mathematics

MDPI

Article

Quantum Algorithms for Some Strings Problems Based on Quantum String Comparator †

Kamil Khadiev [1,*], Artem Ilikaev [1] and Jevgenijs Vihrovs [2]

[1] Institute of Computational Mathematics and Information Technologies, Kazan Federal University, Kremlevskaya 18, 420008 Kazan, Russia; artemka.tema1998@gmail.com
[2] Center for Quantum Computer Science, Faculty of Computing, University of Latvia, Raina 19, LV-1586 Riga, Latvia; jevgenijs.vihrovs@lu.lv
* Correspondence: kamil.hadiev@kpfu.ru
† This paper is an extended version of our paper published in Proceedings of TPNC2019 Conference.

Abstract: We study algorithms for solving three problems on strings. These are sorting of n strings of length k, "the Most Frequent String Search Problem", and "searching intersection of two sequences of strings". We construct quantum algorithms that are faster than classical (randomized or deterministic) counterparts for each of these problems. The quantum algorithms are based on the quantum procedure for comparing two strings of length k in $O(\sqrt{k})$ queries. The first problem is sorting n strings of length k. We show that classical complexity of the problem is $\Theta(nk)$ for constant size alphabet, but our quantum algorithm has $\tilde{O}(n\sqrt{k})$ complexity. The second one is searching the most frequent string among n strings of length k. We show that the classical complexity of the problem is $\Theta(nk)$, but our quantum algorithm has $\tilde{O}(n\sqrt{k})$ complexity. The third problem is searching for an intersection of two sequences of strings. All strings have the same length k. The size of the first set is n, and the size of the second set is m. We show that the classical complexity of the problem is $\Theta((n+m)k)$, but our quantum algorithm has $\tilde{O}((n+m)\sqrt{k})$ complexity.

Keywords: quantum computation; quantum algorithms; string processing; sorting

Citation: Khadiev, K.; Ilikaev, A.; Vihrovs, J. Quantum Algorithms for Some Strings Problems Based on Quantum String Comparator. *Mathematics* **2022**, *10*, 377. https://doi.org/10.3390/math10030377

Academic Editors: Fernando L. Pelayo, Mauro Mezzini and Jan Sładkowski

Received: 12 November 2021
Accepted: 22 January 2022
Published: 26 January 2022

1. Introduction

Quantum computing [1–3] is one of the hot topics in computer science in the last few decades. There are many problems where quantum algorithms outperform the best known classical algorithms [4–12].

One of these problems are problems for strings. Researchers show the power of quantum algorithms for such problems in [13–22].

In this paper, we consider three problems:

- Strings Sorting problem;
- the Most Frequent String Search problem;
- Intersection of Two String Sequences problem.

Our algorithms use some quantum algorithms as a subroutine, and the remaining part is classical. We investigate the problems in terms of query complexity. The query model is one of the most popular in the case of quantum algorithms. Such algorithms can do a query to a black box that has access to the sequence of strings. As a running time of an algorithm, we mean a number of queries to the black box.

In the paper, we suggested a quantum comparison procedure for two strings. We show that its quantum complexity is $\Theta(\sqrt{k})$, where k is the length of strings. The classical complexity is $\Theta(k)$. Thus, the quantum algorithm has a quadratic speed-up compared to classical algorithms. We propose a quantum algorithm that is based on "the first one search" (The minimal element satisfying a condition) problem algorithm from [23–26]. This algorithm is a modification of Grover's search algorithm [27,28]. Another important

algorithm for the search is described in [29]. Using this idea, we obtain quantum algorithms for several problems.

The first problem is the String Sorting problem. Assume that we have n strings of length k. It is known [30] that no quantum algorithm can sort arbitrary comparable objects faster than $O(n \log n)$. At the same time, several researchers tried to improve the hidden constant [31,32]. Other researchers investigated the space bounded case [33]. We focus on sorting strings. In a classical case, we can use an algorithm that is better than arbitrary comparable objects sorting algorithms. It is radix sort that has $O(nk)$ query complexity [34] for a finite size alphabet. It is also a lower bound for classical (randomized or deterministic) algorithms that is $\Omega(nk)$. Our quantum algorithm for the string sorting problem has query complexity $O(n(\log n) \cdot \sqrt{k}) = \tilde{O}(n\sqrt{k})$, where $\tilde{O}$ does not consider log factors. It is based on standard sorting algorithms [34] or Heapsort [34,35] and the quantum algorithm for comparing strings. Additionally, we use the idea of a noisy comparison procedure for sorting [36].

The second problem is the following. We have n strings of length k. We can assume that string symbols are letters from any constant size alphabet, for example, binary, Latin alphabet, or Unicode. The problem is finding the string that occurs in the sequence most often. The problem [37] is one of the most well-studied ones in the area of data streams [38–41]. Many applications in packet routing, telecommunication logging, and tracking keyword queries in search machines are critically based on such routines. The best-known classical (randomized or deterministic) algorithms require $\Omega(nk)$ queries because an algorithm should at least test all symbols of all strings. The deterministic solution can use the radix sort algorithm [34] or the Trie (prefix tree) [42–45] that allow achieving the required complexity.

We propose a quantum algorithm that is based on the sorting algorithm from the first problem. Our algorithm for the most frequent string search problem has query complexity $O(n(\log n) \cdot \sqrt{k}) = \tilde{O}(n\sqrt{k})$. If $\log_2 n = o(\sqrt{k})$, then our algorithm is better than classical counterparts. Note that this setup makes sense in practical cases.

