Computation

What does it mean to "compute"?
→ Not obvious! Took until 1936 for Alan Turing to formalise this properly.

Def (Turing Machine)

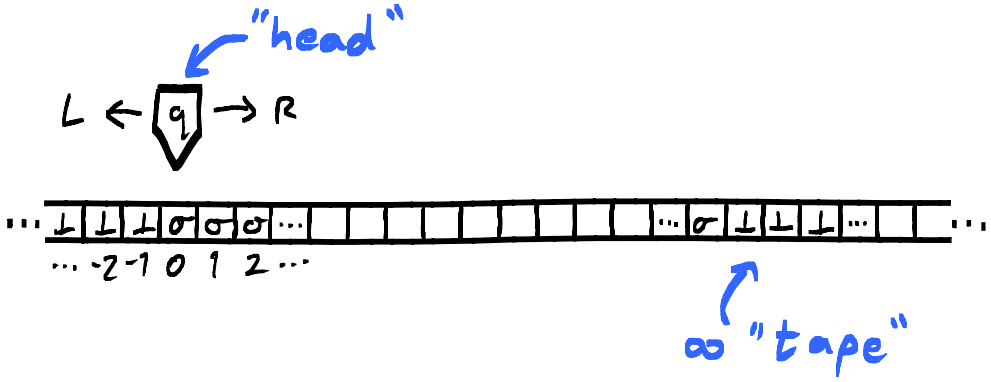
Tuple (Q, Σ, δ)

$Q = \{q_0, q_1, \ldots, q_f\}$
finite set of "states"

$\Sigma = \{\sigma_0, \sigma_1, \ldots, \perp\}$
finite set of "symbols"; distinguished "blank" symbol $\perp$

$\delta : Q \times \Sigma \rightharpoonup Q \times \Sigma \times \{L, R\}$
partial function from state, symbol pairs to state, symbol, direction triples

How does a TM work?



- Start:
  - head over tape cell 0
  - input: finite string of non-blank symbols written on tape from cell 0 onwards
  - rest of tape filled with blank symbols
  - TM in state q_0

- Each time step:
  - tape symbol σ under head
    TM currently in internal state q
  - $\delta(\sigma, q) = (\sigma', q', D)$ ← $\in \{L, R\}$
  - write σ' on cell under head
    update internal state to q'
    move head one step in direction D

- Finish:
  - if TM ever enters state q_f, halt
  - output: whatever is left written on tape

Can formalise this rigorously if desired.
(Exercise)

Def (Computable function)

$f: D \subseteq \mathbb{N} \longrightarrow \mathbb{N}$ is (partial) **computable** if $\exists$ Turing Machine which, given input $n \in D$, halts in finite time with output $f(n)$; and given input $n \in \mathbb{N} \setminus D$ it never halts.

If $D = \mathbb{N}$, f is **total computable**.

Why is this the right definition of computation?
E.g. why not a semi-∞ tape?
Or allow head to stay still as well as move L/R?
Or completely different models of computation?
(Lambda calculus, μ-recursive functions,
cellular automata, game of life, billiard balls...)

Turns out details of computational model don't matter.

Church-Turing thesis

The class of computable functions is independent of the model of computation, for all reasonable models of computation.

Not a theorem, since not a statement that can be proven. Viewed as an axion (Gödel), definition (kleene), natural law (Post), empirical fact (Copeland) by various people.

Proven for many specific models of computation e.g. lamda calculus [Turing], μ-recursive functions [Turing, Church, kleene], cellular automata [von Neumann], game of life [Conway?], billiard balls [Friedkin & Toffoli], ...

There exist uncomputable functions!

Thm (Undecidability of Halting) [Turing 1936]

$h : \mathbb{N} \times \mathbb{N} \to \{0, 1\}$

$$h(m, i) = \begin{cases} 1 & \text{TM } m \text{ halts on input } i \\ 0 & \text{otherwise.} \end{cases}$$

h is not computable.

Proof (informal)

Let $f(m, i)$ be any total computable function of two arguments.

