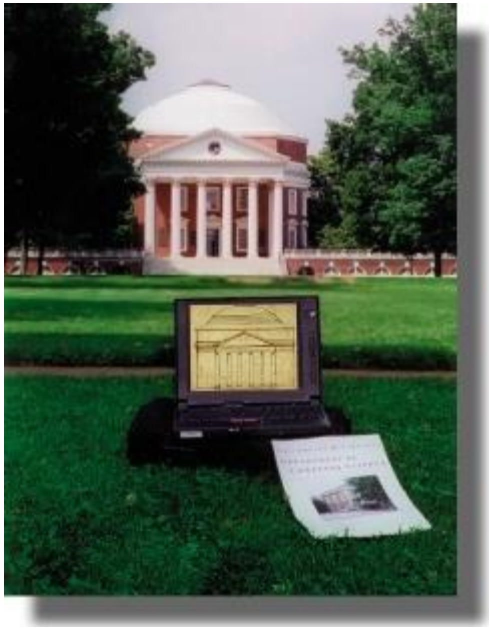


Algorithms
CS6161

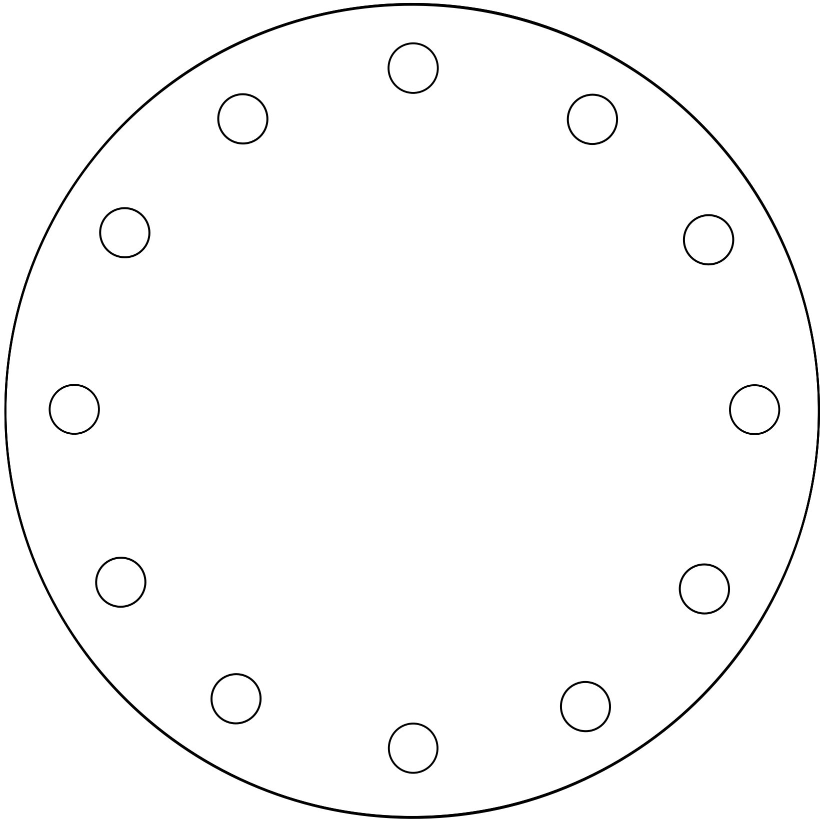
Gabriel Robins

Department of
Computer Science

University of Virginia

www.cs.virginia.edu/robins

Problem: Can 5 test tubes be spun simultaneously in a 12-hole centrifuge in a balanced way?



- What approaches fail?
- What techniques work and why?
- Lessons and generalizations

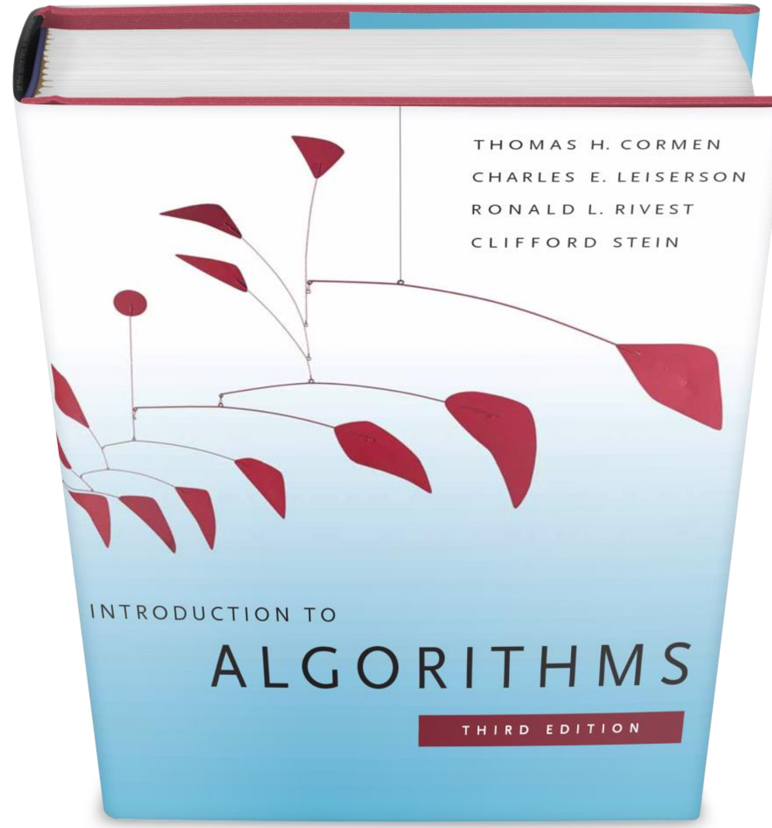
Algorithms (CS6161) Textbook

Textbook:

Introduction to Algorithms

by Cormen et al (MIT)

Third Edition, 2009

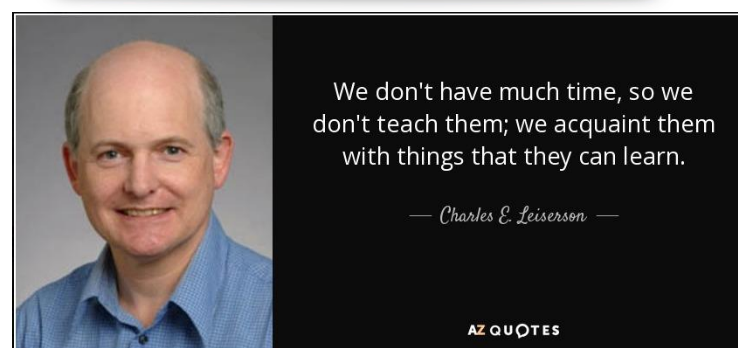



Thomas Cormen


Charles Leiserson


Ronald Rivest


Clifford Stein



computer science

Introduction to Algorithms

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein

Third Edition

Some books on algorithms are rigorous but incomplete; others cover masses of material but lack rigor. *Introduction to Algorithms* uniquely combines rigor and comprehensiveness. The book covers a broad range of algorithms in depth, yet makes their design and analysis accessible to all levels of readers. Each chapter is relatively self-contained and can be used as a unit of study. The algorithms are described in English and in a pseudocode designed to be readable by anyone who has done a little programming. The explanations have been kept elementary without sacrificing depth of coverage or mathematical rigor.

The first edition became a widely used text in universities worldwide as well as the standard reference for professionals. The second edition featured new chapters on the role of algorithms, probabilistic analysis and randomized algorithms, and linear programming. The third edition has been revised and updated throughout. It includes two completely new chapters, on van Emde Boas trees and multithreaded algorithms, and substantial additions to the chapter on recurrences (now called "Divide-and-Conquer"). It features improved treatment of dynamic programming and greedy algorithms and a new notion of edge-based flow in the material on flow networks. Many new exercises and problems have been added for this edition.

As of the third edition, this textbook is published exclusively by the MIT Press.

Thomas H. Cormen is Professor of Computer Science and former Director of the Institute for Writing and Rhetoric at Dartmouth College. Charles E. Leiserson is Professor of Computer Science and Engineering at MIT. Ronald L. Rivest is Andrew and Erna Viterbi Professor of Electrical Engineering and Computer Science at MIT. Clifford Stein is Professor of Industrial Engineering and Operations Research at Columbia University.

"In light of the explosive growth in the amount of data and the diversity of computing applications, efficient algorithms are needed now more than ever. This beautifully written, thoughtfully organized book is the definitive introductory book on the design and analysis of algorithms. The first half offers an effective method to teach and study algorithms; the second half then engages more advanced readers and curious students with compelling material on both the possibilities and the challenges in this fascinating field."
—Shang-Hua Teng, University of Southern California

"*Introduction to Algorithms*, the 'bible' of the field, is a comprehensive textbook covering the full spectrum of modern algorithms: from the fastest algorithms and data structures to polynomial-time algorithms for seemingly intractable problems, from classical algorithms in graph theory to special algorithms for string matching, computational geometry, and number theory. The revised third edition notably adds a chapter on van Emde Boas trees, one of the most useful data structures, and on multithreaded algorithms, a topic of increasing importance."
—Daniel Spielman, Department of Computer Science, Yale University

"As an educator and researcher in the field of algorithms for over two decades, I can unequivocally say that the Cormen book is the best textbook that I have ever seen on this subject. It offers an incisive, encyclopedic, and modern treatment of algorithms, and our department will continue to use it for teaching at both the graduate and undergraduate levels, as well as a reliable research reference."
—Gabriel Robins, Department of Computer Science, University of Virginia

978-0-262-03384-8

The MIT Press
Massachusetts Institute of Technology
Cambridge, Massachusetts 02142
http://mitpress.mit.edu

90000
9 780262 033848

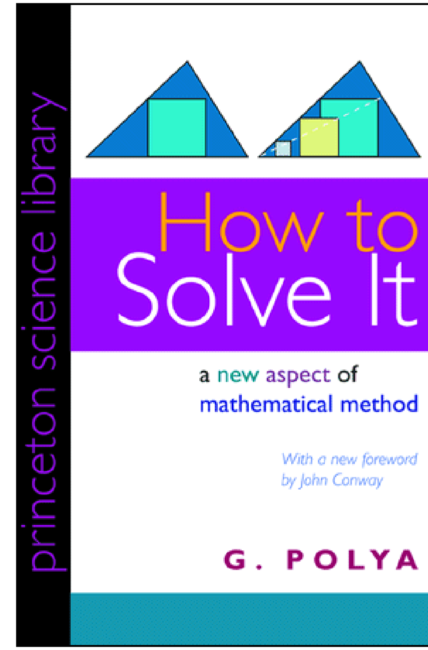
Algorithms (CS6161) Textbook

Supplemental reading:

How to Solve It, by George Polya (MIT)

Princeton University Press, 1945

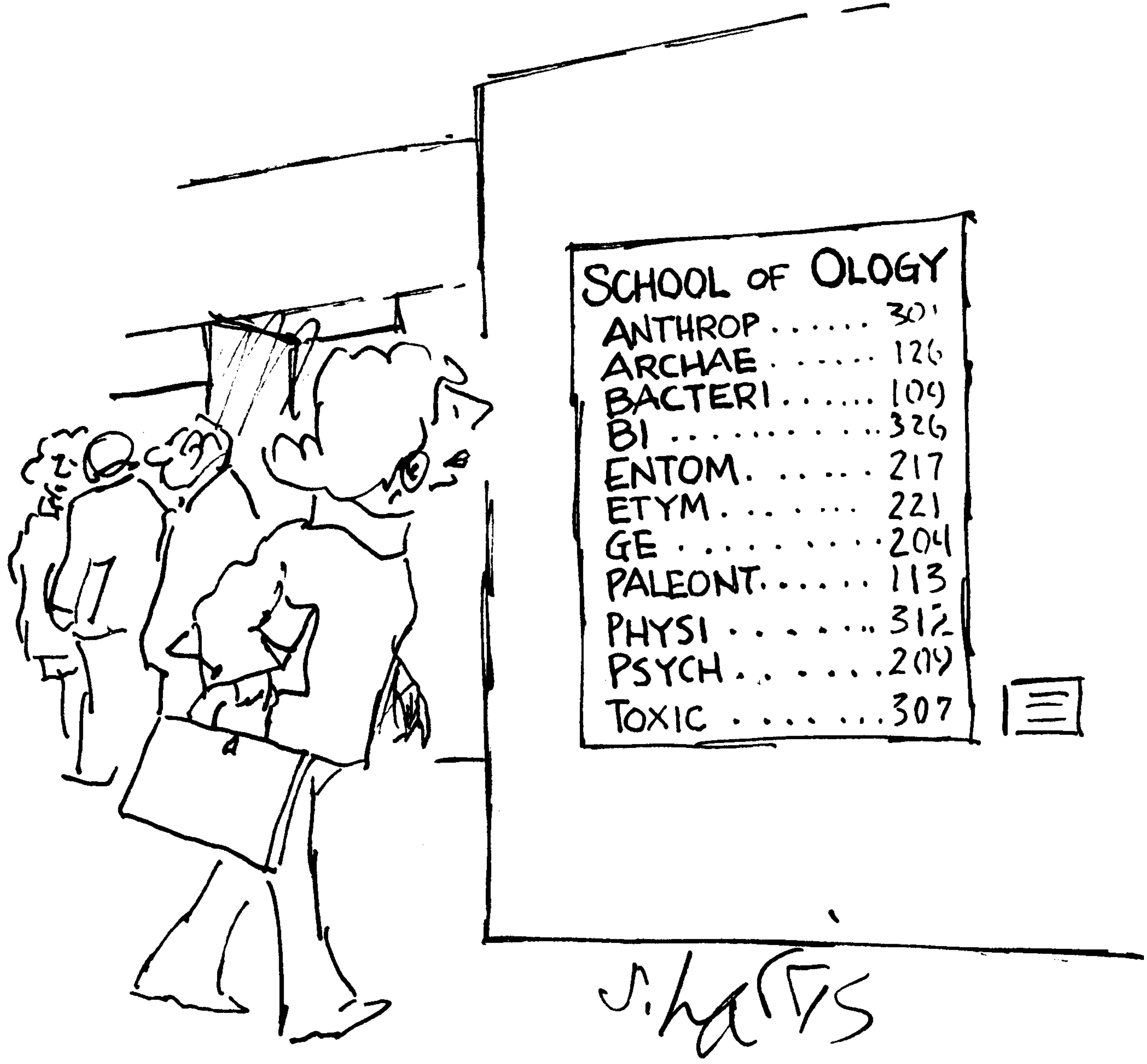
- A classic on problem solving



Good Articles / videos:

www.cs.virginia.edu/robins/CS_readings.html

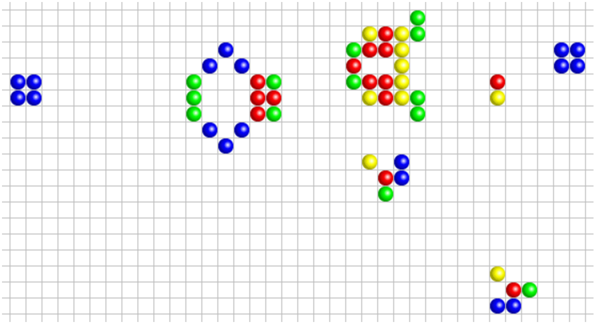
George Polya (1887-1985)

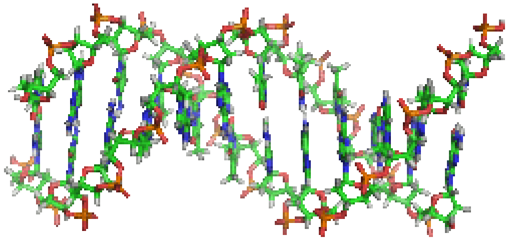

SCHOOL OF OLOGY
ANTHROP 301
ARCHAE 126
BACTERI 109
BI 326
ENTOM 217
ETYM 221
GE 204
PALEONT 113
PHYSI 312
PSYCH 209
TOXIC 307
J. Harris

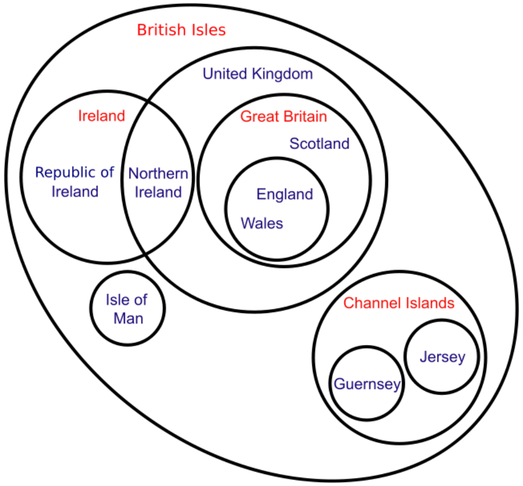
Algorithms Syllabus

Fundamentals:

- History of algorithms
- Problem solving
- Pigeon-hole principle
- Occam's razor
- Uncomputability
- Universality
- Asymptotic complexity
- Set theory and logic



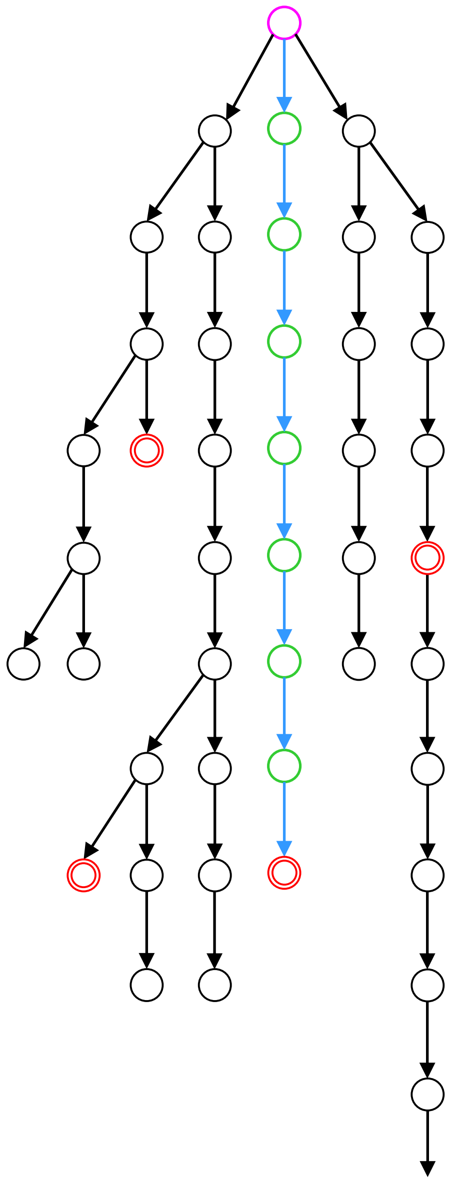




Algorithms Syllabus

Data structures:

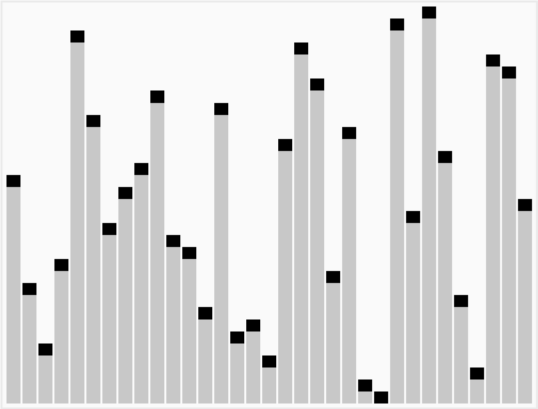
- Arrays
- Stacks and queues
- Linked lists
- Binary and general trees
- Height-balanced trees
- Heaps
- Hash tables

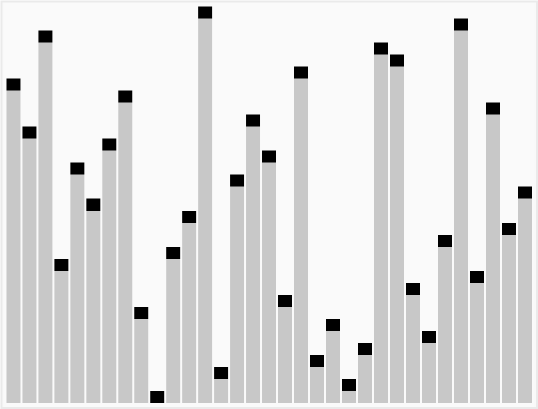


Algorithms Syllabus

Sorting and searching:

- Classical sorting methods
- Specialized sorting techniques
- Finding max & min
- Median finding and K^{th} selection
- Majority detection
- Meta algorithms

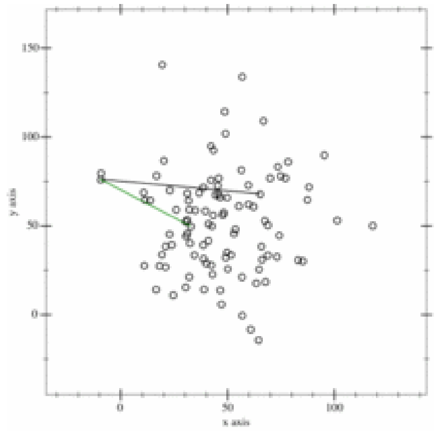




Algorithms Syllabus

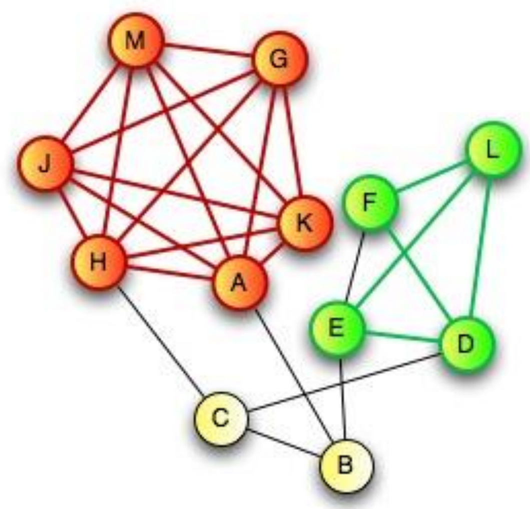
Computational geometry:

- Convex hulls
- Lower bounds
- Line segment intersection
- Planar subdivision search
- Voronoi diagrams
- Nearest neighbors
- Geometric minimum spanning trees
- Delaunay triangulations
- Distance between convex polygons
- Triangulation of polygons
- Collinear subsets

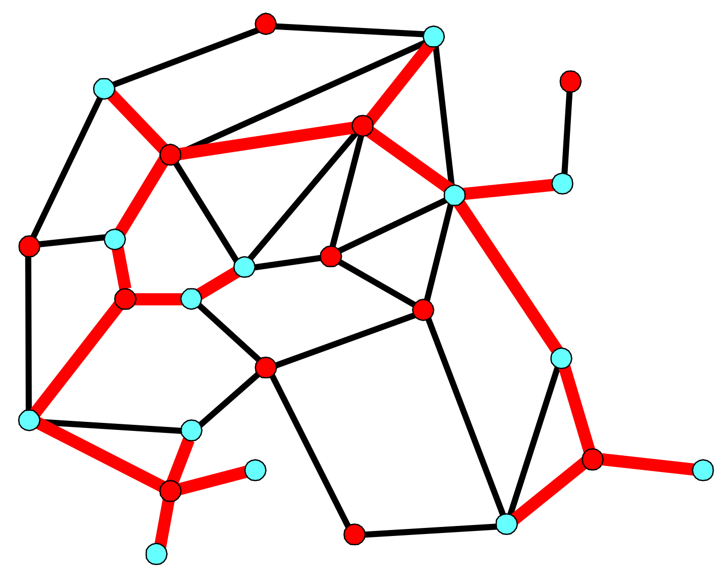


Algorithms Syllabus

Graph algorithms:



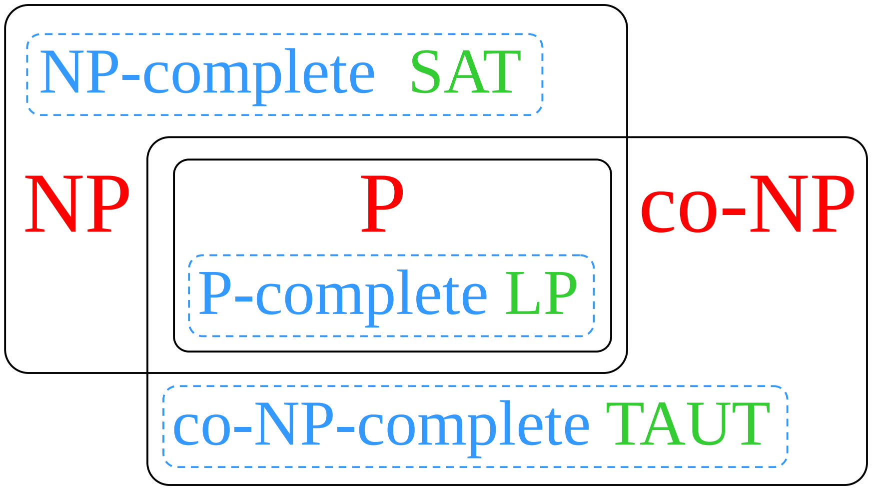
- Depth-first search
- Breadth-first search
- Minimum spanning trees
- Shortest paths trees
- Radius-cost tradeoffs
- Steiner trees
- Degree-constrained trees



Algorithms Syllabus

NP-completeness:

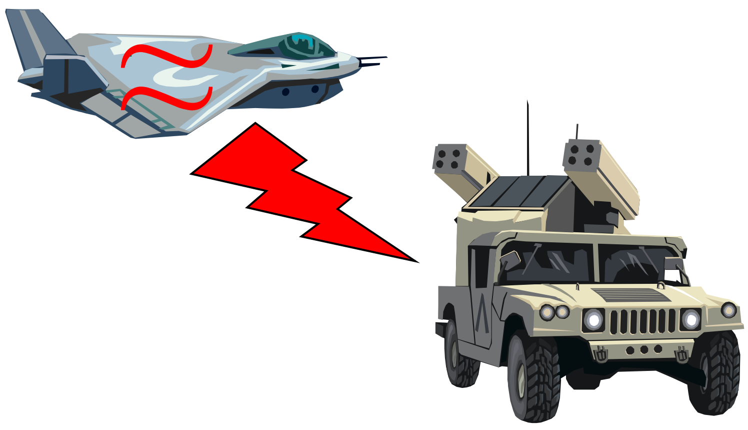
- Resource-constrained computation
- Complexity classes
- Intractability
- Boolean satisfiability
- Cook-Levin theorem
- Transformations
- Graph clique problem
- Independent sets
- Hamiltonian cycles
- Colorability problems
- Heuristics



Algorithms Syllabus

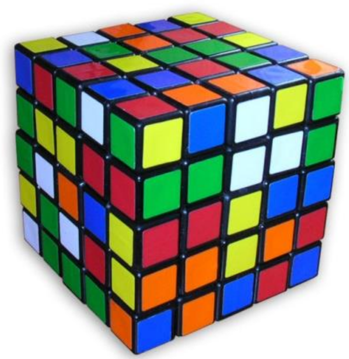
Other topics in algorithms:

- Linear programming
- Matrix multiplication
- String matching
- Minimum matchings
- Network flows
- Distributed algorithms
- Amortized analysis
- Zero knowledge proofs



Overarching Philosophy

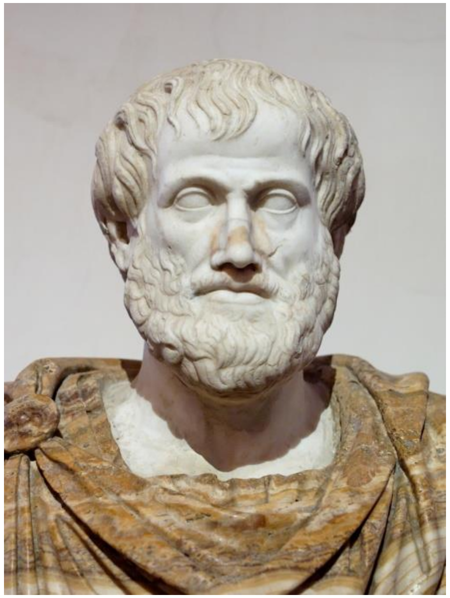
- Focus on the “big picture” & “scientific method”
- Emphasis on problem solving & creativity
- Discuss applications & practice
- A primary objective: have fun!

