# QUANTUM COMPUTING

UNIT 2

Quantum Algorithms

Dr Anjaneyulu Pasala

# Quantum Algorithms

- It may be possible to solve a problem on a quantum system much faster (i.e., using fewer steps) than on a classical computer
- Factoring and searching are examples of problems where quantum algorithms are known and are faster than any classical ones
- Implications for cryptography, information security
- What makes a quantum algorithm potentially faster than any classical one?
  - **Quantum parallelism:** by using superpositions of quantum states, the computer is executing the algorithm on all possible inputs at once
  - **Dimension of quantum Hilbert space:** the "size" of the state space for the quantum system is exponentially larger than the corresponding classical system
  - **Entanglement capability:** different subsystems (qubits) in a quantum computer become entangled, exhibiting nonclassical correlations

# Deutsch-Jozsa Problem

Problem: Given a Black-box (oracle) function: $f:\{0,1\}^n \rightarrow \{0,1\}$ which takes an n-bit input and returns 0 or 1.
Promised that the function is either:

- Constant: all outputs are the same (always 0 or always 1). OR
- Balanced: exactly half of the inputs give 0 and half give 1.

The task is: determine whether f is constant or balanced using fewest possible evaluations

Example:

- Consider a binary function f(x) that takes one bit, 0 or 1 as input and outputs either 0 or 1
- There are 4 possible combinations.
- In 2 cases, the function is balanced (i.e. output is dependent): returns the same value (0 or 1)
- In other 2 cases, the function is constant (i.e. output is independent): Returns 0 for half of the inputs, and 1 for the other half.

.Balanced:
f(0) = 0, f(1) = 1
f(0) = 1, f(1) = 0

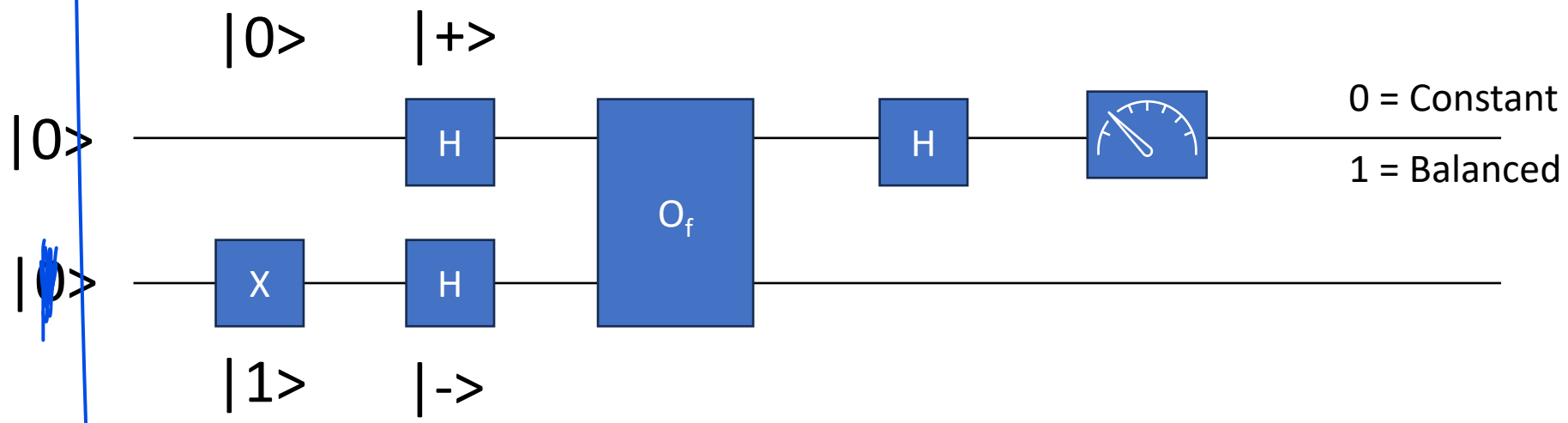
Constant:
f(0) = 0, f(1) = 0
f(0) = 1, f(1) = 1

