

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“DESARROLLO E IMPLEMENTACIÓN DE ALGORITMO
DE ORDENAMIENTO PARA COMPUTADORES
CUÁNTICOS”

IGNACIO ANDRÉS OÑATE LÓPEZ

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Mauricio Solar
Profesor Correferente: Claudio Torres

Diciembre - 2022

DEDICATORIA

Con todo cariño a mis abuelos Yita y Arturo, quienes desde que tengo memoria, motivaron y fomentaron la búsqueda de conocimiento en todo ámbito, tanto en las letras, como en las artes y la ciencia.

AGRADECIMIENTOS

Cuando se trata de agradecimientos la lista es larga, primero reiterar en agradecer a mis abuelos quienes fueron la base de mi motivación en los estudios en general, por supuesto también agradezco a todos mis familiares, mi madre, padre y tíos, quienes me brindaron su apoyo a lo largo de mi vida dándome acceso a las distintas herramientas y establecimientos con los que desarrollé mi aprendizaje.

También es necesario agradecer a todos los amigos que hice a lo largo de mi carrera, quienes me apoyaron, animaron y acompañaron en el sufrimiento que conlleva el aprendizaje. Sufrimiento que eventualmente se tornó placentero una vez nos volvimos capaces de resolver problemas que antes no teníamos ni la más mínima idea de cómo abordar. Es necesario también incluir a mis amigos fuera del ámbito académico, con quienes descansaba y escapaba de la tortura mental que significa estudiar, como Matías quien ha estado en gran parte de mi vida apoyándome y dándome mucha alegría y cariño en los momentos difíciles, también a Inés quien ha sido un pilar fundamental en esta última etapa, motivando mi esfuerzo y dedicación.

Debo incluir a mis profesores a quienes recordaré siempre con mucho cariño, muchas figuras inspiradoras a quienes vi como entes poderosos mientras aprendía de ellos. Agradezco también a mi profesor guía quien me dio la oportunidad de aprender y realizar un trabajo sobre un área poco explorada dentro de la carrera, pero que sin embargo es muy interesante.

RESUMEN

Resumen— El ordenamiento eficiente es uno de los problemas mas fascinantes en ciencias de la computación, existe una gran diversidad de algoritmos que emplean distintas técnicas con sus respectivas complejidades para enfrentar este problema. Entre las propuestas existentes, está la red de ordenamiento de K.E. Batcher basado en procesamiento paralelo para lograr el objetivo de ordenar. Dadas las limitaciones de hardware no es posible implementarlo en un computador clásico. Por esto se propone una adaptación de este algoritmo de ordenamiento haciendo uso de la propiedad de paralelismo en computación cuántica.

Palabras Clave—

- Computación cuántica
- Ordenamiento
- Redes de Baraja-Intercambio

ABSTRACT

Abstract— Efficient sorting is one of the most fascinating problems in computer science, there is a great diversity of algorithms that use different techniques with their respective complexities to face this problem. Among the existing proposals, there is the sorting network of K.E. Batcher that is based on parallel processing to achieve the sorting objective, which given the hardware limitations it is not possible to implement in a classic computer. This is why an adaptation of this sorting algorithm using quantum parallelism is proposed.

Keywords—

- Quantum Computation
- Sorting

- Shuffle-Exchange Networks

GLOSARIO

Por comodidad, las siglas usadas en este trabajo serán a partir de los términos en el idioma inglés.

Qubit: Quantum Bit.

QC: Quantum Computer.

QI: Quantum Information.

SEN: Shuffle Exchange Network.

QBS: Quantum Bitonic Sort.

QPS: Quantum Perfect Shuffle.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	VI
ÍNDICE DE FIGURAS	IX
ÍNDICE DE TABLAS	X
INTRODUCCIÓN	1
CAPÍTULO 1: ORDENAMIENTO	3
1.1 Complejidad de Algoritmos	3
1.2 Algoritmos de Ordenamiento	4
1.3 <i>Merge Sort</i>	5
1.4 Redes de Ordenamiento	6
1.5 <i>Shuffle Exchange Network</i>	6
1.6 <i>Bitonic Sort</i>	7
1.7 El <i>shuffle</i> Perfecto	8
1.8 <i>Shuffle</i> Perfecto + <i>Bitonic Sort</i>	9
1.8.1 Teorema de Batcher	11
CAPÍTULO 2: COMPUTACIÓN CUÁNTICA Y ORDENAMIENTO	14
2.1 El Qubit	14
2.2 Compuertas Cuánticas	15
2.2.1 Las compuertas de Pauli	15
2.3 Circuitos Cuánticos	17
2.4 Reversibilidad	18
2.5 Paralelismo Cuántico	18
2.5.1 Algoritmo de Deutsch-Jozsa	19
2.6 <i>Delay</i>	21
2.7 Ordenamiento cuántico	22
2.8 Algoritmo de ordenamiento cuántico	22
2.9 <i>Quantum Bitonic Sort</i>	23
2.10 <i>Quantum Perfect Shuffle</i>	23
CAPÍTULO 3: SOLUCIÓN PROPUESTA	25
3.1 <i>Quantum Bitonic Sort</i>	25
3.2 <i>Quantum Perfect Shuffle</i>	26
3.3 <i>Quantum Sort</i> (QPS+ QBS)	26
3.4 <i>Delay</i>	28

3.5	Implementación en <i>IBM Quantum Composer</i> para validación	28
3.5.1	QBS en <i>IBM Quantum Composer</i>	28
3.5.2	QBS + QPS en <i>IBM Quantum Composer</i>	28
CAPÍTULO 4: CONCLUSIONES Y TRABAJO FUTURO		30
REFERENCIAS BIBLIOGRÁFICAS		31

ÍNDICE DE FIGURAS

1.1	Símbolo para módulo comparador	8
1.2	Representación gráfica del <i>Shuffle</i> perfecto	9
1.3	Red que puede ordenar secuencias de largo N	10
1.4	Red de ordenamiento para 8 objetos basado en el ordenamiento bitónico de Batcher, todos los módulos son comparaciones-intercambios y los de color celeste son módulos que revierten su salida	10
2.1	Representación de un Qubit como <i>Bloch Sphere</i>	15
2.2	Circuito para la operación Swap entre dos qubits, a la izquierda se puede observar las compuertas aplicadas y a la derecha la representación tradicional para la operación	17
2.3	Representación de la compuerta controlled-U	18
2.4	Representación de la operación de medición	18
2.5	Circuito cuántico implementando el algoritmo general de Deutsch-Jozsa, el cable con la n representa un conjunto de n qubits	20
2.6	Algoritmo <i>Bitonic Sort + Perfect Shuffle</i>	22
2.7	Módulo comparador modo ascendente	23
2.8	Módulo comparador modo descendente	23
2.9	Circuito para <i>Perfect Shuffle</i>	24
3.1	Circuito para el módulo comparador de 2 qubits, q_0 y q_1 son los qubits a comparar y q_2 es el qubit que indica si el ordenamiento se hace de forma ascendente o descendente	25
3.2	Funciones para generar los intercambios mediante la operación <i>Swap</i>	26
3.3	Funciones para generar los intercambios mediante la operación <i>Swap</i>	26
3.4	Circuito <i>Shuffle</i> para 16 qubits	27
3.5	QPS + QBS para 4 qubits	27
3.6	QBS para ordenar de forma ascendente.	28

3.7 QBS para ordenar de forma descendente.	28
3.8 Visualización de ordenamiento para 4 qubits en <i>IBM Quantum Composer</i> . . .	29

ÍNDICE DE TABLAS

1.1 Complejidad temporal y espacial de algoritmos de ordenamiento	5
---	---

INTRODUCCIÓN

No hace falta dedicar muchas palabras a lo que es el problema del ordenamiento, que es bastante conocido en ciencias de la computación, donde se busca ordenar de manera eficiente. Se puede decir que aproximadamente el 25 % de la computación promedio funciona en base a ordenamiento y que es de gran utilidad cuando se necesita buscar y recuperar información. Existen diversas propuestas para enfrentar este problema, cada una de estas posee su respectiva complejidad, uso de memoria y desempeño frente a distintas situaciones, lo cual motiva a seguir desarrollando alternativas que busquen la mayor eficiencia al momento de aplicarse en la realidad.

Por otra parte, la computación cuántica es un tema que se lleva estudiando hace muchos años de forma teórica. En 1976 surgió por primera vez de la teoría de información cuántica [Ingarden, 1976] pero no fue hasta hace unos cuantos años que se comenzó a tener un acercamiento más práctico con computadores cuánticos reales que poseen una cantidad considerable de qubits (Bits cuánticos). Varios algoritmos se han desarrollado en el transcurso de estos 50 años, entre los que se destacan los algoritmos de Shor [Shor, 1994] y Grover [Grover, 1997] en la década de los 90, que fueron la base para muchos algoritmos desarrollados posteriormente y que a su vez entregan una visión clara del potencial que tiene un computador cuántico.

Cuando se ve la ventaja en complejidad al abordar problemas como la factorización o la búsqueda en secuencias de datos no ordenados, da para pensar en las posibles aplicaciones que se le puede dar al paralelismo presente en su forma de operar. En 1968 K. E. Batcher propone un método de ordenamiento diseñado para un procesador en paralelo [Batcher, 1968], que funciona a modo de barajar los datos e ir comparando entre pares de elementos para posteriormente realizar un intercambio entre los valores según corresponda con el objetivo de obtener el ordenamiento. Algoritmo pensado para un procesador paralelo por lo que no es aplicable en la computación clásica, pero podría ser llevado a cabo en un computador cuántico dada la capacidad de los qubits para ser procesados en forma paralela, modificando ciertos aspectos de la propuesta de Batcher, pero manteniendo la idea base, lo cual es el objetivo de este trabajo.

