Quantum Information IV

Lecture notes for 2026 IF Summer School
The Institute for Fundamental Study (IF)
Naresuan University, Phitsanulok, Thailand

Ninnat Dangniam
ninnatd@nu.ac.th

version:May 5, 2026

Contents

The symbol (✍) indicates that you should try to verify the preceding statement as an exercise.

4 Quantum Computer Science

4.1 Notations

We begin by fixing some notation that will be used throughout.

- A finite set with N elements is written as
$$[N] = \{1, 2, \ldots, N\}.$$
- A bit string is denoted by a bold lower-case letter $\mathbf{x} = x_1 x_2 \ldots x_n$, where each $x_j \in \{0, 1\}$ and $j \in [n]$.
- The set of all n-bitstrings is denoted by $\{0, 1\}^n$.
- The set of all finite bistrings (including the empty string) is denoted by $\{0, 1\}^*$.
- All logarithms are base 2 unless stated otherwise.

4.2 Introduction to algorithmic complexity

An algorithm is any procedure that follows a precise, step-by-step set of instructions. No room for interpretation or creativity! Why do we care about such rigid procedures? Because they allow us to make guarantees about resources used to solve problems. Efficiency of an algorithm is measured by how the required resources grow as the input size increases. The resources could be time, memory, or the quality of approximation. Here we focus on **time**.

- Usually what counts as the input size is clear. For example, for graph problems with n vertices and at most $\binom{n}{2} \approx n^2$ edges, n or n^2 is a reasonable measure of problem size as we may encode a graph as a binary, $n \times n$ adjacency matrix, which requires n^2 bits to store. In general, it makes sense to take the input size to be the time to read the input.
- By the same principle, for problems with integer input N, the size is not N itself but the number of bits to write N, which is
$$n = \log N.$$
Writing N in other bases $b > 2$ only changes the size by a usually unimportant constant factor:
$$\log_b N = \frac{\log N}{\log b}.$$
For example, in base 10, the number of digits is $\log N / \log 10 \approx 0.30 \log N$.[1]
The only encoding we avoid is unary (base 1), in which N is represented by exactly N symbols, making it exponentially more expensive than other encoding.

When trying to quantify runtime, we immediately run into a problem: different machines operate at different speeds. To avoid hardware-dependent measures like FLOPS (Floating point operations per second), we define a more robust notion of runtime using the big-O notation.

[1] This uses fewer digits compared to bits, but we have to pay for extra alternative symbols 2 to 9.

- An algorithm is said to run in $T(n)$ time if, for *every input* $x \in \{0,1\}^n$, it produces the correct output in at most $T(n)$ steps. The qualifier "for every input" means that this is a **worst-case** notion of runtime.

- For two positive functions f and g, we write $f = O(g)$ if there exists a a constant $c > 0$ and integer N such that and all $n \geq N$,

$$f(n) \leq cg(n).$$

  Equivalently,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty.$$

  By construction, the big-O notation **hides constant factors** and **small-n behavior**, see Fig. 1. As a result, big-O complexity only cares about the leading growth rate. For example, if f is $O(n^3 + 100n^2)$ then $f = O(n^3)$. (✍) These hidden factors can absolutely matter in practice. An algorithm with runtime $2^{10000}n$, while technically linear, would be beaten by a 2^n-time algorithm for $n < 10000$.

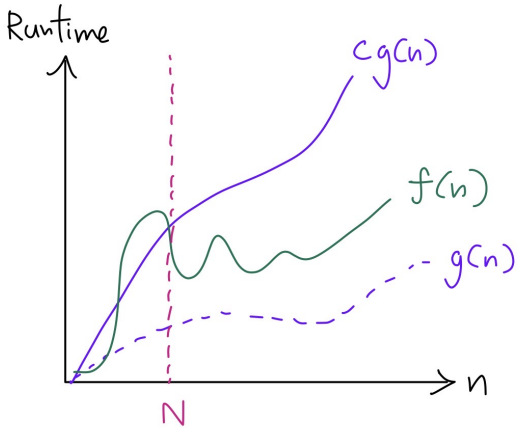


Figure 1: What it means for $f(n)$ to be $O(g(n))$: whenever $n \geq N$, $f(n)$ is upper bounded by $g(n)$ up to a constant factor. (Here $c > 1$.)

4.2.1 The importance of being polynomial

- An algorithm is considered **efficient** if it runs in **polynomial time**:

$$T(n) = O(n^c)$$

  for some constant $c \geq 0$. Why is this notion useful, you say? Again, an algorithm that runs in $O(n^{500})$-time would still count as a polynomial-time algorithm. But it is an empirical observation that most algorithms we consider tractable do not have such a big exponent.

- What's nice for theorists about this definition is that it makes efficient algorithms **composable** or **closed under subroutines**. Suppose an $O(n^c)$-time algorithm calls an $O(n^d)$-time algorithm as a subroutine, then the overall runtime is $O((n^d)^c) = O(n^{c+d})$, which is still polynomial.

□ The class of problems solvable in polynomial time is therefore the smallest class containing problems solvable in linear time (the most efficient ones) and closed under subroutines. The complexity class **P** consists of all **decision problems** that admit polynomial-time *classical and deterministic* algorithms. A decision problem asks whether an input object (such as a graph or a Boolean formula) satisfies a certain property. That is, given an input, we are required to answer YES = 1 or NO = 0.

For concreteness, let's consider a few representative problems.

1. **Integer multiplication:** Given p and q, compute pq.
2. **Integer factorization:** Given $N = pq$, find the prime factors p and q.

□ As stated, these are not decision problems, but this is not a serious limitation. For example, we can rephrase the factoring problem as follows: Given $N = pq$, decide whether there exists a prime factor less than a threshold k. If we can solve this decision version efficiently, we can search for a factor by performing a binary search over k in the range 2 to $\sqrt{N}$. This takes roughly $\log \sqrt{N} = O(n)$ tries. (✍)

□ The familiar "grade-school" algorithm multiplies numbers digit by digit before adding the results together. This costs n^2 multiplications, The cost of adding two n-bitstrings is linear, and we have to do that n times, so that adds another n^2 operations, making a total of $2n^2 = O(n^2)$ elementary arithmetic operations.

□ There are faster multiplication algorithms. For example, the Karatsuba algorithm reduces the number of multiplications by cleverly regrouping terms. We start by turning one multiplication of $2n$-bit numbers into four multiplications of n-bit integers by writing

$$pq = (2^n a + b)(2^n c + d) = 4^n ac + 2^n(ad + bc) + bd. \tag{1}$$

This still takes $4n^2$ multiplications. But observe that by rewriting

$$ad + bc = (a + b)(c + d) - ac - bd, \tag{2}$$

we only need to do three multiplications because we can reuse the products ac and bd. Doing this recursively leads to a runtime of $O(n^{\log 3}) \approx O(n^{1.585})$ [1, p.37, 2, p.55].

Even more advanced algorithms exist, many based on the idea of Fourier transform [1, p.37].

□ In contrast, no fast classical algorithm is known for factoring. A brute-force algorithm would be to check all the primes up to $\sqrt{N}$, which takes time exponential in the input size $n = \log N$. There are algorithms such as the general number field sieve with sub-exponential runtime of $\exp(O(n^\alpha))$ where heuristically $\alpha = 1/3$ [3], or less heuristically but still not fully rigorous $\alpha = 1/2$ [4].

□ Thus, (the decision version of) multiplication is in **P**, while (the decision version of) factoring is believed to be outside **P**. So much so, in fact, that much of modern cryptography is built on this assumption. They provide a striking example of two problems that are inverses of each other yet appear to have very different difficulty, at least on classical computers.

4.2.2 One problem to rule them all

Consider another problem:

3. **Satisfiability (SAT):** Given an efficiently computable[2] Boolean function $f : \{0,1\}^n \to \{0,1\}$, find an input $\mathbf{x}$ such that $f(\mathbf{x}) = 1$.

