Electronic Colloquium on Computational Complexity, Report No. 5 (2023)

Cumulative Memory Lower Bounds for Randomized and Quantum Computation

Paul Beame*
Computer Science & Engineering
University of Washington

Niels Kornerup
Computer Science
University of Texas at Austin

January 13, 2023

Abstract

Cumulative memory—the sum of space used over the steps of a computation—is a fine-grained measure of time-space complexity that is a more accurate measure of cost for algorithms with infrequent spikes in memory usage in the context of technologies such as cloud computing that allow dynamic allocation and de-allocation of resources during their execution. We give the first lower bounds on cumulative memory complexity that apply to general sequential classical algorithms. We also prove the first such bounds for bounded-error quantum circuits. Among many possible applications, we show that any classical sorting algorithm with success probability at least $1/\text{poly}(n)$ requires cumulative memory $\tilde{\Omega}(n^2)$, any classical matrix multiplication algorithm requires cumulative memory $\Omega(n^6/T)$, any quantum sorting circuit requires cumulative memory $\Omega(n^3/T)$, and any quantum circuit that finds k disjoint collisions in a random function requires cumulative memory $\Omega(k^3 n/T^2)$. More generally, we present theorems that can be used to convert a wide class of existing time-space tradeoff lower bounds to matching lower bounds on cumulative memory complexity.

*Research supported by NSF grant CCF-2006359

ISSN 1433-8092

1 Introduction

There are many problems where algorithms can use additional memory for faster running times or expend additional time to reduce memory requirements. While there are many different kinds of tradeoffs between time and space, the most common complexity metric for such algorithms is the maximum time-space (TS) product. This metric is appropriate when a machine must allocate an algorithm's maximum space throughout its computation. However, recent technologies like AWS Lambda [BBB+21], Flex [LL20], and CloudScale [SSGW11] suggest that in the context of cloud computing, space can be allocated to a program only as it is needed. When using such services, analyzing the average memory used per step leads to a more accurate picture than merely measuring the maximum space used.

Cumulative memory (CM) - the sum over time of the space used per step of an algorithm - is an alternative notion of time-space complexity that is more fair to algorithms that only require rare spikes in memory. The term cumulative memory complexity was first coined by Alwen and Serbinenko [AS15] who introduced it as a way to discuss time-space tradeoffs for "memory hard functions" like password hashes. Since then, lower and upper bounds on the CM complexity of problems in structured computational models using the black pebble game have been extensively studied, beginning with the work of [AS15, AB16, RD16, ACP+17, ACK+16, ABP17]. These structured models via pebble games are particularly natural in the context of the random oracle assumptions that are common in cryptography. By carefully interweaving their memory intensive steps, these authors devise algorithms for cracking passwords that compute many hashes in parallel using only slightly more space than is necessary to compute a single hash. While such algorithms can use parallelism to amortize costs and circumvent proven single instance TS complexity lower bounds, their cumulative memory scales linearly with the number of computed hashes. Thus cumulative memory complexity is a more robust metric than TS complexity.

Surprisingly there has been little research into CM complexity outside the setting of cryptography. In [AdRNV17] the authors showed strong CM complexity results for the black-white pebble game and used them to derive related results for resolution proof systems. Our work is the first to explore CM complexity outside the regime of pebbling and the first to obtain results that apply to general models of computation without cryptographic or black-box assumptions.

Our Results

In this work, we give *generic methods* that allow one to convert existing paradigms for obtaining time-space tradeoff lower bounds involving worst-case space to new lower bounds that replace the time-space product by cumulative space, immediately yielding a host of new lower bounds on cumulative memory complexity. With these methods, we show how to extend almost all known proofs for time-space tradeoffs to equivalent lower bounds on cumulative memory complexity. Our results, like those of existing time-space tradeoffs, apply in models in which arbitrary sequential computations may be performed between queries to a read-only input. Our lower bounds also apply to randomized and quantum algorithms that are allowed to make errors.

Classical computation We first focus on lower bound paradigms that apply to computations of multi-output functions. We give general theorems showing how to translate the basic ideas that yield virtually all time-space tradeoffs known for such functions to yield lower bounds on cumulative memory complexity. As applications of our general methodology, we prove that the cumulative memory required by any sorting algorithm is $\tilde{\Omega}(n^2)$ which generalizes [BC82, Bea91] and the cumulative memory required for any matrix multiplication algorithm using time T is $\Omega(n^6/T)$, generalizing [Abr91].

We also show how the paradigm can be extended to correspond to the best time-space tradeoff lower bounds for single-output Boolean functions. In particular, we give examples of functions for which algorithms running in time T require cumulative memory $\Omega((n^2 \log n)/2^{cT/n})$ for some constant $c > 0$, generalizing [BSSV03]. This means, for example, that algorithms computing these functions in time $o(n \log n)$ require cumulative memory $\Omega(n^2 \log n)$.

Quantum computation We generalize the quantum time-space tradeoff for sorting proven in [KŠdW07], which requires that the time order in which output values are produced must correspond to the sorted order, to a matching cumulative memory complexity bound of $\Omega(n^3/T)$ that works for any fixed time-ordering of output production which yields a more general lower bound[1]. We then show how our classical general theorems can be applied to known quantum time-space tradeoffs and extend the quantum time-space tradeoff for k-collision pairs finding from [HM21] to the matching cumulative memory complexity bound of $\Omega(k^3 n/T^2)$.

Previous work

Memory hard functions and cumulative memory complexity Memory hard functions (MHFs) are functions designed to require large space to compute. In [AS15] Alwen and Serbinenko introduced parallel cumulative (memory) complexity as a metric for analyzing the space footprint required to compute these functions. Most MHFs are constructed using hashgraphs (see [DNW05]) of DAGs whose output is a fixed length string and their proofs of security are based on pebbling arguments on these DAGs while assuming access to truly random hash functions for their complexity bounds [AS15, BCGS16, BDK16, RD16, ABP17, ACP+17, BZ17]. More recent MHF constructions do not rely on random hash functions; however, they still require some cryptographic assumptions [CT19, ABB21]. In general the major focus of these results has been on savings with parallel rather than sequential computation.

Classical time-space tradeoffs While early work focused on the kinds of restricted pebbling models similar to those considered to date for cumulative memory complexity [Tom80, BFK+81], the gold standard model for time-space tradeoff analysis is that of unrestricted branching programs, which simultaneously capture both time and space for general sequential computation. This analysis began with the work of [BC82] who proved lower bounds for sorting and introduced a general methodology for multi-output functions that has been extended to many problems (e.g., [Yes84, Abr87, Abr90, Bea91, MNT93]), including universal hashing and a wide array of problems in linear algebra [Abr91]. A separate methodology for single-output functions, first introduced in the context of restricted branching programs [BRS93, Oko93], was extended to general branching programs in [BJS01], with further applications to other problems [Ajt02] including multi-precision

[1] For example, an algorithm may be able to determine the median output long before it determines the other outputs.

integer multiplication [SW03] and error-correcting codes [Juk09] as well as over Boolean input domains [Ajt05, BSSV03].

Both of these methodologies involve breaking the branching program into blocks. For multi-output functions one needs to show that for any fixed node at the beginning of a block, the probability over a random input that the program started at that node produces k correct output values in that block decays exponentially in k. For single-output functions, one decomposes the space of inputs based on the "trace" of nodes traversed at segment boundaries. Based on the traces, one can determine the size and density properties of "embedded rectangles" of inputs on which the function must be constant. Lower bounds require showing the given function does not have such rectangles.

Quantum time-space tradeoffs Though the basic notion of exponential decay in producing correct outputs is similar to the classical multi-output bounds, the arguments are substantially more subtle in the quantum setting. The quantum query model gives us access to an input $X = x_1, \ldots, x_n$ via an oracle Q_X. Since the result of a single quantum query can change if we flip any bit of the input, we need arguments that limit the sensitivity of a query to changes in the oracle. These arguments generally follow one of two techniques: the adversary method [BBBV97, Amb02, ŠS05] or the polynomial method [BBC+01].

To obtain quantum time-space tradeoffs for multi-output functions, it is important to have lemmas showing that query-bounded computations only yield a slight advantage over randomly guessing outputs. Such lemmas often take the form of direct product theorems, which state that if T queries are necessary to solve one instance of a problem with constant probability, then kT queries are insufficient to solve k independent instances of that problem with probability $2^{-\Omega(k)}$. While such results seems intuitive, Shaltiel proved that they are not true in general [Sha03]. The polynomial method [Aar05, KŠdW07, She11] and the adversarial method [AŠdW09] have both been extended to prove quantum direct product theorems.

In [KŠdW07] the authors use direct product theorems to prove a tight time-space tradeoff for sorting and a time-space tradeoff for matrix multiplication in Boolean algebra. They also proved somewhat weaker lower bounds for computing matrix-vector products for fixed matrices A; those bounds were extended in [AŠdW09] to systems of linear inequalities. However, both of these latter results apply to computations where the fixed matrix A defining the problem depends on the space bound and, unlike the case of sorting or Boolean matrix multiplication, do not yield a fixed problem for which the lower bound applies at all space bounds. More recently [HM21] extended the recording query technique of Zhandry in [Zha19] to obtain time-space lower bounds for the k-collision problem and match the aforementioned result for sorting.

Our methods

At the highest level, we employ part of the same paradigms previously used for time-space tradeoff lower bounds. namely breaking up the computations into blocks of time and analyzing properties of the branching programs or quantum circuits based on what happens at the boundaries between those time blocks. However, for cumulative memory complexity, those boundaries cannot be at fixed locations in time and their selection needs to depend on the space used in those time steps.

Further, in many cases, the time-space tradeoff lower bound needs to set the lengths of those time blocks in a way that depends on the specific space bound. When extending the ideas to bound cumulative memory usage, there is no single space bound that can be used throughout the computation; this sets up a tricky interplay between the choices of boundaries between time blocks and the lengths of the time blocks. Because the space usage within a block may grow and shrink radically, even with optimal selection of block boundaries, the contribution of each time block to the overall cumulative memory may be significantly lower than the time-space product lower bound one would obtain for the individual block.

