**Probabilistic Versus Quantum Algorithms**

**1. Introduction**
Algorithms can be broadly classified based on how they handle uncertainty and computation. Two important classes are **probabilistic algorithms** and **quantum algorithms**. Both aim to solve problems more efficiently than classical deterministic algorithms, but they do so using different principles.

**2. Probabilistic Algorithms**
Probabilistic algorithms use randomness to make decisions during computation. They do not always guarantee a correct answer but can give the correct result with high probability.

- **Example**: The **Monte Carlo algorithm** estimates $\pi$ by random sampling.
- **Working**: Random inputs are generated, and outcomes are analyzed statistically to produce an approximate solution.
- **Time Complexity**: Often faster than deterministic algorithms, especially for problems like primality testing or optimization.
- **Advantage**: Simpler and faster for certain problems where exact computation is expensive.
- **Limitation**: There is always a small chance of error, which may require repeated runs to increase accuracy.

**3. Quantum Algorithms**
Quantum algorithms leverage the principles of **quantum mechanics**: superposition, entanglement, and interference. They can explore many possible solutions simultaneously, achieving exponential speedups for certain problems.

- **Example**: **Shor's Algorithm** for factoring large integers.
- **Working**: Quantum superposition allows simultaneous evaluation of a function for many inputs. Quantum Fourier Transform (QFT) helps find periodicity efficiently.
- **Time Complexity**: Solves problems in **polynomial time** that would take **exponential time** classically.
- **Advantage**: Can solve specific hard problems like integer factorization and unstructured search much faster than classical or probabilistic algorithms.
- **Limitation**: Requires a quantum computer; practical implementation is still in early stages.

**4. Comparison**

| **Feature** | **Probabilistic Algorithms** | **Quantum Algorithms** |
|---|---|---|
| Principle | Uses randomness | Uses quantum superposition & entanglement |
| Accuracy | High probability, not always exact | Exact or high-probability solutions |
| Example Problem | Primality testing, Monte Carlo | Shor's Algorithm, Grover's Algorithm |
| Speed | Faster than classical in some cases | Exponentially faster for specific problems |
| Hardware Requirement | Classical computer | Quantum computer |

**5. Applications**

- **Probabilistic Algorithms**: Simulation, randomized optimization, cryptography, approximate counting.
- **Quantum Algorithms**: Breaking RSA encryption, unstructured search, quantum chemistry simulations, optimization problems.

**Phase Kick-Back – Explanation**

**Phase Kick-Back** is a **quantum phenomenon** where the **phase of a qubit** in a **control register** is modified (or "kicked back") as a result of applying a **controlled unitary operation** on a target qubit.

- Instead of changing the **state of the target qubit**, the **control qubit's phase** is altered.
- This is extremely useful in algorithms that encode information in the **phase** rather than the amplitude of a qubit.

**2. How it Works**

Suppose we have:

- **Control qubit:** $|x\rangle$
- **Target qubit:** $|y\rangle$
- **Unitary operation U** such that $U|y\rangle = e^{i\phi}|y\rangle$

A **controlled-U operation** applies $U$only if the control qubit is $|1\rangle$:

$$C_U \mid x\rangle \mid y\rangle = \mid x\rangle U^x \mid y\rangle$$

- If the target qubit is prepared in a special state, e.g., $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$, then applying controlled-U can **transfer the phase** from the target qubit to the control qubit:

$$C_U \mid x\rangle \mid -\rangle = e^{ix\phi} \mid x\rangle \mid -\rangle$$

✅ The **phase is "kicked back"** to the control qubit, without altering the target qubit.

**3. Example**

**Simple example:**

- Let target qubit = $\mid -\rangle = (\mid 0\rangle - \mid 1\rangle)/\sqrt{2}$
- Controlled-Z gate $CZ$is applied:

$$CZ \mid x\rangle \mid y\rangle = \begin{cases} -\mid x\rangle \mid y\rangle & \text{if } x = y = 1 \\ \mid x\rangle \mid y\rangle & \text{otherwise} \end{cases}$$

- Applying $CZ$when target is $\mid -\rangle \rightarrow$ control qubit picks up **phase −1** if it is $\mid 1\rangle$.
- Target remains unchanged: $\mid -\rangle$

This is **phase kick-back in action**.

