CMSC 391 (Physical Computation) Notes

Rio Weil

This document was typeset on January 8, 2026

Introduction:
This is a set of lecture notes taken from UChicago's CMSC 391 (Physical Computation), taught by Bill Fefferman. Topics discussed include basic quantum mechanics, computational models, computational complexity theory, and quantum algorithms.

Contents

1 Overview and History

1.1 Class Overview and Schedule

Note: Lectures 1-4 are based on the provided powerpoint slides, as I was away for these lectures.

This course is a graduate course on the theory of quantum computation - the focus will be on results relevant to physics, with a particular focus on the power/limitations of near-term ("NISQ"-era) quantum experiments. The goal is to not be comprehensive, but sample some of the relevant results in the field.

The lectures will be based on self-contained slides, but for extra references one can consult "Quantum Computation and Quantum Information" by Nielsen and Chuang or "Classical and Quantum Computation" by Kitaev, Shen, and Vyalvi.

The rough agenda (amenable to modification) looks like:

- Lectures 1-4: Basics of Quantum computation, Quantum algorithms, BBBV lower bound on the query complexity of Grover's algorithm
- Lectures 3-6: Hamiltonian complexity and **QMA**. We will discuss the ultimate limits of quantum computation and the hardness of computing the ground state energy of a local Hamiltonian.
- Lectures 7-10: Complexity theoretic background for quantum supremacy experiments
- Lectures 11-14: Introduction to the theory of quantum advantage/supremacy experiments
- Lectures 15-16: Student presentations

1.2 The Truth about Quantum Computation

There has been a rising level of interest in quantum computation; huge media hype and industry interest from huge companies (Google, Microsoft, IBM etc.) and startups (IonQ, QuEra, etc.) alike.

Let us look past the hype. Quantum computing is a genuinely different model of quantum computation, which can use the principles of quantum mechanics to perform very particular problems exponentially faster than classical computers. It will not be a panacea; they will not be able to solve **NP**-complete problems (e.g. most optimization problems) efficiently. It will require us to rethink many ideas in classical computer science.

1.3 History

- 1980s: Benioff, Deutsch, Feynman propose quantum computation as a way to simulate quantum systems; to date the quantum simulation problem (e.g. simulate time evolution or estimate ground state energies) remains as one of the most useful potential applications.
- 1993: Vazirani and Bernstein formalize the quantum computing model and give an "oracle problem" which can be solved in 1 quantum query and requires $\Omega(n)$ classical queries.
- 1994: Simon (while trying to prove otherwise!) shows the first exponential quantum speedup ($O(n)$ quantum queries vs. $\Omega(2^{n/2})$ classical queries) for period finding. Both this and the above cases were contrived problems structured to find a quantum advantage.
- 1994: Shor adapts Simon's algorithm to solve integer factorization in poly($\log(N)$) steps! This is still one of the primary motivating algorithms for the field, and tells us that large fault-tolerant devices will break public key cryptography.
- 1995/1996: Grover realizes that quantum computers have a quadratic ($O(2^{n/2})$ quantum vs. $O(2^n)$ classical) advantage in the unstructured search problem. To date, most provable algorithms are based on Shor or Grover.

1.4 Why the interest?

As theoretical computer scientists, contemporary theory is based on the "Extended Church-Turing thesis", wherein every problem that can be solved efficiently by a feasible model of computation can be solved with at most polynomial slowdown on a classical Turin gmachine. This motivates the Turing machine as *the* model for understanding computation. But it can never be proven if true. Quantum computation is the only viable(?) model that seems to violate this thesis, putting the foundations of theoretical computer science into question. This comes with some skepticism - we have yet to see Shor implemented, and we don't know if factoring is provably classical hard (and indeed, there are skeptics that view Shor as an impossibility "proof" of quantum computation).

Much of the recent excitement comes from experimental advances leading into the Noisy Intermediate Scale Quantum/NISQ era, where several groups have implemented small, noisy, but programmable devices. The noise and space limitations are such that many quantum algorithms (e.g. Shor) cannot be run, but also are big enough that they cannot be naively simulated by classical computers!

As a comment, research in QC mirrors the early 90s - we have not yet discovered many useful speedups for near-term devices, and rather the focus is on demonstrating the first experimental violation of the Extended Church-Turing thesis. We have candidate problems that are within reach of experiment and have provable evidence of classical hardness - but at the moment these are contrived and reverse-engineered problems (i.e. not useful). We hope that (like the 90s), these results may inspire the foundations of useful quantum algorithms. Indeed, these initial "quantum supremacy" results will be one of the focuses of this course.

2 Basics Quantum Mechanics

2.1 Qubits

The simplest quantum system is the "qubit", or quantum bit. It is mathematically, a complex unit vector in the vector space $\mathbb{C}^2$. For our finite-dimensional setting, we can consider the Hilbert space as a complex vector space with an inner product. The states on the space look like:

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2.1}$$

where $|\alpha|^2 + |\beta|^2 = 1$ for $\alpha, \beta \in \mathbb{C}$. The qubit is thus the generalization of the classical $0, 1$ bit states of

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.2}$$

Notice these are orthogonal vectors. What Eq. (2.1) says is then that $|\psi\rangle$ is a superposition of $|0\rangle, |1\rangle$:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.3}$$

where α, β are amplitudes.

In Dirac's notation, $|\psi\rangle$ is a "ket", representing a column vector. Its Hermitian conjugate is the "bra", representing a row vector:

$$\langle\psi| = \begin{pmatrix} \alpha^* & \beta^* \end{pmatrix} = \alpha^*\langle 0| + \beta^*\langle 1| \tag{2.4}$$

The inner product between $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and $|\psi'\rangle = \alpha'\langle 0| + \beta'\langle 1|$ is given by:

$$\langle\psi|\psi'\rangle = (\alpha'^*\langle 0| + \beta'^*\langle 1|)(\alpha^*\langle 0| + \beta^*\langle 1|) = \alpha'^*\alpha + \beta'^*\beta \tag{2.5}$$

where we have used the orthonormality of the $|0\rangle, |1\rangle$ states.

2.2 Multi-qubit systems and Entanglement

A two-qubit quantum state is a unit vector in $\mathbb{C}^2 \otimes \mathbb{C}^2$, i.e.:

$$|\psi\rangle = \sum_{i,j=1}^{2} \alpha_{ij}|i\rangle \otimes |j\rangle \tag{2.6}$$

where $\sum_{ij}|\alpha_{ij}|^2 = 1$. Notice that in general, a two-qubit state $|\psi\rangle$ cannot be written as a tensor product/separable state of the form $|\phi\rangle \otimes |\gamma\rangle$, for one-qubit states $|\phi\rangle, |\gamma\rangle \in \mathbb{C}^2$. As an example of an entangled (not separable) state, we have the Bell/EPR pair:

$$|B_{00}\rangle = \frac{1}{\sqrt{2}}|0\rangle_A|0\rangle_B + \frac{1}{\sqrt{2}}|1\rangle_A|1\rangle_B \tag{2.7}$$

and as an example of a separable state we have:

$$\frac{1}{\sqrt{2}}(|0\rangle_A + |1\rangle_A) \otimes \frac{1}{\sqrt{2}}(|0\rangle_B + |1\rangle_B) = \frac{1}{2}\left(|0\rangle_A|0\rangle_B + |0\rangle_A|1\rangle_B + |1\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B\right) \tag{2.8}$$

More generally, the state of an n qubit system is a unit vector in $(\mathbb{C}^2)^{\otimes n} \cong \mathbb{C}^{2^n}$. Just as in the two-qubit case, we can divide the n qubits into two registers A/B of size n_1, n_2, where the state of register A is in the Hilbert space $\mathcal{H}_A = \mathbb{C}^{2^{n_1}}$ and the state of register B lives in the Hilert space $\mathcal{H}_B = \mathbb{C}^{2^{n_2}}$. The state of a bipartite system is a unit vector in $\mathcal{H}_A \otimes \mathcal{H}_B$ of dimension $2^{n_1+n_2}$ that may be entangled. There are generalizations to tripartite systems and beyond, though notions of entanglement are more subtle there.

3 Models of Computation

3.1 Classical deterministic computation

We consider a deterministic classical computer with $m = \text{poly}(n)$ bits of memory. The state of the system is an m-bit string, which we may equivalently think of as a 2^m-dimensional column vector indexed by all possible m-bit strings $y \in \{0,1\}^m$, denoted by $|y\rangle$. The computation proceeds as follows:

1. Start in the initial state with the memory set to $|x0^{m-n}\rangle$, with x the input.
2. Apply $\text{poly}(n)$ number of local gates, which act non-trivially on some constant number of input bits and modify a constant number of output bits. As an example, we can consider the XOR gate on input $(x_1, x_2, \ldots x_m)$ which takes two bits x_1, x_2 and outputs $(x_1, x_2, \ldots, x_k = (x_1 \oplus x_2), \ldots, x_m)$.
3. The measurement is then simply reading out the unique output string.

