Quantum Fourier Transform & Shor's Algorithm

R. Flütsch

FS 2020

1 Introduction

Factorizing Numbers is a famously hard problem, the best known classical algorithm has a running time of $\mathcal{O}\left(\exp\left(\sqrt[3]{\frac{64}{9}n\log^2(n)}\right)\right)$ (where n is the number of bits in the number to be factorized). The famous quantum algorithm known as Shor's algorithm in contrast only has runtime complexety of $\mathcal{O}\left(n^3\log(n)\right)$. It therefore represents a significant speedup over classical methods.
For the final project I therefore decided to try to understand this algorithm and try to implement it in Qiskit.

2 Problem

Given an number $N \in \mathbb{N}$ find two numbers $N_1, N_2 \in \mathbb{N}$ such that $N = N_1 * N_2$. This problem is hardest when N_1, N_2 are both prime numbers $\mathbb{P}$. As noted above this problem scales exponentially on a classical computer in practice it is thus impossible to factor numbers with much more than a few hundert bits. This practical impossibility is the basis for the widely used RSA encryption scheme.

2.1 Factorization based on period finding

Besides the more intuitive approach of "just trying all prime numbers" (aka. the number sieve) there's also a possibility to factor numbers using modular exponentiation. While not faster in the classical case it is this approach that allows use of quantum effects to efficiently solve the problem. In the following we assume N to be odd (since finding a factor of an even number is trivial).
Pick any integer $x < N$ such that x and N share no factors. There exists an exponent k such that $x^k mod N = 1$. From this follows that $x^k - 1$ is divisible by N. If we assume that our k is even we can write $x^k - 1 = \left(x^{\frac{k}{2}} - 1\right)\left(x^{\frac{k}{2}} + 1\right) = n_1 * n_2$.
Since $n_1 - n_2 = 2$ they share no factors > 2. Since $N = N_1 * N_2$ is odd and n_1, n_2 share maximally a factor of 2 N_1 must be a factor of either n_1 or n_2. Without loss of generality we assume N_1 to be a factor of n_1. Given that N_1 divides both N and n_1 we find N_1 as the greatest common divisor between N and n_1. Finding the greatest common divisor can be done in polynomial time - given we find a suitable k we can therefore factor N in polynomial time.
Unfortunately finding k is classically not easier than searching for the factors on the straightforward approach. As it involves finding the period p of the modular exponentiation sequence $a_0, a_1, a_2, a_3, \ldots$ with $a_i = x^i mod N$. As the first element of such a sequence $x^0 mod N$ will all ways equal 1 the smallest exponent k that we need in the approach outlined above is equal the the period p of this sequence. Finding this period is the bottle neck of this approach and results in it not being faster than the intuitive approach.

2.1.1 Example

Example for $N = 21$ and $x = 10$

$$a_i = 10^i mod 21$$

$$a_0 = 1,\ a_1 = 10,\ a_2 = 16,\ a_3 = 13,\ a_4 = 4,\ a_5 = 19,\ a_6 = 1,\ a_7 = 10,\ a_8 = 16,\ a_9 = 13 \ldots.$$

This sequence has period $p = 6$, we therefore choose $k = p = 6$ and find:

$$x^k - 1 = \left(x^{\frac{k}{2}} - 1\right) * \left(x^{\frac{k}{2}} + 1\right) \text{ is divisible by } N$$

$$\left(10^3 - 1\right) * \left(10^3 + 1\right) = 999 * 1001 \text{ is divisible by } 21.$$

We therefore find the factors of $N = N_1 * N_2 = 21$ as the greatest common divisor between 21 and 999, and 21 and 1001 respectively.

$$N_1 = gdc\,(21,\, 999) = 3$$
$$N_2 = gcd\,(21,\, 1001) = 7.$$

Which are indeed the factors of 21.

3 Quantum Fouriertransform

Finding the period of a sequence can of course be accomplished by a Fourier transform, however as stated above classically calculating a Fourier transform is also a non-polynomial operation and thus yields no speedup. However there exists a quantum version of the Fourier transform which we can use to circumvent this bottle neck.
The quantum Fourier transform is a quaNtum version of the discrete Fourier transform which takes a vector $\vec{X}$ of size N and transforms it to a vector $\vec{Y} = W\vec{X}$ with the Fourier matrix W.

$$W = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(N-1)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \dots & \omega^{(N-1)(N-1)} \end{bmatrix}.$$

With ω such that $\omega^N = 1$. The quantum fourier transform (QFT) is then defined as a transformation between two quantum states given by the two vectors $\vec{X}$ and $\vec{Y}$.

$$QFT\left(\sum_{k=0}^{N-1} X_k \left|k\right\rangle\right) = \sum_{n=0}^{N-1} Y_n \left|n\right\rangle.$$

Simmilar to the well known classical fast Fourier transform a QFT can be implemented recursively. See figure 1.

