**Quantum Computing and Artificial Intelligence**
**Certificate Programme IISc Bangalore**
**Due Date: June 20th 2025**

Total 54 points

# ASSIGNMENT 3

**Instructions**: The question paper has three questions:

- **Question 1: Bernstein-Vazirani algorithm**
  **Contains: 5 Questions .**
  **Total: 15 marks**

- **Question 2: : Quantum Fourier Transform**
  **Contains: 8 Questions .**
  **Total: 24 marks**

- **Part C: Simon algorithm**
  **Contains: 5 Questions.**
  **Total: 15 marks**

**Prepare the report and submit it along with the Python file.**

## Question-1

**Bernstein-Vazirani algorithm** [Total: $3 \times 5 = 15$]

### Part-A :Brief about Bernstein-Vazirani

The Bernstein–Vazirani (BV) algorithm is designed to determine a hidden $n$-bit string $s = (s_0, s_1, \ldots, s_{n-1}) \in \{0,1\}^n$ using a quantum oracle for a Boolean function $f_s : \{0,1\}^n \to \{0,1\}$ defined as:

$$f_s(x) = s \cdot x = \bigoplus_{i=0}^{n-1} s_i x_i \mod 2,$$

where $\cdot$ denotes bitwise inner product modulo 2.

### Algorithm Steps

1. Initialize an $n$-qubit register in the state $|0\rangle^{\otimes n}$.
2. Apply Hadamard gates to all qubits:
$$H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle .$$
3. Apply the phase oracle $U_f$, which acts as:
$$U_f |x\rangle = (-1)^{f_s(x)} |x\rangle = (-1)^{s\cdot x} |x\rangle .$$
The state becomes:
$$\frac{1}{\sqrt{2^n}} \sum_x (-1)^{s\cdot x} |x\rangle .$$
4. Apply Hadamard gates again:
$$H^{\otimes n} \left( \frac{1}{\sqrt{2^n}} \sum_x (-1)^{s\cdot x} |x\rangle \right) = |s\rangle .$$
5. Measure the $n$-qubit register to obtain the bit string $s$ with probability 1.

- **Quantum Query Complexity:** Only **1** query to $U_f$ suffices.

### Part-B :Programming Implementation (Python / Qiskit)

There are a total of five questions given below, where all you need to complete the code snippet in your 'ipynb' file.

1. **Dynamic secret bitstring:** Modify the existing tutorial so that the secret string $s$ can be input dynamically (of any length $n$). Update the circuit construction code to use the bits of $s$ in the oracle and verify that measuring the output qubits recovers $s$. Test with several random secrets and increasing $n$. This demonstrates the signature property that the quantum algorithm finds the $n$-bit secret in one shot (whereas a classical procedure needs $n$ queries)

2. **Alternative Oracle using Phase Kickback:** Replace the standard CNOT-based oracle with a phase oracle. Show that this "phase kickback" version yields the same measured result bits as the original (*i.e.* the secret string) after the final Hadamards. This tests the understanding of different oracles constructions that encode $s$ via phase instead of X-flips. Write your views on how this phase oracle will be beneficial as compared to the previous one(CNOT's one).

3. **Classical vs. quantum performance:**We implemented the classical strategy for finding the secret: query the oracle (simulate the function $f(x) = s \cdot x \bmod 2$) on each basis vector to recover one bit of s at a time. The code requires $n$ queries to learn all bits of s. Contrast this with the quantum algorithm's single query. Measure (or count) the number of operations or the time taken by both approaches for increasing $n$. Write a report to discuss the observed quantum speedup in terms of queries and actual run time.

4. **Circuit Metrics: Depth and Gate Count:** For several values of $n$ (*e.g.* 4, 6, 8), construct the B_V circuit and use QuantumCircuit.depth() and QuantumCircuit.count_ops() to obtain the circuit depth and the number of gates. Plot how the depth and total gate count scale with $n$. Discuss whether the depth grows linearly or faster with $n$, and how this impacts practical run time and identifies which operations (Hadamards or CNOTs) dominate the circuit complexity in your report.

5. **Advance variant of B-V algorithm:** Consider an oracle that hides two secret bit-strings (*i.e.* $f(x)$ returns two bits, each a dot product with a different secret). Describe and implement the modified circuit for your variant, and run experiments to see if the secrets can be recovered. Discuss any limitations or differences compared to the standard BV case in the report.

