

# Quantum Computing

Ace Chun

July 22, 2022

## Contents

<b>1</b>	<b>Quantum Concepts</b>	<b>3</b>
1.1	Locality . . . . .	3
1.2	EPR Paradox . . . . .	4
1.3	Bell's theorem . . . . .	4
1.4	No-cloning Theorem . . . . .	4
<b>2</b>	<b>Qubits</b>	<b>6</b>
2.1	Statevectors . . . . .	6
2.2	Dirac (Bra-Ket) Notation . . . . .	7
2.3	Tensor Product . . . . .	8
2.4	Bloch Sphere . . . . .	8
2.5	Amplitude . . . . .	9
2.6	Phase . . . . .	9
<b>3</b>	<b>Single-Qubit Interactions and Gates</b>	<b>10</b>
3.1	Fundamental gates . . . . .	10
3.1.1	Identity Gate . . . . .	10
3.1.2	X Gate . . . . .	10
3.1.3	Z Gate . . . . .	11
3.1.4	H Gate . . . . .	11
3.2	Standard Gates . . . . .	11
3.2.1	Y gate . . . . .	11
3.2.2	S Gate . . . . .	11
3.2.3	T Gate . . . . .	12
3.3	Rotation Gates . . . . .	12

3.3.1	Rx Gate . . . . .	12
3.3.2	Ry Gate . . . . .	12
3.3.3	Rz Gate . . . . .	12
<b>4</b>	<b>Entanglement and Multi-Qubit Gates</b>	<b>13</b>
4.1	Qubit Registers . . . . .	13
4.2	Entanglement . . . . .	13
4.3	Gates . . . . .	13
4.3.1	CNOT Gate . . . . .	13
4.3.2	SWAP Gate . . . . .	14
4.3.3	CCNOT/Toffoli Gate . . . . .	14
4.3.4	CZ Gate . . . . .	14
4.3.5	0-Controlled Gates . . . . .	15
4.4	Disentanglement . . . . .	15
4.5	Bell States . . . . .	15
<b>5</b>	<b>Circuits</b>	<b>16</b>
5.1	Single Qubit . . . . .	16
5.2	Measurement . . . . .	16
5.3	Multi Qubit . . . . .	16
<b>6</b>	<b>Quantum Interference</b>	<b>17</b>
6.1	Hadamard Gate . . . . .	17
<b>7</b>	<b>Thought Experiments</b>	<b>17</b>
7.1	CHSH Game . . . . .	17
7.2	GHZ Game . . . . .	19
<b>8</b>	<b>Protocols</b>	<b>20</b>
8.1	Superdense Coding . . . . .	20
8.2	Quantum Communication without Entanglement . . . . .	21
8.2.1	Quantum Key Distribution . . . . .	21
8.2.2	Direct Communication . . . . .	22
<b>9</b>	<b>Algorithms</b>	<b>22</b>
9.1	Deutsch-Jozsa Algorithm . . . . .	22
9.1.1	2 input qubits . . . . .	23
9.1.2	Generalized circuit . . . . .	24
9.2	Grover's Algorithm . . . . .	24

9.2.1	Explanation at length . . . . .	25
9.2.2	Complexity Analysis . . . . .	28
9.2.3	Circuit . . . . .	31
9.3	Simon's Algorithm . . . . .	31
9.4	Quantum Fourier Transform . . . . .	33
9.4.1	Purpose . . . . .	34
9.4.2	Circuit . . . . .	35
9.5	Shor's Algorithm . . . . .	35
9.5.1	Modular Exponentiation . . . . .	36
9.5.2	Periodicity of Modular Exponentiation . . . . .	38
9.5.3	Aggregating Factors . . . . .	41
9.5.4	Summary . . . . .	42
9.5.5	Relevance to Factorization . . . . .	43
9.6	Bernstein-Vazirani Algorithm . . . . .	45
9.7	Quantum Clustering Algorithm . . . . .	48
9.7.1	K-means Algorithm . . . . .	48
9.7.2	Q-means Algorithm . . . . .	48
<b>10</b>	<b>Error Correction</b>	<b>49</b>
10.1	Bit-flip Error Correction . . . . .	49
10.2	Steane's Error Correction Code . . . . .	50
<b>11</b>	<b>Other</b>	<b>53</b>
11.1	Constructing the H gate . . . . .	53

# 1 Quantum Concepts

## 1.1 Locality

The concept of locality comes from the field theories of classical mechanics; the principle of locality states that an object can only be influenced directly by its own surroundings (or, there cannot be instantaneous action at a distance). For one point to have an influence on a point at some other location, some space between the two points must be a medium for the action; a wave or a particle must physically travel through space between the two points to carry the influence. In the quantum space, the argument is that: should locality exist, quantum states from distances greater than the speed of light should not be able to be instantaneously affected by each other. Based on

conducted experiences, it seems that this argument is not true, since quantum states can affect (be entangled with) each other at distances. Thus, quantum mechanics must be nonlocal for theory to match up with reality.

## 1.2 EPR Paradox

In 1935, Einstein, Podolsky, and Rosen published a paper that contained what is now known as the EPR paradox, which suggests that there is a paradox hiding in the predictions for measurement outcomes of entangled quantum objects. Given an experiment with which the outcome is known before the experiment takes place, there's an 'element of reality' associated with the experiment that determines the outcome of the measurement. These 'elements of reality' should be local; they should belong to a coordinate in spacetime which cannot be changed by something that moves faster than the speed of light. A choice of measurement done on a particle should not lead to two different quantum states for another particle that is far away.

## 1.3 Bell's theorem

In 1964, Bell created a rigorous test, known as Bell's inequality or Bell's theorem. It states that local theories about quantum mechanics will have a bound on correlations between measurements of the z-component of spin for entangled pairs of spin-half particles, or fermions. Since nonlocal theories can violate this bound, the question of local versus nonlocal can be solved if the inequality can be broken.

## 1.4 No-cloning Theorem

The no-cloning theorem states that any quantum state  $|\Psi\rangle$  cannot be duplicated with a single unitary gate. In other words,

$$U(|\Psi\rangle \otimes |0\rangle) \Rightarrow |\Psi\rangle \otimes |\Psi\rangle$$

cannot exist. The simplest proof of this would be to check if it works for the most basic of situations: for example, let's try duplicating the states  $|0\rangle$ ,  $|1\rangle$  and  $|+\rangle$ . Suppose that such a gate does exist, and that it can successfully duplicate a specific quantum state. Then, it should be able to clone the basic

states. Let's assume that  $U|0\rangle \otimes |0\rangle = |0\rangle \otimes |0\rangle$ , and  $U|1\rangle \otimes |0\rangle = |1\rangle \otimes |1\rangle$ .  
Or:

$$U \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$U \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

By linearity, this should extend to the  $|+\rangle$  state:

$$U|+\rangle \otimes |0\rangle \Rightarrow U \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle$$

However, given the previously assumed states, we end up with a result:

$$\frac{1}{\sqrt{2}}(U(|0\rangle \otimes |0\rangle) + U(|1\rangle \otimes |1\rangle))$$

$$\frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)$$

which is not mathematically equivalent to what we should have received as an output:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

Thus, by contradiction, such a gate  $U$  cannot exist because it cannot correctly duplicate a basic state.

We can try another variation of this problem, considering global phase. Suppose that there is a gate,  $V$ , which can perform the following:

$$V(|\Psi\rangle \otimes |0\rangle) \Rightarrow e^{i\alpha} |\Psi\rangle \otimes |\Psi\rangle$$

Again, we can try to assume that

$$V|0\rangle \otimes |0\rangle = |00\rangle$$

$$V|1\rangle \otimes |0\rangle = -|11\rangle$$

We can, again, try to duplicate  $|+\rangle$  with these assumptions.

$$\begin{aligned} V|+\rangle \otimes |0\rangle &= \frac{1}{\sqrt{2}}V(|0\rangle + |1\rangle) \otimes |0\rangle \\ &= \frac{1}{\sqrt{2}}(V|00\rangle + V|10\rangle) \\ &= \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \end{aligned}$$

Again, we run into the same problem; the above result, obtained through algebraic laws, is not mathematically equivalent to our expected answer of  $|++\rangle$ , or

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

Despite the consideration to global phase, such a unitary gate fails to exist.

## 2 Qubits

A qubit is the quantum analogue of the classical binary bit. Just like classical computers, quantum computers operate by way of manipulating these units of information through quantum gates. However, most of the similarities stop there; a quantum computer, for example, does not have irreversible operations (aside from measurement), as opposed to a classical computer, which has operations like SET and CLEAR (which are not time-reversible). A quantum computer's operations are always reversible because they consist of unitary operators.

### 2.1 Statevectors

The state of a particular quantum computer is represented with a statevector. A statevector is represented as a single column vector, describing the probability that the state would collapse into any of its positions. For example,

$$|\Psi\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

indicates that  $|\Psi\rangle$  has a 100% chance of being in the third state. There are two definitive states that a measured qubit can take: the 0 state and the 1 state.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

When a qubit in superposition is measured, it collapses into either one of the aforementioned states. Since the two states form an orthonormal basis, it is possible to form any state with a combination of them. Thus, any qubit can be expressed in the forms:

$$|\Psi\rangle = a|0\rangle + b|1\rangle = \begin{bmatrix} a \\ b \end{bmatrix}$$

When the function collapses, the probabilities that the qubit will fall into either the 0 or the 1 is its coefficient squared. Thus, this presents the slight caveat that  $a^2 + b^2 = 1$ , since probabilities fall between 0 and 1 and must always add up to 1. This is known as normalization.

## 2.2 Dirac (Bra-Ket) Notation

Introduced by Paul Dirac, Bra-Ket notation is a specific and convenient way of expressing vectors in quantum mechanics. A Ket ( $|\Psi\rangle$ ) represents a quantum state, a statevector consisting of complex numbers. A Ket is always matched with a Bra; they are each other's conjugate transpose. For example, the ket

$$|a\rangle = \begin{bmatrix} 2 - 3i \\ 6 + 4i \\ 3 - i \end{bmatrix}$$

matches with the bra

$$\langle a| = [2 + 3i \quad 6 - 4i \quad 3 + i]$$

Inner multiplication with bras and kets are notated with

$$\langle a|b\rangle$$

where the product is the dot product of the two vectors. This inner product is contrasted with the outer product of a bra and a ket:

$$|b\rangle\langle a|$$

which returns an  $N \times N$  matrix as a result.

$$|b\rangle\langle a| = \begin{bmatrix} b_0a_0 & b_0a_1 & \cdots & b_0a_{n-1} \\ b_1a_0 & b_1a_1 & \cdots & b_1a_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n-1}a_0 & b_{n-1}a_1 & \cdots & b_{n-1}a_{n-1} \end{bmatrix}$$

In general, kets in quantum computing are simply used to express the states of a system.

## 2.3 Tensor Product

The tensor product ( $\otimes$ ) is an operation between two matrices or vectors that outputs a single object as the result. When two vectors are tensored, the second vector of the two is scalar-multiplied with each of the elements in the first vector. Generally:

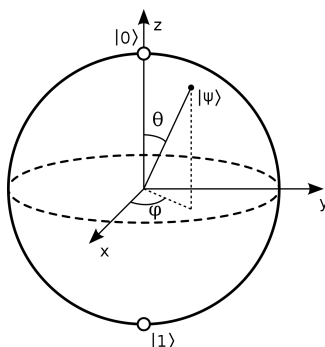
$$\begin{aligned} |x\rangle \otimes |y\rangle &= \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \otimes \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \\ &= \begin{bmatrix} x_0 \cdot \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \\ x_1 \cdot \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \end{bmatrix} \end{aligned}$$

Tensor products are associative ( $|x\rangle \otimes (|y\rangle \otimes |z\rangle) = (|x\rangle \otimes |y\rangle) \otimes |z\rangle$ ) and are distributive over a sum.

