On the Quantum Time Complexity of Divide and Conquer

Jonathan Allcock
Tencent Quantum Laboratory, Hong Kong, China

Jinge Bao
School of Informatics, University of Edinburgh, UK
Centre for Quantum Technologies, National University of Singapore, Singapore

Aleksandrs Belovs
Center for Quantum Computing Science, Faculty of Science and Technology, University of Latvia, Riga, Latvia

Troy Lee
Centre for Quantum Software and Information, University of Technology Sydney, Ultimo, Australia

Miklos Santha
Centre for Quantum Technologies, National University of Singapore, Singapore
CNRS, IRIF, Paris, France

Abstract

In this work, we initiate a systematic study of the time complexity of quantum divide and conquer (QD&C) algorithms for classical problems, and propose a general framework for their analysis. We establish generic conditions under which search and minimization problems with classical divide and conquer algorithms are amenable to quantum speedup, and apply these theorems to various problems involving strings, integers, and geometric objects. These include Longest Distinct Substring, Klee's Coverage, several optimization problems on stock transactions, and k-Increasing Subsequence. For most of these problems our quantum time upper bounds match the quantum query lower bounds, up to polylogarithmic factors.

We give a structured framework for describing and classifying a wide variety of QD&C algorithms so that quantum speedups can be more easily identified and applied, and prove general statements on QD&C time complexity covering a range of cases, accounting for the time required for all operations. In particular, we explicitly account for memory access operations in the commonly used QRAM (read-only) and QRAG (read-write) models, which are assumed to take unit time in the query model, and which require careful analysis when involved in recursion.

Our generic QD&C theorems have several nice features.

1. To apply them, it suffices to come up with a *classical* divide and conquer algorithm satisfying the conditions of the theorem. The quantization of the algorithm is then completely handled by the theorem. This can make it easier to find applications which admit a quantum speedup, and contrast with dynamic programming algorithms which can be difficult to quantize due to their highly sequential nature.
2. As these theorems give bounds on time complexity, they can be applied to a greater range of problems than those based on query complexity, e.g., where the best-known quantum algorithms require super-linear time.
3. It can handle minimization problems as well as boolean functions, which allows us to improve on the query complexity result of Childs et al. [13] for k-Increasing Subsequence by a logarithmic factor.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Quantum Computing, Quantum Algorithms, Divide and Conquer

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.9

Category Track A: Algorithms, Complexity and Games

© Jonathan Allcock, Jinge Bao, Aleksandrs Belovs, Troy Lee, and Miklos Santha;
licensed under Creative Commons License CC-BY 4.0
52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025).
Editors: Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis
Article No. 9; pp. 9:1–9:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version *Full Version*: https://arxiv.org/abs/2311.16401 [4]

Funding This project was supported by the National Research Foundation, Singapore through the National Quantum Office, hosted in A*STAR, under its Centre for Quantum Technologies Funding Initiative (S24Q2d0009). A.B. is supported by the QuantERA ERA-NET project QOPT. T.L. is supported in part by the ARC DP grant DP200100950. J.B. is supported by the Quantum Advantage Pathfinder (QAP) project of the UKRI Engineering and Physical Sciences Research Council (Grant No. EP/X026167/1), and was funded by a CQT PhD Scholarship when the work was initialized.

1 Introduction

Divide and conquer is a basic algorithmic technique that gained prominence in the 1960s via a series of beautiful and powerful algorithms for fundamental problems including integer multiplication [25], sorting [22], the Fourier transform [14] and matrix multiplication [28]. In fact, the technique of divide and conquer was already used much earlier: Gauss designed the first fast Fourier transform algorithm in 1805, published only after his death [17, 21], and von Neumann invented mergesort in 1945 [19].

Divide and conquer algorithms do not have a generic mathematical description unlike, for example, greedy algorithms. Similarly, there are no known combinatorial structures on which they achieve optimality, unlike greedoids where the greedy solution is optimal for a large class of objective functions [27]. In order to capture the variety of divide and conquer algorithms, it is helpful to keep three examples in mind: quicksort, mergesort, and a divide and conquer algorithm for the Single Stock Single Transaction (SSST) problem, where the goal is to compute $\arg\max_{i<j} A_j - A_i$ for a given array $A \in \mathbb{Z}^n$. We assume that the reader is familiar with quicksort and mergesort. The divide and conquer algorithm for SSST is based on the principle that the indices achieving the maximum are either both contained in the left half of the array, both contained in the right half of the array, cases that can be solved recursively, or one is contained in the left half and one in the right half. The latter case can be directly solved because in this case $A_j - A_i$ is maximized when A_j is the maximum value in the second half of A and A_i is the minimum value in the first half of A.

With these examples in mind, we break down the work in a divide and conquer algorithm into four C's: create, conquer, complete, and combine. In the *create* (or divide) step the algorithm constructs the subproblems to be solved. In mergesort, the construction of subproblems is trivial as this simply involves dividing a string into left and right halves. In quicksort, on the other hand, the create step is where the main work of partitioning the input into elements at most the pivot and at least the pivot takes place. In the *conquer* step the algorithm (typically recursively) solves the subproblems created in the previous step. In the *complete* step the algorithm computes anything that is needed to solve the original problem but not contained in the solution to the subproblems. SSST is a prime example of this step, as here in the complete step one solves the special case of finding the maximum profit when buying in the first half of the array and selling in the second half. Finally in the *combine* step the algorithm integrates the solutions to the subproblems and from the complete step to solve the original problem. A canonical example of this is the merge step of mergesort. Table 1 summarizes this discussion.

Classical divide and conquer algorithms are usually analyzed in the RAM model where accessing a memory register containing an element of the input string takes constant time. Our quantum algorithms will be given in two quantum memory models. In both cases, we assume coherent access to the memory register, that is, the index register can store a superposition of addresses. The first model we consider is QRAM, where the access to data is

Table 1 Breakdown of the steps in a divide and conquer algorithm.

	Description	Example
Create	Create subproblems.	Partition step of quicksort.
Conquer	Solve subproblems.	Recursive call to quicksort.
Complete	Compute additional information not covered by subproblems.	In SSST, the maximum profit from buying in the first half, and selling in the second half.
Combine	Combine subproblem solutions to solve the original problem.	The merge step of mergesort.

read-only, and we further assume that the data stored is classical, i.e., the memory register is not in superposition. The second model we consider is QRAG, where both reading and writing to memory are permitted, and we assume the memory register itself can be in superposition (indeed we require this in some of our algorithms). By a *query* we mean any of these memory operations. While we also will derive query complexity results about our algorithms, our main emphasis will be on time complexity. For a memory of size N, we denote the time required to perform a QRAM or QRAG query by parameters QR_N (quantum reading) and QW_N (quantum writing), respectively. Standard one and two-qubit quantum operations are counted as one time step. The exact definition of these models is given in Section A.

1.1 Our contributions

Table 2 Applications of our divide and conquer technique in the third column, assuming quantum memory access times $\mathrm{QR}_n, \mathrm{QW}_n = O(\log^2 n)$ (see Section A.2.2). For each sublinear quantum upper bound, there is a simple quantum query lower bound that matches up to a polylogarithmic factor.

	Classical Time	Quantum Time
Longest Distinct Substring	$\widetilde{O}(n)$	$\widetilde{O}(n^{2/3})$
Klee's Coverage in $\mathbb{R}^d$	$O(n^{d/2}), d \geq 3$ [12]	$O(n^{d/4+\varepsilon}), d \geq 8$
Single Stock Single Transaction	$O(n)$	$O(\sqrt{n}\log^{5/2}(n))$
k-Increasing Subsequence k-Signed Sum	$O(n \log\log k)$ [15] $\widetilde{O}(n)$	$O(\sqrt{n}\log^{k+1}(n)(\log\log n)^{k-1})$
Maximum 4-Combination	$O(n^3)$ [8]	$O(n^{3/2}\log^{5/2}(n))$

In this work, we focus on divide and conquer algorithms where the combine step is either the OR function or minimization/maximization, meaning that it is amenable to a quantum speedup by Grover search or quantum minimum/maximum finding. When the complete step is also relatively simple, we can show a generic theorem that transforms a classical divide and conquer algorithm into a quantum one and bounds its time complexity. We show two versions of this theorem depending on the nature of the create step.

Our generic quantum divide and conquer theorems have several nice features. The first is that, to apply them, it suffices to come up with a *classical* divide and conquer algorithm satisfying the conditions of the theorem. The quantization of this algorithm is then completely handled by the theorem. This can make it easier to find applications of these theorems. The second is that these theorems give bounds on *time complexity*, not just the query complexity. To the best of our knowledge, the quantum time complexity of the divide and conquer method

has not been systematically studied before. Working with time complexity also lets us apply these theorems to a greater range of problems, e.g. those where the best-known classical algorithms require super-linear time.

As divide and conquer algorithms are typically recursive, our theorems must handle the error probability corresponding to the composition of a super-constant number of bounded-error algorithms. While the composition of bounded-error quantum algorithms is now well understood in the quantum query complexity setting, much less work has focused on this in the time complexity setting [9]. We show several basic results about the time complexity of the composition of quantum search or minimum/maximum finding over bounded-error subroutines. In most cases, these results are relatively easy adaptations of analogous algorithms in the query setting. However, we feel these are fairly fundamental algorithmic primitives that will find further application in the future.

