CMPT 476 Lecture 13

Quantum computing

Most general measurement we care about
Projectors: such that they are orthogonal and they sum to the identity.

Up to this point we've looked in general at quantum mechanics, quantum information, and a few example protocols that we can implement with a few qubits. Now we're ready to shift gears and develop a general notion of quantum computation and see how it compares to classical computation.

First let's review computational complexity and the circuit model so that we'll be able to make a meaningful comparison.

(Computational complexity)

Our original motivation for looking at quantum computation was that **simulating quantum dynamics on a classical computer** was not "efficient." We can formally state what we mean by this using **order notation** and **complexity theory**.

We say an algorithm (e.g. a python program) with input x runs in time

$$O(f(n)), \quad f: \mathbb{N} \to \mathbb{R}$$

if there exists $n_0 \in \mathbb{N}$, $c \in \mathbb{R}$ such that whenever x has length $n \geq n_0$, the **runtime** of the algorithm is at most

$$c \cdot f(n)$$

We say that an algorithm **runs in polynomial time** if

$$f(n) \approx n^k, \quad k \in \mathbb{R}$$

and **exponential time** if

$$f(n) \approx k^n, \quad k \in \mathbb{R}$$

The key thing to note is that the runtime of an algorithm running in exponential time grows **much, much faster** than one running in polynomial time.

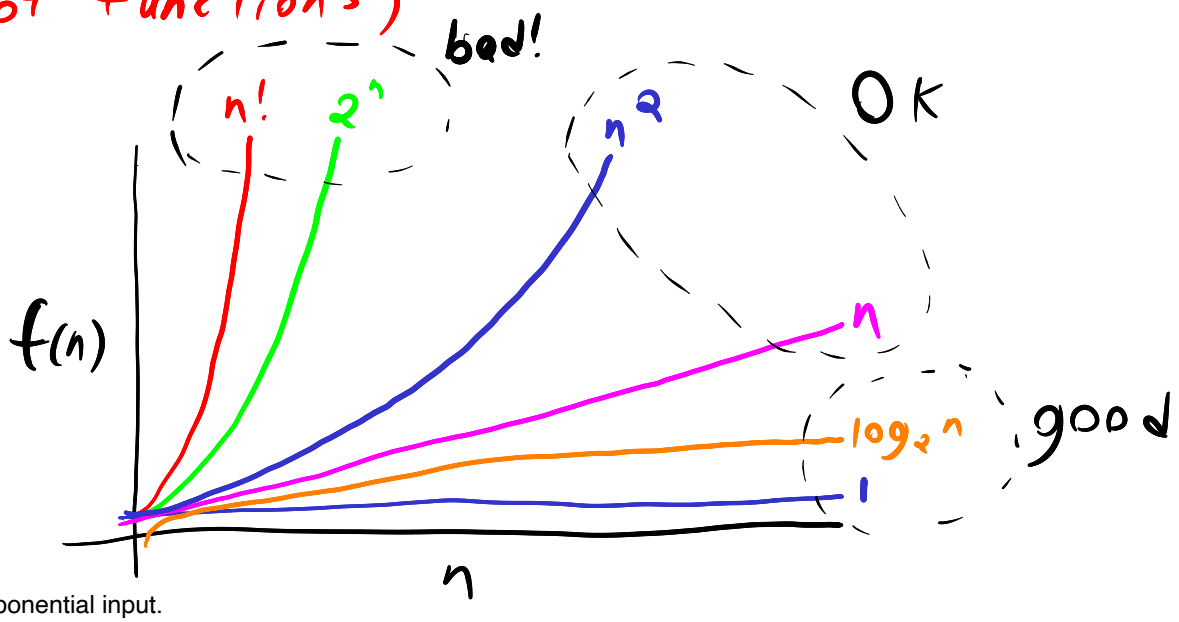
Ex.

Let algorithm A run in time $O(n^2)$ and algorithm B run in time $O(2^n)$. Suppose the input has length 100. Then, if the runtime is in seconds, Alg. A would take

$$100^2 \approx 3 \text{ hours.}$$

Algorithm B on the other hand would take 2^{100} seconds, which is **older than the age of the universe!**

(Growth of functions)



PSPACE: takes an exponential input.
Ask professor to clarify about example.
oop until we reach the logarithm, is it not logarithmic

(Complexity classes)

Tractable: polynomial time
Intractable: exponential time

In theoretical computer science, we classify problems (e.g. sorting an array) in terms of their complexity.

