# Optimizing Space in Regev's Factoring Algorithm

Seyoon Ragavan
MIT
sragavan@csail.mit.edu

Vinod Vaikuntanathan
MIT
vinodv@csail.mit.edu

October 2, 2023

**Abstract**

We improve the space efficiency of Regev's quantum factoring algorithm [Reg23] while keeping the circuit size the same. Our main result constructs a quantum factoring circuit using $O(n \log n)$ qubits and $O(n^{3/2} \log n)$ gates. In contrast, Regev's circuit requires $O(n^{3/2})$ qubits, while Shor's circuit requires $O(n^2)$ gates. As with Regev, to factor an $n$-bit integer $N$, one runs this circuit independently $\approx \sqrt{n}$ times and apply Regev's classical post-processing procedure.

Our optimization is achieved by implementing efficient and reversible exponentiation with Fibonacci numbers in the exponent, rather than the usual powers of 2.

## 1 Introduction

Shor's landmark result from 1994 [Sho97] showed us how to factor numbers in quantum polynomial time. In particular, he constructed an $\tilde{O}(n^2)$-size quantum circuit that uses $O(n \log n)$ qubits (including ancillas), such that $O(1)$ runs of his circuit followed by polynomial-time classical postprocessing suffices to factor an $n$-bit integer. A number of followup works [BCDP96, VBE96, Cop02, CW00, Bea03, TK06, EH17, Gid17, HRS17, Gid19, GE21] improved the Shor circuit in several ways, for example by constructing a log-depth implementation [CW00] and by reducing the number of ancilla qubits to a mere $2n+1$ [Gid17] (albeit with a larger number of gates, $\widetilde{\Theta}(n^5)$ and $\widetilde{\Theta}(n^3)$ respectively).

However, the size of the factoring circuit stood steady at $\tilde{O}(n^2)$ for nearly three decades, until Regev recently demonstrated an $\tilde{O}(n^{1.5})$-size quantum circuit for factoring. Regev shows how to factor $n$-bit integers using $\approx \sqrt{n}$ (parallel) runs of his factoring circuit. While the total number of operations remains the same, the smaller Regev circuit is presumably easier to build and resists decoherence noise better.

A key efficiency consideration in the construction of quantum circuits is their space complexity, namely, the number of qubits including ancillas used by the circuit. (For more discussion on the importance of reducing space complexity in superconducting architectures for quantum computation, see [GE21, Gid23].) The number of qubits used in the original Shor circuit was $O(n \log n)$. Several works [BCDP96, VBE96, Bea03, TK06, Gid17, HRS17, Gid19] optimized the space complexity down to $O(n)$ qubits (and in fact, even $2n+1$). However, none of these optimizations seems to apply to Regev's circuit whose space complexity stands at (the obvious) $\tilde{O}(n^{1.5})$.

In this work, we construct a quantum circuit that asymptotically achieves the best of both worlds: it has a size of $\widetilde{O}(n^{1.5})$ matching Regev's circuit; and it requires $\widetilde{O}(n)$ qubits, nearly matching what is known for optimized versions of Shor's circuit. We compute the concrete number of qubits required (see our Theorem 1 below) which, using schoolbook multiplication, leads us to $(12.32 + o(1))n$ qubits and $\widetilde{O}(n^{2.5})$ gates vis-a-vis the results of [BCDP96, VBE96, Bea03, TK06, Gid17, HRS17] which use $2n+1$ qubits and $\widetilde{O}(n^3)$ gates.

In a nutshell, a key step in Shor's algorithm is the computation of the map $|x\rangle \mapsto |a^x \bmod N\rangle$ in superposition, where $a$ is a *fixed* base. The fact that $a$ is fixed allows one to precompute its powers $a, a^2, \ldots, a^{2^i}, \ldots$ classically. Once this is done, exponentiation in superposition can be done with space $\tilde{O}(n)$ using fast multiplication [HvdH21] and *in-place* exponentiation by multiplying together the appropriate subset of the precomputed values. This technique goes out of the window in Regev's new quantum factoring circuit as observed by Ekerå and Gidney [Gid23], essentially because Regev's circuit crucially cannot use a fixed base $a$. Instead, if one tries to implement the map $|a\rangle |x\rangle \mapsto |a\rangle |a^x \bmod N\rangle$ via the classic square-and-multiply algorithm, one runs into the conundrum of implementing the squaring circuit modulo $N$. On the one hand, doing this reversibly and in-place seems as hard as factoring $N$ in the first place. On the other hand, writing the squared result to a new register consumes extra space; indeed, this is the source of the $O(n^{3/2})$ space requirement for Regev's circuit [Reg23].

We avoid this conundrum with our key technique of Fibonacci exponentation, a method of exponentiation that avoids modular squaring and instead relies solely on modular multiplication. We note that some previous works have considered using Fibonacci numbers rather than powers of two in the exponent [BMT+07, Kle08, Mel07], one of them with the explicit motivation of achieving fast and reversible exponentiation [Kal17]. While Fibonacci exponentiation seems to not be of much use in Shor's circuit, it turns out to be quite powerful in optimizing the space usage of Regev's circuit.

We now proceed to describe our results in more detail.

## 1.1 Our Results

We begin by stating our main theorem, and compare the result to Regev [Reg23] and optimizations of Shor's algorithm [Sho97, BCDP96, VBE96, Bea03, HRS17, TK06, Gid17, Gid19]. In a nutshell, the algorithm by [Reg23] requires $O(n^{3/2})$ qubits, which we bring down to $O(n \log n)$ while retaining the $\widetilde{O}(n^{3/2})$ quantum circuit size. Optimizations of Shor's algorithm that retain the $\widetilde{O}(n^2)$ circuit size require $O(n \log n)$ qubits to the best of our knowledge; thus, compared to these results, our algorithm works with the same number of qubits but achieves a $O(\sqrt{n})$ factor saving in the circuit size (just as [Reg23] does).

**Main Theorem 1.** *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 $G$ 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. (We do not make any assumptions about the circuit's behaviour on inputs where any of $a, b, t$ are $\ge N$.)*

*Under conjecture 1.1, there exists a classical polynomial-time algorithm that outputs a non-trivial factor of $N$ using $\sqrt{n} + 4$ calls to a quantum circuit on*

$$S + \left(\frac{C+2}{\log \phi} + 8 + o(1)\right) n \approx S + (1.44C + 10.88) \cdot n$$

*qubits using $O(n^{1/2} \cdot G + n^{3/2})$ gates, where $\phi = (1+\sqrt{5})/2$ is the golden ratio.*

We now state three corollaries of this general theorem. Using the classical $O(n \log n)$ multiplication algorithm due to [HvdH21] and implementing this using quantum unitary operators (see Appendix C for details) would allow us to set $G = S = O(n \log n)$ and obtain the following corollary:

**Corollary 1.1.** *Under the same assumption as Theorem 1, there is a quantum circuit for factoring that uses $O(n \log n)$ qubits and $O(n^{3/2} \log n)$ gates.*

If we want to push the number of qubits needed all the way down to $O(n)$, we can employ the space-efficient quantum implementation of Karatsuba's algorithm [KO62] due to Gidney [Gid19] which uses $S = O(n)$ ancilla qubits and $G = O(n^{\log_2 3})$ gates, yielding the following corollary:

**Corollary 1.2.** *Under the same assumption as Theorem 1, there is a quantum circuit for factoring that uses O(n) qubits and $O(n^{\log_2 3+1/2})$ gates.*

This latter result achieves the same space complexity as Gidney's version of Shor's algorithm [Gid19], except with $O(\sqrt{n})$ factor smaller number of gates.

