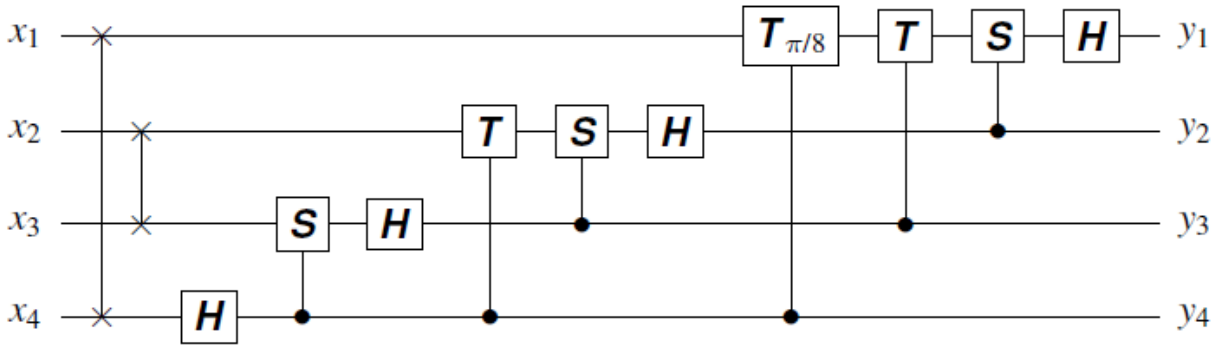
CSE439 Fall 2026 Week 1 Thu: Quantum States, Representations, and Growth Rates

The main difference between quantum circuits and Boolean circuits is this:

- When a **bit** is emitted by one gate and read by another, it is not regarded as having physical existence beyond that one interaction.
- A **qubit**, however, is a physical object that persists through application of **unitary** linear operators (which we call "gates") and *may be* destroyed only by non-unitary operators such as **measurements**, or by nonlinearity.

A system of n qubits is thus represented by n "**lines.**" We will speak of the lines as traveling through time as gates are applied. The lines look like a musical staff, while single-qubit gates are "notes" and multi-qubit gates are "chords". E.g.:



This system has $n = 4$ qubits, but underlying it are $N = 2^4 = 16$ vector dimensions.

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 **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**:

$$\mathbf{a} = a_0\,\mathbf{e_0} + a_1\,\mathbf{e_1} + a_2\,\mathbf{e_2} + \cdots + a_i\,\mathbf{e_i} + \cdots a_{N-1}\,\mathbf{e_{N-1}},$$