**4. Importance in Quantum Algorithms**

Phase kick-back is crucial because it allows **encoding function values into the phase** of a qubit without measuring it.

**Examples:**

1. **Deutsch–Jozsa Algorithm**:
   - Oracle flips the phase of the input qubit according to $f(x)$
   - Uses phase kick-back to encode f(x) as a **phase** in the superposition
2. **Simon's Algorithm**:
   - Encodes hidden XOR function into phases of input qubits
3. **Shor's Algorithm**:
   - Quantum Fourier Transform and modular exponentiation use phase kick-back to find **periods efficiently**

**Deutsch–Jozsa Algorithm – Explanation**

**1. Introduction**

- Developed by **David Deutsch and Richard Jozsa (1992–1993)**.
- Solves a **black-box problem** (oracle problem) **deterministically** using a **quantum computer**.
- Shows that quantum computers can solve some problems in **one query**, whereas classical computers may need **exponentially many queries**.

**2. Problem Definition**

- A function $f: \{0,1\}^n \rightarrow \{0,1\}$implemented as a **black-box oracle**.

**Promise:**

- $f$is either:
  1. **Constant:** $f(x) = 0$or $f(x) = 1$for all $x$
  2. **Balanced:** $f(x) = 0$for exactly half of the inputs, $f(x) = 1$for the other half

**Goal:** Determine whether $f$is **constant** or **balanced**.

**Classical solution:**

- In the worst case, need $2^{n-1} + 1$queries to guarantee correctness.

**Quantum solution:**

- Deutsch–Jozsa algorithm solves it **with just 1 query**.

**3. Steps of Deutsch–Jozsa Algorithm**

**Step 1: Initialize Quantum Registers**

- Use **n qubits for input** and 1 qubit for output.
- Initialize input qubits to $\mid 0\rangle^{\otimes n}$and output qubit to $\mid 1\rangle$:

$$\mid 0\rangle^{\otimes n} \mid 1\rangle$$

**Step 2: Apply Hadamard Transform**

- Apply **Hadamard gate (H)** to all input qubits and output qubit:

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

- Creates **superposition of all input states**.

**Step 3: Apply Oracle $U_f$**

- Oracle $U_f$ maps:

$$U_f \mid x\rangle \mid y\rangle = \mid x\rangle \mid y \oplus f(x)\rangle$$

- After applying oracle, the quantum state becomes:

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

- The output qubit is no longer important; **phase of input qubits encodes f(x)**.

**Step 4: Apply Hadamard Transform Again**

- Apply **Hadamard** to all input qubits again.
- This step performs **interference**, combining amplitudes constructively or destructively.

**Step 5: Measure Input Qubits**

- Measure the input qubits:
  - **All zeros** $\mid 0\rangle^{\otimes n} \rightarrow f$ is **constant**
  - **Any non-zero state** $\rightarrow f$ is **balanced**

✅ Deterministic result with **only one oracle query**.

**4. Example: n = 2**

Suppose $f(x)$ is **balanced**:

$$f(00) = 0, f(01) = 1, f(10) = 1, f(11) = 0$$

**Step 1:** Initialize state: $\mid 00\rangle \mid 1\rangle$

**Step 2:** Apply Hadamard → superposition:

$$\frac{1}{2}(\mid 00\rangle + \mid 01\rangle + \mid 10\rangle + \mid 11\rangle) \cdot \frac{\mid 0\rangle - \mid 1\rangle}{\sqrt{2}}$$

**Step 3:** Apply oracle → add phase $(-1)^{f(x)}$:

$$\frac{1}{2}(\mid 00\rangle - \mid 01\rangle - \mid 10\rangle + \mid 11\rangle)$$

**Step 4:** Apply Hadamard again → input qubits become **non-zero state** → f is **balanced**.

**5. Quantum Advantage**

- **Classical:** Worst case $2^{n-1} + 1$ queries
- **Quantum:** 1 query, deterministic result
- Demonstrates **quantum parallelism and interference**.

**Simon's Algorithm – Explanation**

Simon's Algorithm is a **quantum algorithm** developed by **Daniel Simon in 1994**.