3.2 Randomized classical computation

We can generalize the above computation to a randomized computation to extending the notion of a state to be a probability distribution over m-bit strings $\{0,1\}^m$, or a 2^m-dimensional column vector of positive real numbers that sum to 1, wherein we label the 2^m basis vectors as $|a\rangle, a \in \{0,1\}^m$. Then the procedure goes as:

1. Start in the distribution in which the probability mass is on the input x as well as ancillary bits $|x0^{m-n}\rangle$.
2. As before, apply $\text{poly}(n)$ local gates - but these are now probabilistic gates. Supposing we have the probabilistic state $\mathbf{v} = \sum_{y\in\{0,1\}^m} \alpha_y|y\rangle$ with $\sum_y \alpha_y = 1$. We apply a linear map $F : \mathbb{R}^{2^m} \to \mathbb{R}^{2^m}$ such that:

- If $\mathbf{v}$ is a distribution vector (entries sum to 1) then so is $F(\mathbf{v})$, i.e. F as a $2^m \times 2^m$ is stochastic; it has nonnegative entries where the columns sum to 1.
- F is local, in that it reads and modifies at most 3 bits of the register, leaving the rest untouched. This means it can be described as a $2^3 \times 2^3$ matrix (acts as identity on all other registers).

3. Measurements of the state outputs a sample from the distribution over $\{0,1\}^m$ based on the state at the end of the circuit.

Example; suppose we are in the state:

$$\begin{pmatrix} 1/2 \\ 1/4 \\ 1/4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{2}|000\rangle + \frac{1}{4}|001\rangle + \frac{1}{4}|010\rangle \tag{3.1}$$

and we apply the two-bit CNOT gate:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

between the second/third bits. Then:

$$\begin{pmatrix} 1/2 \\ 1/4 \\ 1/4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \overset{\text{CNOT}}{\rightarrow} \begin{pmatrix} 1/2 \\ 1/4 \\ 0 \\ 1/4 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.2}$$

3.3 Quantum Computation

The state of a quantum computer with m qubits is an m-qubit quantum state $|\psi\rangle \in \mathbb{C}^{2^m}$.

- We start our quantum computation in the computationabl basis state $|x0^{m-n}\rangle$.
- We then apply a sequence of quantum gates G that acts on the state.
- We then perform a quantum measurement as a readout step.

The gates G must be complex unitary matrices, i.e. they preserve l_2-norm and map unit vectors to unit vectors - $\||G|\psi\rangle\|| = \||\,|\psi\rangle\|| = 1$. Equivalently, rows/columns of G are orthonormal. Further equivalently, $U^\dagger U = I$ (the conjugate transpose is the inverse). G must also be local, i.e. it acts nontrivially on some constant number of qubits and as identity on the rest, i.e. $G = B \otimes I_{n-3}$ where B is some $2^3 \times 2^3$ unitary.

An example is starting with:

$$|\psi\rangle = |00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix} \tag{3.3}$$

and then applying the 2-qubit Hadamard gate:

$$H^{\otimes 2} = H \otimes H = (\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}) \otimes (\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}) = \begin{pmatrix}1 & 1 & 1 & 1\\1 & -1 & 1 & -1\\1 & 1 & -1 & -1\\1 & -1 & -1 & 1\end{pmatrix} \tag{3.4}$$

where applying $H^{\otimes 2}$ to $|\psi\rangle$ yields the uniform superposition over n qubits:

$$H^{\otimes 2}|\psi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \tag{3.5}$$

this is the analogue of the uniform distribution and will be the starting point for many quantum algorithms!

3.4 Quantum Measurement

Let us define what we mean on the last step - how do we readout our answer from $|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x |x\rangle$ our final state? We choose an orthonormal basis for the vector space $E = \{|e_1\rangle, |e_2\rangle, \ldots, |e_{2^n}\rangle\}$. And then express $|\psi\rangle = \sum_i c_i|e_i\rangle$ in this basis, with $c_i = \langle e_i|\psi\rangle$. Then the Born rule/Dirac postulate tells us that:

- If we measure $|\psi\rangle$ w.r.t. basis E we obtain $|e_i\rangle$ with probability $|c_i|^2$.
- Furthermore, the measurement collapses the state into $|e_i\rangle$.

Note that if we measure again in the same basis, we obtain the same sate with certainty - measurement breaks the superposition!

A one-qubit example of measurement is starting with $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, doing a measurement in the $\{|0\rangle, |1\rangle\}$ basis yields either $|0\rangle$ or $|1\rangle$ with probability 1/2.

Another single-qubit is example is starting with $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\phi}|1\rangle)$. Measuring in the $\{|0\rangle, |1\rangle\}$ basis yields $|0\rangle$ with probability 1/2 and $|1\rangle$ with again probability $|\frac{e^{i\phi}}{\sqrt{2}}|^2 = \frac{1}{2}$. But, if we measure in the Hadamard basis of $\left\{|\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)\right\}$, we find something interesting; in this basis we can express $|\psi\rangle$ as:

$$|\psi\rangle = \frac{1+e^{i\theta}}{2}|+\rangle + \frac{1-e^{i\theta}}{2}|-\rangle \tag{3.6}$$

so we measure outcome $|+\rangle$ with probability $|\frac{1+e^{i\theta}}{2}|^2 = \cos^2(\frac{\theta}{2})$ and obtain $|-\rangle$ with probability $|\frac{1-e^{i\theta}}{2}|^2 = \sin^2(\frac{\theta}{2})$.

4 Partial Measurement and Interference

4.1 Partial Measurement

Let us consider quantum measurement again. Suppose we now have a bipartite system with composite Hilbert space $\mathcal{H}_A \otimes \mathcal{H}_B$. What happens if we only measure one subsystem? Supposing we have the bipartite state:

$$|\psi\rangle = \sum_{ij} c_{ij} |i\rangle_A |j\rangle_B = \sum_j (\sum_i c_{ij} |i\rangle_A) \otimes |j\rangle_B \tag{4.1}$$

if we measure system B in the standard basis, we obtain $|j\rangle_B$ with probability $\sum_i |c_{ij}|^2$, with the combined system collapsing to the state $(\sum_i c_{ij} |i\rangle_A) \otimes |j\rangle_B$ (up to normalization).

As an example, suppose we have $f : \{0,1\}^n \to \{0,1\}$ a Boolean function. Consider now the bipartite state $\frac{1}{\sqrt{2^n}} \sum_x |x\rangle_A |f(x)\rangle_B$.

If we measure register A (in the computational basis), we just have a uniform probability of measuring any bitstring $|x\rangle$, and then register B is left in the state $|f(x)\rangle$ for the measured x.

If we instead measure register B (in the computational basis), we obtain outcome $|0\rangle$ with probability proportional to the number of inputs x such that $f(x) = 0$ and the same for outcoem $|1\rangle$, i.e. $|0\rangle$ occurs with probbaility $\sum_{x \in f^{-1}(0)} \left(\frac{1}{\sqrt{2^n}}\right)^2 = \frac{|f^{-1}|(0)}{2^n}$ and $|1\rangle$ with $\frac{|f^{-1}|(0)}{2^n}$. For outcome $b \in \{0,1\}$ we have the post-measurement state:

$$\frac{1}{\sqrt{f^{-1}(b)}} \sum_{x \in f^{-1}(b)} |x\rangle |b\rangle \tag{4.2}$$

4.2 Interference

Quantum computers seem to be dramatically faster than their classical counterparts at certain tasks, but where does this power come from? The best answer appears to be a phenomenon known as quantum interference.

Consider the one-qubit state $|\psi\rangle = |0\rangle$ and let $U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$. If we apply U to $|\psi\rangle$, we have:

$$U|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle. \tag{4.3}$$

If we apply U again, we have:

$$U\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) = \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle - \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle = |1\rangle \tag{4.4}$$

so the $|0\rangle$ states cancel out! The outcome of a measurement in the computational basis would now be deterministic. This is one example of destructive interference.

The above example is simple, but tells us that the negative amplitudes are important! In a loose sense, different quantum computational paths cancel due to these relative phases. this is in contrast to classical computation, where randomized pathways always add - randomized algorithms are just deterministic algorithms with random seeds, i.e. a deterministic algorithm B with output probability:

$$\Pr_{r \in \{0,1\}^q}[B(x,r) = 1] = \sum_{r : B(x,r)=1} \Pr[r]. \tag{4.5}$$

This is a sum of all positive terms. Note that there is no way to generically express the output probability of a quantum algorithm in this form (we will return to this later!)

5 Computational Complexity Theory

5.1 Intro to Complexity Theory

Today, we start discussing computational complexity theory, departing from the first 4 lectures worth of discussion of quantum mechanics. As much as I (Bill) loves this subject, we won't be comprehensive here

- we will discuss the bare minimum such that things will make sense here. The language itself is not that complicated, but it does take some getting used to.

At a high level, it is the computational complexity theory is the study of the comparative difficulty of solving computational problems with the various amounts of resources. Traditionally, the most studied aspects are time complexity (how many gates?) and space complexity (how much resources do we need)? We also have query complexity (how many queries do we require?), entanglement resource etc. This is all to say that computational complexity can encompass multiple different resources.

