Regev Factoring Beyond Fibonacci: Optimizing Prefactors

Seyoon Ragavan
MIT
sragavan@mit.edu

April 25, 2024

Abstract

In this note, we improve the space-efficient variant of Regev's quantum factoring algorithm [Reg23] proposed by Ragavan and Vaikuntanathan [RV24] by constant factors in space and/or size. This allows us to bridge the significant gaps in concrete efficiency between the circuits by [Reg23] and [RV24]; [Reg23] uses far fewer gates, while [RV24] uses far fewer qubits.

The main observation is that the space-efficient quantum modular exponentiation technique by [RV24] can be modified to work with more general sequences of integers than the Fibonacci numbers. We parametrize this in terms of a linear recurrence relation, and through this formulation construct three different circuits for quantum factoring:

- A circuit that uses $\approx 12.4n$ qubits and $\approx 54.9n^{1/2}$ multiplications of n-bit integers.
- A circuit that uses $(9+\epsilon)n$ qubits and $O_\epsilon(n^{1/2})$ multiplications of n-bit integers, for any $\epsilon > 0$.
- A circuit that uses $(24+\epsilon)n^{1/2}$ multiplications of n-bit integers, and $O_\epsilon(n)$ qubits, for any $\epsilon > 0$.

In comparison, the original circuit by [Reg23] uses at least $\approx 3n^{3/2}$ qubits and $\approx 6n^{1/2}$ multiplications of n-bit integers, while the space-efficient variant by [RV24] uses $\approx 10.32n$ qubits and $\approx 138.3n^{1/2}$ multiplications of n-bit integers (although a very simple modification of their Fibonacci-based circuit uses $\approx 11.32n$ qubits and only $\approx 103.7n^{1/2}$ multiplications of n-bit integers). The improvements proposed in this note take effect for sufficiently large values of n; it remains to be seen whether they can also provide benefits for practical problem sizes.

1 Introduction

Shor's algorithm from 1994 [Sho97] showed us how to factor n-bit integers quantumly in time polynomial in n. Further optimizations by [Bea03, TK06, Zal06, Gid17, HRS17, Gid19] ultimately led to implementations of Shor's circuit that could use as little space as $\approx 1.5n$ qubits or as few as $O(n/\log n)$ calls[1] to a circuit for multiplying two n-bit integers modulo N.

This state of affairs changed when Regev [Reg23] demonstrated a quantum circuit that needed only $\approx 6\sqrt{n}$ calls to the same multiplication circuit.[2] This circuit could be run $\sqrt{n}$ times in parallel,

[1] This is using Gidney's windowing technique [Gid19], which is applicable when the multiplication circuit being used has $O(n^{1+\Omega(1)})$ gates. To the best of our knowledge, windowing when working with asymptotically faster multiplication circuits [SS71, HvdH21] would only yield savings proportionate to $O(\log\log n)$, but this is not of major interest in this work since we are concerned with concrete costs. We compare our methods with windowing in Section 3.

[2] This prefactor is achieved using similar ideas to the optimizations in this work; see Section 3.1 for details.

Algorithm	Qubits	n-bit multiplications	Qubits $\times$ multiplications
Regev's algorithm [Reg23]	$3n^{3/2}$	$\mathbf{6n^{1/2}}$	$18n^2$
[RV24]	$11.32n$	$103.7n^{1/2}$	$1173.9n^{3/2}$
[RV24]†	$10.32n$	$138.3n^{1/2}$	$1427.3n^{3/2}$
This work, Corollary 2.6	$12.43n$	$54.9n^{1/2}$	$\mathbf{682.3n^{3/2}}$
This work, Corollary 2.5	$O_\epsilon(n)$	$(24+\epsilon)n^{1/2}$	$O_\epsilon(n^{3/2})$
This work, Corollary 2.3†	$\mathbf{(9+\epsilon)n}$	$O_\epsilon(n^{1/2})$	$O_\epsilon(n^{3/2})$

Table 1: Comparison of implementations of Regev's algorithm [Reg23] in terms of both number of qubits and number of gates. Asymptotically best-known results for space, size, or their product are highlighted in bold. Circuits marked with † use dirty ancilla qubits to save space at the expense of needing more gates (see Section 2.3 for more discussion about this). All values here are just for one run of the circuit. For $n \geq 1500$, our circuit from Corollary 2.6 achieves the smallest product of qubits and gates when considering the highest-order terms. However, this comparison ignores lower-order terms so a finer analysis is needed to verify the concrete gains from our results.

and the results classically postprocessed to factor N. The circuit by [Reg23] has since been adapted to the problem of computing discrete logarithms modulo prime p by Ekerå and Gärtner [EG24]. However, Regev's circuit requires at least $3n^{3/2}$ qubits, which is significantly more than Shor's.

Ragavan and Vaikuntanathan [RV24] then leveraged the idea of reversible Fibonacci exponentiation proposed by Kaliski [Kal17] to asymptotically achieve the best of both worlds between Shor's and Regev's circuits: a careful implementation of their circuit would need only $\approx 10.32n$ qubits, but now requires $\approx 138.3\sqrt{n}$ multiplications.[3] A simple modification of their circuit would require $\approx 11.32n$ qubits and $\approx 103.7\sqrt{n}$ multiplications.[4] For cryptographically relevant problem sizes, there is an unsatisfactory gap here; the circuit by [Reg23] requires an extremely large number of qubits, while the circuit by [RV24] requires an extremely large number of gates. These gaps in concrete efficiency become especially important in the case of quantum computers, where scalability has proven difficult due to the issue of decoherence noise [AAB+19, GE21, CCHL22, Cai23].

In this note, we show how to interpolate between these two qualitative extremes in a nontrivial way. In particular, all circuits in this work retain the asymptotic best-of-both-worlds guarantees achieved by [RV24]. At the core, we consider applying reversible exponentiation with respect to more general sequences of integers than the Fibonacci numbers, and show that this can provide benefits for space and/or size depending on the parametrization. We state our results formally in Section 2 and discuss this idea in more detail in Section 3.

Our results are phrased in terms of integer parameters r and s that allow one to control a tradeoff between space and size. Some special cases are tabulated alongside the original implementation by [Reg23] and the variant by [RV24] in Table 1. For a heuristic comparison, we consider the product of the number of qubits and number of multiplications for each of the algorithms proposed. This comparison suggests that our proposal could provide concrete improvements to Regev's algorithm for n as low as 1500, although our methods for optimizing the number of gates appear limited to

[3] This can be seen by plugging $r = 1$ into Theorem 2.2 and doubling the gate count to account for final uncomputation.

[4] This can be seen by plugging $r = s = 1$ into Theorem 2.4 and doubling the gate count to account for final uncomputation.

4× the gate complexity of Regev's original circuit [Reg23]. However, ascertaining this will require a finer and more careful analysis, as our complexity estimates ignore lower-order terms in the number of qubits and gates. We remark that although our results here focus on factoring, these algorithms can also be directly adapted to discrete logarithms by following the methods of [EG24].

