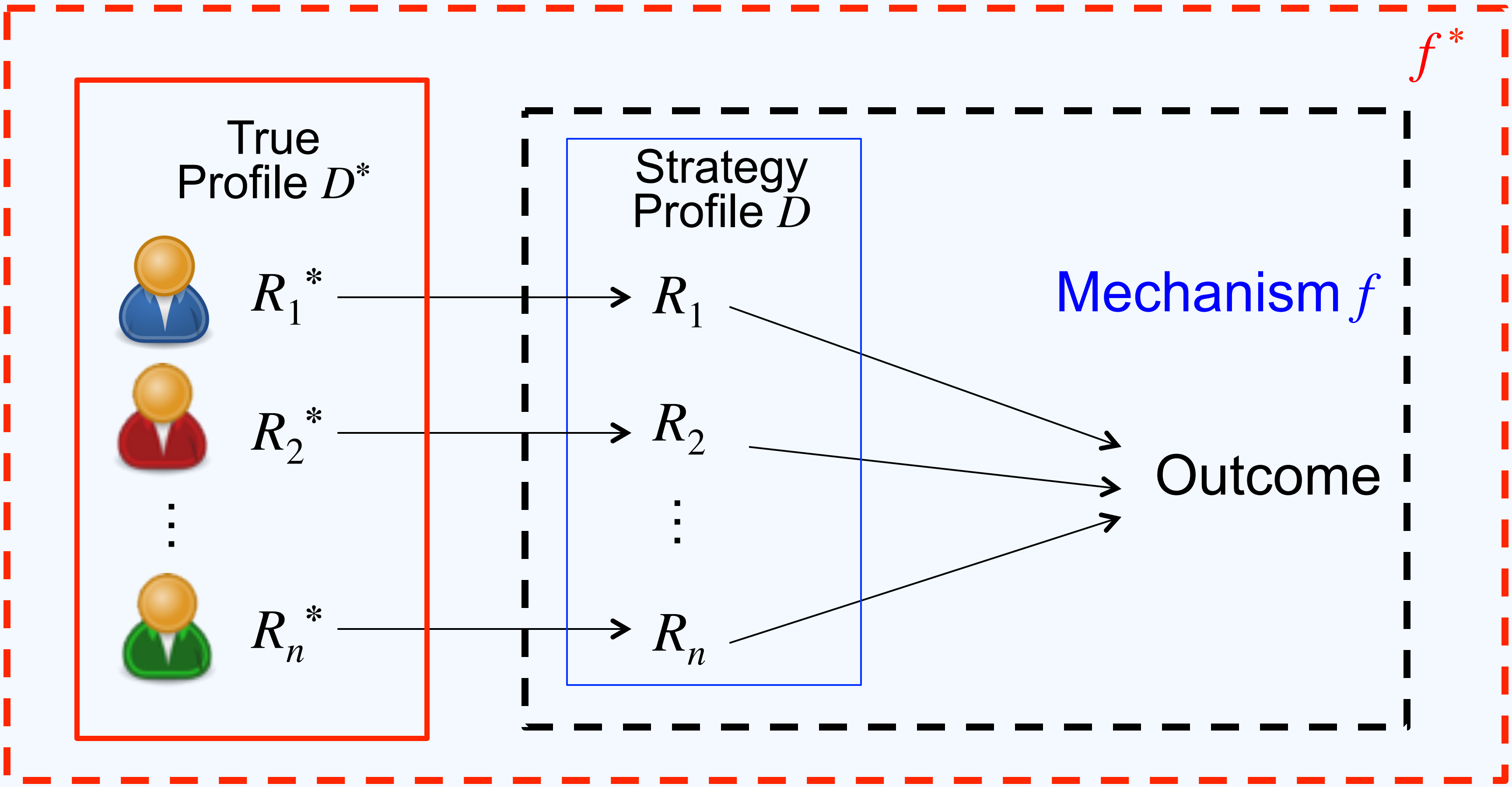
Last class: mechanism design

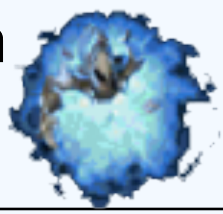


- Agents (players): $N=\{1,\ldots,n\}$
- Outcomes: O
- Preferences (private): total preorders over O
- Message space (c.f. strategy space): S_j for agent j
- Mechanism: $f : \Pi_j S_j \rightarrow O$

Today's schedule

- Computation (completely different from previous classes)!
- Linear programming: a useful and generic technic to solve optimization problems
- Basic computational complexity theorem
  - how can we formally measure computational efficiency?
  - how can we say a problem is harder than another?
- HW2 out
- Remember you will be need to answer the "why what how" question

The last battle

	strength	minerals	gas	supply
Zealot 	1	100	0	2
Stalker 	2	125	50	2
Archon 	10	100	300	4

- Available resource:

mineral	gas	supply
2000	1500	30

- How to maximize the total strength of your troop?