Construct TM $g(i)$ (input) $= \begin{cases} 0 \leftarrow \text{output} & f(i, i) = 0 \\ \text{loops forever} & f(i, i) = 1 \end{cases}$

- $f(g, g) = 0 \Rightarrow g(g)$ halts $\Rightarrow h(g, g) = 1 \neq f(g, g)$
- $f(g, g) = 1 \Rightarrow g(g)$ loops $\Rightarrow h(g, g) = 0 \neq f(g, g)$

$\therefore$ $h \neq$ any computable function. □

To make this formal, would need to
- specify how to represent TM's in $\mathbb{N}$
  → gives enumeration of TM's
- explicitly construct g

(Exercise)

Note: proof is variant of Cantor's diagonal argument:

$s(m,i)$	0	1	2	⋯ i	g	⋯
0	0	1	1	⋯	1	
1	1	1	0	⋯	0	
2	0	0	1	⋯	1	
⋮ m	⋮	⋮	⋮	⋱	⋮	
g	1	0	0	⋯	0	⋯
⋮	⋮	⋮	⋮		⋮	⋱
$g(i)$	0	0	0	⋯	0	⋯
$h(g,i)$	1	0	0	⋯	1	⋯

Corollary (Gödel's first incompleteness theorem)
In any consistent, complete, recursive axiomatisation of arithmetic, there exist statements that can be neither proven nor disproven.

first order logic statements about natural numbers

Proof idea:

- Axiom schema allows us to mechanically enumerate over valid logical statements in arithmetic.
- Halting problem can be expressed as a statement in arithmetic.
- Construct TM that, on input m, i, enumerates over statements until either it finds proof m'th TM halts on input i, or proof that it doesn't.

→ Decides Halting → contradiction. □

Thm (Universal TMs) [Turing 1936]

Can explicitly construct a TM which on input m, i outputs result of running m'th TM on input i (if this halts; otherwise runs forever).

Proof idea:

- Input m contains description of transition rules of m'th TM (in some form one should specify; cf. proof of undecidability of Halting).
- Construct TM which reads m and simulates the specified transition rules. □

Thm (Rice's Thm) [Rice 1953]

Any non-trivial property of partial functions is undecidable.

(Def. "non-trivial": not true (or false) for all functions.)

Proof: Exercise.

Computability theory studies whether things are computable *in principle* on an idealised computer with infinite resources

To study what can be computed on realistic resource-limited computers, need finitary version of theory → Complexity theory...

(Brief) Introduction to Complexity Theory

Complexity theory is the rigorous analysis of the computational resources (space, time, ...) required to solve a problem.

More precisely, it concerns the scaling of the resources as a function of the size of the problem.

Computational Problems

Problem size = amount of information to specify problem

E.g:

Factoring problem

Input: $n \in \mathbb{N}$

Output: factors of n

Problem size ~ # of digits in n ~ $\log(n)$

SAT

Input: boolean expression using vars $\{b_i\}$, operations $\wedge$, $\vee$, $\neg$

Output: { YES if $\exists$ values $b_i \in \{0,1\}$ s.t. expression evaluates to 1
NO otherwise

Problem size ~ # boolean variables

Often interested in decision problems: problems with binary output YES or NO (1 or 0).

Def (Decision Problem)

$f : \{0,1\}^* \longrightarrow \{0,1\}$

↑ input written as bit-string

↑ binary output

(In literature, often see terminology
"Language" $L = \{s \in \{0,1\}^* : f(s) = 1\}$
"Deciding the language" = solving decision problem)

Computational problems often have natural equivalent decision variants,

↖ will make this rigorous later!

E.g:

Factoring (decision variant)

Input: $n, k \in \mathbb{N}$

Output: YES if n has factor $< k$
NO otherwise

Exercise

Show that if you can solve factoring decision problem, you can also find factors.

Circuit Model

Logic gate (or "gate"):

Boolean function on 1 or 2 bits:

$G_1: \{0,1\} \longrightarrow \{0,1\}$ or

$G_2: \{0,1\}^{\times 2} \longrightarrow \{0,1\}$

E.g.

