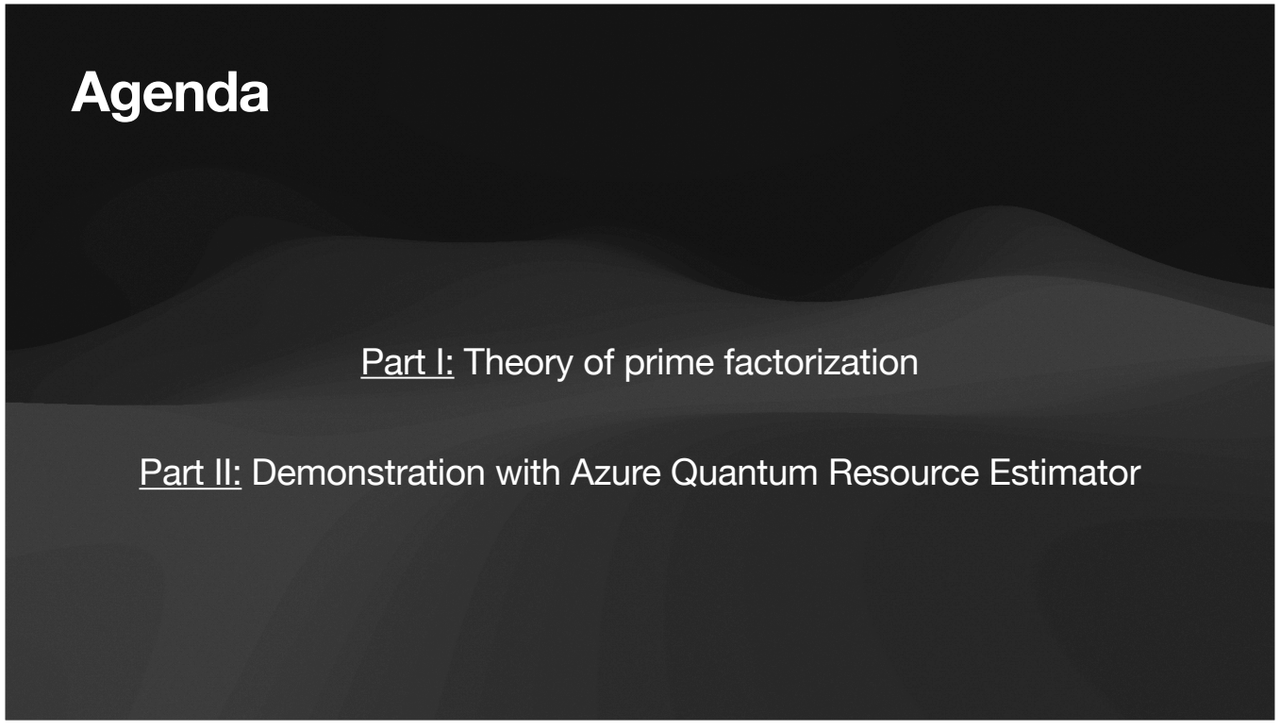
Resource Estimation of Regev’s Algorithm

QRISE 2024 - Microsoft Challenge

Niraj Venkat

Hi there and welcome to this presentation on Resource Estimation of Regev’s Algorithm, a quantum algorithm for prime factorization and discrete logarithm of large numbers.



Our presentation will be divided into 2 parts: theory and implementation. For the first part we talk about the history and importance of prime factorization. We will discuss the new result by Regev that provides an improvement on the quantum circuit for factoring after nearly 3 decades. For the second part we demo our Q# code running on the Resource Estimator within VSCode.

Complexity of three related problems

Consider $h \equiv g^x \mod N$, where $h, g \in \mathbb{Z}_N^*$

Problem	Description	Prime N	Composite N
Exponentiation	Compute h, given g and x	Easy	Easy
Root Finding	Compute g, given h and x	Easy	Hard
Discrete Logarithm	Compute x, given h and g	Hard	Hard

- To set the stage we recall Euclid's fundamental theorem of arithmetic (FTA) which states that every integer N > 1 admits a prime factorization. However Euclid did not give us a recipe for decomposing N. Turns out that this a hard problem for large numbers. Modern day cryptography relies on such hardness assumptions to provide security guarantees. We tabulate here problems regarding the equivalence relation h = g^x mod N, where h and g belong to Z_N^*, which is the multiplicative group over the integers modulo N.