La computación cuántica se ha convertido en un tema tangible en los últimos años debido al desarrollo de tecnologías que han permitido la realización de computadores cuánticos reales, computadores con los que hasta ese entonces solo era posible trabajar vía simulación, donde claramente sus propiedades interesantes siguen siendo limitadas por las propiedades de la computación clásica. El lanzamiento de Qiskit y la disponibilidad de un computador cuántico a cualquier parte del mundo, ha motivado y ha permitido a muchas personas a introducirse en esta área, ya que dados los acontecimientos previos donde se demostró la superioridad absoluta de un computador cuántico frente a uno clásico para ciertas tareas específicas, dio para pensar e investigar en todas las posibles soluciones que se pueden diseñar para problemas en las que un computador clásicos no es eficiente. Cuando se habla de ordenamiento y paralelismo se debe acudir a la idea propuesta por

K.E. Batcher en el año 1968, que a partir de barajar los datos y hacer comparaciones, permite realizar la tarea de ordenamiento a muy baja complejidad, idea que no es posible llevar a cabo en un computador clásico que opera de forma secuencial y no paralelizada.

En el primer capítulo se introduce el problema del ordenamiento, donde se explica en detalle en qué consiste y las distintas formas con las que se aborda el problema, las situaciones en las que cada algoritmo es útil y sus respectivas complejidades y uso de memoria. En este capítulo también se describen las redes de ordenamiento, cómo operan, cuáles son sus componentes y la complejidad lograda al hacer uso de estas redes en la tarea de ordenar, además, se incorpora un teorema que respalda su funcionamiento de forma matemática. En el segundo capítulo se presentan los conceptos fundamentales de la computación cuántica, desde el bit cuántico en si y sus propiedades, hasta el desarrollo de circuitos cuánticos que permiten operar sobre estos bits, también se introducen las propiedades de estos circuitos y se explica el por qué posee tanto impacto al momento de enfrentar ciertos problemas. Donde se cuenta con paralelismo para operar sobre un conjunto de bits cuánticos lo cual permite el funcionamiento de ciertas técnicas como en el caso de este trabajo es el ordenamiento. También se describe el estado del arte en el problema de ordenamiento en los algoritmos cuánticos, donde se puede encontrar el primer acercamiento a la implementación de las redes de ordenamiento en circuitos cuánticos. El capítulo 3 contiene el diseño de los circuitos con sus respectivas mejoras a partir del estado del arte, y la implementación de dichos circuitos en un computador cuántico real haciendo uso de *IBM Quantum Composer*. Herramienta que permite ver en tiempo real el comportamiento de los qubits al pasar por los circuitos. Por último, se presenta el capítulo de Conclusiones y trabajo futuro.

CAPÍTULO 1

ORDENAMIENTO

1.1. Complejidad de Algoritmos

Cuando se habla de algoritmos es necesario introducir conceptos como la complejidad [Cormen *et al.*, 2022], en ciencias de la computación, la complejidad de un algoritmo se refiere a la cantidad de recursos computacionales, como tiempo y espacio, que requiere para resolver un problema en función del tamaño de entrada.

La notación Big O es una forma de describir la complejidad de un algoritmo. Se utiliza para proporcionar una cota superior asintótica en la cantidad de operaciones que realiza el algoritmo en el peor de los casos, a medida que el tamaño de entrada tiende hacia infinito.

Esta se representa como $O(f(n))$, donde $f(n)$ es una función que define la complejidad del algoritmo en términos del tamaño de entrada n . En la notación Big O, se ignoran los términos de menor importancia y se considera solo el término dominante que tiene el mayor impacto en la complejidad a medida que n crece.

Algunos ejemplos comunes de notación Big O son:

- $O(1)$ o complejidad constante: El algoritmo requiere una cantidad constante de tiempo o espacio, independientemente del tamaño de entrada. Ejemplo: acceder a un elemento en un arreglo por índice.
- $O(\log(n))$ o complejidad logarítmica: El algoritmo tiene un tiempo o espacio que crece de manera logarítmica en función del tamaño de entrada. Ejemplo: búsqueda binaria en un arreglo ordenado.
- $O(n)$ o complejidad lineal: El algoritmo tiene un tiempo o espacio proporcional al tamaño de entrada. Ejemplo: recorrer una lista una vez.
- $O(n^2)$ o complejidad cuadrática: El algoritmo tiene un tiempo o espacio que crece cuadráticamente en función del tamaño de entrada. Ejemplo: bucles anidados para comparar todos los pares de elementos en una lista.
- $O(2^n)$ o complejidad exponencial: El algoritmo tiene un tiempo o espacio que crece exponencialmente en función del tamaño de entrada. Ejemplo: algoritmos de fuerza bruta.

Esta notación ayuda a comparar la eficiencia relativa de los algoritmos y predecir cómo aumenta su tiempo de ejecución o uso de memoria a medida que crece el tamaño de entrada. En general, se busca utilizar algoritmos con una complejidad más baja (como $O(n \log(n))$) en lugar de aquellos con una complejidad más alta (como $O(n^2)$) para problemas grandes, ya que los algoritmos más eficientes tienden a escalar mejor.

1.2. Algoritmos de Ordenamiento

Los algoritmos de ordenamiento [Knuth, 1998] son una clase de algoritmos que se utilizan para ordenar una lista de elementos en un orden específico, como de menor a mayor o de mayor a menor. Estos algoritmos son esenciales para diversas aplicaciones informáticas, como la búsqueda, la compresión de datos, la criptografía, la detección de patrones, la estadística, entre otros.

Hay diversos algoritmos de ordenamiento, cada uno con sus propias ventajas y desventajas. En la Tabla 1, se ve una comparación de algunos de los algoritmos existentes. Algunos algoritmos son más rápidos que otros para ordenar grandes conjuntos de datos, mientras que otros son más eficientes para conjuntos de datos pequeños. Algunos algoritmos son más fáciles de implementar, mientras que otros son más complejos pero pueden proporcionar un rendimiento mejorado en ciertas situaciones.

Uno de los algoritmos de ordenamiento más simples y comunes es el algoritmo de ordenamiento burbuja (*bubble sort*), en el que se compara cada elemento de la lista con el siguiente y se intercambian si están en el orden equivocado. Este proceso se repite hasta que se haya pasado por toda la lista y se haya verificado que todos los elementos están en el orden correcto.

Otro algoritmo popular es el algoritmo de ordenamiento por selección (*selection sort*), en el que se busca el elemento más pequeño en la lista y se intercambia con el primer elemento. Luego, se busca el siguiente elemento más pequeño y se intercambia con el segundo elemento, y así sucesivamente hasta que toda la lista esté ordenada.

Un algoritmo de ordenamiento más eficiente es el algoritmo de ordenamiento rápido (*quick-sort*), que utiliza un enfoque de divide y conquista para ordenar la lista. Este algoritmo selecciona un elemento llamado "pivote" se dividen los elementos restantes en dos grupos: aquellos que son menores que el pivote y aquellos que son mayores. Luego, se aplica el mismo proceso a cada uno de estos grupos hasta que todos los elementos estén ordenados.

Otro algoritmo eficiente es el algoritmo de ordenamiento por mezcla (*merge sort*), en el que se divide la lista en dos mitades y se ordenan por separado. Luego, se combinan las dos mitades en una sola lista ordenada.

En resumen, los algoritmos de ordenamiento son herramientas esenciales en la progra-

mación y la informática en general. Los desarrolladores deben elegir cuidadosamente el algoritmo de ordenamiento adecuado para la tarea en cuestión, teniendo en cuenta factores como la cantidad de datos que se van a ordenar, la eficiencia deseada y la complejidad del algoritmo. A continuación se explica en detalle el funcionamiento del algoritmo *Merge Sort*, que tiene una estructura bastante similar a la de la solución propuesta en este trabajo.

Tabla 1.1: Complejidad temporal y espacial de algoritmos de ordenamiento

Algoritmo	Complejidad temporal	Complejidad espacial
Insertion Sort	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(1)$
Bubble Sort	$O(n^2)$	$O(1)$
Quick Sort	$O(n \log n)$ (promedio)	$O(\log n)$ (promedio)
Merge Sort	$O(n \log n)$	$O(n)$
Heap Sort	$O(n \log n)$	$O(1)$
Counting Sort	$O(n + k)$	$O(k)$
Radix Sort	$O(d \cdot (n + k))$	$O(n + k)$
Bucket Sort	$O(n^2)$ (promedio)	$O(n)$
Shell Sort	$O(n \log n)$ (mejor caso)	$O(1)$

1.3. Merge Sort

Merge Sort es un algoritmo de ordenamiento eficiente que utiliza el método de "dividir y conquistar". El proceso de ordenamiento se divide en dos partes: primero se divide la lista en dos mitades iguales y luego se ordenan cada una de las mitades de forma recursiva. Por último, se combinan las dos mitades en una sola lista ordenada.

El proceso de ordenamiento de *Merge Sort* se realiza en tres etapas:

1. División: Se divide la lista original en dos partes iguales, utilizando el concepto de "pivot". Se selecciona el punto medio de la lista y se divide en dos partes, una mitad izquierda y una mitad derecha. Si la lista no se puede dividir en partes iguales, la mitad izquierda tendrá un elemento más que la mitad derecha.
2. Ordenamiento: Cada mitad se ordena de forma recursiva utilizando el algoritmo de *Merge Sort*. Este proceso continúa hasta que las mitades tengan un solo elemento. En este punto, cada una de las mitades está ordenada.
3. Mezcla: Por último, se combinan las dos mitades ordenadas para crear una única lista ordenada. Se compara el primer elemento de cada mitad y se coloca el elemento

más pequeño en la lista final. Luego, se compara el siguiente elemento de la mitad correspondiente y se coloca en la lista final. Este proceso continúa hasta que se hayan mezclado todas las partes de ambas mitades en la lista final.

El proceso de mezcla es el aspecto más importante del *Merge Sort* y lo que lo hace tan eficiente. En lugar de comparar todos los elementos de la lista en cada paso, solo compara los elementos que están en la posición correspondiente en ambas mitades. Esto significa que el proceso de mezcla es muy rápido y eficiente.

El *Merge Sort* es un algoritmo de ordenamiento estable, lo que significa que si hay elementos iguales en la lista original, se mantendrá el orden relativo de esos elementos en la lista ordenada. Además, es un algoritmo de ordenamiento externo, lo que significa que puede ordenar listas que no caben en la memoria principal de la computadora, utilizando la memoria secundaria como disco duro. En la solución propuesta en las secciones futuras, el paso de división, no es necesario aplicarse, esto es la clave para reducir la complejidad de este algoritmo en la versión cuántica de ordenamiento.