Finally, if we want to use schoolbook multiplication because of its efficiency for integers of practical length, we can slightly modify the space-efficient quantum implementation of schoolbook multiplication due to [RNSL17] (see Appendix C for details about our modification). This algorithm uses $S = 2$ ancilla qubits and $G = O(n^2 \log n)$ gates, yielding the following corollary:

**Corollary 1.3.** *Under the same assumption as Theorem 1, there is a quantum circuit for factoring that uses*

$$\left(\frac{C+2}{\log \phi} + 8 + o(1)\right) n \approx (1.44C + 10.88) \cdot n$$

*qubits and $O(n^{5/2} \log n)$ gates.*

We compare these results with previous work on quantum circuits for factoring in Figure 1. We also detail the various sources of space cost in our algorithm and potential areas for further optimizations in Section 3.4.

**The Number-Theoretic Assumption.** We retain all notation from [Reg23] and restate it here for convenience. Let $N \leq 2^n$ be an $n$-bit number. Let $d = \sqrt{n}$ and $b_1, \dots, b_d$ be some small $O(\log n)$-bit integers (e.g. $b_i$ is the $i$th prime number) and let $a_i = b_i^2 \bmod N$. We use log to denote the base-2 logarithm throughout this paper. Also, let $\phi$ denote the golden ratio.

Regev [Reg23] defines the following lattices in $d$ dimensions:

$$\mathcal{L} = \left\{ (z_1, \dots, z_d) \in \mathbb{Z}^d : \prod_{i=1}^{d} a_i^{z_i} \equiv 1 \bmod N \right\}, \text{ and}$$

$$\mathcal{L}_0 = \left\{ (z_1, \dots, z_d) \in \mathbb{Z}^d : \prod_{i=1}^{d} b_i^{z_i} \equiv \pm 1 \bmod N \right\}.$$

We make the following heuristic assumption, the same as the one made in [Reg23].

**Conjecture 1.1.** *There exists a vector in $\mathcal{L} \backslash \mathcal{L}_0$ of $\ell_2$ norm at most $T = 2^{C\sqrt{n}}$, for some given constant $C > 0$.*

As observed by [Reg23], a simple pigeonhole principle argument shows that there exists a nonzero vector in $\mathcal{L}$ with norm at most $2^{(1+o(1))\sqrt{n}}$. Following Regev's heuristic argument hence suggests that taking $C = 1+\epsilon$ should be sufficient. Plugging this into the statement of Theorem 1 suggests that the space complexity of our circuit should be $\approx S + (12.32 + o(1))n$.

| Algorithm | Mult. algorithm | Number of qubits | Circuit size |
|---|---|---|---|
| Shor [Sho97] | [HvdH21] | $O(n \log n)$ | $O(n^2 \log n)$ |
| Shor | Schoolbook | $\boldsymbol{O(n)}$ | $O(n^3)$ |
| Shor | [Gid19] | $\boldsymbol{O(n)}$ | $O(n^{\log_2 3+1})$ |
| Optimized Shor [Bea03, TK06, Gid17, HRS17] | Schoolbook | $\boldsymbol{2n + O(1)}$ | $\widetilde{O}(n^3)$ |
| Regev's algorithm [Reg23] | [HvdH21] | $O(n^{3/2})$ | $\boldsymbol{O(n^{3/2} \log n)}$ |
| Our optimization of Regev | [HvdH21] | $O(n \log n)$ | $\boldsymbol{O(n^{3/2} \log n)}$ |
| Our optimization of Regev | [Gid19] | $\boldsymbol{O(n)}$ | $O(n^{\log_2 3+1/2})$ |
| Our optimization of Regev | Schoolbook [RNSL17] | $\boldsymbol{(1.44C + 10.88 + o(1))n}$ | $O(n^{5/2} \log n)$ |

Figure 1: Comparison of our results with previous work. Asymptotically optimal results for either space or size are highlighted in bold. Note, importantly, that all values here are just for *one run of the circuit*. They do not account for the fact that a circuit for Shor's algorithm only requires $O(\log n)$ independent runs, while a circuit for Regev's algorithm as well as ours requires $\sqrt{n} + 4$ independent runs.

## 2 Overview of the Quantum Part of the Algorithm

Our only modification to Regev's algorithm is in the quantum part; we refer the reader to [Reg23] for a discussion of the classical post-processing. We summarize the three parts of the quantum algorithm in the following sections.

Define $R = 2^{(C+2+o(1))\sqrt{n}}$ and let $D$ be a power of 2 in $[2\sqrt{d} \cdot R, 4\sqrt{d} \cdot R]$. Note that $D$ is also $2^{(C+2+o(1))\sqrt{n}}$. These are the same parameters $R$ and $D$ defined by [Reg23]. Regev defines $R$ to be $2^{C'\sqrt{n}}$ for sufficiently large $C'$ and then $D$ with the same definition that we use; the only difference here is that we explicitly specify $C' = C + 2 + o(1)$ in order to nail down the space complexity of our circuit within $1 + o(1)$ multiplicative error. See Appendix A for a justification that this choice of $C'$ suffices.

To help keep track of our algorithm's space usage, all lemmas will clearly specify the number of ancilla qubits needed (if any).

### 2.1 Constructing a Superposition over Vectors $z$

This proceeds exactly as in [Reg23]; we simply restate the output of this part and its space and size guarantees. For $s > 0$, define the Gaussian function $\rho_s : \mathbb{R}^d \to \mathbb{R}$ as

$$\rho_s(z) = \exp(-\pi||z||^2/s^2).$$

Then in this step, the algorithm constructs a discrete Gaussian state $|\psi\rangle$ within $1/\text{poly}(d)$ trace distance of the state proportional to:

$$\sum_{z \in \{-D/2,\ldots,D/2-1\}^d} \rho_R(z) \left| z \right\rangle .$$

For our purposes, we need the following lemma from [Reg23].

**Lemma 2.1.** *$|\psi\rangle$ can be constructed in-place (that is, without any ancilla qubits) with*

$$O(d(\log D + \mathrm{poly}(\log d))) = O(n)$$

*gates. The number of qubits needed to store the vector $|z\rangle$ is $d \log D = (C + 2 + o(1))n$.*

## 2.2 A Quantum Oracle to Compute $\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N$

Next, the algorithm computes (in superposition)

$$\prod_{i=1}^{d} a_i^{z_i+D/2} \pmod N$$

using a classical procedure (the offset by $D/2$ in the exponent is to make all exponents non-negative).

