Sevag Gharibian

Lecture 10: Shor's quantum factoring algorithm

"Euclid taught me that without assumptions there is no proof. Therefore, in any argument, examine the assumptions."
— Eric Temple Bell.

Contents

1 Introduction

In the previous lecture, we introduced the Quantum Fourier Transform (QFT), and applied it to the Quantum Phase Estimation (QPE) problem. In this lecture, we show how to use QPE to derive a crown jewel of quantum computation — the quantum factoring algorithm. This algorithm, developed by Peter Shor in 1994, forced the classical computer science community to re-evaluate one of its core assumptions. Namely, in 1978, Rivest, Shamir, and Adleman published the public-key cryptosystem known as RSA, which to this day remains widely used in practice. And while the field had grown comfortable assuming RSA is secure, its Achilles heel was hiding in plain sight — the abstract of the original RSA paper itself reads:

"The security of the system rests in part on the difficulty of factoring the published divisor, n".

Indeed, the standard assumption leading into the early 1990's was that factoring is "hard". And this is precisely where Peter Shor made his mark — he showed that a polynomial-time quantum computer could factor a composite integer N into its prime factors, thus breaking RSA.

At a high level, the factoring algorithm proceeds as follows: First, the problem of *integer factorization* is reduced to the problem of *order finding*[1]. Then, the QPE algorithm is used to sample from a certain desired distribution. Finally, these samples are postprocessed using a classical algorithm known as *continued fractions* to solve the order finding problem. Before getting into these details, however, let us begin by formally defining the factoring problem and understanding its context.

2 The integer factorization problem

Recall that the Fundamental Theorem of Arithmetic states that any positive integer N has a prime factorization, i.e.

$$N = p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}, \tag{1}$$

[1]For clarity, this reduction was known prior to Shor's algorithm, due to the 1976 work of Miller [Mil76].

for distinct primes p_i and positive integers k_i, and moreover this decomposition is unique. Thus, intuitively the integer factorization problem asks one to, given N, output the prime factors $\{p_1, \ldots, p_m\}$.

Exercise 1. Consider the following brute force algorithm for finding a factor p_i: Iteratively try to divide N by each of the numbers in sequence $(2, 3, 4, \ldots, \lfloor\sqrt{N}\rfloor)$, and output the first divisor found. Clearly, this algorithm requires $O(\sqrt{N})$ field operations over $\mathbb{R}$. Why is this not a polynomial-time factoring algorithm? (Hint: What is the number of bits required to represent the input, N? Is an $O(\sqrt{N})$-time algorithm polynomial with respect to this number of bits?)

Exercise 2. Given the prime factors $\{p_1, \ldots, p_m\}$, show how to also compute the multiplicities $\{k_1, \ldots, k_m\}$ using polylog(N) field operations. Hint: Prove first that there are at most polylog(N) primes (including multiplicities) in the prime factorization of N.

A formal definition. Although the intuitive definition of the factoring problem is clear, recall that formally the complexity classes P and NP are sets of *decision* problems, meaning roughly that each possible input to the problem is either a YES or NO instance. Thus, we formally define factoring as follows.

Definition 3 (FACTOR)**.**

- *Input: A positive integer $N > 1$, and threshold $K \in \mathbb{Z}^+$.*
- *Output: Does N have a prime factor smaller than K?*

Exercise 4. Show how a subroutine for solving the decision problem FACTOR can be bootstrapped to efficiently compute the smallest non-trivial prime factor of N.

Complexity theoretic considerations. Interestingly, while FACTOR is not known to be in P, its close cousin, the primality testing problem, *is* in P. The latter is defined: *Given $N \in \mathbb{Z}^+$, is N composite (i.e. not prime)?* While this does not, as far as is known, allow us to also solve FACTOR, it does imply that FACTOR $\in$ NP $\cap$ co-NP.

Exercise 5. Why is FACTOR trivially in NP?

Exercise 6. Why is FACTOR in co-NP? (Hint: Use the Fundamental Theorem of Arithmetic.)

Exercise 7. Prove that if a language $L \in$ NP $\cap$ co-NP is NP-complete, then NP = co-NP.

Combining the exercises above, we conclude that if FACTOR is NP-complete, then NP = co-NP. The latter equality, however, is known to collapse the Polynomial-Time Hierarchy (PH) to its first level, a consequence widely believed almost as unlikely as P = NP. Thus, FACTOR is strongly expected not to be NP-complete.