Computing the optimal solution

	str	m	g	s
Z	1	100	0	2
S	2	125	50	2
A	10	100	300	4
Resource		2000	1500	30

- Variables
  - x_Z: number of Zealots
  - x_S: number of Stalkers
  - x_A: number of Archons
- Objective: maximize total strength
- max $1x_Z + 2x_S + 10x_A$
- Constraints
  - mineral: $100x_Z + 125x_S + 100x_A \leq 2000$
  - gas: $0x_Z + 50x_S + 300x_A \leq 1500$
  - supply: $2x_Z + 2x_S + 4x_A \leq 30$
  - $x_Z, x_S, x_A \geq 0$, integers

Linear programming (LP)

- Given
  - Variables x: a row vector of m positive real numbers
  - Parameters (fixed)
    - c: a row vector of m real numbers
    - b: a column vector of n real numbers
    - A: an $n \times m$ real matrix
- Solve

$$\max \quad cx^T$$
$$\text{s.t.} \quad Ax^T \le b,\ x \ge 0$$

- Solutions
  - x is a feasible solution, if it satisfies all constraints
  - x is an optimal solution, if it maximizes the objective function among all feasible solutions

General tricks

- Possibly negative variable x
  - $x = y - y'$
- Minimizing cx^T
  - max $-cx^T$
- Greater equals to $ax^T \geq b$
  - $-ax^T \leq -b$
- Equation $ax^T = b$
  - $ax^T \geq b$ and $ax^T \leq b$
- Strict inequality $ax^T < b$
  - no “theoretically perfect” solution
  - $ax^T \leq b-\varepsilon$

Integrality constraints

- Integer programming (IP): all variables are integers
- Mixed integer programming (MIP): some variables are integers

Efficient solvers

- LP: can be solved efficiently
  - if there are not too many variables and constraints
- IP/MIP: some instances might be hard to solve
  - practical solver: CPLEX free for academic use!

My mini "course project"

- n professors $N = \{1, 2, \ldots, n\}$, each has one course to teach
- m time slots S
  - slot i has capacity c_i
  - e.g. M&Th 12-2 pm is one slot
  - any course takes one slot
- Degree of satisfaction (additive) for professor j
  - S_j^1, S_j^2, …, S_j^k are subsets of S. s_j^1, …, s_j^k are real numbers
    - if j is assigned to a slot in S_j^l, then her satisfaction is s_j^l
    - E.g. S_j^1 is the set of all afternoon classes
  - N_j is a subset of N
    - For each time confliction (allocated to the same slot) of j with a professor in N_j, her satisfaction is decreased by 1
- Objective: find an allocation
  - utilitarian: maximize total satisfaction
  - egalitarian: maximize minimum satisfaction

Modeling the problem linearly

- How to model an allocation as values of variables?
  - for each prof. j, each slot i, a binary (0-1) variable x_{ij}
  - each prof. j is assigned to exactly one course
    - for every j, $\Sigma_i\, x_{ij} = 1$
  - each slot i is assigned to no more than c_i profs.
    - for every i, $\Sigma_j\, x_{ij} \leq c_i$
- How to model the satisfaction of prof. j?
  - allocated to S_j^l if and only if $\Sigma_{i \in S_j^l}\, x_{ij} = 1$
  - confliction with $j^* \in N_j$: $x_{ij}+x_{ij^*}=2$ for some i
    - for each pair of profs. (j, j^*), a variable y_{jj^*}
    - s.t. for every i, $y_{jj^*} \geq x_{ij}+x_{ij^*} - 1$
- How to model the objective?
  - utilitarian: max $\Sigma_j\, [\Sigma_l (s_j^l\, \Sigma_{i \in S_j^l}\, x_{ij}) - \Sigma_{j^* \in N_j}\, y_{jj^*}]$
  - egalitarian: max $\min_j\, [\Sigma_l (s_j^l\, \Sigma_{i \in S_j^l}\, x_{ij}) - \Sigma_{j^* \in N_j}\, y_{jj^*}]$