We show how to bound any loss in going from time-space tradeoff lower bounds to cumulative memory lower bounds in a way that depends solely on the bound on the lengths of blocks as a function h_0 of the target space bound. For many classes of bounding functions we are able to bound the loss by a constant factor, and we are able show that it is always at most an $O(\log n)$ factor loss. Once we have this, if this bounding function h_0 is non-constant, there is still a matter of bounding the optimum way for the algorithm to allocate its space budget for producing the require outputs throughout its computation. This optimization again depends on the bounding function h_0. This involves minimizing a convex function based on h_0 subject to a mix of convex and concave constraints which is not generally tractable. However, assuming that h_0 is nicely behaved, we are able to apply specialized convexity arguments to yield lower bounds on cumulative memory complexity that in many instances match those of previous time-space tradeoffs up to a constant factor.

Road map We give the overall definitions in Section 2, including a review of the standard definitions of the work space used by quantum circuits. We then give the very simple version of our methods that is needed to prove results on the cumulative memory complexity of classical sorting algorithms in Section 3. In Section 4, we give our lower bound for quantum sorting algorithms. This example shows something of the complexity required for our general arguments; in this case, the bounding function is simple enough that we can apply an alternative direct argument to show only constant loss in the choices of boundaries for time blocks, but it still requires some of the complexity of the general optimization of space allocation. This section also includes the additional ideas that allow us to analyze circuits that produce sorted outputs in arbitrary sequential time steps.

We give the full general theorems that let us convert classical time-space tradeoffs for multi-output functions to cumulative memory lower bounds, even for randomized algorithms, in Section 5. In Section 6 we apply these general theorems to a few other problems, particularly those in linear algebra, to give an indication of how they can be used. Next, we show how to convert time-space tradeoff lower bounds for single-output functions to cumulative memory lower bounds in Section 7. Finally, in Section 8 and Section 9 we show how to extend our generic method to quantum circuits and discuss its application to other existing quantum time-space tradeoffs. The appendices contain some of the technical arguments that allow us to bound the loss functions and to give bounds on the optimum allocations of cumulative space budgets to time steps.

2 Preliminaries

Cumulative memory is an abstract notion of time-space complexity that can be applied to any model of computation with a natural notion of space.

Definition 2.1. The *cumulative memory* of a discrete time computation $\mathcal{A}$ that uses $|\mathcal{A}_t|$ space during the t-th step and runs in time T is:

$$CM(\mathcal{A}) = \sum_{t=1}^{T} |\mathcal{A}_t|$$

The *cumulative memory complexity* of a function f with respect to a computational model M is:

$$CMC(f) = \min_{\mathcal{A} \in M \text{ computes } f} CM(\mathcal{A}).$$

In this paper we consider both branching programs and quantum circuits.

Branching Programs Branching programs with input $\{x_1, \ldots, x_n\} \in D^n$ are known as D-way branching programs and are defined using a rooted DAG in which each non-sink vertex is labeled with an $i \in [n]$ and has $|D|$ outgoing edges that correspond to possible values of x_i. Each edge is optionally labeled by some number of output statements expressed as pairs (j, o_j) where $j \in [m]$ is an output index and $o_j \in R$ (if outputs are to be ordered) or simply $o_j \in R$ (if outputs are to be unordered). Evaluation is performed by starting at the root v_0 and following the appropriate labels of the respective x_i. We consider branching programs P that contain $T + 1$ layers where the outgoing edges from nodes in each layer t are all in layer $t + 1$. We impose no restriction on the query pattern of the branching program or when it can produce parts of the output. We define the following complexity measures for such a branching program P. The *time* of the branching program is $T(P) = T$. The *space* of the branching program is $S(P) = \max_t \log_2 |L_t|$ where L_t is the set of nodes in layer t. Observe that in the absence of any limit on its space, a branching program could equally well be a decision tree; hence the minimum time for branching programs to compute a function f is its *decision tree complexity*. The *time-space* (product) used by the branching program is $TS(P) = T(P)S(P)$. The *cumulative memory* used by the branching program is $CM(P) = \sum_t \log_2 |L_t|$.

Branching programs are very general and simultaneously model time and space for sequential computation. In particular they model time and space for random-access off-line multitape Turing machines and random-access machines (RAMs) when time is unit-cost, space is log-cost, and the input and output are read-only and write-only respectively[2]. Branching programs are much more flexible than these models since they can make arbitrary changes to their storage in a single step.

[2] In prior work, branching program space has often been defined to be the logarithm of the total number of nodes (e.g., [BC82, Abr91]) rather than the logarithm of the width (maximum number of nodes per layer), though the latter has been used (e.g., [CFL83]). The natural conversion from an arbitrary space-bounded machine to a branching program produces one that is not leveled (i.e., nodes are not segregated by time step). After leveling the branching program, the space of the original machine becomes the logarithm of the width (cf. [Pip79]). The width-based definition is also the only natural one by which to measure cumulative memory complexity and, in any case, the two definitions differ by at most an additive $\log_2 T$ amount, with lower bounds on width implying lower bounds on size.

Quantum Circuits We also consider quantum circuits $\mathcal{C}$ classical read-only input $X = x_1, \ldots, x_n$ that can be queried using an XOR query oracle as shown in Figure 1. As is normal in circuit models, each output wire is associated with a fixed position in the output sequence, independent of the input. As shown in Figure 2 following [KŠdW07], we abstract an arbitrary quantum circuit $\mathcal{C}$ into layers $\mathcal{C} = \{L_1, \ldots, L_T\}$ where layer L_t starts with the t-th query Q to the input and ends with the start of the next layer. During each layer, an arbitrary unitary transformation V gets applied which can express an arbitrary sub-circuit involving input-independent computation. The sub-circuit/transformation V outputs S_t qubits for use in the next layer in addition to some qubits that are immediately measured in the standard basis, some of which are treated as classical write-only output. The time of $\mathcal{C}$ is lower bounded by the number of layers T and we say that the space of layer L_t is S_t. Observe that to compute a function f, T must be at least the *quantum query complexity* of f since that measure corresponds the above circuit model when the space is unbounded. Note that the cumulative memory of a circuit is lower-bounded by the sum of the S_t. For convenience we define S_0, the space of the circuit before its first query, to be zero. Thus we only consider the space after the input is queried.

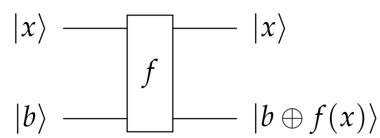

Figure 1: The XOR oracle for a function $f : D \to R$ where $D \subseteq \{0,1\}^n$ and $R \subseteq \{0,1\}^m$ is the linear operator that, for all $x \in D$ and $b \in \{0,1\}^m$, maps the basis state $|x\rangle |b\rangle$ to $|x\rangle |b \oplus f(x)\rangle$. When $x \notin D$, the XOR oracle acts like the identity. The query oracle Q_X (where $X = x_1, \ldots, x_n$) is the XOR oracle for the function $f(i) = x_i$.

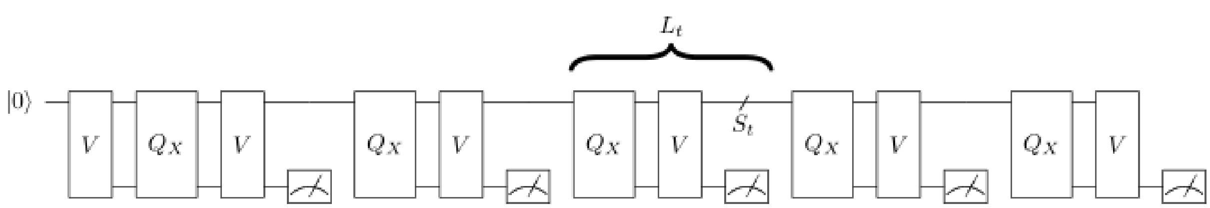

Figure 2: The abstraction of a quantum circuit into layers.

Simulating a quantum query from modified input Without making any additional assumptions on our query oracle, it is possible to simulate a query for a modified (possibly larger) input using at most two queries to the original input. Let Q_X be an XOR query oracle for some input $X = x_1, \ldots x_n$ where $x_i \in \{0,1\}^m$ and let

$$\hat{x}_i = \begin{cases} x_i & i \in [n] \\ 0 & \text{otherwise} \end{cases}$$

By definition, this makes Q_X the permutation that maps any basis state $|i, j, k\rangle$ where $i \in \{0,1\}^{\lceil \log_2 n \rceil}$ and $j \in \{0,1\}^m$ to the basis state $|i, j \oplus \hat{x}_i\rangle$. We want to use Q_X to simulate queries to some modified input $X' = \{x'_1, \ldots, x'_{n+t}\}$ where $x'_i \in \{0,1\}^\ell$ is defined by $x'_i = g(i, \hat{x}_i)$. Let G be the XOR query oracle for $g(i,j)$ and $P_{i>n}$ be the XOR query oracle for the predicate function $p(i) = 1_{i>n}$. Then the circuit in Figure 3 simulates an XOR query on modified input X'.

Note that G and $P_{i>n}$ both compute classical functions and therefore can be computed using a network of Toffoli gates. Since G is independent of X, this circuit simulates a query to X' using at most two queries to X. The second query of Q_X is necessary to uncompute the m qubit ancillary register.

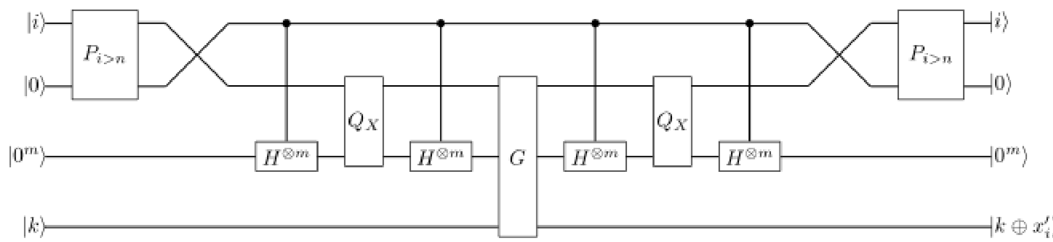

Figure 3: The above circuit uses two calls to a query oracle Q_X in order to simulate one query to the modified input X'.