For simplicity, we do not make comparisons in this work with optimized variants of Shor's algorithm due to the wide variety of optimizations that one could consider [Bea03, TK06, Zal06, Gid17, HRS17, Gid19]. A careful comparison with these variants of Shor's algorithm is an important direction that we leave to future work. We are optimistic that Regev's algorithm [Reg23] combined with the optimizations by [RV24] and this work (and perhaps additional future optimizations) may be sufficient to achieve a concrete improvement over Shor's algorithm [Sho97].

2 Setup

We refer the reader to [RV24] for an overview of Regev's quantum factoring algorithm [Reg23]. In this work, we focus on the modular exponentiation step, which is the bottleneck in terms of both gates and qubits. For simplicity, we will assess gate complexity in terms of the number of multiplications of n-bit integers modulo N. This is a reasonable heuristic since integer multiplications are the asymptotic bottleneck in the circuit by [Reg23] and its modification by [RV24].

2.1 Notation

We retain all notation from [Reg23] and restate it here for convenience. Let $N < 2^n$ be an n-bit number. Let $d = \lfloor \sqrt{n} \rfloor$ and $b_1, \ldots, b_d$ be some small $O(\log n)$-bit integers (e.g. b_i is the ith prime number) and let $a_i = b_i^2 \bmod N$. For any integer t, let $[t]$ denote the set $\{1, 2, \ldots, t\}$. We use log to denote the base-2 logarithm throughout this paper. Also, let ϕ denote the golden ratio. Regev's algorithm uses a number of parameters:

- Let $C > 0$ be an absolute constant given by Regev's [Reg23] number-theoretic conjecture (conjecture 3.1 in [RV24]);
- Let $A > C$ be another constant we specify later. For Regev's factoring algorithm and our space optimization, we will take $A = C + 2 + o(1)$;
- Let $R = 2^{(A+o(1))\sqrt{n}}$; and
- Let D be a power of 2 in $[2\sqrt{d} \cdot R, 4\sqrt{d} \cdot R]$. Note that D is also $2^{(A+o(1))\sqrt{n}}$. These are the same parameters R and D defined by [Reg23].

Following a heuristic argument by [Reg23] suggests that taking $C = 1 + \epsilon$ (and hence $A = 3 + \epsilon$) is likely to be sufficient for his number-theoretic conjecture to hold. For the modular exponentiation step, Regev works with a vector of integers $z = (z_1, z_2, \ldots, z_d)$, where $-D/2 \leq z_i \leq D/2 - 1$ for all i.

We also borrow notation from [RV24] and let $|\psi(x)\rangle$ denote the state $|x\rangle |x^{-1} \bmod N\rangle$, for an integer $x \in [0, N-1]$ relatively prime to N. For the purposes of this work, we also define constant integer parameters $s \leq r$ such that s is a power of 2 and r is a multiple of s. We will also define $\beta = \frac{r+\sqrt{r^2+4}}{2}$, for reasons that will become clear later. (Note that in the special case $r = 1$, β will equal the golden ratio $\phi = \frac{1+\sqrt{5}}{2}$.)

Finally, we use the standard complexity-theoretic $\omega(f(n))$ notation to denote a function $g(n)$ such that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

2.2 Our Results

Our goal is to improve upon Lemma 4.1 by [RV24], which we restate below for reference. All gate costs in these theorems need to doubled to obtain the cost of a factoring circuit, because these circuits must be uncomputed before the final QFT in Regev's circuit [Reg23] can be performed.

Theorem 2.1. *(Lemma 4.1 of [RV24]) Assume there is a quantum circuit that implements the operation*

$$|a\rangle |b\rangle |t\rangle |0^S\rangle \mapsto |a\rangle |b\rangle |(t + ab) \bmod N\rangle |0^S\rangle$$

with $G = \omega(n)$ gates where N, a, b, t are all n-bit integers with $0 \le a, b, t < N$ and $2^{n-1} \le N < 2^n$, and S here is the number of ancilla qubits. Then there exists a quantum circuit mapping

$$|z\rangle |0^M\rangle \mapsto \left| \prod_{i=1}^{d} a_i^{z_i + D/2} \bmod N \right\rangle |\psi\rangle .$$

Here,

$$M = S + \left(\frac{A}{\log \phi} - A + 6 + o(1) \right) n$$

is the initial number of ancilla qubits, and $|\psi\rangle$ is some possibly nonzero state on $M + An - n$ qubits. Moreover, this circuit uses $O(n^{1/2} \cdot G + n^{3/2})$ gates. Since $|z\rangle$ can be stored in $d \log D = An$ qubits, the total number of qubits used (i.e. the space usage) is

$$S + \left(\frac{A}{\log \phi} + 6 + o(1) \right) n.$$

Taking $A = 3 + \epsilon$ implies that the number of qubits is $\approx S + 10.32n$.

We first generalize this theorem to the case where $r > 1$, which as we will see yields an improvement in the space complexity:

Theorem 2.2. *Under the same assumptions as Theorem 2.1, there exists a quantum circuit computing the same mapping with*

$$M = S + \left(\frac{A\lceil \log(r+1) \rceil}{\log \beta} - A + 6 + o(1) \right) n,$$

so that the total number of qubits used is

$$S + \left(\frac{A\lceil \log(r+1) \rceil}{\log \beta} + 6 + o(1) \right) n,$$

and the number of gates used is

$$\frac{(4r + 12 + o(1))A}{\log \beta} \cdot n^{1/2} \cdot G.$$

The $r = 1$ case is exactly Theorem 2.1.

Since $\lim_{r\to\infty}\lceil\log(r+1)\rceil/\log\beta = 1$, taking sufficiently large r and $A = 3+\epsilon$ allows us to obtain the following corollary:

Corollary 2.3. *For any constant $\epsilon > 0$, there exists a quantum circuit computing the same mapping as Theorem 2.1 using space $S + (9+\epsilon)n$ and $O_\epsilon(n^{1/2}\cdot G)$ gates.*

Next, we show that the number of gates can also be dropped by a constant factor (albeit at the expense of blowing up the constant factor in the space), by taking $r > 1$ (the $r = 1$ case can be obtained by a direct simplification of the circuit by [RV24]):

Theorem 2.4. *Under the same assumptions as Theorem 2.1, there exists a quantum circuit computing the same mapping with*

$$S + \left(\frac{A\lceil\log(r+1)\rceil}{\log\beta} + \max(4\log s + 5, 7) + o(1)\right) n$$

qubits, and

$$\frac{(4r/s + 4\log s + 8 + o(1))A}{\log\beta}\cdot n^{1/2}\cdot G$$

gates.

If we set $r = s$ to be a power of 2 and consider the limiting behaviour as $r\to\infty$, we obtain the following corollary (once again taking $A = 3+\epsilon$):

Corollary 2.5. *For any constant $\epsilon > 0$, there exists a quantum circuit computing the same mapping as Theorem 2.1 using $(12+\epsilon)n^{1/2}\cdot G$ gates and space $S + O_\epsilon(n)$.*

Setting $r = 6$ and $s = 2$ yields a result[5] that is concretely efficient in terms of both space and size (once again taking $A = 3+\epsilon$):

Corollary 2.6. *There exists a quantum circuit computing the same mapping as Theorem 2.1 using $\approx 27.4n^{1/2}\cdot G$ gates and space $\approx S + 12.4n$.*

2.3 On the Role of Dirty Ancilla Qubits

The core difference between the algorithms used in Theorem 2.2 and 2.4 (focusing on $s = 1$ for ease of comparison) is the use of dirty ancilla qubits. This leads to three points of difference between these algorithms:

- The use of dirty ancilla qubits allows Theorem 2.2 to use n fewer qubits than it would otherwise need to.
- However, this also leads to a higher gate count, which will ultimately be due to the difference in gate counts between Lemma 3.3 and Lemma 3.5, and similarly between Lemma 3.4 and Lemma 3.6. In each case, the former lemma uses dirty ancilla qubits to save space but at the expense of needing more multiplications.