Full MIP (utilitarian)

- variables
  - x_{ij} for each i, j — Prof. j's course is assigned to slot i
  - integers: y_{jj*} for each j, $j*$ — j and $j*$ have confliction
- max $\Sigma_j [\Sigma_l(s_j^l \Sigma_{i\in S_j^l} x_{ij})\text{-}\Sigma_{j*\in N_j} y_{jj*}]$

s.t. for every j: $\Sigma_i x_{ij} = 1$ — j gets exactly one slot

for every i: $\Sigma_j x_{ij} \leq c_i$ — slot capacity

for every $i,j,j*$: $y_{jj*} \geq x_{ij}+x_{ij*} - 1$ — j and $j*$ are both assigned to i

Full MIP (egalitarian)

- variables
  - x_{ij} for each i, j — Prof. j's course is assigned to slot i
  - integers: y_{jj^*} for each j, j^* — j and j^* have confliction
- max x

s.t. for every j: $\Sigma_i\, x_{ij}$ = 1 — j gets exactly one slot

for every i: $\Sigma_j\, x_{ij} \leq c_i$ — slot capacity

for every i,j,j^*: $y_{jj^*} \geq x_{ij} + x_{ij^*} - 1$ — j and j^* are both assigned to i

for every j: $x \leq \Sigma_l (s_j^l\, \Sigma_{i \in S_j^l}\, x_{ij}) - \Sigma_{j^* \in N_j}\, y_{jj^*}$

Why this solves the problem?

Any optimal solution
to the allocation problem

$$\Updownarrow$$

Any optimal solution
to the MIP

Variantions

- You can prioritize professors (courses) add weights
  - e.g. a big undergrad course may have heavier weight
- You can add hard constraints too
  - e.g. CS 1 must be assigned to W afternoon
- Side comment: you can use other mechanisms e.g. sequential allocation

Your homework

- Given m, and m positional scoring rules, does there exist a profile such that these rules output different winners?
- Objective: output such a profile with fewest number of votes

Theory of computation

- History
- Running time of algorithms
  - polynomial-time algorithms
- Easy and hard problems
  - P vs. NP
  - reduction