□ **Example.** Given two binary variables x and y, find their value assignments such that

$$f(x, y) = (x \vee y) \wedge (\overline{x} \vee \overline{y}) = 1. \tag{3}$$

Verify that $f = 1$ whenever $x \neq y$. (✍)

□ Note that we are not asked to compute f. The function f is already assumed to be efficiently computable (for example, by a polynomial number of logical operations such as AND, OR, and NOT). Our task is instead to find an input in the pre-image of 1. It is not quite accurate to think of this as computing the inverse of f since f^{-1} is generally one-to-many. Still, this viewpoint serves as a useful intuition: we are trying to reverse a computation that is easy to perform.

□ The decision version of SAT asks whether there *exists* an assignment: $\mathbf{x}$ such that $f(\mathbf{x}) = 1$. In this formulation, the input is the Boolean function f, while any $\mathbf{x}$ satisfying $f(\mathbf{x}) = 1$ is said to be a **certificate**, since it lets us check that the answer is YES.

□ SAT is a natural representative of a broad class of decision problems where:

- we are given an input (for instance a graph),
- we are asked whether it satisfies a certain property (say, 3-colorability), and
- whenever the answer is YES, there exists a short certificate (such as a valid coloring) that can be verified efficiently.

The class of such decision problems is called **NP**.[3] It includes many well-known and practically important optimization problems such as traveling salesperson and various scheduling/routing problems.

□ There is an asymmetry here. If the answer is YES, there is a certificates that allows efficient verification. But if the answer is NO, there is no obvious shortcut to find that out. In general, we would need to check all $\mathbf{x} \in \{0,1\}^n$ to conclude that no satisfying assignment exists.

□ A common mistake is to think that all **NP** problems are believed to be hard. In fact, **P** is contained in **NP**, since any polynomial-time algorithm already gives an efficiently verifiable certificate, namely the computation itself.

□ SAT is not only believed to be hard, but it is also extremely expressive: a fast SAT solver would yield a fast algorithm for *all* NP problems [5, p.110 for a rigorous proof]. To have some intuition why, consider encoding factoring as a SAT instance. We can easily build a

[2] To be precise, the problem given here is (the search version of) Circuit-SAT.

[3] **NP** doesn't stand for "non-polynomial". Rather, the **N** stands for "non-deterministic", referring to a hypothetical machine that could solve all **NP** problems efficiently.

Boolean circuit that multiplies two numbers (see Fig. 2), and then constrain the output to equal a desired value. For example, writing 3×5 in binary:

$$11 \times 101 = 1111,$$

we can take the AND of all output bits so that $f(\mathbf{x}) = 1$ if and only if $\mathbf{x}$ encodes the prime factors of 15.[4] In this way, solving the SAT instance amounts to 'inverting" multiplication [6].

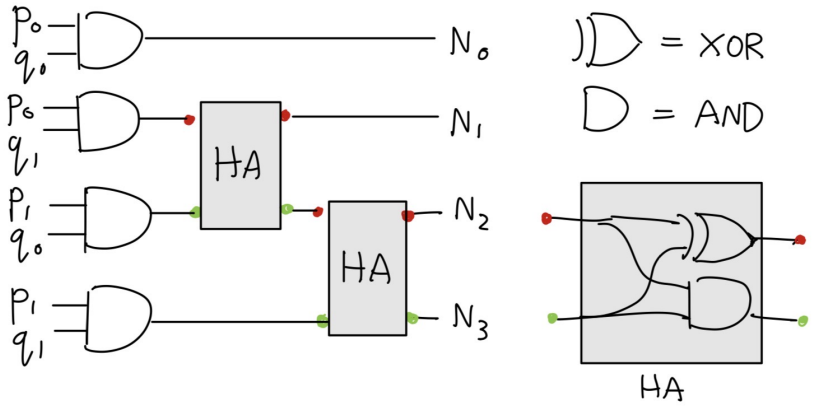


Figure 2: A circuit that multiples two two-bit numbers p and q using elementary logic gates. HA is a "half adder", a gadget that adds two bits and outputs a sum (red) and a carry (green). The output bits can be further constrained to produce 1 if and only if the product is equal to a desired value.

□ No fast classical algorithm is known for SAT. In fact, either finding such an algorithm or proving that none exists would resolve the **P** vs **NP** problem, one of the seven Clay Mathematics Institute's Millennium Prize Problems.

4.3 Quantum algorithms

We will work in the circuit model of quantum computation.

1. **Initialization.** Prepare n input registers and m ancillary registers in the state $|0\rangle^{\otimes(n+m)}$.
2. **Data loading.** Encode the classical input $\mathbf{x}$ into a quantum state $|\psi(\mathbf{x})\rangle$ on the input registers.
3. **Computation.** Apply a polynomial number of "elementary" unitary operations (typically acting on one or two qubits at a time).
4. **Uncomputation/cleanup.** Remove any unwanted entanglement with ancillary m' registers, leaving the n' output registers in a state that encodes the result ($n' + m' = n + m$).
5. **Readout.** Measure the n' output registers in the computational basis, obtaining an outcome $\mathbf{y}$ with probability $|\langle \mathbf{y}|\psi_{\text{out}}\rangle|^2$.

[4] Extra constraints are needed to exclude trivial factors, but these can be done.

4.3.1 Querying in superposition

- First, we need to know how to implement a Boolean function f as a unitary operation. A naive attempt would be to map
$$|\mathbf{x}\rangle \mapsto |f(\mathbf{x})\rangle$$
but this can't be unitary as many different inputs are mapped to the same output. The standard workaround is to keep the input register and write
$$\widehat{U}_f |\mathbf{x}\rangle |0\rangle = |\mathbf{x}\rangle |f(\mathbf{x})\rangle . \tag{4}$$
This is an example of a controlled-unitary operation with the first register serving as the **control** and the second as the **target**. To ensure that $\widehat{U}_f$ is unitary, we must define the action on all basis states to be
$$\widehat{U}_f |\mathbf{x}\rangle |a\rangle = |\mathbf{x}\rangle |a \oplus f(\mathbf{x})\rangle . \tag{5}$$
where $\oplus$ denotes addition modulo 2. This encodes the value of $f(\mathbf{x})$ into the *amplitudes* of the ancilla qubit.

- We can convert this amplitude encoding into a *phase encoding* using a simple quantum trick known as **phase kickback**. Rewriting (5) as (✍)
$$\widehat{U}_f |\mathbf{x}\rangle |a\rangle = |\mathbf{x}\rangle \otimes \left(\widehat{X}^{f(\mathbf{x})} |a\rangle\right), \tag{6}$$
where $\widehat{X}$ is the Pauli-X operator, we now initialize the ancilla in an eigenstate of $\widehat{X}$, namely $|-\rangle$. Then
$$\widehat{U}_f |\mathbf{x}\rangle |-\rangle = |\mathbf{x}\rangle \otimes \left(\widehat{X}^{f(\mathbf{x})} |-\rangle\right) = (-1)^{f(\mathbf{x})} |\mathbf{x}\rangle |-\rangle \tag{7}$$
The value of $f(\mathbf{x})$ is encoded in a global phase, which is unphysical. However, quantum mechanics allows us to make a query on a superposition of inputs!
$$\widehat{U}_f \frac{|\mathbf{x}\rangle + |\mathbf{y}\rangle}{\sqrt{2}} |-\rangle = \frac{(-1)^{f(\mathbf{x})} |\mathbf{x}\rangle + (-1)^{f(\mathbf{y})} |\mathbf{y}\rangle}{\sqrt{2}} |-\rangle \tag{8}$$
The values of $f(\mathbf{x})$ now appear in relative phases which are measurable.

