CSE373: Data Structures & Algorithms

Lecture 22: The P vs. NP question, NP-Completeness

Lauren Milne

Summer 2015

Admin

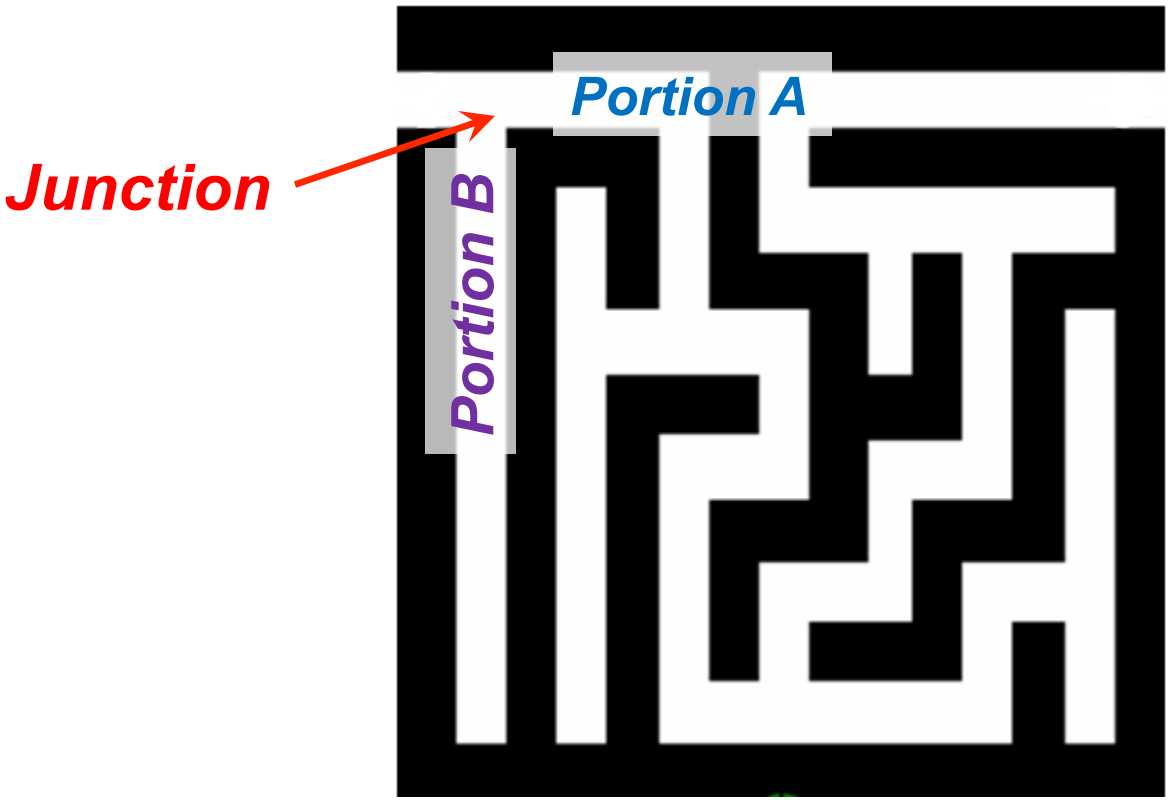
- Homework 6 is posted
  - Due next Wednesday
  - No partners

Algorithm Design Techniques

- Greedy
  - Shortest path, minimum spanning tree, …
- Divide and Conquer
  - Divide the problem into smaller subproblems, solve them, and combine into the overall solution
  - Often done recursively
  - Quick sort, merge sort are great examples
- Dynamic Programming
  - Brute force through all possible solutions, storing solutions to subproblems to avoid repeat computation
- Backtracking
  - A clever form of exhaustive search

Backtracking: Idea

- Backtracking is a technique used to solve problems with a large search space, by systematically trying and eliminating possibilities.
- A standard example of backtracking would be going through a maze.
  - At some point, you might have two options of which direction to go:



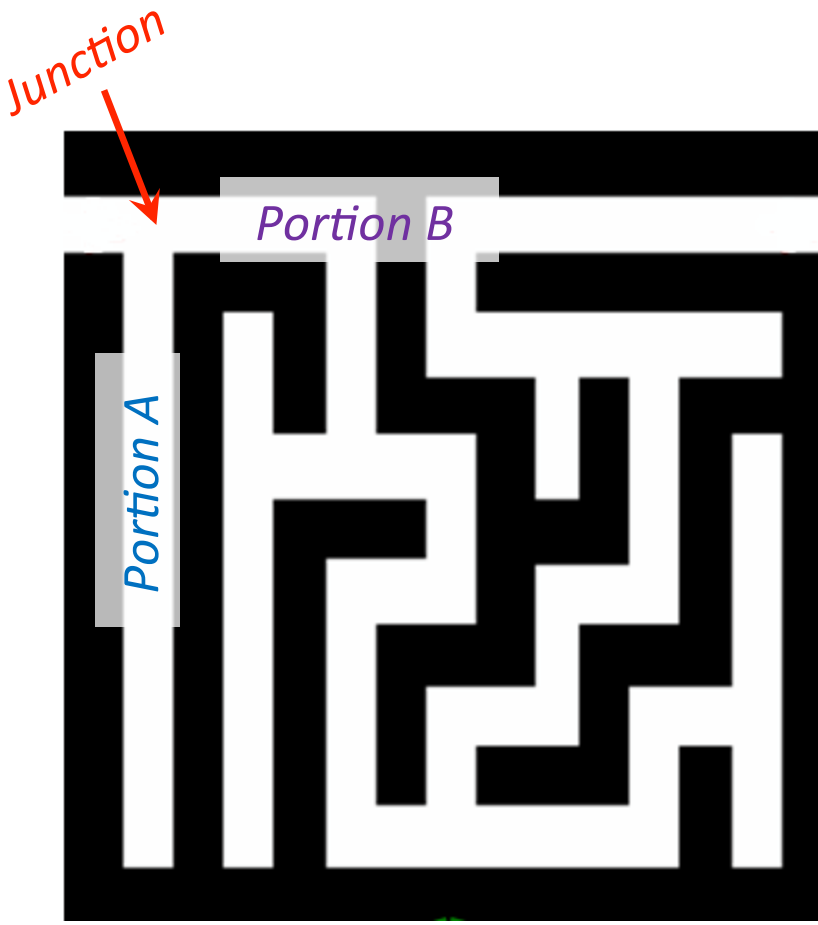
Backtracking

One strategy would be to try going through Portion A of the maze.