# Deutsch-Jozsa Problem

- This can be solved only by evaluating the function twice in classical computer.
- If f is classical:
  - You might have to check $2^{n-1}+1$ inputs in the worst case
  - Only then you can be sure whether f is constant or balanced
- But a quantum computer can solve it by evaluating it only once using superposition.
  - With just one query to the oracle $U_f$.
  - That is exponential speed
- Oracle implements the function as:
  - $U_f|x,y\rangle = |x,y \oplus f(x)\rangle$
  - $|x\rangle$ is the input register (n qubits)
  - $|y\rangle$ is an auxiliary qubit
  - $\oplus$ = XOR operation

# Deutsch-Jozsa Problem

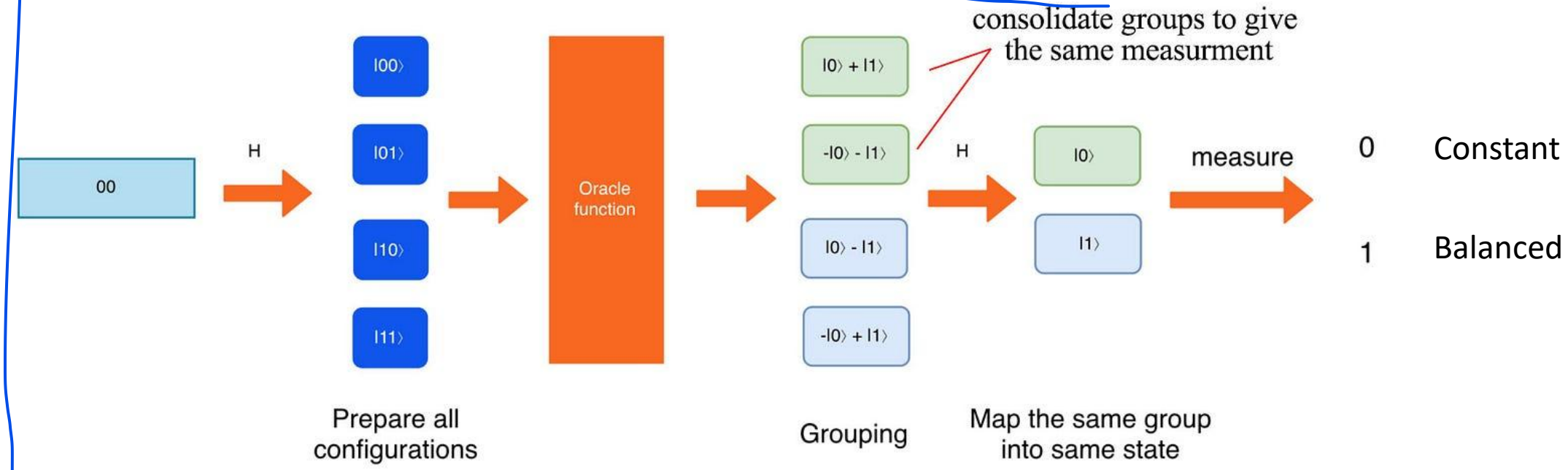
- Step1: Initialize qubits |0> and |1>
- Step2: Place the qubits into superposition
- Step3: Pass the inputs into the oracle function
- Step4: apply transformation to group and measure output