[5] These parameters were experimentally chosen to minimize the product of the two prefactors.

- More subtly, the multiplication functionality we require when using dirty ancilla qubits is more sophisticated. In Lemmas 3.3 and 3.4, the work by [RV24] explicitly relies on the ring structure of $\mathbb{Z}_N$, and assumes a multiplication oracle that maps $|a\rangle|b\rangle|t\rangle \mapsto |a\rangle|b\rangle|(t+ab) \bmod N\rangle$.

  On the other hand, in the simpler case of Lemmas 3.5 and 3.6 where dirty ancilla qubits are not involved, it is straightforward to see that the arguments in the lemmas work even given a multiplication oracle that maps $|a\rangle|b\rangle|t\rangle \mapsto |t \oplus (ab \bmod N)\rangle$, which now only uses the multiplicative structure of $\mathbb{Z}_N$.

  As argued in Appendix A of [RV24], any multiplication oracle with the XOR functionality can be converted into an oracle with the $(t+ab) \bmod N$ functionality, but with some constant-factor overheads that are unlikely to be desirable in practice.

- Another consequence of using dirty ancilla qubits is that quantum errors become more difficult to detect. This is an observation made by Shor [Sho97]. At the end of the computations in Lemmas 3.5 and 3.6, the clean ancilla qubits will be restored to the $|0^n\rangle$ state, which can hence be measured and checked. This certainly does not provide the ability to detect *any* quantum errors, but it does provide the ability to detect some types of quantum errors.

  Without clean ancilla qubits (as in Theorem 2.2 and the circuits by [RV24]), this form of error detection does not appear to possible (to the best of our knowledge).

Given the above observations, we view the use of dirty ancilla qubits in [RV24] and Theorem 2.2 as a way to finely optimize the space usage of the quantum circuit, but believe it is unlikely that these optimizations would be useful in practice. A simpler algorithm such as that in Theorem 2.4 is likely to be of more practical interest.

3 Modular Exponentiation Beyond Powers of 2 and Fibonacci Numbers

The reason why Regev's factoring circuit [Reg23] requires $n^{3/2}$ qubits is that it relies on repeated squaring modulo N for its exponentiation, which is not reversible and hence has to be done out-of-place, consuming extra qubits. Another way to view this exponentiation approach is that it uses the powers of 2 as a "basis" for the exponent z when computing a^z modulo N. This is for the simple reason that the powers of 2 obey the recurrence $G_n = 2G_{n-1}$.

The observation originally made by Kaliski [Kal17] in the context of classical reversible computing and leveraged by [RV24] is that modular exponentiation can be done by instead iterating the operation $(a,b) \mapsto (a, ab \bmod N)$ back and forth. This turns out to use the Fibonacci numbers as a basis for the exponent. This is ultimately because the Fibonacci numbers satisfy the recurrence relation $F_k = F_{k-1} + F_{k-2}$; in particular, the aforementioned operation will send $(a^{F_{k-1}} \bmod N, a^{F_{k-2}} \bmod N) \mapsto (a^{F_{k-1}} \bmod N, a^{F_k} \bmod N)$.

This suggests considering what may happen by considering an arbitrary linear recurrence relation with constant non-negative integer coefficients, and the basis that yields. The first observation is that not every recurrence yields a computation that can be done reversibly; $G_n = 2G_{n-1}$ corresponds to repeated squaring, which is not reversible. However, it turns out that any recurrence relation where the last coefficient is 1 *can* be used for reversible exponentiation. Concretely, let r be a positive integer and let $\{G_k\}$ be defined by $G_0 = 0, G_1 = 1$, and

$$G_k = rG_{k-1} + G_{k-2}.$$

Then we can iterate the operation $(a, b) \mapsto (a, a^r b)$ back and forth to exponentiate using the G_k's as a basis. This is because this operation will send

$$\begin{aligned}(a^{G_{k-1}} \bmod N, a^{G_{k-2}} \bmod N) &\mapsto (a^{G_{k-1}} \bmod N, a^{rG_{k-1}+G_{k-2}} \bmod N) \\ &= (a^{G_{k-1}} \bmod N, a^{G_k} \bmod N).\end{aligned}$$

There are two different reasons that we might hope to benefit from working with such a sequence instead. One is that this could potentially improve space and the other is that this could potentially improve the number of gates, but these two aspects trade against each other so it appears difficult to leverage both types of benefit at once:

- In the algorithm by [RV24], the An qubits needed to store the exponents z gets blown up to $An/\log\phi$, because of redundancy in decomposing the exponent as a sum of Fibonacci numbers. (Indeed, any decomposition that includes two consecutive Fibonacci numbers is already redundant.) It turns out that this redundancy can be reduced by increasing r, thereby reducing the space requirement.

- Secondly, as we will see, there are two types of multiplication operations carried out by the algorithm in [RV24]. The first is the operation $(a, b) \mapsto (a, a^r b)$ (recalling that they restrict attention to $r = 1$), and the second is an intermediate operation where one of the registers is multiplied by a smaller integer c_j.

  The cost of multiplications of the first type appears inherent and independent of r, however we can reduce the number of times we need to carry out the second type of multiplication by increasing r.

We remark that the second optimization listed here bears some high-level similarity to the use of windowing in Shor's algorithm [Gid19, GE21]. Windowing effectively shifts some of the workload from the modular exponentiation to classical precomputation. This crucially relies on the base a being classical and does not appear applicable to Regev's algorithm (see [RV24] for further discussion about the limitations of precomputation in Regev's algorithm). Instead, we can shift some workload from the core modular exponentiation operations to the small-integer multiplications (as captured in Lemma 3.2). A crucial point of difference is that while windowing allows for the saving of logarithmic factors in Shor's algorithm [Gid19], our optimization is limited to constant-factor improvements; asymptotic improvements with this method appear difficult.

3.1 Warm-Up: Prefactors in Regev's Circuit

To begin, we examine how ideas similar to those outlined here can improve the prefactors in Regev's original circuit [Reg23], without the Fibonacci exponentiation optimizations by [RV24]. Suppose firstly that we run Regev's circuit as-is, decomposing the exponents $z_i + D/2$ in binary and then using repeated squaring. Then we have the following:

- The required number of qubits is $(A + o(1))n^{3/2}$, because we need an additional n qubits for each of $An^{1/2}$ squarings.

- The required number of n-bit multiplications is at least $4An^{1/2}$. During the forward execution of the circuit, there are $\log D \approx An^{1/2}$ squarings and additionally $\log D \approx An^{1/2}$ multiplications of a subset of the bases into the register between squarings. This cost then gets doubled due to the need for uncomputation at the end.