If you get stuck before you find your way out, then you *"backtrack"* to the junction.

At this point in time you know that Portion A will *NOT* lead you out of the maze,

so you then start searching in Portion B

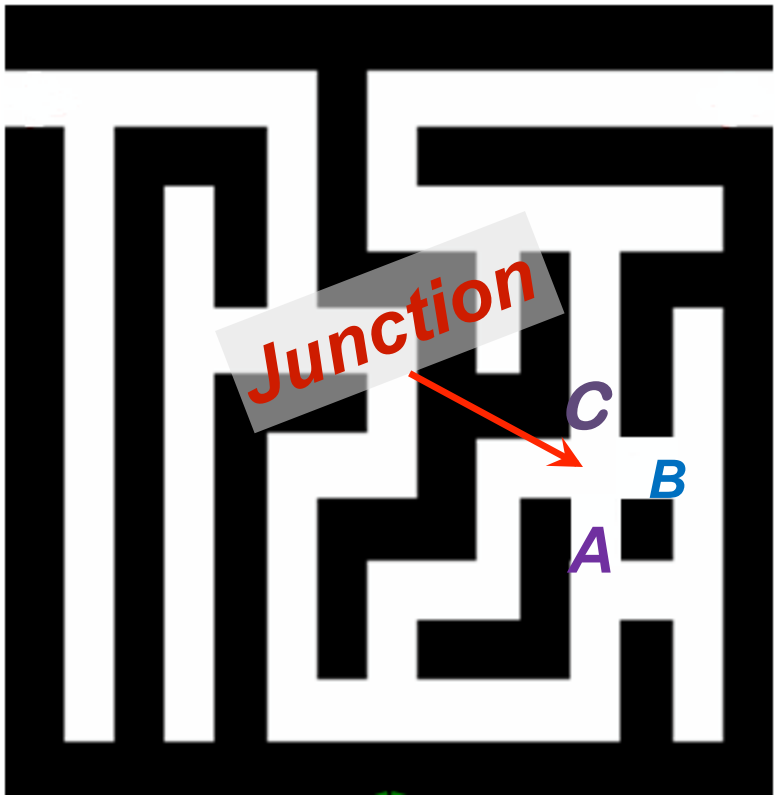


Backtracking

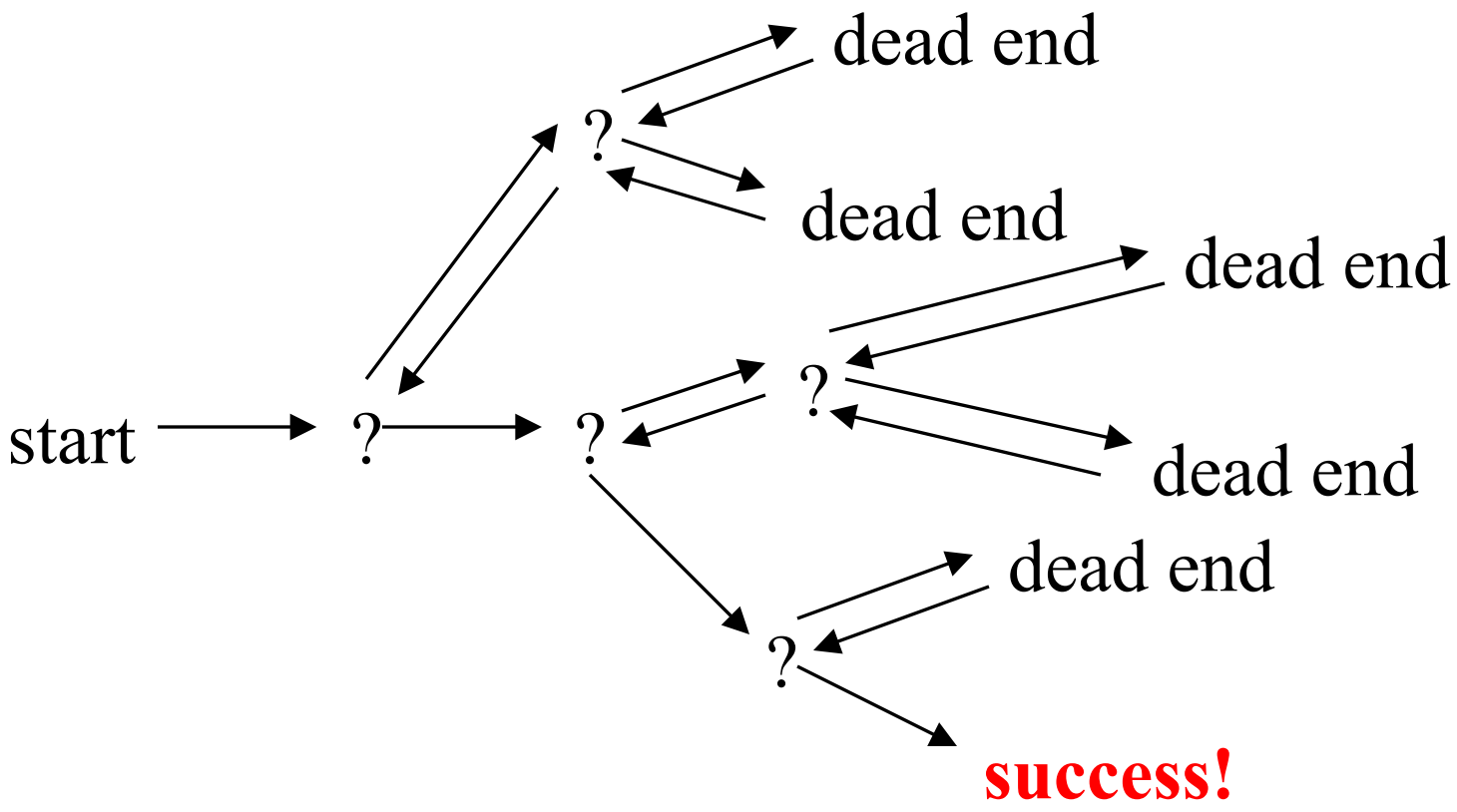
- Clearly, at a single junction you could have even more than 2 choices.