## 2.4 Bloch Sphere

A Bloch sphere is a visual representation of the states of a qubit. By convention, the  $|0\rangle$  state is at the very top of the represented circle, while  $|1\rangle$  is at the very bottom. A vector points outwards to its position on the sphere, which represents the state of the qubit.





We can define a statevector based on the angles  $\theta$  and  $\varphi$ :

$$|\psi\rangle = \begin{bmatrix} \cos(\frac{\theta}{2}) \\ e^{i\varphi} \sin(\frac{\theta}{2}) \end{bmatrix}$$

## 2.5 Amplitude

As stated before, qubits must always be unit vectors; the amplitudes, or the squares of the coefficients of the  $|0\rangle$  and  $|1\rangle$  states, must add up to 1. This means that amplitudes can be any number  $a|a \in \mathbb{C}$ .

Certain properties intrinsic to quantum computing prevent people from directly using the amplitudes of states to hold information. When the qubit is measured, the function collapses, and we are left with either the  $|0\rangle$  or  $|1\rangle$  state, with the information stored inside of its amplitudes destroyed in the process.

## 2.6 Phase

In squaring the amplitudes when measuring a qubit, one might find that the signs on the amplitudes are useless. However, the signs of the amplitudes are crucial when considering the state of the qubit prior to measurement; the difference in signs is called phase. There are two types of phase: relative and global. For example, take two qubits:

$$\begin{aligned} |x\rangle &= \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \\ |y\rangle &= -\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \end{aligned}$$

Notice how we can factor out  $-1$  from the expression of  $|y\rangle$ . Thus,  $|y\rangle$  has a global phase of  $-1$ , and it is functionally equivalent to  $|x\rangle$ . However, in cases where a negative factor cannot be singled out from the expression:

$$\begin{aligned}|x\rangle &= \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \\ |y\rangle &= \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\end{aligned}$$

$|x\rangle$  and  $|y\rangle$  cannot be reduced to one another, and thus have a relative phase between them. These are two different states. Global phase should be a variant of the form  $e^{i\varphi}$ , where  $\varphi$  is any angle. In summary, global phases do not matter in quantum computing, while relative phases do due to their effects on the qubit's state prior to measurement.

## 3 Single-Qubit Interactions and Gates

In quantum computing, qubits are mutable; the states of the qubits themselves are altered. Single-Qubit, or unary gates act on the qubit and perform manipulations akin to rotating or reflecting a vector about a specific axis on the Bloch sphere. In quantum computing, gates are represented as matrices that are applied (matrix-multiplied) to the statevectors.

### 3.1 Fundamental gates

#### 3.1.1 Identity Gate

The Identity gate takes the form of a simple identity matrix: it does nothing to actually change the state of the qubit.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

#### 3.1.2 X Gate

The (Pauli) X gate is the quantum analogue for the classical NOT gate.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Notice how the application of the X gate essentially swaps the amplitudes of  $|0\rangle$  and  $|1\rangle$ . In essence, the X gate flips the qubits around the X axis of the Bloch sphere.

### 3.1.3 Z Gate

The Z gate is specific to quantum computing, as it dictates a property unknown within classical computers: phase. The Z gate flips the phase of the qubit, mirroring the qubits around the Z axis.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

### 3.1.4 H Gate

The Hadamard gate also addresses a property specific to quantum computing: superposition. The H gate essentially places a qubit within a fixed state into an equal superposition of the states  $|0\rangle$  and  $|1\rangle$ .

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Visually, the H gate mirrors qubits around the Hadamard axis, or the line defined by  $x = z$  on the Bloch sphere.

## 3.2 Standard Gates

### 3.2.1 Y gate

The Y gate mirrors qubits around the Y axis of the Bloch sphere.

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Thus, the Y gate functions like an X gate, only rotating with respect to a different axis.

### 3.2.2 S Gate

The S gate is defined to be the square root of the Z gate:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

It rotates qubits by  $\frac{\pi}{2}$  radians counterclockwise about the Z axis, rather than a full  $\pi$  radians. It will turn  $|+\rangle$  into  $|i+\rangle$ .

### 3.2.3 T Gate

The T gate is the square root of the S gate, or the fourth root of the Z gate.

$$T = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1+i}{\sqrt{2}} \end{bmatrix}$$

## 3.3 Rotation Gates

Arbitrary rotation gates allows movement about the entire Bloch sphere. These gates are not considered primitive, as they are constructed from other primitive gates and the CNOT gate.

### 3.3.1 Rx Gate

The  $R_x$  gate performs a rotation about the X axis of the Bloch sphere, or rather, moving qubits around the Y-Z plane.

$$R_x = \begin{bmatrix} \cos \frac{\theta}{2} & -i \cdot \sin \frac{\theta}{2} \\ -i \cdot \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$$

### 3.3.2 Ry Gate

The  $R_y$  gate performs a rotation about the Y axis.

$$R_y = \begin{bmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$$

### 3.3.3 Rz Gate

There are two different ways to describe the  $R_z$  gate.

$$R_z = \begin{bmatrix} e^{i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$$

$$R_\varphi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix}$$

The first matrix simply rotates the qubit and applies a global phase to it. However, it is not commonly used due to its application of a phase to the  $|0\rangle$  state. The second matrix rotates the qubit about the Z axis but is defined in terms of the phase angle,  $\varphi$ .

## 4 Entanglement and Multi-Qubit Gates

### 4.1 Qubit Registers

Within a quantum computer, an  $n$  qubit system has  $2^n$  possible measurement outcomes. During computation, the system can be in a superposition of any of these four states. Multi-qubit states are represented as the tensor product of each individual statevector. So

$$|+\rangle \otimes |0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

We may also express the above state as

$$\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$$

### 4.2 Entanglement

One can construct a system such that the state of one qubit depends on another. For example:

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

The implication here is that, when the system is measured, both qubits will either be in the state  $|0\rangle$  or  $|1\rangle$ , with a 50% probability of each. This means that the states of each qubit are dependent on each other. Such a system can be constructed with entanglement gates.

### 4.3 Gates

#### 4.3.1 CNOT Gate

The CNOT gate consists of a control qubit and a target qubit, in which the state of the target depends on the state of the control. If the control qubit is

in state  $|1\rangle$ , an X (NOT) gate is enacted on the target; if the control is  $|0\rangle$ , nothing happens as a result of the CNOT gate on the target.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

### 4.3.2 SWAP Gate

A SWAP gate swaps the states of the two qubits.

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For example, the state  $|01\rangle$  would change to  $|10\rangle$ .

### 4.3.3 CCNOT/Toffoli Gate

The Toffoli, or CCNOT is a doubly-controlled NOT gate. The target is flipped only if both controls are in a state  $|1\rangle$ . The Toffoli gate matrix is huge, as it must have 8 rows and columns:

$$CCNOT = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

### 4.3.4 CZ Gate

Theoretically, any gate can have a “controlled” form. For example, there exists a CZ gate, which changes phase when both the control and target are

in a state  $|1\rangle$ . This controlled Z gate is also known as a phase kickback.

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

#### 4.3.5 0-Controlled Gates

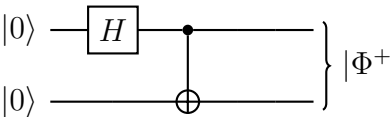
Just like a control gate acts on its target qubit when its controls are in the state  $|1\rangle$ , we may also configure a gate such that it acts on its target when its controls are  $|0\rangle$ . In languages where this is not a built-in functionality, we can just flip the control qubit to a  $|1\rangle$  state, apply the controlled gate, and then flip the qubit back to its original state.

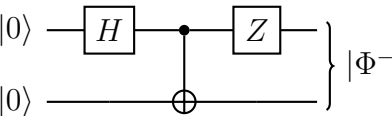
### 4.4 Disentanglement

Applying the CNOT gate to both entangled qubits once again can disentangle two qubits. This is because applying the CNOT once again cancels the “NOT” action on the target qubit.

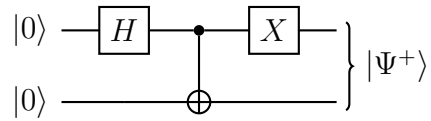
### 4.5 Bell States

Bell states are four different maximum entanglement states for a single pair of qubits.

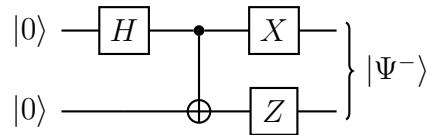
$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|00\rangle$$


$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|00\rangle$$


$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$$



$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle$$



## 5 Circuits

System circuits can be visualized as a series of wires with either multi-qubit or single-qubit operations.

### 5.1 Single Qubit

Single qubit gates are typically represented with a single box with the name of the gate within it. For example:



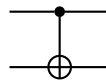
### 5.2 Measurement

Measurement is typically represented with a meter.



### 5.3 Multi Qubit

CNOT Gate

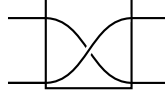


CZ Gate

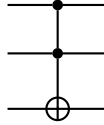




SWAP Gate



CCNOT Gate



## 6 Quantum Interference

Signals with the same frequency interfere with each other based on their phase. When two waves are in phase, constructive interference occurs, thus amplifying the wave; when the waves are out of phase, or have opposite phase, destructive interference occurs and cancels out the waves. Superposition terms with the same value are combined after a gate is applied.

### 6.1 Hadamard Gate

An example of interference is the application of the H gate twice.

$$\begin{aligned}
 H(|0\rangle) &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\
 H\left(\frac{1}{2}(|0\rangle + |1\rangle)\right) &= \frac{1}{\sqrt{2}} \left( \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) + \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \right) \\
 &= \frac{1}{2}(2|0\rangle + 0|1\rangle) \\
 &= |0\rangle
 \end{aligned}$$

## 7 Thought Experiments

### 7.1 CHSH Game

The CHSH game (Clauser, Horne, Shimony, Holt) is a test of Bell's theorem as expressed as a thought experiment or game. It is played with two parties (called Alice and Bob for tradition):

1. The referee sends a random question bit,  $x \in \{0, 1\}$  to Alice, and  $y \in \{0, 1\}$  to Bob.
2. Players have to respond with their own bit ( $a$  for Alice,  $b$  for Bob) without classical communication with the other player.
3. The players win if  $x \wedge y = a \oplus b$  (x AND y is equal to a XOR b)

In the truth table below, we can see what the conditions for  $a$ ,  $b$  have to be so that the players win:

$x$	$y$	$a \oplus b$
0	0	0
0	1	0
1	0	0
1	1	1

The best classical strategy would be for the players to always choose 1; since  $1 \oplus 1 = 0$ , and the result of  $a \wedge b$  is 0 75% of the time. This would be the optimal strategy if Alice and Bob were unable to violate locality, following the bound set by Bell's theorem. Thus, if there exists a more optimal solution with quantum operations, then local theories fail and Bell's theorem is violated. Instead of using bits, let's say Alice and Bob use qubits instead, answering the referee with the measurement results of their qubit. To start, we can create the Bell state  $|\Phi^+\rangle$ , or

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

The next part of the strategy is to define a new basis as a function of an angle  $\theta$ .

$$\begin{aligned}\phi_0(\theta) &= \cos \theta |0\rangle + \sin \theta |1\rangle \\ \phi_1(\theta) &= -\sin \theta |0\rangle + \cos \theta |1\rangle\end{aligned}$$

Recall that the arbitrary rotation gate  $R_y$  looks like:

$$\begin{bmatrix} \cos \frac{\theta}{2} & \sin \frac{\theta}{2} \\ -\sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$$

So, to change the basis to  $\phi_0(\theta)$ ,  $\phi_1(\theta)$  just involves applying the  $R_y$  gate with  $2\theta$  to something in the old basis.

$$R_y(2\theta)|\Psi_i\rangle = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix}$$

From setting this state up, we can now play the game. If Alice receives  $x = 1$ , she can measure her qubit in the basis of  $\theta = \frac{\pi}{8}$ . Otherwise, she can set her basis to  $\theta = -\frac{\pi}{8}$ . It's vice-versa for Bob; if he receives a 0, he can measure his qubit in the basis  $\theta = \frac{\pi}{8}$ , and set  $\theta = -\frac{\pi}{8}$  otherwise. This strategy will win the game 85% of the time, which surpasses the bound set by Bell's theorem. As it turns out, the 85% figure is the most optimal strategy in an environment with nonlocal theories, according to Tsirelson's bound.