1.4. Redes de Ordenamiento

Una de las ideas base fundamentales para el desarrollo de este trabajo, son las redes de intercambio u ordenamiento de Batcher [Batcher, 1968] quien en pocas palabras plantea un algoritmo capaz de ordenar una lista de bits siendo estos procesados de forma paralela tratando la lista como una baraja de cartas. En esta sección se describen los algoritmos mencionados y sus respectivas componentes y propiedades.

Si se busca obtener altas tasas de rendimiento, es importante considerar qué procesos se debe realizar de forma simultánea, a modo de obtener mejores tiempos de respuesta. Dada las limitantes conocidas para las unidades de procesamiento clásicas, diversos algoritmos, como por ejemplo, el de ordenamiento, no son realizables de forma óptima, cada método para ordenar elementos conlleva su respectiva complejidad y uso de memoria. Donde se puede destacar algoritmos como el *QuickSort* o *MergeSort*, mencionados en la sección anterior donde este último es muy parecido a la propuesta de este trabajo en cuanto a su estructura y funcionamiento.

1.5. *Shuffle Exchange Network*

Las redes de intercambio de mezcla (SEN, por sus siglas en inglés) son una clase de redes de interconexión comúnmente utilizadas en sistemas de computación paralelos. Se propusieron por primera vez a principios de la década de 1980 como una alternativa a las topologías de red tradicionales como el hipercubo o la malla.

Los SEN se componen de bloques de construcción simples llamados conmutadores de intercambio aleatorio”, que se pueden usar para redirigir datos entre nodos en la red. Los conmutadores se organizan en una serie de etapas, cada una de las cuales consta de un conjunto de conmutadores que conectan un subconjunto de los nodos de la red.

En una red de intercambio aleatorio, los datos se dividen inicialmente entre los nodos de la red y luego se mezclan a través de una serie de etapas de conmutación hasta que llegan a su destino. La ventaja clave de los SEN es su capacidad para proporcionar patrones de comunicación eficientes para ciertos tipos de algoritmos paralelos, particularmente aquellos que involucran intercambios de datos a gran escala entre nodos.

Los SEN se han utilizado en una variedad de aplicaciones, incluido el procesamiento de imágenes, simulaciones de redes neuronales y simulaciones científicas. Sin embargo, pueden ser relativamente complejos de implementar y analizar, y su rendimiento puede depender en gran medida de las características específicas de la aplicación que se ejecuta.

1.6. Bitonic Sort

Las SEN se pueden utilizar como una implementación de hardware de ciertos algoritmos de ordenamiento paralelo.

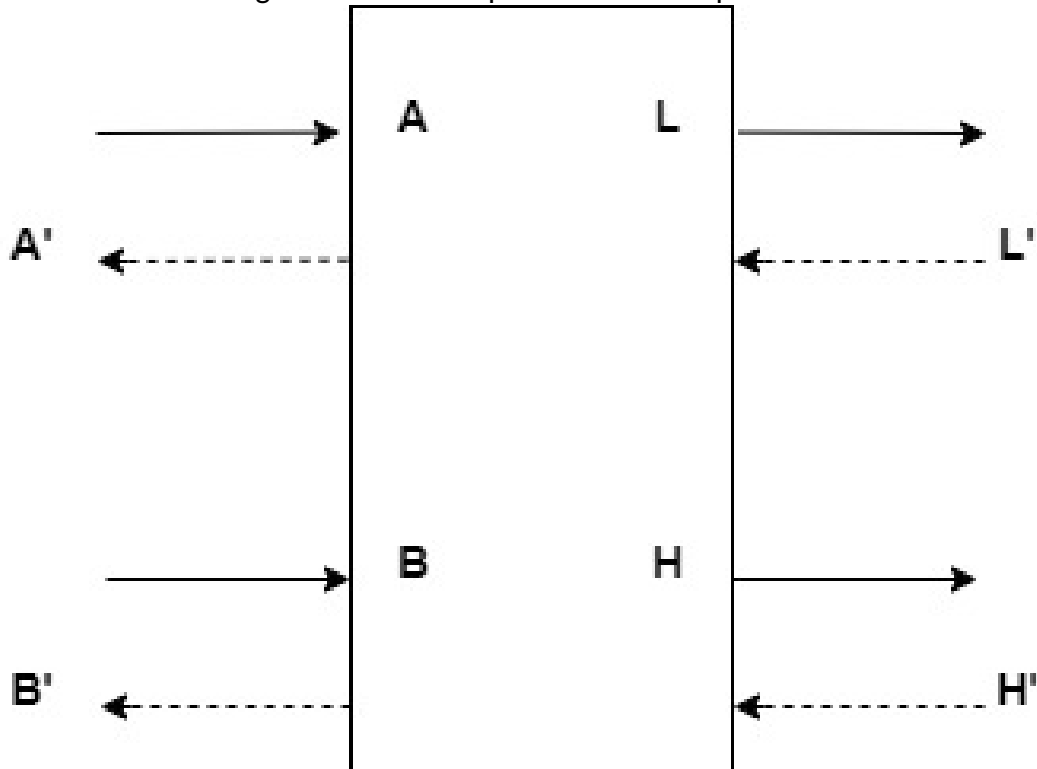
Un ejemplo es el algoritmo de ordenamiento bitónico de Batcher, que se basa en el algoritmo de ordenamiento bitónico que ordena una secuencia de números fusionando repetidamente pares de elementos adyacentes y luego ordenando recursivamente las subsecuencias resultantes. La versión paralela del algoritmo, el ordenamiento bitónico de Batcher, utiliza una SEN para implementar la fusión de pares adyacentes de elementos.

La implementación basada en SEN del ordenamiento bitónico de Batcher consta de múltiples etapas, cada una de las cuales implica mezclar e intercambiar datos entre nodos en la red. En cada etapa, la red realiza una operación de "fusión" comparando pares adyacentes de elementos y reordenándolos si es necesario. En la Figura 1.1 se observa un comparador. La red está configurada de tal manera que las operaciones de fusión se realizan en paralelo en todos los nodos, lo que resulta en una aceleración significativa en comparación con una implementación secuencial.

Otro ejemplo es el algoritmo de ordenamiento paralelo de *radix*, que ordena una secuencia de números dividiéndolos en grupos basados en sus representaciones binarias y luego ordenando recursivamente cada grupo. Una SEN se puede utilizar para implementar el paso de partición distribuyendo cada elemento al nodo correspondiente en la red en función de su representación binaria. Luego, la red se utiliza para realizar el paso de ordenamiento recursivo en paralelo en todos los nodos.

En general, el uso de SEN para algoritmos de ordenamiento puede ofrecer una aceleración

Figura 1.1: Símbolo para módulo comparador

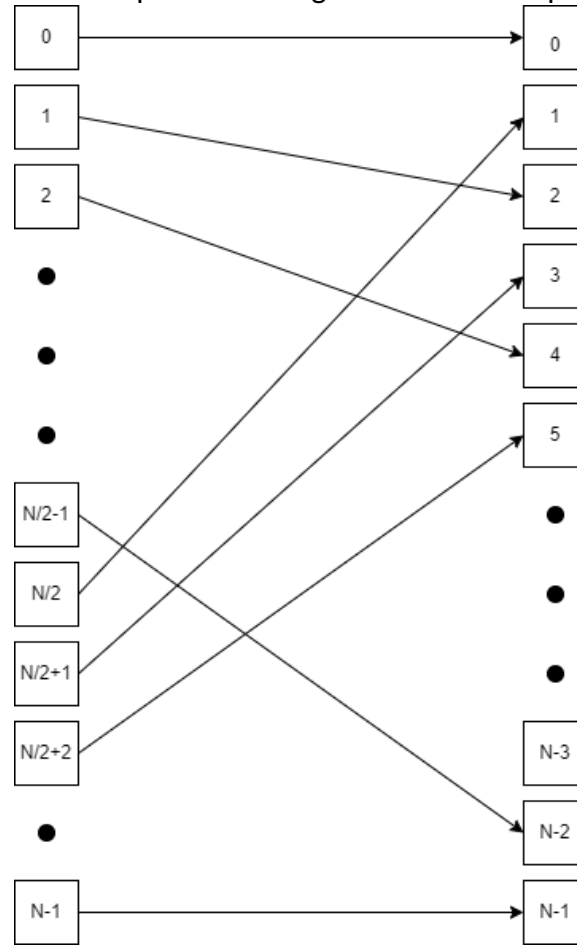


significativa en ciertos casos, especialmente al tratar con conjuntos de datos grandes que se pueden dividir y procesar en paralelo. Sin embargo, la complejidad de diseñar e implementar estas redes significa que típicamente solo se utilizan en aplicaciones especializadas donde sus ventajas superan los costos.

1.7. El *shuffle* Perfecto

El *shuffle* perfecto, también conocido como "intercambio de bits" o "mezcla perfecta", es un método de reordenamiento de datos utilizado en computación paralela y procesamiento de señales digitales. La idea detrás del *shuffle* perfecto es dividir los datos de entrada en grupos de bits y reorganizarlos de tal manera que los bits que originalmente estaban separados ahora estén juntos.

En el paper de Stone, se describe el *shuffle* perfecto como una operación que toma una entrada de datos de longitud N y la reordena en una salida de longitud N , de tal manera que cada bit de entrada esté en una posición única en la salida. Esto se logra mediante una serie de intercambios binarios, que se realizan de manera recursiva hasta que se logra la reorganización deseada.

