CSE439/510 Lecture Tue. Aug. 27: Quantum Computing Overview

Philosophy I: "Simple Realism"

- Show polarizing filters. (Link to chapter with photo.)
- Show part of talk https://cse.buffalo.edu/~regan/Talks/UnionCollege52115.pdf

Philosophy II: Is Nature *Lexical*?

- The idea of *Logos* from 500 BCE. Identified, perhaps incorrectly, with "word".
- The possible meaning of the final sentence of Umberto Eco's novel *The Name of the Rose,* quoting Bernard of Cluny, 1100s*:*

Stat rosa pristina nomine; nomina nuda tenemus

This means: *The [original] rose abides (as a/by its) [original/former] name; we hold the bare name.* It is possibly a misquote of "Stat *Roma*..." meaning that we (in the 1100s or 2000s) only know the glory of ancient Rome through recorded memory of it. I, however, subscribe to a deeper reading that treats "pristina" as meaning "unsullied" rather than "original" and takes some liberties with grammar:

The rose abides unsullied by a name; we hold only the bare name.

Regarding the rose as representing Nature, the issue is whether Nature's workings must be read as paying heed to the symbolic way we describe them. The (theoretically-)efficient quantum factoring algorithm is a real challenge to the idea that nature is symbolically mathematical.

Quantum States

[Note: I have edited the following to number from zero in "underlying co-ordinates" as in the text. This is different from how most linear algebra texts do it. It will however be conventional to number "quantum coordinates" from 1.] Natural systems can be modeled (inefficiently!?) by vectors

$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_2 \\ a_3 \\ \vdots \\ a_i \\ \vdots \\ a_{N-1} \end{bmatrix}.$$

We say that $\mathbf{a}$ has N "underlying coordinates." Often N will be a power of 2, $N = 2^n$, where n will be the number of "quantum coordinates" or **qubits**. We can also have powers of larger numbers d, $N = d^n$. When $d = 3$ we will get **qutrits**, $d = 4$ will give **quarts**, and the general case gives **qudits**.

Maybe over 99% of the "QC" literature is about qubits. But actually, let's first think of N as not being subdivided at all.

One insight of linear algebra is that the entries a_i are not just "things unto themselves" but stand for multiples of corresponding **basis vectors**:

$$\mathbf{a} \;=\; a_0\,\mathbf{e_0} \;+\; a_1\,\mathbf{e_1} \;+\; a_2\,\mathbf{e_2} \;+\; \cdots \;+\; a_i\,\mathbf{e_i} \;+\; \cdots\; a_N\,\mathbf{e_N},$$

where for each i,

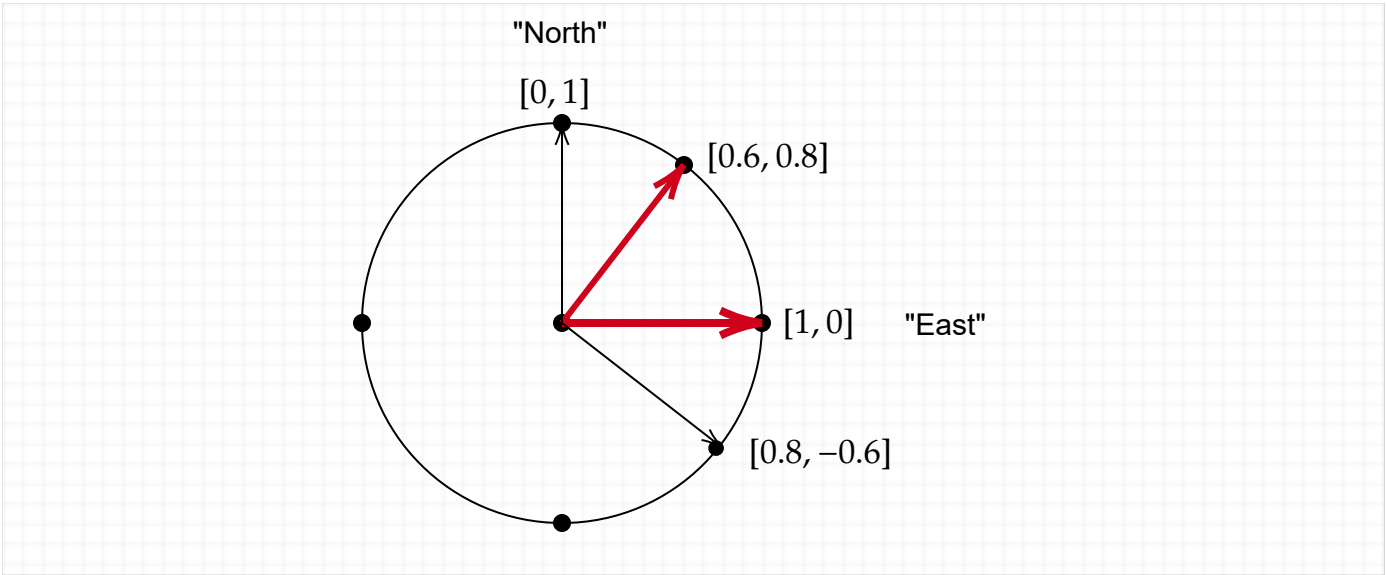
$$\mathbf{e_i} \;=\; [0, 0, 0, \ldots, 0, 1, 0, \ldots, 0]^T$$