P = problems for which there exist a polynomial-time algorithm

NP = problems whose solutions can be checked in polynomial time

Text

WE can simulate quantum computation in PSPACE but not in polynomial time.

Best known algorithm is exponential time

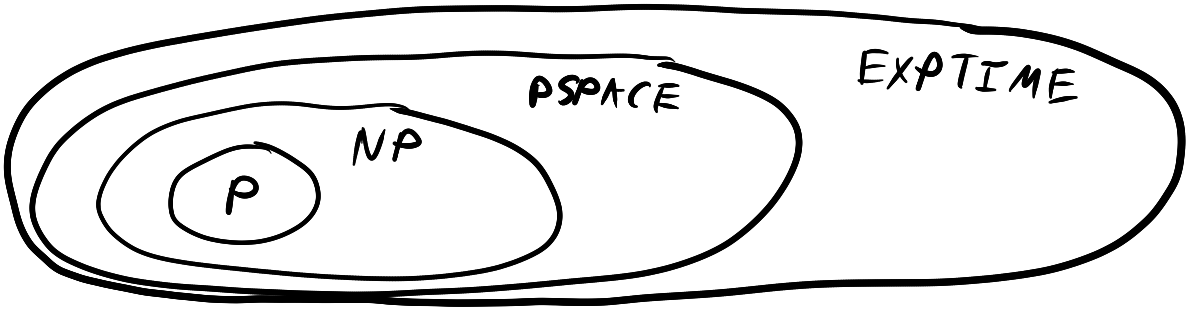
EXPTIME = problems w/ exponential algorithm

PSPACE = problems w/ polynomial space algorithm

Best known algorithm which can be solved in polynomial space

What is the difference between P and PSPACE

Generally we think of P as the class of tractable problems for computation. Clearly $P \subseteq NP$, but whether $P \stackrel{?}{=} NP$ is an open question, and widely believed to be false - that is, NP contains problems which are intractable.



Ex.

- Matrix multiplication is in P ($O(n^3)$ best known)
- Sorting is in P ($O(n \log n)$ best known)
- SAT (determing if a propositional formula is satisfiable) is in NP and is one of the hardest problems in NP.

NP complete:
It is in NP
It is as hard as it gets in NP
We can reduce every other problem in NP to SAT

- Factoring an integer into its prime factors is in NP, but not hard for NP. In particular, if it is in P as well, this does NOT imply $P = NP$

Not known whether it is hard or NP or if it is in P.

The fact that it is P does not

If we let our gates that allow infinitely complex numbers, then we have Turing degrees.
Turing degrees?
If we give our computer arbitrary complex numbers, we can reach Turing degrees.
We need to restrict the power of the computer, so we limit the complex numbers allowed.
Ask professor to clarify about this.

(Aside about polynomial time)

While n^{100} doesn't exactly scream tractable, we use polynomial time to mean tractable or physically realizable computation partially because it works in practice — usually n^k where $1 \leq n \leq 3$, and moreover because we can add and multiply polynomials most differences in computational models come down to polynomial factors, so most models of computation can efficiently simulate one another with respect to P.

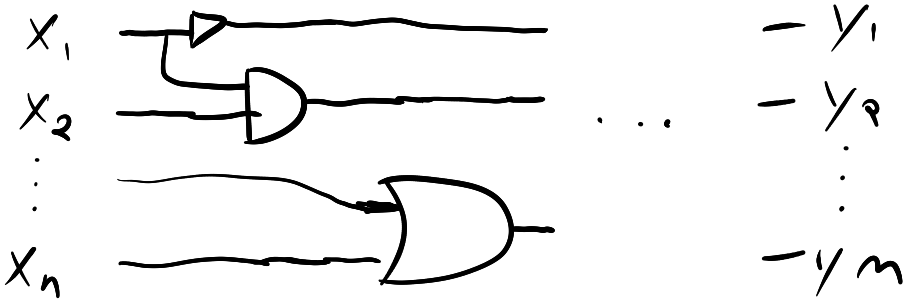
Model 1	Model 2
P <- -	- - -> P
Not P <- -	- - -> Not P

In quantum computation, we have different measures of computation. We have P and NP so we can transition between different complexity models.

So far we've said nothing formal about the model of computation (only that an algorithm is a python program). To compare classical and quantum computation, it will be easiest to work in the circuit model.