Figura 1.2: Representación gráfica del *Shuffle* perfecto

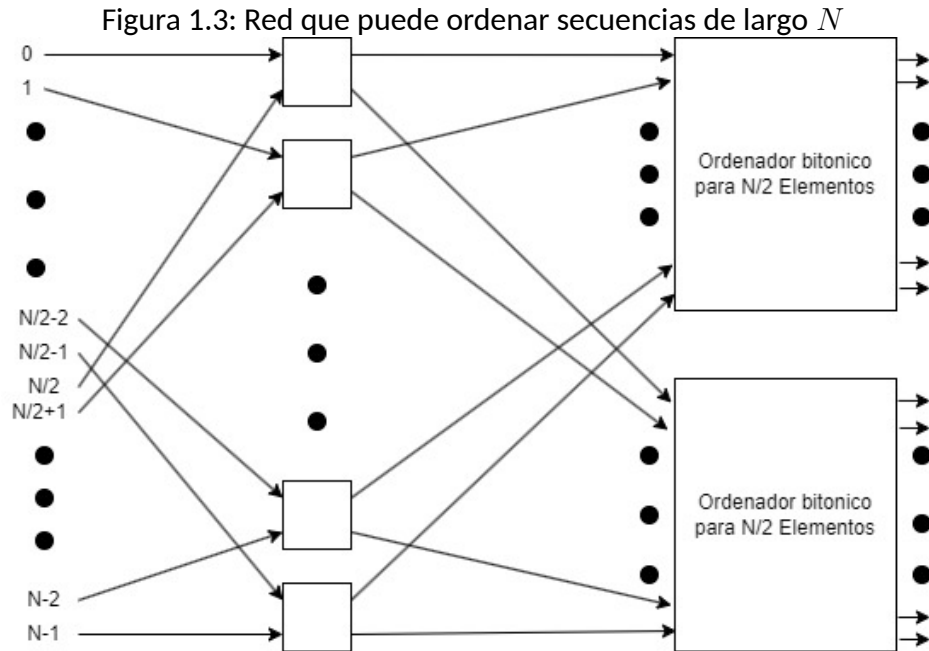
$$P(i) = \begin{cases} 2i & \text{si } 0 \leq i \leq N/2 - 1 \\ 2i - (N - 1) & \text{si } N/2 \leq i \leq (N - 1) \end{cases}$$

Donde N es la longitud total de los datos y i es el índice de la posición que se está reordenando. La función $P(i)$ toma un valor entero i y devuelve el valor entero de su posición después de la operación de *shuffle* perfecto.

1.8. *Shuffle* Perfecto + Bitonic Sort

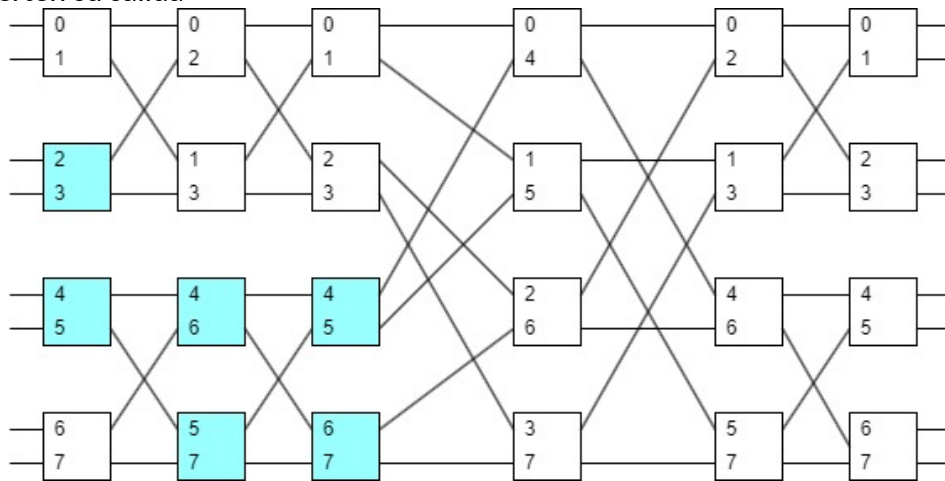
En la Figura 1.3 se muestra la estructura de un *sort* bitónico que permite ordenar secuencias de largo N .

Un ordenador completo para ocho elementos construido a partir de ordenadores bitónicos es mostrado en la Figura 1.4. Los módulos sombreados en la Figura, son los que realizan una operación de intercambio y dejan al mayor de los dos valores en la entrada en la parte supe-



rior de la salida. El primer conjunto de módulos ordena cuatro pares de números en orden ascendente o descendente tal que la salida del primer conjunto sea un par de secuencias bitónicas de largo 4. Los dos conjuntos siguientes ordenan las dos secuencias bitónicas de forma ascendente o descendente de largo 4, formando una secuencia bitónica de largo 8. Los siguientes tres conjuntos de módulos ordenan la secuencia bitónica de largo 8 en una secuencia ascendente de largo 8.

Figura 1.4: Red de ordenamiento para 8 objetos basado en el ordenamiento bitónico de Batcher, todos los módulos son comparaciones-intercambios y los de color celeste son módulos que revierten su salida



Para desarrollar las herramientas para describir y analizar este algoritmo de ordenamiento se incorporan las siguientes definiciones:

- Hay un índice i , donde $0 \leq i \leq N - 1$ tal que a_0 a través de a_i se incrementa monótonamente y a_i a través de a_{N-1} se decrementa monotónicamente: o si
- La secuencia anterior puede ser corrida cíclicamente para que la condición 1 sea satisfecha.

Ejemplos de secuencias bitónicas son las siguientes:

- 1, 3, 5, 7, 8, 6, 4, 2, 0
- 7, 8, 6, 4, 2, 0, 1, 3, 5
- 1, 2, 3, 4, 5, 6, 7, 8

El concepto central del *sort* bitónico es que la red mostrada en la Figura 1.4 puede ordenar una secuencia bitónica. La red realiza una comparación-intercambio de a_i y $a_{i+N/2}$, para $0 \leq i \leq N/2 - 1$. Se debe probar que poco después de comparar-intercambiar las subsecuencias de los primeros $N/2$ números y los últimos $N/2$ números son ambos bitónicos. Además, cada número en la primera subsecuencia no es mayor que cualquier otro número en la segunda subsecuencia. El *output* de la primera comparación-intercambio es pasado a un par de ordenadores bitónicos que operan en la mitad de muchos números. Usando una técnica de construcción recursiva se puede sintetizar la estructura de los ordenadores mas pequeños al copiar la estructura de los mas grandes. Un ordenador completo se puede construir desde muchos ordenadores bitónicos secuencialmente aplicados y uniendo las pequeñas secuencias en las mas largas hasta que se tiene una secuencia bitónica de tamaño N que puede ser puesta en una red como la mostrada en la Figura.

1.8.1. Teorema de Batcher

Sea $a = a_0, a_1, a_2, \dots, a_{N-1}$ una secuencia bitónica y sea $b_i = \min(a_i, a_{i+N/2})$, $c_i = \max(a_i, a_{i+N/2})$ para $0 \leq i \leq N/2 - 1$. Las dos secuencias $b = b_0, b_1, \dots, b_{N/2-1}$ y $c = c_0, c_1, \dots, c_{N/2-1}$ son bitónicas. Además, $b_i \leq c_j$ para todo i y j .

En la implementación del teorema de Batcher, se utilizan subsecuencias de longitud p , donde p es un divisor de n . En este caso señaladas como la subsecuencia b y c . Estas subsecuencias se ordenan de forma independiente utilizando el algoritmo de ordenamiento de Batcher, y luego se combinan utilizando una red de interconexión perfecta para obtener el conjunto de elementos ordenados completo.

Por ejemplo, si $n = 8$, se pueden utilizar subsecuencias de longitud 2 para ordenar los elementos del conjunto. En la primera etapa, se ordenan las subsecuencias (0,1), (2,3), (4,5) y (6,7) de forma independiente utilizando el algoritmo de ordenamiento de Batcher. En la siguiente etapa, se combinan las subsecuencias (0,2), (1,3), (4,6) y (5,7) utilizando la red de interconexión perfecta. Finalmente, se combinan las subsecuencias (0,4) y (1,5), y las subsecuencias (2,6) y (3,7) para obtener el conjunto de elementos completo y ordenado.

El teorema de Batcher establece que la red de ordenamiento de Batcher de tamaño n puede ordenar cualquier conjunto de elementos en un tiempo $O(\log^2(N))$ utilizando $O(N \log N)$ comparaciones. Esta eficiencia se logra al utilizar subsecuencias de longitud p , donde p es un divisor de n , para ordenar los elementos de forma independiente antes de combinarlos.

Prueba:

Para demostrar que la red de ordenamiento de Batcher de tamaño n puede ordenar cualquier conjunto de elementos en un tiempo $O(\log^2(N))$ utilizando $O(N \log N)$ comparaciones, se puede utilizar una demostración por inducción en N .

Base de la inducción: Para $N = 2$, la red de ordenamiento de Batcher consiste en un solo comparador que compara los dos elementos y los intercambia si es necesario. Esto toma un tiempo constante $O(1)$ y una sola comparación.

Hipótesis de inducción: Suponiendo que el teorema de Batcher es cierto para todos los valores de N hasta $k - 1$, donde k es una potencia de 2.

Paso inductivo: Para demostrar que el teorema de Batcher también es cierto para $N = k$, considere una secuencia de longitud k . Primero se divide la secuencia en dos subsecuencias de longitud $k/2$ y se ordena de forma independiente utilizando el teorema de Batcher. Esto toma un tiempo $O(\log^2(k/2)) = O(\log^2(k))$ y $O((k/2)\log(k/2)) = O(k \log k)$ comparaciones.

A continuación, se divide cada una de las subsecuencias de longitud $k/2$ en dos subsecuencias de longitud $k/4$ y se ordenan de forma independiente utilizando el teorema de Batcher. Esto toma un tiempo $O(\log^2(k/4)) = O(\log^2(k))$ y $O((k/4)\log(k/4)) = O(k \log k)$ comparaciones para cada subsecuencia, por lo que en total se realizan $O(k \log k)$ comparaciones.

Se continua dividiendo cada subsecuencia en subsecuencias más pequeñas y ordenándolas de forma independiente utilizando el teorema de Batcher hasta tener subsecuencias de longitud 2. En cada paso, la cantidad de subsecuencias se duplica y la longitud de las subsecuencias se divide por 2, por lo que se realizan $O(\log N)$ pasos.

Finalmente, se combinan las subsecuencias ordenadas utilizando la red de interconexión perfecta para obtener el conjunto de elementos completo y ordenado. La red de interconexión perfecta toma un tiempo $O(\log^2(N))$ y $O(N \log N)$ comparaciones.

