Foundations of computational thinking

Young-suk Lee

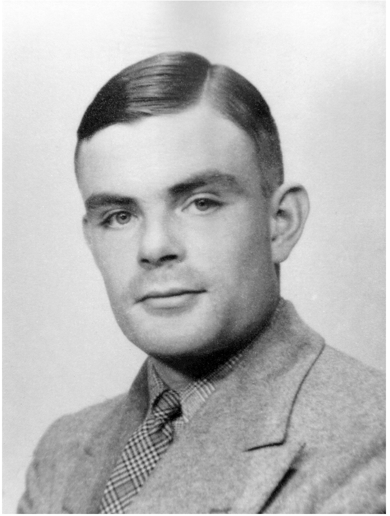
What is computational <u>thinking</u>?

Computer (occupation): The first known written reference dates from 1613

"Computing Machinery and Intelligence" by Alan Turing (1950)

"Human computer" is someone who is "supposed to be following fixed rules; he has no authority to deviate from them in any detail."

“Computing machine” referred to any machine that performed the work of a human computer.

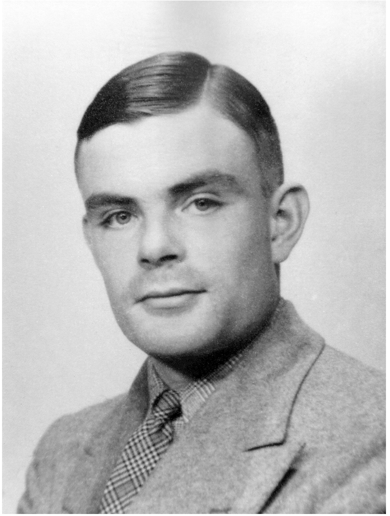


"Computing Machinery and Intelligence" by Alan Turing (1950)

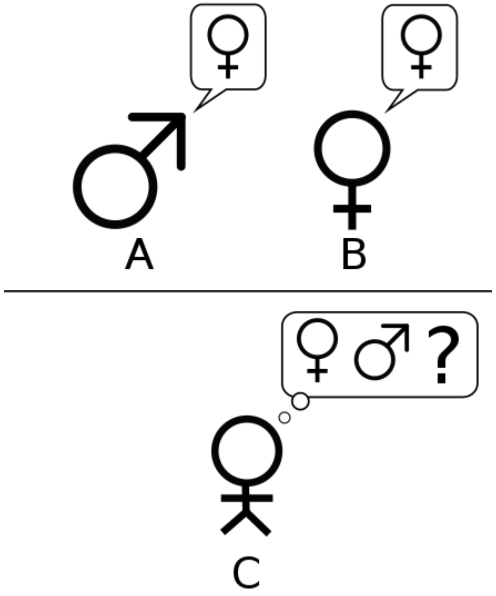
"Human computer" is someone who is "supposed to be following fixed rules; he has no authority to deviate from them in any detail."

“Computing machine” referred to any machine that performed the work of a human computer.

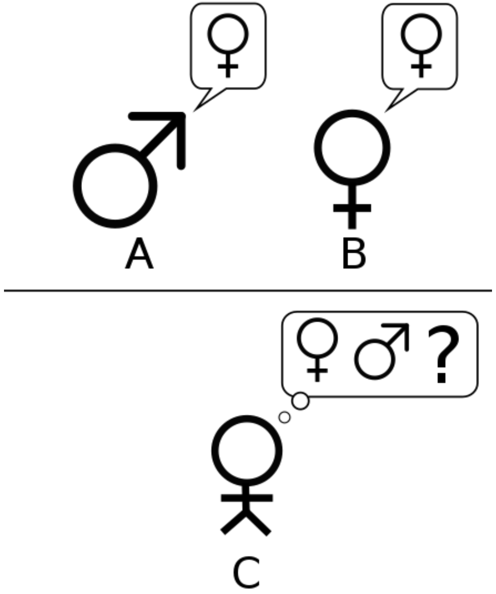
Turing proposed changing the question from whether a machine was intelligent, to "whether or not it is possible for machinery to show intelligent behaviour".