- ~~Exponentiation, which means computing h in this group, can be done using square-multiply technique, which we will cover in more detail in Part 2.~~

- ~~If N is prime, then Z_N^* is a cyclic group which makes root finding easier. Algorithms like NFS (which we discuss shortly) can take advantage of this added structure.~~

- ~~Finally the discrete log problem is hard for both prime and composite cases.~~

- We focus on this part of the table which has applications to cryptography. Root finding with composite N is the hardness assumption underlying the famous RSA cryptosystem. Also widely used today is DH key exchange and elliptic curve cryptography, which both rely on the hardness of the DLP.

History of prime factorization

Trial division	$O(\sqrt{N})$	Classical
Pollard's ρ Method	$O(\sqrt{\#\text{primes}}) \leq O(n^{1/4})$	Classical
Quadratic Sieve	$O(\exp(\sqrt{\log N \cdot \log\log N}))$	Classical
Shor's Algorithm	$O(\log^2 N \log\log N)$	Quantum
Regev's Algorithm	$O(\log^{3/2} N \log\log N)$	Quantum

- Here we present a few algorithms to factor a given number N, and label them as either classical or quantum. We provide this asymptotic analysis as a way to compare classical vs quantum algorithms at a glance. An important caveat is that we ignore any practical overheads of operations like addition and multiplication on large numbers. For classical: the big O notation denotes worst case number of operations, and for quantum: the cost is in terms of circuit size or number of gates.

- ~~First we look at Trial Division (TD) which is the method used to factor primes since antiquity. The simplest version of this taking the range of numbers from 2 to √N and checking for divisors but this is too slow. We can use a sieve to only check against prime divisors and quadratic residues, which we mention later, can be used to speed up TD by skipping some primes that cannot be divisors.~~

- ~~Next we have Pollard's rho Method (PRM). This is a probabilistic algorithm as opposed to a deterministic one like TD. It is also a Monte Carlo algorithm which means we tradeoff accuracy for speed. PRM uses cycle detection because we work with the cyclic group Z mod N. We pick numbers at random from Z mod N and if we get lucky we hit a prime factor. From the birthday paradox, a 50% probability of a collision from n random samples is roughly √n. From the prime number theorem we know #primes < n is roughly log n, and log n is bounded above by √n giving us this heuristic time complexity of O(n^1/4).~~

- ~~Next up we have the Quadratic Sieve (QS). Pomerance proved that the runtime of the QS is actually sub-exponential. It is a deterministic algorithm that produces relations of the form x^2 ≡ q (mod n) which means n | x^2 - q. Here q is called a quadratic residue and needs to be completely factored. This is why need a sieve as a subroutine here. Current state of the art for classical factoring is an extension called the Number Field Sieve, discovered by Pollard from the rho method above. In 1990 it was used to factor F_9, the ninth Fermat number. The NFS has spawned it's own research field and has many variants, including a randomized NFS.~~

- The focus of the rest of this talk will be on the quantum algorithms for prime factorization, namely Shor's and Regev's. Our analysis will be done using the Resource Estimator where we consider practical factors like T gate count and the space-time diagram.

Shor's Algorithm recap

- Quantum algorithm with exponential speedup (break cryptography in poly-time)
- Factoring can be reduced to finding a nontrivial square root: $x^2 \equiv 1 \mod N$
- We choose some random a coprime to N. Shor's algorithm uses a modular multiplication operator $U_{N,a}$ such that $U_{N,a}^m |1\rangle = |a^m \mod N\rangle$
- Because U is periodic: $U^r |1\rangle = |1\rangle$, we would like to know the minimum period $r \neq 0$. QPE allows us to find the eigenvalue for the eigenstate $|1\rangle$
- If r is even, then $a^{r/2}$ is a nontrivial square root. Use GCD to factor N!