## 7.2 GHZ Game

The GHZ game is similar in concept to the CHSH game, but it is played with three players. The question bits sent by the referee  $xyz$  are not random:

1. The referee chooses  $xyz$  uniformly from the set  $\{000, 011, 101, 110\}$
2. The winning condition for the game is  $x \vee y \vee z = a \otimes b \otimes c$ .

Again, a table can be constructed:

$xyz$	$a \otimes b \otimes c$
000	0
011	1
101	1
110	1

The most optimal strategy here would be, again, for all players to choose 1 every time: since 75% of the outcomes are of the output 1, and we know that  $1 \otimes 1 \otimes 1 = 0 \otimes 1 = 1$ , this will win 75% of the time. However, there is a more optimal solution with entangled qubits. Alice, Bob, and Charlie can construct the initial state in a register with their qubits:

$$|\psi\rangle = \frac{1}{2}(|000\rangle - |011\rangle - |101\rangle - |110\rangle)$$

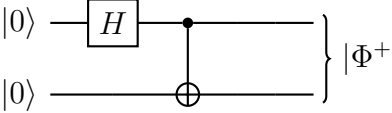
In the game, the players can apply an  $H$  gate to their qubit if they receive a 1 from the referee and return the measured outcome in the standard basis.

This game was formulated to show that, while impossible for a classical strategy, one can win the game every time with a quantum strategy. It works as a stronger proof for nonlocality than the CHSH game, because it is absolute while CHSH relies on probabilistic disagreements. We can see these effects by breaking this down into casework and applying each  $H$  gate conditionally. What we will see is that there is some constructive and destructive interference between states that eventually return four states, all of which have an odd number of qubits in the  $|1\rangle$  state. Since an odd number of ones XORed with each other return a 1 as a result, the players will always win with this strategy.

## 8 Protocols

### 8.1 Superdense Coding

We can encode information to a single qubit by using the reversibility of the Bell states. With all four Bell states, we have the standard operations (placing the qubit into a  $\Phi^+$  state):

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|00\rangle$$


We know that we may generate other Bell states by applying other gates, like  $X$  and  $Z$ . When we then decode the quantum state by applying the  $CNOT$  and  $H$  gates in reverse, there would be some “leftover breadcrumbs” depending on the Bell state that was first applied to the two qubits.

State	Gate	Result
$\Phi^+$	-	$ 00\rangle$
$\Psi^+$	$X$	$ 01\rangle$
$\Phi^-$	$Z$	$ 10\rangle$
$\Psi^-$	$XZ$	$ 11\rangle$

Superdense coding generates a secure channel. Imagine a situation where Alice is sending a message to Bob, where qubits were previously entangled

and put in the Bell state by a third-party satellite. Alice and Bob hold each hold one qubit of the two-qubit system. Thus, to disentangle the system and find out what the message is, one would need both Alice’s transmitted qubit and Bob’s qubit, which prevents an eavesdropper (Eve) from intercepting and decoding the message mid-transmission.

## 8.2 Quantum Communication without Entanglement

Quantum memory is very costly, and keeping entangled qubits coherent at large distances is challenging. These physical limitations motivate the search for an algorithm to encode qubits without using entanglement. On the hardware side of things, qubits can be sent as photons through fiber optic cables, sent through a beam splitter. By simply sending qubits through the fiber optic channel, Eve cannot “win” the game; when she observes, or measures the state of the qubit, the state collapses. However, if Eve knows that Alice is sending either a  $|+\rangle$  or  $|-\rangle$ , then she can measure with respect to the Hadamard basis, which would not collapse the superposition. Therefore, Alice must choose bases at random while encoding the channel to protect the message from Eve. The drawback to this is that Bob will also have no idea what basis Alice is sending each qubit in.

There are two protocols that implement these ideas.

### 8.2.1 Quantum Key Distribution

The idea behind QKD is that Alice and Bob have a shared private key. The BB84 protocol (Bennet-Brassard 1984) states that:

1. Alice randomly generates two n-bit strings, named  $t$  and  $y$ .
2. If  $y_i = 0$ , Alice sends a  $|t_i\rangle$ . If  $y_i = 1$ , then Alice sends  $H|t_i\rangle$ .
3. Bob randomly generates an n-bit string, named  $z$ . If  $z_i = 1$ , Bob applies an  $H$  gate and measures. Otherwise, he does not apply the  $H$  gate. He then records the measurements into a string  $r$ .
4. Alice and Bob share  $y$ ,  $z$ . They then throw out any of the bits  $t_i$ ,  $r_i$  if  $y_i \neq z_i$ .
5.  $t$ ,  $r$  should thus be identical. Sharing just a few of the bits will confirm whether or not Eve intercepted their communication or not; if

the bitstrings are inconsistent with each other, then Eve must have intervened and collapsed a superposition in the process, indicating that their channel is not safe.

You might notice that there are several ways that Eve gets away with intercepting the channel. She may choose the correct basis, or accidentally measure every bit the correct way each time; she ends up have a 75% chance of success in intercepting a single bit. However, her chance of success goes down for each bit checked between Alice and Bob in step 4. This way, QKD is as secure as the users want it to be. To reconcile some inconsistencies due to quantum error, there are two specific protocols: Cascade protocol, and Privacy amplification. In the Cascade protocol, Alice and Bob disclose some information in the form of parity checks to correct discrepancies. In Privacy amplification, Alice and Bob decrease the sizes of their keys by hashing, accounting for the information they shared. A major advantage of QKD is that the qubit transfer is purely one-sided.

### 8.2.2 Direct Communication

Direct communication supposes that qubits are used to send a message directly. In Kak's Three-Stage protocol,

1. Alice randomly generates an angle,  $\alpha$ , and sends  $R_y(\alpha)|t\rangle$  to Bob.
2. Bob randomly generates an angle,  $\beta$ , and sends  $R_y(\beta)R_y(\alpha)|t\rangle$  to Alice.
3. Alice then applies  $R_y(-\alpha)$  and sends Bob  $R_y(-\alpha)R_y(\beta)R_y(\alpha)|t\rangle = R_y(\beta)|t\rangle$ .
4. Bob applies  $R_y(-\beta)$  and measures the resulting state,  $|t\rangle$ .

The main issue with this protocol is that there is much more communication in process; therefore, there is a greater chance that the qubits will become decoherent.

## 9 Algorithms

### 9.1 Deutsch-Jozsa Algorithm

Suppose that there exists a black-box function that outputs a 0 or a 1 based on a binary input. We are promised that the output is either constant (it

outputs the same value for all possible input combinations) or it is balanced (outputs 0 for half of the input combinations, and 1 for the other half). On a classical computer, the worst case scenario would require us to test  $2^{n-1} + 1$  solutions, with  $n$  being the number of bits in the input. However, this problem could be solved in one operation on a quantum computer; utilizing a quantum oracle (an operation that phase-flips a target bit based on the input).

### 9.1.1 2 input qubits

Suppose we're given a 2 qubit input oracle with a truth table, where  $a$ ,  $b$ ,  $c$ ,  $d$  are either 1 or -1.

Input	Output
$ 00\rangle$	$a$
$ 01\rangle$	$b$
$ 10\rangle$	$c$
$ 11\rangle$	$d$

Before the oracle is applied, we have a state of

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

After the oracle, any terms that cause a phase flip in the target qubit are also flipped, giving a state of

$$\frac{1}{2}(a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle)$$

We can then apply a second round of H gates to all of the input qubits, which, after expansion, would leave us with

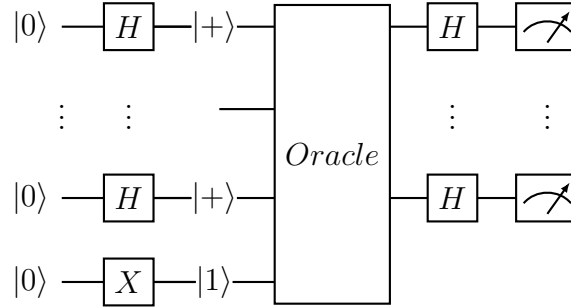
$$H_{all} \left[ \frac{1}{2}(a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle) \right]$$

$$\Downarrow$$

$$\frac{1}{4}[(a+b+c+d)|00\rangle + (a-b+c-d)|01\rangle + (a+b-c-d)|10\rangle + (a-b-c+d)|11\rangle]$$

Because of the concept of phase interference, we are left with the idea that if  $a + b + c + d = 0$ , or otherwise, if there's a 0% chance of measuring all zeroes in the qubit register, then we know the operation was balanced; so if we measure all of the qubits and exactly one of the results turns out to be in a  $|1\rangle$  state, then we are certain that the operation was balanced. Conversely, if we measure the qubits in the register and the states are all  $|0\rangle$ , then we can be certain that the operation was constant.

### 9.1.2 Generalized circuit



In the end, if the measured output is all  $|0\rangle$ s, then we are guaranteed that the operation in the oracle is a constant function.

## 9.2 Grover's Algorithm

Grover's algorithm would, theoretically, reduce the operations required for a classical searching algorithm by a square root of it: it has  $O(\sqrt{N})$  time. The algorithm takes in an input of  $n$  qubits and returns a state in which the correct output exists with high probability.

1. Apply H to all of the input qubits, which results in a uniform superposition of  $2^n$  possible states
2. Apply an X gate to the target qubit so that it is in the  $|1\rangle$  state
3. Run the Grover iteration  $2^{\frac{n}{2}}$  times
4. Measure the input register, in which the the correct state is measured with high probability

The Grover iteration consists of the steps:

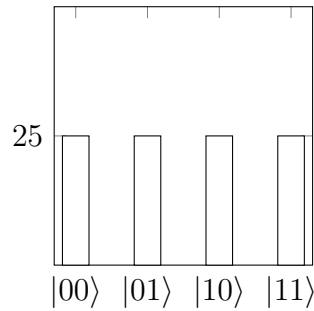
1. Run the oracle (that would flip the phase of the correct state) on the input register and target qubit
2. Apply H to all of the input qubits
3. Negate the state with all zeroes  $|0 \cdots 0\rangle$  with a zero-controlled Z on the output, with all of the inputs as controls
4. Apply H to all of the input qubits



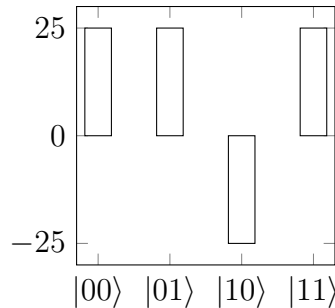
### 9.2.1 Explanation at length

Obviously, the numbered list does not necessarily do much to increase an understanding of Grover's algorithm. Why are we negating the state with all zeroes? Why are we applying H to all of the qubits?