3 Cumulative memory complexity of classical sorting algorithms

For a natural number N, the standard version of *sorting* is a function $Sort_{n,N} : [N]^n \to [N]^n$ that on input $x \in [N]^n$ produces an output $y \in [N]^n$ in non-decreasing order where y is a permutation of x; that is, there is some permutation π such that $y_i = x_{\pi(i)}$ for all $i \in [n]$. A related problem is the *ranking* problem $Rank_{n,N} : [N]^n \to [n]^n$ which on input $x \in [N]^n$ produces a permutation π represented as the vector $(\pi(1), \ldots, \pi(n))$ such that $Sort_{n,N}(x) = (x_{\pi(1)}, \ldots, x_{\pi(n)})$ and whenever $x_i = x_j$ for $i < j$ we have $\pi(i) < \pi(j)$.

Proposition 3.1 ([BC82]). *(a) If there is an $[nN]$-way branching program P computing $Sort_{n,nN}$ then there is an $[N]$-way branching program P' computing $Rank_{n,N}$ with $T(P') \le T(P)$, $S(P') \le S(P)$, and $CM(P') \le CM(P)$.*

(b) If there is an $[N]$-way branching program P'' computing $Rank_{n,N}$ then there is an $[N]$-way branching program P''' computing $Sort_{n,N}$ with $T(P''') \le 2T(P'')$, $S(P''') \le S(P'') + \log_2 N$, and $CM(P''') \le 2CM(P'') + T(P''') \log_2 N$.

Proof. For part (a), the program P' is exactly P except that when P queries $x_i \in [Nn]$, P' reads $x'_i \in [N]$ and branches on value $x_i = (x'_i, i)$ and when P outputs $(i, y_i) = (i, x_{\pi(i)})$ on an edge for $x_{\pi(i)} = (x'_{\pi(i)}, \pi(i))$, P' outputs $(i, \pi(i))$. For part (b), the program P''' is exactly P'' except that whenever P'' outputs $(i, \pi(i))$ on an edge, P''' queries $x_{\pi(i)}$ and outputs $(i, x_{\pi(o)})$. One layer becomes two layers and the number of nodes per layer of P''' is at most N times that of P''. □

Following [BC82], we focus on inputs where the x_i are distinct. In this case, the tie-breaking we enforced in defining $Rank_{n,N}$ when there are equal elements is irrelevant.

Proposition 3.2 ([BC82]). *There is an $\alpha > 0$ such that the following holds. Let n be sufficiently large and μ be the uniform distribution over lists of n distinct integers from $[n^2]$. Then for any branching program B of height $h \leq \alpha n$ and for all integers $k \leq 2\alpha n$, the probability for $x \sim \mu$ that B produces at least k correct output values of $Rank_{n,n^2}$ on input x is at most $2^{-k/\lceil \log_2 n \rceil}$.*

Theorem 3.3. *Let P be a branching program computing $Sort_{n,n^3}$ with probability at least $n^{-O(1)}$ and $T = T(P)$. Then T is $\Omega(n^2/\log^2 n)$ or $CM(P)$ is $\Omega(n^2/\log n)$. Further, any random access machine computing $Sort_{n,n^3}$ with $n^{-O(1)}$ probability requires cumulative memory of $\Omega(n^2/\log n)$ bits.*

Proof. We prove the same bounds for branching programs P computing $Rank_{n,n^2}$ which, by Proposition 3.1, implies the bounds for computing $Sort_{n,n^3}$.

For simplicity we first assume that P is determistic and is always correct. Let α be the constant and μ be the probability distributuon on $[n^2]^n$ from Proposition 3.2, and let $H = \lfloor \frac{\alpha}{2} n \rfloor$. We partition P into $\ell = \lceil T/H \rceil$ intervals $\{I_1, \ldots, I_\ell\}$, all of length H except for the first, which may be shorter than the rest. Let $t_1 = 0$, $t_{\ell+1} = T$, and for $i \in [2, \ell]$, t_i be the time-step in I_i with the fewest number of nodes. We define $S_i = \log_2(|L_{t_i}|)$ where L_j is the set of nodes of P in layer j. The i-th time block B_i will contain all layers from t_i to t_{i+1}. We observe:

$$CM(P) \geq \sum_{i=2}^{\ell} S_i\, H = H \sum_{i=1}^{\ell} S_i \tag{1}$$

since $S_1 = 0$. Define $k_i = \lceil \lceil \log_2 n \rceil (S_i + \log_2(2T)) \rceil$, which will be our target number of outputs for block B_i. By our choice of B_i we know its length is at most αn and it starts at a layer with 2^{S_i} nodes. So, by Proposition 3.2, combined with a union bound, the probability for $x \sim \mu$ that B_i produces at least k_i correct output values of $Rank_{n,n^2}$ on input $x \sim \mu$ is at most $1/(2T)$. Thus the probability over μ that at least one block B_i produces at least k_i correct output values is at most $1/2$ and the probability that the total number of outputs produced is at most $\sum_{i=1}^{\ell}(k_i - 1)$ is at least $1/2$. Since P must always produce n correct outputs, we must have:

$$\sum_{i=1}^{\ell}(k_i - 1) \geq n.$$

Inserting the definition of k_i we get:

$$\sum_{i=1}^{\ell}(\lceil \log_2 n \rceil (S_i + \log_2(2T))) \geq n.$$

Using Equation (1) to express this in terms of $CM(P)$ gives us:

$$CM(P)/H + \ell \log_2(2T) \geq \frac{n}{\lceil \log_2 n \rceil}$$

or

$$CM(P) + T \log_2(2T) \geq \frac{n \lfloor \frac{\alpha}{2} n \rfloor}{\lceil \log_2 n \rceil} \geq \frac{\alpha n^2}{3 \log_2 n}.$$

Thus at least one of $T \log_2(2T)$ or $CM(P)$ is at least $\alpha n^2/(6 \log_2 n)$, as required, since $\log T$ is $O(\log n)$ wlog. The bound for random-access machines comes from observing that such a machine requires at least one memory cell of $\Omega(\log T)$ bits at every time step.

To prove the bound for algorithms with success probability n^{-c}, we multiply $\log_2(2T)$ in the above argument by $(c+1)$. Since any sorting algorithm must have $T \geq n$, on randomly chosen inputs the probability that it produces at least $\sum_{i=1}^{\ell}(k_i - 1)$ correct outputs becomes $\frac{1}{2n^c} < \frac{1}{n^c}$ and hence the above bounds (reduced by the constant factor $c+1$) apply to deterministic algorithms with success probability $1/n^c$. By Yao's lemma this implies the same lower bound for randomized algorithms with success probability n^{-c}. □

Theorem 3.3 applies to cumulative working memory of any algorithm that produces its sorted output in a write-only output vector and can compute those values in arbitrary time order. If the algorithm is constrained to produce its sorted output in the natural time order then, following [Bea91], one can obtain a slightly stronger bound.

Theorem 3.4. *Any branching program P computing the outputs of $Sort_{n,n}$ in order in time T and probability at least $4/5$ requires T to be $\Omega(n^2/\log n)$ or $CM(P)$ to be $\Omega(n^2)$. Further, any random access machine computng $Sort_{n,n}$ in order with probability at least $4/5$ requires cumulative memory $\Omega(n^2)$.*

Proof Sketch. Any such algorithm can easily determine all the elements of the input that occur uniquely and the lower bounds follow from the bounds on Unique Elements that we prove in Section 6. □

4 Quantum cumulative memory complexity of sorting

We now show with a similar argument that the quantum cumulative memory complexity of sorting is $\Omega(n^3/T)$, matching the ST complexity bounds given in [KŠdW07, HM21]. This involves the quantum circuit model which, as we have noted, produces each output position at a predetermined input-independent layer. We restrict our attention to circuits that output all elements in the input according to their sorted order with a constant total success probability. While our proof is inspired by the time-space lower bound of [KŠdW07], it can be easily adapted to follow the proof in [HM21] instead.

Definition 4.1. In the *k-threshold problem* we receive an input $X = x_1, \ldots, x_n$ where $x_i \in \{0,1\}$. We want to accept iff there are at least k distinct values for i where $x_i = 1$.

We say that a quantum circuit $\mathcal{C}$ that computes a boolean function $f : \{0,1\}^n \to \{0,1\}$ has completeness a and soundness b on inputs in domain D iff for all $x \in D$, $\Pr[\mathcal{C}(x) = 1] \geq a$ when $f(x) = 1$ and $\Pr[\mathcal{C}(x) = 1] \leq b$ when $f(x) = 0$. We say that a circuit has perfect completeness (soundness) iff $a = 1$ (respectively, $b = 0$).

Proposition 4.2 (Theorem 13 in [KŠdW07])**.** *For every $\gamma > 0$ there is an $\alpha > 0$ such that any quantum k-threshold circuit with at most $T \leq \alpha\sqrt{kn}$ queries and with perfect soundness must have completeness $\sigma \leq e^{-\gamma k}$ on inputs with hamming weight k.*

Using the above theorem, we present a generalization of a lemma first proven in [KŠdW07].

Lemma 4.3. *Choose any constant $\gamma > 0$. Let n be sufficiently large and $\mathcal{C}(X)$ be a quantum circuit with input $X = x_1, \ldots, x_n$. There exists a constant β that depends only on γ such that for all $k \leq \beta^2 n$ and*

$R \subseteq \{n/2+1, \ldots, n\}$ where $|R| = k$, if $\mathcal{C}(X)$ makes at most $\beta\sqrt{kn}$ queries, then the probability that $\mathcal{C}(X)$ can correctly output all k pairs (x_i, r_j) where $r_j \in R$ and x_i is the r_j'th smallest element of X is at most $e^{(1-\gamma)k-1}$. If R is a contiguous set of integers, then the probability is at most $e^{-\gamma k}$.