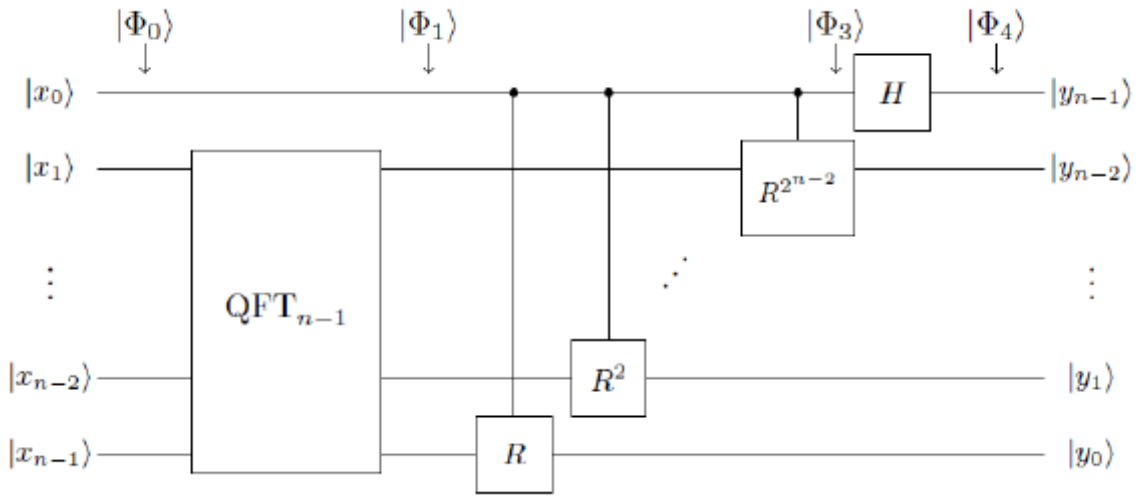


Figure 1: Recursive implementation of the QFT
blatantly stolen from https://qudev.phys.ethz.ch/static/content/QSIT15/Shors%20Algorithm.pdf

4 Shor's Algorithm

By using the approach outlined above to use the period of a modular exponentiation sequence to find the factors of a number N combined with the QFT we can efficiently use a quantum computer to factor a number. By first

creating an equal superposition over qubits in a first register and then applying the function calculating the modular exponentiation we get a periodic superposition which we can then apply a QFT to to find it's period.

4.1 Details of the quantum subroutine in the algorithm

Starting with two registers both in the zero state

$$|\psi\rangle = |0,0\rangle$$

We then apply Hadamart-gates on all the qbits in the first register giving us the following state:

$$|\psi\rangle = \frac{1}{\sqrt{B}} \sum_{n=0}^{B} |n,0\rangle \,.$$

Applying the exponentiation cirqiut yields:

$$\frac{1}{\sqrt{B}} \sum_{n=0}^{B} |n, f(n)\rangle \;\text{ ,with } f(n) = x^n \bmod N.$$

The next step is to apply the QFT which can be used to compute the period of an input. The specific case we're interested in is the following one (justification on why this case arises later):
If the vector $\vec{X}$ of length B has a period r, such that $\frac{B}{r} \in \mathbb{N}$, and the form

$$X_i = \begin{cases} \sqrt{\frac{r}{B}} & \text{if } i \bmod r = s \\ 0 & \text{otherwise} \end{cases}$$

for some offset s, and we write $QFT\left(\sum_{n=0}^{B} X_n |n\rangle\right) = \sum_{k=0}^{B} Y_k |k\rangle$ then the Y_k are given by

$$Y_i = \begin{cases} \frac{1}{\sqrt{r}} & \text{if } i \bmod \frac{B}{r} = 0 \\ 0 & \text{otherwise} \end{cases}$$