The third problem is the Intersection of Two String Sequences problem. Assume that we have two sequences of strings of length k. The size of the first set is n, and the size of the second one is m. The first sequence is given, and the second one is given in an online fashion, one by one. After each requested string from the second sequence, we want to check whether this string belongs to the first sequence. We propose a quantum algorithm for the problem with quantum query complexity $O((n + m(\log m + \log\log n)) \cdot \log n \cdot \sqrt{k}) = \tilde{O}((n + m)\sqrt{k})$. The algorithm uses a quantum algorithm for sorting strings. At the same time, the best-known classical (randomized or deterministic) algorithm requires $\Omega((n + m)k)$ queries, and this bound is achieved using the radix sort algorithm or the Trie data structure.

The paper is an extended version of a conference paper [46].

The structure of the paper is the following. Discussion on the computation model is situated in Section 2. We present the quantum subroutine that compares two strings in Section 3. Then, we discuss three problems: Strings Sorting problem in Section 4, the Most Frequent String Search problem in Section 5, and Intersection of Two String Sequences problem in Section 6. Section 7 contains the conclusions.

2. Preliminaries

We use the standard form of the quantum query model. Let $f : D \to \{0,1\}, D \subseteq \{0,1\}^N$ be an N variable function. An input for the function is $x \in D$. We are given an oracle access to the input x, i.e., it is realized by a specific unitary transformation usually defined as $|i\rangle|z\rangle|w\rangle \to |i\rangle|z + x_i \pmod 2\rangle|w\rangle$, where the $|i\rangle$ register indicates the index of the variable we are querying, $|z\rangle$ is the output register, and $|w\rangle$ is some auxiliary workspace. Note that we use Dirac notation vectors. An algorithm in the query model consists of alternating applications of arbitrary unitaries independent of the input and the query unitary and a measurement in the end.

In the case of non-binary input, we present the input variables in binary form. Using alternating unitaries independent of the input, we can store bits in auxiliary work-space $|w\rangle$ and use the obtained variable in an algorithm. In the case of computing a complex function f and additionally non-binary input, we can consider a block of alternating unitaries independent of the input and the query unitary that stores required variables in the auxiliary work-space $|w\rangle$. Then, we compute the Boolean value of the function f on arguments and store them in the auxiliary work-space $|w\rangle$. After that, we can use the value of the function f in our algorithms.

The smallest number of queries for an algorithm that outputs $f(x)$ with a probability that is at least $\frac{2}{3}$ on all x is called the quantum query complexity of the function f and is denoted by $Q(f)$. We refer the readers to [1–3] for more details on quantum computing.

In the quantum algorithms in this article, we discuss quantum query complexity. We use modifications of Grover's search algorithm [27,28] as quantum subroutines. For these subroutines, time complexity (number of gates in a circuit) is more than query complexity for an additional log factor. Note that the query can be implemented using the CNOT gate.

3. The Quantum Algorithm for Comparing Two Strings

Firstly, we discuss a quantum subroutine that compares two strings of length k. Assume that this subroutine is COMPARE_STRINGS(s, t, k), and it compares s and t in the lexicographical order. It returns:

- -1 if $s < t$;
- 0 if $s = t$;
- 1 if $s > t$.

As a base for our algorithm, we use the algorithm of finding the minimal argument with 1-result of a Boolean-value function. Formally, we have:

Lemma 1 ([24,25], Theorem 10; [23], Section 2.2; [26], Proposition 4)**.** *Suppose we have a function $f : \{1, \ldots, N\} \to \{0, 1\}$ for some integer N. There is a quantum algorithm for finding $j_0 = \min\{j \in \{1, \ldots, N\} : f(j) = 1\}$. The algorithm finds j_0 with the expected query complexity $O(\sqrt{j_0})$ and error probability that is, at most, $\frac{1}{2}$.*

Let us choose the function $f(j) = (s_j \neq t_j)$. Thus, we search for j_0 that is the index of the first unequal symbol of the strings. Then, we can claim that s precedes t in the lexicographical order if the symbol s_{j_0} precedes the symbol t_{j_0}. The claim is right by the definition of the lexicographical order. If there are no unequal symbols, then the strings are equal.

If we discuss the implementation of the f, then we can say that for computing the value $f(j)$, we store the binary representation of s_j and t_j in the auxiliary work-space, for example, $|\psi_s\rangle$ and $|\psi_t\rangle$. Then, compute the value of $f(j)$ and store it in a qubit $|\phi\rangle$. After that, we can use this value in the algorithm. The last step is clearing $|\phi\rangle$ using values of $|\psi_s\rangle$ and $|\psi_t\rangle$ and the CNOT gate; then, clearing $|\psi_s\rangle$ and $|\psi_t\rangle$ repeatedly using the same queries (that use CNOT gates). All these manipulations take a constant number of queries because of the constant size of the input alphabet.

We use THE_FIRST_ONE_SEARCH(f, k) as a subroutine from Lemma 1, where $f(j) = (s_j \neq t_j)$. Assume that this subroutine returns $k + 1$ if it does not find any solution or the found argument j' is such that $f(j') = 0$.

We use the standard technique of boosting success probability. Thus, we repeat the subroutine $\lceil \log_2(\delta^{-1}) \rceil$ times and return the minimal answer.

Suppose the subroutine has an error. There are two cases. The first one is finding the index of unequal symbols that is not the minimal one. In the second case, the algorithm does not find unequal symbols. Then, we assume that it returns $k + 1$. Thus, in a case of an error, the subroutine returns a value that is bigger than the correct answer.

Therefore, if at least one subroutine invocation has no error, then the whole algorithm succeeds. All error events are independent. The error probability of the whole algorithm is the probability of error for all invocations of the subroutine, that is $O\left(\frac{1}{2^{\log_2(\delta^{-1})}}\right) = O(\delta)$.

Let us present the Algorithm 1.