- This trick was used by David Deutsch to demonstrate the first, albeit toy, quantum computational advantage [7]. Since the ancilla remains unentangled, we don't have to do the uncomputation/clean up part. Then,
$$\frac{(-1)^{f(\mathbf{x})} |\mathbf{x}\rangle + (-1)^{f(\mathbf{y})} |\mathbf{y}\rangle}{\sqrt{2}} = (-1)^{f(\mathbf{x})} \left(\frac{|\mathbf{x}\rangle + (-1)^{f(\mathbf{x})\oplus f(\mathbf{y})} |\mathbf{y}\rangle}{\sqrt{2}}\right). \tag{9}$$
We obtain information about the difference $f(\mathbf{x}) \oplus f(\mathbf{y})$ with a single query to f, whereas classically we would need two.

4.3.2 Solving two-bit SAT

Let's try out these new quantum tricks on SAT. Let $N = 2^n$, and suppose for simplicity that there is only one solution string $\mathbf{s} \in \{0,1\}^n$ satisfying $f(\mathbf{s}) = 1$.

□ How to prepare the equal superposition state of all inputs? This is easy since it is simply the product state obtained by applying Hadamard gates to the state $|0\rangle^{\otimes n}$ in parallel:

$$|+^n\rangle := \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}\in\{0,1\}^n} |\mathbf{x}\rangle = |+\rangle^{\otimes n} = \widehat{H}^{\otimes n} |0\rangle^{\otimes n} . \tag{10}$$

□ This multi-qubit **Walsh-Hadamard transform** is defined by its action on the basis states as

$$\widehat{H}^{\otimes n} |\mathbf{x}\rangle = \bigotimes_{j=1}^{n} \left(\widehat{H} |x_j\rangle\right) = \frac{1}{\sqrt{2^n}} \bigotimes_{j=1}^{n} (-1)^{k_j x_j} |k_j\rangle \tag{11}$$

$$= \frac{1}{\sqrt{2^n}} \sum_{\mathbf{k}\in\{0,1\}^n} (-1)^{k_1 x_1 + \cdots + k_n x_n} |\mathbf{k}\rangle \tag{12}$$

$$= \frac{1}{\sqrt{2^n}} \sum_{\mathbf{k}} (-1)^{\mathbf{k}\cdot\mathbf{x}} |\mathbf{k}\rangle . \tag{13}$$

□ If we were to apply U_f to the state $|+^n\rangle$ without phase kickback, we would obtain

$$\widehat{U}_f |+^n\rangle |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} |\mathbf{x}\rangle |f(\mathbf{x})\rangle = \frac{1}{\sqrt{2^n}} \left(\sum_{\mathbf{x}\neq\mathbf{s}} |\mathbf{x}\rangle |0\rangle + |\mathbf{s}\rangle |1\rangle \right) . \tag{14}$$

Immediately measuring the second register would give nothing useful, as the chance to find the solution $\mathbf{s}$ decreases exponentially as n increases.

□ With phase kickback, we end up with the state

$$\frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} (-1)^{\delta_{\mathbf{x},\mathbf{s}}} |\mathbf{x}\rangle .$$

It is not obvious what to do next, so let's specialize to the simplest non-trivial case $n = 2$:

$$\frac{1}{2} \sum_{\mathbf{x}} (-1)^{\delta_{\mathbf{x},\mathbf{s}}} |\mathbf{x}\rangle = |+^2\rangle - |\mathbf{s}\rangle \tag{15}$$

In the X-basis,

$$\widehat{H}^{\otimes 2}(|\mathbf{s}\rangle - |+^2\rangle) = \frac{1}{2} \sum_{\mathbf{k}} (-1)^{\mathbf{k}\cdot\mathbf{s}} |\mathbf{k}\rangle - |0\rangle^{\otimes 2} . \tag{16}$$

Notice that the right hand side is almost the Walsh-Hadamard transform of the solution $|\mathbf{s}\rangle$ except that the term $|0\rangle^{\otimes 2}$ has a sign flip. So we define a unitary transformation (✍) that flips the sign back:

$$\widehat{F} = \hat{\mathbb{1}} - 2(|0\rangle\langle 0|)^{\otimes 2} \simeq \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{17}$$

And there we have it. We have solved two-bit SAT with a single query to f, whereas classically, we would need four! The algorithm is summarized in Fig. 3.

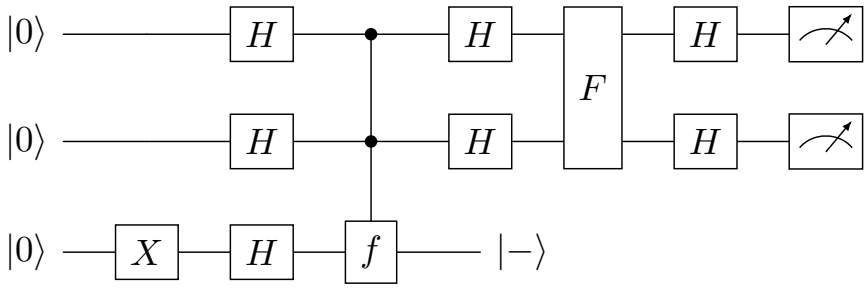


Figure 3: A summary of our quantum algorithm for two-bit SAT, expressed as a circuit diagram.

4.3.3 Grover's search

Does our algorithm generalize to SAT instances of arbitrary size? The answer is both yes and no.[5] A similar idea does extend to larger instances, but we no longer find the solution in a single query. This is the quantum search algorithm of Lov Grover [8].

- □ While Grover's algorithm is often described as unstructured database search, it doesn't really search through an explicitly stored list, but through an implicit list of possible inputs to a function much like in the SAT setting. If all you have is a list, then using Grover means you first need to implement such a function, but if this process itself requires looking up the entire list, taking $O(N)$ time, then Grover won't help.
- □ With phase kickback, the effective action of $\widehat{U}_f$ on the input register can be written as

$$\widehat{O}_f := \hat{\mathbb{1}} - 2\,|\mathbf{s}\rangle\langle\mathbf{s}|\,. \tag{18}$$

- □ The operator $-\widehat{H}^{\otimes n}\widehat{F}\widehat{H}^{\otimes n}$ generalizes to the n-qubit "diffusion operator"

$$\widehat{D} := 2\,|+^n\rangle\langle+^n| - \hat{\mathbb{1}}. \tag{19}$$

- □ From their projector forms, it is easy to see that (✍)
  - $\widehat{O}_f$ maps any state to the span of that state and $|\mathbf{s}\rangle$, and
  - $\widehat{D}$ maps any state to the span of that state and $|+^n\rangle$.

  So if we start from $|+^n\rangle$, repeated application of $\widehat{D}\widehat{O}_f$ keeps the state inside the two-dimensional subspace spanned by $|+^n\rangle$ and $|\mathbf{s}\rangle$ at all time. The operator $\widehat{D}\widehat{O}_f$ is called a **Grover iterate**.
- □ To understand what's going on, let's choose an ONB $\{|\mathbf{s}\rangle\,, |\mathbf{s}^\perp\rangle\}$, where

$$|\mathbf{s}^\perp\rangle = \frac{1}{\sqrt{N-1}} \sum_{\mathbf{x}\neq\mathbf{s}} |\mathbf{x}\rangle \tag{20}$$

[5] One might say the answer is

$$\frac{|\text{Yes}\rangle + |\text{No}\rangle}{\sqrt{2}}.$$

In this basis, the initial state can be written as

$$|+^n\rangle = \sqrt{\frac{N-1}{N}}\,|\mathbf{s}^\perp\rangle + \frac{1}{\sqrt{N}}\,|\mathbf{s}\rangle =: \cos\theta\,|\mathbf{s}^\perp\rangle + \sin\theta\,|\mathbf{s}\rangle \tag{21}$$