- It solves **Simon's Problem**, which is designed to show that **quantum computers can be exponentially faster than classical computers** for certain problems.
- Simon's Algorithm was one of the earliest examples showing an **exponential speedup of quantum algorithms over classical ones**.

**2. Simon's Problem**

Given a **black-box (oracle) function** $f: \{0,1\}^n \rightarrow \{0,1\}^n$ with the property that:

1. There exists a secret string $s \in \{0,1\}^n$ such that:

$$f(x) = f(y) \iff y = x \oplus s$$

- Here, $\oplus$ is **bitwise XOR**.
- $s$ is **unknown**.

2. If $s = 0^n$, then $f$ is **1-to-1** (injective).
3. Otherwise, $f$ is **2-to-1**, meaning every output value has **exactly two inputs** mapping to it: $x$ and $x \oplus s$.

**Goal:** Find the secret string $s$.

- **Classical computers** require $O(2^{n/2})$ queries to the oracle.
- **Simon's quantum algorithm** solves it in **O(n) queries**, exponentially faster.

**3. Steps of Simon's Algorithm**

**Step 1: Initialize Quantum Registers**

- Use **two quantum registers**:
  1. Input register: $n$ qubits (for $x$)
  2. Output register: $n$ qubits (for $f(x)$)
- Initialize both registers to $\mid 0\rangle^{\otimes n}$.

**Step 2: Apply Hadamard Transform**

- Apply **Hadamard gates (H)** to all qubits in the **input register**, creating a **superposition of all possible inputs**:

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

**Step 3: Query the Oracle**

- Apply the **oracle** $U_f$ to map $|x\rangle |0\rangle \rightarrow |x\rangle |f(x)\rangle$
- Now, the quantum state encodes **all input-output pairs** in superposition.

**Step 4: Measure Output Register**

- Measure the **output register**.
- Measurement collapses the output to a specific value $f(x_0)$, leaving the **input register in a superposition of two states**:

$$|x_0\rangle + |x_0 \oplus s\rangle$$

- This is key: the input register now contains **information about the secret string** $s$.

**Step 5: Apply Hadamard to Input Register**

- Apply **Hadamard transform** to the **input register** again.
- This produces a superposition of all strings $y \in \{0,1\}^n$ such that:

$$y \cdot s = 0 \pmod 2$$

- Here, $y \cdot s$ is the **dot product modulo 2**.

**Step 6: Measure Input Register**

- Measure the **input register**, giving a random string $y$ such that $y \cdot s = 0$.
- Repeat Steps 1–6 **O(n) times** to collect **n independent equations**.

**Step 7: Solve Linear System**

- Solve the **linear system of equations modulo 2** to find the secret string $s$.

✅ Done! Simon's Algorithm finds $s$ **exponentially faster than classical methods**.

**4. Example (n=2)**

Suppose $n = 2$, and the secret string $s = 10$.

- Oracle maps:

$$f(00) = f(10) = 01, f(01) = f(11) = 11$$

**Step 1:** Input register in superposition:

$$|00\rangle + |01\rangle + |10\rangle + |11\rangle$$

**Step 2:** Apply oracle → entangled state:

$$|00\rangle |01\rangle + |01\rangle |11\rangle + |10\rangle |01\rangle + |11\rangle |11\rangle$$

**Step 3:** Measure output → e.g., $f(x) = 01$ → input register collapses to:

$$|00\rangle + |10\rangle$$

**Step 4:** Hadamard on input → get a string $y$ such that $y \cdot s = 0$

- Repeat and solve → find $s = 10$

**5. Quantum Advantage**

- **Classical approach:** Need $O(2^{n/2})$ oracle queries.
- **Quantum approach:** Only $O(n)$ queries needed → **exponential speedup**.
- Shows that **quantum computers can outperform classical computers** for some problems.

**Shor's Algorithm – Detailed Explanation**

Shor's Algorithm is a **quantum algorithm** developed by **Peter Shor in 1994** for **integer factorization**. It can factor a large number $N$ in **polynomial time**, which is exponentially faster than the best-known classical factoring algorithms (like trial division or the general number field sieve).

