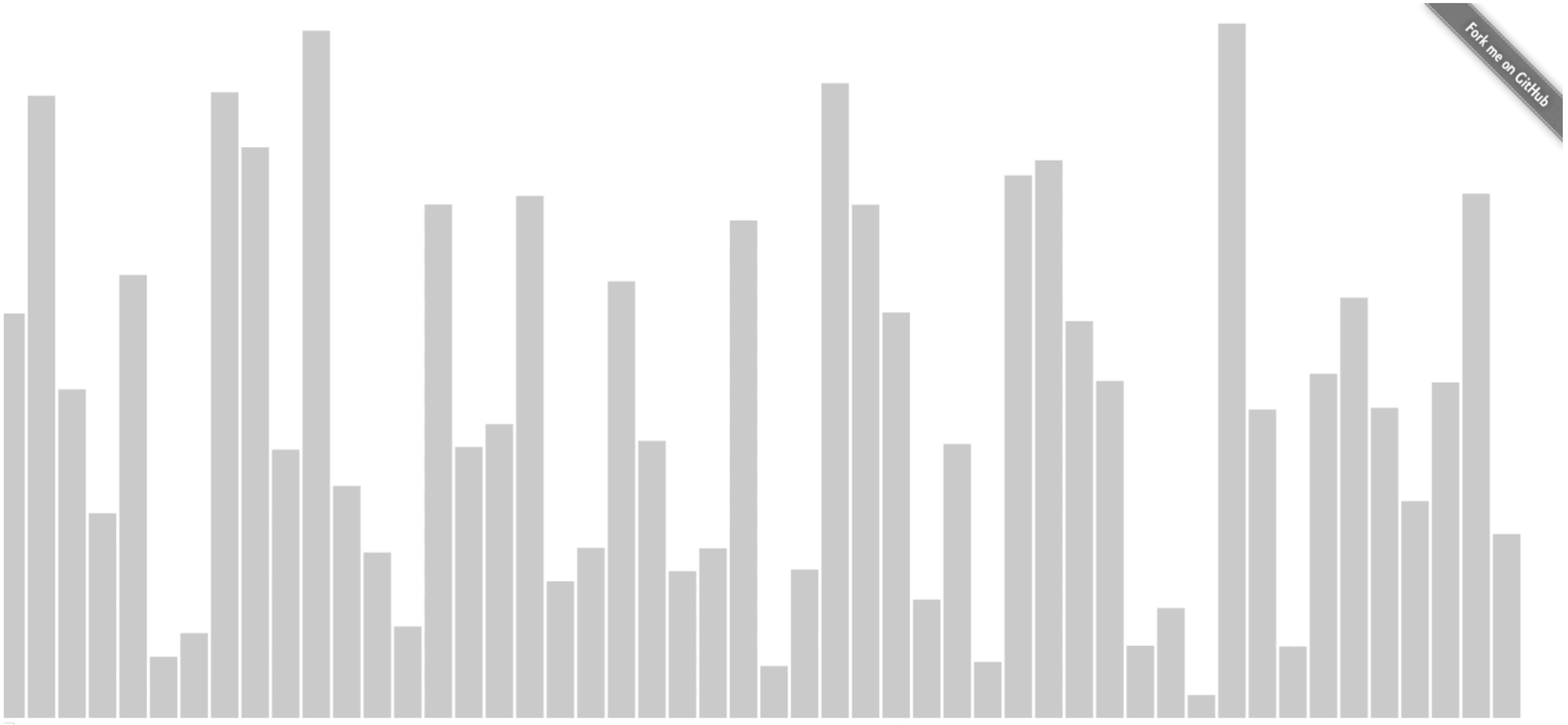
Fill in the code around it

If you can't *directly* translate a statement to code then either:

- You need to break the statement into smaller pieces and try again
- or ... you **DO** have a small enough statement, but don't know the language syntax well enough

```ruby
array = [7,1,2,9,4,5]

def bubble_sort(array)
  loop do
    # Assume the list is sorted
    sorted = true

    # Go through the indexes of the array in pairs
    (0...array.length).each_cons(2) do |first, second|

      # if the first is larger
      if array[first] > array[second]
        # Swap
        array[first], array[second] = array[second], array[first]

        # Remember array isn't sorted
        sorted = false
      end

    end

    # If array is sorted, stop
    break if sorted

    # end of loop/do go back to the first step
  end
end
```

```ruby
array = [7,1,2,9,4,5]

def bubble_sort(array)
  loop do
    # Assume the list is sorted
    sorted = true

    # Go through the indexes of the array in pairs
    (0...array.length).each_cons(2) do |first, second|

      # if the first is larger
      if array[first] > array[second]
        # Swap
        array[first], array[second] = array[second], array[first]

        # Remember array isn't sorted
        sorted = false
      end

    end

    # If array is sorted, stop
    break if sorted

    # end of loop/do go back to the first step
  end
end
```



- https://github.com/AIRTucha/SortVis


A moment of zen

Algorithm Complexity

Measuring Time and Space

Measure

time	the number of operations required
space	the amount of memory required

Best case	If the data is perfect for *this* algorithm, how fast are we?
Worst case	If the data is terrible for *this* algorithm, how slow are we?
Average case	Considering all possible inputs, what is the average speed of the algorithm

What are we measuring?

When we speak of algorithm complexity, what are we measuring?

- Operations
- Comparisons
- Increments

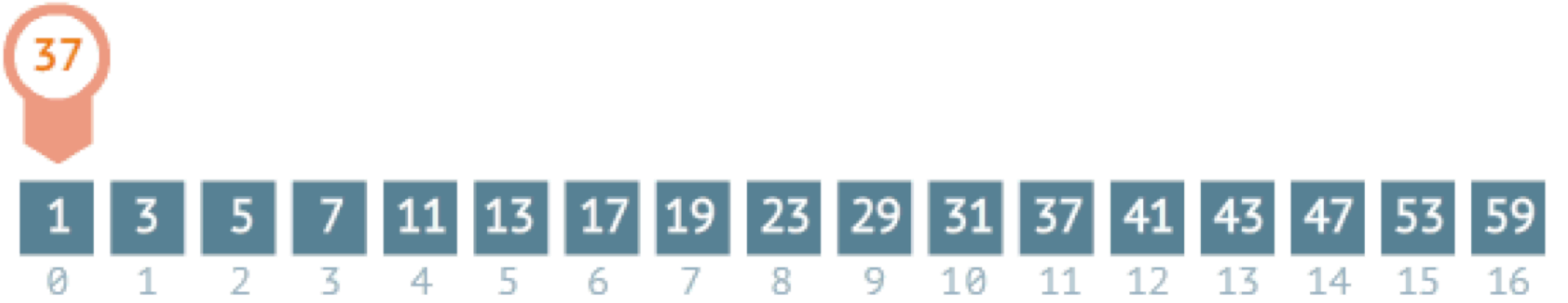
Example: Searching

Binary search

steps: 0

Sequential search

steps: 0



Big O notation

```
O(1)
O(log n)
O(n)
O(n log n)
O(n^2)
O(2^n)
```

- These grow at a different rate based on how n changes

O(1)

Takes a constant amount of time regardless of input size

Example: looking up an index in an array
looking up a key in a hash/dictionary (most cases)

O(n)

If n doubles, the algorithm takes twice as long

Example: linear search animation

O(n^2)

If n doubles, the algorithm takes FOUR times as long

Example: bubble sort!

O(2^n)

If n doubles, the algorithm takes many times as long

e.g., if n grows from 20 to 40, O(2^n) grows by over a MILLION times

Example: Tower of Hanoi

Tower of Hanoi

The Tower of Hanoi is a mathematical game or puzzle.

It consists of three rods and several disks
of different sizes, which can slide onto any rod.

The puzzle starts with the disks in a neat stack
in ascending order of size on one rod, the smallest
at the top, thus making a conical shape.