- The backtracking strategy says to try each choice, one after the other,
  - if you ever get stuck, *"backtrack"* to the junction and try the next choice.

- If you try all choices and never found a way out, then there IS no solution to the maze.



Backtracking (animation)



Backtracking

- Dealing with the maze:
  - From your start point, you will iterate through each possible starting move.
  - From there, you recursively move forward.
  - If you ever get stuck, the recursion takes you back to where you were, and you try the next possible move.

- Make sure you don't try too many possibilities,
  - Mark which locations in the maze have been visited already so that no location in the maze gets visited twice.
  - If a place has already been visited, there is no point in trying to reach the end of the maze from there again.

Backtracking

The neat thing about coding up backtracking is that it can be done recursively, without having to do all the bookkeeping at once.

- Instead, the stack of recursive calls does most of the bookkeeping
- (i.e., keeps track of which locations we've tried so far.)

On to Complexity theory!

The $1M question

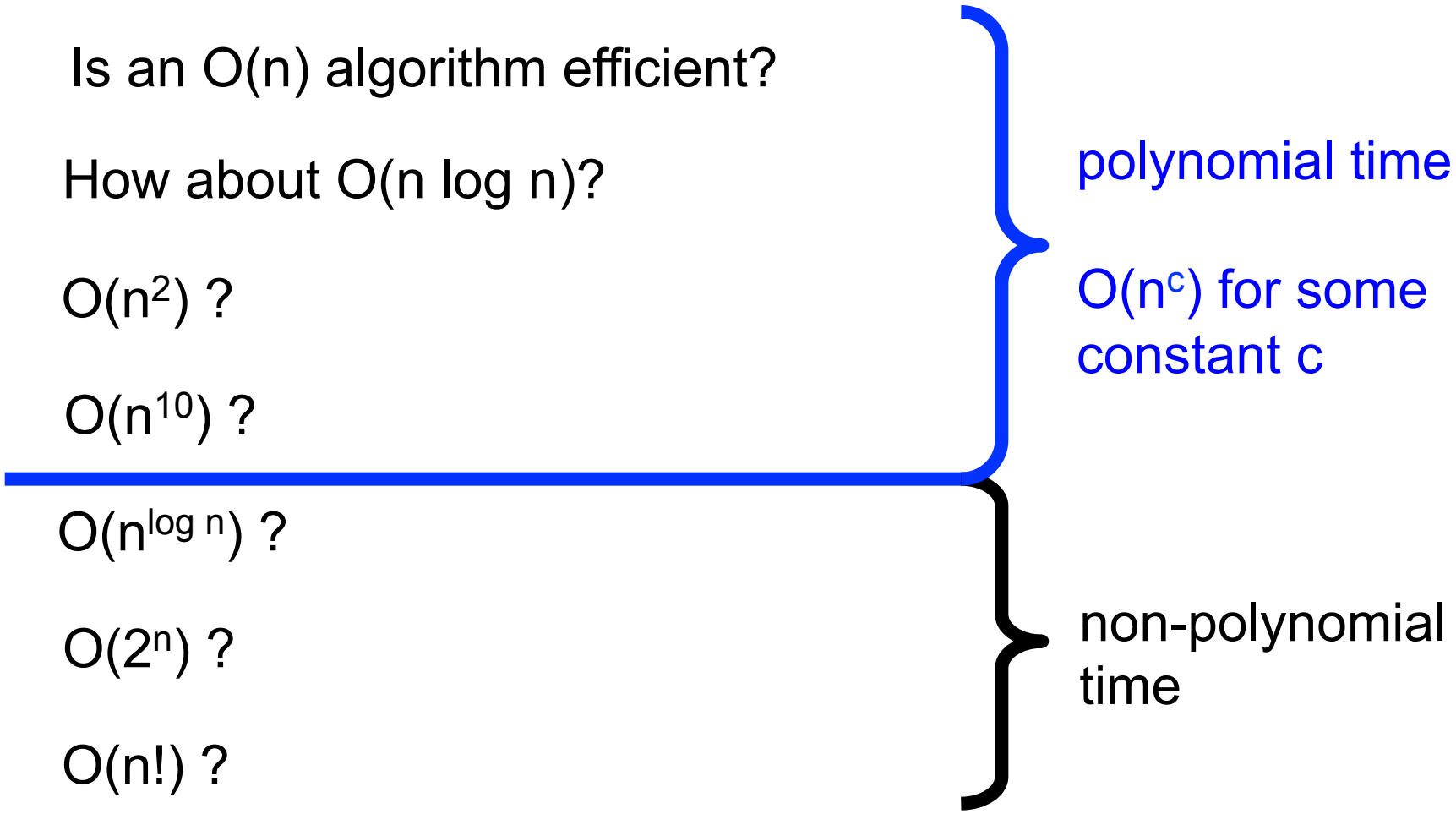
The Clay Mathematics Institute
Millennium Prize Problems