with the lone 1 in position i. Notice we're being picky about considering vectors to be column vectors and writing transpose ^{T} to make $\mathbf{e_i}$ be a column vector. (Whether Nature really makes this distinction is a real question. We took the "no" side in the first edition, but using the angle-bracket notation from physics makes an initial commitment to the "yes" side.) With this notation, the vectors $\mathbf{e_i}$ are collectively called the **standard basis**.

A second insight of linear algebra is that one need not be "wedded to the standard basis"---one can do a **change-of-basis**. In general N-dimensional linear algebra, any set of N *linearly independent* vectors can be a basis. For instance, in $N = 2$ dimensions, the vectors

$[1, 0]$ and $[0.6, 0.8]$

are linearly independent (since there are only two vectors, the point is that neither is a multiple of the other). However, the second one is kind-of redundant in the first coordinate with the first. Whereas $\mathbf{e_0} \;=\; [1, 0]$ is "only East" and $\mathbf{e_1} \;=\; [0, 1]$ is "only North"---they are **orthogonal**, meaning that their *inner product* is zero.



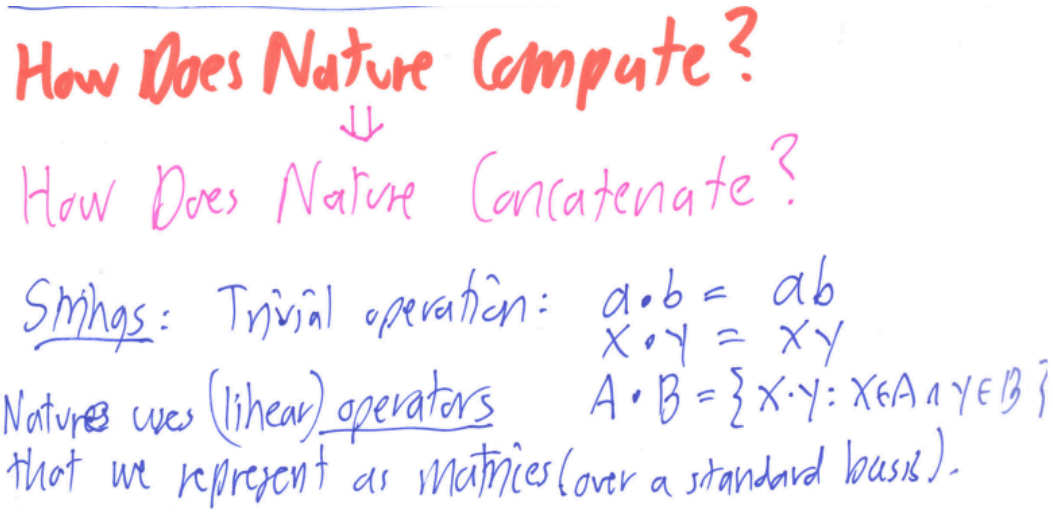
We can diagram these vectors on the *unit circle*---note that $0.6^2 + 0.8^2 = 0.36 + 0.64 = 1$. The inner product of $[0.6, 0.8]$ and our "East" vector is $0.6 \cdot 1 + 0.8 \cdot 0 = 0.6$.

There are several ways to write the inner product of two vectors $\mathbf{a}$ and $\mathbf{b}$:

$$\mathbf{a} \cdot \mathbf{b}, \quad \langle \mathbf{a}, \mathbf{b} \rangle, \quad \langle \mathbf{a} \mid \mathbf{b} \rangle.$$