Instead, suppose we did the following: take r to be a power of 2 (in this section only) and decompose the exponents $z_i + D/2$ in base r. This leads to the following changes to the algorithm:

- The multiplications of subsets of small integer bases now uses a factor of r more gates because these integer bases could be raised to the power of r (see Lemma 3.2 for details).
- In the forward pass, we still need $\log D \approx An^{1/2}$ squarings (to raise something to the power of r, we would just square it $\log r$ times). However, the number of times we need to multiply a subset of bases into the register is now only $\log D / \log r \approx An^{1/2} / \log r$. The total number of n-bit multiplications is now only $\approx 2An^{1/2}(1 + 1/\log r)$.

Note that the number of qubits is still just $An^{3/2}$. Hence we can take r to be slightly super-constant to obtain an implementation of Regev's circuit [Reg23] using $\approx An^{3/2} \approx 3n^{3/2}$ qubits and $\approx 2An^{1/2} \cdot G \approx 6n^{1/2} \cdot G$ gates. We will see that we can obtain similar benefits for the circuit by [RV24] in terms of the number of gates as well as for the number of qubits (whereas this method did not save any qubits in the case of Regev's circuit).

3.2 Outline of Our Algorithm

Now, we analyze how this exponentiation would work in the case of the space-efficient circuit by [RV24]. Let K be maximal such that $G_K \leq D$. We want to compute $\prod_{i=1}^{d} a_i^{z_i + D/2}$, which suggests decomposing

$$z_i + D/2 = \sum_{j=1}^{K} z_{i,j} G_j,$$

for $z_{i,j} \in \{0, 1, \ldots, r\}$. Then, if we let $c_j = \prod_{i=1}^{d} a_i^{z_{i,j}}$, we just need to compute $\prod_{j=1}^{K} c_j^{G_j}$. We have two straightforward properties that we verify in Appendix B:

- Define $\beta = \frac{r + \sqrt{r^2+4}}{2} \in (r, r+1)$, then we have $K = (1 + o(1)) \log D / \log \beta = (\alpha + o(1))\sqrt{n}$, letting $\alpha = A / \log \beta$.
- A straightforward greedy algorithm can be applied to calculate a decomposition of the above form with $z_{i,j} \in \{0, 1, \ldots, r\}$ for all i, j.

Then applying the intuition stated above to the space-efficient exponentiation algorithm by [RV24] suggests the following outline:

1. Compute and store values $z_{i,j} \in \{0, 1, \ldots, r\}$ for $i \in [d]$ and $j \in [K]$ such that for all i we have $\sum_{j=1}^{k} z_{i,j} G_j = z_i + D/2$.
2. Initialize x_1 and x_2 to both be 1.
3. Repeat the following for $j = K, K-1, \ldots, 1$ in that order:
   (a) Update $x_1 \leftarrow x_1 x_2^r$.
   (b) Update $x_1 \leftarrow x_1 c_j$.
   (c) Swap x_1 and x_2.

To show correctness, we argue that for all $j \leq K - 1$ we will have at the end of round j that $x_1 = \prod_{i=j+1}^{K} c_i^{G_{i-j}}$ and $x_2 = \prod_{i=j}^{K} c_i^{G_{i+1-j}}$. When $j = 1$, we will hence have $x_2 = \prod_{i=1}^{K} c_i^{G_i}$, as desired. This claim follows by a straightforward induction, which we defer to Appendix A.

3.3 Greedy Decomposition and Small-Integer Multiplications

In this section, we adapt the primitives used by [RV24] that specifically work with Fibonacci numbers to our more general setting.

Lemma 3.1. *(Adapted from Lemma 5.5 in [RV24]) There exists a quantum circuit using $O(n^{3/2})$ gates mapping the state*

$$|z\rangle\,|0^{dK\lceil\log(r+1)\rceil-d\log D}\rangle\,|0^{O(\sqrt{n})}\rangle \mapsto |z_{i,j} : i\in[d], j\in[K]\rangle\,|0^{O(\sqrt{n})}\rangle\,.$$

The $\lceil\log(r+1)\rceil$ term is because each $z_{i,j}$ is an integer in $[0,r]$, and hence this many bits are needed to represent it. Note that the prefactor in the number of gates here is bounded independently of r.

Proof. We repeat the following greedy procedure for each $i\in[d]$. Note that integers here are computed in absolute terms, rather than modulo N. We need the ability to compute in-place additions and subtractions on $O(\sqrt{n})$-bit integers with $O(\sqrt{n})$ ancilla qubits; it was shown by [Dra00] that this is possible. We also need to be able to compare integers of length $O(\sqrt{n})$, but this need not be in-place so can also be done with $O(\sqrt{n})$ ancilla qubits.

1. Let t denote the number in the register currently holding z_i. First update $t\leftarrow t+D/2$ (so that this register now holds $z_i+D/2$).
2. Set aside $K\lceil\log(r+1)\rceil$ ancilla qubits to hold $z_{i,j}$ for $j\in[K]$, so that $z_{i,j}=0$ for all j initially.
3. Now for each $j=K,K-1,\dots,1$ and for each $b=\lceil\log(r+1)\rceil-1,\dots,2,1,0$, check whether $t\geq 2^bG_j$ and write the output of this comparison to the bth qubit of the register containing $z_{i,j}$. Then use this qubit as a control qubit to conditionally update $t\leftarrow t-2^bG_j$.
4. By Lemma B.1, this greedy algorithm will correctly decompose $z_i+D/2$ as a linear combination $\sum_{j=1}^K z_{i,j}G_j$ of the G_j's, and we will have $t=0$ at the end. Hence we have freed up those $\log D$ bits as ancilla qubits to use in later steps.

We have already observed that correctness follows from Lemma B.1. For the runtime, each step of the innermost loop over j uses $O(\lceil\log(r+1)\rceil\sqrt{n})$ gates. So, each step of the outer loop over i uses $O(K\lceil\log(r+1)\rceil\sqrt{n})=O(A\lceil\log(r+1)\rceil n/\log\beta)=O(n)$ gates (where the prefactor in this bound is independent of r since $\lim_{r\to\infty}\frac{\lceil\log(r+1)\rceil}{\log\beta}=1$). Finally, multiplying by $d=\sqrt{n}$ yields a gate complexity of $O(n^{3/2})$.

Finally, we address space. All individual steps in the loop can clearly be done using $O(\sqrt{n})$ ancilla qubits (which we can then reuse). Other than that, each step of the loop consumes $K\lceil\log(r+1)\rceil$ ancilla qubits but then frees up $\log D$ ancilla qubits. The total initial ancilla requirement is hence $d(K\lceil\log(r+1)\rceil-\log D)+O(\sqrt{n})$ as desired. □

Lemma 3.2. *(Adapted from [Reg23]) There exists a quantum circuit using $O(r\sqrt{n}\log^3 n)$ gates mapping*

$$|t_1\rangle\dots|t_d\rangle\,|0^{\widetilde{O}(r\sqrt{n})}\rangle \mapsto |t_1\rangle\dots|t_d\rangle\left|\prod_{i=1}^d a_i^{t_i}\right\rangle|0^{\widetilde{O}(r\sqrt{n})}\rangle\,.$$

Here, the $t_i\in\{0,1,\dots,r\}$ for all i.