The statement of our quantum divide and conquer theorems is rather technical and we defer a discussion to section 1.2, and give further details in Section B and Section C. In the rest of this section, we describe applications of our generic theorems to different problems. A summary of the major applications can be found in Table 2, where we assume that the cost of the two memory access gates is $O(\log^2 n)$, as per some proposals for implementations of quantum memory (see Section A.2.2).

Some problems have a simple create step, by which we mean that the locations of the subproblems are identical for all inputs of the same size. The quantum algorithm we give for Single Stock Single Transaction turns out to be quite paradigmatic, and we are able to use a very similar approach for the following three problems. In the Longest Increasing Substring problem (LISst), we have to find the longest increasing substring in an array of integers. In the Longest Identical Substring problem (LSIC), we are looking for the longest substring of identical characters in a string over some finite alphabet. Finally, in the Longest 20^*2 Substring problem (L20^*2S) we have to find the longest substring belonging to 20^*2 in a string from $\{0,1,2\}^*$. All these problems can be solved by classical divide and conquer along the lines of the algorithm we sketched for Single Stock Single Transaction, in time $O(n \log n)$, and they require time $\Omega(n)$. Our algorithms achieve an almost quadratic quantum speed up. For the rest of the paper, for $k \geq 1$, we define the function

$$\lambda_k(n,m) = \min\{\log^k n + m, \log^{(k+1)/2}(n)(\log\log n)^{k-1} + \log^{(k-1)/2}(n)(\log\log n)^{k-1} \cdot m\}.$$

Observe that for $k = 1$ we have $\lambda_1(n, \mathrm{QR}_n) = \log n + \mathrm{QR}_n$, and for $k = 2$ we have $\lambda_2(n, \mathrm{QR}_n) = \min\{\log^2 n + \mathrm{QR}_n, \log^{3/2}(n)\log\log n + \sqrt{\log n}\log\log n \cdot \mathrm{QR}_n\}$.

▶ **Theorem 1.** *The quantum query and time complexities of the problems SSST, LISst, LSIC and L20^*2S are respectively $O(\sqrt{n\log n})$ and $O(\sqrt{n\log n} \cdot \lambda_2(n, \mathrm{QR}_n))$.*

One generalization of the stock problem is d-Multiple Stocks Single Transaction, for $d \geq 1$, where we want to find $\max_{i<j} A_j - A_i$ in a d-dimensional array A. Here, by definition, $i < j$ when $i_k < j_k$, for $1 \leq k \leq d$. Our quantum algorithm easily generalizes to that problem.

▶ **Theorem 2.** *For every $d \geq 1$, the quantum query complexity of d-MSST is $O((n\log n)^{d/2})$ and its time complexity is*

$$O\left((n\log n)^{d/2} \cdot \min\{\log^{d+1} n + \mathrm{QR}_{n^d}, \log^{1+d/2}(n)\log\log n + \log^{d/2}(n)\log\log n \cdot \mathrm{QR}_{n^d}\}\right).$$

In the k-Increasing Subsequence problem, we would like to decide if, in an array of integers, there is a subsequence of k increasing numbers. As discussed in [13], this is the natural parametrization of the classically well-studied Longest Increasing Subsequence problem, closely related to patience sorting. There are $O(n \log n)$ query classical dynamic programming algorithms solving Longest Increasing Subsequence, and it is easy to show an $\Omega(n)$ quantum query lower bound for it. This implies that no substantial quantum improvement can be obtained for k-Increasing Subsequence when k is unbounded, and makes the case of constant k an interesting research question. In [13] an $O(\sqrt{n \log^{3(k-1)} n})$ quantum query algorithm was obtained for k-Increasing Subsequence, and we improve this result by a factor of $O(\log^{k-1} n)$ and implement it time-efficiently. It turns out that a very similar quantum algorithm can solve the k-Signed Sum problem with the same complexity. In this problem, given an array of integers and a sign pattern $\varepsilon \in \{-1, 1\}^k$, we want to maximize the signed sum $\sum_{m=1}^{k} \varepsilon_m A_{i_m}$, for k indices satisfying $i_1 < i_2 < \ldots < i_k$.

▶ **Theorem 3.** *There are quantum algorithms that solve k-Increasing Subsequence and k-Signed Sum with $O(\sqrt{n \log^{k-1} n})$ queries. The time complexity of the algorithms is $O(\sqrt{n \log^{k-1} n} \cdot \lambda_k(n, \mathrm{QR}_n))$.*

The problems above have simple create steps. We also consider problems with simple complete steps, and where the main work lies instead in the create step. One interesting example of this is the Longest Distinct Substring (LDS) problem. Given a string $a \in \Sigma^n$ over an alphabet Σ, this problem is to find the longest contiguous substring $a_i a_{i+1} \cdots a_j$ where all letters are unique. The famous Element Distinctness problem is a special case of Longest Distinct Substring where the task is to determine if the length of a longest distinct substring is equal to the length of a itself. Ambainis [5] famously gave a quantum walk algorithm showing the time complexity of element distinctness is $\tilde{O}(n^{2/3})$, which is tight [2]. We apply our divide and conquer theorem, (Theorem 8), in conjunction with a novel classical divide and conquer algorithm for LDS, to show that a quantum algorithm can solve LDS in time $\widetilde{O}(n^{2/3} \cdot \mathrm{QW}_{O(n)})$. Thus, up to logarithmic factors, finding the longest distinct substring has the same quantum time complexity as element distinctness.

▶ **Theorem 4.** *There is a quantum algorithm that solves LDS in time $\widetilde{O}(n^{2/3}) \cdot \mathrm{QW}_{O(n)}$ and a quantum algorithm that solves LDS with $O(n^{2/3} \log(n) \log\log(n))$ queries.*

Another example with a simple complete step is the Klee's Measure problem from computational geometry, which asks to compute the volume of the union of axis-parallel hyperrectangles in d-dimensional real space. In the special case of the Klee's Coverage problem, the question is to decide if the union of the hyperrectangles covers a given base hyperrectangle. In 2-dimensions, the classical complexity of the Klee's Measure problem is $O(n \log n)$ [26], and for any constant $d \geq 3$, Chan [12] has designed an $O(n^{d/2})$ time classical algorithm for it. We give a quantum algorithm for Klee's Coverage that achieves almost quadratic speedup over the classical divide and conquer algorithm of Chan [12], when $d \geq 8$.

▶ **Theorem 5.** *For every constant $\varepsilon > 0$, the quantum time complexity of the Klee's Coverage problem is $O(n^{d/4+\varepsilon} \cdot \mathrm{QW}_{N(n)})$ when $d \geq 8$, and is $O(n^2 \cdot \mathrm{QW}_{N(n)})$ for $5 \leq d \leq 7$, where $N(n) = O(n^{d/2+\varepsilon})$.*

Perhaps not surprisingly, our results have some consequences for fine-grained complexity, in particular for the class **APSP** of problems that are solvable in time $\widetilde{O}(n^3)$ on a classical computer and are sub-n^3 equivalent to the All-pairs Shortest Paths problem in the

sense that either all of them or none of them admit an $O(n^{3-\varepsilon})$ algorithm, for some constant $\varepsilon > 0$. **APSP** is one of the richest classes in fine-grained complexity theory [32, 31] and, in particular, contains various path, matrix, and triangle problems.

Quantum fine-grained complexity is a relatively new research area [1, 11, 10] where one possible direction is to study the quantum complexity of problems in the same classical fine-grained equivalence class. Indeed, the work [6] specifically considered **APSP**. Of course, there is no guarantee that classically equivalent problems remain equivalent in the quantum model of computing, and this is indeed the case for **APSP**. All problems in the class receive some quantum speedup, but the degree of speedup can differ from problem to problem. It turns out that many of the problems in **APSP** can be solved either in time $\widetilde{O}(n^{5/2})$ or in time $\widetilde{O}(n^{3/2})$ by simple quantum algorithms and, concretely, All-pairs Shortest Paths falls in the former category. We consider the quantum complexity of two problems from the class **APSP**: Maximum 4-Combination and Maximum Submatrix, both of which take an $n \times n$ matrix B as input. We want to compute, for the former, the maximum of $B_{ik} + B_{j\ell} - B_{i\ell} - B_{jk}$ and, for the latter, the maximum of $\sum_{i \le u \le j, k \le v \le \ell} B_{uv}$, under the conditions $1 \le i \le j \le n$ and $1 \le k \le \ell \le n$. Our quantum algorithm for Maximum 4-Combination uses the quantum divide and conquer method designed for Single Stock Single Transaction.

▶ **Theorem 6.** *The quantum time complexity of* Maximum Submatrix *is* $O(n^2 \log n)$ *and its query complexity is* $\Omega(n^2)$. *The quantum time complexity of* Maximum 4-Combination *is* $O(n^{3/2} \log^{5/2}(n))$.

1.2 Our techniques

As mentioned, our techniques all suppose that the combine step of the divide and conquer is search or minimization and, moreover, the complete or the create steps (or both) are relatively simple. We now describe in more detail how we are able to exploit these properties.