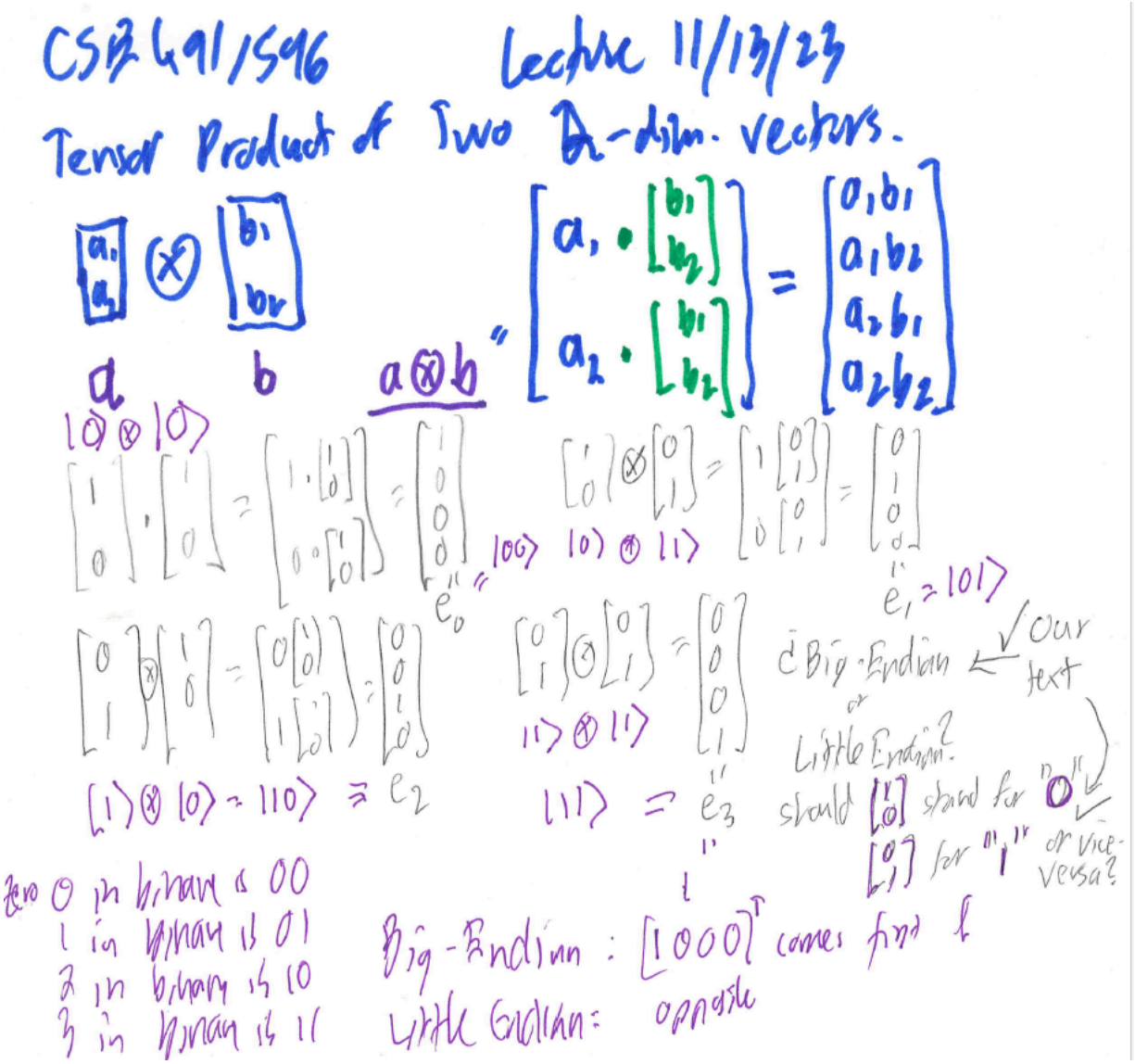
The last is what feeds into **Dirac Notation**, as the **bra**(c)**ket** of the row vector $\langle \mathbf{a}|$ and the column vector $|\mathbf{b}\rangle$. I will mention alongside various notations in chapter 2 and onward, but not require it. In order to motivate the notation scheme, I will briefly jump ahead to the topic of tensor products (chapter 3, section 3.2) but come right back out of it.

Tensor Products



When you think of matrices and vectors, the first idea that pops into mind is the ordinary matrix product AB of an $\ell \times m$ and an $m \times n$ matrix. But this is "lossy," whereas concatenation must be lossless (except possibly for memory of the place where the strings got concatenated). Instead, Nature uses **tensor product**, which applies also to vectors and doesn't need the "shapes" of the operands to agree.

Here are some handwritten and typeset examples.



[Pickup of lecture 2 was here]

An n-qubit quantum state is denoted by a unit vector in $\mathbb{C}^N$ where $N = 2^n$. Thus, a 2-qubit state is represented by a unit vector in $\mathbb{C}^4$. That takes up 8 real dimensions. There are tricks that get this down to a 6-dimensional hypersurface in $\mathbb{R}^7$, but until we have a Hyper-Zoom able to help us visualize 7-dimensional space, we have to rely on linear algebra and some general ideas about Hilbert Spaces (that don't care whether they are real or complex).

