Shor's Algorithm in Qiskit

From controlled modular exponentiation to IQFT cancellation and measurement

Instruction reconstructed from the handwritten summary notebook

8 August 2026

Problem and goal. Given an odd composite integer N, Shor's algorithm aims to find a non-trivial factor of N. The quantum circuit does not search for that factor directly; it finds the order r of a modulo N, after which classical gcd calculations recover the factors.

$$\boxed{N \text{ (composite)} \longrightarrow \text{order finding } r \longrightarrow \gcd(a^{r/2} \pm 1, N) \longrightarrow \text{non-trivial factors}}$$

Key idea. The notebook's central quantum picture is

$$\boxed{\text{modular exponentiation} \longrightarrow \text{periodic structure in } x \xrightarrow{QFT^{-1}} \text{peaks in } y \longrightarrow \text{measurement} \longrightarrow r}$$

For the worked example $N = 15$, $a = 2$, and an 8-qubit counting register, the period is $r = 4$, so the IQFT produces counting-register peaks at

$$y = 0, 64, 128, 192.$$

Contents

1 What Shor's Algorithm Is

1.1 The factoring problem

The input is an **odd composite integer** N. The goal is to find a **non-trivial factor** of N; that is, a divisor d satisfying

$$\boxed{1 < d < N, \qquad d \mid N.}$$

In the simplest textbook case,

$$N = pq,$$

where p and q are unknown distinct odd primes. For example, when $N = 15$, the desired final result is

$$15 = 3 \times 5.$$

A crucial point is that the quantum circuit does *not* search directly for p and q. Shor's algorithm first converts factoring into an **order-finding problem**:

$$\boxed{\text{factor } N \longrightarrow \text{ find an order } r \longrightarrow \text{ use classical gcds to obtain factors.}}$$

1.2 Choose a base a

Choose an integer

$$1 < a < N.$$

Before using the quantum circuit, compute

$$\gcd(a, N).$$

If

$$1 < \gcd(a, N) < N,$$

then a factor has already been found and the quantum subroutine is unnecessary. Otherwise we continue with

$$\boxed{\gcd(a, N) = 1.}$$

In the Qiskit example used throughout this guide,

$$\boxed{N = 15, \qquad a = 2.}$$

1.3 The quantum goal: find the order r

Consider the modular-exponentiation function

$$f(x) = a^x \bmod N.$$

Because a is coprime to N, the powers of a modulo N repeat. The **order** r of a modulo N is the smallest positive integer for which

$$\boxed{r = \min\{t > 0 : a^t \equiv 1 \pmod{N}\}.}$$

Thus the specific goal of the quantum part of Shor's algorithm is

$$\boxed{\text{discover the period } r \text{ of } f(x) = a^x \bmod N.}$$

For the notebook example,

$$2^0 \equiv 1, \quad 2^1 \equiv 2, \quad 2^2 \equiv 4, \quad 2^3 \equiv 8, \quad 2^4 \equiv 1 \pmod{15}.$$

Hence

$$2^x \bmod 15 = 1, 2, 4, 8, 1, 2, 4, 8, \ldots$$

and the period is

$$\boxed{r = 4}.$$

1.4 Why knowing r can reveal factors of N

By the definition of the order,

$$a^r \equiv 1 \pmod{N}.$$

This does *not* mean $a^r - 1 = 1$. It means that $a^r - 1$ is a multiple of N:

$$\boxed{a^r - 1 = kN}$$

for some integer k.

If r is even, then

$$a^r - 1 = \left(a^{r/2}\right)^2 - 1 = \left(a^{r/2} - 1\right)\left(a^{r/2} + 1\right).$$

Therefore

$$\boxed{N \mid \left(a^{r/2} - 1\right)\left(a^{r/2} + 1\right).}$$

This is why the classical post-processing computes

$$\boxed{\gcd\left(a^{r/2} - 1, N\right), \qquad \gcd\left(a^{r/2} + 1, N\right).}$$

The integer k may be very large, but this causes no problem: a gcd with N can only return a divisor of N, so

$$\gcd(A, N) \leq N$$

for any integer A.

Define

$$x = a^{r/2}.$$

Then

$$x^2 \equiv 1 \pmod{N}.$$

For useful factor recovery, x must be a **non-trivial square root of** 1 **modulo** N:

$$\boxed{x \not\equiv \pm 1 \pmod{N}.}$$

The case $x \equiv 1 \pmod{N}$ cannot occur when r is truly the order, because then $r/2 < r$ would already satisfy $a^{r/2} \equiv 1 \pmod{N}$, contradicting the minimality of r.

1.5 The risk: $a^{r/2} + 1$ can be a multiple of N

An even order r is **necessary, but not sufficient** for successful factor recovery. Define

$$x = a^{r/2}.$$

Because r is the order,

$$x^2 = a^r \equiv 1 \pmod{N}.$$

The dangerous case is

$$\boxed{x \equiv -1 \pmod{N}.}$$

This congruence means exactly that $x + 1$ is a multiple of N: there exists an integer m such that

$$\boxed{x + 1 = mN.}$$

Equivalently,

$$\boxed{a^{r/2} + 1 = mN.}$$

Therefore the second gcd becomes

$$\boxed{\gcd(a^{r/2} + 1, N) = N,}$$

which is useless because N is already known. On the other side,

$$x - 1 \equiv -2 \pmod{N}.$$

Since the factoring problem here assumes odd N, $\gcd(2, N) = 1$, and hence

$$\boxed{\gcd(a^{r/2} - 1, N) = 1.}$$

Thus this bad case returns only the trivial pair

$$\boxed{1 \quad \text{and} \quad N.}$$

This is why Shor's post-processing does not merely require r to be even; it also requires

$$\boxed{a^{r/2} \not\equiv -1 \pmod{N}.}$$

A concrete example is

$$N = 15, \qquad a = 14.$$

Since

$$14^2 \equiv 1 \pmod{15},$$

the order is $r = 2$, which is even. However,

$$a^{r/2} = 14 \equiv -1 \pmod{15},$$

so

$$a^{r/2} + 1 = 15 = 1 \cdot N.$$

Consequently,

$$\gcd(14 - 1, 15) = \gcd(13, 15) = 1,$$

while

$$\gcd(14 + 1, 15) = \gcd(15, 15) = 15.$$

No non-trivial factor is obtained even though r is even.

The case $x \equiv 1 \pmod{N}$ cannot occur when r is truly the order, because then $r/2 < r$ would already satisfy $a^{r/2} \equiv 1 \pmod{N}$, contradicting the minimality of r. Therefore the full success conditions are

$$\boxed{r \text{ is even} \qquad \text{and} \qquad a^{r/2} \not\equiv -1 \pmod{N}.}$$

When these hold, $x = a^{r/2}$ is a non-trivial square root of 1 modulo N.

For the notebook example, $r = 4$, so

$$x = a^{r/2} = 2^2 = 4.$$

Since

$$4 \not\equiv \pm 1 \pmod{15},$$

the gcds are non-trivial:

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

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

Hence

$$\boxed{15 = 3 \times 5.}$$

1.6 What the quantum circuit contributes

Once a valid order r is known, the final gcd calculations are ordinary classical arithmetic. The hard part is obtaining r efficiently.