Q: Are query and time complexity the same? A: Related, but not the same. Time can be drastically smaller than query complexity. It may be that we can use something about the structure of the problem (whitebox notion) and that we require a lot less time than queries. An interesting remark - it tends to be much easier to prove things in blackbox settings, and this is related to the difficulty of the $P \overset{?}{=} NP$ problem.

Definition: Decision Problem

Consider a decision language $L \subseteq \{0,1\}^*$. This represents the "yes" inputs to a computational problem whose answer is "Yes", or "No". The decision problem is: Given input $x \in \{0,1\}^n$, decide if $x \in L$.

Not all computational problems are decision problems. E.g. sampling problems are not (and Bill thinks NISQ demonstration of quantum advantage is not possible for decision problems! Noise makes decision problems trivial... but this is an open problem). Factoring looks like its not, but can be framed as a decision problem (more on this later).

Somewhat equivalent way of thinking about the problem - computability of Boolean functions. But this isn't quite fully general, because you can consider *promise languages*, which is not defined over all inputs.

Definition: Promise Languages

Promise languages are those where $L_{\text{yes}} \cup L_{\text{no}}$ (the union of the yes/no answers) is not hte full space $\{0,1\}^*$.

We can see that in the Boolean function case, it needs to be defined on all possible inputs, so it does not capture this notion of a promise language.

Our interest will be in the minimal amount of computational resources needed for an algorithm to decide specific languages L, as a function of the input length $|x|$.

We can consider worst case complexity (how hard is the worst x?) which is the usual notion, though there are others (such as average case complexity, where we consider a random x).

Definition: SAT

$$\text{SAT} = \left\{\psi : \{0,1\}^n \to \{0,1\}\text{, a Boolean function on } n \text{ variables } \exists y \text{ s.t. } \psi(y) = 1\right\}$$

The computational task we try to solve is to decide, given an inpu tof a Boolean formula ρ, is $\rho \in \text{SAT}$?

Recall a Boolean formula on n variables is a Boolean expression involving the variables, negation of the variables, and OR/AND of the two variables.

Sometimes, we restrict to the 3SAT problem (the first nontrivial SAT) in which the Boolean formula is to be a 3CNF, e.g. $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_4 \vee x_5 \vee \neg x_3) \ldots$. Restricting to a constant is important because then considerations of locality can enter.

5.2 Complexity Classes

Complexity classes are sets of problems/decision languages that are grouped together due to the existence of an algorithm that uses similar amounts of resources to solve the problems.

The most important complexity classes are **P** and **NP**.

Definition: P

P is "deterministic polynomial time", which is roughly the computational problems that can be solved in some poly($|x|$) amount of time (classical gates) by a deterministic turing machine.

A nontrivial example is the primality algorithm (AKS algorithm)/determining whether a number is prime or not (this was only proven in 2007!). Another is a the problem of determining the shortest path in a weighted graph (Dijkstra's algorithm). There are also trivial problems, e.g. given a string is the first bit 0 or 1.

Definition: NP

NP is "nondeterministic polynomial time", roughly the computational problems whose "yes" answers can be verified in poly(n) time.

Example: SAT, Factoring. How do we frame factoring as a decision problem? Textbook answer: Given a tuple (n, m), is there a nontrivial factor of n that is less than m? E.g. $(15, 14)$ is yes as $15 = 5 \cdot 3$ both of which are less than 14.

Why is this equivalently powerful? We can do binary search by looking at $(n, n/2^1), (n, n/2^2), (n, n/2^3)$. We only have to do this a logarithmic number of times, which is efficient. The other way is even clearer - if we can prime factorize in the gradeschool sense, then it is very easy to solve this (n, m) decision problems (just check m against the smallest nontrivial prime factor of n).

Definition: coNP

coNP is "nondeterministic polynomial time", roughly the computational problems whose "no" answers can be verified in poly(n) time.

Example: Version of SAT where we ask "does there exist a Boolean function which is *unsatisfiable*?"

Interesting feature - factoring is in both **NP** and also **coNP**. Why is factoring in **NP**? If I send you (n, m) and send you the list of the prime factorization of n (first I would check that all the factors are prime (primality, in **P**) and then multiply them together to make sure that indeed it is in n), then it is easy to check if the instance is "yes". With the identical setup, then it is easy to check if the instance is "no". The same witness serves both the yes and no answers! Why is this important? We strongly believe that **NP**-complete problems cannot be in **coNP**, because if this were true then **coNP** $=$ **NP** but this would result in the collapse of the polynomial hierarchy. Something that is often got very wrong in the media - quantum computers are thought to *not* to be able to solve NP-complete problems, and factoring is not (thought to be) one (again, if factoring is **NP**-hard then the polynomial hierarchy collapses).

P vs. **NP** is the main problem in computer science. **P** $\subseteq$ **NP** is clear because for a **P** problem we can just solve the problem efficiently. The 1-million dollar question is **NP** $\subseteq$ **P**. In order to prove equality, it suffices to give an efficient algorithm to so-called any **NP**-complete problem (e.g. 3-SAT). There is mounting evidence that this is not possible, and so it is thought that **P** $\neq$ **NP**, but showing this is very hard. Proving two classes are equal is just producing an algorithm - proving two classes are unequal is showing that there exists no possible algorithm.

6 Computational Complexity Theory II

Remark: **NP** does not mean non-polynomial, it just means polynomial with respect to a new resource (i.e. already having the solution).

Also, **NP**-complete means both in **NP** and is being **NP**-hard (it is possible for a problem to be **NP**-hard, but not in **NP**, i.e. its harder)

6.1 Randomized Computation

P is a completely deterministic class. But what if we give the algorithm the power to toss random coins, i.e. the output of the algorithm can depend on the input + a random sequence of coin tosses? Then we arrive at **BPP** (bounded-error, probabilistic, polynomial time), the class of decision problems that can be decided in poly(n) time with bounded probability over random coin tosses.

What do we mean about probability? There are two directions:

- Completeness probability: If the answer is "yes", what is the probability we say yes?
- Soundness probability: If the answer is "no", what is the probability we say yes?

Bounded probability means that the errors we make in both cases are bounded away from a half. I.e. completeness probability is greater than 2/3 (the actual number is not so important, as long as its a constant away from no) and the soundness probability is less than or equal than 1/3 (i.e. teh probability that we accept "no"s as a yes).

Why do we want these bounded away from 1/2? Then we can amplify the success probabilities by repeating the algorithm + taking a majority vote. If the gap (the difference between the completeness and soundness probabilities) is tiny, then this amplification strategy fails, but if the gap is large then this is a successful strategy.

It's clear that $\mathbf{P} \subseteq \mathbf{BPP}$. After more than 3 decades of research, we strongly believe that the other inclusion $\mathbf{BPP} \subseteq \mathbf{P}$ is also true, but an unconditional proof remains elusive.

6.2 Quantum Computation

Finally to **BQP** (bounded error, quantum, polynomial). This is the dream/ideal quantum computation. Let us formalize this precisely so we can talk about the computational power of this class.

Definition: BQP (Informal)

The class of decision problems that can be solved with high probability by applying a quantum circuit with poly(n) gates, measuring the first qubit, and accepting iff the outcome is 1.

To formalize this, let's note that if we apply the circuit U to the initial state, and get $|\psi\rangle = U|x\rangle|0^{q(n)}\rangle$ and measure the first qubit in the standard basis we have:

- Pr[Output 1] = $\langle\psi|\Pi_1|\psi\rangle$
- Where $\Pi_1 = |1\rangle\langle 1| \otimes \mathbb{I}$ a projector (projector: idempotent, equivalently eigenvalues are 1/0) onto the first qubit. In this case, a 2x2 matrix which kills the $|0\rangle$ eigenspace on the first qubit.

I.e. it is the set of decision languages $L \subseteq \{0,1\}^*$ so that there exists some polynomial size family of quantum circuits $\{Q_n\}$ so that after measuring the first qubit:

- Letting $|\psi\rangle = Q_n|x\rangle|0^{q(n)}\rangle$
- $x \in L \implies \text{Pr[Output 1]} \geq \frac{2}{3}$

- $x \notin L \implies \Pr[\text{Output } 1] \leq \frac{1}{3}$

Remark: The probability here is *not* coming from some probability distribution over classical coin flips. If it were true that we could reproduce quantum randomness with classical randomness, it turns out the polynomial hierarchy collapses!