En total, se realizan $O(\log N)$ pasos de ordenamiento de Batcher, cada uno tomando un tiempo $O(\log^2(N))$ y $O(N \log N)$ comparaciones, más la combinación utilizando la red de interconexión perfecta, lo que da un tiempo total de $O(\log^2(N))$ y $O(N \log N)$ comparaciones.

Por lo tanto, la demostración por inducción muestra que el teorema de Batcher es cierto para todas las potencias de 2, y por lo tanto para cualquier tamaño de conjunto de elementos.

CAPÍTULO 2

COMPUTACIÓN CUÁNTICA Y ORDENAMIENTO

Para comprender la idea de la propuesta, es necesario tener noción de lo que es la computación cuántica, la cual funciona en base a Quantum bits, luego explicar las compuertas lógicas cuánticas y como operan estas sobre los Quantum bits, que posteriormente permitirá construir el algoritmo de ordenamiento propuesto en este trabajo. Todas las definiciones y algunas de las imágenes enseñadas en este documento están en base al libro *Quantum Computation and Quantum Information* de Michael A. Nielsen e Isaac L. Chuang [Nielsen y Chuang, 2002] el cual se recomienda en caso de querer profundizar los conceptos abordados aquí.

2.1. El Qubit

El bit es el concepto fundamental de la computación e información clásica, la computación cuántica y la información cuántica están construidas sobre un concepto análogo denominado Quantum bit, al cual nos referiremos como Qubit. En esta sección se introduce las propiedades de uno o varios qubits, comparando y contrastando sus propiedades respecto a los bits clásicos.

Los qubits se describen como objetos matemáticos con ciertas propiedades específicas, lo cual da la libertad de construir una teoría general de la QCQI. Dos posibles estados para el qubit son $|0\rangle$ y $|1\rangle$, que corresponde a los estados 0 y 1 para un bit clásico. La notación $|\psi\rangle$ es conocida como *Dirac notation* la cual se usa para representar los estados con los que se trabaja. Un qubit no solo se encuentra en los estados $|0\rangle$ y $|1\rangle$ que son los estados fundamentales base, sino que se puede encontrar en la forma de combinación lineal que viene dada por:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (2.1)$$

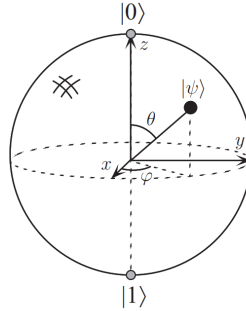
Los números α y β son números complejos, aunque no hay mucha pérdida al considerarlos números reales. Dicho todo esto es necesario recalcar que los estados intermedios como combinaciones lineales, son denominados estados en superposición. Al ser observado el qubit, este colapsa resultando en $|0\rangle$ o $|1\rangle$ con probabilidad $|\alpha|^2$ o $|\beta|^2$ respectivamente. Y dado que $|\alpha|^2 + |\beta|^2 = 1$ Se puede reescribir la ecuación (2.1) como:

$$|\psi\rangle = e^{i\gamma} \left(\cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle \right) \quad (2.2)$$

Para representar esto con matemáticas de forma menos engorrosa se usa los vectores ortogonales $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ y $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Se usa esta notación para representar estados de qubits 0 o

1, de modo de distinguirlos de cosas como los valores de bit 0 y 1 o los números 0 y 1. Los números θ y φ son números reales que definen un punto sobre la esfera unitaria tridimensional, como la mostrada en la Figura 2.1, la cual es llamada *Bloch Sphere* que provee una útil forma de visualizar el estado de un solo qubit.

Figura 2.1: Representación de un Qubit como *Bloch Sphere*



2.2. Puertas Cuánticas

Una característica importante de las puertas cuánticas en comparación con las clásicas es que en el proceso desde que se inicializan los qubits y son medidos finalmente, las operaciones (puertas cuánticas) son siempre reversibles, estas puertas reversibles pueden ser representadas como matrices y actúan como rotaciones sobre la *Bloch Sphere* presentada anteriormente.

2.2.1. Las puertas de Pauli

Si se tiene conocimientos de álgebra lineal, las matrices de Pauli probablemente resulten familiares, En este contexto las matrices de Pauli representan puertas lógicas aplicadas sobre un qubit. Por ejemplo:

La puerta X Es representada por la matriz de Pauli X :

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0| \quad (2.3)$$

Para ver los efectos que tiene una puerta sobre un qubit, simplemente se debe multiplicar el vector del estado del qubit por la matriz que representa la puerta:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \quad (2.4)$$

La compuerta X es comúnmente llamada la *NOT-gate* en referencia a la versión clásica.

Las compuertas Y & Z Al igual que con la compuerta X, las matrices Y & Z también actúan como las compuertas Y & Z en los circuitos cuánticos:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = -i|0\rangle\langle 1| + i|1\rangle\langle 0| \quad (2.5)$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = |0\rangle\langle 0| + |1\rangle\langle 1| \quad (2.6)$$

Estas 3 compuertas pueden ser vistas como rotaciones por π en torno al eje X, Y Z de la *Bloch Sphere* respectivamente.

La compuerta de Hadamard Esta compuerta es fundamental, y permite crear una superposición de los estados $|0\rangle$ y $|1\rangle$. Su matriz viene dada por:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (2.7)$$

Esta compuerta realiza las siguientes transformaciones:

$$H|0\rangle = |+\rangle \quad (2.8)$$

$$H|1\rangle = |-\rangle \quad (2.9)$$

Donde $|+\rangle$ y $|-\rangle$ describen los estados:

$$|+\rangle = |0\rangle + |1\rangle \quad (2.10)$$

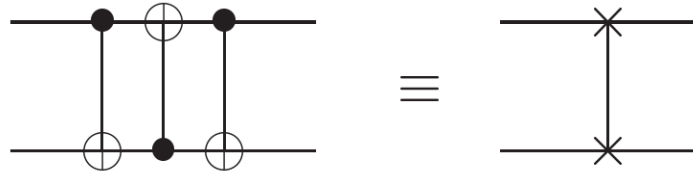
$$|-\rangle = |0\rangle - |1\rangle \quad (2.11)$$

Existen numerosas mas compuertas cuánticas, pero de momento basta con tener claro el como funcionan las básicas para un Qubit. Otras puertas bastante comunes en los circuitos se presentan en las siguientes secciones, ya que a partir de ahora se habla de circuitos cuánticos compuestos por más de un Qubit.

2.3. Circuitos Cuánticos

En esta sección se observa en detalle los elementos presentes en un circuito cuántico, en la figura 2.2, se puede ver un circuito compuesto por 3 compuertas. Este circuito debe leerse de izquierda a derecha, cada línea en el circuito representa un cable. Este cable no necesariamente corresponde a un cable físico, podría corresponder a un paso del tiempo, o tal vez a una partícula física como un protón o una partícula de luz moviéndose de una ubicación a otra en el espacio. Por convención se asume que el estado del input en el circuito es un estado base computacional, usualmente los estados consisten de puros $|0\rangle$.

Figura 2.2: Circuito para la operación Swap entre dos qubits, a la izquierda se puede observar las compuertas aplicadas y a la derecha la representación tradicional para la operación



Existe ciertas características permitidas en los circuitos clásicos que no se presentan en los circuitos cuánticos, por ejemplo, no se permiten los ciclos, es decir feedback de un circuito desde una parte del circuito a otra. Se dice que los circuitos son acíclicos, tampoco soporta operaciones como el FANIN o FANOUT que son operaciones usadas en los circuitos clásicos, las cuales son operaciones irreversibles. En realidad se tiene demostrado que la mecánica cuántica prohíbe el copiar un Qubit, para mayor detalle sobre estas propiedades se recomienda leer las secciones correspondientes en el libro.

A medida que se avance se introducirán nuevas compuertas, es conveniente introducir otra convención sobre los circuitos cuánticos, la cual es ilustrada en la Figura 2.3. Suponga que U representa cualquier matriz unitaria actuando en un número n de qubits, tal que U puede ser vista como una compuerta aplicada a esos qubits. Entonces se puede definir una compuerta *controlled- U* que es una extensión natural de la compuerta *Controlled-Not*, donde la compuerta tiene un qubit de control, indicada por la línea con el punto negro y n *target qubits*. Si el qubit de control es 0, entonces nada pasa con el qubit objetivo, de ser 1 entonces U es aplicado al qubit objetivo.

Otra operación importante es la medición que se representa con el símbolo de un medidor como se muestra en la Figura 2.4. Cuya función es convertir un vector de estado en un bit clásico con las probabilidades mencionadas en la sección anterior.

Figura 2.3: Representación de la compuerta controlled-U

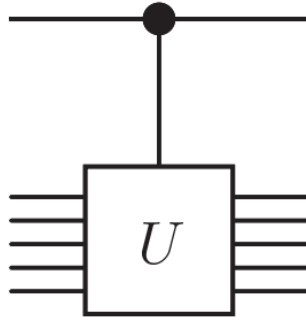
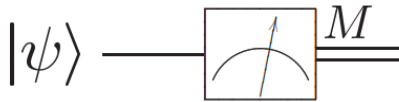


Figura 2.4: Representación de la operación de medición



2.4. Reversibilidad

En la computación clásica, los circuitos lógicos son usualmente irreversibles porque se pierde información en ciertos pasos del procesamiento de datos, lo que no representa un problema en la práctica. Sin embargo, en la computación cuántica, los circuitos deben ser reversibles para cumplir con las leyes de la física cuántica y evitar la pérdida de información.

En la computación cuántica, el estado final de un sistema cuántico depende del estado inicial del sistema y de las operaciones realizadas sobre él. Debido a la naturaleza probabilística de la mecánica cuántica, cualquier operación irreversible o eliminación de información en un circuito cuántico tendría el efecto de romper la coherencia cuántica y, por lo tanto, interferir con el cálculo.

Además, los circuitos cuánticos necesitan ser reversibles porque la evolución de un sistema cuántico es reversible según las leyes de la física cuántica. En otras palabras, cualquier estado cuántico puede evolucionar de manera determinista hacia su estado anterior. Por lo tanto, para evitar la pérdida de información y mantener la coherencia cuántica, los circuitos cuánticos deben ser diseñados de tal manera que todas las operaciones sean reversibles.