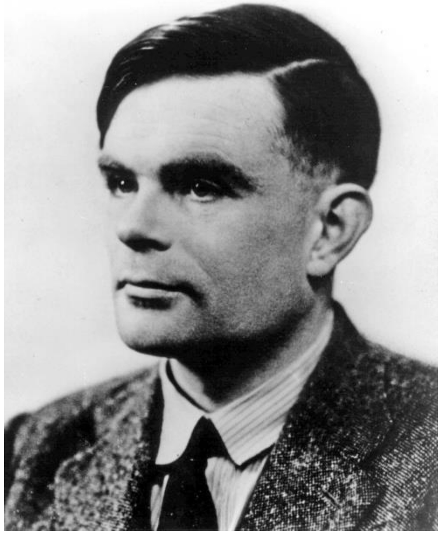


Algorithms Throughout History

A brief history of computing:

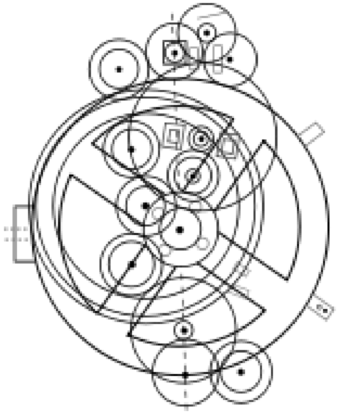
- Aristotle, Euclid, Archimedes, Eratosthenes
- Abu Ali al-Hasan ibn al-Haytham
- Fibonacci, Descartes, Fermat, Pascal
- Newton, Euler, Gauss, Hamilton
- Boole, De Morgan, Babbage, Ada Agusta
- Venn, Carroll, Cantor, Hilbert, Russell
- Hardy, Ramanujan, Ramsey
- Godel, Church, Turing, von Neumann
- Shannon, Kleene, Chomsky

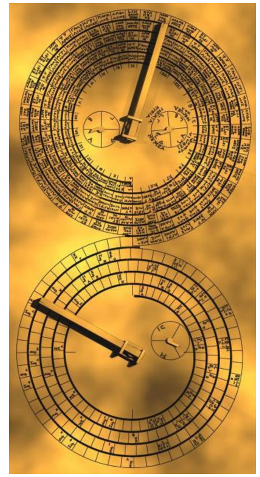




An Ancient Computer: The Antikythera

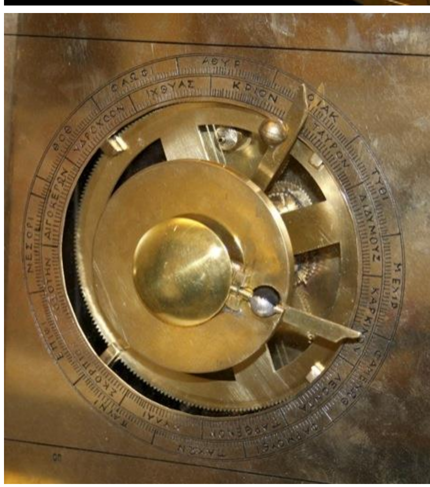
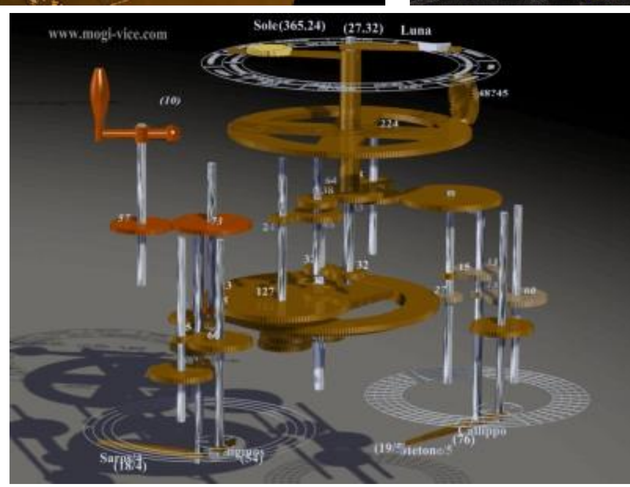
- Oldest known mechanical computer
- Built around 150-100 BCE !
- Calculates eclipses and astronomical positions of sun, moon, and planets
- Very sophisticated for its era
- Contains dozens of intricate gears
- Comparable to 1700’s Swiss clocks
- Has an attached “instructions manual”
- Still the subject of ongoing research

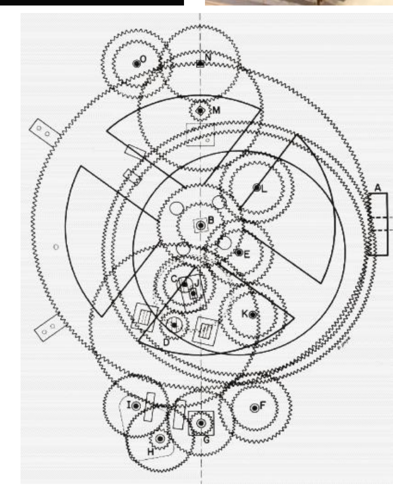


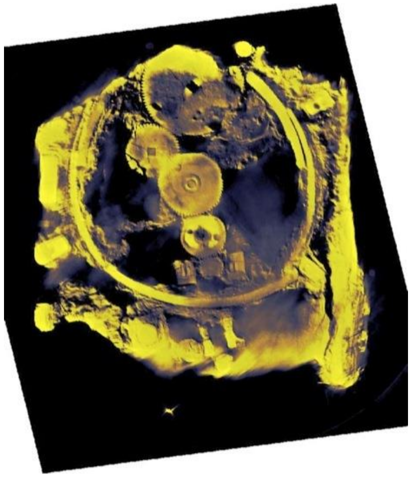



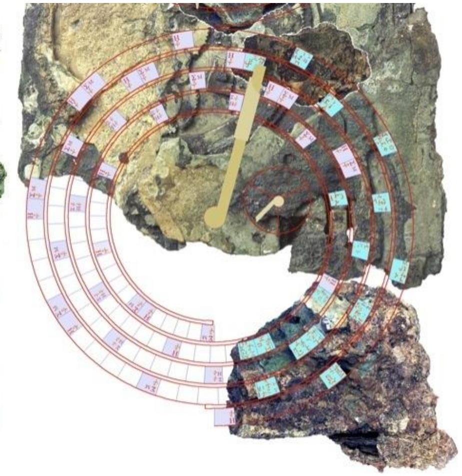
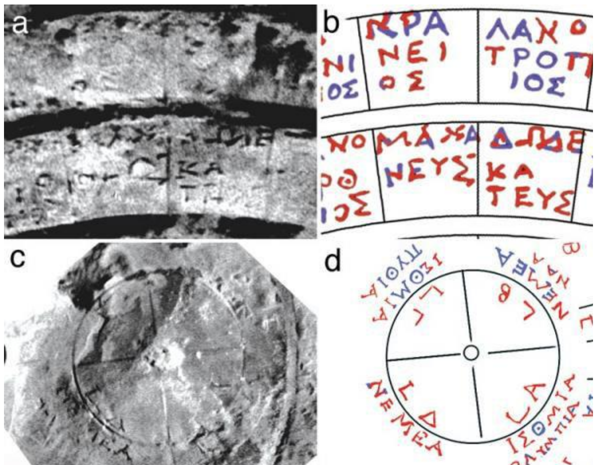
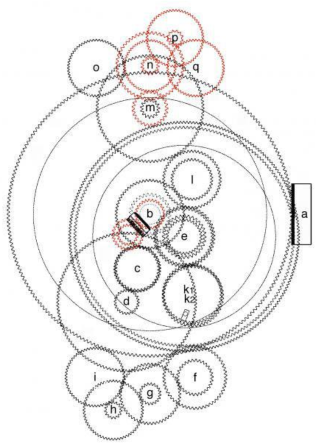
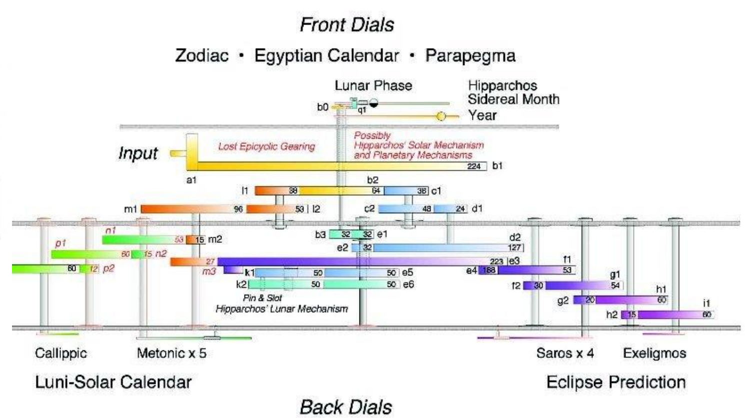
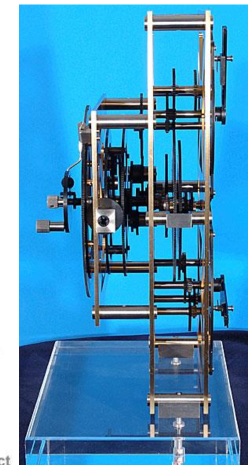
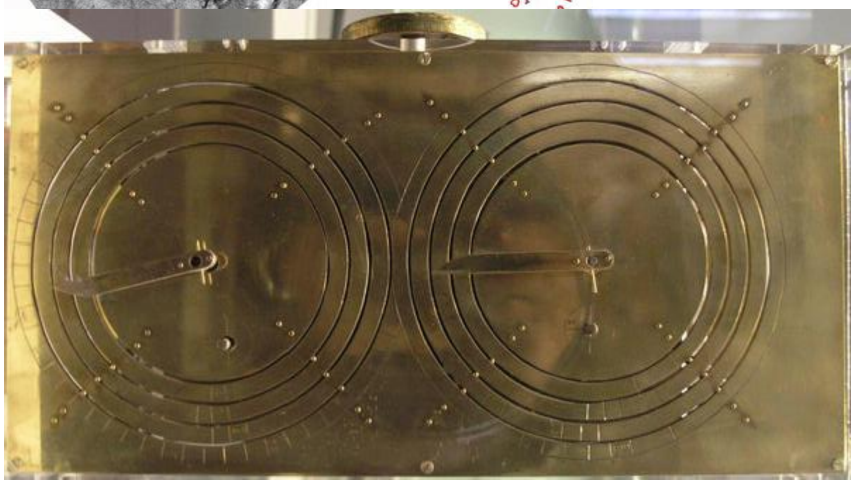







www.mogi-vice.com
Sole(365.24)
(27.32)
Luna
224
127
Calippo
(76)





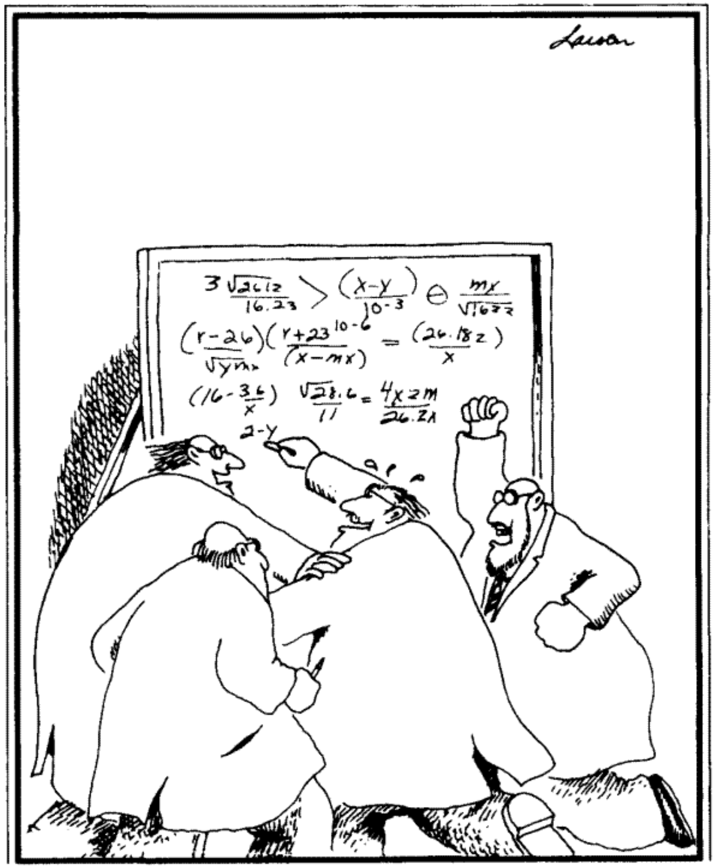

© Antikythera Mechanism Research Project


a
b
c
d


Front Dials
Zodiac • Egyptian Calendar • Parapegma
Lunar Phase
Hipparchos
Sidereal Month
Year
Input
Lost Epicyclic Gearing
Possibly Hipparchos' Solar Mechanism and Planetary Mechanisms
Pin & Slot
Hipparchos' Lunar Mechanism
Callippic
Metonic x 5
Saros x 4
Exeligmos
Luni-Solar Calendar
Eclipse Prediction
Back Dials
© Antikythera Mechanism Research Project




Prerequisites

- Some discrete math & algorithms knowledge
- Ideally, should have taken CS4102
- Course will "bootstrap" (albeit quickly) from first principles
- Critical: Tenacity, patience

Course Organization

- Exams: probably take home
  - Decide by vote
  - Flexible exam schedule
- Problem sets:
  - Lots of problem solving
  - Work in groups!
  - Not formally graded
  - Many exam questions will come from homeworks!
- Project and demo
- Extra credit problems
  - In class & take-home
  - Find mistakes in slides, handouts, etc.
- Course materials posted on Web site www.cs.virginia.edu/robins/algorithms



"Go for it, Sidney! You've got it! You've got it! Good hands! Don't choke!"

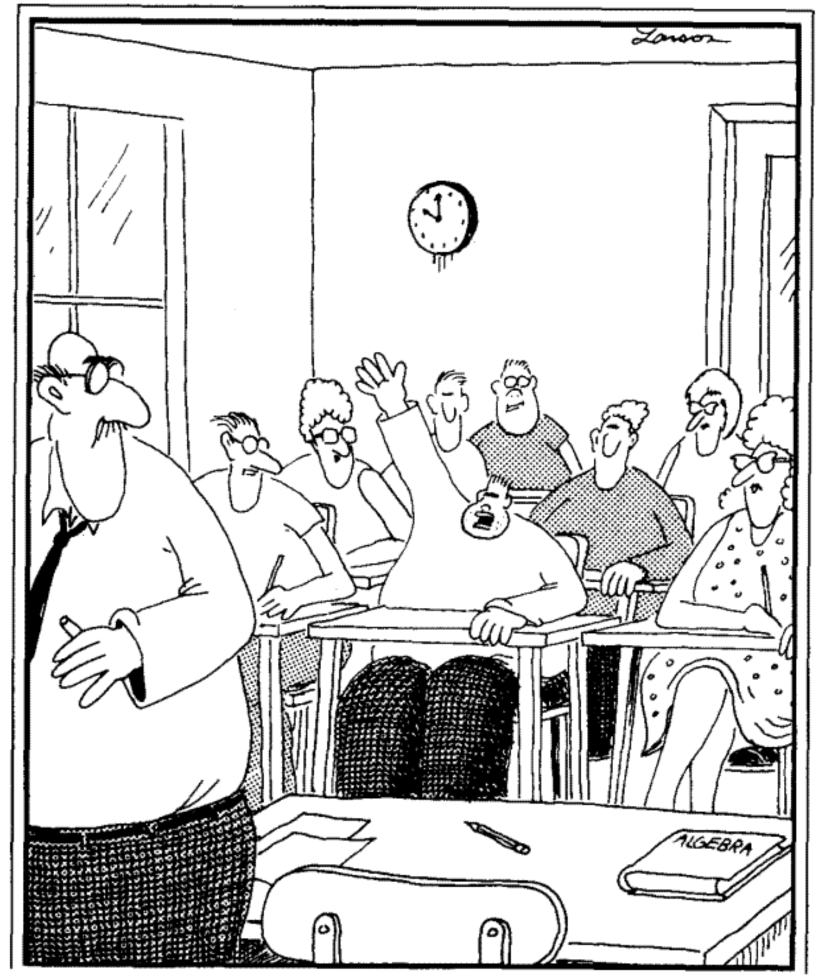
Grading Scheme

- Attendance 10%
- Readings 20%
- Midterm 20%
- Final 20%
- Project 30%
- Extra credit 10%

Total: 110% +

Best strategy:

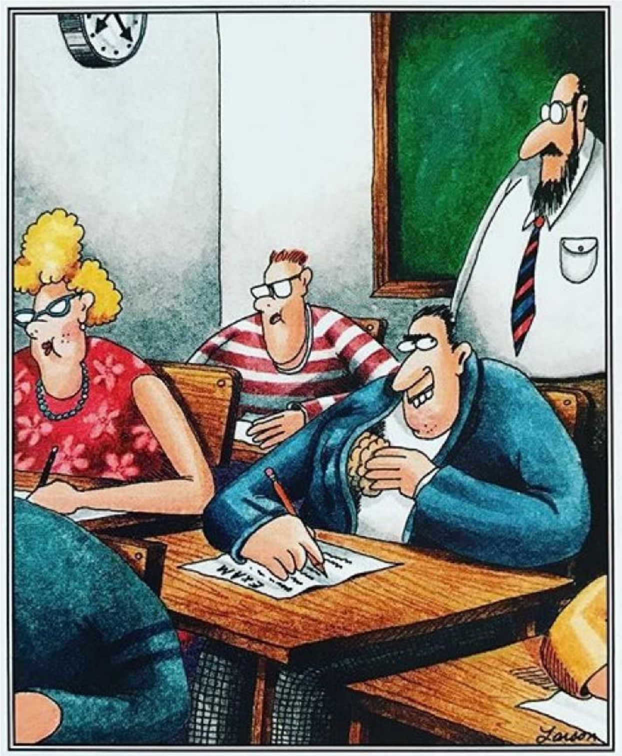
- Solve lots of problems!
- Do lots of readings / EC!
- "Ninety percent of success is just showing up." – Woody Allen



"Mr. Osborne, may I be excused? My brain is full."

Cheating Policy

- Cheating / plagiarism is strictly prohibited
- Serious penalties for violators
- Please review the UVa Honor Code
- Examples of Cheating / plagiarism:
  - Mass-copying of solutions from others / Web
  - Mass-sharing of solutions with others / Web
  - Cutting-and-pasting from other people / Web
  - Copying article/book/movie reviews from people / Web
  - Other people / Web solving entire problems for you
  - Providing other people / Web with verbatim solutions
  - This list is not exhaustive!



Midway through the exam, Allen pulls out a bigger brain.

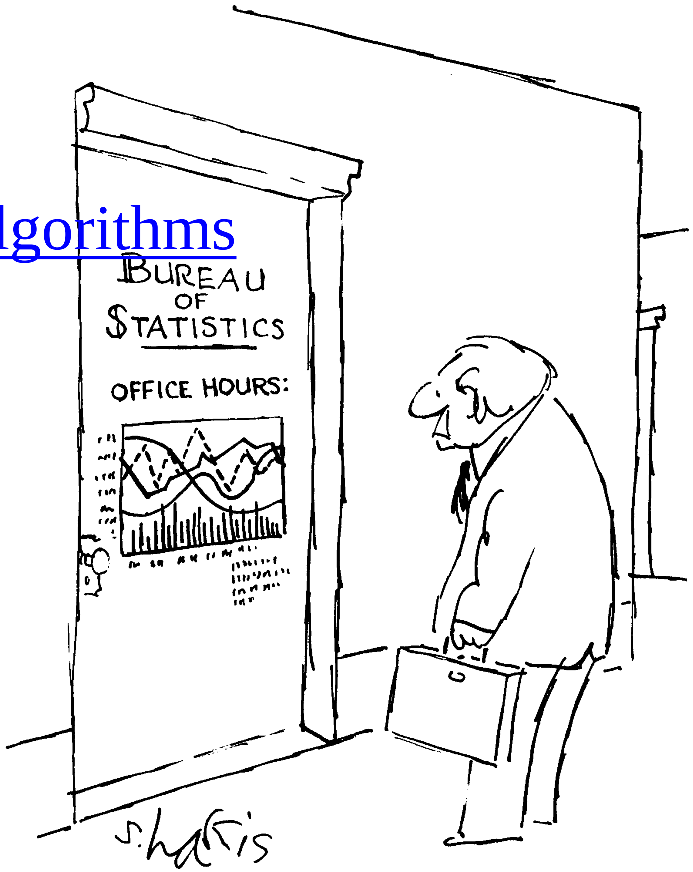
- We have automated cheating / plagiarism detection tools!
- We encourage collaborations / brainstorming
- Lets keep it positive (and not play “gotcha”)

Contact Information

Professor	Gabriel Robins
Office:	406 Rice Hall
Phone:	(434) 982-2207
Email:	robins@cs.virginia.edu
Web:	www.cs.virginia.edu/robins
	www.cs.virginia.edu/robins/algorithms

Office hours: right after class

- Any other time
- By email (preferred)
- By appointment
- Q&A blog posted on class Web site



Course Readings

www.cs.virginia.edu/robins/CS_readings.html

Goal: broad exposure to lots of cool ideas & technologies!

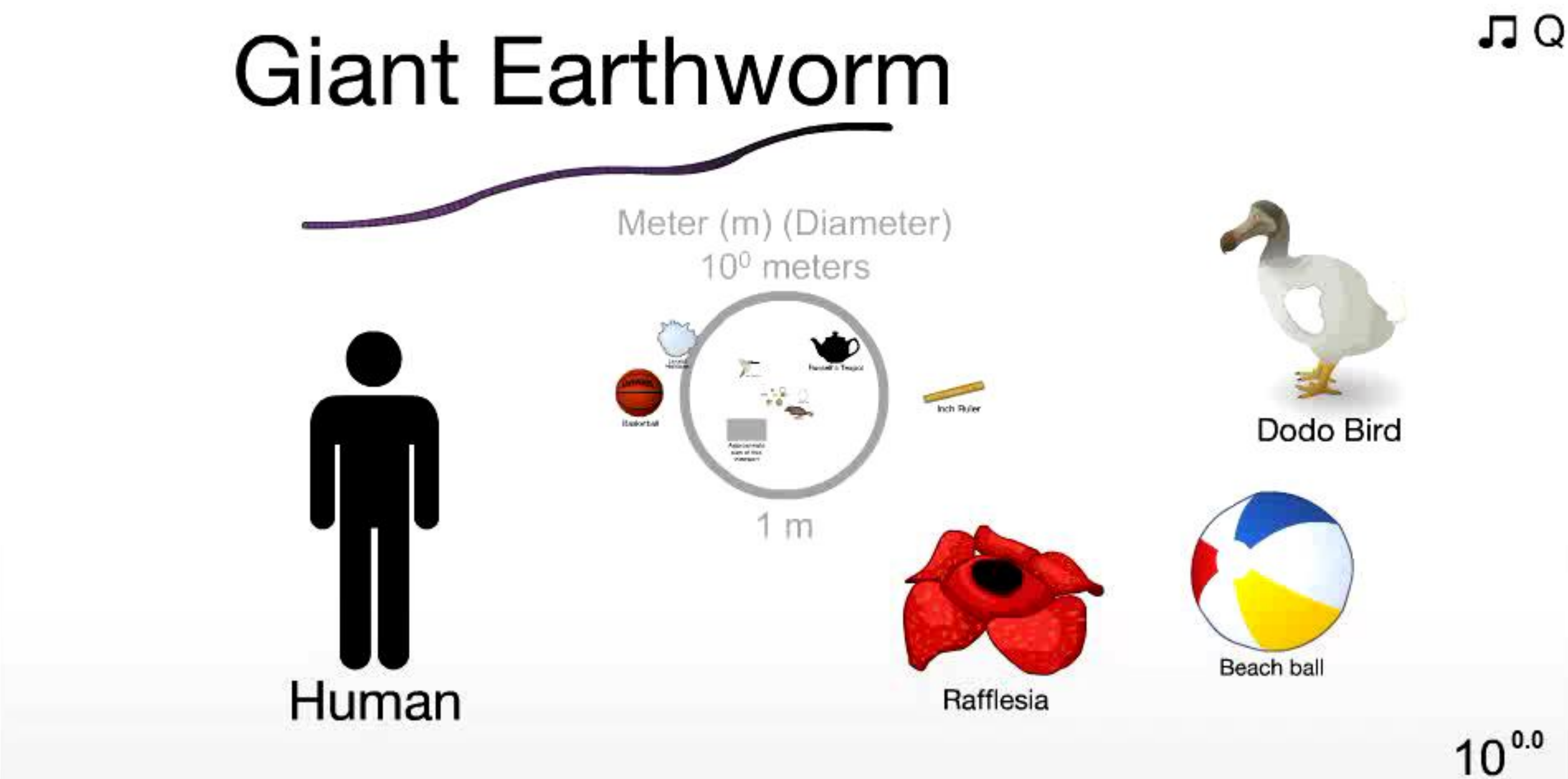
- Required: total of at least 20 items over the semester
- Diverse categories: videos / Web sites / papers / books
- More than 20 total is even better! (extra credit)
- Pace: minimum two per week, and maximum two per day
- Don’t procrastinate!
- Email all submissions to: homework.cs6161@gmail.com

Required Readings
www.cs.virginia.edu/robins/CS_readings.html

- Required videos:
  - Last Lecture, Randy Pausch, 2007
  - Time Management, Randy Pausch, 2007
  - Powers of Ten, Charles and Ray Eames, 1977

Required Readings

"[Scale of the Universe]", Cary and Michael Huang, 2012

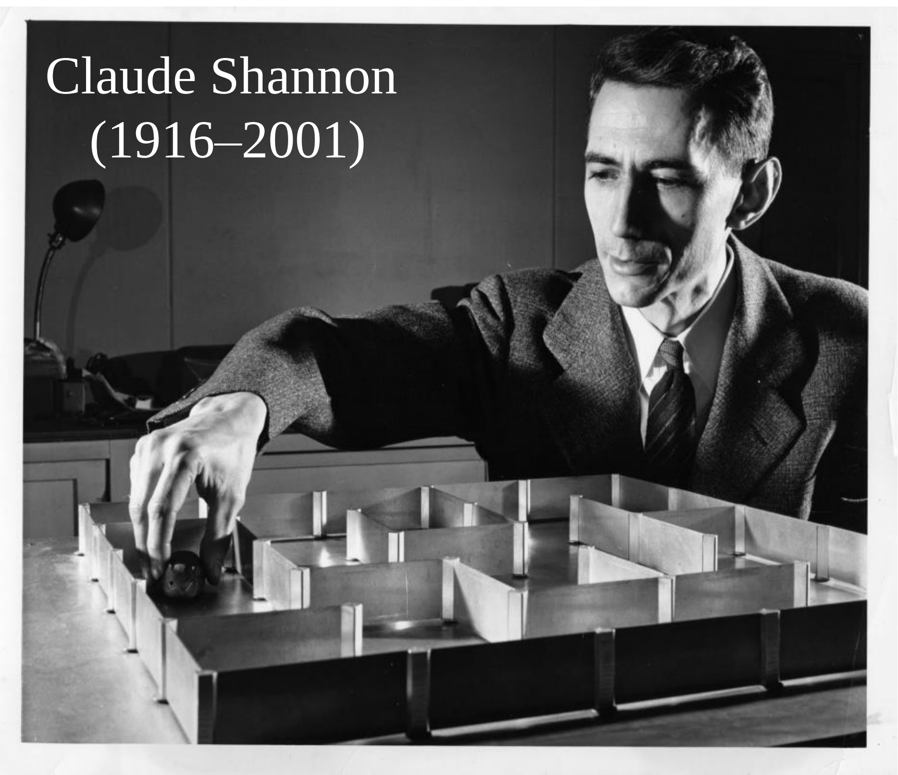


- 10^{-24} to 10^{26} meters $\Rightarrow$ 50 orders of magnitude!

Required Readings

www.cs.virginia.edu/robins/CS_readings.html

- More required videos:
  - Claude Shannon - Father of the Information Age, UCTV
  - The Pattern Behind Self-Deception, Michael Shermer, 2010



Claude Shannon (1916–2001)

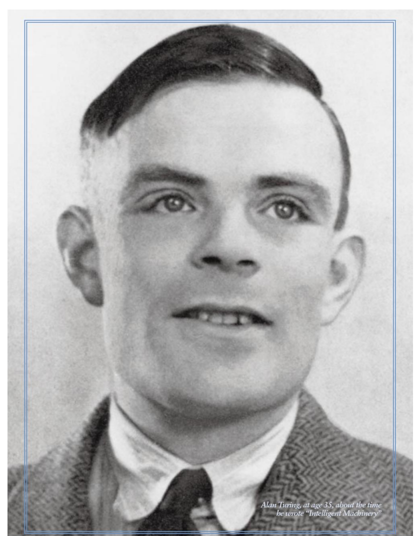
Michael Shermer

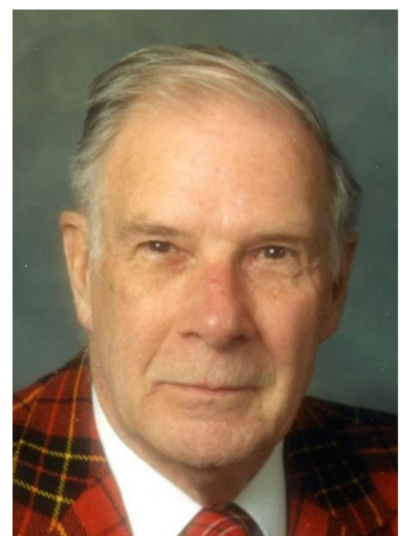
Required Readings

www.cs.virginia.edu/robins/CS_readings.html

- Required articles:
  - Decoding an Ancient Computer, Freeth, 2009
  - Alan Turing's Forgotten Ideas, Copeland and Proudfoot, 1999
  - You and Your Research, Richard Hamming, 1986
  - Who Can Name the Bigger Number, Scott Aaronson, 1999


Antikythera computer, 200BC


Alan Turing


Richard Hamming


Scott Aaronson


"BENEDICT CUMBERBATCH IS OUTSTANDING"
RADIO TIMES
"THE BEST BRITISH FILM OF THE YEAR"
THE INDEPENDENT
"AN INSTANT CLASSIC"
GLAMOUR
"A SUPERB THRILLER"
EMPIRE
TIME OUT
THE TIMES
THE IMITATION GAME
BENEDICT CUMBERBATCH
KEIRA KNIGHTLEY
12A MODERATE SEX REFERENCES
BASED ON THE INCREDIBLE TRUE STORY
/ImitationGameUK
IN CINEMAS NOVEMBER 14
Extra credit!

Basic Concepts and Notation

Gabriel Robins

Required reading

"When I use a word," Humpty Dumpty said, in a rather scornful tone, "it means just what I choose it to mean -- neither more nor less."