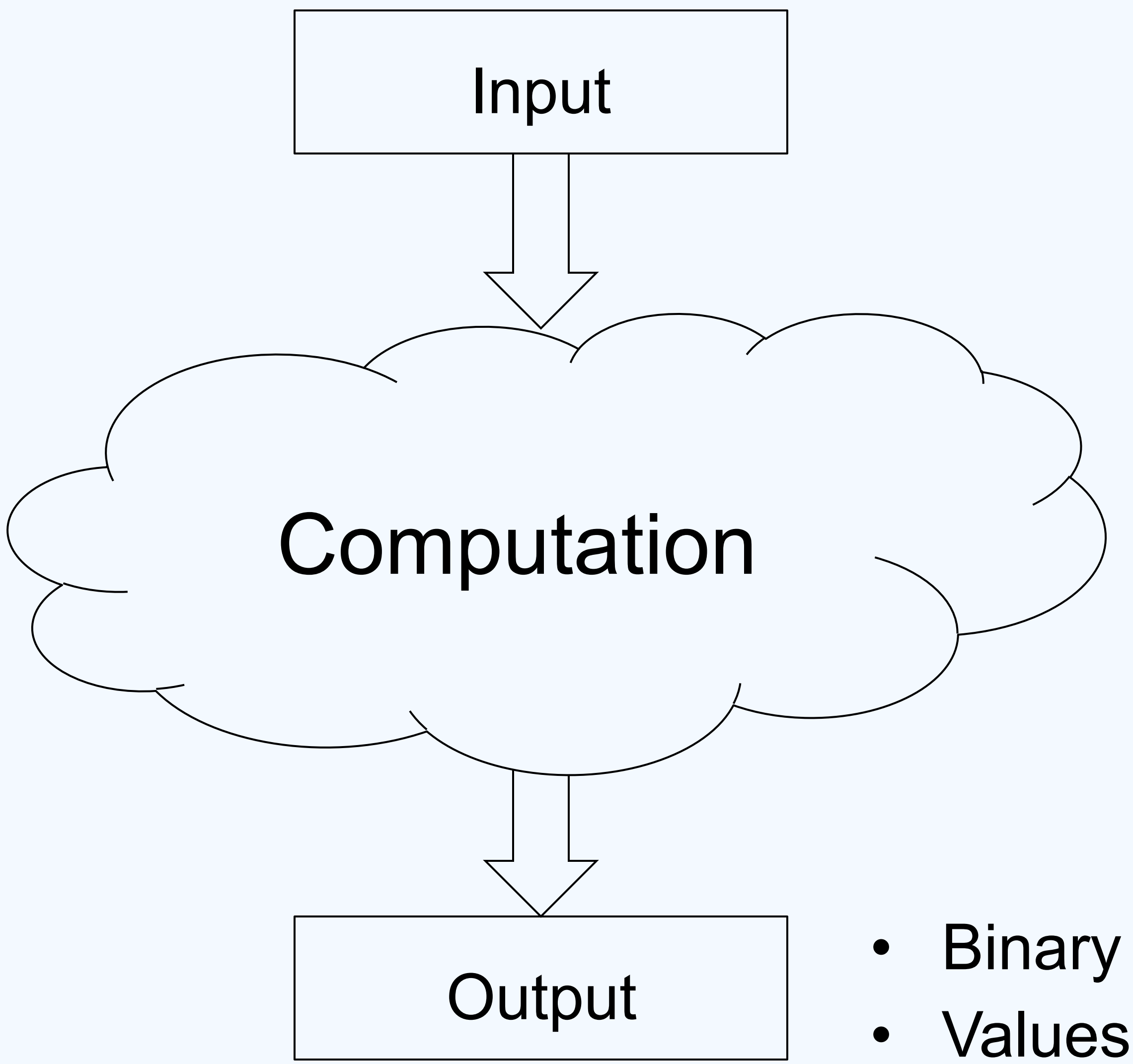
Hilbert's tenth problem

"Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients:

To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers."

- Diophantine equation: $p(x_1,\ldots,x_m) = 0$
  - p is a polynomial with integer coefficients
- Does there exist an algorithm that determines where the equation has a solution?
- Answer: No!

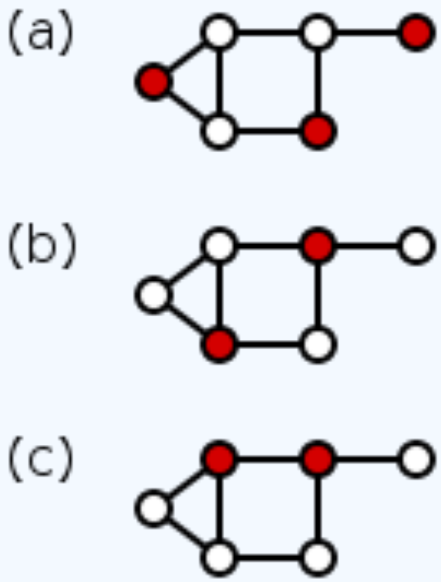
What is computation?



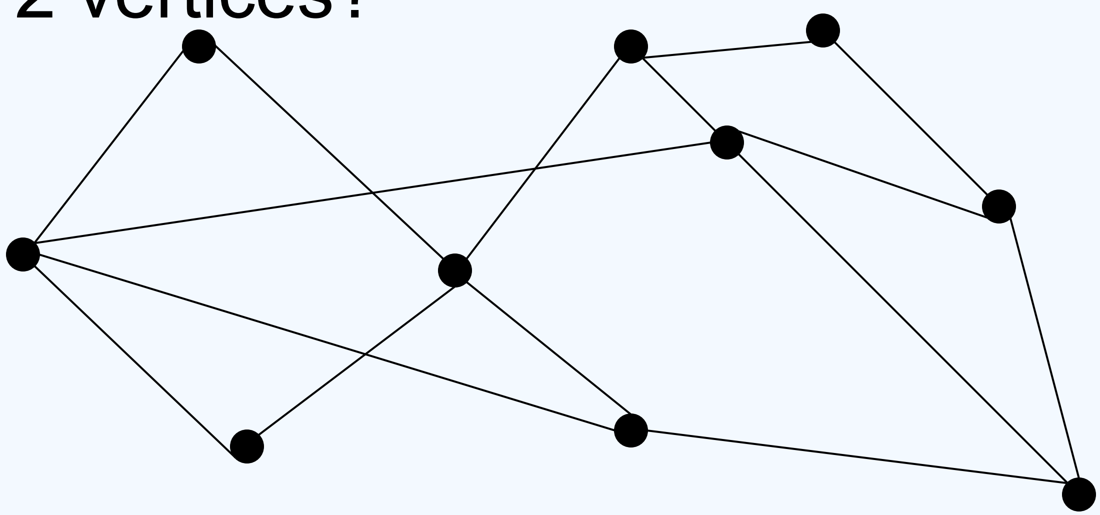
- Binary (yes-no): decision problem
- Values: optimization problem

