Posted on Authorea 23 Nov 2020 — The copyright holder is the author/funder. All rights reserved. No reuse without permission. — https://doi.org/10.22541/au.160614804.47667838/v1 — This a preprint and has not been peer reviewed. Data may be preliminary.

A Comparison of Quantum and Traditional Fourier Transform Computations

Damian Musk[1]

[1]Affiliation not available

November 23, 2020

Abstract

The quantum Fourier transform (QFT) can calculate the Fourier transform of a vector of size N with time complexity $\mathcal{O}(\log^2 N)$ as compared to the classical complexity of $\mathcal{O}(N \log N)$. However, if one wanted to measure the full output state, then the QFT complexity becomes $\mathcal{O}(N \log^2 N)$, thus losing its apparent advantage, indicating that the advantage is fully exploited for algorithms when only a limited number of samples is required from the output vector, as is the case in many quantum algorithms. Moreover, the computational complexity worsens if one considers the complexity of constructing the initial state. In this paper, this issue is better illustrated by providing a concrete implementation of these algorithms and discussing their complexities as well as the complexity of the simulation of the QFT in MATLAB.

The principles of quantum computing

Quantum computation poses a fundamentally different framework than our more familiar classical computation. Specifically, this is due to the existence of qubits, as opposed to bits, the fundamental components of computing. Whereas bits can only hold the binary values of 0 or 1, qubits can instead hold a superposition of states. We use Dirac's bra-ket notation, which utilizes two kets analogous to the classical states 0 and 1, to represent an orthogonal basis,

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{1}$$

A more generic qubit state is a superposition of the previous basis states:

$$|\psi\rangle = x_0|0\rangle + x_1|1\rangle. \tag{2}$$

where x_0 and x_1 are complex-valued amplitudes obeying the relation:

$$|x_0|^2 + |x_1|^2 = 1. \tag{3}$$

It is important to clarify that such a superposition does not take a value "in between 0 and 1": instead, in accordance with the Born rule(Born, 1954), a measurement of the state of the qubit corresponds to a binary value. The qubit has a *probability* $|x_0|^2$ to be found in state "0" and a *probability* $|x_1|^2$ to be found in state "1."

We can use the Kronecker tensor product to construct the basis of a system comprised of multiple (n) qubits. For example, the product basis states of a four-dimensional ($n = 4$) linear vector space are:

Posted on Authorea 23 Nov 2020 — The copyright holder is the author/funder. All rights reserved. No reuse without permission. — https://doi.org/10.22541/au.160614804.47667838/v1 — This a preprint and has not been peer reviewed. Data may be preliminary.

$$|\mathbf{0}\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, |\mathbf{1}\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \tag{4}$$

$$|\mathbf{2}\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, |\mathbf{3}\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{5}$$

Given n qubits, the general state is a superposition state vector in a 2^n-dimensional Hilbert space:

$$|X\rangle = \sum_{j=0}^{j<2^n} x_j |j\rangle \tag{6}$$

with $\sum_{j=0}^{j<2^n} |x_j|^2 = 1$.

Given an arbitrary state result of a computation, to measure one x coefficient j, one would have to perform a projection on the corresponding base vector:

$$x_k = \langle k|x\rangle = \sum_{j=0}^{j<2^n} x_j \delta_{kj}, \tag{7}$$

δ_{kj} is the Kronecker delta. This measurement destroys the state and therefore only one coefficient (or one linear combination of coefficients) can be known. To know more coefficients, one would have to repeat the experiment that produced the original state. In other words, each measurement only provides one x_j of information, and one thus has to perform $\mathcal{O}(N)$ measurements to obtain all N coefficients to a certain precision.

This is a point of major importance that will affect our analysis of the Quantum Fourier Transform.