A **set** is formally an undefined term, but intuitively it is a (possibly empty) collection of arbitrary objects. A set is usually denoted by curly braces and some (optional) restrictions. Examples of sets are {1,2,3}, {hi, there}, and {k | k is a perfect square}. The symbol $\in$ denotes set **membership**, while the symbol $\notin$ denotes set **non-membership**; for example, $7 \in \{p \mid p$ prime$\}$ states that 7 is a prime number, while $q \notin \{0,2,4,6,\ldots\}$ states that q is not an even number. Some **common sets** are denoted by special notation:

The **natural numbers**: $\mathbb{N} = \{1,2,3,\ldots\}$

The **integers**: $\mathbb{Z} = \{\ldots,-3,-2,-1,0,1,2,3,\ldots\}$

The **rational numbers**: $\mathbb{Q} = \{\frac{a}{b} \mid a,b \in \mathbb{Z}, b \neq 0\}$

The **real numbers**: $\mathbb{R} = \{x \mid x$ is a real number$\}$

The **empty set**: $\varnothing = \{\}$

http://www.cs.virginia.edu/robins/cs6161/basics.pdf

JOHANNES KEPLER'S UPHILL BATTLE

Discrete Math Review Slides

Required reading

Symbolic Logic

Def: *proposition* - statement either true (T) or false (F)

Ex: 1 + 1 = 2

2 + 2 = 3

3 < 7

x + 4 = 5

"today is Monday"

Boolean Functions

- "and" $\wedge$
- "or" $\vee$
- "not" $\neg$
- "xor" $\oplus$
- "nand"
- "nor"
- "implication" $\Rightarrow$
- "equivalence" $\Leftrightarrow$

Logical Implication

- "implies" $\Rightarrow$

Truth table:

p	q	p⇒q
T	T	T
T	F	F
F	T	T
F	F	T

Ex: $(x<0) \wedge (x>0) \Rightarrow (x=0)$

$1 < x < y \Rightarrow x^3 < y^3$

"today is Sunday" $\Rightarrow$ 1+1=3

Logical Equivalence

- "biconditional" $\Leftrightarrow$

or "if and only if" ("iff")

or "necessary and sufficient"

or "logically equivalent" $\equiv$

Truth table:

p	q	p⇔q
T	T	T
T	F	F
F	T	F
F	F	T

Ex: $p \Leftrightarrow p$

$[(x=0) \vee (y=0)] \Leftrightarrow (xy=0)$

$\min(x,y)=\max(x,y) \Leftrightarrow x=y$

Predicates

Def: *predicate* - a function or formula involving some variables

Ex: let P(x) = "x > 3"

x is the variable

"x>3" is the predicate

P(5)

P(1)

Ex: Q(x,y,z) = "$x^2+y^2=z^2$"

Q(2,3,4)

Q(3,4,5)

Quantifiers

- Universal: "for all" $\forall$

$\forall x\ P(x)$

$\Leftrightarrow P(x_1) \wedge P(x_2) \wedge P(x_3) \wedge \ldots$

Ex: $\forall x \quad x < x + 1$

$\forall x \quad x < x^3$

- Existential: "there exists" $\exists$

$\exists x\ P(x)$

$\Leftrightarrow P(x_1) \vee P(x_2) \vee P(x_3) \vee \ldots$

Ex: $\exists x \quad x = x^2$

$\exists x \quad x < x - 1$

Combinations:

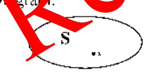
$\forall x \exists y \quad y>x$

Sets

Def: *set* - an unordered collection of elements

Ex: {1, 2, 3} or {hi, there}

Venn Diagram:



Def: two sets are *equal* iff they contain the same elements

Ex: {1, 2, 3} = {2, 3, 1}

{0} ≠ {1}

{3, 5} = {3, 5, 3, 3, 5}

Common Sets

Naturals: N = {1, 2, 3, 4, ...}

Integers: Z = {...,-2, -1, 0, 1, 2,...}

Rationals: Q = { $\frac{a}{b}$ | a,b∈Z, b≠0}

Reals: ℜ = {x | x a real #}

Empty set: Ø = {}

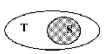
Z^+ = non-negative integers

$ℜ^-$ = non-positive reals, etc.

Subsets

- Subset notation: $\subseteq$

$S \subseteq T \Leftrightarrow (x\in S \Rightarrow x\in T)$



- Proper subset: $\subset$

$S \subset T \Leftrightarrow ((S \subseteq T) \wedge (S\neq T))$

$S=T \Leftrightarrow ((T \subseteq S) \wedge (S \subseteq T))$

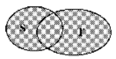
$\forall S \quad Ø \subseteq S$

$\forall S \quad S \subseteq S$

Union / Intersection

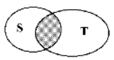
- Union: $\cup$

$S\cup T=\{x \mid x\in S \vee x\in T\}$



- Intersection: $\cap$

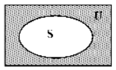
$S\cap T=\{x \mid x\in S \wedge x\in T\}$



Complement

- Universal set: U (everything)
- Set complement: S' or $\bar{S}$

S' = {x | x∉S} = U - S



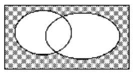
- Disjoint sets: S∩T=Ø

S - T = S ∩ T'

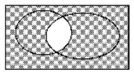
S - S = Ø

DeMorgan's Laws

(S∪T)' = S'∩T'



(S∩T)' = S'∪T'



Boolean logic version:

(X∧Y)'=X'∨Y'

(X∨Y)'=X'∧Y'

Function Types

- One-to-one function: "1-1"

a,b∈S ^ a≠b ⇒ *f*(a)≠*f*(b)

Ex: *f*: ℜ→ℜ, f(x)=2x is 1-1

g(x)=x² is not 1-1

- Onto function:

∀ t ∈T ∃ s∈S ∋ *f*(s)=t

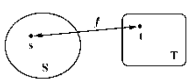
Ex: *f*: Z→Z, f(x)=13-x is onto

g(x)=x² is not onto

1-to-1 Correspondence

- 1-to-1 correspondence: *f*:S↔T

f is both 1-1 and onto



Ex: *f*: ℜ↔ℜ ∋ *f*(x)=x (identity)

h: N↔Z ∋ h(x)= $\frac{x-1}{2}$, x odd,

$\frac{-x}{2}$, x even.

Generalized Cardinality

- S is at least as large as T:

|S ≥ T| ⇒ ∃ *f*:S→T, *f* onto

i.e., "S covers T"

Ex: r:ℜ→Z, r(x)=round(x)

⇒ |ℜ|≥|Z|

- S and T have same cardinality:

|S =|T| ⇒ |S|≥|T| ^ |T|≥|S

or

∃ 1-1 correspondence S↔T

- Generalizes finite cardinality:

{1, 2, 3, 4, 5} ≥ {a, b, c}

Infinite Sets

- Infinite set: |S > k ∀k∈Z

or

∃ 1-1 corres. *f*:S↔T, S⊂T

Ex: {p | p prime}, ℜ

- Countable set: |S ≤ N|

Ex: Ø, {p | p prime}, N, Z

- S is strictly smaller than T:

|S < |T| ⇒ |S|≤|T| ^ |S|≠|T|

- Uncountable set: |N| < |S|

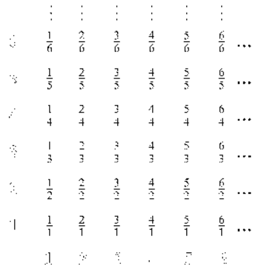
Ex: |N| < ℜ

|N| < [0,1] = {x | x∈ℜ, 0≤x≤1}

Dove-tailing

Thm: ∃ 1-1 correspondence Q↔N

Pf (dove-tailing):



Diagonalization

Thm: |ℜ|> N|

Pf (diagonalization):

Assume ∃ 1-1 corres. *f*: ℜ↔N

Construct X ∈ ℜ:

f(1)=2.718281828... →

f(2)=1.414213562... →

f(3)=1.618033989... →

X = 0. ... ≠ *f*(K) ∀K∈N

⇒ *f* not a 1-1 correspondence

→ contradiction

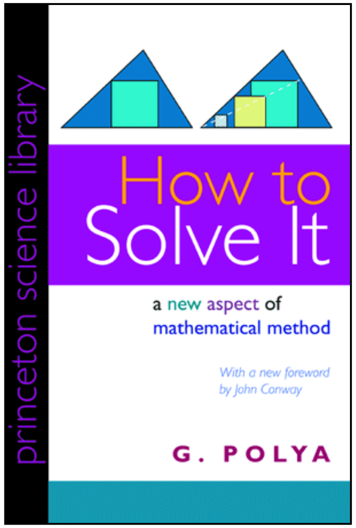
⇒ ℜ is uncountable

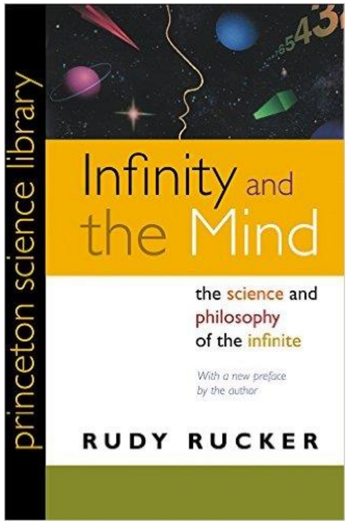
http://www.cs.virginia.edu/robins/cs6161/discrete_math_review_slides.pdf

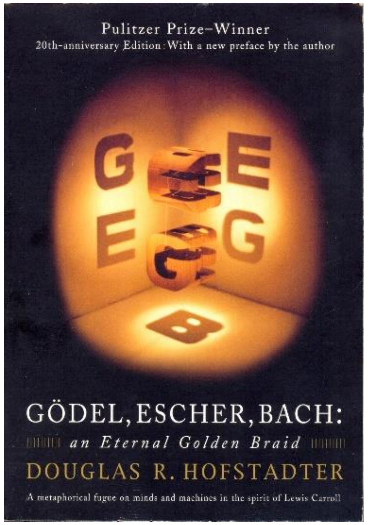
Required Readings

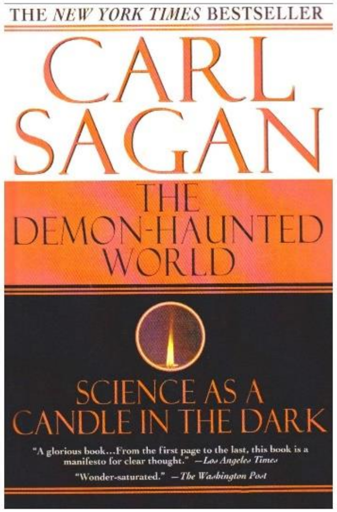
www.cs.virginia.edu/robins/CS_readings.html

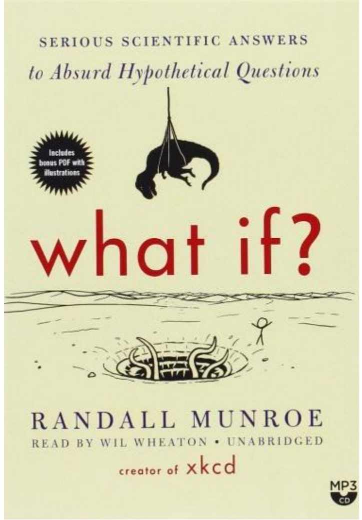
- Required books:
  - "How to Solve It", Polya, 1957
  - "Infinity and the Mind", Rucker, 1995
  - "Godel, Escher, Bach", Hofstadter, 1979
  - "The Demon-Haunted World", Sagan, 2009
  - "What If", Munroe, 2014











Required Readings

www.cs.virginia.edu/robins/CS_readings.html

- 20 required readings, including all red font –marked ones
- Remaining videos / articles / books are “electives”
- Pacing: intended to help you avoid “cramming”
  - at least 2 submissions per week (due 5pm each Monday)
  - at most 2 submissions per day
- Min length: 2 paragraphs per article / video
  2 pages per book
- Books are worth more credit than articles / videos
- Email all submissions to: homework.cs6161@gmail.com
- Additional readings beyond 20 are welcome! (extra credit)

Other "Elective" Readings
www.cs.virginia.edu/robins/CS_readings.html

- Theory and Algorithms:
  - Who Can Name the Bigger Number, Scott Aaronson, 1999
  - The Limits of Reason, Gregory Chaitin, Scientific American, March 2006, pp. 74-81.
  - Breaking Intractability, Joseph Traub and Henryk Wozniakowski, Scientific American, January 1994, pp. 102-107.
  - Confronting Science's Logical Limits, John Casti, Scientific American, October 1996, pp. 102-105.
  - Go Forth and Replicate, Moshe Sipper and James Reggia, Scientific American, August 2001, pp. 34-43.
  - The Science Behind Sudoku, Jean-Paul Delahaye, Scientific American, June 2006, pp. 80-87.
  - The Traveler's Dilemma, Kaushik Basu, Scientific American, June 2007, pp. 90-95.

Other "Elective" Readings

www.cs.virginia.edu/robins/CS_readings.html

- Biological Computing:
  - Computing with DNA, Leonard Adleman, Scientific American, August 1998, pp. 54-61.
  - Bringing DNA Computing to Life, Ehud Shapiro and Yaakov Benenson, Scientific American, May 2006, pp. 44-51.
  - Engineering Life: Building a FAB for Biology, David Baker et al., Scientific American, June 2006, pp. 44-51.
  - Big Lab on a Tiny Chip, Charles Choi, Scientific American, October 2007, pp. 100-103.
  - DNA Computers for Work and Play, Macdonald et al, Scientific American, November 2007, pp. 84-91.

Other “Elective” Readings

www.cs.virginia.edu/robins/CS_readings.html

- Quantum Computing:
  - Quantum Mechanical Computers, Seth Lloyd, Scientific American, 1997, pp. 98-104.
  - Quantum Computing with Molecules, Gershenfeld and Chuang, Scientific American, June 1998, pp. 66-71.
  - Black Hole Computers, Seth Lloyd and Jack Ng, Scientific American, November 2004, pp. 52-61.
  - Computing with Quantum Knots, Graham Collins, Scientific American, April 2006, pp. 56-63.
  - The Limits of Quantum Computers, Scott Aaronson, Scientific American, March 2008, pp. 62-69.
  - Quantum Computing with Ions, Monroe and Wineland, Scientific American, August 2008, pp. 64-71.

Other "Elective" Readings
www.cs.virginia.edu/robins/CS_readings.html

- History of Computing:
  - The Origins of Computing, Campbell-Kelly, Scientific American, September 2009, pp. 62-69.
  - Ada and the First Computer, Eugene Kim and Betty Toole, Scientific American, April 1999, pp. 76-81.
- Security and Privacy:
  - Malware Goes Mobile, Mikko Hypponen, Scientific American, November 2006, pp. 70-77.
  - RFID Powder, Tim Hornyak, Scientific American, February 2008, pp. 68-71.
  - Can Phishing be Foiled, Lorrie Cranor, Scientific American, December 2008, pp. 104-110.

Other "Elective" Readings
www.cs.virginia.edu/robins/CS_readings.html

- Future of Computing:
  - Microprocessors in 2020, David Patterson, Scientific American, September 1995, pp. 62-67.
  - Computing Without Clocks, Ivan Sutherland and Jo Ebergen, Scientific American, August 2002, pp. 62-69.
  - Making Silicon Lase, Bahram Jalali, Scientific American, February 2007, pp. 58-65.
  - A Robot in Every Home, Bill Gates, Scientific Am, January 2007, pp. 58-65.
  - Ballbots, Ralph Hollis, Scientific American, October 2006, pp. 72-77.
  - Dependable Software by Design, Daniel Jackson, Scientific American, June 2006, pp. 68-75.
  - Not Tonight Dear - I Have to Reboot, Charles Choi, Scientific American, March 2008, pp. 94-97.
  - Self-Powered Nanotech, Zhong Lin Wang, Scientific American, January 2008, pp. 82-87.

Other "Elective" Readings
www.cs.virginia.edu/robins/CS_readings.html

- The Web:
  - The Semantic Web in Action, Lee Feigenbaum et al., Scientific American, December 2007, pp. 90-97.
  - Web Science Emerges, Nigel Shadbolt and Tim Berners-Lee, Scientific American, October 2008, pp. 76-81.
- The Wikipedia Computer Science Portal:
  - Theory of computation and Automata theory
  - Formal languages and grammars
  - Chomsky hierarchy and the Complexity Zoo
  - Regular, context-free &Turing-decidable languages
  - Finite & pushdown automata; Turing machines
  - Computational complexity
  - List of data structures and algorithms

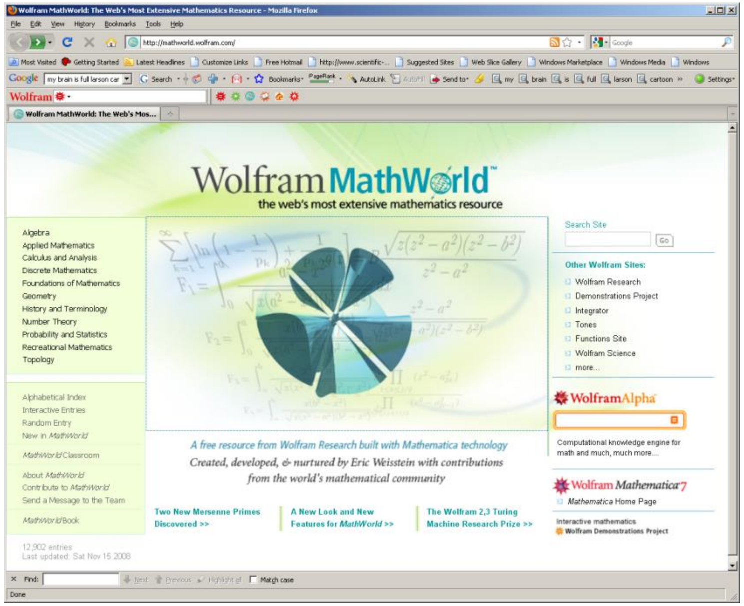
Email all submissions to: homework.cs6161@gmail.com

Other “Elective” Readings

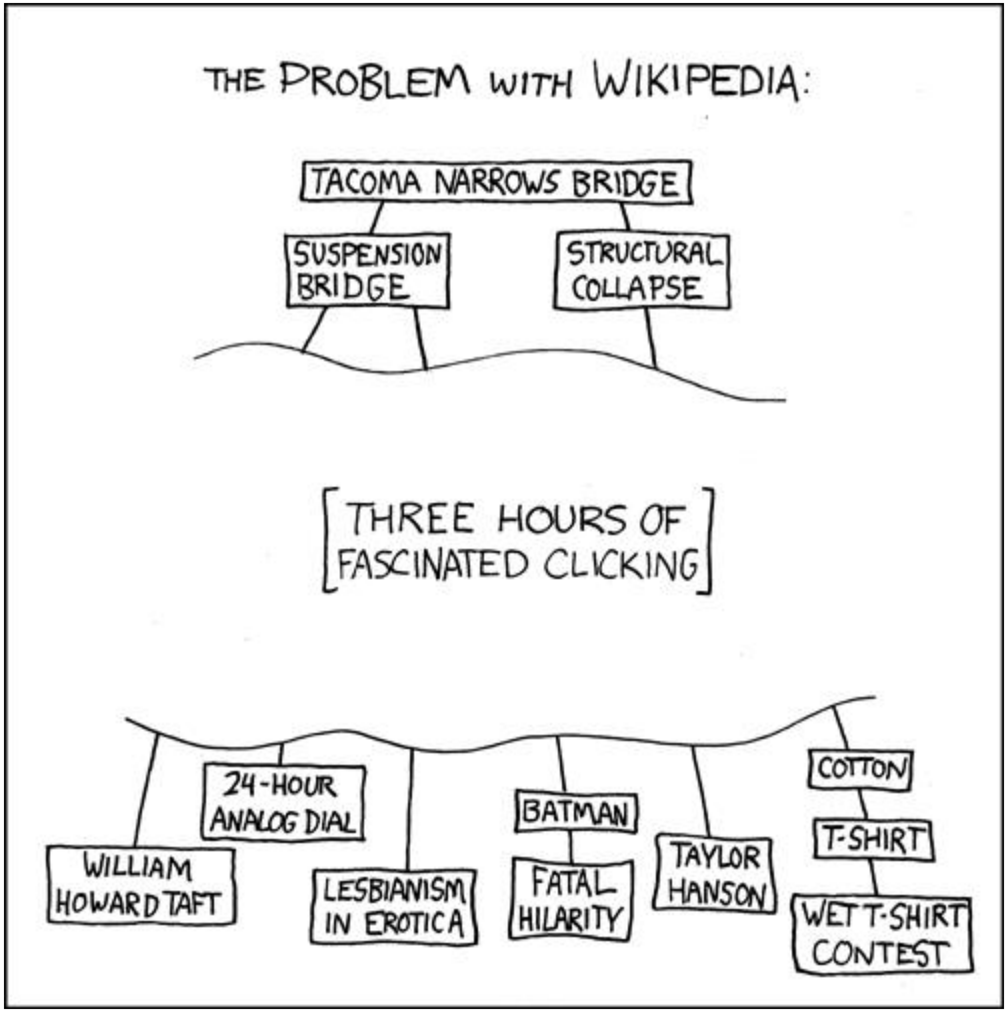
www.cs.virginia.edu/robins/CS_readings.html

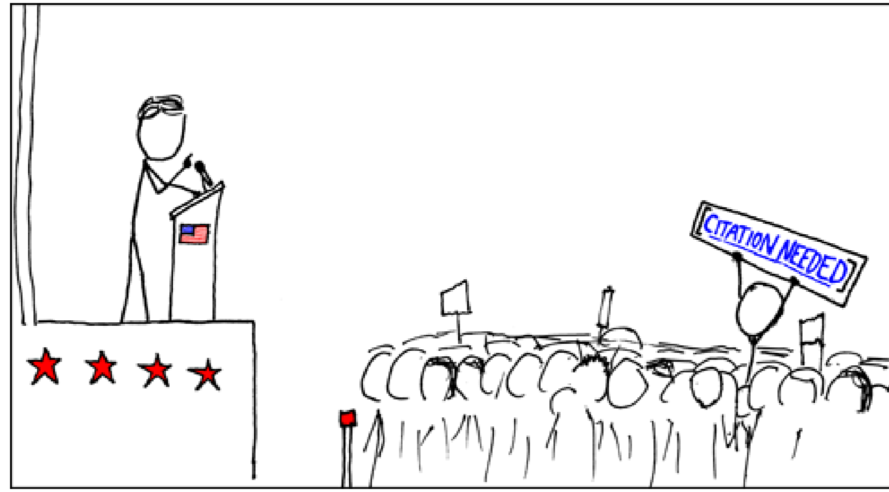
- The Wikipedia Math Portal:
  - Problem solving
  - List of Mathematical lists
  - Sets and Infinity
  - Discrete mathematics
  - Proof techniques and list of proofs
  - Information theory & randomness
  - Game theory
- Mathematica's “Math World”

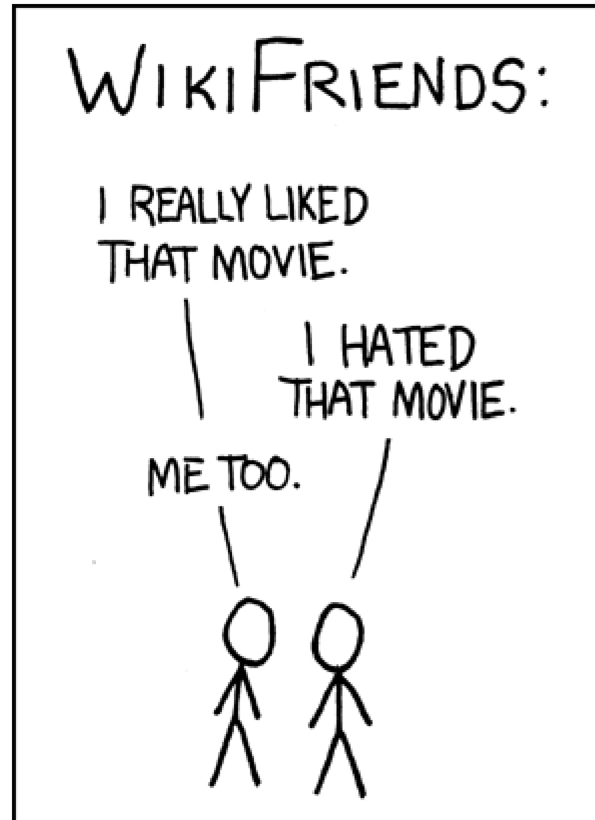
WIKIPEDIA
The Free Encyclopedia



Email all submissions to: homework.cs6161@gmail.com

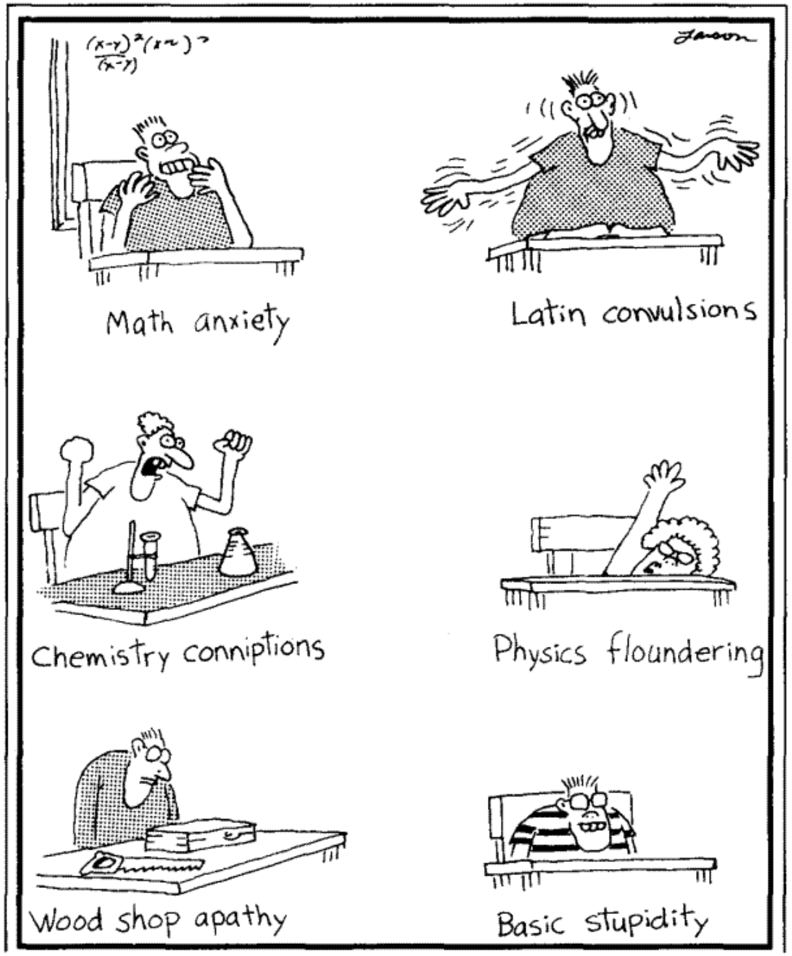

THE PROBLEM WITH WIKIPEDIA:
TACOMA NARROWS BRIDGE
SUSPENSION BRIDGE
STRUCTURAL COLLAPSE
[THREE HOURS OF FASCINATED CLICKING]
WILLIAM HOWARD TAFT
24-HOUR ANALOG DIAL
LESBIANISM IN EROTICA
BATMAN
FATAL HILARITY
TAYLOR HANSON
COTTON
T-SHIRT
WET T-SHIRT CONTEST


[CITATION NEEDED]


WIKIFRIENDS:
I REALLY LIKED THAT MOVIE.
I HATED THAT MOVIE.
ME TOO.

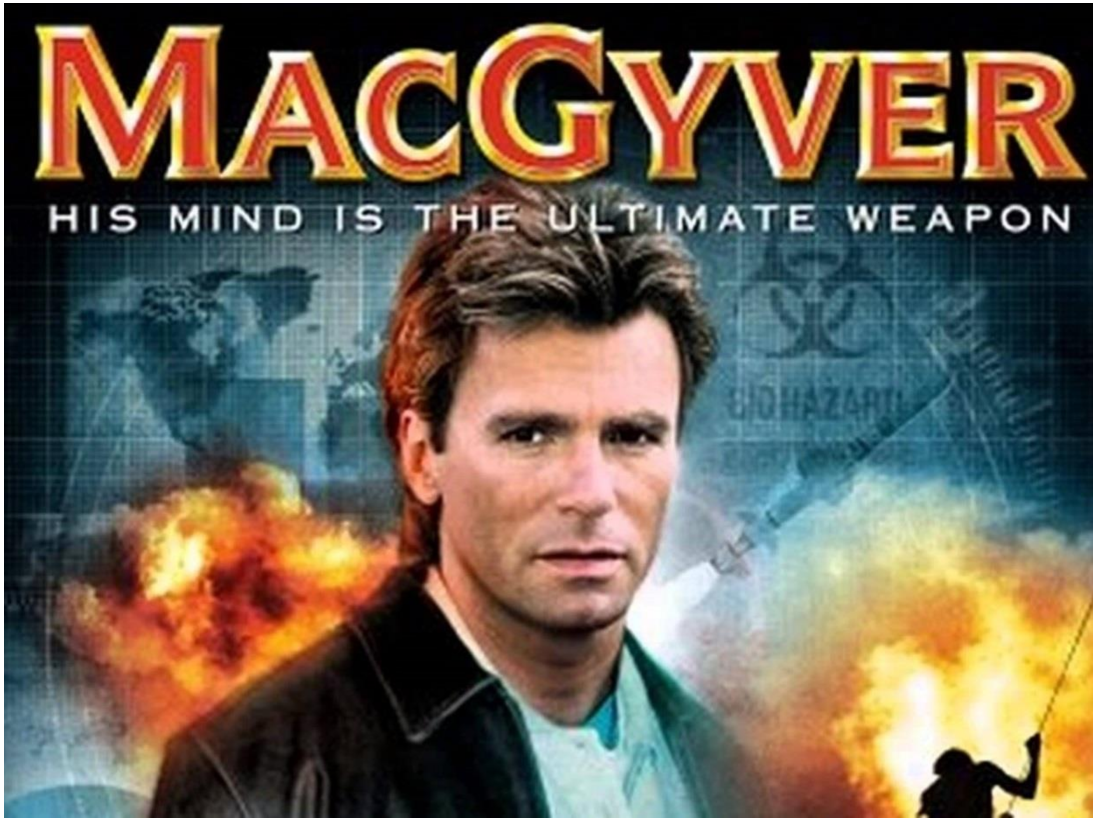
Good Advice

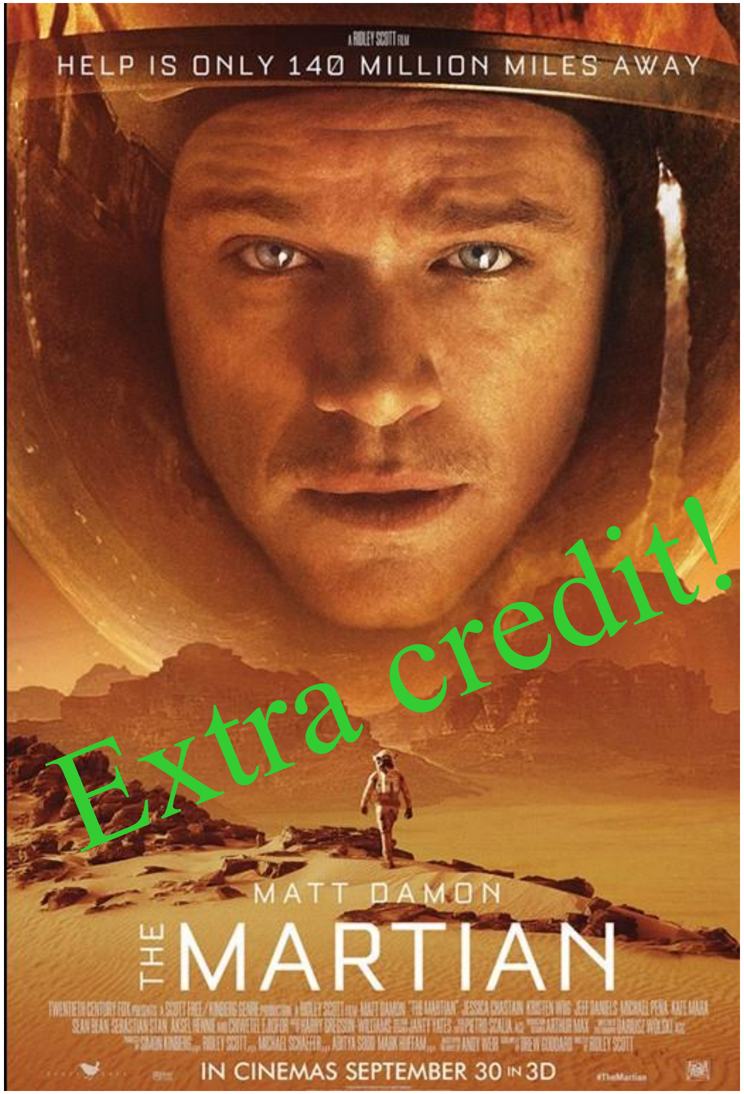
- Ask questions ASAP
- Solve problems ASAP
- Work in study groups
- Do not fall behind
- "Cramming" won't work
- Do lots of extra credit
- Attend every lecture
- Visit class Website often
- Solve lots of problems



Classroom afflictions

Goal: Become a more effective problem solver!