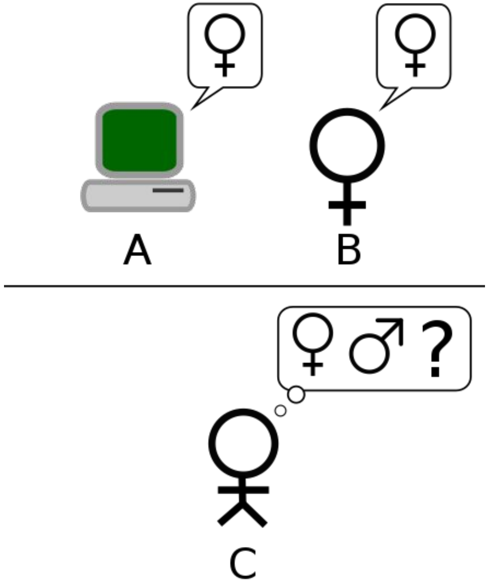


The imitation game

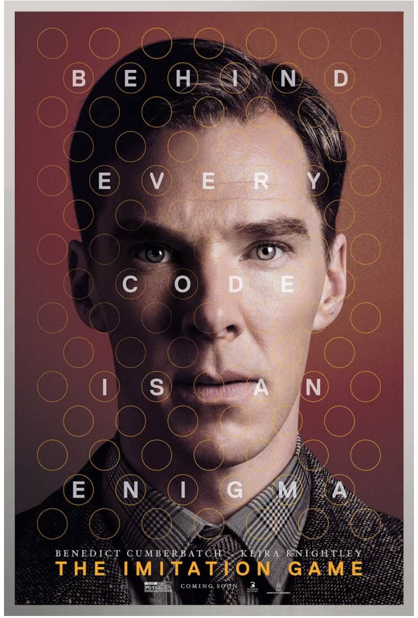


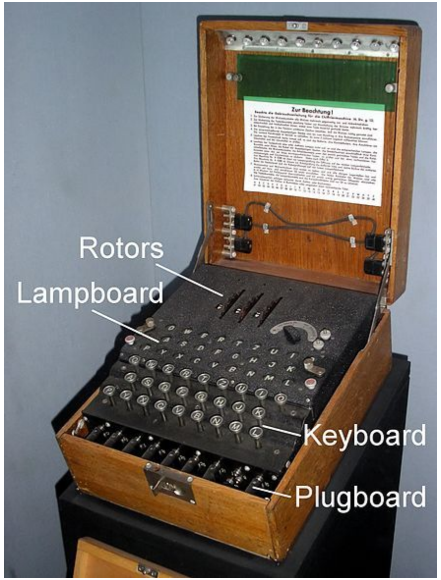
The imitation game





The imitation game (the movie)

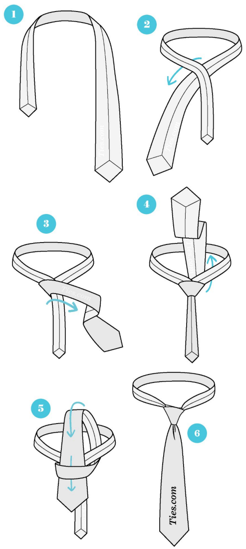


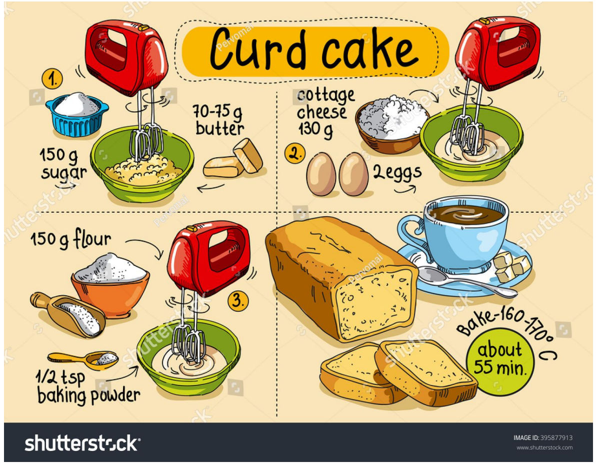


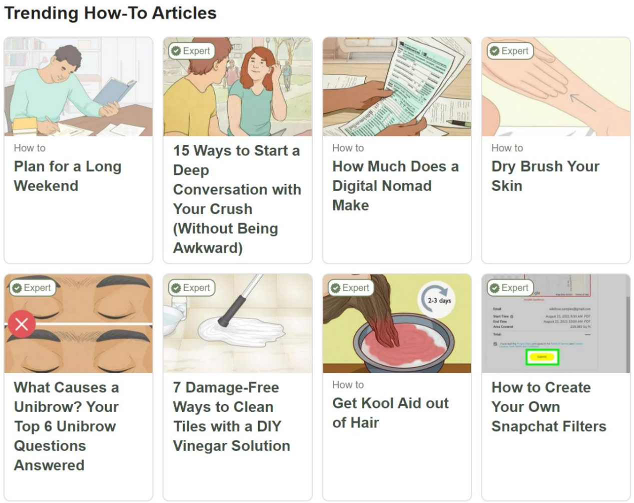
Algorithms and Greatest Common Divisors

What is an algorithm?

