Optimization Theory

Rod Van Meter

Lecture 11: Computational Complexity Theory

December 13, 2016

@慶應湘南藤沢

with help from

Eisuke Abe 阿部英介

Science

Daily News

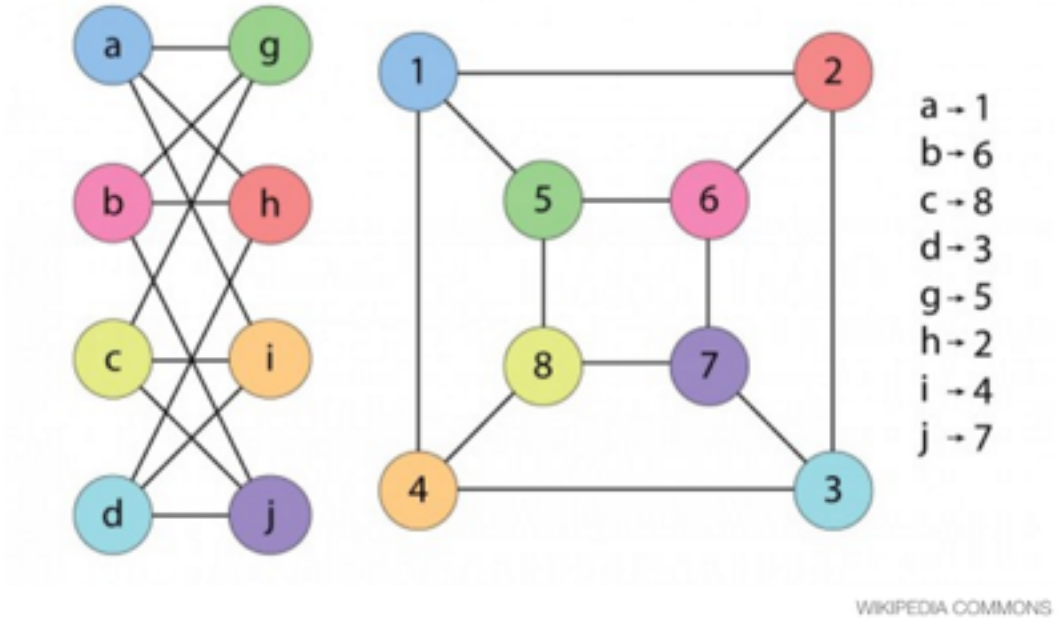
Enter Search Term

AAAS NEWS SCIENCE JOURNALS CAREERS MULTIMEDIA

News Home Hot Topics Categories From the Magazine ScienceInsider Sc

News > Math > Mathematician claims breakthrough in complexity theory

LATEST NEWS



WIKIPEDIA COMMONS

In the graph isomorphism problem, the challenge is to determine whether two apparently different graphs can be rearranged to be identical, as these two can.

Mathematician claims breakthrough in complexity theory

László Babai

http://people.cs.uchicago.edu/~laci/

http://news.sciencemag.org/math/2015/11/mathematician-claims-breakthrough-complexity-theory

http://jeremykun.com/2015/11/12/a-quasipolynomial-time-algorithm-for-graph-isomorphism-the-details/

What We Want to Know

- What problems is it possible for a computer to solve for us?
- In particular, as *problem size* scales up
- Known as *complexity class*
- *Caveat: constant factors matter, too!*

Alan M. Turing

Alonzo Church

"All computing machines above a minimum level of capability are equivalent." (1930s)

"Not everything can be computed." (Turing, 1936)

Juris Hartmanis & Richard E. Stearns

"We can characterize algorithms according to their complexity." (1965)

Richard M. Karp

"Many problems can be mapped to other, apparently different, problems with 'only' a polynomial overhead." (1972)

(photos from Wikipedia, except Turing)

(quotes are artificial paraphrases)

Lecture Outline

- O(.) notation
- Classical Computational Complexity Review
  - P, PSPACE, EXP
  - P v. NP
- Quantum Computational Complexity

What Is Computational Complexity?

An algorithm or a problem has a *computational complexity class* (and, more specifically, its particular computational complexity) that describes how the resources needed to solve the problem grow as the size of the problem grows.

O(.) (Big O) Notation

$$\Omega(\cdot), \Theta(\cdot), O(\cdot)$$

- Already been using it...
- Describes *asymptotic* behavior as the problem size grows.

O(.)

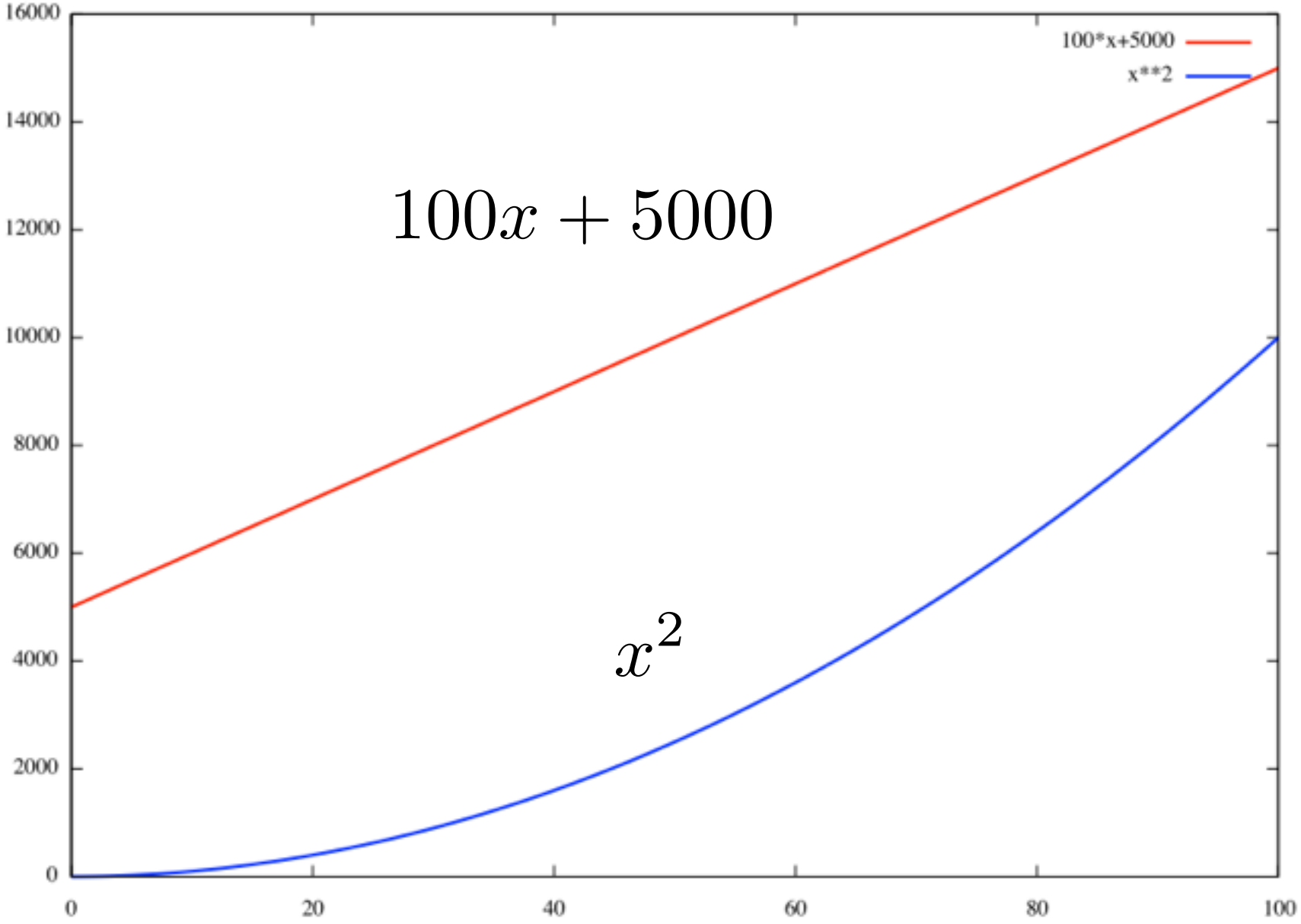
$$f(x) = O(g(x)) \text{ as } x \to \infty$$