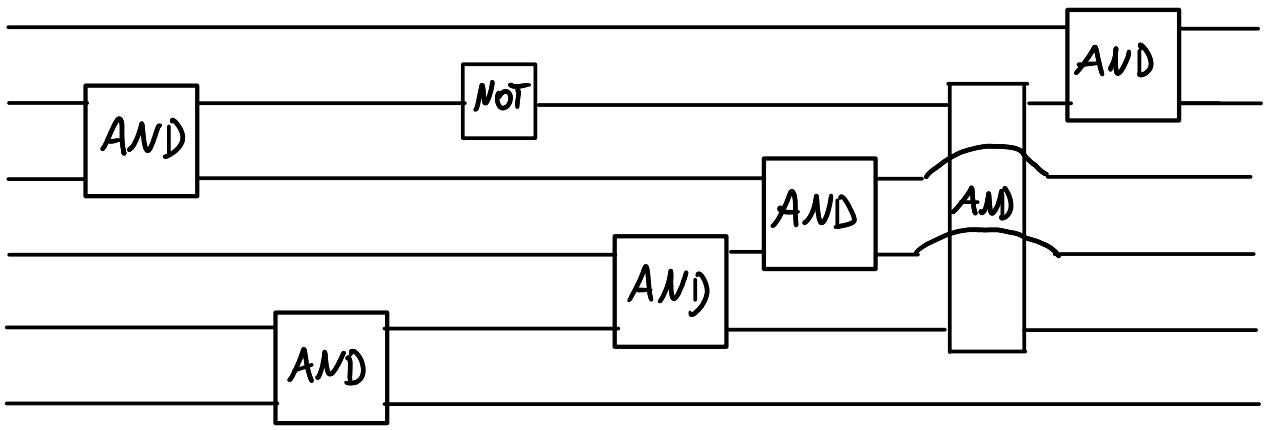
$$AND(x,y) = \begin{cases} 1 & x=y=1 \\ 0 & \text{otherwise} \end{cases}$$

$$NOT(x) = \begin{cases} 0 & x=1 \\ 1 & x=0 \end{cases}$$

Circuit:

Finite sequence of gates each acting on specified subset of bits.

E.g. (circuit diagram):



(Gates don't have to only act on "adjacent" bits — just easier to draw!)

Quantum computation

Quantum gate

Unitary operator on 1 or 2 qubits

$U_1 : \mathbb{C}^2 \longrightarrow \mathbb{C}^2$

$U_2 : \mathbb{C}^2 \otimes \mathbb{C}^2 \longrightarrow \mathbb{C}^2 \otimes \mathbb{C}^2$

E.g.

$$U_{CNOT} = |00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 10| + |10\rangle\langle 11| \cong \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 0 & 1 \\ & & 1 & 0 \end{pmatrix}$$

$$U_Z = |0\rangle\langle 0| - |1\rangle\langle 1| \cong \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Note: unitarity $\Rightarrow$ reversibility:

$U^\dagger U = \mathbb{1} \Rightarrow U$ has inverse gate $U^\dagger$

(Classical computation can also be done reversibly
$\rightarrow$ special case of quantum computation)

Quantum circuit:

Finite sequence of quantum gates each acting on specified subset of qubits

Input $|\psi\rangle$ is "computational basis" state, i.e.