The circuit therefore does not read the work register term by term to discover

$$1, 2, 4, 8, 1, 2, 4, 8, \ldots$$

Instead, controlled modular exponentiation encodes the periodic function coherently across the quantum state, and the inverse QFT converts that periodicity into constructive and destructive interference in the counting register. Measurement then provides a value y from which classical continued fractions recover a candidate r.

The purpose of the quantum circuit can therefore be summarized as

$$\boxed{\begin{array}{c}\text{superposition over } x \to \text{controlled } a^x \bmod N \to \text{periodic quantum structure} \\ \xrightarrow{QFT^{-1}} \text{Fourier peaks in } y \to r\end{array}}$$

The rest of this document explains how each Qiskit instruction realizes one part of this chain.

2 How Would You Solve It with a Quantum Circuit?

2.1 Register layout and initialization

With $n_{\text{count}} = 8$, the circuit allocates

$$8 + 4 = 12$$

qubits in total:

$$\underbrace{q_0, \ldots, q_7}_{\text{counting register (CR)}} \qquad \underbrace{q_8, \ldots, q_{11}}_{\text{work register (WR)}} .$$

There are only 8 classical bits because only the counting register is measured.

The Qiskit allocation is therefore:

```
n_count = 8
qc = QuantumCircuit(n_count + 4, n_count)
```

The first argument creates 12 qubits in total; the second creates 8 classical bits for the final counting-register measurement.

Initially,

$$|0\rangle_{CR}^{\otimes 8} |0000\rangle_{WR} .$$

The work register is initialized to $|1\rangle$, giving

$$\boxed{|0\rangle_{CR}^{\otimes 8} |1\rangle_{WR}}.$$

In Qiskit this is one line:

```
qc.x(n_count)   # q8: |0000> -> |0001> = |1>
```

The value 1 is used because it is the multiplicative identity. Since the work register starts at $|1\rangle$, repeated applications of U_a directly generate $|a^x \bmod N\rangle$.

2.2 Step 1: Hadamards create every exponent x

A Hadamard is applied to each counting qubit:

```
for q in range(n_count):
    qc.h(q)
```

For one qubit,

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}.$$

For 8 counting qubits,

$$H^{\otimes 8}|0\rangle^{\otimes 8} = \frac{1}{\sqrt{256}} \sum_{x=0}^{255} |x\rangle = \frac{1}{16} \sum_{x=0}^{255} |x\rangle .$$

Thus the whole state becomes

$$\boxed{\frac{1}{16} \sum_{x=0}^{255} |x\rangle_{CR} |1\rangle_{WR}}.$$

The counting register now contains a uniform superposition over all $x \in \{0, \ldots, 255\}$.

2.3 Step 2: Controlled modular exponentiation

The controlled modular arithmetic is added by the following Qiskit loop:

```
for q in range(n_count):
    qc.append(
        c_amod15(a, 2**q),
        [q] + [n_count + i for i in range(4)]
    )
```

Here `q` is the counting/control qubit and the remaining four indices are the work-register wires. Mathematically, the entire row of controlled gates implements

$$|x\rangle |1\rangle \longmapsto |x\rangle |a^x \bmod N\rangle .$$

For $N = 15$ and $a = 2$,

$$\frac{1}{16} \sum_{x=0}^{255} |x\rangle |1\rangle \longmapsto \boxed{\frac{1}{16} \sum_{x=0}^{255} |x\rangle |2^x \bmod 15\rangle}.$$

Because

$$2^x \bmod 15 = 1, 2, 4, 8, 1, 2, 4, 8, \ldots,$$

the work-register value repeats with period 4. The counting and work registers are now correlated (entangled).

2.4 Reading the actual Qiskit circuit diagram

Figure 1 is the circuit produced by Qiskit's circuit drawer for the same $N = 15$, $a = 2$, $n_{\text{count}} = 8$ implementation used in this guide.

Read the circuit from left to right:

1. **Hadamards on $q_0, \ldots, q_7$.** The eight red H boxes create

$$\frac{1}{16} \sum_{x=0}^{255} |x\rangle_{CR} .$$

   Thus all possible exponents x are represented coherently.

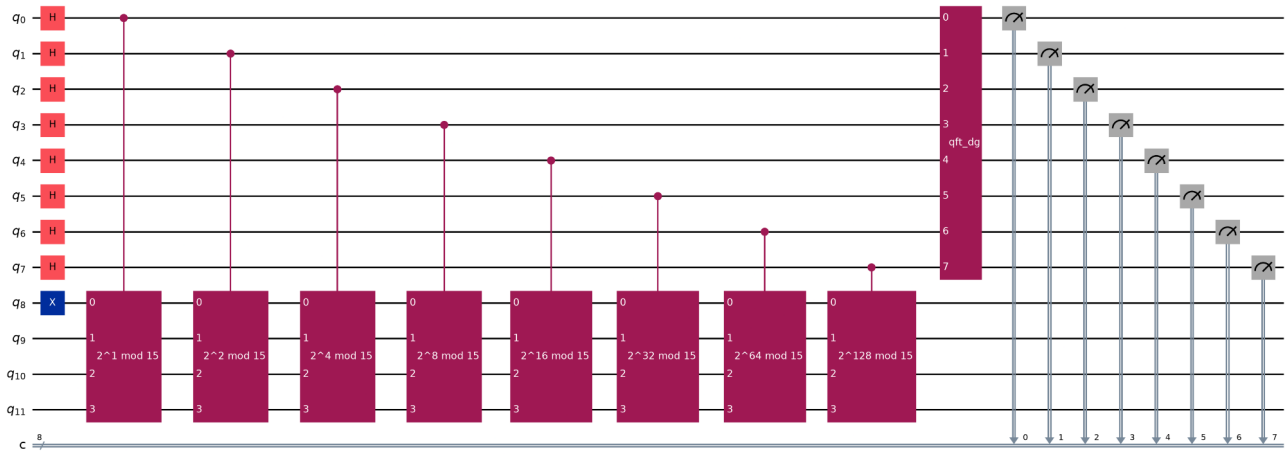

Figure 1: The complete Shor order-finding circuit for $N = 15$, $a = 2$. Qubits q_0–q_7 are the counting register; q_8–q_{11} are the four-qubit work register.

2. **Initialize the work register.** The blue X on q_8 changes the four work qubits from $|0000\rangle$ to the integer state $|1\rangle$. In Qiskit's little-endian convention, q_8 is the least-significant work qubit.

3. **Controlled modular-power gates.** Each magenta dot on a counting qubit is the control of the large four-wire gate directly below it. The first gate is controlled by q_0, the second by q_1, and so on. These are
$$CU_2^1,\ CU_2^2,\ CU_2^4,\ CU_2^8, \ldots, CU_2^{128}.$$
The labels 2^1 mod 15, 2^2 mod 15, 2^4 mod 15, etc. indicate the corresponding modular multipliers.

4. **Inverse QFT.** The block labelled `qft_dg` acts only on $q_0, \ldots, q_7$. The suffix "dg" means dagger, so this is QFT^{-1}.

5. **Measurement.** Only the counting qubits are measured into the eight classical bits. There are no measurement symbols on $q_8, \ldots, q_{11}$.

The vertical magenta line in Figure 1 is important: it does *not* mean that the counting qubit is copied into the work register. It means that the work-register unitary is applied conditionally on the control qubit.