Tower of Hanoi

The objective of the puzzle is to move the entire

stack to another rod, obeying simple rules

Tower of Hanoi

1. Only one disk can be moved at a time.

2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

3. No disk may be placed on top of a smaller disk.

Tower of Hanoi

$$2^n$$

To understand recursion you must first understand recursion

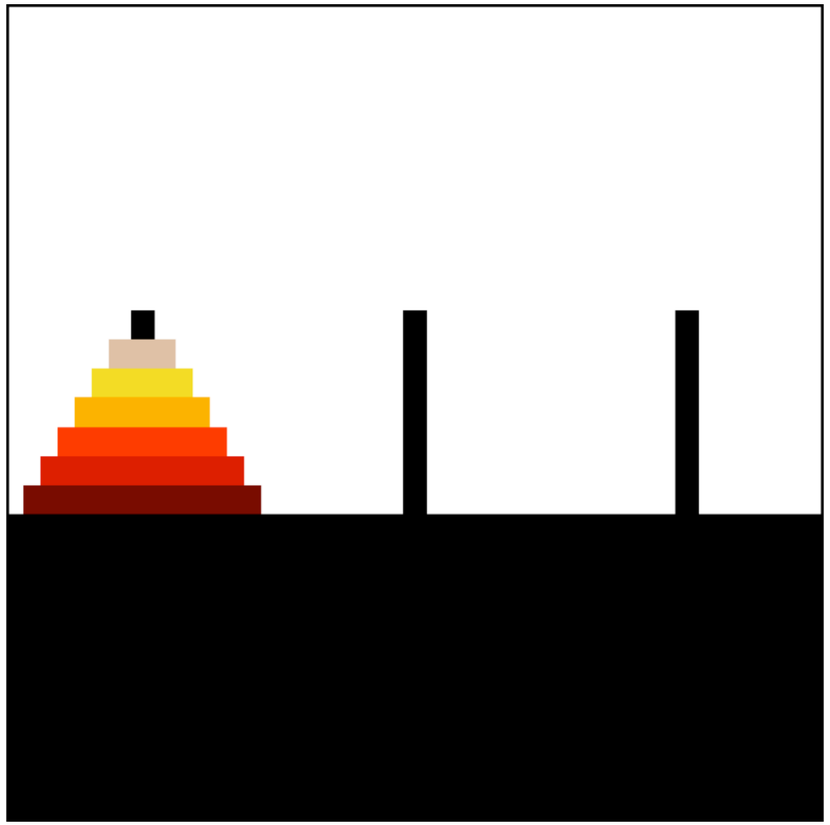
Tower of Hanoi Recursively

Move all but the bottom disk to the "spare" peg (using this exact algorithm)

Move the bottom disk to the "destination" peg (this is a simple move)

Move all the disks from the "spare" peg to the "destination" peg (using this exact algorithm)

Tower of Hanoi



Tower of Hanoi

If we could move a disk per second how long would this take?

Disks	Time
1	**a second**
8	**4 minutes**
12	**about 1 hour**
17	**1 day and a half**

