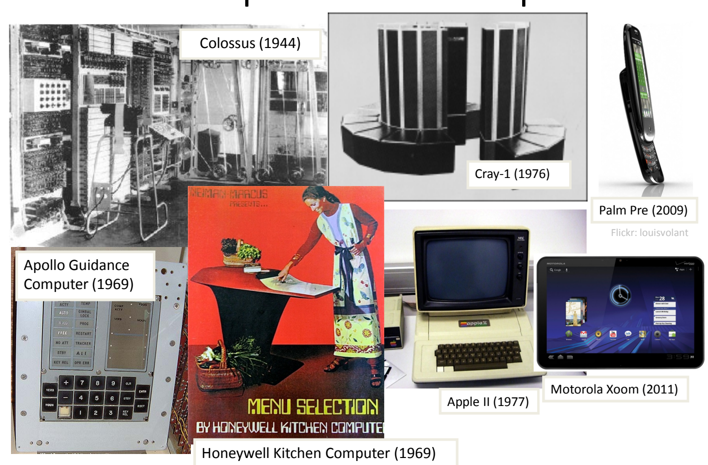
Biggest Number Game

- When I say "GO", write down the biggest number you can in 30 seconds.
- Requirement:
  - Must be an exact number
  - Must be defined mathematically
- Biggest number wins!

Countdown Clock

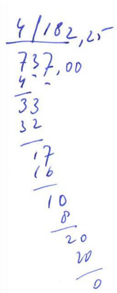
What's so special about computers?



Toaster Science?

"Computers" before WWII

Mechanical Computing



Modeling Computers

Input

Without it, we can’t describe a problem

Output

Without it, we can’t get an answer

Processing

Need a way of getting from the input to the output

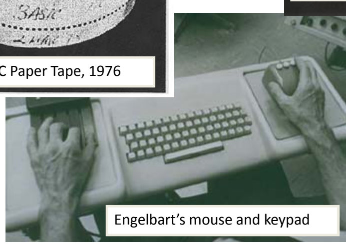
Memory

Need to keep track of what we are doing

Modeling Input

Altair BASIC Paper Tape, 1976

Punch Cards



Engelbart’s mouse and keypad

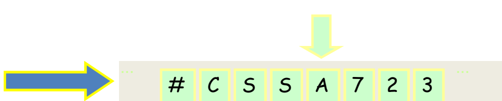
Apple's Newton MessagePad

Turing’s Model

“Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child’s arithmetic book.”

Alan Turing, *On computable numbers, with an application to the Entscheidungsproblem*, 1936

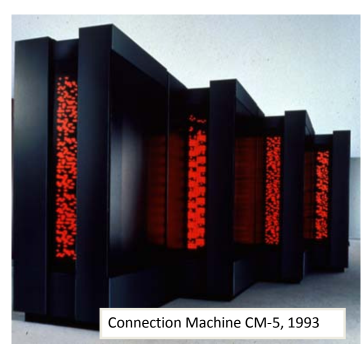
Modeling Pencil and Paper



How long should the tape be?

Modeling Output

- Blinking lights are cool, but hard to model
- Use the tape: output is what is written on the tape at the end



Connection Machine CM-5, 1993

Modeling Processing (Brains)

Look at the current state of the computation

Follow simple rules about what to do next

Modeling Processing

Evaluation Rules

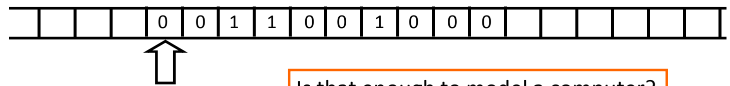
Given an input on our tape, how do we evaluate to produce the output

What do we need:

Read what is on the tape at the current square

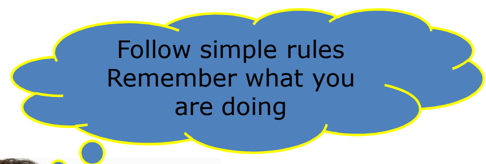
Move the tape one square in either direction

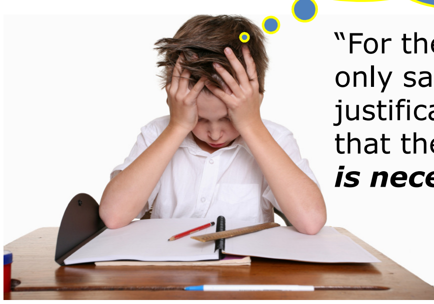
Write into the current square



Is that enough to model a computer?