NP-intermediate problems. While at first glance this may seem an unsatisfying state of affairs, there is a silver lining: FACTOR is one of the few known natural candidate "NP-intermediate" problems. Here, an NP-intermediate problem is a language in NP which is neither in P nor NP-complete. Of course, noone can *prove* that FACTOR is NP-intermediate. However, what we *can* prove (or more accurately, what Richard Ladner proved in 1975) is that, assuming P $\neq$ NP, NP-intermediate problems do exist.

Extended Church-Turing thesis. The backbone on which essentially all of theoretical computer science rests is the *Church-Turing Thesis*. This thesis posits that a Turing machine (TM) can simulate any other model of computing. (Note this is a *thesis*, not a proven theorem.) In other words, TMs tell us everything we need to know about computing. Except that the Church-Turing thesis says nothing about the *efficiency* with which

a TM can simulate other models. For this, we rely on the (less commonly accepted) *Extended Church-Turing Thesis*, which states that TMs can *efficiently* simulate any *physically realizable* model of computing.

The added constraints of "efficiency" and "physical realizability" of the Extended Church-Turing Thesis imply that, thanks to the quantum factoring algorithm, we have arrived at an exciting crossroads, at which one of the following three statements must be *false*:

1. The Extended Church-Turing Thesis is true.
2. FACTOR cannot be solved efficiently on a classical computer.
3. Large-scale universal quantum computers can be built.

Exercise 8. Convince yourself that the truth of any two of these statements implies the falsehood of the remaining statement. Where do the keywords "efficiently" and "physically realizable" from the Extended Church-Turing Thesis fit into your arguments?

3 The factoring algorithm

Having formally defined FACTOR, we discuss the quantum factoring algorithm, which consists of three high-level steps: Reducing FACTOR to order-finding (Section 3.1), and solving order-finding by combining QPE (Section 3.2) and continued fractions (Section 3.3).

3.1 Reducing FACTOR to order-finding

We begin by formally defining order-finding.

Definition 9 (Order-finding (ORDER))**.**

- *Input: Positive integers N and $1 \leq x < N$, with x co-prime to N.*
- *Output: The least positive integer r such that $x^r \equiv 1 \mod N$.*

Exercise 10. As stated, ORDER is not a decision problem. How can it be converted to one?

The reduction. We now show how FACTOR reduces to ORDER. We begin with a simple observation, given as the following exercise. Define $\gcd(x, y)$ as the greatest common divisor of $x, y \in \mathbb{Z}$. Recall that x and y are *co-prime* if $\gcd(x, y) = 1$.

Exercise 11. Let $x \in \mathbb{Z}$ be a non-trivial solution to equation $(x-1)(x+1) \equiv 0 \mod N$, where non-trivial means $x \not\equiv 1 \mod N$ and $x \not\equiv -1 \mod N$. Show that either $\gcd(x-1, N)$ or $\gcd(x+1, N)$ is a non-trivial factor of N. (Hint: Observe that, by assumption, N cannot divide either $x-1$ nor $x+1$.)

The exercise above gives us a route for factoring a given $N \in \mathbb{Z}^+$ — namely, we "just" need to find a non-trivial x satisfying $x^2 \equiv 1 \mod N$. The only question is — where do get our hands on such an x? To solve this difficult-sounding problem, we resort to the most advanced of algorithmic techniques — simply pick an x at random. More accurately, pick (uniformly at random) an integer x in set $Z_N := \{0, 1, \ldots, N-1\}$ which is co-prime to N. It turns out that with high probability, the *order* of x, i.e. the smallest r such that $x^r \equiv 1 \mod N$, allows us to construct the solution to $x^2 \equiv 1 \mod N$ we are looking for. Before formally proving that random sampling of x works, let us see why the order r of x helps.

Exercise 12. Let r be an even positive integer satisfying $x^r \equiv 1 \mod N$. Give a y satisfying $y^2 \equiv 1 \mod N$. (Hint: Note that r is even by assumption.)

Exercise 13. Does your y from the previous exercise necessarily allow you to extract a factor of N? (Hint: Is it necessarily true that y will be a *non-trivial* solution to $y^2 \equiv 1 \mod N$?)

The last exercise above suggests that simply picking an x with even order r will not suffice — we will need slightly more structure.

Finding a non-trivial solution to $x^2 \equiv 1 \mod N$. The proof that random sampling can lead to a non-trivial solution for $x^2 \equiv 1 \mod N$ requires three well-known facts, which we first state.

