

CS187 - Science Gateway Seminar for CS and Math

Fall 2013 – Class 3

Sep. 10, 2013

What is (not) Computer Science?

- Network and system administration?
- Playing video games?
- Learning to use software packages?
- Using and fixing computers?
- Computer programming and code writing? (well, yes and no).

"Computer science is no more about computers than astronomy is about telescopes" (attributed to Edsger Dijkstra, 1970)

What is it Really, Then?

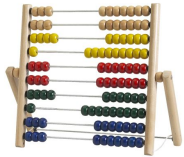
The science of computation – theory and applications.

- Theory – Algorithms, data structures, complexity, computability.
- Programming languages.
- Software engineering and design.
- Hardware design, compilers, operating systems, networks (this is actually **about computers!**).
- Application: Bioinformatics, robotics, computer vision, graphics, databases...

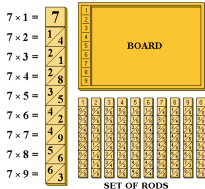
A Brief History

- What we know as modern computer science was formulated in the 1930's.
- First electronic computers were built in the 1940's.
- Often computer science did not exist as an independent department/school until fairly recently (at UMB – only since 2001).
- The term "Computer" was used to describe people until the 1920's!
- Humans used mechanical devices for calculations for thousands of years.

Early History to 17th Century – First Mechanical Devices and Theoretical Developments



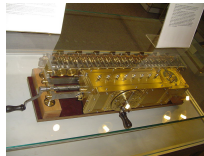
Abacus, 2700–2300 BC – predating written numbers, Middle East and Asia



John Napier (1550–1617), discovery of logarithms, decimal points, "Napier bones" – a calculating instrument



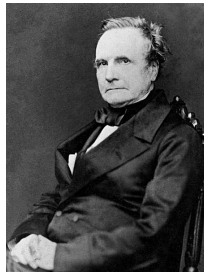
Blaise Pascal (1623–1662), first mechanical adding and subtracting device.



Gottfried Wilhelm Leibniz (1646–1716), binary system, formal binary logic, "stepped reckoner" – first mechanical device capable of performing all 4 arithmetic operations.

19th Century – First Attempt at a "Real" Computer

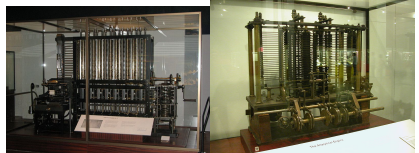
- The "difference engine" – a mechanical computer for tabulating polynomial functions.
- The "analytical engine" – a general-purpose mechanical computer containing a logic unit, conditional branching, loops, and integrated memory.
- Punched cards provided the input.
- Designed, never fully built.



Charles Babbage (1791–1871), invented the first computer.



Ada Lovelace (1815–1852), first computer programmer.



Difference engine and analytical engine

Turing and the Birth of Modern Computer Science

- 1930's – the "Church-Turing thesis" – formalization of the algorithm, the notion of computability.
- Lambda calculus (Church) – a framework for defining functions.
- The Turing machine (Turing) – a theoretical framework for a computer.

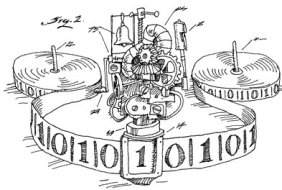


Alonzo Church (1903–1995)



Alan Turing (1912–1954)

The Turing Machine

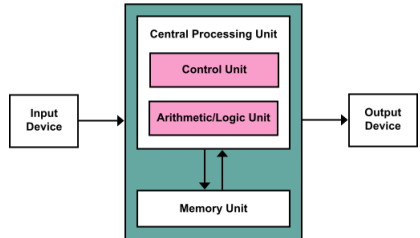
- A (hypothetical) device representing a computing machine.
 - Contains an infinite tape with a (finite) set of symbols from a finite alphabet – the input.
 - A read/write head that can move to the left or right.
 - A (finite) set of *state*, including an initial state and a set of final states.
- 
- A (finite) list of instructions that tells us how to move from one state to another: Given a state q_i and a symbol s_i , move to state q_j , write symbol s_j and move the head to the left or right.

Computability and the Universal Turing Machine

- This is a very simple model... right?
- Yet, it is as powerful as any computing device... even today.
- As a matter of fact, any task can be computed (performed by any computational device following a list of instructions, just like any of today's computer programs) if and only if it can be performed by a Turing machine.
- The Universal Turing machine – A Turing machine that can simulate any Turing machine on any input.
- This is the foundation of modern day computers.