An **algorithm** is a sequence of steps used to solve a problem. An algorithm does not need to involve a computer.



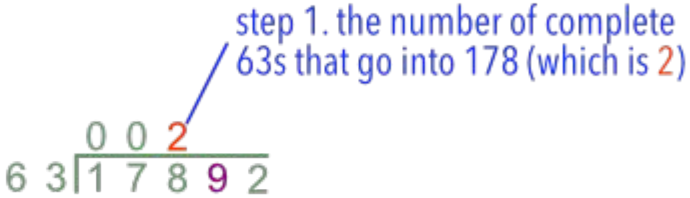




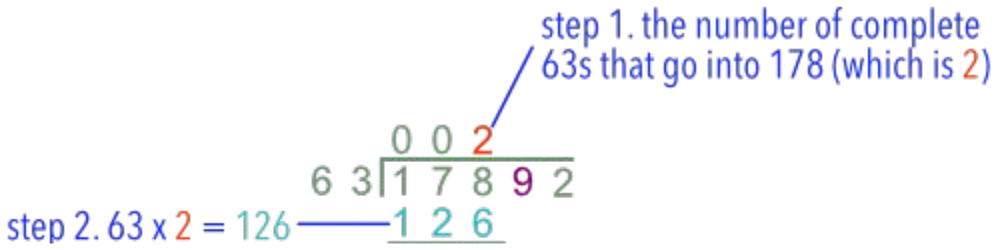
Long division is also an algorithm (4 steps)