Fact 14 (Fermat's Little Theorem (FLT))**.** *For any prime p and integer a, $a^p \equiv a \mod p$. Moreover, if p does not divide a, then $a^{p-1} \equiv 1 \mod p$.*

Fact 15 (Chinese Remainder Theorem (CRT))**.** *Let $\{m_1, \ldots, m_n\}$ be a set of integers larger than 1, each pair of which is co-prime. Then, for any integers $\{a_1, \ldots, a_n\}$, the linear system of equations*

$$\begin{aligned} x &\equiv a_1 \mod m_1 && (2)\\ x &\equiv a_2 \mod m_2 && (3)\\ &\vdots && (4)\\ x &\equiv a_n \mod m_n && (5) \end{aligned}$$

has a solution $x \in \mathbb{Z}$. Moreover, all solutions are equivalent modulo $M = m_1 \cdots m_n$, implying there is a unique solution $x \in \{0, \ldots, M-1\}$.

Fact 16 (Multiplicative group Z_p is cyclic)**.** *Fix any positive prime $p \in \mathbb{Z}$. Then, there exists a generator $g \in Z_p$, such that any non-zero element $e \in Z_p$ can be written $g^k \equiv e \mod p$ for some $k \in \{1, \ldots, p-1\}$.*

Exercise 17. Use Fermat's Little Theorem to prove that there is a bijection between elements e and powers k in Fact 16 above. Conclude that picking e uniformly at random is equivalent to picking k uniformly at random. (For this exercise, do not assume *a priori* that $k \in \{1, \ldots, p-1\}$ as stated in Fact 16.)

Exercise 18. Let r be the order of x modulo N, and consider $r' > r$ satisfying $x^{r'} \equiv 1 \mod N$. Prove that r divides r'.

The main theorem of this section is the following. Its proof requires Lemma 25, which is stated and proven subsequently.

Theorem 19. *Let N be an odd, composite natural number. Define $Z_N^* := \{x \in Z_N \mid x \text{ is co-prime to } N\}$. Choose $x \in Z_N^*$ uniformly at random. Then, with probability at least $1/2$, the order r of x is even and satisfies*

$$x^{r/2} \not\equiv -1 \mod N. \quad (6)$$

Exercise 20. Why does Theorem 19 allow us to factor N with probability at least $1/2$, assuming we have an oracle for order-finding? In your answer, make sure you account for the fact that Theorem 19 does not explicitly state that $x^{r/2} \not\equiv 1 \mod N$.

Exercise 21. Is $0 \in Z_N^*$? What is Z_p^* for prime p?

Proof of Theorem 19. We prove the claim for the special case relevant to RSA (Section 4), i.e. $N = pq$ for distinct primes p and q. The proof can be generalized to arbitrary $N = p_1^{k_1} \cdots p_m^{k_m}$ for distinct primes $p_1, \ldots, p_m$.

To begin, the only thing we know about N is its prime factorization $N = pq$, so let us start by rephrasing what it means to "choose $x \in Z_N^*$ uniformly at random" in terms of "local" factors p and q.

Exercise 22. Show that choosing $x \in Z_N^*$ uniformly at random is equivalent to choosing $(a, b) \in S_{pq}$ uniformly at random, for set $S_{pq} := \{(a, b) \mid a \in Z_p^* \text{ and } b \in Z_q^*\}$. (Hint: Use the Chinese Remainder Theorem to demonstrate a bijection between Z_N^* and S_{pq}.)

Thus, assume we equivalently sample (a, b) uniformly at random from S_{pq}. We now have to bound the probability that r is even and $x^{r/2} \notin \{\pm 1\}$ modulo N.

Focus: Largest powers of 2. Recall that $x^r \equiv 1 \mod pq$, $x \equiv a \mod p$, and $x \equiv b \mod q$. Defining r_a and r_b as the orders of a modulo p and b modulo q, respectively, we thus have $x^{r_a} \equiv a^{r_a} \equiv 1 \mod p$, and $x^{r_b} \equiv b^{r_b} \equiv 1 \mod q$.

Exercise 23. Prove that r_a and r_b both divide r.

To bound the probability of our bad events (i.e. r is odd or $x^{r/2} \in \{\pm 1\}$ modulo N), it turns out the right place to look is the exact *power of* 2 which appears in each of the prime decompositions of r_a and r_b, respectively. Specifically, we proceed by showing two claims:

(A) Either bad event causes r_a and r_b to have precisely the *same* power of 2 in their prime factorizations.

(B) But since (a, b) is drawn uniformly at random from S_{pq}, the odds that r_a and r_b have this same factor of 2 is precisely $1/2$.