To compute with qubits, one can form quantum logic gates, which are analogous to classical logic gates but instead operate on quantum states. For example, the following gate H inputs state $|X\rangle$ and outputs $|Y\rangle$:

$$|Y\rangle = H|X\rangle. \tag{8}$$

A more specific example would the mapping of the basis state $|0\rangle$ to $(|0\rangle + |1\rangle)/\sqrt{2}$ and $|1\rangle$ to $(|0\rangle - |1\rangle)/\sqrt{2}$ which would be represented by the one-qubit Hadamard matrix:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{9}$$

Another common example would be the controlled phase shift gate, a single-qubit phase shift gate that will leave the basis state $|0\rangle$ unchanged and map $|1\rangle$ to $\exp(i\phi)|1\rangle$, for a phase shift ϕ (such that the probability of measuring either a $|0\rangle$ or a $|1\rangle$ also remains unchanged).

In general, quantum gates are unitary operators that map the Hilbert space into itself. They are time evolution operators and they thus conserve energy and information. Moreover, although free quantum evolution requires no energy, true quantum gates instead require energy to operate, as one has to initiate and stop the interactions. Additionally, because quantum error corrections techniques make use of classical hardware, error correction is expected to be a major part of the energy budget in quantum computing.

Posted on Authorea 23 Nov 2020 — The copyright holder is the author/funder. All rights reserved. No reuse without permission. — https://doi.org/10.22541/au.160614804.47667838/v1 — This a preprint and has not been peer reviewed. Data may be preliminary.

The definition and use of the classical discrete Fourier transform

Before we move on to the first quantum algorithms, let us first examine the limitations of classical computation via one particular algorithm: the general number field sieve (GNFS).

The GNFS is currently the most efficient known classical algorithm for large integer factorization (particularly for integers exceeding 10^{100})(Pomerance, 1996). A generalization of the special number field sieve, this algorithm works in sub-exponential time, specifically of complexity $\mathcal{O}[\exp(1.9(\log N)^{1/3}(\log\log N)^{2/3})]$.

However, we compare it to one of the first quantum algorithms to ever spark interest in the field of quantum computation: Shor's algorithm(Shor, 1994). Via fast multiplication algorithms(Beckman et al., 1996), the algorithm need only take quantum gates of order $\mathcal{O}[(\log N)^2(\log\log N)(\log\log\log N)]$, thus being a member of the bounded-error quantum polynomial time (BQP) complexity class.

One distinguishing feature of the Shor's algorithm is that it requires only a few repeated measurements of the output state to obtain the desired result such that one does not need to know all the coefficients of the output state. This is done via the quantum period-finding subroutine, which does not load classical data and evaluate a full vector but instead finds the period by measuring the result of the QFT multiple times, which will be valuable for quantum speedup. In fact, basing factoring on period finding by using the QFT is the great innovation in Shor's quantum factoring algorithm.

In fact, there is an almost exponential difference between the complexities of the GNFS and Shor's algorithm, showing that quantum computing is, in principle, capable of performing tasks that no classical computer could ever perform: this phenomenon is often referred to as *quantum speedup*. To better visualize this complexity, Fig. 1 graphs the number of operations as function of the input size.

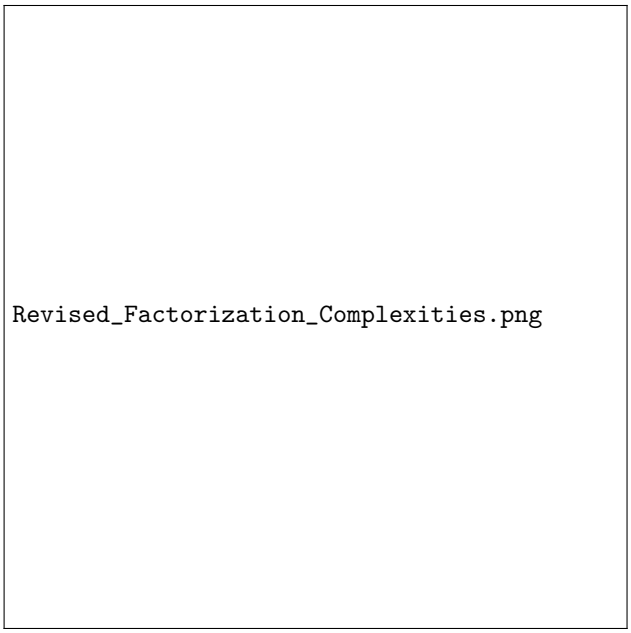

Figure 1: Scaling, represented by the number of operations as function of the algorithm's input size. (This graph was normalized to start at the origin, with SA denoting Shor's algorithm.)

