CSE439/510 Fall 2025 Week 1: QC Overview, Chapters 1, 2, and 3.1--3.2.

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 names.* It *misquotes* Cluny's "Stat ***Roma***..." meaning that we (in the 1100s or 2000s) know the glory of ancient Rome only through recorded memory of it. I, however, subscribe to a deeper reading that treats "pristina" as meaning "unsullied" rather than "original", takes some liberties with Latin grammar, and brings in Shakespeare's "a rose by any other name would smell as sweet" (as I believe Eco intended):

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

Regarding the rose as representing Nature, and "names" as our lexical mathematical and programming-language formalisms, 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 by Peter Shor is a real challenge to the idea that nature is symbolically mathematical.

Even before computational complexity theory developed, there were loomings that caused Richard Feynman to exclaim, "Nature isn't classical, dammit!" (whole quote). Feynman wrote a paper about building a quantum machine to do simulations circa 1980. David Deutsch took up the technical mantle in the 1980s while he was a postdoc at Oxford, where I was a student. We will pick up his story in detail later in chapters 8 and 9. He first thought that quantum computers could solve the Halting Problem in definitive time, but when that was rebuffed, he started again on a smaller scale. That is where we will begin.

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

$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_i \\ \vdots \\ a_{N-1} \end{bmatrix}.$$

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,

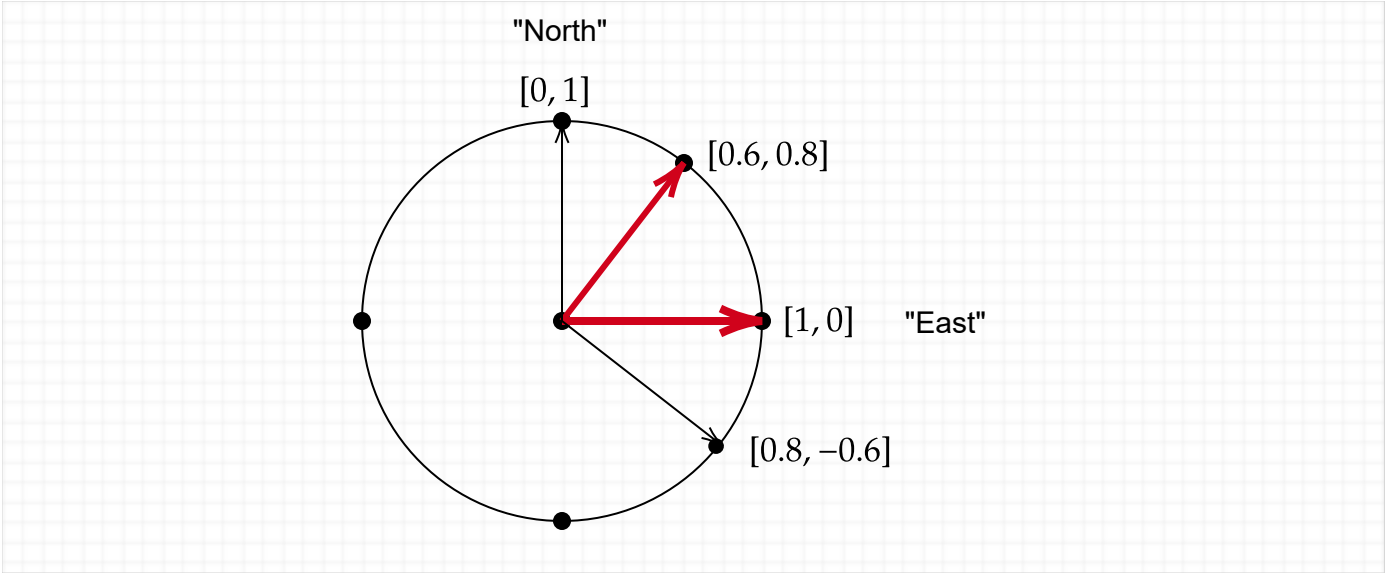
$$\mathbf{e_i} = [0, 0, 0, \ldots, 0, 1, 0, \ldots, 0]^T$$

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 $0.6 \cdot 1 + 0.8 \cdot 0 = 0.6$.