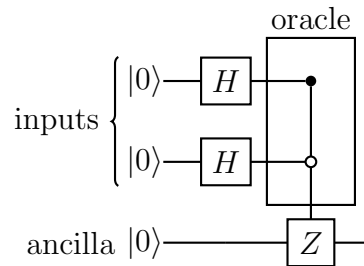
To begin, we should start with the question: How do we set a specific state apart from all of the others in an equal superposition? How Grover's algorithm seeks to achieve this is by applying a phase-flip to a certain state that fits exactly the right conditions. In other words, we need to go from this:



to this:



We can do this using a controlled Z gate, with zero-controls implemented as necessary. For example, let's say we're searching for the state  $|10\rangle$  in a two-qubit system. We can then flip the phase of the state:



We put a control on the first qubit and a 0-control on the second qubit so that we get a phase flip when we are in the state  $|10\rangle$ . We then apply the H gate in order to manipulate the amplitudes of our entire system to increase the probability that we'll get the right answer when we measure. So, after step 1, we end up with a uniform superposition:

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

After the application of the oracle, or the controlled Z, gate, we end up with

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

**Side note: how do we apply 2 single-qubit gates at once?**

When we first start with a blank slate, we are in the state  $|00\rangle$ , or, as expressed as a vector:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We can't simply apply a Hadamard gate to the vector as it is now; the H gate was designed for a single qubit. However, we can tensor the gate with itself to find what is effectively a single gate that applies an H gate to two qubits. This is the identity,

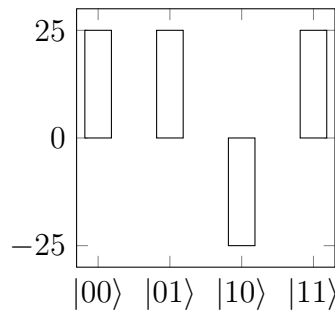
$$A(v_1) \otimes B(v_2) = (A \otimes B)(v_1 \otimes v_2)$$

Thus, the 2-qubit Hadamard operation is the matrix

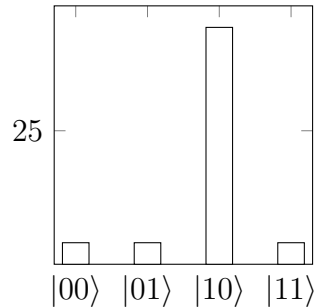
$$\frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Back to the algorithm. After this first step, we're left with a phase-flip on our desired state, differentiating it from the other states. This is good, right? Wrong. We can't end here, since we still have an equal likelihood that we will measure any of the four states; the phase-flip doesn't do anything to change the probability, since our probability is our amplitude-squared

(thus destroying any information we have about signs and phases). So, there arises a need for an amplification operator. The problem here is that we need some sort of operator that will amplify the probabilities of any states with a negative phase. This is where Grover's Diffusion operator comes in. Graphically, this can be represented as flipping each amplitude of the component about the *average* of the amplitudes. As seen in our graph from earlier, this will then correspond to our desired state flipping over the axis by a large delta, while the other states change by just a small margin. We go from:



to



(diagram not to scale). We notice that each time, from here on, that we perform these operators, the amplitude of our desired state will only increase (which is what we want).

Now, how do we translate this into gates? One way to do this: we apply an H gate, once again, to our state. Keep in mind that we should apply the H gate afterwards, as well. The motivation behind this is to transform our state into a vector with all zeroes ( $|00\rangle$ ), and we can apply the H gate after the fact to undo this operation. We now end up with the state

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

Then, we apply a controlled Z operator such that it will flip the phase of anything in a  $|00\rangle$  state. This leaves us with

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

Once we apply the H gate again, we're left with

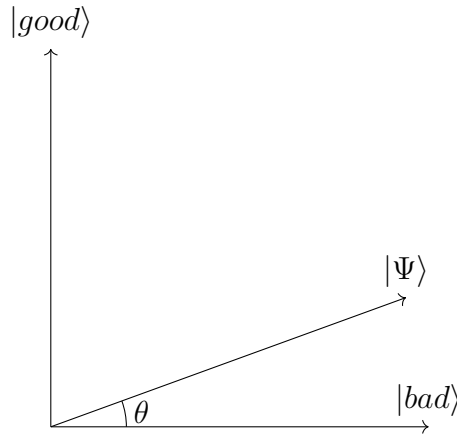
$$\frac{1}{4}(-4|10\rangle) = -|10\rangle$$

and we're left with our desired state. Since this system is on the simpler side (with 2 qubits), we were able to find our state in one iteration. However, the algorithm normally takes on the order of  $\sqrt{N}$ ,  $N = 2^n$  iterations to achieve a confident result.

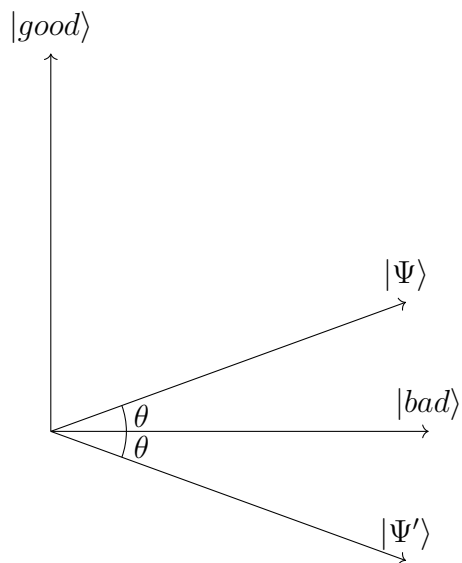
### 9.2.2 Complexity Analysis

It's been said that the time complexity of Grover's algorithm is something on the order of  $\mathcal{O}(\sqrt{N})$  (or, more accurately,  $\mathcal{O}(\frac{\pi}{4}\sqrt{N})$ ). Where does this figure come from?

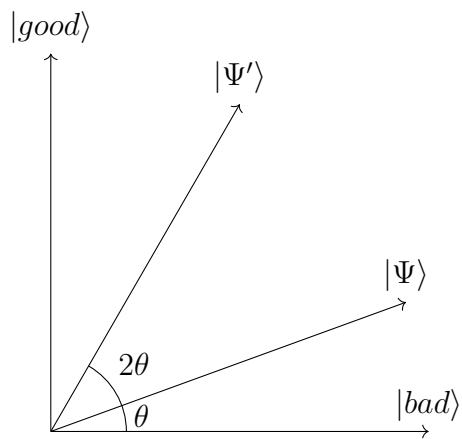
There is another geometric interpretation of Grover's algorithm: we think of our desired state (say,  $|good\rangle$ ) as some sort of vertical axis, and all of the states we don't want ( $|bad\rangle$ ) as a horizontal axis. This works because  $|bad\rangle$  and  $|good\rangle$  are perpendicular to each other; they are exclusive towards each other. Between these two axes, we can have a vector  $|\Psi\rangle$  that represents our state.



When we apply the oracle operator that would flip the phase of our desired state, then we end up reflecting our  $|\Psi\rangle$  vector over the  $|bad\rangle$  axis (similar in principle to the bar-graph visualization).



Then, when we apply the Diffusion operator, we are effectively flipping  $|\Psi'\rangle$  about the axis  $|\Psi\rangle$ .



So, effectively, the usage of both the oracle and the diffusion operator rotates our state counterclockwise by the angle  $2\theta$ . Basically, the closer we are to the  $|good\rangle$  axis, the more likely we are to find our  $|good\rangle$  state when we measure  $|\Psi'\rangle$ . Ideally, the final angle between  $|\Psi'\rangle$  and  $|bad\rangle$  would be  $\frac{\pi}{2}$  (or at least,

very close to it). So, let's define a number  $a$  that is the number of Grover iterations we perform. Then,

$$\frac{\pi}{2} \approx \theta + 2a\theta$$

We need to find  $\theta$  somehow. The inner product (or dot product) between  $|good\rangle$  and  $\Psi$  is  $||good\rangle| \cdot ||\Psi\rangle| \cdot \cos(\frac{\pi}{2} - \theta)$ . Since quantum states are normalized, the equation reduces down to  $\cos(\frac{\pi}{2} - \theta)$ , which can be transformed into  $\sin(\theta)$  by trigonometric identities. The dot product between the states is equal to  $\frac{1}{\sqrt{2^n}}$  — we know that, since  $\Psi$  is in equal superposition, the amplitude of each of these states is  $\frac{1}{\sqrt{2^n}}$ . We also know that  $|good\rangle$  contains a  $|1\rangle$  in the only “correct” state; so, the dot product should be  $\frac{1}{\sqrt{2^n}}$ . So:

$$\sin(\theta) = \frac{1}{\sqrt{2^n}}$$

By the small angle approximation for sine:

$$\theta \approx \frac{1}{\sqrt{2^n}}$$

We know that the resulting angle will be  $(2a + 1)\theta$  for  $a$  Grover iterations. Since we want it to be close to  $\frac{\pi}{2}$ :

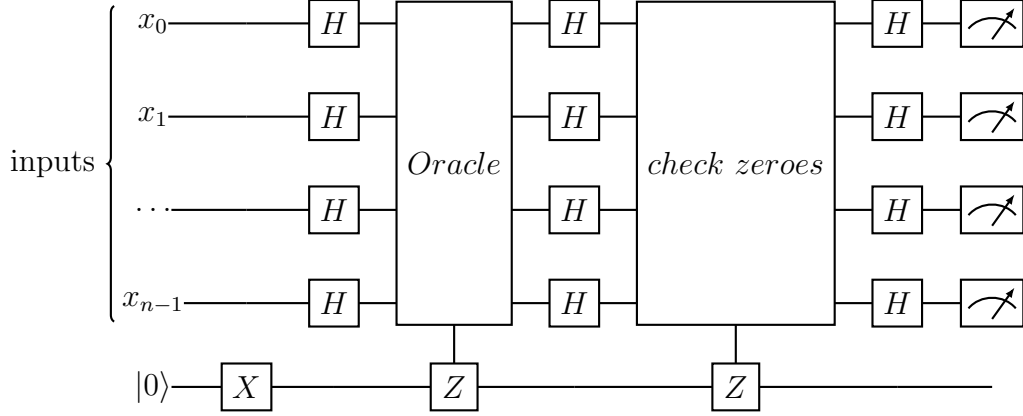
$$\frac{\pi}{2} \approx (2a + 1)\theta \approx \frac{2a + 1}{\sqrt{2^n}}$$

Doing some rearrangement, we get that

$$a \approx \frac{\pi}{4}\sqrt{2^n} = \frac{\pi}{4}\sqrt{N}$$

So the number of iterations we need is around the order of  $\sqrt{N}$ . Note that if we overshoot this number, we will rotate our state “backwards”; we will get further from our target than we want. This is why it's important to hit a “perfect spot” in determining the number of iterations needed.