Recall that we use $G$ to denote the number of gates used by our $n$-bit multiplication circuit; see the statement of Theorem 1 for a precise description. Then Regev achieves this step with $O(\log D \cdot G)$ gates by using a repeated squaring procedure, while also exploiting the fact that the $a_i$'s are small to reduce the number of large-integer multiplications required. Our main technical improvement is stated in the following lemma; we devote Section 3 to its proof.

**Lemma 2.2.** *As in Theorem 1, let $G$ and $S$ be the number of gates and ancilla qubits respectively for our multiplication circuit on $n$-bit integers. Then there exists a quantum circuit mapping*

$$|z\rangle\,|0^A\rangle \mapsto |z\rangle\,|\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N\rangle\,|0^{A-n}\rangle\,.$$

*Here,*

$$A = S + \left(\frac{C+2}{\log\phi} - C + 6 + o(1)\right)n$$

*is the initial number of ancilla qubits. Moreover, this circuit uses $O(n^{1/2} \cdot G + n^{3/2})$ gates. Therefore, the total space usage here is*

$$S + \left(\frac{C+2}{\log\phi} + 8 + o(1)\right)n$$

*qubits.*

## 2.3 Measurement and QFT

This final stage also proceeds exactly as in [Reg23]. The algorithm measures the register storing $\prod_{i=1}^{d} a_i^{z_i+D/2} \pmod N$ to collapse the $|z\rangle$ register to a superposition over some coset of $\mathcal{L}$. The algorithm then applies an approximate QFT [Cop02] to the $|z\rangle$ register and measures to obtain a vector close to the dual lattice $\mathcal{L}^*$. This is the output of the quantum part of the algorithm. What we need is the following lemma from [Reg23, Cop02].

**Lemma 2.3.** *The approximate QFT can be computed inplace with circuit size*

$$O(d \log D(\log\log D + \log d)) = O(n \log n)$$

*gates. The space usage here is the number of qubits needed to store $|z\rangle$, which is $d \log D = (C + 2 + o(1))n$ qubits.*

## 3 Proof of Lemma 2.2

We construct the quantum oracle that computes the desired mapping in Lemma 2.2.

### 3.1 Reversible Fibonacci exponentiation

This is the key idea of our optimization. The bottleneck in [Reg23]'s construction of this oracle is in the use of repeated squaring. As a simple example, suppose we have some $n$-bit integer $|a\rangle$ and we wish to compute $|a^{2^k} \bmod N\rangle$ for some $k$. The natural classical-inspired approach would be to use repeated squaring as follows:

$$|a\rangle \mapsto |a^2 \bmod N\rangle \mapsto |a^4 \bmod N\rangle \mapsto \ldots \mapsto |a^{2^k} \bmod N\rangle .$$

The problem is that none of these operations are easily reversible; computing square roots mod $N$ is hard, and there is also the fact that squaring mod $N$ is not one-to-one. Thus carrying out this computation requires storing all of these quantities in separate registers. This leads to an $O(k)$ blow-up in the space complexity, which appears wasteful.

To get around this, we use the fact that it is much easier to implement reversible multiplication than squaring. Concretely, if we could implement an operation resembling $|a\rangle |b\rangle \mapsto |a\rangle |ab \bmod N\rangle$, we could proceed as follows:

$$|a\rangle |a\rangle \mapsto |a\rangle |a^2 \bmod N\rangle \mapsto |a^3 \bmod N\rangle |a^2 \bmod N\rangle \mapsto |a^3 \bmod N\rangle |a^5 \bmod N\rangle \mapsto \ldots$$

We now find Fibonacci numbers rather than powers of 2 in the exponent. For this reason, previous works have also considered using Fibonacci numbers rather than powers of 2 in the exponent for fast and reversible exponentiation [BMT+07, Kal17, Kle08, Mel07].

Notice that when using this idea, we will have an extra register lingering around. For example, if our goal is to compute $|a^5 \bmod N\rangle$ in one register, this would give us $|a^3 \bmod N\rangle$ in the other register. We would like to be able to clean this up, but fortunately this is straightforward; we can copy the final output to a new register, and then uncompute our circuit to clean up all intermediate qubits. This will just leave us with the inputs and outputs of our computation. We will use this idea repeatedly in our algorithm.

We now state the critical lemmas that formalize this intuition. As in Theorem 1, we assume throughout that we have a multiplication circuit that maps

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

where $a$, $b$, $t$ are all $n$-bit integers and $0 \leq t < N$. $S$ hence denotes the number of ancilla qubits used by the circuit, and we also use $G$ to denote the number of gates in this circuit.

Our lemmas will also make use of some "dirty ancillas" that may not necessarily be in the $|0\rangle$ state. This will allow us to reuse qubits from other parts of our algorithm and cut down on the number of qubits needed in our circuit. The idea of using dirty ancillas to achieve space savings was introduced by [HRS17] in relation to implementing Shor's algorithm with fewer qubits.

The first lemma we need is essentially due to Shor [Sho97].

**Lemma 3.1.** *Let $a \in [0, N-1]$ be an integer coprime to $N$. Then there exists a circuit using $O(G + n)$ 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 $N$.*

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

*Proof.* We can classically precompute $a^{-1} \bmod N$ efficiently using the extended Euclidean algorithm. Now proceed as follows using our quantum multiplication circuit given in Theorem 1:

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

This runs our multiplication circuit three times and does $O(n)$ bit flips and bit swaps, for a total of $O(G+n)$ gates. This completes our proof. □

We also adapt this idea to the case where $a$ may be a superposition of integers rather than a classical constant in the next lemma.

**Lemma 3.2.** *There exists a quantum circuit using $O(G+n)$ 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*[1]

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

*Hence this computation uses and restores $S$ clean ancillas and $n$ dirty ancillas.*

*Proof.* The quantum multiplication circuit assumed in Theorem 1 allows us the following computation when $0 \le t < N$:

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

We can also hence use the inverse of this circuit, which will behave as follows when $0 \le t < N$:

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

[1]We are not concerned about the circuit's behavior on other basis states.