means

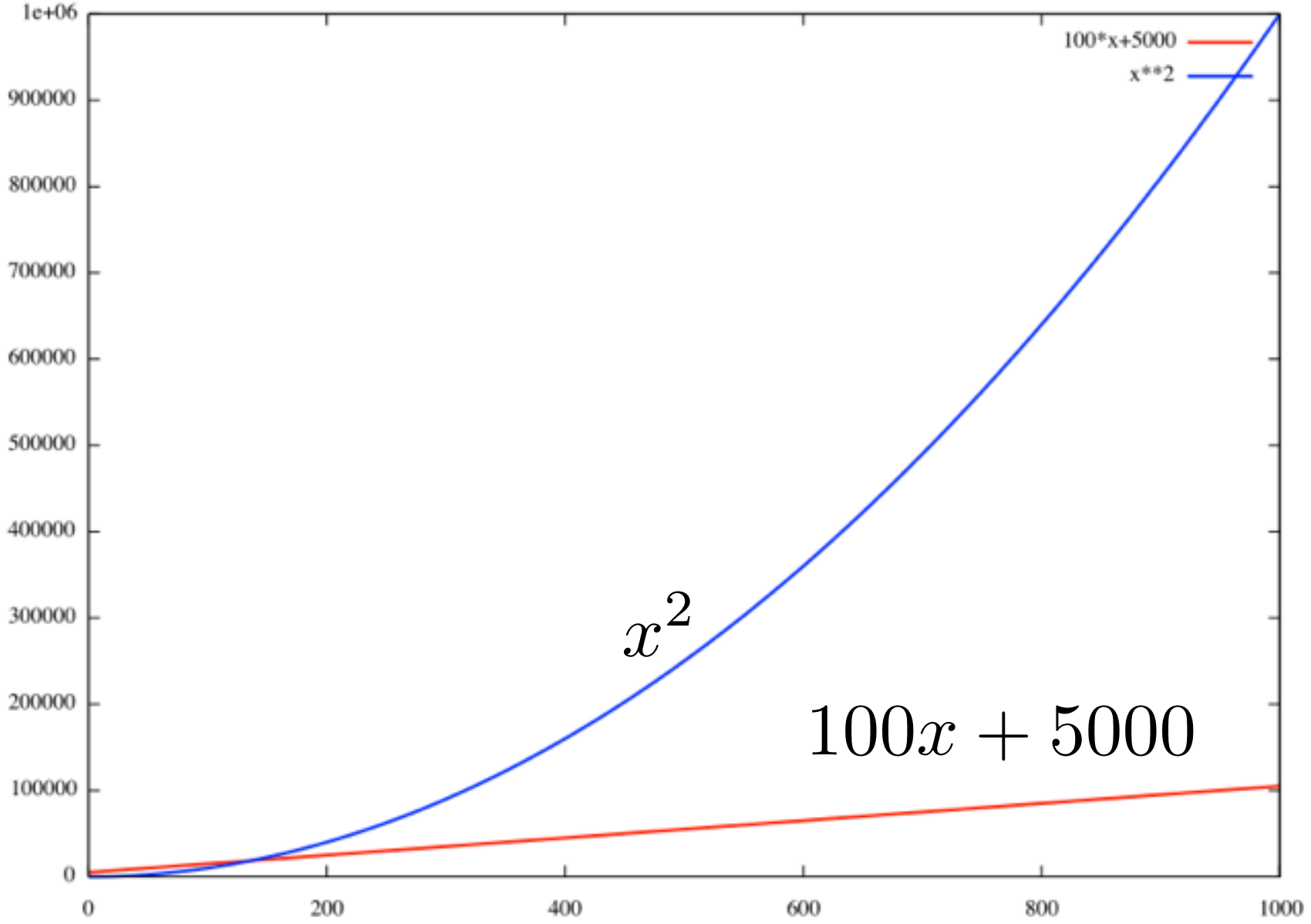
$$|f(x)| \leq M|g(x)| \text{ for all } x > x_0$$

for some positive constant M and some value x_0

Example


16000
14000
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
100*x+5000
x**2
$100x + 5000$
x^2

Example


1e+06
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0
200
400
600
800
1000
100*x+5000
x**2
x^2
$100x + 5000$

Example

So the x^2 curve *dominates* the $100x + 5000$ curve. This is independent of the constants involved. We can say "*f(x)* is oh-of (or big-oh of) (something)"

$$f(x) = 100x + 5000$$

$$f(x) = O(x)$$

$$f(x) = O(x^2)$$

$$f(x) = O(e^x)$$

Example 2

So the x^2 curve *dominates* the $100x + 5000$ curve. This is independent of the constants involved. We can say "*f(x)* is oh-of (or big-oh of) (something)"