At this point, the overlap with the solution $|\mathbf{s}\rangle$ is still very small. What each Grover iterate does is gradually amplifying this component. How it does this is that geometrically $\widehat{O}_f$ acts as a reflection through $|\mathbf{s}^\perp\rangle$, while $\widehat{D}$ acts as a reflection through $|+^n\rangle$. The net effect of these two reflections is a rotation toward the solution $|\mathbf{s}\rangle$ by an angle 2θ, see Fig. 4. After T iterations, the state has rotated to an angle $(2T+1)\theta$ away from $|\mathbf{s}^\perp\rangle$. To land close to $|\mathbf{s}\rangle$, we therefore want

$$(2T+1)\theta \approx \frac{\pi}{2}.$$

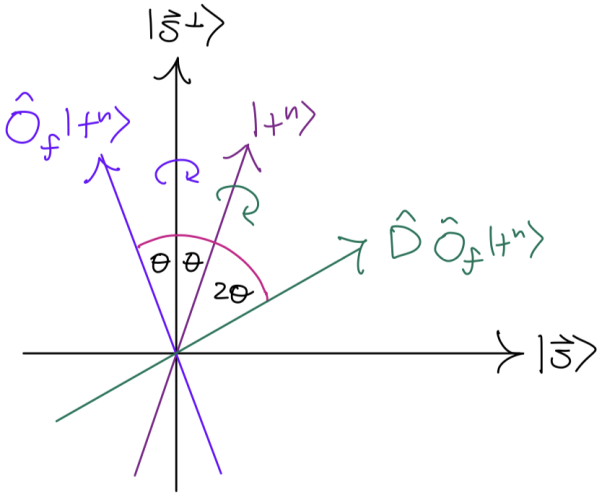


Figure 4: Geometric interpretation of each Grover iterate $\widehat{D}\widehat{O}_f$. The violet (resp. green) curved arrow indicates that $\widehat{O}_f$ (resp. $\widehat{D}$) acts as a reflection through $|\mathbf{s}^\perp\rangle$ (resp. $|+^n\rangle$).

□ Let's work out this number T asymptotically. From the above, we know that

$$T \approx \frac{\pi}{4\theta}$$

dor large N. Since $\sin\theta = \theta + O(\theta^3)$, we have that

$$\theta = \frac{1}{\sqrt{N}} - O(N^{-3/2}) \approx \frac{1}{\sqrt{N}}. \tag{22}$$

Hence,

$$T \approx \frac{\pi}{4}\sqrt{N}.$$

If we round T to the nearest integer, the final state is guaranteed to lie within an angle θ from $|\mathbf{s}\rangle$. Therefore,

$$\Pr(\mathbf{s}) \geq \cos^2\theta = 1 - O(\theta^2) = 1 - O(1/N) \tag{23}$$

which is overwhelmingly close to 1 for large N.

- The $\sqrt{N}$ scaling turns out to be the best one can do under the laws of quantum mechanics [1, p.874, 9, p.188]. Interestingly, this lower bound was proven before Grover discovered his algorithm [10].

Grover's algorithm has many variants.

- **Multiple solutions and unknown number of solutions [11].** Suppose there are M equally good solutions. We can run Grover in the exact same way, except that the target state is now the equal superposition of all solutions.

$$|M\rangle = \frac{1}{\sqrt{M}} \sum_{j=1}^{M} |\mathbf{s}_j\rangle \tag{24}$$

$$|+^n\rangle = \sqrt{\frac{N-M}{M}} |M^\perp\rangle + \frac{1}{\sqrt{M}} |M\rangle \tag{25}$$

  The same analysis goes through and Grover reaches success probability $\Pr(M) = 1 - O(M/N)$ after about $\pi/4\sqrt{N/M}$ iterates. Thus, Grover's algorithm becomes faster when there are more solutions, but with reduced success probability.

- What if $M \geq N/2$? In that case, a random guess already succeeds with probability at least $1/2$, so we might not really need Grover. Alternatively, we can add an extra qubit to double the search space and reduce the fraction of solutions.

- Even when the number of solutions is unknown, Grover's algorithm still succeeds with constant probability if we randomize the number of iterates $1 \leq T \leq \sqrt{N}$.

- **Amplitude amplification [12, 9, p.163].** Grover's algorithm is really a special case of a more general idea. Suppose we have an algorithm $\widehat{U}$, classical or quantum, with success probability p, and comes with a flag indicating when the "good" outcome occurs:

$$\widehat{U} |\psi\rangle = \sqrt{p} |\text{good}\rangle |\psi_1\rangle + \sqrt{1-p} |\text{bad}\rangle |\psi_0\rangle . \tag{26}$$

  Repeating until success takes $O(1/p)$ repetitions on average. But if we apply a Grover-like iterate, where now

$$\widehat{O}_f = (\hat{\mathbb{1}} - 2 |\text{good}\rangle\langle\text{good}|) \otimes \hat{\mathbb{1}}, \tag{27}$$

  and

$$\widehat{D} = \widehat{U}(2 |\psi\rangle\langle\psi| - \hat{\mathbb{1}})\widehat{U}^\dagger, \tag{28}$$

  the state will have large overlap with the good subspace after $O(1/\sqrt{p})$ iterations.

- **Fixed point search.** If we iterate more than $\pi\sqrt{N}/4$ steps, we start to overshoot the target and the success probability drops again. A relatively new design framework called *quantum signal processing* or *quantum singular value transformation* can be employed to construct an optimal fixed point Grover search/amplitude amplification that avoids this overshooting behavior while retaining the quadratic speedup [13, 14].

4.3.4 Simon's algorithm

Simon's problem was the first example to exhibit an exponential quantum speedup [15].

□ In Simon's problem, we are given a function $f : \{0,1\}^n \to A$ where $|A| = 2^{n-1}$ with the promise that

$$f(\mathbf{x}) = f(\mathbf{y}) \iff \mathbf{x} \oplus \mathbf{s} = \mathbf{y} \tag{29}$$

for some secret bitstring $\mathbf{s}$. The goal is to find $\mathbf{s}$, which is sometimes called a hidden *XOR shift*.

□ **Example.** Let $n = 3$, $A = \{R, G, B, O\}$, and $\mathbf{s} = 101$.

$$f(000) = f(101) = R \qquad\qquad f(001) = f(100) = G \tag{30}$$

and so on (see Fig. 5).

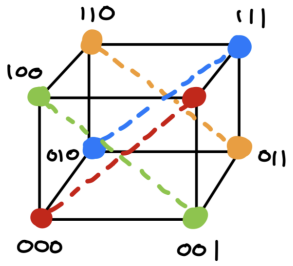


Figure 5: Illustration of Simon's problem with $n = 3$ and $\mathbf{s} = 101$. Two vertices have the same color if and only if they are connected by $\mathbf{s}$, represented by the diagonal shifts.

□ Classically, we would try to find a **collision**, two inputs $(\mathbf{x}, \mathbf{y})$ such that $f(\mathbf{x}) = f(\mathbf{y})$. How many queries does it take to see a collision? This is closely related to the famous birthday paradox: it takes only a room of 23 people to have more than 50% chance that at least two share a birthday. Why?

Consider the generalized birthday problem where with m people (input samples) and N days (outputs). There are $\binom{m}{2}$ input pairs, and each pair collides with probability $1/N$. What is the probability that at least one pair collides? The **union bound** says that for events $A_1, \ldots, A_k$, an upper bound for the probability that any one event happens is simply the sum

$$\Pr(\bigvee_i A_i) \leq \sum_i \Pr(A_i). \tag{31}$$

Therefore,

$$\Pr(\text{collision}) \leq \binom{m}{2} \frac{1}{N} \approx \frac{m^2}{2N}. \tag{32}$$

This number remains small until m is about $\sqrt{2N}$.

For $N = 365$, this estimates give $m \approx \sqrt{2N} \approx 27$. The reason we don't get 23 is that the union bound is very loose, but still we get the asymptotic right; finding a collision classically requires $O(\sqrt{2^n})$ queries for $N = 2^n$. Once a collision $\mathbf{x}$ and $\mathbf{y}$ is found, we compute $\mathbf{x} \oplus \mathbf{y} = \mathbf{x} \oplus \mathbf{x} \oplus \mathbf{s} = \mathbf{s}$.

□ Grover's algorithm lets us find a collision in $O(N^{1/3})$ queries [16]. For Simon's problem, though, we can do even better.