We can use these two circuits to proceed as follows.

$$\begin{aligned}
&\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|b\right\rangle\left|b^{-1} \bmod N\right\rangle\left|g\right\rangle\left|0^S\right\rangle\\
\to&\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|b\right\rangle\left|b^{-1} \bmod N\right\rangle\left|(g+ab) \bmod N\right\rangle\left|0^S\right\rangle\\
\to&\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|(b-(a^{-1}\cdot(g+ab))) \bmod N\right\rangle\left|b^{-1} \bmod N\right\rangle\left|(g+ab) \bmod N\right\rangle\left|0^S\right\rangle\\
&\qquad=\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|(-a^{-1}g) \bmod N\right\rangle\left|b^{-1} \bmod N\right\rangle\left|(g+ab) \bmod N\right\rangle\left|0^S\right\rangle\\
\to&\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|(-a^{-1}g) \bmod N\right\rangle\left|b^{-1} \bmod N\right\rangle\left|((g+ab)+a\cdot(-a^{-1}g)) \bmod N\right\rangle\left|0^S\right\rangle\\
&\qquad=\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|(-a^{-1}g) \bmod N\right\rangle\left|b^{-1} \bmod N\right\rangle\left|ab \bmod N\right\rangle\left|0^S\right\rangle\\
\to&\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|(-a^{-1}g+a^{-1}b^{-1}) \bmod N\right\rangle\left|b^{-1} \bmod N\right\rangle\left|ab \bmod N\right\rangle\left|0^S\right\rangle\\
\to&\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|(-a^{-1}g+a^{-1}b^{-1}) \bmod N\right\rangle\left|(b^{-1}-(a\cdot(-a^{-1}g+a^{-1}b^{-1}))) \bmod N\right\rangle\left|ab \bmod N\right\rangle\left|0^S\right\rangle\\
&\qquad=\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|(-a^{-1}g+a^{-1}b^{-1}) \bmod N\right\rangle\left|g\right\rangle\left|ab \bmod N\right\rangle\left|0^S\right\rangle\\
\to&\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|((-a^{-1}g+a^{-1}b^{-1})+a^{-1}\cdot g) \bmod N\right\rangle\left|g\right\rangle\left|ab \bmod N\right\rangle\left|0^S\right\rangle\\
&\qquad=\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|(ab)^{-1} \bmod N\right\rangle\left|g\right\rangle\left|ab \bmod N\right\rangle\left|0^S\right\rangle\\
\to&\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle\left|ab \bmod N\right\rangle\left|(ab)^{-1} \bmod N\right\rangle\left|g\right\rangle\left|0^S\right\rangle.
\end{aligned}$$

This runs our multiplication circuit four times and its inverse twice 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. ☐

For the next lemma, we quickly set up some notation. Define the Fibonacci numbers by $F_0 = 0, F_1 = 1$, and then $F_k = F_{k-1} + F_{k-2}$ for $k \geq 2$. Let $K$ be maximal such that $F_K \leq D$. We have:

$$\begin{aligned}
K &= (1+o(1))\cdot\frac{\log D}{\log\phi}\\
&= (1+o(1))\cdot\frac{(C+2+o(1))\sqrt{n}}{\log\phi}\\
&= (\alpha+o(1))\sqrt{n},
\end{aligned}$$

where we let $\alpha = \frac{C+2}{\log\phi}$.

Also, let $\psi(a)$ denote the state $\left|a\right\rangle\left|a^{-1} \bmod N\right\rangle$ for an $n$-bit integer $a$ coprime to and reduced modulo $N$. Each $\psi(a)$ hence requires $2n$ qubits to store.

**Lemma 3.3.** *There exists a quantum circuit using $O(K(G+n)) = O(n^{1/2}(G+n))$ gates such that, for all $n$-bit integers $c_1,\ldots,c_K$ coprime to and reduced modulo $N$, it will map*

$$\psi(c_1)\psi(c_2)\ldots\psi(c_K)\left|0^{S+5n}\right\rangle \mapsto \psi(c_1)\psi(c_2)\ldots\psi(c_K)\psi(\prod_{j=2}^{K} c_j^{F_{j-1}})\psi(\prod_{j=1}^{K} c_j^{F_j})\left|0^{S+n}\right\rangle$$

*(All registers throughout this lemma and its proof work with integers mod $N$, so we drop the $\bmod N$ everywhere for convenience.)*

*Proof.* We will use $4n$ of our ancillas to store two states $\psi(x_1)$ and $\psi(x_2)$. Thus the state at any given point in our algorithm will be $\psi(c_1)\psi(c_2)\ldots\psi(c_K)\psi(x_1)\psi(x_2)\left|0^{S+n}\right\rangle$. We will update the values of $x_1$ and $x_2$ throughout the algorithm.

Lemma 3.2 tells us that we can perform the operation $\psi(x)\psi(y)\left|0^{S+n}\right\rangle \mapsto \psi(x)\psi(xy)\left|0^{S+n}\right\rangle$ using $O(G+n)$ gates. We denote this as updating $y \leftarrow xy$ and will iteratively use this, as described in Algorithm 3.1.

**Algorithm 3.1:** Fibonacci multi-exponentiation

```
Data: Initial state $\psi(c_1)\psi(c_2)\dots\psi(c_K)\,|0^{S+5n}\rangle$
Result: Final state $\psi(c_1)\psi(c_2)\dots\psi(c_K)\psi(\prod_{j=2}^K c_j^{F_{j-1}})\psi(\prod_{j=1}^K c_j^{F_j})\,|0^{S+n}\rangle$
  1. Initialize $x_1 \leftarrow 1$ and $x_2 \leftarrow 1$. This is just a matter of copying some qubits into ancillas, and
     the state is now
     $\psi(c_1)\psi(c_2)\dots\psi(c_K)\psi(x_1)\psi(x_2)\,|0^{S+n}\rangle = \psi(c_1)\psi(c_2)\dots\psi(c_K)\psi(1)\psi(1)\,|0^{S+n}\rangle$.
  2. Repeat the following for $j = K, K-1, \dots, 1$ in that order:
       • Update $x_1 \leftarrow x_1x_2$ using Lemma 3.2.
       • Update $x_1 \leftarrow x_1c_j$ using Lemma 3.2.
       • Swap $x_1$ and $x_2$ (i.e. swap $\psi(x_1)$ and $\psi(x_2)$).
```

Before we show the correctness of our algorithm, let us quickly analyze its complexity. The initialization takes $O(n)$ gates. Within the loop, there are $O(G+n)$ gates from calls to Lemma 3.2 and then $O(n)$ gates to swap $x_1$ and $x_2$. The loop has $K$ iterations, so the total number of gates is $O(K(G+n)) = O(n^{1/2}(G+n))$.

Now we show that the algorithm runs correctly. The claim is that for all $j \le K-1$, at the end of round $j$, we will have $x_1 = \prod_{i=j+1}^K c_i^{F_{i-j}}$ and $x_2 = \prod_{i=j}^K c_i^{F_{i+1-j}}$. We show this by induction.

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: $(1, c_K) \to (c_K, c_K) \to (c_{K-1}c_K, c_K) \to (c_K, c_{K-1}c_K)$. 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:

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

After the first update, $x_1$ becomes:

$$\begin{aligned}\prod_{i=j+2}^{K} c_i^{F_{i-j-1}} \cdot \prod_{i=j+1}^{K} c_i^{F_{i-j}} &= c_{j+1}^{F_1} \cdot \prod_{i=j+2}^{K} c_i^{F_{i-j-1}+F_{i-j}} \\ &= c_{j+1} \cdot \prod_{i=j+2}^{K} c_i^{F_{i-j+1}} \\ &= \prod_{i=j+1}^{K} c_i^{F_{i-j+1}}.\end{aligned}$$

After the second update, it becomes:

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

Swapping the registers now gives us the state $(x_1, x_2) = (\prod_{i=j+1}^K c_i^{F_{i-j}}, \prod_{i=j}^K c_i^{F_{i-j+1}})$, completing our induction.