One of those ideas is the **standard basis**. In 4-space, this is given by the vectors:

$$e_0 = (1,0,0,0),\ e_1 = (0,1,0,0),\ e_2 = (0,0,1,0),\ e_3 = (0,0,0,1).$$

The indexing scheme for **quantum coordinates** changes the labels to come from $\{0,1\}^2$ instead of

from $\{1, 2, 3, 4\}$, using the canonical binary order $00, 01, 10, 11$. Then we have:

$$e_{00} = (1,0,0,0),\ e_{01} = (0,1,0,0),\ e_{10} = (0,0,1,0),\ e_{11} = (0,0,0,1).$$

The big advantage is that these basis elements are all separable and the labels respect the tensor products involved:

$$\begin{aligned}
|00\rangle &= e_{00} = (1,0,0,0) = (1,0)\otimes(1,0) = e_0 \otimes e_0 = |0\rangle\otimes|0\rangle = |0\rangle|0\rangle \\
|01\rangle &= e_{01} = (0,1,0,0) = (1,0)\otimes(0,1) = e_0 \otimes e_1 = |0\rangle\otimes|1\rangle = |0\rangle|1\rangle \\
|10\rangle &= e_{10} = (0,0,1,0) = (0,1)\otimes(1,0) = e_1 \otimes e_0 = |1\rangle\otimes|0\rangle = |1\rangle|0\rangle \\
|11\rangle &= e_{11} = (0,0,0,1) = (0,1)\otimes(0,1) = e_1 \otimes e_1 = |1\rangle\otimes|1\rangle = |1\rangle|1\rangle
\end{aligned}$$

It is OK to picture the tensoring with row vectors, but because humanity chose to write matrix-vector products as Mv rather than vM, they need to be treated as column vectors. This will lead to cognitive dissonance when we read quantum circuits left-to-right but have to compose matrices right-to-left. Lipton and I are curious whether a "non-handed" description of nature can work.

There is an even more immediate "left-right" issue to get to. What the text in chapter 2 calls the **canonical numbering of strings** is actually a choice. For two qubits, the above amounts to:

$$\begin{aligned}
00 &= 0 \\
01 &= 1 \\
10 &= 2 \\
11 &= 3.
\end{aligned}$$

This is indeed canonical in being how we write binary numbers. It also orders the (same-length) binary strings in **lexicographical order**, as used by ASCII. However, this makes column 1 (which we will soon call "qubit 1") the *most* significant bit. This is **big-endian**. The other way is to make the leftmost column be the *least* significant bit:

$$\begin{aligned}
00 &= 0 \\
10 &= 1 \\
01 &= 2 \\
11 &= 3.
\end{aligned}$$

This is **little endian**. Here are the comparisons for length-3 strings:

$$\begin{bmatrix} Big & End & ian \\ 000 & = & 0 \\ 001 & = & 1 \\ 010 & = & 2 \\ 011 & = & 3 \\ 100 & = & 4 \\ 101 & = & 5 \\ 110 & = & 6 \\ 111 & = & 7 \end{bmatrix} \qquad \begin{bmatrix} Little & End & ian \\ 000 & = & 0 \\ 100 & = & 1 \\ 010 & = & 2 \\ 110 & = & 3 \\ 001 & = & 4 \\ 101 & = & 5 \\ 011 & = & 6 \\ 111 & = & 7 \end{bmatrix}$$

An important curveball with little endian is that the relation to tensor product of basis elements does not work---it needs another reversal. For instance:

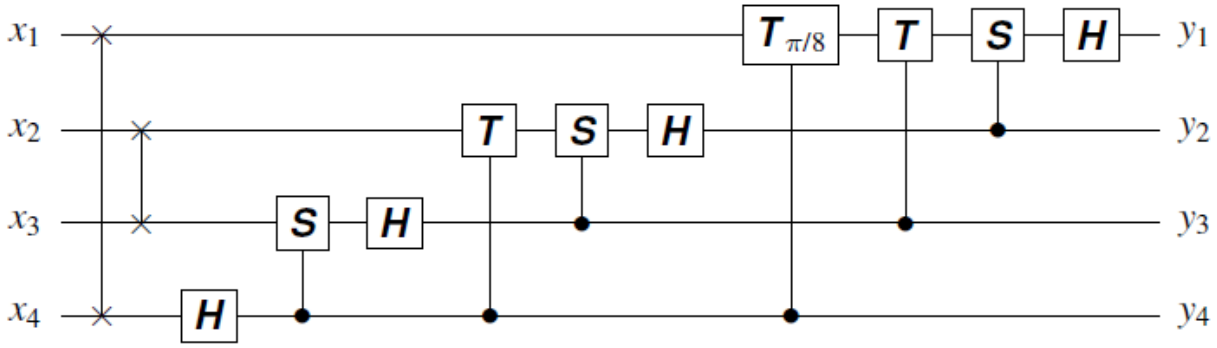
e_0 is still $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ in little-endian, because the order of 0 and 1 by themselves is the same.

