Quantum B+ Tree

ABSTRACT

Quantum computing is a popular topic in computer science. In recent years, many new studies came out in different areas such as machine learning, network and cryptography. However, the topic of quantum data structures seems long neglected. There is an open problem in the database area: Can we make an improvement on existing data structures by quantum techniques? Consider a dataset of key-record pairs. Given an interval as a query range, a B+ tree can report all the records with keys within this interval, which is called a range query. A classical B+ tree answers a range query in $O(\log_B N + k)$ time, where B is the branching factor of the B+ tree, N is the total number of records, and k is the number of records in the interval. It is asymptotically optimal in a classical computer but not efficient enough in a quantum computer. In this paper, we proposed the quantum range query problem. To solve the problem, we first propose the static quantum B+ tree that answers a static quantum range query in $O(\log_B N)$ time, which is exponentially faster than the classical data structure and asymptotically optimal in quantum computers. To achieve this significant improvement on range queries, we design a hybrid quantum-classical algorithm to do the range search on the quantum B+ tree. Furthermore, we extend it to a dynamic quantum B+ tree. The dynamic quantum B+ tree performs an insertion or a deletion in $O(\log_B N)$ time and answers a dynamic quantum range query in $O(\log_B^2 N)$ time. We also discuss the high-dimensional quantum range query. Based on the quantum B+ tree, we propose the quantum range tree which answers a d-dimensional quantum range query in $O(\log_B^d N)$ time, which cannot be achieved by any classical data structure. In the experiment, we did simulations to show that to answer a range query, the quantum B+ tree is up to 1000× faster than the classical B+ tree. The quantum B+ tree is the first tree-like quantum data structure that achieves better complexity than classical data structures as far as we know.

ACM Reference Format:
. . Quantum B+ Tree. In *Proceedings of* . ACM, New York, NY, USA, 15 pages.

1 INTRODUCTION

Consider a dataset of key-record pairs. The range query problem, which is to report all the key-record pairs with keys within a given range, has been widely used in many real-world applications. For example, a teacher may need to list all the students who obtained 40-60 marks in an exam, a smartphone buyer may need to list all the smartphones that sit around \$200-\$400, and a traveler may want to list all the nearby spots.

Consider the scenario that a user Alice wants to find an interesting movie. Each movie is described as a key-record pair where the key is the release date of the movie and the record is a feature vector which contains some attributes such as the name, the genre, and the cast. Alice may want a movie of the 1990s. In the literature [21, 30], we usually first find out all the movies from 1990 to 1999 as the candidates, then give recommendations based on these candidates. The interval $[1990, 1999]$ given by Alice is the query range, where 1990 is the lower bound and 1999 is the upper bound. A range query returns all the movies with release dates within the query range.

To answer range queries, many data structures [6, 20, 50] have been proposed to store the key-record pairs. One representative data structure is the B+ tree. The B+ tree [20] is a widely-used existing data structure that answers a range query in $O(\log_B N+k)$ time, where B is the branching factor of the B+ tree, N is the total number of key-record pairs, and k is the number of records in the query range. Based on Yao's proof [67], it can be proved that the B+ tree is asymptotically optimal for range queries in classical computers. However, efficiency issues occur in quantum computers. Recently, quantum algorithms have attracted more and more attention. Many quantum algorithms [28, 59] have been proposed and are expected to show quadratic or even exponential speedup compared to classical algorithms, so many linear-complexity problems can be solved in sub-linear time in quantum computers. For example, following [15], consider using the KNN to give movie recommendations based on the candidates returned by a range query. Yue et al. [52] proposed a quantum KNN algorithm that only costs $O(d^3)$ time, where d is the dimension of the feature vector. Since $k = O(N)$, the classical B+ tree answers a range query in $O(N)$ time in the worst case. This linear-time data-preprocessing step will ruin all the advantages of the sublinear-time quantum algorithm. In fact, any classical range query algorithm has the same issue, which limits the potential of quantum algorithms in real-world applications.

Motivated by the serious limitations of the classical range query in quantum computers, we want to combine the advantages of data structures and quantum computation. Therefore, we proposed the problem called the *quantum range query*, which returns the answer in quantum bits. Specifically, the quantum range query $QUERY(x, y)$ does not return a list of key-record pairs, but returns quantum bits in a superposition of all the key-record pairs with keys within the query range $[x, y]$, where superposition is the ability of a quantum system to be in multiple states simultaneously. For example, consider a movie dataset $\{(key_0, rec_0), (key_1, rec_1), \cdots, (key_{N-1}, rec_{N-1})\}$, where N is the number of movies and each movie is described as a key-record pair as above mentioned. Assume we can construct a data structure in the quantum computer to store all the key-record pairs. If Alice wants to obtain all the movies of the 1990s, a quantum range query $QUERY(1990, 1999)$ will search on the data structure and return quantum bits in a superposition of all the movies (key_i, rec_i) whose key_i is within the interval $[1990, 1999]$. Motivated by the different scenarios in real-world applications, we study the quantum range query problems in three different cases. The first case is that the dataset is immutable, so the problem is called the *static quantum range query*. The second case is that insertions and deletions are supported in the dataset,

, ,

and the problem is called the *dynamic quantum range query*. The third case is that the keys in key-record pairs are high-dimensional points (e.g., the locations of restaurants), so the problem is called *high-dimensional quantum range query*.

The quantum range query problem has two distinctive characteristics. The first distinctive characteristic is that it allows the utilization of data structures. Existing quantum algorithms cannot solve this problem efficiently. There are many quantum search algorithms [13, 24, 28, 29], but all of them focused on the unstructured search, where the dataset is unsorted and we cannot assume that the key-record pairs are stored in a data structure. By the study in [13], we can obtain a quantum range query algorithm which returns k key-value pairs in $O(\sqrt{Nk})$ time. Although the cost of a range query grows sublinearly with k, this quantum range query algorithm cannot outperform its classical competitors such as the B+ tree. The main reason is that the existing quantum search algorithms do not take the advantage of data structures, and the unstructured search hardly occurs in real-world applications.

The second distinctive characteristic is the use of quantum computation. Motivated by the quantum algorithms in other research fields, we have a question: Can we use the quantum techniques on the data structures to improve the complexity? The existing classical data structures [6, 20, 50] are asymptotically optimal in classical computers, but there is a lot of room for improvement in quantum computers as above mentioned. Since the number of quantum states increases exponentially when the number of quantum bits increases linearly, the quantum computer is supposed to help a lot of tasks. Quantum search should also be considered on data structures, but this topic has never been studied before.

Therefore, we first propose the *static quantum B+ tree*, which is the first tree-like quantum data structure to our best knowledge. We choose the B+ tree [20] as the first data structure to study in quantum computers. Since the B+ tree is the most fundamental and widely-used data structure, it is considered the most suitable one to open a new world of quantum data structures. To take the full advantages of quantum parallelism, we design a hybrid quantum-classical algorithm to answer the static quantum range query. The hybrid design is very popular, especially in quantum machine learning. According to [1], many studies such as [2, 7, 16, 45, 52, 56, 57] used hybrid quantum-classical algorithms to do machine learning, since this design can reduce the circuit depth so that high performance can be obtained. The quantum range query algorithm has the following two main steps.

(1) Global classical search: We do a breadth-first search on the B+ tree to narrow the search range, where the search range is the range of all the tree nodes in the BFS queue. When we find that the ratio of the query range to the search range becomes high enough, we turn to the local quantum search;
(2) Local quantum search: Starting from the tree nodes from the global classical search, we perform quantum operations to access all the leaves below these tree nodes. Then, we do a post-selection to obtain the answer.

We proved that a static quantum B+ tree answers a static quantum range query in $O(\log_B N)$ time and it is asymptotically optimal in quantum computers, where B is the branching factor and N is the number of records. The static quantum B+ tree achieves an exponential speedup compared to a classical B+ tree. We also discuss the dynamic quantum range query. A *dynamic quantum B+ tree* can perform classical insertions and deletions. For any key-record pair (key, rec), we use $INSERT(key, rec)$ or $DELETE(key, rec)$ to insert or delete it in $O(\log_B N)$ time, and the complexity of a dynamic quantum range query is $O(\log_B^2 N)$. Furthermore, we extend the quantum B+ tree to a d-dimensional *quantum range tree*, which answers a d-dimensional range query in $O(\log_B^d N)$ time.

To the best of our knowledge, we are the first to study the quantum range query problem, and we are the first to propose a tree-like quantum data structure. We consider that the first tree-like quantum data structure will open a new world of quantum data structures. Unlike the classical range query, a quantum range query returns answers in quantum bits. We consider the quantum output very useful, and discuss the real-world utilization in Section 4.2. In summary, our contributions are shown in the following:

- We are the first to study the quantum range query problems.
- We are the first to propose a quantum tree-like data structure, which is the quantum B+ tree.
- We design a hybrid quantum-classical algorithm that can answer a range query on a static quantum B+ tree in $O(log_B N)$ time, which is exponentially faster than a classical B+ tree and asymptotically optimal in quantum computers.
- We further make the static quantum B+ tree support insertions and deletions in $O(\log_B N)$ time, and the complexity of the range query becomes $O(\log_B^2 N)$.
- We extend the quantum B+ tree to the quantum range tree, which answers a d-dimensional range query in $O(\log_B^d N)$ time.
- We conducted the experiment to confirm the exponential speedup of the quantum range queries both in 1-dimensional case and 2-dimensional case on real-world datasets. In the experiment, the quantum range query is up to 1000× faster than a classical range query.

The rest of the paper is organized as follows. In Section 2, we introduce some basic knowledge used in this paper about quantum algorithms. In Section 3, we introduce some related works. In Section 4, we define the problem and further discuss the problem. In Section 5, we show the design of a quantum B+ tree. In Section 6, we do simulations to show the efficiency and correctness of our algorithms. In Section 7, we draw a conclusion.

2 PRELIMINARIES

The bit is a basic unit when we want to store data in the memory or on disks. The bit in classical computers usually has two *states*, which are 0 and 1. In a quantum computer, we also have the bit, which is called the *quantum bit*. For simplicity, we use *qubit* in the following. Similar to the bit in a classical computer, the qubit also has states. To describe the states of a qubit, we usually use the *Dirac notation* [22] which looks like $|\psi\rangle$. For example, $|0\rangle$ and $|1\rangle$ are two states of a qubit. This corresponds to the state 0 and state 1 of a classical bit. Different from the classical bit, the state of a qubit can be 'between' the $|0\rangle$ and $|1\rangle$. We can express this *superposition* state of a qubit with a linear combination of the two basis states: $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$. α and β are two complex numbers, which are usually called the *amplitudes*. We have $|\alpha|^2 + |\beta|^2 = 1$, where $|\alpha|^2$ means the absolute square of α. As Dirac suggested in [6], we call the column vector a *ket*. Like reading a classical bit, we can measure a qubit. The measurement will collapse the superposition of the qubit so that we can only get the result state 0 with probability $|\alpha|^2$ or state 1 with probability $|\beta|^2$. For example, there is a qubit:

$$|\psi\rangle = (0.36 + 0.48i)\,|0\rangle + (0.48 - 0.64i)\,|1\rangle\,.$$

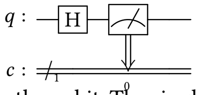
If we measure this qubit, we can get 0 with probability $|0.36 + 0.48i|^2 = 0.36^2 + 0.48^2 = 0.36$ or 1 with probability $|0.48 - 0.64i|^2 = 0.48^2 + 0.64^2 = 0.64$. Since the probability should sum to 1, $|0.36 + 0.48i|^2 + |0.48 - 0.64i|^2 = 1$ holds.

Considering there are two basis states of a qubit, if we have two qubits, the basis states will become $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. If we measure a qubit, the state of another qubit could be changed. Such a phenomenon is called *quantum entanglement* [55]. For example, assume there are two qubits q_0 and q_1:

$$\begin{aligned}|q_1q_0\rangle &= \frac{1}{\sqrt{2}}\,|0\rangle\,(0.6\,|0\rangle + 0.8\,|1\rangle) + \frac{1}{\sqrt{2}}\,|1\rangle\,(\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle)\\ &= \frac{0.6}{\sqrt{2}}\,|0\rangle + \frac{0.8}{\sqrt{2}}\,|1\rangle + \frac{1}{2}\,|2\rangle + \frac{1}{2}\,|3\rangle\end{aligned}$$

If we measure q_1, we will get 0 or 1 with probability 0.5. If the result is 0, $|q_0\rangle = 0.6\,|0\rangle + 0.8\,|1\rangle$ and if the result is 1, $|q_0\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$. Similarly, we can extend this system to n qubits $q_{n-1}, q_{n-2}, \cdots q_0$ and they have 2^n basis states and corresponding 2^n amplitudes.