2.5 What the controlled U gate means

Define the modular-multiplication unitary

$$\boxed{U_a\,|y\rangle = |ay \bmod N\rangle}.$$

In the Qiskit implementation, `c_amod15` first builds the four-qubit work-register transformation `U`, converts it to a gate, and finally turns it into a controlled gate:

```
def c_amod15(a, power):
    U = QuantumCircuit(4)
    # reversible modular multiplication is built inside U
    ...
    U = U.to_gate()
    U.name = f"{a}^{power} mod 15"
    return U.control()
```

The call `U.control()` is exactly what creates the magenta control dot connected to the four-wire U box in Figure 1. Because $\gcd(a, N) = 1$, multiplication by a modulo N is a permutation of the modular

residues and can therefore be implemented reversibly as a unitary operation. For the present example,

$$\boxed{U_2 \left|y\right\rangle = \left|2y \bmod 15\right\rangle}.$$

On the orbit that begins at $|1\rangle$,

$$\left|1\right\rangle \xrightarrow{U_2} \left|2\right\rangle \xrightarrow{U_2} \left|4\right\rangle \xrightarrow{U_2} \left|8\right\rangle \xrightarrow{U_2} \left|1\right\rangle .$$

Thus

$$\boxed{U_2^4 \left|1\right\rangle = \left|1\right\rangle},$$

which is the circuit-level manifestation of the order $r = 4$.

A **controlled** $U_a^{2^j}$ gate acts according to

$$\boxed{\left|c\right\rangle \left|y\right\rangle \longmapsto \left|c\right\rangle \left(U_a^{2^j}\right)^c \left|y\right\rangle} \qquad c \in \{0, 1\}.$$

Equivalently,

$$\left|0\right\rangle \left|y\right\rangle \longmapsto \left|0\right\rangle \left|y\right\rangle ,$$

while

$$\left|1\right\rangle \left|y\right\rangle \longmapsto \left|1\right\rangle \left|a^{2^j} y \bmod N\right\rangle .$$

For $a = 2$, this becomes

$$\left|1\right\rangle \left|y\right\rangle \longmapsto \left|1\right\rangle \left|2^{2^j} y \bmod 15\right\rangle .$$

The counting qubit is not normally in a definite classical state after the Hadamard. For example,

$$\frac{\left|0\right\rangle + \left|1\right\rangle}{\sqrt{2}} \left|y\right\rangle \xrightarrow{CU_2^{2^j}} \frac{1}{\sqrt{2}} \left(\left|0\right\rangle \left|y\right\rangle + \left|1\right\rangle \left|2^{2^j} y \bmod 15\right\rangle\right).$$

If the two work states are different, this operation entangles the control qubit with the work register. This is exactly where the counting register becomes correlated with modular exponentiation.

2.5.1 Why one controlled gate expands into many elementary gates

Figure 2 shows Qiskit's decomposition of the first controlled modular-multiplication gate, CU_2^1, into lower-level reversible gates.

In this decomposition, a vertical column with two filled dots and one $\oplus$ is a Toffoli (CCX) operation. Groups of these operations implement *controlled swaps* of work-register bits. For $a = 2$, the source code that builds one application of U_2 is

```
if a in [2, 13]:
    U.swap(2, 3)
    U.swap(1, 2)
    U.swap(0, 1)
```

Therefore the original `c_amod15(2,1)` circuit uses the work-register swaps

$$\mathrm{SWAP}(2, 3), \qquad \mathrm{SWAP}(1, 2), \qquad \mathrm{SWAP}(0, 1),$$

and making the whole U_2 gate controlled turns the corresponding work-bit permutation into a conditional permutation. The top wire in Figure 2 therefore remains a control throughout the decomposition: if it is 0, the net operation is the identity; if it is 1, the swaps collectively implement multiplication by 2 modulo 15.

For the states relevant to Shor's example, the binary effect can be seen directly:

$$0001\ (1) \longrightarrow 0010\ (2) \longrightarrow 0100\ (4) \longrightarrow 1000\ (8) \longrightarrow 0001\ (1).$$

The swaps are therefore not arbitrary wiring tricks; they are a reversible implementation of the modular permutation required by U_2.

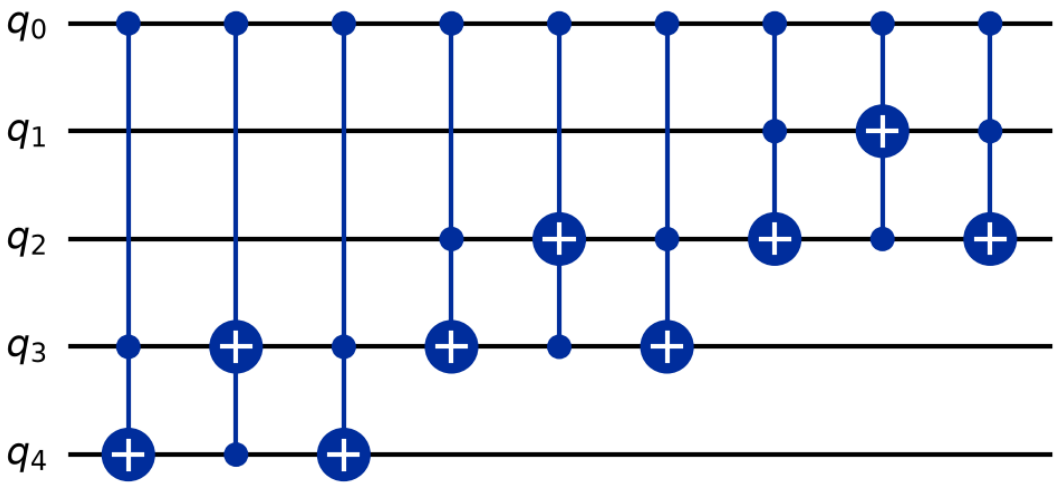

Figure 2: Decomposition of the controlled U_2 gate. Here local wire q_0 is the counting/control qubit and local wires q_1–q_4 are the four work qubits. A filled dot is a control and $\oplus$ is an X target.

2.5.2 How all eight control bits build the exponent x

Let the measured-basis value of the counting register be written in binary as

$$x = \sum_{j=0}^{7} x_j 2^j, \qquad x_j \in \{0, 1\}.$$

The jth counting qubit controls $U_2^{2^j}$. This is visible directly in the Qiskit exponent `2**q`:

```
for q in range(n_count):
    qc.append(c_amod15(2, 2**q),
              [q] + [n_count + i for i in range(4)])
```

Thus q_0 controls U_2^1, q_1 controls U_2^2, q_2 controls U_2^4, and so on. Hence, on a computational-basis component $|x\rangle$, all controlled gates together apply

$$\prod_{j=0}^{7} \left(U_2^{2^j}\right)^{x_j} = U_2^{\sum_j x_j 2^j} = \boxed{U_2^x}.$$

Starting with the work register in $|1\rangle$,

$$U_2^x |1\rangle = |2^x \bmod 15\rangle .$$

Therefore the circuit as a whole realizes

$$\boxed{|x\rangle_{CR} |1\rangle_{WR} \longmapsto |x\rangle_{CR} |2^x \bmod 15\rangle_{WR}}.$$

This equation is the precise connection between the row of controlled boxes in Figure 1 and the periodic joint state used in the IQFT calculation later in this document.