## Question-2

**Quantum Fourier Transform** [Total: $3 \times 8 = 24$]

### Part-A :Basic maths behind QFT

The quantum Fourier transform on $n$ qubits is defined as the transformation

$$|x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} e^{2\pi i xy/2^n} |y\rangle$$

where we identify $n$-bit strings and the integers they represent in binary. More generally, for any nonnegative integer $N$, we can define the quantum Fourier transform modulo $N$ as

$$|x\rangle \xrightarrow{F_N} \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i xy/N} |y\rangle$$

where the state space is $\mathbb{C}^N$, with orthonormal basis $\{|0\rangle, |1\rangle, \ldots, |N-1\rangle\}$. Let $P$ denote the unitary operation that adds 1 modulo $N$: for any $x \in \{0, 1, \ldots, N-1\}$, $P|x\rangle = |x + 1 \bmod N\rangle$.

1. Show that $F_N$ is a unitary transformation.
2. Show that the Fourier basis states are eigenvectors of $P$. What are their eigenvalues? (Equivalently, show that $F_N^{-1} P F_N$ is diagonal, and find its diagonal entries.)
3. Let $|\psi\rangle$ be a state of $n$ qubits. Show that if $P|\psi\rangle$ is measured in the Fourier basis (or equivalently, if we apply the inverse Fourier transform and then measure in the computational basis), the probabilities of all measurement outcomes are the same as if the state had been $|\psi\rangle$.

### Part-B: Complexity of FFT and QFT

The straight forward method for performing the discret fourier transform on $(N = 2^n)$ elements, based upon direct evaluation requires $\mathcal{O}(N^2)$ operations. The best classical algorithm, Fast Fourier Transform (FFT), which computes the discrete Fourier transform using $\mathcal{O}(N \log N)$ gates. The Quantum Fourier Transform (QFT), which is the quantum analogue of the discrete Fourier transform, can be implemented on a quantum computer using $\mathcal{O}(\log^2 N)$ gates. It requires exponentially more operations to compute the Fourier transform on a classical computer than it does to implement the quantum Fourier transform on a quantum computer.

1. Explain why the classical Fast Fourier Transform (FFT) requires $\mathcal{O}(N \log N)$ operations to compute the Fourier transform of an $N$-element vector.
2. Describe how the Quantum Fourier Transform (QFT) achieves a significantly lower complexity of quantum gates $\mathcal{O}(\log^2 N)$, and what assumptions about the input and output quantum states allow this speedup

### Part-C: Programming Implementation (Python / Qiskit)

...

1. Implement a Python function `qft(n)` that builds the Quantum Fourier Transform (QFT) circuit for $n$ qubits using Qiskit. Give a decomposition of the controlled-phase gate into single-qubit and CNOT gates.
2. Apply the QFT circuit on the basis state $|5\rangle$ for $n = 4$. Print the resulting quantum statevector and verify that the amplitudes match the expected Fourier coefficients.
3. Implement the inverse QFT and show that applying QFT followed by inverse QFT (`iqft`) recovers the original state for several test inputs.

## Question-3

**Simon algorithm** [Total: $3 \times 5 = 15$]

### Part-A :Basic overview and maths behind Simon algorithm

Given a function $f : \{0,1\}^n \to \{0,1\}^n$ that satisfies the property:

$$f(x) = f(y) \iff x = y \oplus s \quad \forall x, y \in \{0,1\}^n,$$

for some unknown string $s \in \{0,1\}^n$, the task is to determine the value of $s$.
There are two possibilities:

- If $s = 0^n$, then $f(x) = f(y) \iff x = y$, and the function is *one-to-one*: each input maps to a unique output.
- If $s \neq 0^n$, then for every $x$, $f(x) = f(x \oplus s)$, and the function is *two-to-one*: each output corresponds to exactly two inputs separated by $s$.

**Simon's Problem:** Given oracle access to the function $f$, determine whether $s = 0^n$ (i.e., whether $f$ is one-to-one) or $s \neq 0^n$, and if so, find the secret string $s$.