# Deutsch-Jozsa Problem

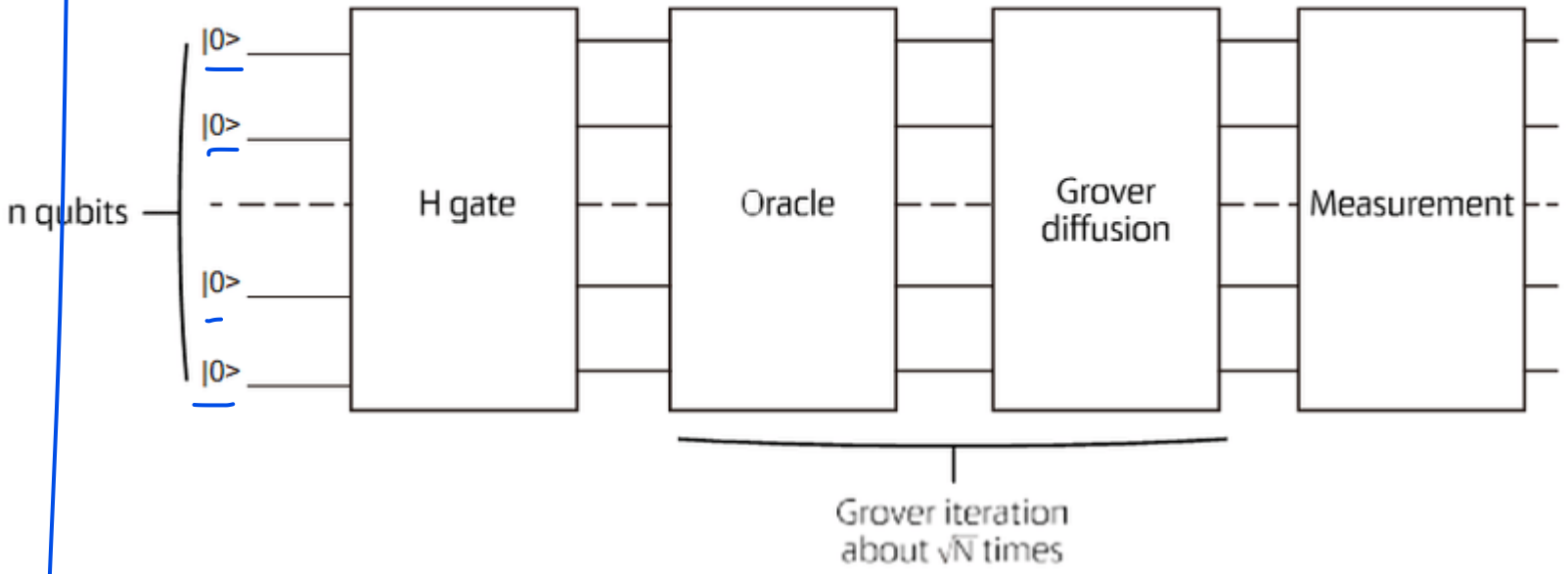
- For ‘n’ bit input, we need at least $2^{n-1}+1$ evaluations.
- Eg: 32 bit problem will take $2^{32-1}+1 = 2,14,74,83,649$ evaluations.

- In quantum computer, this problem can be solved with 100% confidence after only one call to the function f(x) [for a single bit]



# Grover's Algorithm

- **Grover's Algorithm** helps to find a specific item in an unsorted database much faster than any classical method.
- Instead of checking each item one by one, it uses quantum properties to locate the target in about √N steps, where N is the total number of items.



# Grover's Algorithm – Step 1

The **Hadamard gate (H)** is applied to each qubit at the start of Grover's algorithm.

- It creates an **equal superposition** of all possible states.
- It means the quantum system is now "looking at" all possible solutions **at the same time**.

**Example**:

For 2 qubits (N = 4), initial state is: $|00\rangle$

After Hadamard on both qubits: $\frac{1}{2}(|00\rangle+|01\rangle+|10\rangle+|11\rangle) = \frac{1}{4}$ the possibility of each state

Now the system holds **all 4 possibilities equally**—this is key to **quantum parallelism**.

# Grover's Algorithm – Step 2 – Mark the answer

The **oracle** is a black-box quantum function that **flips the sign** (i.e., multiplies by -1) of the amplitude of the **correct (target)** solution.

- Think of the oracle as a "magnet" that tags the right answer by inverting its amplitude.
- It doesn't identify the answer directly but sets it apart so that the next step (the diffuser) can amplify it.

Example:

Let's say the correct answer is $|11\rangle$. The state after oracle function

$$\frac{1}{2}\,(|00\rangle+|01\rangle+|10\rangle+|11\rangle) \;\rightarrow\; \frac{1}{2}(|00\rangle+|01\rangle+|10\rangle-|11\rangle)$$

[0.5, 0.5, 0.5, -0.5] ← oracle flipped $|11\rangle$

Only the amplitude of $|11\rangle$ is **flipped**. That's how Grover "marks" it.

