Complexity of a problem

- We measure the time to solve a problem of input size n by a function $T(n)$ which measures the running time.
- $T(n)$ is an upper bound on the time for all inputs of size n .
- We only focus on the most significant term in $T(n)$. (Big-O notation).

Examples

$$T(n) = 3n^2 - n + 6. \qquad O(n^2)$$

$$T(n) = 3n \log n + 50n - 1 \qquad O(n \log n)$$

$$T(n) = 2^n + n^3 \qquad O(2^n)$$

Some problems

- Adding n numbers. $O(n)$
- Sorting n items. $O(n \log n)$
- Multiplying 2 n by n matrices $O(n^{2.37})$
- Finding the shortest route between 2 points in a network with n roads $O(n \log n)$
- Solving a system of n linear equations $O(n^3)$

These are all polynomial functions.

2^n , $n!$, and n^n , are non-polynomial functions.

P

- P = {Problems that can be solved in polynomial time }
- P is roughly the class of problems that can be solved efficiently.
- P is independent of
  - computer hardware (non-quantum)
  - operating system
  - programming language

What about problems which we have not put into P?

Problems not known to be in P

- Traveling Salesman Problem
- Formula satisfiability
- Many more ....

Satisfiability:

Input: A boolean formula, e.g.

$$(x_1 \cup x_2) \cap (\overline{x_1} \cup x_4 \cup x_6)$$

Is there a setting that makes this true?
Yes: e.g. $x_1 = T; x_6 = T$

Not always possible

$$(x_1 \cup x_2) \cap (\overline{x_1} \cup x_2) \cap (x_1 \cup \overline{x_2}) \cap (\overline{x_1} \cup \overline{x_2})$$

Hard problems

- We'd like to be able to say - There is no polynomial time algorithm for TSP.
- Unfortunately, we are really bad at making statements about problems being hard.

Hardness - The state of the art

- Some problems are not solvable by any computer (Does a program have an infinite loop).
- You have to read the input.
- You have to print the output.
- Sorting takes at least $n \log n$ time (assuming a reasonable model of a computer).
  - There are $n!$ possible orderings for n numbers.
  - Each step of the algorithm can "eliminate" half the orderings.
  - You can halve $n!$ $\log_2(n!) \approx n \log n$ times.

Not much else is known

Hard problems

You are asked to solve new problem X. You can't.

- We'd like to be able to say - There is no polynomial time algorithm for X.
- We can say - I've worked on it for a while, and I'm not smart enough to solve X.
- Not good for job security, self-respect, impressing people at cocktail parties, etc.

Hard problems

You are asked to solve new problem X. You can't.

- We'd like to be able to say - There is no polynomial time algorithm for X.
- We can say - I've worked on it for a while, and I'm not smart enough to solve X.
- Not good for job security, self-respect, impressing people at cocktail parties, etc.

Face saving theory:

- We can say - I've worked on it for a while, and I'm not smart enough to solve X, but neither are thousands of other very smart people, who have been working for many years.
- Good for job security and self-respect. Not too successful at most cocktail parties.

NP-completeness

Brief introduction to NP-completeness

NP: The set of problems whose solution can be verified in polynomial time.

Verification of TSP: Given a permutation, is its length less than some value B ?

Verification of satisfiability: Given a setting of the booolean variables, is the formula true?

$$(x_1 \cup x_2) \cap (\overline{x_1} \cup x_4 \cup x_6)$$

$$x_1 = T; x_2 = F; x_4 = F; x_6 = T$$

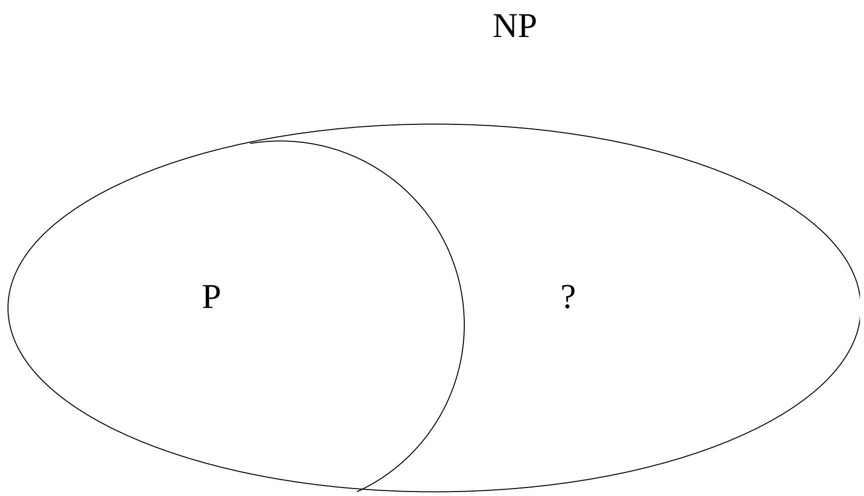
Verification of sorting: Given a list of numbers, is it already in sorted order.

$$3, 6, 9, 2$$

Verification

- Clearly, verification is no harder than solving a problem from scratch.
- Informally, problems for which you can enumerate all possible solutions and check them are in NP.
- Is verification significantly easier than solving a problem?