Similar to the classical computer, we also have wires and gates in quantum computers. The wire starting at the left hand side and ending at the right hand size is time. The gates in quantum computers are called *quantum gates*, which is an instruction at the time. For example, *Hadamard gate* [31, 48] is a very useful quantum gate, which turns $|0\rangle$ into $\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$ and turns $|1\rangle$ into $\frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle$. The following shows an example of the visualization of a quantum circuit which consists of a qubit, a classical register, a Hadamard gate, and a measurement.



As above mentioned, q is the qubit. The single-line wire is a timeline, which denotes the order of the instructions coming to the qubit. The double-line wire denotes the classical register with 1 bit. The box containing H denotes the Hadamard gate. The box containing a meter denotes a measurement, which will store the result into the 0-th bit in the classical register c.

Here, a Hadamard gate can be represented by the Dirac notation:

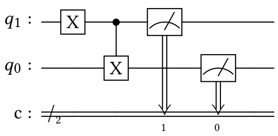
$$|0\rangle \rightarrow \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle\,;|1\rangle \rightarrow \frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle\,.$$

For example, given an input $|q\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$, before the measurement, then quantum bit will become

$$|q\rangle = \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle) + \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle) = |0\rangle\,.$$

If we measure the result, we obtain 0.

Also similar to the classical computer, the gates in the quantum computer can be applied on multiple qubits. The following quantum circuit shows an example.



We have two qubits q_0 and q_1 in this example. We apply an *X-gate* [48] (also known as a NOT-gate) on q_0, and a *controlled-X* gate on both q_0 and q_1. The X-gate is to swap the amplitudes of $|0\rangle$ and $|1\rangle$. We can represent it by the Dirac notation: $|0\rangle \longrightarrow |1\rangle\,;|1\rangle \longrightarrow |0\rangle$. The controlled-$X$ gate has two parts: q_1 is the *control* qubit and q_0 is the *target* qubit. The controlled-X gate applies an X-gate on the target qubit if and only if the controlled qubit is $|1\rangle$, otherwise keep the target qubit unchanged. We can represent it by the Dirac notation: $|0\rangle\,|q_0\rangle \longrightarrow |0\rangle\,|q_0\rangle\,;|1\rangle\,|q_0\rangle \longrightarrow |1\rangle\,X\,|q\rangle_0$. In programming language, we can express this operation as: If $q_1 = 1$, then $q_0 =$ NOT q_0. Consider the following three inputs:

(1) $|q_1q_0\rangle = |00\rangle$. After the X-gate on q_1, $|q_1q_0\rangle = |10\rangle$. Since $|q_1\rangle = |1\rangle$, apply an X-gate on q_0, so the result is $|11\rangle$.
(2) $|q_1q_0\rangle = |11\rangle$. After the X-gate on q_1, $|q_1q_0\rangle = |01\rangle$. Since $|q_1\rangle = |0\rangle$, do nothing, so the result is $|01\rangle$.
(3) $|q_1q_0\rangle = \frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|11\rangle$. After the X-gate on q_1, $|q_1q_0\rangle = \frac{1}{\sqrt{2}}\,|10\rangle + \frac{1}{\sqrt{2}}\,|01\rangle$. The result is $\frac{1}{\sqrt{2}}\,|11\rangle + \frac{1}{\sqrt{2}}\,|01\rangle\,.$

Usually, we describe such a quantum operation consisting of a series of quantum gates as a *quantum oracle*. A quantum oracle can be regard as a black box of a quantum circuit which we focus on the quantum operation it can perform but do not need to know the quantum circuit in detail. The concept of the quantum oracle has been widely used in many studies such as [28, 59, 64, 68]. Since different gate sets (which correspond with instruction sets in classical computers) may cause different time complexity and the general quantum computer is still at a very early stage, we usually use the query complexity to study quantum algorithms, which is to measure the number of queries to the quantum oracles. In the quantum algorithm area, many studies such as [33, 36, 39, 41, 42, 46, 47] assume that quantum oracles costs $O(1)$ time, and analyze the time complexity based on this assumption.

3 RELATED WORK

The classical range query problem has been studied for decades but it is not such a popular topic now. The reason is that a lot of different data structures have been proposed to solve this problem in different scenarios and there is very little room for further significant improvement.

For the static range query and the dynamic range query, the B-tree [6] and the B+ tree [20] are widely-used. The B+ tree is a tree data structure which can find a key in $O(\log_B N)$ time, where N is the number of key-record pairs and B is the branching factor. Then, $O(k)$ time is needed to load the k key-record pairs in the query range, so the range query on a B+ tree costs $O(\log_B N + k)$ time. The membership problem is another problem that given a query key, we need to decide if the key exists in the data structure. Yao [67] proved that for a large key space, any cell-probe data structure needs $\Omega(\log N)$ time to answer a membership problem in the worst case, where N is the size of the data structure. Since a membership problem can be transformed into a point query, $\Omega(\log N)$ is also the lower bound on a tree data structure, so the B+ tree is asymptotically optimal for range queries in classical computers.

For the high-dimensional range query, Bentley [8] proposed the range tree to answer it in $O(\log^d N + k)$ time, where d is the dimension of the keys. In addition, the range tree needs $O(N \log^{d-1} N)$

storage space. Then, Chazelle further improved the complexity. Chazelle proposed the lower bounds for the d-dimensional cases: $O(k+polylog(N))$ time complexity with $\Omega(N(\log N/\log\log N)^{d-1}))$ storage space in [17] and $\Omega((\log N/\log(2C/N))^{d-1}+k)$ time complexity with C units storage space in [19] where the lower bound of query I/O cost is provably tight for $C=\Omega(N(\log N)^{d-1})$.

However, all the above classical data structures have the same problem: The execution time grows linearly with k, which makes them useless for quantum algorithms. For example, the $O(\log N)$ quantum algorithm for linear systems of equations [32], the $O(\sqrt{N})$ quantum Bayesian inference algorithm [44], and the $O(\log N)$ quantum support vector machine [51] are all sub-linear time algorithms. If a linear time is needed to load the data, the advantages of sub-linear time algorithms will disappear. So, the quantum data structures for range queries should be considered.

The quantum database searching problem is also very popular in the quantum algorithm field. Grover's algorithm is described as a database search algorithm in [28]. It solves the problem of searching a marked record in an unstructured list, which means that all the N records are arranged in random order. On average, the classical algorithm needs to perform $N/2$ queries to a function f which tells us if the record is marked. More formally, for each index i, $f(i)=1$ means the i-th record is marked and $f(i)=0$ means the i-th record is unmarked. Note that only one record is marked in the database. If we have a quantum circuit to calculate this function, then we can build a Grover oracle $G:|x\rangle\rightarrow(-1)^{f(x)}|x\rangle$. Taking the advantage of quantum parallelism, Grover's algorithm can find the index of the marked record with $O(\sqrt{N})$ queries to the oracle. The main idea is to first "flip" the amplitude of the answer state and then reduce the amplitudes of the other states. One such iteration will enlarge the amplitude of the answer state and $O(\sqrt{N})$ iterations should be performed until the probability that the qubits are measured to be the answer comes close to 1. Then, an improved Grover's algorithm was proposed in [13]. We are also given the function f to mark the record, but k records are marked at this time. The improved Grover's algorithm can find one of the k marked records in $O(\sqrt{N/k})$ time. If we make the function f to mark the records with keys within a query range, then this algorithm returns one of the k key-record pairs in $O(\sqrt{N/k})$ time, and it needs $O(\sqrt{Nk})$ time to answer a range query.

The execution time of the quantum database search algorithm grows linearly with the square root of k, which is better than the classical data structures. However, the time complexity is even worse than the classical structures. Due to this observation, although Grover described the Grover's algorithm as a database search algorithm in [28], we do not consider it capable of real-world database searching. The main reason is that in most real-world scenario, the database is not unstructured. Thus, the data structure should be considered in real-world databases.

In the database area, there are also plenty of studies discussing how to use quantum computers to further improve traditional database queries. For example, [25, 54, 61, 62] discussed quantum query optimization, which is to use quantum algorithms like quantum annealing [26] to optimize a traditional database query. However, compared with the quantum query discussed in this paper, the existing studies are in a different direction. The existing studies are discussing how to use a quantum optimizer to do classical queries in a classical computer. In this paper, we discuss how to use a classical optimizer to do quantum queries in a quantum computer. We focus on a quantum database stored in a quantum computer, where the classical computer is only to assist the query. To our best knowledge, in the database area, we are the first to discuss quantum algorithms in this direction.

In conclusion, the existing classical data structures and the existing quantum database searching algorithms cannot solve the range query problem in quantum computers perfectly.

4 PROBLEM DEFINITION

In this section, we formally define the quantum range query problems and further discuss them.

Consider an immutable dataset $D=\{(key_0,rec_0),(key_1,rec_1),\cdots,(key_{N-1},rec_{N-1})\}$ where key_i is an integer stored in n_k bits, rec_i is a bit-string record stored in n_r bits, and (key_i,rec_i) is a key-record pair for each $i\in[0,N-1]$, we need to answer a range query in quantum bits. Note that in our discussion, the keys are integers, but they can be extended to real numbers easily. A range query is expressed as $QUERY(x,y)$ where x is the lower bound and y is the upper bound. A classical range query returns a list of key-record pairs whose keys are in the query range. The returned list $\{(key_{l_0},rec_{l_0}),\cdots,(key_{l_{k-1}},rec_{l_{k-1}})\}$ is a subset of D, where k is the length of the list, l_i is the index of the key-record pair in D for each $i\in[0,k-1]$, and $x\leq key_{l_i}\leq y$ holds for each $i\in[0,k-1]$. A quantum range query returns a superposition of all the key-record pairs in the list. We have the following Definition 4.1.

Definition 4.1 (Static Quantum Range Query). Given an immutable dataset $D=\{(key_0,rec_0),(key_1,rec_1),\cdots,(key_{N-1},rec_{N-1})\}$, a quantum range query $QUERY(x,y)$ is to return the quantum state

$$QUERY(x,y)=\frac{1}{\sqrt{k}}\sum_{i=0}^{k-1}\left|key_{l_i}\right\rangle\left|rec_{l_i}\right\rangle,$$

where for each $i\in[0,k-1]$, $x\leq key_{l_i}\leq y$.

To discuss the lower bound of the static quantum range query problem, we introduce another problem. The membership problem is another problem that given a query key, we need to decide if the key exists in the data structure. Ambainis [3] proved an $\Omega(\log N)$ lower bound on quantum binary search, and claimed that quantum algorithms can only achieve a constant speedup over classical binary search. Based on this lower bound, Pranab et al. [58] further proved that the lower bound of the static membership problem in the quantum cell-probe model is $\Omega(\log N)$. Then, we have the following Theorem 4.2.

Theorem 4.2. *The time complexity to answer a static quantum range query is $\Omega(\log N)$.*

Proof. Consider a static quantum range query $QUERY(x,x)$, it answers whether the key x exists in the dataset, so it answers a static membership problem. By [58], the time complexity to answer a static membership problem in quantum computers is $\Omega(\log N)$. Therefore, the time complexity to answer a static quantum range query is also $\Omega(\log N)$. □

Then, consider the dynamic case. Insertions and deletions are supported in the dynamic range query. An insertion $INSERT(key,rec)$ inserts the key-record pair (key,rec) into the dataset, so we obtain a new dataset $D \cup \{(key, rec)\}$. A deletion $DELETE(key, rec)$ deletes the key-record pair (key, rec) from the dataset, so we obtain a new dataset $D \setminus \{(key, rec)\}$. We have the following Definition 4.3.