- We will now give a recap of Shor's algorithm.

- It is among just a handful of quantum algorithms that can claim an exponential speedup over classical. Shor extended the original paper to include discrete logarithms as well. It is no exaggeration to say that this discovery has kicked off the vibrant field of quantum computing.

- Let N be an integer that is a product of two primes, as is true for RSA numbers. Choose some x between 2 and N-2. ~~If the quadratic residue of x is 1, in other words~~ if x is a nontrivial square root of N, we can factor the left hand side as (x+1) and (x-1), and it can be shown that GCD of x±1 with N gives one of the prime factors of N.

- Now that we know how to factor classically, let us show how a quantum algorithm does it. We start by choosing a base `a` between 2 and N-2 such that gcd(a, N) = 1. Two points worth mentioning here. First is that we are sampling `a` from a subset of Z_N^*, this will be useful in the next step. Second, we could get incredibly lucky at this step and find that `a` is actually a factor of N in which case we're done! But suppose we aren't so lucky. Given a and N that are coprime, Shor's algorithm relies on an operator U that when applied to |1> does a modular multiplication in the computational basis resulting in |a mod N>.

- Because of Euler's theorem from group theory, U is periodic, so let's call the period r. It can be shown that there are r eigenstates of U. Very conveniently, if we sum up all these eigenstates, the different phases cancel out all computational basis states except |1>. Recall that QPE uses controlled unitaries to kick back a phase on the control register, which we then measure in the Fourier basis. In our case, the phases contain information about the eigenvalues of U, and by performing a series of measurements, we can recover the period r.

- If the period r is even, then the square root of a^r is a nontrivial square root. We simply perform the classical procedure mentioned previously to factor N.

Regev's Algorithm overview

- Multidimensional analog of Shor's algorithm that samples from a d-dimensional lattice
- Calculate the product $\prod_{i=1}^{d} a_i^{z_i} \mod N$ where b_i is a small prime and $a_i = b_i^2$
- Apply QFT and measure. Collect noisy samples $w = (w_1, \ldots, w_{d+4}) \in \mathbb{R}^d/\mathbb{Z}^d$ of the dual lattice $\mathscr{L}^*/\mathbb{Z}^d$. Run classical post-processing (lattice reduction, Gram-Schmidt) on w
- $w \mapsto z = (z_1, \ldots, z_d) \in \mathscr{L} \backslash \mathscr{L}_0$. Reached the goal: GCD$((\prod_{i=1}^{d} b_i^{z_i} \mod N) \pm 1, N)$
- Technical detail: The control register is prepared in a Gaussian superposition

- We are now ready to discuss the centerpiece of this project which is Regev's algorithm. We hope to provide a brief glimpse on the application of number theory and lattices. For the purposes of resource estimation however, we only need to know what goes in the quantum circuit and our project does not currently implement the classical post-processing that happens afterwards.

- If we view Shor's algorithm as working with a 1-D line of integers, then Regev's algorithm works with a d-dim lattice. Another difference is that Shor's needs a fault tolerant quantum computer whereas Ragavan and Vaikuntanathan showed that Regev's is resilient to noise by adding more samples. Ekera and Gartner showed that Regev's can solve the discrete log problem as well.

- We perform modular multiplication as before but the product is much larger. Earlier we chose the base a randomly but this time we take the base a_i as the square of the i_th prime number b_i. ~~Modular multiplication on a quantum computer is an entire research topic that is being explored for Shor's algorithm. We have implemented it in 2 ways, more of this in Part 2.~~

- Applying QFT gives the dual of the lattice that we want. We measure to sample the dual but these samples are noisy so we post-process the result. First we perform a lattice reduction algorithm like LLL to give us the closest vector on the dual. Then we perform Gram-Schmidt orthogonalization to construct a basis of the solution lattice L

- So what we've done is to map the noisy samples w to z which lies in the solution space L minus L_0, where L is a subset of the d-dimensional integers and L_0 is all the trivial square roots that we dont care about. We have now reached our goal, just calculate the GCD given here to compute one of the factors.

