Classical CS, computational complexity

Turing machine (Alan Turing, 1936)

Toy model of computation: tape, state register, table

- Infinite tape, cells with symbols (x) from a finite alphabet;
  at start the tape contains finite amount of data
- Read/write head (read, write, and move by -1, 0, or $+1$)
- State register (finite for each machine number of states q; overall not limited)
- Table (program): finite set of instructions $(q, x) \to (q', x', m)$

state, symbol on tape, move $(-1,0,1)$

(many small variations of this description)

Universal Turing machine: a Turing machine, which can simulate any other Turing machine

Construction: Let us assign a number T to each possible program (machine); countable infinity. We can write this number T on the tape; then tape initially contains T (program) and X (data)

Church-Turing Thesis

Simple formulation of Church-Turing Thesis:

Any algorithm can be simulated using a Turing machine

Several modifications of this statement

- simulated efficiently ("strong Church-Turing Thesis")
  (ratio of steps at most polynomial in the problem size)

- simulated using a probabilistic Turing machine
  (it was realized that random numbers can make algorithms more efficient)

QC probably violates the Church-Turing Thesis (though this is not clear)

Computable and uncomputable

Not everything is computable

Example: **Halting problem** Turing machine stops (halts) for a specific state (read from the table, need certain combination q, x). Either eventually stops or runs forever (e.g., cycle).

Can we create a machine, which can look at a program (+ initial data) and tell us if this program eventually stops or runs forever?

No, this is impossible

Proof 1) Proof by contradiction: Assume such machine exits

$$h(T,X) = \begin{cases} 1, \text{halts} \\ 0, \text{does not halt} \end{cases}$$

here T is a number assigned to any program (finite alphabet), X is data

Consider the case $T = X$ (somewhat strange)

2) Construct a machine M, which works in the opposite way:
$M(X)$ calculates $h(X,X)$ and then halts if $h(X,X) = 0$ (i.e., if machine X with data X does not stop), but cycles if $h(X,X) = 1$ (i.e., when X with X does not stop)

3) Machine M has its own number (T_M). What about $M(T_M)$? Does it halt or not?

If halts, then $h(T_M, T_M) = 1$, so $M(T_M)$ does not halt

If does not halt, then $h(T_M, T_M) = 0$, so $M(T_M)$ should halt

Contradiction, QED

Computational complexity classes

Besides computable/uncomputable, there is a question of how much time (crudely) it takes to compute.

Computational complexity classes

P (polynomial-time): solvable on a Turing machine in a polynomial time, $poly(n)$, where n is the size of the problem (e.g., number of bits, size of a matrix, etc.)

Examples: multiplication, matrix inversion, etc.

NP (non-deterministic, polynomial-time): can be solved in polynomial time by chance (e.g., by a non-deterministic Turing machine), solution can be checked in polynomial time

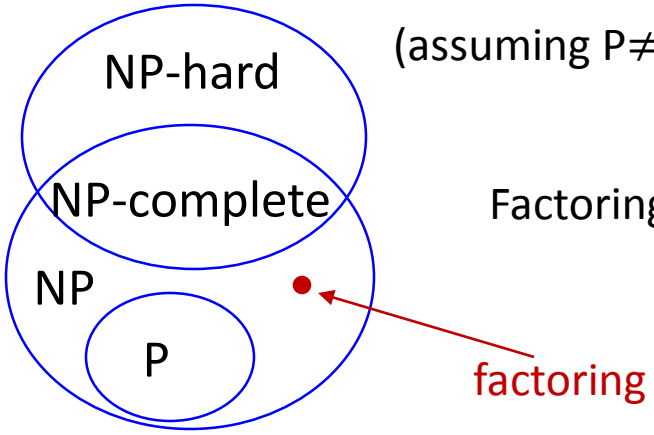
NP-complete: problems, to which any NP problem can be reduced (if we can solve one NP-complete problem, then we can solve any NP-problem)

Examples: Boolean satisfiability (SAT): is $f(x) = 1$ for at least one input x?

Travelling salesman (decision version): is there a route for a cost less than x?

NP-hard: not easier than NP-complete

Complexity classes (cont.)



(assuming P≠NP; not proven, but suspected)

Factoring is <u>not</u> an NP-complete problem (most likely)

Some other classes and relations

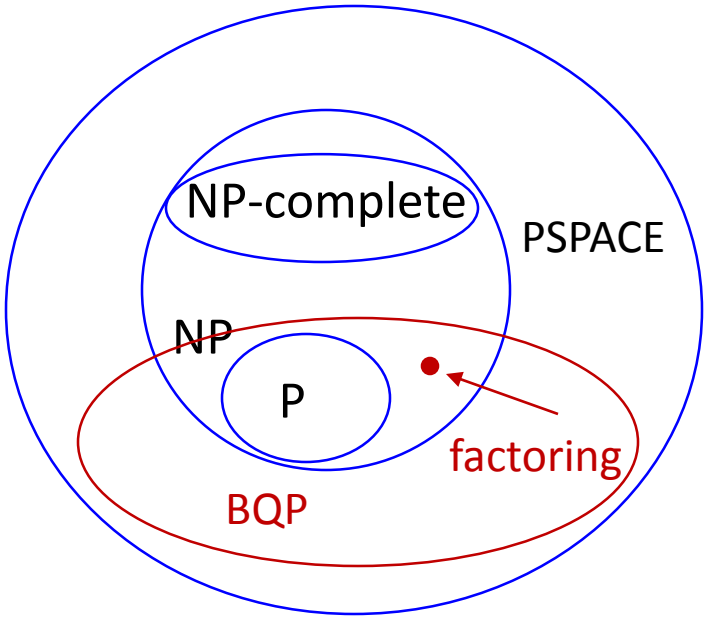
$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PP} \subseteq \mathrm{PSPACE}$ $\mathrm{P} \subseteq \mathrm{BPP} \subseteq \mathrm{PP}$ (unknown relation between NP and BPP)

PP (probabilistic, polynomial time): class of decision problems, solvable by a probabilistic Turing machine in polynomial time, with error probability $< 1/2$.
Answers "yes" or "no"; if actually "yes", algorithm should say "yes" with probability $> 1/2$; if actually "no", it says "yes" with prob. $\leq 1/2$.

PSPACE: polynomial in space (length of the tape)

BPP (bounded-error, probabilistic, polynomial time): decision problem solvable by a probabilistic Turing machine with error probability $< 1/3$ always.

Complexity class for QC



BQP (bounded-error, quantum, polynomial time): class of decision problems solvable by a quantum computer in polynomial time, with error probability $< 1/3$ in all instances (quantum analogue of BPP).

$$\mathrm{P} \subseteq \mathrm{BPP} \subseteq \mathrm{BQP} \subseteq \mathrm{PP} \subseteq \mathrm{PSPACE}$$

Relation between BQP and NP is not known

Likely, BQP is larger than BPP, but does not include NP-complete; possibly extends beyond NP into PSPACE

Scalable quantum computer

(DiVincenzo criteria)

(arXiv:quant-ph/0002077,
Fortschr. Phys., 48, 771, 2000)

1) Scalable physical system of well-characterized qubits
2) Initializable in a simple state
3) Long decoherence time ($\gg$ gate time)
4) Universal set of quantum gates
5) Efficient measurement capability

+ for networking (added later):

6) Interconversion of stationary and “flying” qubits
7) Faithful transmission of flying qubits between “stations”

In principle, all these criteria have been demonstrated, but separately (in different systems). We need everything in the same system.

This is very hard. Is it possible? Time will tell.