2.5. Paralelismo Cuántico

Una de las características fundamentales para este trabajo es el paralelismo cuántico, lo cual es la base para muchos algoritmos cuánticos. Heurísticamente y bajo el riesgo de sobre-

simplificar, el paralelismo cuántico permite evaluar una función $f(x)$ para diferentes valores de x de manera simultánea, en esta sección se explicará como funciona el paralelismo cuántico a partir de un algoritmo bastante conocido en la computación cuántica.

2.5.1. Algoritmo de Deutsch-Jozsa

Este algoritmo fue el primer ejemplo de un algoritmo cuántico que posee un desempeño mejor al del mejor algoritmo clásico. Muestra las ventajas que puede haber al usar un computador cuántico como una herramienta computacional para un problema específico.

El Problema Deutsch-Jozsa Dada una función Booleana f , que toma como entrada una serie de bits y retorna 1 o 0, esto es:

$$f(\{x_0, x_1, x_2, \dots\}) \rightarrow 0 \vee 1,$$

Donde x_n es 0 o 1

La propiedad de esta función booleana es tal que garantiza ser constante o balanceada. Una función constante retorna ya sea solo 1s o solo 0s para cualquier entrada, mientras que una función balanceada retorna para la mitad de los valores 0s y para la otra mitad 1s. La tarea consiste en determinar si la función es balanceada o constante.

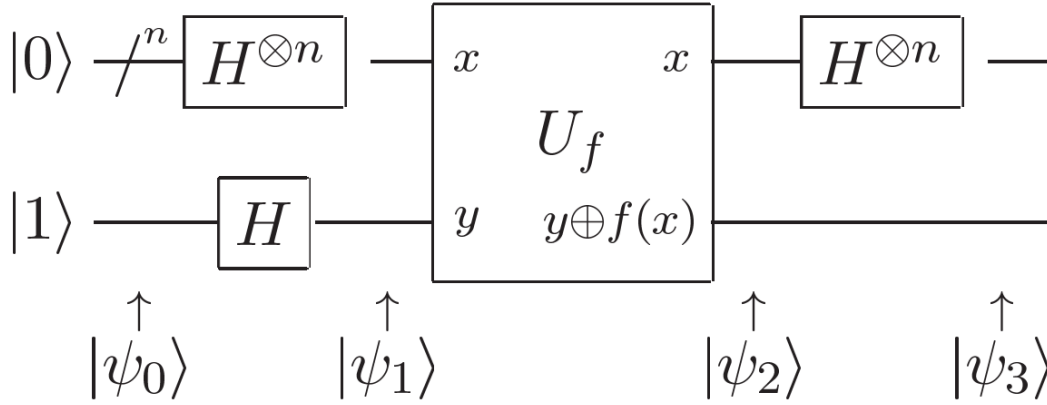
Tener en cuenta de que el problema de Deutsch-Jozsa es una extensión de n -qubits del problema de Deutsch.

La Solución Clásica Clásicamente en el mejor de los casos es necesario de hacer dos solicitudes a un oráculo para poder determinar si la función booleana, $f(x)$, es balanceada, por ejemplo si se obtiene para $f(0, 0, 0, \dots) \rightarrow 0$ y $f(1, 0, 0, 0, \dots) \rightarrow 1$ entonces la función está balanceada ya que se obtuvo dos salidas distintas. En el peor de los casos si se continúa viendo la misma salida para cada entrada que se intente, entonces se tendría que revisar la mitad de todas las entradas y una más para garantizar que la función es constante.

La Solución Cuántica Usando un computador cuántico se puede resolver este problema con toda confianza tras haber realizado solo un llamado a la función $f(x)$. Se cuenta con una función f implementada como un oráculo cuántico, que asigna los estados $|X\rangle|y\rangle$ a $|x\rangle|y \oplus f(x)\rangle$, donde $\oplus$ es la adición módulo 2. En la Figura 2.5 se ve una representación del circuito cuántico para el algoritmo de Deutsch-Jozsa.

Ahora corresponde explicar el como se desenvuelve este algoritmo paso a paso.

Figura 2.5: Circuito cuántico implementando el algoritmo general de Deutsch-Jozsa, el cable con la n representa un conjunto de n qubits



1. Se preparan dos registros cuánticos, el primero es un registro de n -qubit inicializado en $|0\rangle$ y el segundo es un simple qubit inicializado en $|1\rangle$

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

2. Aplicando la compuerta de Hadamard a estos qubits se obtiene:

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

3. Aplicando el oráculo cuántico $|X\rangle|y\rangle$ a $|x\rangle|y \oplus f(x)\rangle$:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle (|f(x)\rangle - |1 \oplus f(x)\rangle) = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle (|0\rangle - |1\rangle)$$

4. En este punto el segundo qubit simple puede ser ignorado, se aplica la compuerta Hadamard a cada uno de los qubits del primer registro:

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \left[\sum_{y=0}^{2^n-1} (-1)^{xy} |y\rangle \right] = \frac{1}{2^n} \sum_{y=0}^{2^n-1} \left[\sum_{x=0}^{2^n-1} (-1)^{f(x)} (-1)^{xy} \right] |y\rangle$$

Donde $xy = x_0y_0 \oplus x_1y_1 \oplus \dots \oplus x_{n-1}y_{n-1}$ es la suma de los productos entre los bits.

5. Medir el primer registro, notar que la probabilidad de medir $|0\rangle = \left| \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \right|^2$, que evalúa para 1 si $f(x)$ es constante y 0 si $f(x)$ es balanceada.

Una de las conclusiones más importantes sobre el algoritmo de Deutsch-Jozsa es que demuestra la capacidad de la computación cuántica para acelerar la resolución de problemas en comparación con los algoritmos clásicos equivalentes. En el caso del algoritmo de Deutsch-Jozsa, el algoritmo cuántico puede determinar si una función es constante o equilibrada con una sola consulta, mientras que los algoritmos clásicos equivalentes requieren al menos dos consultas para llegar a una conclusión.

Otra conclusión importante es que el algoritmo de Deutsch-Jozsa se basa en la propiedad de superposición cuántica y la interferencia cuántica. Esto significa que el algoritmo utiliza el hecho de que un qubit puede estar en múltiples estados cuánticos simultáneamente, y que estos estados pueden interferir entre sí para producir resultados finales que son diferentes de los resultados clásicos.

En resumen, el algoritmo de Deutsch-Jozsa es un importante hito en la investigación de la computación cuántica porque demuestra la ventaja cuántica en la resolución de problemas. Además, el algoritmo se basa en propiedades cuánticas fundamentales como la superposición y la interferencia, lo que demuestra la capacidad de la computación cuántica para aprovechar estas propiedades para la resolución de problemas.

2.6. Delay

Debido a la naturaleza cuántica de los qubits, los algoritmos para computadores cuánticos pueden ser más susceptibles a errores y ruido en comparación con los algoritmos clásicos. Esto significa que se requiere una corrección de errores cuidadosa para mantener la integridad de los cálculos. Además, los qubits pueden tener tiempos de vida muy cortos, lo que significa que los cálculos deben realizarse rápidamente antes de que los qubits pierdan su información.

Todo esto puede resultar en un *delay* o retardo en la ejecución de los algoritmos cuánticos. Se debe trabajar para minimizar los errores y el ruido, y maximizar la eficiencia en el uso de los qubits disponibles. A medida que la tecnología de la computación cuántica avanza, es posible que se desarrollen nuevas técnicas para reducir o eliminar estos retrasos.

Una medida común del tiempo de ejecución de un algoritmo cuántico es el número de puertas cuánticas necesarias para completar la tarea. Otra medida común es el número de iteraciones que se necesitan para realizar una tarea. Por ejemplo, en algunos algoritmos de optimización cuántica, se realizan múltiples iteraciones para encontrar el mejor resultado posible. Cada iteración puede requerir varias operaciones cuánticas y mediciones, lo que puede aumentar el tiempo de ejecución total del algoritmo.

2.7. Ordenamiento cuántico

Actualmente para el problema del ordenamiento en computadores cuánticos, existen varias propuestas para abordar este problema, se plantean algoritmos de ordenamiento para objetos abstractos basado en el funcionamiento de la computación cuántica como tal, haciendo uso de oráculos como compuertas para obtener resultados deseados. Pero entre las propuestas sólo una existente toma este problema a nivel de qubits, también haciendo uso de las SEN, propuesta presentada en esta sección y considerada como otra base para desarrollar la propuesta del presente trabajo.

2.8. Algoritmo de ordenamiento cuántico

En el trabajo de [Mahmud *et al.*, 2019] se plantea el algoritmo mostrado en la Figura 3.6, que logra el ordenamiento con gran eficiencia, para esta tarea se hace uso de las SEN, donde se hace una primera adaptación a los circuitos cuánticos. Para datos codificados en n qubits, el algoritmo se descompone en m etapas donde $m = \log_2(n)$. Cada etapa, s , consta de $m-s$ *quantum perfect shuffles* consecutivos (QPS) operaciones seguidas por s QPS-comparaciones de pares, lo que resulta en un total de m QPS y s operaciones de comparación (ver Figura 3.6). Cada comparación consta de $n/2$ circuitos comparadores y se aplica ya sea una operación min-max o max-min sobre dos qubits dependiendo de un modo establecido. El *mode* 0 corresponde a una operación min-max y el *mode* 1 corresponde a una operación max-min.

Figura 2.6: Algoritmo *Bitonic Sort + Perfect Shuffle*

```

for s = 1 to m do
  mode(s)
  for i = 1 to m - s do
    QPS(qubits)
  end for
  for i = m - s + 1 to m do
    QPS(qubits)
    comp(qubits, mode)
    QPS(mode)
  end for
end for

```

2.9. Quantum Bitonic Sort

Para el módulo comparador de qubits se proponen los circuitos presentados en las Figura 2.7 y 2.8, donde según el qubit *mode* se logra realizar ordenamiento ascendente o descendente.