□ We again use quantum parallelism, but this time we won't need phase kickback; since we can prepare the state (✍)

$$\frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} |\mathbf{x}\rangle |f(\mathbf{x})\rangle ,$$

if we measure the second register, the state of the first register collapses to an equal superposition of two inputs that collide!

$$\frac{|\mathbf{x}\rangle + |\mathbf{x} \oplus \mathbf{s}\rangle}{\sqrt{2}}$$

However, we can't naively add the two terms to recover $|\mathbf{x} \oplus \mathbf{x} \oplus \mathbf{s}\rangle = |\mathbf{s}\rangle$, nor does measuring in the computational basis reveal anything useful.

□ Instead, what we can do is applying the Walsh-Hadamard transform once more. Recall its action on the basis states:

$$\widehat{H}^{\otimes n} |\mathbf{x}\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{k}} (-1)^{\mathbf{k}\cdot\mathbf{x}} |\mathbf{k}\rangle . \tag{33}$$

For a general superposition, it's helpful to think in terms of how the amplitudes transform. If

$$|\psi\rangle = \sum_{\mathbf{x}} a_{\mathbf{x}} |\mathbf{x}\rangle = \sum_{\mathbf{k}} \underbrace{\sum_{\mathbf{x}} a_{\mathbf{x}} \langle \mathbf{k}|\mathbf{x}\rangle}_{\tilde{a}_{\mathbf{k}}} |\mathbf{k}\rangle , \tag{34}$$

then after applying $\widehat{H}^{\otimes n}$, the amplitude in the k-basis becomes

$$\tilde{a}_{\mathbf{k}} = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}} a_{\mathbf{x}} (-1)^{\mathbf{k}\cdot\mathbf{x}}. \tag{35}$$

For our special case of the superposition of colliding inputs, this is

$$\tilde{a}_{\mathbf{k}} = \frac{1}{\sqrt{2^n}} \frac{(-1)^{\mathbf{k}\cdot\mathbf{x}} + (-1)^{\mathbf{k}\cdot(\mathbf{x}\oplus\mathbf{s})}}{\sqrt{2}} \tag{36}$$

$$= (-1)^{\mathbf{k}\cdot\mathbf{x}} \frac{1}{\sqrt{2^{n-1}}} \left(\frac{1 + (-1)^{\mathbf{k}\cdot\mathbf{s}}}{2} \right) \tag{37}$$

Therefore, the probability is concentrated entirely on bitstrings $\mathbf{k}$ orthogonal to $\mathbf{s}$ modulo 2:

$$\Pr(\mathbf{k}) = |\tilde{a}_{\mathbf{k}}|^2 = \begin{cases} 1/2^{n-1}, & \text{if } \mathbf{k} \perp \mathbf{s}, \\ 0, & \text{otherwise} \end{cases} \tag{38}$$

□ By repeating the algorithm $t = O(n)$ times, we obtain a homogeneous system of linear equations

$$\begin{aligned} \mathbf{k}_1 \cdot \mathbf{s} &\equiv 0 \pmod 2, \\ \mathbf{k}_2 \cdot \mathbf{s} &\equiv 0 \pmod 2, \\ &\vdots \\ \mathbf{k}_t \cdot \mathbf{s} &\equiv 0 \pmod 2 \end{aligned}$$

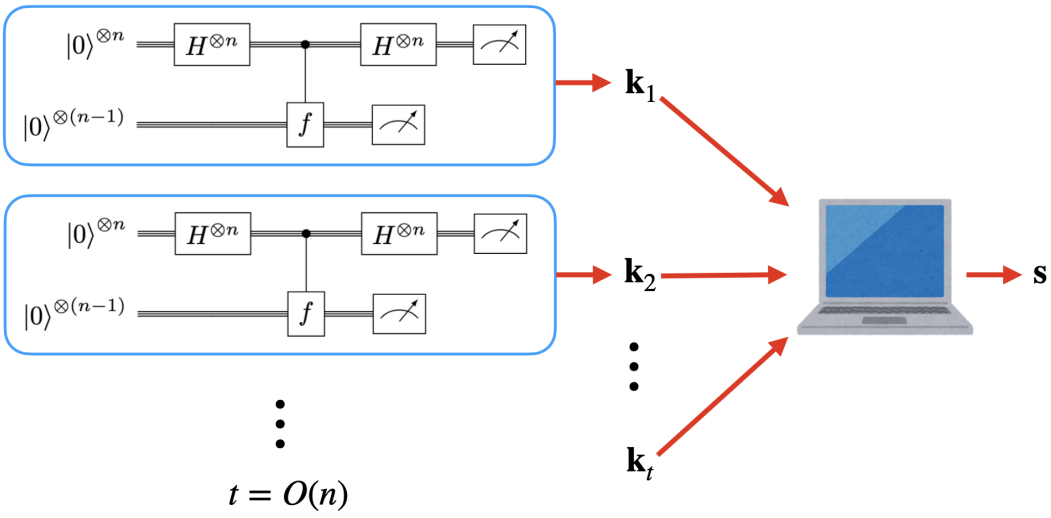


Figure 6: Summary of Simon's algorithm. Each query to f gives a sample $\mathbf{k}$ orthogonal to $\mathbf{s}$. After $t = O(n)$ samples, we can (with high probability) recover the unique $\mathbf{s}$ using a classical algorithm.

that, with high probability, uniquely determines $\mathbf{s}$.[6] For example, for $\mathbf{s} = 101$, we would only find $\mathbf{k} = 000, 010, 101$, or 101. (Note that $\mathbf{s} \cdot \mathbf{s} \equiv 0 \mod 2$.)

$\mathbf{k}$	000	001	010	011	100	101	110	111
$\mathbf{k} \cdot \mathbf{s}$	0	1	0	1	1	0	1	0

We can solve this system of equations efficiently on a classical computer to recover the secret bitstring $\mathbf{s}$. Overall, this solves Simon's problem using $O(n)$ queries, compared to $O(\sqrt{2^n})$ queries required classically. A summary of Simon's algorithm is depicted in Fig. 6.

□ The exponential speedup for Simon's problem is a **black-box speedup**. Unlike in the Grover's setting where a function f is assumed to be efficiently implementable, here f is only abstractly specified with the promise of the hidden XOR shift structure. We don't know how to implement such an f efficiently in general, so the exponential speedup doesn't directly translate into a real-world speedup.

4.3.5 Generalizing to period finding

Recall that in Simon's problem, we are given a function $f : \{0,1\}^n \to A$, where $|A| = 2^{n-1}$, with the promise that it is two-to-one, and the only collisions come from a hidden XOR shift $\mathbf{s}$:

$$f(\mathbf{x}) = f(\mathbf{y}) \iff \mathbf{x} \oplus \mathbf{s} = \mathbf{y}$$

We can think of $\mathbf{s}$ as a "period" of f except the shift is defined using XOR instead of ordinary addition. It turns out that the same ideas can be adapted to find a hidden shift of a function on

[6] More precisely, with $n + \log n$ samples, the probability that the samples span the full subspace orthogonal to $\mathbf{s}$ is $1 - 1/n$ [1, p. 893].

integers mod N, leading to a quantum algorithm for period finding. In this case, the hidden shift is with respect to addition modulo N, and the role of the Walsh-Hadamard transform is played by the **Quantum Fourier Transform (QFT)**.