Proof. The proof by [Reg23] shows that such a computation can be done classically with $O(d \log^3 d) = O(\sqrt{n} \log^3 n)$ gates in the case that $t_i \in \{0, 1\}$; we just adapt this to our slightly more general setting. Let $M(k)$ denote the number of gates needed to classically multiply two k-bit integers. Using the multiplication circuit by [HvdH21], we may take $M(k) = O(k \log k)$.

Our classical circuit will first compute $a_i^{t_i}$ for each $i \in [d]$ using repeated squaring. For each i, this involves $O(\log r)$ multiplications of integers of length $O(r \log n)$. Hence the total cost here is $O(\sqrt{n} \cdot \log r \cdot M(r \log n)) = O(\sqrt{n} \cdot \log r \cdot r \log n \log \log n) = O_r(\sqrt{n} \log n \log \log n)$.

Now we follow Regev's procedure [Reg23] exactly using the integers $a_1^{t_1}, \ldots, a_d^{t_d}$: we organize them into a binary tree and recursively calculate their product accordingly. At the bth level, for $b \in [\log d]$, we will need to carry out $d/2^b$ multiplications of integers of length $2^b r \log n$. The number of gates is hence:

$$\begin{aligned}\sum_{b \in [\log d]} \frac{d}{2^b} \cdot M(2^b r \log n) &= O\left(\sum_{b \in [\log d]} dr \log n (b + \log \log n) \right) \\ &= O(dr \log^3 n) \\ &= O(r\sqrt{n} \log^3 n),\end{aligned}$$

which is the dominant term as desired. This classical circuit can be implemented quantumly, as explained in Lemma 5.6 of [RV24]. □

Since we are concerned with concrete costs in this note, we remark that the procedure in Lemma 3.2 is still a lower-order cost in terms of the number of gates and qubits if we use schoolbook multiplication everywhere i.e. we take $M(k) = O(k^2)$. Indeed, the number of gates in the first part would be $O(\sqrt{n} \cdot \log r \cdot (r \log n)^2) = O_r(\sqrt{n} \log^2 n)$, and the number of gates in the second part would be:

$$\begin{aligned}\sum_{b \in [\log d]} \frac{d}{2^b} \cdot (2^b r \log n)^2 &= O\left(\sum_{b \in [\log d]} d \cdot 2^b r^2 \log^2 n \right) \\ &= O(r^2 n \log^2 n).\end{aligned}$$

This has to be done $\log D = O(\sqrt{n})$ times, so the total number of gates used by calls to Lemma 3.2 would be $O(r^2 n^{3/2} \log^2 n)$. When using schoolbook multiplication, the total number of gates in the remainder of Regev's circuit will be $O(n^{5/2})$ which is hence the dominant cost. We note that even with schoolbook multiplication, the space usage of Lemma 3.2 can be kept down to $\widetilde{O}(r\sqrt{n})$; this is because any ancilla qubits used for multiplications can be reused between multiplications.

3.4 Large-Integer Multiplications

We begin by restating the primitives for large-integer multiplications constructed by [RV24], but additionally state the number of calls made to the multiplication circuit assumed in Theorem 2.1.

Lemma 3.3. *(Lemma 5.1 in [RV24], adapted from Shor [Sho97]) Let $a \in [0, N-1]$ be an integer coprime to N. Then there exists a circuit using $(3 + o(1))G$ gates mapping*

$$|x\rangle |0^{S+n}\rangle |g\rangle \mapsto |ax \bmod N\rangle |0^{S+n}\rangle |(-a^{-1}g) \bmod N\rangle$$

for any integers x, g that are reduced mod N.

Note that this computation uses and restores $S + n$ clean ancilla qubits, while applying some reversible transformation to n dirty ancilla qubits that initially store the state $|g\rangle$.

Lemma 3.4. *(Lemma 5.2 in [RV24]) There exists a quantum circuit using $(6 + o(1))G$ gates such that, for all n-bit integers $a, b, g \in [0, N-1]$ such that a and b are coprime to N, it will map*

$$\begin{aligned}|a\rangle |a^{-1} \bmod N\rangle |b\rangle |b^{-1} \bmod N\rangle |g\rangle |0^S\rangle \\ \mapsto |a\rangle |a^{-1} \bmod N\rangle |ab \bmod N\rangle |(ab)^{-1} \bmod N\rangle |g\rangle |0^S\rangle .\end{aligned}$$

Note that this computation uses and restores S clean ancilla qubits and n dirty ancilla qubits.

We also state straightforward simplifications of these results that use slightly more space but are $\approx 1.5\times$ more efficient in terms of the number of multiplications:

Lemma 3.5. *(Essentially due to Shor [Sho97], compare with Lemma 3.3) Let $a \in [0, N-1]$ be an integer coprime to N. Then there exists a circuit using $(2 + o(1))G$ gates mapping*

$$|x\rangle |0^{S+2n}\rangle \mapsto |ax \bmod N\rangle |0^{S+2n}\rangle$$

for any integer x that is reduced mod N.

Proof. We can classically precompute $a^{-1} \bmod N$ efficiently using the extended Euclidean algorithm. Now proceed as follows; we omit $\bmod N$ throughout for brevity:

$$\begin{aligned}|x\rangle |0^n\rangle |0^n\rangle |0^S\rangle &\to |x\rangle |a\rangle |0^n\rangle |0^S\rangle \text{ (writing in a classical constant using some bit-flips)} \\ &\to |x\rangle |a\rangle |ax\rangle |0^S\rangle \\ &\to |x\rangle |-a^{-1}\rangle |ax\rangle |0^S\rangle \text{ (writing in a classical constant again)} \\ &\to |x - a^{-1} \cdot ax\rangle |-a^{-1}\rangle |ax\rangle |0^S\rangle \\ &= |0^n\rangle |-a^{-1}\rangle |ax\rangle |0^S\rangle \\ &\to |0^n\rangle |0^n\rangle |ax\rangle |0^S\rangle \text{ (writing in a classical constant again)} \\ &\to |ax\rangle |0^n\rangle |0^n\rangle |0^S\rangle\end{aligned}$$

This runs our multiplication circuit twice, and the remaining operations are just $O(n)$ bit flips and bit swaps. This completes our proof. □

Lemma 3.6. *(Compare with Lemma 3.4) There exists a quantum circuit using $(4 + o(1))G$ gates such that, for all n-bit integers $a, b \in [0, N-1]$ such that a and b are coprime to N, it will map*

$$\begin{aligned}|a\rangle |a^{-1} \bmod N\rangle |b\rangle |b^{-1} \bmod N\rangle |0^{S+n}\rangle \\ \mapsto |a\rangle |a^{-1} \bmod N\rangle |ab \bmod N\rangle |(ab)^{-1} \bmod N\rangle |0^{S+n}\rangle .\end{aligned}$$