where for each i,

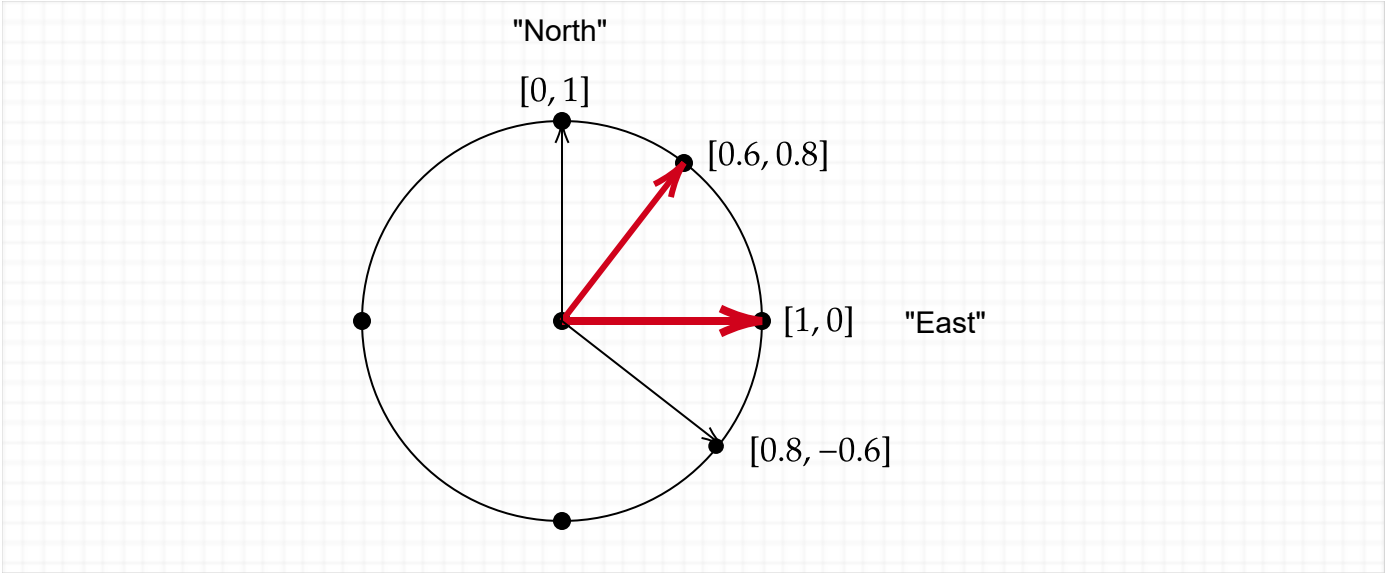
$$\mathbf{e_i} = [0, 0, 0, \ldots, 0, 1, 0, \ldots, 0]^T$$

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, which will be covered in earnest next week (chapter 3, section 3.2), but come right back out of it.

Quantum versus Classical Coordinates, and Concatenation (and Endianness)

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:

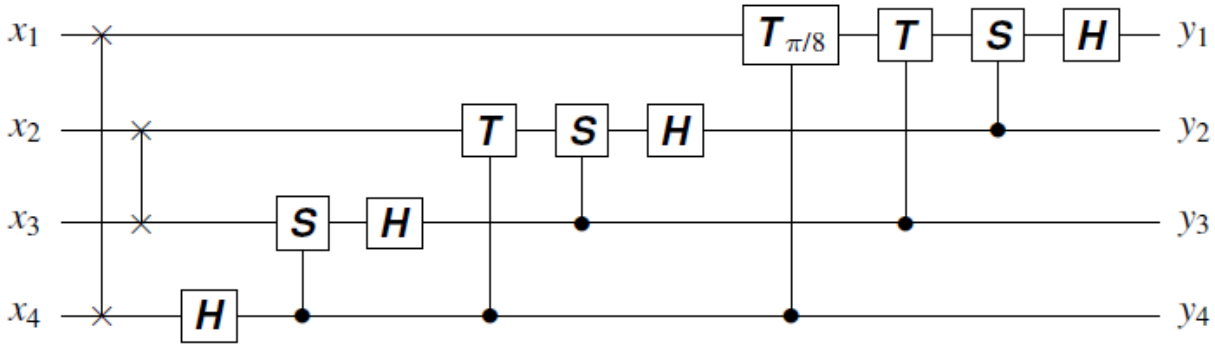
Definition: The "little endian" tensor product of two "aggregate objects" A and B is obtained by multiplying every element of B by a whole copy of A, and adjusting dimensions accordingly.

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.

A Brief Glimpse of the Whole Idea

The 2×2 **Hadamard matrix**, denoted by $\mathbf{H}$ in the diagram, is given by $\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$. The inverse square root of 2 in front makes the determinant become -1 rather than -2, and more to the point, creates a **unitary matrix**. Just like **unit vectors** are legal quantum states, unitary matrices are legal operations on qubits.