□ Given a function $f(x)$ on $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$, its Fourier transform is

$$\tilde{f}(k) = \frac{1}{\sqrt{N}} \sum_x f(x)\, \omega^{kx}, \quad \text{where } \omega = e^{2\pi i/N}. \tag{39}$$

The inverse transform is

$$f(x) = \frac{1}{\sqrt{N}} \sum_k \tilde{f}(k)\, \omega^{-kx}. \tag{40}$$

□ In linear algebra language, the Fourier transform is just a unitary change of basis.

$$|\psi\rangle = \sum_x a_x |x\rangle \overset{\widehat{U}_{QFT}}{\longleftrightarrow} |\tilde{\psi}\rangle = \sum_k \tilde{a}_k |k\rangle, \tag{41}$$

where

$$\tilde{a}_k = \frac{1}{\sqrt{N}} \sum_x a_x\, \omega^{kx}. \tag{42}$$

In particular, the unitary operator (✍) $\widehat{U}_{QFT}$ that implements the QFT is defined by the action on basis states $\{|0\rangle, |1\rangle, \ldots, |N-1\rangle\}$ as

$$\widehat{U}_{QFT} |x\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{kx} |k\rangle. \tag{43}$$

or in the matrix form

$$\widehat{U}_{QFT} \simeq \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)^2} \end{pmatrix}. \tag{44}$$

For $N = 2^n$, QFT can be implemented using $O(n^2)$ single-qubit and controlled-unitary operations [1, p.859].[7]

□ It will be useful when summing up the amplitudes to recall the geometric series:

$$\sum_{k=0}^{N-1} b^k = \frac{1 - b^N}{1 - b}. \tag{45}$$

In particular, if $\omega = e^{2\pi i/N}$,

$$\sum_{k=0}^{N-1} (\omega^j)^k = \begin{cases} N & \text{if } j \equiv 0 \pmod N, \\ 0 & \text{otherwise.} \end{cases} \tag{46}$$

[7] QFT can also be implemented efficiently for N not a power of 2 [17].

□ Suppose $f : \mathbb{Z}_N \to A$ has a period r:

$$f(x + r \pmod N) = f(x), \tag{47}$$

and otherwise all values within one period are distinct.

□ **Example.** Let $N = 8$ and $r = 4$.

x	0	1	2	3	4	5	6	7	0
$f(x)$	a	b	c	d	a	b	c	d	a

□ First notice that $f(x + N) = f(x)$ automatically, so any period r must divide N. In the special case where $N = 2^n$, this actually makes the problem classically easy: we can simply compare $f(x)$ with $f(x+2), f(x+2^2), f(x+2^3), \ldots$ until we see a repeat. This takes at most $n = \log N$ queries.

Having said that, we'll still study this setting because it sets up the more interesting case where we instead sample x from a larger range $\{0, \ldots, M-1\}$ with $M = 2^m > N$. In that situation, the period r need not divide M, and the problem becomes genuinely nontrivial.

□ The period finding algorithm begins in the same way as in Simon's algorithm: prepare a uniform superposition

$$\frac{1}{\sqrt{N}} \sum_x |x\rangle \, |f(x)\rangle \, ,$$

and measure the second register. This collapses the first register to a superposition of inputs that all map to the same value:

$$\sqrt{\frac{r}{N}} \sum_{t=0}^{N/r-1} |x_0 + rt\rangle \, .$$

The amplitudes are nonzero on exactly N/r evenly spaced basis states, separated by r. We can think of this as a (finite) Dirac comb:

$$\text{Ш}_r(x) = \sum_{t=0}^{N/r-1} \delta_{x,\, x_0 + rt}, \tag{48}$$

for some offset x_0 set by the measurement outcome of the second register.

□ Again, we can't "reach in" the quantum state and read off the spacing r directly. Measuring in the computational basis just gives us one of these peaks, and there's no obvious way to tell whether the spacing we see actually corresponds to the smallest period r. So we need to do something more.

□ The key is to apply the Fourier transform. A Dirac comb turns into another Dirac comb, but now with reciprocal spacing (see Fig. 7):

$$\text{Ш}_r(x) \;\mapsto\; \frac{N}{r} \sum_{q=0}^{r-1} \delta_{k,\, qN/r}. \tag{49}$$

Let's see this explicitly. Starting from the amplitudes,

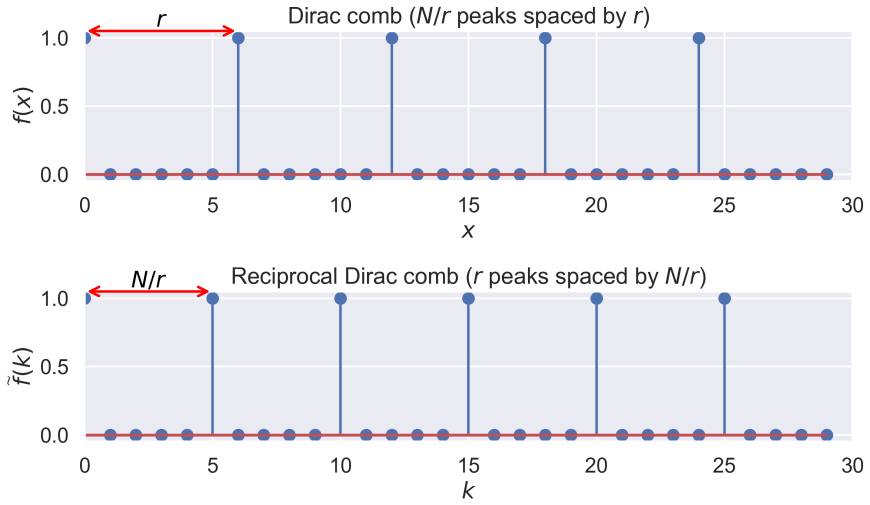


Figure 7: Dirac comb and its Fourier transform for $r = 6$ and $N = 30$

$$\tilde{a}(k) = \frac{1}{\sqrt{N}} \sum_x a_x \, \omega^{kx} \tag{50}$$

$$= \frac{\sqrt{r}}{N} \sum_x \delta_{x,\, x_0 + rt} \, \omega^{kx} \tag{51}$$

$$= \frac{\sqrt{r}}{N} \sum_{t=0}^{N/r-1} \omega^{k(x_0+rt)} \tag{52}$$

$$= e^{2\pi i k x_0/N} \cdot \frac{\sqrt{r}}{N} \sum_{t=0}^{N/r-1} \omega^{krt}. \tag{53}$$

The sum is a geometric series, which evaluates to

$$\tilde{a}(k) = \begin{cases} \dfrac{1}{\sqrt{r}} \, e^{2\pi i k x_0/N} & \text{if } kr \equiv 0 \pmod{N} \\ 0 & \text{otherwise.} \end{cases} \tag{54}$$

□ So, up to a global phase, the quantum state becomes

$$\frac{1}{\sqrt{r}} \sum_{q=0}^{r-1} \left| q \cdot \frac{N}{r} \right\rangle. \tag{55}$$

Instead of being hidden in the spacing between basis states, the period r now shows up in the *locations* of the peaks themselves, which are multiples of N/r. That's something we can actually access by measurement. In particular, as soon as we obtain two samples

$$k_1 = \frac{q_1 N}{r}, \quad k_2 = \frac{q_2 N}{r}.$$

whose q_1 and q_2 are coprime,[8] computing

$$\gcd(k_1, k_2) = \frac{N}{r},$$

[8]Two integers are *coprime* if $\gcd(q_1, q_2) = 1$. gcd stands for the greatest common divisor, which is efficiently computable using Euclid's algorithm.

immediately gives us the period r.

- How likely is this to happen? It turns out the probability to find such a pair of outcomes is constant, and we can give a rough estimate (not rigorous) as follows . A given integer c divides a random integer with probability $1/c$ (for example, half the integers are divisible by 2). So for a prime p, the probability that p divides both q_1 and q_2 is $1/p^2$. Therefore, q_1 and q_2 are coprime exactly when no prime divides both of them. Assuming independence across primes, the probability is

$$\prod_{p \text{ prime}} \left(1 - \frac{1}{p^2}\right) = \frac{1}{\zeta(2)} = \frac{6}{\pi^2} \approx 0.607, \tag{56}$$

where ζ is the Riemann zeta function. So we are likely to succeed after a constant number of repetitions.

Now we briefly go over a trickier variant when we **don't know the size of the input domain** N.