Algorithm 1 COMPARE_STRINGS(s, t, k). The Quantum Algorithm for Comparing Two Strings.

```
j_0 ← THE_FIRST_ONE_SEARCH(f, k)                    ▷ The initial value
for i ∈ {1, ..., ⌈log_2 δ^{-1}⌉} do
    j_0 ← min(j_0, THE_FIRST_ONE_SEARCH(f, k))
end for
if j_0 = k + 1 then
    result ← 0                                      ▷ The strings are equal.
else
    if (s_{j_0} < t_{j_0}) then
        result ← −1                                 ▷ s precedes t.
    else
        result ← 1                                  ▷ s succeeds t.
    end if
end if
return result
```

The next property follows from the previous discussion.

Lemma 2. *Algorithm 1 compares two strings of length k in the lexicographical order with query complexity* $O(\sqrt{k}\log\delta^{-1})$ *and error probability* $O(\delta)$ *for some integer k and* $0 < \delta < 1$.

The algorithm finds the minimal index of unequal symbols j_0. We can say that $j_0 - 1$ is the length of the longest common prefix for these strings.

We can show that the lower bound for the problem is $\Omega(\sqrt{k})$.

Lemma 3. *Any quantum algorithm for Comparing Two Strings problem has* $\Omega(\sqrt{k})$ *query complexity.*

Proof. Let us show that the problem is at least as hard as the unstructured search problem. Let $s_{\lfloor k/2 \rfloor} = 1$ and $s_j = 0$ for all $j \in \{1, \ldots, \lfloor k/2 \rfloor - 1, \lfloor k/2 \rfloor + 1, \ldots k\}$. The string t is such that there is only one 1 in position z. In other words, there is $z \in \{1, \ldots, k\}$ such that $t_z = 1$ and $t_j = 0$ for all $j \in \{1, \ldots, z-1, z+1, \ldots k\}$.

If $z < \lfloor k/2 \rfloor$, then $t > s$. If $z = \lfloor k/2 \rfloor$, then $t = s$. If $z > \lfloor k/2 \rfloor$, then $t < s$. Therefore, the problem is at least as hard as the search for 1 among the first $\lfloor k/2 \rfloor$ variables in the string t.

It is known [14] that the quantum query complexity of the unstructured search among $\lfloor k/2 \rfloor$ variables is $\Omega(\sqrt{k})$. □

At the same time, the classical complexity of the problem is $\Theta(k)$.

Lemma 4. *Randomized query complexity for Comparing Two Strings problem is* $\Theta(k)$.

Proof. Due to the proof of Lemma 3, the problem is at least as hard as the search for 1 among the first $k/2$ variables in the string t.

It is known [14] that the randomized query complexity of the unstructured search among $k/2$ variables is $\Omega(k)$.

At the same time, we can check all symbols sequentially to search the first unequal symbol. This algorithm has $O(k)$ query complexity. □

Additionally, we can compute the complexity of any algorithm based on the two strings comparison procedure.

Lemma 5. *Suppose we have some integer n, integer $A = A(n)$ and ε such that $\lim\limits_{n\to\infty} \varepsilon/A = 0$. Then, if a quantum algorithm does $A(n)$ comparisons of strings of length k and has $O(\varepsilon)$ error probability, then it does at most $O(A\sqrt{k}\log(A/\varepsilon))$ queries.*

Proof. As a strings comparison procedure, we use COMPARE_STRINGS subroutine for $\delta = \varepsilon/A$. Because of Lemma 2, the complexity of the subroutine is $O(\sqrt{k}\log(A/\varepsilon))$, and the error probability is $O(\varepsilon/A)$. Because of A comparison operations, the total complexity of the algorithm is $O(A\sqrt{k}\log(A/\varepsilon))$.

Let us discuss the error probability. Events of error in the algorithm are independent. Thus, all events should be correct. The error probability for one event is $1-(1-\varepsilon/A)$. Hence, the error probability for all A events is at least $1-(1-\varepsilon/A)^A = 1-(1-\varepsilon/A)^A$.

Note that

$$\lim_{n\to\infty} \frac{1-\left(1-\frac{\varepsilon}{A}\right)^A}{\varepsilon} = \lim_{n\to\infty} \frac{1-\left(1-\frac{\varepsilon}{A}\right)^{\frac{A}{\varepsilon}\cdot\varepsilon}}{\varepsilon} \leq 1;$$

Hence, the total error probability is at most $O(\varepsilon)$. □

4. Strings Sorting Problem

Let us consider the following problem.

Problem. For some positive integers n and k, we have the sequence of strings $s = (s^1, \ldots, s^n)$. Each $s^i = (s^i_1, \ldots, s^i_k) \in \Sigma^k$ for some finite size alphabet Σ. We search an order $ORDER = (i_1, \ldots, i_n)$ such that for any $j \in \{1, \ldots, n-1\}$, we have $s^{i_j} \leq s^{i_{j+1}}$ in the lexicographical order.

We use one of the existing sorting algorithms (for example, Heapsort algorithm [34,35] or the Merge sort algorithm [34]) as a base and the quantum algorithm for string comparison from Section 3. In fact, our comparison function can have errors. That is why we use the result for "noisy computation" from [36]. The result is presented in the following lemma.

Lemma 6 ([36], Theorem 3.5)**.** *Suppose we have a comparison procedure that works with error probability ε. Then there is a sorting algorithm with query complexity $O(n\log(n/\varepsilon))$ and error probability at most ε.*

The complexity of the algorithm is presented in the following theorem.

Theorem 1. *The algorithm sorts $s = (s^1, \ldots, s^n)$ with query complexity $O(n(\log n)\cdot\sqrt{k}) = \tilde{O}(n\sqrt{k})$ and constant error probability.*

Proof. The correctness of the algorithm follows from the description. Let $\varepsilon = 0.1$. Then, we apply the result from Lemma 6 and use the quantum comparison procedure that has ε error probability and $O(\sqrt{k}\log\varepsilon^{-1}) = O(\sqrt{k})$ query complexity. Therefore, the query complexity of the algorithm is $O(n(\log(n/\varepsilon))\cdot\sqrt{k}) = O(n(\log n)\cdot\sqrt{k}) = \tilde{O}(n\sqrt{k})$, and the error probability is ε. □

We can show the lower bound for the problem.