Posted on Authorea 23 Nov 2020 — The copyright holder is the author/funder. All rights reserved. No reuse without permission. — https://doi.org/10.22541/au.160614804.47667838/v1 — This a preprint and has not been peer reviewed. Data may be preliminary.

Limitations of Classical Algorithms

Before introducing the QFT, for clarity, we define its classical counterpart: the discrete Fourier transform (DFT). The DFT maps a sequence of N complex numbers $\boldsymbol{x} = \{x_0, x_1, ..., x_{N-1}\}$ into another sequence, $\boldsymbol{y} = \{X_0, X_1, ..., X_{N-1}\}$, such that:

$$y_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn},\ k = 0, 1, 2, ..., N-1. \tag{10}$$

Moreover, an N-point DFT is often expressed as

$$\mathbf{y} = W\mathbf{x} \tag{11}$$

where W is the DFT matrix:

$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \dots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \dots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \dots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \dots & \omega^{(N-1)(N-1)} \end{bmatrix} \tag{12}$$

Here, $\omega = \exp(-2\pi i/N)$ is the primitive Nth root of unity.

This mathematical operation is typically implemented as an algorithm that minimizes the number of elementary arithmetic operations, which is known as the fast Fourier transform (FFT)(Zhou et al., 1992). This reduces the complexity by directly computing the DFT from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$(Lohne, 2017). Here, we will be using the Cooley-Tukey algorithm(Bekele, 2006) to implement the FFT.

The simplest implementation of the Cooley-Tukey algorithm is the radix-2 decimation-in-time (DIT) case, which divides a DFT of size N into two interleaved DFTs with size $N/2$ for each stage in the algorithm's recursion; the next section examines an implementation of this method in MATLAB.

The implementation of the radix-2 DIT case in MATLAB

The radix-2 DIT is a recursive algorithm, taking the following inputs: x, the relevant data, the input data x and its size $N = 2^n$, and the stride z (that is the distance in memory between consecutive values of x; we are taking an initial value of $z = 1$).

A possible implementation of the algorithm in MATLAB is as follows:

```
function d = DIT(x, N, z)
    if N == 1
        d(1) = x(1)
    else
        d(1:N/2) = DIT(x, N/2, 2*z)
        d(N/2+1:N) = DIT(x+z, N/2, 2*z)
        for k = 1:N/2
            t = d(k)
            d(k) =
              t + exp(-2*pi*i*(k-1)/N)*d(k+N/2)
            d(k+N/2) =
              t - exp(-2*pi*i*(k-1)/N)*d(k+N/2)
        end
    end
end
```

Posted on Authorea 23 Nov 2020 — The copyright holder is the author/funder. All rights reserved. No reuse without permission. — https://doi.org/10.22541/au.160614804.47667838/v1 — This a preprint and has not been peer reviewed. Data may be preliminary.

In the function's initial if statement, we account for the trivial case wherein x is a single element list; we thus set our output equal to the first member of x (which will, of course, always be equivalent to x).

In the else statement, we truly access the essence of the algorithm by first setting the first half of the output equal to the recursion of the function with a halved N and a doubled z (thus reiterating through the function until the elimination case is reached) and then setting the second half of the output to the shifted input by z with a halved N a doubled z. Afterwards, we interleave the two evaluations to get our full DIT. This results in some interesting complexity analysis, which is reviewed in the following section.