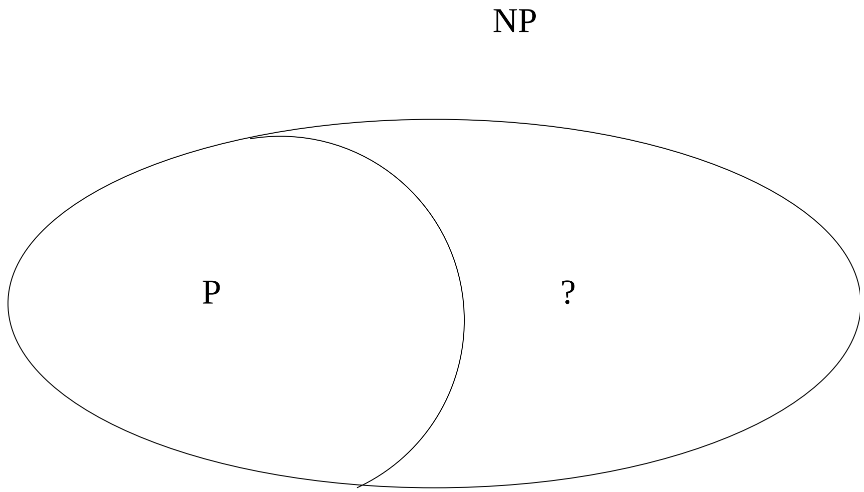
NP-completeness



What does the question mark area look like? (Is it empty?)

NP-complete problems are the "hardest" problems in NP.

NP-completeness

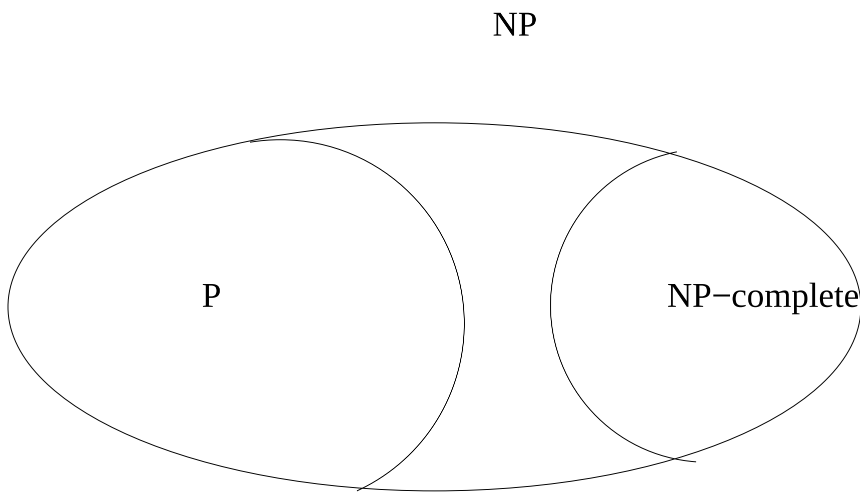


What does the question mark area look like? (Is it empty?)

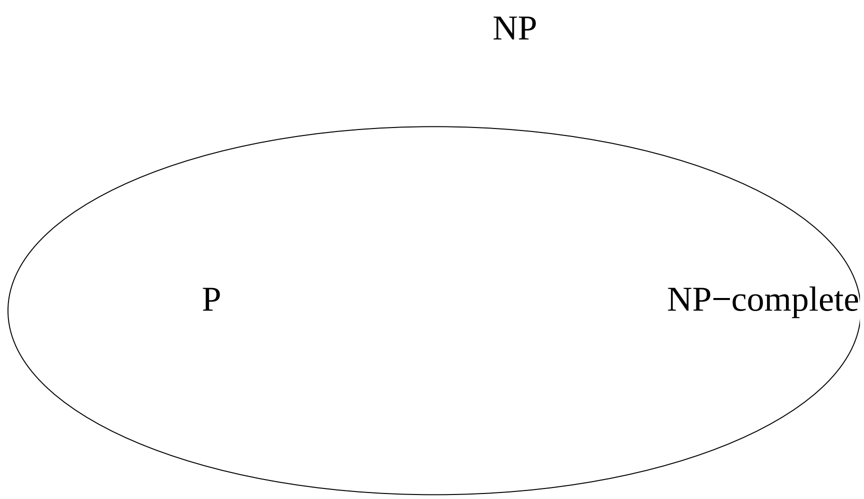
NP-complete problems are the "hardest" problems in NP.

NP-completeness

If $P \neq NP$



If $P = NP$



The power of NP-completeness

The power comes from the diverse group of problems, e.g.

- traveling salesman problem
- formula satisfiability
- longest path between two points
- assigning frequencies in a cellphone network
- minimum phylogenetic tree
- minimum energy protein folding
- scheduling a factory
- 3-dimensional ising model
- the game geography
- nearest vector in a lattice
- ...

Either all of these are in P, or none are in P.

How do we show a new problem N is NP-complete?

- Choose a known NP-complete problem K .
- Show that K reduces to N.

K reduces to N means that we can use N as a "subroutine" for solving K.
N easy $\Rightarrow K$ easy

Contrapositive:
K hard $\Rightarrow N$ hard