1. Birch and Swinnerton-Dyer Conjecture
2. Hodge Conjecture
3. Navier-Stokes Equations
4. P vs NP
5. Poincaré Conjecture
6. Riemann Hypothesis
7. Yang-Mills Theory

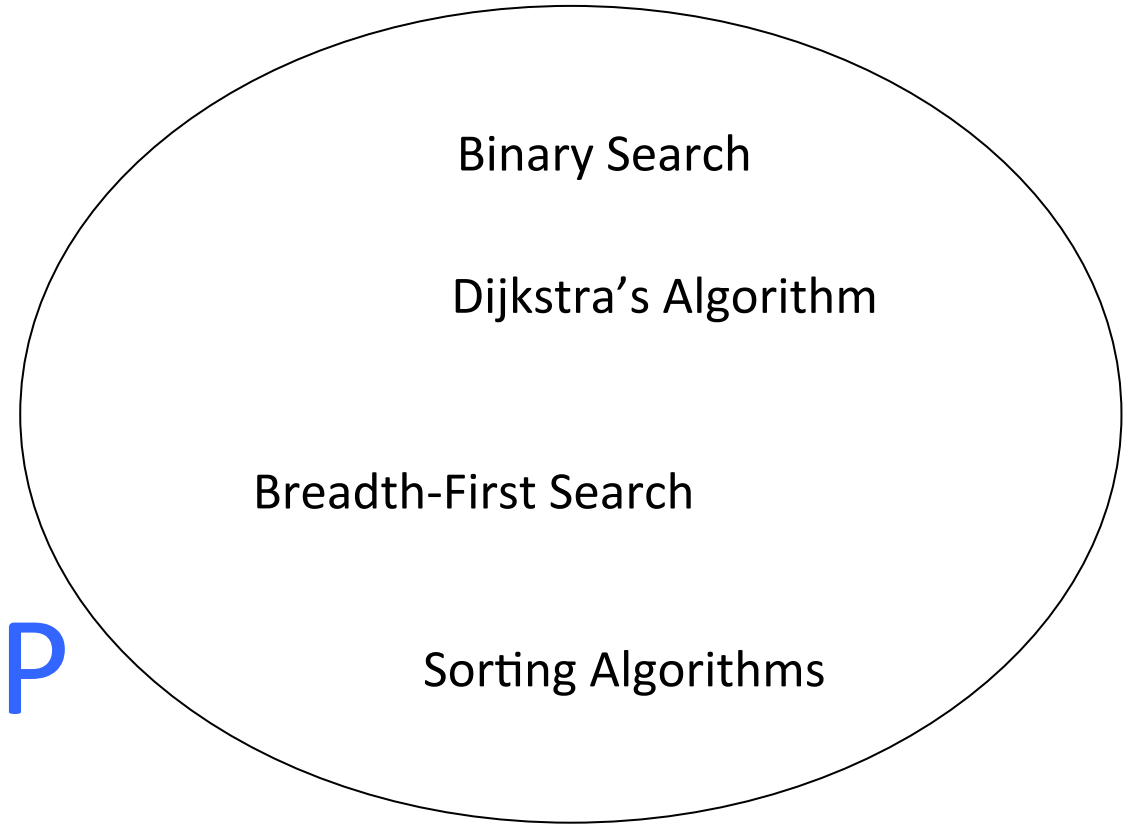
The P versus NP problem (informally)

Can every problem whose solution can be *quickly* verified by a computer also be *quickly* solved by a computer?

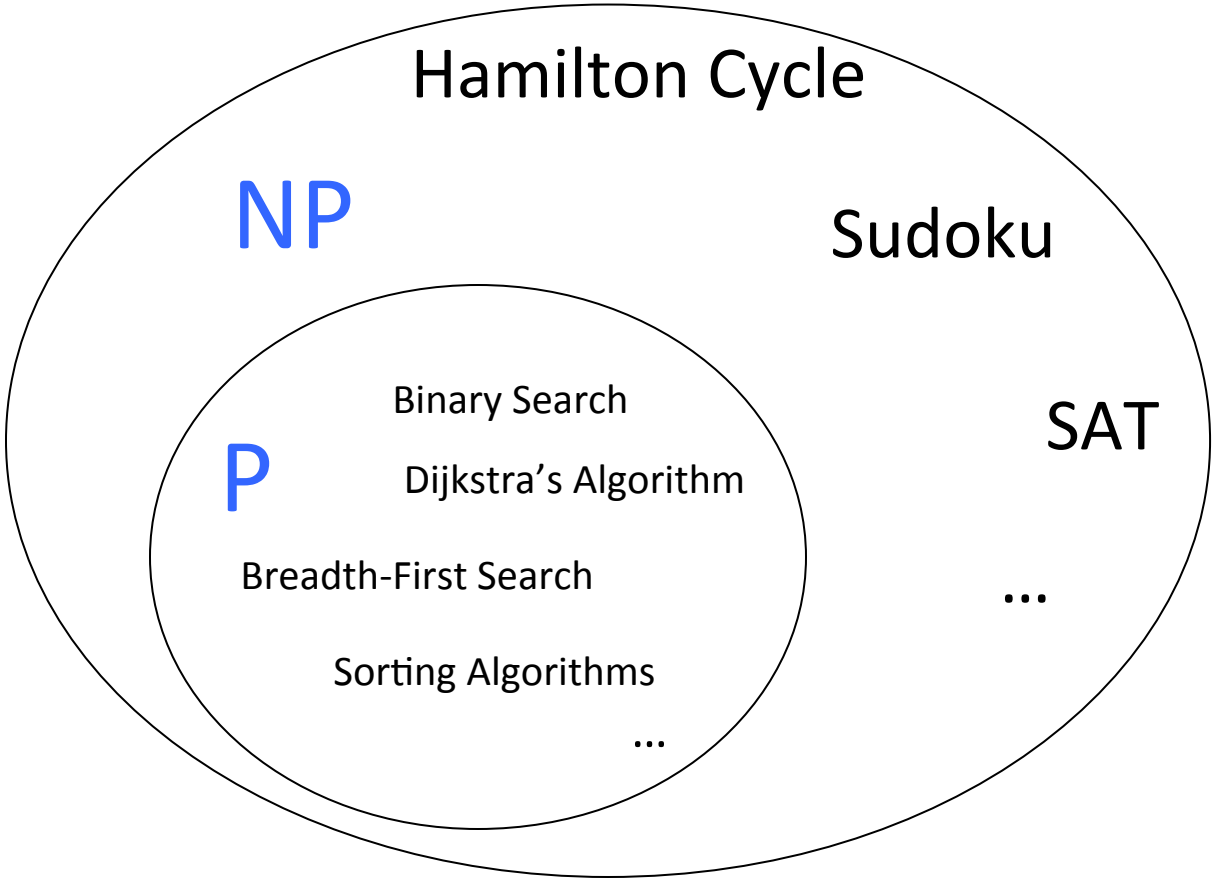
What is an efficient algorithm?