The version of Lemma 4.3 proved in [KŠdW07] had the additional assumption that the set of output ranks R is a contiguous set of integers; this was sufficient to show that any quantum circuit that produces its sorted output in sorted time order requires that T^2S is $\Omega(n^3)$. The authors stated that their proof can be generalized to any fixed rank ordering, but the generalization is not obvious. We generalize their lemma to non-contiguous R, which is sufficient to obtain an $\Omega(n^3/T)$ lower bound on the cumulative complexity of sorting independent of the time order in which the sorted output is produced.

Proof of Lemma 4.3. Choose α as the constant for γ in Proposition 4.2 and let $\beta = \sqrt{2}\alpha/6$. Let $\mathcal{C}$ be a circuit with at most $\beta\sqrt{kn}$ layers that outputs the k correct pairs (x_i, r_j) with probability p. Let $R = \{r_1, \ldots r_k\}$ where $r_1 < r_2 < \ldots < r_k$. We describe our construction of a circuit $\mathcal{C}'(X)$ solving the k-threshold problem on inputs $X = x_1, \ldots, x_{n/2}$ with exactly k ones in terms of a function $f : [n/2] \to R$. Given f, we re-interpret the input as follows: we replace each x_i with $x'_i = f(i)x_i$, add k dummy values of 0, and add one dummy value of j for each $j \in \{n/2+1, \ldots, n\} \setminus R$. Doing this gives us an input $X' = x'_1, \ldots, x'_n$ that has $n/2$ zeroes. If we assume that f is 1-1 on the k ones of X, then the image of the ones of X will be R and there will be precisely one element of X' for each $j \in \{n/2+1, \ldots, n\}$. Therefore the element of rank $j > n/2$ in X' will have value j, and hence the rank $r_1, \ldots, r_k$ elements of X' will be the images of precisely those elements of X with $x_i = 1$.

To obtain perfect soundness, we cannot rely on the output of $\mathcal{C}(X')$ and must be able to check that each of the output ranks was truly mapped to by a distinct one of X. For each element x_i of X we simply append its index i as $\log_2 n$ low order bits to its image x'_i and append an all-zero bit-vector of length $\log_2 n$ to each dummy value to obtain input X''. Doing so will not change the ranks of the elements in X', but will allow recovery of the k indices that should be the ones in X. In particular, circuit $\mathcal{C}'(X)$ will run $\mathcal{C}(X'')$ and then for each output x''_j with low order bits i, $\mathcal{C}'(X)$ will query x_i, accepting if and only if all of those $x_i = 1$. More precisely, since the mapping from each x_i to the corresponding x''_i is only a function of f, x_i, and i, as long as $\mathcal{C}'(X)$ has an explicit representation of f, it can simulate each query of $\mathcal{C}(X'')$ with two oracle queries to X (see Section 2 for details). Since $\mathcal{C}'$ has at most

$$2\beta\sqrt{kn} + k \leq 3\beta\sqrt{kn} \leq \alpha\sqrt{kn/2}$$

layers, by Proposition 4.2, it can only accept with probability at most $e^{-\gamma k}$ when the input has k ones.

We now observe that for each fixed X with exactly k ones, for a randomly chosen function $f : [n/2] \to R$, the probability that f is 1-1 on the ones of X' is exactly $k!/k^k \geq e^{1-k}$. Therefore $\mathcal{C}'(X)$ will give the indices of the k ones in X with probability[3] at least $p \cdot e^{1-k}$. However, this probability must be at most $e^{-\gamma k}$, so we can conclude that $p \leq e^{(1-\gamma)k-1}$. In the event that R is a contiguous set of integers, observe that any choice for the function f will make X'' have the ones of X become ranks $r_1, \ldots, r_k$. So the probability of finding the ones is at least $p \leq e^{-\gamma k}$. ☐

[3] Note that though this is exponentially small in k it is still sufficiently large compared to the completeness required in the lower bound for the k-threshold problem.

By setting k and γ appropriately, Lemma 4.3 gives a useful upper bound on the number of fixed ranks successfully output by any $\beta\sqrt{Sn}$ query quantum circuit that has access to S qubits of input dependent initial state. To handle input-dependent initial state, we will need the following proposition.

Proposition 4.4 ([Aar05]). *Let $\mathcal{C}$ be a quantum circuit, ρ be any S qubit (possibly mixed) state, and I be the S qubit maximally mixed state. If $\mathcal{C}$ with initial state ρ produces some output $\mathcal{O}$ with probability p, then $\mathcal{C}$ with initial state I produces $\mathcal{O}$ with probability at least $p/2^{2S}$.*

This allows us to bound the overall progress made by any short quantum circuit.

Lemma 4.5. *Let $\gamma = 1 + \ln(4)$ and β be the constant from Lemma 4.3 that depending on γ. Then for any fixed set of $S \leq \beta^2 n$ ranks that are greater than $n/2$, the probability that any quantum circuit $\mathcal{C}$ with at most $\beta\sqrt{Sn}$ queries and S qubits of input-dependent initial state correctly produces the outputs for these S ranks is at most $1/e$.*

Proof. Applying Proposition 4.4 to the bound in Lemma 4.3 gives us that a quantum circuit with S qubits of input-dependent state can produce a fixed set of $k \leq \beta^2 n$ outputs larger than median with a probability at most $2^{2S}e^{(1-\gamma)k-1}$. Since $\gamma = 1 + \ln(4)$ setting $k = S$ yields a probability bound on of at most $1/e$ on the event in question. ☐

Theorem 4.6. *When n is sufficiently large, any quantum circuit $\mathcal{C}$ for sorting a list of length n with success probability at least $1/e$ and at most T layers that produces its sorted outputs in any fixed time order requires cumulative memory that is $\Omega(n^3/T)$.*

Proof. We partition $\mathcal{C}$ into blocks with large cumulative memory that can only produce a small number of outputs. We achieve this by starting at last unpartitioned layer and finding a suitably low space layer before it so that we can apply Lemma 4.5 to upper bound the number of correct outputs that can be produced in that block with a success probability of at least $1/e$. Let β be the constant from Lemma 4.5 and $k^*(t)$ be the least non-negative integer value of k such that the interval:

$$I(k,t) = \left[t - \frac{\beta}{2}(2^{k+1}-1)\sqrt{n}, t - \frac{\beta}{2}(2^k - 1)\sqrt{n}\right]$$

contains some t' such that $S_{t'} \leq 4^k - 1$. We recursively define our blocks as follows. Let ℓ be the number of blocks generated by this method. The final block $\mathcal{C}_\ell$ starts with the first layer $t_{\ell-1} \in I(k^*(T), T)$ where $S_{t_{\ell-1}} \leq 4^{k^*(T)} - 1$ and ends with layer $t_\ell = T$. Let t_i be the first layer of block $\mathcal{C}_{i+1}$. Then the block $\mathcal{C}_i$ starts with the first layer $t_{i-1} \in I(k^*(t_i), t_i)$ where $S_{t_{i-1}} \leq 4^{k^*(t_i)} - 1$ and ends with t_i. See Figure 4 for an illustration of our partitioning. Since $S_0 = 0$ we know that $k^*(t) \leq \log(T)$. Likewise since $S_t > 0$ when $t > 0$, for all $t > \frac{\beta}{2}\sqrt{n}$ we know that $0 < k^*(t) \leq \log(T)$.

By construction, block $\mathcal{C}_i$ starts with less than $4^{k^*(t_i)}$ qubits of initial state and has length at most $\beta 2^{k^*(t_i)}\sqrt{n}$; so by Lemma 4.5, if $4^{k^*(t_i)} \leq \beta^2 n$, the block $\mathcal{C}_i$ can output at most $4^{k^*(t_i)}$ inputs with failure probability at most $1/e$. Additionally $\mathcal{C}_i$ has at least $\frac{\beta}{2}2^{k^*(t_i)-1}\sqrt{n}$ layers that each have at least $4^{k^*(t_i)-1}$ qubits,[4] so the cumulative memory of $\mathcal{C}_i$ is at least $\frac{\beta}{2}2^{3k^*(t_i)-3}\sqrt{n}$.

[4]This may not hold for $\mathcal{C}_1$ with length less than $\frac{\beta}{2}\sqrt{N}$, but Lemma 4.3 with Appendix C give us that this number of layers is insufficient to find a fixed rank input with probability at least $1/e$. Thus we can omit such a block from our analysis.

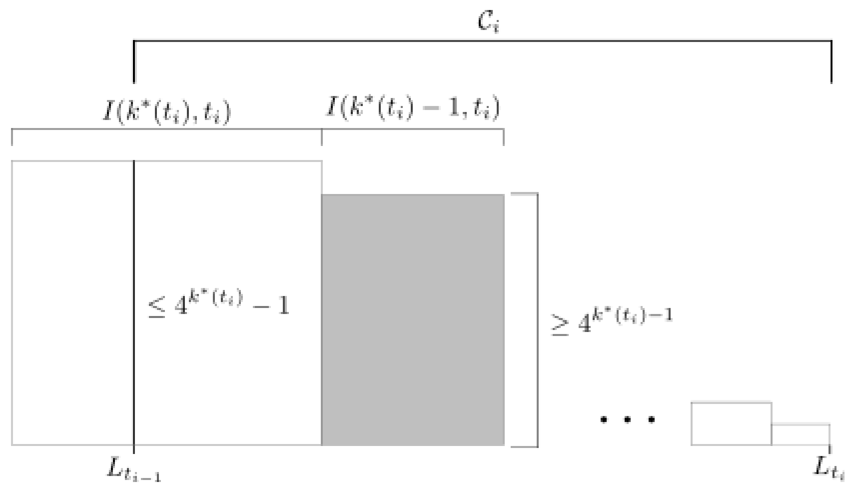

Figure 4: How we define the block $\mathcal{C}_i$ that ends at layer L_{t_i}. The grey layers are the ones used to lower bound the cumulative memory complexity of $\mathcal{C}_i$, as each of these layers uses at least $4^{k^*(t_i)-1}$ qubits and the length of this interval is $\frac{\beta}{2}2^{k^*(t_i)-1}\sqrt{n}$.