Modeling Processing (Brains)

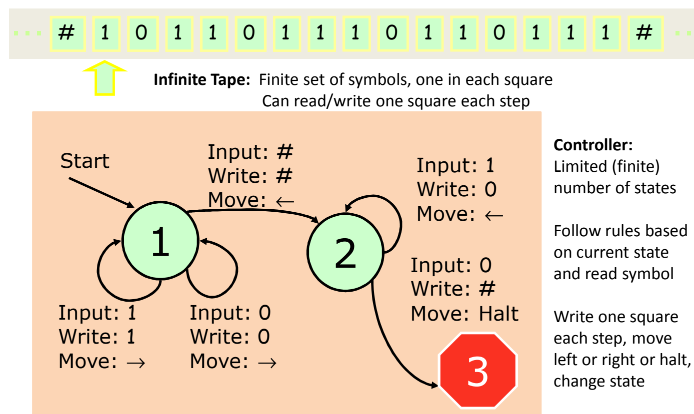




"For the present I shall only say that the justification lies in the fact that the ***human memory is necessarily limited***."

Alan Turing

Turing's Model: **Turing Machine**



Church-Turing Thesis

- All mechanical computers are equally powerful*

  *Except for practical limits like memory size, time, display, energy, etc.

- There exists some Turing machine that can simulate *any* mechanical computer
- *Any* computer that is powerful enough to simulate a Turing machine, can simulate any mechanical computer

Power of Turing Machine

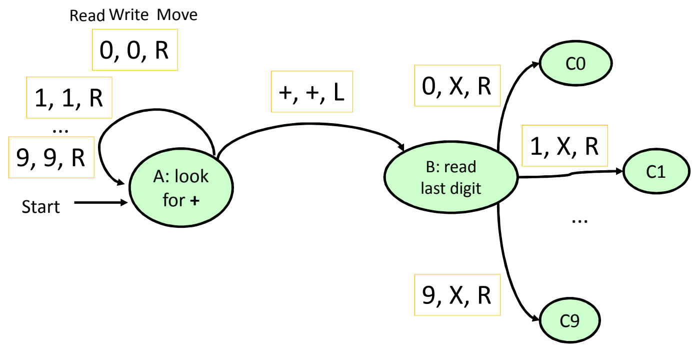
- Can it add?
- Can it carry out any computation?
- Can it solve any problem?

Performing Addition

- **Input:** a two sequences of digits, separated by + with # at end.
  - e.g., # 1 2 9 3 5 2 + 6 3 5 9 4 #
- **Output:** sum of the two numbers
  - e.g., # 1 9 2 9 4 6 #

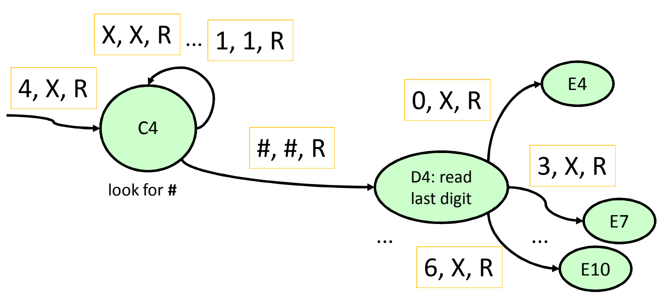
Addition Program

Find the rightmost digit of the first number:



Addition, Continued

Find the rightmost digit of the second number:



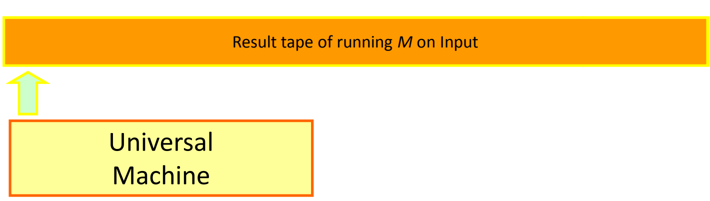
Must duplicate this for each first digit – states keep track of first digit!

