CS621/CSL611
Quantum Computing For Computer Scientists

Quantum Search

Dhiman Saha

Winter 2024

IIT Bhilai

Quantum Search

Shor's Factoring Algorithm

3.1 Period finding, factoring, and cryptography

Simon's problem (Section 2.5) starts with a subroutine that calculates a function $f(x)$, which satisfies $f(x) = f(y)$ for distinct x and y if and only if $y = x \oplus a$, where $\oplus$ denotes the bitwise modulo-2 sum of the n-bit integers a and x. The number of times a classical computer must invoke the subroutine to determine a grows exponentially with n, but with a quantum computer it grows only linearly.

Periodicity Under Modulo-2 Addition

This is a rather artificial example, of interest primarily because it gives a simple demonstration of the remarkable computational power a quantum computer can possess. It amounts to finding the unknown period a of a function on n-bit integers that is "periodic" under bitwise modulo-2 addition. A more difficult, but much more natural problem is to find the period r of a function f on the integers that is periodic under *ordinary* addition, satisfying $f(x) = f(y)$ for distinct x and y if and only if x and y differ by an integral multiple of r. Finding the period of such a periodic function turns out to be the key to factoring products of large prime numbers, a mathematically natural problem with quite practical applications.

Motivation Why Period-Finding Under Addition is Difficult?

One might think that finding the period of such a periodic function ought to be easy, but that is only because when one thinks of periodic functions one tends to picture slowly varying continuous functions (like the sine function) whose values at a small sample of points within a period can give powerful clues about what that period might be. But the kind of periodic function to keep in mind here is a function on the integers whose values within a period r are virtually random from one integer to the next, and therefore give no hint of the value of r.

The best known classical algorithms for finding the period r of such a function take a time that grows faster than any power of the number n of bits of r (exponentially with $n^{1/3}$). But in 1994 Peter Shor discovered that one can exploit the power of a quantum computer to learn the period r, in a time that scales only a little faster than n^3.

Shor's Algorithm

1995, AT&T **Peter Shor**

"Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer."

- Solves a more general class of problems than factoring and discrete logarithms
- Computing period of a periodic function
- a function $f()$ is periodic, if theres a ω (the period) such that $f(x+\omega) = f(x)$ for any x,
- Shors algorithm will efficiently find ω.

Major Implication in Crypto

The ability of Shors algorithm to efficiently compute the period of a function can be used to attack public-key cryptography

- Finding the values of the function

$$f_{a,N}(x) = a^x \bmod N$$

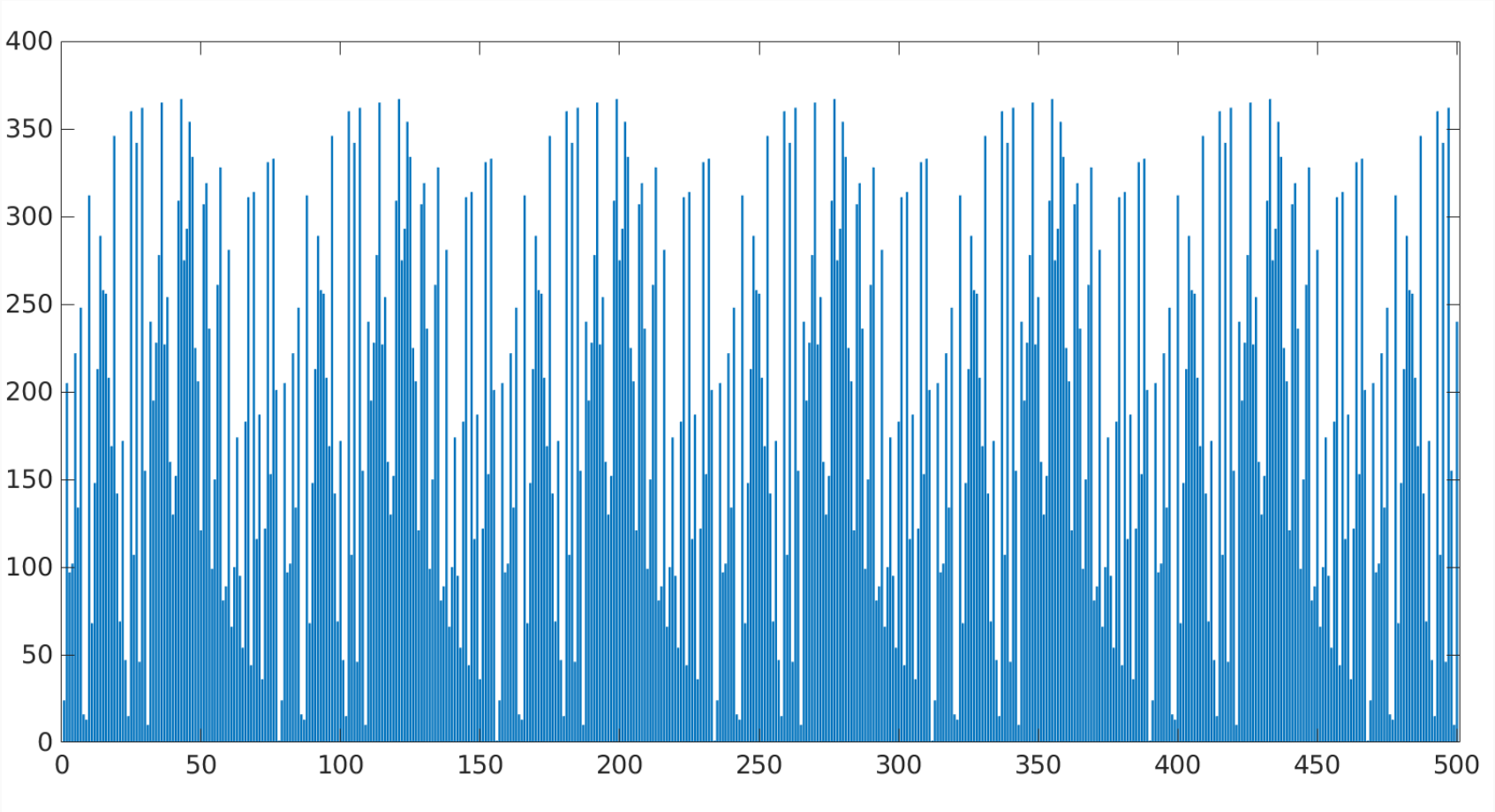
Some examples

x	0	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
$f_{2,15}(x)$	1	2	4	8	1	2	4	8	1	2	4	8	1	$\cdots$
$f_{4,15}(x)$	1	4	1	4	1	4	1	4	1	4	1	4	1	$\cdots$
$f_{13,15}(x)$	1	13	4	7	1	13	4	7	1	13	4	7	1	$\cdots$

- Can you spot the pattern?

Visualizing The Pattern

$f_{24,371}(x)$



$24^x \bmod 371$

Theorem

For any co-prime $a \leq N$, the function $f_{a,N}$ will output a 1 for some $r < N$.

- **Implication**: After it hits 1, the sequence of numbers will simply repeat.

$$f_{a,N}(r) = 1$$
$$f_{a,N}(r+1) = f_{a,N}(1)$$
$$\vdots$$
$$f_{a,N}(r+s) = f_{a,N}(s)$$

- r is known as the order of a mod N

Definition (Order of a modulo N)

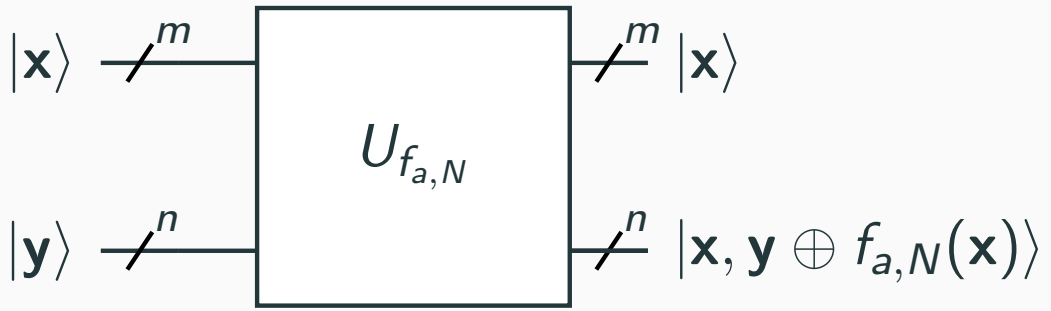
For $a \in \mathbb{Z}_N^*$, the **order** of a in $\mathbb{Z}_N^*$ is the smallest positive integer r such that

$$a^r \equiv 1 \bmod N$$

- **Input:** A positive integer $N \geq 2$ and an element $a \in \mathbb{Z}_N^*$.
- **Output:** The order of a in $\mathbb{Z}_N^*$.
- Classical Solution
  - Hard to solve.
  - Computing powers of a modulo N until 1 is obtained
  - Can take time exponential in $\log N$.

