History
Computers

Mathematics
+ Theory on Computation
+ Engineering
= Computers
Mathematics – Counting

The Túngara frog understands (add1 ...)

Early humans (apx. 19,000 BC) did some basic (?) mathematical operations (Ishango bones).
Mathematics – Calculations

With the advent of complex societies arose...

- the need for record-keeping: Old Kingdom, apx. 2,600 BC
- the need for land-surveys: Third Dynasty of Ur, apx. 2,000 BC
- the demand for mysticism: Vedic period, apx. 1,200 BC; Shang Dynasty, apx. 1,200 BC
Mathematics – Formal systems

First mathematical system:
• Euclidean geometry: Euclid of Alexandria, apx. 300 BC

Number systems:
• First decimal positional system: Archimedes of Syracuse, apx. 250 BC; Muhammad ibn Musa al-Khwarizmi apx. 800 AD
Mathematics – Formal systems

Differential and Integral Calculus
• Newton, (1666) & 1693 & 1704
• Leibniz, (1674) & 1684

You stole my idea!!!
How could I??? You didn’t tell anyone!!!
Language

[English] Compute: calculate

[French] computer: drawing calendars according to astronomical data

[Latin] com puto: arranging together

*I have read the truest computer of times*

-- Richard Brathwait, The Yong Mans Gleanings, 1613

Until the late 1800s, a computer was strictly a person, not a device.
Engineering – Basic operations

Abacus:
• Invented by most civilizations (e.g., Incan Quipu, Chinese Suanpan)
• Manual operation
• Adding, subtraction, multiplication, division
• Data storage (memory)
Engineering – Specific operations

Clocks, astrolabe:

• Invented by many civilizations (e.g., France, apx. 1600)
• Mechanical (gears)
• Telling time, longitude, latitude
Engineering – Calculators

Step reckoner

- Leibnitz (1694): *It is beneath the dignity of excellent men to waste their time in calculation when any peasant could do the work just as accurately with the aid of a machine.*

- Mechanical (gears)

- Adding, subtraction, multiplication, division
“Engineering” – Calculators

Mathematical tables

• Pre-calculated look-up values for common complex operations

• Trigonometry, logarithms
Engineering – Advanced operations

Slide rule

• Invented apx. 1620 by William Oughtred, possibly earlier

• Manual operation

• Multiplication, division, exponentials, logarithms
Engineering – Advanced calculators

Difference Engine, 1822
• Charles Babbage
• Mechanical (gears)
• +, -, ·, /, exponentials, logarithms, trigonometry, solving polynomials

Never built 😞
Engineering – Programmable calculators

Analytical Engine, 1837
• Charles Babbage
• Mechanical (gears)
• +, -, ·, /, exponentials, logarithms, trigonometry, solving polynomials
• Programmable: could be given data & a sequence of operations

Never built 😞
Engineering – Programmable calculators

Programming the Analytical Engine

• Ada King, Countess of Lovelace (1843): A new, a vast, and a powerful language is developed for the future use of analysis.
Engineering – Electro-mechanical calculators

Tabulating machine, 1890

• Herman Hollerith
• Electro-mechanical (gears and mercury)
• Adding

10 x faster than manual tabulation
Engineering – Electro-mechanical calculators

In 1896, Herman Hollerith founded the
• Tabulating Machine Company
which got amalgamated in 1911 into the
• Computing-Tabulating-Recording Company
which was renamed in 1924 to
• International Business Machines Corporation
Engineering – First crisis of mechanics

Analog computers (e.g., computers that use cogs internally) were too slow due to physical constraints of the gearboxes (e.g., momentum, friction).

Digital computers had to be designed.

Problems:

• How to represent data?
• How to build the required circuitry?
Voltage as data: associate certain voltage ranges with certain discrete values

How many ranges should there be?
Engineering – Analog to digital

Voltage as data: quinary computer (5 states), for example
Engineering – Analog to digital

Quinary computer: not stable
Engineering – Analog to digital

Binary computer (2 states): (more) stable
Mathematics – Boolean algebra

Binary (Boolean) algebra

- George Boole, The Laws of Though, 1854
- Extension of Aristotle’s logic with the goal to allow truth to be systematically and formally proven.

Operators:
- and
- or
- not
- xor: (or (and (not x) y) (and x (not y)))
Engineering – Boolean algebra

Switch
• Allows for manual control of electricity: a human operator closes or opens a lever

Relay
• Allows for electric control of electricity: a control current closes or opens a lever
Engineering – Boolean algebra

Relay, early 1830s
• Maybe Gauss invented it?

With relays, the output of one relay can be the control for another!
Engineering – Digital computers

Z3, 1941
• Konrad Zuse
• Electro-mechanical (relays)
• +, -, ·, /, sqrt
• Digital
• Programmable
• 1.2 Hz (+), 0.3 Hz (·)
Engineering – Digital computers

Harvard Mark I, 1944

• John von Neumann
• Electro-mechanical (relays)
• +, -, ·, /, logarithms, trigonometry
• Digital
• Programmable
• 3 Hz (+), 0.15 Hz (·)
Engineering – Digital computers

Grace Hopper

• Programmer for the Harvard Mark I
• Coined the term “bug” (according to Computer Science folklore...)
Engineering – Second crisis of mechanics

Electro-mechanical computers (e.g., computers that use relays internally) were too slow due to physical constraints of the relays (e.g., momentum, friction).