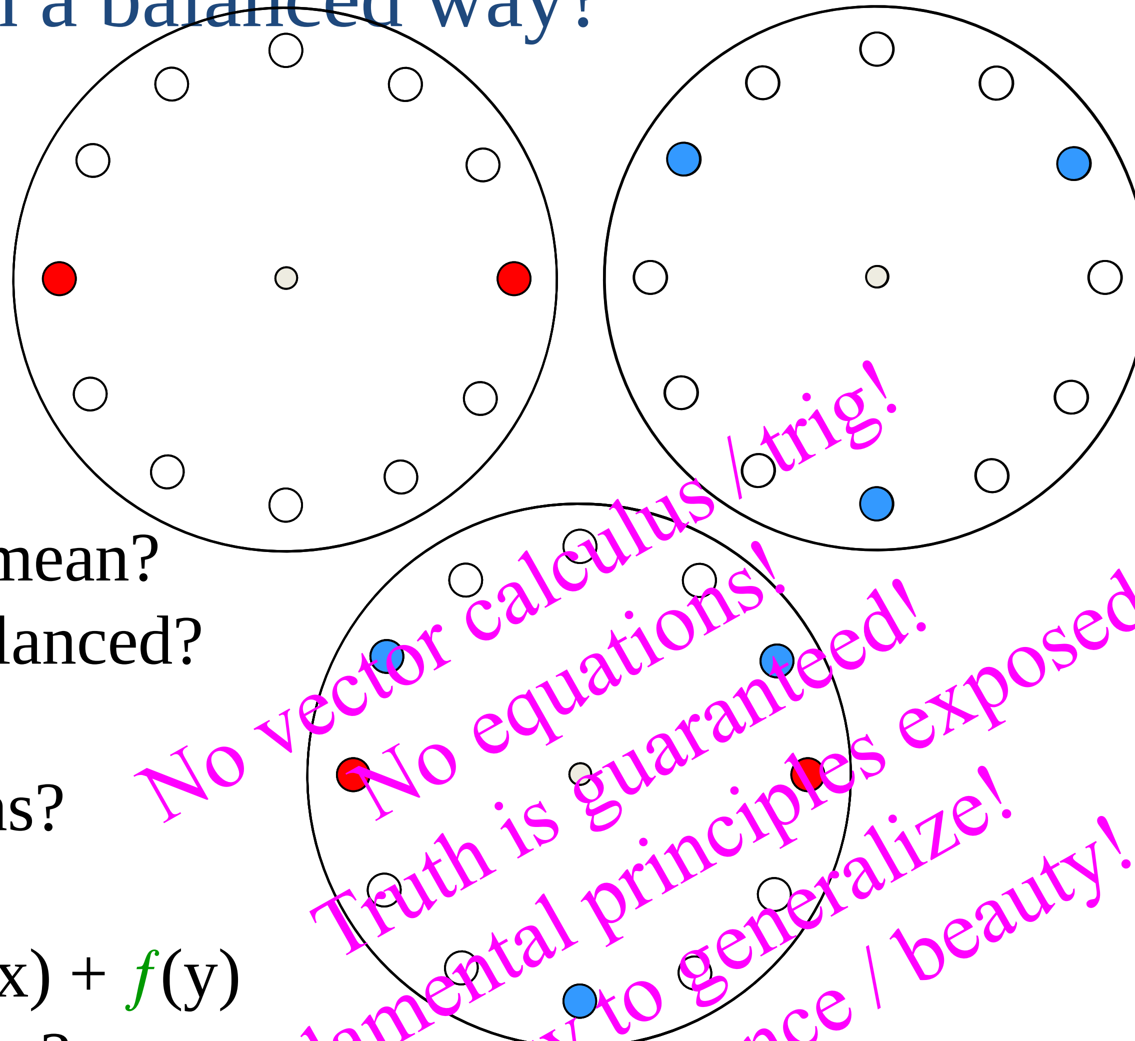




Extra credit!

Email all submissions to: homework.cs6161@gmail.com

Problem: Can 5 test tubes be spun simultaneously in a 12-hole centrifuge in a balanced way?



- What does "balanced" mean?
- Why are 3 test tubes balanced?
- Symmetry!
- Can you merge solutions?
- Superposition!
- Linearity! $f(x + y) = f(x) + f(y)$
- Can you spin 7 test tubes?
- Complementarity!
- Empirical testing…

No vector calculus / trig!
No equations!
Truth is guaranteed!
Fundamental principles exposed!
Easy to generalize!
High elegance / beauty!

Problem: 1 + 2 + 3 + 4 + …+ 100 = ?

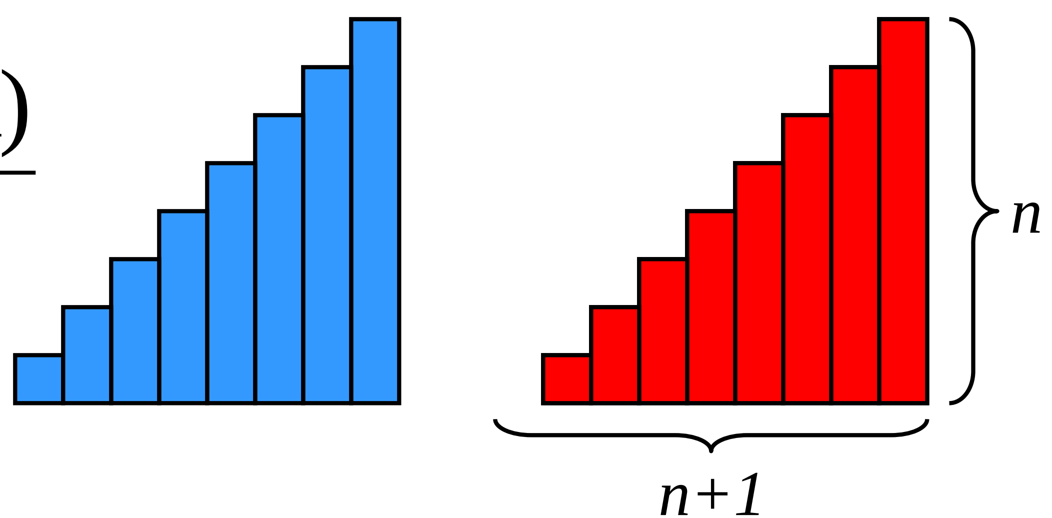
= (100*101)/2

= 5050

Proof: ~~Induction~~…

1	+	2	+	3	+	…	+	99	+	100	
100	+	99	+	98	+	…	+	2	+	1	
101	+	101	+	101	+	…	+	101	+	101	= 100*101

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$



Drawbacks of Induction

- You must a priori know the formula / result
- Easy to make mistakes in inductive proof
- Mostly "mechanical" – ignores intuitions
- Tedious to construct
- Difficult to check
- Hard to understand
- Not very convincing
- Generalizations not obvious
- Does not "shed light on truth"
- Obfuscates connections

Conclusion: only use induction as a last resort! (i.e., rarely)

Problem: $(1/4) + (1/4)^2 + (1/4)^3 + (1/4)^4 + \ldots = ?$

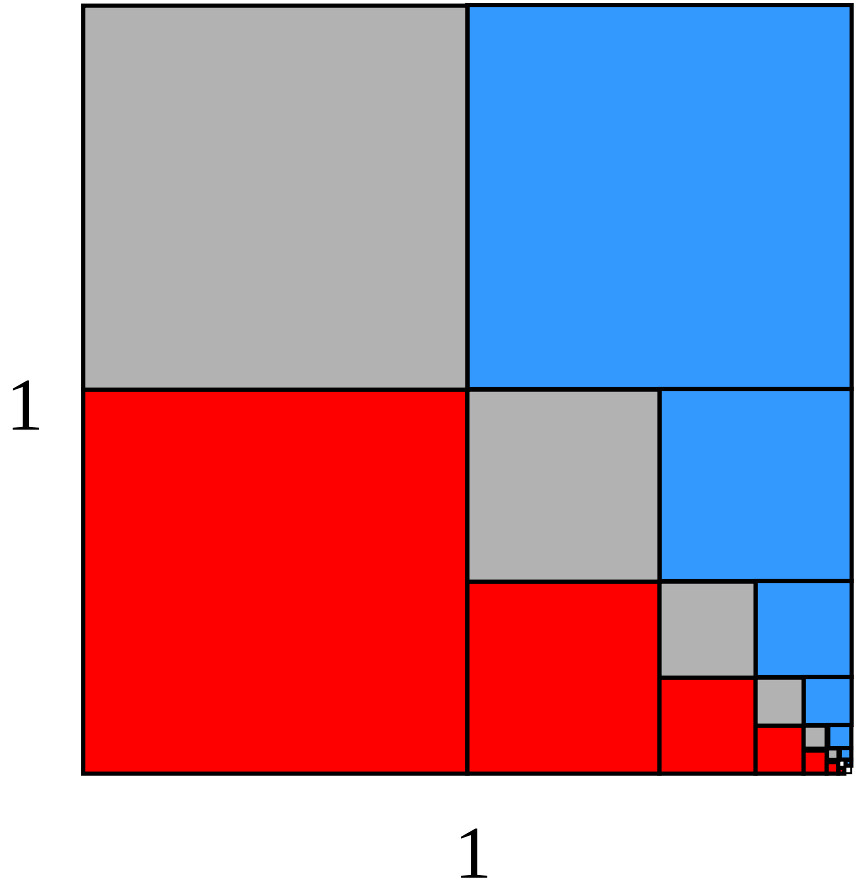
$$\sum_{i=1}^{\infty} \frac{1}{4^i} = ?$$

Extra Credit:

Find a short, geometric, induction-free proof.

Problem: $(1/4) + (1/4)^2 + (1/4)^3 + (1/4)^4 + \ldots = ?$

Find a short, geometric, induction-free proof.



$$\sum_{i=1}^{\infty} \frac{1}{4^i} = \frac{1}{3}$$

Problem: $(1/8) + (1/8)^2 + (1/8)^3 + (1/8)^4 + \ldots = ?$

$$\sum_{i=1}^{\infty} \frac{1}{8^i} = ?$$

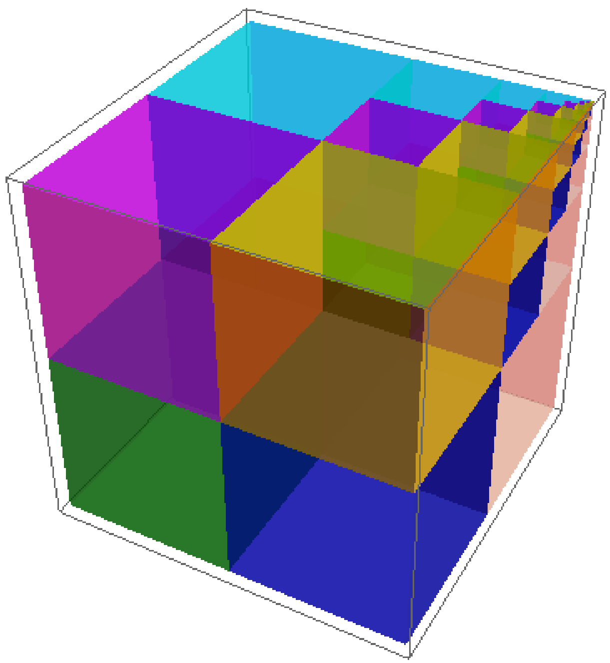
Extra Credit:

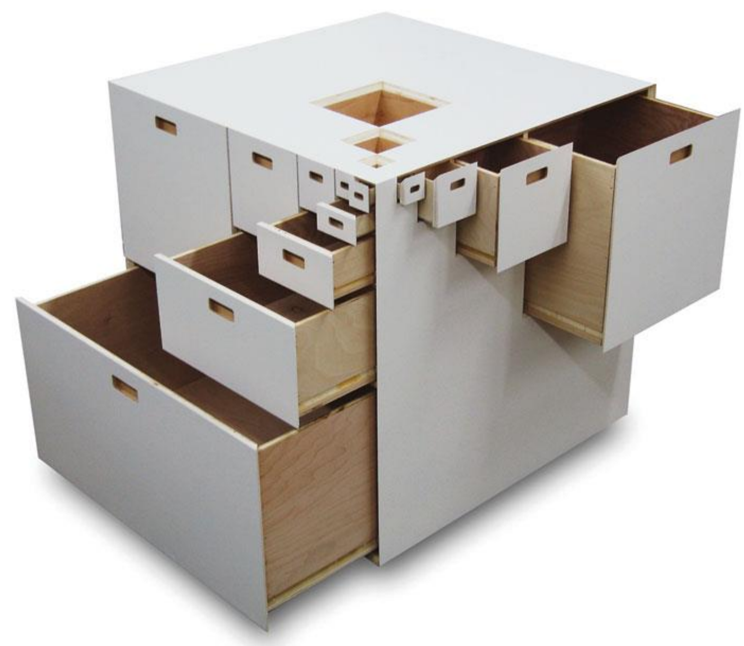
Find a short, geometric, induction-free proof.

Problem: $(1/8) + (1/8)^2 + (1/8)^3 + (1/8)^4 + \ldots = ?$

Find a short, geometric, induction-free proof.

$$\sum_{i=1}^{\infty} \frac{1}{8^i} = \frac{1}{7}$$



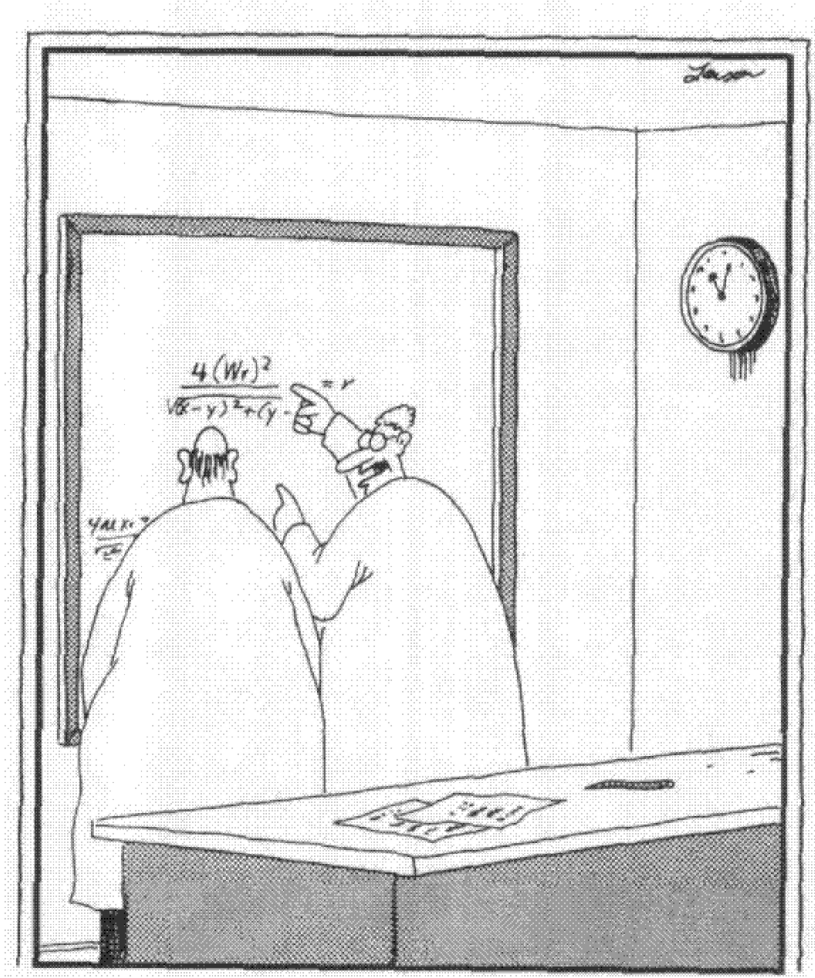


Problem: $1^3 + 2^3 + 3^3 + 4^3 + \ldots + n^3 = ?$

$$\sum_{i=1}^{n} i^3 = ?$$

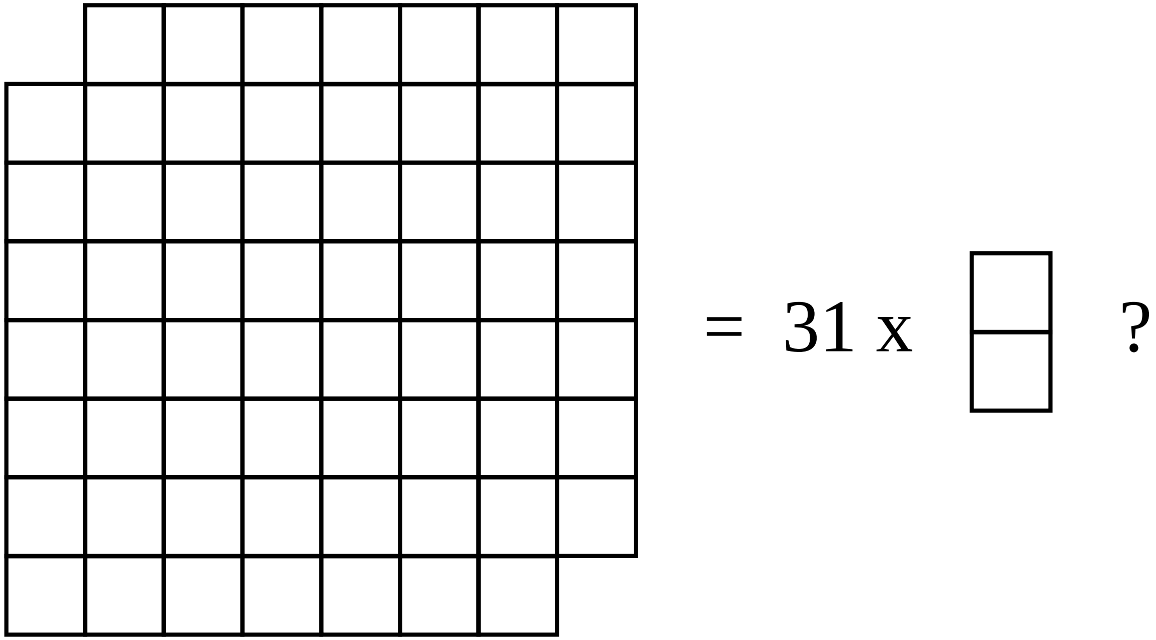
Extra Credit:

find a short, geometric, induction-free proof.



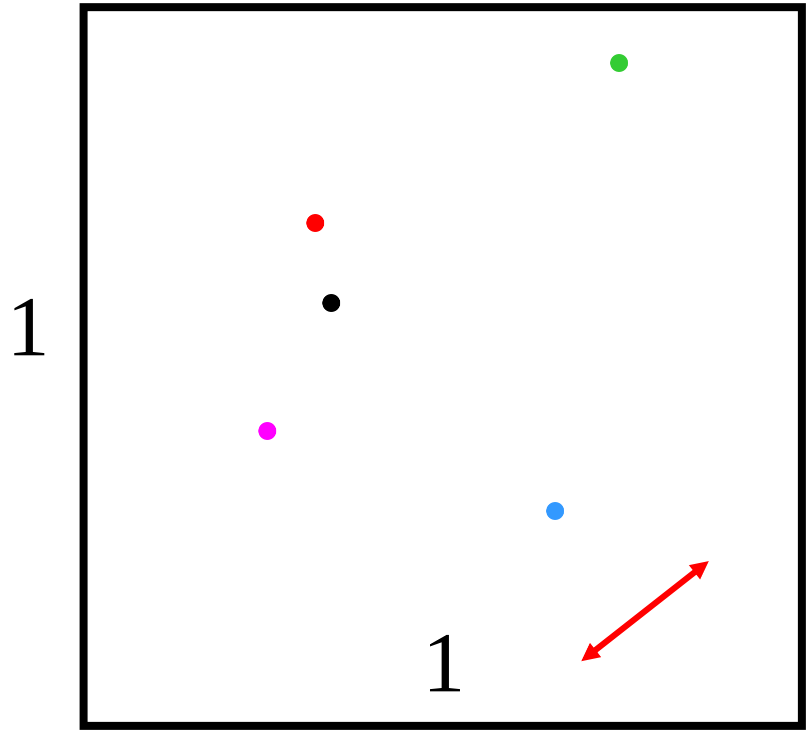
"Yes, yes, I *know* that, Sidney ... everybody knows *that*! ... But look: Four wrongs *squared*, minus two wrongs to the fourth power, divided by this formula, *do* make a right."

Problem: Can an 8x8 board with two opposite corners missing be tiles with 31 dominoes?

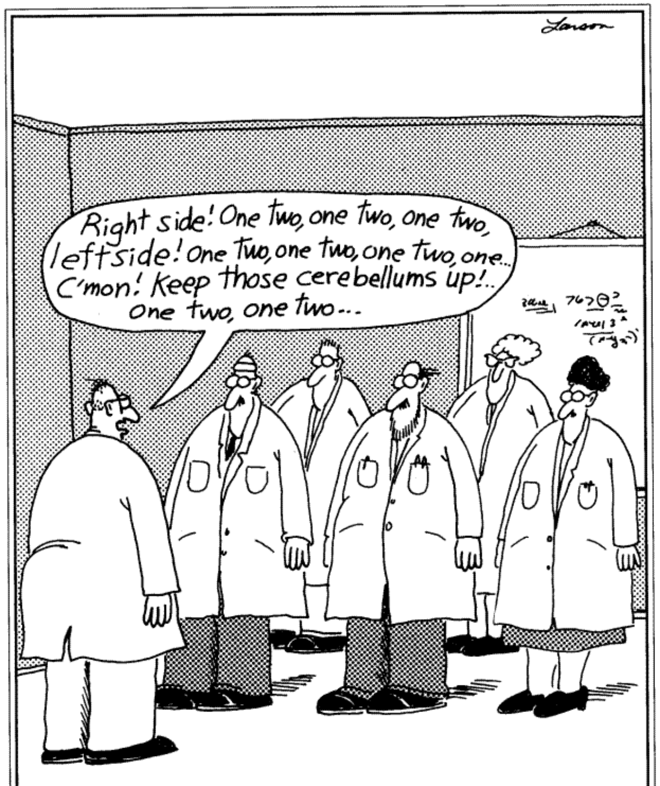


- What approaches fail?
- What techniques work and why?
- Lessons and generalizations

Problem: Given any five points in/on the unit square, is there always a pair with distance $\leq \frac{1}{\sqrt{2}}$?

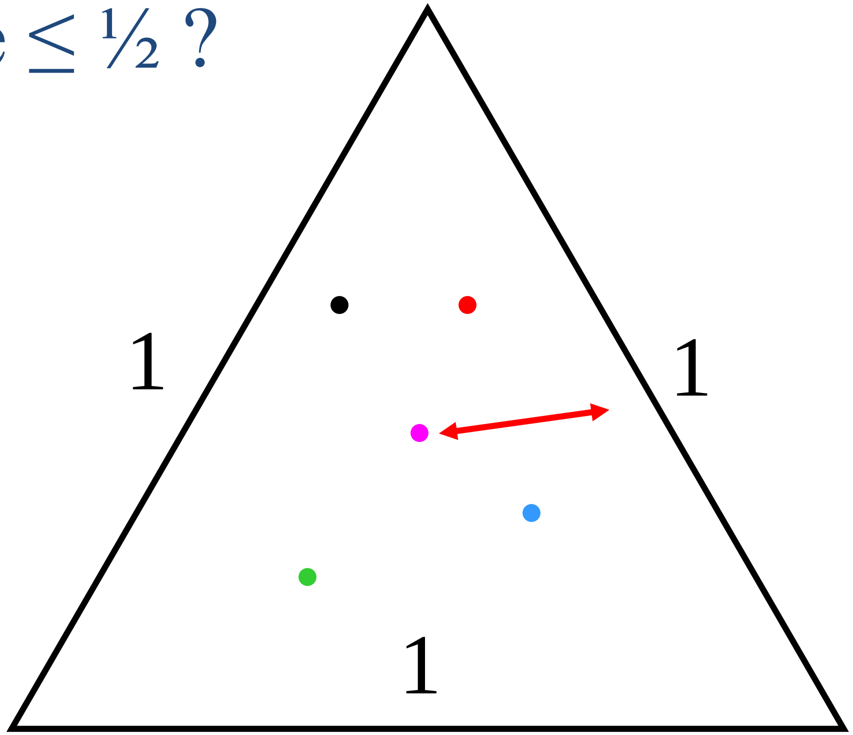


- What approaches fail?
- What techniques work and why?
- Lessons and generalizations

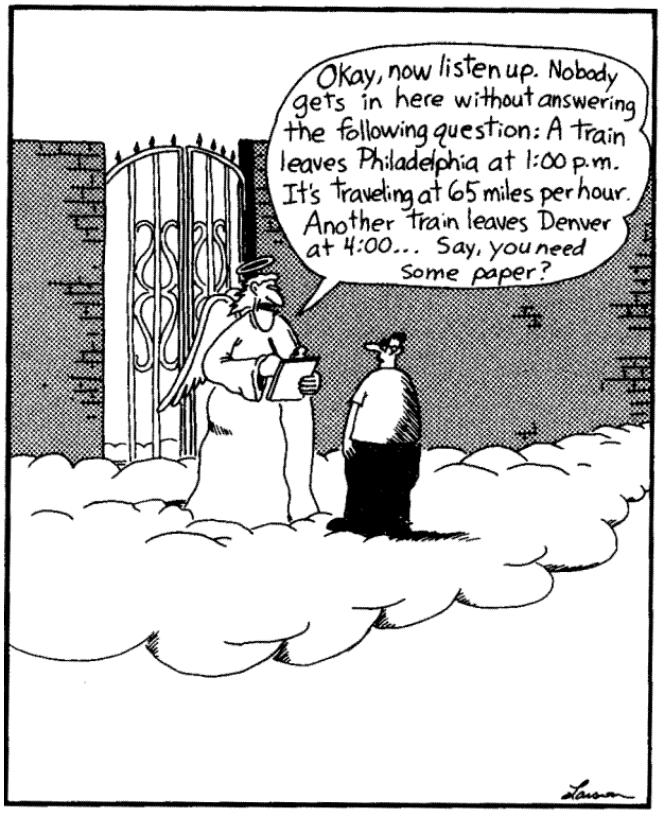


Brain aerobics

Problem: Given any five points in/on the unit equilateral triangle, is there always a pair with distance ≤ ½ ?



- What approaches fail?
- What techniques work and why?
- Lessons and generalizations



Math phobic's nightmare

Problem: Prove that there are an infinity of primes.

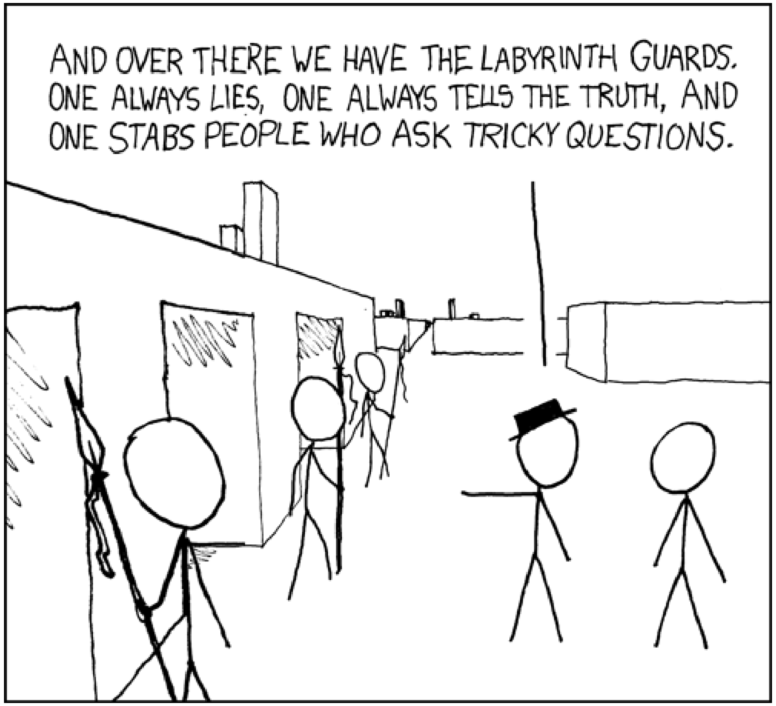
Extra Credit: Find a short, induction-free proof.