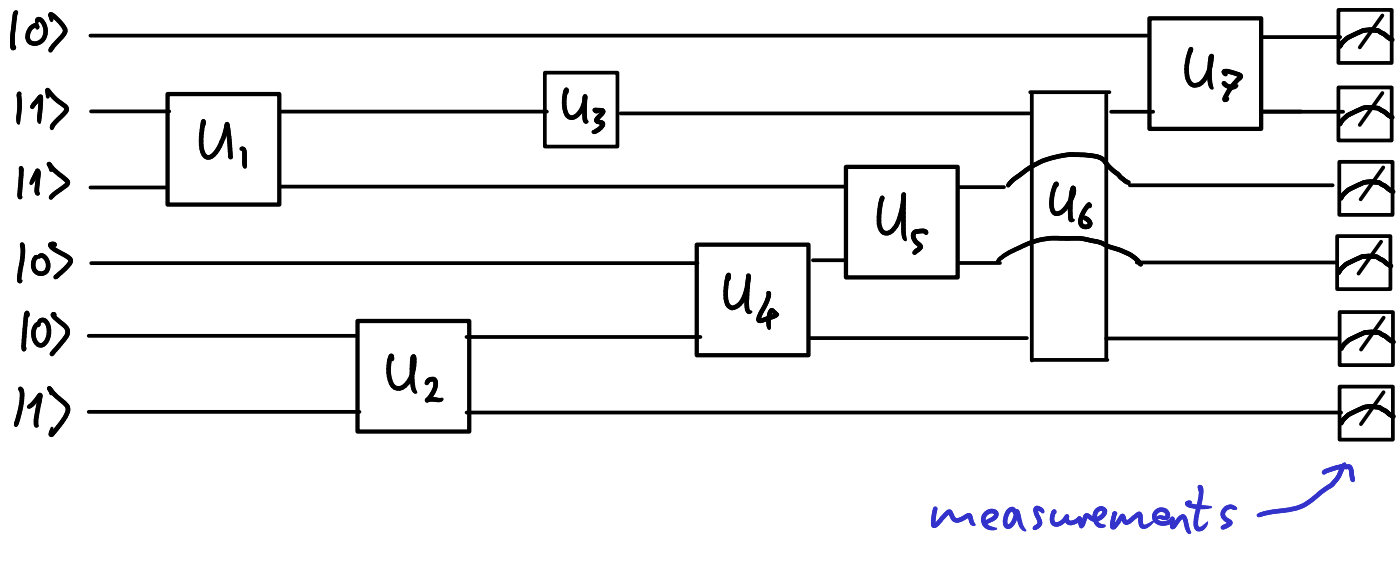
$$|\psi\rangle = \bigotimes_i |x_i\rangle , \quad |x_i\rangle \in \{|0\rangle, |1\rangle\}.$$

Output is outcome of measuring qubits in computational basis, i.e. measure $\{\Pi^{(0)}, \Pi^{(1)}\}$ on each qubit.

⟶ outcome probabilistic !

(Can also define probabilistic classical computation ... but not in this course!)

E.g. (quantum circuit diagram):



Complexity Classes

Classify computational problems according to computational resources required to solve them. (Typically time or space; we will only study time complexity classes here.)

In Turing Machine model of computation:

problem size := # of non-blank cells in input

time := # of steps until halt

space := # of different tape cells accessed by head during computation

Def. P (polynomial-time)

Class of all <u>decision</u> problems solvable in time that scales as some polynomial of input size.

Examples: decision versions of addition, subtraction, finding eigenvalues, primality testing [2000]

Def. NP (non-deterministic polynomial time) ← ignore name!

Class of decision problems for which ∃ poly-time verifier TM V s.t. if answer for input x is

YES:

∃ polynomial-sized "witness" w s.t. $V(x, w) = 1$.

NO:

∀ witnesses w, $V(x, w) = 0$.

Think of NP as game between all-powerful (but untrustworthy) Merlin, and less powerful Arthur who can only run polynomial-time computations.

Arthur asks Merlin question x.
Merlin replies with answer YES/NO.
But Arthur doesn't trust Merlin, so Merlin also gives Arthur a simple (poly-time checkable) proof w that his answer is correct.

NP is class of all problems for which Merlin can <u>convince</u> Arthur of a YES answer ($\exists w$), and <u>cannot trick</u> him into believing a wrong NO answer ($\forall w$).

(Note asymmetry between YES & NO.
Class co-NP is very similar, but with roles of YES & NO swapped.)

Want to generalise these classes to quantum setting. But we have only introduced circuit model of quantum computation.

How are P, NP defined in classical circuit model?

problem size := # input bits

time := # gates in circuit

First attempt:

Def. P/poly

Decision problems solvable with polynomial-sized (classical) circuit.

