School of
Information Systems

SMU
SINGAPORE MANAGEMENT UNIVERSITY

COR-IS1702: Computational Thinking
Week 11: Limits of Computation

Limits of Computation video (25 mins): https://youtu.be/gV12PS19YL8

2020/21 Term 1

Road Map

Algorithm Design and Analysis

(Weeks 1 - 5)

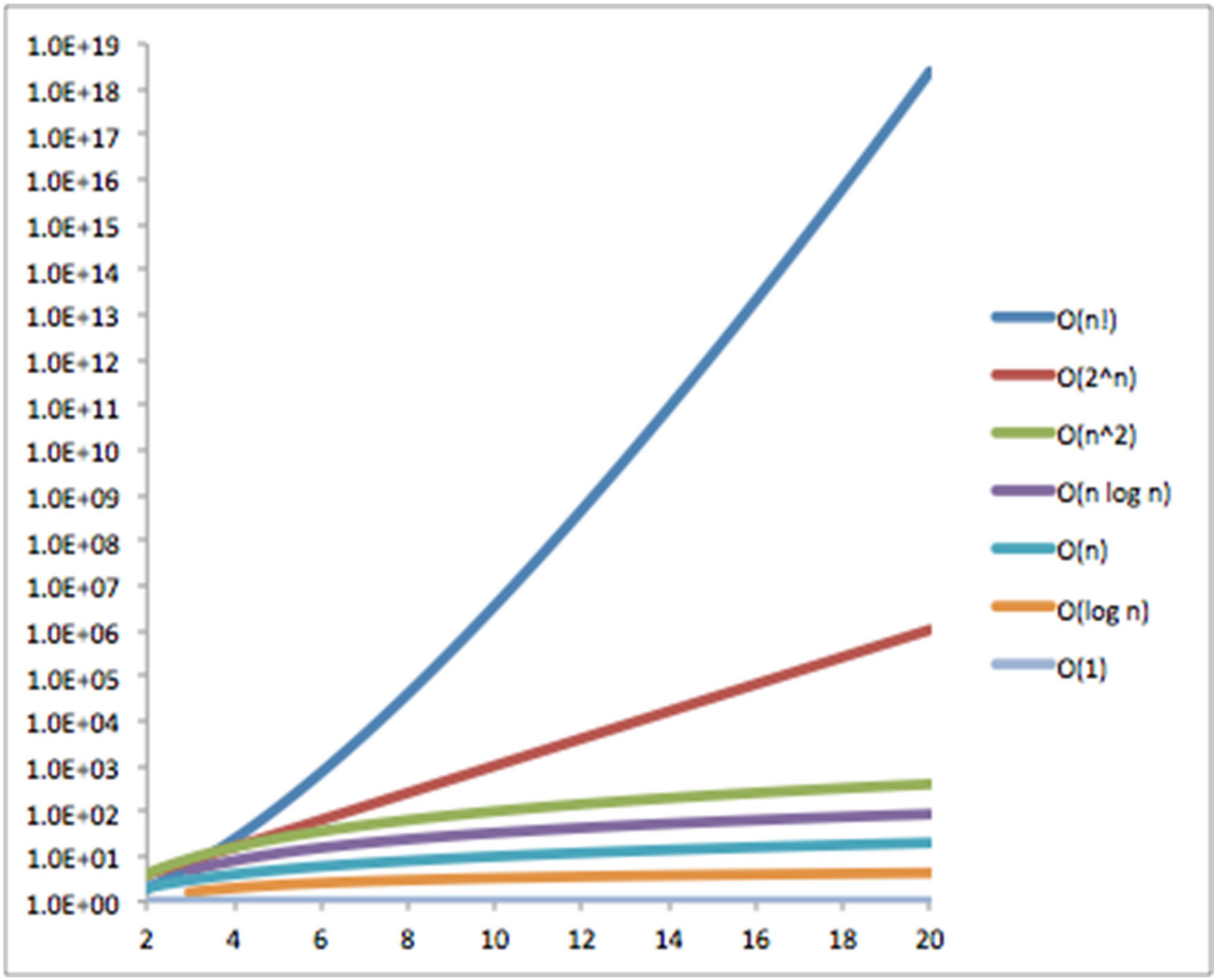
Fundamental Data Structures

(Weeks 6 - 9)