For example, if

$$x = 3 = 00000011_2,$$

then $x_0 = x_1 = 1$ and the other x_j are zero. Only U_2^1 and U_2^2 are active, so

$$U_2^1 U_2^2 = U_2^3, \qquad U_2^3 |1\rangle = |2^3 \bmod 15\rangle = |8\rangle .$$

If

$$x = 5 = 00000101_2,$$

then the active powers are U_2^1 and U_2^4, giving

$$U_2^5 |1\rangle = |2^5 \bmod 15\rangle = |2\rangle .$$

The periodicity of these results is what the inverse QFT converts into the peaks $y = 0, 64, 128, 192$.

2.6 The notebook's $WR = 1$ calculation

Focus on the part of the joint state whose work register is $|1\rangle$. This occurs whenever

$$x = 0, 4, 8, 12, \ldots, 252.$$

Write

$$x = 4k, \qquad k = 0, 1, \ldots, 63.$$

Conditioned on $WR = 1$, the normalized counting-register state is

$$\boxed{\frac{1}{8}\sum_{k=0}^{63} |4k\rangle}.$$

The essential information is the spacing of the surviving x values:

$$\boxed{4}.$$

This is exactly the order r.

The other work-register values give shifted versions with the same spacing:

$$WR = 2 : \; x = 1, 5, 9, \ldots,$$

$$WR = 4 : \; x = 2, 6, 10, \ldots,$$

$$WR = 8 : \; x = 3, 7, 11, \ldots.$$

Therefore every work-register sector encodes the same period $r = 4$.

2.7 Step 3: Apply the inverse QFT to the counting register

Only the counting register receives the inverse QFT. The corresponding Qiskit instruction is

```
qc.append(QFTGate(n_count).inverse(), range(n_count))
```

Because `range(n_count)` is only $q_0, \ldots, q_7$, the work register is not passed to the IQFT gate. Mathematically, the operation is

$$\boxed{QFT_{CR}^{-1} \otimes I_{WR}}.$$

The work register is not transformed by the IQFT.

For an 8-qubit counting register,

$$Q = 2^8 = 256.$$

The inverse QFT acts on a computational basis state as

$$\boxed{QFT^{-1} |x\rangle = \frac{1}{\sqrt{256}} \sum_{y=0}^{255} e^{-2\pi i x y/256} |y\rangle = \frac{1}{16} \sum_{y=0}^{255} e^{-2\pi i x y/256} |y\rangle}.$$

Here x is the input basis-state label and y is the output basis-state label.

2.8 Apply the IQFT to the periodic state

Starting from

$$\frac{1}{8}\sum_{k=0}^{63}|4k\rangle\,,$$

apply QFT^{-1}:

$$\begin{aligned} QFT^{-1}\left(\frac{1}{8}\sum_{k=0}^{63}|4k\rangle\right) &= \frac{1}{8}\sum_{k=0}^{63}\left(\frac{1}{16}\sum_{y=0}^{255}e^{-2\pi i(4k)y/256}\,|y\rangle\right) \\ &= \frac{1}{128}\sum_{y=0}^{255}\left(\sum_{k=0}^{63}e^{-2\pi i(4k)y/256}\right)|y\rangle \\ &= \boxed{\frac{1}{128}\sum_{y=0}^{255}\left(\sum_{k=0}^{63}e^{-2\pi iky/64}\right)|y\rangle}. \end{aligned}$$

The amplitude associated with a particular output y is therefore controlled by

$$\boxed{A_y=\sum_{k=0}^{63}e^{-2\pi iky/64}}.$$

2.9 Where cancellation actually happens

The terms

$$1,\ e^{-2\pi iy/64},\ e^{-2\pi i2y/64},\ldots,\ e^{-2\pi i63y/64}$$

are complex amplitudes. For most y, they point around the complex plane in different directions and sum to zero.

Let

$$\omega=e^{-2\pi iy/64}.$$

Then

$$A_y=1+\omega+\omega^2+\cdots+\omega^{63}.$$

If $\omega\neq 1$, the geometric-series formula gives

$$A_y=\frac{1-\omega^{64}}{1-\omega}.$$

But

$$\omega^{64}=e^{-2\pi iy}=1,$$

so

$$\boxed{A_y=0}\qquad(\omega\neq 1).$$

This is exact destructive interference.

If $\omega=1$, every term has the same phase and

$$A_y=\sum_{k=0}^{63}1=64.$$

The condition $\omega=1$ is

$$e^{-2\pi iy/64}=1,$$

which requires

$$\frac{y}{64}\in\mathbb{Z}.$$

Thus

$$\boxed{y = 0, 64, 128, 192}.$$

These four outputs reinforce instead of cancelling.

After normalization, the relevant counting-register state is

$$\boxed{\frac{1}{2}\left(|0\rangle + |64\rangle + |128\rangle + |192\rangle\right)}.$$

Each has probability

$$\left|\frac{1}{2}\right|^2 = \boxed{\frac{1}{4}}.$$

2.10 Why the output spacing is 64

The period is

$$r = 4,$$

while the counting-register size is

$$Q = 256.$$

Fourier peaks therefore occur at

$$y = \frac{sQ}{r}, \qquad s = 0, 1, 2, 3.$$

Hence

$$y = \frac{s(256)}{4} = 64s,$$

so

$$\boxed{y \in \{0, 64, 128, 192\}}.$$

Equivalently,

$$\frac{y}{256} \in \left\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}\right\}.$$

These are the possible values

$$\boxed{\theta = \frac{s}{r}}.$$

2.11 What about $WR = 2, 4, 8$?

For a shifted periodic state

$$\frac{1}{8}\sum_{k=0}^{63} |4k + j\rangle\,,$$

the inverse QFT gives an additional phase factor:

$$QFT^{-1}\left(\frac{1}{8}\sum_{k=0}^{63} |4k + j\rangle\right) = \frac{1}{128}\sum_{y=0}^{255} e^{-2\pi i j y/256}\left(\sum_{k=0}^{63} e^{-2\pi i k y/64}\right)|y\rangle\,.$$

The factor

$$e^{-2\pi i j y/256}$$

changes only the phase of each surviving output, not whether its amplitude is zero. Therefore every translated sequence gives peaks at the same four y values.

This is why the counting-register measurement can reveal the period even though the work register is never measured in the Qiskit code.

2.12 Step 4: Measurement

The circuit measures only the counting register:

```
qc.measure(range(n_count), range(n_count))
```

The work qubits $q_8, \ldots, q_{11}$ are deliberately absent from this call. The IQFT does *not* choose one y value. It first builds amplitudes for all computational basis states. Interference makes most amplitudes vanish and leaves strong amplitudes at the Fourier peaks.

Measurement then samples from

$$\boxed{P(y) = |A_y^{\text{norm}}|^2}.$$

For this exact example, a shot returns one of

$$0, 64, 128, 192$$

with ideal probability $1/4$ each.

In 8-bit binary,

$$0 = 00000000,$$
$$64 = 01000000,$$
$$128 = 10000000,$$
$$192 = 11000000.$$

2.13 Step 5: Recover the period from y/Q

Suppose measurement returns

$$y = 64.$$

Then

$$\frac{y}{Q} = \frac{64}{256} = \frac{1}{4}.$$

The denominator suggests

$$\boxed{r = 4}.$$

If $y = 192$,

$$\frac{192}{256} = \frac{3}{4},$$

which again has denominator 4.