Theorem 2. *Any quantum algorithm for the Sorting problem has $\Omega(\sqrt{nk})$ query complexity.*

Proof. Let us show that the problem is at least as hard as the unstructured search problem. Assume that strings $s^1, \ldots, s^n$ are such that

- There is a pair (u, v) such that $s^u_v = 1$;
- For all pairs $(i, j) \neq (u, v)$, $s^i_j = 0$.

In that case, the answer is $ORDER = (i_1, \ldots, i_{n-1}, u)$, where $(i_1, \ldots, i_{n-1})$ is a permutation of integers from $\{1, \ldots, u-1, u+1, \ldots, n\}$. The searching for the required index u is at least as hard as the search for the 1-value variable s^u_v.

It is known [14] that the quantum complexity of the unstructured search among nk variables is $\Omega(\sqrt{nk})$. □

The lower bound for classical complexity can be proven by the same way as in Theorem 2.

Theorem 3. *The randomized query complexity of the Sorting problem is* $\Theta(nk)$.

Proof. Due to the proof of Theorem 2, the problem is at least as hard as the search for 1 among nk variables in the strings $s^1, \ldots, s^n$.

It is known [14] that the randomized query complexity of the unstructured search among nk variables is $\Omega(nk)$.

The Radix sort [34] algorithm reaches this bound and has $O(nk)$ complexity in a case of a finite alphabet. □

5. The Most Frequent String Search Problem

Let us formally present the problem.

Problem. For some positive integers n and k, we have a sequence of strings $s = (s^1, \ldots, s^n)$. Each $s^i = (s^i_1, \ldots, s^i_k) \in \Sigma^k$ for some finite size alphabet Σ. Let $\#(t) = |\{i \in \{1, \ldots, n\} : s^i = t\}|$ be the number of occurrences of a string t. We search for $i = argmax_{i \in \{1,\ldots n\}} \#(s^i)$. If several strings satisfy the condition, then the answer is the index of the string with minimal index in the set s. Formally, i is such that:

$$i = min\{j : \#(s^j) = \max_{z \in \{1,\ldots,n\}} \#(s^z)\}$$

Firstly, we present an idea of the algorithm.

The algorithm contains two steps. The first step is sorting the sequence of strings and obtaining $ORDER = (i_1, \ldots, i_n)$ such that for any $j \in \{1, \ldots, n-1\}$, we have $s^{i_j} \leq s^{i_{j+1}}$ in the lexicographical order. In that case, equal strings are situated sequentially. On the second step, we find each segment $[i_\ell, i_r]$ of indexes for equal strings, i.e., $s^j = s^{i_\ell}$ for $j \in \{i_\ell, \ldots, i_r\}$ and $s^{i_{\ell-1}} \neq s^{i_\ell}$ or $\ell = 1$, and $s^{i_{r+1}} \neq s^{i_r}$ or $r = n$. We check segments for different strings one by one. We store the longest segment's length as c_{max} and the minimal index of the string that corresponds to this segment in j_{max}. As in the sorting algorithm, in the second step of the algorithm, we apply the COMPARE_STRINGS subroutine for checking the equality of strings. Assume that we have the SORT_STRINGS(s) subroutine that implements the algorithm from Section 4.

Let us present the algorithm formally in Algorithm 2.

Let us discuss the complexity of the algorithm.

Theorem 4. *Algorithm 2 finds the most frequent string from* $s = (s^1, \ldots, s^n)$ *with query complexity* $O(n(\log n) \cdot \sqrt{k}) = \tilde{O}(n\sqrt{k})$ *and constant error probability.*

Proof. The correctness of the algorithm follows from the description. Let us discuss the query complexity. Because of Theorem 1, the sorting algorithm's complexity is $O(n(\log n)\sqrt{k})$, and the error probability is constant. The second step does $O(n)$ comparison operations. Let $\varepsilon' = 0.1$. Thus, because of Lemma 5, the second step of the algorithm algorithm does $O(n(\log n)\sqrt{k})$ queries, and the error probability is constant. The total complexity is $O(n(\log n)\sqrt{k} + n(\log n)\sqrt{k}) = O(n(\log n)\sqrt{k})$.

Error events of two steps are independent. Therefore, the error probability of the whole algorithm is also constant. We can achieve any required constant error probability by repetition. The technique is standard in both one-side and two-side errors. It can be seen, for example, in [16]. □

Algorithm 2 The Quantum Algorithm for the Most Frequent String Problem.

$(i_1, \ldots, i_n) = ORDER \leftarrow$ SORT_STRINGS(s) ▷ We sort $s = (s^1, \ldots, s^n)$.
$c_{max} \leftarrow 0, j_{max} \leftarrow -1$
$c \leftarrow 1, j \leftarrow i_1$
for $b \in (1, \ldots, n)$ **do**
  if $b = n$ or ($b \neq n$ and COMPARE_STRINGS$(s^{i_b}, s^{i_{b+1}}, k) \neq 0$) **then** ▷ We find the end of a segment
    if $c > c_{max}$ **then** ▷ If the current segment is longer than the current longest one
      $c_{max} \leftarrow c, j_{max} \leftarrow j$
    end if
    $c \leftarrow 1$
    if $b \neq n$ **then**
      $j = i_{b+1}$
    end if
  else
    $c \leftarrow c + 1$
    if $i_{b+1} < j$ **then** ▷ j is the minimal index of the current segment
      $j \leftarrow i_{b+1}$.
    end if
  end if
end for
return j_{max}

Theorem 5. *Suppose we have a constant ε such that $0 < \varepsilon < 3/4$. If the length of the strings $k \geq \log_2 n$, then any quantum algorithm for the Most Frequent String Search problem has $\Omega(\sqrt{nk} + n^{3/4-\varepsilon})$ query complexity. If $k < \log_2 n$, then any quantum algorithm for the Most Frequent String Search problem has $\Omega(\sqrt{nk})$ query complexity*

Proof. Let us show that the problem is at least as hard as the unstructured search problem. Assume that $n = 2t$ and $k > 1$ for some integer t. Then, let $s^{t+1}, \ldots, s^{2t} = 0^k$, where 0^k is a string of k zeros. Other strings can be $s^1, \ldots, s^t = 1^k$ or there are $z \in \{1, \ldots, t\}$ and $u \in \{1, \ldots, k\}$ such that $s^z_u = 0$ and $s^z_{u'} = 1$ for all $u' \in \{1, \ldots, u-1, u+1, \ldots, k\}$.