Power of Turing Machine

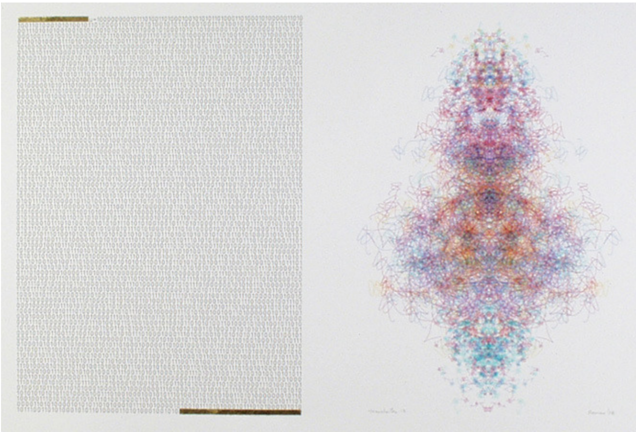
✓ Can it add?

- Can it carry out any computation?
- Can it solve any problem?

Universal Machine

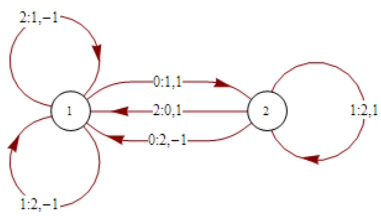


A Universal Turing Machine can simulate any Turing Machine running on any Input!



Manchester Illuminated Universal Turing Machine, #9
from http://www.verostko.com/manchester/manchester.html

Universal Computing Machine



2-state, 3-symbol Turing machine proved universal by Alex Smith in 2007

What This Means

- Your cell phone, watch, iPod, etc. has a processor powerful enough to simulate a Turing machine
- A Turing machine can simulate the world's most powerful supercomputer
- Thus, your cell phone can simulate the world's most powerful supercomputer (it'll just take a lot longer and will run out of memory)

In **Theory**

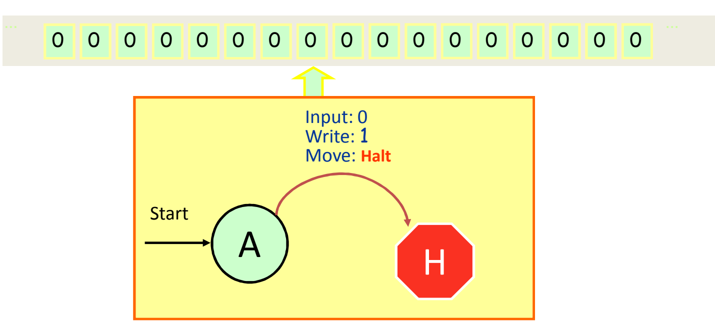
Are there problems computers can't solve?

The "Busy Beaver" Game

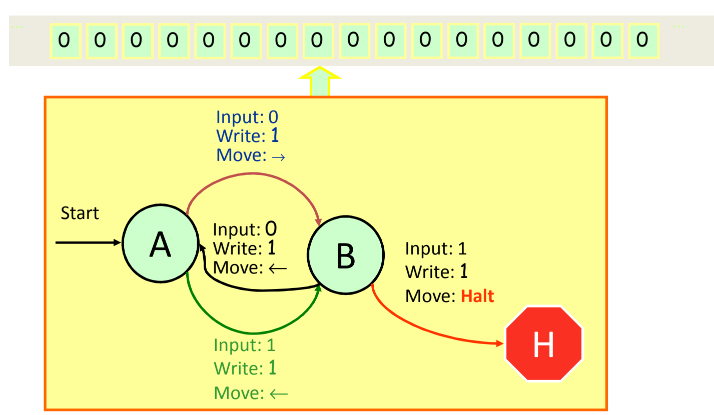
- Design a Turing Machine that:
  - Uses two symbols (e.g., "0" and "1")
  - Starts with a tape of all "0"s
  - Eventually halts (can't run forever)
  - Has *N* states
- Goal: machine runs for as many steps as possible before **eventually** halting