Definition 4.3 (Dynamic Quantum Range Query). Given a dynamic dataset $D = \{(key_0, rec_0), (key_1, rec_1), \cdots, (key_{N-1}, rec_{N-1})\}$ which supports insertions and deletions, a quantum range query $QUERY(x, y)$ is to return the quantum state

$$QUERY(x, y) = \frac{1}{\sqrt{k}} \sum_{i=0}^{k-1} \left|key_{l_i}\right\rangle \left|rec_{l_i}\right\rangle,$$

where for each $i \in [0, k-1]$, $x \leq key_{l_i} \leq y$.

In the high-dimensional cases, the keys in the dataset are not integers but d-dimensional vectors. For each $i \in [0, N-1]$, key_i consists of d integers $(key_{i,0}, key_{i,1}, \cdots, key_{i,d-1})$. Correspondingly, x and y in a range query $QUERY(x, y)$ are two d-dimensional vectors, where $x = (x_0, x_1, \cdots, x_{d-1})$ and $y = (y_0, y_1, \cdots, y_{d-1})$. Then, we have the following Definition 4.4.

Definition 4.4 (High-Dimensional Quantum Range Query). Given an immutable dataset $D = \{(key_0, rec_0), (key_1, rec_1), \cdots, (key_{N-1}, rec_{N-1})\}$ where the keys are d-dimensional vectors, a quantum range query $QUERY(x, y)$ is to return the quantum state

$$QUERY(x, y) = \frac{1}{\sqrt{k}} \sum_{i=0}^{k-1} \left|key_{l_i}\right\rangle \left|rec_{l_i}\right\rangle,$$

where for each $i \in [0, k-1]$ and $j \in [0, d-1]$, $x_j \leq key_{l_i,j} \leq y_j$.

To further discuss the problems, we introduce the quantum random access memory (QRAM) in Section 4.1, since the QRAM should be used to store the dataset in quantum computers. Instead of returning a list of all the answers, a quantum range query returns quantum bits. The result is a superposition of all the key-record pairs with keys within the query range. Unlike the classical answer, if we measure the quantum bits, we can only randomly get one of the key-record pairs, and then the superposition will be collapsed. Although quantum range query cannot return a classical list of all the answers, we still consider it very meaningful, since we do not regard the result as a final output available to a user but as an input to other quantum algorithms. The motivations of the quantum range query are concluded in Section 4.2.

4.1 Quantum Random Access Memory

Following [38, 53], in this paper, we assume a classical-write quantum-read QRAM. It takes $O(1)$ time to store a classical record. Also, it takes $O(1)$ time to accept a superposition of addresses and return a superposition of the corresponding records. The main difference from a classical memory is that we assume that it takes $O(1)$ time to read multiple records. Therefore, we have the following Definition 4.5.

Definition 4.5 (Quantum Random Access Memory (QRAM)). A QRAM Q is an ideal model which perform store and load operations in $O(1)$ time.

- A store operation: $Q[address] = value$, where $value$ is a bit-string;
- A load operation:

$$Q \frac{1}{\sqrt{k}} \sum_{i=0}^{k-1} |address_i\rangle |x_i\rangle = \frac{1}{\sqrt{k}} \sum_{i=0}^{k-1} |address_i\rangle |x_i \oplus value_i\rangle,$$

where k is the number of required values, x_i is the initial value of the destination register, and $\oplus$ denotes the XOR operation.

Assume there is a QRAM Q, then Q needs to support two operations. The first operation is to store classical data. An operation $Q[address] = value$ will store a classical value at the specified address in $O(1)$ time, where $address$ and $value$ are two integers. In this operation, we can regard Q as an array A in a classical computer, where $A(i)$ is the value stored in the i-th entry and i is the address. For simplicity, we also use $Q[address]$ to denote the value stored at the address. The second operation is to load quantum data. In this operation, Q works as a quantum mapping from the addresses to the values: $Q\,|address\rangle\,|x\rangle = |address\rangle\,|x \oplus value\rangle$, where $\oplus$ denotes the XOR operation and $|x\rangle$ is the initial state of the returned qubits. Furthermore, the loading operation

$$Q \frac{1}{\sqrt{k}} \sum_{i=0}^{k-1} |address_i\rangle |0\rangle = \frac{1}{\sqrt{k}} \sum_{i=0}^{k-1} |address_i\rangle |A(address_i)\rangle$$

costs $O(1)$ time, where k is the number of required values. The load and store operation cost $O(1)$ time is a basic assumption in the database area when we calculate the complexity. In the quantum algorithm area, this quantum mapping can be regarded as a quantum oracle, and many studies such as [33, 36, 39, 41, 42, 46, 47] also assume the quantum oracle costs $O(1)$ time.

The concept of a QRAM has been widely adopted in many quantum algorithms [35, 37, 51, 63, 64]. Since many quantum algorithms were proposed to have a sub-linear time complexity compared to their linear classical competitors, if a linear time is needed to load the data, they may lose the advantage. For example, consider the Grover's algorithm [28] introduced in Section 3. If we want to use Grover's algorithm as a common database search algorithm, we need a quantum random access memory (QRAM) to store all the records. Assume the query is to find the position of the word "unicorn" in *The Witcher* and the word only appears once in the book. If all the N words are stored in the QRAM, then the QRAM can do the following quantum operation efficiently:

$$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle |0\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle |w_i\rangle,$$

where N is the total number of words in the book and w_i is the i-th word in the book. Then, we can build a quantum oracle that flips the amplitude of $|i\rangle$ if w_i is "unicorn". With this oracle, we can find the position of the word in $O(\sqrt{N})$ time. However, without a QRAM, we need to use a linear time to load all the words in the book, then Grover's algorithm will lose the advantage of quantum parallelism.

Note that even with the assumption of QRAMs, the quantum range query is still non-trivial. The reason is that the QRAM only maps the addresses to values, but to fetch all the addresses we need is a non-trivial task. Considering there is no existing quantum data structure, if we use quantum algorithms with a QRAM and without any data structure, we can use quantum counting [14] to estimate the number of answers k in $O(\sqrt{N})$ time, then use the unstructured quantum search [13] to find the answers in $O(\sqrt{Nk})$ time. The search can even be slower than a classical data structure search.

4.2 Application of Quantum Range Query

In this section, we introduce the motivations of the quantum range query. We discuss several real-world applications where other quantum algorithms can be benefitted from the quantum range query.

The first application is for label binning in quantum machine learning. Assume there is a classification task with a training dataset $\{(x_1, y_1), \cdots, (x_N, y_N)\}$ where for each $i \in [1, N]$, x_i is the feature vector and y_N is a continues label. There are many studies showing that packing labels into bins can improve the accuracy of the label prediction. In [23], Dougherty et.al. introduced equal-width binning, which is to bin the labels into k equal-sized bins, where k is a user parameter. They conducted experiments on 16 datasets to show that the label binning can improve the accuracy of Naive Bayes. In [66], Xue et.al. used equal-size binning to bin the labels into k bins. They conducted experiments to show that label binning can improve the accuracy of SVM and the best performance is obtained when the bin size is set to $O(N^{1/3})$. They pointed out that k will affect the quality of the result. In [10], Berg et.al. used a neural network to do image classification. They did three tasks: Age estimation, head pose estimation, and historical image dating. In the experiment, two tasks were improved by label binning. They also claimed that k has an impact on the prediction performance, and it is difficult to select k for a given problem without an extensive parameter search.

The above studies are based on classical machine learning. We can learn that many machine learning algorithms can be improved by label binning in some scenarios, so it is reasonable to consider it may also improve the accuracy of quantum machine learning algorithms. For example, to train a quantum model, there is an existing method called single-shot training [2]. To train a classical model, we input (x_i, y_i) one by one and N iterations are needed in an epoch. In single-shot training, we generate a superposition of all the x_i with the same label y_i and input the data in one iteration, so an epoch contains only k iterations, where k is the number of bins. To obtain such a superposition, we can make a quantum range query of the bin boundaries to generate the superposition, which should be exponentially faster than existing methods like [43] to encode the data in a bin into quantum bits one by one.

The second application is for a range filter of a recommendation system. For example, a recommendation system [21, 30] usually has two steps. The first step is to select a small part of all the items as the candidates. The second step is to use a prediction model (e.g., a neural network) to predict the scores of the items for a user. For example, assume that we want the system to recommend us a movie from 1990 to 1999. Now we have a model [11] to predict the score of the movie for a user. The first step is to select all the k movies from 1990 to 1999 from the database with N movies to obtain $\{movie_1, movie_2, \cdots, movie_k\}$, which costs $O(\log N + k)$ time . The second step is that for each $movie_i$, we use the model to predict the score $s(movie_i)$ for a recommendation. Assume the model prediction costs T time for a movie, then it totally costs $O(kT)$ time. Therefore, the classical recommendation system costs $O(\log N + k + kT)$ time. Now consider a quantum system for the movie recommendation. Assume that all the movies have been stored in a quantum data structure, and we can obtain the k movies in quantum bits by a quantum range query in $O(\log_B N)$ time:

$$QUERY(1990, 1999) = \frac{1}{\sqrt{k}} \sum_{i=0}^{k-1} |i\rangle\, |movie_i\rangle\, .$$

Then, with a quantum model, we can predict the score of all the movies in T time:

$$\frac{1}{\sqrt{k}} \sum_{i=0}^{k-1} |i\rangle\, |movie_i\rangle\, |0\rangle \rightarrow \frac{1}{\sqrt{k}} \sum_{i=0}^{k-1} |i\rangle\, |movie_i\rangle\, |s(movie_i)\rangle\, .$$

Therefore, the quantum recommendation system only needs $O($ $\log_B N{+}T)$ time. A multi-dimensional example is the location-based restaurant recommendation [30]. We can use a high-dimensional quantum range query to obtain a speedup.

Another example is the time-periodic-biased KNN. Campos et.al. [15] studied how to use KNN to do movie recommendations. They found that it can improve the accuracy to only consider the rating datasets from the last month before the recommendation time in different years. That means they need to select the records produced in the same month but in different years. Given a B+ tree containing all the movies sorted by the dates in the year, the calculation costs $O(\log_B N + N/12)$ time since the range query returns $N/12$ records on average. Assume that we can use a quantum range query to select the $N/12$ records in $O(\log_B N)$ time. Note that there is an existing $O(d^3)$ quantum KNN algorithm [52] where d is the dimension of the feature vector. Therefore, the whole process only costs $O(\log_B N + d^3)$. In most cases, d is much smaller than N. Assuming $d = O(\log N)$, we can find that the quantum time-periodic-biased KNN is exponentially faster than its classical competitor.

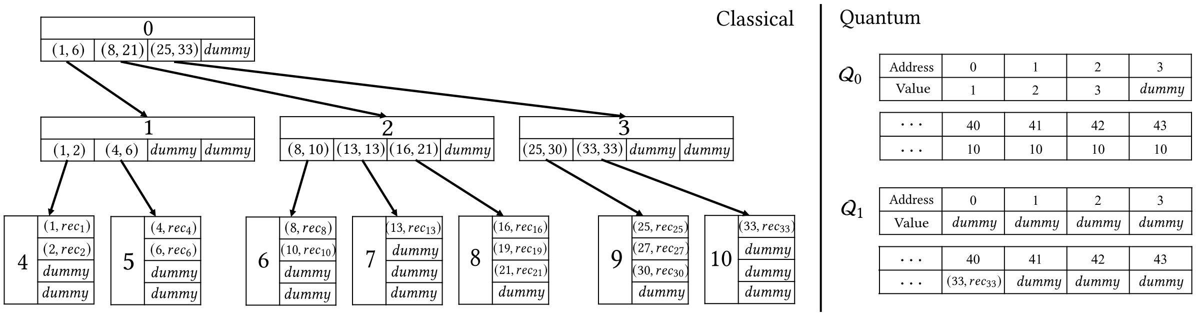
Besides the above applications, we believe that more and more quantum algorithms will be benefitted in the future. Since quantum algorithms always receive quantum data as the input, we think it is very common that quantum algorithms return answers in a superposition. For example, the HHL algorithm [32] returns the answer to a linear system of equations in a superposition, and it becomes a subroutine of the quantum SVM [51]. Specifically, in the database area, we consider the further quantum top-k algorithms and quantum query optimizer will be benifitted by quantum data structures, for which a quantum B+ tree can preprocess data to narrow the searching space.