### 9.2.3 Circuit



## 9.3 Simon's Algorithm

This algorithm is an example of a hybrid algorithm; an algorithm that uses the abilities of both quantum and classical computers to solve problems. Suppose that we are given a blackbox operation that takes in two qubit registers: an input, and output, both of the same size. This is a  $2^n - > 1$  function; there are two inputs for every possible output. We define a string  $s$  such that:

$$f(x_0) = y, f(x_1) = y$$

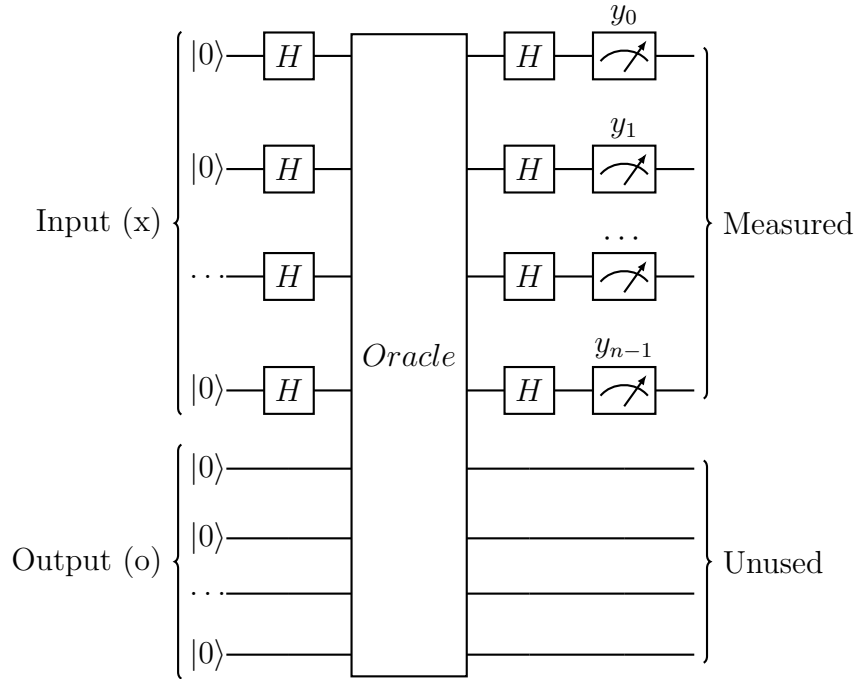
$$x_0 \oplus x_1 = s$$

For any pair of inputs that produce the same output, we always get the same string  $s$  after XORing the input strings. For example,  $s = |100\rangle$  for the left-shift operation on a 3-qubit input-output system. Given this setup, how many iterations are needed to find a pair of inputs that produce the same output? Or, how fast can one find the secret string for a given black-box function? This algorithm utilizes classical computing for pre and post processing. Briefly, the algorithm works like this:

1. Apply the quantum oracle to an input register in a uniform superposition. Apply the  $H$  gate to all of the qubits, and then measure them.
2. The previous operation will produce a bitstring  $y$  where the mod-2 dot product of the bitstring and  $s$  is 0 ( $(y_0 \cdot s_0) \oplus (y_1 \cdot s_1) \oplus \dots \oplus (y_{n-1} \cdot s_{n-1}) = 0$ )

3. We can add the bitstring to our list of outputs. Repeat the quantum step again, and if the output is not linearly independent from the other outputs in the list, discard it.
4. We repeat the quantum step until we have a total of  $n - 1$  linearly independent bit strings.
5. We are left with  $n - 1$  linearly independent bitstrings which forms a matrix of  $n - 1$  linear equations.
6. We can use a classical process to uncover the  $n^{th}$  linearly independent equation, and we can add this to our matrix.
7. We can use a classical mod-2 Gaussian elimination/back substitution process to solve the matrix.
8. The solution to the matrix will give us the string  $s$ , with which we can check with a further classical process.

The corresponding circuit to the quantum component is as follows:



As you can see, the crucial part of the algorithm depends on the classical processing.



## 9.4 Quantum Fourier Transform

Data is commonly collected in terms of a time domain. For some periodic data, plotting it against a time axis allows the observer to see (mostly) clearly defined peaks and troughs and identify a period in the data. However, with this interpretation, some patterns in the periodicity are hard to observe. This is where plotting information on a frequency domain comes in. In general, a wave is defined by:

$$y = a \cdot \sin(f \cdot x + p)$$

where  $a$  is amplitude,  $f$  is frequency, and  $p$  is phase. However, most data will not behave as a simple and smooth sine wave. Thus, we can think of some wave as a sum of different wave functions with different frequencies and phases, a compound wave. The problem statement, then, is thus: we need to find some algorithm that will find all of the different waves that one must combine to approximate some function to a certain degree; or, we must find an algorithm to find a Fourier series. The way we do this is the Fourier Transform. The Discrete Fourier Transform is described as:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{\frac{-2\pi kni}{N}}$$

where  $N$  is the number of values in the original data,  $x_n$  is the  $n$ th value in the original data, and  $k$  is the index of the corresponding output value. To calculate the entire transform for the input set, we would have to run this operation from  $k = 0$  to  $k = N - 1$ .  $X_k$  frequencies in the outputs aren't exact; they're ranges that the corresponding waves fall into. The fidelity of the DFT, then, is dependent on the number of samples that exist in the original data.

What we get out of the DFT is a sort of set of data that tells you the strengths of various frequencies in your data set. For each of the outputs from the discrete time periods (which are complex numbers), we can use formulas to extract the frequency, amplitude, and phase of the specific wave — values that will be used to recreate the original periodic function. For a

value  $y_k = a + bi$ :

$$\begin{aligned} frequency_i &= \frac{i}{r} \\ amplitude_i &= \frac{2\sqrt{a^2 + b^2}}{N} \\ phase_i &= atan2(b, a) \end{aligned}$$

where  $i$  is your index,  $r$  is your range of your values in your initial domain, and  $N$  is your sample size.

Now, we need some way of translating this to a quantum algorithm. We can encode our state as:

$$|\psi\rangle = x_0|0\rangle + x_1|1\rangle + \dots + x_{N-1}|N-1\rangle = \sum_{a=0}^{N-1} x_a|a\rangle$$

Traditionally, the QFT is represented as an inverse transform (from frequency to time), rather than the other way around. Thus, to get the same effect, we need to apply the inverse of the QFT. So, the QFT is just

$$X_k = \sum_{n=0}^{N-1} x_n * e^{\frac{2\pi kni}{N}}$$

Define  $N$  as the classical array size, and  $n$  as the number of qubits used. So,  $N = 2^n$ .

$$\begin{aligned} |\psi\rangle &= \sum_{a=0}^{N-1} x_a|a\rangle \\ IQFT(\psi), |\psi'\rangle &= \sum_{k=0}^{2^n-1} \sum_{a=0}^{2^n-1} x_a \cdot e^{\frac{-2\pi kai}{2^n}}|a\rangle \end{aligned}$$

#### 9.4.1 Purpose

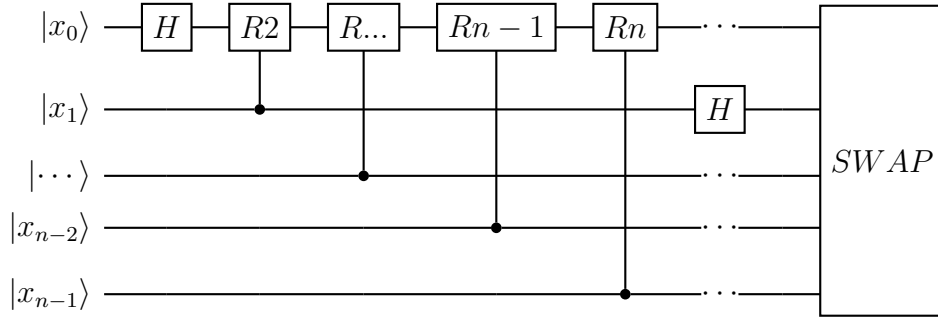
We have to ask: why is using the Fourier transform useful? There are many benefits provided by the idea of deconstructing a periodic function into simple sine and cosine waves. For example, when transmitting audio information, it is less resource-intensive to send a set of discrete points that represent the Fourier transform of the wave, rather than sending samples of the wave itself. Along this line of reasoning, it is also much easier to reconstruct a wave from a series of sine and cosine waves, rather than from discrete samples of it.

### 9.4.2 Circuit

To start, we can define some rotation gate:

$$R_a = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^a}} \end{bmatrix}$$

The QFT (non-inverted version) is as shown:



Basically, in a sequence from qubits  $0 - (n - 1)$ , we:

1. Apply an H gate
2. Sequentially apply a controlled  $R_a$  gate to every qubit after it, with the other qubit being its control
3.  $a$  is equal to one more than the difference between the indexes of the qubits
4. After all gates have been applied, swap the entire register

## 9.5 Shor's Algorithm

The problem that Shor's algorithm attempts to solve is: given some non-prime integer  $N$ , find an integer greater than 1 and less than  $N$  that divides it evenly.

$$\begin{aligned} &\text{Find } a \text{ where } \frac{N}{a} = b \\ &1 < a < N \\ &a, b, N \in \mathbb{Z} \end{aligned}$$

For large numbers, this problem is extremely resource-intensive. Integer factorization is classified as an NP problem, which makes it useful for cryptography; since it is hard to find some divisor without a brute-force algorithm, integer divisors are more secure to use in a cryptographical context. Shor's algorithm has the potential to bring this problem into the BQP space - bringing the time-scale that the algorithm takes down to a human-recognizable period. Thus, it has the potential to break many asymmetrical key cryptographic functions.

### 9.5.1 Modular Exponentiation

Consider the equation

$$y = a^x \mod b$$

So, for example,

$$\begin{aligned} 11^0 \mod 21 &= 1 \\ 11^1 \mod 21 &= 11 \\ &\dots \end{aligned}$$

We can represent this operation as a quantum circuit so that we may utilize it in quantum algorithms. The equation takes in four arguments:

1.  $x$ : input (perhaps in superposition)
2.  $y$ : output
3.  $a$ : classical integer
4.  $b$ : classical integer

Recall that a binary expansion of a binary number  $x$  (in big-endian notation) is:

$$dec(x) = (x_0 \cdot 2^{n-1} + x_1 \cdot 2^{n-2} + \dots + x_{n-1} \cdot 2^0)$$

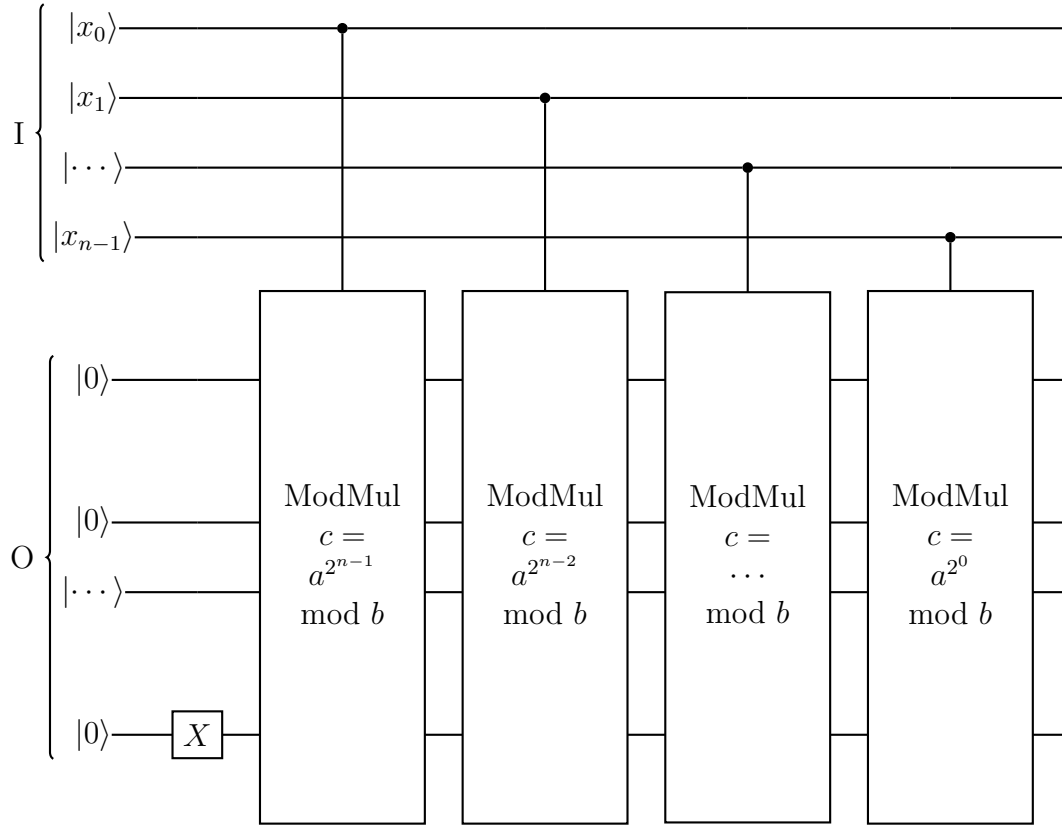
Thus,

$$\begin{aligned} y &= a^x \mod b \\ &= a^{x_0 \cdot 2^{n-1} + x_1 \cdot 2^{n-2} + \dots + x_{n-1} \cdot 2^0} \mod b \\ &= a^{x_0 \cdot 2^{n-1}} \cdot a^{x_1 \cdot 2^{n-2}} \dots a^{x_{n-1} \cdot 2^0} \mod b \\ &= (a^{x_0 \cdot 2^{n-1}} \mod b) \cdot (a^{x_1 \cdot 2^{n-2}} \mod b) \dots (a^{x_{n-1} \cdot 2^0} \mod b) \mod b \end{aligned}$$