Now let us apply $\mathbf{H}$ to our state $[0.6, 0.8]^T$ shown above. We get:

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 0.6 \\ 0.8 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1.4 \\ -0.2 \end{bmatrix} = \begin{bmatrix} 0.9899... \\ 0.1414... \end{bmatrix}.$$

The resulting vector is exactly a unit vector because $1.4^2 + |-0.2|^2 = 1.96 + 0.04 = 2$, but the square of the $1/\sqrt{2}$ in front makes the sum become 1. But notice how lopsided it is: Most of the **amplitude** is on the first coordinate. If you measure, the probability of getting the 0 bit outcome is $\left(1.4/\sqrt{2}\right)^2 = 1.96/2 =$ exactly 0.98, i.e., 98%. The chance of getting the 1 answer is only 2%.

> The object of a quantum algorithm is to transform the input x into a state vector y such that most of the amplitude in y has been "mushed" onto the desired answer(s).

Alas, I cannot show this by directly inputting $x = [0.6, 0.8]^T$ into a quantum simulator. Those only let you start with states in the standard basis or a few other special orthonormal bases. So we have to do extra work to **prepare** the state $[0.6, 0.8]^T$ to begin with.

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:

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

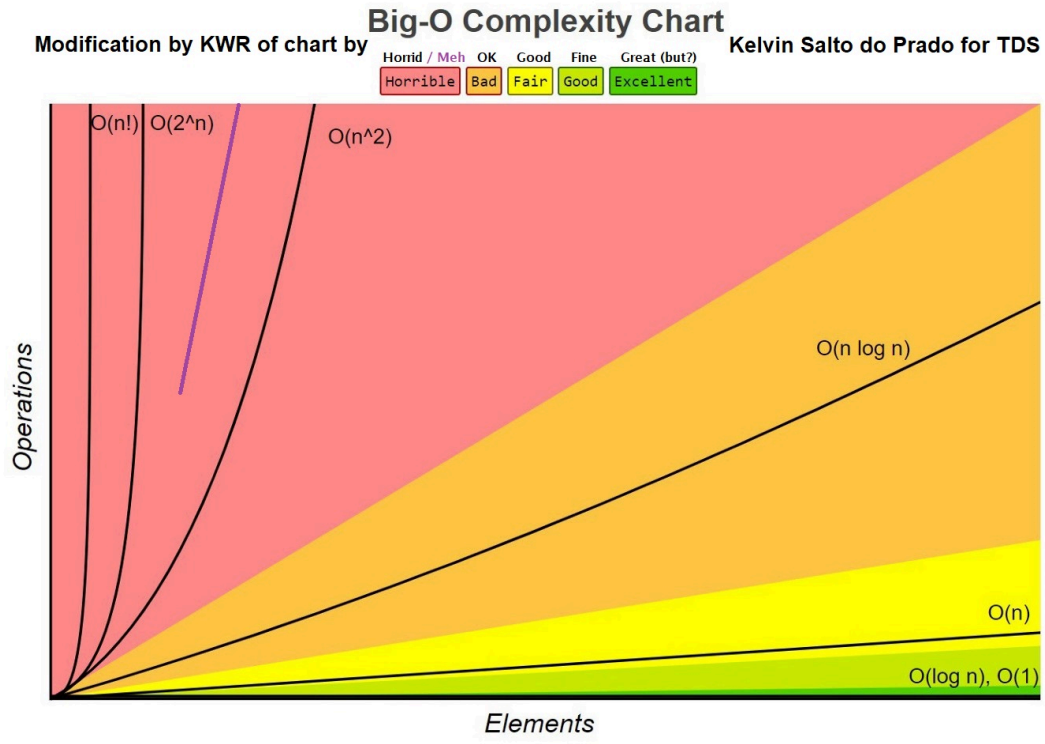
Here c is called "the constant in the O" and should be estimated and minimized as well, even though "$t = O(g)$" does not depend on it. Having n_0 be not excessive is also important. (Often we think of "c" as being ≥ 1.)

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.]

Reading for next week: Chapter 3 in https://cse.buffalo.edu/~regan/cse439/LRQMIT2Echs1-3.pdf and also up to section 14.3 of https://cse.buffalo.edu/~regan/cse439/LRQmitbook2pp131-147.pdf