Formally, define t_a and t_b as the exact powers of 2 which divide r_a and r_b, respectively (e.g. $r_a = 2^{t_a} p_2^{k_2} p_3^{k_3} \cdots$ is the prime factorization of r_a, with $p_2, p_3, \ldots$ being distinct primes larger than 2). Let us first prove point (A). Consider the first bad event, r is odd. By Exercise 23, r_a and r_b divide r, and hence are both odd. Thus, $t_a = t_b = 0$. The second bad event is when r is even but $x^{r/2} \equiv -1 \mod N$. We know from Exercise 23 that r_a and r_b both divide r. Thus, t_a and t_b are *at most* the power of 2 in the prime decomposition of r. We show matching lower bounds. Since $x^{r/2} \equiv -1 \mod N$, we have $x^{r/2} \equiv -1 \mod p$ and $x^{r/2} \equiv -1 \mod q$. But by definition $x^{r_a} \equiv a^{r_a} \equiv 1 \mod p$ and $x^{r_b} \equiv b^{r_b} \equiv 1 \mod q$, so r_a and r_b cannot divide $r/2$. (Otherwise, for example, $x^{r/2} \equiv 1 \mod p$.) Since r_a and r_b divide r but not $r/2$, we obtain our desired lower bound — t_a and t_b must be *at least* the power of 2 in the prime decomposition of r. This concludes the proof of point (A). Point (B) now follows from Lemma 25, as the following exercise shows.

Exercise 24. Confirm that Point (B) follows from Lemma 25. □

Lemma 25. *Fix an odd prime p, and choose $a \in Z_p^*$ uniformly at random. Let r_a be the order of a. Then, with probability $1/2$, the largest power of 2 which divides $p - 1$ also divides r_a.*

Proof. Let t be the largest power of 2 which divides $p - 1$, i.e. 2^t divides $p - 1$.

Exercise 26. Why can we assume $t > 0$?

The proof strategy exploits Exercise 17, which recall states that choosing $a \in Z_p^*$ uniformly at random is equivalent to choosing $k \in \{1, \ldots, p-1\}$ uniformly at random (where $g^k \equiv a \mod p$ for fixed generator g of Z_p). In particular, since Z_p^* has even size, this means that k is odd with probability $1/2$. The key point is that, as we show next, k is odd if and only if 2^t divides r_a, yielding the desired claim regarding r_a.

- (Case 1: k odd) We claim 2^t divides r_a. Since

$$g^{kr_a} \equiv a^{r_a} \equiv 1 \equiv g^{p-1} \mod p,$$

where the last equivalence follows from FLT, we conclude $p - 1$ divides kr_a. But $p - 1$ is even and k is odd, implying the factor 2^t which divides $p - 1$ does not divide k, but rather r_a, as claimed.

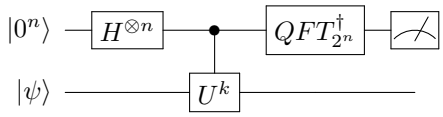

Figure 1: A high-level schematic of the QPE algorithm.

- (Case 2: k even) We claim 2^t does not divide r_a. To show this, note

$$g^{kr_a} \equiv a^{r_a} \equiv 1 \equiv g^{p-1} \equiv g^{k\frac{p-1}{2}} \mod p,$$

  where the third equivalence uses FLT, and the last that k is even. We conclude that r_a divides $(p-1)/2$. But $p-1$ has factor 2^t, meaning $(p-1)/2$ has factor 2^{t-1}, and thus the largest power of 2 appearing in the prime factorization of r_a is also at most 2^{t-1}. Thus, 2^t does not divide r_a, as claimed.

□

3.2 Sampling via QPE

Having reduced FACTOR to ORDER, we are now tasked with solving the latter: Given $x, N \in \mathbb{Z}^+$ such that x is co-prime to N, what is the order r of x? It turns out previous lectures have already given us all the toys needed to play this game. Indeed, we now detail the only quantum component of Shor's algorithm, consisting of an application of QPE (a high-level view of QPE is given in Figure 1 to jog your memory). This use of QPE requires overcoming three challenges: (1) Which unitary U do we apply QPE to? (2) Where do we find an eigenvector $|\psi\rangle$ of U to plug into QPE? (3) What precision do we need to run QPE to, and can we efficiently compute the corresponding controlled-U^k operation for this precision?

Challenge 1: The choice of U. Recalling that (x, N) is the input to ORDER, we shall run QPE on unitary U_x defined via action

$$U_x|y\rangle = |xy \mod N\rangle. \tag{7}$$

Exercise 27. Let V be a $d \times d$ complex matrix. Prove that V is unitary if and only if V maps any given orthonormal basis $B \subseteq \mathbb{C}^d$ to an orthonormal basis $B' \subseteq \mathbb{C}^d$. Can you give an outer product representation of V?