I.e. problem size n → circuit size $O(\text{poly}(n))$.

But this def. doesn't give same class P as before.

In fact, P/poly contains undecidable problems!

Exercise: Show Halting $\subset$ P/poly.

Issue is that have to construct different circuits for different problem sizes
→ can smuggle too much computation into this.

Need some kind of uniformity condition on sequence of circuits of increasing size

Second attempt:

Def. Uniform circuit family

Sequence of circuits C_n on n bits is uniform if $\exists$ TM which, on input n, outputs description of C_n in polynomial time

Def. P

Class of decision problems solvable with uniform family of polynomial-sized circuits.

Exercise: Show def^{n} of P in TM model & circuit model coincide.

Define NP analogously using uniform family of verifier circuits.

Having dealt with this subtlety, can almost always forget about it. In practice, circuits we construct are invariably uniform.

"Only important thing about TMs is they exist!"

≈ "quantum P"

Def. BQP (bounded-error q. poly-time)

Decision problems:

∃ poly-sized quantum circuit U s.t.

$\Pr(U \text{ outputs } "1") \begin{cases} \geq 2/3 & \text{YES instance} \\ \leq 1/3 & \text{NO instance} \end{cases}$

(Recall

$\Pr(U \text{ outputs } "1")$ — input $|x\rangle \to$ density matrix $|x\rangle\langle x|$

$= \mathrm{tr}[\Pi_1^{(1)} \otimes \mathbb{1}_{2\ldots n}\, U |x\rangle\langle x| U^\dagger]$

$= \langle x| U^\dagger \Pi_1^{(1)} \otimes \mathbb{1}_{2\ldots n}\, U |x\rangle$)

↖ wlog output is given by 1st qubit

Note: Probs. $2/3$, $1/3$ arbitrary.
Could choose $1-\varepsilon$, ε for any const. ε, or even $\varepsilon = \Omega(1/\mathrm{poly}(n))$.

Exercise

Prove class BQP is independent of choice of ε.

Def. QMA (Quantum Merlin-Arthur)

Decision problems.

$\exists$ poly-sized quantum verifier circuit U s.t. if answer on (classical) input x is:

YES:

$\exists$ poly-sized quantum witness $|w\rangle \in \mathbb{C}^{\text{poly}(n)}$ s.t.

$\Pr(U \text{ outputs "1" on input } |x\rangle|w\rangle) \geq \frac{2}{3}$

NO:

$\forall$ states $|w\rangle$,

$\Pr(U \text{ outputs "1" on input } |x\rangle|w\rangle) \leq \frac{1}{3}$

Note: again, probs. $\frac{2}{3}$, $\frac{1}{3}$ arbitrary.

Exercise

Prove this.

Reduction

Lets us compare <u>difficulty</u> of different computational problems.

⟶ rigorously say problem B is harder than problem A.

// "Karp reduction"

<u>Def</u> (poly-time reduction)

A <u>reduces</u> to B if

$\exists$ map $A \longmapsto B$

instance$_a$ ⟶ instance$_b$

s.t. b has answer YES iff a has answer YES.

<u>and</u> map $A \to B$ can be computed by poly-size circuit.

I.e. can solve A by transforming it into B & solving B.

Write "$A \leq B$".

$A \leq B$ & $B \leq A$ $\Rightarrow$ A, B <u>equivalent</u>

⟶ write "$A = B$".

Def. (NP-hard)

Problem B is NP-hard if $\forall A \in NP \quad A \leq B$.

Def. (NP-complete)

Problem A is NP-complete if $A \in NP\text{-hard}$ & $A \in NP$.