- The quantum procedure starts by preparing a discrete Gaussian state on the control qubits. The reason for this technical, our guess is it has something to do with

worst case to average case reductions in order to simplify the proofs in Regev's paper. For our purposes we implemented Gaussian state preparation in three ways and compared them with resource estimation. Lets go into

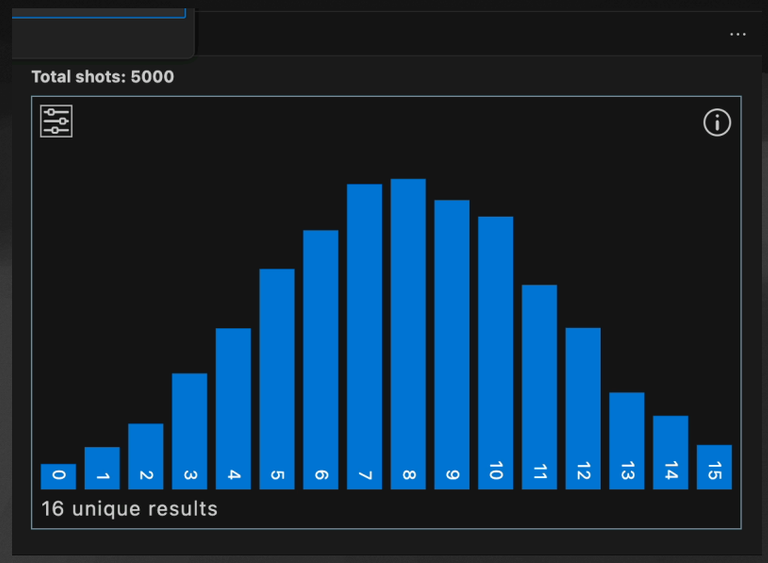
What is a Gaussian state?

- Gaussian distribution: $\mathcal{N}_{\mu,\sigma}(x) \propto e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$ where $x \in \mathbb{R}$
- Gaussian wavefunction: $\psi_{\mu,\sigma}(x) \propto e^{-\frac{1}{4}(\frac{x-\mu}{\sigma})^2}$ where $x \in \mathbb{Z}$
- Quantum state: $|\tilde{\mathcal{N}}_{\mu,\sigma}(x)\rangle = \sum_{x=0}^{2^n-1} \psi_{\mu,\sigma}(x)\,|x\rangle$ for n qubits
- Gaussians are everywhere!