- Classically, this makes the problem much harder. Quantum mechanically, we begin by picking some $M = 2^m$ and perform the QFT over $\mathbb{Z}_M$. In general, M won't be divisible by r. So the number of peaks, which we denote by ℓ, is no longer exactly M/r, but instead either $\lfloor M/r \rfloor$ or $\lceil M/r \rceil$, depending on which value of f we observe in the second register (the offset x_0).

- **Example.** If $M = 30$ and $r = 10$, then $M/r = 3$, and we always see exactly 3 peaks. But if $r = 12$, then $M/r \approx 2.7$, so we might see either 2 or 3 peaks depending on the shift. For instance, we could get $x = 0, 12, 24$ (three peaks), or $x = 6, 18$ (two peaks).

- After applying the QFT, the peaks in the probability distribution are no longer perfectly sharp: instead of complete destructive interference away from multiples of M/r, we now get broadened peaks with small oscillations around them (see Fig. 8). It turns out that to recover the period r from these "blurry" peaks, it suffices to choose $M > r^2$ [1, p.857]. Under this condition, the recovery can be done efficiently based on a classical method of continued-fraction expansion using Euclid's algorithm.

4.3.6 From period finding to factoring

With period finding at hand, Shor's factoring algorithm is only a few steps away. The main idea is that factoring reduces to *order finding*, which is simply period finding applied to a specific function.

- **Order finding.** Given integers a and N with $\gcd(a, N) = 1$, the task is to fine the *order* of a modulo N, defined as the smallest positive integer r such that

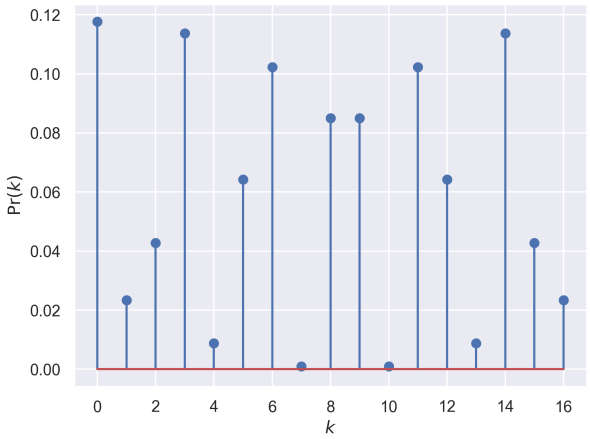
$$a^r \equiv 1 \pmod{N}. \tag{57}$$

This is precisely a period-finding problem for the function $f(x) = a^x \bmod N$.

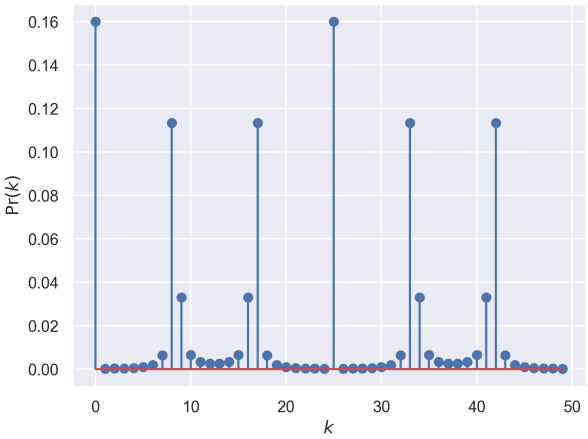
Note that the maximum possible period of this function (the domain size of f) is not N but the so-called Euler totient function $\varphi(N)$ according to Fermat's little theorem:

$$a^{\varphi(N)} \equiv 1 \mod N, \tag{58}$$

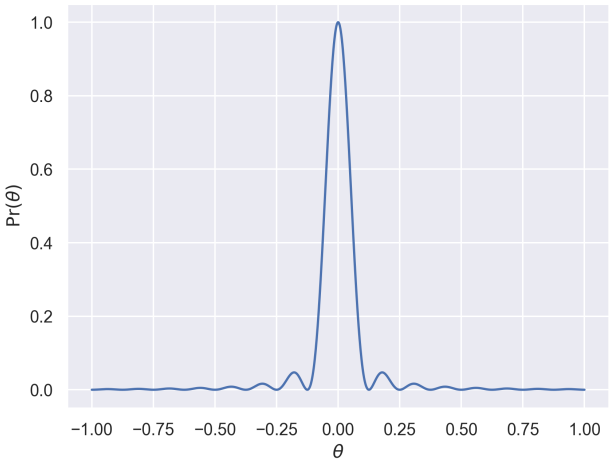
but $\varphi(n)$ contains the same information as prime factors of N which we don't know! Fortunately, as we have seen in the period-finding algorithm, we only need to choose $M > r^2$. Since $\varphi(N) \leq N$, taking $M > N^2$ guarantees that this condition is satisfied.



(a) Exact distribution ($M = 17 < r^2$)



(b) Exact distribution ($M = 50 > r^2$)



(c) Continuous sinc^2 approximation ($M = 50$)

Figure 8: Outcome distributions for period finding with period $r = 6$ and $\ell = \lfloor M/r \rfloor$ always rounded down for (my) convenience. The top row shows the exact probability distributions for small and large M, illustrating how the peaks sharpen as M increases. The bottom plot shows the continuous sinc^2 approximation of the shape of each peak, where $\theta := kr/M - q$. In the continuous approximation (derived in Appendix A), all peaks have equal height. In contrast, in the discrete distribution, the peak heights depend on how close k is to a multiple of M/r, with larger deviations resulting in lower peaks.

□ How does knowing the order helps us factor N? The trick is to use r to construct a *nontrivial* square root of 1 modulo N. This is, an integer b such that

$$b^2 \equiv 1 \pmod{N}, \qquad b \not\equiv \pm 1 \pmod{N}. \tag{59}$$

Given such a b, we have

$$(b-1)(b+1) \equiv 0 \pmod{N}. \tag{60}$$

Since $b \not\equiv \pm 1 \pmod{N}$, neither factor is divisible by N, but their product is. This implies that each prime factor of N must divide each of $b-1$ and $b+1$ separately, so

$$\gcd(b-1, N), \qquad \gcd(b+1, N) \tag{61}$$

are nontrivial factors of N.

□ A factoring algorithm would go like this: pick a random integer $a < N$ with $\gcd(a, N) = 1$, and compute its order r. If r is even, define

$$b \coloneqq a^{r/2} \bmod N. \tag{62}$$

Then $b^2 \equiv a^r \equiv 1 \pmod{N}$. If moreover $b \not\equiv \pm 1 \pmod{N}$, then b is a nontrivial square root of 1, and we can extract a factor as above. It can be shown that with probability at least $1/2$, a random choice of a yields an even r and a nontrivial b.[9] So we can simply repeat the algorithms a few times with different candidates a.

□ **Example.** Let $N = 21$ and choose $a = 2$. First check that $\gcd(2, 21) = 1$. By repeated squaring,

x	0	1	2	3	4	5	6	7	8	9	10	11
$2^x \bmod 21$	1	2	4	8	16	11	1	2	4	8	16	11

We see that the order is $r = 6$. Since r is even, compute

$$b = 2^{r/2} = 2^3 = 8.$$

Then $b^2 = 64 \equiv 1 \pmod{21}$, and $b \not\equiv \pm 1 \pmod{21}$. Hence

$$\gcd(8-1, 21) = \gcd(7, 21) = 7, \qquad \gcd(8+1, 21) = \gcd(9, 21) = 3,$$

so $21 = 3 \times 7$.

□ Classically, the only slow step is finding the order of a. That's where Shor's order-finding algorithm comes in. A summary of the quantum factoring algorithm is given in Fig. 9.

[9] If N is an odd prime power, such r may not exists, but this case can be handled separately.