We now have two possibilities. If we have some i such that $4^{k^*(t_i)} > \beta^2 n$, the cumulative memory of $\mathcal{C}_i$ alone is at least $\beta^4 n^2/16$ which is $\Omega(n^2)$ and hence $\mathcal{C}$ has cumulatively memory $\Omega(n^3/T)$ since $T \geq n$. Otherwise, since we require that the algorithm is correct with probability at least $1/e$, each block $\mathcal{C}_i$ can produce at most $4^{k^*(t_i)}$ outputs. Since our circuit must output all $n/2$ elements larger than the median, we know $\sum_{i=1}^{\ell} 4^{k^*(t_i)} \geq n/2$. For convenience we define $w_i = 2^{k^*(t_i)}$ and get the following bound on the sum of the w_i^2:

$$\sum_{i=1}^{\ell} w_i^2 \geq n/2$$

We obtain the following lower bound on the cumulative memory:

$$CM(\mathcal{C}) \geq \sum_{i=1}^{\ell} \frac{\beta}{2} 2^{3k^*(t_i)-3}\sqrt{n} = \frac{\beta}{16}\sqrt{n}\sum_{i=1}^{\ell} w_i^3 \tag{2}$$

To lower bound the cumulative complexity, this gives us the non-convex optimization problem in Figure 5. In Appendix A we prove that if $\sum x_i \leq \sum x_i^2$, then $\sum x_i^2 \leq \sum x_i^3$. This gives us that:

$$\sum_{i=1}^{\ell} x_i^3 \geq \xi$$

Reversing the variable substitution gives us:

$$\sum_{i=1}^{\ell} w_i^3 \geq \frac{\beta n^{5/2}}{16T}$$

Then applying Equation (2) gives us the bound:

$$CM(\mathcal{C}) \geq \frac{\beta^2 n^3}{256T}$$

(a)

$$
\begin{aligned}
CM(\mathcal{C}) \geq \min\ & \frac{\beta}{16}\sqrt{n}\sum_{i=1}^{\ell} w_i^3 \\
\text{s.t.}\ & \sum_{i=1}^{\ell} w_i^2 \geq n/2 \\
& \frac{\beta}{4}\sqrt{n}\sum_{i=1}^{\ell} w_i \leq T
\end{aligned}
$$

(b)

$$
\begin{aligned}
\min\ & \sum_{i=1}^{\ell} x_i^3 \\
\text{s.t.}\ & \sum_{i=1}^{\ell} x_i^2 \geq \xi \\
& \sum_{i=1}^{\ell} x_i \leq \xi
\end{aligned}
$$

Figure 5: The non-convex optimization problem that bounds the cumulative memory for quantum sorting. The objective function of system (a) is a lower bound on the cumulative memory complexity and system (b) is the same system after scaling the objective function and applying the variable substitutions $w_i = \frac{\beta n^{3/2}}{8T} x_i$ and $\xi = \frac{32T^2}{\beta^2 n^2}$.

And therefore the cumulative memory of $\mathcal{C}$ is $\Omega(n^3/T)$. □

To extend our results to arbitrary success probability at most $1-\delta$, it is important to know how α and γ are related in Proposition 4.2. In Appendix C we show that we can have α that is $\Omega(e^{-\gamma/2})$ and get a probability of at most $e^{-\gamma k}$. Thus for any S, k, and $\delta \in (0,1)$, we can choose

$$\gamma = \frac{\ln(2^{2S}/(1-\delta)) - 1}{k} + 1$$

to get a probability of at most $1-\delta$ for circuits with

$$\Omega\left(\left(\frac{2^{2S}}{e(1-\delta)}\right)^{1/2k}\sqrt{kn}\right)$$

layers and S qubits of advice to produce k outputs. When $S = k$ and $\delta = 1 - 1/e$, this is exactly the bound from Lemma 4.5. If we repeat the proof of Theorem 4.6 for failure probability at most δ, we can set β to a value that is $\Omega(1/\sqrt{1-\delta})$ to obtain a lower bound on the cumulative memory that is $\Omega(n^3/((1-\delta)T))$.

5 A general method for proving cumulative memory complexity lower bounds

Our method involves adapting techniques previously used to prove tradeoff lower bounds on worst-case time and worst-case space. We show that the same properties that yield lower bounds on the product of time and space in the worst case can also be used to produce nearly identical lower bounds on cumulative memory. To do so, we first revisit the standard approach to such time-space tradeoff lower bounds.

The standard method for time-space tradeoff lower bounds for multi-output functions

Consider a multi-output function f on D^n where the output $f(x)$ is either unordered (the output is simply a set of elements from R) or ordered (the output is a vector of elements from R). Then $|f(x)|$ is either the size of the set or the length of the vector of elements. The standard method for obtaining an ordinary time-space tradeoff lower bounds for multi-output functions on D-way branching programs is the following:

The part that depends on f: Choose a suitable probability distribution μ on D^n, often simply the uniform distribution on D^n and then:

(A) Prove that $\Pr_{x\sim\mu}[|f(x)| \geq m] \geq \alpha$.

(B) Prove that for all $k \leq m'$ and any branching program B of height $\leq h'(k,n)$, the probability for $x \sim \mu$ that B produces at least k correct output values of f on input x is at most $C \cdot |R|^{-k/r(n)}$ for some m', h', r and constant C independent of n.

Observe that under any distribution μ, a branching program with ordered outputs that makes no queries can produce k outputs that are all correct with probability at least $|R|^{-k}$, so the bound in (B) shows that, roughly, up to the power $1/r(n)$ there is not much gained by using a branching program of height h.

The generic completion: In the following outline we omit integer rounding for readability.

- Suppose that
$$S \leq \frac{\log_2 |R|}{r(n)} \cdot m' - \log_2(2C/\alpha). \tag{3}$$
- Let $k = [S + \log_2(2C/\alpha)] \cdot r(n)/\log_2 |R|$, which is at most m' by hypothesis on S, and define $h(S,n) = h'(k,n)$.
- Divide time T into $\ell = T/h$ blocks of length $h = h(S,n)$.
- The original branching program can be split into at most 2^S sub-branching programs of height $\leq h$, each beginning at a boundary node between layers. By property (B) and a union bound, for $x \sim \mu$ the probability that at least one of these $\leq 2^S$ sub-branching programs of height at most h produces k correct outputs on input x is at most
$$2^S \cdot C \cdot |R|^{-k/r(n)} \leq \alpha/2$$
by our choice of k.
- Under distribution μ, by (A), with probability at least α, an input $x \sim \mu$ has some block of time during which at least $m/\ell = m \cdot h(S,n)/T$ outputs of f must be produced on input x.
- If $m \cdot h(S,n)/T \leq k$, this can occur for at most an $\alpha/2$ fraction of inputs under μ. Therefore we have
$$m \cdot h(S,n)/T > k = [S + \log_2(2C/\alpha)] \cdot r(n)/\log_2 |R|$$

and hence, combining with Equation (3), we have

$$T \cdot S \geq \min\left(m \cdot h(S,n),\ m' \cdot n'\right) \cdot \frac{\log_2 |R|}{r(n)} - \log_2(C/\alpha) \cdot T$$

where $n' \leq n$ is the decision tree complexity of f and hence a lower bound on T.

Remark 5.1. Though it will not impact our argument, for many instances of the above outline, the proof of property (B) is shown for a decision tree of the same height by proving an analog for the conditional probability along each path in the decision tree separately; this will apply to the tree as a whole since the paths are followed by disjoint inputs, so property (B) follows from the alternative property below:

(B') For any partial assignment τ of $k \leq m'$ output values over R and any restriction (i.e., partial assignment) π of $h'(k,n)$ coordinates within D^n,

$$\Pr_{x \sim \mu}[f(x) \text{ is consistent with } \tau \mid x \text{ is consistent with } \pi] \leq C \cdot |R|^{-k/r(n)}.$$

Remark 5.2. The above method still gives lower bounds for many multi-output functions $g : D^N \to R^M$ that have individual output values that are easy to compute or large portions of the input space on which they are easy to compute. The bounds follow by applying the method to some subfunction f of g given by $f(x) = \Pi_O(g(x,\pi))$ where π is a partial assignment to the input coordinates and Π_O is a projection onto a subset O of output coordinates. In the subsequent discussions we ignore this issue, but the idea can be applied to all of our lower bound methods.

A general extension to cumulative memory bounds

To give a feel for the basic ideas of the method, we first show this for a simple case. Observe that, other than the separate bound on time, the lower bound on cumulative memory usage we prove in this case is asymptotically identical to the bound achieved for the product of time and worst-case space using the standard outline.

Theorem 5.3. *Let $c > 0$. Suppose that properties (A) and (B) apply for $h'(k,n) = h(n)$, $m' = m$, and $\alpha = C = 1$. If*

$$T \log_2 T \leq \frac{m \cdot h(n) \cdot \log_2 |R|}{6(c+1)r(n)}$$

then the cumulative memory used in computing $f : D^n \to R^m$ in time T with success probability at least T^{-c} is at least

$$\frac{m \cdot h(n) \cdot \log_2 |R|}{6r(n)}.$$

Proof. Fix a deterministic branching program P of length T computing f. Rather than choosing fixed blocks of height $h = h(n)$, layers of nodes at a fixed distance from each other, and a fixed target of k outputs per block, we choose the block boundaries depending on the properties of P and the target k depending on the property of the boundary layer chosen.

Let $H = \lfloor h(n)/2 \rfloor$. We break P into $\ell = \lceil T/H \rceil$ time segments of length H working backwards from step T so that the first segment may be shorter than the rest. We let $t_1 = 0$ and for $1 < i \leq \ell$ we let

$$t_i = \arg\min\{ \, |L_t| \; : \; T - (\ell - i + 1) \cdot H \leq t < T - (\ell - i) \cdot H \, \}$$

be the time step with the fewest nodes among all time steps $t \in [T - (\ell - i + 1) \cdot H, T - (\ell - i) \cdot H]$.