The Class P (polynomial time)



NP (Nondeterministic Polynomial Time)



The P versus NP problem

Is one of the biggest open problems in computer science (and mathematics) today

It's currently unknown whether there exist polynomial time algorithms for NP-complete problems

- We know $P \subseteq NP$, but does $P = NP$?
- People generally believe $P \neq NP$, but no proof yet

What do these NP problems look like?

Sudoku

2			3		8		5	
		3		4	5	9	8	
		8			9	7	3	4
6		7		9				
9	8						1	7
				5		6		9
3	1	9	7			2		
	4	6	5	2		8		
	2		9		3			1

3x3x3

Sudoku

2	9	4	3	7	8	1	5	6
1	7	3	6	4	5	9	8	2
5	6	8	2	1	9	7	3	4
6	5	7	1	9	2	3	4	8
9	8	2	4	3	6	5	1	7
4	3	1	8	5	7	6	2	9
3	1	9	7	8	4	2	6	5
7	4	6	5	2	1	8	9	3
8	2	5	9	6	3	4	7	1

3x3x3

Sudoku

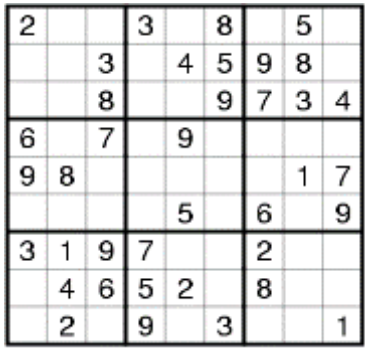
	F		2						6			C	B	3	
	C				4	8	E	A			0		D		
D	A	8			3		2	7	F			6		5	
6			E	D	F		C		8						7
	9	3		7					A						2
E						6	F	5		8	4		3		1
C	8		1	3	9	D		0	2		E				
	D		6		5	E	B		1					0	4
9	6					1		F	3	2		0		A	
				4		A	8		D	0	9	B		2	5
2		A		0	D		5	6	C						F
5						2					A		4	8	
B						4		1		A	2	F			0
	0		7			F	3	C		D			2	9	B
		5		1			A	9	0	B				D	
	2	D	A			9						1		4	

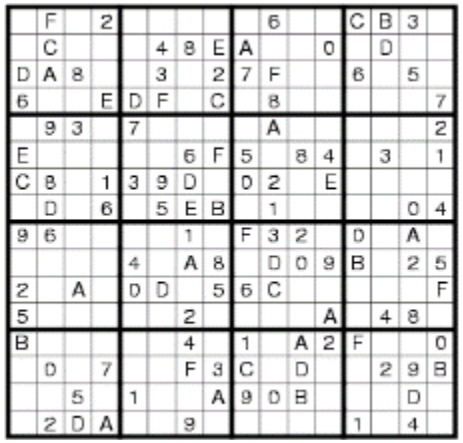
4x4x4

Sudoku

0	F	9	2	A	7	5	1	4	6	E	D	C	B	3	8
7	C	1	3	6	4	8	E	A	B	5	0	2	D	F	9
D	A	8	4	9	3	B	2	7	F	C	1	6	0	5	E
6	5	B	E	D	F	0	C	2	8	9	3	4	A	1	7
4	9	3	5	7	1	C	0	D	A	F	B	8	E	6	2
E	B	7	0	2	A	6	F	5	9	8	4	D	3	C	1
C	8	F	1	3	9	D	4	0	2	6	E	5	7	B	A
A	D	2	6	8	5	E	B	3	1	7	C	9	F	0	4
9	6	4	8	E	B	1	7	F	3	2	5	0	C	A	D
3	7	C	F	4	6	A	8	E	D	0	9	B	1	2	5
2	1	A	B	0	D	3	5	6	C	4	8	7	9	E	F
5	E	0	D	F	C	2	9	B	7	1	A	3	4	8	6
B	3	6	9	C	E	4	D	1	5	A	2	F	8	7	0
1	0	E	7	5	8	F	3	C	4	D	6	A	2	9	B
8	4	5	C	1	2	7	A	9	0	B	F	E	6	D	3
F	2	D	A	B	0	9	6	8	E	3	7	1	5	4	C