Computational Intractability and Heuristic Reasoning

- ✦ Week 10: Heuristics
- This week ⟶ ✦ Week 11: Limits of Computation
- ✦ Week 13: Review

Log scale



factorial
exponential
polynomial
linearithmic
linear
logarithmic
constant

If each operation takes 1 nanosecond (10^{-9} sec)

input size	log n	n	n log n	n^2	2^n	n!
10	0.003 µs	0.01 µs	0.033 µs	0.1 µs	1 µs	3.63 ms
20	0.004 µs	0.02 µs	0.086 µs	0.4 µs	1 ms	77.1 yrs
30	0.005 µs	0.03 µs	0.147 µs	0.9 µs	1 sec	8 x 10^{15} yrs
40	0.005 µs	0.04 µs	0.213 µs	1.6 µs	18.3 min	
50	0.006 µs	0.05 µs	0.282 µs	2.5 µs	13 days	
100	0.007 µs	0.1 µs	0.644 µs	10 µs	4 x 10^{13} yrs	
1,000	0.010 µs	1.00 µs	9.966 µs	1 ms		
10,000	0.013 µs	10 µs	130 µs	100 ms		
100,000	0.017 µs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 µs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 µs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 µs	0.10 sec	2.66 sec	115.7 days		

Source: Figure 2.4 in Steven S. Skiena, "The Algorithm Design Manual," Springer, 2nd Edition, 2010

Algorithm Efficiency & Problem Tractability

- Efficient vs. inefficient algorithm
  - An algorithm is considered "efficient" if it has polynomial time complexity or lower, i.e., $O(n^k)$
  - An algorithm with exponential $O(k^n)$ or factorial $O(n!)$ complexity is considered inefficient

- Tractable vs. intractable problems
  - A problem is considered tractable or solvable if there is a known solution with polynomial complexity or lower.
  - Otherwise, the problem is considered intractable

- Problems need not be complicated to be intractable

Example: Prime Factorization

3 x 5 = ?

Multiplication

Example: Prime Factorization

? x ? = 15

Factoring

Example: Prime Factorization

<u>212 digits</u>

9091213529597818878440658302600437485892608310328358720428512168960411528640933367824950788367956756806141

x

8143859259110045265727809126284429335877899002167627883200914172429324360133004116702003240828777970252499

=

74037563479561712828046796097429573142593188889231289084936232638972765034028266276891996419625117843995894330502127585370118968098286733173273108930900552505116877063299072396380786710086096962537934650563796359

Example: Prime Factorization

? x ? =

```
7403756347956171282804679609742
9573142593188889231289084936232
6389727650340282662768919964196
2511784399589433050212758537011
8968098286733173273108930900552
5051168770632990723963807867100
86096962537934650563796359
```

2000 CPU Years!!

A $30,000 problem

The RSA Factoring Challenge

Why is Factoring Difficult?

- How to find factors of a number?
  - Divide by 2, 3, 5, 7, 11, …
  - Essentially a “brute force search”.
  - For very large number, we also have very large number of possibilities to search.

- Important observation:
  - Difficult to solve the factoring problem.
  - Yet with factors, very easy to verify.