A decision problem

- Dominating set (DS):
  - Given a undirected graph and a natural number k.
  - Does there exists a set S of no more than k vertices so that every vertex is either
    - in S, or
    - connected to at least one vertex in S

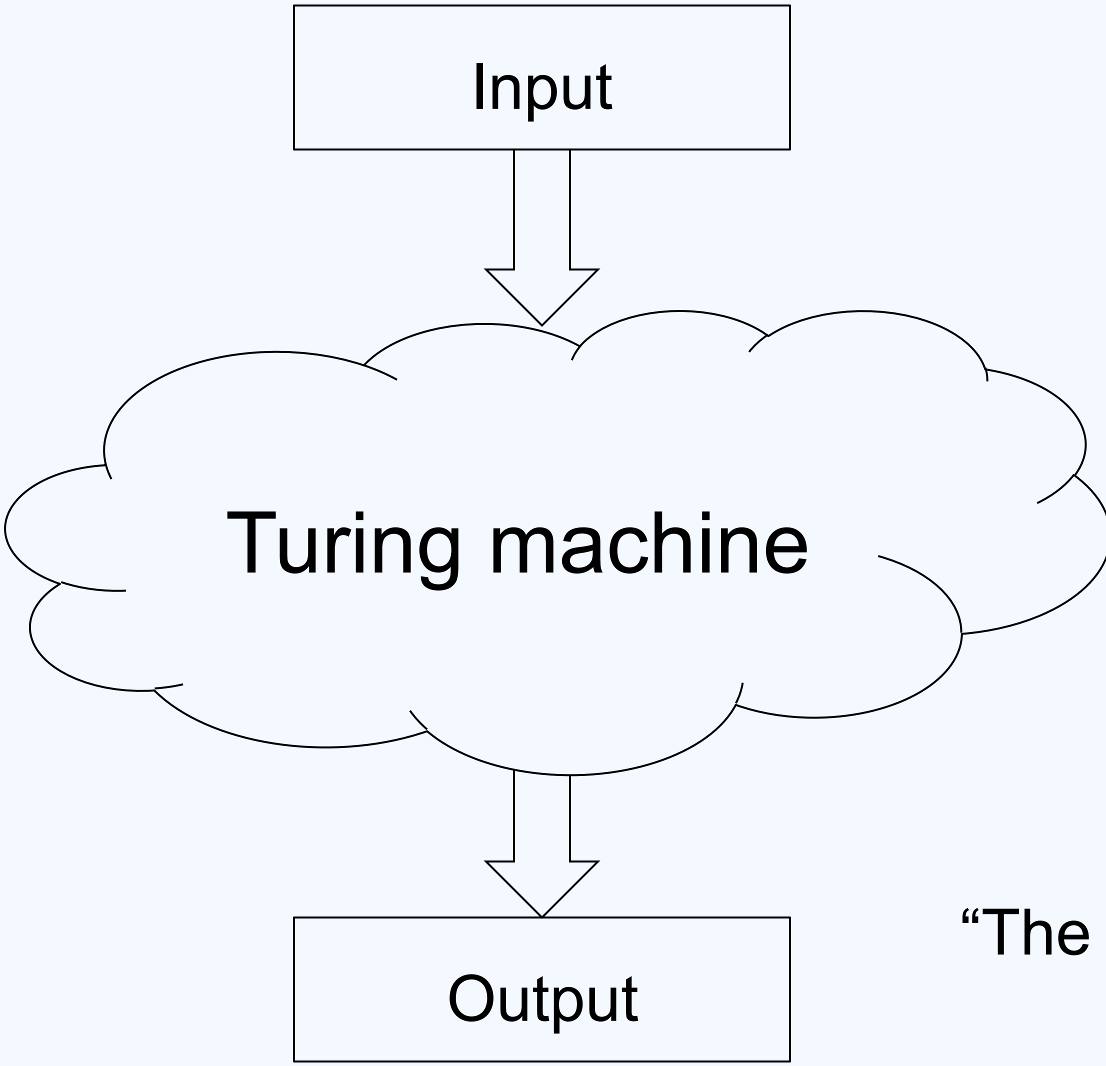


- Example: Does there exists a dominating set of 2 vertices?



How to formalize computation?

Church-Turing conjecture



Alonzo Church 1903–1995

Alan Turing 1912–1954

Turing's most cited work
"The chemical basis of morphogenesis"