Disks	Time	
21	**24 days**	About a month

Disks	Time	
25	**388 days**	About a year

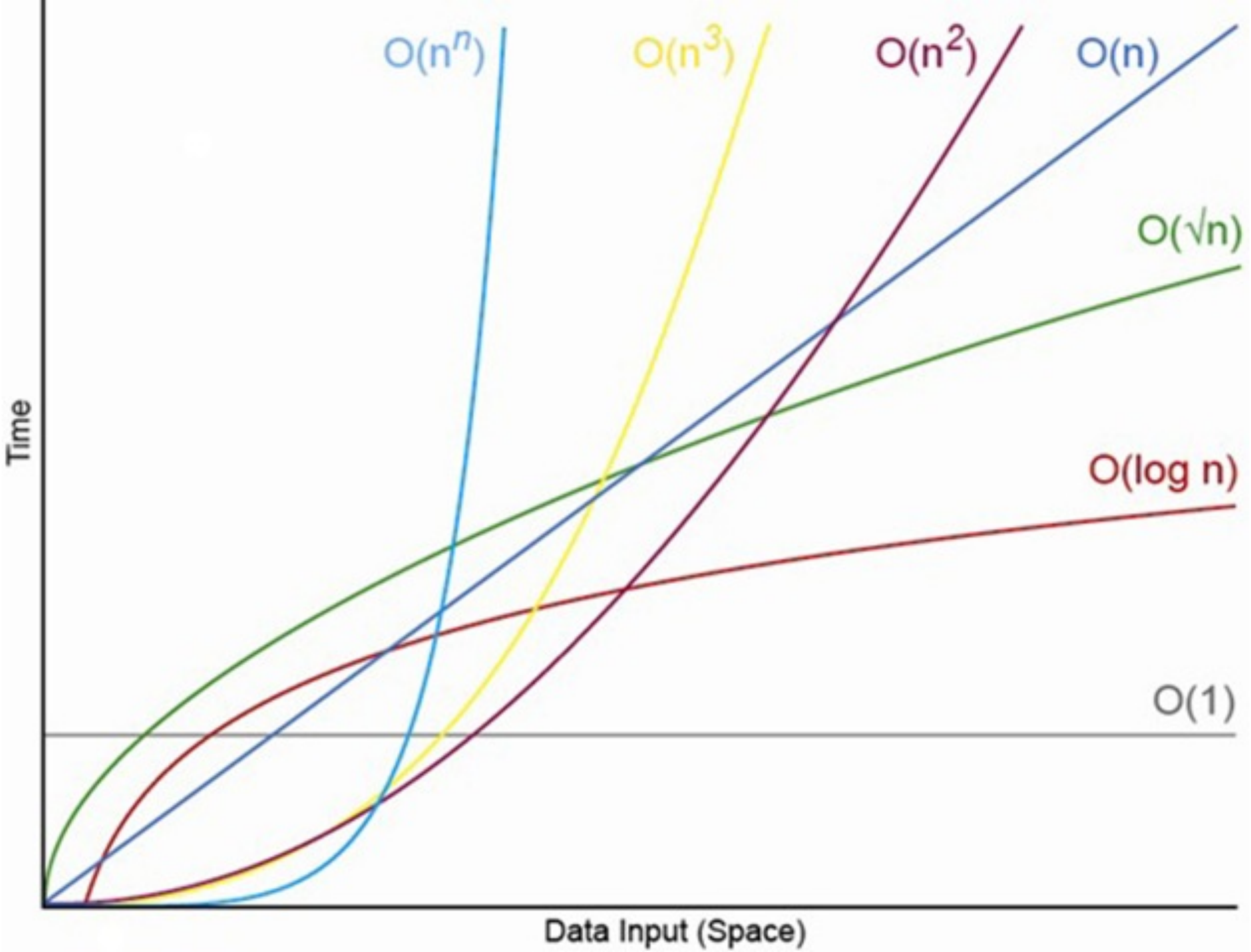
Disks	Time	
31	**about 68 years**	About an (average) lifetime

Disks	Time	
35	… **over 1,089 years**	About a millennium

Disks	Time
50	**35.7 million years** 😱

Disks	Time	
59	**More than 13 billion years**	Estimated age of the universe

Complexity


O(n^n)
O(n^3)
O(n^2)
O(n)
O(√n)
O(log n)
O(1)
Time
Data Input (Space)