But $e_{01} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ rather than $\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ because 10 now comes before 01 in little-endian. And:

$$e_0 \otimes e_{01} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\ 0 \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ which alas is not the index for 001. You have to flip the tensor product too:

$$e_{01} \otimes e_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 0 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 1 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 0 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ which is the basis vector for 001 in little-endian.

Where this really matters is when we write qubits going down rather than across, like notes on a musical staff:



However, we can mentally convert if we imagine rotating this 90 degrees right and reading across---so x_4 is in the leftmost column and gets read as if it were "x_1", etc. Some other discussion:

https://quantumcomputing.stackexchange.com/questions/8244/big-endian-vs-little-endian-in-qiskit
https://pasqal-io.github.io/qadence/v1.5.2/content/state_conventions/

We will use Big Endian officially in this course---needing Little Endian only to read optional quantum circuit widgets that use it.

Tensor products can be repeated---but they get exponentially big when you do so. Simply for instance:

$$e_0 \otimes e_0 \otimes e_0 = (e_0 \otimes e_0) \otimes e_0 = (1,0,0,0)^T \otimes e_0 = (1,0,0,0,0,0,0,0)^T = e_{000}$$

$$e_0 \otimes e_0 \otimes e_0 \otimes e_0 = e_{000} \otimes e_0 = (1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)^T = e_{0000}$$

$e_0^{\otimes 8} = e_{00000000}$ = (a 1 followed by 255 0s).

Time Complexity and O-Notation

The number n will generally stand for "the total number of unit-size data points." The concepts "time at most order-of", "time proportional to", and "vanishingly smaller than" are necessarily rough. We can, however, give a precise mathematical definition of them in a way that incorporates their roughness:

The key definition is: Given two numerical functions $f(n)$ and $g(n)$,

- $f(n) = O(g(n))$ if there are constants c and n_0 such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$.
- $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $g(n) = O(f(n))$.

- $f(n) = o(g(n))$ if the limit of $f(n)/g(n)$ goes to 0 as n goes to infinity.

Big-O Notation

Suppose that on problems with n data items (counting chars or small ints/doubles), your program takes at most $t(n)$ steps. Let $g(n)$ stand for a performance target. Then

$$t(n) = O(g(n)),$$

meaning your *program design* achieves the target, if there are constants $c > 0$ and $n_0 \geq 0$ such that:

$$\text{for all } n \geq n_0,\ t(n) \leq cg(n).$$

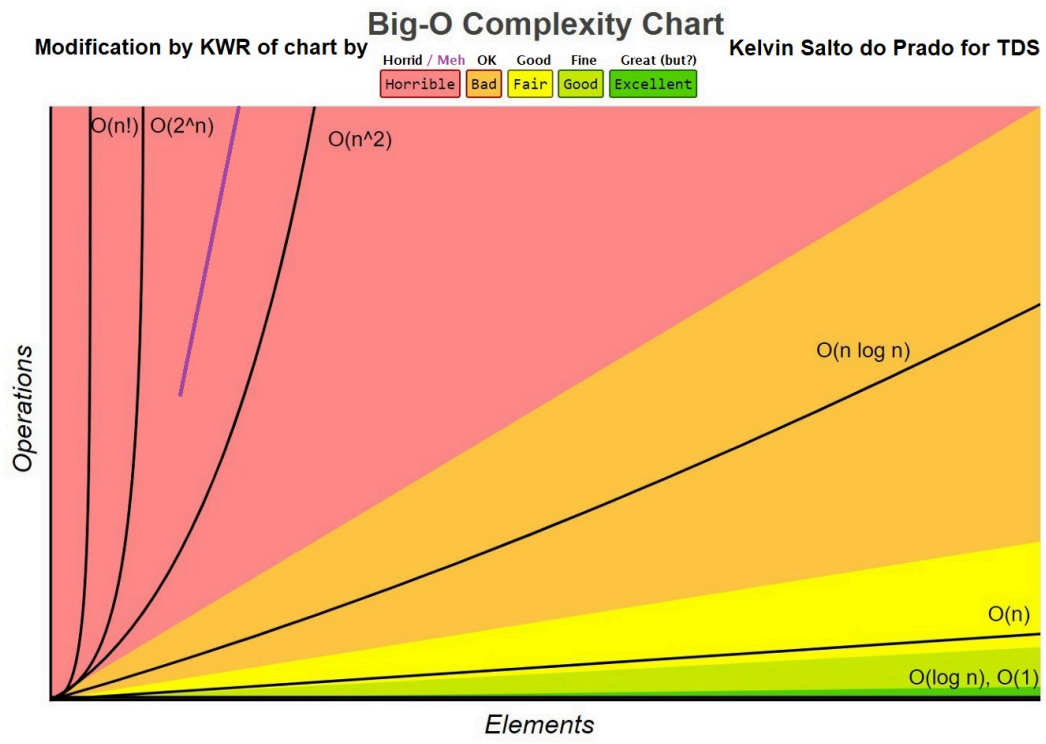
Here c is called "the constant in the O" and should be estimated and minimized as well, even though "$t = O(g)$" does not depend on it. Having n_0 be not excessive is also important. (Often we think of "c" as being ≥ 1.)