$$f(x) = 5x^2$$
$$f(x) = O(x^2)$$
$$f(x) = O(e^x)$$
$$f(x) \neq O(x)$$

Functions can be Complicated

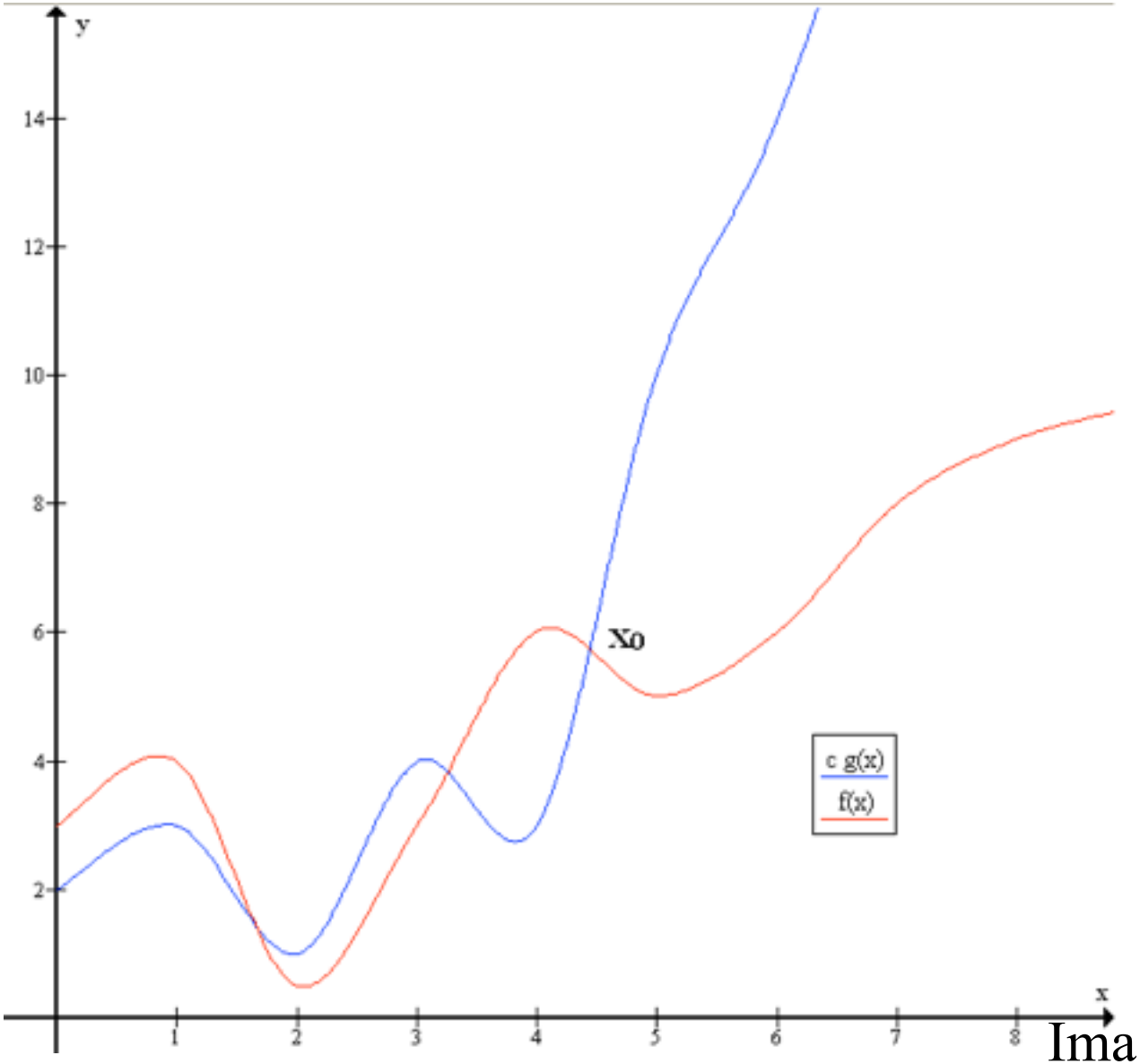


Image from Wikipedia

Applying to Code (1)

Someone write me code for:

* averaging an array of numbers
* dot product of two vectors

for size DIM

This is *O()* what as DIM gets bigger?

Applying to Code (2)

```
for (i = 0 ; i < DIM ; i++) {
  for (j = 0 ; j < DIM ; j++) {
    c[i][j] = 0.0;
    for (k = 0 ; k < DIM ; k++) {
      c[i][j] += a[k][j]*b[j][k];
    }
  }
}
```

This is *O()* what as DIM gets bigger?

Applying to Code (3)

```
for (i = 0 ; i < DIM ; i++) {
  BigHairyFunction(a[i]);
}
```

This is *O()* what as DIM gets bigger?

$$\Omega(\cdot), \Theta(\cdot), O(\cdot)$$

O(.) is “no worse than”

Omega(.) is “no better than”

Theta(.) is “is the same as”
(both O() and Omega())