Exercise 28. Using the previous exercise, prove that U_x is unitary. (Hint: Use the fact that for any x co-prime to N, there exists an inverse $x^{-1} \in Z_N^*$ such that $xx^{-1} \equiv 1 \mod N$.)

The claim now is that certain eigenvalues of U_x encode the order r of x. Let us "design" the corresponding eigenvectors from scratch to build intuition. First, what do we know, and why did we choose U_x the way we did? Well, we know $x^r \equiv 1 \mod N$, and the operation $y \mapsto xy \mod N$ induced by U allows us to iteratively produce the elements of the infinite sequence $(1, x, x^2, x^3, \ldots, x^{r-1}, 1, x, \ldots)$. It follows that a first guess at an eigenvector of U_x is

$$|\eta\rangle := \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} |x^j \mod N\rangle. \tag{8}$$

Exercise 29. Show that $U_x|\eta\rangle = |\eta\rangle$, i.e. $|\eta\rangle$ has eigenvalue 1.

Unfortunately, an eigenvalue of 1 is not very exciting. However, we are only a tweak away from what we want. Recall the other setting in which we have a "cyclic" or "wraparound" property: The complex unit

circle. In particular, for principal rth root of unitary $\omega_r := \exp(2\pi i/r)$, we have $\omega_r^r = 1$. Thus, let us try injecting rth roots of unity as relative phases into $|\eta\rangle$ to obtain:

$$|\phi\rangle := \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} e^{2\pi ij/r} |x^j \mod N\rangle. \tag{9}$$

Exercise 30. Show that $U_x|\phi\rangle = e^{-2\pi i/r}|\phi\rangle$, i.e. $|\phi\rangle$ has eigenvalue $e^{-2\pi i/r}$.

We conclude that running QPE with U_x on $|\phi\rangle$ would yield an approximation to $1/r$, from which we could in principle recover r, as desired. There's only one problem — *how do we prepare* $|\phi\rangle$? Indeed, to prepare $|\phi\rangle$, it would seem we need to know r, which is precisely what we had originally set out to find!

Exercise 31. In Lecture 9, we assumed in the definition of QPE that the input phase $\theta \in [0, 1)$, whereas above we have $\theta = -i/r$. Show how a simple modification to the QPE algorithm allows us to nevertheless extract i/r.

Challenge 2: How to prepare $|\phi\rangle$. Unfortunately, it is not known how to construct $|\phi\rangle$ directly. There is, however, a slick way around this problem. Let us extend $|\phi\rangle$ to an orthonormal basis $B := \{|\phi_k\rangle\}$ for the Hilbert space $\mathcal{H}$ which U_x acts on, where we define $|\phi_1\rangle := |\phi\rangle$. Then, any unit vector $|\psi\rangle \in \mathcal{H}$ may be written $|\psi\rangle = \sum_k \alpha_k |\phi_k\rangle$ with $\sum_k |\alpha_k|^2 = 1$. Since QPE is unitary (omitting the final measurement), by linearity we have

$$|0^n\rangle|\psi\rangle = \sum_k \alpha_k |0^n\rangle|\phi_k\rangle \xmapsto{QPE} \sum_k \alpha_k |\theta_k\rangle|\phi_k\rangle, \tag{10}$$

where θ_k is an approximation to the phase corresponding to $|\phi_k\rangle$.

Exercise 32. Suppose we measure the first register of the right side of Equation (10). With what probability do we obtain our desired outcome θ_1 corresponding to $|\phi_1\rangle$?

The takeaway is that we need not prepare $|\phi_1\rangle$ *exactly*; rather, it may be easier to prepare some $|\psi\rangle$ which has non-zero amplitude α_1 on $|\phi_1\rangle$. It turns out this is not only easy, but *trivial*. Indeed, there is a clever choice of the first r basis vectors $|\phi_k\rangle$ in B which satisfies

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\phi_k\rangle = |1 \mod N\rangle. \tag{11}$$

(Note that $|1 \mod N\rangle$ is indeed trivial to prepare.) So what are $|\phi_2\rangle, \ldots, |\phi_r\rangle$? In particular, how do we systematically construct states orthonormal to $|\phi_1\rangle$? Observe that $|\phi_1\rangle$ is essentially a column from a Vandermonde matrix, obtained by raising the principal rth root of unity ω_r to increasing powers. Thus, we know precisely how to generate $r-1$ more vectors orthonormal to $|\phi_1\rangle$ — simply consider the Vandermonde matrix whose kth (for $k \in \{0, \ldots, r-1\}$) row exponentiates ω_r^k to increasing powers (up to $r-1$, inclusive). Formally, define

$$|\phi_k\rangle := \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} e^{2\pi ijk/r} |x^j \mod N\rangle. \tag{12}$$

Exercise 33. Prove that vectors $\{|\phi_k\rangle\}_{k=1}^r$ are orthonormal.