Search or minimization combine step. The key technical tool we will use repeatedly is based on a relatively old result of Høyer, Mosca and de Wolf [23] (see Fact 1 in Section A). Their result is stated for query complexity and essentially says that if, for J boolean functions $f_1, \ldots, f_J : \{0,1\}^n \to \{0,1\}$ there exists a quantum algorithm F that on input $|j\rangle|x\rangle$ correctly computes $f_j(x)$ with probability at least 8/10, then there exists a quantum algorithm which uses $O(\sqrt{J})$ repetitions of F and with probability at least 9/10 finds a marked index, that is $1 \le j \le J$ such that $f_j(x) = 1$, if there is one. The main point here is that the number of repetitions is of the same order of magnitude as one would need when F does the computation without error. We analyze the time complexity of the above algorithm and show in Corollary 9 that it is $O(\sqrt{J}(\log J + \tau))$, where τ is the time complexity of F. A similar result was also known for the query complexity of finding the index of a minimum element (see Fact 2), and we obtain the analogous result for the time complexity in Corollary 10. In the remaining part of the section, while we only speak about minimization algorithms, everything is valid for search problems as well.

What can we say about the time complexity of F if, for every $1 \le j \le J$, we have at our disposal an algorithm A_j of known complexity computing $f_j(x)$? Let us call these *base* algorithms, and for the simplicity of discussion let us suppose that the time complexity of every base algorithm is $S + q \cdot \mathrm{QR}_n$, where S is the total number of one and two-qubit gates and J, S, q depend on n. If there is no particular relation between the J base algorithms (that is, they can be very different), we do not have a particularly clever implementation of F. One possibility is to compose the quantum circuits to compute them, where the non-query

gates of A_j are controlled by j. The query gates can be executed without control, giving an overall complexity of F of $O(JS + q \cdot \mathrm{QR}_n)$, and yielding a minimization algorithm of complexity $O(\sqrt{J}(JS + q \cdot \mathrm{QR}_n))$. Another trivial way to solve the minimization problem is to reduce the error of each base algorithm via $\log J$ repetitions and then classically compute the minimum. The complexities of these approaches are stated in Lemma 11.

However, there is one situation where we can do much better, and this is exactly the case of recursive algorithms such as divide and conquer. In this case, the base algorithms correspond to the recursive calls and are therefore the same by definition. Thus, for implementing F there is no need to use controlled operations or to repeat the base algorithms, provided that we are able to determine, for every j, the memory indices where the QRAM gates for A_j are executed (the "index problem"). There are two broad cases we consider, corresponding to simple create and simple complete steps:

Simple create step. In some cases, such as in the Single Stock Single Transaction problem, the create step is independent of the input. Suppose that the input length is a power of 2. If we unfold the successive recursive steps for this problem, then it is easy to see that, for every input A of length n, there exists $t \in [\log n]$, such that if we partition $[n]$ into 2^t consecutive intervals of length $n/2^t$ then one of these intervals C contains both i and j, with $i < j$, such that $A_j - A_i$ is an optimal solution. Moreover, i is in the left half C_ℓ, and j is in the right half C_r of C, and therefore under this assumption they can be found easily in time $\sqrt{n/2^t}(\log n + \mathrm{QR}_n)$.

Now the main point is that, for a given t, we don't know which interval C contains the solution, but to find this good interval we can use quantum maximum finding with an erroneous oracle. These 2^t intervals are consecutive and therefore the elements in all intervals can be indexed uniformly, once we settle the first t bits specific to each interval. For every t, this results in a uniform cost $\sqrt{n}(\log n + \mathrm{QR}_n)$. After this, we still have to search over the $\log n$ possible values of t, again using Corollary 10. This time there are additional costs for implementing F, captured in Lemma 11, because the above procedures are quite different for different values of t. However, as the search is only over a domain of logarithmic size the additional cost is not substantial.

We call the resulting divide and conquer method *bottom-up* since the method actually completely eliminates the recursive calls, and we give a general statement about this approach in Theorem 12. Avoiding recursion is possible because we know the subproblems in advance, as they are independent of the input. Moreover, under the condition that the solution is contained in some interval C, in our case the additional information that i is in C_ℓ and j is in C_r makes the solution easier. This, of course, isn't always the case. Consider the element distinctness problem on an interval of size n. The additional knowledge that the two colliding elements are in different halves of the interval does not make the task of finding them easier. But it does help for Single Stock Single Transaction as well as for Longest Increasing Substring, Longest Identical Substring, Longest 20*2 Substring and d-Multiple Stocks Single Transaction. In fact, to some extent, we can even generalize the method to k-ary relations, for $k > 2$, which is illustrated by our algorithms for k-Increasing Subsequence and k-Signed Sum.

Simple complete step. In this case, the main work of the algorithm is done in the create step. We give two relatively generic theorems describing situations where the index problem can be handled well.

The first situation concerns *constructible instance* problems, where the recursive calls are executed on subproblems that are explicitly constructed by the create step. Such a create step may be expensive to perform and, in particular, all the inputs of the recursive calls (which we call *constitutive strings*, and which are not necessarily substrings or subsequences of the input) have to be written down, likely taking at least linear time. This implies that this approach can only yield nontrivial results for time complexity, but not for query complexity.

Constructible instance problems are specified by two functions $h : \mathbb{N} \to \mathbb{N}$, $m : \mathbb{N} \to \mathbb{R}_+$. At each level of recursion, from the input a of length n, the $h(n)$ constitutive strings, $a^{(1)}, \dots a^{h(n)}$, each of length $n/m(n)$, are constructed. Then, the solution $P(a)$ has a particularly simple form. For disjunctive problems (when the combine step is the OR function), $P(a) = \mathrm{OR}_{j \in h(n)} \gamma(a, j, P(a^{(j)})$, where $\gamma : \Sigma^n \times [h(n)] \times \{0,1\} \to \{0,1\}$ is a function used in the complete step, and depends on the input a, the constitutive string number, and the solution for the subproblem $a^{(j)}$. Minimizing problems (when the combine step is minimization) are defined analogously.

We can suppose that the constitutive strings are written in a way that the indices of the memory used by the J functions calls A_j, for $j \in J$, differ only in the first $\log J$ bits where, for every j, the algorithm A_j has j in binary.

▶ **Theorem 7** (Informal). *Let P be a constructible instance (h, m)-disjunctive (respectively (h, m)-minimizing) problem. Suppose that*

- *$h(n)$ and $m(n)$ can be computed by classical algorithms in time $O(V(n))$;*
- *there is a classical algorithm that computes the constitutive strings $a^{(1)}, \dots, a^{(h(n))}$ in time $O(S(n))$;*
- *the computation of $\gamma(a, j, P(a^{(j)}))$ can be done by a quantum algorithm in time $O(G(n) \cdot \mathrm{QW}_N)$ and with error at most $1/10$.*

Then, there exists a quantum divide and conquer algorithm $\mathcal{A}$ which on input α, computes $P(\alpha)$ with probability at least $9/10$ and has a running time $\overline{T}$ that satisfies the recurrence relation

$$\overline{T}(n) \leq c\sqrt{h(n)}\left(\overline{T}(n/m(n)) + G(n) \cdot \mathrm{QW}_N + \log h(n)\right) + O\left(S(n) \cdot \mathrm{QW}_N + V(n)\right),$$

for some constant $c > 0$, where N is the total amount of quantum memory required, and which can be calculated by a recursive equation.

An application of this result is the quantum algorithm for Klee's Coverage where the create step requires quadratic time. However, in high dimensions, the time of the homogeneous recursion, which doesn't include the create step, is much larger and therefore we are able to achieve an almost quadratic speed up over the classical algorithm.

The second situation concerns *t-decomposable instance* problems, where the recursive calls are made on subsequences of the input. As for constructible instance problems, each recursive call involves determining $h(n)$ constitutive strings, each of length $n/m(n)$. However, in this case, the create step does not need to create the constitutive strings. Rather, the constitutive strings are required to be the concatenation of t substrings of the original input string α, where t is a constant. Then, the work during the create step is to determine the indices that delimit the constitutive strings in the input. We call this set of indices a t-description of the constitutive string and, in principle, computing this can take much less than linear time.

▶ **Theorem 8** (Informal). *Let P be a t-decomposable instance (h, m)-disjunctive (respectively (h, m)-minimizing) problem. Let the input α to the problem have length n^*. Suppose that*

- *$h(n)$ and $m(n)$ can be computed by classical algorithms in time $O(V(n))$;*

- *there is a quantum algorithm that computes the t-description of constitutive strings in time $O(D(n) \cdot \mathrm{QW}_{n^*})$, and with probability of error at most $1/10$;*
- *the completion function γ can be done by a quantum algorithm in time $O(G(n) \cdot \mathrm{QW}_{n^*})$, with error at most $1/10$.*

Then, in the QRAM *model, there exists a quantum divide and conquer algorithm $\mathcal{A}$ which computes the problem P with probability at least $8/10$, and has running time given by the recurrence relation*

$$\overline{T}(n^*, n) \leq c\sqrt{h(n)}\left(\overline{T}(n^*, n/m(n)) + G(n) \cdot \mathrm{QR}_{n^*} + \log h(n)\right) + O\left(D(n) \cdot \mathrm{QR}_{n^*} + V(n)\right).$$

An analogous result holds in the QRAG *model.*

As an application of this, by identifying a problem, that we call Bipartite Longest Distinct Substring, as a t-decomposable instance problem whose create and complete functions can be computed in time $\widetilde{O}(n^{2/3}) \cdot \mathrm{QW}_{O(n)}$, we are able to give a quantum algorithm for solving the Longest Distinct Substring problem in time $\widetilde{O}(n^{2/3}) \cdot \mathrm{QW}_{O(n)}$.