In general, the observed value only approximates s/r, so classical continued fractions are used to recover a candidate denominator r:

$$\frac{y}{Q} \approx \frac{s}{r}.$$

The corresponding classical Qiskit/Python-side post-processing is

```
y = int(bitstring, 2)
phase = y / (2**n_count)
frac = Fraction(phase).limit_denominator(N)
r = frac.denominator
```

This code is not quantum: it interprets the measured integer y and extracts a rational approximation to the phase.

2.14 Step 6: Convert the order into factors

The quantum circuit has now done its job: it has supplied information from which we recovered the candidate order r. Factor recovery from r is entirely classical.

The code first checks the two success conditions discussed in Section 1:

```
if r % 2 != 0:
    return None

x = pow(a, r // 2, N)    # x = a^(r/2) mod N

if x == N - 1:           # x == -1 (mod N)
    return None

p = gcd(x - 1, N)
q = gcd(x + 1, N)
```

Why these gcds? Since r is an order,

$$a^r \equiv 1 \pmod{N},$$

so for even r, with $x = a^{r/2}$,

$$(x-1)(x+1) = a^r - 1 = kN.$$

Thus every prime factor of N must divide the product $(x-1)(x+1)$. In the common textbook setting $N = pq$ with distinct odd primes, each of p and q must divide one of the two factors. They cannot divide both, because any common divisor of $x-1$ and $x+1$ must also divide their difference 2.

There is one important failure mode before taking the gcds. If

$$x \equiv -1 \pmod{N},$$

then

$$x + 1 = mN$$

for some integer m, so

$$\gcd(x+1, N) = N$$

and, because N is odd,

$$\gcd(x-1, N) = 1.$$

Thus **even r alone is not enough**: the algorithm must reject the case in which $a^{r/2} + 1$ is a multiple of N. This is exactly what the code checks with

```
if x == N - 1:           # a^(r/2) == -1 (mod N)
    return None
```

If $x \not\equiv \pm 1 \pmod{N}$, the two prime factors cannot both lie on the same side. Hence they split between $x-1$ and $x+1$, and the two gcds recover p and q (possibly in the opposite order):

$$\boxed{\gcd(x-1, N) = p, \qquad \gcd(x+1, N) = q}$$

or vice versa.

For the present example,

$$r = 4, \qquad a = 2,$$

so

$$x = a^{r/2} = 2^2 = 4.$$

Therefore

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

and

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

Hence

$$\boxed{15 = 3 \times 5}.$$

2.15 The entire calculation in one chain

$$
\begin{aligned}
&|0\rangle^{\otimes 8} |1\rangle \\
&\xrightarrow{H^{\otimes 8}} \frac{1}{16} \sum_{x=0}^{255} |x\rangle |1\rangle \\
&\xrightarrow{\text{controlled modular exponentiation}} \frac{1}{16} \sum_{x=0}^{255} |x\rangle |2^x \bmod 15\rangle \\
&\xrightarrow{QFT_{CR}^{-1} \otimes I_{WR}} \text{interference in the counting-register amplitudes} \\
&\longrightarrow \boxed{y \in \{0, 64, 128, 192\}} \\
&\longrightarrow \frac{y}{256} \in \left\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}\right\} \\
&\longrightarrow r = 4 \\
&\longrightarrow \gcd(2^{4/2} - 1, 15) = 3, \qquad \gcd(2^{4/2} + 1, 15) = 5.
\end{aligned}
$$

2.16 Gate-by-gate mathematical meaning

Circuit operation	**Mathematical meaning**
Hadamard on all counting qubits	$\|0\rangle^{\otimes 8} \mapsto \frac{1}{16} \sum_{x=0}^{255} \|x\rangle$
Controlled $U_a^{2^q}$	Encodes $a^x \bmod N$ in the work register
QFT^{-1} on CR	Converts periodic spacing in x into peaks in y
Measurement of CR	Samples y according to the squared Fourier amplitudes
Continued fractions	Recovers a candidate r from $y/Q \approx s/r$

2.17 IQFT intuition from one and two qubits

For one qubit,

$$QFT^{-1} = H,$$

so

$$QFT^{-1} |0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \qquad QFT^{-1} |1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

The minus sign is a relative phase. When amplitudes contributing to the same output basis state are later added, relative phases determine whether they reinforce or cancel.

For two qubits, $Q = 4$ and

$$QFT^{-1} |x\rangle = \frac{1}{2} \sum_{y=0}^{3} e^{-2\pi i x y/4} |y\rangle .$$

Each input basis state contributes to every output basis state, but with a phase depending on xy. For a superposition of several input x values, the amplitudes for the same output $|y\rangle$ add. This small case is the same mechanism used by the 8-qubit Shor example.

3 Summary

Shor's algorithm separates the factoring problem into a **quantum order-finding stage** and a **classical factor-recovery stage**. The quantum computer is not asked to guess the factors of N directly. Its job is to expose the period of the modular function

$$f(x) = a^x \bmod N.$$

The complete logical flow is

$$\boxed{\begin{array}{c} N, a \longrightarrow \text{superposition over } x \longrightarrow \text{controlled } U_a^{2^j} \longrightarrow \text{periodic correlations} \\ \xrightarrow{QFT^{-1}} \text{peaks in } y \longrightarrow \dfrac{y}{Q} \approx \dfrac{s}{r} \longrightarrow r \\ \longrightarrow \gcd(a^{r/2}-1, N),\ \gcd(a^{r/2}+1, N) \end{array}}$$

For the worked example $N = 15$ and $a = 2$, the modular sequence

$$1, 2, 4, 8, 1, 2, 4, 8, \ldots$$

has order

$$\boxed{r = 4}.$$

With an 8-qubit counting register, $Q = 256$, so the IQFT produces the four peaks

$$\boxed{y = 0, 64, 128, 192}.$$

A useful non-zero measurement, such as $y = 64$, gives

$$\frac{y}{Q} = \frac{64}{256} = \frac{1}{4},$$

from which classical continued fractions recover $r = 4$. Finally,

$$2^{r/2} = 2^2 = 4,$$

and the classical gcd calculations give

$$\gcd(4-1, 15) = 3, \qquad \gcd(4+1, 15) = 5.$$

Thus

$$\boxed{15 = 3 \times 5}.$$

The important conceptual points are:

- the Hadamards create a coherent superposition of possible exponents x;
- controlled modular exponentiation implements

$$|x\rangle\,|1\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle\,;$$

- the counting and work registers become correlated through the controlled U operations;
- the IQFT acts only on the counting register;
- amplitudes corresponding to incompatible output frequencies cancel, while frequencies compatible with the period reinforce;
- measurement samples one surviving Fourier peak rather than reading the work register term by term;
- continued fractions recover a candidate order r from $y/Q \approx s/r$;
- even r is necessary but not sufficient: if $a^{r/2} \equiv -1 \pmod N$, then $a^{r/2}+1$ is a multiple of N, so the two gcds collapse to N and 1;
- therefore the successful post-processing condition is r even together with $a^{r/2} \not\equiv -1 \pmod N$; under this condition the final gcd calculations can reveal non-trivial factors of N.