When we proceed to measure the qbits on the second register we will find a random one of the values from the exponentiation sequence s. Our state is now

$$\frac{1}{\sqrt{\frac{B}{r}}} \sum_{n=0,\; f(n)=s}^{B} |n, s\rangle$$

we now have a state exactly as described above, applying a QFT to this state thus yields:

$$\frac{1}{\sqrt{r}} \sum_{n=0}^{r} \left| n\left(\frac{B}{r}\right), s \right\rangle .$$

For an illustration of the circuit see figure 2. Measuring this superposition on the first register will thus always return a multiple of $\frac{B}{r}$. Running the the calculation multiple times will thus allow us to calculate $\frac{B}{r}$ and, since B is known, we can also calculate the period r. Note: when the assumption that $\frac{B}{r} \in \mathbb{N}$ the quantum circuit stays the same but the classical process of finding the period p is harder.

4.2 Complete Shor's algorithm

1. Determine whether N is even, if it is one of the factors is 2, exit.

2. Pick a random number $x \in \mathbb{N}$ and calculate $\gcd(x, N)$ if it is not one x is a factor of N, else continue.

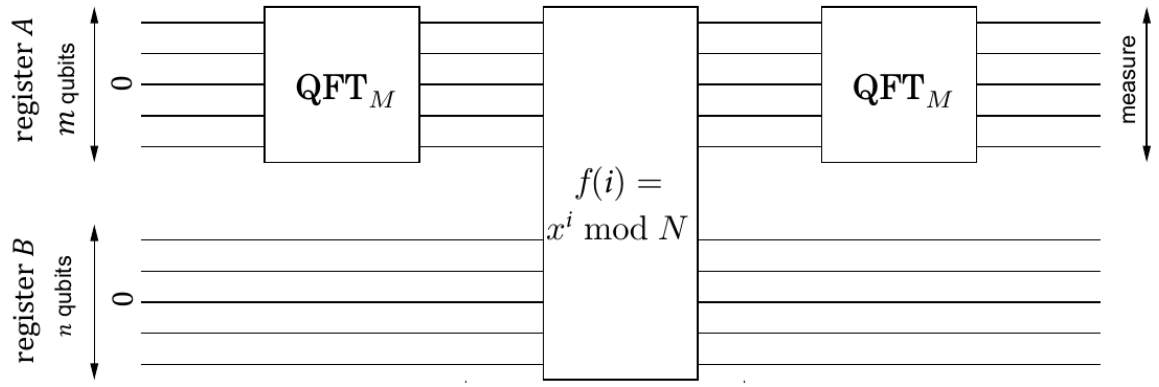


Figure 2: Illustration of the implementation of Shor's algorithms
blatantly stolen from https://arxiv.org/pdf/1804.03719.pdf

3. Use the quantum routine outlined above to find the period p of the modular exponentiation sequence $a_i = x^i \bmod N$

4. If p is odd go back to 2. else continue

5. the factors of N are given by $N_1, N_2 = \gcd\left(x^{\frac{p}{2}} \pm 1, N\right)$

5 Examples

All my code for the following two examples can be found in the attached Jupyter Notebook.

5.1 Optimized circuit found in sources

I found the following circuit (figure 3) in various sources, it is always described as on optimized circuit for the Shor's algorithm with $N = 15$ and $x = 11$. Unfortunately I was unable to find a non-optimized version so that I could try to understand the optimization. Non-the-less I used Qiskit to implement it and ran it on simulators as well as a real device just as a little exercise before attempting my own implementation.