1.3 Previous work

Recently, several quantum divide and conquer algorithms were presented for various string problems. In the first of these papers, Akmal and Jin [3] considered the k-Length Minimal Substring problem for $k \geq n/2$, where given a string a of length n over some finite alphabet with a total order, the output is a substring v of a of length k such that for every substring w of a of length k, we have $v \leq w$, for lexicographic ordering among strings. In the decision version of the problem, the input also includes a string v of length k and the question is whether v is lexicographically smallest among the k-length substrings of a.

The algorithm in [3] works in time $\sqrt{n}2^{O(\log^{2/3} n)}$, and the high-level structure of the proof goes along the lines of our Theorem 8 for t-decomposable instance minimizing problems, but without involving our Corollary 10 to deal efficiently with the errors of the recursive calls. The combinatorial contents of the proof, however, are quite different. In [3] it is also shown that the problems Minimal String Rotation and Minimal Suffix are easily reducible to k-Length Minimal Substring and therefore have the same quantum time complexity upper bound. In the former problem, one is looking for $j \in [n]$ such that $a[j:n]a[1:j-1] \leq a[i:n]a[1:i-1]$, for all $i \in [n]$, while in the latter problem one looks for $j \in [n]$ such that $a[j:n] < a[i:n]$, for all $i \neq j$. The decision versions of these problems, when j is also given in the input, are defined naturally.

In the query complexity model, two papers improved on these results. Childs et al. in [13] showed that the decision versions of the above problems can be solved in $O(\sqrt{n}\log^{5/2} n)$ queries. For Minimal String Rotation Wang [29] improved this to $\sqrt{n}2^{O(\sqrt{\log n})}$, and for the decision version of the problem further improved this to $O(\sqrt{n}\log^{3/2} n\sqrt{\log\log n})$. Interestingly, Wang uses Fact 2 on bounded-error oracles in his argument, but the paper does not deal with time complexity.

The result of Childs et al. above was part of a powerful and general framework in the query complexity model for quantum divide and conquer algorithms computing boolean functions. Their framework includes, for example, the case where the complete step is computed by an AND-OR formula in the recursive calls and some auxiliary boolean functions. Besides the problems from the previous paragraph, it is also applied to give algorithms for k-Increasing Subsequence and for k-Common Subsequence where one has to decide whether two strings share a common subsequence of length k. This framework makes use of

specific properties of the query model (for example, the equality of the query complexity and the adversary bound), and it is unlikely that it can directly yield results for time complexity. With respect to this framework ours has the following advantages:

- It deals with time complexity and not only with query complexity.
- As a consequence, it can also deal with problems of super-linear complexity.
- It can handle minimization problems and not just boolean functions.

Finally we mention that for the k-Increasing Subsequence problem we improve their query complexity result of $O(\sqrt{n}\log^{3(k-1)/2} n)$ to $O(\sqrt{n}\log^{(k-1)/2} n)$.

In independent work, Jeffery and Pass also give a time-efficient quantum algorithm for divide and conquer [24, Theorem 4.13]. Their theorem is incomparable with our results. They allow a symmetric Boolean formula as the combination function, while we consider the combination function to be OR or maximization. Another difference surrounds the flexibility of the create step. They require the create function to work in unit time. Our theorems build the complexity of the create function into the recurrence for the running time, and we allow the create function to either explicitly construct the input to the subproblems (Theorem 7) or reference the input to the subproblems via the original input (Theorem 8).

References

1. Scott Aaronson, Nai-Hui Chia, Han-Hsuan Lin, Chunhao Wang, and Ruizhe Zhang. On the quantum complexity of closest pair and related problems. In *Proceedings of the 35th Computational Complexity Conference (CCC)*, pages 16:1–16:43, 2020. doi:10.4230/LIPICS.CCC.2020.16.
2. Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM (JACM)*, 51(4):595–605, 2004. doi:10.1145/1008731.1008735.
3. Shyan Akmal and Ce Jin. Near-optimal quantum algorithms for string problems. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 2791–2832. SIAM, 2022. doi:10.1137/1.9781611977073.109.
4. Jonathan Allcock, Jinge Bao, Aleksandrs Belovs, Troy Lee, and Miklos Santha. On the quantum time complexity of divide and conquer. *arXiv preprint arXiv:2311.16401*, 2023. doi:10.48550/arXiv.2311.16401.
5. Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. doi:10.1137/S0097539705447311.
6. Andris Ambainis, Harry Buhrman, Koen Leijnse, Subhasree Patro, and Florian Speelman. Matching triangles and triangle collection: Hardness based on a weak quantum conjecture. *arXiv preprint arXiv:2207.11068*, 2022. doi:10.48550/arXiv.2207.11068.
7. Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. On the robustness of bucket brigade quantum RAM. In *Proceedings of the 10th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC)*, pages 226–244, 2015. doi:10.4230/LIPICS.TQC.2015.226.
8. Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. Tight hardness results for maximum weight rectangles. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 81:1–81:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.81.
9. Aleksandrs Belovs, Stacey Jeffery, and Duyal Yolcu. Taming quantum time complexity. *Quantum*, 8:1444, 2024. doi:10.22331/Q-2024-08-23-1444.
10. Harry Buhrman, Bruno Loff, Subhasree Patro, and Florian Speelman. Limits of quantum speed-ups for computational geometry and other problems: Fine-grained complexity via quantum walks. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 31:1–31:12, 2022. doi:10.4230/LIPICS.ITCS.2022.31.

11 Harry Buhrman, Subhasree Patro, and Florian Speelman. A framework of quantum strong exponential-time hypotheses. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 19:1–19:19, 2021. doi:10.4230/LIPICS.STACS.2021.19.

12 Timothy M. Chan. Klee's measure problem made easy. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 410–419, 2013. doi:10.1109/FOCS.2013.51.

13 Andrew Childs, Robin Kothari, Matt Kovacs-Deak, Aarthi Sundaram, and Daochen Wang. Quantum divide and conquer. *ACM Transactions on Quantum Computing*, 6(2):1–26, 2025.

14 James Cooley and John Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

15 Maxime Crochemore and Ely Porat. Computing a longest increasing subsequence of length k in time $O(n \log \log k)$. In Erol Gelenbe, Samson Abramsky, and Vladimiro Sassone, editors, *Visions of Computer Science - BCS International Academic Conference, Imperial College, London, UK, 22-24 September 2008*, pages 69–74. British Computer Society, 2008. URL: http://www.bcs.org/server.php?show=ConWebDoc.22851.

16 Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *arXiv preprint quant-ph/9607014*, 1996.

17 Carl Friedrich Gauss. Theoria interpolationis methodo nova tractata. *Carl Friedrich Gauss Werke, Königlichen Gesellschaft der Wissenschaften: Göttingen*, 3:265–327, 1876.

18 Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Architectures for a quantum random access memory. *Physical Review A*, 78(5), 2008. doi:10.1103/physreva.78.052310.

19 Herman Goldstone and John von Neumann. Planning and coding of problems for an electronic computing instrument. *IAS Princeton*, 2(2):49–68, 1947.

20 Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th annual ACM symposium on Theory of computing (STOC)*, pages 212–219, 1996. doi:10.1145/237814.237866.

21 Michael Heideman, Don Johnson, and Sidney Burrus. Gauss and the history of the fast fourier transform. *IEEE ASSP Magazine*, 1(4):14–21, 1984. doi:10.1109/MASSP.1984.1162257.

22 Tony Hoare. Quicksort. *Comput. J.*, 5(1):10–15, 1962. doi:10.1093/comjnl/5.1.10.

23 Peter Høyer, Michele Mosca, and Ronald de Wolf. Quantum search on bounded-error inputs. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 291–299, 2003. doi:10.1007/3-540-45061-0_25.

24 Stacey Jeffery and Galina Pass. Multidimensional quantum walks, recursion, and quantum divide & conquer. *CoRR*, 2023. arXiv:2401.08355.

25 Anatoly Karatsuba and Yuri Ofman. Multiplication of many-digital numbers by automatic computers (in russian). *Doklady Akademii Nauk SSSR*, 145(2):293–294, 1962.

26 Victor Klee. Can the measure of $\bigcup_1^n [a_i, b_i]$ be computed in less than $O(n \log n)$ steps? *American Mathematical Monthly*, 84(4):284–285, 1977.

27 Bernhard Korte and László Lovász. Greedoids - a structural framework for the greedy algorithm. In WILLIAM R. PULLEYBLANK, editor, *Progress in Combinatorial Optimization*, pages 221–243. Academic Press, 1984. doi:10.1016/B978-0-12-566780-7.50019-2.

28 Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. doi:10.1007/BF02165411.

29 Qisheng Wang. A note on quantum divide and conquer for minimal string rotation. *Theoretical Computer Science*, page 115120, 2025. doi:10.1016/J.TCS.2025.115120.

30 Qisheng Wang and Mingsheng Ying. Quantum algorithm for lexicographically minimal string rotation. *Theory of Computing Systems*, 68(1):29–74, 2024. doi:10.1007/S00224-023-10146-8.

31 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians (ICM)*, pages 3447–3487, 2019.

32 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

Additional details and some proofs of results stated in the main text can be found here. Omitted proofs and details of all other claims can be found in the full version of the manuscript.