Order Finding forms the **Quantum** part of Shor's factoring algorithm

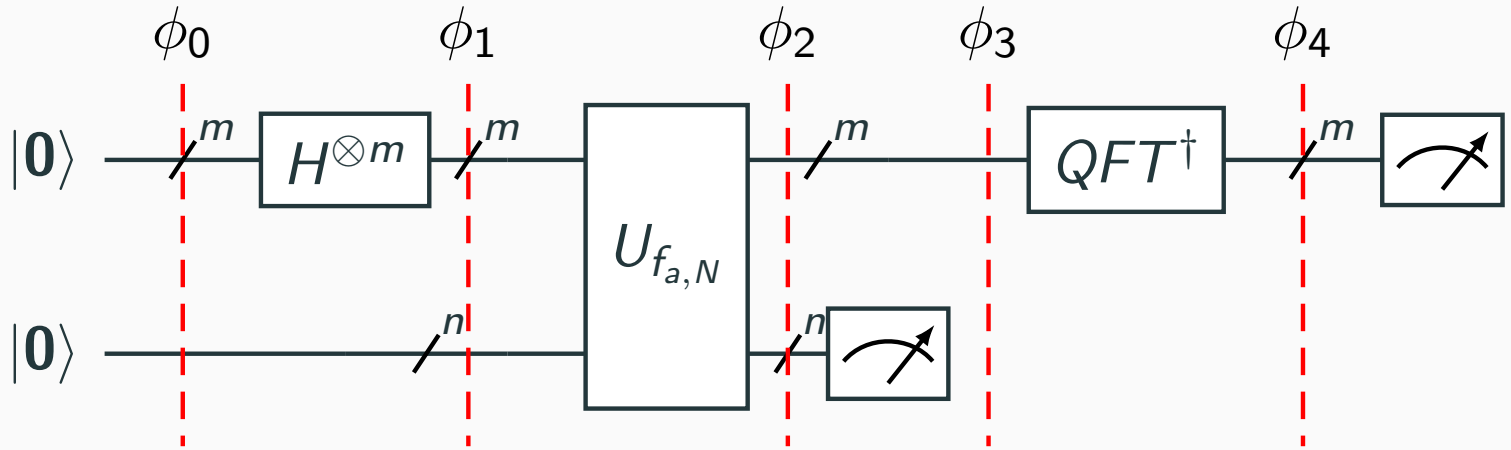
- Deploy a quantum computer with its ability to be in a **superposition** to calculate $f_{a,N}(x)$ for all needed x.
- The output of this function will always be less than N, and so we will need $n = \log_2 N$ output bits
- Need to evaluate $f_{a,N}$ for at least the first N^2 values of x and so will need at least $m = \log_2 N^2 = 2\log_2 N =$ $2n$ input qubits.



$$|\mathbf{x}\rangle, |\mathbf{y}\rangle \rightarrow |\mathbf{x}\rangle |\mathbf{y} \oplus a^{\mathbf{x}} \bmod N\rangle$$

Circuit formation of $U_{f_{a,N}}$ will be discussed later

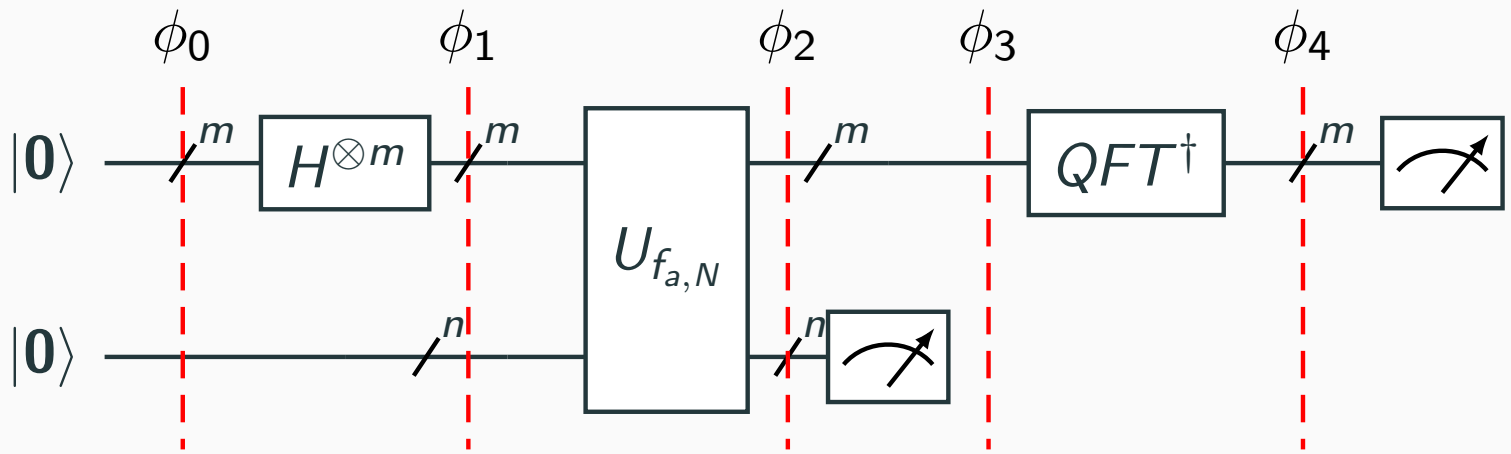
Quantum Period Finding



$$(Measure \otimes I)(QFT^\dagger \otimes I)(I \otimes Measure)U_{f_{a,N}}(H^{\otimes m} \otimes I)\,|\mathbf{0}_m\rangle\,|\mathbf{0}_n\rangle$$

- Here $\mathbf{0}_m$ and $\mathbf{0}_n$ are qubit strings of length m and n, respectively.

Circuit Slicing: $|\phi_1\rangle$



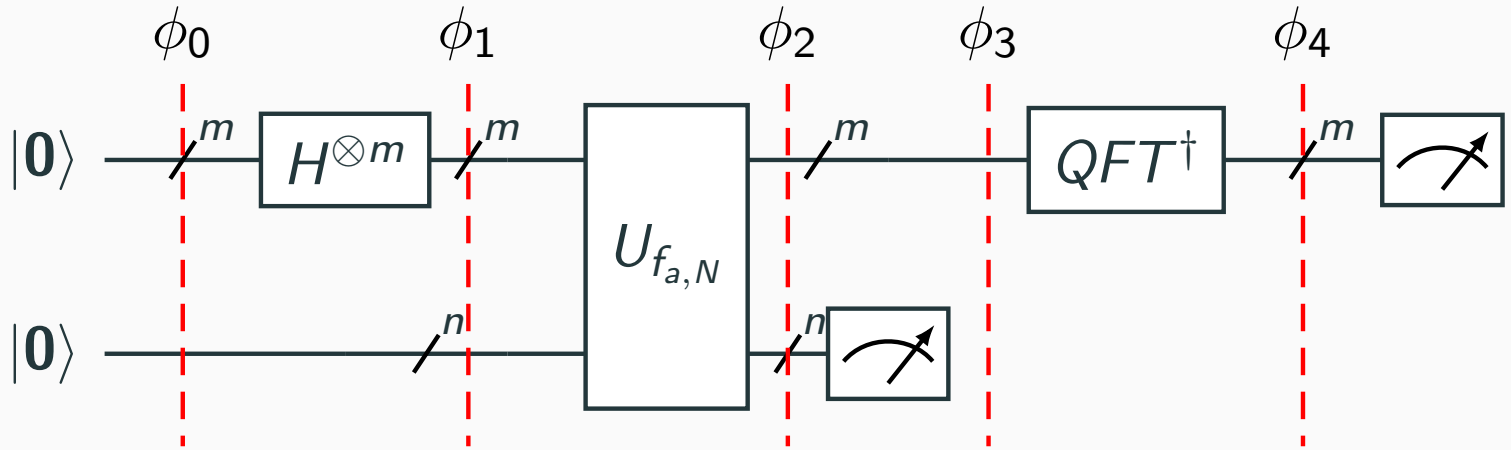
- Starting state:

$$|\phi_0\rangle = |\mathbf{0}_m\rangle\,|\mathbf{0}_n\rangle$$

- After $H-$transforms, we get equally weighted superposition of all possible inputs

$$|\phi_1\rangle = \frac{\sum_{\mathbf{x}\in\{0,1\}^m} |\mathbf{x}\rangle\,|\mathbf{0}_n\rangle}{\sqrt{2^m}}$$

Circuit Slicing: $|\phi_2\rangle$



- Next we get evaluation of f on all these possibilities:

$$|\phi_2\rangle = \frac{\sum_{\mathbf{x}\in\{0,1\}^m} |\mathbf{x}\rangle\,|f_{a,N}(\mathbf{x})\rangle}{\sqrt{2^m}} = \frac{\sum_{\mathbf{x}\in\{0,1\}^m} |\mathbf{x}\rangle\,|a^{\mathbf{x}} \bmod N\rangle}{\sqrt{2^m}}$$

- These outputs repeat and repeat $\implies$ they are **periodic**

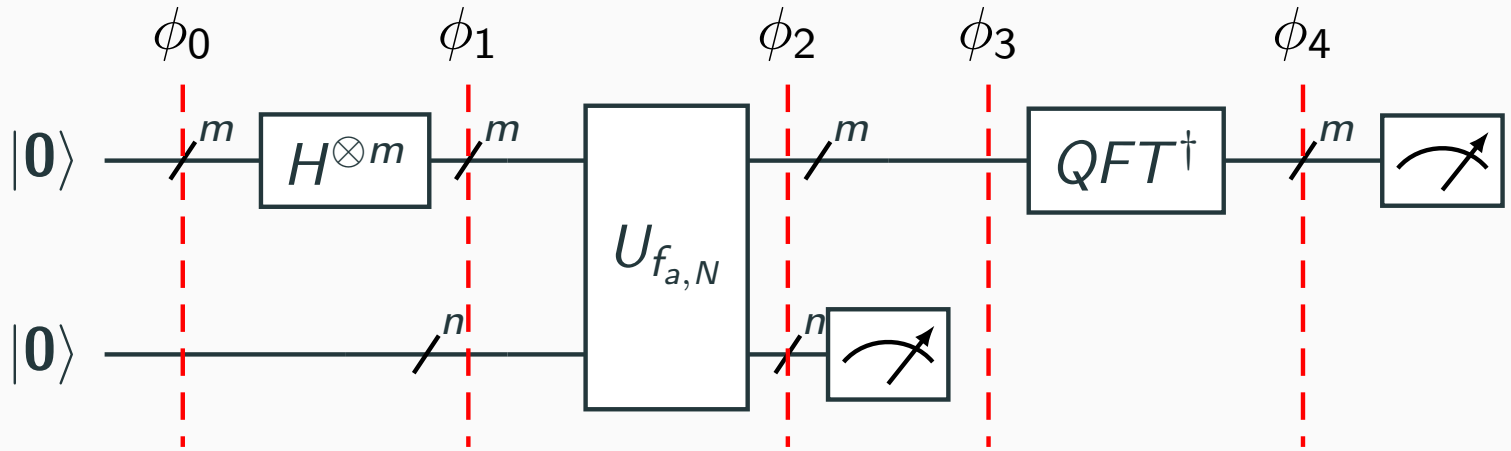
Some Examples of $|\phi_2\rangle$

- **Case-1:** $N = 15$, we have $n = 4$ and $m = 8$. For $a = 13$:

$$\frac{|0,1\rangle + |1,13\rangle + |2,4\rangle + |3,7\rangle + |4,1\rangle + \cdots + |254,4\rangle + |255,7\rangle}{\sqrt{256}}$$