5 QUANTUM B+ TREE

In this section, we propose the quantum B+ tree to solve the quantum range query problem. In Section 5.1, for static quantum range query, we introduce how to store a static quantum B+ tree in a QRAM, and give the algorithms for bulk-loadings and range queries. Then, we discuss two variants of the static quantum B+ tree. The first one in Section 5.2 is the dynamic quantum B+ tree for dynamic quantum range query, which supports two more kinds of operations: Insertions and deletions. The second one in Section 5.3 is the quantum range tree for high-dimensional quantum range query, which is based on the static quantum B+ tree and can do a high-dimensional range search.

5.1 Static Quantum B+ Tree

In our design, a static quantum B+ tree has two parts: A classical part and a quantum part.



Q_0

Address	0	1	2	3
Value	1	2	3	*dummy*

$\cdots$	40	41	42	43
$\cdots$	10	10	10	10

Q_1

Address	0	1	2	3
Value	*dummy*	*dummy*	*dummy*	*dummy*

$\cdots$	40	41	42	43
$\cdots$	$(33, rec_{33})$	*dummy*	*dummy*	*dummy*

Figure 1: An example of a quantum B+ tree: $N = 14$ **and** $B = 4$

The classical part is stored in either a classical computer or a quantum computer, since it can be regarded as a quantum system only using $|0\rangle$ and $|1\rangle$ without any superposition. It is based on a weight-balanced B+ tree [4]. A weight-balanced B+ tree is a tree data structure that consists of internal nodes and leaves. Each node has a unique ID number, which can be expressed by n_i bits. A user parameter B is a constant in the B+ tree, which is the branching factor usually set by the block size. We assume that B is a power of 2 and $B \geq 4$, which means there exists a constant integer $n_b \geq 2$ such that $B = 2^{n_b}$. A leaf node contains B key-record pairs sorted by the keys. An internal node has B children $c_0, c_1, \cdots, c_{B-1}$ and B routing keys $(l_0, r_0), (l_1, r_1), \cdots, (l_{B-1}, r_{B-1})$ where l_i is the least key in the subtree of c_i, and r_i is the largest key in the subtree of c_i. A node or a key-record pair can be empty, which is stored as a special mark *dummy*. In practice, *dummy* can be the maximum number of the data type so that it will always be sorted to the last positions. We have the following definitions.

- The level of a tree node is the distance from the root. All the leaves are in the same level.
- The height of a tree node is the distance from the leaves. The height of the tree is the height of the root node.
- The weight of a tree node is defined to be the number of non-dummy key-record pairs below the node. If a non-root node of height h has a weight between $\frac{1}{4}B^{h+1}$ and B^{h+1}, then the node is *balanced.* Moreover, if the non-root node of height h has a weight between $\frac{1}{2}B^{h+1}$ and B^{h+1}, then the node is *perfectly balanced.* If all the non-dummy node is perfectly balanced, then the B+ tree is perfectly balanced.

In a static quantum B+ tree, each non-root node has between $B/4$ and B non-dummy children, and the root node has at least two non-dummy children, so the height of the tree is $O(\log_B N)$.

The quantum part is stored in 2 QRAMs Q_0 and Q_1. The first QRAM Q_0 is to store the mapping from the internal nodes to their children. Consider an internal node u in the classical part has B children $c_0, \cdots, c_{B-1}$ which contains f non-dummy children $u_0, \cdots, u_{f-1}$ followed by $B - f$ *dummies*. In the quantum part, $c_0, c_1, \cdots, c_{B-1}$ are stored in $Q_0[u*B+0], Q_0[u*B+1], \cdots, Q_0[u*B+B-1]$ such that

$$Q_0 \frac{1}{\sqrt{B}} \sum_{i=0}^{B-1} |u\rangle |i\rangle |0\rangle = \frac{1}{\sqrt{B}} \sum_{i=0}^{B-1} |u*B+i\rangle |c_i\rangle$$

$$= \frac{1}{\sqrt{B}} \sum_{i=0}^{f-1} |u*B+i\rangle |u_i\rangle + \frac{1}{\sqrt{B}} \sum_{i=f}^{B-1} |u*B+i\rangle |dummy\rangle$$

$$= \frac{1}{\sqrt{B}} \sum_{i=0}^{f-1} |u\rangle |i\rangle |u_i\rangle + \frac{1}{\sqrt{B}} \sum_{i=f}^{B-1} |u\rangle |i\rangle |dummy\rangle .$$

Then, if we only consider the first n_i qubits and the last n_i qubits, we obtain a quantum transformation $|u\rangle |0\rangle \rightarrow \frac{1}{\sqrt{B}} \sum_{i=0}^{f-1} |u\rangle |u_i\rangle + \frac{\sqrt{B-f}}{\sqrt{B}} |u\rangle |dummy\rangle$. By the above quantum transformation, we can read out the IDs of all the children of the node u by only one memory access. Consider a leaf node u with no children. We define that for each $i \in [0, B-1]$, $Q_0[u*B+i] = u$. The second QRAM Q_1 is to store the mapping from the leaves to the key-record pairs. Consider a leaf node u in the classical part has B key-record pairs which contains f non-dummy key-record pairs $(key_0, rec_0), \cdots, (key_{f-1}, rec_{f-1})$ followed by $B - f$ *dummies*. Similarly, in the quantum part, $(key_0, rec_0), \cdots, (key_{f-1}, rec_{f-1})$ and $B - f$ *dummies* are stored in $Q_1[u*B+0], Q_1[u*B+1], \cdots, Q_1[u*B+B-1]$ such that

$$Q_1 \frac{1}{\sqrt{B}} \sum_{i=0}^{B-1} |u\rangle |i\rangle |0\rangle$$

$$= \frac{1}{\sqrt{B}} \sum_{i=0}^{f-1} |u\rangle |i\rangle |key_i\rangle |rec_i\rangle + \frac{1}{\sqrt{B}} \sum_{i=f}^{B-1} |u\rangle |i\rangle |dummy\rangle .$$

Specially, for a dummy node, we define that for each $i \in [0, B-1]$, we have $Q_0[dummy*B+i] = dummy$ and $Q_1[dummy*B+i] = dummy$. Figure 1 shows an example of a quantum B+ tree stored in a quantum computer. The left side of the vertical line is the classical part. There is a classical weight-balanced B+ tree with $N = 14$ and $B = 4$. Each box is a node in the tree where the big number is the node ID. An internal node has 4 children and it has a pair of bounds (l_i, r_i) for each non-dummy child. A leaf node has 4 key-record pairs. The right side of the vertical line is the quantum part. Q_0 is to store the edges. For example, node 1, node 2, node 3 and *dummy* are 4 children of node 0, so we have $Q_0[0*4+0] = 1$, $Q_0[0*4+1] = 2$, $Q_0[0*4+2] = 3$ and $Q_0[0*4+3] = dummy$. Since node 10 is a leaf node, we have $Q_0[10 * 4 + i] = 10$ for $i \in [0, 3]$, and $Q_1[10 * 4 + 0] = (33, rec_{33})$.

To analyze the cost of building a quantum B+ tree, we have the following Theorem 5.1.

Theorem 5.1. *Given a list of N key-record pairs sorted by the keys, a perfectly balanced quantum B+ tree can be built in $O(N)$ time.*

Proof. Let $h = \lfloor \log_B N \rfloor$. First, we evenly distribute the N key-record pairs into $\lceil N/B^h \rceil \cdot B^{h-1}$ leaves. Then, we build the tree from the bottom to the top, starting from the leaves. Since there are $O(N)$ leaves, there are $O(N)$ internal nodes, so the building costs $O(N)$ time.

Next, we prove the B+ tree is perfectly balanced by induction. We only discuss the case that N is not a power of B. Otherwise, the B+ tree is full, so it is obviously perfectly balanced. For a leaf node u, we can calculate the weight $w(u)$:

$$\begin{aligned} w(u) &\geq \left\lfloor \frac{N}{\lceil N/B^h \rceil \cdot B^{h-1}} \right\rfloor \geq \frac{N}{(N/B^h + 1) \cdot B^{h-1}} - 1 \\ &= \frac{N}{N/B + B^{h-1}} - 1 > \frac{N}{N/B + B^{\log_B N - 1}} - 1 \\ &= \frac{N}{N/B + N/B} - 1 = B/2 - 1. \end{aligned}$$

Then, we know that a leaf node has a weight between $B/2$ and B such that it is perfectly balanced. Consider a non-root node u of height $h > 0$, it must has B perfectly balanced non-dummy children such that $w(u) \geq B \cdot B^h/2 = B^{h+1}/2$. Therefore, the B+ tree is perfectly balanced. □

Then, we propose the hybrid quantum-classical range search algorithm on a static quantum B+ tree. We first introduce the quantum post-selection search in Section 5.1.1. The post-selection returns the answer to a range query efficiently when $\frac{k}{N}$ is large, where N is the number of records for search and k is the number of keys in the range. Obviously, we cannot assume $\frac{k}{N}$ is large, since the range query can return only one record, so we further proposed a hybrid quantum-classical range search algorithm to solve this problem. This algorithm contains two main steps: a global classical search and a local quantum search. In Section 5.1.2, we introduce the classical part. In Section 5.1.3, we introduce the quantum part based on a quantum post-selection. Then, we analyze the combination of the global classical search and the local quantum search in Section 5.1.4.

5.1.1 Quantum Post-Selection search.

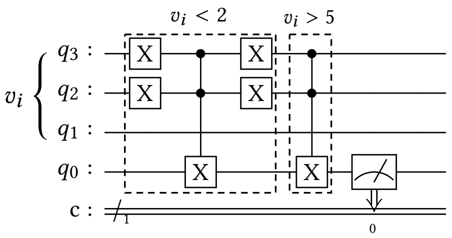


Figure 2: Post-selection search for $QUERY(2, 5)$

To understand the quantum range search on a quantum B+ tree, we first discuss how to do an unstructured range search for $QUERY(x, y)$. Consider a QRAM Q such that $Q \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle |0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle |v_i\rangle$, where v_i is an integer value and N is the number of values. Given Q and a pair of integers x and y, we are to obtain the quantum states in the interval $[x, y]$ i.e., $\frac{1}{\sqrt{k}} \sum_{x \leq v_i \leq y} |v_i\rangle$, where k is the number of values in the result. For simplicity, we use $|in\rangle = \frac{1}{\sqrt{k}} \sum_{x \leq v_i \leq y} |v_i\rangle$ to denote the values in the result and use $|out\rangle = \frac{1}{\sqrt{N-k}} \sum_{v_i < x \vee v_i > y} |v_i\rangle$ to denote the values not in the result. Then, we add an auxiliary qubit to obtain

$$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |v_i\rangle |0\rangle = \frac{\sqrt{k}}{\sqrt{N}} |in\rangle |0\rangle + \frac{\sqrt{N-k}}{\sqrt{N}} |out\rangle |0\rangle .$$

Next, we need to change the quantum state from $|0\rangle$ to $|1\rangle$ after $|out\rangle$ by a quantum oracle $O_{x,y}$:

$$|v_i\rangle |0\rangle \rightarrow \begin{cases} |v_i\rangle |1\rangle & v_i < x \text{ or } v_i > y; \\ |v_i\rangle |0\rangle & \text{otherwise.} \end{cases}$$

Then, we obtain $\frac{\sqrt{k}}{\sqrt{N}} |in\rangle |0\rangle + \frac{\sqrt{N-k}}{\sqrt{N}} |out\rangle |1\rangle$. The last step is to post-select the last qubit to be $|0\rangle$, which means we measure the last qubit, and if it is 1, we repeat the whole process until it is 0. After the post-selection, the state $|out\rangle$ is destroyed in the superposition so that we obtain $|in\rangle$. To analyze the complexity of the post-selection search, we obtain the following Lemma 5.2.