A Preliminaries

A.1 Notation

We denote by $\widetilde{O}(f(n))$ the family of functions of the form $f(n)\,\mathrm{polylog}(n)$. For a positive integer n, we denote by $[n]$ the set $\{1,\ldots,n\}$. Let Σ be a finite set and $a = a_1 \ldots a_n \in \Sigma^n$. We define the *length* of a as n, and we denote it by $|a|$. We call $a_{i_1}a_{i_2}\ldots a_{i_j}$ a *subsequence* of a, for any $1 \leq i_1 < i_2 < \ldots < i_j \leq n$. A *substring* is a subsequence of consecutive symbols, and for $1 \leq i \leq j \leq n$, we use $a[i:j]$ to denote the substring $a_i a_{i+1} \ldots a_j$ of a. If $i > j$ then $a[i:j]$ is the empty string. For two strings $a, b \in \Sigma^*$ we denote the concatenation of a and b by $a \mathbin{+\!\!+} b$. For $n^* \in \mathbb{N}$ we define $\Sigma^{\leq n^*} = \bigcup_{n \leq n^*} \Sigma^n$. When $\Sigma \subseteq \mathbb{Z}$, we use capital letters A, B, C... for elements of Σ^n or $\Sigma^{n\times n}$, that is, for arrays and matrices. For ease of notation, we often neglect taking floors and ceilings in our computations, however, this never affects the correctness of the asymptotic results.

A.2 Quantum computational models

We use the standard quantum circuit model of all single qubit gates and the two-qubit CNOT gate, augmented with random access to quantum memory.

A.2.1 QRAM and QRAG

We will use two quantum memory models. In the more restricted QRAM model the memory can only be accessed for reading. For any positive integer N, we define the QRAM_N gate as

$$\mathsf{QRAM}_N|i, e, x\rangle = |i, e \oplus x_i, x\rangle,$$

where $i \in [N], e, x_1, \ldots x_N \in \{0,1\}^r$. We refer to the three registers involved in a QRAM_N gate as the memory *index*, the memory *output*, and the memory *content* registers. The memory index and output registers can be in superposition, but the memory content register always only contains the input string. The memory content register can only be accessed via the QRAM_N gates and these gates can only applied to the memory registers. In the more general QRAG model the memory content register can also be accessed for writing.

To formalize this is, for any positive integer N, we define the QRAG_N gate by

$$\mathsf{QRAG}_N|i, e, x\rangle = |i, x_i, x_1, \ldots, x_{i-1}, e, x_{i+1} \ldots, x_N\rangle,$$

where $i \in [N]$ and $e, x_1, \ldots, x_N \in \{0,1\}^r$. Similar to the other QRAM case, here the memory content register can only be accessed via QRAG_N gates and these gates can only applied to the memory registers. Since writing to memory is permitted, the memory content register can also be in quantum superposition.

In our running time analyses, single and two-qubit gates count as one time step. We will use the parameter QR_N ("quantum read") to denote the cost of a QRAM_N gate, and the parameter QW_N ("quantum write") to denote the cost of a QRAG_N gate. These models are also adapted to query complexity analyses and we define a *query* as one application of the QRAM_N or the QRAG_N gate, in the respective model of computation.

A.2.2 Inputs vs. Instances

The parameterization by N of the memory access times QR_N and QW_N allows for quantum memories of different sizes to be accessed at different rates. We assume that the size N of the memory is fixed during the running of any algorithm, and thus the cost of the memory accesses will remain constant during the algorithm, even when recursive calls are made on shorter and shorter strings. We therefore make a distinction between an *instance*, which is any string on which the algorithm might be recursively called, and an *input*, which is the initial string the algorithm is given. We denote the size of the input by n^*. The size of an instance can be any integer $n \leq n^*$.

Consequently, the memory content register will be a sufficiently large function $N(n^*) \geq n^*$ of the input size, depending on the problem P we are solving. In the QRAM model we choose $N(n^*) = n^*$ and suppose that, at the beginning of the computation, the input $a \in \Sigma^{n^*}$ is in the memory content register. In the QRAG model, the size $N(n^*)$ of the memory content register can be strictly larger than n^* and we suppose that, at the beginning of the computation, it contains $a\mathbf{0}^{N(n^*)-n^*}$, where $\mathbf{0} = 0^r$.

Thus, for any problem with input of size n^*, the cost of quantum read and write operations will be $\mathrm{QR}_{N(n^*)}$ and $\mathrm{QW}_{N(n^*)}$, respectively. According to some proposals [18, 7], $\mathrm{QR}_{N(n^*)}$ and $\mathrm{QW}_{N(n^*)}$ might scale as $O(r \log N(n^*))$, where r is the number of bits each memory cell can store. In that case, when $N(n^*)$ is a polynomial function of n^*, the cost of the memory access operations would be $O(r \log(n^*))$. For both QRAM and QRAG models, we assume that basic arithmetic operations and comparisons can be performed in the same time as memory access.

For notational simplicity, we will quote all final algorithmic running times in our theorems in terms of n rather than n^*, i.e., we take the *input* size to be n in our final results.

A.3 Quantum algorithms

We state here several basic quantum algorithms we will use in the paper. All, except the last result on element distinctness, are in the QRAM model. The first two are generalizations of Grover's search algorithm [20] and quantum minimum finding [16] to cases where the oracle can be erroneous.

▶ **Fact 1** (Quantum query search with an erroneous oracle, [23]). *For $j \in [J]$, let $f_j : \Sigma^n \to \{0,1\}$ be a boolean function. Let F be a quantum algorithm that for every $(j,x) \in [J] \times \Sigma^n$, when $|x\rangle$ is given in the quantum memory content register, with q queries computes $f_j(x)$ with probability at least $7/10$. Then there exists a quantum algorithm that uses $O(\sqrt{J})$ repetitions of F and with probability at least $9/10$ finds an index $j \in [J]$ such that $f_j(x) = 1$, if there is one. The query complexity of the algorithm is $O(\sqrt{J}q)$.*

▶ **Fact 2** (Quantum query minimum finding with erroneous oracle, Lemma 3.4 in[30]). *For $j \in [J]$, let $f_j : \Sigma^n \to \mathbb{Z}$ be a function. Let F be a quantum algorithm that for every $(j,x) \in [J] \times \Sigma^n$, when $|x\rangle$ is given in the quantum memory content register, with q queries computes $f_j(x)$ with probability at least $7/10$. Then there exists a quantum algorithm which uses $O(\sqrt{J})$ repetitions of F and with probability at least $9/10$ finds the index $j \in [J]$ of a minimal element in $\{f_1(x), \ldots, f_J(x)\}$. The query complexity of the algorithm is $O(\sqrt{J}q)$.*

In the following corollaries, we extend the above results for time complexity. These will be our main tool for the analysis of quantum divide and conquer algorithms since the results of the recursive calls might also be erroneous. Proofs are given in the full version of the manuscript.

▶ **Corollary 9** (Quantum search with an erroneous oracle). *For $j \in [J]$, let $f_j : \Sigma^n \to \{0,1\}$ be a boolean function. Let F be a quantum algorithm that for every $(j,x) \in [J] \times \Sigma^n$, when $|x\rangle$ is given in the quantum memory content register, in time τ computes $f_j(x)$ with probability at least $7/10$. Then there exists a quantum algorithm which in time $O(\sqrt{J}(\log J + \tau))$ with probability at least $9/10$ finds an index $j \in [J]$ such that $f_j(x) = 1$, if there is one.*

▶ **Corollary 10** (Quantum minimum finding with an erroneous oracle). *For $j \in [J]$, let $f_j : \Sigma^n \to \mathbb{Z}$ be a function. Let F be a quantum algorithm that for every $(j,x) \in [J] \times \Sigma^n$, when $|x\rangle$ is given in the quantum memory content register, in time τ computes $f_j(x)$ with probability at least $7/10$. Then there exists a quantum algorithm which in time $O(\sqrt{J}(\log J + \tau))$ with probability at least $9/10$ finds the index $j \in [J]$ of a minimal element in $\{f_1(x), \ldots, f_J(x)\}$.*

In the applications of Corollary 9 and Corollary 10 it is important to have good bounds on the complexity of F. The following Lemma that we state in the QRAM model describes two simple but very generic algorithms for computing F that can be used without any assumption on the functions f_j. A similar lemma can be proven in the QRAG model.

▶ **Lemma 11.** *Let $J : \mathbb{N} \to \mathbb{N}$, and for all $j \in [J(n)]$, let $f_j : \Sigma^n \to \{0,1\}$, for all $j \in [J(n)]$, let $f_j : \Sigma^n \to \mathbb{Z}$. Suppose that for every $j \in [J(n)]$, $f_j(x)$ can be computed with probability at least $9/10$ by a circuit C_j having $S_j(n)$ one and two-qubit gates and $q_j(n)$ query gates. We set $S_{\text{sum}}(n) = \sum_{j\in[J]} S_j(n)$, $q_{\text{sum}}(n) = \sum_{j\in[J](n)} q_j(n)$, and $q_{\max}(n) = \max_{j\in[J(n)]} q_j(n)$. Then, there exist two quantum algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ which, with probability at least $9/10$, find the index $j \in [J(n)]$ of a marked or minimal element in $\{f_1(x), \ldots, f_J(x)\}$, respectively. The complexities of the algorithms are as follows:*

1. *$\mathcal{A}_1$ makes $O(\sqrt{J(n)}q_{\max}(n))$ queries and takes time $O(\sqrt{J(n)}(S_{\text{sum}}(n) + q_{\max}(n) \cdot \text{QR}_n))$,*
2. *$\mathcal{A}_2$ makes $O(\log J(n) \cdot q_{\text{sum}}(n))$ queries and takes time $O(\log J(n)(S_{\text{sum}}(n) + q_{\text{sum}}(n) \cdot \text{QR}_n))$.*

B Bottom-up divide and conquer

When the create function is trivial then the subproblems that arise in a divide and conquer algorithm are known in advance. In this case, we can directly write an iterative algorithm rather than a recursive one, by sequentially solving the subproblems in an appropriate order.