Analogy to $<, =, >$

- The real numbers enjoy a property called trichotomy: for all a, b, either $a < b$ or $a = b$ or $a > b$.
- Functions $f, g : \mathbf{N} \longrightarrow \mathbf{N}$ do not, e.g. $f(n) = \lfloor n^2 \sin n \rfloor$ and $g(n) = n$ [a quick hand-drawn graph was enough to show this in class].
- However, the British mathematicians Hardy and Littlewood proved that *for all real-number functions f, g built up from $+, -, *, /$ and* $\exp, \log$ *only*,

$$f = o(g) \quad \text{or} \quad f = \Theta(g) \quad \text{or} \quad g = o(f).$$

- Thus common functions fall into a nice linear order by growth rate (see chart from text).



More examples of curves, tradeoffs, and the role of the leading constant are in the graphs of Jim Marshall from a course at Sarah Lawrence:
http://science.slc.edu/~jmarshall/courses/2002/spring/cs50/BigO/index.html

Some useful instances:

$\sqrt{N} = \sqrt{2^n} = \left(2^n\right)^{1/2} = 2^{n/2}$ which is still $2^{\Theta(n)}$ exponential in n.

But $2^{O(\log n)} = \left(2^{(\log n)}\right)^{O(1)} = n^{O(1)} = polynomial.$

Concretely with 3 as the "constant in the O": $2^{3(\log n)} = \left(2^{(\log n)}\right)^3 = n^3 = polynomial.$

Computational Complexity

We have talked about the running times of ~~Turing machines~~ *algorithms in general*, already. It is finally time to formalize this. We will call a collection $\{C_n\}$ of classical Boolean (or quantum) circuits, where each C_n handles data of size n, a single "machine" or "algorithm"---presupposing that the C_n have common characteristics for any individual n.

Definition:

1. Given a function $t : \mathbb{N} \to \mathbb{N}$, a machine M **runs in time** $t(n)$ if for all n and inputs x of length n, $M(x) \downarrow$ within $t(n)$ steps.
2. Given a function $s : \mathbb{N} \to \mathbb{N}$, a machine M **runs in space** $s(n)$ if for all n and inputs x of length n, $M(x) \downarrow$ while **changing** the character in at most $s(n)$ tape cells.
3. A nondeterministic Turing machine runs within a given time or space bound if **all** of its possible computations obey the bound.

Note that although a computation can "loop" within a finite amount of space, the machine is not regarded as running within that space (in practice, the activation stack or some other tracker would overflow). When the input tape is read-only, the space measure is essentially equivalent to the number of cells accessed on the initially-blank worktapes. For some examples:

Definition: For any time function $t(n)$ and space function $s(n)$, using M to mean DTM and N *for NTM*:

1. DTIME$[t(n)] = \{L(M) : M \text{ runs in time } t(n)\}$
2. NTIME$[t(n)] = \{L(N) : N \text{ runs in time } t(n)\}$
3. DSPACE$[s(n)] = \{L(M) : M \text{ runs in space } s(n)\}$
4. NSPACE$[s(n)] = \{L(N) : N \text{ runs in space } s(n)\}$

Convention: For any collection T of time or space bounds, in particular one defined by O-notation,

DTIME[T] means the union of DTIME[$t(n)$] over all functions $t(n)$ in T, and so on.