3.1 Code-to-math map

The main Qiskit instructions correspond directly to the mathematical operations:

$$\texttt{qc.h(q)} \quad \longleftrightarrow \quad H^{\otimes 8},$$

$$\texttt{c\_amod15(a,2**q)} \quad \longleftrightarrow \quad \text{controlled } U_a^{2^q},$$

$$\texttt{QFTGate(n\_count).inverse()} \quad \longleftrightarrow \quad QFT_{CR}^{-1} \otimes I_{WR},$$

$$\texttt{qc.measure(...)} \quad \longleftrightarrow \quad P(y) = |\langle y \mid \psi_{CR} \rangle|^2,$$

$$\texttt{Fraction(phase).limit\_denominator(N)} \quad \longleftrightarrow \quad \frac{y}{Q} \approx \frac{s}{r},$$

$$\texttt{gcd(x - 1,N), gcd(x + 1,N)} \quad \longleftrightarrow \quad \gcd(a^{r/2} - 1, N),\ \gcd(a^{r/2} + 1, N).$$

4 Reference

The following is the handwritten summary page used as the mathematical reference for this guide.

Summary

Saturday, August 8, 2026 15:27

I Learned what the Shor's algorithm is.

eg say you want to find the factor N = 314191.

and candidates are [101, 127, etc. . .]

1st: You will determine how many qubits you are going to use.

$2^n \le 314191 < 2^{n+1}$

determine the value

since $\log_2 314191 \approx 18.2618$

so we need 19 qubits to represent remainder (Work Registry)

we need $\lceil \log_2 N^2 \rceil$ qubits for storing y^x (Counting Registry)

So in total we need 19 + 37 = 56

56 qubits — 19 qubits for Work Registry; 37 qubits for Counting Registry.

2nd: Use Q comp to find value r which satisfies

$y^r - 1 = m \cdot N$ ... ①

$(y^{\frac{r}{2}}+1)\cdot(y^{\frac{r}{2}}-1) = m \cdot N$ ... ② so r must be an even number.

Choose y from candidate.

y = 101 ~ Q computer ~

2.1 Apply H gate on CR

so let quantum state at CR be $|\psi\rangle_{CR}$

then $|\psi\rangle_{CR} = \frac{1}{\sqrt{2^{37}}} \sum_{x=0}^{2^{37}-1} |x\rangle$

2.2 At the start of work registry, you will apply X gate

why? so that when you calculate $y^x \bmod N$, you will get $1 \cdot y^x \bmod N$

in other word. when you apply a function, you will make it applicable.

ask GPT for better explanation or at least " why it is $I \otimes U$?

$I \to$ [Apply U] $\to U \to$ [Apply U] $\to U^2 \to$ [Apply U] $\to U^3$

2.3: You will create a U registry and apply the U registries for calculating modular exponential.

image: $y^x \to$ [U] $\to y^x \bmod 15$ → stored on Work registry.

2.4: Then you will entangle the qubits as mentioned before.

let whole quantum states to be represented as $|\psi\rangle$ and let 2^{37} be denoted as N for simplicity.

$|\psi\rangle = \frac{1}{\sqrt{2^{37}}} \sum_{x=0}^{2^{37}-1} |x\rangle_{CR} \otimes U|x\rangle_{WR}$

2.5: Apply Inverse Quantum Fourier Transformation to CR.

If I apply QFT^{-1} only to CR

$QFT^{-1}_{CR} \otimes I_{WR}$

this means you will apply QFT^{-1} on Count Register but not on Work Register so you will Apply Identity phase to it.

so $QFT^{-1}_{CR} \otimes I_{WR}$

$= \frac{1}{\sqrt{2^{37}}} \sum_{x=0}^{2^{37}-1} QFT^{-1}|x\rangle \otimes U|x\rangle$

$= \frac{1}{\sqrt{2^{37}}} \sum_{x=0}^{2^{37}-1} \left(\frac{1}{\sqrt{2^{37}}} \sum_{y=0}^{2^{37}-1} e^{\frac{-2\pi i \cdot xy}{2^{37}}} |y\rangle \right) \otimes U|x\rangle$

$= \frac{1}{2^{37}} \sum_{x=0}^{2^{37}-1} \sum_{y=0}^{2^{37}-1} e^{\frac{-2\pi i \cdot xy}{2^{37}}} |y\rangle \otimes U|x\rangle$

The key is here

on QFT^{-1}, you will calculate the direction or in other word direction of the arrow.

How would you do that?

↳ calculating $\sum_{y=0}^{2^{37}-1} e^{\frac{-2\pi i \cdot xy}{2^{37}}}$

say you're calculating QFT on such state.

$|\delta\rangle = \frac{1}{\sqrt{4}} \left(|0\rangle + |1\rangle + |2\rangle + |3\rangle \right)$

$|\delta\rangle = \frac{1}{\sqrt{4}} \sum_{x=0}^{4-1} (-1)^x |x\rangle$

$QFT^{-1}|\delta\rangle = \frac{1}{\sqrt{4}} \sum_{x=0}^{4-1} (-1)^x QFT^{-1}|x\rangle$

$= \frac{1}{\sqrt{4}} \sum_{x=0}^{4-1} (-1)^x \frac{1}{\sqrt{4}} \sum_{y=0}^{4-1} e^{\frac{-2\pi i \cdot xy}{4}} |y\rangle$

$= \frac{1}{\sqrt{4}} \cdot \frac{1}{\sqrt{4}} \sum_{x=0}^{4-1} \sum_{y=0}^{4-1} e^{\frac{-2\pi i \cdot xy}{4}} \cdot (-1)^x |y\rangle$

$= \frac{1}{4} \sum_{x=0}^{4-1} \sum_{y=0}^{4-1} e^{\frac{-2\pi i \cdot xy}{4}} \cdot (-1)^x |y\rangle$

when x=0, $(e^0 \cdot |0\rangle + e^0 \cdot |1\rangle + e^0 \cdot |2\rangle + e^0 \cdot |3\rangle) \cdot (-1)^0$

$= |0\rangle + |1\rangle + |2\rangle + |3\rangle$

when x=1, $(e^0 \cdot |0\rangle + e^{\frac{-2\pi i \cdot 1}{4}} |1\rangle + e^{\frac{-2\pi i \cdot 2}{4}} |2\rangle + e^{\frac{-2\pi i \cdot 3}{4}} |3\rangle) \cdot (-1)^1$

$= (|0\rangle - i|1\rangle - |2\rangle + i|3\rangle) \cdot (-1)^1$

$= -|0\rangle + i|1\rangle + |2\rangle - i|3\rangle$

when x=2, $|0\rangle + e^{\frac{-2\pi i \cdot 2}{4}} |1\rangle + e^{\frac{-2\pi i \cdot 4}{4}} |2\rangle + e^{\frac{-2\pi i \cdot 6}{4}} |3\rangle$

$= |0\rangle - |1\rangle + |2\rangle - |3\rangle$

when x=3, $(|0\rangle + e^{\frac{-2\pi i \cdot 3}{4}} |1\rangle + e^{\frac{-2\pi i \cdot 6}{4}} |2\rangle + e^{\frac{-2\pi i \cdot 9}{4}} |3\rangle) \cdot (-1)^3$

$= (|0\rangle + i|1\rangle - |2\rangle - i|3\rangle) \cdot (-1)$

$= -|0\rangle - i|1\rangle + |2\rangle + i|3\rangle$

→ So the results of QFT^{-1} is:

x=0	$\|0\rangle +$	$\|1\rangle +$	$\|2\rangle +$ $\|3\rangle$
x=1	$-\|0\rangle +$	$i\|1\rangle +$	$\|2\rangle -$ $i\|3\rangle$
x=2	$+\|0\rangle -$	$\|1\rangle +$	$\|2\rangle -$ $\|3\rangle$
x=3	$-\|0\rangle -$	$i\|1\rangle +$	$\|2\rangle +$ $i\|3\rangle$

$= 4|2\rangle$

so sub in $\sum_{x=0}^{3} \sum_{y=0}^{3} e^{\frac{-2\pi i \cdot xy}{4}} \cdot (-1)^x |y\rangle = 4|2\rangle$.