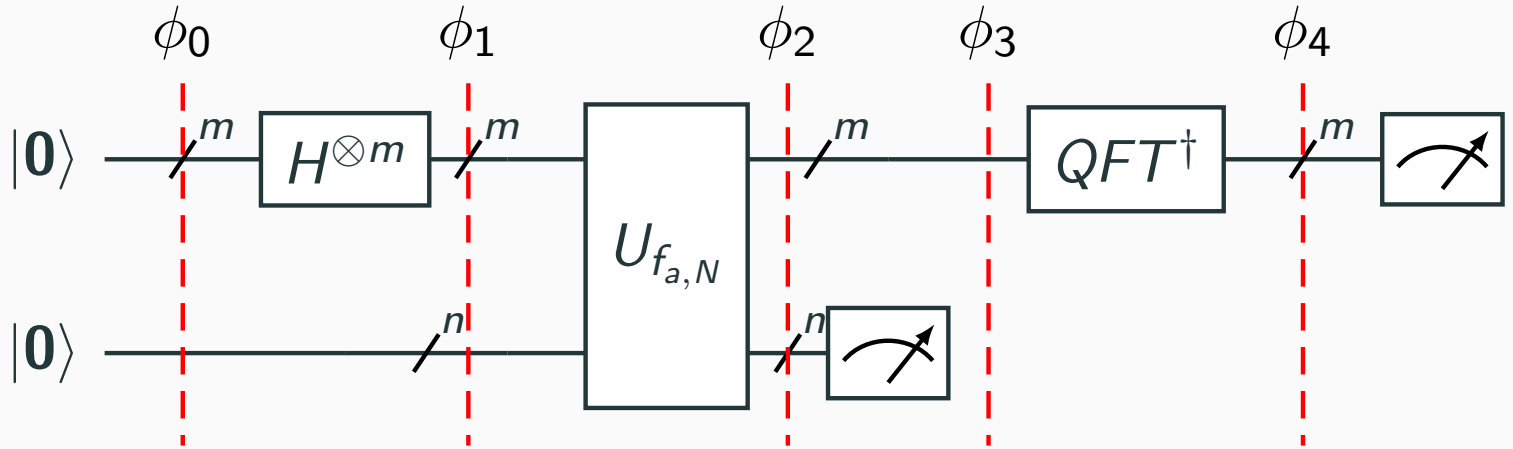
- **Case-2:** $N = 371$, we have $n = 9$ and $m = 18$. For $a = 24$:

$$\frac{\begin{aligned}&|0,1\rangle + |1,24\rangle + |2,205\rangle + |3,97\rangle + |4,102\rangle + \cdots + \\ &\left|2^{18}-1, 24^{2^{18}-1} \bmod 371\right\rangle\end{aligned}}{\sqrt{2^{18}}}$$



- Next, measure the bottom qubits of $|\phi_2\rangle$
- Let us say that after measuring the bottom qubits we find $a^{\bar{\mathbf{x}}} \bmod N$ for some $\bar{\mathbf{x}}$
- Due to periodicity of $f_{a,N}$ we also have: $a^{\bar{\mathbf{x}}} \bmod N \equiv a^{\bar{\mathbf{x}}+r} \bmod N$, $a^{\bar{\mathbf{x}}} \bmod N \equiv a^{\bar{\mathbf{x}}+2r} \bmod N, \cdots$
- For any $s \in \mathbb{Z}$, we have $a^{\bar{\mathbf{x}}} \bmod N \equiv a^{\bar{\mathbf{x}}+sr} \bmod N$

Circuit Slicing: $|\phi_3\rangle$



- How many of the 2^m superpositions $\mathbf{x}$ in $|\phi_2\rangle$ have $a^{\bar{\mathbf{x}}} \bmod N$ as the output? **Answer:** $\left\lfloor \frac{2^m}{r} \right\rfloor$

$$|\phi_3\rangle = \frac{\sum_{a^{\mathbf{x}} \equiv a^{\bar{\mathbf{x}}} \bmod N} |\mathbf{x}\rangle \left| a^{\bar{\mathbf{x}}} \bmod N \right\rangle}{\left\lfloor \frac{2^m}{r} \right\rfloor}$$

Alternative View of $|\phi_3\rangle$

$$|\phi_3\rangle = \frac{\sum_{a^{\mathbf{x}} \equiv a^{\bar{\mathbf{x}}} \bmod N} |\mathbf{x}\rangle \left|a^{\bar{\mathbf{x}}} \bmod N\right\rangle}{\left\lfloor \frac{2^m}{r} \right\rfloor}$$

$$|\phi_3\rangle = \frac{\sum_{j=0}^{\frac{2^m}{r-1}} |t_0 + jr\rangle \left|a^{\bar{\mathbf{x}}} \bmod N\right\rangle}{\left\lceil \frac{2^m}{r} \right\rceil}$$

- Here t_0 is the first time that a $t_0 \equiv a^{\bar{\mathbf{x}}} \bmod N$, i.e., the **first time** that the measured value occurs.
- t_0 is referred to as the **offset** of the **period**

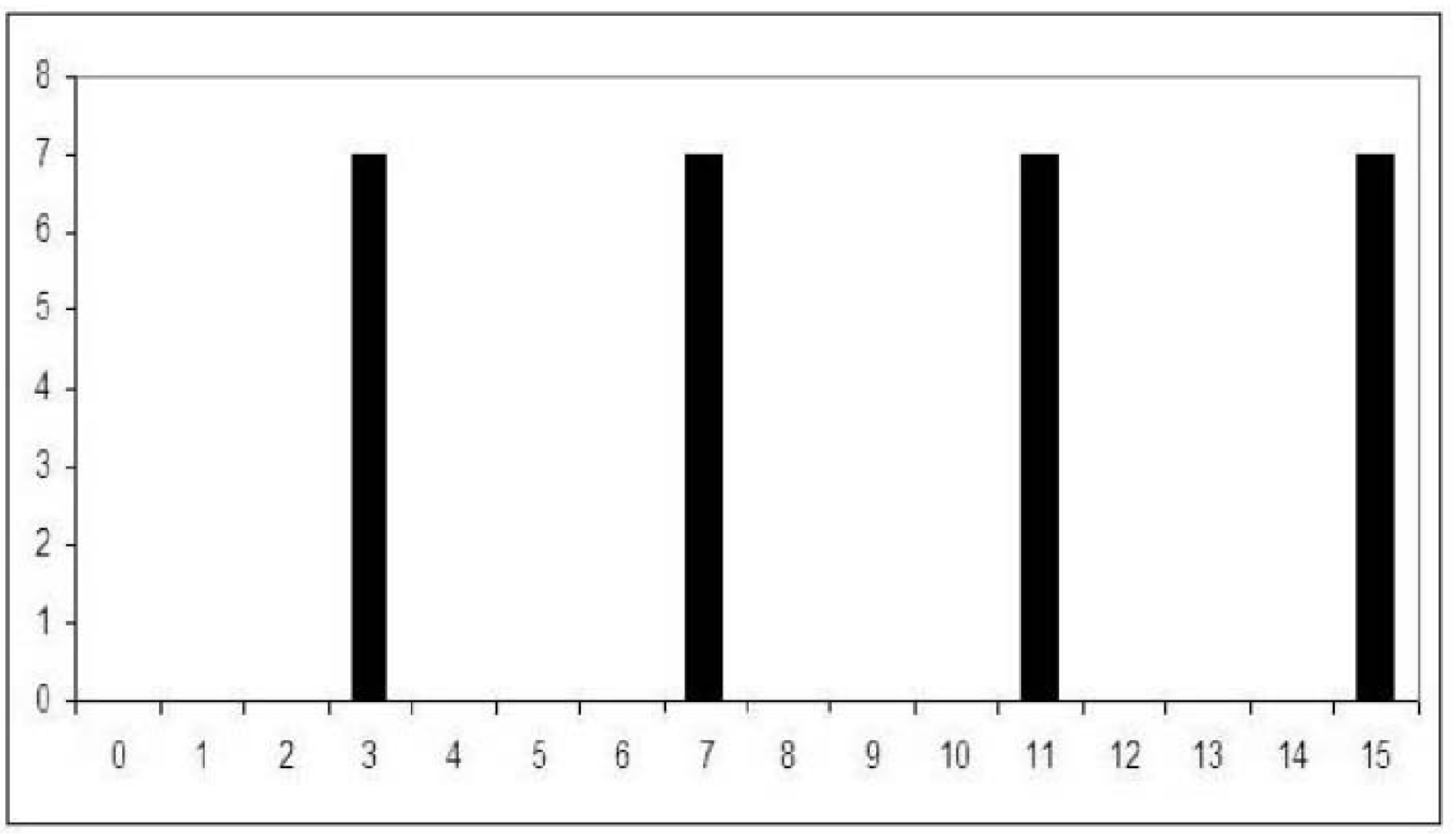
- $N = 15$, we have $n = 4$ and $m = 8$. For $a = 13$:

$$\frac{|0,1\rangle + |1,13\rangle + |2,4\rangle + |3,7\rangle + |4,1\rangle + \cdots + |254,4\rangle + |255,7\rangle}{\sqrt{256}}$$