4x4x4





n x n x n

Sudoku

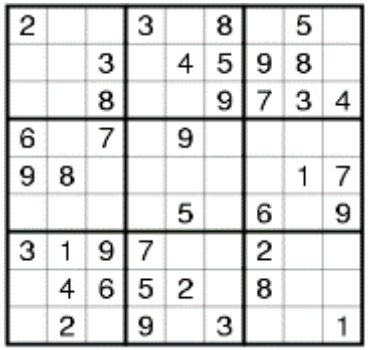
Suppose you have an algorithm S(n) to solve n x n x n

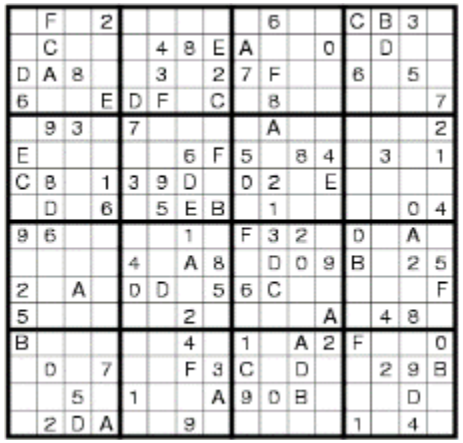
V(n) time to verify the solution

Fact: $V(n) = O(n^2 \times n^2)$

Question: is there some constant such that

$S(n) = O(n^{constant})$?





⋮

n x n x n

Sudoku

P vs NP problem

=

Does there exist an algorithm for solving n x n x n Sudoku that runs in time p(n) for some polynomial p( ) ?

The P versus NP problem (informally)

Can every problem whose solution can be verified *in polynomial time* by a computer also be solved *in polynomial time* by a computer?

To check if a problem is in NP

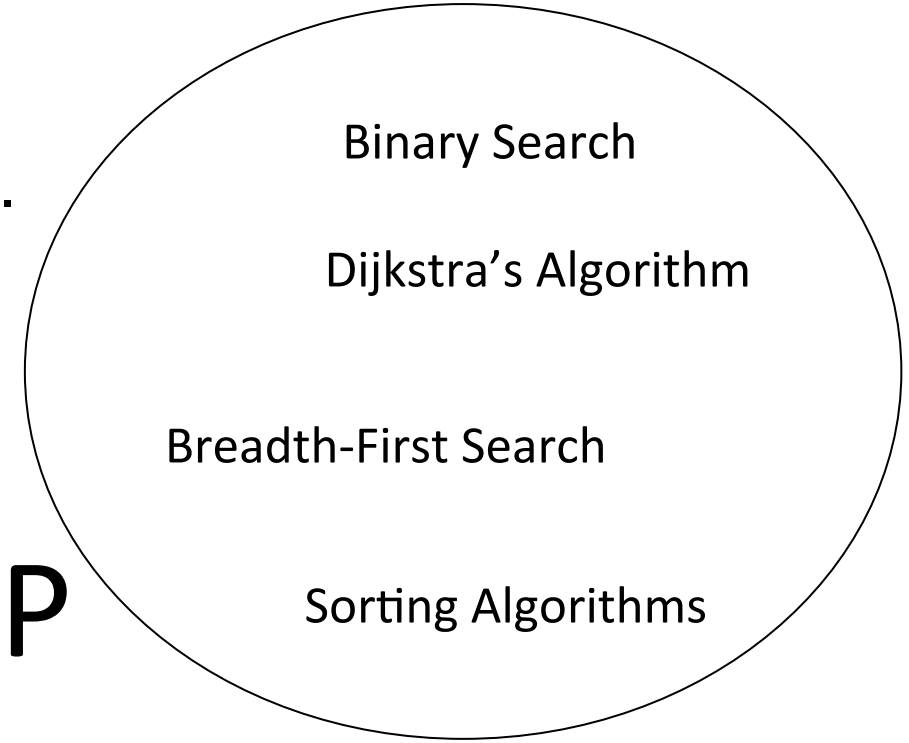
- Phrase the problem as a yes/no question
  - If we can prove any yes instance is correct (in polynomial time), it is in NP
  - If we can also answer yes or no to the problem (in polynomial time) without being given a solution, it is in P

The Class P

The class of all sets that can be verified in polynomial time.