$= \frac{1}{4} \cdot 4|2\rangle$

$= |2\rangle$

therefore

$QFT^{-1}|\psi\rangle = |2\rangle$ ↔ what does this mean? the "2" sign for frequency of 2

Important notice:

the frequency is not always repped as a single state, it usually expressed as superposition

2.6: After QFT,

always frequency is represented as sth like that.

$\alpha|m\rangle + \beta|n\rangle + \cdots$

and you will measure the value

this will be either m or n or whichever.

then you will do $\frac{m}{2^{37}}$, $\frac{n}{2^{37}}$ to work out $\frac{1}{p}$ and then eventually you will get value "p"

But if such p is an odd number then you will do it again with different y

eg factorise N = 15 with a = 2

When N = 15, we need 2^4 for work registry since

$2^3 < 15 < 2^4$

and since $\lceil \log_2 15^2 \rceil = 8$, the counting registry is 8 qubits.

$2^0 \equiv 1 \pmod{15}$

$2^1 \equiv 2 \pmod{15}$

$2^2 \equiv 4 \pmod{15}$

$2^3 \equiv 8 \pmod{15}$

$2^4 \equiv 1 \pmod{15}$

So work registry will store $|0\rangle$, $|2\rangle$, $|4\rangle$, and $|8\rangle$ and nothing else after the circuit calculate the modular exponential with U gate.

let whole state to be denoted as $|\psi\rangle$.

first apply H gate on $|\psi\rangle_{CR}$

$|\psi\rangle_{CR} = \frac{1}{\sqrt{2^8}} \sum_{x=0}^{2^8-1} |x\rangle_{CR}$

then apply U gate (modular exponential) on the controlled bit.

$|\psi\rangle = \frac{1}{\sqrt{2^{8}}} \sum_{x=0}^{2^{8}-1} |x\rangle \otimes U|x\rangle$

$= \frac{1}{\sqrt{2^{8}}} \sum_{x=0}^{2^{8}-1} |x\rangle_{CR} |2^x \bmod 15\rangle_{WR}$ ← since it move in |0⟩ |2⟩ |4⟩ and |8⟩ regularly ...

$= \frac{1}{\sqrt{2^{8}}} \sum_{k=0}^{63} |4k\rangle + |4k+1\rangle + |4k+2\rangle + |4k+3\rangle |8\rangle.$

$= \frac{1}{\sqrt{2^{8}}} \sum_{k=0}^{63} \sum_{\alpha=0}^{3} |4k+\alpha\rangle |2^\alpha\rangle$

↳ notice there 256 did not change.

that is because there are still 256 states left in the quantum state.

Then apply QFT between CR ~ 7th qubit (which is whole counting registry)

$QFT^{-1}_{CR} \otimes I_{WR}$

$QFT^{-1}|\psi\rangle = \frac{1}{\sqrt{2^{8}}} \sum_{k=0}^{63} \sum_{\alpha=0}^{3} QFT^{-1}|4k+\alpha\rangle_{CR} |2^\alpha\rangle_{WR}$

$= \frac{1}{\sqrt{2^{8}}} \sum_{k=0}^{63} \sum_{\alpha=0}^{3} \frac{1}{\sqrt{2^8}} \sum_{y=0}^{255} e^{\frac{-2\pi i (4k+\alpha) \cdot y}{2^8}} |y\rangle_{CR} |2^\alpha\rangle_{WR}$

$= \frac{1}{2^8} \sum_{k=0}^{63} \sum_{\alpha=0}^{3} \sum_{y=0}^{255} e^{\frac{-2\pi i (4k+\alpha) \cdot y}{2^8}} |y\rangle_{CR} |2^\alpha\rangle_{WR}$

$= \frac{1}{2^8} \sum_{k=0}^{63} \sum_{\alpha=0}^{3} \sum_{y=0}^{255} e^{\frac{-2\pi i \cdot 4k \cdot y}{2^8}} \cdot e^{\frac{-2\pi i \cdot \alpha \cdot y}{2^8}} |y\rangle_{CR} |2^\alpha\rangle_{WR}$

If the value $e^{\frac{-2\pi i \cdot 4k \cdot y}{2^8}}$ is always one, it will just increment forever.

$\frac{-2\pi i \cdot 4k \cdot y}{2^8} = \frac{-2\pi i \cdot k y}{64}$

so when y = 64 · n for any natural number when $y \le 255$, the value $e^{\frac{-2\pi i \cdot ky}{64}}$ will be 1.

What if y is not a multiple of 64

let's name $\sum_{k=0}^{63} e^{\frac{-2\pi i \cdot ky}{64}}$

$= e^1 + e^{\frac{-2\pi i \cdot y}{64}} + e^{\frac{-2\pi i \cdot 2y}{64}} + \cdots + e^{\frac{-2\pi i \cdot 63y}{64}}$

let $r = e^{\frac{-2\pi i \cdot y}{64}}$

$= 1 + r + r^2 + \cdots + r^{63}$

$= \frac{1 - r^{64}}{1 - r}$

sub in back

$= \frac{1 - e^{64 \cdot \frac{-2\pi i \cdot y}{64}}}{1 - e^{\frac{-2\pi i \cdot y}{64}}}$

$= \frac{1 - e^{-2\pi i \cdot y}}{1 - e^{\frac{-2\pi i \cdot y}{64}}}$ ← let's have a closer look at numerator.

$e^{-2\pi i \cdot y}$ → for any natural number y, $-2\pi i \cdot y$ will just be divisible by 2π so $e^{-2\pi i \cdot y} = 1$.

therefore numerator is always 1-1=0.

so $\sum_{k=0}^{63} e^{\frac{-2\pi i \cdot ky}{64}} = \frac{0}{1 - e^{\frac{-2\pi i \cdot y}{64}}}$

and this is only when denominator is not 0

$0 = 1 - e^{\frac{-2\pi i \cdot y}{64}}$

$e^{\frac{-2\pi i \cdot y}{64}} = 1$

let d be any natural number $d \in \mathbb{Z}$

$\frac{-2\pi i \cdot y}{64} = -2\pi i d$

$\frac{y}{64} = d$

$y = 64 \cdot d$

so if $y \neq 64 \cdot d$ for some $d \in \mathbb{Z}$, then denominator will be greater than 0 and will eventually be 0.