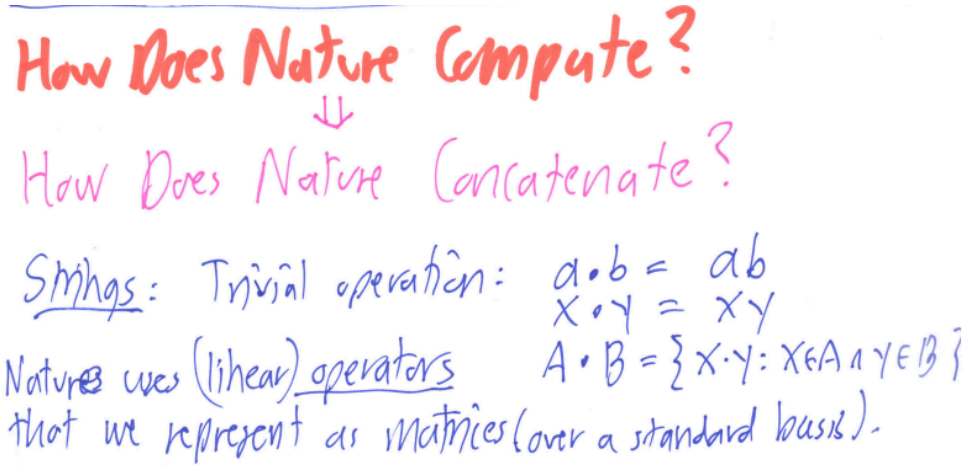
There are several ways to write the inner product of two vectors $\mathbf{a}$ and $\mathbf{b}$:

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

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.

[The first lecture then covered syllabus information: homework and exams; expectations; grade objectives and policies; sequence of material to be covered. The second lecture picked up here.]

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 is an informal definition that is more general than matrices:

[This part of the lecture was done at the blackboard.]

Definition: The **tensor product** $A \otimes B$ of two "aggregate objects" A and B is obtained by multiplying every element of A by a whole copy of B, and adjusting dimensions accordingly.

The most basic important example also introduces another general question in physics: does Nature distinguish "handedness"? The technical word for this is chirality. Whatever the answer for "nature's rose", we with our mathematical nomenclature have to be careful. Note that whereas ordinary multiplication associates left-to-right, matrix multiplication goes right-to-left. A concrete illustration for $n \times n$ matrices is, how would you calculate $(A \cdot B)u$ given a length-n vector u? If you multiply A and B first, you spend order-of n^3 steps doing that the usual way. Whereas, if you work right-to-left by first computing $v = Bu$ and then doing Av, you spend order-of n^2 operations twice, which is still order-of n^2 time. (We will define the notation $O(n^2)$ etc. for "order-of" shortly.)

Example: We will represent the qubit "object" **0** by the standard basis vector $e_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and **1** by $e_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ as column vectors. Then

$$e_0 \otimes e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 0 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \text{ and } e_1 \otimes e_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 1 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

Note that the latter is the standard basis vector e_2, continuing our policy of numbering the N-many "underlying coordinates" from 0. If we write 2 in binary as 10 and 1 as 01, then we get:

$$e_0 \otimes e_1 = e_{01} \quad \text{and} \quad e_1 \otimes e_0 = e_{10}.$$

And:

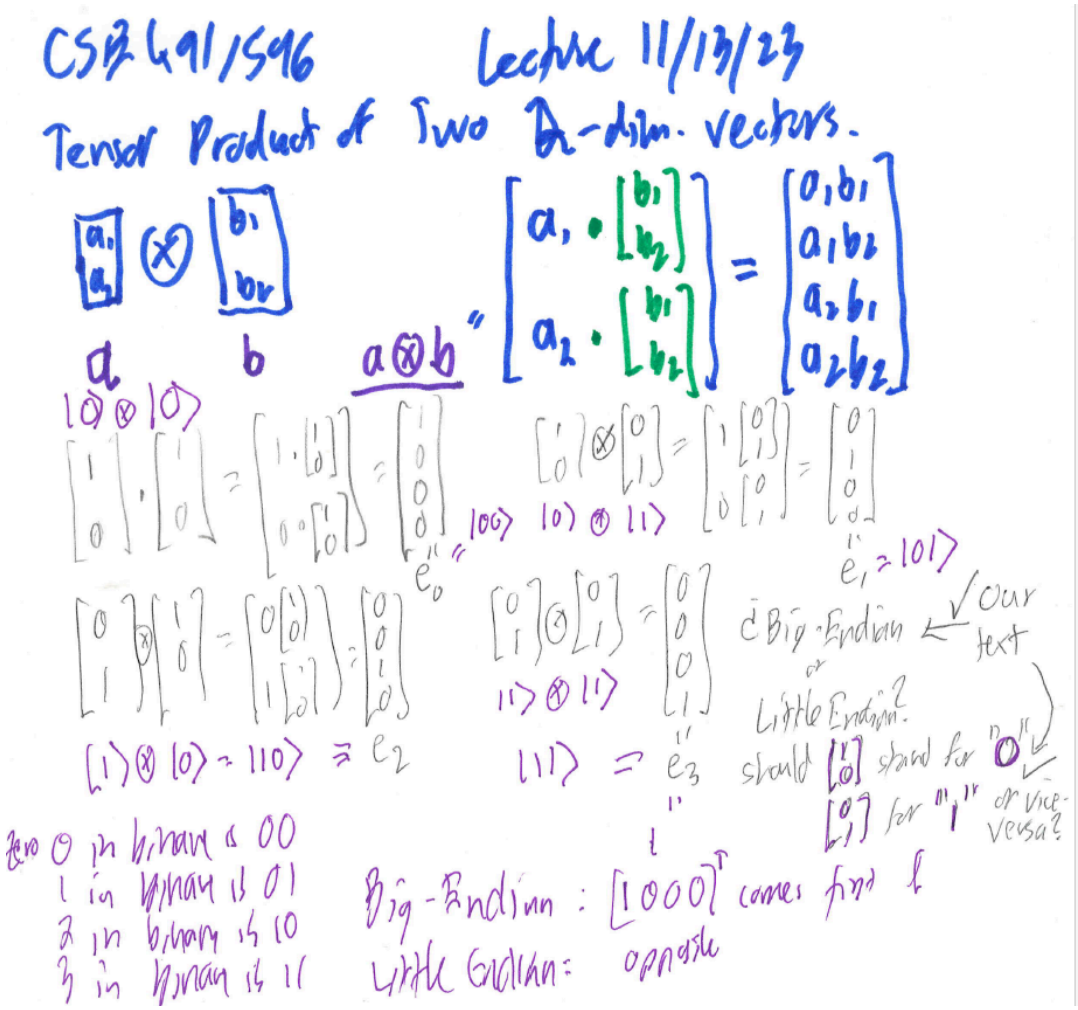
$$e_0 \otimes e_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 0 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = e_{00}; \; e_1 \otimes e_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 1 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = e_{11}.$$

Thus tensor product does concatenation on the binary strings that index the standard basis vectors, first in dimension 2 and then in dimension 4. The resulting ordering of the binary strings as

$00, 01, 10, 11$ is called **lex**(icographic) **order**. A more visceral way to refer to the standard basis vectors is to put "brackets" around the binary strings themselves: $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. This notation was invented circa 1930 by the physicist Paul Dirac, so is called **Dirac notation**. It is defined in the posted excerpt of Chapter 14 of the text, which we will read in parallel starting next week. Here are the two notations side-by-side:

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

Now we can picture this example as a case of the general rule for when A is a 2-vector $[a_1, a_2]^T$ and $B = [b_1, b_2]^T$. Here I'm using the superscript T for "transpose" just to avoid typing column vectors, which are bulkier with vertical space. (Actually, you can apply the tensor rule to two row vectors without transposing; then you get a longer row vector.)



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).

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, using a revised definition:

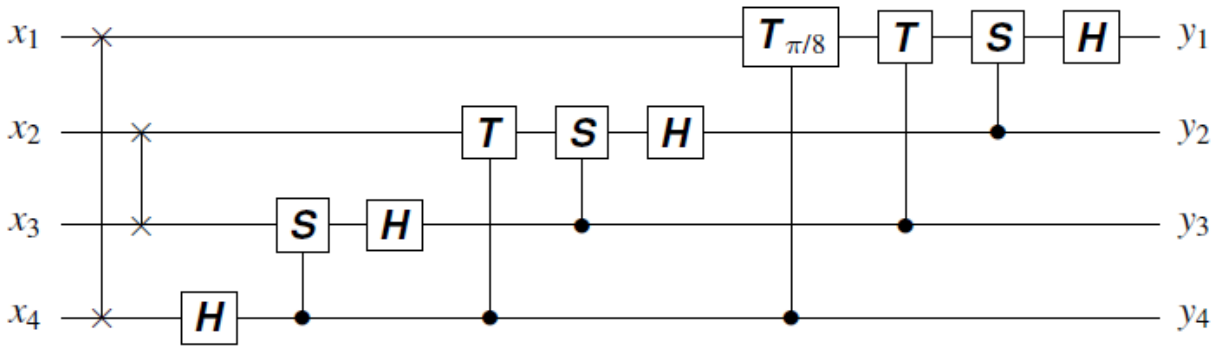
Definition: The "little endian" tensor product of two "aggregate objects" A and B is obtained by multiplying every element of B by a whole copy of A, and adjusting dimensions accordingly.