- What approaches fail?
- What techniques work and why?
- Lessons and generalizations

Problem: True or false: there are arbitrary long blocks of consecutive composite integers (i.e., big "prime deserts")

Extra Credit: find a short, induction-free proof.

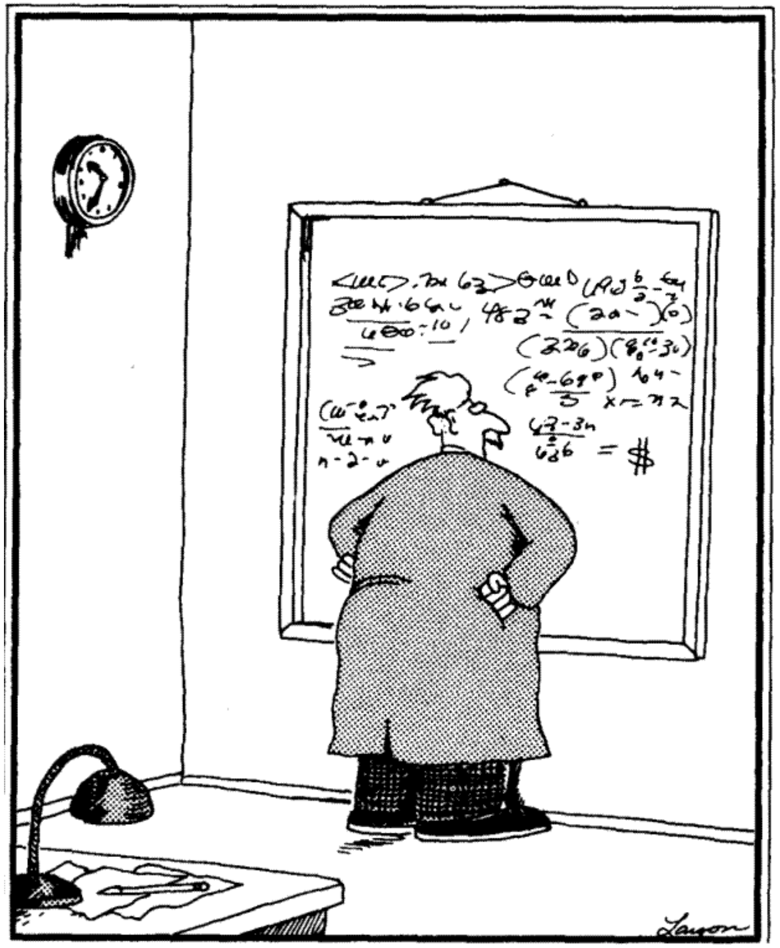
- What approaches fail?
- What techniques work and why?
- Lessons and generalizations



Problem: Prove that $\sqrt{2}$ is irrational.

Extra Credit: find a short, induction-free proof.

- What approaches fail?
- What techniques work and why?
- Lessons and generalizations

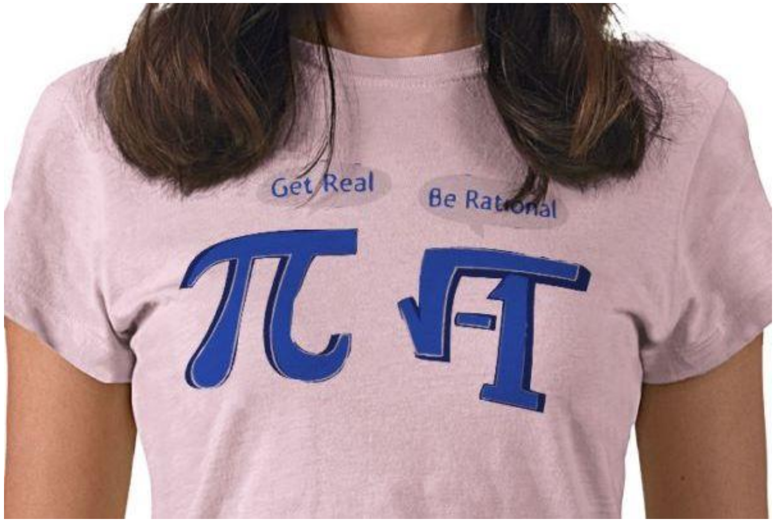


Einstein discovers that time is actually money.

Problem: Does exponentiation preserve irrationality?
i.e., are there two irrational numbers x and y such that x^y is rational?

Extra Credit: find a short, induction-free proof.

- What approaches fail?
- What techniques work and why?
- Lessons and generalizations

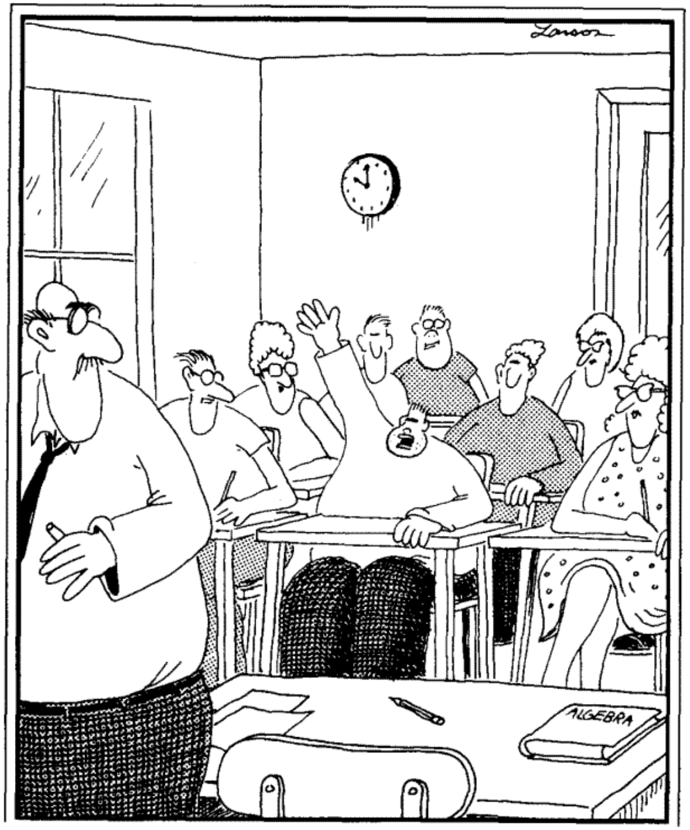


Problem: Solve the following equation for X:

$$X^{X^{X^{X^{\cdot^{\cdot^{\cdot}}}}}} = 2$$

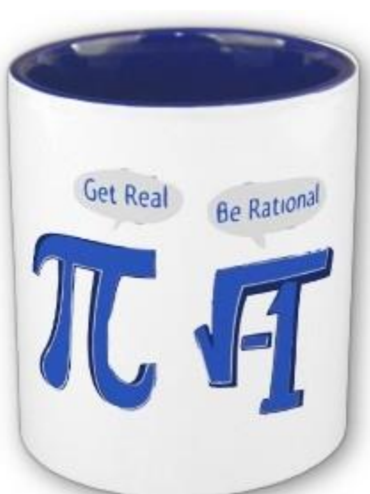
where the stack of exponentiated x's extends forever.

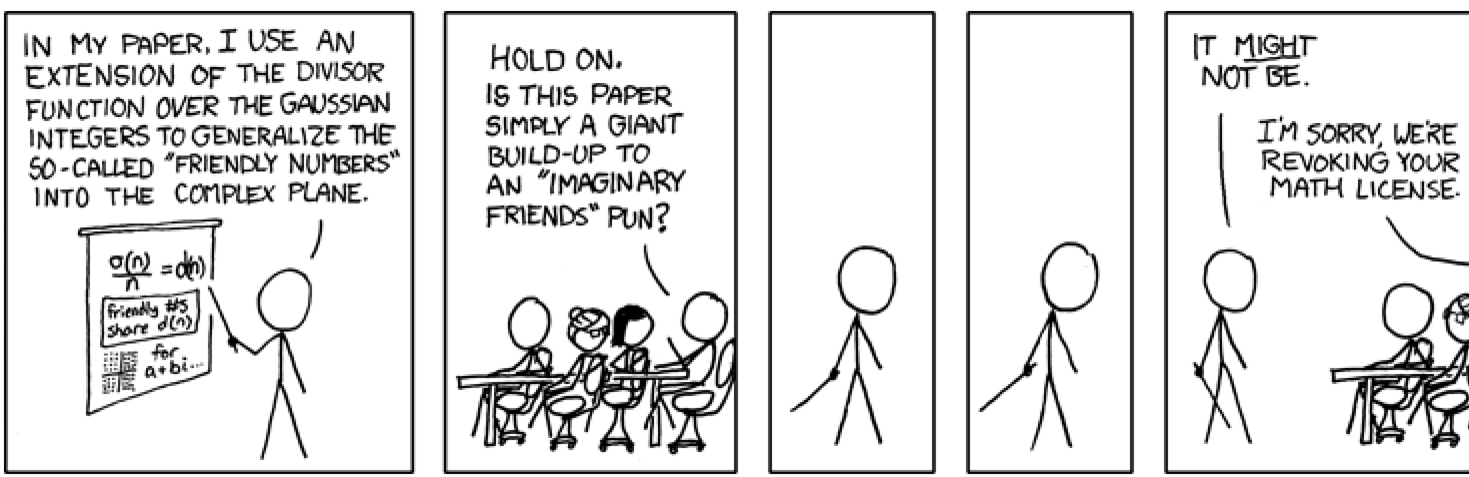
- What approaches fail?
- What techniques work and why?
- Lessons and generalizations



"Mr. Osborne, may I be excused? My brain is full."

Problem: Are the complex numbers closed under exponentiation ? E.g., what is the value of i^i?





Theorem [Turing]: not all problems are solvable by algorithms.
Theorem: not all functions are computable by algorithms.
Theorem: not all Boolean functions are computable by algorithms.
Theorem: most Boolean functions are not computable!

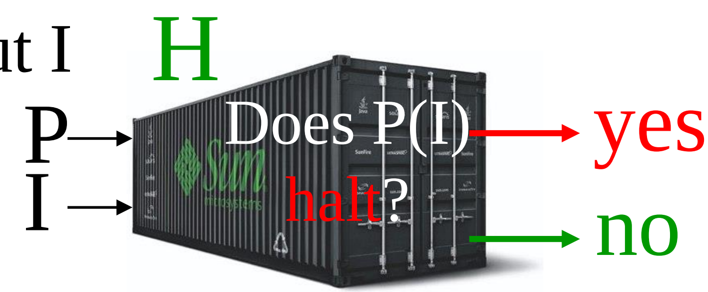
Q: Can we find a concrete example of an uncomputable function?
A: [Turing] Yes, for example, the Halting Problem.

Definition: The Halting problem: given a program P and input I, will P ever halt if we ran it on I?

Define $H:\mathbb{N}\times\mathbb{N}\rightarrow\{0,1\}$

$H(P,I)=1$ if program P halts on input I

$H(P,I)=0$ otherwise



- Both P and I can be encoded as strings
- P and I can also be encoded as integers (in some canonical order)
- H is an everywhere-defined Boolean function on natural #'s

Theorem [Turing]: the halting problem (H) is not computable.

Ex: the "3X+1" problem (the Ulam conjecture):

- Start with any integer X>0
- If X is even, then replace it with X/2
- If X is odd then replace it with 3X+1
- Repeat until X=1 (i.e., short cycle 4, 2, 1, ...)

Ex: 26 terminates after 10 steps

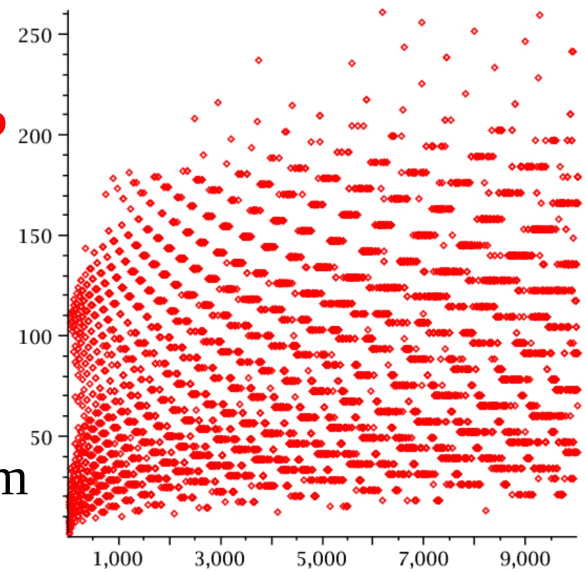
27 terminates after 111 steps

Termination verified for $X<10^{18}$

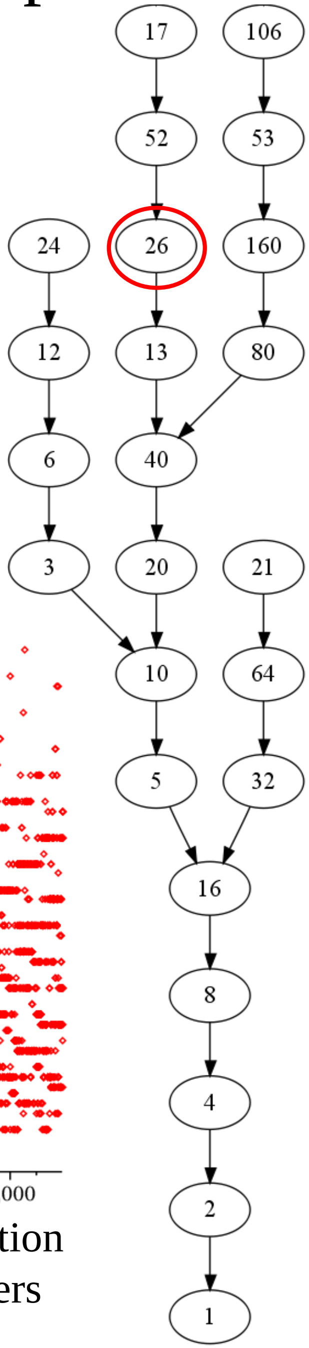
Q: Does this terminate for every X>0 ?

A: Open since 1937!

"Mathematics is not yet ready for such confusing, troubling, and hard problems." - Paul Erdős, who offered a $500 bounty for a solution to this problem



Number of steps to termination for the first 10,000 numbers



Observation: termination is in general difficult to detect!

Theorem [Turing]: the halting problem (H) is not computable.

Corollary: we can not algorithmically detect all infinite loops.

Q: Why not? E.g., do the following programs halt?

```
main()
{ int k=3; }
```

Halts!

```
main()
{ while(1) {} }
```

Runs forever!

Windows Vista

?

main()
{ Find a Fermat
triple $a^n+b^n=c^n$
with $n>2$ then stop}

Runs forever!

Open from 1637-1995!

```
main()
{ Find a Goldbach
  integer that is not a sum
  of two primes & stop}
```

?

Still open since 1742!

Theorem: solving the halting problem is at least as hard as solving arbitrary open mathematical problems!

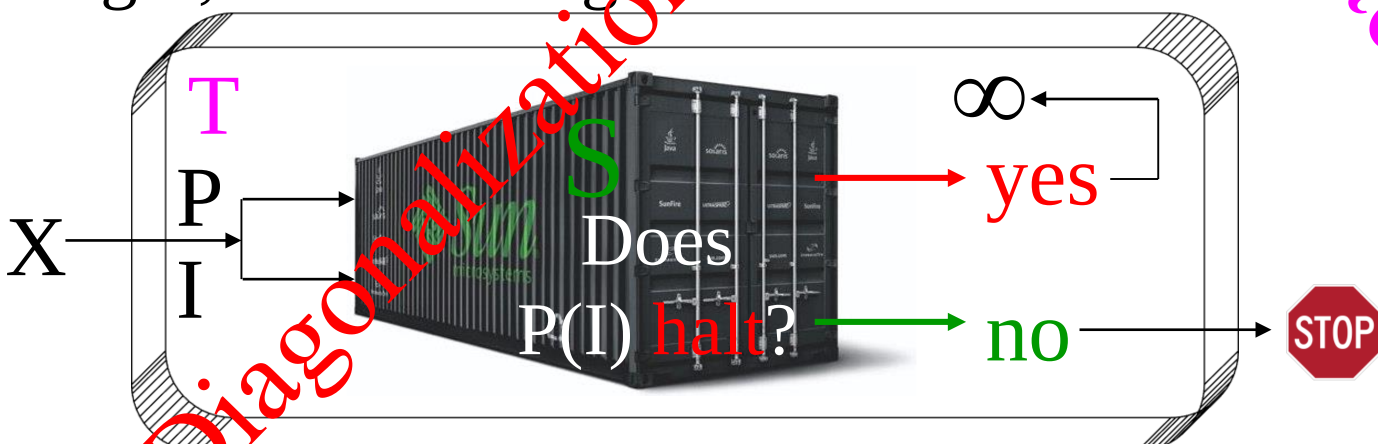
Corollary: Its not about size!

Theorem [Turing]: the halting problem (H) is not computable.

Proof: Assume ∃ algorithm S that solves the halting problem H, that always stops with the correct answer for any P & I.

Non-existence proof!

Using S, construct algorithm / TM T:



Diagonalization

T(T) halts ⇒ T(T) does not halt

T(T) does not halt ⇒ T(T) halts

} Q ⇔ ~Q ⇒ Contradiction!

⇒ S cannot exist! (at least as an algorithm / program / TM)

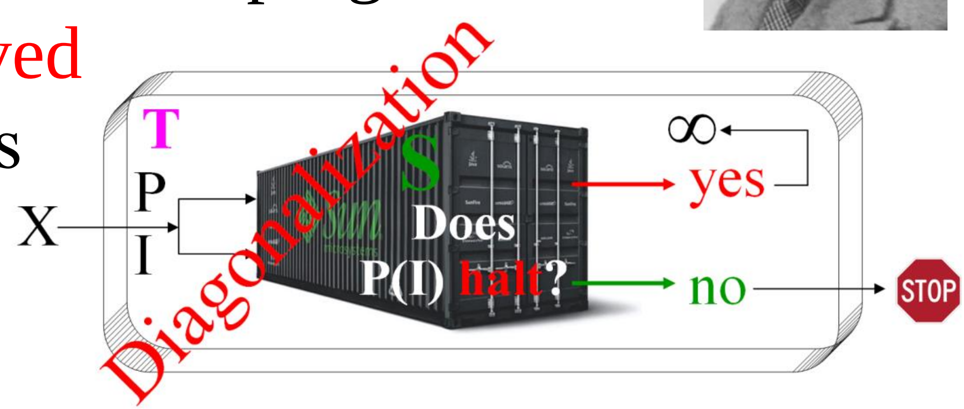
Theorem: all computable numbers are finitely describable.
Proof: A computable number can be outputted by a TM.
A TM is a (unique) finite description.

What the unsolvability of the Halting Problem means:

There is no single algorithm / program / TM that correctly solves all instances of the halting problem in finite time each.

This result does not necessarily apply if we allow:

- Incorrectness on some instances
- Infinitely large algorithm / program
- Infinite number of finite algorithms / programs
- Some instances to not be solved
- Infinite "running time" / steps
- Powerful enough oracles



Q: When do we want to feed a program to itself in practice?

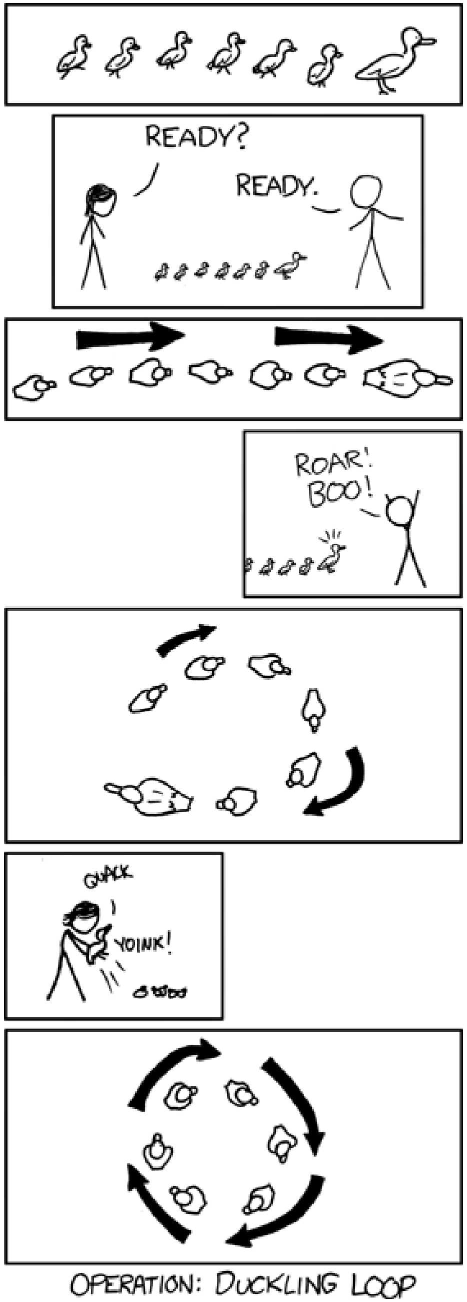
A: When we build compilers.

Q: Why?

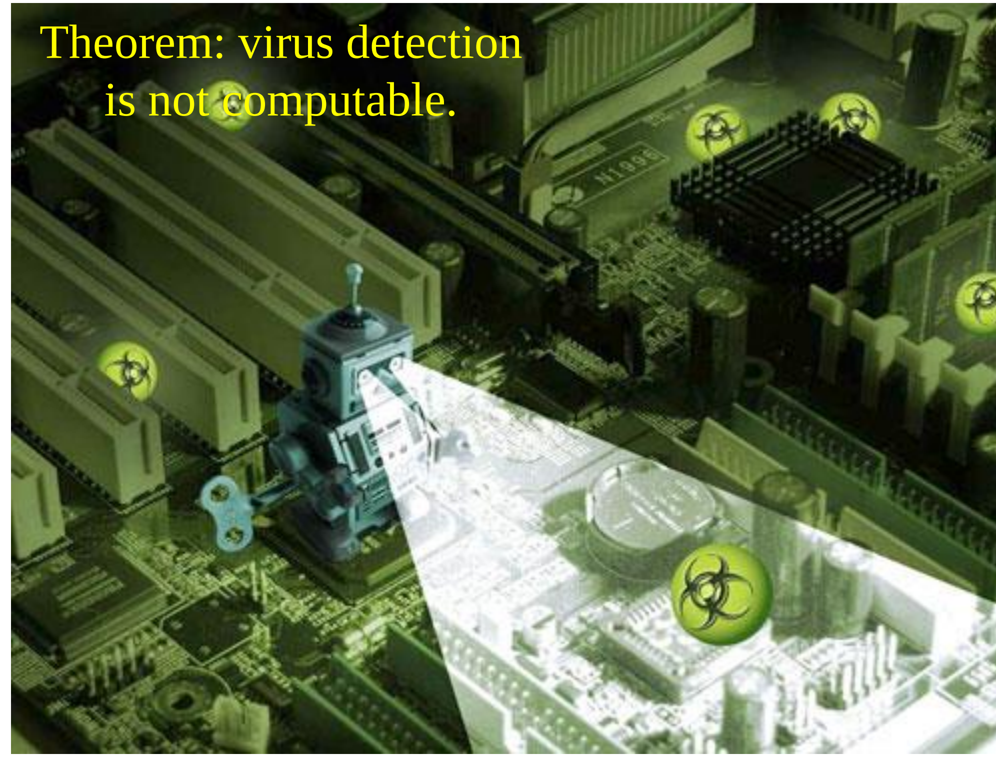
A: To make them more efficient!

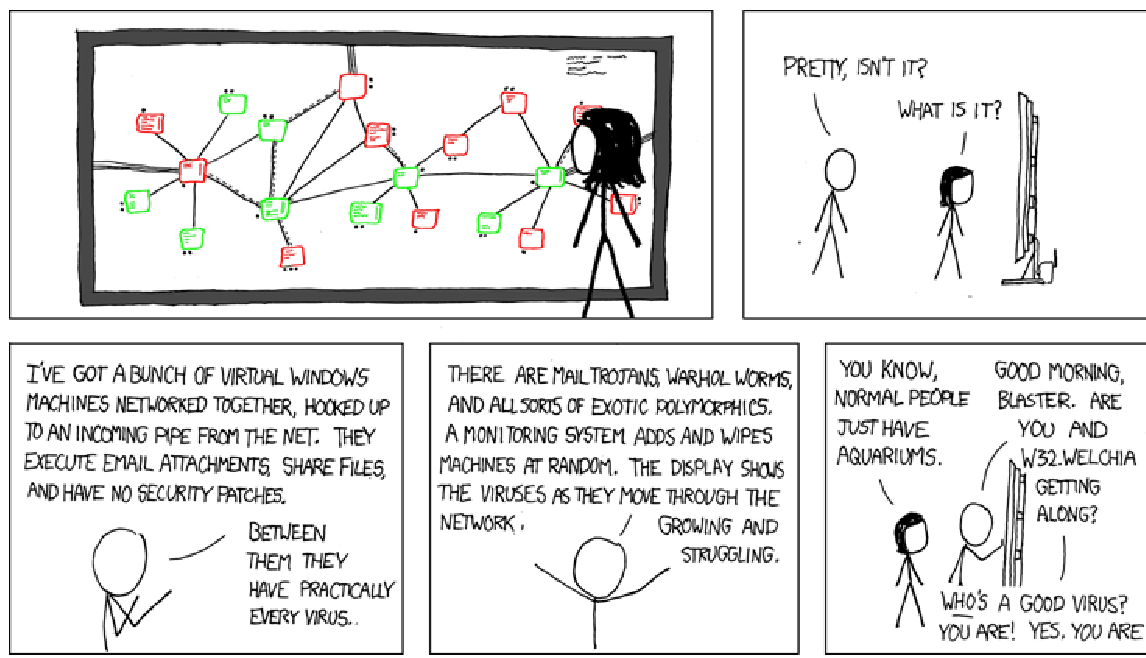
To boot-strap the coding in the compiler's own language!

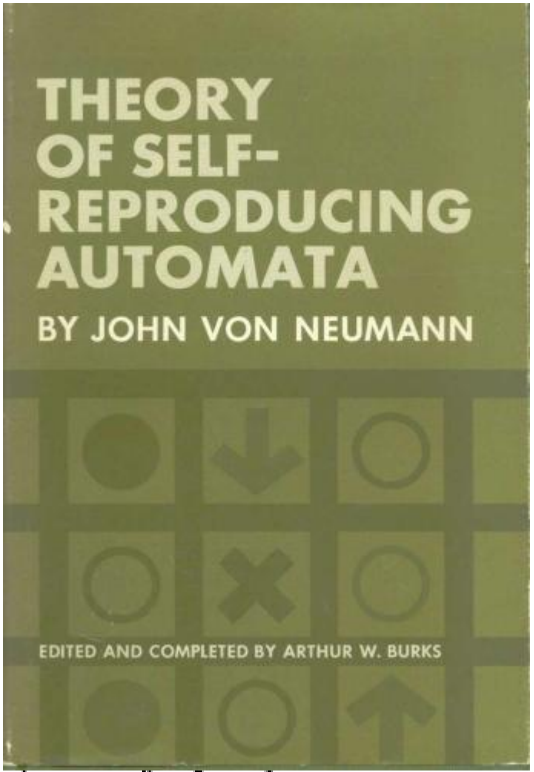
Theorem: Infinite loop detection is not computable.



Theorem: virus detection is not computable.

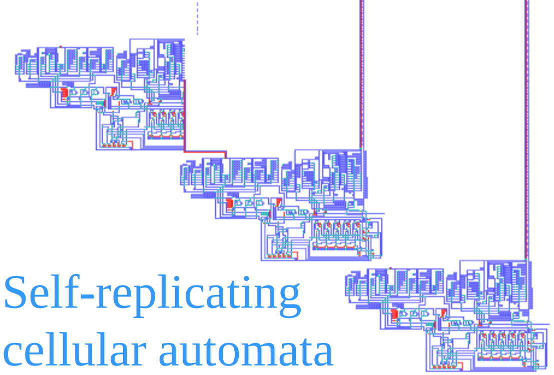




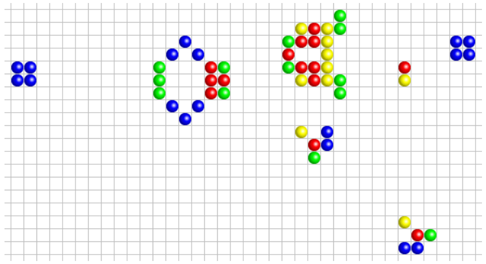


Self-Replication

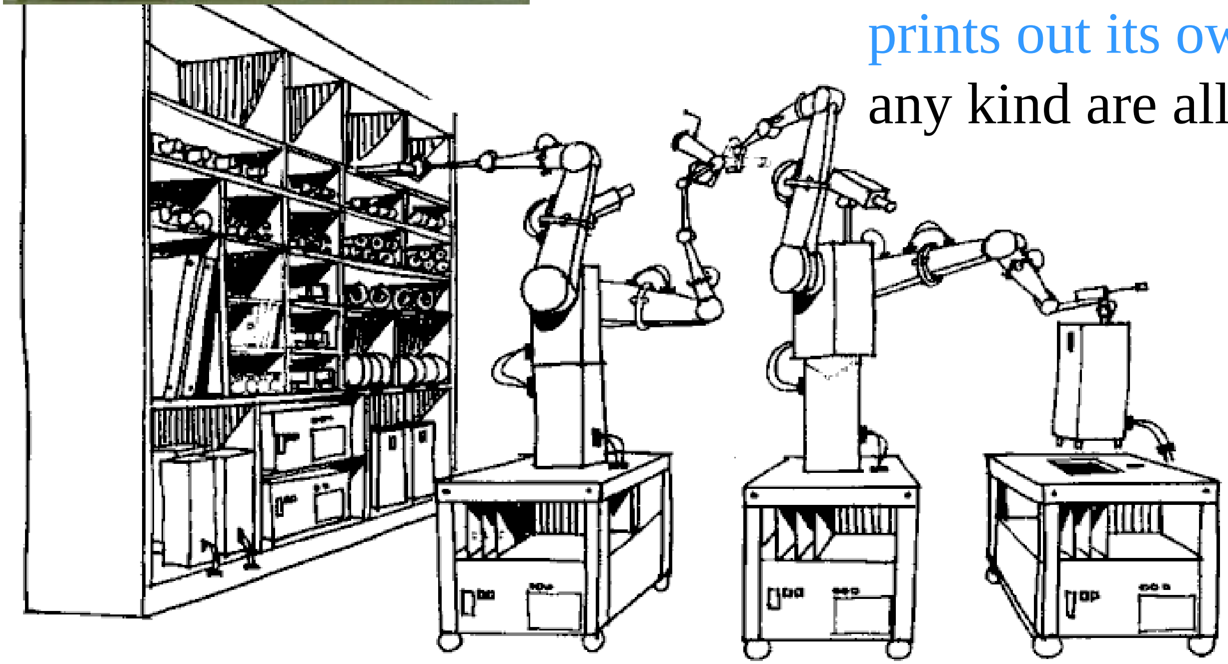
- Biology / DNA
- Nanotechnology
- Computer viruses
- Space exploration
- Memetics / memes
- “Gray goo”



Self-replicating cellular automata designed by von Neumann



Problem (extra credit): write a program that prints out its own source code (no inputs of any kind are allowed).