After the $j = 1$ stage, our state is hence $(\prod_{i=2}^{K} c_i^{F_{i-1}}, \prod_{i=1}^{K} c_i^{F_i})$, as desired. $\square$

Note, importantly, that we would not want to use Lemma 3.3 directly in our algorithm; storing $c_1, \ldots, c_K$ at the same time is an undesirable space overhead. Instead, we will compute $c_1, \ldots, c_K$ on an as-needed basis and apply the same idea; this way, we only need to store one of them at any given time. We explain how to do this and how we define the $c_j$'s in the next section.

## 3.2 Combining Fibonacci exponentiation with [Reg23]'s optimization

To use the results from Section 3.1, we want to decompose each of our exponents as a sum of distinct Fibonacci numbers. Concretely, recall that we want to compute the following expression:

$$\prod_{i=1}^{d} a_i^{z_i+D/2}.$$

So for each $i \in [d]$, we would like to write:

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

for $z_{i,j} \in \{0, 1\}$. It is well-known that this can be achieved with a simple greedy algorithm; see Appendix B for details. Our first task is to compute these coefficients:

**Lemma 3.4.** *There exists a quantum circuit using $O(n^{3/2})$ gates mapping the state*

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

*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 $O(\sqrt{n})$ ancillas; 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 $O(\sqrt{n})$ ancillas.

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 $K$ ancillas to hold $z_{i,j}$ for $j \in [K]$, so that $z_{i,j} = 0$ for all $j$ initially.
3. Now for each $j = K, K-1, \ldots, 1$, check whether $t \geq F_j$ and write the output of this comparison to the qubit $z_{i,j}$. Then use $z_{i,j}$ as a control qubit to conditionally update $t \leftarrow t - F_j$.
4. By Lemma B.1, this greedy algorithm will correctly decompose $z_i+D/2$ as a sum $\sum_{j=1}^{K} z_{i,j} F_j$ of distinct Fibonacci numbers, and we will have $t = 0$ at the end. Hence we have freed up those $\log D$ bits as ancillas 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 $O(\log D) = O(\sqrt{n})$ gates. So, each step of the outer loop over $i$ uses $O(K \log D) = O(n)$ gates. 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 $O(\sqrt{n})$ ancillas (which we can then reuse). Other than that, each step of the loop consumes $K$ ancillas but then frees up $\log D$ ancillas. The total initial ancilla requirement is hence $d(K - \log D) + O(\sqrt{n})$ as desired. $\square$

Now that we can calculate the $z_{i,j}$'s, let us write our desired expression in those terms:

$$\begin{aligned}\prod_{i=1}^{d} a_i^{z_i+D/2} &= \prod_{i=1}^{d}\prod_{j=1}^{K} a_i^{z_{i,j}F_j} \\ &= \prod_{j=1}^{K}(\prod_{i=1}^{d} a_i^{z_{i,j}})^{F_j}.\end{aligned}$$

We can calculate each $\prod_{i=1}^{d} a_i^{z_{i,j}}$ efficiently following the idea by [Reg23] to exploit the fact that the $a_i$'s are very small integers:

**Lemma 3.5.** *[Reg23] There exists a quantum circuit using $O(\sqrt{n}\log^3 n)$ gates mapping*

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

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

*Proof.* [Reg23] shows that such a computation can be done classically with $O(d\log^3 d) = O(\sqrt{n}\log^3 n)$ gates. We can implement this using unitary gates and just use $\sqrt{n}\log^3 n = \widetilde{O}(\sqrt{n})$ ancillas for all intermediate values in the circuit. Then we copy the final output $|\prod_{i=1}^{d} a_i^{t_i}\rangle$ to a fresh register; each $a_i$ is $O(\log d)$ bits so we need $O(d\log d) = \widetilde{O}(\sqrt{n})$ ancillas and $O(d\log d) = O(\sqrt{n}\log n)$ gates to do this. Finally, we can just uncompute the original circuit to restore all ancillas to $|0\rangle$. In total, we have used $\widetilde{O}(\sqrt{n})$ ancillas and $O(\sqrt{n}\log^3 n)$ gates, as desired. □

Combining these ideas allows us to outline the algorithm. Define $c_j = \prod_{i=1}^{d} a_i^{z_{i,j}}$ for each $j \in [K]$. Then we have:

$$\begin{aligned}\prod_{i=1}^{d} a_i^{z_i+D/2} &= \prod_{j=1}^{K}(\prod_{i=1}^{d} a_i^{z_{i,j}})^{F_j} \\ &= \prod_{j=1}^{K} c_j^{F_j}.\end{aligned}$$

We can use Lemma 3.5 to compute each $c_j$, and Lemma 3.3 to finally compute this product. The only missing detail is that of computing $c_j^{-1}$, which we do as follows:

$$\begin{aligned}c_j^{-1} &= \prod_{i=1}^{d} a_i^{-z_{i,j}} \\ &= (\prod_{i=1}^{d} a_i^{-1}) \cdot \prod_{i=1}^{d} a_i^{1-z_{i,j}}.\end{aligned}$$

We can compute the latter expression using Lemma 3.5 since the exponents are now in $\{0,1\}$. Then we can apply Lemma 3.1 using the classical constant $\prod_{i=1}^{d} a_i^{-1}$ to finish computing $c_j^{-1}$. We formally detail all of this in the next section.

**Algorithm 3.2:** Quantum oracle for $\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N$

**Data:** Initial state $|z\rangle$ and $S+(\alpha-C+6+o(1))n$ ancilla qubits in the $|0\rangle$ state.

**Result:** Final state comprising $|z\rangle\,|\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N\rangle$ and $S+(\alpha-C+5+o(1))n$ qubits in the $|0\rangle$ state.

We will reserve $n$ ancillas for the final step of the algorithm, and for the most part use the other $S+(\alpha-C+5+o(1))n$ ancillas. Call these the *primary ancillas*.