Lemma 5.2. *On average, the post-selection needs to be iterated N/k times to obtain the answer to the range query, where N is the total number of values and k is the number of values in the query range.*

Proof. After we measure the last auxiliary qubit, we will obtain 0 with probability $\frac{k}{N}$ or 1 with probability $\frac{N-k}{N}$ so that the expected number of measurements to obtain 0 is N/k. □

Figure 2 shows an example. Use $q_3q_2q_1$ to denote the binary representation of v_i and use q_0 to denote the auxiliary qubit. Assume we have $|q_3q_2q_1\rangle = \frac{1}{2}(|0\rangle + |1\rangle + |4\rangle + |7\rangle)$ and we are to answer $QUERY(2, 5)$ with post-selection. We add q_0 to the last, then apply the quantum circuit in Figure 2. In the first dashed-line box, we consider the case $v_i < 2$ such that we obtain $|q_3q_2q_1\rangle |q_0\rangle = \frac{1}{2}(|0\rangle + |1\rangle) |1\rangle + \frac{1}{2}(|4\rangle + |7\rangle) |0\rangle$. In the second dashed-line box, we consider the case $v_i > 5$ such that we obtain $|q_3q_2q_1\rangle |q_0\rangle = \frac{1}{2}(|0\rangle + |1\rangle + |7\rangle) |1\rangle + \frac{1}{2} |4\rangle |0\rangle$. Then, if we measure q_0 to be 0 with probability $\frac{1}{4}$, then we can obtain $|q_3q_2q_1\rangle = |4\rangle$.

5.1.2 Global Classical Search.

As described above, post-selection search costs $O(N/k)$ time, so it is obviously unacceptable to only use post-selection search to answer a range query. The reason is that if there is only one key-record pair in the query range, then $k = 1$, which means the time complexity becomes $O(N)$. On the contrary, if k is large, the post-selection search can be very fast. That is the reason why we use a data structure to do the range search even in a quantum computer.

Given a query $QUERY(x, y)$, in our quantum range search algorithm, the first step is to do a global classical search. The motivation of this step is to check if k is large enough. We do the search level-wise starting from the root node. We call the nodes to be searched in a level the candidate nodes in the level. For example, the root node is the only candidate node in level 0. Consider a candidate node u in level j has f non-dummy children $c_0, c_1, \cdots, c_{f-1}$ and f

routing keys $(l_0, r_0), (l_1, r_1), \cdots, (l_{f-1}, r_{f-1})$. For each $i \in [0, f-1]$, there are the following three cases.

- Case 1: $r_i < x$ or $l_i > y$. That means c_i does not have an answer. We do nothing.
- Case 2: $l_i < x$ and $x \le r_i \le y$ or $r_i > y$ and $x \le l_i \le y$. A part of the non-dummy key-record pairs below c_i is in the answer, so add c_i to the candidate nodes in level $j+1$.
- Case 3: $l_i \ge x$ and $r_i \le y$. That means all the non-dummy key-record pairs below c_i are in the answer. Turn to a local quantum search starting from the candidate nodes in level j.

To analyze the complexity of the global classical search, we have the following Lemma 5.3.

Lemma 5.3. *The global classical search costs $O(\log_B N)$ time.*

Proof. Assume in level j, the candidate nodes are $u_0, u_1, \cdots, u_{t-1}$, where $t \ge 3$ is the number of candidate nodes in the level. Obviously, all the key-record pairs below u_1 are in the answer, so we turn to the local quantum search in level $j-1$ and cannot reach level j, which makes a contradiction. Therefore, in each level, there are at most 2 candidate nodes. Since the height of the tree is $O(\log_B N)$, there are totally $O(\log_B N)$ candidate nodes in the B+ tree. Hence, the global classical search costs $O(\log_B N)$ time. □

For example, consider a query $QUERY(5, 11)$ given to the quantum B+ tree in Figure 5.1. We first check node 0. There is no child in Case 3. Node 1 and node 2 are candidate nodes since they are in Case 2. Then, we check node 1 and node 2. We find node 6 is a child in Case 3, so we turn to a local quantum search starting from node 1 and node 2.

5.1.3 *Local Quantum Search.*

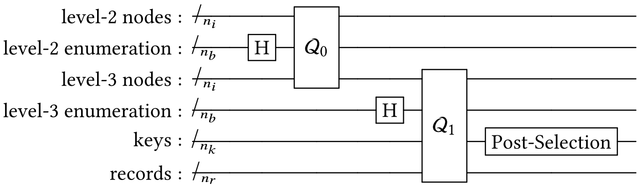


Figure 3: Local quantum search for $QUERY(5, 11)$

Since the local quantum search starts from at most two candidate nodes, consider answering $QUERY(x, y)$ starting from a node u and another node v as the candidate nodes in level j and the height of the tree is h. Step 1 is to initialize the first n_i quantum qubits to be $\frac{1}{\sqrt{2}}(|u\rangle + |v\rangle)$. Then, in Step 2, we add n_b auxiliary qubits $|0\rangle$ to the last and also apply a Hadamard gate on each auxiliary qubit. We obtain

$$\frac{1}{\sqrt{2}}(|u\rangle + |v\rangle)\mathcal{H}|0\rangle\mathcal{H}|0\rangle \cdots \mathcal{H}|0\rangle = \frac{1}{\sqrt{2B}}(|u\rangle + |v\rangle)\sum_{i=0}^{B-1}|i\rangle .$$

This step is to enumerate all the edges from the candidate nodes. Then, in Step 3, we add n_i qubits to the end and apply Q_0 to obtain all the children of the candidate nodes, so we obtain

$$\frac{1}{\sqrt{2B}}|u\rangle\sum_{i=0}^{B-1}|i\rangle|c_i\rangle + \frac{1}{\sqrt{2B}}|v\rangle\sum_{i=0}^{B-1}|i\rangle|c_{i+B}\rangle ,$$

where $c_0, \cdots, c_{B-1}$ are the B children of u and $c_B, \cdots, c_{2B-1}$ are the B children of v. If we only look at the last n qubits, we obtain $\frac{1}{\sqrt{2B}}\sum_{i=0}^{2B-1}|c_i\rangle$, which is the $2B$ children of u and v. We repeat Step 2 and Step 3 for $h-j$ times so that we obtain all the $2B^{h-j}$ leaves below u and v. Then, we do the same thing as Step 2 to enumerate all the $2B^{h-j+1}$ key-record pairs in the $2B^{h-j}$ leaves. In the last step, we apply Q_1 to obtain all the key-record pairs below u and v and then do a post-selection search. Denote the $2B^{h-j+1}$ key-record pairs as $(key_0, rec_0), \cdots, (key_{2B^{h-j+1}-1}, rec_{2B^{h-j+1}-1})$. Then the quantum states become $\frac{1}{\sqrt{2B^{h-j+1}}}\sum_{i=0}^{2B^{h-j+1}-1}|key_i\rangle|rec_i\rangle$. Similar to Section 5.1.1, we use $|in\rangle$ to denote the k key-record pairs in the query range and use $|out\rangle$ to denote the other dummy key-record pairs and non-dummy key-record pairs which are not in the query range. We obtain $\frac{\sqrt{k}}{\sqrt{2B^{h-j+1}}}|in\rangle + \frac{\sqrt{2B^{h-j+1}-k}}{\sqrt{2B^{h-j+1}}}|out\rangle$. If we do a post-selection, we can obtain $|in\rangle$ with probability $\frac{k}{2B^{h-j+1}}$. To analyze the complexity of a local quantum search, we have the following Lemma 5.4.

Lemma 5.4. *On average, the local quantum search needs $O(\log_B N)$ time.*

Proof. By Lemma 5.2, we repeat all the steps for $\frac{2B^{h-j+1}}{k}$ times on average. By the condition to trigger a local quantum search mentioned in Section 5.1.2, all the non-dummy key-record pairs below one of the children of u and v are all in the answer, therefore $k \ge \frac{1}{4}B^{h-j}$ by the definition of our quantum B+ tree. So, we need to repeat all the steps for at most $8B$ times, which is a constant time. In each iteration, we do Step 2 and Step 3 for at most $O(\log_B N)$ times, so the local quantum search needs $O(\log_B N)$ time. □

For example, consider a query $QUERY(5, 11)$ on the quantum B+ tree in Figure 5.1. As mentioned in Section 5.1.2, the candidate nodes are node 1 and node 2. First, we initialize $|\psi\rangle = \frac{1}{\sqrt{2}}(|1\rangle + |2\rangle)$. Then, we apply the quantum circuit in Figure 3. After applying Hadamard gates and Q_0, we obtain $|\psi\rangle = \frac{1}{\sqrt{8}}(|4\rangle + |5\rangle + |6\rangle + |7\rangle + |8\rangle) + \frac{\sqrt{3}}{\sqrt{8}}|dummy\rangle$, which consists of all the children of node 1 and node 2. Then, after applying Hadamard gates and Q_1, we obtain all the key-record pairs below node 1 and node 2, which is $|\psi\rangle = \frac{1}{\sqrt{32}}(|1\rangle|rec_1\rangle + |2\rangle|rec_2\rangle + |4\rangle|rec_4\rangle + |6\rangle|rec_6\rangle + |8\rangle|rec_8\rangle + |10\rangle|rec_{10}\rangle + |13\rangle|rec_{13}\rangle + |16\rangle|rec_{16}\rangle + |19\rangle|rec_{19}\rangle + |21\rangle|rec_{21}\rangle) + \frac{\sqrt{22}}{\sqrt{32}}|dummy\rangle$. Finally, by a post-selection, we can obtain $|\psi\rangle = \frac{1}{\sqrt{3}}(|6\rangle|rec_6\rangle + |8\rangle|rec_8\rangle + |10\rangle|rec_{10}\rangle)$ with probability $\frac{3}{32}$.

5.1.4 *Quantum Range Query.*

In summary, we obtain Algorithm 1. The quantum range query algorithm on a static quantum B+ tree works as follows. First, we do a global classical search in the classical part. We can obtain at most two candidate nodes. Then, we do a local quantum search in the quantum part starting from the two candidate nodes. In the local quantum search, we use the QRAM to obtain all the key-record pairs below the candidate nodes, then do a post-selection search to obtain the answer. To analyze the complexity of the algorithm, we have the following Theorem 5.5.

Theorem 5.5. *On average, the quantum range query algorithm returns the answer in $O(\log_B N)$ time.*

Proof. By Lemma 5.3, the global classical search needs $O(\log_B N)$ time. By Lemma 5.4, the local quantum search needs $O(\log_B N)$

Algorithm 1: QUERY(x, y)

```
Input: a lower bound $x$ and an upper bound $y$.
Output: $\frac{1}{\sqrt{k}} \sum_{x \le key_i \le y} |key_i\rangle |rec_i\rangle$ where $k$ is the number
        of records with keys in $[x, y]$.
// A global classical search
Create lists $L$ and $L'$ to store candidate nodes;
$L.add(root)$;                    // Initialize $L$ with the root node
$Found \leftarrow False$;         // to denote if we need a quantum search
while $L$ consists of internal nodes do
    $L' \leftarrow \emptyset$;
    foreach node $u$ in $L$ do
        foreach child $c_i$ of $u$ do
            if $l_i \ge x$ and $r_i \le y$ then $Found \leftarrow True$; break;
            if $l_i \le y$ and $r_i \ge x$ then $L'.add(c_i)$;
        if Found then break;
    if Found then break;
    $L \leftarrow L'$;
if not Found then
    $L' \leftarrow \emptyset$;
    foreach node $u$ in $L$ do
        foreach key-record pair $(key_i, rec_i)$ stored in $u$ do
            if $x \le key_i \le y$ then $L'.add((key_i, rec_i))$;
    return $\frac{1}{\sqrt{|L'|}} \sum |L'_i\rangle$
else
    // A local quantum search
    $|\psi\rangle \leftarrow \frac{1}{\sqrt{|L|}} \sum |L_i\rangle$;
    while $|\psi\rangle$ consists of internal nodes do
        $|\psi\rangle = \frac{1}{\sqrt{t}} \sum_{i=0}^{t-1} |u_i\rangle \xmapsto{H-gates} \frac{1}{\sqrt{tB}} \sum_{i=0}^{t-1} |u_i\rangle \sum_{j=0}^{B-1} |j\rangle$
        $\xmapsto{Q_0} \frac{1}{\sqrt{tB}} \sum_{i=0}^{tB-1} |c_i\rangle$;     // search for the children
    $|\psi\rangle = \frac{1}{\sqrt{t}} \sum_{i=0}^{t-1} |u_i\rangle \xmapsto{H-gates} \frac{1}{\sqrt{tB}} \sum_{i=0}^{t-1} |u_i\rangle \sum_{j=0}^{B-1} |j\rangle \xmapsto{Q_1}$
    $\frac{1}{\sqrt{tB}} \sum_{i=0}^{tB-1} |key_i\rangle |rec_i\rangle \xmapsto{\text{post-selection}}$
    $\frac{1}{\sqrt{k}} \sum_{x \le key_i \le y} |key_i\rangle |rec_i\rangle$;
    return $\frac{1}{\sqrt{k}} \sum_{x \le key_i \le y} |key_i\rangle |rec_i\rangle$
```

time. Therefore, the quantum range query algorithm needs $O(\log_B N)$ time. □

By Theorem 5.5 and Theorem 4.2, this algorithm is asymptotically optimal in a quantum computer.