- Let us say that after measurement of the bottom qubits, 7 is found.
- In that case $|\phi_3\rangle$ would be

$$\frac{|3,7\rangle + |7,7\rangle + |11,7\rangle + |15,7\rangle + \cdots + |251,7\rangle + |255,7\rangle}{\left[\frac{256}{4}\right]}$$

$f_{13,15}$ after a measurement of 7



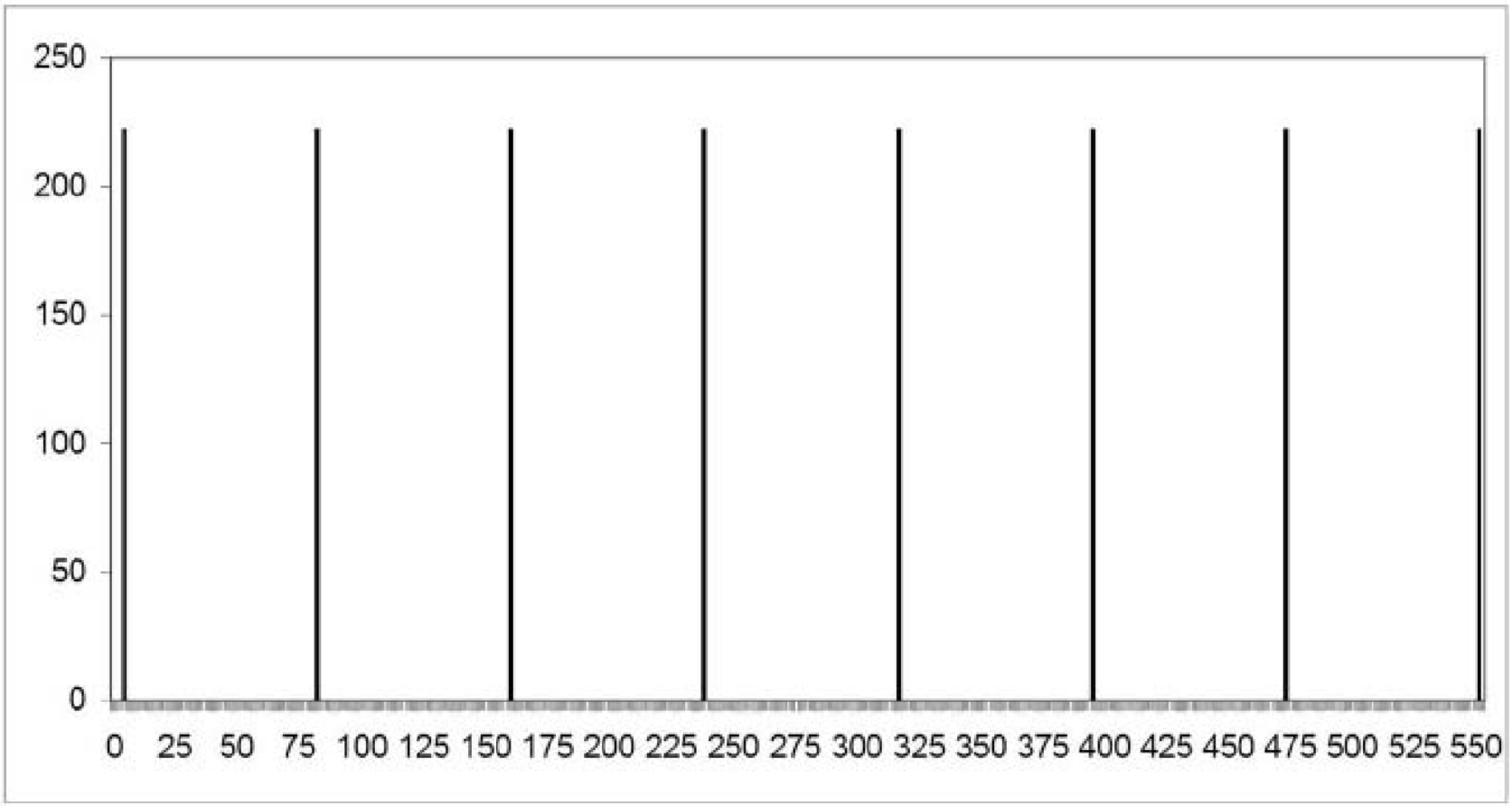
- $N = 371$, we have $n = 9$ and $m = 18$. For $a = 24$:

$$\frac{|0,1\rangle + |1,24\rangle + |2,205\rangle + |3,97\rangle + |4,102\rangle + \cdots + \left|2^{18}-1, 24^{2^{18}-1} \bmod 371\right\rangle}{\sqrt{2^{18}}}$$

- Let us say that after measurement of the bottom qubits we find 222 (which is $24^5 \bmod 371$)
- In that case $|\phi_3\rangle$ would be

$$\frac{|5,222\rangle + |83,222\rangle + |161,222\rangle + |239,222\rangle + \cdots}{\left\lceil \frac{2^{18}}{78} \right\rceil}$$

$f_{24,371}$ after a measurement of 222



- The final step of the quantum part of the algorithm is to take such a superposition and return its **period**
- This will be done with a type of **Fourier transform**
- For the rest of the lecture **no background** on Fourier transforms is assumed.
- We now take a detour.

- Consider the polynomial

$$P(x) = a_0 + a_1 x^1 + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1}$$

- The polynomial can be represented by a column vector

$$\left[a_0, a_1, a_2, \ldots, a_{n-1} x^{n-1}\right]^T$$

- Task: Evaluate the polynomial as the numbers

  $x_0, x_1, x_2, \ldots, x_{n-1}$
- To find: $P(x_0), P(x_1), P(x_2), \ldots P(x_{n-1})$
- Can this be done with a matrix multiplication?

The Vandermonde Matrix

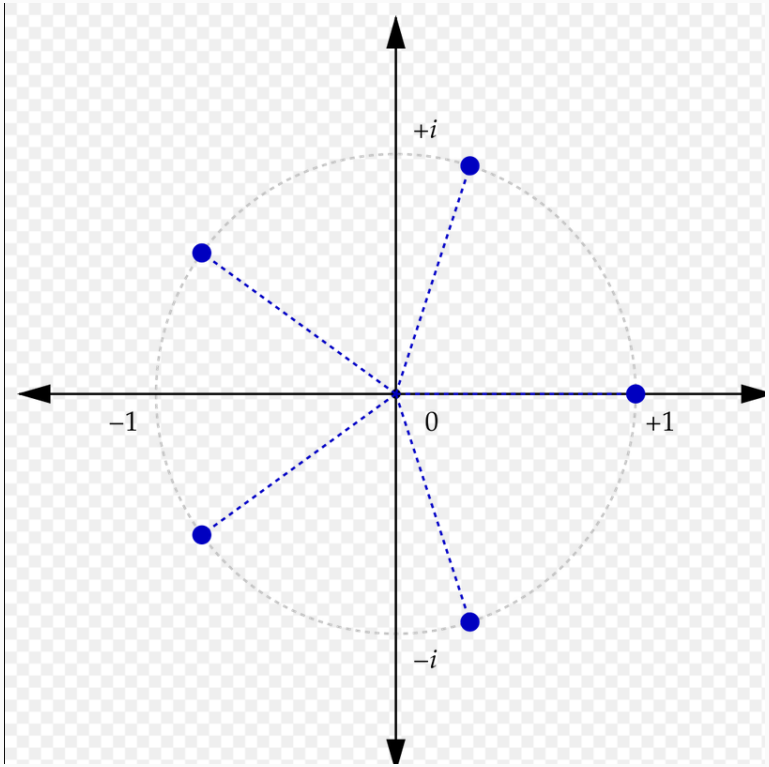
$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^j & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^j & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^j & \cdots & x_2^{n-1} \\ \vdots & & & & \vdots & & \vdots \\ 1 & x_k & x_k^2 & \cdots & x_k^j & \cdots & x_k^{n-1} \\ \vdots & & & & \vdots & & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^j & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_k \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} P(x_0) \\ P(x_1) \\ P(x_2) \\ \vdots \\ P(x_k) \\ \vdots \\ P(x_{n-1}) \end{bmatrix}$$

- The matrix on the left is called the **Vandermonde** matrix
- Here every row is a **geometric** series
- The **Vandermonde** matrix is denoted by $\mathcal{V}(x_0, x_1, \ldots, x_{n-1})$