We can apply this to quantum states as well. When  $x_i = 0$ , we get something like:

$$\begin{aligned}
 |r'\rangle &= (|r\rangle \cdot a^{0 \cdot 2^{n-i-1}} \bmod b) \bmod b \\
 &= (|r\rangle \cdot a^0 \bmod b) \bmod b \\
 &= (|r\rangle \bmod b) \bmod b \\
 &= |r\rangle
 \end{aligned}$$

So, for bits where  $x_i = 0$ , we wouldn't do anything to the output; this can be implemented with controlled operations. Basically, the circuit for quantum modular exponentiation looks something like:



$$|r'\rangle = (|r\rangle \cdot c) \bmod b$$

The input register in the circuit holds the binary value  $x$ , the exponent in  $y = a^x \mod b$ . The output register starts with its last value set to  $|1\rangle$  and has a chain of controlled modular multiplications. Each modular multiplier is parameterized by the constant  $c$ , as defined in each successive gate.

### 9.5.2 Periodicity of Modular Exponentiation

Recall

$$y = a^x \mod b$$

If  $a$  and  $b$  are coprime, then the output of the function  $y$  is periodic (cyclical, repeats forever). More generally, the period of the modExp function is the smallest value of  $p$  that satisfies the equation

$$a^p \mod b = 1, \quad p > 0$$

Classically, finding the period of a modular exponentiation function is an NP-hard problem. There is no trick to doing this classically. However, we can solve this in a reasonable time with quantum operations. Set a register  $|x\rangle$  in equal superposition so that

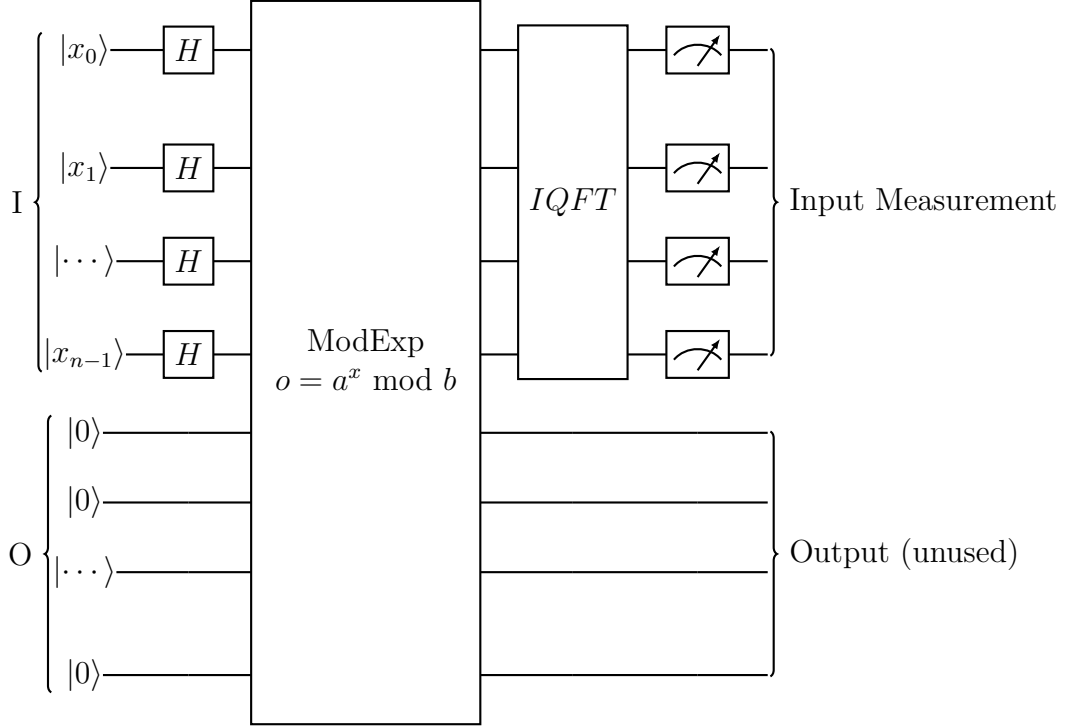
$$|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$$

We can then run modular exponentiation:

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

$$ModExp(|x\rangle, |0\rangle), \quad |x, y\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i, a^i \mod b\rangle$$

With the results, we now have a superposition with a periodic component to it. Because the input and output registers themselves do not have information about the periodicity of the outcomes, the period information is encoded within the relationship between the input and output. Now, we can just run the IQFT to find the period (and consequentially, the frequency). The circuit looks something like:



We go from

$$|x, o\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j, a^j \mod b\rangle$$

to

$$|x, o\rangle = \frac{1}{2^n} \sum_{k=0}^{2^n-1} \sum_{j=0}^{2^n-1} e^{-2\pi i j k / 2^n} |k, a^j \mod b\rangle$$

The operation will cause quantum interference such that most states will destructively interfere with each other so that the amplitudes are decreased by a large amount. However, some states will constructively interfere, creating “peaks” (the number of which correspond to the value of the period of the modular exponentiation function). We can call the peaks special states, denoting the  $i$ -th state as  $s_i$ . The states amplified by the algorithm follow the equation

$$s_i = \frac{2^n \cdot i}{p}, \quad 0 \leq i < p$$

where  $p$  is our period. Because  $|x_i\rangle$  is an integer, it is rounded to the closest value of the corresponding special state.

$$|x_i\rangle \approx \frac{2^n \cdot i}{p}$$

The formula can be rearranged into

$$\frac{|x_i\rangle}{2^n} \approx \frac{i}{p}$$

So when we measure  $x_i$ , we have a close approximation for  $\frac{i}{p}$ . We can't get the value of  $p$  exactly with this alone. We can, however, get the precise value of  $\frac{i}{p}$  using continued fractions, or a fraction of the form

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots}}$$

Any real number can be expressed in this form. For irrational numbers, the continued fraction will repeat forever; for rational numbers, the fraction will end at some point. Because  $i, p \in \mathbb{Z}$ , we know that  $\frac{i}{p} \in \mathbb{Q}$ . There is a simple, iterative algorithm for getting the coefficients  $a_i$  of the continued fraction:

1. Call  $P_0$  the numerator used for the first iteration, and  $Q_0$  the denominator for the first iteration.  $P_0 = P$ ,  $Q_0 = Q$
2. The first coefficient,  $a_0$ , will be the  $P_0$  integer divided by  $Q_0$ :  $a_0 = \left\lfloor \frac{P_0}{Q_0} \right\rfloor$
3. Define a variable  $r_0$  so that  $r_0 = P_0 \bmod Q_0$
4. For the next iteration,  $P_1 = Q_0$ ,  $Q_1 = r_0$
5. We can repeat step 2 to obtain the next coefficient, and steps 3/4 to get the next  $P$  and  $Q$ .
6. Repeat until we get to a point where  $r_i = 0$

We can approximate these continued fractions to a degree; these are known as convergents. Define  $v_i$  such that it will be the  $i$ -th convergent of some



continued fraction. Then,

$$\begin{aligned} v_0 &= a_0 \\ v_1 &= a_0 + \frac{1}{a_1} \\ v_2 &= a_0 + \frac{1}{a_1 + \frac{1}{a_2}} \end{aligned}$$

and so on and so forth. There also exists an algorithm to calculate the numerator ( $n_i$ ) and denominator ( $d_i$ ) for a convergent.

1. Start with the values  $n_{-2} = 0$ ,  $d_{-2} = 1$ ,  $n_{-1} = 1$ ,  $d_{-1} = 0$
2. Calculate  $n_0 = a_0 \cdot n_{-1} + n_{-2}$  and  $d_0 = a_0 \cdot d_{-1} + d_{-2}$
3. Repeat the process:  $n_i = a_i \cdot n_{i-1} + n_{i-2}$ ,  $d_i = a_i \cdot d_{i-1} + d_{i-2}$
4. Keep going until  $d_i$  is larger than some arbitrary cutoff, after which it is determined that the subsequent denominators become too large (and, the subsequent fractions become too small) to matter.

In the period-finding algorithm, the cutoff can be set to  $b$ ; that is, the modulus value used in the modular exponentiation. We return the last denominator that is less than the modulus. After we've calculated some arbitrarily close fraction, we can apply the knowledge that the fraction is, in some way, equal to  $\frac{i}{p}$ . However, since the output will be in some reduced form, we should check to see if the answer is the actual period; if not, then the period is actually some multiple of the denominator.

### 9.5.3 Aggregating Factors

After running the quantum subroutine to find the special states of the periodic modExp function, we can construct a list of factors of our period (given by the denominators of the outputs of the convergent-finding algorithm). Let's say that, after running the algorithm twice, we have two values:  $d_0$  and  $d_1$ , which are the denominators of the reduced function output by the period finding algorithm. Now, we can find the actual period of the function by taking the least common multiple (LCM) of the values, given by the formula:

$$\text{lcm}(d_0, d_1) = \frac{d_0 \cdot d_1}{\text{gcd}(d_0, d_1)}$$

where the greatest common divisor (GCD) can be found efficiently with Euclid's algorithm. We can run this over and over with a new period and new measurements until we find a period  $p$  that satisfies the initial equation:

$$a^p \bmod b = 1$$

#### 9.5.4 Summary

The goal of Shor's algorithm is to find the period of  $a^x \bmod b$ . We know that  $a \perp b$ .