In the first case, the answer is 1^k. In the second case, the answer is 0^k. Therefore, solving the problem for this instance is equivalent to the search for 0 among the first $tk = nk/2$ variables.

According to [14], the quantum complexity of the unstructured search among $nk/2$ is $\Omega(\sqrt{nk})$.

In the case of odd n, we assign $s^n = 1^{k/2}0^{k/2}$, and it is not used in the search. Then, we can consider only $n-1$ strings. Thus, $n-1$ is even.

Let us consider the case of $k = 1$. If n is odd, then $s^n = 2$. Let $s^i = 0$ for $i \geq t+1$, and $t = \lfloor n/2 \rfloor$. Let us consider two cases. The first one is $s^i = 1$ for all $i \in \{1, \ldots, t\}$. The second case is $s^i = 1$ for all $i \in \{1, \ldots, t\} \backslash \{i_1\}$ and $s^{i_1} = 0$ for some $i_1 \in \{1, \ldots, t\}$. In the first case, the answer is 1. In the second case, the answer is 0. Therefore, solving the problem for this instance is equivalent to the search for 0 among the first $t = n/2 = nk/2$ variables.

Let us show that the problem is at least as hard as the d-distinctness problem [47]. Let d be such that $\frac{1}{4d} = \varepsilon/2$. Let b be the maximal integer that satisfies $n \geq b \cdot (d-1) + 1$. Let u^j be a binary representation of j for $j \in \{0, \ldots, b\}$.

Assume that $s^1 = u^1$ for other strings. We have two cases:

- **Case 1.** The sequence s contains $d-1$ copies of each u^j, where $j \geq 1$ and other strings are u^0. Formally:
  - $\#(u^j) = d-1$ for $j \in \{1, \ldots, b\}$;
  - $\#(u^0) = n - b \cdot (d-1)$.

- **Case 2.** The sequence s contains $d-1$ copies of each u^j, where $j \geq 1$ except some $j_m \in \{2,\ldots,b\}$; d copies of u^{j_m} and other strings are u^0. Formally:
  - $\#(u^{j_m}) = d$ for some $j_m \in \{2,\ldots,b\}$;
  - $\#(u^j) = d-1$ for $j \in \{1,\ldots,b\}\backslash\{j_m\}$;
  - $\#(u^0) = n - b\cdot(d-1)+1$.

In the first case, $\#(u^j) = d-1$ for $j \in \{1,\ldots,b\}$, $\#(u^0) \leq d-1$ and $s^1 = u^1$. Therefore, the answer is 1. In the second case, $\#(u^j) = d-1$ for $j \in \{1,\ldots,b\}\backslash\{j_m\}$, $\#(u^0) \leq d-1$ and $\#(u^{j_m}) = d$. Therefore, the answer is $i_m = \min\{i : s^i = u^{j_m}\}$. Note that $i_m \neq 1$ because $j_m \geq 2$ and $s^1 = u^1 \neq u^{j_m}$.

Hence, solving the problem for this instance is equivalent to checking whether there is a string that occurs in the input at least d times. It is the d-distinctness problem from [47]. It is known that the complexity of the problem is $\Omega\left(\frac{1}{4^d d^2 \cdot \log^{5/2} R}\cdot R^{3/4-1/(4d)}\right)$ for $R = \Theta(d^{d/2}n)$. In our case, the complexity is $\Omega(n^{3/4-\varepsilon})$. □

Secondly, let us discuss the classical complexity of the problem.

Theorem 6. *Any randomized algorithm for the Most Frequent String Search problem has* $\Theta(nk)$ *query complexity.*

Proof. The best-known classical algorithm uses the radix sort algorithm and does steps similar to the steps of the quantum algorithm.

The running time of this algorithm is $O(nk)$. At the same time, we can show that it is also a lower bound.

As it was shown in the proof of Theorem 5, the problem is at least as hard as the unstructured search problem among $nk/2$ variables. It is known [14] that the randomized complexity of the unstructured search among $nk/2$ variables is $\Omega(nk)$. □

6. Intersection of Two Sequences of Strings Problem

Let us consider the following problem.

Problem. For some positive integers n, m and k, we have the sequence of strings $s = (s^1,\ldots,s^n)$. Each $s^i = (s^i_1,\ldots,s^i_k) \in \Sigma^k$ for some finite size alphabet Σ. Then, we obtain m requests $t = (t^1 \ldots t^m)$, where $t^i = (t^i_1,\ldots,t^i_k) \in \Sigma^k$. The answer for a request t^i is 1 if there is $j \in \{1,\ldots,n\}$ such that $t^i = s^j$. We should answer 0 or 1 to each of m requests.

Let us present the algorithm that is based on the sorting algorithm from Section 4. We sort strings from s. Then, we answer each request using a binary search in the sorted sequence of strings [34] and COMPARE_STRINGS quantum subroutine for strings comparison during the binary search.