$$63\overline{)\,1\;7\;8\;9\;2}$$

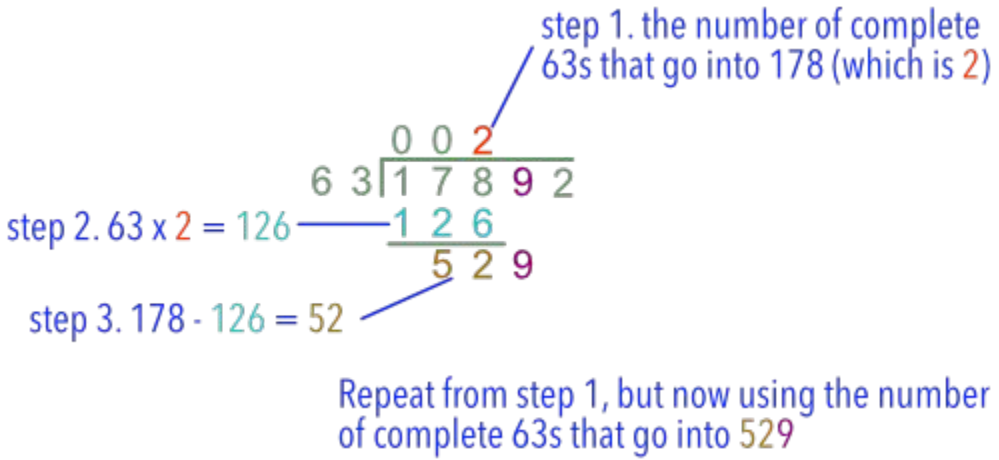
Long division is also an algorithm



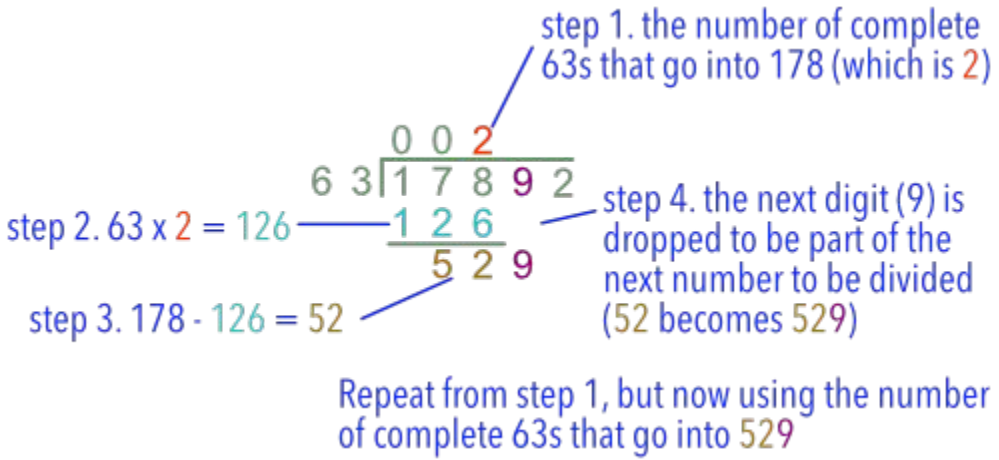
Long division is also an algorithm



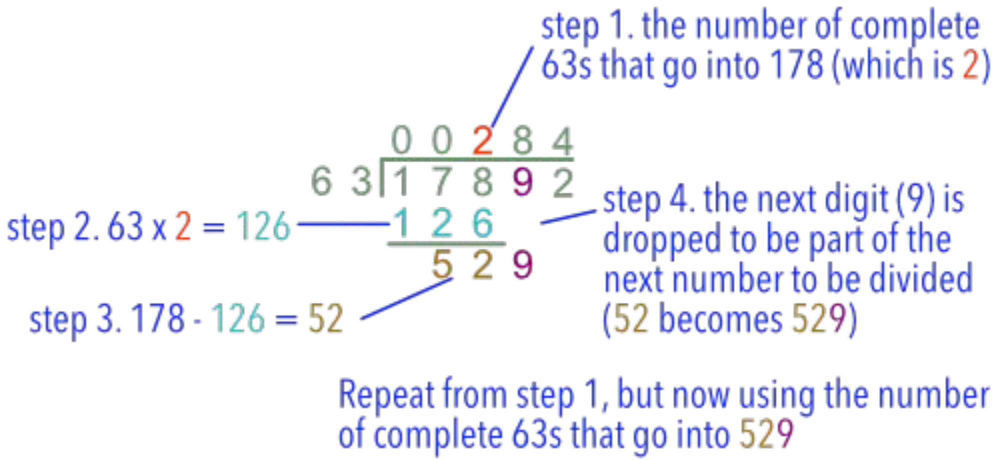
Long division is also an algorithm



Long division is also an algorithm

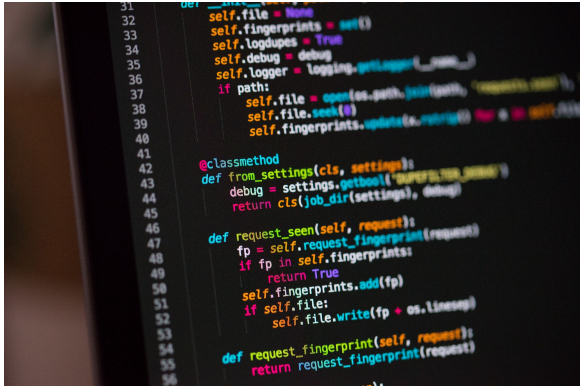


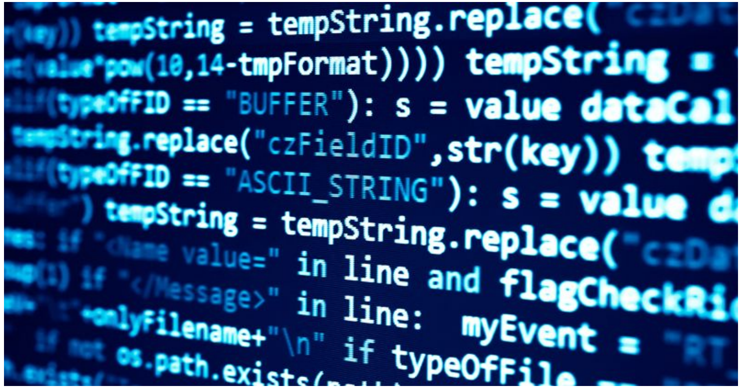
Long division is also an algorithm



What is programming?

Computers can only follow very precise instructions. For this reason, we will need to be exact when we convert our ideas into instructions that a computer can understand; this is what we mean by **programming** the computer.