Figura 2.7: Módulo comparador modo ascendente

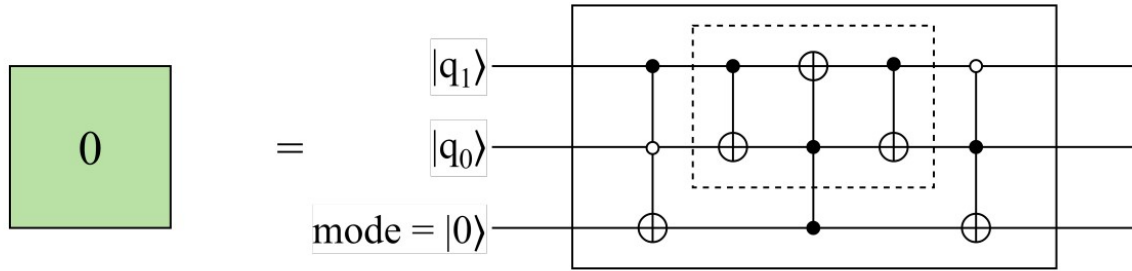
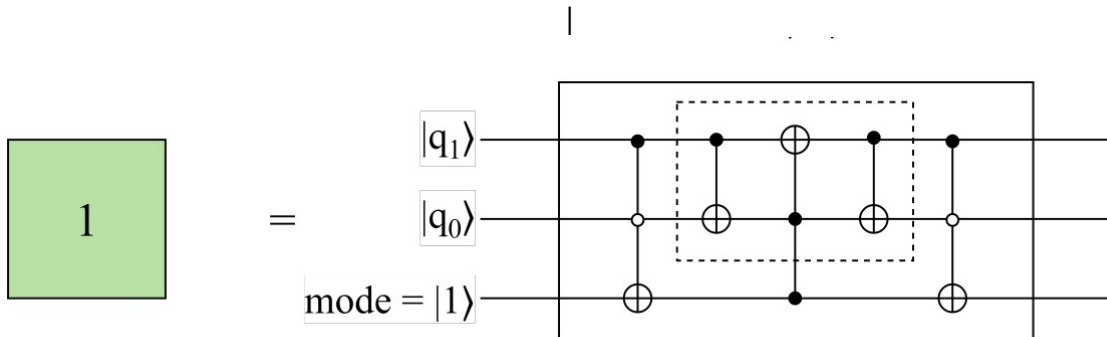
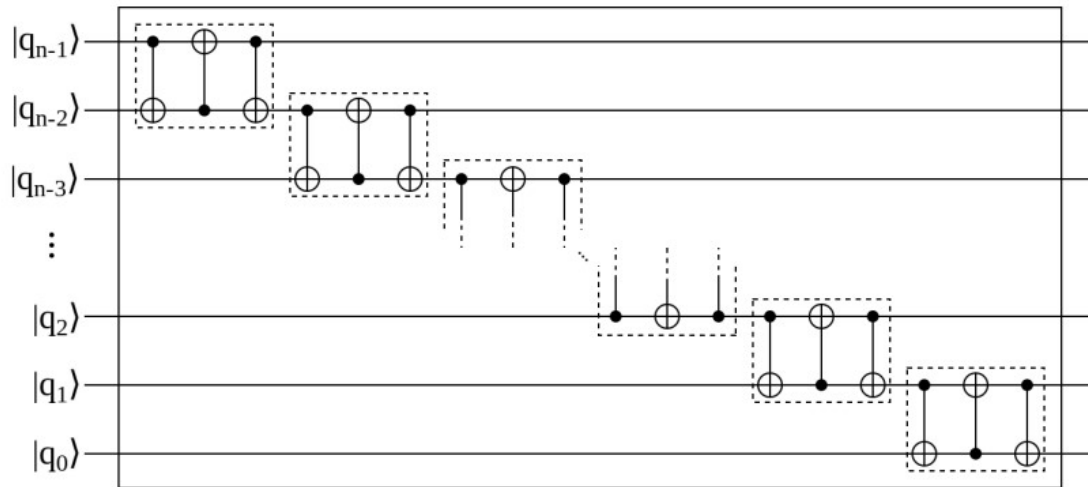


Figura 2.8: Módulo comparador modo descendente



2.10. Quantum Perfect Shuffle

Para el *Perfect Shuffle* se plantea un circuito que permuta los qubits en forma de cascada haciendo uso de compuertas *Swap* entre cada qubit consecutivo, como se observa en la Figura 2.9.

Figura 2.9: Circuito para *Perfect Shuffle*

CAPÍTULO 3

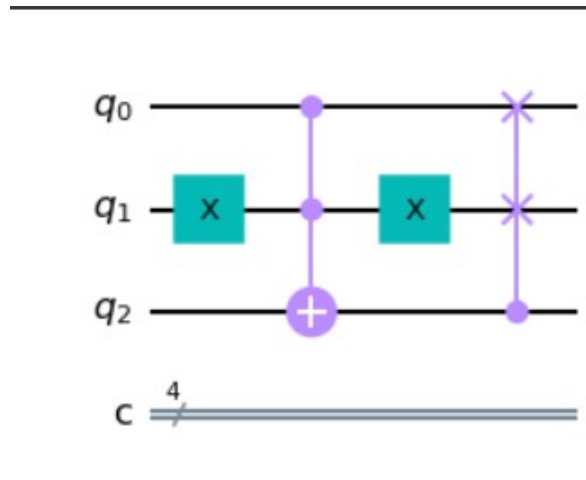
SOLUCIÓN PROPUESTA

Tras explicar las componentes de las redes de ordenamiento de Batcher, corresponde presentar las respectivas adaptaciones en base a los circuitos cuánticos que fueron introducidos en un comienzo y las implementaciones ya realizadas en el estado del arte, los cuales permiten construir el algoritmo que logra ordenar qubits.

3.1. Quantum Bitonic Sort

En primer lugar se diseña un módulo para comparar dos qubits y ordenarlos de forma ascendente o descendente, unificando los dos circuitos presentados anteriormente. Para esto se hace uso de 3 qubits, los dos a comparar y uno auxiliar que permite realizar las operaciones de comparación e intercambio de ser necesario. En la Figura 3.1, se observa el circuito, el cual comprueba inicialmente si los qubits son distintos haciendo uso de una compuerta de Toffoli, la cual activa el qubit auxiliar que, de activarse, este realiza un intercambio según si el tercer qubit es 1 o 0 el cual indica si el ordenamiento se hace de forma ascendente o descendente. El qubit 1 es necesario pasarlo por una puerta *Not* dos a modo de lograr detectar cuando es distinto al qubit 0.

Figura 3.1: Circuito para el módulo comparador de 2 qubits, q_0 y q_1 son los qubits a comparar y q_2 es el qubit que indica si el ordenamiento se hace de forma ascendente o descendente



3.2. Quantum Perfect Shuffle

Para realizar la operación de barajar los qubits, a diferencia de la propuesta en el estado del arte que solo intercambia en forma de cascada no haciendo un real *Shuffle*, es necesario diseñar un circuito que realice las operaciones de intercambio, esto se logra mediante compuertas *Swap* aplicadas en los distintos qubits, para construir estos circuitos se crearon funciones en Python usando la librería Qiskit (Figura 3.2 y 3.3) la cual de acuerdo a los índices genera los intercambios entre qubits, cabe señalar que estos intercambios se muestran de forma separada, pero en la práctica los que no son aplicados sobre un mismo qubit, se realizan de forma simultánea, logrando una complejidad de $\log(N)$.

Figura 3.2: Funciones para generar los intercambios mediante la operación *Swap*

```
def generate_swaps_from_base(base: int, qubit_length: int) -> list:
    last_swap = base * 2
    swaps = []
    swaps.append((base, last_swap))
    while last_swap != base:
        tmp_swap = last_swap * 2
        if tmp_swap >= qubit_length:
            tmp_swap = last_swap * 2 + 1 - qubit_length
        swaps.append((base, tmp_swap))
        last_swap = tmp_swap
    return swaps[:-1]

def QPS_swaps(qubit_length: int) -> list:
    swap_initiators = [value for value in range(1, qubit_length//2, 2)]
    print("Initiators are", swap_initiators)
    swaps = []
    for base in swap_initiators:
        swaps += generate_swaps_from_base(base, qubit_length)
    return swaps
```

Figura 3.3: Funciones para generar los intercambios mediante la operación *Swap*

```
def generate_QPS_circuit(qubit_length: int) -> QuantumCircuit:
    qreg_q = QuantumRegister(qubit_length, 'q')
    creg_c = ClassicalRegister(qubit_length, 'c')
    qps = QuantumCircuit(qreg_q, creg_c)
    swaps = QPS_swaps(qubit_length=qubit_length)
    for swap in swaps:
        qps.swap(qreg_q[swap[0]], qreg_q[swap[1]])
    return qps
```

En la Figura 3.4, se observa un circuito generado con una operación *Shuffle* para 16 qubits.

3.3. Quantum Sort(QPS+ QBS)

Finalmente combinando estos dos circuitos, se pueda generar uno nuevo que permite realizar las operaciones *Shuffle* y luego una comparación entre cada par de qubits, logrando así el ordenamiento. Por practicidad se muestra en la Figura 3.5 el circuito para ordenar 4 qubits.