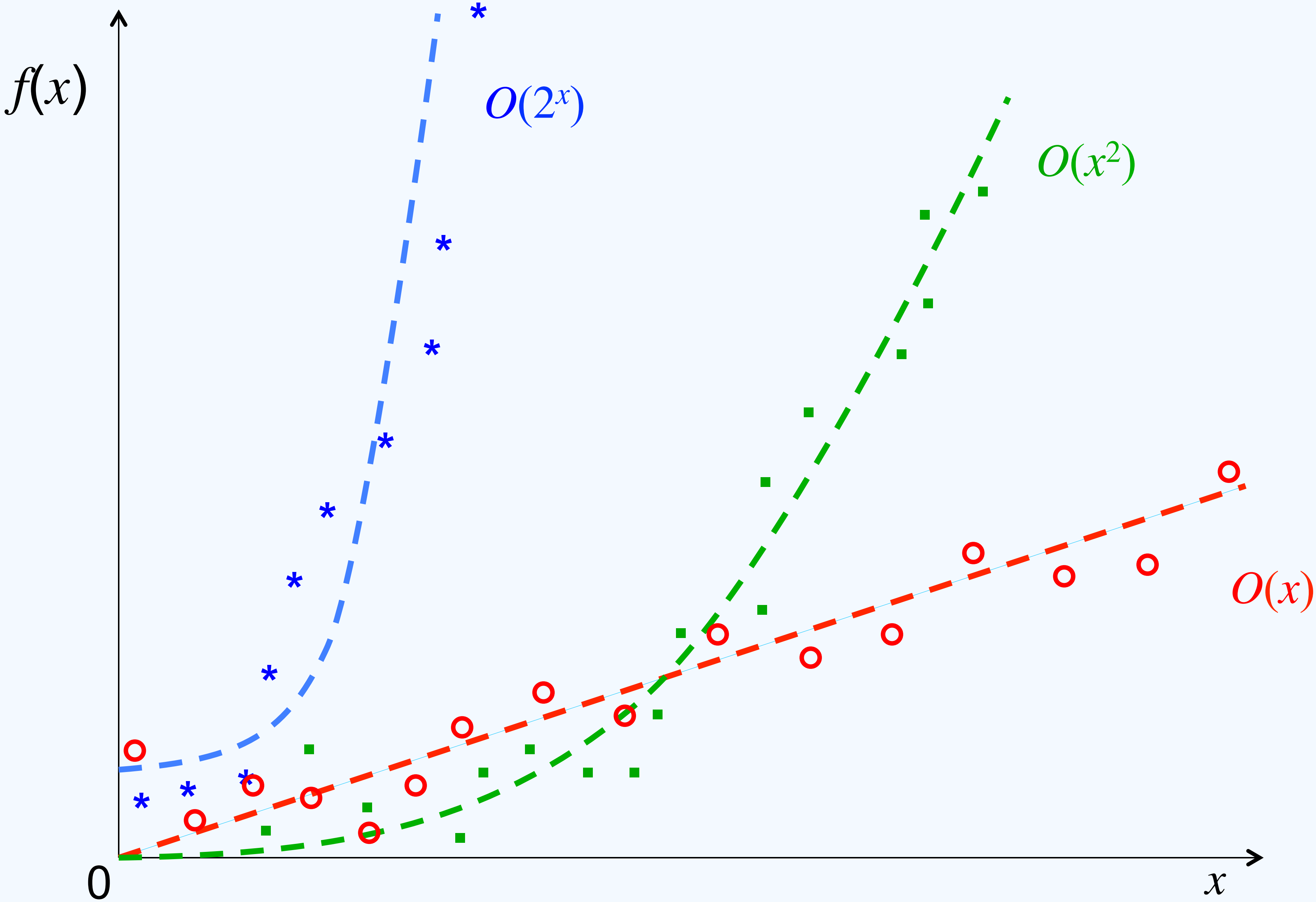
Running time of an algorithm

- Number of "basic" steps
  - basic arithmetic operations, basic read/write, etc
  - depends on the input size
- $f(x)$: number of "basic" steps when the input size is x
  - computer scientists care about running time of "large" problems

The big O notation

- Given two real-valued functions $f(x)$, $g(x)$
- f is $O(g)$ if there exists a constant c and x_0 such that
  - for all $x>x_0$, $|f(x)| \le cg(x)$
  - f is $O(2x) \Leftrightarrow f$ is $O(x) \Rightarrow f$ is $O(x^2)$
- g is an asymptotic upper bound of f
  - up to a constant multiplicative factor
- Can we say an $O(x^2)$ algorithm always runs faster than an $O(x)$ algorithm?
  - No
- Can we say an $O(x)$ algorithm runs faster than an $O(x^2)$ algorithm?
  - No

Example



Polynomial-time algorithms

- An algorithm is a polynomial-time algorithm if
  - there exists $g(x)$ = a polynomial of x
    - e.g. $g(x) = x^{100}-89x^7+3$
  - such that f is $O(g)$
- Running time is asymptotically bounded above by a polynomial function
- Considered "fast" in most part of the computational complexity theory

P vs. NP

- P (polynomial time): all decision problems that can be solved by deterministic polynomial-time algorithms
  - "easy" problems
  - Linear programming