Proof. We proceed as follows:

$$\begin{aligned}|a\rangle |a^{-1}\rangle |b\rangle |b^{-1}\rangle |0^n\rangle |0^S\rangle &\to |a\rangle |a^{-1}\rangle |b\rangle |b^{-1}\rangle |ab\rangle |0^S\rangle \\ &\to |a\rangle |a^{-1}\rangle |b - a^{-1} \cdot ab\rangle |b^{-1}\rangle |ab\rangle |0^S\rangle\end{aligned}$$

$$
\begin{aligned}
&= |a\rangle\,|a^{-1}\rangle\,|0^n\rangle\,|b^{-1}\rangle\,|ab\rangle\,|0^S\rangle \\
&\to |a\rangle\,|a^{-1}\rangle\,|a^{-1}b^{-1}\rangle\,|b^{-1}\rangle\,|ab\rangle\,|0^S\rangle \\
&\to |a\rangle\,|a^{-1}\rangle\,|a^{-1}b^{-1}\rangle\,|b^{-1} - a\cdot a^{-1}b^{-1}\rangle\,|ab\rangle\,|0^S\rangle \\
&= |a\rangle\,|a^{-1}\rangle\,|(ab)^{-1}\rangle\,|0^n\rangle\,|ab\rangle\,|0^S\rangle \\
&\to |a\rangle\,|a^{-1}\rangle\,|ab\rangle\,|(ab)^{-1}\rangle\,|0^n\rangle\,|0^S\rangle\,.
\end{aligned}
$$

This runs our multiplication circuit four times and then does a constant number of swaps of n-bit registers at the end, for a total of $O(G+n)$ gates. This completes our proof. □

Finally, the following lemma generalizes Lemma 3.6 and is the key workhorse for the quantum circuits we present in this work. Intuitively, to compute $(a,b)\mapsto(a,a^r b)$, we can use repeated squaring as in Regev's circuit [Reg23]. This incurs a space overhead but allows us to save gates. However, we clean up the intermediate registers to prevent the space overheads from accumulating.

Lemma 3.7. *There exists a quantum circuit using $(4r/s + 4\log s + o(1))G$ gates such that, for all n-bit integers $a, b \in [0, N-1]$ such that a and b are coprime to N, it will map*

$$
\begin{aligned}
&|a\rangle\,|a^{-1} \bmod N\rangle\,|b\rangle\,|b^{-1} \bmod N\rangle\,|0^{S+(4\log s+1)n}\rangle \\
&\qquad\mapsto |a\rangle\,|a^{-1} \bmod N\rangle\,|a^r b \bmod N\rangle\,|(a^r b)^{-1} \bmod N\rangle\,|0^{S+(4\log s+1)n}\rangle\,.
\end{aligned}
$$

When $r = s = 1$, this is equivalent to Lemma 3.6.

Proof. All computations are mod N, which we omit for brevity. First, observe that if we have a value $|c\rangle$ in a certain register (for $c \in [0, N-1]$ coprime to N), we can carry out the following computation:

$$
|c\rangle\,|0^{S+2n}\rangle \mapsto |c\rangle\,|c\rangle\,|0^{S+n}\rangle \mapsto |c\rangle\,|c\rangle\,|c^2\rangle\,|0^S\rangle\,,
$$

using $(1+o(1))G$ gates[6].

Iterating this, we obtain the state

$$
|a\rangle^{\otimes 2}\,|a^{-1}\rangle^{\otimes 2}\,|a^2\rangle^{\otimes 2}\,|a^{-2}\rangle^{\otimes 2}\,|a^4\rangle^{\otimes 2}\,|a^{-4}\rangle^{\otimes 2}\ldots|a^{s/2}\rangle^{\otimes 2}\,|a^{-s/2}\rangle^{\otimes 2}\,|a^s\rangle\,|a^{-s}\rangle\,|b\rangle\,|b^{-1}\rangle\,|0^{S+n}\rangle\,,
$$

in $(2\log s + o(1))G$ gates. Next, we apply Lemma 3.6 r/s times to the last five registers in the above expression, to obtain the state

$$
|a\rangle^{\otimes 2}\,|a^{-1}\rangle^{\otimes 2}\,|a^2\rangle^{\otimes 2}\,|a^{-2}\rangle^{\otimes 2}\,|a^4\rangle^{\otimes 2}\,|a^{-4}\rangle^{\otimes 2}\ldots|a^{s/2}\rangle^{\otimes 2}\,|a^{-s/2}\rangle^{\otimes 2}\,|a^s\rangle\,|a^{-s}\rangle\,|a^r b\rangle\,|(a^r b)^{-1}\rangle\,|0^{S+n}\rangle\,,
$$

using $(4r/s + o(1))G$ gates. Finally, we uncompute the terms involving $a^2, a^4, \ldots, a^s$ and their inverses, using another $(2\log s + o(1))G$ gates. □

[6]The additional copying of c is to conform to the syntax of our multiplication circuit, where two input registers are required. If one works with a circuit for out-of-place squaring mod N that does not need to pseudo-copy its input, then this space overhead would not be needed.

4 Our Algorithms

4.1 Proof of Theorem 2.4

Our algorithm is shown in Algorithm 4.1, and makes two key changes to Algorithm 5.2 in [RV24]: it carries out simpler arithmetic operations via Lemma 3.7 to save gates, and additionally works with the G_k basis instead of the F_k basis. Our exponentiation procedure can hence thought of as a hybrid between those by [Reg23] and [RV24]; the high-level steps follow a Fibonacci-type exponentiation approach to avoid consuming significant space, however internally we rely on Lemma 3.7 which uses repeated out-of-place squaring.

Algorithm 4.1: Quantum oracle for $\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N$

Data: Initial state $|z\rangle$ and $S + (\alpha\lceil\log(r+1)\rceil - A + \max(4\log s + 5, 7) + o(1))n$ ancilla qubits in the $|0\rangle$ state.

Result: Final state comprising $|\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N\rangle$ and $S + (\alpha\lceil\log(r+1)\rceil + \max(4\log s + 4, 6) + o(1))n$ qubits in some state (which may not be $|0\rangle$).

1. Use Lemma 3.1 to compute and store the values $|z_{i,j}\rangle$ for all $i \in [d]$ and $j \in [K]$. Note that this step also "overwrites" the qubits storing $|z\rangle$. (This consumes $dK\lceil\log(r+1)\rceil - d\log D = (\alpha\lceil\log(r+1)\rceil - A + o(1))n$ qubits, leaving $S + (\max(4\log s + 5, 7) + o(1))n$ ancilla qubits.)

2. Set aside $4n$ ancilla qubits. These will store our states $|\psi(x_1)\rangle$ and $|\psi(x_2)\rangle$. Initialize $x_1 \leftarrow 1$ and $x_2 \leftarrow 1$. (This leaves $S + (\max(4\log s + 1, 3) + o(1))n$ ancilla qubits.)

3. Repeat the following for $j = K, K-1, \ldots, 1$ in that order:

   (a) Update $x_1 \leftarrow x_1 x_2^r$ by applying Lemma 3.7. (This temporarily uses and restores $S + (4\log s + 1)n$ clean ancilla qubits, which we have.)