Go Forth and Replicate

Birds do it, bees do it, but could machines do it? New computer simulations suggest that the answer is yes

By Moshe Sipper and James A. Reggia

Photoillustrations by David Emmite

Apples beget apples, but can machines beget machines? Today it takes an elaborate manufacturing apparatus to build even a simple machine. Could we endow an artificial device with the ability to multiply on its own? Self-replication has long been considered one of the fundamental properties separating the living from the nonliving. Historically our limited understanding of how biological reproduction works has given it an aura of mystery and made it seem unlikely that it would ever be done by a man-made object. It is reported that when René Descartes averred to Queen Christina of Sweden that animals were just another form of mechanical automata, Her Majesty pointed to a clock and said, "See to it that it produces offspring."

The problem of machine self-replication moved from philosophy into the realm of science and engineering in the late 1940s with the work of eminent mathematician and physicist John von Neumann. Some researchers have actually constructed physical replicators. Forty years ago, for example, geneticist Lionel Penrose and his son, Roger (the famous physicist), built small assemblies of plywood that exhibited a simple form of self-replication [see "Self-Reproducing Machines," by Lionel Penrose; SCIENTIFIC AMERICAN, June 1959]. But self-replication has proved to be so difficult that most researchers study it with the conceptual tool that von Neumann developed: two-dimensional cellular automata.

Implemented on a computer, cellular automata can simulate a huge variety of self-replicators in what amount to austere universes with different laws of physics from our own. Such models free researchers from having to worry about logistical issues such as energy and physical construction so that they can focus on the fundamental questions of information flow. How is a living being able to replicate unaided, whereas mechanical objects must be constructed by humans? How does replication at the level of an organism emerge from the numerous interactions in tissues, cells and molecules? How did Darwinian evolution give rise to self-replicating organisms?

The emerging answers have inspired the development of self-repairing silicon chips [*see box on page 40*] and autocatalyzing molecules [see "Synthetic Self-Replicating Molecules," by Julius Rebek, Jr.; SCIENTIFIC AMERICAN, July 1994]. And this may be just the beginning. Researchers in the field of nanotechnology have long proposed that self-replication will be crucial to manufacturing molecular-scale machines, and proponents of space exploration see a macroscopic version of the process as a way to colonize planets using in situ materials. Recent advances have given credence to these futuristic-sounding ideas. As with other scientific disciplines, including genetics, nuclear energy and chemistry, those of us who study self-replication face the twofold challenge of creating replicating machines and avoiding dystopian predictions of devices running amok. The knowledge we gain will help us separate good technologies from destructive ones.

Playing Life

SCIENCE-FICTION STORIES often depict cybernetic self-replication as a natural development of current technology, but they gloss over the profound problem it poses: how to avoid an infinite regress. A system might try to build a clone using a blueprint—that is, a self-description. Yet the self-description is part of the machine, is it not? If so, what describes the description? And what describes the description of the description? Self-replication in this case would be like asking an architect to make a perfect blueprint of his or her own studio. The blueprint would have to contain a miniature version of the blueprint, which would contain a miniature version of the blueprint and so on. Without this information, a construction crew would be unable to re-create the studio fully; there would be a blank space where the blueprint had been.

Von Neumann's great insight was an explanation of how to break out of the infinite regress. He realized that the self-description could be used in two distinct ways: first, as the instructions whose interpretation leads to the construction of an identical copy of the device; next, as data to be copied, uninterpreted, and attached to the newly created child so that it too possesses the ability to self-replicate. With this two-step process, the self-description need not contain a description of itself. In the architectural analogy, the blueprint would include a plan for building a photocopy machine. Once the new studio and the photocopier were built, the construction crew would simply run off a copy of the blueprint and put it into the new studio.

Living cells use their self-description, which biologists call the genotype, in exactly these two ways: transcription (DNA is copied mostly uninterpreted to form mRNA) and translation (mRNA is interpreted to build proteins). Von Neumann made this transcription-translation distinction several years before molecular biologists did, and his work has been crucial in understanding self-replication in nature.

To prove these ideas, von Neumann and mathematician Stanislaw M. Ulam came up with the idea of cellular automata. A cellular-automata simulation involves a chessboardlike grid of squares, or cells, each of which is either empty or occupied by one of several possible components. At discrete intervals of time, each cell looks at itself and its neighbors and decides whether to metamorphose into a different component. In making this decision, the cell follows relatively simple rules, which are the same for all cells. These rules constitute the basic physics of the cellular-automata world. All decisions and actions take place locally; cells do not know directly what is happening outside their immediate neighborhood.

The apparent simplicity of cellular automata is deceptive; it does not imply ease of design or poverty of behavior. The most famous automata, John Horton Conway's Game of Life, produces amazingly intricate patterns. Many questions about the dynamic behavior of cellular automata are formally unsolvable. To see how a pattern will unfold, you need to simulate it fully [see Mathematical Games, by Martin Gardner; SCIENTIFIC AMERICAN, October 1970 and February 1971; and "The Ultimate in Anty-Particles," by Ian Stewart, July 1994]. In its own way, a cellular-automata model can be just as complex as the real world.

Copy Machines

WITHIN CELLULAR AUTOMATA, self-replication occurs when a group of components—a "machine"—goes through a sequence of steps to construct a nearby duplicate of itself. Von Neumann's machine was based on a universal constructor, a machine that, given the appropriate instructions, could create any pattern. The constructor consisted of numerous types of components spread over tens of thousands of cells and required a book-length manuscript to be specified. It has still not been simulated in its entirety, let alone actually built, on account of its complexity. A constructor would be even more complicated in the Game of Life because the functions performed by single cells in von Neumann's model—such as transmission of signals and generation of new components—have to be performed by composite structures in Life.

Going to the other extreme, it is easy to find trivial examples of self-replication. For example, suppose a cellular automata has only one type of component, labeled +, and that each cell follows only a single rule: if exactly one of the four neighboring cells contains a +, then the cell becomes a +; otherwise it becomes vacant. With this rule, a single + grows into four more +'s, each of which grows likewise, and so forth.

> Her Majesty pointed to a clock
> and said, "See to it that it produces offspring."

THE AUTHORS

MOSHE SIPPER and *JAMES A. REGGIA* share a long-standing interest in how complex systems can self-organize. Sipper is a senior lecturer in the department of computer science at Ben-Gurion University in Israel and a visiting researcher at the Logic Systems Laboratory of the Swiss Federal Institute of Technology in Lausanne. He is interested mainly in bio-inspired computational paradigms such as evolutionary computation, self-replicating systems and cellular computing. Reggia is a professor of computer science and neurology, working in the Institute for Advanced Computer Studies at the University of Maryland. In addition to studying self-replication, he conducts research on computational models of the brain and its disorders, such as stroke.

Such weedlike proliferation does not shed much light on the principles of replication, because there is no significant machine. Of course, that invites the question of how you would tell a "significant" machine from a trivially prolific automata. No one has yet devised a satisfactory answer. What is clear, however, is that the replicating structure must in some sense be complex. For example, it must consist of multiple, diverse components whose interactions collectively bring about replication—the proverbial "whole must be greater than the sum of the parts." The existence of multiple distinct components permits a self-description to be stored within the replicating structure.

In the years since von Neumann's seminal work, many researchers have probed the domain between the complex and the trivial, developing replicators that require fewer components, less space or simpler rules. A major step forward was taken in 1984 when Christopher G. Langton, then at the University of Michigan, observed that looplike storage devices—which had formed modules of earlier self-replicating machines—could be programmed to replicate on their own. These devices typically consist of two pieces: the loop itself, which is a string of components that circulate around a rectangle, and a construction arm, which protrudes from a corner of the rectangle into the surrounding space. The circulating components constitute a recipe for the loop—for example, "go three squares ahead, then turn left." When this recipe reaches the construction arm, the automata rules make a copy of it. One copy continues around the loop; the other goes down the arm, where it is interpreted as instructions.

By giving up the requirement of universal construction, which was central to von Neumann's approach, Langton showed that a replicator could be constructed from just seven unique components occupying only 86 cells. Even smaller and simpler self-replicating loops have been devised by one of us (Reggia) and our colleagues [*see box on next page*]. Because they have multiple interacting components and include a self-description, they are not trivial. Intriguingly, asymmetry plays an unexpected role: the rules governing replication are often simpler when the components are not rotationally symmetric than when they are.

Emergent Replication

ALL THESE SELF-REPLICATING structures have been designed through ingenuity and much trial and error. This process is arduous and often frustrating; a small change to one of the rules results in an entirely different global behavior, most likely the disintegration of the structure in question. But recent work has gone beyond the direct-design approach. Instead of tailoring the rules to suit a particular type of structure, researchers have experimented with various sets of rules, filled the cellular-automata grid with a "primordial soup" of randomly selected components and checked whether self-replicators emerged spontaneously.

In 1997 Hui-Hsien Chou, now at Iowa State University, and Reggia noticed that as long as the initial density of the free-floating components was above a certain threshold, small self-replicating loops reliably appeared. Loops that collided underwent annihilation, so there was an ongoing process of death as well as birth. Over time, loops proliferated, grew in size and evolved through mutations triggered by debris from past collisions. Although the automata rules were deterministic, these mutations were effectively random, because the system was complex and the components started in random locations.

Such loops are intended as abstract machines and not as simulacra of anything biological, but it is interesting to compare them with biomolecular structures. A loop loosely resembles circular DNA in bacteria, and the construction arm acts as the enzyme that catalyzes DNA replication. More important, replicating loops illustrate how complex global behaviors can arise from simple local interactions. For example, components move around a loop even though the rules say nothing about movement; what is actually happening is that individual cells are coming alive, dying or metamorphosing in such a way that a pattern is eliminated from one position and reconstructed elsewhere—a process that we perceive as motion. In short, cellular automata act locally but appear to think globally. Much the same is true of molecular biology.

In a recent computational experiment, Jason Lohn, now at the NASA Ames Research Center, and Reggia experimented not with different structures but with different sets of rules. Starting with an arbitrary block of four components, they found they could determine a set of rules that made the block self-replicate. They discovered these rules via a genetic algorithm, an automated process that simulates Darwinian evolution.

The most challenging aspect of this work was the definition of the so-called fitness function—the criteria by which sets of rules were judged, thus separating good solutions from bad ones and driving the evolutionary process toward rule sets that facilitated replication. You cannot simply assign high fitness to those sets of rules that cause a structure to replicate, because none of the initial rule sets is likely to allow for replication. The solution was to devise a fitness function composed of a weighted sum of three measures: a growth measure (the extent to which each component type generates an increasing supply of that component), a relative position measure (the extent to which neighboring components stay together) and a replicant measure (a function of the number of actual replicators present). With the right fitness function, evolution can turn rule sets that are sterile into ones that are fecund; the process usually takes 150 or so generations.

Self-replicating structures discovered in this fashion work in a fundamentally different way than self-replicating loops do. For example, they move and deposit copies along the way—unlike replicating loops, which are essentially static. And although these newly discovered replicators consist of multiple, locally interacting components, they do not have an identifiable self-description—there is no obvious genome. The ability to replicate without a self-description may be relevant to questions about how the earliest biological

Continued on page 43

BUILD YOUR OWN REPLICATOR

SIMULATING A SMALL self-replicating loop using an ordinary chess set is a good way to get an intuitive sense of how these systems work. This particular cellular-automata model has four different types of components: pawns, knights, bishops and rooks. The machine initially comprises four pawns, a knight and a bishop. It has two parts: the loop itself, which consists of a two-by-two square, and a construction arm, which sticks out to the right.

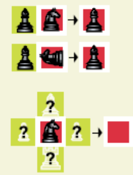
The knight and bishop represent the self-description: the knight, whose orientation is significant, determines which direction to grow, while the bishop tags along and determines how long the side of the loop should be. The pawns are fillers that define the rest of the shape of the loop, and the rook is a transient signal to guide the growth of a new construction arm.

As time progresses, the knight and bishop circulate counterclockwise around the loop. Whenever they encounter the arm, one copy goes out the arm while the original continues around the loop.

HOW TO PLAY: You will need two chessboards: one to represent the current configuration, the other to show the next configuration. For each round, look at each square of the current configuration, consult the rules and place the appropriate piece in the corresponding square on the other board. Each piece metamorphoses depending on its identity and that of the four squares immediately to the left, to the right, above and below. When you have reviewed each square and set up the next configuration, the round is over. Clear the first board and repeat. Because the rules are complicated, it takes a bit of patience at first. You can also view the simulation at lslwww.epfl.ch/chess

The direction in which a knight faces is significant. In the drawings here, we use standard chess conventions to indicate the orientation of the knight: the horse's muzzle points forward. If no rule explicitly applies, the contents of the square stay the same. Squares on the edge should be treated as if they have adjacent empty squares off the board. *—M.S. and J.A.R.*

KNIGHT

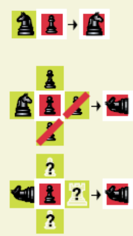


IF THERE is a bishop just behind or to the left of the knight, replace the knight with another bishop.

OTHERWISE, if at least one of the neighboring squares is occupied, remove the knight and leave the square empty.

PAWN

IF THERE is a neighboring knight, replace the pawn with a knight with a certain orientation, as follows:



IF A NEIGHBORING knight is facing away from the pawn, the new knight faces the opposite way.

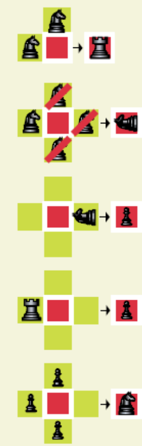
OTHERWISE, if there is exactly one neighboring pawn, the new knight faces that pawn.

OTHERWISE the new knight faces in the same direction as the neighboring knight.

BISHOP OR ROOK

REPLACE IT with a pawn.

EMPTY SQUARE



IF THERE are two neighboring knights and either faces the empty square, fill the square with a rook.

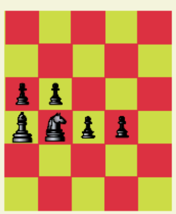
IF THERE is only one neighboring knight and it faces the square, fill the square with a knight rotated 90 degrees counterclockwise.

IF THERE is a neighboring knight and its left side faces the square, and the other neighbors are empty, fill the square with a pawn.

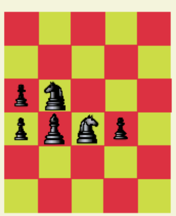
IF THERE is a neighboring rook, and the other neighbors are empty, fill the square with a pawn.

IF THERE are three neighboring pawns, fill the square with a knight facing the fourth, empty neighbor.

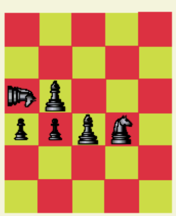
STAGES OF REPLICATION



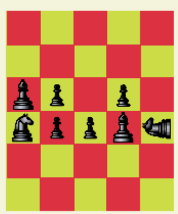
INITIALLY, the self-description, or "genome"—a knight followed by a bishop—is poised at the start of the construction arm.



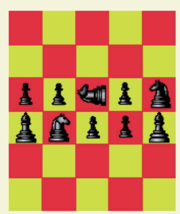
1 The knight and bishop move counterclockwise around the loop. A clone of the knight heads out the arm.



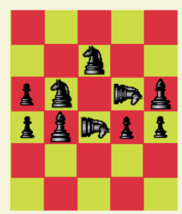
2 The original knight-bishop pair continues to circulate. The bishop is cloned and follows the new knight out the arm.



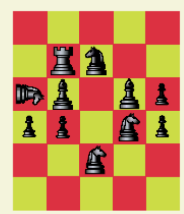
3 The knight triggers the formation of two corners of the child loop. The bishop tags along, completing the gene transfer.



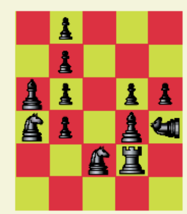
4 The knight forges the remaining corner of the child loop. The loops are connected by the construction arm and a knight-errant.



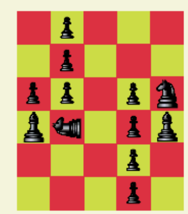
5 The knight-errant moves up to endow the parent with a new arm. A similar process, one step delayed, begins for the child loop.



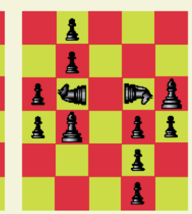
6 The knight-errant, together with the original knight-bishop pair, conjures up a rook. Meanwhile the old arm is erased.



7 The rook kills the knight and generates the new, upward arm. Another rook prepares to do the same for the child.



8 At last the two loops are separate and whole. The self-descriptions continue to circulate, but otherwise all is calm.



9 The parent prepares to give birth again. In the following step, the child too will begin to replicate.

ROBOT, HEAL THYSELF

Computers that fix themselves are the first application of artificial self-replication

LAUSANNE, SWITZERLAND—Not many researchers encourage the wanton destruction of equipment in their labs. Daniel Mange, however, likes it when visitors walk up to one of his inventions and press the button marked KILL. The lights on the panel go out; a small box full of circuitry is toast. Early in May his team unveiled its latest contraption at a science festival here—a wall-size digital clock whose components you can zap at will—and told the public: Give it your best shot. See if you can crash the system.

The goal of Mange and his team is to instill electronic circuits with the ability to take a lickin' and keep on tickin'—just like living things. Flesh-and-blood creatures might not be so good at calculating π to the millionth digit, but they can get through the day without someone pressing Ctrl-Alt-Del. Combining the precision of digital hardware with the resilience of biological wetware is a leading challenge for modern electronics.

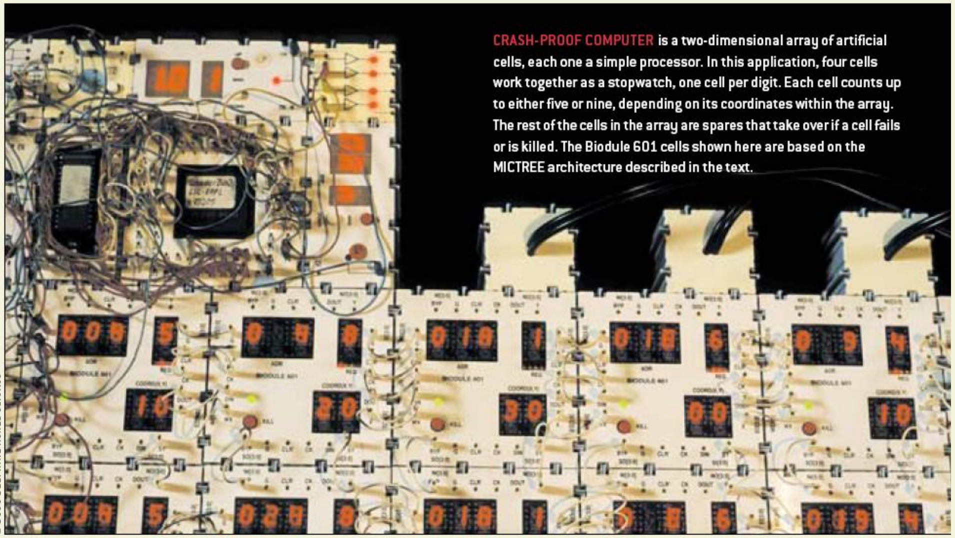
Electronics engineers have been working on fault-tolerant circuits ever since there were electronics engineers [see "Redundancy in Computers," by William H. Pierce; SCIENTIFIC AMERICAN, February 1964]. Computer modems would still be dribbling data at 1200 baud if it weren't for error detection and correction. In many applications, simple quality-control checks, such as extra data bits, suffice. More complex systems provide entire backup computers. The space shuttle, for example, has five processors. Four of them perform the same calculations; the fifth checks whether they agree and pulls the plug on any dissenter.

The problem with these systems, though, is that they rely on centralized control. What if that control unit goes bad?

Nature has solved that problem through radical decentralization. Cells in the body are all basically identical; each takes on a specialized task, performs it autonomously and, in the event of infection or failure, commits hara-kiri so that its tasks can be taken up by new cells. These are the attributes that Mange, a professor at the Swiss Federal Institute of Technology here, and others have sought since 1993 to emulate in circuitry, as part of the "Embryonics" (embryonic electronics) project.

One of their earlier inventions, the MICTREE (microinstruction tree) artificial cell, consisted of a simple processor and four bits of data storage. The cell is contained in a plastic box roughly the size of a pack of Post-its. Electrical contacts run along the sides so that cells can be snapped together like Legos. As in cellular automata, the models used to study the theory of self-replication, the MICTREE cells are connected only to their immediate neighbors. The communication burden on each cell is thus independent of the total number of cells. The system, in other words, is easily scalable—unlike many parallel-computing architectures.

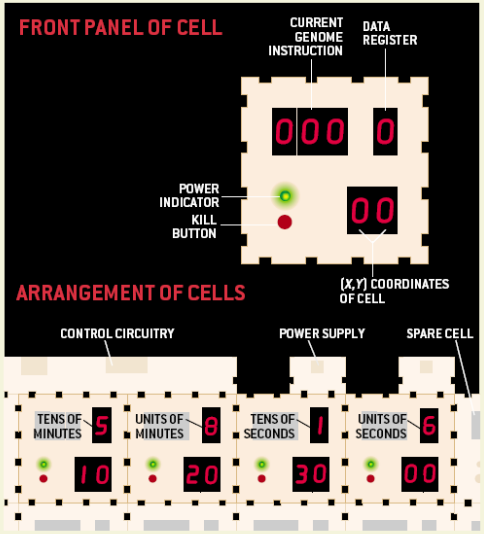
Cells follow the instructions in their "genome," a program written in a subset of the Pascal computer language. Like their biological antecedents, the cells all contain the exact same genome and execute part of it based on their position within the array, which each cell calculates relative to its neighbors. Wasteful though it may seem, this redundancy allows the array to withstand the loss of any cell. Whenever someone presses the KILL button on a cell, that cell shuts down, and its left and right neighbors become directly connected. The right neighbor recalculates its position and starts executing the deceased's program. Its tasks, in turn, are taken up by the next cell to the right, and so on, until a cell designated as a spare is pressed into service.



CRASH-PROOF COMPUTER is a two-dimensional array of artificial cells, each one a simple processor. In this application, four cells work together as a stopwatch, one cell per digit. Each cell counts up to either five or nine, depending on its coordinates within the array. The rest of the cells in the array are spares that take over if a cell fails or is killed. The Biodule 601 cells shown here are based on the MICTREE architecture described in the text.

Writing programs for any parallel processor is tricky, but the MICTREE array requires an especially unconventional approach. Instead of giving explicit instructions, the programmer must devise simple rules out of which the desired function will emerge. Being Swiss, Mange demonstrates by building a superreliable stopwatch. Displaying minutes and seconds requires four cells in a row, one for each digit. The genome allows for two cell types: a counter from zero to nine and a counter from zero to five. An oscillator feeds one pulse per second into the rightmost cell. After 10 pulses, this cell cycles back to zero and sends a pulse to the cell on its left, and so on down the line. The watch takes up part of an array of 12 cells; when you kill one, the clock transplants itself one cell over and carries on. Obviously, though, there is a limit to its resilience: the whole thing will fail after, at most, eight kills.

The prototype MICTREE cells are hardwired, so their processing power cannot be tailored to a specific application. In a finished product, cells would instead be implemented on a field-programmable gate array, a grid of electronic components that can be reconfigured on the fly [see "Configurable Computing," by John Villasenor and William H. Mangione-Smith; SCIENTIFIC AMERICAN, June 1997]. Mange's team is now custom-designing a gate array, known as MUXTREE (multiplexer tree), that is optimized for artificial cells. In the biological metaphor, the components of this array are the "molecules" that constitute a cell. Each consists of a logic gate, a data bit and a string of configuration bits that determines the function of this gate.



Building a cell out of such molecules offers not only flexibility but also extra endurance. Each molecule contains two copies of the gate and three of the storage bit. If the two gates ever give different results, the molecule kills itself for the greater good of the cell. As a last gasp, the molecule sends its data bit (preserved by the triplicate storage) and configuration to its right neighbor, which does the same, and the process continues until the rightmost molecule transfers its data to a spare. This second level of fault tolerance prevents a single error from wiping out an entire cell.

A total of 2,000 molecules, divided into four 20-by-25 cells, make up the BioWall—the giant digital clock that Mange's team has just put on display. Each molecule is enclosed in a small box and includes a KILL button and an LED display. Some molecules are configured to perform computations; others serve as pixels in the clock display. Making liberal use of the KILL buttons, I did my utmost to crash the system, something I'm usually quite good at. But the plucky clock just wouldn't submit. The clock display did start to look funny—numerals bent over as their pixels shifted to the right—but at least it was still legible, unlike most faulty electronic signs.

That said, the system did suffer from display glitches, which Mange attributed mainly to timing problems. Although the processing power is decentralized, the cells still rely on a central oscillator to coordinate their communications; sometimes they fall out of sync. Another Embryonics team, led by Andy Tyrrell of the University of York in England, has been studying making the cells asynchronous, like their biological counterparts. Cells would generate handshaking signals to orchestrate data transfers. The present system is also unable to catch certain types of error, including damaged configuration strings. Tyrrell's team has proposed adding watchdog molecules—an immune system—that would monitor the configurations (and one another) for defects.

Although these systems demand an awful lot of overhead, so do other fault-tolerance technologies. "While Embryonics appears to be heavy on redundancy, it actually is not that bad when compared to other systems," Tyrrell argues. Moreover, MUXTREE should be easier to scale down to the nano level; the "molecules" are simple enough to really be molecules. Says Mange, "We are preparing for the situation where electronics will be at the same scale as biology."

On a philosophical level, Embryonics comes very close to the dream of building a self-replicating machine. It may not be quite as dramatic as a robot that can go down to Radio Shack, pull parts off the racks, and take them home to resolder a connection or build a loving mate. But the effect is much the same. Letting machines determine their own destiny—whether reconfiguring themselves on a silicon chip or reprogramming themselves using a neural network or genetic algorithm—sounds scary, but perhaps we should be gratified that machines are becoming more like us: imperfect, fallible but stubbornly resourceful.

—*George Musser, imperfect but resourceful staff editor and writer*

Continued from page 39

replicators originated. In a sense, researchers are seeing a continuum between nonliving and living structures.

Many researchers have tried other computational models besides the traditional cellular automata. In asynchronous cellular automata, cells are not updated in concert; in nonuniform cellular automata, the rules can vary from cell to cell. Another approach altogether is Core War [see Computer Recreations, by A. K. Dewdney; SCIENTIFIC AMERICAN, May 1984] and its successors, such as ecologist Thomas S. Ray's Tierra system. In these simulations the "organisms" are computer programs that vie for processor time and memory. Ray has observed the emergence of "parasites" that co-opt the self-replication code of other organisms.

In a sense, researchers are seeing a continuum between nonliving and living structures.

Getting Real

SO WHAT GOOD are these machines? Von Neumann's universal constructor can compute in addition to replicating, but it is an impractical beast. A major advance has been the development of simple yet useful replicators. In 1995 Gianluca Tempesti of the Swiss Federal Institute of Technology in Lausanne simplified the loop self-description so it could be interlaced with a small program—in this case, one that would spell the acronym of his lab, "LSL." His insight was to create automata rules that allow loops to replicate in two stages. First the loop, like Langton's loop, makes a copy of itself. Once finished, the daughter loop sends a signal back to its parent, at which point the parent sends the instructions for writing out the letters.

Drawing letters was just a demonstration. The following year Jean-Yves Perrier, Jacques Zahnd and one of us (Sipper) designed a self-replicating loop with universal computational capabilities—that is, with the computational power of a universal Turing machine, a highly simplified but fully capable computer. This loop has two "tapes," or long strings of components, one for the program and the other for data. The loops can execute an arbitrary program in addition to self-replicating. In a sense, they are as complex as the computer that simulates them. Their main limitation is that the program is copied unchanged from parent to child, so that all loops carry out the same set of instructions.

In 1998 Chou and Reggia swept away this limitation. They showed how self-replicating loops carrying distinct information, rather than a cloned program, can be used to solve a problem known as satisfiability. The loops can be used to determine whether the variables in a logical expression can be assigned values such that the entire expression evaluates to "true." This problem is NP-complete—in other words, it belongs to the family of nasty puzzles, including the famous traveling-salesman problem, for which there is no known efficient solution. In Chou and Reggia's cellular-automata universe, each replicator received a different partial solution. During replication, the solutions mutated, and replicators with promising solutions were allowed to proliferate while those with failed solutions died out.

Although various teams have created cellular automata in electronic hardware, such systems are probably too wasteful for practical applications; automata were never really intended to be implemented directly. Their purpose is to illuminate the underlying principles of replication and, by doing so, inspire more concrete efforts. The loops provide a new paradigm for designing a parallel computer from either transistors or chemicals [see "Computing with DNA," by Leonard M. Adleman; SCIENTIFIC AMERICAN, August 1998].