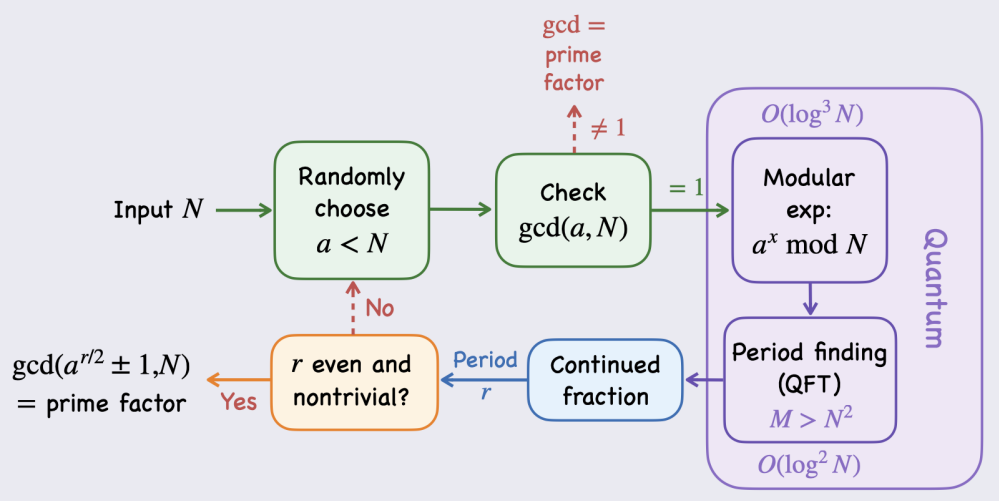


Figure 9: Summary of Shor's factoring algorithm

4.3.7 Impact on modern cryptography

□ Much of modern (public-key) cryptography such as RSA and elliptic curve cryptography relies on the assumed hardness of factoring and the discrete logarithm problem, both of which are efficiently solved by Shor's algorithm [18–20]. Why does this matter? Because these are exactly the tools used to secure things like HTTPS, digital signatures, and online banking.

□ Given the steady progress in quantum hardware and error correction [21, 22], relying on these cryptographic systems is becoming increasingly risky. This is why there is a global push toward *post-quantum cryptography*, that is, classical cryptographic schemes that are believed to remain secure even against quantum attacks.

A particular subtle threat is that someone with malicious intent doesn't need a quantum computer *today*. It is possible to record encrypted data now and decrypt it later once quantum computers become powerful enough—a strategy known as "harvest now, decrypt later."

□ Another approach is to use quantum mechanics itself to protect information. In *quantum key distribution* (QKD), two parties create a shared secret key by sending quantum states, typically single-photons states, and If someone tries to listen in, they would have to make a measurement, which generally disturbs the states, leaving a detectable trace. In this way, the security comes directly from the laws of physics, rather than from assumptions about computational difficulty.

A Approximate period-finding distribution when M is not divisible by r.

To understand the shape of the peaks, we start from the exact expression

$$\Pr(k) = \frac{1}{M\ell}\left|\sum_{t=0}^{\ell-1} e^{2\pi ikrt/M}\right|^2 = \frac{1}{M\ell}\left|\frac{1-e^{2\pi ikr\ell/M}}{1-e^{2\pi ikr/M}}\right|^2. \tag{63}$$

Applying the identity

$$\left|1-e^{ix}\right| = 2|\sin(x/2)|. \tag{64}$$

to both numerator and denominator gives

$$\Pr(k) = \frac{1}{M\ell}\left(\frac{\sin(\pi kr\ell/M)}{\sin(\pi kr/M)}\right)^2. \tag{65}$$

To approximate the probabilities near the peaks qM/r, we write

$$k = \frac{qM}{r} + \delta \tag{66}$$

for small δ, making the numerator

$$\sin(\pi kr\ell/M) = \sin(\pi q\ell + \pi\delta r\ell/M) \approx \sin(\pi\delta r\ell/M)\,. \tag{67}$$

Similarly, the denominator becomes

$$\sin(\pi kr/M) \approx \sin(\pi\delta r/M) \approx \pi\delta r/M, \tag{68}$$

where we have used $\sin x \approx x$ so that the combined expression becomes the sinc function:

$$\Pr(k) \;\approx\; \left(\frac{\sin(\pi\delta r\ell/M)}{\pi\delta r\ell/M}\right)^2. \tag{69}$$

To further simplify, write

$$\theta := \frac{\delta r}{M} = \frac{kr}{M} - q, \tag{70}$$

so that

$$\Pr(k) \approx \left(\frac{\sin(\pi\ell\theta)}{\pi\ell\theta}\right)^2. \tag{71}$$

The width of the sinc function can be measured by the distance between its first zero crossings, $2/\ell$, so larger ℓ (that is, larger M) gives sharper peaks.

References

[1] Cristopher Moore and Stephan Mertens. *The nature of computation*. Oxford University Press, 2011.

[2] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani. *Algorithms*. McGraw-Hill Higher Education, 2006. ISBN: 9780077388492.

[3] Arjen K Lenstra and Hendrik W Lenstra. *The development of the number field sieve*. Vol. 1554. Springer Science & Business Media, 1993.

[4] Hendrik W Lenstra and Carl Pomerance. "A rigorous time bound for factoring integers". *Journal of the American Mathematical Society* 5, pp. 483–516, 1992.

[5] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[6] Polylog. *What P vs NP is actually about*. 2024. URL: https://www.youtube.com/watch?v=6OPsH8PK7xM (visited on 04/19/2026).

[7] David Deutsch. "Quantum theory, the Church–Turing principle and the universal quantum computer". *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400, pp. 97–117, 1985.

[8] Lov K Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996. Pp. 212–219.

[9] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An introduction to quantum computing*. OUP Oxford, 2006.

[10] Charles H Bennett et al. "Strengths and weaknesses of quantum computing". *SIAM journal on Computing* 26, pp. 1510–1523, 1997.

[11] Michel Boyer et al. "Tight bounds on quantum searching". *Fortschritte der Physik: Progress of Physics* 46, pp. 493–505, 1998.

[12] Gilles Brassard et al. "Quantum Amplitude Amplification and Estimation". In: *Quantum Computation and Quantum Information*. Ed. by Samuel J. Lomonaco Jr. Vol. 305. AMS Contemporary Mathematics. American Mathematical Society, 2002. Pp. 53–74.

[13] Theodore J Yoder, Guang Hao Low, and Isaac L Chuang. "Fixed-point quantum search with an optimal number of queries". *Physical review letters* 113, p. 210501, 2014.

[14] John M Martyn et al. "Grand unification of quantum algorithms". *PRX quantum* 2, p. 040203, 2021.

[15] Daniel R Simon. "On the power of quantum computation". *SIAM journal on computing* 26, pp. 1474–1483, 1997.

[16] Gilles Brassard, Peter Høyer, and Alain Tapp. "Quantum Algorithm for the Collision Problem". In: *Proceedings of the Third Latin American Symposium on Theoretical Informatics (LATIN'98)*. Ed. by Cláudio L. Lucchesi and Arnaldo V. Moura. Vol. 1380. Lecture Notes in Computer Science. Springer, 1998. Pp. 163–169.

[17] Michele Mosca and Christof Zalka. "Exact quantum Fourier transforms and discrete logarithm algorithms". *International Journal of Quantum Information* 2, pp. 91–100, 2004.

[18] Peter W Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". *SIAM review* 41, pp. 303–332, 1999.

[19] Ryan Babbush et al. “Securing Elliptic Curve Cryptocurrencies against Quantum Vulnerabilities: Resource Estimates and Mitigations”. *arXiv preprint arXiv:2603.28846*, 2026.

[20] Madelyn Cain et al. “Shor’s algorithm is possible with as few as 10,000 reconfigurable atomic qubits”. *arXiv preprint arXiv:2603.28627*, 2026.

[21] Dolev Bluvstein et al. “Logical quantum processor based on reconfigurable atom arrays”. *Nature* 626, pp. 58–65, 2024.

[22] Google Quantum AI. “Quantum error correction below the surface code threshold”. *Nature* 638, pp. 920–926, 2025.