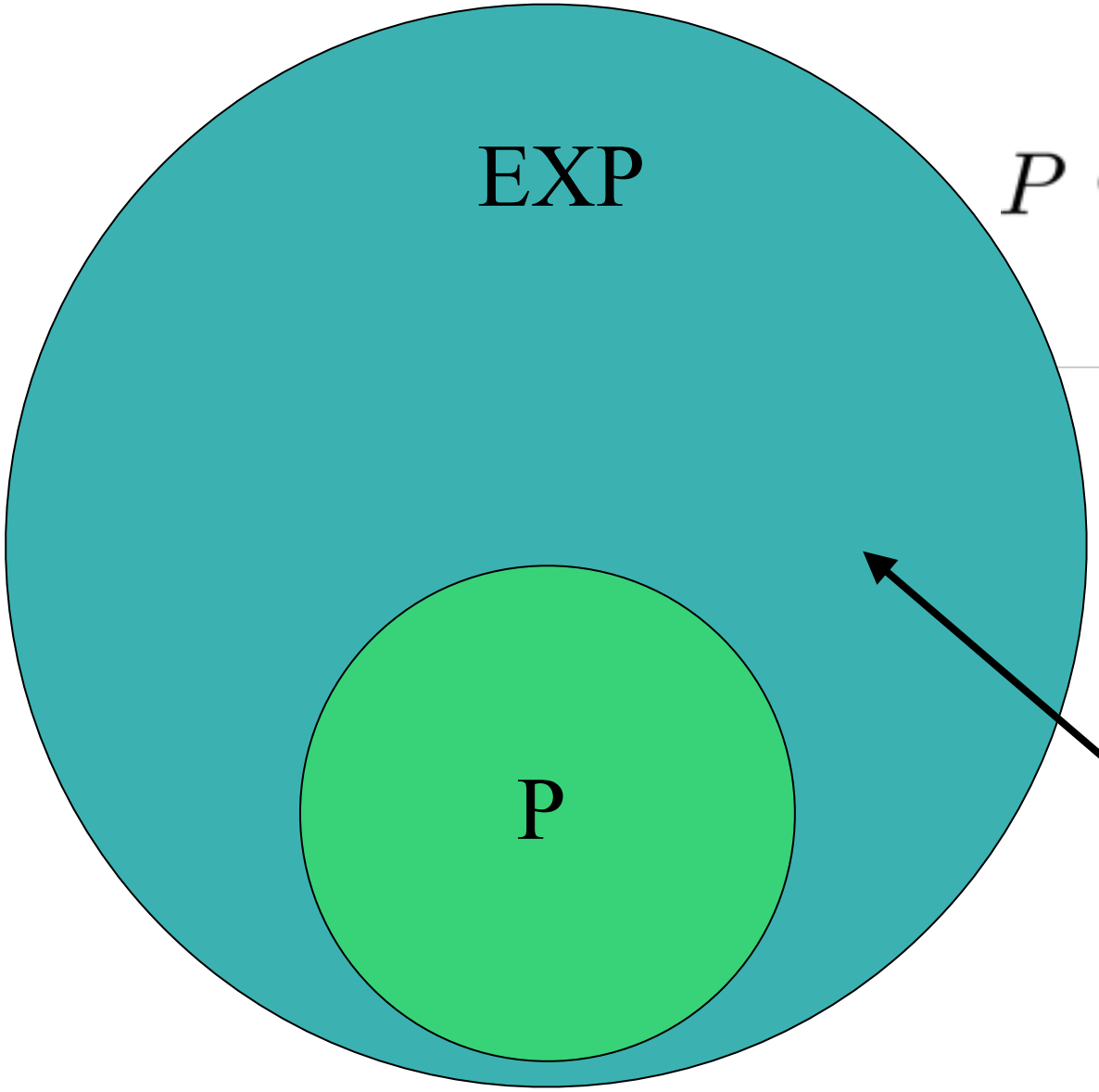
O(.) Notation

- In the real world, constant factors matter!
- Must be defined relative to an architecture
  - One-tape Turing machine, two-tape TM, random-access machine, etc.
  - All are polynomially equivalent, but the polynomials matter!
- Sometimes used for time (circuit depth) with parallel computation; usually represents total computation cost

“Language Accepted by…”

A string is “accepted by” a Turing machine if the machine *halts* when given the string as input. The set of strings accepted by a TM is the *language* accepted by the machine. The TM may return “yes”, “no”, or “no decision” on the string, as long as it halts.

What We're Certain Of



$$P \subsetneq EXPTIME$$

$$P \subsetneq EXP$$

$$P \neq EXP$$

This means that there is *definitely* a problem in this area (inside EXP, outside P).

What Does That Mean?

EXP is *practically* infinite resources.
P is finite resources (space and time).
We know it is possible to solve more problems using infinite (well, very, very large) resources.

(n.b.: there are actually harder things, but for all practical purposes, EXP is the “biggest” thing we need to worry about.)

More Precisely...

- EXP: exponential time on a Turing machine
- P: polynomial time on a TM
  trying *all* possible answers requires only polynomial # of tries, *or* if the # of possibilities is larger, there is an algorithm for always getting the right answer in only a polynomial # of tries

EXPTIME-Complete Problems

- Known to be in EXPTIME, and *not* in P
- Go (囲碁) with

  Japanese *ko* rules (not known if Chinese or American rules are outside of P)
- Determining if a deterministic Turing machine (DTM) halts in *k* steps

Truly Undecidable

- Whether a given DTM *ever* halts, for a given input.

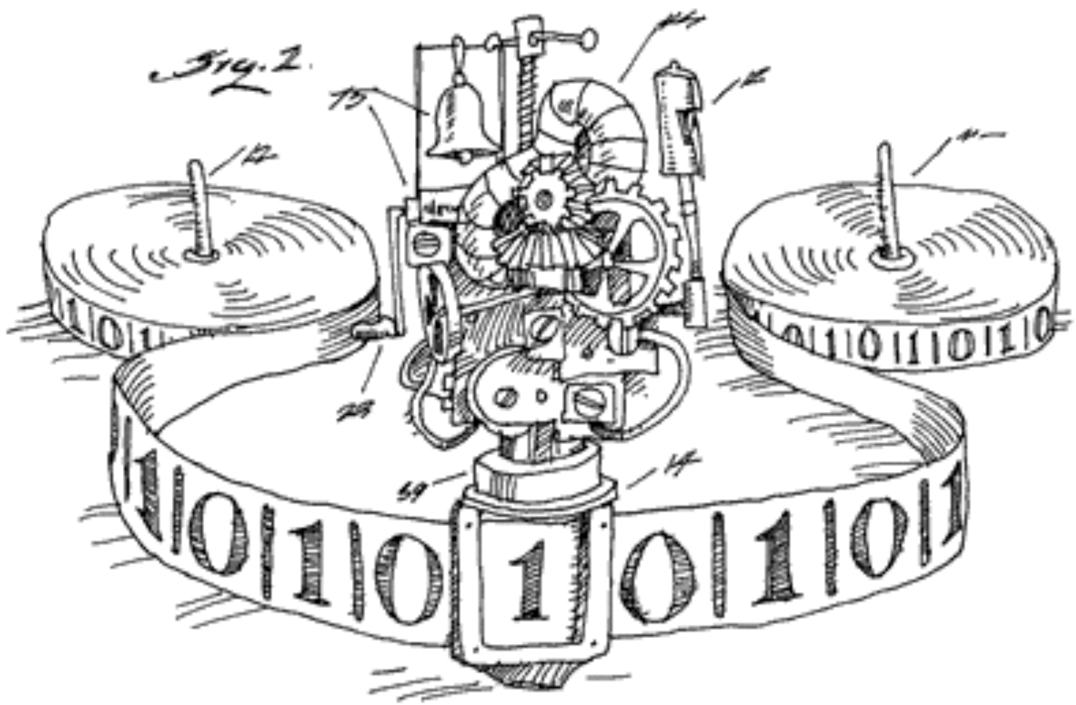


Image from http://www.cis.upenn.edu/~dietzd/CIT596/index.html

Poorly Defined

Many problems of simulating the real world can't be said to be in a particular complexity class, because there isn't a solution that can be checked.

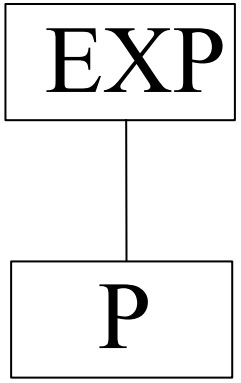
"Calculate the chemical bonding energy of…"

"Calculate the lift of an airfoil…"

So to formally evaluate their complexity, you have to define the problem somewhat artificially.

"Run this algorithm until the result converges to *n* digits of precision."