In 1980 a NASA team led by Robert Freitas, Jr., proposed planting a factory on the moon that would replicate itself, using local lunar materials, to populate a large area exponentially. Indeed, a similar probe could colonize the entire galaxy, as physicist Frank J. Tipler of Tulane University has argued. In the nearer term, computer scientists and engineers have experimented with the automated design of robots [see "Dawn of a New Species?" by George Musser; SCIENTIFIC AMERICAN, November 2000]. Although these systems are not truly self-replicating—the offspring are much simpler than the parent—they are a first step toward fulfilling the queen of Sweden's request.

Should physical self-replicating machines become practical, they and related technologies will raise difficult issues, including the *Terminator* film scenario in which artificial creatures outcompete natural ones. We prefer the more optimistic, and more probable, scenario that replicators will be harnessed to the benefit of humanity [see "Will Robots Inherit the Earth?" by Marvin Minsky; SCIENTIFIC AMERICAN, October 1994]. The key will be taking the advice of 14th-century English philosopher William of Ockham: *entia non sunt multiplicanda praeter necessitatem*—entities are not to be multiplied beyond necessity. SA

MORE TO EXPLORE

Simple Systems That Exhibit Self-Directed Replication. J. Reggia, S. Armentrout, H. Chou and Y. Peng in *Science*, Vol. 259, No. 5099, pages 1282–1287; February 26, 1993.

Emergence of Self-Replicating Structures in a Cellular Automata Space. H. Chou and J. Reggia in *Physica D*, Vol. 110, Nos. 3–4, pages 252–272; December 15, 1997.

Special Issue: Von Neumann's Legacy: On Self-Replication. Edited by M. Sipper, G. Tempesti, D. Mange and E. Sanchez in *Artificial Life*, Vol. 4, No. 3; Summer 1998.

Towards Robust Integrated Circuits: The Embryonics Approach. D. Mange, M. Sipper, A. Stauffer and G. Tempesti in *Proceedings of the IEEE*, Vol. 88, No. 4, pages 516–541; April 2000.

Moshe Sipper's Web page on artificial self-replication is at **lslwww.epfl.ch/~moshes/selfrep/**

Animations of self-replicating loops can be found at **necsi.org/postdocs/sayama/sdsr/java/**

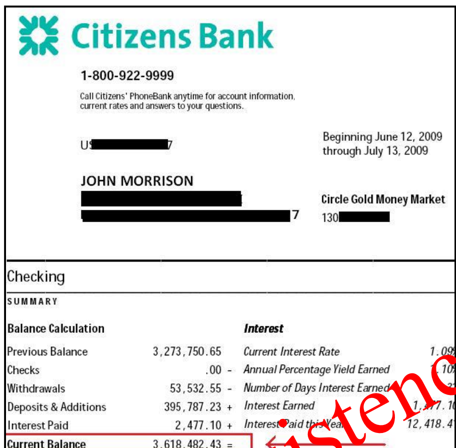
For John von Neumann's universal constructor, see **alife.santafe.edu/alife/topics/jvn/jvn.html**

Non-Existence Proofs

- Must cover all possible (usually infinite) scenarios!
- Examples / counter-examples are not convincing!
- Not "symmetric" to existence proofs!

Ex: proofs that you
are a millionaire:

"Proofs" that you
are not a millionaire ?


Citizens Bank
1-800-922-9999
Call Citizens' PhoneBank anytime for account information, current rates and answers to your questions.

U[redacted]7

Beginning June 12, 2009
through July 13, 2009

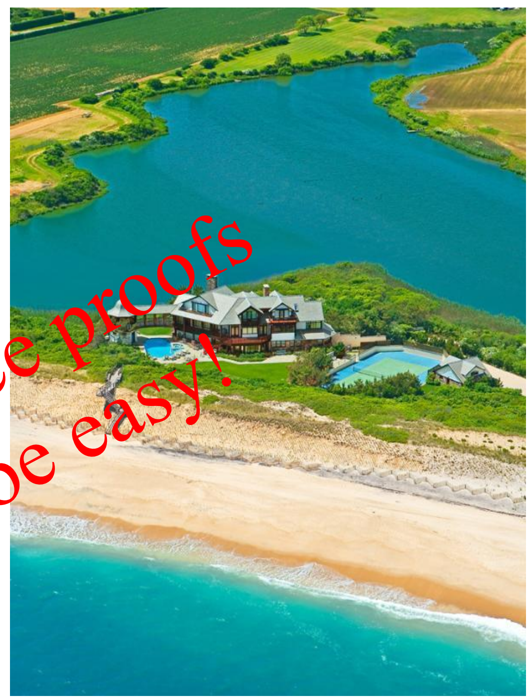
JOHN MORRISON
[redacted]7

Circle Gold Money Market
130[redacted]

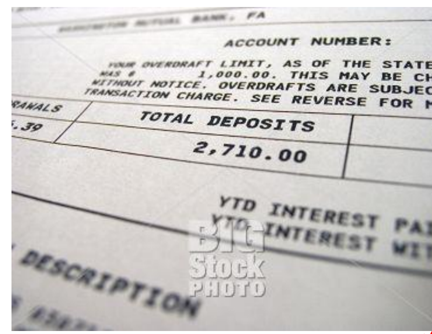
Checking

SUMMARY

Balance Calculation		Interest	
Previous Balance	3,273,750.65	Current Interest Rate	1.09
Checks	.00 -	Annual Percentage Yield Earned	1.10
Withdrawals	53,532.55 -	Number of Days Interest Earned	
Deposits & Additions	395,787.23 +	Interest Earned	1,477.10
Interest Paid	2,477.10 +	Interest Paid this Year	12,418.4
Current Balance	3,618,482.43 =		

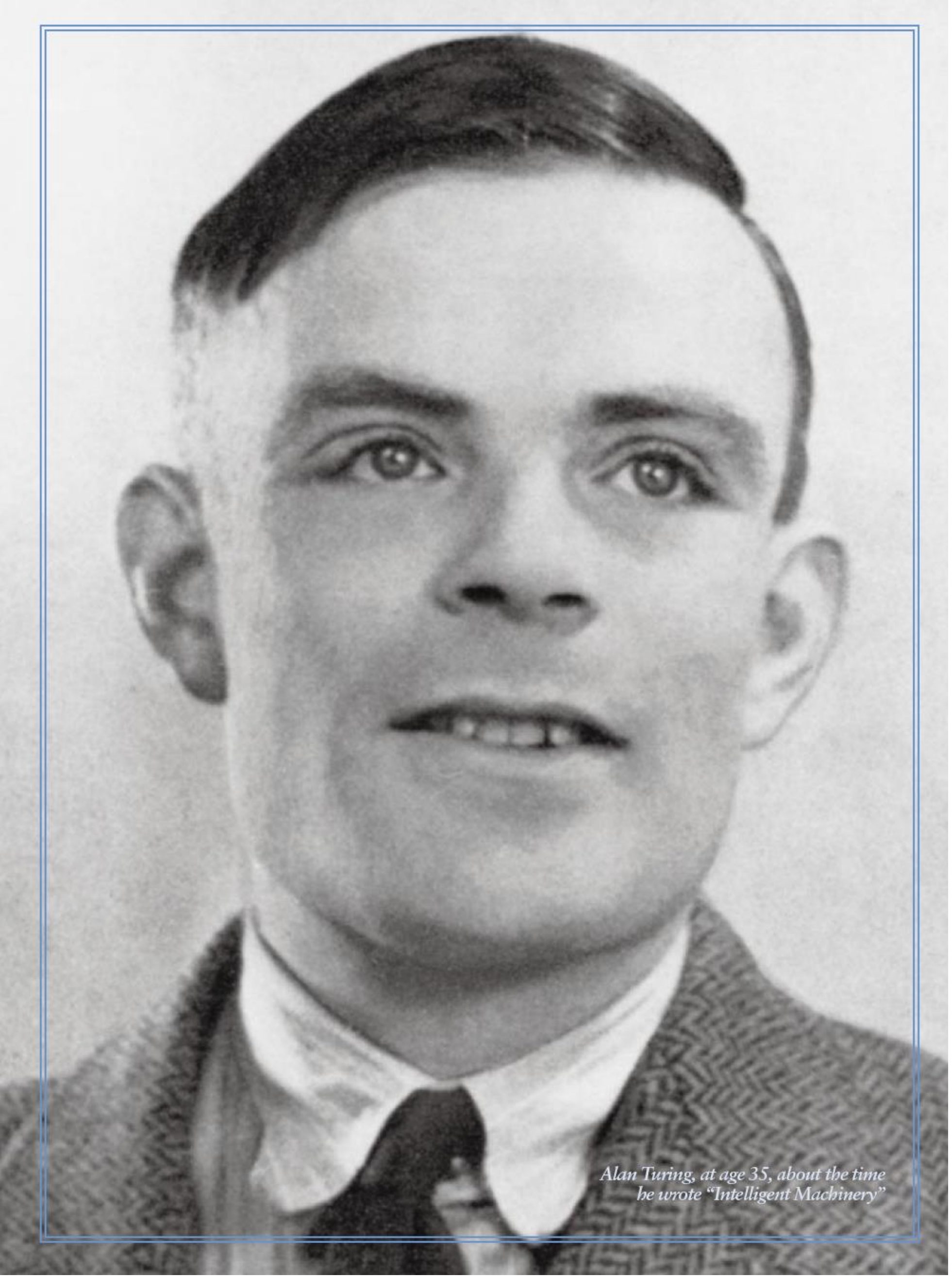


Existence proofs
can be easy!



Non-existence proofs
are often hard!

P≠NP



Alan Turing, at age 35, about the time he wrote "Intelligent Machinery"

Alan Turing's Forgotten Ideas in Computer Science

Well known for the machine, test and thesis that bear his name, the British genius also anticipated neural-network computers and "hypercomputation"

by B. Jack Copeland and Diane Proudfoot

Alan Mathison Turing conceived of the modern computer in 1935. Today all digital computers are, in essence, "Turing machines." The British mathematician also pioneered the field of artificial intelligence, or AI, proposing the famous and widely debated Turing test as a way of determining whether a suitably programmed computer can think. During World War II, Turing was instrumental in breaking the German Enigma code in part of a top-secret British operation that historians say shortened the war in Europe by two years. When he died at the age of 41, Turing was doing the earliest work on what would now be called artificial life, simulating the chemistry of biological growth.

Throughout his remarkable career, Turing had no great interest in publicizing his ideas. Consequently, important aspects of his work have been neglected or forgotten over the years. In particular, few people—even those knowledgeable about computer science—are familiar with Turing's fascinating anticipation of connectionism, or neuronlike computing. Also neglected are his groundbreaking theoretical concepts in the exciting area of "hypercomputation." According to some experts, hypercomputers might one day solve problems heretofore deemed intractable.

The Turing Connection

Digital computers are superb number crunchers. Ask them to predict a rocket's trajectory or calculate the financial figures for a large multinational corporation, and they can churn out the answers in seconds. But seemingly simple actions that people routinely perform, such as recognizing a face or reading handwriting, have been devilishly tricky to program. Perhaps the networks of neurons that make up the brain have a natural facility for such tasks that standard computers lack. Scientists have thus been investigating computers modeled more closely on the human brain.

Connectionism is the emerging science of computing with networks of artificial neurons. Currently researchers usually simulate the neurons and their interconnections within an ordinary digital computer (just as engineers create virtual models of aircraft wings and skyscrapers). A training algorithm that runs on the computer adjusts the connections between the neurons, honing the network into a special-purpose machine dedicated to some particular function, such as forecasting international currency markets.

Modern connectionists look back to Frank Rosenblatt, who published the first of many papers on the topic in 1957, as the founder of their approach. Few realize that Turing had already investigated connectionist networks as early as 1948, in a little-known paper entitled "Intelligent Machinery."

Written while Turing was working for the National Physical Laboratory in London, the manuscript did not meet with his employer's approval. Sir Charles Darwin, the rather headmasterly director of the laboratory and grandson of the great English naturalist, dismissed it as a "schoolboy essay." In reality, this farsighted paper was the first manifesto of the field of artificial intelligence. In the work—which remained unpublished until 1968, 14 years after Turing's death—the British mathematician not only set out the fundamentals of connectionism but also brilliantly introduced many of the concepts that were later to become central to AI, in some cases after reinvention by others.

In the paper, Turing invented a kind of neural network that he called a "B-type unorganized machine," which consists of artificial neurons and devices that modify the connections between them. B-type machines may contain any number of neurons connected in any pattern but are always subject to the restriction that each neuron-to-neuron connection must pass through a modifier device.

Few realize that Turing had already investigated connectionist networks as early as 1948.

All connection modifiers have two training fibers. Applying a pulse to one of them sets the modifier to "pass mode," in which an input—either 0 or 1—passes through unchanged and becomes the output. A pulse on the other fiber places the modifier in "interrupt mode," in which the output is always 1, no matter what the input is. In this state the modifier destroys all information attempting to pass along the connection to which it is attached.

Once set, a modifier will maintain its function (either "pass" or "interrupt") unless it receives a pulse on the other training fiber. The presence of these ingenious connection modifiers enables the training of a B-type unorganized machine by means of what Turing called "appropriate interference, mimicking education." Actually, Turing theorized that "the cortex of an infant is an unorganized machine, which can be organized by suitable interfering training."

Each of Turing's model neurons has two input fibers, and the output of a neuron is a simple logical function of its two inputs. Every neuron in the network executes the same logical operation of "not and" (or NAND): the output is 1 if either of the inputs is 0. If both inputs are 1, then the output is 0.

Turing selected NAND because every other logical (or Boolean) operation can be accomplished by groups of NAND neurons. Furthermore, he showed that even the connection modifiers themselves can be built out of NAND neurons. Thus, Turing specified a network made up of nothing more than NAND neurons and their connecting fibers—about the simplest possible model of the cortex.

In 1958 Rosenblatt defined the theoretical basis of connectionism in one succinct statement: "Stored information takes the form of new connections, or transmission channels in the nervous system (or the creation of conditions which are functionally equivalent to new connections)." Because the destruction of existing connections can be functionally equivalent to the creation of new ones, researchers can build a network for accomplishing a specific task by taking one with an excess of connections and selectively destroying some of them. Both actions—destruction and creation—are employed in the training of Turing's B-types.

At the outset, B-types contain random interneural connections whose modifiers have been set by chance to either pass or interrupt. During training, unwanted connections are destroyed by switching their attached modifiers to interrupt mode. Conversely, changing a modifier from interrupt to pass in effect creates a connection. This selective culling and enlivening of connections hones the initially random network into one organized for a given job.

Turing wished to investigate other kinds of unorganized machines, and he longed to simulate a neural network and its training regimen using an ordinary digital computer. He would, he said, "allow the whole system to run for an appreciable period, and then break in as a kind of 'inspector of schools' and see what progress had been made." But his own work on neural networks was carried out shortly before the first general-purpose electronic computers became available. (It was not until 1954, the year of Turing's death, that Belmont G. Farley and Wesley A. Clark succeeded at the Massachusetts Institute of Technology in running the first computer simulation of a small neural network.)

Paper and pencil were enough, though, for Turing to show that a sufficiently large B-type neural network can be configured (via its connection modifiers) in such a way that it becomes a general-purpose computer. This discovery illuminates one of the most fundamental problems concerning human cognition.

From a top-down perspective, cognition includes complex sequential processes, often involving language or other forms of symbolic representation, as in mathematical calculation. Yet from a bottom-up view, cognition is nothing but the simple firings of neurons. Cognitive scientists face the problem of how to reconcile these very different perspectives.

Turing's discovery offers a possible solution: the cortex, by virtue of being a neural network acting as a general-purpose computer, is able to carry out the sequential, symbol-rich processing discerned in the view from the top. In 1948 this hypothesis was well ahead of its time, and today it remains among the best guesses concerning one of cognitive science's hardest problems.

Computing the Uncomputable

In 1935 Turing thought up the abstract device that has since become known as the "universal Turing machine." It consists of a limitless memory that stores both program and data and a scanner that moves back and forth through the memory, symbol by symbol, reading the information and writing additional symbols. Each of the machine's basic actions is very simple—such as "identify the symbol on which the scanner is positioned," "write '1'" and "move one position to the left." Complexity is achieved by chaining together large numbers of these basic actions. Despite its simplicity, a universal Turing machine can execute any task that can be done by the most powerful of today's computers. In fact, all modern digital computers are in essence universal Turing machines [see "Turing Machines," by John E. Hopcroft; Scientific American, May 1984].

Turing's aim in 1935 was to devise a machine—one as simple as possible—capable of any calculation that a human mathematician working in accordance with some algorithmic method could perform, given unlimited time, energy, paper and pencils, and perfect concentration. Calling a machine "universal" merely signifies that it is capable of all such calculations. As Turing himself wrote, "Electronic computers are intended to carry out any definite rule-of-thumb process which could have been done by a human operator working in a disciplined but unintelligent manner."

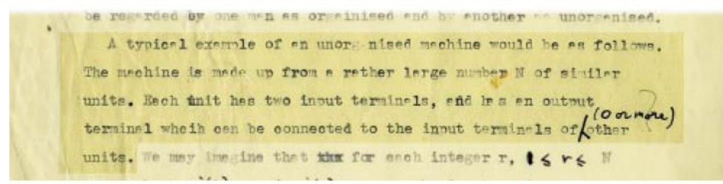
Such powerful computing devices notwithstanding, an intriguing question arises: Can machines be devised that are capable of accomplishing even more? The answer is that these "hypermachines" can be described on paper, but no one as yet knows whether it will be possible to build one. The field of hypercomputation is currently attracting a growing number of scientists. Some speculate that the human brain itself—the most complex information processor known—is actually a naturally occurring example of a hypercomputer.

Before the recent surge of interest in hypercomputation, any information-processing job that was known to be too difficult for universal Turing machines was written off as "uncomputable." In this sense, a hypermachine computes the uncomputable.

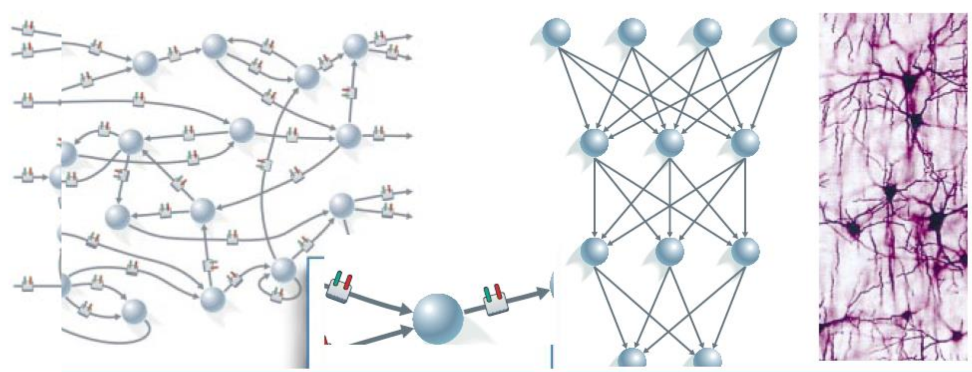
Examples of such tasks can be found in even the most straightforward areas of mathematics. For instance, given arithmetical statements picked at random, a universal Turing machine may not always be able to tell which are theorems (such as "7 + 5 = 12") and which are nontheorems (such as "every number is the sum of two even numbers"). Another type of uncomputable problem comes from geometry. A set of tiles—variously sized squares with different colored edges—"tiles the plane" if the Euclidean plane can be covered by copies of the tiles with no gaps or overlaps and with adjacent edges always the same color. Logicians William Hanf and Dale Myers of the University of Hawaii have discovered a tile set that tiles the plane only in patterns too complicated for a universal Turing machine to calculate. In the field of computer science, a universal Turing machine cannot always predict whether a given program will terminate or continue running forever. This is sometimes expressed by saying that no general-purpose programming language (Pascal, BASIC, Prolog, C and so on) can have a foolproof crash debugger: a tool that detects all bugs that could lead to crashes, including errors that result in infinite processing loops.

Turing himself was the first to investigate the idea of machines that can perform mathematical tasks too difficult

Turing's Anticipation of Connectionism


be regarded by one man as organised and by another as unorganised.

A typical example of an unorganised machine would be as follows. The machine is made up from a rather large number N of similar units. Each unit has two input terminals, and has an output terminal wheih can be connected to the input terminals of (0 or more) other units. We may imagine that for each integer r, 1 ≤ r ≤ N



In a paper that went unpublished until 14 years after his death (*top*), Alan Turing described a network of artificial neurons connected in a random manner. In this "B-type unorganized machine" (*bottom left*), each connection passes through a modifier that is set either to allow data to pass unchanged (*green fiber*) or to destroy the transmitted information (*red fiber*). Switching the modifiers from one mode to the other enables the network to be trained. Note that each neuron has two inputs (*bottom left, inset*) and executes the simple logical operation of "not and," or NAND: if both inputs are 1, then the output is 0; otherwise the output is 1.

In Turing's network the neurons interconnect freely. In contrast, modern networks (*bottom center*) restrict the flow of information from layer to layer of neurons. Connectionists aim to simulate the neural networks of the brain (*bottom right*).

TOM MOORE (*illustrations*); KING'S COLLEGE MODERN ARCHIVES, CAMBRIDGE UNIVERSITY LIBRARY (*top*); PETER ARNOLD, INC. (*bottom right*)

Alan Turing's Forgotten Ideas in Computer Science

Using an Oracle to Compute the Uncomputable

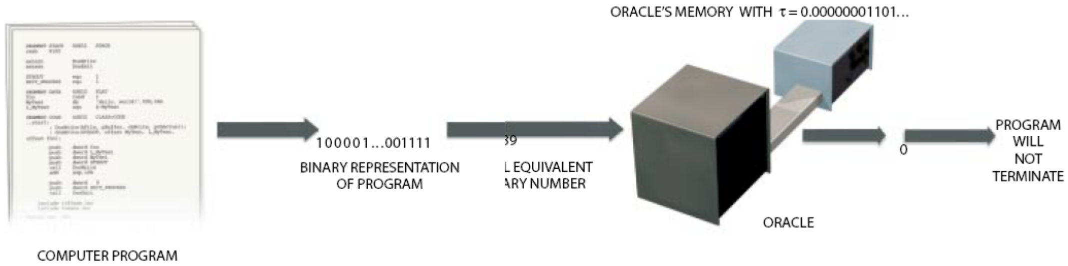
Alan Turing proved that his universal machine—and by extension, even today's most powerful computers—could never solve certain problems. For instance, a universal Turing machine cannot always determine whether a given software program will terminate or continue running forever. In some cases, the best the universal machine can do is execute the program and wait—maybe eternally—for it to finish. But in his doctoral thesis (*below*), Turing did imagine that a machine equipped with a special "oracle" could perform this and other "uncomputable" tasks. Here is one example of how, in principle, an oracle might work.

Consider a hypothetical machine for solving the formidable "terminating program" problem (*above*). A computer program can be represented as a finite string of 1s and 0s. This sequence of digits can also be thought of as the binary representation of an integer, just as 1011011 is the equivalent of 91. The oracle's job can then be restated as, "Given an integer that represents a program (for any computer that can be simulated by a universal Turing machine), output a '1' if the program will terminate or a '0' otherwise."

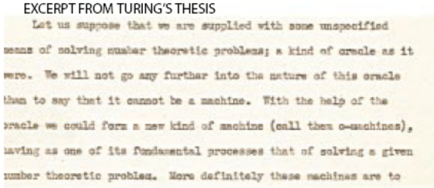
The oracle consists of a perfect measuring device and a store, or memory, that contains a precise value—call it τ for Turing—of some physical quantity. (The memory might, for example, resemble a capacitor storing an exact amount of electricity.) The value of τ is an irrational number; its written representation would be an infinite string of binary digits, such as 0.00000001101…

The crucial property of τ is that its individual digits happen to represent accurately which programs terminate and which do not. So, for instance, if the integer representing a program were 8,735,439, then the oracle could by measurement obtain the 8,735,439th digit of τ (counting from left to right after the decimal point). If that digit were 0, the oracle would conclude that the program will process forever.

Obviously, without τ the oracle would be useless, and finding some physical variable in nature that takes this exact value might very well be impossible. So the search is on for some practicable way of implementing an oracle. If such a means were found, the impact on the field of computer science could be enormous. *—B.J.C. and D.P.*



EXCERPT FROM TURING'S THESIS


Let us suppose that we are supplied with some unspecified means of solving number theoretic problems; a kind of oracle as it were. We will not go any further into the nature of this oracle than to say that it cannot be a machine. With the help of the oracle we could form a new kind of machine (call them o-machines), having as one of its fundamental processes that of solving a given number theoretic problem. More definitely these machines are to

PRINCETON ARCHIVES

TOM MOORE

for universal Turing machines. In his 1938 doctoral thesis at Princeton University, he described "a new kind of machine," the "O-machine."

An O-machine is the result of augmenting a universal Turing machine with a black box, or "oracle," that is a mechanism for carrying out uncomputable tasks. In other respects, O-machines are similar to ordinary computers. A digitally encoded program is fed in, and the machine produces digital output from the input using a step-by-step procedure of repeated applications of the machine's basic operations, one of which is to pass data to the oracle and register its response.

Turing gave no indication of how an oracle might work. (Neither did he explain in his earlier research how the basic actions of a universal Turing machine—for example, "identify the symbol in the scanner"—might take place.) But notional mechanisms that fulfill the specifications of an O-machine's black box are not difficult to imagine [*see box above*]. In principle, even a suitable B-type network can compute the uncomputable, provided the activity of the neurons is desynchronized. (When a central clock keeps the neurons in step with one another, the functioning of the network can be exactly simulated by a universal Turing machine.)

Even among experts, Turing's pioneering theoretical concept of a hypermachine has largely been forgotten.

In the exotic mathematical theory of hypercomputation, tasks such as that of distinguishing theorems from nontheorems in arithmetic are no longer uncomputable. Even a debugger that can tell whether any program written in C, for example, will enter an infinite loop is theoretically possible.

If hypercomputers can be built—and that is a big if—the potential for cracking logical and mathematical problems hitherto deemed intractable will be enormous. Indeed, computer science may be approaching one of its most significant advances since researchers wired together the first electronic embodiment of a universal Turing machine decades ago. On the other hand, work on hypercomputers may simply fizzle out for want of some way of realizing an oracle.

The search for suitable physical, chemical or biological phenomena is getting under way. Perhaps the answer will be complex molecules or other structures that link together in patterns as complicated as those discovered by Hanf and Myers. Or, as suggested by Jon Doyle of M.I.T., there may be naturally occurring equilibrating systems with discrete spectra that can be seen as carrying out, in principle, an uncomputable task, producing appropriate output (1 or 0, for example) after being bombarded with input.

Outside the confines of mathematical logic, Turing's O-machines have largely been forgotten, and instead a myth has taken hold. According to this apocryphal account, Turing demonstrated in the mid-1930s that hypermachines are impossible. He and Alonzo Church, the logician who was Turing's doctoral adviser at Princeton, are mistakenly credited with having enunciated a principle to the effect that a universal Turing machine can exactly simulate the behavior of any other information-processing machine. This proposition, widely but incorrectly known as the Church-Turing thesis, implies that no machine can carry out an information-processing task that lies beyond the scope of a universal Turing machine. In truth, Church and Turing claimed only that a universal Turing machine can match the behavior of any human mathematician working with paper and pencil in accordance with an algorithmic method—a considerably weaker claim that certainly does not rule out the possibility of hypermachines.

Even among those who are pursuing the goal of building hypercomputers, Turing's pioneering theoretical contributions have been overlooked. Experts routinely talk of carrying out information processing "beyond the Turing limit" and describe themselves as attempting to "break the Turing barrier." A recent review in *New Scientist* of this emerging field states that the new machines "fall outside Turing's conception" and are "computers of a type never envisioned by Turing," as if the British genius had not conceived of such devices more than half a century ago. Sadly, it appears that what has already occurred with respect to Turing's ideas on connectionism is starting to happen all over again.

The Final Years

In the early 1950s, during the last years of his life, Turing pioneered the field of artificial life. He was trying to simulate a chemical mechanism by which the genes of a fertilized egg cell may determine the anatomical structure of the resulting animal or plant. He described this research as "not altogether unconnected" to his study of neural networks, because "brain structure has to be … achieved by the genetical embryological mechanism, and this theory that I am now working on may make clearer what restrictions this really implies." During this period, Turing achieved the distinction of being the first to engage in the computer-assisted exploration of nonlinear dynamical systems. His theory used nonlinear differential equations to express the chemistry of growth.

But in the middle of this groundbreaking investigation, Turing died from cyanide poisoning, possibly by his own hand. On June 8, 1954, shortly before what would have been his 42nd birthday, he was found dead in his bedroom. He had left a large pile of handwritten notes and some computer programs. Decades later this fascinating material is still not fully understood.

The Authors

B. JACK COPELAND and DIANE PROUDFOOT are the directors of the Turing Project at the University of Canterbury, New Zealand, which aims to develop and apply Turing's ideas using modern techniques. The authors are professors in the philosophy department at Canterbury, and Copeland is visiting professor of computer science at the University of Portsmouth in England. They have written numerous articles on Turing. Copeland's *Turing's Machines* and *The Essential Turing* are forthcoming from Oxford University Press, and his *Artificial Intelligence* was published by Blackwell in 1993. In addition to the logical study of hypermachines and the simulation of B-type neural networks, the authors are investigating the computer models of biological growth that Turing was working on at the time of his death. They are organizing a conference in London in May 2000 to celebrate the 50th anniversary of the pilot model of the Automatic Computing Engine, an electronic computer designed primarily by Turing.

Further Reading

X-MACHINES AND THE HALTING PROBLEM: BUILDING A SUPER-TURING MACHINE. Mike Stannett in *Formal Aspects of Computing*, Vol. 2, pages 331–341; 1990.

INTELLIGENT MACHINERY. Alan Turing in *Collected Works of A. M. Turing: Mechanical Intelligence*. Edited by D. C. Ince. Elsevier Science Publishers, 1992.

COMPUTATION BEYOND THE TURING LIMIT. Hava T. Siegelmann in *Science*, Vol. 268, pages 545–548; April 28, 1995.