- Factoring large numbers is the **basis of RSA encryption**.
- On a quantum computer, Shor's Algorithm can break RSA encryption efficiently.
- Classical computers struggle because factoring large numbers requires **exponential time**, whereas Shor's Algorithm does it in **O((log N)^3)** operations.

**2. Mathematical Foundation**

Shor's Algorithm is based on the idea that **factoring can be reduced to period finding**:

1. Suppose $N$ is the number we want to factor.
2. Pick a number $a < N$ such that gcd(a, N) = 1.
3. Consider the function:

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

This function is **periodic**, meaning there exists a smallest positive integer $r$ such that:

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

- Finding this period $r$ is **the hardest part classically**, but quantum computers can do it efficiently using **quantum Fourier transform (QFT)**.
- Once the period $r$ is known, the **factors of $N$ can be computed using gcd(a^(r/2) ± 1, N))**, provided $r$ is even.

**3. Detailed Steps**

**Step 1: Random Selection of $a$**

- Choose a random number $a$ less than $N$.
- If gcd(a, N) ≠ 1, then we already have a factor.
- Otherwise, proceed to find the period.

**Step 2: Quantum Period Finding**

- Construct a quantum superposition of all possible values of $x$.
- Compute $f(x) = a^x \bmod N$ for all $x$ **simultaneously** using quantum parallelism.
- Apply **quantum Fourier transform (QFT)** to find the period $r$.

**Step 3: Classical Post-processing**

- Check if $r$ is even. If not, choose another $a$.
- If $r$ is even, compute:

$$\text{factor}_1 = \gcd\left(a^{r/2} - 1, N\right) \quad \text{factor}_2 = \gcd\left(a^{r/2} + 1, N\right)$$

- These usually give **non-trivial factors of $N$**.

**4. Example: Factor N = 15**

Let's see a small example:

- Pick $a = 2$ (gcd(2,15) = 1).
- Compute $f(x) = 2^x \bmod 15$:

**x 0 1 2 3 4**

2^x mod 15 1 2 4 8 1

- Period $r = 4$ (smallest x where 2^x ≡ 1 mod 15)
- $r$ is even → compute gcd(2^(4/2) ± 1, 15) = gcd(4-1, 15), gcd(4+1, 15) = gcd(3,15), gcd(5,15)
- Factors = **3 and 5**

**5. Quantum Advantage**

- **Classical Approach:** Finding period $r$ is hard. For large $N$, classical algorithms take **exponential time**.
- **Quantum Approach:** Quantum superposition and interference allow the algorithm to **find the period efficiently**, in **polynomial time**.

**6. Applications and Importance**

1. **Breaking RSA Encryption:**
   - RSA security depends on factoring large semiprime numbers.
   - Shor's Algorithm can factor them efficiently, threatening RSA.
2. **Cryptography:**
   - The development of quantum-resistant algorithms is crucial because Shor's Algorithm can break current public-key cryptography.
3. **Number Theory and Quantum Computing:**
   - Shor's Algorithm demonstrates that quantum computing can solve certain **mathematical problems exponentially faster** than classical computing.

**Factoring Integers**

**1. Introduction**

Factoring integers is the process of breaking a number into smaller numbers (factors) that, when multiplied together, give the original number. It is a fundamental problem in **number theory** and has major applications in **cryptography**, especially in RSA encryption.

**2. Classical Factoring Methods**

Classical algorithms try to find factors using deterministic or probabilistic approaches:

- **Trial Division**: Check divisibility by all numbers up to √N. Simple but **slow for large numbers**.
- **Fermat's Method**: Express N as a difference of squares, useful when factors are close together.
- **Pollard's Rho Algorithm**: A probabilistic method that can find non-trivial factors faster than trial division.
- **Time Complexity**: For very large numbers, classical methods take **exponential time**, making them inefficient for cryptography.

**3. Quantum Factoring (Shor's Algorithm)**
Quantum algorithms can factor integers **efficiently** using quantum mechanics.