Some issues/oddities with **BQP**:

- Noise/Decoherence. A polynomial size circuit results in a fidelity of $\sim 2^{-m} = 2^{-\text{poly}(n)}$ where m is the number of gates.
- Locality/connectivity. Computer science locality is k-locality (bounded degree of interaction) but experimental locality is geometric/spatial. Note that however this is self-consistent within the definition - we allow for polynomial number of operations, and hence we can always use a polynomial number of swap operations to recover the spatial locality from k-locality. This would no longer be true if we enforced the notion of constant depth quantum circuits, for example.
- Number of measurement appears to be just 1... but indeed we can use the principle of deferred measurement here. Instead of measuring in the middle of the computation, we can associate one measurement per ancilla (apply a CNOT between a physical qubit and the ancilla), and then defer the measurement.
- The polynomial depth is notable here. Different depths gives you different amount of power. Shor for example can be done in log depth, but a classical computer would call the log depth circuit multiple times.
- This is an asymptotic class: n here is growing. But for some finite number, it doesn't really make sense to ask whether a single/fixed datapoint of number of gates as a function of n is polynomial or exponential... One of the most common NISQ era mistakes. What does it mean to claim an "exponential speedup" for $n = 53$ qubits? Doesn't make sense! Such claims can be made only asymptotically/scaling an algorithm.
- Gatesets: We have a fixed number of possible gates that we can perform in experiment. Not a problem though, because we have a universal gateset (dense subgroup of $SU(2^n)$). Then, Solovay-Kitaev tells us that we can simulate a different gateset with $\text{polylog}(1/\epsilon)$ overhead. This tells us that in the context of **BQP** we don't have to worry! With polynomial overhead we are totally able to simulate any other gateset with exponential precision. But pragmatically, note we do care about numbers, and hence a lot of work goes into compilation and optimization.
- Decision problems: Many problems that we may want to solve on a quantum computer may not be expressable as a decision problem! (e.g. there exists no known decision problem analog of the sampling problem).

Next week, we'll start discussing how **BQP** fits into the landscape of complexity classes we've introduced.

7 Computational Complexity Theory III

Size of the gap (between completeness and soundness) can be $1/\text{poly}(n)$ for BQP (as we can repeat the algorithm $\text{poly}(n)$ times to amplify the success probability).

7.1 P $\subseteq$ BQP

This proof tells us that quantum computing at least subsumes classical computation. This maybe seems a bit obvious, but where does the subtlety come in? The problem is that quantum computations are reversible due to the unitarity of the gates, i.e. U (a quantum circuit) is unitary with $UU^\dagger = U^\dagger U = I \implies U^\dagger = U^{-1}$. This of not the case for arbitrary classical gates, e.g. consider the AND gate, where AND(0, 1) = AND(1, 0) = AND(0. 0) = 0 (information is lost).

We will resolve this by showing that classical computations can always be made reversible. There are many different proofs of this fact, ours will use the Fredkin, our controlled-SWAP gate.

Definition: Fredkin Gate

The Fredkin is the classical gate which takes inputs (a, b, c) and outputs (a, b, c) if $a = 0$ and (a, c, b) if $a = 1$.

The Fredkin gate is controlled because the first bit determins what to do. Note that it is self-inverse.

Classically, $\{AND, NOT\}$ is universal. Simulating AND using Fredkin is done by AND(a, b) = third register of Fredkin(a, b, 0). I.e. the third register is 1 iff a, b are 1. Similarly, simulating NOt can be done by observing that NOT(a) = third register of Fredkin(a, 0, 1).

Note that this reversibility did not come to free - we had to add 1/2 bits for each AND/NOT. This is not a problem because we allow ourselves access to a polynomial number of ancillas.

7.2 Classical Functions in Quantum Superposition, Phase Kickback

Corollary: Classically computing functions in quantum superpositions. Suppose we have $f : \{0,1\}^n \to \{0,1\}$ that is computable by a deterministic classical algorithm in poly(n) time. We'll often need to compute this in quantum superposition, i.e. prepare $\frac{1}{\sqrt{2^n}} \sum_x |x\rangle|f(x)\rangle$.

You might want to ask why can't we just prepare $|f(x)\rangle$ - the issue is that $x \to f(x)$ is not reversible and not natively implementable. But, we can implement:

$$U_f : |x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle \tag{7.1}$$

with a poly(n) size quantum circuit. $\oplus$ is XOR. We implement it in this fashion so that this is a reversible operation.

Now, to prepare the desired state, we first create the uniform superposition on the n-qubit register by applying $H^{\otimes n}$, then by initializing the $|y\rangle$ register in the $|0\rangle$ register, we can do:

$$|0\rangle^{\otimes n}|0\rangle \overset{H^{\otimes n} \otimes I}{\to} \frac{1}{\sqrt{2^n}} \sum_x |x\rangle|0\rangle \overset{U_f}{\to} \frac{1}{\sqrt{2^n}} |x\rangle|f(x)\rangle \tag{7.2}$$

Sometimes we are interested in preparing $\frac{1}{\sqrt{2^n}} \sum_x (-1)^{f(x)}|x\rangle$. Preparing this is easy:

- First prepare $\frac{1}{\sqrt{2^n}} \sum_x |x\rangle|f(x)\rangle$.
- Then, apply a phase-flip unitary/Z-gate to the second register $U = Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$. This gives $\frac{1}{\sqrt{2^n}} \sum_x (-1)^{f(x)}|x\rangle|f(x)\rangle$
- Apply U_f again, which yields $\frac{1}{\sqrt{2^n}} \sum_x (-1)^{f(x)}|x\rangle|f(x) \oplus f(x)\rangle = \frac{1}{\sqrt{2^n}} \sum_x (-1)^{f(x)}|x\rangle|0\rangle$. Now the second register is unentangled from the first register, and can be removed.

The above procedure is suboptimal in the sense that it requires 2 queries to f/two uses of U_f. We can actually be more clever and do it with one query:

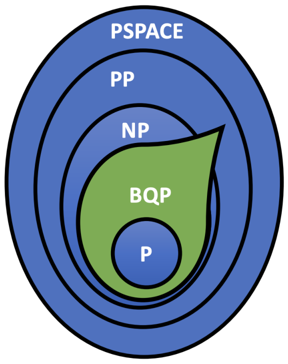
- Initialize the second register in $|1\rangle$ and apply a Hadamard to both registers, i.e. prepare $\frac{1}{\sqrt{2^n}} \sum_x |x\rangle \otimes |-\rangle = \frac{1}{\sqrt{2^n}} \sum_x |x\rangle \otimes (\frac{|0\rangle - |1\rangle}{\sqrt{2}})$
- Now, apply U_f, which yields $\frac{1}{\sqrt{2^n}} \sum_x |x\rangle \otimes (\frac{|f(x)\rangle - |1+f(x)\rangle}{\sqrt{2}})$. If $f(x) = 0$ then the second register is $|-\rangle$ and if $f(x) = 1$ then the state is $-|-\rangle$. So we are in the overall state $\frac{1}{\sqrt{2^n}} \sum_x (-1)^{f(x)} |x\rangle \otimes |-\rangle$. The desired state!
- Optionally, measure the last qubit in the Hadamard basis and discard the last register.

7.3 The Complexity Hierarchy and BQP

So, we've shown that **P** $\subseteq$ **BQP**. But why shouldn't **BQP** $\subseteq$ **P** or **BQP** $\subseteq$ **NP** Or should **NP** $\subseteq$ **BQP**? We have no idea/no proofs, but we have strong reason to believe that the answer to all questions is "no".

The best known upper bound (roughly speaking) can simulate any problem in **BQP** in unbounded error randomized polynomial time (called **PP**). This class is so powerful because its a class that involves almost random guessing.

Bill's personal belief of the hierarchy:



We haven't seen **PSPACE** before - this is the class of decision problems where we have space-limited memory but exponential amount of time. It's clear to see that SAT is in **PSPACE** because we can just try every possible solution, erasing the memory each time.

Upper bound 1: **BQP** $\subseteq$ **PSPACE**

Main idea: Feynman path integral - write acceptance probability of a quantum circuit as a sum with exponentially many terms, each of which can be computed in polynomial space.

Suppose we have a quantum circuit $Q = Q_m Q_{m-1} \dots Q_1$ on $n + q(n)$ qubits. We can write the acceptance probability of a quantum circuit Q as:

$$\Pr[Q \text{ outputs } 1] = \Pr[\langle\psi|\Pi_1|\psi\rangle] = \langle x| \otimes \langle 0^{q(n)}|Q_1^* \dots Q_m^* \Pi_1 Q_m \dots Q_1|0^{q(n)}\rangle|x\rangle \tag{7.3}$$

Now consider the following:

- Insert I between each pair of operators
- Note that for the n qubit identity, $I = \sum_x |x\rangle\langle x|$
- We can then write:

$$\begin{aligned}\Pr[Q \text{ outputs } 1] &= \langle x| \otimes \langle 0^{q(n)}|Q_1^* I Q_2^* \dots I Q_m^* \Pi_1 Q_m I Q_{m-1} \dots I Q_1 |0^{q(n)}\rangle |x\rangle \\ &= \sum_{x_1, x_2, \dots x_{2m+2}} (\langle x| \otimes \langle 0^{q(n)}|)|x_1\rangle \langle x_1|Q_m^*|x_2\rangle \langle x_2| \dots Q_1|x_{2m+1}\rangle \langle x_{2m+2}|(|x\rangle \otimes |0\rangle^{q(n)})\end{aligned}$$

So we have an exponential sum. But each of the matrix elements appearing in the sum above is easy to compute! We know what the gates are! So each term in the sum can be computed in polynomial space. Then by erasing the register each time and then doing the exponential sum, we conclude that **BQP** $\subseteq$ **PSPACE**.