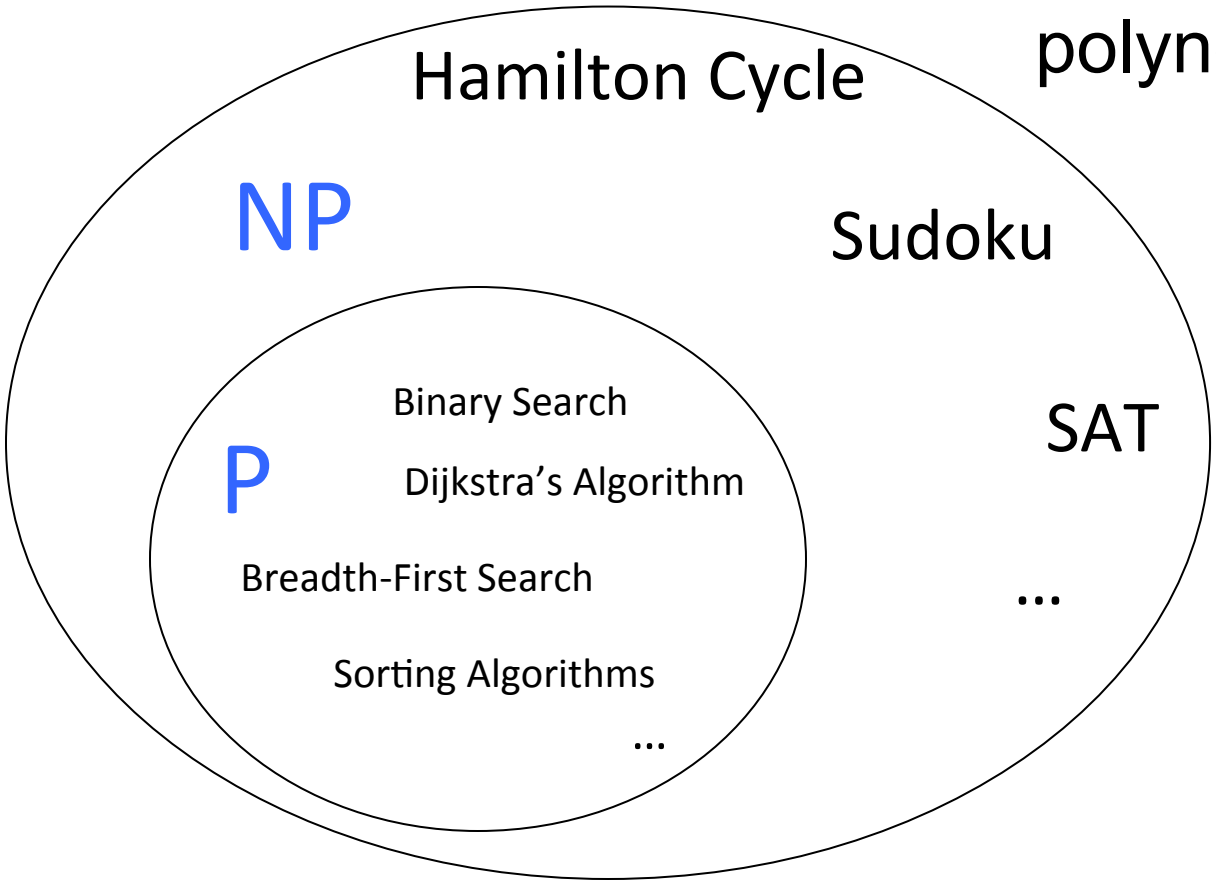
AND

The class of all decision problems that can be decided in polynomial time.



NP

The class of all sets that can be verified in polynomial time.



Sudoku

Input: n x n x n sudoku instance

Output: YES if this sudoku has a solution

NO if it does not

The Set “SUDOKU”

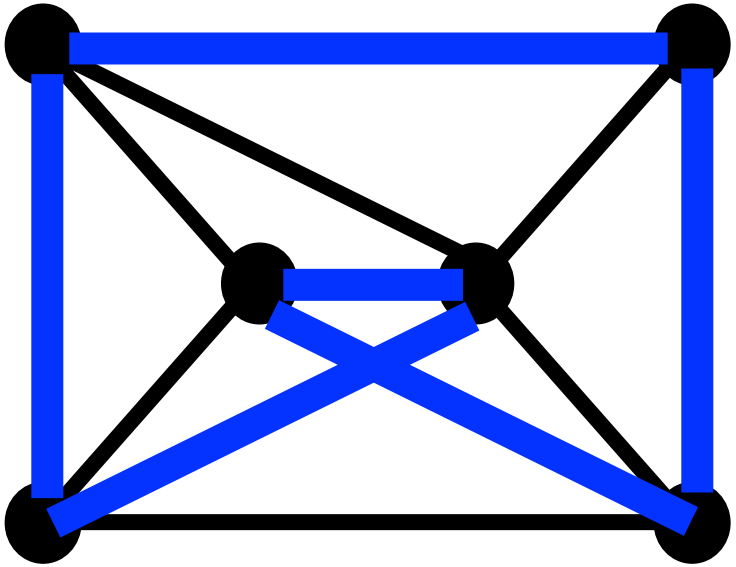
SUDOKU = { All solvable sudoku instances }

Hamilton Cycle

Given a graph G = (V,E), is there a cycle that visits all the nodes exactly once?

YES if G has a Hamilton cycle

NO if G has no Hamilton cycle



The Set "HAM"

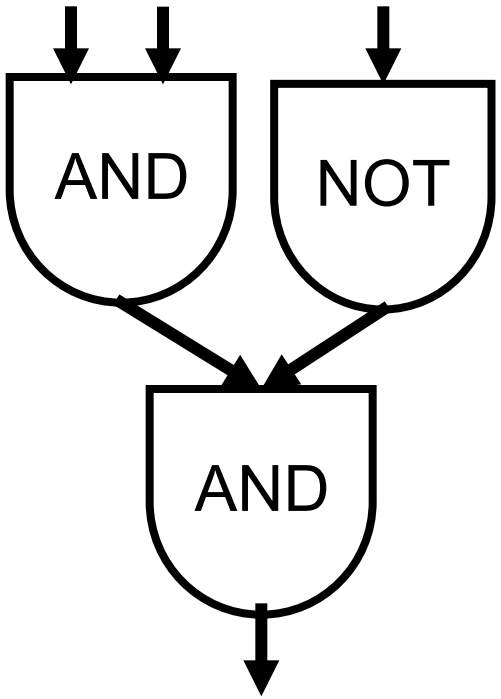
HAM = { graph G | G has a Hamilton cycle }

Circuit-Satisfiability

Input: A circuit C with one output

Output: YES if C is satisfiable

NO if C is not satisfiable



The Set “SAT”

SAT = { all satisfiable circuits C }

Verifying Membership

Is there a short "proof" I can give you to verify that:

$G \in$ HAM?
$G \in$ Sudoku?
$G \in$ SAT?

Yes: I can just give you the cycle, solution, circuit

The Class NP

The class of sets for which there exist
"short" proofs of membership
(of polynomial length)
that can "quickly" verified
(in polynomial time).

Fact: $P \subseteq NP$

Recall: The algorithm doesn't have to find the proof; it just needs to be able to verify that it is a "correct" proof.

Summary: P versus NP

NP: "proof of membership" in a set can be verified in polynomial time.