P and NP Problems

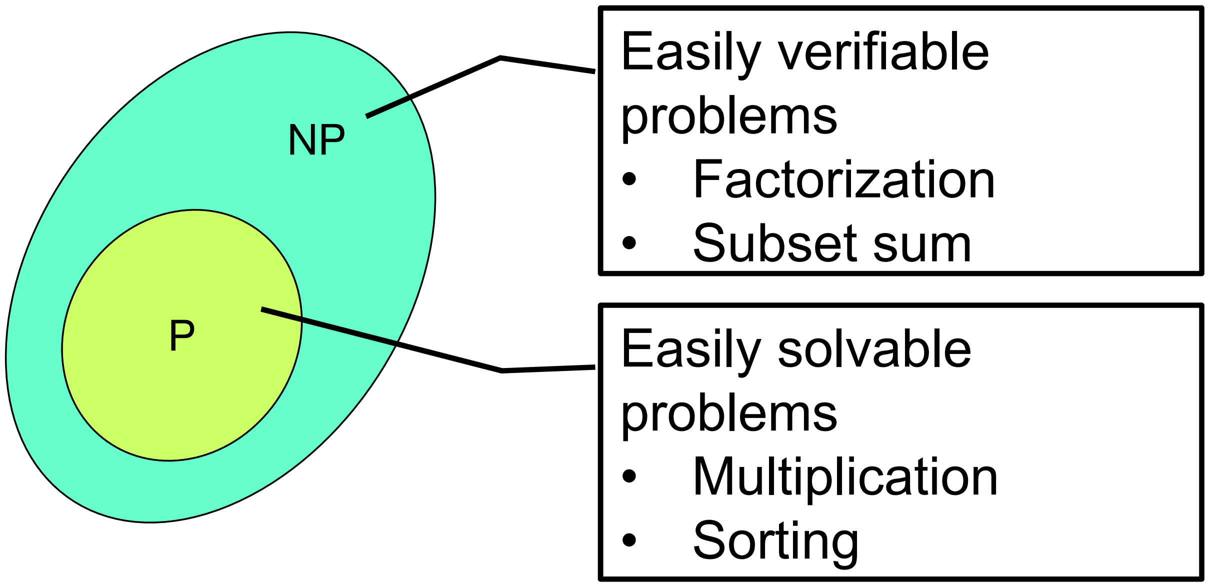
Let’s first talk about the difference between solving a problem, and verifying a proposed solution…

Example: Subset Sum Problem

- Given a set of integers, find a subset whose elements sum to zero.
- Example: Does any subset of {11, 5, -3, -1, -2, 7, -4} sum to zero?
- To solve the problem (using brute force):
  - Iterate over all the possible subsets ... O(2^n)
    - Check that the subset sums to zero ... O(n) worst case
  - Overall complexity: O($2^n \times n$)
- To verify a solution:
  - Check that the subset sums to zero ... O(n)
  - Overall complexity: O(n)

Putting Problems in Categories

- P (polynomial)
  - Solvable in polynominal time.
- NP (non-deterministic polynomial)
  - Easy to verify a proposed solution in polynominal time.



Easily verifiable problems
- Factorization
- Subset sum

Easily solvable problems
- Multiplication
- Sorting

- If a problem can be solved in polynominal time, it can definitely be verified in polynominal time, because you can verify a solution by solving the problem!

→ All P problems are NP problems.

Class of Problems: **P** (Polynomial)

A problem is in **P** if there is a known polynomial time algorithm to **solve** it.

- Examples:
  - Closest Pair Problem
    - $O(n \log n)$
  - Shortest Path Problem
    - $O(n^2)$

Class of Problems: **NP** (Non-deterministic Polynomial)

A problem is in **NP** if there is a known polynomial time algorithm to **verify** a given solution.

- Examples:
  - Prime Factorization Problem
    - Given a (very large) integer `N`, find its prime factors.
    - Difficult to solve
    - Easy to check
  - Subset Sum Problem
    - Find a subset of integers that sum up to a given goal.

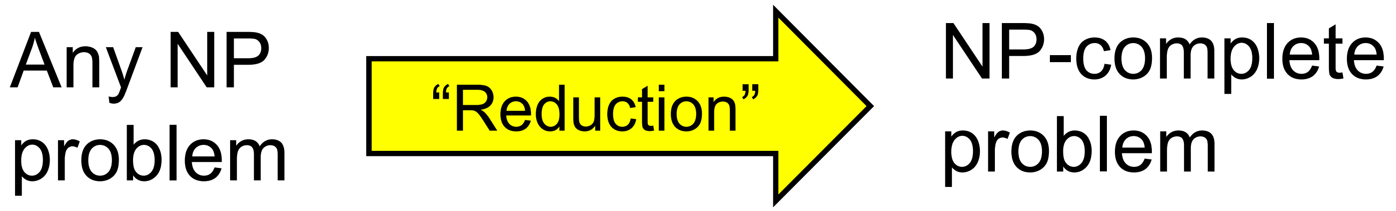
NP-Complete & NP-Hard Problems

Two more categories…

Class of Problems: **NP-Complete**

An NP problem is **NP-complete** if all other NP problems can be **reduced** to it.