1. Run the quantum subroutine:
  - (a) Prepare an output register  $|o\rangle = |0\rangle$  of  $m$  qubits, where  $m = \lceil \log_2(b+1) \rceil$  because  $m$  must be large enough to hold  $b$ .
  - (b) Prepare an input register  $|x\rangle = |0\rangle$  of  $n$  qubits, which usually works out to  $n = 2m$ .
  - (c)  $H_{ALL}|x\rangle$ , or put  $|x\rangle$  into a uniform superposition.
  - (d) Run the modular exponentiation function with the input  $|x\rangle$ , output  $|o\rangle$ , and parameterized with  $a, b$ 
    - i. Apply X to  $|o_{m-1}\rangle$  such that  $|o\rangle = |1\rangle$ .
    - ii. Iterating through all of the qubits in the input, take its index,  $i$ .
      - A. Calculate  $c = a^{2^{n-1-i}} \bmod b$
      - B. Apply the controlled ModMul function, where the qubit  $x_i$  is the control bit, the register  $|o\rangle$  is the target,  $c$  is the multiplier, and  $b$  is the modulus.
    - iii. Repeat for all qubits in  $|x\rangle$ .
  - (e) Run the inverse QFT on  $|x\rangle$ .
  - (f) Measure the qubits in  $|x\rangle$  and convert them to a decimal integer, called  $x'$ .
2. If  $x' = 0$ , this is the trivial case: discard it and try again.
3. Find the closest continued fraction convergent of  $\frac{x'}{2^n}$ .
  - (a) Start with the values:  $n_{-2} = 0, d_{-2} = 1, n_{-1} = 1, d_{-1} = 0, P_0 = x', Q_0 = 2^n, i = 0$

- (b) Calculate the coefficients of each step of the continued fraction:  

$$a_i = \left\lfloor \frac{P_i}{Q_i} \right\rfloor$$
  - (c) Calculate the remainder,  $r_i = P_i \bmod Q_i$
  - (d) Calculate the convergent denominator  $d_i = a_i \cdot d_{i-1} + d_{i-2}$
  - (e) For the next iteration: increment  $i$ , set  $P_i = Q_{i-1}$ , set  $Q_i = r_{i-1}$
  - (f) Repeat until  $r_i = 0$  or  $d_i \geq b$ . Return the last value of  $d_i$  that is greater than  $b$ .
4. Check if  $a^{d_i} \bmod b = 1$ . If it is, then  $d_i$  is the period; if not, then we know  $d_i \nmid p$ . Set  $d_i = f_0$
  5. Run the steps over to find a new denominator,  $f_1$ .
  6. Calculate a larger factor of  $p$  with  $f = f_0 \cdot f_1 / \gcd(f_0, f_1)$ .
  7. If our new  $f$  fulfills the test requirement, then  $f = p$ . If not, set  $f = f_2$  and run again until the period is found.

### 9.5.5 Relevance to Factorization

The integer factorization problem is as follows:

Given some non-prime integer  $b$ , find an integer  $b_0$  that divides it evenly.  
 $b \mid b_0$ .

With Shor's Algorithm, we can find the periodicity of some parameterized modExp function,  $a^p \bmod b = 1$ . With the rules of modular arithmetic, that can be rewritten to

$$(a^p - 1) \bmod b = 0$$

Again, this can be rewritten to

$$a^p - 1 = z \cdot b, \quad z \in \mathbb{Z}$$

So, given our knowledge of the period of the function, we can calculate  $z$  easily. Let's express our integers in the following way:  $b = b_0 \cdot b_1$ ,  $z = z_0 \cdot z_1$ . Then,

$$a^p - 1 = z_0 z_1 b_0 b_1$$

We can also rewrite  $a^p - 1$  as  $(a^{\frac{p}{2}})^2 - 1$ , which can be written in a difference of squares format:

$$(a^{\frac{p}{2}} - 1)(a^{\frac{p}{2}} + 1) = z_0 z_1 b_0 b_1$$

Basically, we're guaranteed that  $(a^{\frac{p}{2}} - 1)$  and  $(a^{\frac{p}{2}} + 1)$  contains any product combination of  $z_0, z_1, b_0, b_1$ . We can calculate the factors using the GCD of one of the two factors listed above with  $b$ . For example:

$$\begin{aligned}\gcd(b, a^{\frac{p}{2}} + 1) &= b_0 \\ \gcd(b, a^{\frac{p}{2}} - 1) &= b_1\end{aligned}$$

We've found the factors using Shor's algorithm and a few tricks. We can run a few tests to make sure our result holds and is not the result of an overlook of some trivial case. In summary:

1. Select an integer  $b$  to be factored. It should be composite, which can be verified with a primality test.
2. If  $b$  is even, then its factors will just be 2 and  $\frac{b}{2}$ . We can just return them.
3. Choose a number  $a$  such that  $2 < a < b$ .
4. If  $a$  and  $b$  are not coprime (if  $\gcd(a, b) \neq 1$ ) then the factors of  $b$  are  $\gcd(a, b)$  and  $\frac{b}{\gcd(a, b)}$ .
5. If  $\gcd(a, b) = 1$ , then  $a^x \bmod b$  is periodic - use Shor's algorithm to find the period  $p$ .
6. If  $p$  is odd, then it can't be used for factoring, so we have to run the algorithm again with a different value of  $a$ .
7. Check if  $a^{\frac{p}{2}} \bmod b$  is equal to either 1 or  $-1$ ; if it is, then we will get the trivial factors of  $b$ , 1.
8. Calculate the factors of  $b$ :

$$(a) \quad b_0 = \gcd(b, (a^{\frac{p}{2}} \bmod b) + 1)$$

$$(b) \quad b_1 = \gcd(b, (a^{\frac{p}{2}} \bmod b) - 1)$$

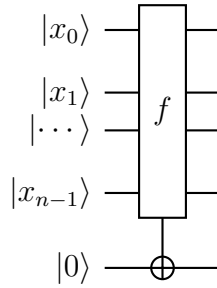
9. Perform a sanity check, verifying that  $2 < b_0 < b, 2 < b_1 < b, b_0 \cdot b_1 = b$ .

## 9.6 Bernstein-Vazirani Algorithm

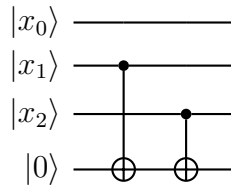
We have a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  defined using a fixed and unknown string, called  $s$ . where  $s = s_0 s_1 \cdots s_{n-1}$ . The function operates under the condition

$$f(x_0 x_1 \cdots x_{n-1}) = (s_0 \cdot x_0) \oplus (s_1 \cdot x_1) \oplus \cdots \oplus (s_{n-1} \cdot x_{n-1})$$

Or,  $f$  performs the bitwise dot product of  $x$  and  $s$ . Our goal is to find  $s$  in as few iterations as possible. Classically, we would need  $n$  iterations, as we can modify one bit of  $x$  at a time and find a correspondence to a bit in  $s$ . With a quantum algorithm, we only need 1 iteration. We can implement the oracle, which will take in some input  $x$  and apply a bitflip to an output qubit if  $f(x) = 1$ .



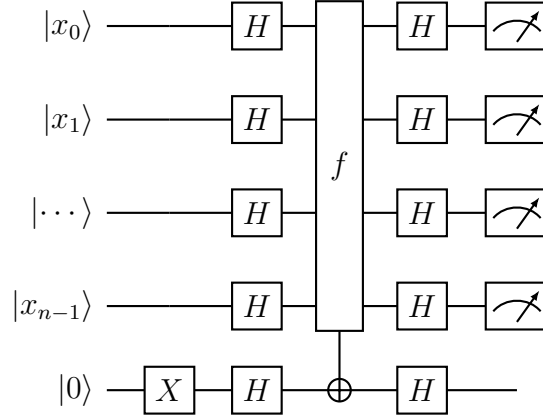
Internally, the oracle will apply a CNOT gate for every  $s_i$  such that  $s_i = 1$ , with  $s_i$  as the control and the output as the target. For  $s = 011$ :



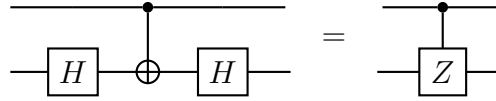
Now, our problem is to find which input bits are controls for a CNOT gate, and which ones don't affect the output. Our algorithm to solve this problem is as follows:

1. Apply  $X$  to the output qubit so that it starts in the state  $|1\rangle$
2. Apply  $H_{ALL}$  to the entire register, including the output
3. Run the oracle on the input register with the output as the target

4. Apply  $H_{ALL}$  to the register again
5. Measure all of the inputs to find  $s$

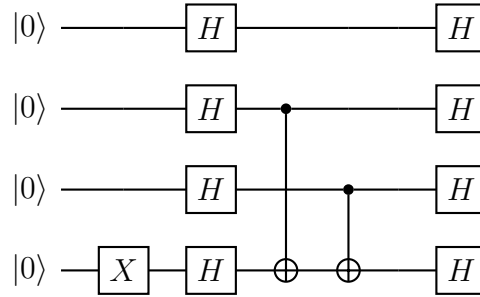


The algorithm demonstrates the implementation of phase kickback; applying  $H$  gates before and after a CNOT gate swaps the control and target of the CNOT gate. If we flip all of the CNOT gates in the earlier diagram upside down (or, make the output qubit the control and the corresponding controls into targets), then an output qubit in the  $|1\rangle$  state would simply reveal the string  $s$  encoded into the oracle with an input of all zeroes. We can prove that this works with circuit diagram equivalence. Recall that  $H^2 = I$  and that  $HXH = Z$ . We can apply a control to this; if the control bit is 1, then an  $HXH = Z$  gate is applied; otherwise, an  $HH = I$  gate is applied instead. Therefore,

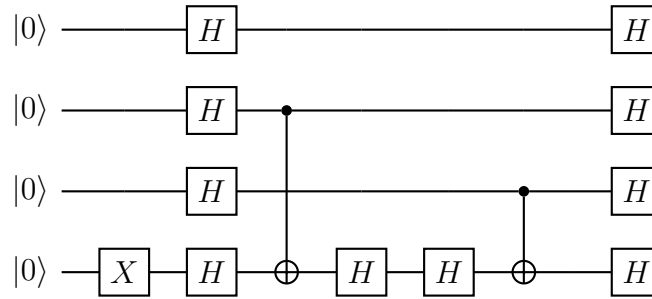


In a CZ gate, the target and control are interchangeable. Then, if the output bit is set to  $|1\rangle$  before having an  $H$  gate applied to it, the CZ gates will apply. The gates will then act on the input register bits that correspond to an  $s_i = 1$ . When  $H$  gates are applied to all of the input qubits, they will be in the  $|+\rangle$  state; when the oracle is applied, the qubits that correspond to  $s_i = 0$  will remain unaffected and will be placed back to  $|0\rangle$  due to the  $H_{ALL}$  operation at the end of the circuit. The qubits that correspond to  $s_i = 1$ , however, will be targeted by an effective CZ gate that will change them into

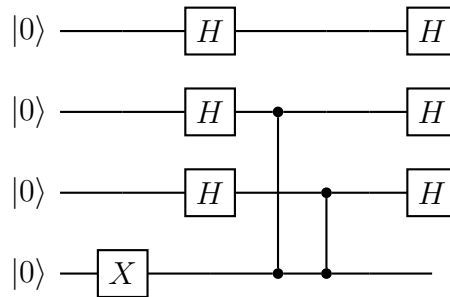
a  $|-\rangle$  state, which will then measure as  $|1\rangle$  after the  $H_{ALL}$  is applied. So, our oracle from before:



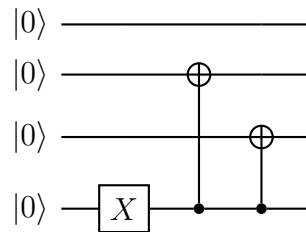
can be turned into



because  $H^2 = I$ . With the identity  $HXH = Z$ , the circuit can then be turned into



This, in turn, can be turned into



## 9.7 Quantum Clustering Algorithm

A big problem in computing is the formulation of an algorithm for clustering unlabeled data. Currently, there are many classical algorithms (such as K-means) that attempt to cluster unlabeled data; however, they are relatively expensive, and there arises a need for efficient algorithms. The process here would be:

1. Obtain unlabeled data
2. Encode data points into quantum space
3. Execute a quantum clustering algorithm
4. Obtain clusters from the unlabeled data

### 9.7.1 K-means Algorithm

The K-means algorithm partitions unlabeled data into  $k$  clusters.