8 Computational Complexity IV, Quantum Algorithms I

Last class, we talked about **BQP**, and showed that **P** $\subseteq$ **BQP** - this was a priori not obvious because there are classical gates that are not reversible, but indeed we came up with a construction that allows for all classical computations to be reversible through the use of (polynomially many) ancillas. We also showed that **BQP** $\subseteq$ **PSPACE** via a Feynman path integral.

8.1 PP

Note that the tighter inclusion is **BQP** $\subseteq$ **PP**. We will show this later when we discuss **QMA**. But let's just give some intuition for why **PP** is powerful. **PP** is **BPP** with the "bounded error" part removed. In other words, the $\frac{1}{2} \pm \epsilon$ completeness/soundness probability can now have $\epsilon = \frac{1}{\exp(n)}$ (rather than $\frac{1}{\text{poly}(n)}$), so the gap can be inverse exponential, which in some sense allows us to answer questions just via near-guessing.

We want a **PP** algorithm for SAT - it suffices to have an algorithm that if "yes" succeeds with $\frac{1}{2^n}$ and if "no" succeeds with probability 0. If the answer is "yes", then there is at least 1 satisfying assignment. If the answer is "no", then there are no satisfying assignment. So, pick a random bitstring x, plug it into the formula. If it is satisfied, answer "yes", if it is not satisfied, then answer "no". Actually, to ensure that we are centered around 1/2, if $\phi(x) = 1$ then we accept, otherwise we accept with probability $\frac{1}{2} - \frac{1}{2^{n+1}}$ and reject with probability $\frac{1}{2} + \frac{1}{2^{n+1}}$. This is very simple, but is nevertheless a **PP** algorithm for solving SAT (and most **PP** algorithms tend to be sort of "stupid" in this way - exponential gaps give us a lot of leeway). This shows that **NP** $\subseteq$ **PP**.

8.2 Discrete Fourier Transform

We go through this section quickly, because you should have seen it already. First, let's review the discrete Fourier transform (note: we will always be thinking about discrete FTs in this course). Suppose M is an integer and $f : \mathbb{Z}_M \to \mathbb{C}$ is a complex valued function, so its Fourier transform $\hat{f} : \mathbb{Z}_M \to \mathbb{C}$ is given by:

$$\hat{f}(t) = \frac{1}{\sqrt{M}} \sum_{x \in \mathbb{Z}_M} f(x) \omega^{xt} \tag{8.1}$$

where $\omega = \frac{2\pi i}{M}$ is a primitive M-th root of unity. If we write the function **f** as a vector:

$$\mathbf{f} = \begin{pmatrix} f(0) \\ f(1) \\ \vdots \\ f(M-1) \end{pmatrix} \in \mathbb{C}^M \tag{8.2}$$

and we also do the same for the Fourier transform $\hat{f}$, then these vectors are related by a change of basis $\hat{\mathbf{f}} = F_M \mathbf{f}$ where the matrix F_M has the form:

$$F_M = \frac{1}{\sqrt{M}} \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{M-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(M-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{M-1} & \omega^{(M-1)^2} & \omega^{(M-1)^3} & \cdots & \omega^{(M-1)^{M-1}} \end{pmatrix} \tag{8.3}$$

Straightforward multiplication of $\mathbf{f}$ by F_M would take M^2 operations by naive matrix multiplication.

Note that by exploiting the symmetry of F_M it is possible to recursively perform the transform in $O(M \log M)$, this is the fast fourier transform (FFT). Specifically, we can use Divide and conquer to recursively break down a DFT into many smaller DFTs:

$$\begin{aligned} \hat{f}(j) &= \sum_{i=0}^{M-1} \omega^{ij} f(i) \\ &= \sum_{i \text{ even}} \omega^{ij} f(i) + \sum_{i\text{odd}} \omega^{ij} f(i) \quad \text{(splitting into odd/even terms)} \\ &= \sum_{i'=0}^{M/2-1} (\omega^{2i'})^j f(2i') + \omega^j \sum_{i'=0}^{M/2-1} (\omega^{2i'})^j f(2i'+1) \\ &= (F_{M/2} \vec{f_{\text{even}}})(j) + \omega^j (F_{M/2} \vec{f_{\text{odd}}})(j) \end{aligned}$$

8.3 Quantum Fourier Transform

An important property of F_M is that it is unitary. Let's go with the simple geometric proof; the inner product rows of $i \neq j$ yields:

$$\frac{1}{M} \sum_{k=0}^{M-1} \omega^{ik} \omega^{-jk} = \frac{1}{M} \sum_{k=0}^{M-1} \omega^{(i-j)k} = \delta_{ij} \tag{8.4}$$

The last equality is a bit nontrivial - how do we see this? If $i \neq j$ then ω^{i-j} is an Nth root of unity, let's call it ω_1. Geometrically, the sum over this root of unity has center of mass at the origin. Analytically, we can see via the geometric series formula.

$$\frac{1}{M} \sum_{k=0}^{M-1} \omega^{(i-j)k} = \frac{1}{M} \sum_{k=0}^{M-1} \omega_k^k = \frac{1 - \omega_1^M}{1 - \omega_1} = 0 \tag{8.5}$$

Ok, so we've shown that F_M is a unitary. But how do we implement it in terms of gates? Using similar ideas to the FFt, we can decompose F_M into a quantum circuit of size $O(\log^2 M) = O(n^2)$ (with $M = 2^n$) which is polynomial.

This itself is insufficient for showing a quantum speedup - there's a bit of sleight of hand here, and it has to do with the fact that the DFT computed classically and the quantum computer computing the QFT are different. Let's think about the QFT - we start with $|0\rangle^{\otimes n}$, then apply $H^{\otimes n}$ to get the uniform superposition state, then apply the QFT, now we have a pure quantum state whose amplitudes are Fourier coefficients. But when we measure, we get a bitstring out, *not* the Fourier coefficient (unlike the classical case) - in other words the QFT allows us to do Fourier sampling - sample from the distribution with weight determined by the Fourier coefficients. Actually getting the distribution requires then $O(n2^n)$ time... so its not clear we have a speedup. Actually, this is often misconstrued. For example the HHL algorithm prepares a quantum state which corresponds to the solution to a system of linear equations... you can't directly read out the solution from this!

Note that if a function has a coefficient that is very large (e.g. inverse poly) then by repeat sampling (polynomial times) we can approximate the answer, but generically most Boolean functions are "flat" and so we require an exponential number of samples.

The upshot - the QFT is a good *subroutine*, but in itself does not yield a useful speedup. We will see it arise in Shor's algorithm, soon.

8.4 Phase Estimation

This construction is from Kitaev in 1995. He turns out to rediscover Shor's algorithm 6 months later. Apparently, he was aware of Shor insofar as he was told about it at a conference, but allegedly in Moscow he did not have internet access to Shor's paper... so he decides to reprove it for himself.

The algorithm turns out to be similar, but different. The quantum circuit is the same, but the analysis is completely different. In this class we'll look at Kitaev's analysis, which (beyond him being Bill's supervisor...) is simpler and introduces a phase estimation subroutine, which gives rise to a wide class of quantum algorithms. The fact that Shor can be viewed as a special case of phase estimation gives intuition for why Shor's algorithm works!

9 Quantum Algorithms II

Factoring arguably the most impressive result in quantum computing, from a conceptual (and pragmatic!) point of view. At the same time, the way that physicists present this often overcomplicates the algorithm. The goal for today is to present this as clearly as possible. We will first describe phase estimation, and then discuss factoring as a special case.

9.1 Phase Estimation

Let U be an $N \times N$ unitary matrix, then it has orthonormal eigenvectors which we will call $|\psi_1\rangle, |\psi_2\rangle, \ldots |\psi_N\rangle$, each with eigenvalues $\lambda_j = e^{2\pi i \theta_j}$ where $\theta_j \geq 0$ is called the phase. The proof is left as an exercise.

Proof. Since $UU^\dagger = U^\dagger U (= I)$, U is normal and hence has a spectral decomposition and can be diagonalized as $U = \sum_i \lambda_j |j\rangle\langle j|$ for an orthnormal basis $\{|i\rangle\}$ (for the proof of the spectral decomposition, see, e.g., Box 2.2 of Nielsen and Chuang). Now, by unitarity, we have that:

$$UU^\dagger = I \implies \sum_j |\lambda_j|^2 |j\rangle\langle j| = \sum_j 1|j\rangle\langle j| \tag{9.1}$$

which implies that all $\lambda_j = e^{i\theta_j}$ for some real θ_j. $\square$

The phase estimation problem is as follows; as input, we have a quantum circuit for implementing a $2^n \times 2^n$ unitary U, and an eigenvector $|\psi_j\rangle$, and as output we want an approximation to the phase θ_j.

The classical complexity of this problem is a bit subtle. The issue is not in describing U (has a polynomial description of gates), but in describing the eigenvector $|\psi_j\rangle$ - this is described by 2^n numbers - but there is no succint description of this eigenvector (if we only have access to a classical computer). So for this problem we can't directly compare the quantum and classical cases.