- NP-complete problems are the hardest problem in NP
- If we can find a solution to an NP-complete problem, it implies that we can find a solution to all NP problems.
- Examples:
  - Subset Sum problem
  - Boolean Satisfiability problem



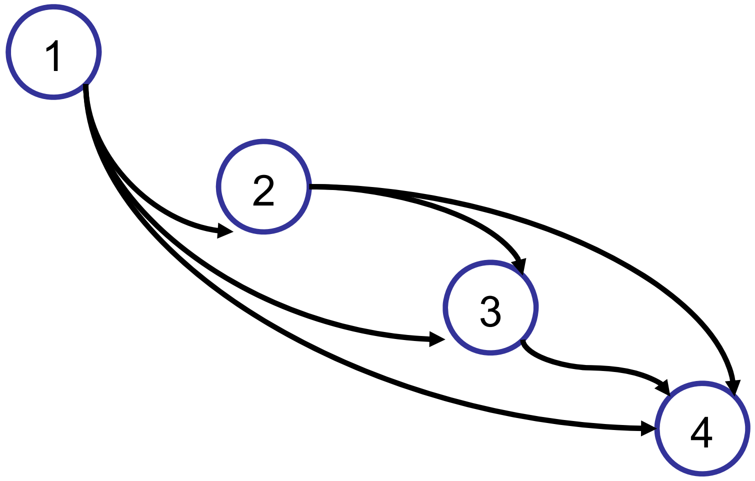
- You can convert any NP problem into a NP-complete problem (reverse is not necessarily true)
- NP-complete problem is the "grand-daddy" of all NP problems
- If you can solve the NP-complete problem quickly → you can solve any NP problem quickly.

1st Example of "Reduction"

- Problem#1: Sorting an array of integers
  - Given an array of integers, sort them in ascending order.
- Problem#2: Topological sorting
  - Given a directed acyclic graph G, find a topological ordering.
- Problem#1 can be "reduced" into Problem#2
  - We can show that Problem#1 is a special case of Problem#2.
  - Thus, an algorithm that solves Problem#2, will also solve Problem#1.

Reducing Integer Sorting to Topological Sorting

- Suppose we need to sort an array [2, 1, 4, 3]
- Construct a graph G:
  - Each vertex is an integer.
  - There is an edge from a vertex to another vertex, if the first is smaller than the second.
  - Do topological sorting on G.



2nd Example of "Reduction"

- Hamiltonian Cycle Problem (HCP)
  - Given a graph G (might not be *complete*), decide if there exists a cycle that visits every node of G exactly once.

- Travelling Salesman Problem (TSP)
  - Given a *complete weighted* graph G, find the cycle with the shortest distance.

- HCP can be "reduced" into TSP
  - In other words, we can show that HCP is a special case of TSP.
  - Thus, an algorithm that solves TSP, will also solve HCP.
  - Further, HCP cannot be “harder” than TSP.

Reducing HCP into TSP

- Suppose we have an instance of Hamiltonian Cycle Problem (HCP)
  - The input is graph G with N vertices, which may be incomplete (some vertices are not connected) and non-weighted.
- Transforming HCP into Travelling Salesman Problem (TSP)
  - TSP requires a complete, weighted graph.
  - We create a second graph G’ for the same set of N vertices.
  - All the original edges in G will be added into G’, each with edge weight 1.
  - Edges that are not yet in G’ will be added with edge weight 2.
  - G’ is now a complete, weighted graph.
  - Running TSP on G’ will result in the shortest path in G’.
- Solving TSP on G’ will solve HCP on G
  - If the TSP shortest path in G’ has tour length of N, this tour must consist of all original edges in G, which means there exists a HCP cycle in G.
  - If the TSP shortest path in G’ has tour length greater than N, the tour contains some edges not in G, which means there does NOT exist a HCP cycle in G.