The i-th time block of P will be between times t_i and t_{i+1}. Observe that by construction $|t_{i+1} - t_i| \leq h(n)$ so each block has length at most $h(n)$. Set $S_i = \log_2 |L_{t_i}|$ so that L_{t_i} has at 2^{S_i} nodes. By definition of each t_i, the cumulative memory used by P,

$$CM(P) \geq \sum_{i=1}^{\ell} S_i \cdot H. \tag{4}$$

(Note that since $S_1 = 0$, it does not matter that the first segment is shorter than the rest[5].)

We now define the target k_i for the number of output values produced in each time block to be the smallest integer such that $|R|^{-k_i/r(n)} \leq 2^{-S_i}/T^{c+1}$. That is,

$$k_i = \lceil r(n) \cdot (S_i + (c+1)\log_2 T)/\log_2 |R| \rceil.$$

For $x \sim \mu$, for each $i \in [\ell]$ and each sub-branching program B rooted at some node in L_{t_i} and extending until time t_{i+1}, by our choice of k_i and property (B), if $k_i \leq m$, the probability that B produces at least k_i correct outputs on input x is at most $2^{-S_i}/T^{c+1}$. Therefore, by a union bound, for $x \sim \mu$ the probability that P produces at least k_i correct outputs in the i-th time block on input x is at most

$$|L_{t_i}| \cdot 2^{-S_i}/T^{c+1} = 1/T^{c+1}.$$

Therefore, if each $k_i \leq m$, the probability for $x \sim \mu$ that there is some i such that P produces at least k_i correct outputs on input x during the i-th block is at most $\ell/T^{c+1} < T^c$. Therefore, if each $k_i \leq m$, the probability for $x \sim \mu$ that P produces at most $\sum_{i=1}^{\ell}(k_i - 1)$ correct outputs in total on input x is $> 1 - 1/T^c$.

If each $k_i \leq m$, since P must produce m correct outputs on $x \in D^n$ with probability at least $1/T^c$, we must have $\sum_{i=1}^{\ell}(k_i - 1) \geq m$. On the other hand, if some $k_i > m$ we have the same bound. Using our definition of k_i we have

$$\sum_{i=1}^{\ell} [r(n) \cdot (S_i + (c+1)\log_2 T)]/\log_2 |R|)] \geq m$$

or

$$\sum_{i=1}^{\ell} (S_i + (c+1)\log_2 T) \geq \frac{m \cdot \log_2 |R|}{r(n)}.$$

In particular, plugging in the bound (4) on the cumulative memory and the value of ℓ, it implies that

$$CM(P)/H + (c+1)\lceil T/H \rceil \cdot \log_2 T \geq \frac{m \cdot \log_2 |R|}{r(n)}$$

[5]This simplifies some calculations and is the prime reason for starting the time segment boundaries at T rather than at 0.

or that

$$CM(P) + (c+1)T\log_2 T \geq \frac{m \cdot h(n) \cdot \log_2 |R|}{3 \cdot r(n)},$$

where the 3 on the right rather than a 2 allows us to remove the integer rounding. Therefore either

$$T\log_2 T > \frac{m \cdot h(n) \cdot \log_2 |R|}{6(c+1) \cdot r(n)}$$

or

$$CM(P) \geq \frac{m \cdot h(n) \cdot \log_2 |R|}{6r(n)},$$

which is what we wanted to show. □

In the general version of our theorem there are a number of additional complications, most especially because the branching program height limit $h(k,n)$ in property (B) *can depend on* k, the target for the number of outputs produced. This forces the lengths of the blocks and the space used at the boundaries between blocks to depend on each other in a quite delicate way. In order to discuss the impact of that dependence and state our general theorem, we need the following definition.

Definition 5.4. Given a non-decreasing function $p : \mathbb{R} \to \mathbb{R}$ with $p(1) = 1$, we define $p^{-1} : \mathbb{R} \to \mathbb{R} \cup \{\infty\}$ by $p^{-1}(R) = \min\{j \mid p(j) \geq k\}$. We also define the *loss*, $\mathcal{L}_p$, of p by

$$\mathcal{L}_p(n) = \min_{1 \leq k \leq p(n)} \frac{\sum_{j=1}^{k} p^{-1}(j)}{k \cdot p^{-1}(k)}.$$

Lemma 5.5. *The following hold for every non-decreasing function* $p : \mathbb{R} \to \mathbb{R}$ *with* $p(1) = 1$*:*

(a) $1/p(n) \leq \mathcal{L}_p(n) \leq 1$.

(b) If p is a polynomial function $p(s) = s^{1/c}$ then $\mathcal{L}_p(n) > 1/2^{c+1}$.

(c) For any $c > 1$, $\mathcal{L}_p(n) \geq \min_{1 \leq s \leq n} \frac{p(s) - p(s/c)}{cp(s)}$.

(d) We say that p is nice *if it is differentiable and there is an integer $c > 1$ such that for all x, $p'(cx) \geq p'(x)/c$. If p is nice then $\mathcal{L}_p(n)$ is $\Omega(1/\log_2 n)$. This is tight for p with $p(s) = 1 + \log_2 s$.*

We prove these technical statements in Appendix B. The following is our full general theorem.

Theorem 5.6. *Let $c > 0$. Suppose that function f defined on D^n has properties (A) and (B) with α that is $1/n^{O(1)}$ and m' that is $\omega(\log_2 n)$. For $s > 0$, define $h(s,n)$ to be $h'(k,n)$ for $k = s \cdot r(n)/\log_2 |R|$. Suppose that $h(s,n) = h_0(s)\, h_1(n)$ with $h_0(1) = 1$ and h_0 a differentiable function such that $s/h_0(s)$ is increasing and concave. Define $S^* = S^*(T,n)$ by*

$$\frac{S^*}{h_0(S^*)} = \frac{m \cdot h_1(n) \cdot \log_2 |R|}{6r(n)T}.$$

(a) *Either*

$$T \log_2(2CT^{c+1}/\alpha) > \frac{m \cdot h_1(n) \cdot \log_2 |R|}{6r(n)},$$

which implies that T *is* $\Omega(\frac{m \cdot h_1(n) \cdot \log |R|}{r(n) \log n})$, *or the cumulative memory used by a randomized branching program in computing* f *in time* T *with error* $\varepsilon \leq \alpha(1 - 1/(2T^c))$ *is at least*

$$\mathcal{L}_{h_0}(n \log_2 |D|) \cdot \min\left(m \cdot h(S^*(T,n),n),\ 3m' \cdot h'(m'/2,n)\right) \cdot \frac{\log_2 |R|}{6r(n)}.$$

(b) *Further any randomized random-access machine computing* f *in time* T *with error* $\varepsilon \leq \alpha(1 - 1/(2T^c))$ *requires cumulative memory*

$$\Omega\left(\mathcal{L}_{h_0}(n \log_2 |D|) \cdot \min\left(m \cdot h(S^*(T,n),n),\ m' \cdot h'(m'/2,n)\right) \cdot \frac{\log_2 |R|}{r(n)}\right).$$

Before we give the proof of the theorem, we note that by Lemma 5.5, in the case that h_0 is constant or a polynomial function of its input, which together account for all existing applications we are aware of, the function $\mathcal{L}_{h_0}$ is lower bounded by a constant. Further, the value S^* in the statement of this theorem is at least a constant factor times the value of S used in the generic time-space tradeoff lower bound methodology. Therefore, for example, the cumulative memory lower bound derived for random-access machines via Theorem 5.6 is close to the lower bound on the product of time and worst-case space given by standard methods.

Proof of Theorem 5.6. We prove both (a) and (b) directly for branching programs, which can model random-access machines, and will describe the small variation that occurs in the case that the branching program in question comes from a random-access machine. To prove these properties for randomized branching programs, by Yao's Lemma [Yao77] it suffices to prove the properties for deterministic branching programs that have error at most ε under distribution μ. Fix a (deterministic) branching program P of length T computing f with error at most ε under distribution μ. Without loss of generality, P has maximum space usage at most $S^{max} = n \log_2 |D|$ space since there are at most $|D^n|$ inputs.

Let $H = \lfloor h_1(n)/2 \rfloor$. We break P into $\ell = \lceil T/H \rceil$ time segments of length H working backwards from step T so that the first segment may be shorter than the rest. We then choose a sequence of *candidates* for the time steps in which to begin new blocks, as follows: We let $\tau_1 = 0$ and for $1 < i \leq \ell$ we let

$$\tau_i = \arg\min\{\ |L_t|\ :\ T - (\ell - i + 1) \cdot H \leq t < T - (\ell - i) \cdot H\ \}$$

be the time step with the fewest nodes among all time steps $t \in [T - (\ell - i + 1) \cdot H, T - (\ell - i) \cdot H]$. Set $\sigma_i = \log_2 |L_{\tau_i}|$ so that L_{τ_i} has at 2^{σ_i} nodes. This segment contributes at least $\sigma_i \cdot H$ to the cumulative memory bound of P.

To choose the beginning t_{i^*} of the last time block[6]. we find the smallest k such that $h_0(\sigma_{\ell-k+1}) < k$. Such a k must exist since h_0 is a non-decreasing non-negative function, $h_0(1) = 1$ and $\sigma_1 = 0 < 1$.

[6]Since we are working backwards from the end of the branching program and we do not know how many segments are included in each block, we don't actually know this index until things stop with $t_1 = 0$

We now observe that the length of the last block is at most $k \cdot H$ which by choice of k is less than $h(\sigma_{\ell-k+1}, n)$ and hence we have satisfied the requirements for property (B) to apply at each starting node of the last time block.

By our choice of each τ_i, the total cumulative memory used in the last k segments is at least

$$\sum_{j=1}^{k} \sigma_{\ell+1-j} \cdot H.$$

Further, since k was chosen as smallest with the above property, we know that for every $j \in [k-1]$ we have

$$h_0(\sigma_{\ell-j+1}) \geq j$$

Hence we have $\sigma_{\ell-j+1} \geq h_0^{-1}(j)$ and we get a cumulative memory bound for the last k segments of at least

$$(\sigma_{\ell-k+1} + \sum_{j=1}^{k-1} h_0^{-1}(j)) \cdot H. \tag{5}$$

Claim: $\sigma_{\ell-k+1} + \sum_{j=1}^{k-1} h_0^{-1}(j) \geq \mathcal{L}_{h_0}(S^{max}) \cdot \sigma_{\ell-k+1} \cdot k$.