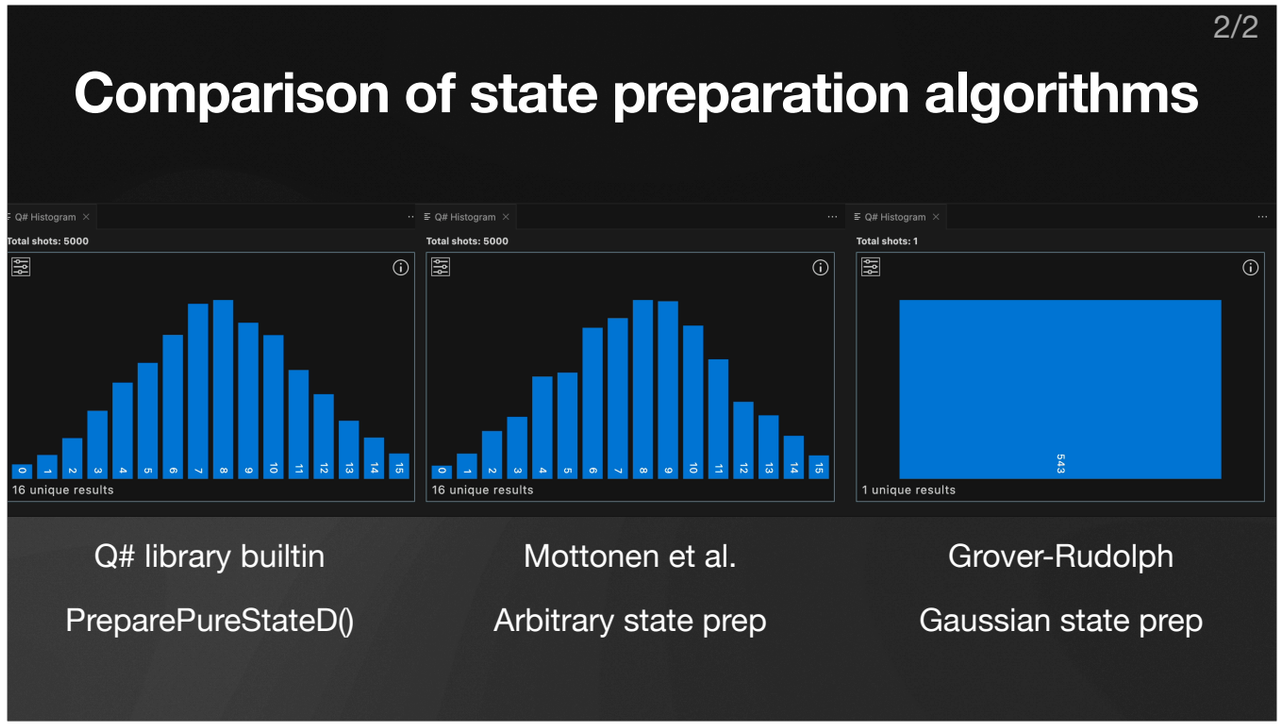
Starting with the normal or Gaussian distribution with mean mu and variance sigma^2, we can construct a wavefunction by requiring that its norm square equals the PDF
A common approach to representing this wavefunction as the state of some qubits is to discretize the domain of the distribution
For example, if we have n qubits, then we integrate over 2^n regions to get their probabilities and set the amplitude of that corresponding basis state as the square root of the probability

Gaussians show up in many areas of science, particularly in physics. For example: The ground state of a simple harmonic oscillator is a Gaussian ~~for a particle of mass m centered at μ corresponds to the choice of σ^2 = hbar/2mω~~. It is a building block for bosonic degrees of freedom and ground states of scalar field theories

Comparison of number of shots



Using the histogram in VSCode, we can see how our quantum state converges to a Gaussian distribution as the number of shots increases. With a hundred shots it won't look like much of a Gaussian, but with more shots a clear pattern emerges.