Class of Problems: **NP-hard**

A problem is in **NP-hard** if it is at least as hard as NP-complete problems. It is not necessarily in NP, i.e., may not be verifiable in polynomial time.

If an optimization problem is not in P, it most likely cannot be NP, as verifying the solution would be as hard as solving the problem. In this case, it will be NP-hard.

Potential confusion between NP-hard and NP-complete problems:
See, for example, a blog from an avid Java and Ruby programmer!

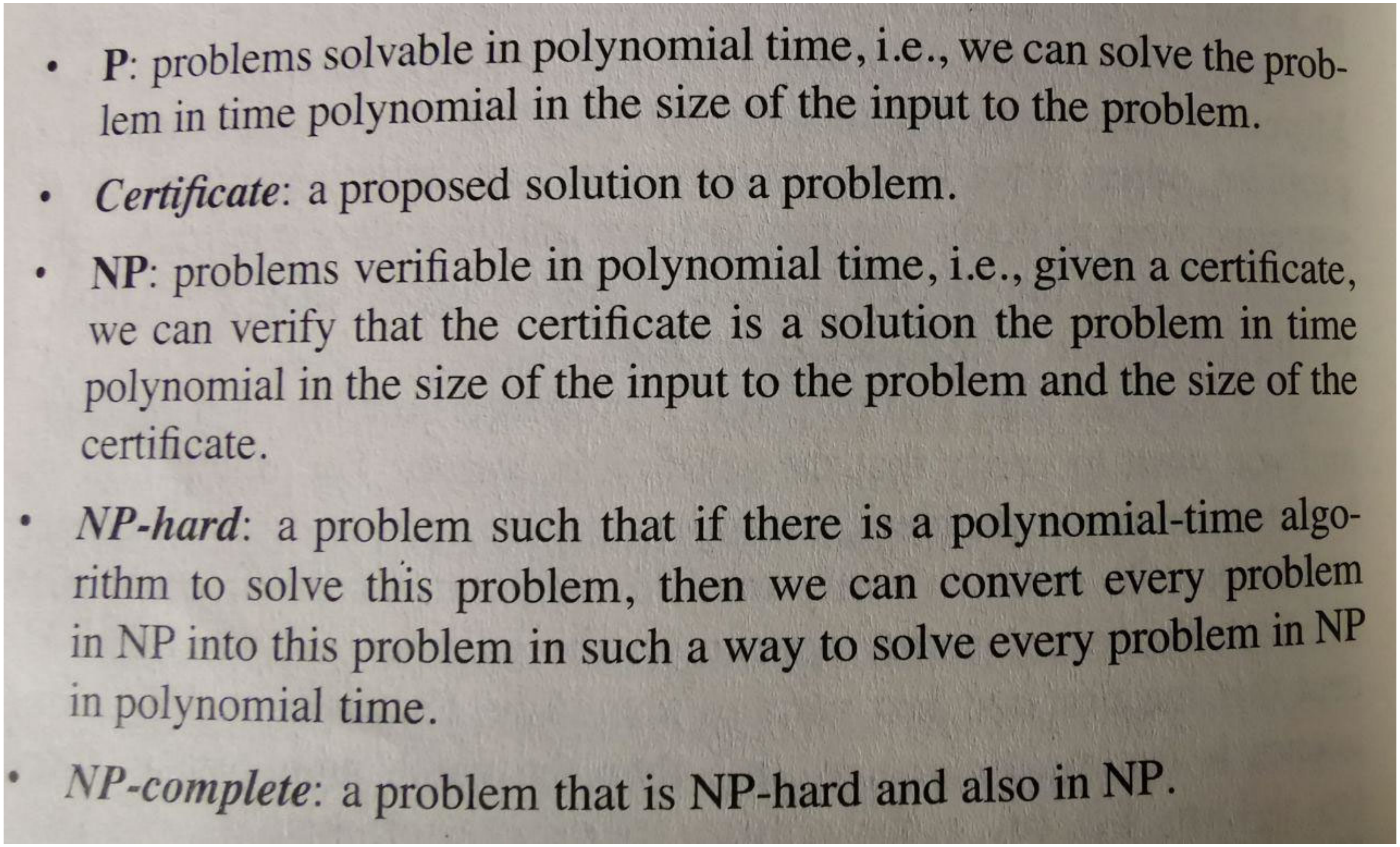
http://www.nomachetejuggling.com/2012/09/14/traveling-salesman-the-most-misunderstood-problem/