Tibor Radó, 1962

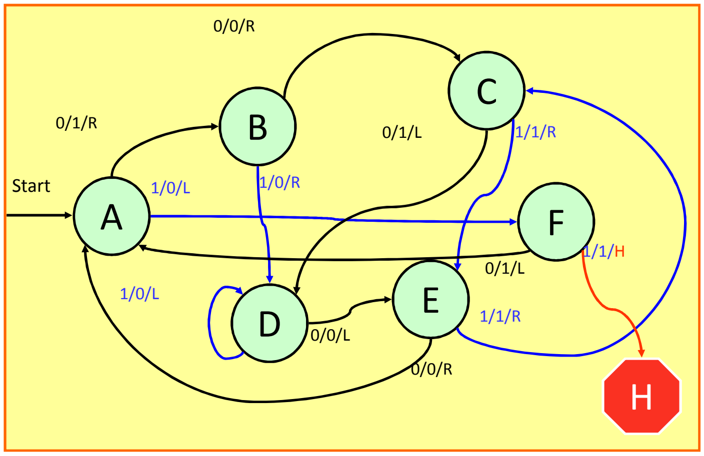
Busy Beaver: N = 1



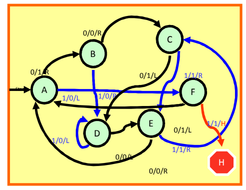
BB(1) = 1 Most steps a 1-state machine that halts can make



BB(2) = 6



6-state machine found by Buntrock and Marxen, 2001



Best found before 2001, only 925 digits!

In Dec 2007, Terry and Shawn Ligocki beat this: 2879 digits!

300232771652356282895510301834134018514775433724675250037338
180173521424076038326588191208297820287669898401786071345848
280422383492822716051848585583668153797251438618561730209415
487685570078538658757304857487222040030769844045098871367087
615079138311034353164641077919209890837164477363289374225531
955126023251172259034570155087303683654630874155990822516129
938425830691378607273670708190160525534077040039226593073997
923170154775358629850421712513378527086223112680677973751790
032937578520017666792246839908855920362933767744760870128446
883455477806316491601855784426860769027944542798006152693167
452821336689917460886106486574189015401194034857577718253065
541632656334314242325592486700118506716581303423271748965426
160409797173073716688827281435904639445605928175254048321109
306002474658968108793381912381812336227992839930833085933478
853176574702776062858289156568392295963586263654139383856764
728051394965554409688456578122743296319960808368094536421039
149584946758006509160985701328997026301708760235500239598119
410592142621669614552827244429217416465494363891697113965316
892660611709290048580677566178715752354594049016719278069832
866522332923541370293059667996001319376698551683848851474625
152094567110615451986839894490885687082244978774551453204358
588661593979763935102896523295803940023673203101744986550732
496850436999753711343067328676158146269292723375662015612826
924105454849658410961574031211440611088975349899156714888681
952366018086246687712098553077054825367434062671756760070388
922117434932633444773138783714023735898712790278288377198260
380065105075792925239453450622999208297579584893448886278127
629044163292251815410053522246084552761513383934623129083266
949377380950466643121689746511996847681275076313206

(1730 digits)

Busy Beaver Numbers

BB(1) = 1
BB(2) = 6
BB(3) = 21
BB(4) = 107
BB(5) = Unknown!

Best so far is 47,176,870

BB(6) > 10^{2879}

Discovered 2007

Winning the "Biggest number" game: BB(BB(BB(BB(111111111))))

Computing Busy Beaver Numbers

- Input: N (number of states)
- Output: BB(N)
  - The maximum number of steps a Turing Machine with N states can take before halting

Is it possible to design a Turing Machine that solves the Busy Beaver Problem?