# Grover's Algorithm – Step 3 - Amplitude Amplification

- The **diffuser** (or Grover diffusion operator) reflects all amplitudes about their **average** value.
- This step is often called **inversion about the mean**.

- This step increases the **amplitude of the marked state**, making it **more likely to be measured**.
- The rest of the amplitudes go down slightly, ensuring the right answer "stands out" over several repetitions.

**Calculate the average**:

$$\text{Avg} = \frac{0.5+0.5+0.5-0.5}{4} = \frac{1.0}{4} = 0.25$$

**Apply transformation to each amplitude**:

| Initial state | Value | 2*avg - Initial | New state |
|---|---|---|---|
| $\|00\rangle$ | 0.5 | = 0.5 - 0.5 | 0 |
| $\|01\rangle$ | 0.5 | = 0.5 - 0.5 | 0 |
| $\|10\rangle$ | 0.5 | = 0.5 - 0.5 | 0 |
| $\|11\rangle$ | -0.5 | = 0.5 + 0.5 | 1 |

- Repeat **Oracle + Diffuser** roughly **√N times**.
- After that, you **measure** the state and with high probability, you get the correct answer.

# Grover's Algorithm

Grover's algorithm solves the following problem efficiently:
Given a function $f:\{0,1\}^n \to \{0,1\}$,
find an input $x_0$ such that $f(x_0) = 1$.

**The Setup**

We use two registers:

- **Input register**: $n$ qubits initialized to $|0\rangle^{\otimes n}$
- **Ancilla qubit**: 1 qubit initialized to $|1\rangle$

The overall initial state:

$$|\psi_0\rangle = |0\rangle^{\otimes n} |1\rangle$$

Then apply **Hadamard gates** to all qubits.

After applying Hadamard gates:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \cdot \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle)$$

The ancilla qubit (last one) prepares the oracle to "flip the sign" correctly later.

# Grover's Algorithm

**The Oracle $O_f$**

The **oracle** is a quantum subroutine that marks the correct solution by flipping its phase.

$$O_f \mid x\rangle = (-1)^{f(x)} \mid x\rangle$$

- If $f(x) = 1$ :phase becomes negative (sign flip)
- If $f(x) = 0$ :unchanged

So it identifies the "target" state $\mid x_0\rangle$.

**The Diffusion Operator (Amplification Step)**

After marking the target state, we apply the **diffusion operator** (also called "inversion about the mean"):

$$D = 2 \mid \psi\rangle\langle\psi \mid -I$$

Where:

$$\mid \psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \mid x\rangle$$

**Purpose:**

The diffusion operator amplifies the amplitude of the correct state and reduces the others, gradually making the probability of measuring the correct $\mid x_0\rangle$very high.

# Grover's Algorithm

**Grover Iteration**

Each Grover iteration = **Oracle + Diffusion**

1.Oracle flips phase of target state

$$|x_0\rangle \rightarrow -|x_0\rangle$$

2.Diffusion operator increases its amplitude

After $k$iterations:

$$k \approx \frac{\pi}{4}\sqrt{N}$$

you get maximum probability of measuring the correct state.

# Grover's Algorithm: Example

**Example:** $N = 4$**(2 qubits)**

Let's take 2 qubits $\Rightarrow N = 2^2 = 4$

We want to find $f(x) = 1$for $x_0 = 11$.

**Step 1: Initialize** $|\psi_0\rangle = |00\rangle$

After Hadamard gates:

$$|\psi_1\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

**Step 2: Oracle**

Oracle flips phase of marked state $|11\rangle$:

$$O_f\,|\psi_1\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$$

**Step 3: Diffusion Operator**

The diffusion operator reflects all amplitudes about their average.