   (b) Now we prepare the state $|\psi(c_j)\rangle$:

      i. Calculate the state $|\prod_{i=1}^{d} a_i^{r-z_{i,j}} \bmod N\rangle$ using Lemma 3.2 and store it in an n-bit register. We now have $S + (\max(4\log s, 2) + o(1))n$ ancilla qubits available.

      ii. Use Lemma 3.5 with the classical constant $\prod_{i=1}^{d} a_i^{-r} \bmod N$ to update the register from the above step to contain $|c_j^{-1} \bmod N\rangle$. (This temporarily uses and restores $S + 2n$ ancilla qubits, which we have.)

      iii. Calculate the state $|c_j\rangle$ using Lemma 3.2 and store it in an n-bit register. We now have $S + (\max(4\log s - 1, 1) + o(1))n$ ancilla qubits available.

   (c) We now have the state $|\psi(c_j)\rangle$, so we can update $x_1 \leftarrow x_1 c_j$ using Lemma 3.6. (This temporarily uses and restores $S + n$ ancilla qubits, which we have.)

   (d) Now we uncompute the state $|\psi(c_j)\rangle$, returning all qubits to $|0\rangle$.

   (e) Swap x_1 and x_2 (i.e. swap the registers $|\psi(x_1)\rangle$ and $|\psi(x_2)\rangle$).

We track the space inside the algorithm; it remains to account for large-integer multiplications.

The first call to Lemma 3.7 uses $(4r/s + 4\log s + o(1))G$ gates. Preparing the state $|\psi(c_j)\rangle$ only requires large-integer multiplications in the use of Lemma 3.5, which uses $(2 + o(1))G$ gates. The call to Lemma 3.6 uses $(4 + o(1))G$ gates, and then finally uncomputing $|\psi(c_j)\rangle$ uses $(2 + o(1))G$ gates. The total number of gates is hence:

$$(4r/s + 4\log s + 8 + o(1))K \cdot G = \frac{(4r/s + 4\log s + 8 + o(1))A}{\log \beta} \cdot G \cdot \sqrt{n}.$$

□

4.2 Proof of Theorem 2.2

We also show that the optimizations for space by [RV24] by using dirty ancilla qubits carry over to the G_k basis instead of the F_k basis. Our algorithm is shown in Algorithm 4.2, and is a direct adaptation of Algorithm 5.2 in [RV24] to the G_k basis.

We track the space inside the algorithm; it remains to account for large-integer multiplications. For each j, there are r calls to Lemma 3.6, each of which uses $(4 + o(1))G$ gates. Constructing $|\psi(c_j)\rangle$ uses $(3 + o(1))G$ gates from the call to Lemma 3.3. The call to Lemma 3.4 uses $(6 + o(1))G$ gates, and then finally uncomputing $|\psi(c_j)\rangle$ uses another $(3 + o(1))G$ gates. The total number of gates used by our oracle is hence:

$$(4r + 12 + o(1))K \cdot G = \frac{(4r + 12 + o(1))A}{\log \beta} \cdot G \cdot \sqrt{n}.$$

□

Acknowledgements. I am grateful to Vinod Vaikuntanathan for introducing me to the problem of optimizing Regev's algorithm, and for guidance and numerous helpful discussions throughout this project. I would also like to thank Gregory D. Kahanamoku-Meyer, Katherine van Kirk, Martin Ekerå, and Joel Gärtner for insightful comments and discussions, some of which inspired the ideas in this note. This work was supported by an Akamai Presidential Fellowship.

References

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel,

Algorithm 4.2: Space-optimized quantum oracle for $\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N$, adapted from [RV24]

Data: Initial state $|z\rangle$ and $S+(\alpha\lceil\log(r+1)\rceil-A+6+o(1))n$ ancilla qubits in the $|0\rangle$ state.

Result: Final state comprising $|\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N\rangle$ and $S+(\alpha\lceil\log(r+1)\rceil+5+o(1))n$ qubits in some state (which may not be $|0\rangle$).

1. Use Lemma 3.1 to compute and store the values $|z_{i,j}\rangle$ for all $i \in [d]$ and $j \in [K]$. Note that this step also "overwrites" the qubits storing $|z\rangle$. (This consumes $dK\lceil\log(r+1)\rceil - d\log D = (\alpha\lceil\log(r+1)\rceil - A + o(1))n$ qubits, leaving $S+(6+o(1))n$ ancilla qubits.)

2. Set aside $4n$ ancilla qubits. These will store our states $|\psi(x_1)\rangle$ and $|\psi(x_2)\rangle$. Initialize $x_1 \leftarrow 1$ and $x_2 \leftarrow 1$. (This leaves $S+(2+o(1))n$ ancilla qubits.)

3. Repeat the following for $j = K, K-1, \ldots, 1$ in that order:

   (a) Consider the qubits comprising $z_{i,j'}$ for $i \in [d]$ and $j' \neq j$. There are $d(K-1)\lceil\log(r+1)\rceil \geq (\alpha - o(1))\lceil\log(r+1)\rceil n \geq (A\lceil\log(r+1)\rceil/\log\beta - o(1))n > 2n$ such qubits (noting that $A > 2$ and $r+1 > \beta$), and we will not use any of them for this iteration of the loop. Hence we may take $n-1$ of these qubits and pre-pend one clean qubit in the $|0\rangle$ state to obtain n dirty ancilla qubits. We now have $S+2n$ clean ancilla qubits available.

   (b) Update $x_1 \leftarrow x_1 x_2^r$ by applying Lemma 3.6 r times. (This temporarily uses and restores $S+n$ clean ancilla qubits, which we have.)

   (c) Now we prepare the state $|\psi(c_j)\rangle$:

      i. Calculate the state $|\prod_{i=1}^{d} a_i^{r-z_{i,j}} \bmod N\rangle$ using Lemma 3.2 and store it in an n-bit register. We now have $S+n$ ancilla qubits available.

      ii. Use Lemma 3.3 with the classical constant $\prod_{i=1}^{d} a_i^{-r} \bmod N$ to update the register from the above step to contain $|c_j^{-1} \bmod N\rangle$. (This uses and restores all $S+n$ clean ancilla qubits, as well as modifying our n dirty ancilla qubits.)

      iii. Calculate the state $|c_j\rangle$ using Lemma 3.2 and store it in an n-bit register. We now have S clean ancilla qubits available.

   (d) We now have the state $|\psi(c_j)\rangle$, so we can update $x_1 \leftarrow x_1 c_j$ using Lemma 3.4. (This uses and restores S clean ancilla qubits and n dirty ancilla qubits.)

   (e) Now we uncompute the state $|\psi(c_j)\rangle$, returning all qubits to $|0\rangle$. (Note that this will return all dirty ancilla qubits to their original value, since they are only modified in the construction of $|\psi(c_j)\rangle$ when we use Lemma 3.3.)

   (f) Swap x_1 and x_2 (i.e. swap the registers $|\psi(x_1)\rangle$ and $|\psi(x_2)\rangle$).

Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, October 2019. 2

[Bea03] Stéphane Beauregard. Circuit for Shor's algorithm using 2n+3 qubits. *Quantum Inf. Comput.*, 3(2):175–185, 2003. 1, 3

[Cai23] Jin-Yi Cai. Shor's algorithm does not factor large integers in the presence of noise. *CoRR*, abs/2306.10072, 2023. 2

[CCHL22] Sitan Chen, Jordan Cotler, Hsin-Yuan Huang, and Jerry Li. The complexity of NISQ. *CoRR*, abs/2210.07234, 2022. 2

[Dra00] Thomas G Draper. Addition on a quantum computer. *arXiv preprint quant-ph/0008033*, 2000. 9

[EG24] Martin Ekerå and Joel Gärtner. Extending Regev's factoring algorithm to compute discrete logarithms, 2024. 2, 3

[GE21] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021. 2, 7

[Gid17] Craig Gidney. Factoring with $n + 2$ clean qubits and $n - 1$ dirty qubits. *arXiv preprint arXiv:1706.07884*, 2017. 1, 3

[Gid19] Craig Gidney. Windowed quantum arithmetic, 2019. 1, 3, 7

[HRS17] Thomas Häner, Martin Roetteler, and Krysta M. Svore. Factoring using $2n + 2$ qubits with Toffoli based modular multiplication. *Quantum Inf. Comput.*, 17(7&8):673–684, 2017. 1, 3

[HvdH21] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2), March 2021. 1, 10

[Kal17] Burton S. Kaliski Jr. Targeted Fibonacci exponentiation. *arXiv preprint arXiv:1711.02491*, 2017. 2, 6

[Reg23] Oded Regev. An efficient quantum factoring algorithm. *arXiv preprint arXiv:2308.06572*, 2023. 1, 2, 3, 4, 6, 7, 8, 9, 10, 12, 13

[RV24] Seyoon Ragavan and Vinod Vaikuntanathan. Space-efficient and noise-robust quantum factoring. 2024. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. 1, 3, 6, 10, 11

[SS71] Arnold Schönhage and Volker Strassen. Fast multiplication of large numbers. *Computing*, 7:281–292, 1971. 1

[TK06] Yasuhiro Takahashi and Noboru Kunihiro. A quantum circuit for Shor's factoring algorithm using 2n+ 2 qubits. *Quantum Information & Computation*, 6(2):184–192, 2006. 1, 3

[Zal06] Christof Zalka. Shor's algorithm with fewer (pure) qubits, 2006. 1, 3

[Zec72] Édouard Zeckendorf. Representations of natural numbers by a sum of Fibonacci numbers and Lucas numbers. *Bulletin of the Royal Society of Sciences of Liege*, pages 179–182, 1972. 18

A Proof of Correctness of our Algorithms

First we check the base case i.e. the first two rounds where $j = K, K-1$. In round K, (x_1, x_2) evolves as follows: $(1,1) \to (1,1) \to (c_K, 1) \to (1, c_K)$. Then in round $K-1$, it evolves as follows: $(1, c_K) \to (c_K^r, c_K) \to (c_{K-1}c_K^r, c_K) \to (c_K, c_{K-1}c_K^r)$. This is consistent with our claim for $j = K-1$.

For the inductive step, assume the current round is j, and the previous round (indexed by $j+1$) has ended according to our claim. The current state is hence:

$$(x_1, x_2) = \left(\prod_{i=j+2}^{K} c_i^{G_{i-j-1}}, \prod_{i=j+1}^{K} c_i^{G_{i-j}} \right).$$

After the first update, x_1 becomes:

$$\begin{aligned} \prod_{i=j+2}^{K} c_i^{G_{i-j-1}} \cdot \prod_{i=j+1}^{K} c_i^{rG_{i-j}} &= c_{j+1}^{rG_1} \cdot \prod_{i=j+2}^{K} c_i^{G_{i-j-1}+rG_{i-j}} \\ &= c_{j+1}^{G_2} \cdot \prod_{i=j+2}^{K} c_i^{G_{i-j+1}} \\ &= \prod_{i=j+1}^{K} c_i^{G_{i-j+1}}. \end{aligned}$$

After the second update, it becomes:

$$c_j \cdot \prod_{i=j+1}^{K} c_i^{G_{i-j+1}} = \prod_{i=j}^{K} c_i^{G_{i-j+1}}.$$

Swapping the registers now gives us the state

$$(x_1, x_2) = \left(\prod_{i=j+1}^{K} c_i^{G_{i-j}}, \prod_{i=j}^{K} c_i^{G_{i-j+1}} \right),$$

completing our induction. □

B Properties of $\{G_k\}$

Recall that G_k is defined by $G_0 = 0$, $G_1 = 1$, and $G_k = rG_{k-1} + G_{k-2}$ for $k > 1$.

B.1 Closed Form and Asymptotics

A straightforward induction tells us that for any $k \geq 0$ we have:

$$\begin{aligned} G_k &= \frac{1}{\sqrt{r^2+4}}\left(\frac{r+\sqrt{r^2+4}}{2}\right)^k - \frac{1}{\sqrt{r^2+4}}\left(\frac{r-\sqrt{r^2+4}}{2}\right)^k \\ &= \frac{1}{\sqrt{r^2+4}}\beta^k + o(1). \end{aligned}$$

Hence we will have

$$K = (1+o(1))\frac{\log D}{\log \beta}.$$

We clearly have $G_k \leq G_{k+1}$ for all $k \geq 0$. It follows that for $k > 0$ we have $G_{k+1} - rG_k = G_{k-1} \leq G_k \Leftrightarrow G_{k+1} \leq (r+1)G_k$.

B.2 Decomposing Positive Integers as a Sum of G_k

It was shown by [Zec72] that any positive integer has a unique decomposition as a sum of Fibonacci numbers, if we enforce that no two of the Fibonacci numbers should be consecutive. A simple generalization of the greedy algorithm by [Zec72] yields the following lemma:

Lemma B.1. *Consider the following algorithm, that takes as input a non-negative integer $t_0 < G_{k+1}$. Initialize $t \leftarrow t_0$. Now for $j = k, k-1, \ldots, 1$, set $y_j = \lfloor t/G_j \rfloor$ and update $t \leftarrow t - y_j G_j$.*

Then at the end of the algorithm, we will have $t = 0$, $\sum_{j=1}^{k} y_j G_j = t_0$, and $y_j \in \{0, 1, \ldots, r\}$ for all j.

Proof. First, observe that the algorithm clearly ensures that $t \geq 0$ at all times, and that t and all the $y_j G_j$'s constructed thus far add to t_0. Hence it suffices to show that at the end of the algorithm, we will have $t = 0$ and that all the y_j's are in $[0, r]$.

We do this by strong induction on k. The base case $k = 0$ follows trivially since $t < G_1 = 1$ forces $t = 0$, so we are already done. We also consider the base case $k = 1$; in this case we have $t < G_2 = r$. Then we have $y_1 = \lfloor t/G_1 \rfloor = t \in [0, r]$, so we will have $y_1 G_1 = t$ and t will become 0.

Now for the inductive step, consider $k > 1$. We have two cases:

- If $t < G_k$, then in the $j = k$ round we will simply have $y_k = 0$, t will be unchanged, and we reduce directly to the $k-1$ case.
- If $t \geq G_k$, then in the $j = k$ round we will have $y_k = \lfloor t/G_k \rfloor$ and t will be replaced by $t - y_k G_k$. Firstly, observe that $t < G_{k+1} \leq (r+1)G_k \Rightarrow t/G_k < r+1 \Rightarrow y_k \leq r$. Secondly, we have by definition that $t - y_k G_k < G_k$, so we have now reduced to the $k-1$ case.

Either way, the conclusion follows by induction. □