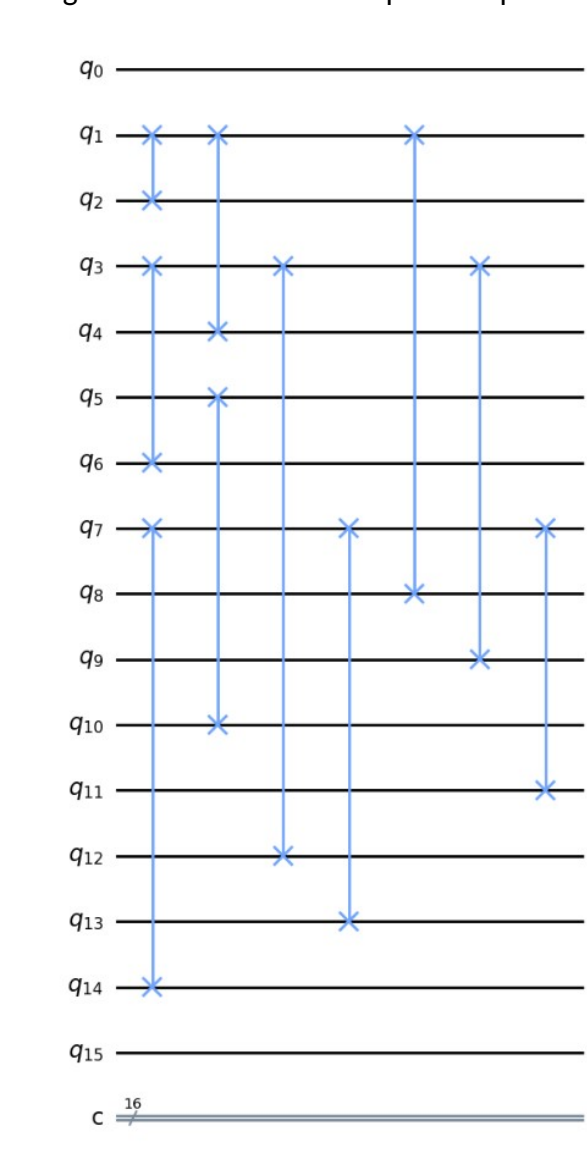
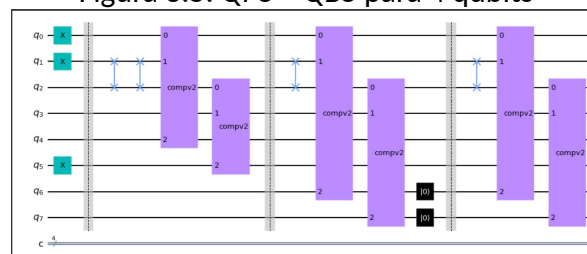
Figura 3.4: Circuito *Shuffle* para 16 qubits

Figura 3.5: QPS + QBS para 4 qubits



3.4. Delay

Uno de los aspectos a considerar en el desarrollo de algoritmos cuánticos, tal como se mencionó en el marco conceptual, es el *delay*, en este caso para cada algoritmo se puede medir el *delay* de acuerdo a la cantidad de compuertas por las que debe pasar el conjunto de qubits. Para el *sort* bitónico, el *delay* es de 3, en el caso del QBS+QPS, considerando tanto al QBS como al QPS, como compuertas únicas, el *delay* global del algoritmo es un total de $\text{Log}(N)$.

3.5. Implementación en IBM Quantum Composer para validación

Para validar estos circuitos, se hace uso de los computadores cuánticos de IBM, los cuales están disponibles para uso gratuito. Sin embargo, la cantidad de qubits disponibles para experimentos es limitado. Para ver los resultados en tiempo real, se hace uso de *IBM Composer*, una herramienta que permite monitorear en vivo el como cambian los qubits al interactuar con las compuertas.

3.5.1. QBS en IBM Quantum Composer

En los circuitos de la Figura 3.6 y 3.7, se pueden observar los circuitos para ordenar dos qubits de forma ascendente y descendente, esto solo cambiado el valor inicial del qubit auxiliar.

Figura 3.6: QBS para ordenar de forma ascendente.

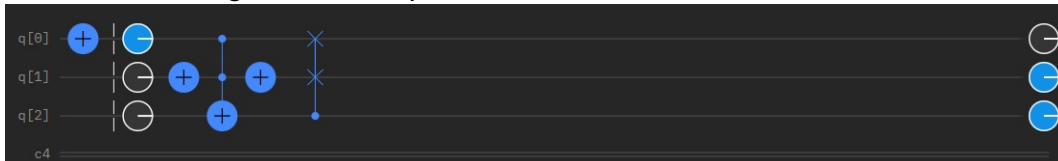
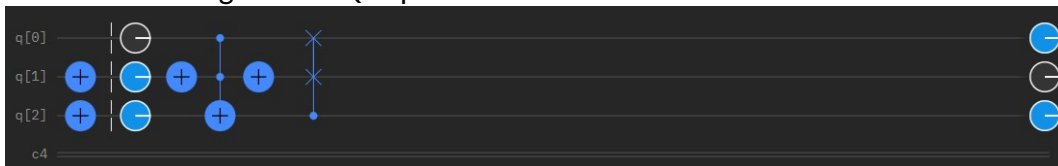


Figura 3.7: QBS para ordenar de forma descendente.

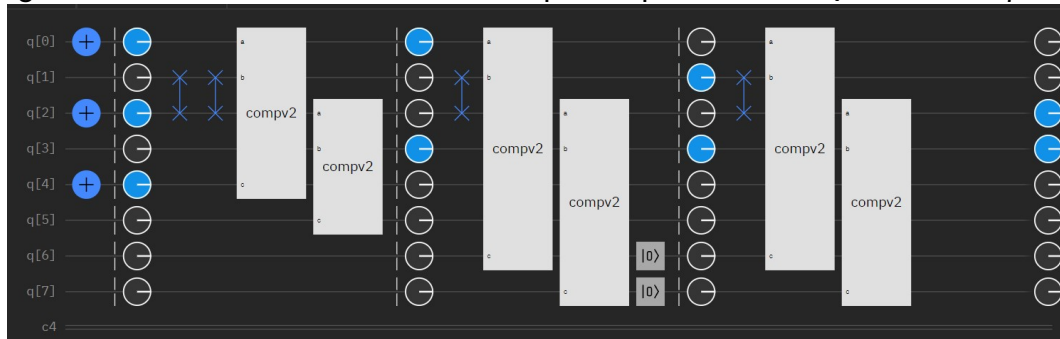


3.5.2. QBS + QPS en IBM Quantum Composer

El *IBM Composer* para su visualización está limitado a 8 qubits, por lo que solo se pone a prueba el algoritmo en el ordenamiento de 4 qubits para el *Bitonic Sort* junto al QPS, debido

a que se requieren de 4 qubits auxiliares para ordenar los 4 objetivos. En la Figura 3.8 a la izquierda se pueden observar desde el qubit 0 hasta el qubit 3, los qubits a ordenar, donde tras pasar por todas las compuertas quedan de forma ordenada ascendentemente. Los otros 4 qubits sirven para auxiliar las compuertas donde se realiza el ordenamiento bitónico. Los cuales son los que después van quedando inutilizables, por lo que es necesario volver a dejarlos en 0 tras el segundo paso por la compuerta.

Figura 3.8: Visualización de ordenamiento para 4 qubits en *IBM Quantum Composer*



CAPÍTULO 4

CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se realizó la tarea de ordenamiento en un computador cuántico, donde los objetos a ordenar son qubits. Este ordenamiento se logró con una complejidad de $\text{Log}(N)$ para cada módulo de intercambio y ordenamiento, un *delay* de $\text{Log}(N)$ para cada *Shuffle*, esto es una mejora técnica en cuanto a complejidad respecto a los algoritmos clásicos conocidos, lo cual es un indicador de que la información destinada a procesar en un algoritmo cuántico, se pueda ordenar dentro de la misma unidad de procesamiento cuántica, para lograr mejores tiempos de cómputo.

Tal como se señaló en un inicio, el ordenamiento es base importante para muchos algoritmos de la computación en general y la computación cuántica no es la excepción, la entrada o salida para los algoritmos cuánticos debe ser legible para un humano y para esto es fundamental poder lograr ordenar la información desde su base y por supuesto también de forma eficiente, sin embargo, en este ultimo punto aun queda mucho trabajo que realizar, como se mencionó en la sección de la solución, el comparador aun cuenta con qubits que tras la operación quedan inutilizables, lo cual es un problema consistente en los distintos algoritmos cuánticos que se han desarrollado hasta la fecha. Es por esto que el trabajo futuro se debería centrar no solo en el diseño de nuevos algoritmos, sino también en la mejora y optimización de los ya existentes, a fin de reducir la cantidad de qubits utilizados en realizar una operación, también es necesario considerar el problema con la generación y corrección de errores en estos algoritmos al ser implementados en circuitos mas grandes con una mayor cantidad de qubits, lo cual de momento es difícil de poner a prueba, dado el limite de recursos disponibles para los experimentos, como se mencionó en la sección de validación, el máximo de qubits a prueba fue de 8, lo cual es un numero muy pequeño para concluir respecto al desempeño del algoritmo planteado. Sin embargo es, cosa de tiempo y no mucho, para que se desarrollen y se hagan disponibles nuevos computadores con mayor capacidad a modo de poder poner a prueba no solo este algoritmo, sino la gran variedad existente en la actualidad, gracias a los múltiples aportes alrededor del mundo en el desarrollo de esta disciplina de la información. En la actualidad ya se sabe y se ha experimentado con la teletransportación cuántica y sus posibles usos, el potencial de esto es enorme y sin ninguna duda revolucionará la forma de comunicarse. Reiterando en la idea anterior, es necesario que esta información sea entendible para los humanos en determinado punto, es por eso que así como se debe desarrollar operaciones y algoritmos que aprovechen el paradigma cuántico, también se debe adaptar los algoritmos clásicos para poder ser llevados a este paradigma, porque es posible que eventualmente dada la ley de Moore, sea mas rentable migrar la computación clásica a la computación cuántica, con el fin de conseguir la mayor cantidad de registros dentro de una unidad única de procesamiento.

REFERENCIAS BIBLIOGRÁFICAS

- [Batcher, 1968] Batcher, K. E. (1968). Sorting networks and their applications. En *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), p. 307–314, New York, NY, USA. Association for Computing Machinery.
- [Cormen et al., 2022] Cormen, T. H., Leiserson, C. E., Rivest, R. L., y Stein, C. (2022). *Introduction to algorithms*. MIT press.
- [Grover, 1997] Grover, L. K. (1997). Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79:325–328.
- [Ingarden, 1976] Ingarden, R. S. (1976). Quantum information theory. *Reports on Mathematical Physics*, 10(1):43–72.
- [Knuth, 1998] Knuth, D. E. (1998). *The art of computer programming: Volume 3: Sorting and Searching*. Addison-Wesley Professional.
- [Mahmud et al., 2019] Mahmud, N., Srimoungchanh, B., Haase-Divine, B., Blankenau, N., Kuhnke, A., y El-Araby, E. (2019). Combining perfect shuffle and bitonic networks for efficient quantum sorting. En *2019 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, pp. 42–49.
- [Nielsen y Chuang, 2002] Nielsen, M. A. y Chuang, I. (2002). Quantum computation and quantum information. *American Journal of Physics*, 70(5):558–559.
- [Shor, 1994] Shor, P. (1994). Algorithms for quantum computation: discrete logarithms and factoring. En *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134.