Exercise 34. Prove that $U|\phi_k\rangle = e^{-\frac{2\pi ik}{r}}|\phi_k\rangle$.

Exercise 35. Show that defining $|\phi_k\rangle$ as in Equation (12) yields Equation (11).

Plugging the exercise above into Equations (10) and (11), we have that if we run QPE on $|1 \mod N\rangle$ and measure the first register, with probability $1/r$ we get (an approximation to) phase k/r for some $k \in [r-1]$. This is almost what we want, except for the pesky k variable in the numerator. So we must settle for something less — the ability to sample uniformly at random from set $S := \{1/r, 2/r, \ldots, (r-1)/r\}$. Is this enough to extract r? Incredibly, the answer is yes, and will be covered in Section 3.3.

Exercise 36. Consider the following naive algorithm for extracting r. Run QPE repeatedly t times to collect t samples from S, for t some polynomial in the input size, $\log N$. Let t^* denote the smallest sample drawn. Return $1/t^*$ as the guess for r. Intuitively, how large might t have to be for this algorithm to extract r with (say) constant probability?

Challenge 3: Precision and the controlled-U^k operation. Before discussing how sampling from S allows us to extract r, we must address a key issue we have thus far swept under the rug: *Can we even efficiently run the QPE algorithm to the precision required for sampling from S, i.e. to represent k/r for $k \in [r-1]$?* To analyze this, recall that the input size to FACTOR is the number of bits needed to encode N, i.e. $n := \lceil \log N \rceil + 1$ bits.

Exercise 37. Consider arbitrary $k, r \in \mathbb{Z}^+$ for $k < r$, which can both be expressed exactly using n bits. How many bits are required to express the ratio k/r as a real number, i.e. as a sequence of bits $0.b_1b_2b_3\ldots b_m$? (Hint: Does k/r necessarily have a finite binary expansion, even if k and r have finite expansions?)

As the exercise above suggests, even if k and r have finite binary expansions, it is not necessarily true that k/r has a finite expansion. Thus, *a priori* it is not clear what precision we should require from QPE. Naively, one might "guess" that since kr requires at most $2n+1$ bits to represent, perhaps k/r can be sufficiently well-approximated also using $2n+1$ bits (at least close enough for us to recover k and r as integers). Indeed, it turns out that $2n+1$ bits of QPE precision will suffice for the continued fractions algorithm of Section 3.3 to extract k and r.

To obtain $2n+1$ bits of precision in QPE with probability at least $1-\epsilon$ for fixed $\epsilon > 0$, however, we require two things: Expanding the first register of Figure 1 to $n' = 2n + 1 + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ qubits (which is just a constant factor overhead), and the ability to raise U to power $2^{n'} \in O(2^n)$ in time $\text{poly}(n)$ (this smells like trouble). Stated differently, the latter requires us to compute $x^k y \mod N$ for $k \in O(2^n)$ in time $\text{poly}(n)$. As luck would have it, there is a well-known classical algorithm which accomplishes essentially this, known as *square-and-multiply*.

The square-and-multiply algorithm. This algorithm allows us to compute $a^b \mod M$ using $\text{polylog}(b)$ operations, which is exponentially faster than the brute force method of multiplying a by itself b times.

Basic idea. The basic idea is most easily seen when b is a power of 2, e.g. $b = 2^n$. In this case, note that (trivially) $a^b = (a^2)^{b/2}$. In other words, we can cut the exponent *in half* simply by performing a single squaring of a. So let us repeat this tradeoff, each time squaring the result of the previous square and cutting the remaining exponent in half. How many times can we repeat this before it terminates? Precisely $\log b = n$ times, hence yielding an exponential speedup over brute force, which would have required b operations.

General case. When b is not a power of 2, the principle is similar, but the recursive step requires two cases instead of one:

$$a^b = \begin{cases} (a^2)^{\frac{b}{2}} & \text{if } b \text{ is even} \\ a(a^2)^{\frac{b-1}{2}} & \text{if } b \text{ is odd.} \end{cases} \tag{13}$$

The square-and-multiply algorithm repeatedly applies this recursive step until $b = 0$, at which point it terminates. Note that since we are working mod M, in order to prevent the expression being computed from blowing up exponentially quickly, we may apply the mod operation after each application of the recursive rule.