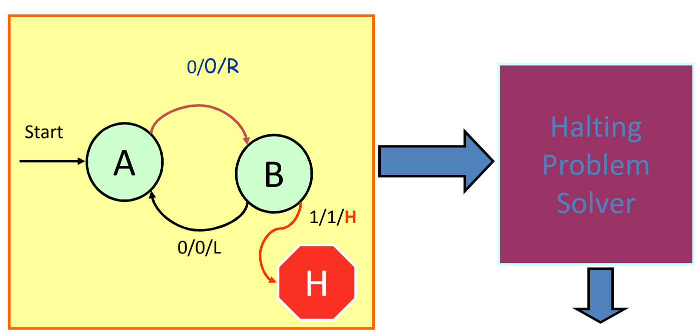
The Halting Problem

- Input: a description of a Turing Machine
- Output: "1" if it eventually halts, "0" if it never halts, starting on a tape full of "0"s.

Is it possible to design a Turing Machine that **solves** the Halting Problem?

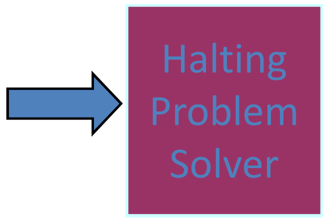
"Solves" means for all inputs, the machine finishes and produces the right answer.

Example

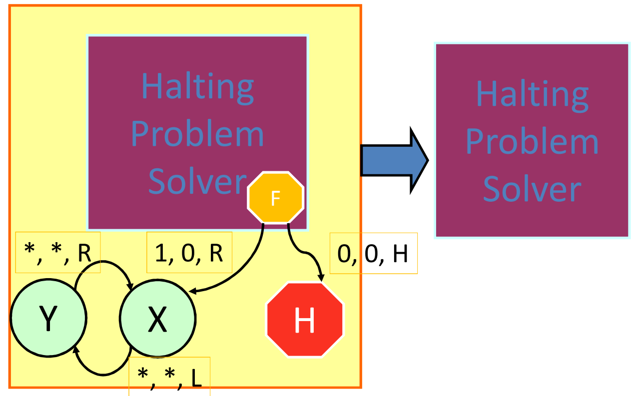


0 (it never halts)

Example



Impossibility Proof!



Impossible to make Halting Problem Solver

- If it outputs "0" on the input, the input machine would halt (so "0" cannot be correct)
- If it outputs "1" on the input, the input machine never halts (so "1" cannot be correct)

If it halts, it doesn't halt!
If it doesn't halt, it halts!

Busy Beaver is Impossible Too!

- If you could solve it, could solve Halting Problem:
  - Input machine has N states
  - Compute BB(N)
  - Simulate input machine for BB(N) steps
  - If it ever halts, it must halt by now
- ... but we know that is impossible, so it must be impossible to computer BB(N)

The BB numbers are so big you can't even compute them!

Recap

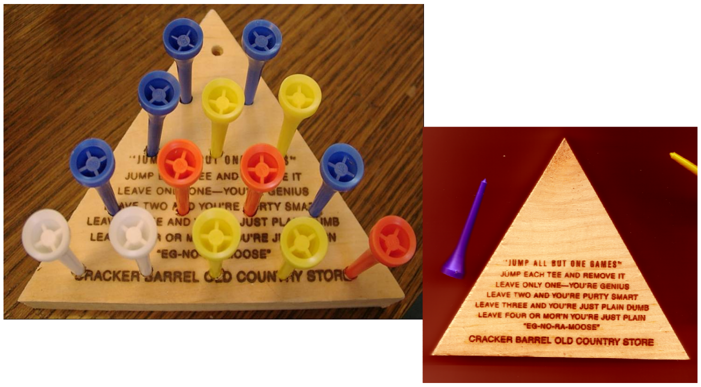
- A *computer* is something that can carry out well-defined steps:
  - Read and write on scratch paper, follow rules, keep track of state
- All computers are equally powerful
  - If a machine can simulate any step of another machine, it can simulate the other machine (except for physical limits)
  - What matters is the *program* that defines the steps

In **Practice**

Are there problems (real) computers can't solve?