Some NP-Hard Graph Problems

- Traveling Salesperson Problem
  - Given a weighted graph, find a tour whose length is **minimized**.
- Graph Coloring Problem
  - Given a graph, find a coloring that uses the **minimum** number of colors.

More hard problems at:

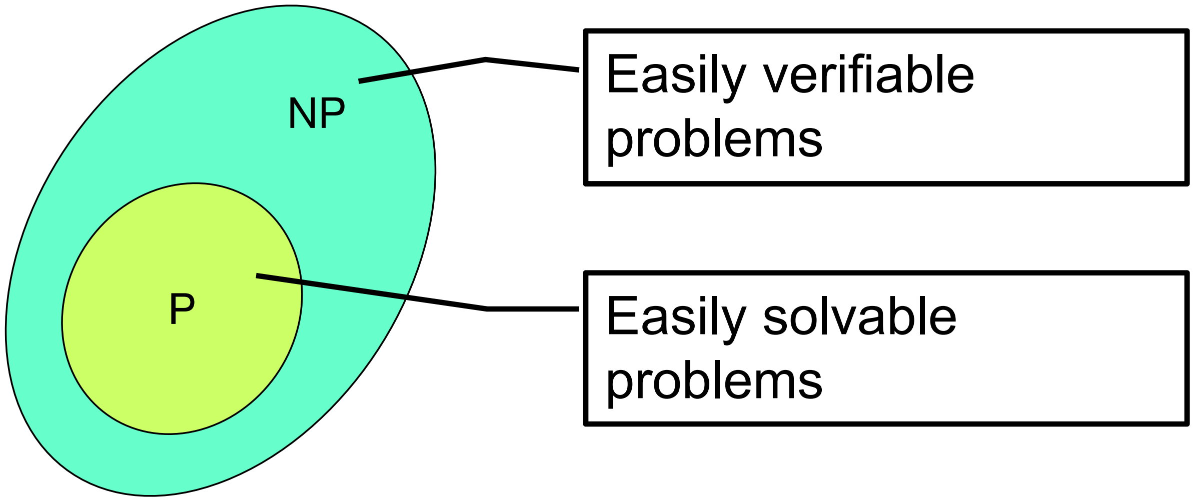
http://www.cs.sunysb.edu/~algorith/major_section/1.5.shtml



- **P**: problems solvable in polynomial time, i.e., we can solve the problem in time polynomial in the size of the input to the problem.
- ***Certificate***: a proposed solution to a problem.
- **NP**: problems verifiable in polynomial time, i.e., given a certificate, we can verify that the certificate is a solution the problem in time polynomial in the size of the input to the problem and the size of the certificate.
- ***NP-hard***: a problem such that if there is a polynomial-time algorithm to solve this problem, then we can convert every problem in NP into this problem in such a way to solve every problem in NP in polynomial time.
- ***NP-complete***: a problem that is NP-hard and also in NP.