Proof of Claim. Observe that it suffices to prove the claim when we replace $\sigma_{\ell-k+1}$, which appears on both sides, by a larger quantity. In particular, we show how to prove the claim with $h_0^{-1}(k)$ instead, which is larger since $h_0(\sigma_{\ell-k+1}) < k$. But this follows immediately since by definition

$$\mathcal{L}_{h_0}(S^{max}) \leq \frac{\sum_{j=1}^{k} h_0^{-1}(j)}{k \cdot h_0^{-1}(k)},$$

which is equivalent to what we want to prove. □

Write $S_{i^*} = \sigma_{\ell-k+1}$. By the claim, the cumulative memory contribution associated with the last block beginning at t_{i^*} is at least

$$\mathcal{L}_{h_0}(S^{max}) \cdot S_{i^*} \cdot h_0(S_{i^*})H.$$

We repeat this in turn to find the time step for the beginning of the next block from the end, t_{i^*-1}. One small difference now is that there is a last partial segment of height at most H from the beginning of segment containing t_{i^*} to layer t_{i^*}. However, this only adds at most $h_1(n)/2$ to the length of the segment which still remains well within the height bound of $h(S_{i^*-1}, n) = h_0(S_{i^*-1})h_1(n)$ for property (B) to apply.

Repeating this back to the beginning of the branching program we obtain a decomposition of the branching program into some number i^* of blocks, the i-th block beginning at time step t_i with 2^{S_i} nodes, height between $h_0(S_i)H$ and $h_0(S_i)H + H \leq 2h_0(S_i)H$, and with an associated cumulative memory contribution in the i-th block of at least

$$\mathcal{L}_{h_0}(S^{max}) \cdot S_i \cdot h_0(S_i)H.$$

(This is correct even for the partial block starting at time $t_1 = 0$ since $S_1 = 0$.) Since we know that $i^* \le \ell$, for convenience, we also define $S_i = 0$ for $i^* + 1 \le i \le \ell$. Then, by definition we have

$$CM(P) \ge \mathcal{L}_{h_0}(S^{max}) \cdot \left(\sum_{i=1}^{i^*} S_i \cdot h_0(S_i) \right) \cdot H = \mathcal{L}_{h_0}(S^{max}) \cdot \left(\sum_{i=1}^{\ell} S_i \cdot h_0(S_i) \right). \tag{6}$$

and

$$\sum_{i=1}^{\ell} h_0(S_i) \le T/H. \tag{7}$$

As in the previous argument for the simple case, for $i \le i^*$, we define the target k_i for the number of output values produced in each time block to be the smallest integer such that $C|R|^{-k_i/r(n)} \le 2^{-S_i}\alpha/(2T^{c+1})$. That is,

$$k_i = \lceil r(n) \cdot (S_i + \log_2(2CT^{c+1}/\alpha))/\log_2 |R| \rceil.$$

If $k_i > m'$ for some i, then $S_i \ge m' \cdot \log_2 |R|/r(n) - \log_2(2CT^{c+1}/\alpha) \ge m' \cdot \log_2 |R|/(2r(n))$ since m' is $\omega(\log n)$ and $1/\alpha$ and T are $n^{O(1)}$. Therefore $h_0(S_i) \ge h'(m'/2, n)$ and hence

$$CM(P) \ge \mathcal{L}_{h_0}(S^{max}) \cdot m' \cdot h'(m'/2, n) \cdot \frac{\log_2 |R|}{2r(n)}$$

Suppose instead that $k_i \le m'$ for all $i \le i^*$. Then, for $x \sim \mu$, for each $i \in [i^*]$ and each sub-branching program B rooted at some node in L_{t_i} and extending until time t_{i+1}, by our choice of k_i and property (B), the probability that B produces at least k_i correct outputs on input x is at most

$$\alpha \cdot 2^{-S_i}/(2T^{c+1}).$$

Therefore, by a union bound, for $x \sim \mu$ the probability that P produces at least k_i correct outputs in the i-th time block on input x is at most

$$|L_{t_i}| \cdot \alpha \cdot 2^{-S_i}/(2T^{c+1}) = \alpha/(2T^{c+1})$$

and hence the probability for $x \sim \mu$ that there is some i such that P produces at least k_i correct outputs on input x during the i-th block is at most $\ell \cdot \alpha/(2T^{c+1}) < \alpha/(2T^c)$. Therefore, the probability for $x \sim \mu$ that P produces at most $\sum_{i=1}^{\ell}(k_i - 1)$ correct outputs in total on input x is $> 1 - \alpha/(2T^c)$.

Since, by property (A) and the maximum error it allows, P must produce at least m correct outputs with probability at least $\alpha - \epsilon \ge \alpha - \alpha(1 - 1/(2T^c)) = \alpha/(2T^c)$ for $x \sim \mu$, we must have $\sum_{i=1}^{i^*}(k_i - 1) \ge m$. Using our definition of k_i we have

$$\sum_{i=1}^{i^*} [r(n) \cdot (S_i + \log_2(2CT^{c+1}/\alpha))]/\log_2 |R|)] \ge m$$

or

$$\sum_{i=1}^{i^*} (S_i + \log_2(2CT^{c+1}/\alpha)) \ge \frac{m \cdot \log_2 |R|}{r(n)}.$$

This is the one place in the proof where there is a distinction between an arbitrary branching program and one that comes from a random access machine.

We first start with the case of arbitrary branching programs: Note that $i^* \leq \ell = \lceil T/H \rceil = \lceil T/\lfloor h_1(n)/2 \rfloor \rceil$. Suppose that $T \log_2(2CT^{c+1}/\alpha) \leq \frac{m \cdot h_1(n) \cdot \log_2 |R|}{6r(n)}$. Then, even with rounding, we obtain $\sum_{i=1}^{i^*} S_i \geq \frac{m \cdot \log_2 |R|}{2r(n)}$.

Unlike an arbitrary branching program that may do non-trivial computation with sub-logarithmic S_i, a random-access machine with even one register requires at least $\log_2 n$ bits of memory (just to index the input for example) and hence $S_i + \log_2(2CT^{c+1}/\alpha)$ will be $O(S_i)$, since T is at most polynomial in n without loss of generality and $1/\alpha$ is at most polynomial in n by assumption. Therefore we obtain that $\sum_{i=1}^{i^*} S_i$ is $\Omega(\frac{m \cdot \log_2 |R|}{r(n)})$ without the assumption on T.

In the remainder we continue the argument for the case of arbitrary branching programs and track the constants involved. The same argument obviously applies for programs coming from random-access machines with slightly different constants that we will not track. In particular, since $S_i = 0$ for $i > i^*$ we have

$$\sum_{i=1}^{\ell} S_i \geq \frac{m \cdot \log_2 |R|}{2r(n)}. \tag{8}$$

From this point we need to do something different from the argument in the simple case because the lower bound on the total cumulative memory contribution is given by Equation (6) and is not simply $\sum_{i=1}^{\ell} S_i \cdot H$. Instead, we combine Equation (8) and Equation (7) using the following technical lemma that we prove in Appendix A.

Lemma 5.7. *Let $p : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ be a differentiable function such that $q(x) = x/p(x)$ is a concave increasing function of x. For $x_1, x_2, \ldots \in \mathbb{R}^{\geq 0}$, if $\sum_i x_i \geq K$ and $\sum_i p(x_i) \leq L$ then $\sum_i x_i p(x_i) \geq q^{-1}(K/L) \cdot L$.*

In our application of the lemma $p = h_0$, $K = \frac{m \cdot \log_2 |R|}{2r(n)}$, and $L = T/H$. Let S^* be the solution to

$$\frac{S^*}{h_0(S^*)} = K/L = \frac{m \cdot H \cdot \log_2 |R|}{2r(n)T} \geq \frac{m \cdot h_1(n) \log_2 |R|}{6r(n)T}.$$

Then Lemma 5.7 implies that

$$\sum_{i=1}^{\ell} S_i \cdot h_0(S_i) \geq S^* \cdot T/H = \frac{m \cdot h_0(S^*) \cdot \log_2 |R|}{2r(n)}.$$

and hence

$$CM(P) \geq \mathcal{L}_{h_0}(S^{max}) \cdot \frac{m \cdot h_0(S^*) \cdot H \cdot \log_2 |R|}{2r(n)} \geq \mathcal{L}_{h_0}(S^{max}) \cdot \frac{m \cdot h(S^*, n) \cdot \log_2 |R|}{6 \cdot r(n)}$$

since $H = \lfloor h_1(n)/2 \rfloor$ and $h(S^*, n) = h_0(S^*) \cdot h_1(n)$. □

By Lemma 5.5, in the case that h_0 is a polynomial function of its input, the function $\mathcal{L}_{h_0}$ is lower bounded by a constant and the bound in Theorem 5.6 only loses a constant factor in moving from the product of worst-case space and time to cumulative memory complexity. In that special case (and indeed for any nice function h_0), there is an alternative variant of the above in which one breaks up time into exponentially growing segments starting with time step T. We use that alternative approach to obtain lower bounds on the cumulative memory complexity of sorting by quantum algorithms in Section 4.

Remark 5.8. If we restrict our attention to $o(\frac{m' \log |R|}{r(n)})$-space bounded computation, then each $k_i \le m'$ and the cumulative memory bound for a branching program in Theorem 5.6 becomes

$$\mathcal{L}_{h_0}(n \log_2 |D|) \cdot m \cdot h(S^*(T,n),n) \cdot \frac{\log_2 |R|}{6r(n)}.$$

And the bound for RAM cumulative memory becomes

$$\Omega\left(\mathcal{L}_{h_0}(n \log_2 |D|) \cdot m \cdot h(S^*(T,n),n) \cdot \frac{\log_2 |R|}{r(n)}\right).$$

6 Sample applications to cumulative complexity of classical algorithms

Theorems 5.3 and 5.6 are powerful tools that can convert most existing time-space lower bounds into asymptotically equivalent lower bounds on the required cumulative memory. We give a few examples to indicate how our general theorems can be used.