Sure…all the undecidable problems. Are there others?

Pegboard Problem



Pegboard Problem

Input: a configuration of n pegs on a cracker barrel style pegboard (of size large enough to hold the pegs)

Output: if there is a sequence of jumps that leaves a single peg, output that sequence of jumps. Otherwise, output **false**.

How hard is the Pegboard Problem?

How much work is the Pegboard Problem?

Upper bound: $O(n!)$

Try all possible permutations

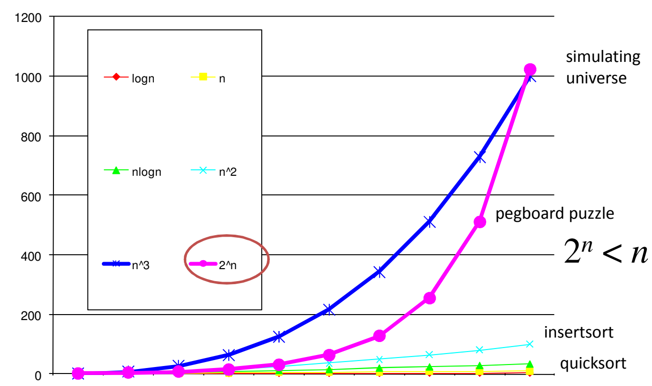
Lower bound: $\Omega(n)$

Must at least look at every peg

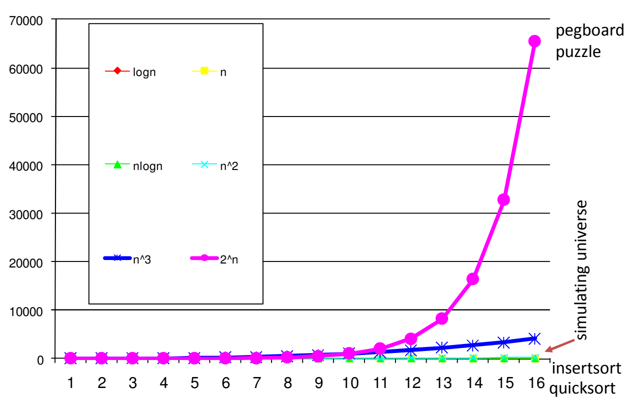
Tight bound: $\Theta(?)$

No one knows!

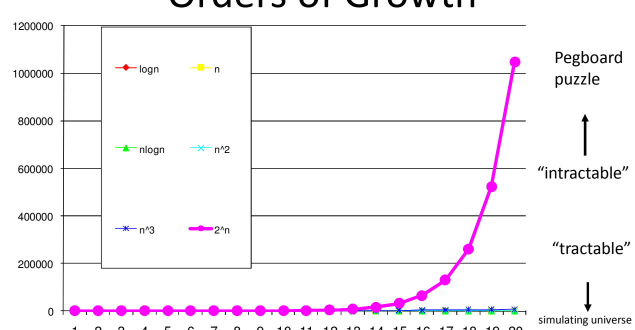
Orders of Growth



Orders of Growth



Orders of Growth



I do nothing that a man of unlimited funds, superb physical endurance, and maximum scientific knowledge could not do.
– Batman (may be able to solve intractable problems, but computer scientists can only solve tractable ones for large n)

Complexity Class P
"Tractable"

Class P: problems that can be solved in a *polynomial* ($< an^k$ for some constants a and k) number of steps by a deterministic TM.

Easy problems like sorting, genome alignment, and simulating the universe are all in **P**.

Complexity Class NP

Class NP: Problems that can be solved in a polynomial number of steps by a *nondeterministic* TM.

Omnipotent: If we could try all possible solutions at once, we could identify the solution in polynomial time.