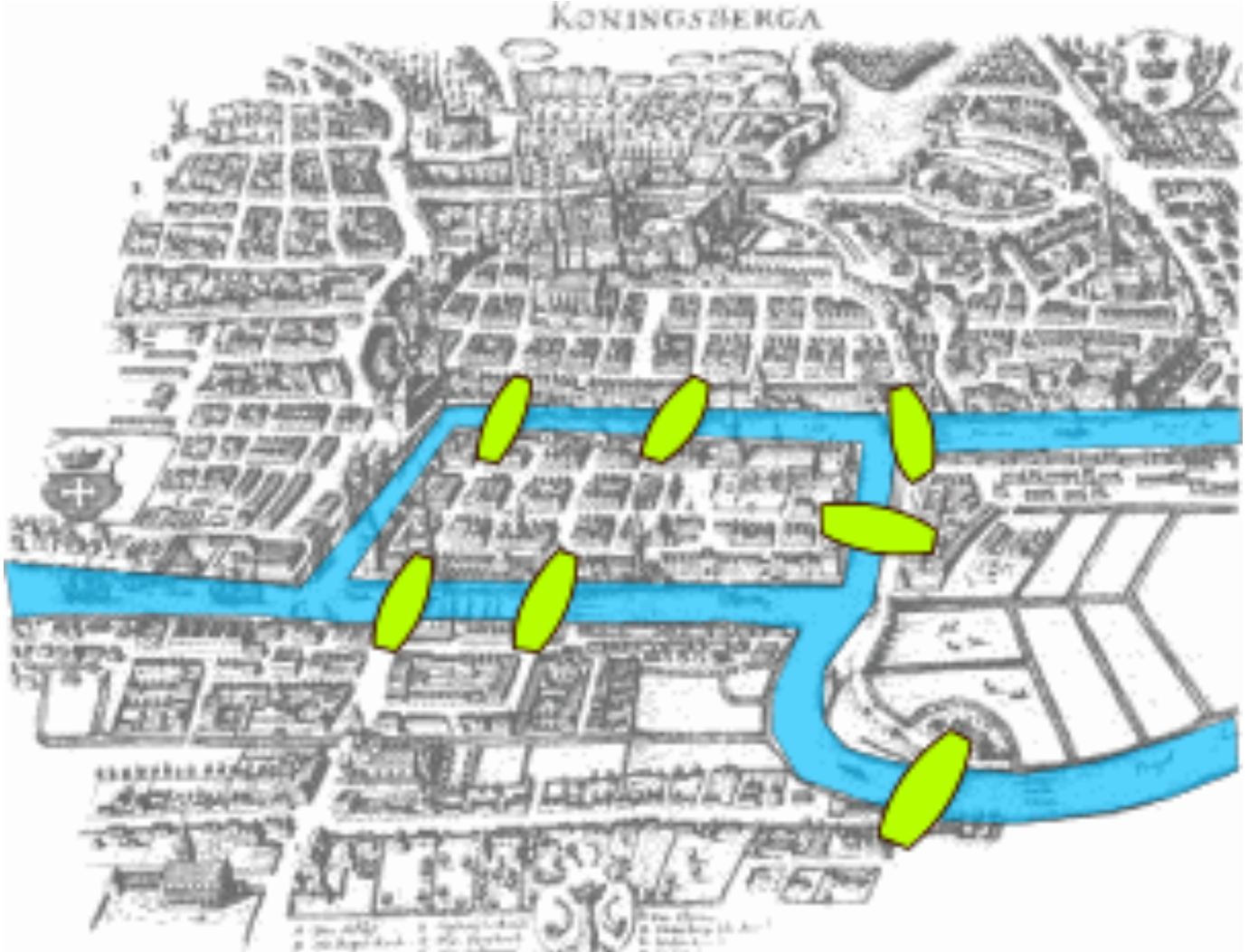
What We're Certain Of



P Examples

- **Deterministic Polynomial (P)**
- *O(.)* is polynomial
- Addition & other basic arithmetic
  - Matrix multiplication we just saw
- Eulerian cycles
  - Euler's Seven Bridges of Koenigsberg
- Sorting

Seven Bridges of Koenigsberg



https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

NP

- Nondeterministic Polynomial time
- Answers are hard to find, but easy to verify
- If you were clairvoyant (magic), you would find the answer on the first try
- Not known if P = NP
  (the answer is worth a million dollars)

NP Definition

- **Non-deterministic Polynomial (NP)**
- Key point: in practice, exponential # of operations to *find* the answer, polynomial # of operations to *check* the answer
  - *O(.)* generally exponential
- Actual (non-technical) definition: imagine you can test all possible solutions at once, and quit when you find the answer!
- Or, that you have some magic knack for always trying the correct solution first.

NP-Complete

- NP-Complete if it's in NP and in NP-hard
- NP-hard, “at least as hard as the hardest problems in NP”
- There is a polynomial-time **reduction** to another problem in NP-complete

NP-Complete Examples

- Traveling salesman problem
  - Hamiltonian cycle, hit each vertex once, looking for shortest cycle
- Clique
  - If every pair of people is either friends or enemies, what's the largest group where everyone is friends with everyone else?
- Satisfiability
- Increasingly large sudoku games
- Packing problems: knapsack, Tetris
- ...thousands more...

Some NP-Complete Problems

- Hamiltonian circuit
- Subgraph isomorphism
- *Graph isomorphism is not known to be NP-complete!*
- Boolean Satisfiability
- Some compiler code generation problems!
- Karp's original 21 problems (1972)

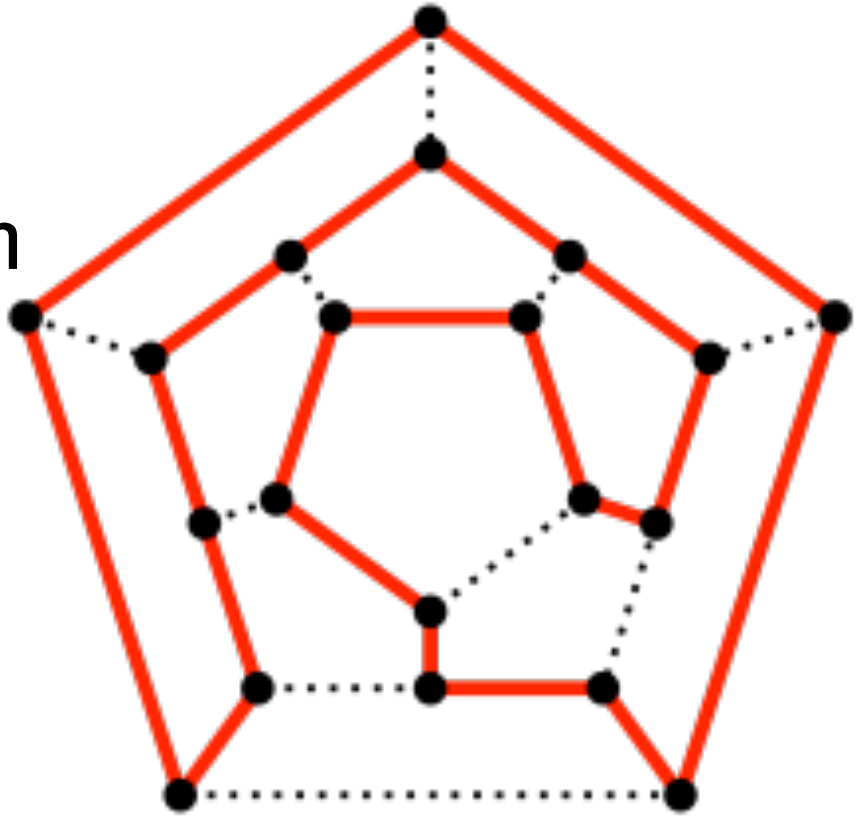


Image from Wikipedia

Traveling Salesman Problem

What is the shortest path (Hamiltonian cycle) through the graph that takes us to all the cities?