5.2 Dynamic Quantum B+ Tree

In this section, we introduce how to make the static quantum B+ tree dynamic. In Section 5.2.1, we introduce how to use the LSM-technique to make the quantum B+ tree support insertions. In Section 5.2.2, we give a solution to perform deletions. In Section 5.2.3, we analyze the complexity of a range query in the dynamic version.

To use the quantum range query algorithm introduced in Section 5.1.4, the quantum B+ tree needs to meet 3 requirements:

(1) Fanout limit: The fanout of each node should be at most B.
(2) Weight limit: The weight of each node of height h should be at least $\frac{1}{4}B^{h+1}$.
(3) Level limit: All the leaves in a tree should be in the same level.

The reason is that Lemma 5.4 is based on these 3 requirements. To the best of our knowledge, there is no existing data structure that meets our all requirements. For example, the original B+ tree [20] has no weight limit, and the original weight-balanced B+ tree [4] has no fanout limit, since the fanout can be larger than B. The reason is that to meet the three requirements is meaningless in a classical computer. That is the motivation of us to make a new data structure that can work with our quantum range query algorithm.

5.2.1 *Insertion.*

We use the logarithmic method [9] to make the quantum B+ tree support insertions. To store the key-record pairs, we build at most $\lfloor \log_B N \rfloor + 1$ forests $F_0, \cdots, F_{\lfloor \log_B N \rfloor}$ where for each $i \in [0, \lfloor \log_B N \rfloor]$, the forest F_i contains at most $B-1$ static quantum B+ trees of height i.

To perform an insertion like $INSERT(key, rec)$, we insert the key-record pair into a sorted list first. When the length of the sorted list reaches B, we flush it into F_0. Then, whenever a forest F_i has B quantum B+ trees, which means we have B quantum B+ trees of height i, we merge the B quantum B+ trees of height i into a quantum B+ trees of height $i + 1$, and add it into F_{i+1}. To analyze the complexity, we have the following Theorem 5.6.

Theorem 5.6. *The amortized cost of an insertion in a dynamic quantum B+ tree is $O(\log_B N)$.*

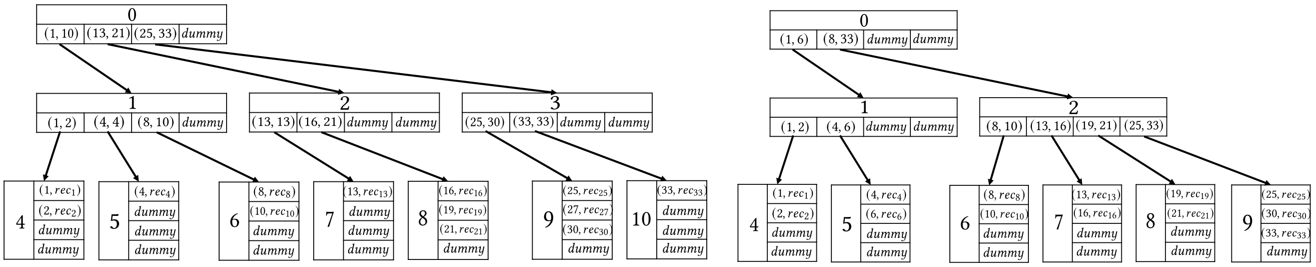
Proof. By Theorem 5.1 in this paper and Theorem 3.1 in [9], N insertions totally cost $O(N \log_B N)$ time. Therefore, the amortized cost of one insertion is $O(\log_B N)$. □

Note that all quantum B+ trees have their classical parts and quantum parts. When doing insertions, any modification in the classical part has to be done in the quantum part. Since a QRAM operation costs $O(1)$ time as defined in Section 4.1, Theorem 5.6 also holds in the quantum part.

5.2.2 *Deletion.*

To perform a deletion like $DELETE(key, rec)$, we have two steps. The first step is to find the quantum B+ tree which contains the key-record pair. The second step is to delete the key-record pair in both the classical part and the quantum part of the quantum B+ tree. We then discuss these two steps one by one.

To locate the tree to do the deletion, we assign each key-record pair a unique ID in ascending order (e.g., by a counter). When doing an insertion $INSERT(key, rec)$, we assign the ID to the key-record pair, and maintain a B+ tree T_0 to store the one-to-one mapping from the key-record pair to its ID. When a key-record pair is inserted, we insert the key-record pair and its ID into T_0. When the key-record pair is deleted, we delete the key-record pair in T_0 accordingly. There is no other case to update T_0. Furthermore, we maintain another B+ tree T_1 to store the mapping from the key-record pair ID to the forest F_i it belongs to. Similar to T_0, we do insertions and deletions in T_1 accordingly. In addition, when we merge F_i, we update T_1 for all the key-record pairs in F_i. To analyze the update cost in a merge, we have the following Lemma 5.7.


0
(1, 10) (13, 21) (25, 33) dummy
1
(1, 2) (4, 4) (8, 10) dummy
2
(13, 13) (16, 21) dummy dummy
3
(25, 30) (33, 33) dummy dummy
4 (1, rec1) (2, rec2) dummy dummy
5 (4, rec4) dummy dummy dummy
6 (8, rec8) (10, rec10) dummy dummy
7 (13, rec13) dummy dummy dummy
8 (16, rec16) (19, rec19) (21, rec21) dummy
9 (25, rec25) (27, rec27) (30, rec30) dummy
10 (33, rec33) dummy dummy dummy
0
(1, 6) (8, 33) dummy dummy
1
(1, 2) (4, 6) dummy dummy
2
(8, 10) (13, 16) (19, 21) (25, 33)
4 (1, rec1) (2, rec2) dummy dummy
5 (4, rec4) (6, rec6) dummy dummy
6 (8, rec8) (10, rec10) dummy dummy
7 (13, rec13) (16, rec16) dummy dummy
8 (19, rec19) (21, rec21) dummy dummy
9 (25, rec25) (30, rec30) (33, rec33) dummy

(a) $DELETE(6, rec_6)$: Delete $(6, rec_6)$ in node 5. The weight of node 1 becomes 3 which is less than $\frac{1}{4}B^2 = 4$, so node 1 becomes imbalanced. It borrows node 6 from node 2 such that node 1 and node 2 are both balanced.

(b) $DELETE(27, rec_{27})$: Delete $(27, rec_{27})$ in node 9. Node 3 becomes imbalanced, and it cannot borrow node 8 from node 2. Node 2 and node 3 cannot be directly merged since they have 5 children, so the whole subtree is rebuilt.

Figure 4: Examples of deletions

Lemma 5.7. *When merging F_i, the time complexity to update T_1 is $O(\log_B N)$.*

Proof. Let ID_l denote the least ID in F_i. For each $j < i$ and each key-record pair in F_j, the ID of the key-record pair is smaller than ID_l. Let ID_r denote the greatest ID in F_i. Then, for each $j > i$ and each key-record pair in F_j, the ID of the key-record pair is greater than ID_r. Therefore, the update operations for the key-record pairs in F_i can be merged into a range update. Then, we can use lazy propagation [34] to do the range update in $O(\log_B N)$ time. □

By Lemma 5.7, the extra cost to maintain T_0 and T_1 has no impact on Theorem 5.6, which means the amortized cost of insertion is still $O(\log_B N)$.

To delete the key-record pair in a B+ tree in F_i, we do a classical search to find the leaf that contains the key-record pair. Replace the key-record pair with a *dummy*. Then, check if its ancestors are still balanced. If an imbalanced ancestor is found, we first check if it can borrow a child from its sibling. Figure 6(a) shows an example in this case. If not, we check if it can be directly merged with its sibling, which means the node and its sibling have at most B children. If not, we merge the node and its sibling by rebuilding the subtrees below them. Figure 6(b) shows an example in this case. Then, after rebalancing the B+ tree, we check if the root node still has at least two children. If not, we check if it can borrow a child from another B+ tree in F_i. If not, then if there are at least two B+ trees in F_i, we merge the root nodes of the two B+ trees. Otherwise, we remove the root node and downgrade the B+ tree from F_i to F_{i-1}.

Theorem 5.8. *The amortized cost of a deletion in a dynamic quantum B+ tree is $O(\log_B N)$.*

Proof. The first step costs $O(\log_B N)$ time, since it consists of two point queries in B+ trees.

Then consider the second step. Motivated by the analysis of partial rebuilding in [49], we consider a node u of height h just after a rebuild. The node u is perfectly balanced such that its weight $w(u) \geq \frac{1}{2}B^{h+1}$. Since it will become imbalanced if and only if $w(u) < \frac{1}{4}B^{h+1}$, there must be $\Omega(B^{h+1})$ deletions below the node u or its siblings before that. So, it is charged $O(1)$ time for each deletion below it and its siblings. Then, for a deletion in a leaf node, each ancestor and its siblings are charged $O(1)$ time, so they are totally charged $O(\log_B N)$ time.

Therefore, the amortized cost of a deletion is $O(\log_B N)$. □

Theorem 5.8 shows that the complexity of deletion is the same as insertion, which is $O(\log_B N)$.

5.2.3 Query.

To answer a query $QUERY(x, y)$, we do a global classical search and a local quantum search on $F_0, \cdots, F_{\lfloor \log_B N \rfloor}$. We initialize $\lfloor \log_B N \rfloor +$ $1x$ lists $L_0, \cdots, L_{\lfloor \log_B N \rfloor}$. For each $i \in [0, \lfloor \log_B N \rfloor]$, we add the root nodes in F_i into L_i. Then, we do the global classical search from $L_{\lfloor \log_B N \rfloor}$ to L_0. For L_i, we scan all the nodes in it one by one. Consider a node u in L_i. We scan all u's children. Similar to the algorithm in Section 5.1.2, if there is a child such that all the non-dummy key-record pairs are in the answer, then we turn to the local quantum search starting from $L_0, \cdots, L_i$. Otherwise, we add at most two children of u into L_{i-1}, which may contain the answers. By Lemma 5.3, since there are initially $O(\log_B N)$ quantum B+ trees, the classical global search costs $O(\log_B^2 N)$ time.

Then, consider the local quantum search starting from $L_0, \cdots, L_i$. Consider the nodes $u_0, u_1, \cdots, u_{m-1}$ in the lists, where m is the total number of nodes in $L_0, \cdots, L_i$. We initialize the quantum bits to be $\sum_{i=0}^{m-1} \frac{\sqrt{B^{h(u_i)+1}}}{\sqrt{\sum_{j=0}^{m-1} B^{h(u_j)+1}}} |u_i\rangle$, where $h(u_i)$ is the height of u_i, and $\frac{\sqrt{B^{h(u_i)+1}}}{\sqrt{\sum_{j=0}^{m-1} B^{h(u_j)+1}}}$ is the normalized amplitude of each u_i such that each of the key-record pair below the nodes in the lists has the same amplitude in the result. By the amplitude encoding scheme in [43], the initialization costs $O(m)$ time. Since by Lemma 5.3, $m = O(\log_B N)$, the quantum bits can be obtained in $O(\log_B N)$ time. Then, we do the same Step 2 and Step 3 in Section 5.1.3 to obtain the leaves below the nodes. Finally, we do a post-selection search as discussed in Section 5.1.1. To analyze the query cost, we have the following Theorem 5.9.