Exercise 38. Asymptotically, how many operations does the algorithm require in the general case?

3.3 Postprocessing via continued fractions

In Section 3.2, we used QPE to approximately sample from set $S := \{1/r, 2/r, \ldots, (r-1)/r\}$. More specifically, defining $n := \lceil \log N \rceil + 1$, each time we ran the QPE subroutine, we obtained (with probability at least $1 - \epsilon$) the $2n+1$ most significant bits of the binary expansion of some random element k/r of S. Given access to such a sample x, it thus remains to extract a guess for r. To do so, we run the *continued fractions* algorithm, for which we first overcome the following minor obstacle.

A minor obstacle. If we are to have any hope of extracting r given k/r, we must have k and r co-prime (in other words, k/r is already in lowest terms). *A priori*, there is no way of guaranteeing this, since k is drawn uniformly at random from $\{0, \ldots, r-1\}$. However, the uniform sampling is simultaneously the key to the solution. Namely, given positive integer r, the number of positive integers less than r which are *co-prime* to r has been studied since 1763, and goes by the name of the Euler totient function $\phi(r)$. The totient function scales as $\phi(r) \in \Omega(r/\log\log r)$ for $r > 2$. Thus, a randomly chosen k in $\{0, \ldots, r-1\}$ is co-prime to r with probability scaling as $1/\log\log r \geq 1/\log\log N$. It follows that repeating the QPE sampling step $O(\log\log N)$ times (i.e. logarithmic in the encoding size of N) suffices to obtain k/r with k co-prime to r with high probability. Assuming we have such a k/r, we can now apply continued fractions.

The continued fractions algorithm. In a nutshell, the continued fractions expansion is just another representation for a given rational number a/b for $a, b \in \mathbb{Z}^+$. Specifically, given a/b, the algorithm outputs a finite sequence of integers $C := (c_0, c_2, \ldots, c_m)$, such that

$$\frac{a}{b} = c_0 + \cfrac{1}{c_1 + \cfrac{1}{c_2 + \cfrac{1}{\cdots + \cfrac{1}{c_m}}}}. \tag{14}$$

Why should this representation of a/b be useful? First, instead of viewing C as representing a *single* rational number, note we may view it as encoding a *list* of m rational numbers, each defined via the continued fraction subsequence (formally, *kth convergent*) $C_k := (c_0, \ldots, c_k)$. Recall now that what QPE gave us was not a random sample k/r, but rather the first $2n+1$ bits of k/r, denoted x. Thus, x itself is rational, and is "close" to k/r. Amazingly, the following theorem thus tells us that the continued fractions expansion of x will yield a sequence of m convergents, one of which is precisely a/b.

Theorem 39. *Let $k, r \in \mathbb{Z}^+$ and $x \in \mathbb{Q}^+$ satisfy*

$$\left| \frac{k}{r} - x \right| \leq \frac{1}{2r^2}. \tag{15}$$

Let $C = (c_0, \ldots, c_M)$ be the continued fractions expansion of x. Then, there exists a $k \in \{0, \ldots, M\}$ such that the kth convergent $(c_0, \ldots, c_k)$ represents k/r exactly.

Exercise 40. Show that since x is the $2n+1$ most significant bits of k/r, Equation (15) indeed holds, and thus we can apply Theorem 39.

This *almost* gives us what we want — instead of an approximation x to k/r, we now actually have k/r exactly. There are three questions to be answered: (1) How large is M in Theorem 39? (2) How do we

compute the continued fractions expansion of x? (3) Given the convergent of x which yields k/r, how do we extract r (recall we don't know k)? For (1), it turns out since k and r are n-bit integers, one can choose $M \in O(n)$. The answers to the other two questions we give below.

Computing the continued fractions expansion. The algorithm essentially "implements" Equation (14) in a "greedy" fashion. Let us demonstrate with an explicit example, the expansion of 23/9:

$$\frac{23}{9} \xrightarrow{\text{split}} 2 + \frac{5}{9} \xrightarrow{\text{invert}} 2 + \frac{1}{\frac{9}{5}} \xrightarrow{\text{split}} 2 + \frac{1}{1 + \frac{4}{5}} \xrightarrow{\text{invert}} 2 + \frac{1}{1 + \frac{1}{\frac{5}{4}}} \xrightarrow{\text{split}} 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{4}}}.$$

As demonstrated above, the algorithm repeatly applies the *split* step (extract the integer component) and *invert* step (flip the fractional component) until the final fraction is of form $1/z$ for some $z \in \mathbb{Z}$.