1. Use Lemma 3.4 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 $dK - d\log D = (\alpha - C - 2 + o(1))n$ qubits, leaving $S+(7+o(1))n$ primary ancillas.)
2. Set aside $4n$ ancillas. These will store our states $\psi(x_1)$ and $\psi(x_2)$. Initialize $x_1 \leftarrow 1$ and $x_2 \leftarrow 1$. (This leaves $S+(3+o(1))n$ primary ancillas.)
3. Repeat the following for $j = K, K-1, \ldots, 1$ in that order:
   (a) Consider the qubits $z_{i,j'}$ for $i \in [d]$ and $j' \neq j$. There are $d(K-1) \geq (\alpha - o(1))n \geq (\frac{2}{\log\phi} - o(1))n > 2n$ such qubits, 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 ancillas. We now have $S+3n$ clean primary ancillas available.
   (b) Update $x_1 \leftarrow x_1 x_2$ using Lemma 3.2. (This temporarily uses and restores $S$ clean ancillas and $n$ dirty ancillas, which we have.)
   (c) Now we prepare the state $\psi(c_j)$:
      i. Calculate the state $|c_j\rangle$ using Lemma 3.5 and store it in an $n$-bit register. (We now have $S+2n$ primary ancillas.)
      ii. Calculate the state $|\prod_{i=1}^{d} a_i^{1-z_{i,j}} \bmod N\rangle$ using Lemma 3.5 and store it in another $n$-bit register. (We now have $S+n$ primary ancillas.)
      iii. Use Lemma 3.1 with the classical constant $\prod_{i=1}^{d} a_i^{-1} \bmod N$ to update the register from the above step to contain $|c_j^{-1} \bmod N\rangle$. (This will use and then restore all $S+n$ of our remaining primary ancillas, as well as modifying our $n$ dirty ancillas.)
   (d) We now have the state $\psi(c_j)$, so we can update $x_1 \leftarrow x_1 c_j$ using Lemma 3.2. (This will use and then restore $S$ clean primary ancillas and $n$ dirty ancillas.)
   (e) Now we uncompute the state $\psi(c_j)$, returning all qubits to $|0\rangle$. This can be done by just applying in reverse order the inverses of the circuits used to construct this state. (This will use $S+n$ of our available primary ancillas and then also free up the $2n$ ancillas containing $\psi(c_j)$, so we are now back to $S+3n$ primary ancillas. Additionally, this will return all dirty ancillas to their original value, since the dirty ancillas are only modified in the construction of $\psi(c_j)$ when we use Lemma 3.1.)
   (f) Swap $x_1$ and $x_2$ (i.e. swap the registers $\psi(x_1)$ and $\psi(x_2)$).
4. Copy the state $|x_2\rangle$ (i.e. the first $n$ qubits of $\psi(x_2)$) to our $n$ reserved ancillas. Now uncompute the entire algorithm up to this point (i.e. apply the inverse circuit); this will restore the state $|z\rangle$ and reset all primary ancillas to $|0\rangle$.

### 3.3 Proof of Lemma 2.2

Our procedure is detailed in Algorithm 3.2. We track the use of ancillas in the algorithm description.

First we address the correctness of our algorithm. We make some important observations to justify our use of dirty ancillas:

- At step $j$ in the loop, our algorithm does use any qubit $z_{i,j'}$ for $j' \neq j$, so the fact that these qubits might change value during the loop will not affect how $x_1$ and $x_2$ are updated.
- We also need to check that our dirty ancillas are usable in Lemmas 3.1 and 3.2 i.e. that $g \in [0, N-1]$, where $|g\rangle$ is the value stored by these ancillas. Initially, this is ensured by the fact that the first bit of $g$ is 0 i.e. $g < 2^{n-1} \leq N$. Additionally, even though Lemma 3.1 modifies $g$, it replaces $g$ with some other value that is reduced mod $N$. So $g$ will always be in $[0, N-1]$.
- Any application of Lemma 3.2 will preserve the value of these dirty ancillas.
- The one application of Lemma 3.1 when constructing $\psi(c_j)$ will modify the dirty ancillas, but this will be reversed at the end of step $j$ in the loop when uncomputing $\psi(c_j)$. Hence the value of these qubits is ultimately restored at the end of the loop.

Now that we have shown that our use of dirty ancillas does not affect anything, correctness follows from the proof of Lemma 3.3; at the end of step 3, we will have $|x_2\rangle = |\prod_{j=1}^{K} c_j^{F_j} \bmod N\rangle = |\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N\rangle$, and this will be copied into the reserved ancillas. Then after step 4, the remaining qubits will be restored to the initial state containing $|z\rangle$ and the primary ancillas set to $|0\rangle$.

Finally, we check the size of our circuit. The steps before the loop use $O(n^{3/2})$ gates. Now, for each step of the loop over $j$:

1. Each application of Lemma 3.2 uses $O(G+n)$ gates.
2. Computing and later uncomputing the state $\psi(c_j)$ uses $O(G+n)$ gates.
3. The final swap uses $O(n)$ gates.

The loop runs for $K$ iterations, so the total circuit size from the loop will be $O(K(G+n)) = O(n^{1/2}(G+n))$ gates. Combining this with the initialization steps yields a circuit of size $O(n^{1/2} \cdot G + n^{3/2})$ up to and including step 3. Step 4 simply doubles this circuit complexity (due to the uncomputation) with an $O(n)$ overhead to copy $|x_2\rangle$. Hence the asymptotic circuit size is still $O(n^{1/2} \cdot G + n^{3/2})$, completing the proof of Lemma 2.2.

### 3.4 Potential Future Space Optimizations

We have shown that Regev's factoring algorithm can be implemented using just

$$S + \left(\frac{C+2}{\log \phi} + 8 + o(1)\right)n$$

qubits, while retaining the asymptotic $O(n^{1/2} \cdot G + n^{3/2})$ circuit size. We remind the reader that here $G$ and $S$ denote the number of gates and the number of ancilla qubits respectively for our multiplication circuit on $n$-bit integers.

A natural question is whether this can be optimized further, particularly the constant $\frac{C+2}{\log \phi}+8$ in the linear term. To this end, we describe the various space costs incurred by our algorithm below:

1. $\alpha n = \left(\frac{C+2}{\log \phi} + o(1)\right)n$ qubits are used to store the qubits $z_{i,j}$ for $i \in [d]$ and $j \in [K]$. This is a slight blow-up from the $(C + 2 + o(1))n$ bits that we really need to store all the $z_i$'s.

   This does appear inefficient, because at any given point in our algorithm we are only interested in one particular value of $j$. However, the natural way to compute the $z_{i,j}$'s involves iterating through each value of $i$ (since we decompose $z_i + D/2$ as a sum of Fibonacci numbers to obtain the $z_{i,j}$'s). A potential workaround would be to find a way to compute $z_{i,j}$ given $i$ and $j$ in a more efficient way than actually calculating the entire representation of $z_i + D/2$ as a sum of Fibonacci numbers, but we are not aware of how to do this.

2. $S + n$ clean ancillas are needed for the multiplication operations given in Lemmas 3.1 and 3.2. This is already quite optimized due to our use of dirty ancillas in these lemmas. However, we believe this can be brought down to $S$ (which is necessary for us to be able to use our multiplication circuit).

   The only reason we need the extra $n$ ancillas is in Lemma 3.1, which we use with $a = (\prod_{i=1}^{d} a_i^{-1}) \bmod N$. These $n$ qubits are used to write in $a$ and $-a^{-1}$ so that we can use our multiplication circuit to multiply by these integers. We stated our algorithm in this way so that all our multiplication operations could be carried out via one convenient abstraction.

   However, following ideas by Shor [Sho97], it should be possible to instead construct circuits for "multiplication by $a$" and "multiplication by $-a^{-1}$", thereby baking these values into the circuit architecture without requiring $a$ and $-a^{-1}$ to explicitly be written down in another register. This would save us $n$ qubits of space cost.

3. $4n$ ancillas are needed to store the "accumulator" states $\psi(x_1)$ and $\psi(x_2)$. We are not aware of any approaches to bypass this requirement.