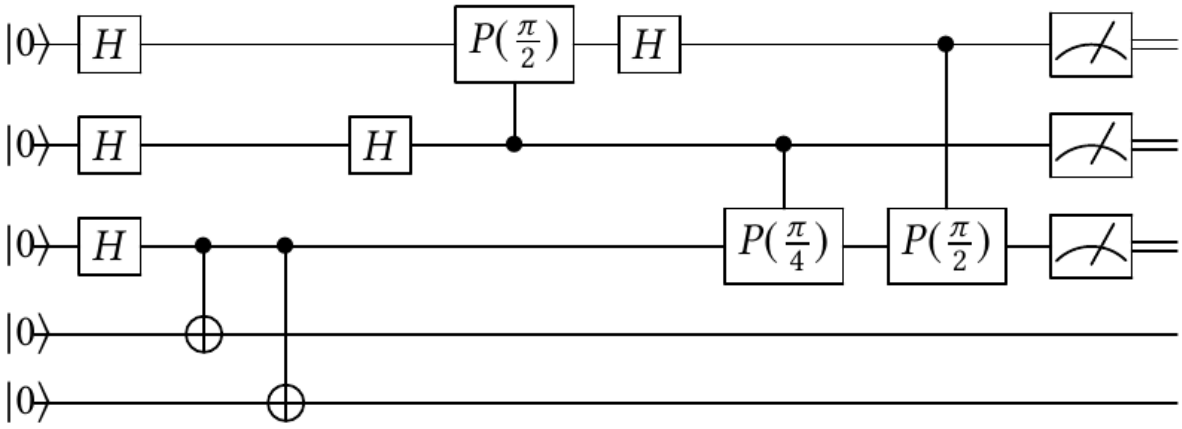
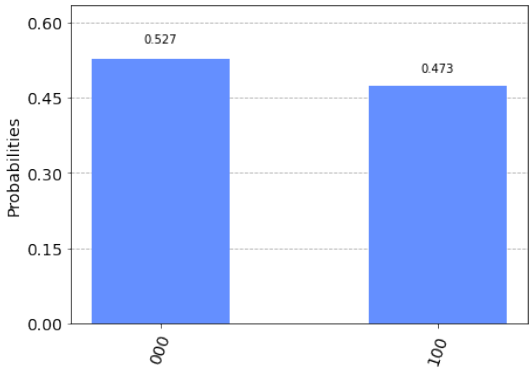
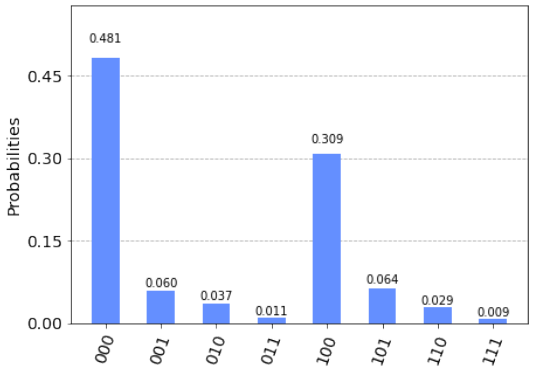


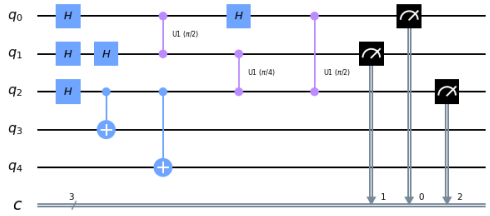
Figure 3: The 'optimized' cirqiut to factorize $N = 15$ using $x = 11$
blatantly stolen from https://arxiv.org/pdf/1804.03719.pdf



(a) Results when above circuit was run on the ibmq_qasm_simulator



(b) Results when above circuit was run on the ibmq_london 5 Qbit quantum computer



(c) My own version of the circuit above

Figure 4: Details on my own implementation of the optimized circuit

5.1.1 Results from optimized circuit

The results (both from the simulation and from the real device, even before error correction) are clear enough to read of two possible results (figure 4) . The first is the trivial case with $p = 0$ and the second the interesting case with $p = b100 = 4$. We worked with $B = 8$ and thus know that $4 = p = \frac{B}{r} \implies r = 2$. Using $\left(x^{\frac{r}{2}} - 1\right) * \left(x^{\frac{r}{2}} + 1\right)$ we thus find

$$N_1 = \gcd\left(11 - 1, 15\right) = 5$$
$$N_2 = \gcd\left(11 + 1, 15\right) = 3.$$

Which are indeed the factors of 15.

5.2 My own implementation

I decided to implement Shor's algorithm for $N = 15$ and $x = 4$. The sequence $a_i = 4^i \bmod 15$ thus is given by $1, 4, 1, 4, 1, 4, \ldots$ with period $p = 2$. I opted for a circuit with 7 qubits, 4 in the first register and 3 in the second. We therefor have $B = 16$ and expect the measurement results to be of the form $n * \frac{B}{p} = n * \frac{16}{2} = n * 8$. An exponent of 2 would lead us to find the factors of $N = N_1 * N_2$ as $\gcd\left(4^1 \pm 1, 15\right) = N_{1,2} = 3, 5$.
The first step was to implement the quantum Fourier transform, followed by the circuit that calculates the exponentiation.