P: in NP (membership verified in polynomial time)

AND membership in a set can be decided in polynomial time.

Fact: $P \subseteq NP$

Question: Does $NP \subseteq P$?

i.e. *Does P = NP?*

People generally believe $P \neq NP$, but no proof yet

Why Care?

NP Contains Lots of Problems We Don't Know to be in P

Classroom Scheduling

Packing objects into bins

Scheduling jobs on machines

Finding cheap tours visiting a subset of cities

Finding good packet routings in networks

Decryption

…

OK, OK, I care...

How could we prove that NP = P?

We would have to show that every set in NP has a polynomial time algorithm…

How do I do that?
It may take a long time!
Also, what if I forgot one of the sets in NP?

How could we prove that NP = P?

We can describe just one problem L in NP, such that if this problem L is in P, then NP $\subseteq$ P.

It is a problem that can capture all other problems in NP.

The “Hardest” Set in NP

We call these problems NP-complete

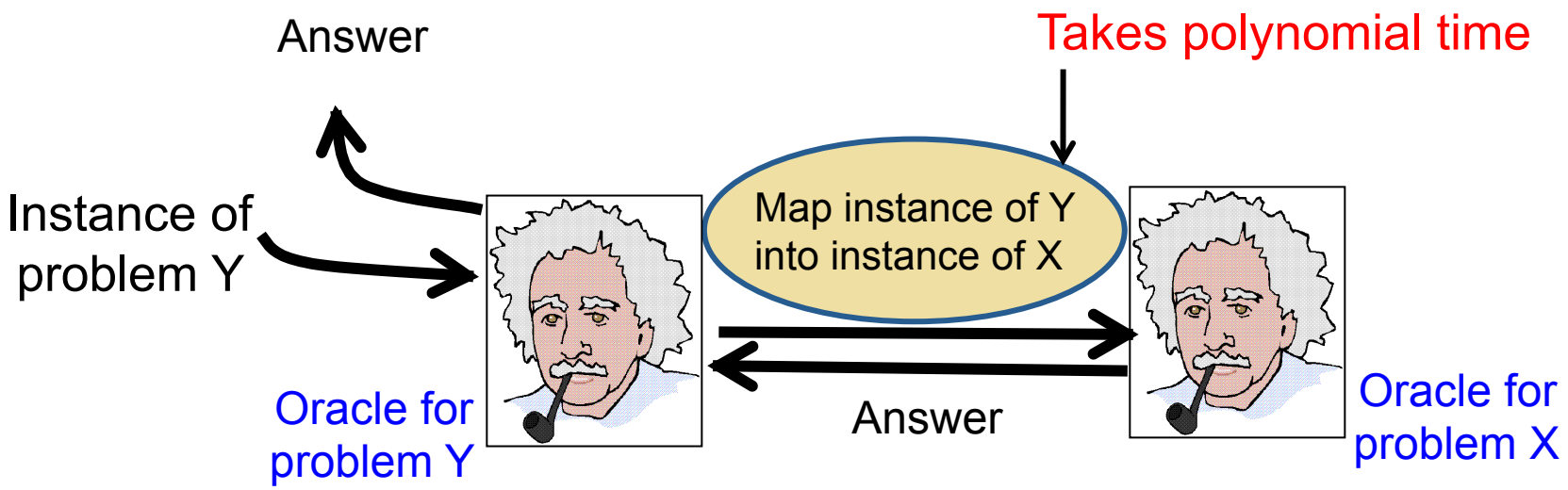
Theorem [Cook/Levin]

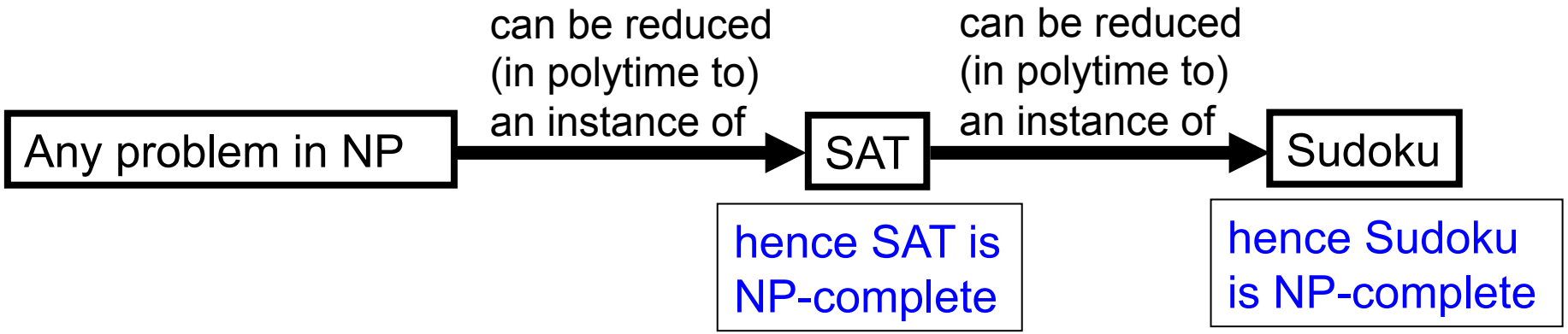
SAT is one problem in NP, such that if we can show SAT is in P, then we have shown NP = P.

SAT is a problem in NP that can capture all other languages in NP.

We say SAT is NP-complete.

Poly-time reducible to each other





NP-complete: The "Hardest" problems in NP

Sudoku Clique

SAT Independent-Set

3-Colorability HAM

These problems are all "polynomial-time equivalent"
i.e., each of these can be reduced to any of the others in polynomial time

If you get a polynomial-time algorithm for one,
you get a polynomial-time algorithm for ALL.
(you get millions of dollars, you solve decryption, … etc.)