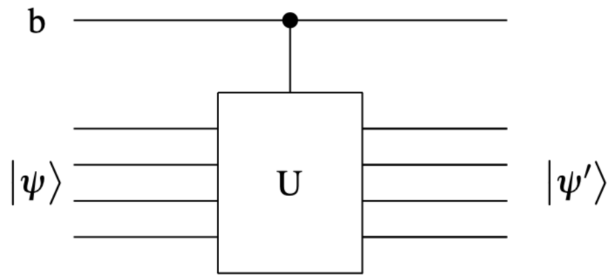
9.2 Controlled-Quantum Circuits

Given a quantum circuit U on n qubits, we call the following circuit of the controlled-U, acting on $n+1$ qubits:

$$C(U)|0\rangle|\psi\rangle \to |0\rangle|\psi\rangle \tag{9.2}$$

$$C(U)|1\rangle|\psi\rangle \to |1\rangle U|\psi\rangle \tag{9.3}$$

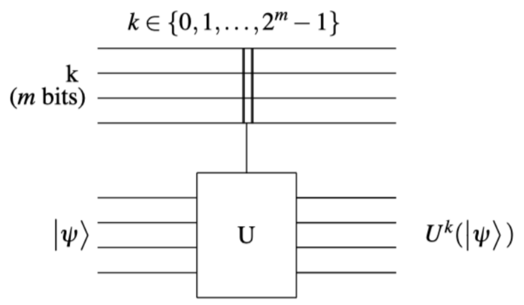
In circuit form, we write it as the following:



We can generalize this to a k-controlled quantum circuit. This is the circuit that takes m ancilla bits and uses this to implement a power of the unitary:

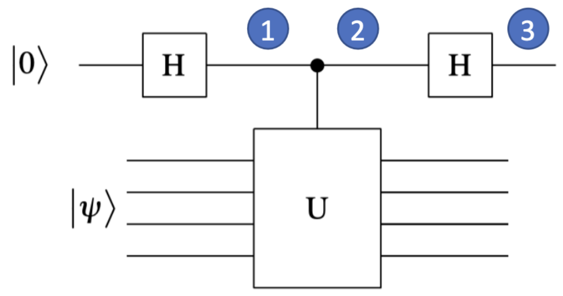
$$C_k(U)|k\rangle|\psi\rangle \to |k\rangle U^k|\psi\rangle \tag{9.4}$$

Subtlety - although U^k is unitary, we cannot necessarily efficiently implement it given an efficient representation of U - i.e. k can be exponential in the number of ancilla bits n, and U^{2^n} is of course not guaranteed to be efficient!



9.3 Poor Man's Phase Estimation

Now that we can control our circuits, we can solve the phase estimation problem (with limited precision). Say we are given U and eigenstate $|\psi\rangle$ with eigenvalue λ. Consider the following quantum circuit:



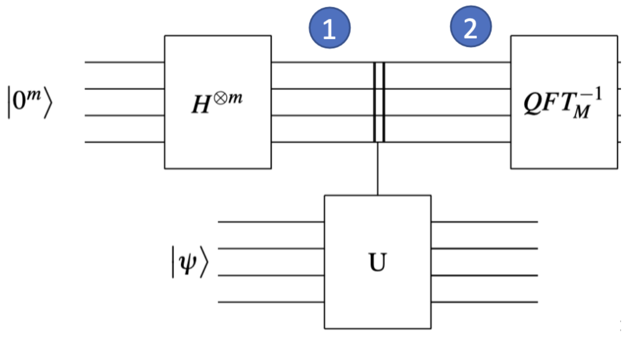
The action of this circuit is:

$$
\begin{aligned}
|0\rangle|\psi\rangle &\overset{H_1}{\rightarrow} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|\psi\rangle \\
&\overset{C(U)}{\rightarrow} \frac{1}{\sqrt{2}}|0\rangle|\psi\rangle + \frac{1}{\sqrt{2}}|1\rangle U|\psi\rangle = \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{\lambda}{\sqrt{2}}|\sqrt{2}\rangle|1\rangle\right)|\psi\rangle \\
&\overset{H_1}{\rightarrow} \frac{1+\lambda}{2}|0\rangle + \frac{1-\lambda}{2}|1\rangle = \frac{1+e^{i\theta}}{2}|0\rangle + \frac{1-e^{i\theta}}{2}|1\rangle
\end{aligned}
$$

the probability of seeing $|0\rangle$ is $\frac{1+\cos(2\pi\theta)}{2}$ and the probability of $|1\rangle$ is $\frac{1-\cos(2\pi\theta)}{2}$ so there's a bias that depends on θ. We can estimate the bias $\cos(2\pi\theta)$ by repeating the circuit many times. By Chernoff/Hofting bound (which is tight), to estimate the bias of a coin to within ϵ with probability at least $1-\delta$, we require $\frac{\log(\frac{1}{\delta})}{\epsilon^2}$ samples. Suppose then we want to estimate θ within m bits of accuracy, i.e. $\epsilon \sim \pm 2^{-m}$. This presents a problem because for this level of accuracy we require $\exp(m)$ number of samples. So, this strategy works well if we want a course estimate of the phase, but not for high precision.

9.4 Better Phase Estimation

In fact with m ancillas we can output θ to within m bits of accuracy. Let $M = 2^m$ and let $\lambda = e^{2\pi i\theta}$ and $\theta = \frac{j}{2^m}$ (note - this is a simplification! But note this all goes through even if θ is approximately a power of 2). Now consider the circuit with m ancilla bits:



The action of this circuit is:

$$
\begin{aligned}
|0^m\rangle|\psi\rangle &\overset{H^{\otimes m}}{\rightarrow} \left(\frac{1}{2^{m/2}}\sum_{k=0}^{M-1}|k\rangle\right)|\psi\rangle \\
&\overset{C_k(U)}{\rightarrow} \left(\frac{1}{2^{m/2}}\sum_{k=0}^{M-1}\lambda^k|k\rangle\right)|\psi\rangle = \left(\frac{1}{2^{m/2}}\sum_{k=0}^{M-1}\omega_M^{jk}|k\rangle\right)|\psi\rangle \\
&\overset{QFT_M^{-1}}{\rightarrow} |j\rangle|\psi\rangle
\end{aligned}
$$

Now, by measuring the ancillas in the computational basis, in a single shot we get j! The caveat - because we apply a k-controlled unitary, if we want exponential precision in the phase, we require the capacity to implement exponential powers of the unitary U. Unless we have access to this, the cost of this algorithm does not outperform the poor man's version.

A relevant application of phase estimation - if we have a Hamiltonian H, and want the ground state energy E_0 given the ground state $|\psi_0\rangle$, we can use phase estimation on $U = e^{-iHt}$. The difficulty is we require $|\psi_0\rangle$, or an approximation to it - most of the time we use a physically motivated ansatz.

9.5 Shor's Algorithm for Integer Factorization

Seeing Shor as a case of phase estimation is very cool, but nontrivial. We show that factoring reduces to order finding which reduces to phase estimation. To build up to this, we require some basic number theory.

The claim: Factoring is as easy as finding the non-trivial square roots of 1 mod N.

Definition: Non-trivial square root problem

Given odd natural N which is not prime, find $x = \pm 1 \pmod{N}$ such that $x^2 \cong 1 \pmod{N}$.

We can solve factoring given the ability to find such an x:

$$x^2 \equiv 1 \pmod{N} \implies x^2 - 1 \equiv \pmod{N} \implies (x+1)(x-1) \equiv 0 \pmod{N} \tag{9.5}$$

by assumption, both $x+1, x-1$ are non-zero mod N, which means that N must share a nontrivial factor with $x+1, x-1$. So, we can use Euclid's algorithm to find $\gcd(x+1, N)$ or $\gcd(x-1, N)$, and call it c- this is a factor of N!

Example: Let $N = 15$. We want the non-trivial square root of $1 \pmod{15}$. It is $x = 4$ because $4^2 \cong 1 \pmod{15}$ and $4 \neq \pm 1 \pmod{15}$. Both $\gcd(4-1, 15) = 3$ and $\gcd(4+1, 15) = 5$ are nontrivial factors of 15.

Next, we claim that finding non-trivial square roots is as easy as order finding.

Definition: Order finding

Given a, N relatively prime, find least $1 \leq r \leq N$ such that $a^r \cong 1 \pmod{N}$.

How do we reduce this to the non-trivial square root problem? The answer is to pick a random $a \in \mathbb{Z}_N$ and find its order r. Suppose we are lucky and r is even. Then, set $x \cong a^{r/2} \pmod{N}$ - in this case $x^2 \cong a^r \pmod{N} = 1 \pmod{N}$, by the definition of order. What happens if we get it wrong and r is odd? We just throw it away and do it again (there is a sufficiently large probability (over 1/2) we get it right). Of course we also want $x \neq \pm 1 \pmod{1}$, and again there is a high enough probability (over 1/2) that this is true.