- **Shor's Algorithm**: Developed by Peter Shor (1994), it factors a number N in **polynomial time**.
- **Key Idea**: Reduce factoring to **period finding**. For a number a<N such that gcd(a, N)=1, consider f(x) = a^x mod N. The smallest period r such that a^r ≡ 1 mod N is used to compute factors.
- **Steps**:
    1. Randomly choose a number a < N. If gcd(a, N) ≠ 1, a factor is found.
    2. Use **quantum superposition** and **Quantum Fourier Transform (QFT)** to find the period r.
    3. Compute gcd(a^(r/2) ± 1, N) to get non-trivial factors.
- **Advantage**: Exponentially faster than classical algorithms for large integers.

**4. Example (N = 15)**

- Pick a = 2, gcd(2,15) = 1.
- f(x) = 2^x mod 15 → sequence: 1, 2, 4, 8, 1 …
- Period r = 4 → factors = gcd(2^(4/2) ± 1, 15) = gcd(3,15), gcd(5,15) = 3 and 5.

**5. Importance**

- **Cryptography**: RSA encryption security relies on the difficulty of factoring large integers. Quantum factoring threatens current public-key systems.
- **Number Theory**: Studying factoring leads to deeper understanding of prime numbers and mathematical structures.

**Grover's Algorithm – Explanation**

**1. Introduction**

Grover's Algorithm, developed by **Lov Grover in 1996**, is a **quantum search algorithm** that can find a specific item in an **unsorted database** of size $N$in roughly $\mathbf{O}(\sqrt{\mathbf{N}})$operations.

- Classical search in an unsorted database requires $O(N)$steps.
- Grover's Algorithm achieves a **quadratic speedup**, which is significant for large $N$.

**Applications:**

- Searching large unsorted databases
- Breaking symmetric cryptography (e.g., brute-forcing keys)
- Solving combinatorial problems

**2. Problem Setup**

Given:

- An **unsorted database** of $N$elements.
- A **black-box (oracle) function** $f(x)$that outputs:

$$f(x) = \begin{cases} 1 & \text{if } x \text{ is the target item} \\ 0 & \text{otherwise} \end{cases}$$

**Goal:** Find $x$such that $f(x) = 1$.

**3. Steps of Grover's Algorithm**

**Step 1: Initialize Quantum State**

- Use $n$qubits to represent $N = 2^n$items.
- Initialize all qubits to $|0\rangle$.

$$|0\rangle^{\otimes n}$$

- Apply **Hadamard transform** to create **equal superposition** of all states:

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

**Step 2: Oracle (Phase Inversion)**

- Apply **oracle** $O$which flips the **phase of the target state** $|\ x_t\rangle$by multiplying it by -1:

$$O\ |\ x\rangle = \begin{cases} -|\ x\rangle & x = x_t \\ |\ x\rangle & x \neq x_t \end{cases}$$

- This marks the correct item in the superposition.

**Step 3: Amplify Target (Grover Diffusion Operator)**

- Apply the **Grover Diffusion Operator** $D$to amplify the probability amplitude of the target state:

$$D = 2\ |\ \psi_0\rangle\langle\psi_0\ |\ - I$$

- Intuition: Inverts the amplitudes about the **average amplitude**, increasing the target's probability.

**Step 4: Repeat Steps 2–3**

- Repeat the **Oracle + Diffusion** operation approximately:

$$R = \frac{\pi}{4}\sqrt{N} \text{ times}$$

- After these iterations, the **target state's probability** is very close to 1.

**Step 5: Measure**

- Measure the qubits.
- The result will most likely be the **target element** $x_t$.

**4. Example: Search in a 4-Item Database**

Suppose database: $[00,01,10,11]$and the target is $x_t = 10$.

**Step 1:** Initialize superposition:

$$|\ \psi_0\rangle = \frac{1}{2}(|\ 00\rangle + |\ 01\rangle + |\ 10\rangle + |\ 11\rangle)$$

**Step 2:** Oracle flips target phase:

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

**Step 3:** Apply Grover diffusion:

- Reflect about average amplitude → probability of $|\ 10\rangle$increases.

**Step 4:** Repeat (~1 iteration for N=4)

**Step 5:** Measure → result: $|\ 10\rangle$with high probability.

**5. Quantum Advantage**

- **Classical search:** O(N) queries to find the item.
- **Grover's search:** O(√N) queries.
- Quadratic speedup may seem modest compared to Shor's Algorithm (exponential speedup), but it is significant for **large unstructured datasets**.