Evaluating the complexity of the radix-2 DIT implementation

The runtime of the input of our implementation need only depend on the size of our input, N. Thus, we can describe this runtime via the usage of some unknown function, $T(N)$. As a result, the total runtime of our DIT implementation can be expressed as $T(N)$, and the recursive computations in line 5 and 6 as $T(N/2)$ (since we are merely reevaluating the function with a reduced N). Lastly, the runtime of the computations present in the final for loop is proportional to N. However, since we are to evaluate the overall complexity of our algorithm, we can ignore any such scaling factors and simply write N. Therefore, we have:

$$T(N) = 2T(\frac{N}{2}) + N. \tag{13}$$

Next, we can utilize the master theorem for divide-and-conquer recurrences for a more concrete answer. The master theorem states that for recurrence relations of the form $T(N) = aT(N/B) + f(N)$, we determine the critical exponent $c_{\text{crit}} = \log_b a$. Therefore, we utilize the case of the master theorem stating that $f(N) = \mathcal{O}(N^{c_{\text{crit}}} \log^k N)$ for any $k \geq 0$, since we merely have $c_{\text{crit}} = \log_2 2 = 1$ and we can thus simply set $k = 0$ such that $\log^k N = \log^0 N = 1$. Therefore, in accordance to the master theorem, we determine the function as having complexity:

$$T(N) = \mathcal{O}(N^{c_{\text{crit}}} \log^{k+1} N) = \mathcal{O}(N \log N). \tag{14}$$

As expected, our implementation makes correct usage of the Cooley-Tukey algorithm, showing the aforementioned complexity.

Definition and usage of quantum Fourier transform

Akin to the DFT that we examined in Section V, the quantum Fourier transform (QFT) maps a quantum state $|x\rangle = \sum_{i=0}^{N-1} x_i|i\rangle$ to $\sum_{i=0}^{N-1} y_i|i\rangle$ in accordance to the formula:

$$y_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n \omega^{nk},\ k = 0, 1, 2, ..., N-1. \tag{15}$$

(We continue to use the notation of ω as a function of N: for further clarity, $\omega = \exp(-2\pi i/N)$ for $N = 2^n$.) Moreover, we can also re-express the QFT in a way that will be easier to simulate for our implementation(Tellez et al., 2008), which represents our input as a tensor product of states like so (here, s is used to better distinguish it from x):

$$|s\rangle = |s_1 s_2 \dots s_n\rangle = |s_1\rangle \otimes |s_2\rangle \otimes \cdots \otimes |s_n\rangle. \tag{16}$$

We also borrow fractional binary notation such that:

$$[0.s_1 \dots s_m] = \sum_{k=1}^{m} s_k 2^{-k}. \tag{17}$$

Posted on Authorea 23 Nov 2020 — The copyright holder is the author/funder. All rights reserved. No reuse without permission. — https://doi.org/10.22541/au.160614804.47667838/v1 — This a preprint and has not been peer reviewed. Data may be preliminary.

Therefore, we can express the QFT as:

$$\mathrm{QFT}(|s\rangle) = \frac{1}{\sqrt{2^n}} \bigotimes_{j=1}^{n} (|0\rangle + e^{2\pi i[0.s_j s_{j+1} \ldots s_n]}|1\rangle). \tag{18}$$

More precisely, eq. 18 represents a time evolution operator constituted by the composition of n^2 gates in parallel, some Hadamard gates, and some controlled phase shift gates. Here, the larger tensor product symbol functions as an indexed product of the Kronecker tensor product. Via decomposition, the QFT on 2^n amplitudes can be implemented as a quantum circuit consisting of $\mathcal{O}(n^2)$ Hadamard gates and controlled phase shift gates for a number of qubits n.

The previously classical DFT would, in our implementation, require $\mathcal{O}(n2^n)$ gates, which is exponentially greater than the aforementioned quantum complexity. This may also be compared with $\mathcal{O}(n \log n)$, the number of gates required by the most efficient known quantum Fourier transform algorithms for an approximate result.