4. $2n$ ancillas are used to store the state $\psi(c_j) = |c_j\rangle\, |c_j^{-1} \bmod N\rangle$. This is potentially unnecessary; $c_j = \prod_{i=1}^{d} a_i^{z_{i,j}}$ is a product of at most $d$ integers of $O(\log d)$ bits each, so $c_j$ only comprises $O(d \log d) = \widetilde{O}(\sqrt{n})$ bits. While the same cannot be said of $c_j^{-1} \bmod N$, it might be possible to instead store $\prod_{i=1}^{d} a_i^{1-z_{i,j}}$ (which also only comprises $\widetilde{O}(\sqrt{n})$ bits) in its place, and bake the fixed multiplier $(\prod_{i=1}^{d} a_i^{-1}) \bmod N$ into the circuit architecture.

   Once again, we used $n$ qubits for each of these registers so that we could maintain a clean abstraction for multiplication operations, namely multiplying two $n$-bit integers that are reduced mod $N$. It may be possible to improve this if the multiplication algorithm being used can multiply a small integer with an $n$-bit integer without padding to make both integers $n$ bits.

5. Finally, $n$ ancillas are reserved for copying the final output of our quantum oracle, so that we can uncompute the entire algorithm and clean up intermediate registers. We are not aware of a way to bypass this.

**Acknowledgements.** We thank Oded Regev for his detailed comments on this manuscript. SR was supported by an Akamai Presidential Fellowship. VV was supported in part by DARPA under Agreement No. HR00112020023, NSF CNS-2154149, a grant from the MIT-IBM Watson AI and a Thornton Family Faculty Research Innovation Fellowship from MIT. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

## References

[BCDP96] David Beckman, Amalavoyal N Chari, Srikrishna Devabhaktuni, and John Preskill. Efficient networks for quantum factoring. *Physical Review A*, 54(2):1034, 1996. 1, 2

[Bea03] Stéphane Beauregard. Circuit for Shor's algorithm using 2n+3 qubits. *Quantum Inf. Comput.*, 3(2):175–185, 2003. 1, 2, 4

[BMT$^+$07] Andrew Byrne, Nicolas Meloni, Arnaud Tisserand, Emanuel M. Popovici, and William Peter Marnane. Comparison of simple power analysis attack resistant algorithms for an elliptic curve cryptosystem. *Journal of Computers*, 2(10), December 2007. 2, 6

[Cop02] Don Coppersmith. An approximate Fourier transform useful in quantum factoring. *arXiv preprint quant-ph/0201067*, 2002. 1, 5

[CW00] Richard Cleve and John Watrous. Fast parallel circuits for the quantum fourier transform. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 526–536. IEEE Computer Society, 2000. 1

[Dra00] Thomas G Draper. Addition on a quantum computer. *arXiv preprint quant-ph/0008033*, 2000. 10

[EH17] Martin Ekerå and Johan Håstad. Quantum algorithms for computing short discrete logarithms and factoring RSA integers. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 347–363. Springer, 2017. 1

[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. 1

[Gid17] Craig Gidney. Factoring with $n + 2$ clean qubits and $n - 1$ dirty qubits. *arXiv preprint arXiv:1706.07884*, 2017. 1, 2, 4

[Gid19] Craig Gidney. Asymptotically efficient quantum Karatsuba multiplication. *arXiv preprint arXiv:1904.07356*, 2019. 1, 2, 3, 4, 18

[Gid23] Craig Gidney. Comment on Scott Aaronson's blog, 2023. 1, 2

[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, 2, 4, 6

[HvdH21] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2), March 2021. 2, 3, 4, 18

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

[Kle08] Shmuel T. Klein. Should one always use repeated squaring for modular exponentiation? *Information Processing Letters*, 106(6):232–237, June 2008. 2, 6

[KO62] Anatolii Alekseevich Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, volume 145, pages 293–294. Russian Academy of Sciences, 1962. 3

[Mel07] Nicolas Meloni. New point addition formulae for ECC applications. In *Arithmetic of Finite Fields: First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007. Proceedings 1*, pages 189–201. Springer, 2007. 2, 6

[Reg23] Oded Regev. An efficient quantum factoring algorithm. *arXiv preprint arXiv:2308.06572*, 2023. 1, 2, 3, 4, 5, 6, 10, 11, 16

[RNSL17] Martin Roetteler, Michael Naehrig, Krysta M Svore, and Kristin Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23*, pages 241–270. Springer, 2017. 3, 4, 17, 18

[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, 2, 4, 6, 14

[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, 2, 4

[VBE96] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Physical Review A*, 54(1):147, 1996. 1, 2

[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. 16

## A Justifying the choice of $R$ (and hence $D$)

Let $\delta = \frac{\sqrt{d}}{\sqrt{2}R}$. Then the inequality in the analysis by [Reg23] that requires $R$ to be sufficiently large is the following (see section 5 of [Reg23]:

$$\begin{aligned}
&(2d+4)^{1/2} \cdot 2^{d+2} \cdot (d+5)^{1/2} \cdot T < \delta^{-1}(4 \cdot 2^n)^{-1/(d+4)}/6 \\
\Leftrightarrow\ &(2d+4)^{1/2} \cdot 2^{d+2} \cdot (d+5)^{1/2} \cdot 2^{C\sqrt{n}} < \frac{\sqrt{2}R}{\sqrt{d}} \cdot \frac{(4 \cdot 2^n)^{-1/(d+4)}}{6} \\
&\Leftrightarrow \log_2 R > \log_2(6d^{1/2} \cdot (d+2)^{1/2} \cdot 2^{d+2} \cdot (d+5)^{1/2} \cdot 2^{C\sqrt{n}} \cdot 2^{(n+2)/(d+4)}) \\
&\qquad = o(\sqrt{n}) + C\sqrt{n} + (d+2) + \frac{n+2}{d+4}) \\
&\qquad = o(\sqrt{n}) + C\sqrt{n} + \sqrt{n} + \sqrt{n} \\
&\qquad = (C+2+o(1))\sqrt{n},
\end{aligned}$$

thus our choice of $R$ (and hence $D$) is sufficient.

## B Decomposing Positive Integers as a Sum of Fibonacci Numbers

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. We only need a weaker property, which we restate and prove here:

**Lemma B.1.** *Consider the following algorithm, that takes as input a non-negative integer $t_0 < F_{k+1}$. Initialize $t \leftarrow t_0$ and $S$ to be the empty set. Now for $j = k, k-1, \ldots, 1$, check whether $t \geq F_j$. If it is, update $t \leftarrow t - F_j$ and add $F_j$ to $S$.*

*Then at the end of the algorithm, we will have $t = 0$ and the elements of $S$ adding to $t_0$.*

*Proof.* First, observe that the algorithm clearly ensures that $t \geq 0$ at all times, and that $t$ and the elements of $S$ together add to $t_0$. Hence it suffices to show that at the end of the algorithm, we will have $t = 0$.