Let us present Algorithm 3. Assume that the sorting algorithm from Section 4 is the subroutine SORT_STRINGS(s), and it returns the order $ORDER = (i_1,\ldots,i_n)$. The subroutine BINARY_SEARCH_FOR_STRINGS$(t^i, s, ORDER)$ is the binary search algorithm with the COMPARE_STRINGS subroutine as a comparator, and it searches for t^i in the ordered sequence $(s^{i_1},\ldots,s^{i_n})$. Suppose that the subroutine BINARY_SEARCH_FOR_STRINGS returns 1 if it finds t and 0 otherwise.

Algorithm 3 The Quantum Algorithm for Intersection of Two Sequences of Strings Problem using sorting algorithm.

```
ORDER ← SORT_STRINGS(s)                          ▷ We sort s = (s^1,...,s^n).
for i ∈ {1,...,m} do
    ans ← BINARY_SEARCH_FOR_STRINGS(t^i, s, ORDER)  ▷ We search t^i in the ordered sequence.
    return ans
end for
```

The algorithms have the following query complexity.

Theorem 7. *Algorithm 3 solves Intersection of Two Sequences of Strings Problem with query complexity* $O((n+m)\sqrt{k}\cdot\log n\cdot\log(n+m)) = \tilde{O}((n+m)\sqrt{k})$ *and error probability* $O\left(\frac{1}{n+m}\right)$.

Proof. The correctness of the algorithm follows from the description.

Because of Theorem 1, the sorting algorithm's complexity is $O(n\log n\cdot\sqrt{k})$ and constant error probability.

Let us consider the second part of the algorithm. It does $O(m\log n)$ comparison operations for all invocations of the binary search algorithm. Let $\varepsilon = 0.1$. Thus, because of Lemma 5, the second part of the algorithm does

$$O(m\sqrt{k}\log n\log(m\log n)) = O(m\sqrt{k}\log n(\log m + \log\log n))$$

queries, and the error probability is constant.

Thus, the total complexity is $O((n + m(\log m + \log\log n))\sqrt{k}\log n)$. Error events of the two steps are independent. Therefore, the error probability of the whole algorithm is also constant. We can achieve any required constant error probability by repetition. □

The lower bound for the classical case can be proven using a result stated in [48] (Lemma 7, Section 5.1).

Theorem 8. *The randomized query complexity of Intersection of Two Sequences of Strings Problem is* $\Theta((n+m)k)$.

Proof. Assume that $n > m$. Let us consider $t^1 = 0^k$, and s^i contains only 0 s and 1 s, i.e., $s^i_j \in \{0,1\}$ for all $i \in \{1,\ldots,n\}, j \in \{1,\ldots,k\}$.

For checking $s^i = t^1$, it is enough to check $\neg \bigvee_{j=1}^{k}(s^i_j = 1)$ because this implies $s^i_j = 0$ for all $j \in \{1,\ldots,k\}$. In that case, checking for the existence of t^1 among s^i is the same as checking the following condition:

$$\neg \bigwedge_{i=1}^{n}\bigvee_{j=1}^{k}(s^i_j = 1)$$

This condition means that not all string s^i contains at least one 1.

The randomized complexity of computing $\neg \bigvee_{j=1}^{k}(s^i_j = 1)$ is the same as the complexity of the unstructured search for 1 among k variables, which is $\Omega(k)$. According to [48] (Lemma 7, Section 5.1), the total complexity of the function is $\Omega(nk)$.

Assume that $m > n$. Let us consider $s^i = 0^k$ for all $i \in \{1,\ldots,n\}$. The checking existence t^j among $s^1,\ldots,s^n$ is at least as hard as the search for 1 among $t^j_1,\ldots,t^j_k$ that requires $\Omega(k)$ queries. It is true for all $j \in \{1,\ldots,m\}$. Therefore, the total randomized complexity is $\Omega(mk)$.

Hence, if we join both cases, the randomized complexity of solving the problem is $\Omega(\max(n,m)\cdot k) = \Omega((n+m)\cdot k)$.

This complexity $O((n+m)k)$ can be reached if we use the radix sort algorithm and perform the same operations as in the quantum algorithm. □

Note that we can use the quantum algorithm for element distinctness [49,50] for this problem. The algorithm solves the problem of finding two identical elements in the sequence. The query complexity of the algorithm is $O(D^{2/3})$, where D is the number of elements in the sequence. The complexity is tight because of [51]. The algorithm can be the following. On j-th request, we can add the string t^j to the sequence $s^1,\ldots,s^n$ and invoke the element distinctness algorithm that finds a collision of t^j with other strings. Such approach

requires $\Omega(n^{2/3}\sqrt{k})$ queries for each request and $\Omega(mn^{2/3}\sqrt{k})$ for processing all requests. Note that the online nature of requests does not allow us to access all $t^1, \ldots, t^m$. Thus, each request should be processed separately.

In a case of $n \ll m$, we can use the Grover search algorithm for searching t^j among $(s_1, \ldots, s_n)$. The complexity is $\tilde{O}(m\sqrt{nk})$ in that case.

Because of the probabilistic behavior of the Oracle, we should use the approach similar to [52] that uses ideas of Amplitude Amplification [53].

7. Conclusions

In the paper, we propose a quantum algorithm for a comparison of strings and a general idea for any algorithm that does A string comparison operations. Then, using these results, we construct a quantum strings sorting algorithm that works faster than the radix sort algorithm, which is the best known deterministic algorithm for sorting a sequence of strings.

We propose quantum algorithms for two problems using the sorting algorithm: the Most Frequent String Search and Intersection of Two String Sequences. These quantum algorithms are more efficient than classical (deterministic or randomized) counterparts in a case of $\log_2(n) = o(\sqrt{k})$, where k is the length of strings and n is the number of strings. In a case of the Intersection of Two String Sequences problem, the condition is $\log_2(n)(\log_2 m + \log_2 \log_2 n) = o(\sqrt{k})$, where n and m are the number of strings in two sequences. Note that these assumptions are reasonable.