Fully electric computers had to be designed.

Problems:
• How to construct gates?
Engineering – Boolean algebra

Relay
• Allows for electric control of electricity: a control current closes or opens a lever

Vacuum tube
• Allows for electric control of electricity: a control current makes a section of the circuit conductive or not
Engineering – Electronic computers

Colossus, 1943
• Alan Turing
• Electronic (vacuum tubes)
• Inverting Lorenz SZ40 operations
• Digital
• Programmable
• Apx. 5 kHz (!)
Mathematics – The Foundational Crisis

Foundational Crisis of Mathematics, 1900s

• Mathematicians were not able to express the axioms of mathematics within themselves.

• E.g., Russell’s Paradox, Bertrand Russell, 1901

Lol, I broke math.
Mathematics – The Foundational Crisis

David Hilbert:

• (Hilbert’s Axioms, 1899: New set of axioms to describe Euclidean Geometry)
• (Hilbert’s Problems, 1900: 23, partially still unanswered, problems of mathematics.)

Hilbert’s Program, 1920s

• Attempt to find a new set of axioms of mathematics that are complete and consistent.

Np, I’ll fix it!
Also, check out my cool hat.
Mathematics – The Foundational Crisis

Complete: for any formula $\varphi$, if $\varphi$ is true, then $\varphi$ is provable.

Consistent: for any formula $\varphi$, there are no proofs of both $\varphi$ and ($not$ $\varphi$).
Mathematics – Gödel’s Proof

Complete: for any formula \( \varphi \), if \( \varphi \) is true, then \( \varphi \) is provable.

- Kurt Gödel, 1931: Any axiom system powerful enough to describe arithmetic on integers is not complete.

Consistent: for any formula \( \varphi \), there are no proofs of both \( \varphi \) and (\( not \varphi \)).

- Kurt Gödel, 1931: If an axiom system is consistent, its consistency cannot be proved within the system.
Mathematics – Gödel’s Proof (www.youtube.com/watch?v=O4ndlIDcDSGc)
Mathematics – Hilbert’s questions

Hilbert’s Entscheidungsproblem, 1928

• Does an algorithm exist that takes a logic statement as input and produces Yes if that statement is valid and No otherwise?

First raised by Leibnitz in the 1670s.
Theory on Computation – Hilbert’s questions

Hilbert’s Entscheidungsproblem, 1928

Alonzo Church, 1936
• Not solvable when using $\lambda$-calculus.

Alan Turing, 1936
• Not solvable when using a Turing machine.

Booth proofs are equivalent (Church–Turing thesis), but Turing’s is better.
Theory on Computation – $\lambda$ calculus
(www.youtube.com/watch?v=eis11j_iGMs)
Theory on Computation – The Turing Machine

(www.youtube.com/watch?v=dNRDvLACg5Q)
Theory on Computation – The Halting Problem
(www.youtube.com/watch?v=macM_MtS_w4)
CQ1 – Who is your CS 135 instructor?

A. Alan Turing
B. Alonzo Church
C. Adrian Reetz
D. Byron Weber Becker
E. Mr. Fancy Pants
F. None of the above
G. All of the above
H. Donald Trump
Theory on Computation – Formal Grammar

Noam Chomsky

• “Father of modern linguistics”

• Chomsky hierarchy, 1956
  • Classifies formal languages into 4 categories: Type-0 to Type-3
  • Turing machine implements a Type-0 language
  • Important to answer questions regarding computability, complexity, etc.

More to come...
Engineering – Hardware architecture

Alan Turing
• Automatic Computing Engine, 1945

John von Neumann
• Von-Neumann Architecture, 1945
  • Processing unit
  • Control unit
  • Memory for data and instructions
  • Mass storage
  • Input and output
Engineering – Third crisis of mechanics

Vacuum tubes break too easily and burn out too quickly

Solution: Solid state semi-conductors, a.k.a. transistors, 1947

First transistor, 1947

Intel 8088, 1979: 4.5k trans.

Engineering – Software

Grace Hopper

• Proposed the idea of high-level programming languages, 1950
• Proposed the idea of compilers, 1952
• Developed the first programming language (FLOWMATIC, for the UNIVAC I computer), 1955
Engineering – Software

John McCarthy

• Coined the term Artificial Intelligence
• Invented LISP, 1958, a predecessor for Racket
Hardware-inspired

Math-inspired
Hardware-inspired

Math-inspired
Looking ahead to CS 136

• We have been fortunate to work with very small languages (the teaching languages) writing very small programs which operate on small amounts of data.

• In CS 136, we will broaden our scope, moving towards the messy but also rewarding realm of the “real world”.

• The main theme of CS 136 is scalability: what are the issues which arise when things get bigger, and how do we deal with them?
Looking ahead to CS 136

• How do we organize a program that is bigger than a few screenfuls?
• How do we reuse and share code, apart from cutting-and-pasting it into a new program file?
• How do we design programs so that they run efficiently?
• What changes might be necessary to our notion of types and to the way we handle errors when there is a much greater distance in time and space between when the program is written and when it is run?
Looking ahead to CS 136

• When is it appropriate to abstract away from implementation details for the sake of the big picture, and when must we focus on exactly what is happening at lower levels for the sake of efficiency?

• These are issues which arise not just for computer scientists, but for anyone making use of computation in a working environment.

• We can build on what we have learned this term in order to meet these challenges with confidence.