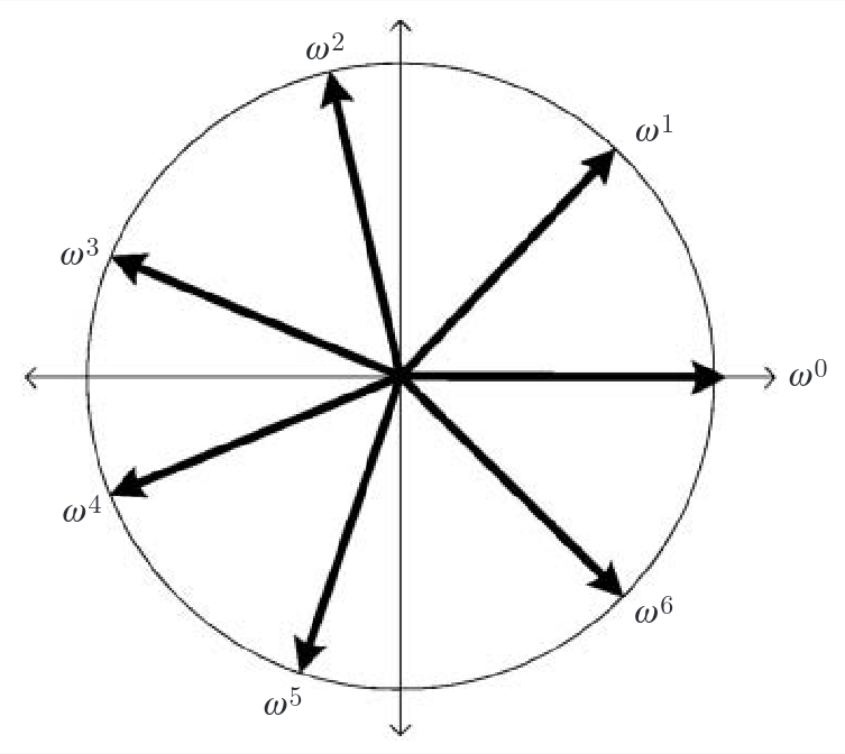
Using the Vandermonde matrix

- No restriction on the type of numbers permitted to use in the Vandermonde matrix
- Permitted to use complex numbers.
- Need them to be powers of the M^{th} **roots of unity**, ω_M
- Since M is fixed we shall simply denote this as ω_M as ω.
- Letting $M = 2^m$
- Recall this is the amount of numbers that can be described with the top qubits
- Required size of Vandermonde matrix is $M-\text{by}-M$

A **root of unity** is any complex number that yields 1 when raised to some positive integer power n.



The 5th roots of unity (blue points) in the complex plane



The seventh root of unity and its powers.

- Note that the roots **equally** partition the unit circle

Using the Vandermonde matrix

- Target: Evaluate the polynomials at $(\omega^0 = 1), \omega, \omega^2, \ldots, \omega^{M-1}$.
- To do this need to look at $\mathcal{V}(\omega^0, \omega^1, \omega^2, \ldots, \omega^{M-1})$
- In order to evaluate $P(x)$ at the powers of the M^{th} root of unity, we must multiply

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \cdots & \omega^j & \cdots & \omega^{M-1} \\ 1 & \omega^2 & \omega^{2\times 2} & \cdots & \omega^{2j} & \cdots & \omega^{2(M-1)} \\ \vdots & & & & \vdots & & \vdots \\ 1 & \omega^k & \omega^{k2} & \cdots & \omega^{kj} & \cdots & \omega^{k(M-1)} \\ \vdots & & & & \vdots & & \vdots \\ 1 & \omega^{M-1} & \omega^{(M-1)2} & \cdots & \omega^{(M-1)j} & \cdots & \omega^{(M-1)(M-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_k \\ \vdots \\ a_{M-1} \end{bmatrix} = \begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^k) \\ \vdots \\ P(\omega^{M-1}) \end{bmatrix}$$

- $\left[P(\omega^0), P(\omega^1), P(\omega^2), \ldots, P(\omega^{M-1})\right]^T$ is the vector of the values of the polynomial at the powers of the M^{th} root of unity

Definition (Discrete Fourier transform)

$$DFT = \frac{1}{\sqrt{M}}\mathcal{V}(\omega^0, \omega^1, \omega^2, \ldots, \omega^{M-1})$$

- Formally, DFT is defined as

$$DFT[j,k] = \frac{1}{\sqrt{M}}\omega^{jk}$$

- It can be shown that DFT is a unitary matrix

The Adjoint of DFT

- $DFT^{\dagger}$ is formally defined as

$$DFT^{\dagger}[j,k] = \frac{1}{\sqrt{M}}\overline{\omega^{kj}} = \frac{1}{\sqrt{M}}\omega^{-jk}$$

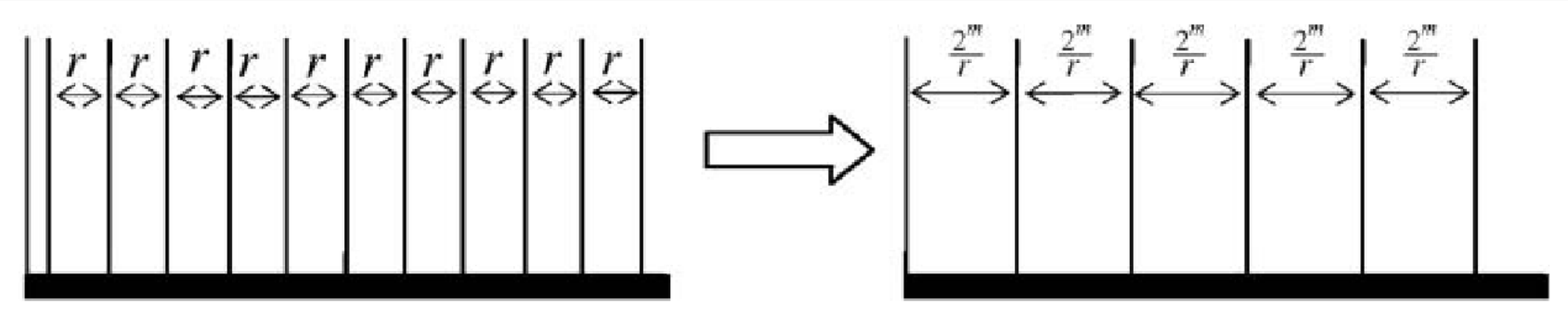
- One can verify that $DFT \star DFT^{\dagger} = I \implies$ DFT is unitary

---

We are skipping the details and concentrating more on the intuition

- Ignore the normalization $\frac{1}{\sqrt{M}}$ for the moment
- The matrix DFT acts on polynomials by evaluating them on different **equally spaced points** of the circle.
- The outcomes of those evaluations will necessarily have periodicity because the points go around and around the circle.
- So multiplying a column vector with DFT takes a sequence and outputs a periodic sequence.
- If we **start with a periodic column vector**, then the DFT will **transform** the periodicity.
- Similarly, the inverse of the Fourier transform, $DFT^{\dagger}$, will also change the periodicity

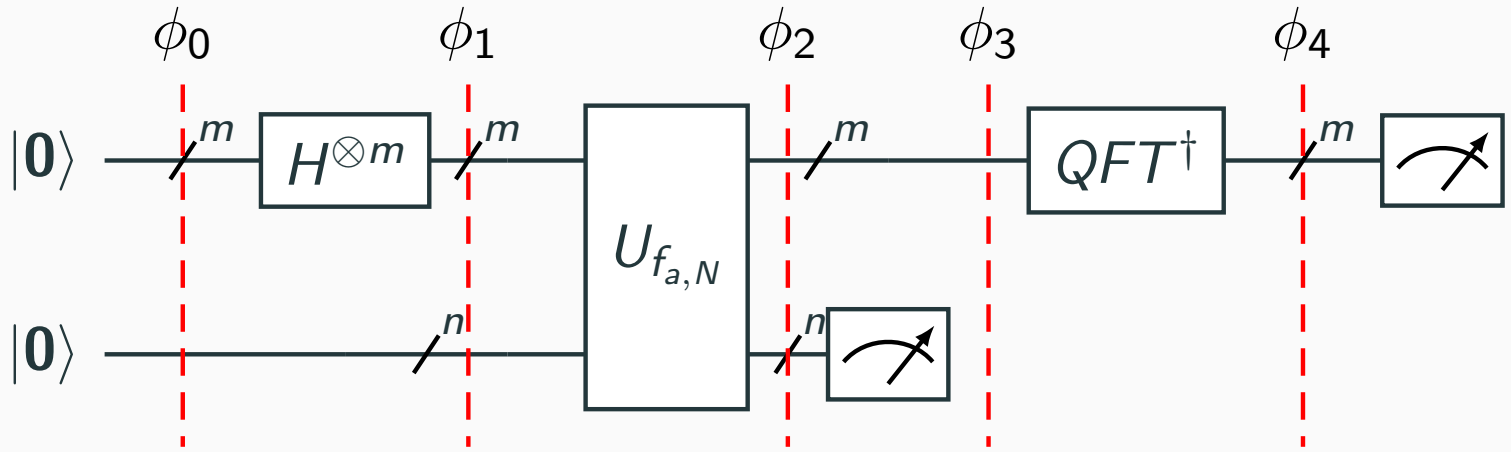
What task does $DFT^{\dagger}$ perform?