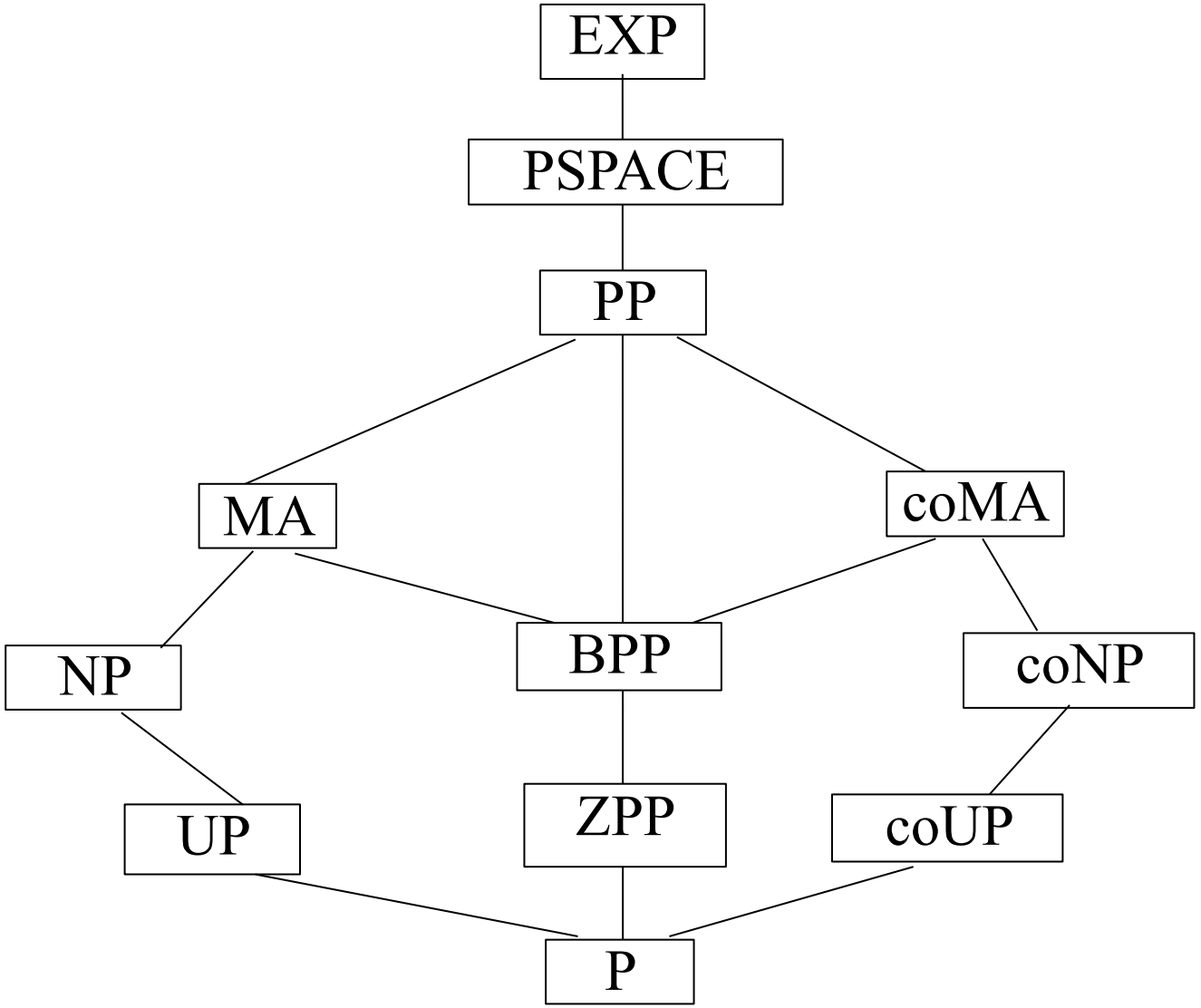
TSP

- Usually expressed with a fully-connected graph (e.g., driving by car)
- Add a city, and how long does it take to find the optimal solution?
- Checking all possible solutions requires O(exp(n)) time!
- *TSP is the most famous NP-hard problem*
- *With a threshold, NP-complete*
- *Without a threshold, NP-hard (no certificate!)*

In NP, not known to be outside P and not known to be NP-Complete

- Factoring!
- Graph isomorphism
  (n.b.: not subgraph isomorphism, which might be harder)
  - —> Laszlo Babai!

The Whole Classical Shebang
(not really, but enough to be useful)



What We Know

$$P \subseteq NP \subseteq PSPACE \subset EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE$$