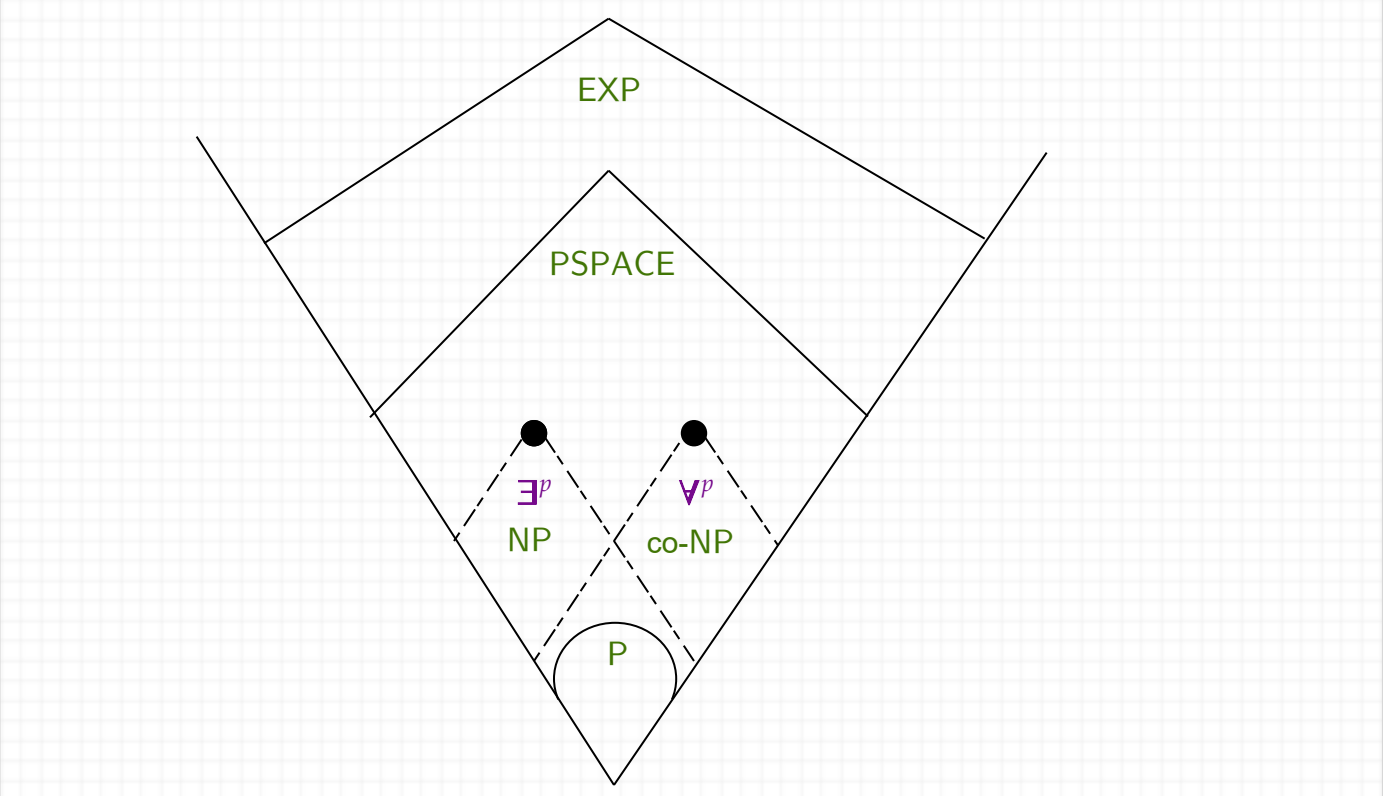
Definition (some of the "Canonical Complexity Classes"):

1. P $=$ DTIME$\left[n^{O(1)}\right]$
2. NP $=$ NTIME$\left[n^{O(1)}\right]$
3. DLOG $=$ DSPACE$[O(\log n)]$. Also called just L for "Logspace."
4. NLOG $=$ NSPACE$[O(\log n)]$. Also called just NL for "Nondeterministic Logspace."
5. PSPACE $=$ DSPACE$\left[n^{O(1)}\right]$
6. EXP $=$ DTIME$\left[2^{n^{O(1)}}\right]$.

The only class we know to contain languages not in P is the last one: we know P $\subsetneq$ EXP. Regarding line 5, it seems we've skipped an analogously-defined class "NPSPACE" but it actually equals PSPACE. Right now P and NP, along with co-NP $= \{\sim L : L \in \text{NP}\}$ will take center stage. Here co-NP means problems where a lucky guess can confirm a *no* answer---which are the complements of problems where a lucky guess can confirm a *yes* answer.

A protypical problem about n-vertex **graphs** G is: given a start node s and a target node t, is there a path from s to t? (And if so, can you find such a path?) Here the lucky guess (in the "yes" case where a path really exists) is a series of right choices of next step to take, without going into a dead end or a vicious cycle. The only memory you need to keep is your current vertex v in the path---we suppose that G itself is given as read-only input that you can consult at will for free. Since v can be a binary number from 1 to n, and since you can forget v once you make the next good step along an edge to a vertex v' in G, you only need to maintain $O(\log n)$ bits of modifiable storage. That is what classifies the problem into Nondeterministic Logspace.