The action of $DFT^{\dagger}$

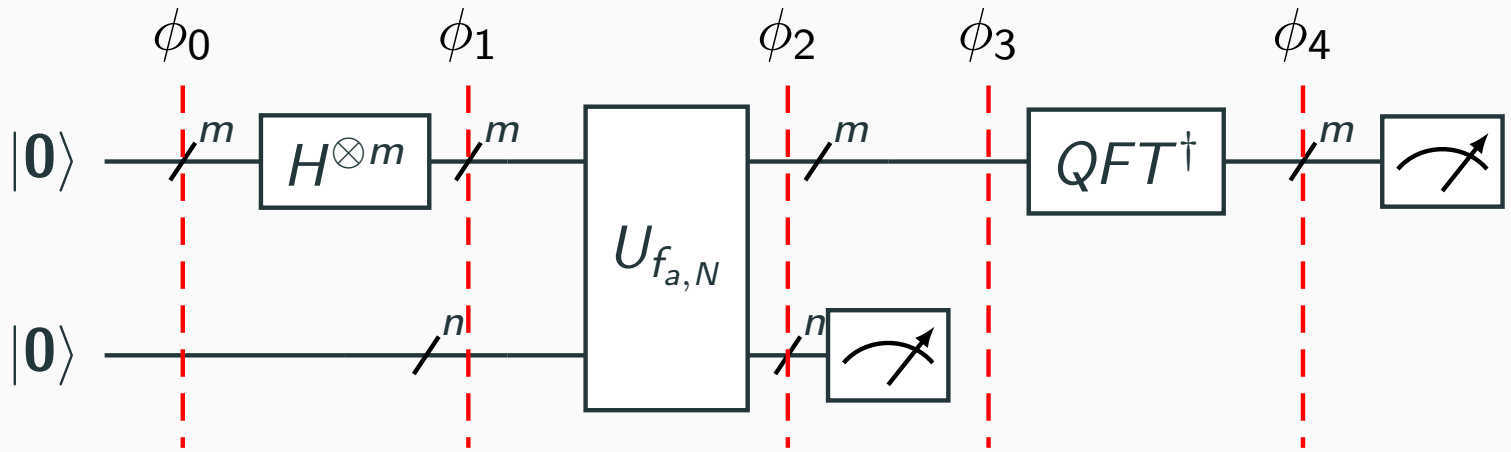
- $DFT^{\dagger}$ does two tasks:
  - It modifies r to $\frac{2^m}{r}$
  - It eliminates the offset

The Quantum Fourier Transform



- Uses a variant of a DFT called a quantum Fourier transform and denoted as QFT
- Its inverse is denoted by $QFT^\dagger$
- Performs the same operation but is constructed in a way that is more suitable for quantum computers

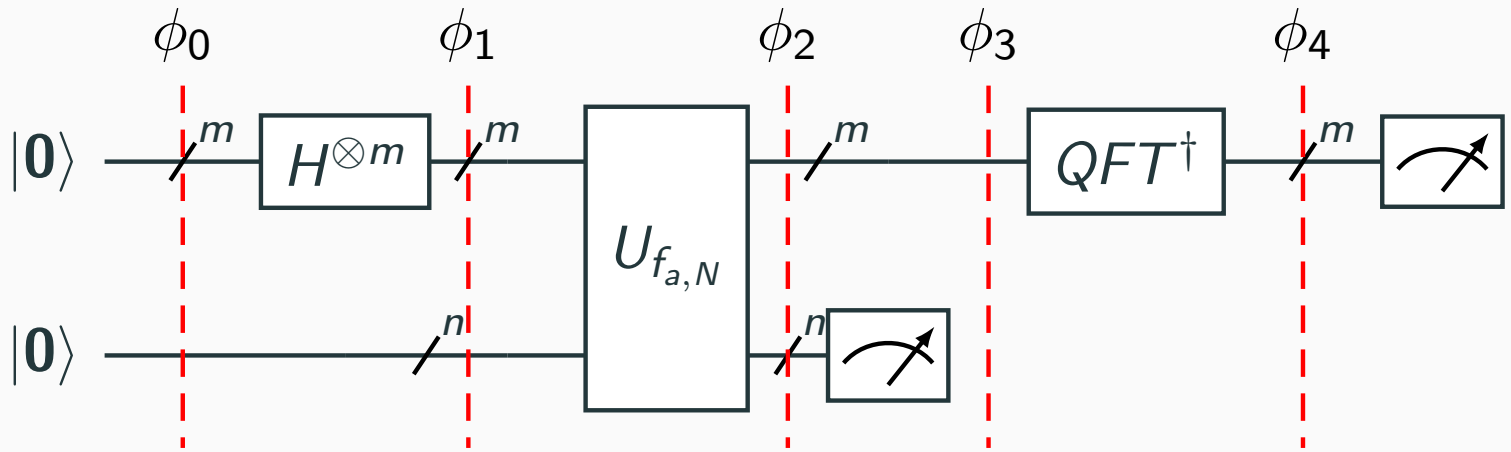
Circuit Slicing: ϕ_4 Final Step



- The final step of the circuit is to measure the top qubits.
- **Simplifying Assumption**: r evenly divides into 2^m
- When we measure the top qubit we will find it to be some multiple of $\frac{2^m}{r}$
- **Implication**: We will measure

$$x = \frac{\lambda 2^m}{r}$$

Circuit Slicing: ϕ_4 Final Step



- We know 2^m, and after measuring we will also know x
- We can divide the whole number x by 2^m and get

$$\frac{x}{2^m} = \frac{\lambda 2^m}{r 2^m} = \frac{\lambda}{r}$$

- One can then reduce this number to an irreducible fraction and take the denominator to be the long sought-after r.

If we do not make the simplifying assumption that r evenly divides into 2^m, then we might have to perform this process several times and analyze the results

Connecting Period Finding to Factoring

- How knowing period $r \implies$ factoring N ?
- **Requirement**: $2 \big| r$ i.e., r is **even**[1]
- If for chosen a, r is odd, then repeat[2] with another a.
- With this constraint, after successful period finding we have:

$$a^r \equiv 1 \bmod N : 2 \big| r$$

[1] Why this is needed will be apparent in a while.

[2] There is a theorem of number theory that tells us that for the **majority** of a, the period of $f_{a,N}$ will be an even number.

Connecting Period Finding to Factoring

- $a^r \equiv 1 \bmod N : 2 \big| r$

  $\implies a^r - 1 \equiv 0 \bmod N$

  $\implies N \big| (a^r - 1)$

  $\implies N \big| (\sqrt{a^r} + 1)(\sqrt{a^r} - 1)$ $\quad \leftarrow x^2 - y^2 = (x+y)(x-y)$

  $\implies N \big| (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1)$ $\quad \leftarrow$ Reason for requirement $2 \big| r$

Implication

Any factor of N is also a factor of $(a^{\frac{r}{2}} + 1)$ or $(a^{\frac{r}{2}} - 1)$ or both.

Connecting Period Finding to Factoring

- $a^r \equiv 1 \bmod N : 2 \big| r$

  $\implies a^r - 1 \equiv 0 \bmod N$

  $\implies N \big| (a^r - 1)$

  $\implies N \big| (\sqrt{a^r} + 1)(\sqrt{a^r} - 1)$ $\quad \leftarrow x^2 - y^2 = (x+y)(x-y)$

  $\implies N \big| (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1)$ $\quad \leftarrow$ Reason for requirement $2 \big| r$

Implication

Any factor of N is also a factor of $(a^{\frac{r}{2}} + 1)$ or $(a^{\frac{r}{2}} - 1)$ or both.

Connecting Period Finding to Factoring

- $a^r \equiv 1 \bmod N : 2 \big| r$

  $\implies a^r - 1 \equiv 0 \bmod N$

  $\implies N \big| (a^r - 1)$

  $\implies N \big| (\sqrt{a^r} + 1)(\sqrt{a^r} - 1)$ $\quad \leftarrow x^2 - y^2 = (x+y)(x-y)$

  $\implies N \big| (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1)$ $\quad \leftarrow$ Reason for requirement $2 \big| r$

Implication

Any factor of N is also a factor of $(a^{\frac{r}{2}} + 1)$ or $(a^{\frac{r}{2}} - 1)$ or both.

Connecting Period Finding to Factoring

- $a^r \equiv 1 \bmod N : 2 \Big| r$

  $\implies a^r - 1 \equiv 0 \bmod N$

  $\implies N \Big| (a^r - 1)$

  $\implies N \Big| (\sqrt{a^r} + 1)(\sqrt{a^r} - 1) \qquad \leftarrow x^2 - y^2 = (x+y)(x-y)$

  $\implies N \Big| (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1) \qquad \leftarrow$ Reason for requirement $2 \Big| r$

Implication

Any factor of N is also a factor of $(a^{\frac{r}{2}} + 1)$ or $(a^{\frac{r}{2}} - 1)$ or both.

Connecting Period Finding to Factoring

- $a^r \equiv 1 \bmod N : 2 \Big| r$

  $\implies a^r - 1 \equiv 0 \bmod N$

  $\implies N \Big| (a^r - 1)$

  $\implies N \Big| (\sqrt{a^r} + 1)(\sqrt{a^r} - 1)$ $\quad \leftarrow x^2 - y^2 = (x+y)(x-y)$

  $\implies N \Big| (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1)$ $\quad \leftarrow$ Reason for requirement $2 \Big| r$

Implication

Any factor of N is also a factor of $(a^{\frac{r}{2}} + 1)$ or $(a^{\frac{r}{2}} - 1)$ or both.

Connecting Period Finding to Factoring

$$N \Big| (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1) \tag{1}$$

- A factor for N can be found by looking at

$$\gcd(a^{\frac{r}{2}} + 1, N) \qquad \textit{and} \qquad \gcd(a^{\frac{r}{2}} - 1, N)$$

- Employ classical Euclidean GCD algorithm

Small caveat: Ensure $(a^{\frac{r}{2}} \neq -1 \bmod N)$

- $(a^{\frac{r}{2}} = -1 \bmod N) \implies (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1) = 0$
- So RHS of Equation (1) becomes 0
- Reveals no information about N
- If $(a^{\frac{r}{2}} = -1 \bmod N)$, repeat with another a

Connecting Period Finding to Factoring

$$N \Big| (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1) \tag{1}$$

- A factor for N can be found by looking at

$$\gcd(a^{\frac{r}{2}} + 1, N) \qquad \textit{and} \qquad \gcd(a^{\frac{r}{2}} - 1, N)$$

- Employ classical Euclidean GCD algorithm

Small caveat: Ensure $(a^{\frac{r}{2}} \neq -1 \bmod N)$

- $(a^{\frac{r}{2}} = -1 \bmod N) \implies (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1) = 0$
- So RHS of Equation (1) becomes 0
- Reveals no information about N
- If $(a^{\frac{r}{2}} = -1 \bmod N)$, repeat with another a

Connecting Period Finding to Factoring

$$N \Big| (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1) \qquad (1)$$

- A factor for N can be found by looking at

$$\gcd(a^{\frac{r}{2}} + 1, N) \qquad \textit{and} \qquad \gcd(a^{\frac{r}{2}} - 1, N)$$

- Employ classical Euclidean GCD algorithm

Small caveat: Ensure $(a^{\frac{r}{2}} \neq -1 \bmod N)$

- $(a^{\frac{r}{2}} = -1 \bmod N) \implies (a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1) = 0$
- So RHS of Equation (1) becomes 0
- Reveals no information about N
- If $(a^{\frac{r}{2}} = -1 \bmod N)$, repeat with another a

- Recall $f_{2,15}$

x	0	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
$f_{2,15}(x)$	1	2	4	8	1	2	4	8	1	2	4	8	1	$\cdots$

.