5.2.1 QFT

For the QFT I used the recursive approach in REF. It is important to note that this implementation flips the bit order on its' head (msb becomes lsb and vice versa, etc.), this problem can be solved either by using quantum SWAP gates to swap around the order after the QFT or (at least in our case where the QFT is the last operation in the circuit) can also be done classically by just reading the resulting bitstring backwards.
My function takes as input a quantum circuit on which it appends the appropriate gates, as well as a list of numbers specifying the qubits on which to perform the QFT (they need not neccesarily be in the same order as on the circuit diagram), the first element of this list is assumed to be the lsb, and the last to be the msb. For an example output of my function for 4 qubits see figure 5.

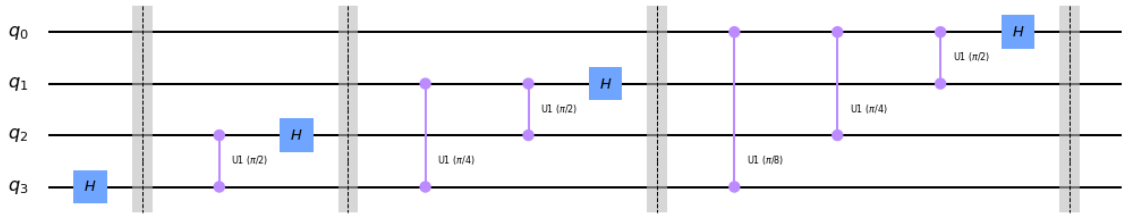


Figure 5: The circuit resulting from my QFT function for 4 qubits, the barriers illustrate the recursive nature of the QFT. Note: this does not include the swaps neccesary to preserve bit order.

5.2.2 Exponentiation Circuit

Since the sequence $a_i = 1, 4, 1, 4, 1, 4, \ldots$ is quite simple the gates required to realize $|i, 0\rangle \rightarrow |i, f(i)\rangle$ are also quite simple. As is obvious from the sequence all the even entries are $a_{2*n} = 1$ and all the odd entries are $a_{2*n+1} = 4$, we therefore only need to look at the first qubit (the least significant bit) to compute $f(i)$. If he first qubit is 1 we need to set the third qubit of the second register (controlled X gate between the first qubit of the first register and the third qubit of the second register) and if the first qubit is 0 we need to set the first qubit of the second register.

5.2.3 Putting it together

The only thing left to do is to put it all together in the right order, the complete circuit is depicted in figure 6. I used a bunch of Hadamart gates on the first register to produce the needed superposition over all states, then applied the gates to perform $|i, 0\rangle \rightarrow |i, f(i)\rangle$ and then stuck my QFT at the end of it. The only thing left to decide was to either use quantum swap gates or to just read the resulting bitstirngs backwards, I opted to include the swapping in the quantum circuit. The complete circuit is depicted below.

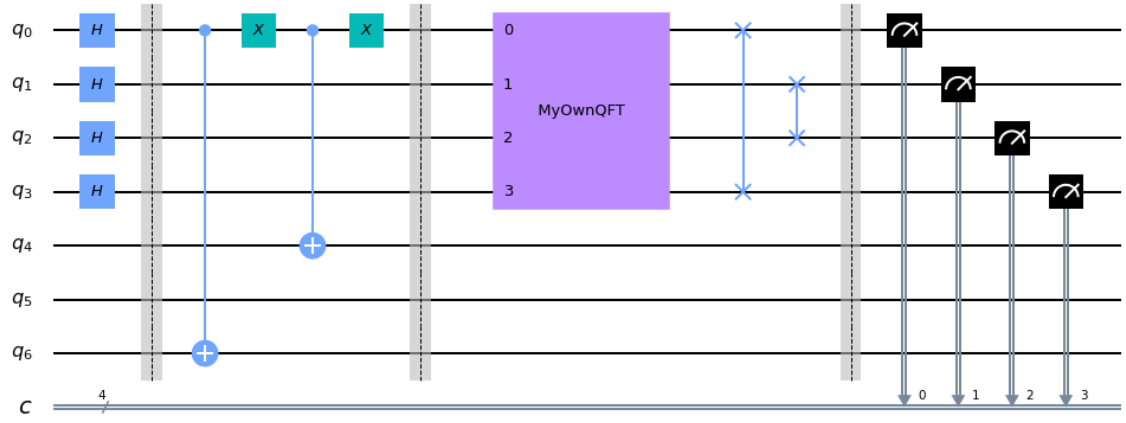


Figure 6: My own implementation of Shor's algorithm for $N = 15$ and $x = 4$. The first section ensures an equal superposition between the qubits of the first register, the second section performs $|i, 0\rangle \rightarrow |i, f(i)\rangle$ and the third section is the QFT followed by the swap operations, finally the last section performs the measurement on the first register.