We do this by strong induction on $k$. The base case $k = 1$ follows trivially since $t < F_2 = 1$ forces $t = 0$, so we are already done. We also consider the base case $k = 2$; in this case we have $t < F_3 = 2$. If $t = 0$ then we are already done. Else we have $t = 1 = F_2$. The algorithm will hence add $F_2$ to $S$ and $t$ will become 0.

Now for the inductive step, consider $k > 2$. We have two cases:

- If $t < F_k$, then in the $j = k$ round nothing will happen, and we simply reduce to the $k-1$ case.
- If $t \geq F_k$, then in the $j = k$ round $t$ will be replaced by $t - F_k < F_{k+1} - F_k = F_{k-1}$. Hence nothing will happen in the $j = k-1$ round, and at this point we have reduced to the $k-2$ case.

Either way, the conclusion follows by induction. $\square$

## C Implementation of Our Multiplication Oracle

In Theorem 1 and our algorithm, we assume a quantum multiplication circuit that takes a specific form, mapping $|a\rangle|b\rangle|t\rangle|0^S\rangle \mapsto |a\rangle|b\rangle|(t+ab) \bmod N\rangle|0^S\rangle$. However, multiplication circuits will typically only offer the guarantee that $|a\rangle|b\rangle|0^n\rangle|0^S\rangle \mapsto |a\rangle|b\rangle|ab \bmod N\rangle|0^S\rangle$. In this section, we show that this does not pose a significant barrier.

We first describe how to take a circuit with the weaker guarantee and construct a circuit with our specific form, at the expense of $n$ additional ancillas. The key ingredients for our constructions in this section are the following space-efficient primitives for modular doubling and addition, due to [RNSL17]:

**Lemma C.1.** *[RNSL17] There exists a circuit using $O(n \log n)$ gates mapping $|x\rangle|y\rangle|0\rangle|0\rangle \mapsto |x\rangle|(x+y) \bmod N\rangle|0\rangle|0\rangle$. Here, $x$ and $y$ are reduced mod $N$, and each $|0\rangle$ is just one individual ancilla qubit.*

**Lemma C.2.** *[RNSL17] Provided $N$ is odd, there exists a circuit using $O(n \log n)$ gates mapping $|x\rangle|0\rangle|0\rangle \mapsto |2x \bmod N\rangle|0\rangle|0\rangle$. Here, $x$ is once again reduced mod $N$, and we only have two ancilla qubits.*

Using the addition lemma, we have our first simple result:

**Lemma C.3.** *Suppose we have a quantum circuit using $G_0$ gates that maps*

$$|a\rangle|b\rangle|0^n\rangle|0^{S_0}\rangle \mapsto |a\rangle|b\rangle|ab \bmod N\rangle|0^{S_0}\rangle,$$

*and $S_0 \geq 2$. Then there is also a quantum circuit using $O(G_0 + n \log n)$ gates that maps*

$$|a\rangle|b\rangle|t\rangle|0^{S_0+n}\rangle \mapsto |a\rangle|b\rangle|(t+ab) \bmod N\rangle|0^{S_0+n}\rangle,$$

*whenever $t$ is reduced mod $N$.*

*Proof.* We proceed as follows:

$$\begin{aligned}|a\rangle|b\rangle|t\rangle|0^n\rangle|0^{S_0}\rangle &\to |a\rangle|b\rangle|t\rangle|ab \bmod N\rangle|0^{S_0}\rangle \text{ (using the given circuit)}\\ &\to |a\rangle|b\rangle|(t+ab) \bmod N\rangle|ab \bmod N\rangle|0^{S_0}\rangle \text{ (using Lemma C.1)}\\ &\to |a\rangle|b\rangle|(t+ab) \bmod N\rangle|0^n\rangle|0^{S_0}\rangle \text{ (using the inverse of the given circuit)}\end{aligned}$$

The number of gates is clearly $O(G_0 + n\log n)$, so this establishes the lemma. □

Note in particular that this justifies our application of the multiplication algorithms by [HvdH21] and [Gid19] to Theorem 1:

- [HvdH21] implies a circuit with $G_0 = O(n\log n)$ and $S_0 = O(n\log n)$, so we can take $G = O(G_0 + n\log n) = O(n\log n)$ and $S = S_0 + n = O(n\log n)$.
- [Gid19] implies a circuit with $G_0 = O(n^{\log_2 3})$ and $S_0 = O(n)$, so we can take $G = O(G_0 + n\log n) = O(n^{\log_2 3})$ and $S = S_0 + n = O(n)$.

Finally, we show how to construct a circuit using schoolbook multiplication where only $O(1)$ ancillas are needed. [RNSL17] already constructs a circuit mapping $|a\rangle|b\rangle|0^n\rangle \mapsto |a\rangle|b\rangle|ab \bmod N\rangle$, but we want to avoid the $n$-qubit overhead of the conversion procedure in Lemma C.3. Fortunately, this can be done with just a slight tweak of their shift-and-add construction:

**Lemma C.4.** *Provided $N$ is odd, there exists a quantum circuit using $O(n^2\log n)$ gates that maps*

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

*whenever $a, b, t$ are all reduced mod $N$.*

*Proof.* Label the bits comprising $a$ as $a_0, a_1, \ldots, a_{n-1}$, so that $a = \sum_{i=0}^{n-1} a_i 2^i$. Then observe that

$$ab = (\sum_{i=0}^{n-1} a_i 2^i)b = \sum_{i=0}^{n-1} a_i \cdot (2^i b).$$

This suggests the following algorithm:

1. Let $x, y, z$ denote the values in the register containing $a, b, t$ respectively. So initially we have $x = a, y = b, z = t$.
2. Repeat the following for $i = 0, 1, \ldots, n-1$:
   (a) Use $a_i$ as a control bit for the circuit in Lemma C.1 to update $z \leftarrow (z + a_i y) \bmod N$. We have the necessary two ancillas for this.
   (b) Use Lemma C.2 to update $y \leftarrow 2y \bmod N$. We have the necessary two ancillas for this.
3. Repeat the following $n$ times: update $y \leftarrow y/2 \bmod N$. This can be done using the inverse of the circuit from Lemma C.2. Once again, this only needs two ancillas.

First, note that we make $O(n)$ calls to the circuits in Lemmas C.1 and C.2, implying the gate complexity of $O(n^2\log n)$.

To see correctness, it is straightforward to show by induction that at the end of step $i$ within the loop in stage 2, we will have:

$$\begin{aligned}&x = a,\\ &y = 2^{i+1}b \bmod N, \text{ and}\\ &z = (t + \sum_{j=0}^{i} a_j \cdot 2^j b) \bmod N.\end{aligned}$$

So at the end of stage 2, we will have $x = a$, $y = 2^n b \bmod N$, and $z = (t + ab) \bmod N$. Then after stage 3, $y$ will become $b$ again. The ancillas are returned to 0 in each step, so the final state after applying our algorithm will be

$$|x\rangle\,|y\rangle\,|z\rangle\,|0\rangle\,|0\rangle = |a\rangle\,|b\rangle\,|(t+ab) \bmod N\rangle\,|0\rangle\,|0\rangle\,,$$

as desired. $\square$