- Period of $f_{2,15}$ is 4, i.e., $2^4 \equiv 1 \bmod 15$
- Now,

$$15 \big| (2^2 + 1)(2^2 - 1)$$

- So we have

$$\gcd(5, 15) = 5 \quad \textit{and} \quad \gcd(3, 15) = 3$$

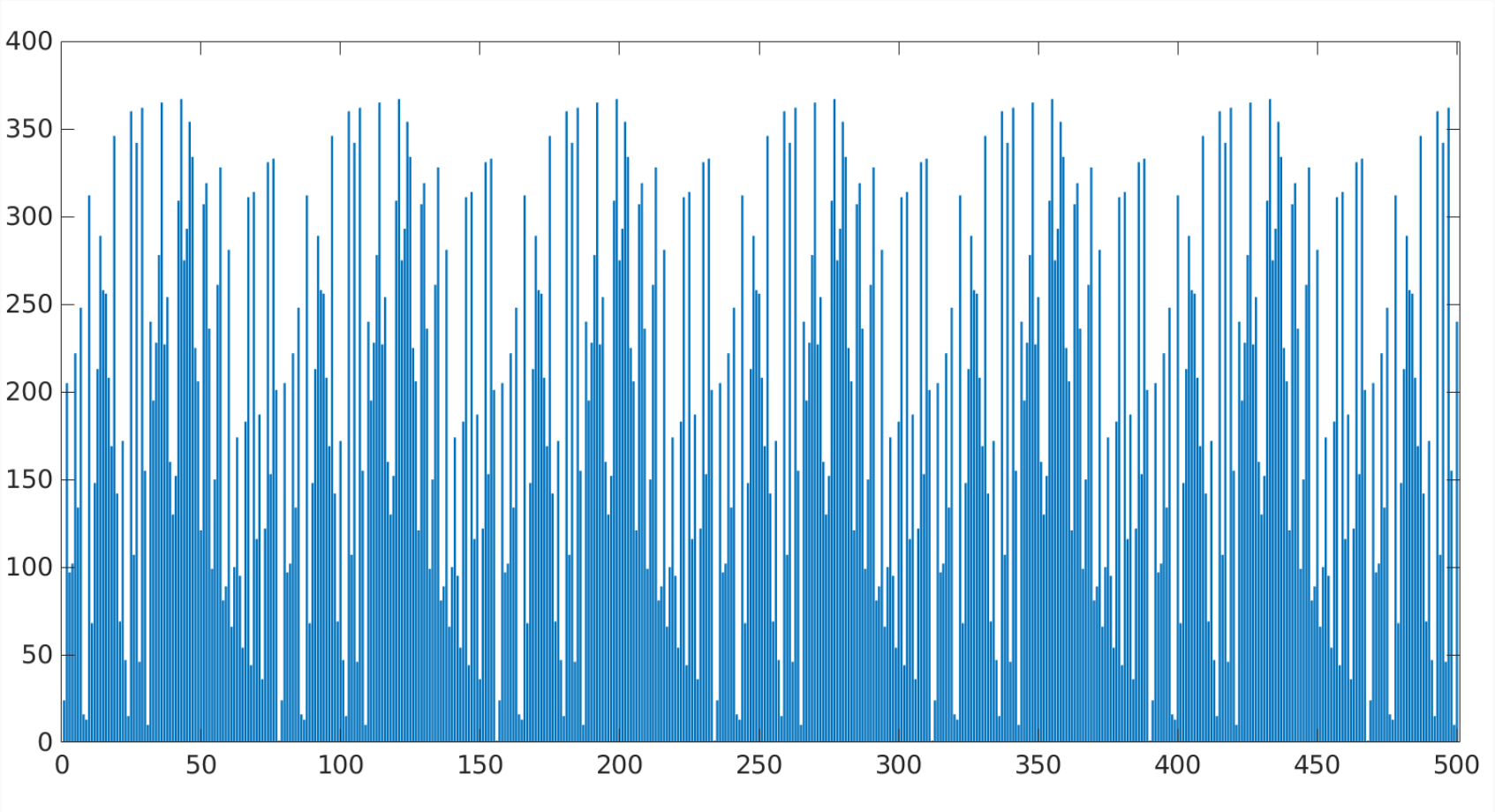
- Consider, $f_{6,371}$

x	0	1	2	3	4	5	6	7	$\cdots$	13	$\cdots$	25	26	27	28	$\cdots$
$f_{6,371}(x)$	1	6	36	216	183	356	281	202	$\cdots$	370	$\cdots$	62	1	6	36	$\cdots$

- Period of $f_{6,371}$ is 26, i.e., $6^{26} \equiv 1 \bmod 371$
- However, $6^{\frac{26}{2}} = 6^{13} \equiv -1 \bmod 371$
- So $a = 6$ has to be discarded
- Let us repeat for $a = 24$

Recall The Pattern

$f_{24,371}(x)$



$24^x \bmod 371$

- Now for $f_{24,371}$

x	0	1	2	3	4	5	6	7	$\cdots$	39	$\cdots$	77	78	79	80	$\cdots$
$f_{24,371}(x)$	1	24	205	97	102	222	134	248	$\cdots$	160	$\cdots$	201	1	24	205	$\cdots$

- Period of $f_{24,371}$ is 78, i.e., $24^{78} \equiv 1 \bmod 371$
- Now,

$$371 \Big| (24^{\frac{78}{2}} + 1)(24^{\frac{78}{2}} - 1)$$

- Also $24^{\frac{78}{2}} = 160 \neq -1 \bmod 371$
- So we can exploit:

$$371 \Big| (24^{39} + 1)(2^{39} - 1)$$

- Thus we have: $\gcd(161, 371) = 7$ *and* $\gcd(159, 371) = 53$
- And $371 = 7 \star 53$

Exercise

- Use the fact that the period of $f_{7,247}$ is 12 to determine the factors of 247.

Input: A positive integer N with $n = \lceil log_2 N \rceil$

Output: A factor p of N if it exists.

- **Step 1.** Use a polynomial time algorithm to determine if N is prime or a power of prime.
  - If it is a prime, declare that it is and exit.
  - If it is a power of a prime number, declare that it is and exit.
- **Step 2.** Randomly choose an integer a such that $1 < a < N$. Perform Euclid's algorithm to determine $\gcd(a, N)$.
  - If the GCD is not 1, then return it and exit.
- **Step 3.** Use quantum order finding circuit to find a period r
- **Step 4.** If r is odd or if $a^r \equiv -1 \bmod N$, then return to **Step 2** and choose another a.
- **Step 5.**Compute using Euclidean GCD algorithm: $\gcd(a^{\frac{r}{2}} + 1, N)$ and $\gcd(a^{\frac{r}{2}} - 1, N)$. Return at least one of the nontrivial solutions

Input: A positive integer N with $n = \lceil log_2 N \rceil$

Output: A factor p of N if it exists.

- **Step 1.** Use a polynomial time algorithm to determine if N is prime or a power of prime.
  - If it is a prime, declare that it is and exit.
  - If it is a power of a prime number, declare that it is and exit.
- **Step 2.** Randomly choose an integer a such that $1 < a < N$. Perform Euclid's algorithm to determine $\gcd(a, N)$.
  - If the GCD is not 1, then return it and exit.
- **Step 3.** Use quantum order finding circuit to find a period r
- **Step 4.** If r is odd or if $a^r \equiv -1 \bmod N$, then return to **Step 2** and choose another a.
- **Step 5.**Compute using Euclidean GCD algorithm: $\gcd(a^{\frac{r}{2}} + 1, N)$ and $\gcd(a^{\frac{r}{2}} - 1, N)$. Return at least one of the nontrivial solutions

Input: A positive integer N with $n = \lceil log_2 N \rceil$

Output: A factor p of N if it exists.

- **Step 1.** Use a polynomial time algorithm to determine if N is prime or a power of prime.
  - If it is a prime, declare that it is and exit.
  - If it is a power of a prime number, declare that it is and exit.
- **Step 2.** Randomly choose an integer a such that $1 < a < N$. Perform Euclid's algorithm to determine $\gcd(a, N)$.
  - If the GCD is not 1, then return it and exit.
- **Step 3.** Use quantum order finding circuit to find a period r
- **Step 4.** If r is odd or if $a^r \equiv -1 \bmod N$, then return to **Step 2** and choose another a.
- **Step 5.**Compute using Euclidean GCD algorithm: $\gcd(a^{\frac{r}{2}} + 1, N)$ and $\gcd(a^{\frac{r}{2}} - 1, N)$. Return at least one of the nontrivial solutions

Input: A positive integer N with $n = \lceil log_2 N \rceil$

Output: A factor p of N if it exists.

- **Step 1.** Use a polynomial time algorithm to determine if N is prime or a power of prime.
  - If it is a prime, declare that it is and exit.
  - If it is a power of a prime number, declare that it is and exit.
- **Step 2.** Randomly choose an integer a such that $1 < a < N$. Perform Euclid's algorithm to determine $\gcd(a, N)$.
  - If the GCD is not 1, then return it and exit.
- **Step 3.** Use quantum order finding circuit to find a period r
- **Step 4.** If r is odd or if $a^r \equiv -1 \bmod N$, then return to **Step 2** and choose another a.
- **Step 5.**Compute using Euclidean GCD algorithm: $\gcd(a^{\frac{r}{2}} + 1, N)$ and $\gcd(a^{\frac{r}{2}} - 1, N)$. Return at least one of the nontrivial solutions

Input: A positive integer N with $n = \lceil log_2 N \rceil$

Output: A factor p of N if it exists.