5.2.4 Results

I ran my circuit on both the IBM quasm simulator as well as the IBM Q Melbourne 15 qubit quantum computer. The results are summarized in figure 7.
While the results from the simulator do indeed yield the expected solutions 0 and 8, the results from running the circuit on a real device are not usable as the expected results are drown out in the noise. Part of this noise could certainly be addressed by error mitigation techniques introduced in the course, another way the error could be mitigated is by streamlining the circuit (some qubits are not even used and could be discarded).

To actually finish the algorithm I will therefore use the simulation results 0 and 8. We can ignore the trivial result of 0 and focus fully on 8. We know the measured result is of the form $n * \frac{B}{r}$ with $n \in \mathbb{N}$, we also know that in this circuit $B = 16$. We thus conclude:

$$8 = n * \frac{B}{r}, \text{ we assume } n = 1 \text{ and find } 8 = \frac{16}{r}$$

$$\implies r = 2 \implies x^r - 1 = 4^2 - 1 \text{ is divided by } N = 15$$

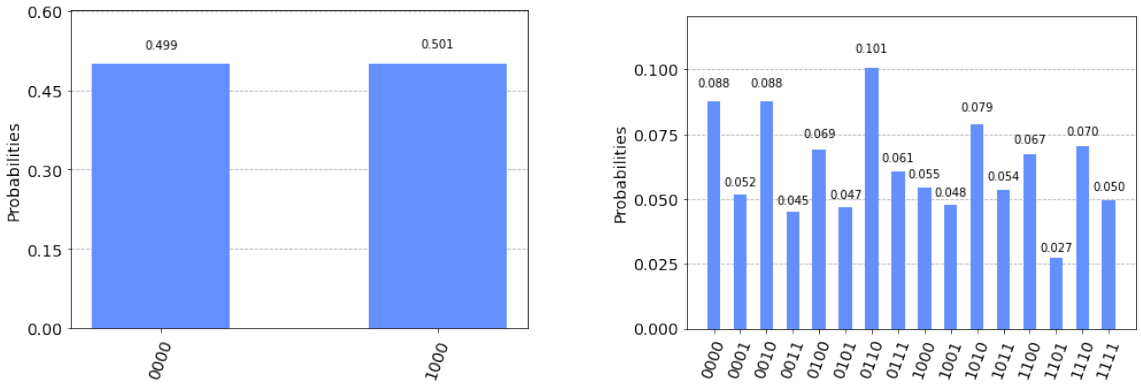
$$\implies \left(4^1 \pm 1\right) \text{ is divided by N}$$

and we find the factors N_1, N_2 of N as

$$N_1 = \gcd(4 - 1, 15) = \gcd(3, 15) = 3$$

$$n_2 = \gcd(4 + 1, 15) = \gcd(5, 15) = 5.$$

Which are, again like expected, the factors of 15.



(a) ibmq_qasm_simulator

(b) ibmq_melburne

Figure 7: Results from both the IBM qasm simulator (a) and the IBM Q Melbourne 15 qubit quantum computer

6 Outlook & Sources

I would very much like to try and get the noise from my own implementation down to a level where the output of a real device can be used to calculate the factors of a number. As already mentioned I would approach this by means of first streamlining the circuit and then if still necessary applying error mitigation methods introduced in this course. Unfortunately I am running out of time and these optimizations will have to wait until after all of the exams :).
To write this little report I used mainly the following sources:

- Quantum Algorithm Implementations for Beginners, ABHIJITH J. et al., Los Alamos National Laboratory, Los Alamos, New Mexico 87545, USA https://arxiv.org/pdf/1804.03719.pdf
- Qiskit Tektbook, Chapter on QFT, https://qiskit.org/textbook/ch-algorithms/quantum-fourier-transform.html
- Shor's Algorithm, Slideshow found on https://qudev.phys.ethz.ch/static/content/QSIT15/Shors%20Algorithm.pdf