$QFT^{-1}|\psi\rangle = \frac{1}{\sqrt{2^{8}}} \sum_{k=0}^{63} \sum_{\alpha=0}^{3} QFT^{-1}|4k+\alpha\rangle_{CR} |2^\alpha\rangle_{WR}$

$= \frac{1}{\sqrt{2^{8}}} \sum_{k=0}^{63} \sum_{\alpha=0}^{3} \frac{1}{\sqrt{2^8}} \sum_{y=0}^{255} e^{\frac{-2\pi i (4k+\alpha) \cdot y}{2^8}} |y\rangle_{CR} |2^\alpha\rangle_{WR}$

$= \frac{1}{2^8} \sum_{k=0}^{63} \sum_{\alpha=0}^{3} \sum_{y=0}^{255} e^{\frac{-2\pi i (4k+\alpha) \cdot y}{2^8}} |y\rangle_{CR} |2^\alpha\rangle_{WR}$

$= \frac{1}{2^8} \sum_{k=0}^{63} \sum_{\alpha=0}^{3} \sum_{y=0}^{255} e^{\frac{-2\pi i \cdot 4k \cdot y}{2^8}} \cdot e^{\frac{-2\pi i \cdot \alpha \cdot y}{2^8}} |y\rangle_{CR} |2^\alpha\rangle_{WR}$

$= \frac{1}{2^8} \sum_{k=0}^{63} \sum_{\alpha=0}^{3} \sum_{m=0}^{\frac{255}{64}} e^{\frac{-2\pi i \cdot \alpha \cdot 64m}{2^8}} e^{\frac{-2\pi i \cdot 4k \cdot 64m}{2^8}} |64m\rangle_{CR} |2^\alpha\rangle_{WR}$

$= \frac{1}{2^8} \sum_{\alpha=0}^{3} \sum_{m=0}^{3} \sum_{k=0}^{63} e^{\frac{-2\pi i \cdot \alpha \cdot 64m}{2^8}} \cdot e^{-2\pi i \cdot km} |64m\rangle |2^\alpha\rangle$

$= \frac{1}{2^8} \sum_{\alpha=0}^{3} \sum_{m=0}^{3} 64 \cdot e^{\frac{-2\pi i \cdot \alpha \cdot m}{4}} |64m\rangle |2^\alpha\rangle$

$= \frac{1}{4} \sum_{\alpha=0}^{3} e^{\frac{-2\pi i \cdot \alpha \cdot m}{4}} |64m\rangle |2^\alpha\rangle$

$= \frac{1}{4} \sum_{\alpha=0}^{3} |0\rangle |2^\alpha\rangle + e^{\frac{-2\pi i}{4} \cdot 1} |64\rangle |2^\alpha\rangle + e^{\frac{-2\pi i}{4} \cdot 2} |128\rangle |2^\alpha\rangle + e^{\frac{-2\pi i}{4} \cdot 3} |192\rangle |2^\alpha\rangle$

when you do $\sum_{\alpha=0}^{3} |0\rangle |2^\alpha\rangle$

$= |0\rangle (|1\rangle + |2\rangle + |4\rangle + |8\rangle)$

then amplitude will be $\sqrt{1^2 + 1^2 + 1^2 + 1^2} = \sqrt{4}$

$= 2$

since amplitude must be 1, you'll normalize it to $\frac{1}{2}$.

so put them in correct formula

$2 \cdot \frac{1}{2} |0\rangle (|1\rangle + |2\rangle + |4\rangle + |8\rangle)$.

But since we will not measure WR, we will ignore WR

then WR part to be denoted as $|WR\rangle = \frac{1}{\sqrt{4}} |2^\alpha\rangle$

$= \frac{1}{4} \cdot (2 \cdot |0\rangle |WR\rangle + 2 \cdot |64\rangle |WR\rangle + 2 \cdot |128\rangle |WR\rangle + 2 \cdot |192\rangle |WR\rangle)$

$= \frac{1}{2} (|0\rangle |WR\rangle + |64\rangle |WR\rangle + |128\rangle |WR\rangle + |192\rangle |WR\rangle)$.

↳ Therefore Q comp will measure $|0\rangle$, $|64\rangle$, $|128\rangle$, and $|192\rangle$ in equal probability

5 Complete Qiskit code

Short code snippets were placed next to the concepts they implement. The entire runnable implementation is also collected here at the very back for reference.

```
from fractions import Fraction
from math import gcd

from qiskit import QuantumCircuit, transpile
from qiskit.circuit.library import QFTGate
from qiskit_aer import AerSimulator

N = 15


def c_amod15(a, power):
    """Controlled multiplication by a^power mod 15."""
    if a not in [2, 4, 7, 8, 11, 13]:
        raise ValueError("'a' must be 2, 4, 7, 8, 11 or 13")

    U = QuantumCircuit(4)

    for _ in range(power):
        if a in [2, 13]:
            U.swap(2, 3)
            U.swap(1, 2)
            U.swap(0, 1)

        if a in [7, 8]:
            U.swap(0, 1)
            U.swap(1, 2)
            U.swap(2, 3)

        if a in [4, 11]:
            U.swap(1, 3)
            U.swap(0, 2)

        if a in [7, 11, 13]:
            for q in range(4):
                U.x(q)

    U = U.to_gate()
    U.name = f"{a}^{power} mod 15"
    return U.control()


def qpe_amod15(a=2, n_count=8, shots=1024):
    qc = QuantumCircuit(n_count + 4, n_count)

    # 1. Uniform superposition over x = 0,...,2^n_count-1.
    for q in range(n_count):
        qc.h(q)

    # 2. Initialize the four-qubit work register to |1>.
    qc.x(n_count)

    # 3. Controlled modular exponentiation.
    for q in range(n_count):
        qc.append(
            c_amod15(a, 2**q),
            [q] + [n_count + i for i in range(4)]
```

```
        )

    # 4. IQFT on the counting register only.
    qc.append(QFTGate(n_count).inverse(), range(n_count))

    # 5. Measure the counting register only.
    qc.measure(range(n_count), range(n_count))

    sim = AerSimulator()
    tqc = transpile(qc, sim)
    result = sim.run(tqc, shots=shots).result()

    return qc, result.get_counts()


def candidate_orders_from_counts(counts, a=2, n_count=8, N=15):
    Q = 2**n_count
    candidates = []

    for bitstring, frequency in counts.items():
        y = int(bitstring, 2)

        if y == 0:
            continue

        phase = y / Q
        frac = Fraction(phase).limit_denominator(N)
        r = frac.denominator

        # Check that r really is an order candidate.
        if pow(a, r, N) == 1:
            candidates.append((y, frequency, frac, r))

    return candidates


def factors_from_order(a, r, N=15):
    if r % 2 != 0:
        return None

    x = pow(a, r // 2, N)

    if x == N - 1:
        return None

    p = gcd(x - 1, N)
    q = gcd(x + 1, N)

    if p in (1, N) or q in (1, N):
        return None

    return p, q


qc, counts = qpe_amod15(a=2, n_count=8, shots=1024)
print(counts)

for y, frequency, frac, r in candidate_orders_from_counts(counts):
    factors = factors_from_order(2, r, 15)
    print(
        "y =", y,
```

```
        "frequency =", frequency,
        "phase ~=", frac,
        "r =", r,
        "factors =", factors,
    )
```