The notion of a "lucky guess" is the same as saying **there exists** a solution that you can verify *in polynomial time*. This is symbolized by an existential quantifier with a little p superscript. When a "*no*" answer is prone to a lucky guess, it may mean that a "yes" answer requires **all** possibilities to confirm "yes". This is symbolized by a universal quantifier with p superscript---like so:



When a lucky guess works for both the "yes" and "no" cases, the problem is in NP ∩ co-NP. We note this below for a natural way to express the task of factoring a number N by yes/no questions that can narrow down a factor. In both cases, the "lucky guess" is the entire unique prime factorization of N.

Problems in NP and co-NP

[Lecture got as far as SAT, 3SAT, G3C,and TAUT]

It is usually easiest to tell that (the language of) a decision problem belongs to NP by thinking of a witness and its verification. For example:

Satisfiability (SAT):
Instance: A logical formula ϕ in variables $x_1, \ldots, x_n$ and operators $\wedge, \vee, \neg$.
Question: Does there exist a truth assignment $a \in \{0,1\}^n$ such that $\phi(a_1, \ldots, a_n) = 1$?

The assignment cannot have length longer than the formula, and *evaluating* a formula on a given assignment is quick to do. Hunting for a possible *satisfying assignment*, on the other hand, takes up to 2^n tries if there is no better way than brute force. This is apparently hard even when the Boolean formula has a simple form.

Definition. A Boolean formula is in **conjunctive normal form** (CNF) if it is a conjunction of **clauses**

$$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m,$$

where each clause C_j is a disjunction of **literals** x_i or $\bar{x}_i$. The formula is in k**-CNF** if each clause has at

most k distinct literals (*strictly* so if each has exactly k).

3SAT
Instance: A Boolean formula $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ in 3CNF.
Question: Is there an assignment $\vec{a} = a_1 a_2 \cdots a_n \in \{0,1\}^n$ such that $\phi(a_1, \ldots, a_n) = 1$?

Now for a problem with a different kind of witness:

Graph Three-Coloring (G3C):
Instance: An undirected graph $G = (V, E)$.
Question: Does there exist a 3-coloring of the nodes of G?

A *3-coloring* is a function $\chi : V \to \{R, G, B\}$ such that for all edges $(u, v) \in E$, $\chi(u) \neq \chi(v)$. The table for χ needs only n entries where $n = |V| \ll N = |G|$, so it has length at most linear in the encoding length N of G (often $N \approx n^2$). And it is easy to *verify* that a *given* coloring χ is correct.

PRIMES $= \{2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots\}$ (encoded as, say, $10, 11, 101, 111, 1011, \ldots$)

This language was formally shown to belong to P only in 2004, but had long been known to be "almost there" in numerous senses. But now consider this one:

FACT:
Instance: An integer N and an integer k.
Question: Does N have a prime factor p such that $p \leq k$?

If you can always answer yes/no in polynomial time $r(n)$, where $n \approx \log_2 N$ is the number of bits in N, then you can do *binary search* to *find* a factor p of N in time $O(nr(n))$. By doing $N' = n/p$ and repeating you can get the complete factorization of N in polynomial time. This is something that the human race currently does **not** want us to be able to solve efficiently, as it would (more than Covid?) "destroy the world economy" by shredding the basket in which most of our security eggs are still placed. (This is the gist of the 1992 movie *Sneakers* with Robert Redford heading an all-star cast.) But to indicate proximity to this peril, we note:

FACT: **FACT** is in NP $\cap$ co-NP.

Proof: The witness for "no" as well as "yes" is the unique prime factorization $N =: p_1^{a_1} p_2^{a_2} \cdots p_\ell^{a_\ell}$. Although the right-hand side may seem long, ℓ cannot be bigger than the number of bits of N in binary because each p_i is at least 2, and bigger powers only make ℓ have to be smaller. The length of the factorization is $O(n)$. To verify it, one must verify that each p_i is prime---but this is in polynomial time as above---and then simply multiply everything together and check that the result is N. Finally, to verify the *yes* answer, check that at least one of the p_i is $\leq k$; *no* if none.

TAUT:
Instance: A Boolean formula ϕ', same as for SAT.
Question: Is ϕ' a **tautology**, that is, true for all assignments?

Note that ϕ is unsatisfiable $\equiv$ every assignment a makes $\phi(a)$ false $\iff$ every assignment a makes $\phi'(a)$ true, where $\phi' = \neg\phi$. Thus TAUT is essentially the complement of SAT.