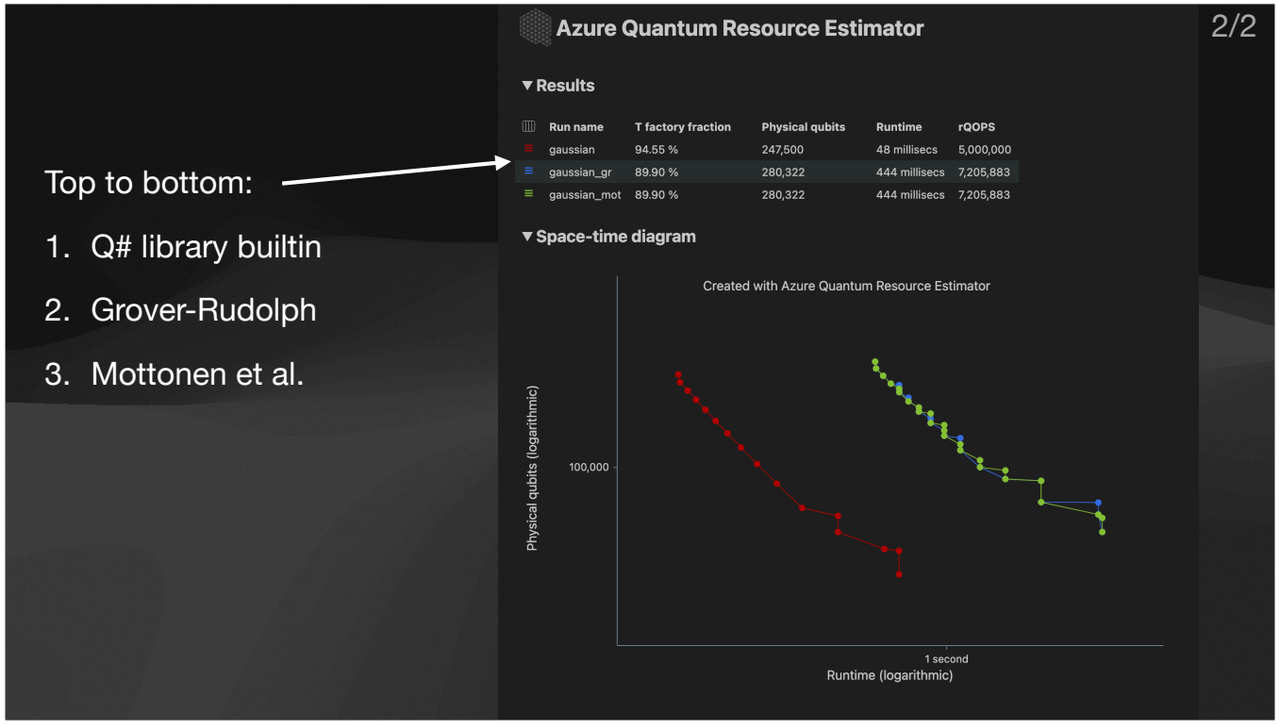


Q# includes a state preparation algorithm and we implemented two additional ones:

The API for Mottonen is similar to the one builtin to Q#, where you just input a vector of 2^n amplitudes and it prepares the state. However, the number of gates is exponential in the number of qubits and we see that cost quite clearly in the resource estimator.

Grover-Rudolph is specifically for distributions and does not accept amplitudes as input. However runs quite slow in the simulator because we generate an exponential number of angles by manually integrating the Gaussian. There is more room for improvement by taking advantage of the symmetry of the bell curve. ~~The resource estimation is quite similar to Mottonen as well.~~

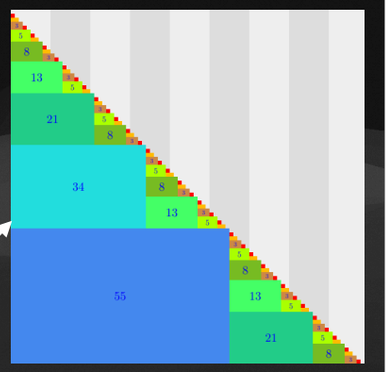
There are many other state preparation methods, ~~like the black box method from Sanders et al., state preparation using QET from McArdle et al. and Kitaev-Webb which is meant specifically for multivariate Gaussians.~~ Worth trying out and this is an exciting area of future work.



This is the spacetime diagram for Gaussian state prep. The Q# library is a lot faster in terms of runtime, and uses tens of thousand less physical qubits. Mottonen and GR have high depth and use multi controlled rotations which is not friendly for nearest neighbor architectures. However they perform a little better in terms of T gates.