We discussed quantum and classical lower bounds for these problems. Classical lower bounds are tight, and at the same time, there is room to improve the quantum lower bounds.

Author Contributions: The main idea and algorithms, K.K. and A.I.; lower bounds, J.V. and K.K.; constructions and concepts, K.K., A.I. and J.V. All authors have read and agreed to the published version of the manuscript.

Funding: This paper has been supported by the Kazan Federal University Strategic Academic Leadership Program ("PRIORITY-2030"). J.V. Supported by the ERDF project 1.1.1.5/18/A/020 "Quantum algorithms: from complexity theory to experiment".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: We thank Aliya Khadieva, Farid Ablayev, Kazan Federal University quantum group and Krišjānis Prūsis from the University of Latvia for useful discussions.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Nielsen, M.A.; Chuang, I.L. *Quantum Computation and Quantum Information*; Cambridge University Press: Cambridge, UK, 2010.
2. Ambainis, A. Understanding Quantum Algorithms via Query Complexity. In Proceedings of the International Congress of Mathematicians, Rio de Janeiro, Brazil, 1–9 August 2018; Volume 4, pp. 3283–3304.
3. Ablayev, F.; Ablayev, M.; Huang, J.Z.; Khadiev, K.; Salikhova, N.; Wu, D. On quantum methods for machine learning problems part I: Quantum tools. *Big Data Min. Anal.* **2019**, *3*, 41–55. [CrossRef]
4. de Wolf, R. *Quantum Computing and Communication Complexity*; Institute for Logic, Language and Computation: Amsterdam, The Netherlands, 2001.
5. Jordan, S. Quantum Algorithms Zoo. 2021. Available online: http://quantumalgorithmzoo.org/ (accessed on 12 November 2021).
6. Khadiev, K.; Safina, L. Quantum Algorithm for Dynamic Programming Approach for DAGs. Applications for Zhegalkin Polynomial Evaluation and Some Problems on DAGs. In Proceedings of the UCNC, Tokyo, Japan, 3–7 June 2019; LNCS: Cham, Switzerland, 2019; Volume 4362, pp. 150–163.
7. Khadiev, K.; Kravchenko, D.; Serov, D. On the Quantum and Classical Complexity of Solving Subtraction Games. In Proceedings of the CSR 2019, Novosibirsk, Russia, 1–5 July 2019; LNCS: Cham, Switzerland, 2019; Volume 11532, pp. 228–236.

8. Khadiev, K.; Mannapov, I.; Safina, L. The Quantum Version Of Classification Decision Tree Constructing Algorithm C5. 0. In Proceedings of the CEUR Workshop Proceedings, Como, Italy, 9–11 September 2019; Volume 2500.
9. Kravchenko, D.; Khadiev, K.; Serov, D.; Kapralov, R. Quantum-over-Classical Advantage in Solving Multiplayer Games. *Lect. Notes Comput. Sci.* **2020**, *12448*, 83–98.
10. Khadiev, K.; Mannapov, I.; Safina, L. Classical and Quantum Improvements of Generic Decision Tree Constructing Algorithm for Classification Problem. *CEUR Workshop Proc.* **2021**, *2842*, 83–93.
11. Glos, A.; Nahimovs, N.; Balakirev, K.; Khadiev, K. Upper bounds on the probability of finding marked connected components using quantum walks. *Quantum Inf. Process.* **2021**, *20*, 6. [CrossRef]
12. Khadiev, K.; Safina, L. The quantum version of random forest model for binary classification problem. *CEUR Workshop Proc.* **2021**, *2842*, 30–35.
13. Montanaro, A. Quantum pattern matching fast on average. *Algorithmica* **2017**, *77*, 16–39. [CrossRef]
14. Bennett, C.H.; Bernstein, E.; Brassard, G.; Vazirani, U. Strengths and weaknesses of quantum computing. *SIAM J. Comput.* **1997**, *26*, 1510–1523. [CrossRef]
15. Ramesh, H.; Vinay, V. String matching in $O(\sqrt{n}+\sqrt{m})$ quantum time. *J. Discret. Algorithms* **2003**, *1*, 103–110. [CrossRef]
16. Ambainis, A.; Balodis, K.; Iraids, J.; Khadiev, K.; Kļevickis, V.; Prūsis, K.; Shen, Y.; Smotrovs, J.; Vihrovs, J. Quantum Lower and Upper Bounds for 2D-Grid and Dyck Language. In Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020), Prague, Czech Republic, 25–26 August 2020; Volume 170, pp. 8:1–8:14.
17. Khadiev, K.; Remidovskii, V. Classical and quantum algorithms for constructing text from dictionary problem. *Nat. Comput.* **2021**, *20*, 713–724. [CrossRef]
18. Khadiev, K.; Remidovskii, V. Classical and Quantum Algorithms for Assembling a Text from a Dictionary. *Nonlinear Phenom. Complex Syst.* **2021**, *24*, 207–221. [CrossRef]
19. Khadiev, K.; Kravchenko, D. Quantum Algorithm for Dyck Language with Multiple Types of Brackets. In Proceedings of Unconventional Computation and Natural Computation (UCNC 2021), Espoo, Finland, 18–22 October 2021; LNCS: Cham, Switzerland, 2021; Volume 12984, pp. 68–83.
20. Gall, F.L.; Seddighin, S. Quantum Meets Fine-grained Complexity: Sublinear Time Quantum Algorithms for String Problems. *arXiv* **2020**, arXiv:2010.12122.
21. Akmal, S.; Jin, C. Near-Optimal Quantum Algorithms for String Problems. *arXiv* **2021**, arXiv:2110.09696.
22. Ablayev, F.; Ablayev, M.; Khadiev, K.; Salihova, N.; Vasiliev, A. Quantum Algorithms for String Processing. *arXiv* **2020**, arXiv:2012.00372.
23. Kothari, R. An optimal quantum algorithm for the oracle identification problem. In Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science, Lyon, France, 5–8 March 2014; p. 482.
24. Lin, C.Y.Y.; Lin, H.H. Upper Bounds on Quantum Query Complexity Inspired by the Elitzur-Vaidman Bomb Tester. In Proceedings of the 30th Conference on Computational Complexity (CCC 2015), Portland, OR, USA, 17–19 June 2015; Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2015.
25. Lin, C.Y.Y.; Lin, H.H. Upper Bounds on Quantum Query Complexity Inspired by the Elitzur–Vaidman Bomb Tester. *Theory Comput.* **2016**, *12*, 1–35. [CrossRef]
26. Kapralov, R.; Khadiev, K.; Mokut, J.; Shen, Y.; Yagafarov, M. Fast Classical and Quantum Algorithms for Online k-server Problem on Trees. *CEUR Workshop Proc.* **2022**, *3072*, 287–301.
27. Grover, L.K. A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; ACM: New York, NY, USA, 1996; pp. 212–219.
28. Boyer, M.; Brassard, G.; Høyer, P.; Tapp, A. Tight bounds on quantum searching. *Fortschritte Phys.* **1998**, *46*, 493–505. [CrossRef]
29. Long, G.L. Grover algorithm with zero theoretical failure rate. *Phys. Rev. A* **2001**, *64*, 022307. [CrossRef]
30. Høyer, P.; Neerbek, J.; Shi, Y. Quantum complexities of ordered searching, sorting, and element distinctness. *Algorithmica* **2002**, *34*, 429–448.
31. Odeh, A.; Elleithy, K.; Almasri, M.; Alajlan, A. Sorting N elements using quantum entanglement sets. In Proceedings of the Third International Conference on Innovative Computing Technology (INTECH 2013), London, UK, 29–31 August 2013; pp. 213–216.
32. Odeh, A.; Abdelfattah, E. Quantum sort algorithm based on entanglement qubits $\{00, 11\}$. In Proceedings of the 2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT), Farmingdale, NY, USA, 29–29 April 2016; pp. 1–5.
33. Klauck, H. Quantum time-space tradeoffs for sorting. In Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, San Diego, CA, USA, 9–11 June 2003; ACM: New York, NY, USA, 2003; pp. 69–76.
34. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*; MIT Press: Cambridge, UK, 2001.
35. Williams, J.W.J. Algorithm 232—Heapsort. *Commun. ACM* **1964**, *7*, 347–349.
36. Feige, U.; Raghavan, P.; Peleg, D.; Upfal, E. Computing with noisy information. *SIAM J. Comput.* **1994**, *23*, 1001–1018. [CrossRef]
37. Cormode, G.; Hadjieleftheriou, M. Finding frequent items in data streams. *Proc. Vldb Endow.* **2008**, *1*, 1530–1541. [CrossRef]
38. Muthukrishnan, S. Data streams: Algorithms and applications. *Found. Trends Theor. Comput. Sci.* **2005**, *1*, 117–236. [CrossRef]
39. Aggarwal, C.C. *Data Streams: Models and Algorithms*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2007; Volume 31.
40. Becchetti, L.; Chatzigiannakis, I.; Giannakopoulos, Y. Streaming techniques and data aggregation in networks of tiny artefacts. *Comput. Sci. Rev.* **2011**, *5*, 27–46. [CrossRef]