- **Step 1.** Use a polynomial time algorithm to determine if N is prime or a power of prime.
  - If it is a prime, declare that it is and exit.
  - If it is a power of a prime number, declare that it is and exit.
- **Step 2.** Randomly choose an integer a such that $1 < a < N$. Perform Euclid's algorithm to determine $\gcd(a, N)$.
  - If the GCD is not 1, then return it and exit.
- **Step 3.** Use quantum order finding circuit to find a period r
- **Step 4.** If r is odd or if $a^r \equiv -1 \bmod N$, then return to **Step 2** and choose another a.
- **Step 5.**Compute using Euclidean GCD algorithm: $\gcd(a^{\frac{r}{2}} + 1, N)$ and $\gcd(a^{\frac{r}{2}} - 1, N)$. Return at least one of the nontrivial solutions

Input: A positive integer N with $n = \lceil log_2 N \rceil$

Output: A factor p of N if it exists.

- **Step 1.** Use a polynomial time algorithm to determine if N is prime or a power of prime.
  - If it is a prime, declare that it is and exit.
  - If it is a power of a prime number, declare that it is and exit.
- **Step 2.** Randomly choose an integer a such that $1 < a < N$. Perform Euclid's algorithm to determine $\gcd(a, N)$.
  - If the GCD is not 1, then return it and exit.
- **Step 3.** Use quantum order finding circuit to find a period r
- **Step 4.** If r is odd or if $a^r \equiv -1 \bmod N$, then return to **Step 2** and choose another a.
- **Step 5.**Compute using Euclidean GCD algorithm: $\gcd(a^{\frac{r}{2}} + 1, N)$ and $\gcd(a^{\frac{r}{2}} - 1, N)$. Return at least one of the nontrivial solutions

Input: A positive integer N with $n = \lceil log_2 N \rceil$

Output: A factor p of N if it exists.

- **Step 1.** Use a polynomial time algorithm to determine if N is prime or a power of prime.
  - If it is a prime, declare that it is and exit.
  - If it is a power of a prime number, declare that it is and exit.
- **Step 2.** Randomly choose an integer a such that $1 < a < N$. Perform Euclid's algorithm to determine $\gcd(a, N)$.
  - If the GCD is not 1, then return it and exit.
- **Step 3.** Use quantum order finding circuit to find a period r
- **Step 4.** If r is odd or if $a^r \equiv -1 \bmod N$, then return to **Step 2** and choose another a.
- **Step 5.**Compute using Euclidean GCD algorithm: $\gcd(a^{\frac{r}{2}} + 1, N)$ and $\gcd(a^{\frac{r}{2}} - 1, N)$. Return at least one of the nontrivial solutions

Input: A positive integer N with $n = \lceil log_2 N \rceil$

Output: A factor p of N if it exists.

- **Step 1.** Use a polynomial time algorithm to determine if N is prime or a power of prime.
  - If it is a prime, declare that it is and exit.
  - If it is a power of a prime number, declare that it is and exit.
- **Step 2.** Randomly choose an integer a such that $1 < a < N$. Perform Euclid's algorithm to determine $\gcd(a, N)$.
  - If the GCD is not 1, then return it and exit.
- **Step 3.** Use quantum order finding circuit to find a period r
- **Step 4.** If r is odd or if $a^r \equiv -1 \bmod N$, then return to **Step 2** and choose another a.
- **Step 5.** Compute using Euclidean GCD algorithm: $\gcd(a^{\frac{r}{2}} + 1, N)$ and $\gcd(a^{\frac{r}{2}} - 1, N)$. Return at least one of the nontrivial solutions

Input: A positive integer N with $n = \lceil log_2 N \rceil$

Output: A factor p of N if it exists.

- **Step 1.** Use a polynomial time algorithm to determine if N is prime or a power of prime.
  - If it is a prime, declare that it is and exit.
  - If it is a power of a prime number, declare that it is and exit.
- **Step 2.** Randomly choose an integer a such that $1 < a < N$. Perform Euclid's algorithm to determine $\gcd(a, N)$.
  - If the GCD is not 1, then return it and exit.
- **Step 3.** Use quantum order finding circuit to find a period r
- **Step 4.** If r is odd or if $a^r \equiv -1 \bmod N$, then return to **Step 2** and choose another a.
- **Step 5.**Compute using Euclidean GCD algorithm: $\gcd(a^{\frac{r}{2}} + 1, N)$ and $\gcd(a^{\frac{r}{2}} - 1, N)$. Return at least one of the nontrivial solutions

- Depends of implementation details of $U_{f_{a,N}}$ and $QFT^{\dagger}$
- Percentage of time we will get an odd period of a choice of a
- Skipping details running time of Shor's Algorihtm is

$$O(n^2 \log n \log\log n), \text{ where } n = \lceil log_2 N \rceil$$

- Best known classical algorithm runs in

$$O(e^{cn^{\frac{1}{3}} \log^{\frac{2}{3}} n}), \text{ where } c \text{ is some constant}$$

- This is exponential in n

Intuition

Break $U_{f_{a,N}}$ into sub-operations

- Binary expansion of x

$$x = x_{n-1}x_{n-2}\cdots x_2x_1x_0$$

- x can be represented as

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_2 2^2 + x_1 2 + x_0$$

- Rewriting $f_{a,N}$ using the above representation

$$\begin{aligned} f_{a,N}(x) &= a^x \bmod N \\ &= a^{x_{n-1}2^{n-1}+x_{n-2}2^{n-2}+\cdots+x_2 2^2+x_1 2+x_0} \bmod N \\ &= a^{x_{n-1}2^{n-1}} \times a^{x_{n-2}2^{n-2}} \times \cdots \times a^{x_2 2^2} \times a^{x_1 2} \times a^{x_0} \bmod N \end{aligned}$$

$$f_{a,N}(x) = a^{x_{n-1}2^{n-1}} \times a^{x_{n-2}2^{n-2}} \times \cdots \times a^{x_2 2^2} \times a^{x_1 2} \times a^{x_0} \bmod N \quad (2)$$

Inductive definition of $f_{a,N}(x)$ from Equation (2)

- Let us define $y_0, y_1, \ldots, y_{n-2}, y_{n-1}$ where $y_{n-1} = f_{a,N}(x)$
  - The base case is: $y_0 = a^{x_0}$
  - Recall $a^s \bmod N = ((a^{s-1} \bmod N) \times a) \bmod N$.
  - Using this, we have $(y_j = y_{j-1} \times a^{x_j 2^j} \bmod N)$

- Note that if $x_j = 0$, then $y_j = y_{j-1}$
- In other words, whether or not we should multiply y_{j-1} by $(a^{2^j} \bmod N)$ is dependent on whether or not $x_j = 1$.

```
expModNaive(x, e, n) {
    y = x
    for i = 1 to e - 1 {
        y = y * x  mod n
    }
    return y
}
```

Naive Exponentiation

The naive way to compute (x^e mod n) takes $e - 1$ multiplications

Fast Exponentiation (Square and Multiply)

- Exponent e consists of bits $e_{m-1}e_{m-2}\cdots e_1e_0$,
- e_0 is the LSB

```
expMod(x, e, n) {
    y = x
    for i = m - 1 to 0 {
        y = y * y  mod n
        if e_i == 1 then
            y = y * x  mod n
    }
    return y
}
```

- Runs in time $O(m)$
- The naive algorithm runs in time $O(2^m)$

Fast Exponentiation (An Example)

$$x^{26} = x^{11010_2} = x^{(h_4 h_3 h_2 h_1 h_0)_2}.$$

The algorithm scans the exponent bits, starting on the left with h_4 and ending with the rightmost bit h_0.

Step		
#0	$x = x^{\mathbf{1}_2}$	inital setting, bit processed: $h_4 = 1$
#1a	$(x^1)^2 = x^2 = x^{\mathbf{10}_2}$	SQ, bit processed: h_3
#1b	$x^2 \cdot x = x^3 = x^{10_2} x^{1_2} = x^{\mathbf{11}_2}$	MUL, since $h_3 = 1$
#2a	$(x^3)^2 = x^6 = (x^{11_2})^2 = x^{\mathbf{110_2}}$	SQ, bit processed: h_2
#2b		no MUL, since $h_2 = 0$
#3a	$(x^6)^2 = x^{12} = (x^{110_2})^2 = x^{\mathbf{1100_2}}$	SQ, bit processed: h_1
#3b	$x^{12} \cdot x = x^{13} = x^{1100_2} x^{1_2} = x^{\mathbf{1101}_2}$	MUL, since $h_1 = 1$
#4a	$(x^{13})^2 = x^{26} = (x^{1101_2})^2 = x^{\mathbf{11010_2}}$	SQ, bit processed: h_0
#4b		no MUL, since $h_0 = 0$

Known

If $a \in \mathbb{Z}_N^*$ (i.e., a and N are co-prime), the operation of multiplying a number times $(a^{2^j} \bmod N)$ is reversible and **unitary**.

- So for each j there is an unitary operator $U_{a^{2^j} \bmod N}$
- Simplified notation $\rightarrow U_{a^{2^j}}$
- Note we want to do this based the value of $x_j \implies$ a **conditional** quantum operator
- So we need controlled$-U_{a^{2^j}}$ or ${}^{C}U_{a^{2^j}}$ gates

- Quantum circuit implementing $f_{a,N}$ in a polynomial number of gates