This analysis, however, does not take into account the complexity of preparing the input state and measuring the output states, which both have complexity $\mathcal{O}(N \log 1/\epsilon)$ for a required resolution ϵ(Shende et al., 2006). It is therefore evident that in the case of the best-known QFT algorithm, the complexity is completely dominated by these measurements. Furthermore, if one wants to read out the full output vector the complexity becomes $\mathcal{O}(N^2 \log^2 1/\epsilon)$. The classical FFT algorithm also has a dependency on the desired precision, but we treat it as a constant depending on machine precision, not considering arbitrary precision implementations of the classical algorithm.

However, the complexity analysis of our simulation on a classical computer will not be fully accurate to the innate properties of a quantum device. On a classical computer, we cannot simulate the full QFT algorithm but we can simulate the application of the unary operator that corresponds to the QFT algorithm to one base state. We first examine a gate-level accurate implementation that we may write in pseudocode. As the following image indicates, the QFT may be constructed with Hadamard gates (written H) and controlled phase shift gates (written R_m).

In pseudocode, we may write:

```
function QFT(s):
    for i in 1..n:
        H(s_i)
        for m in 2...{n-i+1}
            R_m(s_i)
```

Where $H(..)$ is an application of the Hadamard gate and $R_m(...)$ is an application of the controlled phase shift gate as in:

$$R_m = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$$

for some phase shift ϕ. They both may be thought of as operators acting in place on the state s_i.

The above algorithm consists of two nested loops therefore it has a complexity of $\mathcal{O}(n^2) = \mathcal{O}(\log^2 N)$.

Since each quantum gate can be simulated in $\mathcal{O}(N)$, and taking advantage of the possibility of parallelizing some of the transformations implemented by the gates, then we can simulate the above circuit in $\mathcal{O}(N \log^2 N)$.

We provide a possible implementation of this algorithm in MATLAB[1] using the QUBIT4MATLAB(Toth, 2008) library.

[1]Source code of the algorithm in MATLAB may be found at https://github.com/DamianRMusk/MATLAB_FT.

Posted on Authorea 23 Nov 2020 — The copyright holder is the author/funder. All rights reserved. No reuse without permission. — https://doi.org/10.22541/au.160614804.47667838/v1 — This a preprint and has not been peer reviewed. Data may be preliminary.

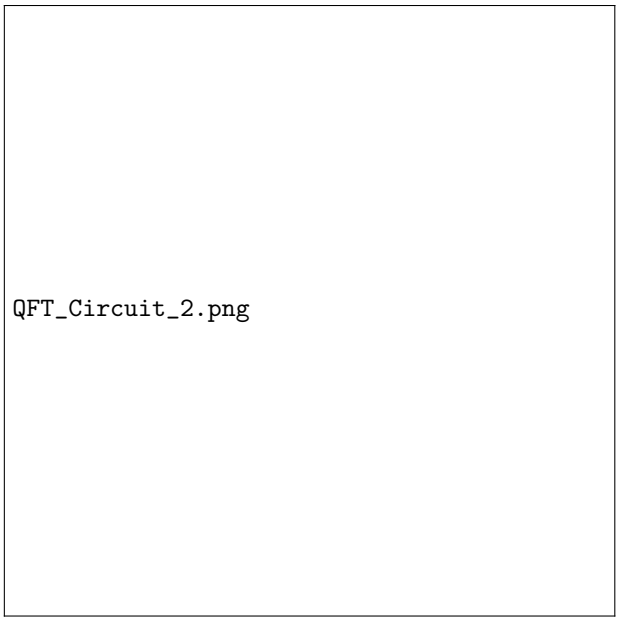

Figure 2: A quantum circuit drawn for the QFT for n qubits.

Evaluating the complexity of the QFT

In the previous section, we discussed the running time of the gate-accurate description of the QFT algorithm and proceeded to multiply by N (the cost of simulating the gate for each possible input) to obtain the gate-accurate simulation time.