These reductions were in the literature for a very long time. The interesting part/insight is that order finding reduces to phase estimation. Suppose we can do phase estimation - now suppose given a we want to find the order $r \pmod{N}$. We have to construct a unitary U corresponding to this problem. We consider the unitary $M_a : |x\rangle \to |xa \mod N\rangle$. Notice the eigenvectors are:

$$|\psi_k\rangle = \frac{1}{\sqrt{r}}\left(|1\rangle + \omega_r^{-k}|a\rangle + \ldots + \omega^{-k(r-1)}|a^{r-1}\rangle\right) \tag{9.6}$$

we can see this because:

$$M_a|\psi_k\rangle = \frac{1}{\sqrt{r}}\left(|a\rangle + \omega_r^{-k}|a^2\rangle + \ldots + \omega_r^{-k(r-1)}|a^r\rangle\right) = \omega_r^k \frac{1}{\sqrt{r}}(|1\rangle + \omega_r^{-k}|a\rangle + \ldots + \omega_r^{-k(r-1)}|a^{r-1}\rangle) = \omega_r^k|\psi_k\rangle \tag{9.7}$$

The upshot is the eigenvalue/phase encodes r, so phase estimation will tell us the order! We can implement powers of M_a efficiently - this is just modular exponentiation/repeated squaring. The only step remaining thus is how to obtain $|\psi_k\rangle$ - how do we prepare this without knowing what r is ahead of time?

10 Quantum Algorithms III

10.1 Shor, Continued

Last time, we discussed how to cast order finding in terms of an instance of phase estimation. We showed how the shift unitary M_a has eigenvalues $\omega_r^k = e^{2\pi i k/r}$ that encode the order and eigenvectors $|\psi_k\rangle$:

$$|\psi_k\rangle = \frac{1}{\sqrt{r}}\left(|1\rangle + \omega_r^{-k}|a\rangle + \ldots + \omega^{-k(r-1)}|a^{r-1}\rangle\right) \tag{10.1}$$

We need to be able to implement exponential powers of M_a - indeed we can do this via a modular exponentiation/repeated squaring algorithm (which reduces an $O(k)$ implementation to an $O(\log k)$ implementation for a power k).

The bow on the story is then - without knowing r, how do we produce the eigenvectors $|\psi_k\rangle$ (i.e. what quantum circuit will prepare this)? The answer is in fact we *cannot* prepare an individual $|\psi_k\rangle$ without knowing r, but we can make use of the trick that the sum over all k gives just the 1 state!:

$$\frac{1}{\sqrt{r}}\sum_{k=0}^{r-1}|\psi_k\rangle = |1\rangle \tag{10.2}$$

because all other terms drop out (sum over roots of unity).

Ok, hold on - now we're trying to do phase estimation without giving an eigenstate of M_a. How does this work then? We use the fact that the unitary is linear - we apply the operator to each of the $|\psi_k\rangle$ states in the linear combination. This gives us a superposition of different outputs, with all different k. When we measure, we get some phase $\frac{k}{r}$, for some uniformly random $k \in \{0, \ldots, r-1\}$.

So, we get some decimal expansion of $\frac{k}{r}$, which we then apply continued fractions to to extract r. This requires that k, r are relatively prime. There is some constant probability that this is not true, and we get the wrong answer. But since factoring is **NP**, we can just check the answer. In that case, we throw it away and try again. There is a sufficiently high (constant) probability that we succeed, and when we fail, we can verify that we fail. So, this does not turn out to be a problem.

10.2 Grover Search Algorithm

Definition: Unstructured Search

Given access to a function $f : \{0,1\}^n \to \{0,1\}$ (as an oracle/a black-box circuit computing f). The problem: Is there an x so that $f(x) = 1$?

Classically, the computational complexity of this is $O(N) = O(2^n)$ (because we need to query every input in the worse case!).

Note that this generalizes many **NP**-complete problems, e.g. SAT. But notice that there can be problems that can be easier than unstructured search due to structure (in some sense this unstructured search is a hard iteration of a problem).

Grover tells us that we can solve the search in $2^{n/2}$ oracle queries, or $2^{n/2}\text{poly}(n)$ time. Generally we want exponential quantum speedups, while this is only a polynomial (square root) speedup - thus we would never want to use this. The speedup is quite small, and the constant is quite large; the system size we require to see such a speedup requires order ~ 1 million qubits... by which time there are better things we could be doing. So be cautious of any startup that tells you that they want to do optimization a la Grover.

Let's now discuss the search algorithm. Suppose that $|a\rangle$ is the marked state, such that $f(a) = 1$.

1. Start with a uniform superposition $|\psi_0\rangle = H^{\otimes n}|0^n\rangle$ on n qubits.

2. Now, consider the angle between the uniform state and the marked state:
$$\frac{\pi}{2} - \theta = \arccos(\langle \psi_0 | a \rangle) = \arccos(\frac{1}{\sqrt{N}}) \implies \sin(\theta) \sim \theta = \frac{1}{\sqrt{N}} \tag{10.3}$$
3. We will apply many rounds each consisting of two unitary reflection operators - after each round, the angle between the current state $|\psi_n\rangle$ and $|a\rangle$ decreases.

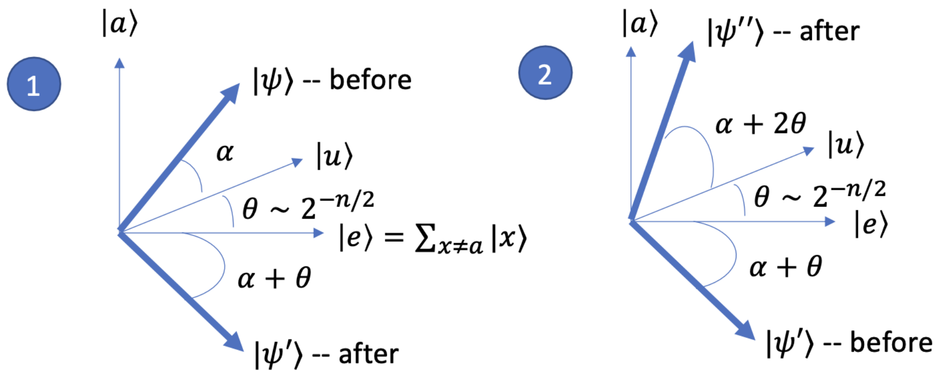
The idea is that we will only need $\sqrt{N}$ rounds.

The motivation for Grover's first operator is as follows: to reflect a vector $\mathbf{w}$ around another vector $\mathbf{v}$, we express $\mathbf{w} = \beta\mathbf{v} + \mathbf{v}^{\perp}$ and we output $\beta\mathbf{v} - \mathbf{v}^{\dagger}$. Thus we want a reflection about $|e\rangle = \sum_{x \neq a} |x\rangle$ (the state orthogonal to $|a\rangle$ in the plane spanned by uniform superposition and $|a\rangle$). We implement this via taking the current state $|\psi\rangle = \sum_x \alpha_x |x\rangle$ which we can think of as $\sum_{x \neq a} \alpha_x |x\rangle + \alpha_a |a\rangle$. We want to flip the sign of this last term (i.e. get $\sum_{x \neq a} \alpha_x |x\rangle - \alpha_a |a\rangle$). Thus we need only query the oracle once! (because the oracle precisely has the action of flipping the sign).

The second operator is as follows; we want to reflect around the uniform superposition $\frac{1}{\sqrt{N}} \sum_x |x\rangle$. Recall that $H^{\otimes n}$ maps to/from uniform superposition and $|0^n\rangle$. So, if we have $\sum_x \alpha_x |x\rangle$, then $H^{\otimes n} |\psi\rangle = \sum_x \beta_x |x\rangle$. Now, implement the unitary that multiplies by -1 if the second register is in anything but $|0^n\rangle$, so we get $\beta_{0^n} |0^n\rangle - \sum_{x \neq 0^n} \beta_x |x\rangle$. Then by taking $H^{\otimes n}$ again we map to $|0^n\rangle \to |u\rangle$.

The proof by picture:

1. Start in $|u\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle$
2. Want to reach $|a\rangle$ with $f(a) = 1$
3. We repeatedly apply two reflections that we claim that take $|\psi\rangle$ closer to $|a\rangle$
4. Consider the plane spanned by $|a\rangle, |u\rangle$



This shows the upper bound of the algorithm. Next class, we will show the lower bound, proving that the Grover algorithm is optimal.

11 Quantum Algorithms IV

11.1 Optimality of Grover's Algorithm

What this optimality proof shows - if QCs can solve **NP**-complete problem efficiently, it must use some meaningful structure of a Boolean formula (that would distinguish it from a blackbox). This square root speedup is the best we can do for unstructured search.

Let us show that *any* quantum algorithm must make $\sqrt{N}$ queries. Define $g : \{0,1\}^n \to \{0,1\}$ such that $g(z) = 1$ for some z and 0 otherwise, and $f(x) = 0$ for all x.

We give a high-level proof sketch. Consider any quantum algorithm that queries f. We show for htis algorithm that there exists a g such that the final state of this same algorithm that queries g is extremely close to the final state querying f, unless the number of queries is at least $\sqrt{N}$.

Thus, consider a general quantum algorithm that queries the all-zero function f, and it looks like $A = U_T F_T U_{T-1} F_{T-1} \ldots F_1 U_1$ (with F_i a query to f which acts as $F_i|x\rangle = (-1)^{f(x)}|x\rangle$ and U_i an f-independent unitary) and starts in $|0^n\rangle$.