- NP (nondeterministic polynomial time, not "Not P"): all decision problems that can be solved by nondeterministic Turing machines in polynomial-time
  - "believed-to-be-hard" problems
- Open question: is it true that P = NP?
  - widely believed that P ≠ NP
  - $1,000,000 Clay Mathematics Institute Prize
- If P = NP,
  - current cryptographic techniques can be broken in polynomial time
  - many hard problems can be solved efficiently

To show a problem is in:

- P: design a polynomial-time deterministic algorithm to give a correct answer
- NP: for every output, design a polynomial-time deterministic algorithm to verify the correctness of the answer
  - why this seems harder than P?
  - Working on Problem 5’ vs reading the solution

How to prove a problem is easier than another?

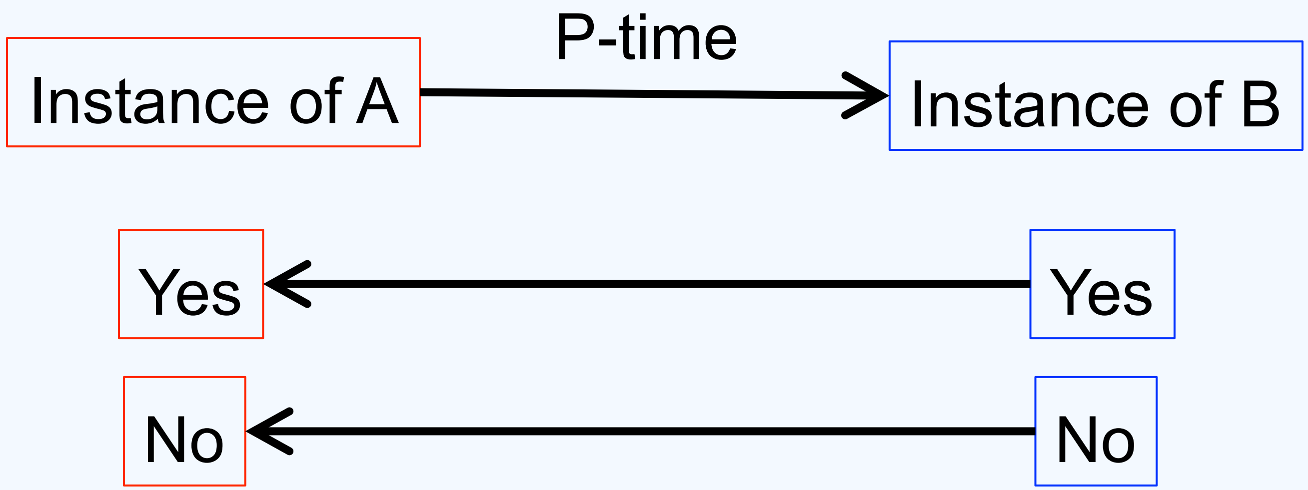
- Of course you can define it by
  - the fastest algorithm for A is faster than the fastest algorithm for B
  - What is the fastest algorithm?
- A mathematician and an engineer are on desert island. They find two palm trees with one coconut each. The engineer climbs up one tree, gets the coconut, eats. The mathematician climbs up the other tree, gets the coconut, climbs the other tree and puts it there. "Now we've reduced it to a problem we know how to solve."