From "Algorithms Unlocked" by Thomas H Cormen. p.184

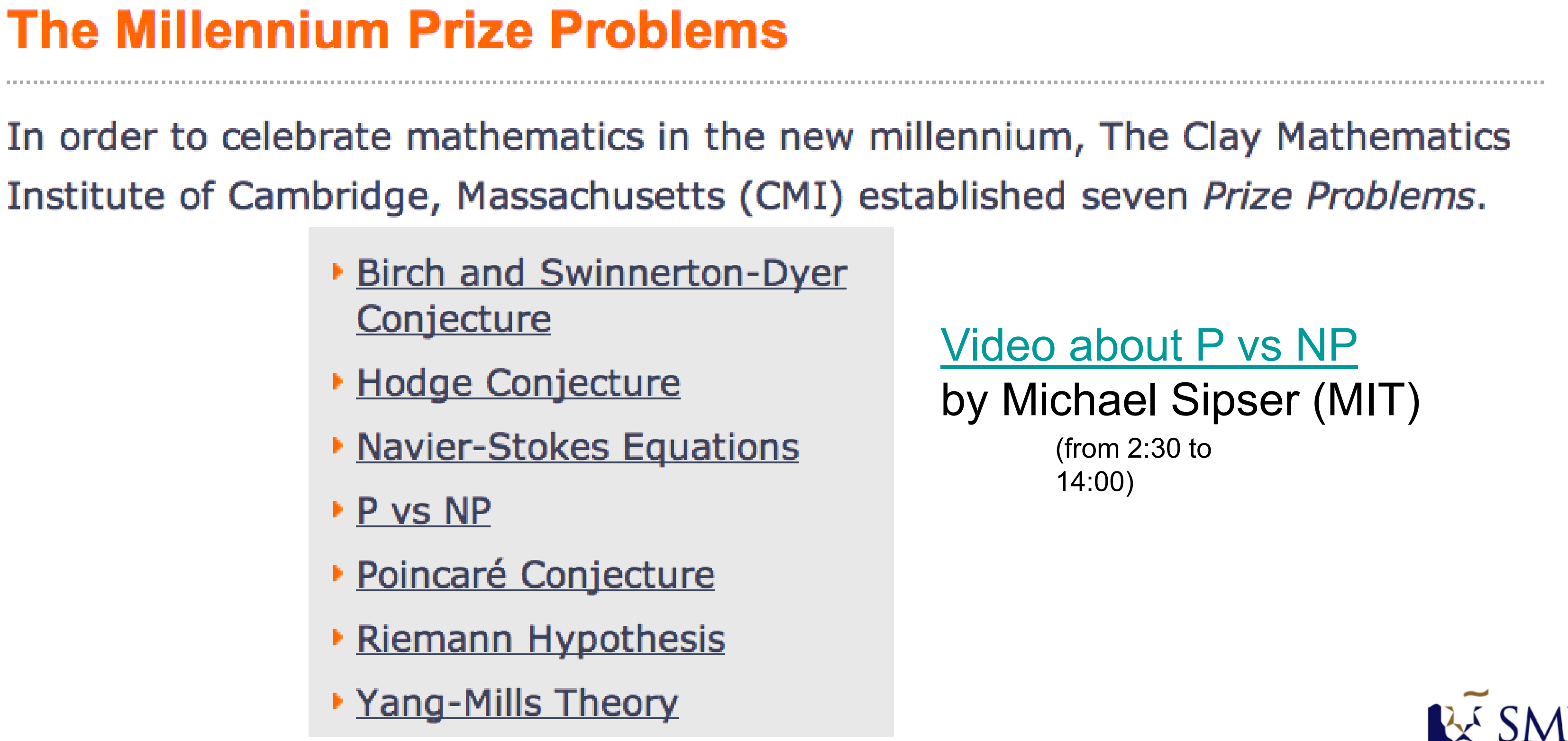
Is P = NP?



Million-Dollar Question

P = NP ?

If a problem can be verified in polynomial time, can it also be solved in polynomial time?



The Millennium Prize Problems

In order to celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) established seven *Prize Problems*.