Theorem 5.9. *On average, the dynamic quantum B+ tree answers a range query in $O(\log_B^2 N)$ time.*

Proof. As above mentioned, the global classical search costs $O(\log_B^2 N)$ time. Consider the cost of the local quantum search. The initialization and the steps to obtain the leaves cost $O(\log_B N)$. Then, we multiply it by the average post-selection times. Let N' denote the total number of key-record pairs below the nodes in $L_0, \cdots, L_i$. Then, by Lemma 5.2, the post-selection needs to be iterated $O(N'/k)$ times on average. Since there are at most $2B\lfloor \log_B N \rfloor$ nodes in the lists and each node has a weight at most B^{i+1}, we have

$N' \leq 2B^{i+2} \log_B N$. Before we turn to the local quantum search, we have found a child of a node in L_i such that all the key-record pairs below the child are in the answer, such that $k \geq \frac{1}{4} B^i$. Therefore, $N'/k = O(\log_B N)$. Hence, the local quantum search costs $O(\log_B^2 N)$ time on average.

Since both the global classical search and the local quantum search cost $O(\log_B^2 N)$ time, the dynamic quantum B+ tree answers a range query in $O(\log_B^2 N)$ time on average. □

5.3 Quantum Range Tree

In this section, we show how to convert the classical range tree [8] into a quantum data structure, as another extension of the static quantum B+ tree. Originally, the range tree answers a d-dimensional range query in $O(\log^d N + k)$ time. Using the quantum B+ tree, we can improve the complexity to $O(\log^d N)$. In the new problem, the key in a key-record pair is a d-dimensional vector $(key_0, \cdots, key_{d-1})$ where each key_i is an integer, and in a query $QUERY(x, y)$, x and y are also two d-dimensional vectors, which are $(x_0, \cdots, x_{d-1})$ and $(y_0, \cdots, y_{d-1})$, respectively. The quantum range tree returns a superposition of all the key-record pairs such that for each $i \in [0, d-1]$, we have $x_i \leq key_i \leq y_i$.

We construct a quantum range tree recursively. To build a d-dimensional quantum range tree, we first build a classical B+ tree indexing the d-th dimension of the keys. Then, we build a $d-1$-dimensional tree for each internal node in the B+ tree based on the key-record pairs below the internal node. Specifically, a 1-dimensional quantum range tree is a static quantum B+ tree. Obviously, the space complexity and the time complexity of the construction are similar to the classical range tree.

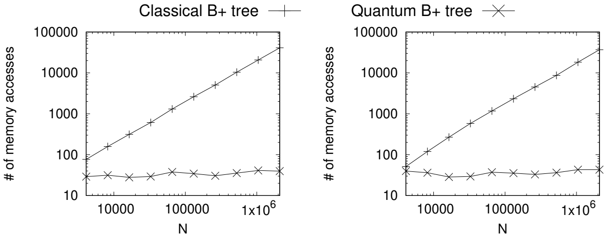
Then, we discuss the d-dimensional quantum range query. Starting from the d-th dimension, we search the B+ tree and find the $O(\log_B N)$ internal nodes in the search path which covers the d-th dimension of the query range. Then, turn to the quantum range trees on the $O(\log_B N)$ internal nodes and do a $d-1$-dimensional range search. Similar to the classical range tree, we then fetch the answers in $O(\log_B^{d-1} N)$ 1-dimensional quantum range trees. By the same algorithm as described in Section 5.1.4, we can obtain the key-record pairs within the query range in $O(O(\log_B^d N))$ time on average. The following Theorem 5.10 shows the result.

Theorem 5.10. *On average, the quantum range tree answers a range query in $O(\log_B^d N)$ time.*

Proof. The first step is to recursively search the quantum range trees to obtain the $O(\log_B^{d-1} N)$ 1-dimensional quantum range trees which contain answers. It costs $O(\log_B^{d-1} N)$ time. The second step is to perform a global classical search on the $O(\log_B^{d-1} N)$ quantum B+ tree. It costs $O(\log_B^d N)$ time. The third step is to perform a local quantum search starting from $O(\log_B^{d-1} N)$ candidate nodes returned in the second step. We first initialize the quantum bits in $O(\log_B^{d-1} N)$ time. Then, we do the same Step 2 and Step 3 in Section 5.1.3 to obtain the leaves below the candidate nodes in $O(\log_B N)$ time. By Lemma 5.2, we need to do a constant number of post-selections. Therefore, this step costs $O(\log_B^{d-1} N)$ time.

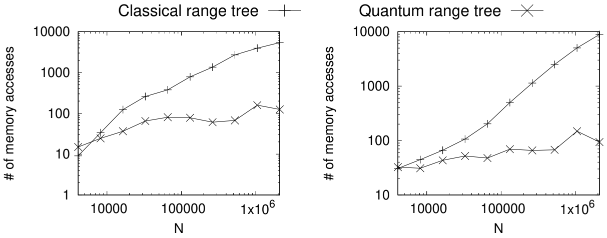
Therefore, the quantum range tree answers a range query in $O(\log_B^d N)$ time on average. □

6 EXPERIMENT



(a) Brightkite (b) Gowalla

Figure 5: Range queries with $d = 1$ and $B = 16$



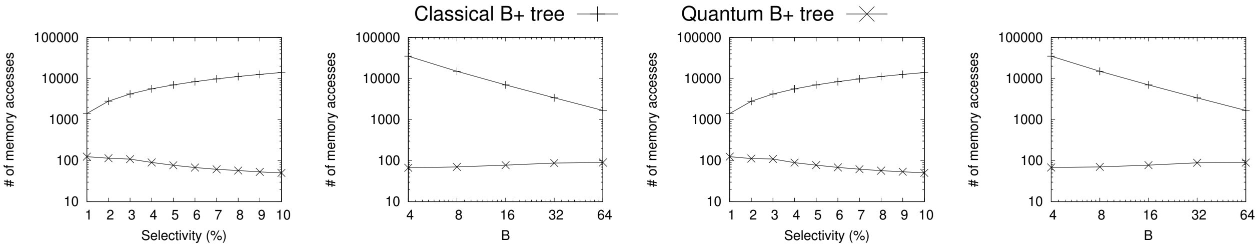
(a) Brightkite (b) Gowalla

Figure 6: Range queries with $d = 2$ and $B = 16$

In this section, we show some experimental results to further discuss the advantages of the quantum B+ tree. The study of the real-world quantum supremacy [5, 12, 60], which is to confirm that a quantum computer can do tasks faster than classical computers, is still a popular topic in the quantum area. We are not going to verify quantum supremacy in this problem, but we believe it will be verified in a future quantum computer. In this paper, we choose to evaluate the *number of memory accesses* to make the comparisons, which corresponds to IOs in traditional searches. In the quantum data structures, a QRAM read operation is counted as 1 IO. In the classical data structures, a page access is counted as 1 IO. IOs cannot be regarded as the real execution time, since we have no any information about the real implementation of a QRAM, but they can reflect the potential of the data structures. In addition, the existing quantum simulators such as Qiskit [65] and Cirq [27] do not have a solution to simulate an efficient QRAM, so we choose to use C++ to perform the quantum simulations.

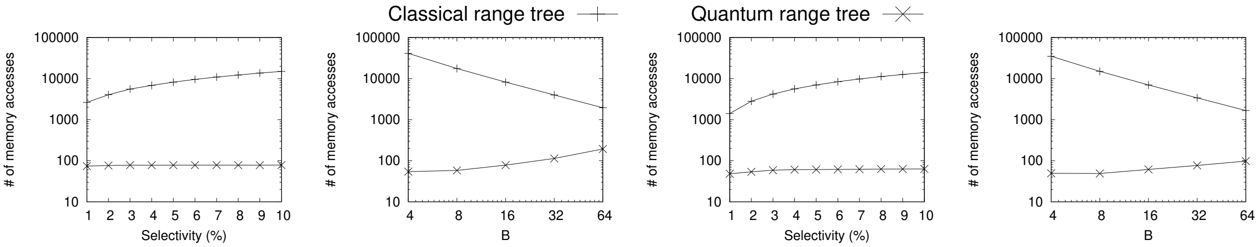
We conducted an experiment by simulating the quantum B+ tree on real-world datasets from SNAP [40] with C++. The two datasets named Brightkite (4m) and Gowalla (6m) are two lists of check-ins with timestamps and locations. To compare the quantum B+ tree and the classical B+ tree, we took the timestamps as the 1-dimensional data, which corresponds to the time-based range queries in real-world applications. We set the data size N from 4096 to 1048576 and randomly generated 10000 range queries. Figure 5 shows the result, where the x-axis N denotes the number of records and the y-axis denotes the number of memory accessed needed to answer a range query on average.

In addition, to show the advantage of the quantum range tree, we compared it with the classical range tree [17]. Note that Chazalle [17] has proved the lower bound of the orthogonal range searching problem, and the range tree is asymptotically optimal. We choose



(a) Brightkite (b) Brightkite (c) Gowalla (d) Gowalla

Figure 7: 1-dimensional range queries



(a) Brightkite (b) Brightkite (c) Gowalla (d) Gowalla

Figure 8: 2-dimensional range queries

the state-of-the-art in the classical computer to make the comparison. We took the locations in the datasets as the 2-dimensional data, which corresponds to the location-based range queries in real-world applications. Figure 6 shows the result.

In Figure 5 and Figure 6, we can learn that the cost of a classical range query grows linearly with N. The reason is that when the size of the dataset becomes larger, the size of the answer to a range query will also probably become larger. Compared to the classical competitors, the cost of a quantum range query only grows logarithmically with N, since it does not depend on the size of the answer. Based on these small datasets, the quantum range query is up to 1000× faster than the classical query, which supports our claim in Section 1 that the quantum range query can be exponentially faster than the classical range query.

Then, we study how the size of the answer and the block size affect the performance. Figure 7 shows the result in 1-dimension. First, we studied the performance with different selectivities (i.e., $\frac{k}{N}$). We varied the selectivity between 1% and 10% and the results are shown in Figure 7(a) and Figure 7(c), where the x-axis denotes the selectivity and the y-axis denotes the number of memory accessed needed to answer a range query on average. We see that the IOs needed by the classical B+ tree grows linearly with k. However, the IOs needed by the quantum B+ tree even decreases. It seems surprising but in fact it is reasonable. The reason is that a larger k shortened the process of the classical global search, such that we can turn to the local quantum search earlier. Also, as mentioned in Section 5.1.1, post-selection costs $O(N/k)$ time, so a larger k also speedup the post-selection step. In addition, we studied how the block size affects the performance, and the results are shown in Figure 7(b) and Figure 7(d), where the x-axis denotes the block size and the y-axis denotes the number of memory accessed needed to answer a range query on average. IOs needed by the classical B+ tree decreases fast, since with a larger B, fewer pages were accessed by the B+ tree. IOs needed by the quantum B+ tree increases, and this observation corresponds to a middle result in the proof of Lemma 5.4 that B has an impact on the success rate of the post-selection step.

Figure 8 shows the result. We also varied the selectivity between 1% and 10%, and varied B from 4 to 64 to study their performance. The observations are almost the same as the 1-dimensional cases. The only difference is that a larger k did not lead to fewer IOs needed by the quantum range tree. The reason is that a larger k leads to more candidate nodes for the subsequent local quantum search, which is different from the 1-dimensional case.

In conclusion, the quantum trees performs far better than the classical trees from the perspective of the number of memory accesses. In 1-dimensional case, a larger k and a smaller B leads to a better performance. In 2-dimensional case, a smaller B leads to a better performance. Since the quantum B+ tree and the quantum range tree needs much fewer memory accesses to answer a range query, we conclude that they have the potential to outperform any classical data structure.

7 CONCLUSION

In this paper, we study the quantum range query problem. We propose the static quantum B+ tree that answers a static quantum range query in $O(\log_B N)$ time, which is asymptotically optimal in quantum computers and exponentially faster than classical B+ trees. Furthermore, we extend it to a dynamic quantum B+ tree. The dynamic quantum B+ tree can support insertions and deletions in $O(\log_B N)$ time and answer a dynamic quantum range query in $O(\log_B^2 N)$ time. We also discuss the high-dimensional quantum range query. Based on the quantum B+ tree, we obtain a quantum range tree which answers a d-dimensional quantum range query in $O(\log_B^d N)$ time, which is also cannot be achieved by any classical algorithm. In the experiment, we did simulations to confirm the exponential speedup. We expect that the quantum data structures will show advantages in the real world.