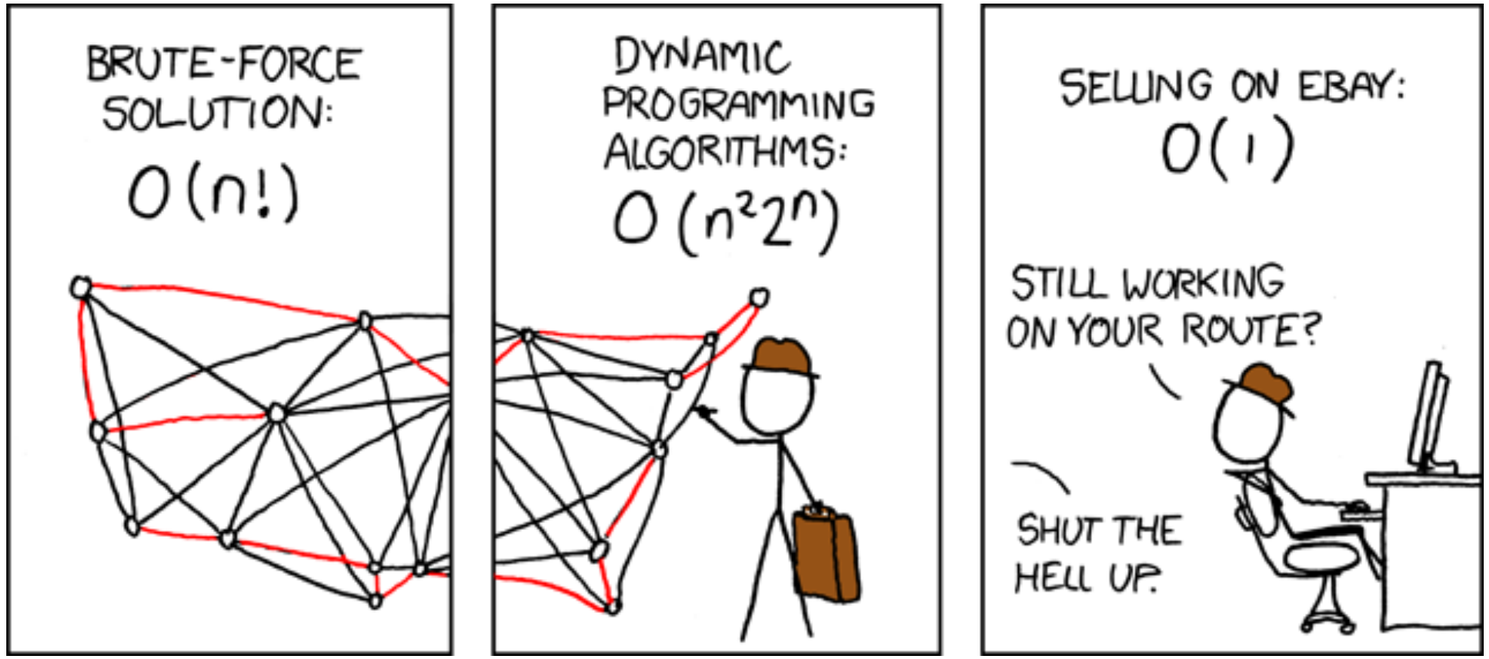
Salesperson


Messina
Palermo
Trapani
Enna
Catania
Caltanissetta
Agrigento
Siracusa
Ragusa

Could you find it by hand?

Probably, for a small number of cities!

Complexity



https://www.explainxkcd.com/wiki/index.php/399:_TravellingSalesmanProblem

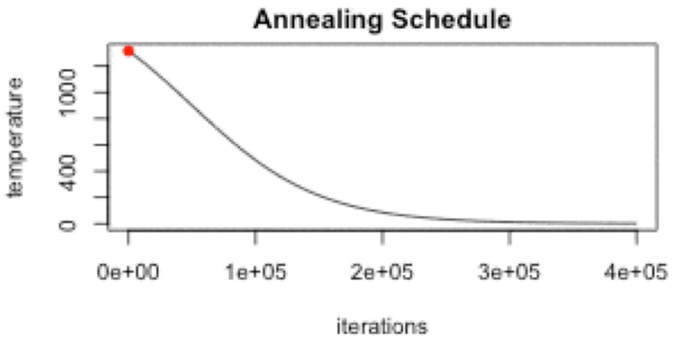
1 Billion Computations per second

Cities	Time
2	less than a minute
3	less than a minute
4	less than a minute
14	1 minute
16	about 6 hours

1 Billion Computations per second

Cities	Time
18	2 months
19	almost 4 years
21	about 1,620 years

Distance: 43,499 miles
Temperature: 1,316
Iterations: 0



The TSP has several applications

planning

logistics

manufacturing of microchips

DNA sequencing

For More info

http://bit.ly/XUXLXWX

Wrap It Up

Algorithms Matter

Understanding Complexity Matters

Understanding How to Break Problems Down Effectively REALLY Matters

Being a Developer

1. Understanding the problem
2. Break it down into small steps
3. Break it into even smaller steps
4. Translate this to code
5. Appreciate the complexity

How to practice?

exercism.io

codewars.com

CoderNight meetup
(http://meetup.com/CoderNight)

Thank You

Questions?