(Classical computation)

Recall from early on that we said in the classical circuit model, a computation is a circuit, e.g.



which computes a function $f: \{0,1\}^n \longrightarrow \{0,1\}^m$

Given a circuit C, we denote the number of gates in C by $|C|$.

(Uniform families of circuits)

Since a single circuit is fixed - i.e. it has a static and finite number of inputs and outputs, we can't talk about an algorithm or complexity. To do so we need a notion of a circuit family which implements an algorithm with varying input size.

Cannot define

Circuit families which are functions which take an input size and outputs

A uniform family of circuits is a family

A uniform circuit family is a family of circuits

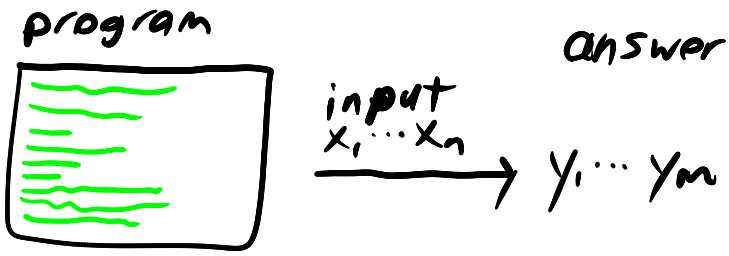
$$\{C_n \mid n \in \mathbb{N}\}$$

wires in classical circuit model carries bits.
Wires in quantum circuit model carries qubits.

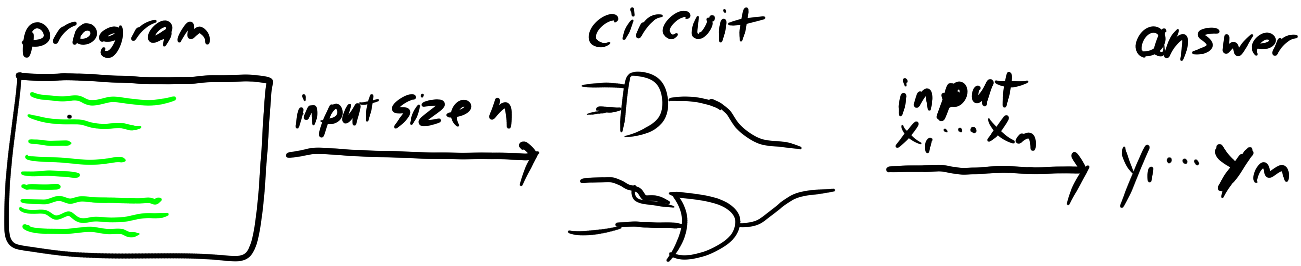
such that C_n has n input wires and C_n can be generated by an "efficient algorithm" e.g. a python program with runtime polynomial in $|C_n|$ and n

Polynomial time algorithm to generate the circuit.