B.1 Generic bottom-up bound

In the problems we look at in this paper, we are frequently trying to find a substring of a string $a \in \Sigma^n$ which maximizes some function.

To see how the bottom-up approach to such a problem works, consider the endpoints $i, j \in [n]$ of a substring which maximize the function of interest. When viewed as bit strings of length $\log n$ corresponding to the binary representation of $i-1, j-1$ respectively, these strings will share a common prefix of length $t \in \{0, 1, \ldots, \log(n) - 1\}$. This means that i, j are contained in a unique interval of length $n/2^t$ of the form $\{(k-1)n/2^t + 1, \ldots, kn/2^t\}$, for some $k \in [2^t]$, such that i is in the left half of this interval, and j is in the right half of this interval.

Thus to solve the original problem it suffices to solve the *crossing problem*: compute the maximum value $P(a,t,k)$ of the function of interest on substrings contained in an interval $\{(k-1)n/2^t + 1, \ldots, kn/2^t\}$ specified by k, t whose left endpoint is in the left half of the interval and the right endpoint is in the right half of the interval. The solution to the original problem is then $\max_t \max_{1\le k\le 2^t} P(a,t,k)$.

The next theorem gives an upper bound on the quantum complexity of the bottom-up approach in terms of the quantum complexity of solving the crossing problem $P(a,t,k)$.

▶ **Theorem 12.** *Let n a power of 2, and $T = \{0,\dots,\log(n)-1\}$. Let $U = \Sigma^n \times T \times [n]$, $S = \{(a,t,k) \in U : k \le 2^t\}$, and $P : S \to \mathbb{Z}$. Suppose that, for every $t \in T$, there is a quantum algorithm A_t that for every $a \in \Sigma^n, 1 \le k \le 2^t$ outputs $P(a,t,k)$ with probability at least $9/10$ in time $\sqrt{n/2^t} \cdot \tau(n,t)$, and with $\sqrt{n/2^t} \cdot \sigma(n,t)$ queries, for some functions $\tau,\sigma : \mathbb{N} \times \mathbb{N} \to \mathbb{R}_+$. Then there is a quantum algorithm that computes $\max_{t,1\le k\le 2^t} P(a,t,k)$ and the t,k realizing the maximum with probability at least $9/10$ in time*

$$O(\sqrt{n}\log\log(n) \cdot (\log n + \sum_{t\in T} \tau(n,t)))$$

and with $O(\sqrt{n}\log\log(n) \cdot \sum_{t\in T} \sigma(n,t))$ queries.

For a proof of Theorem 12, see the full version of the manuscript. Below, we focus on problems that have a quantum running time around $\sqrt{n}$, for which we give a tailored version of Theorem 12.

▶ **Theorem 13.** *Let n a power of 2, and $T = \{0,\dots,\log(n)-1\}$. Let $U = \Sigma^n \times T \times [n]$, $S = \{(a,t,k) \in U : k \le 2^t\}$ and $P : S \to \mathbb{Z}$. Suppose that, for every $t \in T$, there is a quantum algorithm A_t that for every $a \in \Sigma^n, 1 \le k \le 2^t$ outputs $P(a,t,k)$ with probability at least $9/10$ in time $\sqrt{n/2^t} \cdot (\log(n) + \mathrm{QR}_n)$, and with $O(\sqrt{n/2^t})$ queries. Then there is a quantum algorithm that computes $\max_t \max_{1\le k\le 2^t} P(a,t,k)$ and the t,k realizing the maximum with probability at least $9/10$ with $O(\sqrt{n\log(n)})$ queries, and a quantum algorithm to do this with running time $O(\sqrt{n\log(n)}\lambda_2(n,\mathrm{QR}_n))$.*

Proof. For any $a \in \Sigma^n, t \in T$, we can compute $\max_{1\le k\le 2^t} P(a,t,k)$ with probability at least $9/10$ in time $O(t\sqrt{2^t} + \sqrt{n}(\log(n) + \mathrm{QR}_n))$ by Corollary 10 and with $O(\sqrt{n})$ queries by Fact 2. To compute $\max_t \max_{1\le k\le 2^t} P(a,t,k)$ we use Lemma 11. The first item of the lemma gives an algorithm with $O(\sqrt{n\log n})$ queries and running time $O(\sqrt{n\log(n)}(\log^2(n) + \mathrm{QR}_n))$. The second item of the lemma gives an algorithm with running time $O(\sqrt{n}\log\log(n) \cdot (\log^2(n) + \log(n)\mathrm{QR}_n))$. Taking the minimum of the two options for the time complexity gives the claimed bound of $O(\sqrt{n\log(n)}\lambda_2(n,\mathrm{QR}_n))$. ◀

B.2 Applications

We now prove Theorem 1 from the main text.

▶ **Theorem** (Theorem 1 restated)**.** *The quantum query and time complexities of the problems SSST, LISst, LSIC and L20*2S are respectively $O(\sqrt{n\log n})$ and $O(\sqrt{n\log n} \cdot \lambda_2(n,\mathrm{QR}_n))$.*

Proof. Here we prove the theorem for SSST. The remaining cases are dealt with in the full version of the manuscript.

Let n be a power of 2, and for $t \in \{0,\dots,\log(n)-1\}, 1 \le k \le 2^t$ define $C_{n,t,k} = \{(k-1)n/2^t + 1,\dots,kn/2^t\}$. Note that $C_{n,t,k} = C_{n,t+1,2k-1} \cup C_{n,t+1,2k}$. Let a be an array of integers of size n. Define the problem $P(a,t,k)$ to be

$$P(a,t,k) = \max_{\substack{i\in C_{n,t+1,2k-1}\\ j\in C_{n,t+1,2k}}} a[j] - a[i] \ .$$

For any $i < j \in [n]$ there is a t,k such that $i \in C_{n,t+1,2k-1}$ and $j \in C_{n,t+1,2k}$. Thus

$$\max_{i<j} a[j] - a[i] = \max_t \max_{1\le k\le 2^t} P(a,t,k) \ .$$

By the quantum minimum finding algorithm [16] we can compute $P(a,t,k)$ with $O(\sqrt{n/2^t})$ queries and in time $O(\sqrt{n/2^t}(\log n + \mathrm{QR}_{n/2^t}))$. Thus by Theorem 13 there is a quantum algorithm to compute $\max_t \max_{1\le k\le 2^t} P(a,t,k)$ and the t,k realizing this value with probability at least 9/10 using $O(\sqrt{n\log(n)})$ queries and a quantum algorithm to do this with time $O(\sqrt{n\log(n)}\lambda_2(n,\mathrm{QR}_n))$. After we find the t,k realizing the maximum we can again solve $P(a,t,k)$ to find $\arg\max_{i<j} a[j]-a[i]$ and output these values.

We now prove by induction that there is a quantum algorithm for any n with the desired complexity. The base case is $n=2$ which is a power of 2 and so done. Now we design an algorithm for an input of a size $2^m < n < 2^{m+1}$ assuming we have an algorithm of the desired complexity for inputs of size at most 2^m. There are three choices for $\arg\max_{i<j} a[j]-a[i]$: either $1\le i,j\le 2^m$, $2^m < i,j \le n$, or $i\le 2^m, j > 2^m$. The first two cases can be solved by the inductive hypothesis, considering $a[1:2^m]$ and $a[2^m+1:n]$, respectively. For the third case, we can pad a to an array a' of size 2^{m+1} by adding $-\infty$ entries to the end and then solve the problem $P(a',0,1)$. We then take the maximum of the three cases. The running time is dominated by the first two cases and so is as desired. ◀

B.3 Generalizations

There are several interesting generalizations of the paradigmatic SSST problem, including d-Multiple Stocks Single Transaction (d-MSST),k-Increasing Subsequence, k-Signed Sum, k-Single Stock Multiple Transactions and Maximum 4-Combination. Details of these are given in the full version of the manuscript.

C Disjunctive and minimizing problems

We now turn to applications of Corollary 9 and Corollary 10 to design and analyze quantum divide and conquer algorithms. It turns out that we can do that for problems whose complete step is simple, and whose combine step is either the OR function or minimization. These problems are easily amenable to recursively applying Grover search or minimization on subproblems and therefore to quantum divide and conquer algorithms. However, let us emphasize that their create step can be rather complex.

These problems will be characterized by two functions, $h:\mathbb{N}\to\mathbb{N}$ and $m:\mathbb{N}\to\mathbb{R}_+$. In the divide and conquer algorithm an instance of length n will be divided into $h(n)$ instances of size $\lceil n/m(n)\rceil < n$. We call an integer n *small* for $m(n)$ if $\lceil n/m(n)\rceil \ge n$, otherwise n is *large* for $m(n)$. A string $a\in\Sigma^*$ is called *small* if $|a|$ is small. These strings correspond to the base cases of the problem, and the other strings are called *large*.

