The supremacy of quantum algorithms
Lesson 1 – The Bernstein-Vazirani Algorithm

The Bernstein-Vazirani algorithm, developed in 1993, builds upon Deutsch's algorithm and provides another example of how quantum computing can solve certain problems more efficiently than classical computing.

In Lesson 1, you solved a task trying to discover a hidden binary string by asking Yes/No questions one at a time. Using a classical approach as you did, one needs to ask one question per bit to determine the entire string. This means that for a hidden string of n bits, you required n queries in the worst case.

However, quantum computing leverages the Bernstein-Vazirani algorithm to find the hidden string in just one query, regardless of how many bits it contains: Yes, it reduces n queries to a single one.

In this lesson, you will first explore how this quantum speedup is achieved using an example where the hidden string consists of three bits. Then, you will generalize the concept to n bits.

The Problem

Imagine we have a mystery box represented by a function f_a. This box is special because when you give a binary number with n bits, it returns either 0 or 1. The function f_a is defined using a fixed secret binary number $a = (a_1, a_2, \ldots, a_n)$, where each a_i is either 0 or 1.

To compute $f_a(x)$, the function follows these steps[1]:

- Take a n-bit input $x = (x_1, x_2, \ldots, x_n)$, where each x_i is 0 or 1.
- Multiply each input bit by the corresponding bit in a: $a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_n \cdot x_n$
- Check the result of the sum:
  - If the sum is even, the function outputs 0.
  - If the sum is odd, the function outputs 1.

Task 1 Use the hidden binary number $\boldsymbol{a = (1, 0, 1)}$ for the case $\boldsymbol{n = 3}$ and complete the following table by computing $\boldsymbol{f_a(x)}$.

Input x	$a \cdot x = a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot x_3$	Odd	Even	Output $f_a(x)$
$(0,0,0)$	$1 \cdot 0 + 0 \cdot 0 + 1 \cdot 0 = 0$		x	0
$(0,0,1)$	$1 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 = 1$	x		1
$(0,1,0)$				
$(0,1,1)$				
$(1,0,0)$				
$(1,0,1)$				
$(1,1,0)$				
$(1,1,1)$				

[1] This type of addition is called addition modulo 2. It means: if the total number of 1s is even, the result is 0; if it is odd, the result is 1.

Understanding the challenge

Our goal is to discover the secret binary string a used by the function $f_a(x)$. To achieve this, we can ask the function questions by giving it different binary strings $x = (x_1, x_2, \ldots, x_n)$ and observing what it returns.

Classical Solution

Task 2 Determine the minimum number of questions you need to ask to fully uncover the binary string $a = (a_1, a_2, a_3)$. Justify your answer.

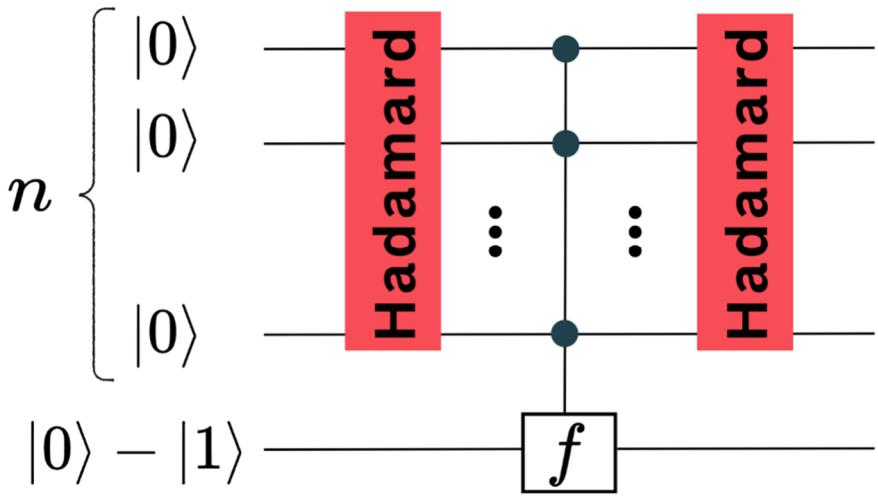
Hint: Think back to the task you solved in Lesson 1, where you had to find a hidden binary string by asking Yes/No questions. How is that related to the problem we are facing now?

Task 3 Now, consider the case where the binary string has n bits: $a = (a_1, a_2, \ldots, a_n)$.

Determine the number of questions needed in this general case. Justify your answer.

Quantum Solution: The Bernstein-Vazirani Algorithm

Instead of querying the function multiple times as in the classical method, the Bernstein–Vazirani algorithm extracts the entire hidden string in a single query using the circuit below:



It follows these steps:

Step I: Creating superposition

Prepare n qubis in state $|0\rangle$ in the first register. Apply a Hadamard gate to all of them. This creates a superposition of all possible states on the n qubits:

$$H|0\rangle H|0\rangle \cdots H|0\rangle = \frac{1}{\sqrt{2^n}}(\ |00\cdots0\rangle + |00\cdots1\rangle + \cdots + |11\cdots1\rangle\ )$$

Step II: Applying the Function f_a and Introducing Phases

Apply the function f_a to the obtained state in Step I, using the second register. This operation adds a phase factor of either $(-1)^0$ or $(-1)^1$ to each term of the superposition:

$$\frac{1}{\sqrt{2^n}}\left[\ (-1)^{f_a(00\cdots0)}|00\cdots0\rangle + (-1)^{f_a(00\cdots1)}|00\cdots1\rangle + \cdots + (-1)^{f_a(11\cdots1)}|11\cdots1\rangle\ \right]$$

Step III: Applying Hadamard Gates

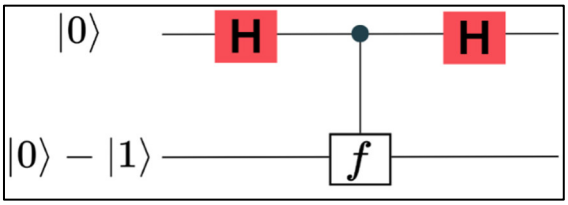
Apply a Hadamard gate to each qubit in the first register:

$$\frac{1}{\sqrt{2^n}}\left[\ (-1)^{f_a(00\cdots0)}H|0\rangle H|0\rangle\cdots H|0\rangle + \ (-1)^{f_a(00\cdots1)}H|0\rangle H|0\rangle\cdots H|1\rangle + \cdots\ (-1)^{f_a(11\cdots1)}H|1\rangle H|1\rangle\cdots H|1\rangle\right]$$

Note: The fact that the function introduces these phase shifts in the first register without affecting the second register directly is a fundamental concept in quantum computing. If you would like to explore in detail why and how these phase shifts occur, check the Quantum Function Evaluation worksheet.

Final Measurement: Measuring the first register now reveals a with 100% probability. Unlike the classical case, where multiple queries are required, the quantum algorithm finds a in one query.
After measurement, we obtain $|a\rangle = |a_1 a_2 \cdots a_n\rangle$.

Task 4 Compare the circuit used in the Deutsch Algorithm (see the figure on the right) with the circuit used in the Bernstein-Vazirani Algorithm and answer the following questions:



Explain the similarities and differences between the two algorithms. How does the Deutsch Algorithm relate to the Bernstein-Vazirani Algorithm?

Similarities	Differences