NP-complete problems are "hardest problems in NP": if you can solve one, you can solve all NP problems

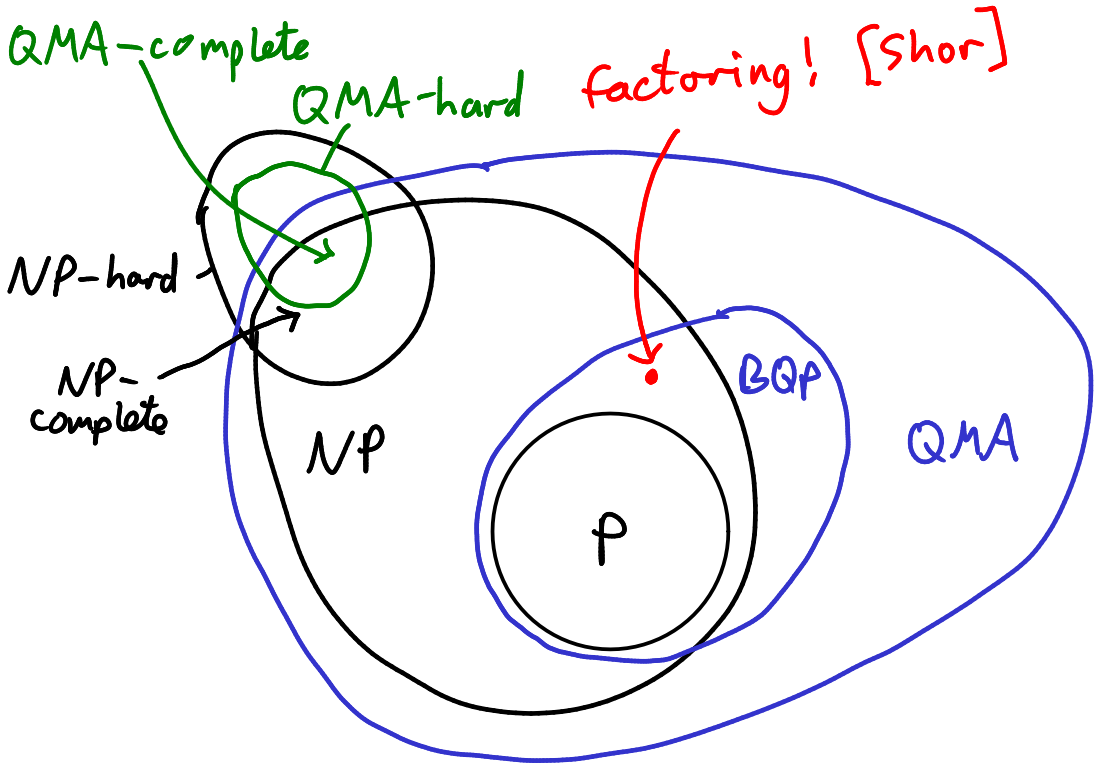
Define QMA-hard & QMA-complete analogously.

Complexity Zoo

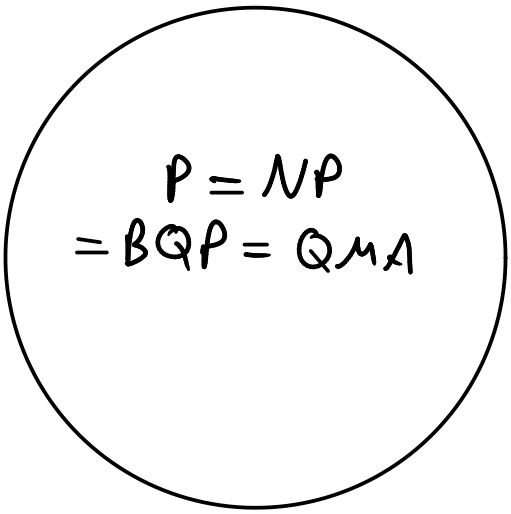
Relationship between P, NP, BQP, QMA is an open problem.

- $P \subseteq BQP$ (classical = special case of quantum)
  $NP \subseteq QMA$
- $P \stackrel{?}{=} NP$ infamous! $1,000,000
- $P \stackrel{?}{=} BQP$ "are quantum computers useful?"
- $BQP \stackrel{?}{=} QMA$ quantum "P vs. NP"
- $NP \stackrel{?}{=} QMA$ unlikely

General belief...



... but could be



!