We will consider two different classes of disjunctive and minimizing problems depending on how the instances of the subproblems are presented during the recursive calls. In the class of *constructible instance* problems, the instances will be computed and written explicitly to the quantum memory. In the class of *t-decomposable instance* problems, all recursive calls will be made on subsequences of the original input string and therefore can be described potentially more succinctly by a sufficient number of memory indices.

C.1 Constructible instance problems

In constructible instance problems, the strings on which the recursive calls are made have to be computed explicitly, and therefore the definition of the problem on an instance doesn't make reference to the initial input. On the other hand, the definition involves the length of the initial input which determines the size of the quantum memory and the cost of all QRAM or QRAG operations made during the algorithm.

Let $h : \mathbb{N} \to \mathbb{N}$ and $m : \mathbb{N} \to \mathbb{R}_+$. We say that $P : \bigcup_{n^*}(\{n^*\} \times \Sigma^{\leq n^*}) \to \{0,1\}$ is a *constructible instance* (h,m)*-disjunctive* problem if:

1. There are only a constant number of small integers for $m(n)$.
2. There exists a completion function $\gamma : \bigcup_{0<n\leq n^*}(\{n^*\} \times \Sigma^n \times [h(n)] \times \{0,1\}) \to \{0,1\}$ such that, for every n^*, for every large $n \leq n^*$, and for every $a \in \Sigma^n$, there exist strings $a^{(1)}, \ldots, a^{(h(n))}$, with $|a^{(1)}| = \cdots = |a^{(h(n))}| = \lceil n/m(n) \rceil$, such that

$$P(n^*, a) = \mathrm{OR}_{j\in[h(n)]}\gamma(n^*, a, j, P(n^*, a^{(j)}))$$

Constructible instance (h,m)*-minimizing* problems $P : \bigcup_{n^*}(\{n^*\} \times \Sigma^{\leq n^*}) \to \mathbb{Z}$ are defined similarly, except with γ defined as $\gamma : \bigcup_{0<n\leq n^*}(\{n^*\} \times \Sigma^n \times [h(n)] \times \mathbb{Z}) \to \mathbb{Z})$ and $P(n^*, a) = \min_{j\in[h(n)]}\gamma(n^*, a, j, P(n^*, a^{(j)}))$.

For $j \in [h(n)]$, we call $a^{(j)}$ the jth *constitutive* string of a. For every n^*, for a disjunctive problem P, we define $P_{n^*} : \Sigma^{\leq n^*} \to \{0,1\}$ by $P_{n^*}(a) = P(n^*, a)$, and similarly for minimizing problems. The above conditions impose constraints on the create, complete and combine step of problems amenable to divide and conquer algorithms.

▶ **Theorem** (Theorem 7 formal version)**.** *Let P be a constructible instance (h,m)-disjunctive (respectively (h,m)-minimizing) problem. Suppose that $h(n)$ and $m(n)$ can be computed by classical algorithms in time $O(V(n))$. Let $N(n)$ be defined by the recursive equations $N(n) = O(1)$ when n is small and, when n is large, $N(n) = 2^k$, where k is the smallest integer such that $2^k > (h(n)+1)\max\{n, N(n)\}$.*

Fix n^* as the input size and let $\mathrm{QW} = \mathrm{QW}_{N(n^*)}$. Suppose that there is a classical algorithm that for every large $n \leq n^*$, for every $a \in \Sigma^n$, computes the $h(n)$ constitutive strings $a^{(1)}, \ldots, a^{(h(n))}$ in time $O(S(n))$. Finally suppose that the computation of $\gamma(n^*, a, j, P(n^*, a^{(j)}))$ can be done by a quantum algorithm in time $O(G(n) \cdot \mathrm{QW})$ and with error at most $1/10$.

Then there exists a quantum divide and conquer algorithm $\mathcal{A}$ which on an input size n^* and an instance $a \in \Sigma^n$, written at the beginning of the memory, where $n \leq n^*$, computes P_{n^*} with probability at least $9/10$ and uses a quantum memory of size $N(n^*)$. Let $\overline{T}(n^*, n)$ denote the time complexity of the algorithm, maximized over all words in Σ^n with $n \leq n^*$. Then $\overline{T}(n^*, n) = O(\log N(n^*) + \mathrm{QW})$ when n is small and, when n is large, $\overline{T}(n^*, n)$ satisfies the recurrence equation

$$\overline{T}(n^*, n) \leq c\sqrt{h(n)}\left(\overline{T}(n^*, n/m(n)) + G(n) \cdot \mathrm{QW} + \log h(n)\right) + O\left(S(n) \cdot \mathrm{QW} + V(n)\right),$$

for some constant $c > 0$.

The following corollary shows that when the completion function is trivial, an almost quadratic time complexity quantum speed-up over classical divide and conquer algorithms can be achieved.

▶ **Corollary 14.** *Let h and m be constants and let P be a constructible instance (h,m)-disjunctive or (h,m)-minimizing problem. Suppose that there is a classical algorithm that, for every large a of length n, computes the h constitutive strings $a^{(1)}, \ldots, a^{(h)}$ in time $O(S(n))$. Finally suppose that the completion function is trivial, meaning that it is the projection to its fourth argument.*

Then, there exists a quantum divide and conquer algorithm $\mathcal{A}$ which for every n^*, computes P_{n^*} with probability at least $9/10$ and uses a quantum memory of size $N(n^*)$. Let $\overline{T}(n^*, n)$ be the time complexity of the algorithm. Then

$$\overline{T}(n^*, n) = O(\log N(n^*) + \mathrm{QW}_{N(n^*)}) \ ,$$

when n is small and, when n is large, $\overline{T}(n^*, n)$ satisfies the recurrence equation

$$\overline{T}(n^*, n) \leq c\sqrt{h}\overline{T}(n^*, n/m) + O\left(S(n) \cdot \mathrm{QW}_{N(n^*)}\right) \quad ,$$

for some constant $c > 0$.

C.2 t-decomposable instance problems

In a constructible instance problem, the main difficulty can be in the create step, computing the constitutive strings which may be non-trivially related to the original input string. The situation is different when, for every instance a, the constitutive strings of a are subsequences of the original input string α. We will consider *t-decomposable instance* problems where, at every level of recursion, every constitutive string is the concatenation of t substrings of the input, for some constant t.

For some integers $n^* \geq n > 0$, let $\alpha \in \Sigma^{n^*}$ and let $t > 0$ be an integer constant. We say that $\mathcal{I} \in [n^*]^{2t}$ is an *n^*-valid t-description* if $\mathcal{I} = (b_1, e_1, \ldots, b_t, e_t)$ with $1 \leq b_1 \leq e_1 < b_2 \leq \ldots < b_t \leq e_t \leq n^*$. We define the *size* of $\mathcal{I}$ as $s(\mathcal{I}) = \sum_{i=1}^{t}(e_i - b_i + 1)$. Let $\mathcal{V}(n^*, t)$ denote the set of n^*-valid t-descriptions, and for $0 < n \leq n^*$, let $\mathcal{V}(n^*, t, n)$ denote the set of n^*-valid t-descriptions of size n. For $\mathcal{I} \in \mathcal{V}(n^*, t)$, where $\mathcal{I} = (b_1, e_1, \ldots, b_t, e_t)$, we denote by $\alpha_{\mathcal{I}}$ the string $\alpha[b_1 : e_1] \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} \alpha[b_t : e_t]$. Observe that $|\alpha_{\mathcal{I}}| = s(\mathcal{I})$. Let $a \in \Sigma^n$, we say that a is *t-decomposable* in α if there exists $\mathcal{I} \in \mathcal{V}(n^*, t, n)$ such that $a = \alpha_{\mathcal{I}}$, and we call $\mathcal{I}$ a *t-description* of a in α. We are interested in t-decomposable instance problems, meaning that for every $\alpha \in \Sigma^{n^*}$, for every large t-decomposable subsequence a of α, the constitutive strings of a are all t-decomposable in α.

We now formally define t-decomposable instance problems. Let $\alpha \in \Sigma^{n^*}$ be an input. An instance $a \in \Sigma^n$ which is a t-decomposable in α will be specified by an n^*-valid t-description $\mathcal{I}$ of a in α of size n. The t-description can be given in some work registers which is distinct from the memory registers. Observe that $\mathcal{I}$ is an $O(\log n^*)$ string. We will involve in the definition the create functions δ where, for every $\alpha \in \Sigma^{n^*}$, for every $\mathcal{I} \in \mathcal{V}(n^*, t, n)$, and for every $j \in [h(n)]$, the value of $\delta(\alpha, \mathcal{I}, j)$ is a t-description of the jth constitutive string $\alpha_{\mathcal{I}}^{(j)}$ of $\alpha_{\mathcal{I}}$.