We don't care about two Fs in a row because the Fs are self-inverse and hence we would never apply them twice in a row. We don't care about two Us in a row because that just forms another unitary and here we do not care about the gate complexity (in this proof, the Us are fully general!)

Let the state of the algorithm on the tth query be $\sum_x \alpha_{x,t}|x\rangle$. To define g (i.e. to set the marked string z), we'll define the notion of "query magnitude" of a string x, defined as $\sum_t |\alpha_{x,t}|^2$. Then, the expectation value of the query magnitude over random x is:

$$E_x\left[\sum_t |\alpha_{x,t}|^2\right] = \frac{1}{N}\sum_x\sum_t |\alpha_{x,t}|^2 = \frac{1}{N}\sum_t\sum_x |\alpha_{x,t}|^2 = \frac{1}{N}\sum_t 1 = \frac{T}{N} \tag{11.1}$$

Now notice that:

$$\min_x\left(\sum_t |\alpha_{x,t}|^2\right) = \frac{T}{N} \tag{11.2}$$

Now let z be the input at which this minimum occurs, then by Cauchy-Shwarz:

$$\sum_t |\alpha_{z,t}| \leq \frac{T}{\sqrt{N}} \tag{11.3}$$

We see this as:

$$\left(\sum_t |\alpha_{z,t}| \cdot 1\right)^2 \leq \langle\vec{\alpha}_z|\vec{\alpha}_z\rangle\langle\vec{1}|\vec{1}\rangle = T\sum_t |\alpha_{z,t}|^2 \leq T\frac{T}{N} \implies \sum_t |\alpha_{z,t}| \leq \frac{T}{\sqrt{N}} \tag{11.4}$$

Define g to be such that $g(z) = 1$ and otherwise 0.

We shall prove that the end state of the algorithm is close regardless of the oracle being g or f. Let the state of the algorithm after the Tth query to f be called $|\phi_T\rangle$ and the same algorithm with g as the oracle be called $|\psi_T\rangle$. In other words, we will show that $\left\||\psi_T\rangle - |\phi_T\rangle\right\| \leq \frac{T}{\sqrt{N}}$. Thus measuring the two states gives us $O(\frac{T}{\sqrt{N}})$ -close distributions (in statistical distance/total variation distance), and hence $T = \Omega(\sqrt{N})$ to distinguish the distributions with constant probability.

We will show this bound by doing a "hybrid" run of the algorithm wherein the algorithm makes some queries to f and some to g (these do not exist! But is useful as a proof/analysis technique of final runs of all f or all g). In these hybrid runs, we consider runs where the queries only differ in one spot (e.g. $fffgggg$ vs. $ffffggg$). These adjacent runs turn out to be easy to bound, because the only difference between the query f/g is the action on $|z\rangle$; this is a very local change. We then apply the bound N times, flipping fs to gs one at a time, e.g. $ffff \to fffg \to ffgg \to fggg \to gggg$. Now that the intuition is clear, let's go to formalizing this argument.

We claim:

$$|\psi_T\rangle = |\phi_T\rangle + |E_0\rangle + |E_1\rangle + \ldots + |E_{T-1}\rangle \tag{11.5}$$

where $\left\||E_t\rangle\right\| \leq 2|\alpha_{z,t}|$. Notice after this we are done since $\left\||\psi_T\rangle - |\phi_T\rangle\right\| \leq \sum_t 2|\alpha_{z,t}| \leq O(\frac{T}{\sqrt{N}})$. To prove the claim, consider two runs of algorithm A, which only differ on the tth-step. The first run queries f on the first t steps and queries g on the remaining $T-t$, vs. the second run queries f on the first $t-1$ steps and g on the remaining $T-t+1$.

After the first $t-1$ steps, both runs are in state $|\phi_t\rangle$. But on the tth step, one run queries f and the other g. So, the output of these queries differ only in a single amplitude, that of $|z\rangle$. So, we can write the states as $|\phi_t\rangle$ and $|\phi_t\rangle + |F_t\rangle$ where $\||F_t\rangle\| \leq 2|\alpha_{z,t}|$.

So, if we let U be the unitary transformation describing the remaining $T-t$ states of the algolrithm, then the ending states of both runs are $U(|\phi_t\rangle)$ and $U(|\phi_t\rangle + |F_t\rangle) = U|\phi_t\rangle + |E_t\rangle$ where $\||E_t\rangle\| \leq 2|\alpha_{z,t}|$ since unitaries preserve the ℓ_2 norm.

Now we can transform the run A_f to A_g by a succession of T changes of this sort. Overall, we can think of the final states as $|\phi_T\rangle$ plus a sum of the individual local differences, which gives us the bound.

12 The limits of quantum computation

The Grover lower bound is a bound on black box queries - it makes no reference to the locality/depth of the unitaries involved. This is the sense in which is it quite strong/very general. But it is weak in the sense that it is an oversimplification for how the real-world works. E.g. SAT formulas do not have to just be queried, often we make use of the structure. In the white-box world, given a white box unitary we can solve SAT in a single application, by encoding the truth table of the encoded Boolean function. In this setting, we care about time/depth lower bounds. Query lower bounds are a rough proxy for this.

12.1 QMA

We now proceed to discussion of QMA:

Computer science motivation - How do we precisely characterize the computational power of quantum computational models/get white box lower bounds?

Physics motivation - What is the ground state energy of a local Hamiltonian? It turns out this problem is QMA-complete, so the two meet in the middle.

We are interested in the limits of computation - what problems are too hard to be solved by computers efficiently? **NP** is the class of problems whose solutions can be verified efficiently. One way to think about this is a simple game between two players.

- Merlin, who knows everything but is dishonest, sends Arthur an efficient Turing machine (capable of solving everything in **P**) a witness to convince him to accept
- i.e. $L \subseteq \{0,1\}^n$ is in **NP** if there exists a deterministic efficient Turing machine (or "verifier", Arthur) V_L such that:
  - "Completeness": $x \in L \implies \exists y \in \{0,1\}^{p(|x|)}, V_L(x,y) = 1$
  - "Soundness": $x \notin L \implies \forall y \in \{0,1\}^{p(|x|)}, V_L(x,y) = 0$
- We get the class **MA** ("Merlin-Arthur") if we allow V to be bounded-error polynomial time and in the YES case V accepts with bounded probability and in the NO case rejects with bounded probability.
- We believe, but cannot prove unconditionally, that **MA** $=$ **NP**.

What is the quantum analogue? We have **QMA**, **QCMA**. In **NP**, we think about the message being sent as classical. In these quantum analogues, in the first case Merlin sends a quantum state to Arthur with a quantum computer in the second case Merlin sends a classical state to Arthur with a quantum computer (the other way does not make sense). It is easy to see that **QCMA** $\subseteq$ **QMA** because quantum states subsume classical states (computational basis states). The opposite inclusion is strongly thought to be false, but not proven.

Arthur is a bounded-error quantum probabilistic verifier. But Merlin, who sends a succint/polynomial sized witness, could be a classical bitstring or quantum state.

Definition: QMA

A promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ is in **QMA(c, s)** if there exists a polynomial sized uniform quantum circuit family $\{Q_n\}$ so that for any input x:

- "Completeness": If $x \in A_{\text{yes}}$ then there exists a witness state $|\psi_x\rangle$ such that $Q_{|x|}$ accepts with probability at least c
- "Soundness": If $x \in A_{\text{no}}$ then for all polynomial length witness states $|\psi\rangle$, the probability $Q_{|x|}$ accepts is at most s.

More formally, let Q_n be Arthur's circuit that acts on $n + p(n)$ qubits (input, witness), which we can also consider the ancilla space intialized as $|0^{p(n)}\rangle$.

- If yes, Pr[accept] $= \langle\psi|\langle x|Q_n^*|1\rangle\langle 1|Q_n|x\rangle|\psi\rangle > c$
- If no, Pr[accept] $= \langle\psi|\langle x|Q_n^*|1\rangle\langle 1|Q_n|x\rangle|\psi\rangle < s$

Generally, we take **QMA** $=$ **QMA**$(2/3, 1/3)$

Some variants:

- **QCMA**, wwhere the basis state $y \in \{0,1\}^{p(|x|)}$
- **QMA$_1$**: perfect completeness ($c = 1$)
- **QMA(2)**: Two unentangled Merlins, i.e. witnesses in this case are of the form $|\psi_1\rangle \otimes |\psi_2\rangle$. This turns out to be very important for checking for entanglement - it turns out this is **NP**-hard, and is not something Arthur can do by himself! Checking that the two Merlins give a tensor product state vs. one Merlin gives an entangled state is hard.

Let us discuss some properties of **QMA**. Let **QMA$_\mathbf{m}$** be **QMA** with witness length m. Then:

$$\mathbf{QMA} = \bigcup_{m \in \text{poly}(n)} \mathbf{QMA_m}$$

We can then amplify so $\mathbf{QMA_m}(c,s) = \mathbf{QMA}_{mr/(c-s)^2}(1 - 2^{-r}, 2^{-r})$.

There was one more lecture on the local Hamiltonian problem (that I unfortunately missed), as well as a set of student lectures for which I did not take notes.