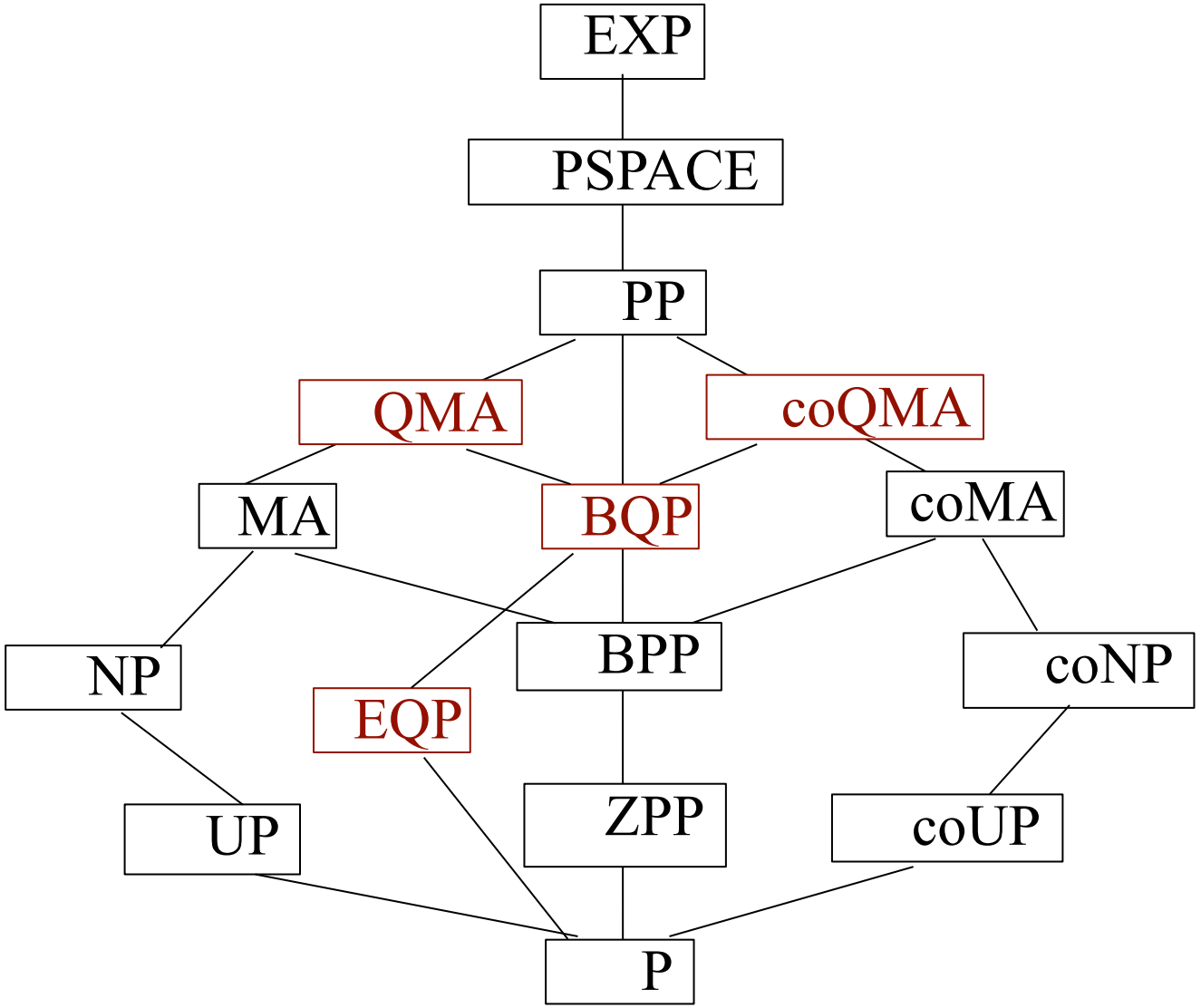
and

$$P \subsetneq EXPTIME$$
$$NP \subsetneq NEXPTIME$$
$$PSPACE \subsetneq EXPSPACE$$

Definitions

- EXP (or EXPTIME): exponential time on Turing machine
- PSPACE: polynomial space (exp. time)
- PP: probabilistic polynomial time
- MA: Merlin-Arthur protocol
- BPP: bounded-error probabilistic polynomial time
- ZPP: zero-error probabilistic polynomial time
- NP: non-deterministic polynomial time
- UP: unambiguous polynomial time
- P: polynomial time on a TM

The Whole Quantum Shebang



Definitions

- QMA: quantum Merlin-Arthur protocol
- BQP: bounded-error quantum polynomial time
- EQP: exact quantum polynomial time
- (many more have been defined...)

Quantum Algorithms

- BQP: Shor's factoring, discrete log, quantum simulation
- NP: Grover's search algorithm
- D-J: ?

Shor's Factoring Algorithm

$$66554087 = ? \quad 6703 \times 9929$$

An “efficient” classical algorithm for this problem is not known.

Using classical number field sieve, factoring an L-bit number is

$$O\left(e^{k\sqrt[3]{L\log^2 L}}\right)$$

Using quantum Fourier transform (QFT), factoring an L-bit number is

$$O\left(L^3\right)$$

Superpolynomial speedup! *But*... what are the constant fac d to ignore clock lea

O\left(e^{k\sqrt[3]{L\log^2 L}}\right)

O\left(L^3\right)

c. STOC 1994, SIA

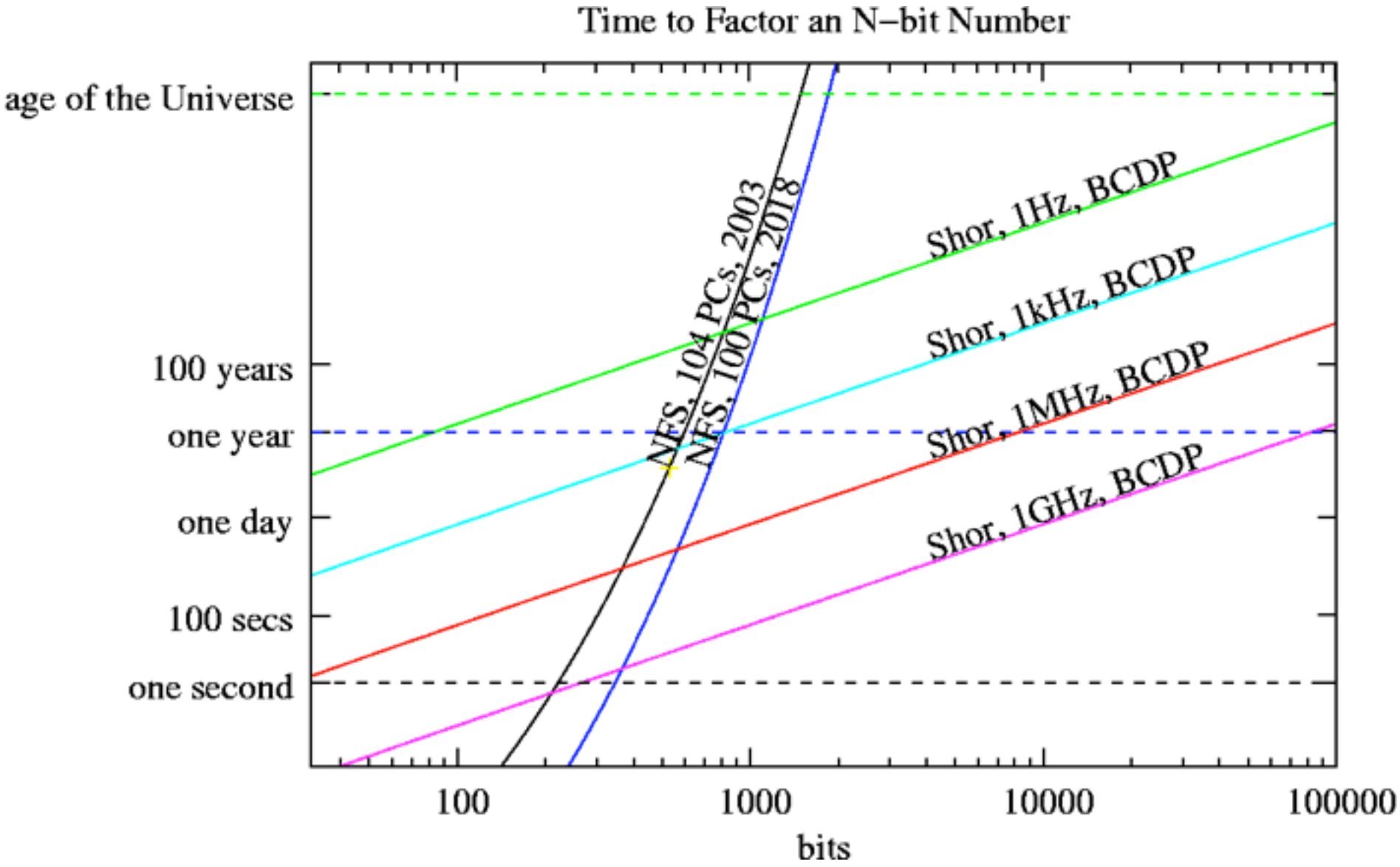
Shor's Algorithm

- Full consideration has to include both classical and quantum parts
- QFT is O(L^2)
- Modular exponentiation is O(L^3) in basic form
- Classical portion is also polynomial
- All polynomials are architecture-dependent in depth!