Applied to $A = e_0$ and $B = e_{01}$ as above, we instead form:

$$\begin{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot 0 \\ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot 0 \\ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot 1 \\ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = e_4.$$

Looking back in the Little Endian table, this is the index for e_{001} as we want. Thinking about the flipped definition a little more, it turns out to be the same as doing $B \otimes A$. Note that like with matrix product---and more pertinently, like with string concatenation---tensor product is not usually commutative. But if we import the details of reversing the tensor product into our mental mindscape, confusions that we already have to deal with could get mushroomed.

Happily, there is a "visual" way to handle the reversals and read diagrams from little-endian web apps such as Quirk and Qiskit while staying inside big-endian notation---which is used by the most immediately user-friendly app, Davy Wybiral's Quantum Circuit Simulator. All of them portray quantum circuits the same way, rather like notes on a musical staff, e.g.:



In little-endian, the qubits might not themselves be labeled woth x_1 and y_1 on the bottom, but the little-endian matrix representations of the gates would look that way. However, we can mentally convert if we imagine **rotating the diagram 90 degrees right and reading across-and down**, so that 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:

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

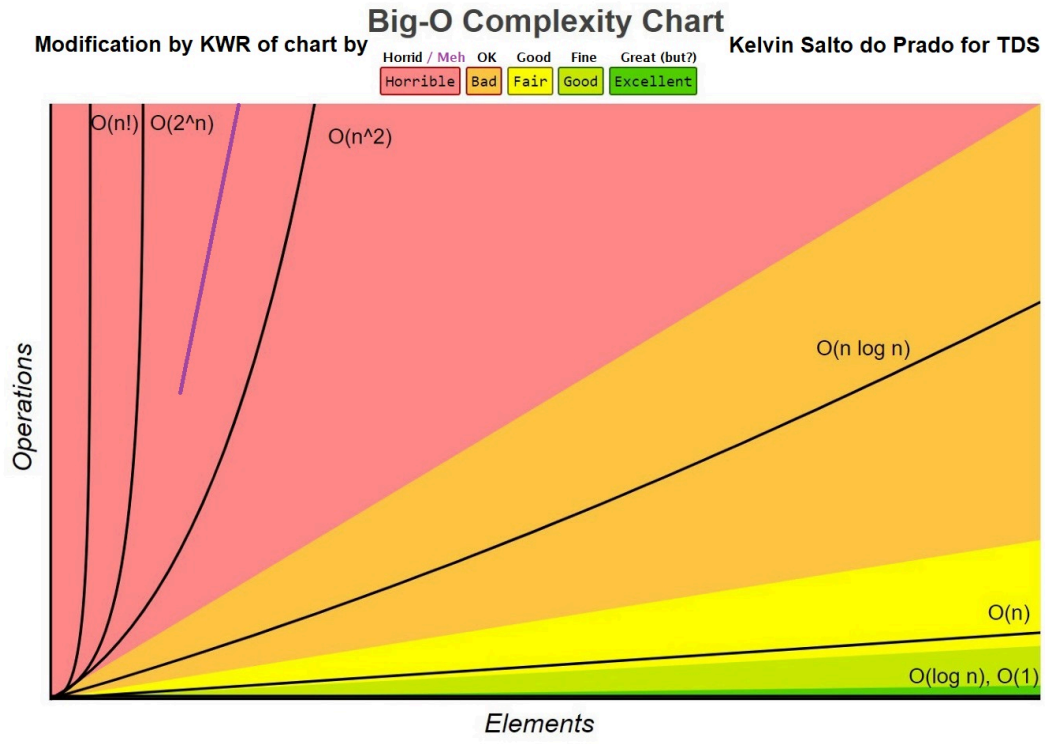
Here c is called "the constant in the O" and should be estimated and minimized as well, even though "$t = O(g)$" does not depend on it. Having n_0 be not excessive is also important. (Often we think of "c" as being ≥ 1.)

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 classes will be introduced later alongside chapters 4 and 5, where they are more technically relevant.]