Current amplitudes: $\left[\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, -\frac{1}{2}\right]$

Average amplitude:

$$\bar{a} = \frac{1}{4}\left(\frac{1}{2} + \frac{1}{2} + \frac{1}{2} - \frac{1}{2}\right) = \frac{1}{4}$$

# Grover's Algorithm: Example

New amplitude after reflection:

$a_i' = 2a - a_i$

Compute:

| State | Old Amplitude | New Amplitude |
|---|---|---|
| 00 | 1/2 | 0 |
| 01 | 1/2 | 0 |
| 10 | 1/2 | 0 |
| 11 | -1/2 | 1 |

So the new state:

$$|\psi_2\rangle = |11\rangle$$

On measurement, we get **$|11\rangle$ with probability 1.**

# Shor's Algorithm

- Imagine you have a big number like **N = 15**, and you want to find its **prime factors** (3 and 5).
- Classical computers can do this by trial and error, but for very large numbers (hundreds of digits), it can take **years**.
- The best-known algorithms for factoring integers (like RSA numbers) take **superpolynomial time**, roughly

$$O\left(e^{(1.9)\left(\log N\right)^{1/3}\left(\log\log N\right)^{2/3}}\right)$$

- Shor's Algorithm can **factor large numbers exponentially faster** using a quantum computer.
- **Shor's algorithm** does it in **polynomial time**, approximately

$$O\left(\left(\log N\right)^{3}\right)$$

- And that's why Shor's algorithm threatens classical encryption systems like RSA.
- Factoring large numbers is the basis of **RSA encryption**

# Shor's Algorithm

Shor's algorithm is a quantum algorithm for integer factorization — it finds the prime factors of a large composite number $N$ efficiently.

**Classical observation:**

If we can find a number $r$ such that

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

then $r$ is called the **order** of $a$ modulo $N$.

Then:

$$(a^{r/2} - 1)(a^{r/2} + 1) \equiv 0 \pmod{N}$$

So, with high probability:

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

will give us a non-trivial factor of $N$.

# Shor's Algorithm

**Quantum Part — Period Finding**

This is the **core quantum advantage**.

**The function**

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

is **periodic** with some period $r$, meaning:

$$f(x + r) = f(x)$$

**The goal is to find** $r$ efficiently using **quantum parallelism and interference**.

# Shor's Algorithm

**It is a two-step process**

**Step 1**: Choose a Random Number a:

a. Pick a number **a** such that **1 < a < N**

b. Compute a **coprime to N** (i.e., gcd(a, N) = 1).

For example, with N = 15, choose a = 2 or 4 or 7 or 11 or 13

If gcd($a$,$N$)≠1, you've already found a factor!
Otherwise, proceed.

**Step 2**: Period Finding **r**

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

This function repeats every r steps called (**period**). A quantum computer uses **superposition**, **quantum Fourier transform**, and **interference** to **find r** efficiently.

# Shor's Algorithm – Step 2

Step 2: Period Finding **r**

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

| x | $7^x$ mod15 | Calculation |
|---|---|---|
| **0** | 1 | $7^0$ mod 15 = 1 |
| **1** | 7 | $7^1$mod 15 = 7 |
| **2** | 4 | $7^2$ mod15 = 49 mod 15 = 4 |
| **3** | 13 | $7^3$ mod 15 = 343 mod 15 = 13 |
| **4** | 1 | $7^4$ mod15= 2401 mod 15 = 1 |
| **5** | 7 | Repeats again like x=1 |
| **6** | 4 | Repeats like x=2 |
| **7** | 13 | Repeats like x=3 |
| **8** | 1 | Repeats like x=0 |

N=15, a=7 , $f(x)=7^x \bmod 15$ , r =?