Omniscient: If we had a magic guess-correctly procedure that makes every decision correctly, we could devise a procedure that solves the problem in polynomial time.

NP Problems

- Can be solved by just trying all possible answers until we find one that is right
- Easy to quickly check if an answer is right
  - Checking an answer is in **P**
- The pegboard problem is in **NP**

  We can easily try ~n! different answers

  We can check if a guess is correct in $O(n)$ (check all n jumps are legal)

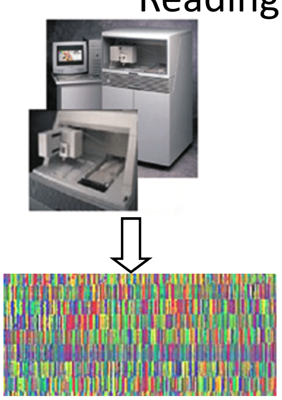
Is the Pegboard Problem in **P**?

No one knows!

We can't find a $O(n^k)$ solution.

We can't prove one doesn't exist.

Reading the Genome



Whitehead Institute, MIT

Gene Reading Machines

- One read: about 700 base pairs
- But…don't know where they are on the chromosome

Read 3 TACCCGTGATCCA

Read 2 TCCAGAATAA

Read 1 ACCAGAATACC

Actual Genome AGGCATACCAGAATACCCGTGATCCAGAATAAGC

Genome Assembly

Read 1 ACCAGAATACC

Read 2 TCCAGAATAA

Read 3 TACCCGTGATCCA

Input: Genome fragments (but without knowing where they are from)

Ouput: The full genome

Genome Assembly

Read 1 ACCAGAATACC

Read 2 TCCAGAATAA

Read 3 TACCCGTGATCCA

Input: Genome fragments (but without knowing where they are from)

Ouput: The smallest genome sequence such that all the fragments are substrings.

Common Superstring

Input: A set of n substrings and a maximum length k.

Output: A string that contains all the substrings with total length $\leq k$, or no if no such string exists.

ACCAGAATACC

TCCAGAATAA

TACCCGTGATCCA

$n = 26$

⇨

ACCAGAATACC

TCCAGAATAA

TACCCGTGATCCA

ACCAGAATACCCGTGATCCAGAATAA

Common Superstring

Input: A set of n substrings and a maximum length k.

Output: A string that contains all the substrings with total length $\leq k$, or no if no such string exists.

ACCAGAATACC

TCCAGAATAA

TACCCGTGATCCA

$n = 25$

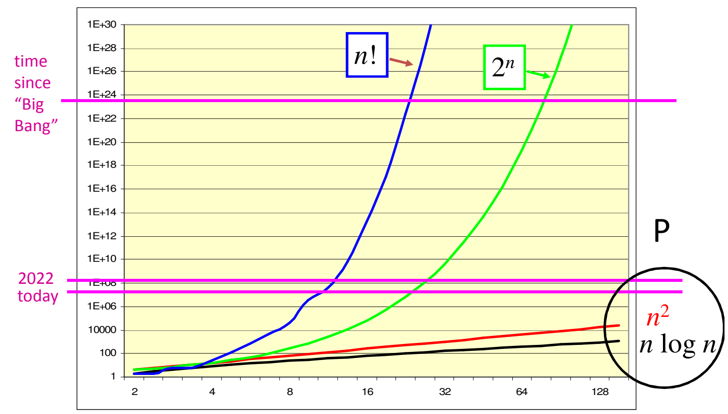
⇨ **Not possible**

Common Superstring

- In **NP:**
  - Easy to verify a "yes" solution: just check the letters match up, and count the superstring length
- In **NP-Complete:**
  - Similar to Pegboard Puzzle!
  - Could transform Common Superstring problem instance into Pegboard Puzzle instance!