We now discuss the complexity of eq. 18 given one input s. One way to implement eq. 18 consists of precomputing all required values of $[0.s_j...s_n]$ (eq. 17) for each j. Since we have n possible values of j and the sum has a complexity proportional to n for each j, this operation has a total complexity of n^2.

We now consider the complexity of computing the Kronecker products in eq. 18. This tensor product is effectively just a form of multiplication: using the associative property, we see that first product involves two vectors of size 2, resulting in a vector of size 4. The second product involves a vector of size 4 and a vector of size 2, resulting in a vector of size 8. This continues to the final result of size $N = 2^n$.

Thus, we effectively sum over 2^j as:

$$\sum_{j=1}^{j<n} 2^j = \frac{2^n - 1}{2 - 1} = 2^n - 1 \in \mathcal{O}(2^n). \tag{19}$$

To find the full complexity, we sum and take the greater term: $\mathcal{O}(n^2 + 2^n) = \mathcal{O}(2^n) = \mathcal{O}(N)$.

This possible implementation of the algorithm is faster than the gate accurate simulation previously discussed.

In principle, we take N basis vectors to account for arbitrary N coefficients, we should repeat these calculations N times. Yet, in practice, in simulation we are allowed to copy vectors therefore this does not affect the overall running time.

In evaluating the complexity of our function, we are mostly concerned with the complexity of the computations present in the repeated Kronecker product. We can compute its computational complexity in the

Posted on Authorea 23 Nov 2020 — The copyright holder is the author/funder. All rights reserved. No reuse without permission. — https://doi.org/10.22541/au.160614804.47667838/v1 — This a preprint and has not been peer reviewed. Data may be preliminary.

simulation; however, this complexity does not appear in the (actual) quantum computer. We can represent the differences between the simulation and the true quantum computations as:

> We start with our initial state X, i.e. a superposition of base vectors. In the actual computation, the calculation of each component is parallelized at no cost but we can only measure one coefficient. We therefore need to rebuild the output state N times in order to be able to perform the N measurements needed to obtain all the coefficients.
>
> However, in the simulation, we must loop through the 2^n vector bases. As a result, the computation to generate the output state appears slower than the true quantum computation. Yet, because we can copy this state to perform mutiple measurments, the net complexity is not slower. This apparent discrepancy is mostly due to the effect of the Kroneker delta computation.
>
> In true quantum information, we are only capable of extracting one measurement at a time. Thus, we compute every number 2^n faster than the solution. This forms a black box which computes every coefficient of the transform incredibly quickly: however, we may only measure one coefficient or linear combination of coefficients per each iteration of the function (since the QFT's wave function will collapse every time we call the black box, thus necessitating that the computations be redone).

In a true quantum computation, there is no cost in the Kronecker products discussed in the previous section. If the QFT was implemented as a series of Hadamard gates, this would require n^2 Hadamard gates (as was mentioned following the definition of the QFT in Section VI). Each of the $n^2 = \log^2 N$ gates require $O(N)$ classical operations to be simulated, thus, the total cost is $O(N \log^2 N)$.

Conclusions

In the following table, we can directly compare the formulated algorithmic complexities (to write terms in a uniform manner, we have reverted to expressing complexities in terms of $N = 2^n$ gates for the quantum algorithms):

Table 1: Our calculated complexities calculated by applying various complexity analysis techniques to our previously defined implementations. ("Sim" and "theo" are abbreviations for "simulated" and "theoretical," respectively.

DFT	$\mathcal{O}(N \log N)$
QFT (gate-level accurate)	$\mathcal{O}(N \log^2 N)$
QFT (theo, no measurement)	$\mathcal{O}(\log^2 N)$
QFT (theo + preparation + measurement)	$\mathcal{O}(N^2 \log^2 1/\epsilon)$

We can graph the complexities as in the figure below.