Complexity theory

- Provides a formal, mathematical way to say "problem A is easier than B"
- Easier in the sense that A can be reduced to B efficiently
  - how efficiently? It depends on the context

How a reduction works?

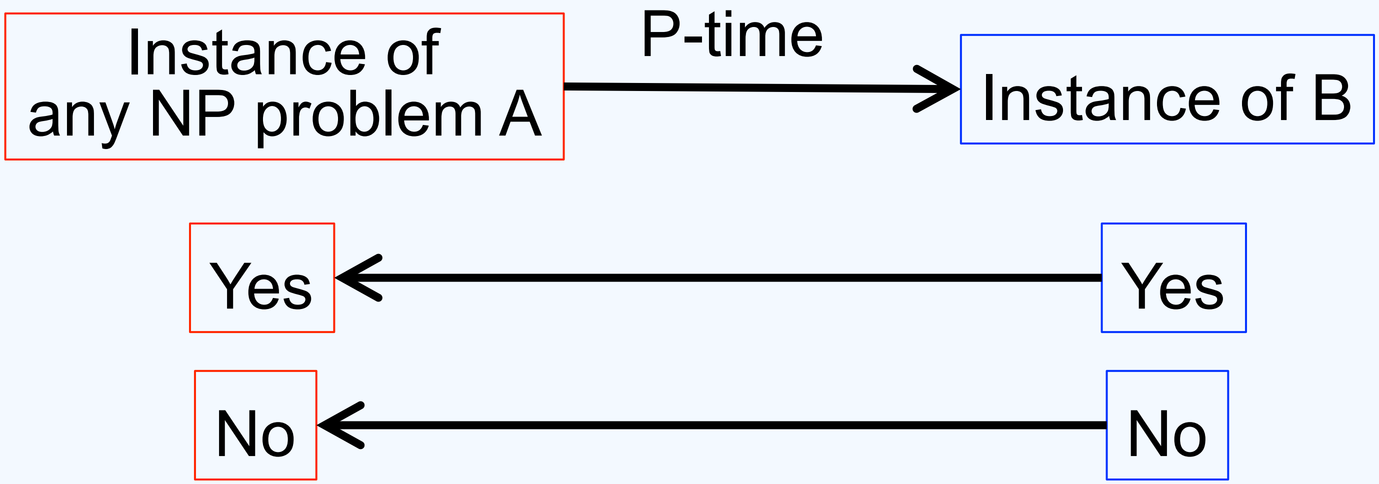
- Polynomial-time reduction: convert an instance of A to an instance of another decision problem B in polynomial-time
  - so that answer to A is “yes” if and only if the answer to B is “yes”



- If you can do this for all instances of A, then it proves that B is HARDER than A w.r.t. polynomial-time reduction
- But it does not mean that B is always harder than A for all instances

NP-hard problems

- "Harder" than any NP problems w.r.t. polynomial-time reduction
  - suppose B is NP-hard



- NP-hard problems
  - Dominating set
  - Mixed integer programming

Any more complexity classes?

- https://complexityzoo.uwaterloo.ca/Complexity_Zoo

Wrap up

- Linear programming
- Basic computational complexity
  - big O notation
  - polynomial-time algorithms
  - P vs. NP
  - reduction
  - NP-hard problems

Next class

- More on computational complexity
  - more examples of NP-hardness proofs
- Computational social choice: the easy-to-compute axiom
  - winner determination for some voting rules can be NP-hard!
  - solve them using MIP in practice