Shor v. Classical

- Number Field Sieve (NFS) is sub-exponential in length of number
- Not yet known if we can do better classically
- In NP, but believed to not be NP-complete (that is, not all NP problems can be reduced to factoring)
- Shor is exponentially faster than NFS, but we don't yet know if this means QC is exponentially faster than classical

Factoring Larger Numbers



Wrap-Up

- Main point: not yet known if quantum computers are really more efficient than classical ones!
- Believed, but not known, that QC CANNOT solve all NP problems in polynomial time
- Even this is just the tip of the complexity iceberg
- Many relationships partially understood
- QC may open new attacks on P =? NP problem

Homework

Pick one of the following and write a 3-4 page report on its *current* status, with references.

(1) one of Karp's original 21 problems

(2) the *halting problem*

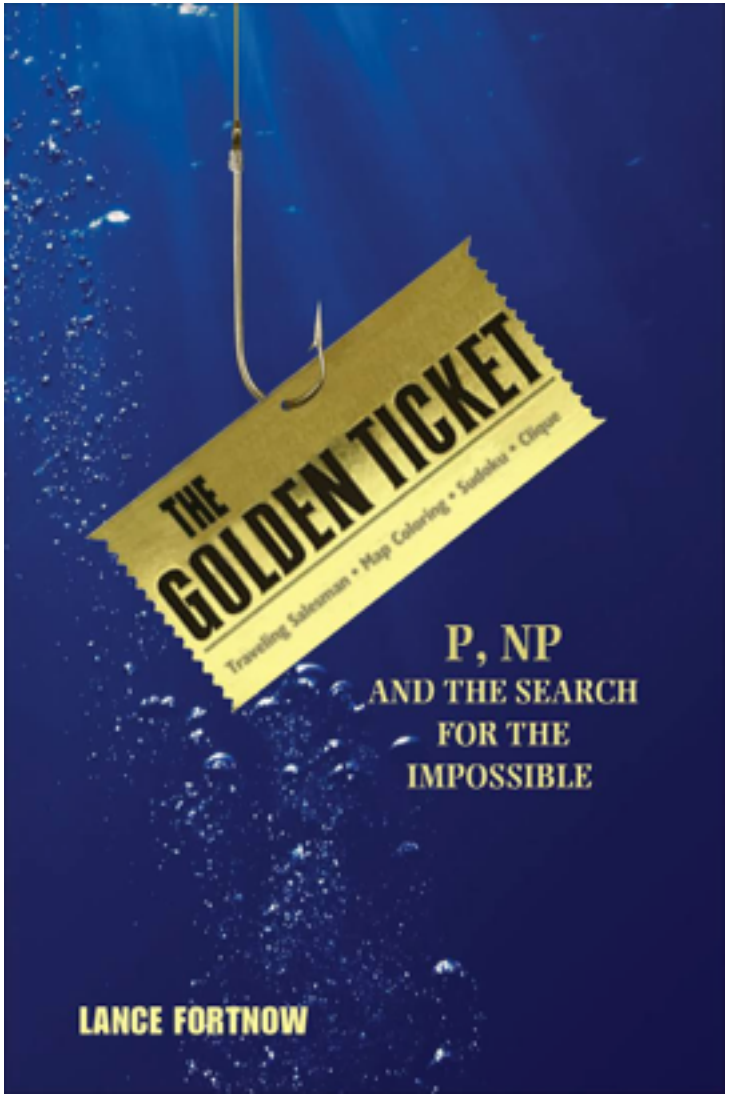
(3) universality of Turing machines

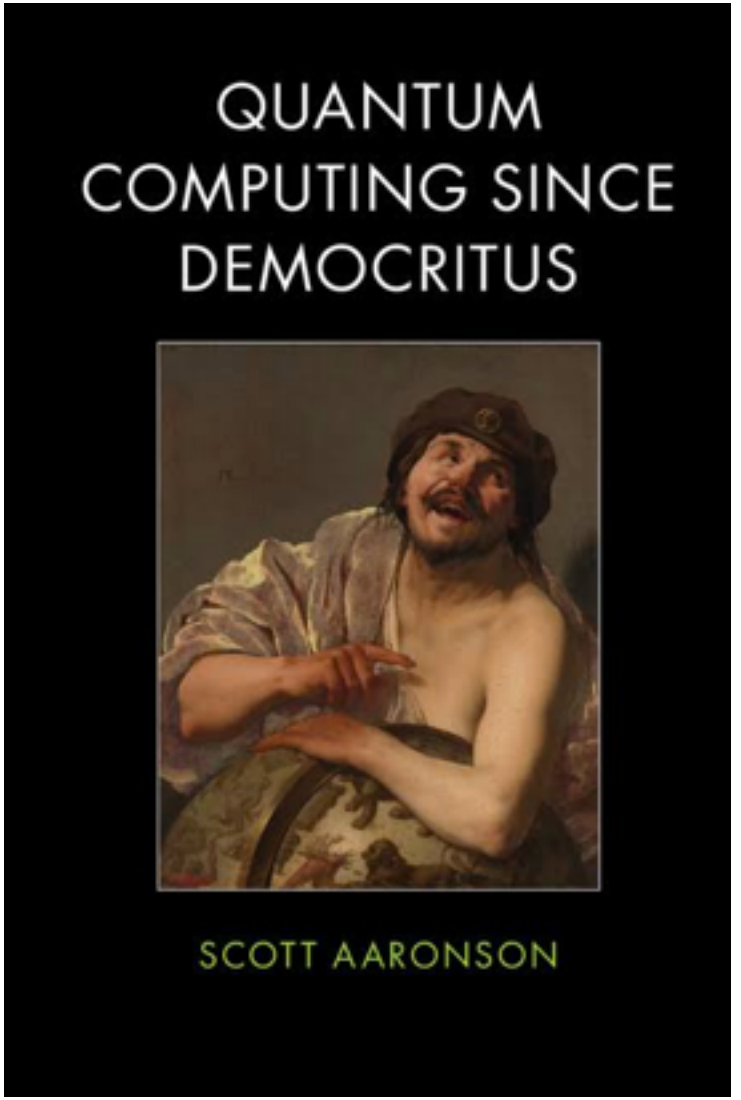
(4) computable v. uncomputable numbers

Research Question

We have described a small set of classes here, but hundreds are known. What if complexity classes are actually continuous, rather than discrete?

References


THE GOLDEN TICKET
Traveling Salesman • Map Coloring • Sudoku • Clique
P, NP
AND THE SEARCH FOR THE IMPOSSIBLE
LANCE FORTNOW


QUANTUM COMPUTING SINCE DEMOCRITUS
SCOTT AARONSON

References

- http://www.complexityzoo.com/
  (407 computational complexity classes!)
- Wikipedia has whole set of pages
- Mike & Ike
- Aaronson's Ph.D. thesis
- Hopcroft, Ullman, *Intro to Automata… (classic)*
- Papadimitriou, *Computational Complexity* (somewhat dated now?)
- Sipser, *Intro to Theory of Computation*
  (somewhat dated now?)
- Lance Fortnow, *The Golden Ticket*
  (no equations)
- Scott Aaronson, *Quantum Computing Since Democritus*
  (few equations, but still highly technical)