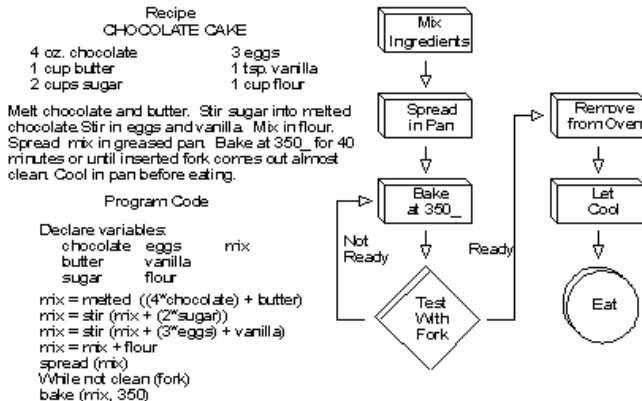
Stored-Program Computers, the Von-Neumann Architecture

- A design that does not require "reprogramming" of a computer for every new task.
- Instead, the programs and instructions can be stored inside the computer.
- Sort of like the Universal Turing Machine...
- Modern computers are still based on this architecture.



What Are Algorithms, Anyway?

A set of instructions/procedures to solve a given problem.



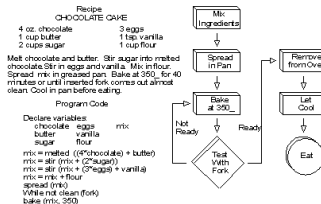
What Are Algorithms, Anyway?

A set of instructions/procedures to solve a given problem.

Input



Algorithm

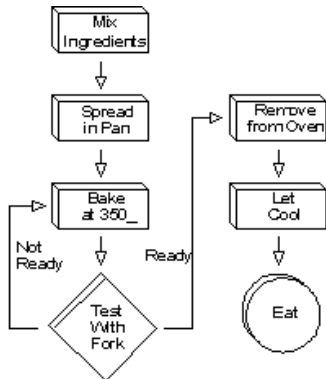


Output



What's in an Algorithm

- Must be unambiguous, solve the problem and terminate.
- One starting point and (one or more) end point(s).
- Input and output (both optional).
- Control flow – we follow the instructions (not necessarily in the order of their appearance) and at any stage we're at some "block".
- Conditional branching – if...then... (else...).
- Loops – repeat some actions for a certain number of times or until some condition is filled.



Smaller Building Blocks to Build a Larger Algorithm

Step 1: Go to the supermarket



Step 2: Buy apples



Step 3: Use in a recipe



step 1 \rightarrow step 2 is common to all recipes. We can write a program called “get_apples” and use it as a **“black-box”** inside all three.

More (CS/Math Relevant) Examples

- 1 Print "Hello, world!" on the screen.
- 2 Given a sequence of integers in no particular order, $(-2, 17, 36, 29, 100, 10)$, find the smallest.
- 3 Given a sequence of integers, sort them from the smallest to the biggest.
- 4 Given a sequence of integers, calculate their sum.

How can you relate (2) and (3)?

Use Smaller Building Blocks to Build a Larger Algorithm

One way to sort a sequence of numbers is using the following algorithm:

Algorithm 1 Sort (list L of N numbers)

- 1: Repeat steps (2-4) N times:
 - 2: $m = \text{Minimum}(L)$
 - 3: $\text{print}(m)$
 - 4: Remove m from L
-

Algorithm 2 Minimum(list L of M numbers)

- 1: $\text{tmpmin} = \text{first element in } L$
 - 2: **for** Each of the remaining elements in L **do**
 - 3: $i = \text{Next element in } L$
 - 4: **if** $i < \text{tmpmin}$ **then**
 - 5: $\text{tmpmin} = i$
 - 6: **end if**
 - 7: **end for**
 - 8: **return** tmpmin
-

Computational Complexity

- Given a problem with an input of size N , how long will it take to compute it as a function of N ?
- Sometimes the choice of the right algorithm can make the difference between hours and seconds.
- Some problems are computable but their computation takes so much time that it is impractical.
- Example: You work for a delivery company and have to visit a number of destinations, each exactly once, and go back to your starting point. What is the shortest way to do it?

Can Anything be Computed?

- In other words – can any problem be formulated as an algorithm?
- No! And it can be shown mathematically.
- As a matter of fact, in every non-trivial axiomatic formulations in mathematics, there are undecidable propositions (statements that are impossible to prove).
- It is called “incompleteness”.
- Example – the halting problem: Given any program and any input, will the program ever stop?
- Turns out you can’t tell, in the general case!

The Halting Problem

Let's say we can tell whether a program stops on an input.

Algorithm 3 Halt (program, input)

```
1: if program stops on input then
2:   return It Stops!
3: else
4:   return It Doesn't stop!
5: end if
```

Now let's look at this program:

Algorithm 4 smartass (program)

```
1: if Halt(program,program) then
2:   Go into an infinite loop
3: else
4:   return It doesn't stop! I'm stopping now!
5: end if
```

What happens if we call *smartass(smartass)*?

Acknowledgements and Sources

- Wikipedia
- Ikea catalog
- <http://www.cgl.uwaterloo.ca/~csk/halt/>
- <http://www.gmtel.net/web/windowshelp/algorithm.htm>
- Kevin Amaral, CS department.