Exercise 41. Give the continued fractions expansion for 31/11.

Exercise 42. Prove that the continued fractions expansion for any rational number is indeed finite.

Recovering r by "inverting" the continued fractions expansion. Let $C = (c_0, c_1, \ldots, c_m)$ be the continued fractions expansion of the $2n+1$-bit estimate x for k/r given by QPE. Since we assumed k and r are co-prime, we are now in a position to extract r via the following theorem. Note we do not know *a priori* which convergent of C correctly encodes k/r; but since $m \in O(n)$, we can simply try all convergents in polynomial time.

Theorem 43. *Given any continued fractions expansion (or more generally, some kth convergent) $C_k := (c_0, c_1, \ldots, c_k)$ where $c_i \in \mathbb{Z}^+$ for all $i \in 0, \ldots, k$, define a_i and b_i recursively as follows:*

$$\begin{array}{llll} a_0 = c_0 & a_1 = 1 + c_0 c_1 & a_i = c_i a_{i-1} + a_{i-2} & \text{for } i \in \{2, \ldots, k\} \\ b_0 = 1 & b_1 = c_1 & b_i = c_i b_{i-1} + b_{i-2} & \text{for } i \in \{2, \ldots, k\}. \end{array}$$

Then, the expansion C_k is equivalent to rational number a_k/b_k, where a_k and b_k are co-prime.

Combining Theorems 39 and 43, we hence have that trying all m possible convergents in the expansion C for x (which recall is a $2n+1$-bit estimate of k/r) will eventually allow us to recover both k and r.

Exercise 44. Apply Theorem 43 to $C = (1, 2, 3, 4)$. What are a_3 and b_3? Confirm that they are co-prime.

4 Application: Breaking RSA

Finally, let us briefly discuss how the quantum factoring algorithm allows us to efficiently break RSA.

The RSA cryptosystem. RSA, named after its founders Rivest-Shamir-Adleman, is a public key cryptosystem still widely used today. Roughly, in a *public key* cryptosystem, Alice wishes to allow the public to send her private messages. For this, she creates a pair of keys: One public key P, one private key S. The public key P is distributed openly to the world; anyone can use P to encrypt a given message M, obtaining ciphertext $E(M)$ for transmission to Alice. Upon receipt of any such $E(M)$, Alice uses her private key S, which is only known to her, to decrypt $E(M)$ and recover M.

What we thus need to specify for RSA are the public and private keys P and S, respectively, as well as the encryption and decryption algorithms. These are given as follows.

- **How Alice creates the keys:**

  1. Alice chooses two large prime integers p and q, and computes $N = pq$.

2. She randomly chooses small odd $e \in \mathbb{Z}$ co-prime to $\phi(N) := (p-1)(q-1)$.
3. She computes d satisfying $ed \equiv 1 \mod \phi(N)$.
4. She publicly releases $P = (e, N)$, and privately stores $S = (d, N)$.

- **How the public encrypts messages given $P = (e, N)$.** Suppose Bob wishes to encrypt string $M \in \{0,1\}^{\lfloor \log N \rfloor}$. Viewing M as the binary encoding of an integer, he computes ciphertext

$$E(M) := M^e \mod N. \tag{16}$$

Note this can be done efficiently even for large e via the square-and-multiply algorithm.

Exercise 45. Why is the length of M assumed to be bounded by $O(\log N)$ above?

- **How Alice decrypts ciphertext $E(M)$.** Given ciphertext $E(M)$, Alice recovers M by computing

$$M = (E(M))^d \mod N. \tag{17}$$

The proof of this equality is not difficult, but is omitted to avoid sidetracking the discussion.

- **How factoring allows one to completely break RSA.** Suppose adversary Eve can factor N efficiently into primes p and q. Then, since e is public knowledge, she can recompute d such that $ed \equiv 1 \mod \phi(N)$ (Step 3 of Alice's creation protocol), giving her Alice's private key $S = (d, N)$. Thus, not only can Eve decrypt any incoming ciphertext to Alice, but she in fact has complete knowledge of Alice's secret key.

In sum, what makes RSA classically "hard" to break is that given just $N = pq$, it is not clear how to compute $\phi(N) = (p-1)(q-1)$. But given the factors p and q, recovering $\phi(N)$, and hence the private key d, is easy.

References

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300 – 317, 1976.