Let $h : \mathbb{N} \to \mathbb{N}$ and $m : \mathbb{N} \to \mathbb{R}_+$ and $t > 0$ an integer constant. We say that $P : \bigcup_{n^*}(\Sigma^{n^*} \times \mathcal{V}(n^*, t)) \to \{0, 1\}$ is a *t-decomposable instance (h, m)-disjunctive* problem if:

1. There are only a constant number of small integers for $m(n)$.
2. There exist two functions

$$\delta : \bigcup_{0<n\leq n^*} (\Sigma^{n^*} \times \mathcal{V}(n^*, t, n) \times [h(n)]) \to \mathcal{V}(n^*, t, \lceil n/m(n) \rceil)$$

and

$$\gamma : \bigcup_{0<n\leq n^*} (\Sigma^{n^*} \times \mathcal{V}(n^*, t, n) \times [h(n)] \times \mathcal{V}(n^*, t, \lceil n/m(n) \rceil \times \{0, 1\})) \to \{0, 1\}$$

such that for every $\alpha \in \Sigma^{n^*}$, for every $\mathcal{I} \in \mathcal{V}(n^*, t, n)$ where n is large,

$$P(\alpha, \mathcal{I}) = \mathrm{OR}_{j\in[h(n)]}\gamma(\alpha, \mathcal{I}, j, \delta(\alpha, \mathcal{I}, j), P(\alpha, \delta(\alpha, \mathcal{I}, j))).$$

We call the functions δ and γ respectively the *create* and the *completion* function, and for $j \in [h(n)]$, we call $\alpha_{\delta(\alpha,\mathcal{I},j)}$ the jth *constitutive* string of $\alpha_{\mathcal{I}}$. *t-decomposable instance (h, m)-minimizing* or *maximizing* problems can similarly be defined.

We now state a formal version of Theorem 8, which stipulates the existence of a quantum divide and conquer algorithm for t-decomposable instance disjunctive or minimizing problems.

▶ **Theorem** (Theorem 8 formal version). *Let P be a t-decomposable instance (h, m)-disjunctive or minimizing problem. Suppose that $h(n)$ and $m(n)$ can be computed by classical algorithms in time $O(V(n))$.*

Suppose there is a quantum algorithm that, for every input string $\alpha \in \Sigma^{n^*}$ given in the quantum memory, for every large n, for every $\mathcal{I} \in \mathcal{V}(n^*, t, n)$, computes $\delta(\alpha, \mathcal{I}, j)$, for all $j \in [h(n)]$, in time $O(D(n) \cdot \mathrm{QW}_{M(n^*)})$, for some function M. Suppose also that each computation has probability of error at most $1/10$.

Let $N'(n)$ be defined by the recursive equations $N'(n) = O(1)$ when n is small and $N'(n) = N'(n/m(n)) + M(n)$ when n is large. Also define $N(n^*) = n^* + N'(n^*)$. Finally suppose that for every input string $\alpha \in \Sigma^{n^*}$, for every large n, for every $\mathcal{I} \in \mathcal{V}(n^*, t, n)$, the computation of the completion function γ can be done by a quantum algorithm in time $O(G(n) \cdot \mathrm{QW}_{N(n^*)})$ and with error at most $1/10$.

Then, there exists a quantum divide and conquer algorithm $\mathcal{A}$ which computes P on t-decomposable instances in α with probability at least $8/10$ and uses a quantum memory of size $N(n^*)$.

Denote by $\overline{T}(n^*, n)$ its time complexity, maximized over all inputs α of length n^* and over all t-decomposable instances in α of length n, specified by a t-description in α. Then, for small n, we have

$$\overline{T}(n^*, n) = O(\log N(n^*) + \mathrm{QW}_{N(n^*)}).$$

When n is large, $\overline{T}(n^*, n)$ satisfies, for some constant $c > 0$, the recurrence

$$\overline{T}(n^*, n) \leq c\sqrt{h(n)} \left(\overline{T}(n^*, n/m(n)) + G(n) \cdot \mathrm{QW}_{N(n^*)} + \log h(n)\right) + O\left(D(n) \cdot \mathrm{QW}_{N(n^*)} + V(n)\right).$$

Proof. See full version of the manuscript, where similar results are stated for the QRAM model, and for query complexity. ◀

D Longest Distinct Substring

Recall that a substring of α is a subsequence of consecutive symbols. We say that a substring is *distinct* if no character in the substring appears more than once.

▶ **Problem 1** (Longest Distinct Substring). *Let Σ be an alphabet and $\alpha \in \Sigma^{n^*}$ be a string. The goal is to output the length of the longest distinct substring of α, denoted $LDS(\alpha)$.*

Computing $\mathrm{LDS}(\alpha)$ is a maximization problem over all substrings of α and fits into the bottom-up divide and conquer framework of Section B. By repeating the last character of α we may assume that its size is a power of 2 without affecting $\mathrm{LDS}(\alpha)$. Letting $P(\alpha, t, k)$ be the length of the longest distinct substring of $\alpha[(k-1)n/2^t + 1 : kn/2^t]$ whose left endpoint is in the left half of this interval and the right endpoint is in the right half of the interval, we see that $\mathrm{LDS}(\alpha) = \max_t \max_{1 \leq k \leq 2^t} P(\alpha, t, k)$. Our main task is thus to compute $P(\alpha, t, k)$. This will be done with a divide and conquer algorithm with a non-trivial create step. Because of the nature of the recursive calls in the divide and conquer algorithm for $P(\alpha, t, k)$, it is helpful to consider a more general version of the problem which we call Bipartite Longest Distinct Substring (BLDS).

▶ **Problem 2** (Bipartite Longest Distinct Substring). *For an n^*-valid 2-description $\mathcal{I}$ let $\mathcal{I}(k)$ refer to the kth element of $\mathcal{I}$. Given α and a n^*-valid 2-description $\mathcal{I}$, the goal of the bipartite longest distinct substring problem is to output the length of a longest distinct substring of $\alpha[\mathcal{I}(1) : \mathcal{I}(2)] +\!\!+ \alpha[\mathcal{I}(3) : \mathcal{I}(4)]$ that includes at least one of $\alpha_{\mathcal{I}(2)}, \alpha_{\mathcal{I}(3)}$. We denote this value by $BLDS(\alpha, \mathcal{I})$.*

In the full version of the manuscript we prove the following theorem.

▶ **Theorem 15.** *For any constant integer $h \geq 2$, BLDS$(\alpha, \mathcal{I})$ is a 2-decomposable instance (h, h)-maximizing problem. When $s(\mathcal{I}) = n$, the create and completion functions can be computed by a quantum algorithm with oracle access to α in time $\widetilde{O}(n^{2/3}) \cdot \mathrm{QW}_{O(n^*)}$ and with $O(n^{2/3} \log(n))$ queries.*

We can now apply Theorem 8 to give the following.

▶ **Theorem 16.** *There is a quantum algorithm that for any n^*-valid 2-description $\mathcal{I}$ with $s(\mathcal{I}) = n$ computes BLDS$(\alpha, \mathcal{I})$ in time $\widetilde{O}(n^{2/3}) \cdot \mathrm{QW}_{O(n^*)} + O(\sqrt{n} \log(n^*))$ and with $O(n^{2/3} \cdot \log(n))$ queries.*

Proof. We prove the time complexity result here and defer the query complexity proof to the full version of the manuscript. For convenience, let $f(n) = \widetilde{O}(n^{2/3}) \cdot \mathrm{QW}_{O(n^*)}$ be an upper bound on the time complexity of the create and completion functions from Theorem 15. By Theorem 8 and Theorem 15, the time complexity $\overline{T}(n)$ for BLDS$(\alpha, \mathcal{I})$ with $s(\mathcal{I}) = n$ satisfies the following recurrence relation for any $h \geq 2$:

$$\overline{T}(1) = O(\log n^* + \mathrm{QW}_{O(n^*)})$$
$$\overline{T}(n) \leq c\sqrt{h}(\overline{T}(n/h) + f(n)) + f(n) \ .$$

By taking $h = 2c^6$, the solution to this recurrence becomes $\widetilde{O}(n^{2/3}) \cdot \mathrm{QW}_{O(n^*)} + O(\sqrt{n} \log n^*)$. ◀

Finally, we can use the algorithm for BLDS as part of a bottom-up algorithm for LDS.

▶ **Theorem 17.** *There is a quantum algorithm that solves LDS in time $\widetilde{O}(n^{2/3}) \cdot \mathrm{QW}_{O(n)}$ and a quantum algorithm that solves LDS with $O(n^{2/3} \log(n) \log\log(n))$ queries.*

Proof. We use the bottom-up approach. Let the input be $a \in \Sigma^n$. We can pad the input by repeating the last character without changing the length of the longest distinct substring. Thus we may assume n is a power of 2 without changing the asymptotic complexity.

Let $P(a, t, k)$ be the length of the longest distinct substring of a contained in the interval $\{(k-1)n/2^t + 1, \ldots, kn/2^t\}$, with the left endpoint in the left half of this interval and right endpoint in the right half of this interval. Letting $i_1 = (k-1)n/2^t + 1, i_2 = (2k-1)n/2^{t+1}, j_1 = (2k-1)n/2^{t+1} + 1, j_2 = kn/2^t$ and $\mathcal{I} = (i_1, i_2, j_1, j_2)$ we see that $P(a, t, k) = \mathrm{BLDS}(a, \mathcal{I})$. Thus by Theorem 16 there is a quantum algorithm that outputs $P(a, t, k)$ with probability at least 9/10 in time $\widetilde{O}(n^{2/3}/2^{2t/3} \cdot \mathrm{QW}_{O(n)})$ and with $O(n^{2/3}/2^{2t/3} \log(n/2^t)$ queries. Applying Theorem 12 gives a quantum algorithm for LDS with running time $\widetilde{O}(n^{2/3}) \cdot \mathrm{QW}_{O(n)}$ and a quantum algorithm with query complexity $O(n^{2/3} \log(n) \log\log(n))$. ◀