In the future, we will study the following two questions.

(1) Can we have a better index complexity in quantum data structures? To answer a range query, a dynamic quantum B+ tree needs $O(\log_B^2 N)$ time and a quantum range tree needs $O(\log_B^d N)$ time. However, the classical data structures can answer a 1-dimensional range query in $O(\log_B N + k)$ time [20] and answer a d-dimensional range query in $O(\log^{d-1} N + k)$ time [18]. Although the quantum data structures is exponentially faster than them, whether we could have the same index complexity as the best classical data structure is still an open problem.

(2) Can we answer a quantum query instead of classical queries? In our discussion, a query $QUERY(x, y)$ is given in classical bits. Consider a sequence of queries $QUERY(x_1, y_1)$, $QUERY(x_2, y_2)$, $\cdots$, $QUERY(x_t, y_t)$. If the input is in quantum bits like $\frac{1}{\sqrt{t}} \sum_{i=1}^{t} |x_i\rangle |y_i\rangle$, the classical queries becomes a quantum query. How to answer a quantum query is also an interesting question.

REFERENCES

[1] Zainab Abohashima, Mohamed Elhosen, Essam H Houssein, and Waleed M Mohamed. 2020. Classification with Quantum Machine Learning: A Survey. *arXiv e-prints* (2020), arXiv–2006.

[2] Soumik Adhikary, Siddharth Dangwal, and Debanjan Bhowmik. 2020. Supervised learning with a quantum classifier using multi-level systems. *Quantum Information Processing* 19, 3 (2020), 1–12.

[3] Andris Ambainis. 1999. A better lower bound for quantum algorithms searching an ordered list. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*. IEEE, 352–357.

[4] Lars Arge and Jeffrey Scott Vitter. 1996. Optimal dynamic interval management in external memory. In *Proceedings of 37th Conference on Foundations of Computer Science*. IEEE, 560–569.

[5] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

[6] Rudolf Bayer and Edward McCreight. 2002. Organization and maintenance of large ordered indexes. In *Software pioneers*. Springer, 245–262.

[7] Marcello Benedetti, Delfina Garcia-Pintos, Oscar Perdomo, Vicente Leyton-Ortega, Yunseong Nam, and Alejandro Perdomo-Ortiz. 2019. A generative modeling approach for benchmarking and training shallow quantum circuits. *npj Quantum Information* 5, 1 (2019), 1–9.

[8] Jon Louis Bentley. 1978. *Decomposable searching problems.* Technical Report. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE.

[9] Jon Louis Bentley and James B Saxe. 1980. Decomposable searching problems I. Static-to-dynamic transformation. *Journal of Algorithms* 1, 4 (1980), 301–358.

[10] Axel Berg, Magnus Oskarsson, and Mark O'Connor. 2021. Deep ordinal regression with label diversity. In *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2740–2747.

[11] Claudio Biancalana, Fabio Gasparetti, Alessandro Micarelli, Alfonso Miola, and Giuseppe Sansonetti. 2011. Context-aware movie recommendation based on signal processing and machine learning. In *Proceedings of the 2nd Challenge on Context-Aware Movie Recommendation*. 5–10.

[12] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J Bremner, John M Martinis, and Hartmut Neven. 2018. Characterizing quantum supremacy in near-term devices. *Nature Physics* 14, 6 (2018), 595–600.

[13] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. 1998. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics* 46, 4-5 (1998), 493–505.

[14] Gilles Brassard, Peter Høyer, and Alain Tapp. 1998. Quantum counting. In *International Colloquium on Automata, Languages, and Programming*. Springer, 820–831.

[15] Pedro G Campos, Alejandro Bellogín, Fernando Díez, and J Enrique Chavarriaga. 2010. Simple time-biased KNN-based recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*. 20–23.

[16] Sanjay Chakraborty, Soharab Hossain Shaikh, Amlan Chakrabarti, and Ranjan Ghosh. 2020. A hybrid quantum feature selection algorithm using a quantum inspired graph theoretic approach. *Applied Intelligence* 50, 6 (2020), 1775–1793.

[17] Bernard Chazelle. 1990. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM (JACM)* 37, 2 (1990), 200–212.

[18] Bernard Chazelle. 1990. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM (JACM)* 37, 2 (1990), 200–212.

[19] Bernard Chazelle. 1990. Lower bounds for orthogonal range searching: II. The arithmetic model. *Journal of the ACM (JACM)* 37, 3 (1990), 439–463.

[20] Douglas Comer. 1979. Ubiquitous B-tree. *ACM Computing Surveys (CSUR)* 11, 2 (1979), 121–137.

[21] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[22] Paul Adrien Maurice Dirac. 1939. A new notation for quantum mechanics. In *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 35. Cambridge University Press, 416–418.

[23] James Dougherty, Ron Kohavi, and Mehran Sahami. 1995. Supervised and unsupervised discretization of continuous features. In *Machine learning proceedings 1995*. Elsevier, 194–202.

[24] Christoph Durr and Peter Hoyer. 1996. A quantum algorithm for finding the minimum. *arXiv preprint quant-ph/9607014* (1996).

[25] Tobias Fankhauser, Marc E Solèr, Rudolf M Füchslin, and Kurt Stockinger. 2021. Multiple query optimization using a hybrid approach of classical and quantum computing. *arXiv preprint arXiv:2107.10508* (2021).

[26] Aleta Berk Finnila, MA Gomez, C Sebenik, Catherine Stenson, and Jimmie D Doll. 1994. Quantum annealing: A new method for minimizing multidimensional functions. *Chemical physics letters* 219, 5-6 (1994), 343–348.

[27] Bacon D Gidney C and contributors. 2018. Cirq: A python framework for creating, editing, and invoking noisy intermediate scale quantum (NISQ) circuits. https://github.com/quantumlib/Cirq.

[28] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 212–219.

[29] Lov K Grover and Jaikumar Radhakrishnan. 2005. Is partial quantum search of a database any easier?. In *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*. 186–194.

[30] Anant Gupta and Kuldeep Singh. 2013. Location based personalized restaurant recommendation system for mobile environments. In *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 507–511.

[31] Jacques Hadamard. 1893. Resolution d'une question relative aux determinants. *Bull. des sciences math.* 2 (1893), 240–246.

[32] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum algorithm for linear systems of equations. *Physical review letters* 103, 15 (2009), 150502.

[33] Akinori Hosoyamada and Yu Sasaki. 2018. Quantum Demiric-Selçuk meet-in-the-middle attacks: applications to 6-round generic Feistel constructions. In *International Conference on Security and Cryptography for Networks*. Springer, 386–403.

[34] Nabil Ibtehaz, M Kaykobad, and M Sohel Rahman. 2021. Multidimensional segment trees can do range updates in poly-logarithmic time. *Theoretical Computer Science* 854 (2021), 30–43.

[35] Ashish Kapoor, Nathan Wiebe, and Krysta Svore. 2016. Quantum perceptron models. *Advances in neural information processing systems* 29 (2016).

[36] Ruslan Kapralov, Kamil Khadiev, Joshua Mokut, Yixin Shen, and Maxim Yagafarov. 2020. Fast Classical and Quantum Algorithms for Online k-server Problem on Trees. *arXiv preprint arXiv:2008.00270* (2020).

[37] Iordanis Kerenidis, Jonas Landman, Alessandro Luongo, and Anupam Prakash. 2019. q-means: A quantum algorithm for unsupervised machine learning. *Advances in Neural Information Processing Systems* 32 (2019).

[38] Iordanis Kerenidis and Anupam Prakash. 2017. Quantum Recommendation Systems. In *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[39] Maria Kieferova, Ortiz Marrero Carlos, and Nathan Wiebe. 2021. Quantum Generative Training Using R\'enyi Divergences. *arXiv preprint arXiv:2106.09567* (2021).

[40] Jure Leskovec and Rok Sosič. 2016. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 1–20.

[41] Tongyang Li, Shouvanik Chakrabarti, and Xiaodi Wu. 2019. Sublinear quantum algorithms for training linear and kernel-based classifiers. In *International Conference on Machine Learning*. PMLR, 3815–3824.

[42] Tongyang Li, Chunhao Wang, Shouvanik Chakrabarti, and Xiaodi Wu. 2021. Sublinear Classical and Quantum Algorithms for General Matrix Games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8465–8473.

[43] Gui-Lu Long and Yang Sun. 2001. Efficient scheme for initializing a quantum register with an arbitrary superposed state. *Physical Review A* 64, 1 (2001), 014303.

[44] Guang Hao Low, Theodore J Yoder, and Isaac L Chuang. 2014. Quantum inference on Bayesian networks. *Physical Review A* 89, 6 (2014), 062315.

[45] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. 2018. Quantum circuit learning. *Physical Review A* 98, 3 (2018), 032309.

[46] Ashley Montanaro. 2017. Quantum pattern matching fast on average. *Algorithmica* 77, 1 (2017), 16–39.

[47] María Naya-Plasencia and André Schrottenloher. 2020. Optimal Merging in Quantum k-xor and k-sum Algorithms. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 311–340.

[48] Michael A Nielsen and Isaac L Chuang. 2001. Quantum computation and quantum information. *Phys. Today* 54, 2 (2001), 60.

[49] Mark H Overmars. 1983. *The design of dynamic data structures*. Vol. 156. Springer Science & Business Media.

[50] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica* 33, 4 (1996), 351–385.

[51] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. 2014. Quantum support vector machine for big data classification. *Physical review letters* 113, 13 (2014), 130503.

[52] Yue Ruan, Xiling Xue, Heng Liu, Jianing Tan, and Xi Li. 2017. Quantum algorithm for k-nearest neighbors classification based on the metric of hamming distance. *International Journal of Theoretical Physics* 56, 11 (2017), 3496–3507.

[53] Seyran Saeedi and Tom Arodz. 2019. Quantum sparse support vector machines. *arXiv preprint arXiv:1902.01879* (2019).

[54] Manuel Schönberger. 2022. Applicability of quantum computing on database query optimization. In *Proceedings of the 2022 International Conference on Management of Data*. 2512–2514.

[55] Erwin Schrödinger. 1935. Discussion of probability relations between separated systems. In *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 31. Cambridge University Press, 555–563.

[56] Maria Schuld, Alex Bocharov, Krysta M Svore, and Nathan Wiebe. 2020. Circuit-centric quantum classifiers. *Physical Review A* 101, 3 (2020), 032308.

[57] Maria Schuld and Nathan Killoran. 2019. Quantum machine learning in feature Hilbert spaces. *Physical review letters* 122, 4 (2019), 040504.

[58] Pranab Sen and Srinivasan Venkatesh. 2001. Lower bounds in the quantum cell probe model. In *International Colloquium on Automata, Languages, and Programming*. Springer, 358–369.

[59] Peter W Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 124–134.

[60] Barbara M Terhal. 2018. Quantum supremacy, here we come. *Nature Physics* 14, 6 (2018), 530–531.

[61] Immanuel Trummer and Christoph Koch. 2015. Multiple query optimization on the D-Wave 2X adiabatic quantum computer. *arXiv preprint arXiv:1510.06437* (2015).

[62] Valter Uotila. 2022. Synergy between Quantum Computers and Databases. (2022).

[63] Nathan Wiebe, Daniel Braun, and Seth Lloyd. 2012. Quantum algorithm for data fitting. *Physical review letters* 109, 5 (2012), 050505.

[64] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. 2015. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *Quantum Information & Computation* 15, 3-4 (2015), 316–356.

[65] Robert Wille, Rod Van Meter, and Yehuda Naveh. 2019. IBM's Qiskit tool chain: Working with and developing for real quantum computers. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1234–1240.

[66] Yanbing Xue and Milos Hauskrecht. 2017. Efficient learning of classification models from soft-label information by binning and ranking. In *The Thirtieth International Flairs Conference*.

[67] Andrew Chi-Chih Yao. 1981. Should tables be sorted? *Journal of the ACM (JACM)* 28, 3 (1981), 615–628.

[68] Kun Zhang and Vladimir Korepin. 2018. Quantum partial search for uneven distribution of multiple target items. *Quantum Information Processing* 17, 6 (2018), 1–20.