### Algorithm Steps

1. **Initialize:** Prepare $2n$ qubits in the state $|0\rangle^{\otimes n} |0\rangle^{\otimes n}$, then apply Hadamard gates to the first $n$ qubits:
$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |0\rangle$$
2. **Oracle Query:** Apply the oracle $U_f$, which maps $|x\rangle |0\rangle \mapsto |x\rangle |f(x)\rangle$, to obtain:
$$\frac{1}{\sqrt{2^n}} \sum_{x} |x\rangle |f(x)\rangle$$
3. **Measure the Second Register:** Measure the second register to obtain some output $f_0 = f(x_0)$. The state collapses based on the value of $f$:
    - If $f$ is **one-to-one** ($s = 0^n$), the measurement uniquely identifies $x_0$, and the first register collapses to:
    $$|x_0\rangle$$
    - If $f$ is **two-to-one** ($s \neq 0^n$), the measurement yields two pre-images $x_0$ and $x_0 \oplus s$, and the first register collapses to:
    $$\frac{1}{\sqrt{2}} \left(|x_0\rangle + |x_0 \oplus s\rangle\right)$$

4. **Apply Hadamard Transform:** Apply Hadamard gates $H^{\otimes n}$ to the first register.

   In the two-to-one case, this gives:

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} \left[(-1)^{x_0 \cdot y} + (-1)^{(x_0 \oplus s) \cdot y}\right] |y\rangle$$

   which simplifies to a superposition over those $y$ satisfying $y \cdot s = 0 \mod 2$, due to destructive interference when $y \cdot s = 1$.

   In the one-to-one case, this gives:

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} [(-1)^{x_0 \cdot y}] |y\rangle$$

5. **Measure the First Register:** In the two-to-one case, each measurement yields a bitstring $y\imath\{0,1\}^n$ such that $y \cdot s = 0$. In the one-to-one case, each measurement returns a random uniformly chosen bitstring $y \in \{0,1\}^n$.

6. **Repeated queries** Repeat this process to collect a set of linearly independent such equations.

7. **Post-Processing:** Solve the resulting system of linear equations to find the hidden string $s$.

8. **Decision Step:** If the only solution is $s = 0^n$, then $f$ is one-to-one. Otherwise, return the nonzero $s$ as the solution.

## Part-B: Classical vs quantum complexity

1. What is the query complexity of a classical algorithm to solve Simon's algorithm - on average, how many times do we need to call $f$ to check if it is two-to-one or one-to-one?

2. After applying the second Hadamard transform, we have the state

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} \left[(-1)^{x_0 \cdot y} + (-1)^{(x_0 \oplus s) \cdot y}\right] |y\rangle$$

   .

   Simplify this further and show how it becomes a superposition over $y$ satisfying $y \cdot s = 0 \mod 2$.

3. What is the query complexity of Simon's algorithm as described above - on average, how many times do we apply $U_f$? What is the speedup over the classical case?

## Part-C: Programming Implementation (Python / Qiskit)

1. Write a function that builds a Qiskit oracle circuit for Simon's function with a secret string $s$. The oracle acts on two registers (each $n$ qubits): input and output registers. It implements

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

   where f(x)=f(z) $\leftrightarrow x = z \oplus s$

```python
    def simon_oracle(n: int, s: str) -> QuantumCircuit:
        """
        Build a Simon oracle circuit for secret string s of length n.

        Args:
            n (int): Number of qubits in input/output registers.
            s (str): Secret binary string of length n, e.g. '110'.

        Returns:
            QuantumCircuit: Oracle circuit acting on 2n qubits.
        """
            oracle = QuantumCircuit(2*n)

        # YOUR CODE HERE: implement oracle that maps |x>|y> -> |x>|y XOR f(x)
        # Hint: Use CNOTs from input qubits to output qubits controlled by bits of s
            .

        return oracle
```

2. Use the oracle from question 1 to implement the quantum side of a single run of Simon's algorithm(steps 1-5) in qiskit for $n$ qubits and print the measured bitstring.

```python
    def simon_circuit_3bit(n: int, s: str) -> QuantumCircuit:
        """
        Build Simon's algorithm circuit using a 3-bit Simon oracle.

        Args:
            n (int): Number of input qubits
            s (str): Secret string, e.g., '000', '101'

        Returns:
            QuantumCircuit: Complete Simon circuit with measurement.
        """
        qc = QuantumCircuit(2*n)

        oracle = simon_oracle(n, s)

        # YOUR CODE HERE: implement oracle that implements steps 1 to 5

        return qc
```

3. **[OPTIONAL]** Write code to run simon's algorithm in full. Note: Algorithmically solving a set of linear equations is not easy, you can try to come up with your own algorithm or look into using the `sympy` or `galois` packages. This problem is **optional** as this is not a linear algebra course!