Intractable Problems

log-log scale



Complexity Classes

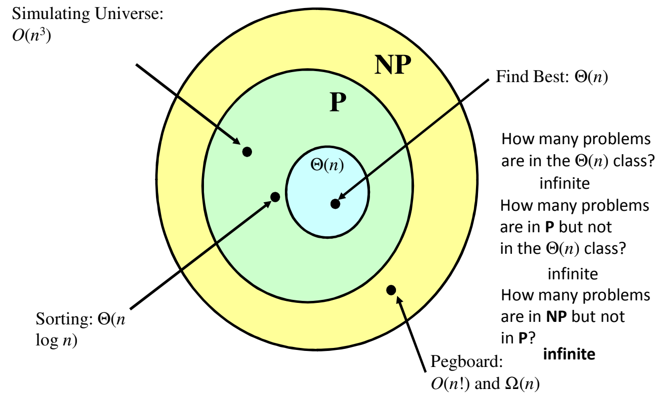
Class P: problems that can be solved in polynomial time by deterministic TM

Easy problems like simulating the universe are all in **P**.

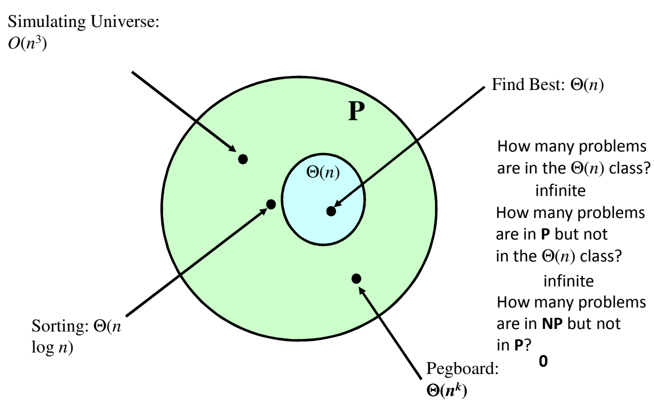
Class NP: problems that can be solved in polynomial time by a nondeterministic TM.

Includes all problems in **P** and some problems possibly outside **P** like the Pegboard puzzle.

Problem Classes if P ≠ NP:



Problem Classes if P = NP:



P = NP?

- Is P different from NP: is there a problem in NP that is not also in P
  - If there is one, there are infinitely many
- Is the "hardest" problem in NP also in P
  - If it is, then every problem in NP is also in P
- The most famous unsolved problem in computer science and math
  - Listed first on Millennium Prize Problems

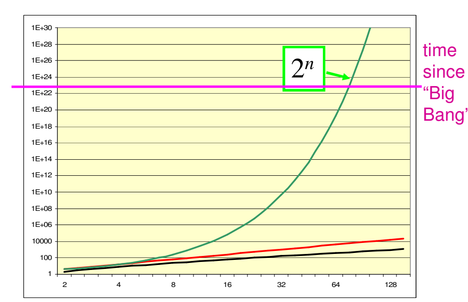
NP-Complete Problems

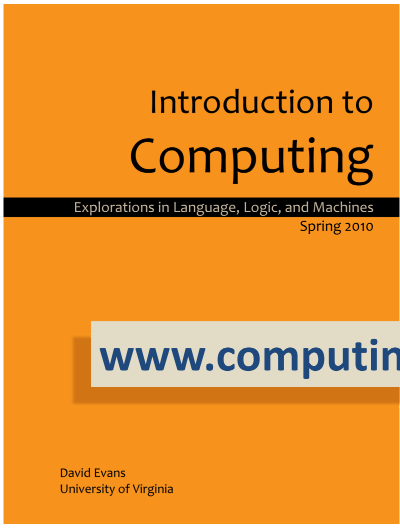
- Easy way to solve by trying all possible guesses
- If given the "yes" answer, quick (in P) way to check if it is right
- If given the "no" answer, no quick way to check if it is right
  - No solution (can't tell there isn't one)
  - No way (can't tell there isn't one)

This part is hard to prove: requires showing you could use a solution to the problem to solve a known NP-Complete problem.

Give up?

No way to solve an NP-Complete problem (best known solutions being $O(2^n)$ for $n \approx$ 20 Million)





Questions
/
Plug

www.computingbook.org