Two approaches to exponentiation: a^n

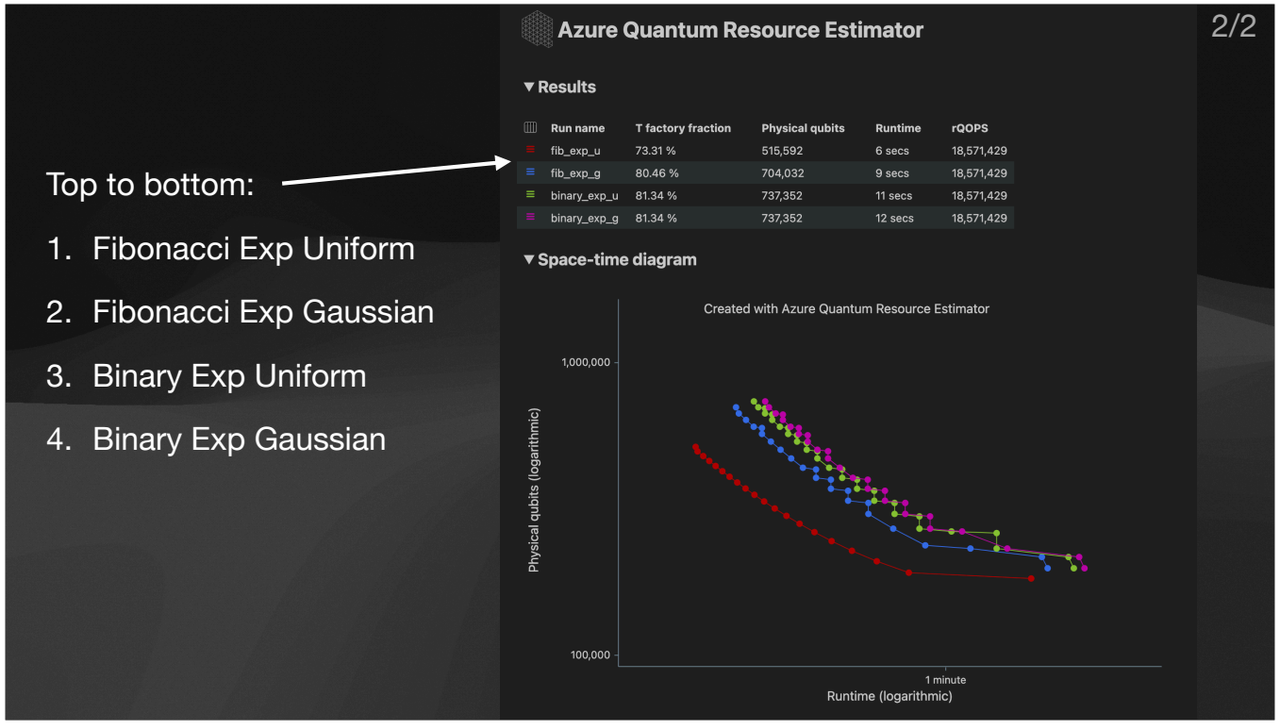
- Binary/square-multiply
  - Write out the exponent n in binary
  - Example: $a^{13} = a^{1101_2} = a^8 \cdot a^4 \cdot a^1$
- Fibonacci
  - Every $n > 0$ has a Zeckendorf representation
  - Example: $4 = \langle 1,0,1 \rangle = \text{Fib}(1) \cdot 1 + \text{Fib}(2) \cdot 0 + \text{Fib}(3) \cdot 1 = 1 + 3$
  - Easier application to quantum computing



- We also explored some ways to implement quantum modular exponentiation. a^n can computed much faster than n-1 multiplications. ~~For modular exponentiation we simply mod out any time that we multiply.~~

- If we write our n in binary we only need to perform at most log n multiplications. ~~Binary exponentiation is the idea that we start at the least significant bit, squaring the base along the way but only multiplying it with the result if the bit is one~~

- Another technique that uses Fibonacci numbers to exponentiate. ~~The Zeckendorf representation of an integer may be determined by a “greedy” high-to-low algorithm that repeatedly selects the largest possible Fibonacci number less than or equal to the remaining balance in the integer.~~ This approach has been used in the reversible computing literature and has direct applications to quantum computing



We profile both binary and Fibonacci exponentiation each with a uniform or Gaussian distribution.
We notice that choice of base significantly impacts runtime and this validates the Regev's strategy of choosing small primes as the base.
We use the Q# builtin method in all the profiles as this is the fastest method we have for Gaussian state prep.
The best performing method is the Fibonacci Exp with Uniform distribution. It would be exciting to know if using a Gaussian actually helps in practice as much as it helps in theory.

References

1. History of integer factorization - Wagstaff
2. Number Theory Review - Vaikuntanathan
3. Shor's Algorithm Tutorial - Qiskit
4. Shor's Algorithm Tutorial - Pennylane
5. Regev's Algorithm (video)
6. Regev's Algorithm (main paper)
7. Regev's Algorithm (optimized)
8. Ekera DLP extension
9. Reason for the Gaussian state
10. PreparePureStateD()
11. Mottonen state preparation
12. Grover-Rudolph state preparation
13. Binary exponentiation
14. Zeckendorf repr image
15. Fibonacci exponentiation

We hope you enjoyed the presentation and thanks for watching!