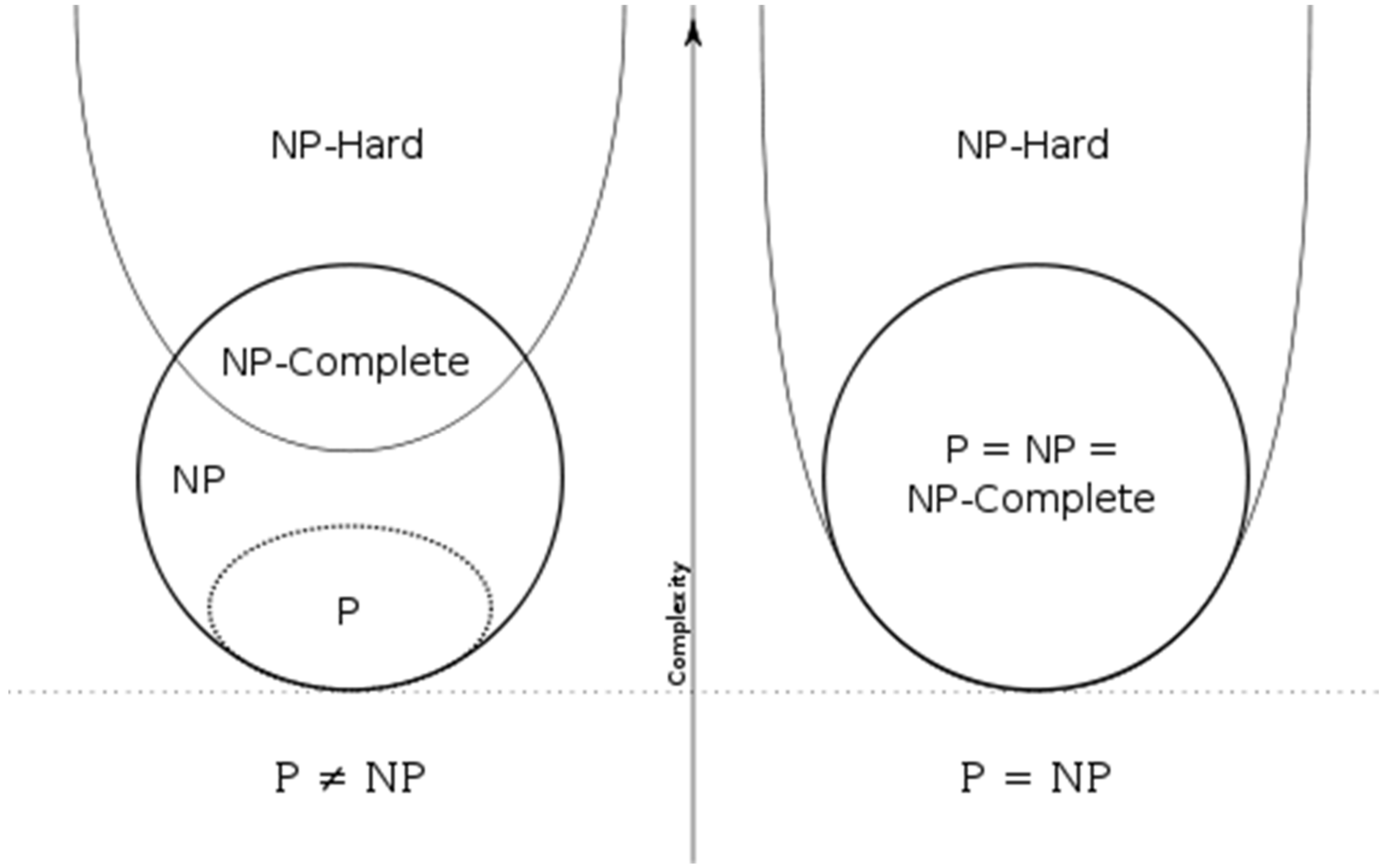
- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- P vs NP
- Poincaré Conjecture
- Riemann Hypothesis
- Yang-Mills Theory

Video about P vs NP by Michael Sipser (MIT)

(from 2:30 to 14:00)

http://www.claymath.org/millennium-problems

What is the difference?



Why is P vs. NP important?

- If P = NP
  - Problems that we used to think are hard suddenly seem “solvable”.
  - Good:
    - There is hope that we can solve problems previously believed to be NP-complete in polynomial time.
  - Bad:
    - Some cryptography depends on the assumption that solving some mathematical operations are extremely hard, while checking them are easy.
- If $P \neq NP$
  - Focus on finding approximations (developing heuristics).
- Most computer scientists “believe” that $P \neq NP$, but it has not been conclusively proven yet.