Algorithm



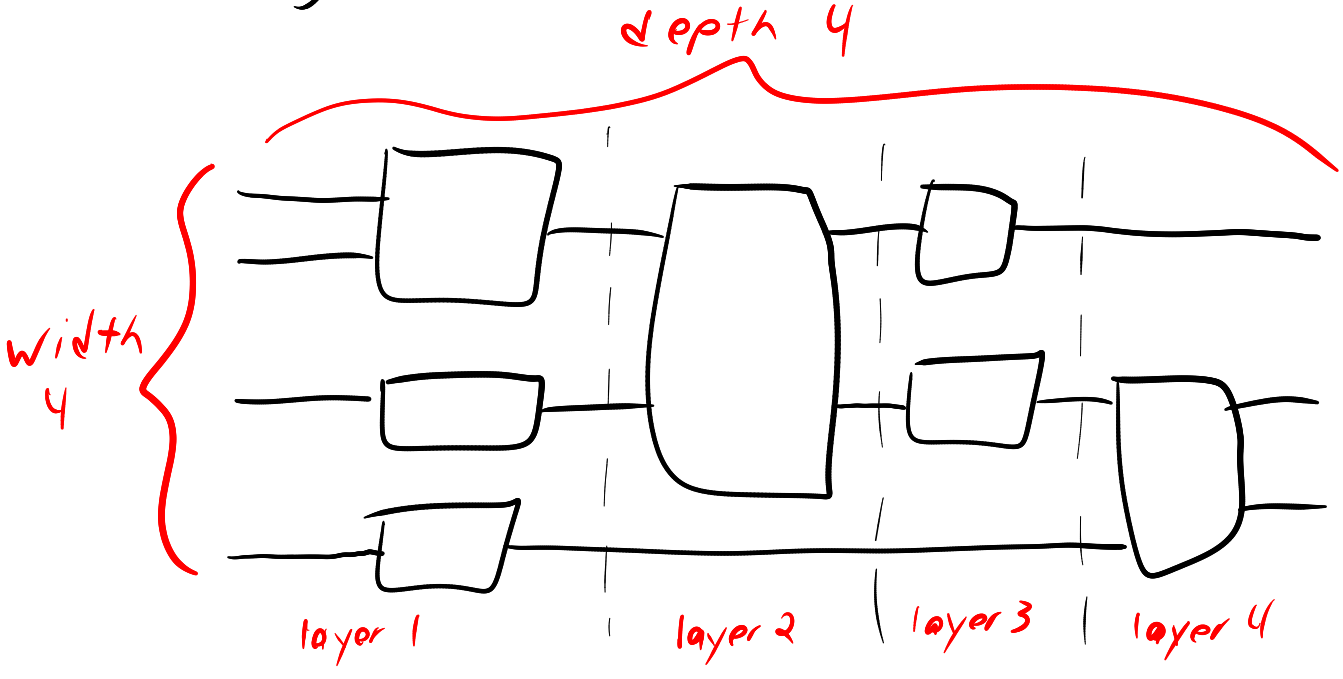
Uniform circuit family



The time to run an algorithm depends on the time to generate the circuit family and the time to run through the algorithm.
Ex: if we can generate a circuit family in polynomial time and the algorithm runs in log(n) time, the entire algorithm runs in polynomial time.

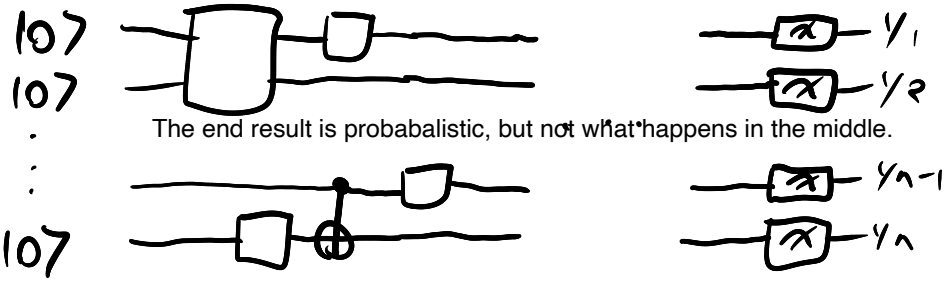
(Circuit complexity)

When we talk about complexity in the circuit model, we typically mean the **time** or **space** complexity of a uniform circuit family — that is, the growth rate of the circuit **depth** or **width**, respectively, as a function of n. Note that the depth — or the number of "layers" of gates, not the number of gates, gives the time complexity. Usually though, depth and # of gates are polynomially related.



(A quantum circuit model)

Recall that quantum circuits are obtained by replacing bits (e.g. wires) with qubits and classical gates by unitary gates (in particular the same number of inputs & outputs). We further restrict our attention to **deterministic** circuits, which only measure qubits **at the end of a computation.**



The end result is probabalistic, but not what happens in the middle.

Ask professor about ensembles in the middle of the computation.

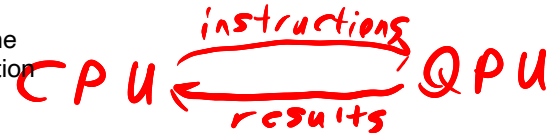
We restrict measurement to the **end of computation** because it turns out not to matter (we'll see this with the **principle of deferred measurement**) and the unitary model is more convenient.

(Other models of QC)

Many other models of quantum computation exist, both circuit-like and non-circuit-like. Here are a few:

- Knill's QRAM (quantum random access machine)

Measurement is in the center of the computation



Quantum computation is a mix of classical and quantum information

- Adiabatic QC (e.g. DWAVE)

DWAVE does a form of this in a restricted format. DWAVE's computer can not solve factoring in polynomial time.
Idea: change state until you obtain the result you want.

- Measurement-based QC

Change measurements on the next state based on the previous measurement.