Greatest Common Divisors

We will specify exactly what we intend the computer to take as input, and exactly what it should return as its output. Here's the **computational problem**:

GCD Problem

Input: Integers a and b.

Output: The greatest common divisor of a and b, denoted GCD(a, b).

Greatest Common Divisors

We will specify exactly what we intend the computer to take as input, and exactly what it should return as its output. Here's the **computational problem**:

variables (or parameters)

GCD Problem

Input: Integers *a* and *b*.

Output: The greatest common divisor of *a* and *b*, denoted GCD(*a*, *b*).

Simple algorithm for computing GCD(*a*, *b*)

1. Set a variable *d* equal to 1. This variable will represent the largest divisor common to *a* and *b* that we have found thus far.

Simple algorithm for computing GCD(*a*, *b*)

1. Set a variable *d* equal to 1. This variable will represent the largest divisor common to *a* and *b* that we have found thus far.
2. For every integer *n* between 1 and the minimum of *a* and *b*, we ask: “Is *n* a divisor of both *a* and *b*?” If “Yes”, then we update the largest identified common divisor *d* to be equal to the current value of *n*.

Simple algorithm for computing GCD(a, b)

1. Set a variable d equal to 1. This variable will represent the largest divisor common to a and b that we have found thus far.
2. For every integer n between 1 and the minimum of a and b, we ask: “Is n a divisor of both a and b?” If “Yes”, then we update the largest identified common divisor d to be equal to the current value of n.
3. After ranging through all these integers, the current value of d must be GCD(a, b).

GCD(378, 273)?

Divisors of 378	1
Divisors of 273	1

GCD(378, 273)?

Divisors of 378	1	2
Divisors of 273	1	

GCD(378, 273)?

Divisors of 378	**1**	2	**3**
Divisors of 273	**1**		**3**

GCD(378, 273) = 21

Divisors of 378	1	2	3	6	7	9	14	18	21	27	42	54	63	126	189	378
Divisors of 273	1		3		7		13		21		39			91		273

GCD(978, 89798763754892653453379597352537489494736)?

The described algorithm is correct but also known as a **trivial** algorithm. Computers have limitations, and the better the algorithms that we provide computers with, the faster they can solve our problems.

Pseudocode and Control Flow

Pseudocode and control flow

Pseudocode is a general (i.e. programming-language agnostic) way of describing algorithms. Here's a **computational problem**:

Minimum of Two Numbers Problem

Input: Numbers *a* and *b*.

Output: The minimum value of *a* and *b*.

Pseudocode and control flow

Pseudocode is a general (i.e. programming language agnostic) way of describing algorithms. Here's a **the pseudocode for this problem**:

Function

Conditional statements

```
min(a, b)
    if a < b
        return a
    else
        return b
```

Pseudocode and control flow

Pseudocode is a general (i.e. programming language agnostic) way of describing algorithms. Here's a **the pseudocode for this problem**:

Function

Conditional statements

```
min(a, b)
    if a < b
        return a
    else
        return b
```

Q. Does this *min* function return the desired answer?

Pseudocode and control flow

Pseudocode is a general (i.e. programming language agnostic) way of describing algorithms. Here's a **the pseudocode for this problem**:

Function

Conditional statements

```
min(a, b)
    if a < b
        return a
    else
        return b
```

Q. Could you write out the pseudocode for returning the minimum value of three numbers (e.g. *a*, *b*, *c*)?

Pseudocode and control flow

Intermediate variables

Loop

```
factorial(n)
  p ← 1
  i ← 1
  while i ≤ n
      p ← p·i
      i ← i+1
  return p
```

Pseudocode and control flow

Loop

```
anotherFactorial(n)
    p ← 1
    for every integer i between 1 and n
        p ← p·i
    return p
```

Q. Write the pseudocode for the trivial GCD algorithm

1. Set a variable *d* equal to 1. This variable will represent the largest divisor common to *a* and *b* that we have found thus far.
2. For every integer *n* between 1 and the minimum of *a* and *b*, we ask: "Is *n* a divisor of both *a* and *b*?" If "Yes", then we update the largest identified common divisor *d* to be equal to the current value of *n*.
3. After ranging through all these integers, the current value of *d* must be GCD(*a*, *b*).

Hint: You probably will need to write pseudocode for the **IntegerDivision** and **Remainder** function.

Next on nontrivial algorithms