The values repeat every **4 steps**:

1→7→4→13→1→7→4→13→1

# Shor's Algorithm – Step 3

Step 3: Use the Period to Find Factors

Once you have the period r , here are two checks:

- If r is **even**, and
- $a^{r/2} \neq 1 \bmod N$
- $\gcd(a^{r/2} - 1, N)$ and $\gcd(a^{r/2} + 1, N)$

These give you **non-trivial factors of N**

$N=15$, $a=7$ , $f(x)=7^x \bmod 15$ , $r = 4$

$\gcd(7^2 - 1, 15) = \gcd(49 - 1, 15) = \gcd(48,15) = 3$

$\gcd(7^2 + 1, 15) = \gcd(49+1,15) = \gcd(50,15) = 5$

the factors: **3 and 5**

# Shor's Algorithm – for Number 221

Step1: Choose a Random Number a = 38 , 1 < a < 221

a is coprime to 221 → that is, gcd($a$,221) = 1, gcd(a,221)=1

Step2: Period Finding r

f(x) = $38^x$ mod 221

| x | $38^x$ mod221 | Calculation |
|---|---|---|
| **0** | 1 | $38^0$ mod 221 = 1 |
| **1** | 38 | $38^1$mod 221 = 38 |
| **2** | 116 | $38^2$ mod 221 = 1444 mod 221 = 116 |
| **3** | 211 | $38^3$ mod 221 = 5472 mod 221 = 211 |
| **4** | 36 | $38^4$ mod221= 200936 mod 221 = 36 |
| **5** | 38 | Repeats again like x=1 |
| **6** | 116 | Repeats like x=2 |
| **7** | 211 | Repeats like x=3 |
| **8** | 36 | Repeats like x=0 |

r = 4 , $38^4$ mod 221 = 36 , $38^8$ mod 221 = 36

N=221, a=38 , f(x)=$38^x$ mod 221 , r = 4

gcd($38^{r/2}$−1,221) = gcd($38^2$−1,221) = gcd(116−1,221) = gcd(115,221) = 17

gcd($38^{r/2}$+1,221) = gcd($38^2$+1,221) = gcd(116+1,221) = gcd(117,221) = 13

the factors: **13 and 17**

# Shor's Algorithm

Quantum Part: Find the period $r$. We use two quantum registers:

| Register | Size | Purpose |
|---|---|---|
| 1st | ( n_1 ) qubits | Store values of ( x ) |
| 2nd | ( n_2 ) qubits | Store values of ( f(x) = $a^x$ mod N ) |

**Step 1: Initialize**

$$|\psi_0\rangle = |0\rangle\,|0\rangle$$

**Step 2: Apply Hadamard to 1st register**

This creates a superposition of all $x$:

$$|\psi_1\rangle = \frac{1}{\sqrt{Q}}\sum_{x=0}^{Q-1}|x\rangle\,|0\rangle$$

where $Q = 2^{n-1}$ (choose $Q > N^2$)

**Step 3: Compute** $f(x) = a^x \bmod N$

Use a quantum modular exponentiation circuit:

$$|\psi_2\rangle = \frac{1}{\sqrt{Q}}\sum_{x=0}^{Q-1}|x\rangle\,|f(x)\rangle$$

This entangles both registers.

**Step 4: Measure the 2nd register**

Suppose the measurement gives some value $y = f(x_0)$.

Then the state collapses to:

$$|\psi_3\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |x_0 + kr\rangle |y\rangle$$

—only those $x$that give the same $f(x_0)$remain.

**Step 5: Apply Quantum Fourier Transform (QFT) to the 1st register**

The QFT converts the periodic structure into **peaks** in the frequency domain.

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \frac{1}{\sqrt{Q}} \sum_{c=0}^{Q-1} e^{2\pi i c(x_0+kr)/Q} |c\rangle$$

This gives constructive interference for values of $c$close to multiples of $Q/r$.

**Step 6: Measure 1st register**

Measurement gives a value $c$that is approximately:

$$c \approx \frac{jQ}{r}$$

where $j$is an integer.

**Step 7: Classical Post-Processing**

From the measured $c$and known $Q$, use **continued fractions** to find a rational approximation of $c/Q$.

The denominator gives the period $r$.

# Shor's Algorithm

## Quantum Advantage

| Method | Time Complexity |
| --- | --- |
| Classical factoring | $O\left(e^{(\log N)^{1/3}}\right)$ |
| Shor's Algorithm | $O\left((\log N)^3\right)$ |

ThankQ