41. Boyar, J.; Larsen, K.S.; Maiti, A. The frequent items problem in online streaming under various performance measures. *Int. J. Found. Comput. Sci.* **2015**, *26*, 413–439. [CrossRef]
42. De La Briandais, R. File searching using variable length keys. In Proceedings of the Papers Presented at the the Western Joint Computer Conference, San Francisco, CA, USA, 3–5 March 1959; ACM: New York, NY, USA, 1959; pp. 295–298.
43. Black, P.E. Dictionary of Algorithms and Data Structures | NIST. 1998. Available online: http://www.nist.gov/dads (accessed on 12 November 2021).
44. Brass, P. *Advanced Data Structures*; Cambridge University Press: Cambridge, UK, 2008; Volume 193.
45. Knuth, D. *Searching and Sorting, the Art of Computer Programming*; Addison-Wesley: Reading, MA, USA, 1973; Volume 3.
46. Khadiev, K.; Ilikaev, A. Quantum Algorithms for the Most Frequently String Search, Intersection of Two String Sequences and Sorting of Strings Problems. In Proceedings of the International Conference on Theory and Practice of Natural Computing, Kingston, ON, Canada, 9–11 December 2019; pp. 234–245.
47. Mande, N.S.; Thaler, J.; Zhu, S. Improved Approximate Degree Bounds for k-Distinctness. In Proceedings of the 15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020), Riga, Latvia, 9–12 June 2020; Schloss Dagstuhl-Leibniz-Zentrum für Informatik: Dagstuhl, Germany, 2020.
48. Göös, M.; Jayram, T.; Pitassi, T.; Watson, T. Randomized communication vs. partition number. In Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017), Warsaw, Poland, 10–14 July 2017; Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2017.
49. Ambainis, A. Quantum walk algorithm for element distinctness. *SIAM J. Comput.* **2007**, *37*, 210–239. [CrossRef]
50. Ambainis, A. Quantum Walk Algorithm for Element Distinctness. In Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, FOCS '04, Rome, Italy, 17–19 October 2004; pp. 22–31.
51. Aaronson, S.; Shi, Y. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM* **2004**, *51*, 595–605. [CrossRef]
52. Høyer, P.; Mosca, M.; de Wolf, R. Quantum Search on Bounded-Error Inputs. In *Automata, Languages and Programming*; Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 291–299.
53. Brassard, G.; Høyer, P.; Mosca, M.; Tapp, A. Quantum amplitude amplification and estimation. *Contemp. Math.* **2002**, *305*, 53–74.