Accordingly, the complexities of our algorithms are rather distinct. The exponential (the DFT) increases far more rapidly than the other functions, the product of the input and the logarithm of the input (the simulated QFT) lies in-between the others, with the squared logarithm (the theoretical QFT) residing far below the two other functions, increasing at an almost linear rate: these visualized results further bring the concept of quantum speedup into question.

The previous results appear to confirm that true quantum algorithms and mere simulations of them have significantly different running times. In the case of the QFT, this is mostly due to the costliness of the iterated Kronecker tensor product for an arbitrary number of basis vectors. However, we arrive at the question of whether or not these complexities can truly be ranked directly. By the principles of quantum information, to acquire results more comparable to those given by the classical DFT, we must make extra preparations to our black box that, in turn, cost us a substantial amount of computing power (potentially indicating that, in actuality, the algorithms have comparable runtimes when evaluating the Fourier transforms in full).

Posted on Authorea 23 Nov 2020 — The copyright holder is the author/funder. All rights reserved. No reuse without permission. — https://doi.org/10.22541/au.160614804.47667838/v1 — This a preprint and has not been peer reviewed. Data may be preliminary.

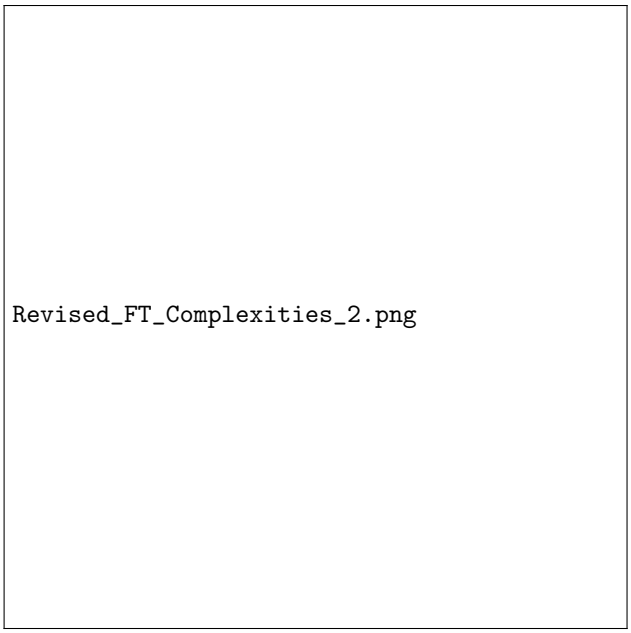

Figure 3: Our complexities in the form of a MATLAB graph, displaying the various forms of growth between the relative size of an algorithm's input and its overall complexity. (We have regularized some complexities to start at the origin.)

More specifically, if we want to measure each coefficient, we must redo our operations for each coefficient (since our wave function will collapse for every measurement). This is not necessary for the classical DFT; upon making these modifications to the theoretical QFT such that it becomes more analogous to the DFT, it is possible that (in the case of Fourier analysis) quantum computing is no better than classical computing. However, this does not necessarily mean that quantum speedup is nonexistent; in the case of integer factorization, wherein we only desire a single answer that meets are conditions, the parallelization of quantum computing can become extremely useful (as mentioned in the Introduction in the case of Shor's algorithm, although this algorithm need only sample the wave function to calculate prime factors). In fact, this parallelization becomes beneficial for *any* type of search problem given the appropriate search criteria(Grover, 1997).

The primary goal of this paper was to present the reader with a non-trivial example to illustrate that directly comparing quantum algorithms to classical ones is a significant issue and there are a considerable number of potential hazards in doing so. Furthermore, in the case of some quantum algorithms, the perceived computational advantage may disappear should all elements of the wave function need to be read out, and there conversely exists speedup potential if only some elements need to be sampled. Therefore, although quantum speedup is certainly real for a number of algorithms, it is not necessarily a given.

I would like to thank Professor Arthur Western for
his guidance while working under him at Pioneer Academics. Moreover, I would like to acknowledge Lorena of article title (-1) disabled
Barba and Matthias Troyer for their very detailed feedback and suggestions(Troyer, 2020).