1. Initialization: Pick  $k$  data points as initial centroids
2. Assignment: Calculate distance between each data point and centroid, and assign each data point to a centroid based on the closest distance
3. Update: Recalculate new centroids with a mean of newly assigned points
4. Finish: No data points change clusters, and cluster centroids no longer move

However, this algorithm has a very large time complexity, depends on Euclidean distance (which generally does not make sense in discrete data sets) and has a curse of dimensionality.

### 9.7.2 Q-means Algorithm

There also exists a Q-means algorithm that serves to be a quantum analogue of the existing K-means algorithm. There are two problems at hand here: encoding the classical data into quantum data, and measuring the proximity between the data points and the centroids. What the Q-means algorithm does differently is that it considers if measuring the distance between the



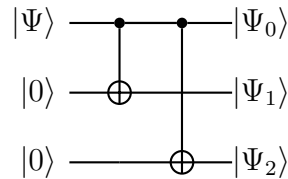
centroid and a data point is really *necessary*; in the quantum algorithm simply determines that the detecting similarity suffices, using the SWAP Test (which compares the angle between two vectors to determine its “closeness”). This Q-means algorithm has a poly-logarithmic time complexity per iteration, and assumes that the data is stored in QRAM (which doesn’t exist at present). Present implementations have various disadvantages, such as hard-coded mapping, harsh termination criteria, and processed things with single threads.

## 10 Error Correction

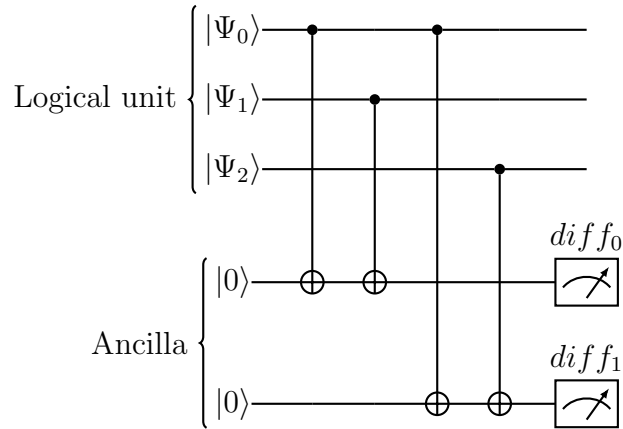
The idea with error correction is that multiple qubits are used to encode one value, with the hope that it will be less prone to error and noise.

### 10.1 Bit-flip Error Correction

Bit-flip code uses two spare qubits to “protect” the original qubit. It works in a way that allows it to identify and correct one single bit-flip error; however, it can’t protect against more than one bit-flip nor phase-flip errors. First, we encode one original qubit into a “logical” qubit:



The second part of the error correction scheme is something known as **Syndrome Measurement**. After we define our single logical qubit, we can perform different checks to make sure which qubit in the logical unit was flipped, if there was a bit flip at all. The circuit looks something like:



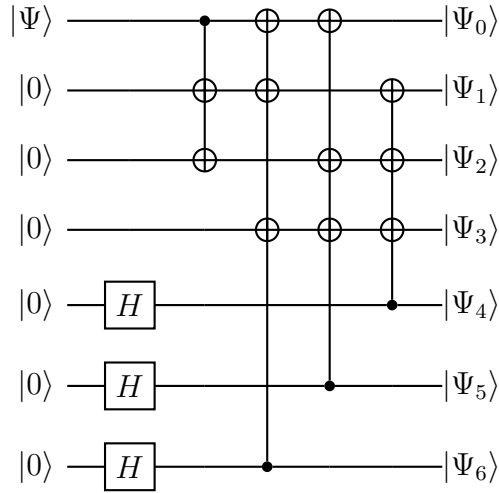
When we measure the ancilla qubits, the placement of the results can tell us about the parity of the bits in the qubit. We can refer to the table:

Register state	0-1 Syndrome Measurement	0-2 Syndrome Measurement
000	$ 0\rangle$ (same parity)	$ 0\rangle$ (same parity)
001	$ 0\rangle$ (same parity)	$ 1\rangle$ (different parity)
010	1 (different parity)	$ 1\rangle$ (different parity)
100	$ 1\rangle$ (different parity)	$ 0\rangle$ (same parity)

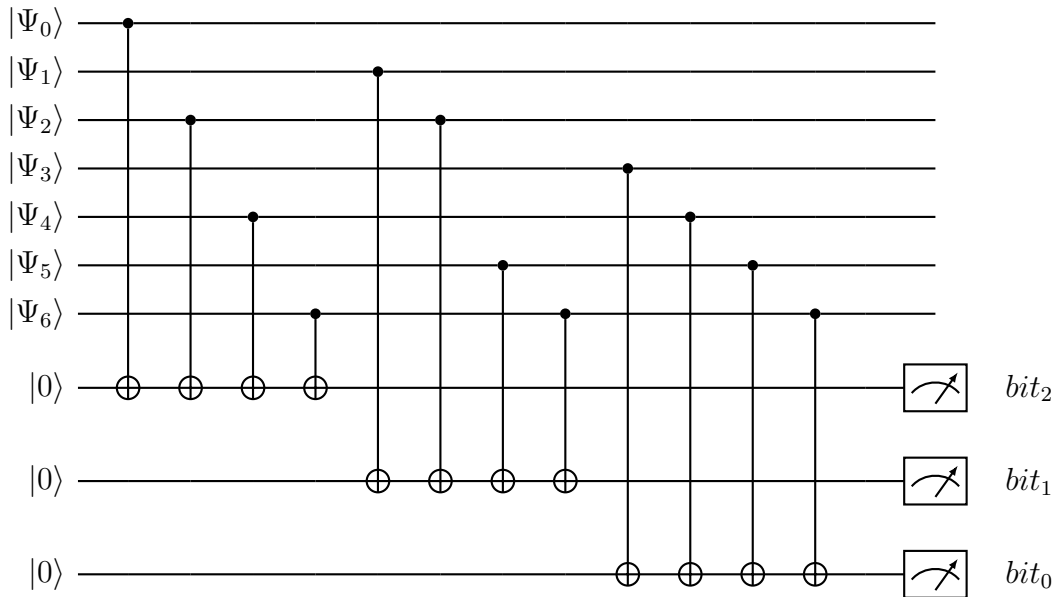
There are four possible outcomes of the syndrome measurement, each corresponding to a different qubit. However, this approach is quite limited in what it can do: it can only detect single-bit flips, and cannot do anything about multi-bit or phase flips.

## 10.2 Steane's Error Correction Code

Steane's Error Correction Code uses 7 physical qubits to comprise a single logical qubit. It is able to detect single bit-flip and phase-flip errors.

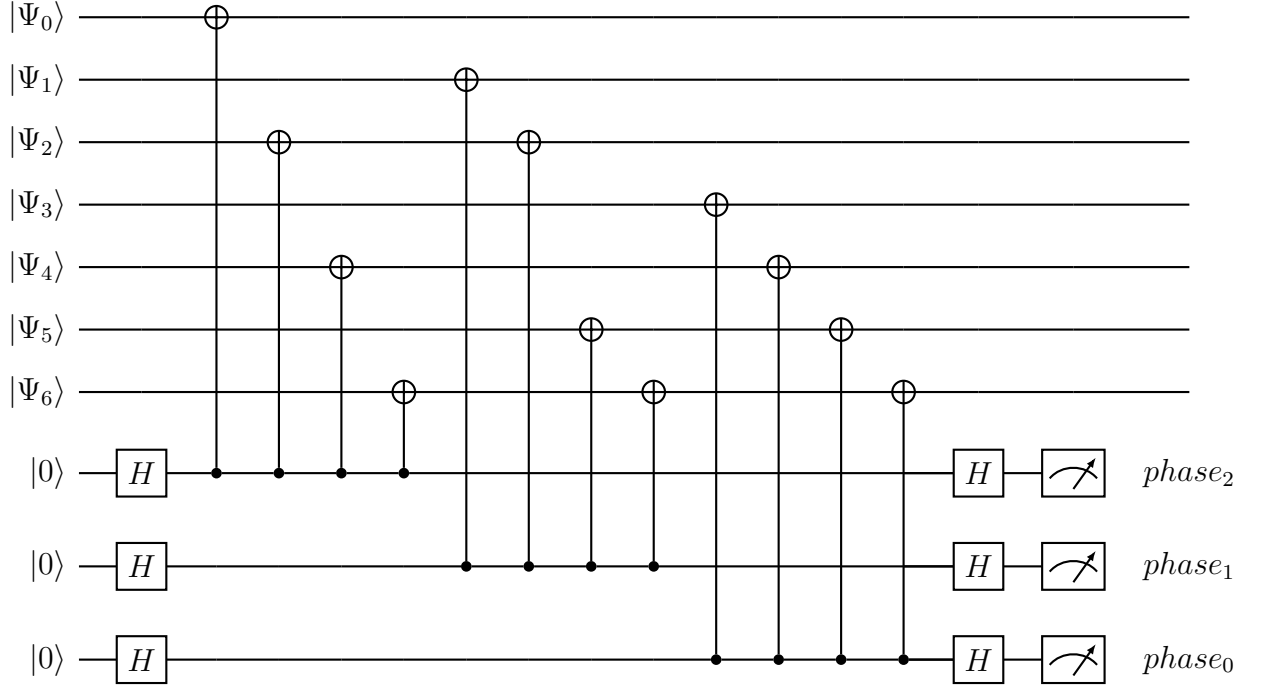


The circuit above creates a single logical, error protected qubit out of 7 physical qubits. Like the simple bit-flip code, we can perform syndrome measurements to determine where the error is in the qubit. The syndrome measurement consists of 6 ancilla qubits, with 3 of each allocated to detecting bit flips and phase flips, respectively.



The diagram above shows the process for finding the bitflip of any physical

qubits in the logical unit. The phase flip circuit is separate, but they can be chained together:



There's a similar table for interpreting the results of the syndrome measurements:

Broken Index	$S_0$	$S_1$	$S_2$
None	0	0	0
0	0	0	1
1	0	1	0
2	0	1	1
3	1	0	0
4	1	0	1
5	1	1	0
6	1	1	1

where  $S_i$  represents the  $i$ -th bit measurement of the particular syndrome measurement (phase or bitflip). Notice the relationship between the broken index and the values of  $S_0$ ,  $S_1$ ,  $S_2$ ; The broken index is, generally, the decimal value of the binary number  $S_0S_1S_2$  (big-endian order)  $-1$ .

## 11 Other

### 11.1 Constructing the H gate

$$R_y = \begin{bmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}, \quad R_x = \begin{bmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\begin{aligned} \begin{bmatrix} \cos \frac{\pi}{2} & -i \sin \frac{\pi}{2} \\ -i \sin \frac{\pi}{2} & \cos \frac{\pi}{2} \end{bmatrix} \cdot \begin{bmatrix} \cos \frac{\pi}{4} & -\sin \frac{\pi}{4} \\ \sin \frac{\pi}{4} & \cos \frac{\pi}{4} \end{bmatrix} &= \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{-i}{\sqrt{2}} & \frac{-i}{\sqrt{2}} \\ \frac{-i}{\sqrt{2}} & \frac{i}{\sqrt{2}} \end{bmatrix} \\ &= \frac{-i}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \end{aligned}$$

Because  $-i$  is a global phase, the resultant matrix is functionally equivalent to the H gate.