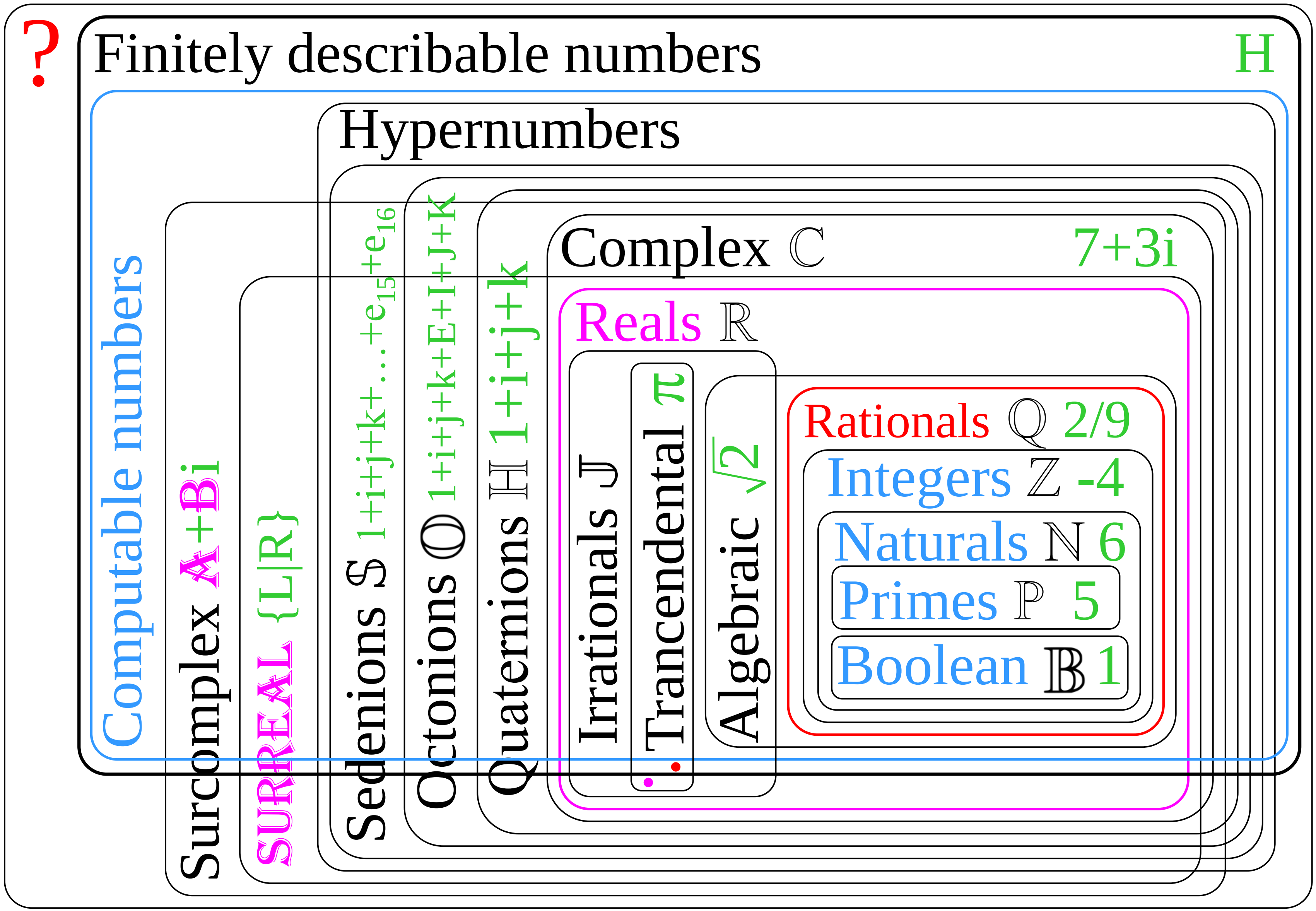
ON ALAN TURING'S ANTICIPATION OF CONNECTIONISM. B. Jack Copeland and Diane Proudfoot in *Synthese*, Vol. 108, No. 3, pages 361–377; March 1996.

TURING'S O-MACHINES, SEARLE, PENROSE AND THE BRAIN. B. Jack Copeland in *Analysis*, Vol. 58, No. 2, pages 128–138; 1998.

THE CHURCH-TURING THESIS. B. Jack Copeland in *The Stanford Encyclopedia of Philosophy*. Edited by Edward N. Zalta. Stanford University, ISSN 1095-5054. Available at http://plato.stanford.edu on the World Wide Web.

"BENEDICT CUMBERBATCH IS OUTSTANDING"
RADIO TIMES
"THE BEST BRITISH FILM OF THE YEAR"
★★★★★
THE INDEPENDENT
"AN INSTANT CLASSIC"
★★★★★
GLAMOUR
"A SUPERB THRILLER"
★★★★
EMPIRE
★★★★
TIME OUT
★★★★
THE TIMES
BENEDICT CUMBERBATCH
KEIRA KNIGHTLEY
THE IMITATION GAME
12A MODERATE SEX REFERENCES
BASED ON THE INCREDIBLE TRUE STORY
/ImitationGameUK
IN CINEMAS NOVEMBER 14
Extra credit!

Generalized Numbers



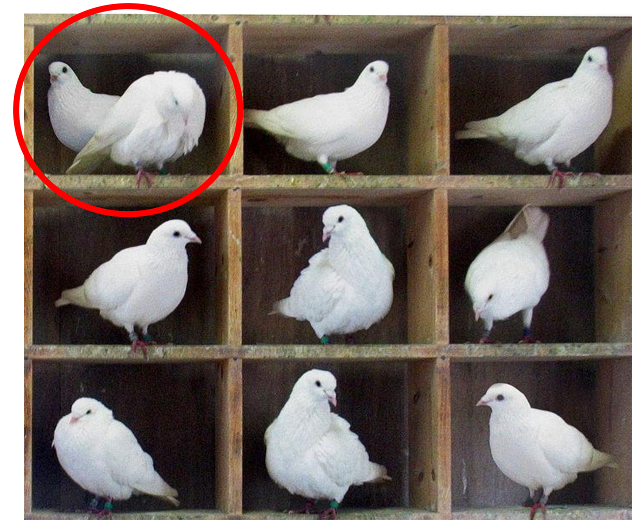
Theorem: some real numbers are not finitely describable!

Theorem: some finitely describable real numbers are not computable!

JOHANNES KEPLER'S UPHILL BATTLE

Pigeon-Hole Principle

- J. Dirichlet (1834)
- "Drawer principle"
- "Shelf Principle"
- "Box principle"



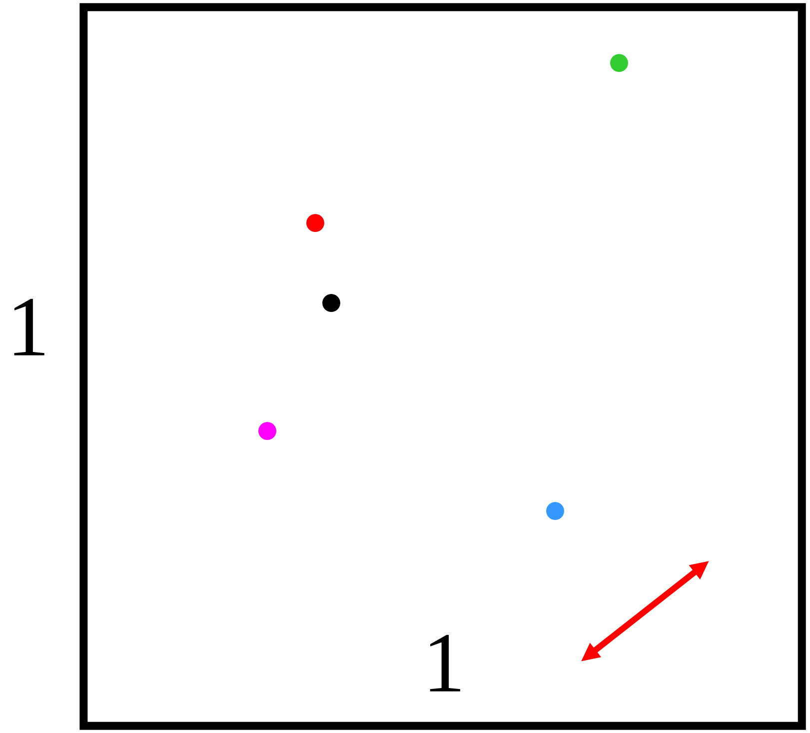
Theorem (pigeon-hole): There is no injective (1-to-1) function from a finite set (domain) to a smaller finite set (range).

Generalization:

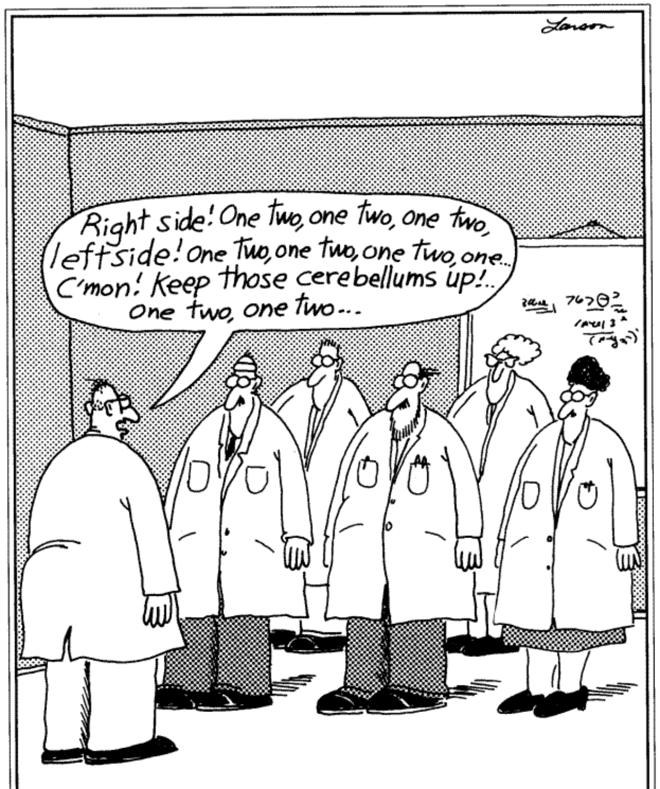
N objects placed in M containers; then:

- at least 1 container must hold $\geq \left\lceil \frac{N}{M} \right\rceil$
- at least 1 container must hold $\leq \left\lfloor \frac{N}{M} \right\rfloor$

Problem: Given any five points in/on the unit square, is there always a pair with distance $\leq \frac{1}{\sqrt{2}}$?

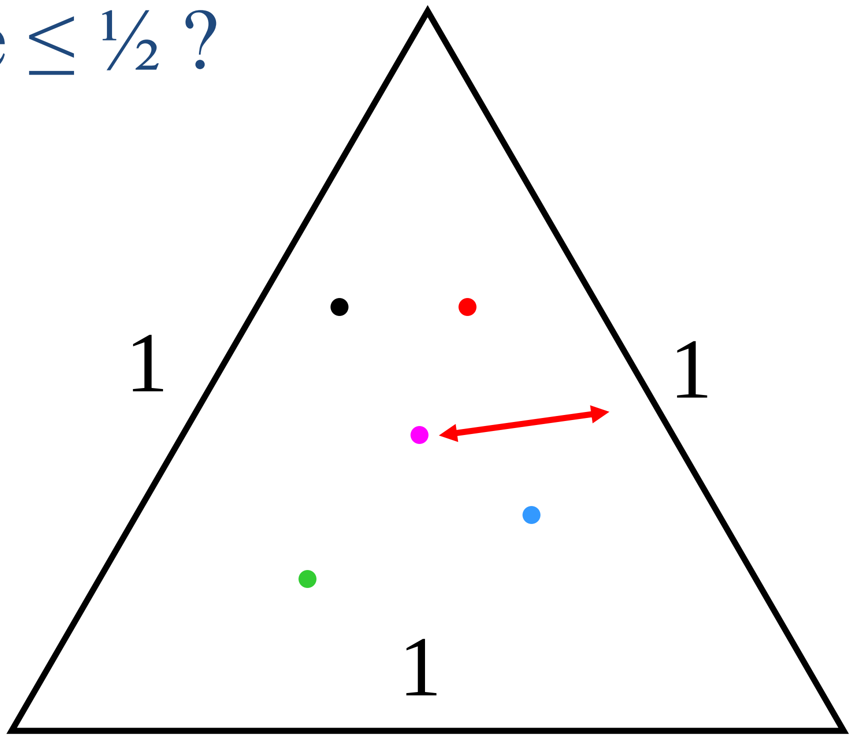


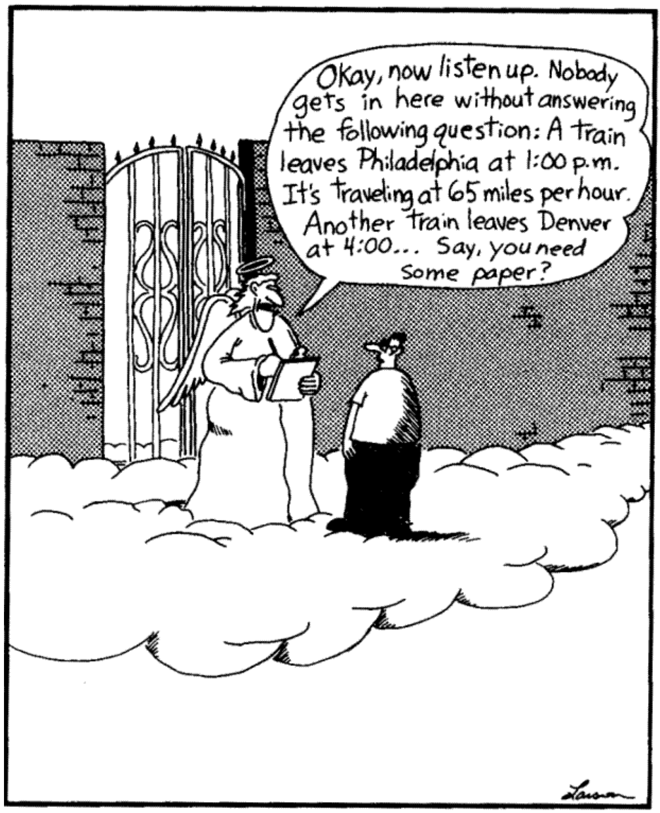
- What approaches fail?
- What techniques work and why?
- Lessons and generalizations



Brain aerobics

Problem: Given any five points in/on the unit equilateral triangle, is there always a pair with distance $\leq \frac{1}{2}$?

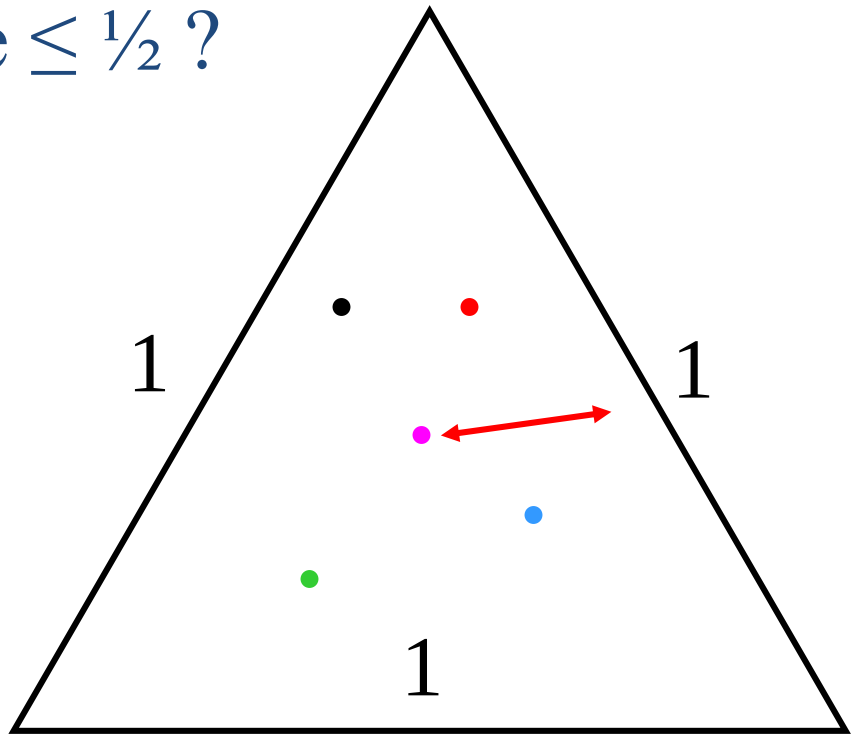




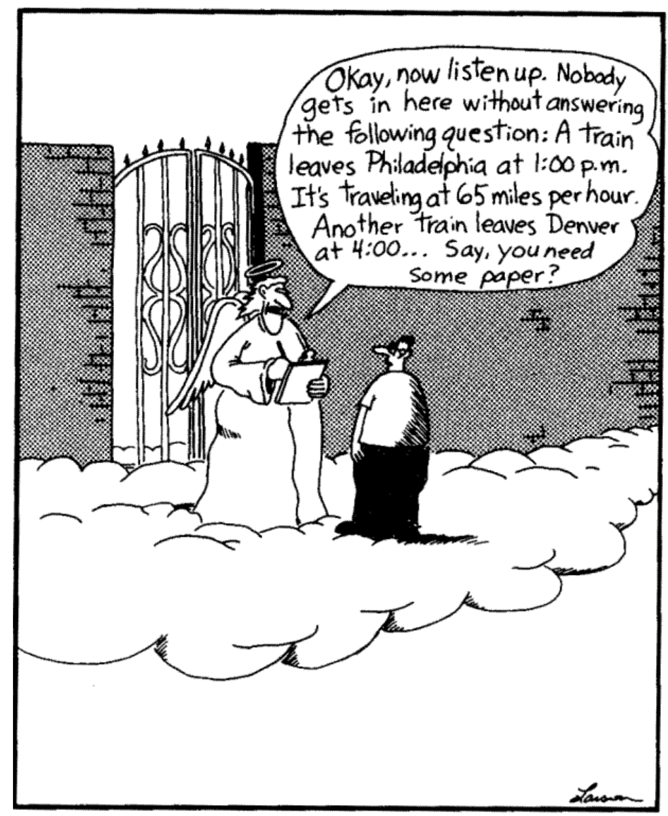
Math phobic's nightmare

- What approaches fail?
- What techniques work and why?
- Lessons and generalizations

Problem: Given any five points in/on the unit equilateral triangle, is there always a pair with distance $\leq$ ½ ?

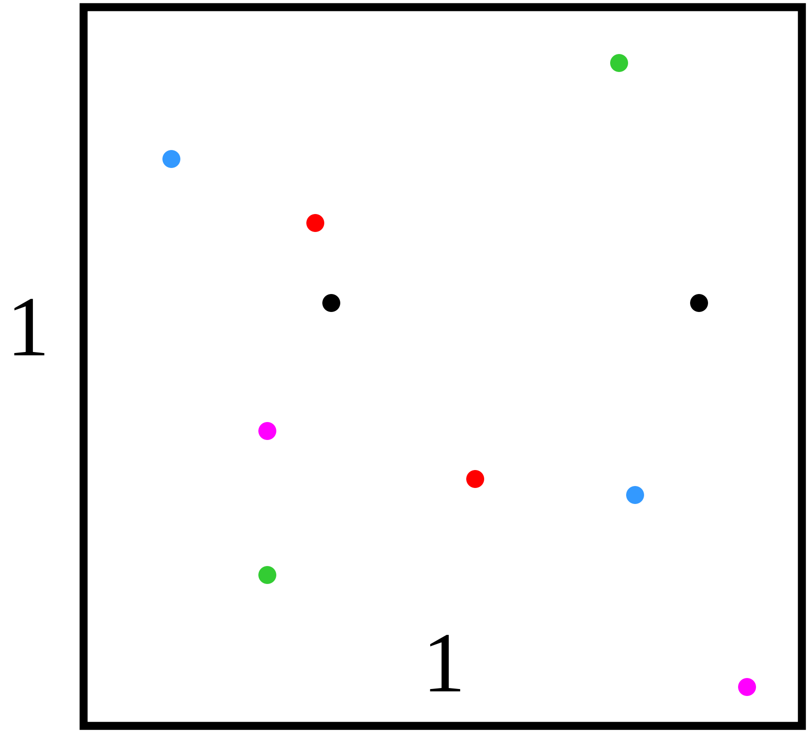


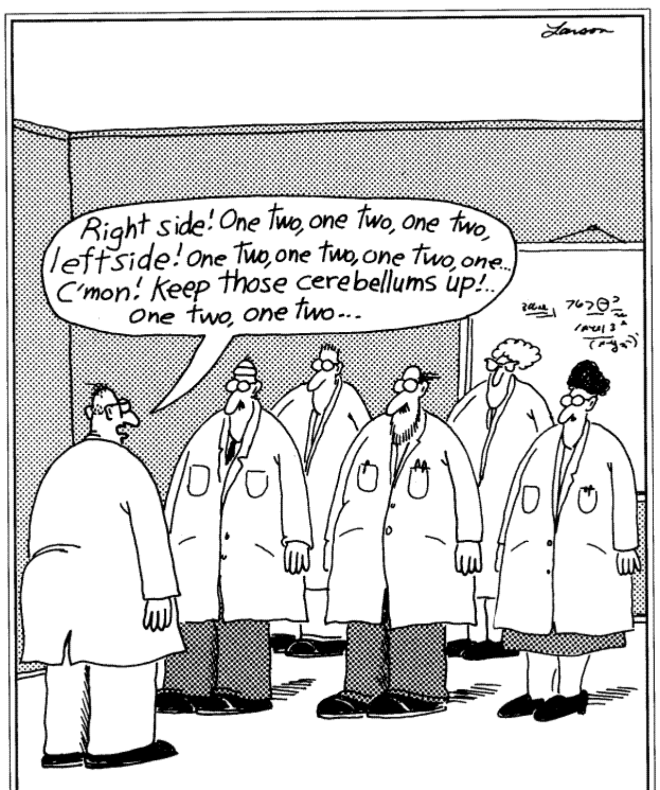
- What approaches fail?
- What techniques work and why?
- Lessons and generalizations



Math phobic's nightmare

Problem: Given any ten points in/on the unit square, what is the maximum pairwise distance?





Brain aerobics

- What approaches fail?
- What techniques work and why?
- Lessons and generalizations

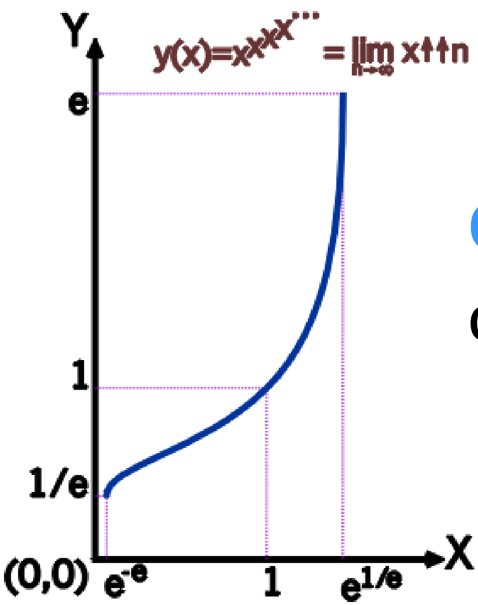
Problem: Solve the following equation for X:

$$X^{\overbrace{X^{X^{X^{\cdot^{\cdot^{\cdot}}}}}}^{=\,2}} = 2 \quad \Rightarrow X^2 = 2 \quad \Rightarrow X = \sqrt{2}$$

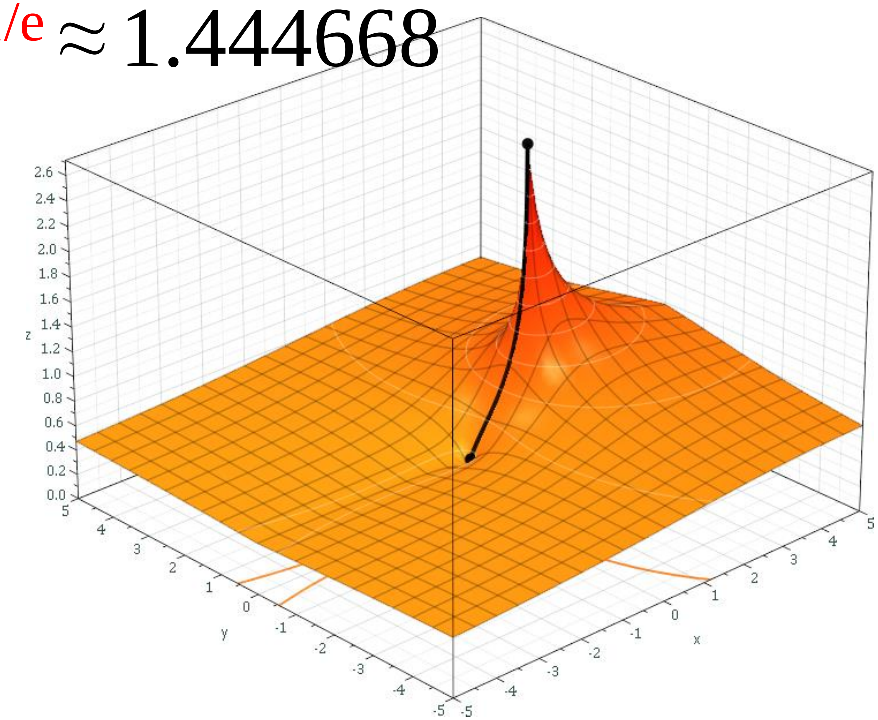
where the stack of exponentiated x's extends forever.

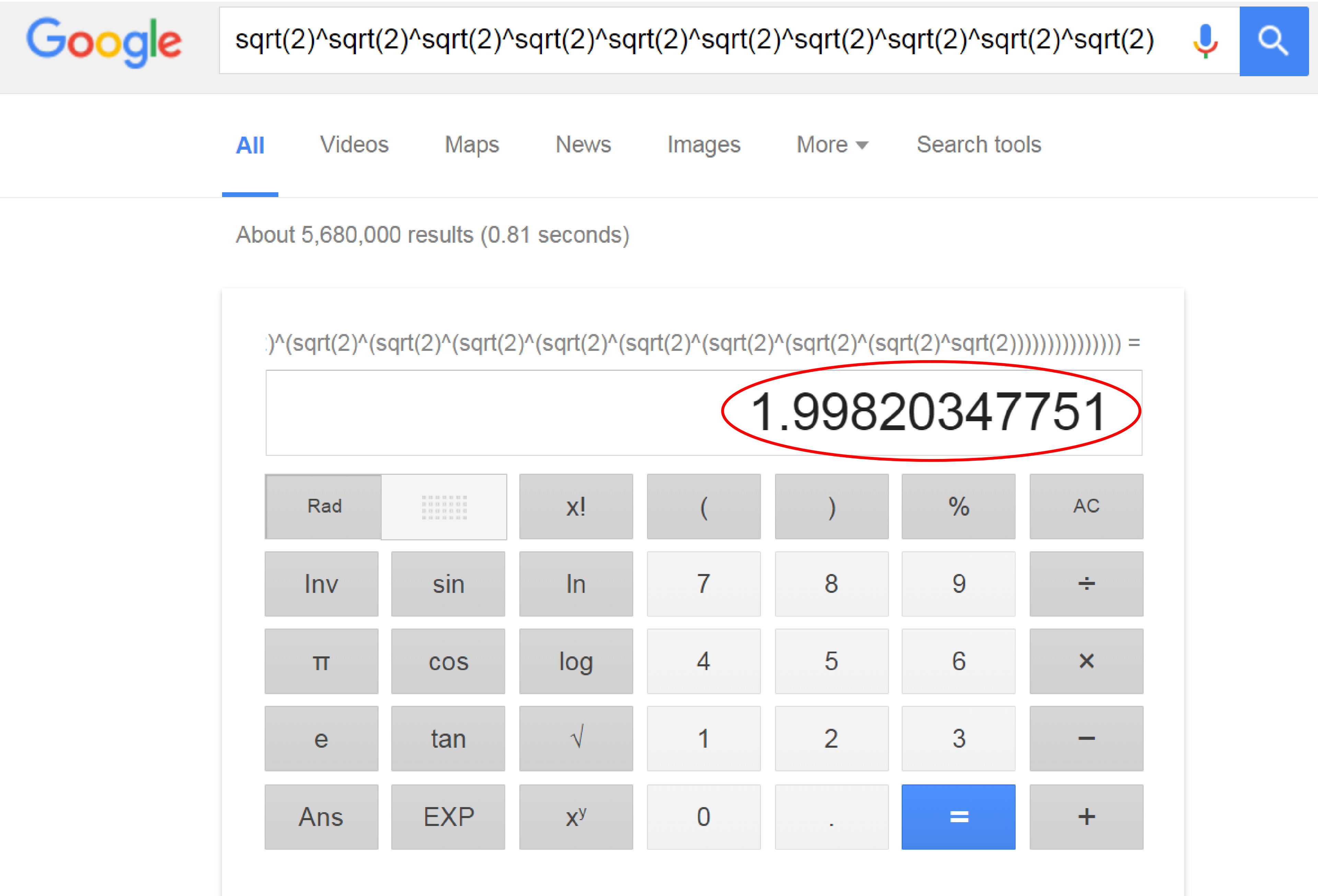
This "power tower" converges for:

$$0.065988 \approx e^{-e} < X < e^{1/e} \approx 1.444668$$



Generalization to complex numbers:




Google
sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)^sqrt(2)
All
Videos
Maps
News
Images
More
Search tools
About 5,680,000 results (0.81 seconds)
)^(sqrt(2)^(sqrt(2)^(sqrt(2)^(sqrt(2)^(sqrt(2)^(sqrt(2)^(sqrt(2)^(sqrt(2)^sqrt(2))))))))))))) =
1.99820347751
Rad
x!
(
)
%
AC
Inv
sin
ln
7
8
9
÷
π
cos
log
4
5
6
×
e
tan
√
1
2
3
−
Ans
EXP
xʸ
0
.
=
+