Unique elements

Define $Unique_{n,N} : [N]^n \to \mathcal{P}([N])$ by $Unique_{n,N}(x) = \{ x_i \mid x_j \ne x_i \text{ for all } j \ne i \}$.

Proposition 6.1 (Lemmas 2 and 3 in [Bea91]). *For the uniform distribution μ on $[N]^n$ with $N \ge n$,*

(A) $\Pr_{x\sim\mu}[|Unique_{n,N}(x)| \ge n/(2e)] \ge 1/(2e-1)$

(B′) For any partial assignment τ of $k \le n/4$ output values over $[N]$ and any restriction π of $n/4$ coordinates in $[n]^n$, $\Pr_{x\sim\mu}[Unique_{n,N}(x)$ is consistent with $\tau \mid x$ is consistent with $\pi] \le e^{-k/2}$.

The above lemma is sufficient to prove that TS is $\Omega(n^2)$ for the unique elements problem, and can be easily extended to a cumulative complexity bound using Theorem 5.6.

Theorem 6.2. *For $n \ge N$, any branching program computing $Unique_{n,N}$ in time T and probability at least $4/5$ requires T to be $\Omega(n^2/\log n)$ or $CM(P)$ to be $\Omega(n^2)$. Further, any random access machine computing $Unique_{n,N}$ with probability at least $4/5$ requires cumulative memory $\Omega(n^2)$*

Proof. By Proposition 6.1, $Unique_{n,N}$ satisfies conditions (A) and (B) of Section 5 with $h'(k,n) = n/4$, $m' = n/4$, $m = n/(2e)$, $C = 1$, $r(n) = 2\ln N$ and $\alpha = 1/(2e-1) \ge 0.2254$. Since $h'(k,n)$ is independent of k, the function h_0 defined in Theorem 5.6 is the constant function 1 and $h_1(n) = n/4$ so $\mathcal{L}_{h_0} \equiv 1$. We then apply Theorem 5.6 to obtain the claimed lower bounds. □

The above theorem is tight for $N = n$ using the algorithm in [Bea91].

Linear Algebra

We consider linear algebra over some finite field $\mathbb{F}$. Let D be a subset of $\mathbb{F}$ with d elements.

Definition 6.3. An $m \times n$ matrix is (g, h, c)*-rigid* iff every $k \times w$ submatrix where $k \leq g$ and $w \geq n - h$ has rank at least ck. We call $(g, h, 1)$-rigid matrices (g, h)-rigid.

Matrix rigidity is a robust notion of rank and is an important property for proving time-space and cumulative complexity lower bounds for linear algebra. Fortunately, Abrahamson proved that there are always rigid square matrices.

Proposition 6.4 (Lemma 4.3 in [Abr91]). *There is a constant $\gamma \in (0, \frac{1}{2})$ where at least a $1 - d^{-1}(2/3)^{\gamma n}$ fraction of the matrices over $D^{n\times n}$ are $(\gamma n, \gamma n)$-rigid.*

Abrahamson shows in [Abr91] that for any constant $c \in (0, \frac{1}{2})$ and $m \times n$ matrix A that is (cm, cn, c)-rigid, any D-way branching program that computes the function $f(x) = Ax$ with expected time $\overline{T} \geq n$ and expected space[7] $\overline{S}$ has $\overline{TS} = \Omega(nm \log d)$ where $d = |D|$. We restate the key property used in that proof.

Proposition 6.5 (Theorem 4.6 in [Abr91]). *Let $c \in (0, \frac{1}{2}]$, A be any $m \times n$ matrix that is (g, h, c)-rigid and f be the function $f(x) = Ax$ over $\mathbb{F}$. Let μ be the uniform distribution on D^n for $D \subseteq \mathbb{F}$ with $|D| = d$. For any restriction π of h coordinates to values in D and any partial assignment τ of $k \leq g$ output coordinates over $\mathbb{F}^m$,*

$$\Pr_{x\sim\mu}[f(x) \text{ is consistent with } \tau \mid x \text{ is consistent with } \pi] \leq d^{-ck}$$

Theorem 6.6. *Let $c \in (0, \frac{1}{2}]$. Let A be an $m \times n$ matrix over D, with $|D| = d$ that is $(g(m), h(n), c)$-rigid. Then, for any D-way branching program P computing $f(x) = Ax$ in T steps with probability at least $n^{-O(1)}$, either T is $\Omega(g(m)h(n)\log_n d)$ or $CM(P)$ is $\Omega(g(m)h(n)\log d)$. Further, computing f on a random access machine requires cumulative memory $\Omega(g(m)h(n)\log d)$ unconditionally.*

Proof. We invoke Theorem 5.3 using Proposition 6.5 to obtain condition (B′). Condition (A) is trivial since $|f(x)| = m$. □

By Proposition 6.4 we know that for some constant γ, a random matrix has a good chance of being $(\gamma m, \gamma n)$-rigid. This means that computing $f(x) = Ax$ for a random matrix A in time at most T is likely to require either the cumulative memory or $T \log T$ to be $\Omega(mn \log d)$. Since Yesha [Yes84] proved that the $n \times n$ DFT matrix is $(n/4, n/4, 1/2)$-rigid, the DFT is a concrete example where the cumulative memory or $T \log T$ is $\Omega(n^2 \log d)$; other examples include generalized Fourier transform matrices over finite fields [BJS01, Lemma 28].

Corollary 6.7. *If A is an $n \times n$ generalized Fourier transform matrix over field $\mathbb{F}$ with characteristic relatively prime to n then any random-access machine computing $f(x) = Ax$ for $x \in D^n$ where $D \subseteq \mathbb{F}$ has $|D| = d$ with probability at least $n^{-O(1)}$ requires cumulative memory that is $\Omega(n^2 \log d)$.*

It is easy to see that our lower bound is asymptotically optimal in these cases.

[7] [Abr91] defines expected space as the expected value of the $\log_2$ of the largest number of a branching program node that is visited during a computation under best case node numbering.

Proposition 6.8 (Theorem 7.1 in [Abr91]). *Let $f : D^{2n^2} \to \mathbb{F}^{n^2}$ for $D \subseteq \mathbb{F}$ and $d = |D|$ be the matrix multiplication function, γ be the constant from Proposition 6.4, and μ be the uniform distribution over $(\gamma m, \gamma n)$-rigid matrices. Choose any integers h and k such that $2(h/\gamma n)^2 \leq k$. If $\gamma n \geq 1$ then for any D-way branching program B of height $\leq h$ the probability that B produces at least k correct output values of f is at most $d^{2-\gamma k/4}$.*

Theorem 6.9. *Multiplying two random matrices in D^{n^2} with $D \subseteq \mathbb{F}$ and $d = |D|$ with probability at least $n^{-O(1)}$ requires time T that is $\Omega((n^3\sqrt{\log d})/\log n)$ or cumulative memory $\Omega((n^6 \log d)/T)$. On random access machines, the cumulative memory bound is unconditional.*

Proof. Proposition 6.8 lets us apply Theorem 5.6 with $m = n^2$, $h'(k,n) = \gamma n\sqrt{k/2}$, $C = d^2$, $\alpha = 1$, $|R| = |\mathbb{F}|$, and $r(n) = (4\log_d |\mathbb{F}|)/\gamma$. This gives us that $h(s,n) = n\sqrt{2\gamma s/\log_2 d}$, so $h_0(s) = \sqrt{s}$. Then we get that $\sqrt{S^*} = \frac{mn\sqrt{2\gamma/\log_2 d}\cdot\log_2|\mathbb{F}|}{6r(n)T}$ and hence

$$S^* \text{ is } \Omega\left(\frac{n^6 \log d}{T^2}\right).$$

Therefore we get that either

$$T \text{ is } \Omega\left(\frac{n^3 \log^{1/2} d}{\log n}\right)$$

or, since the loss function for h_0 is a constant, the cumulative memory is

$$\Omega\left(\min\left((n^6 \log d)/T, n^5 \log^{1/2} d\right)\right).$$

Since the decision tree complexity of matrix multiplication is $\Omega(n^2)$, this is $\Omega((n^6 \log d)/T)$. For random access machines, the same cumulative memory bound applies without the condition on T. ☐

7 Cumulative memory complexity of single-output functions

The time-space tradeoff lower bounds known for classical algorithms computing single-output functions are quite a bit weaker than those for multi-output functions, but the bounds we can obtain on cumulative memory for slightly super-linear time bounds are nearly as strong as those for multi-output functions.

For simplicity we focus on branching programs with Boolean output, in which case, we can simply assume that the output is determined by which of two nodes the branching program reaches at time step T.

The general method for bounds for single output functions is based on the notion of the *trace* of a branching program computation. We fix a branching program P computing $f : D^n \to \{0,1\}$. As in the case of the simple bounds for multi-output functions, we break up P into a sequence of blocks, say ℓ of them, that are separated by time steps $0 = t_1, \ldots, t_\ell, t_{\ell+1} = T$. A trace τ in P is a sequence of ℓ nodes of P, one node in the set of nodes L_{t_i} at time step t_i for each $i = 1, \ldots, \ell$. The set of all traces $\mathcal{T} = L_{t_1} \times \cdots \times L_{t_\ell}$.

A key object under consideration is the notion of an *embedded rectangle*, which is a subset of $R \subseteq D^n$ with associated disjoint subsets $A \subset [n]$ and $B \subset [n]$ with $|A| = |B| = m(R) = m$ and assignment $\sigma \in D^{[n]-A-B}$ such that $R = R_A \times R_B \times \sigma$. We write $\alpha(R) = \min(|R_A|, |R_B|)/|D|^m$.

Proposition 7.1 (Implicit in Corollary 5.2 of [BSSV03]). *Let P be a branching program of length T computing a function $f : D^n \to \{0,1\}$. Suppose that $T \leq kn$ for $k \geq 4$ and $n \geq \ell \geq k^2 2^{k+6}$. If $0 = t_1 < t_2 < \cdots < t_{\ell+1} = T$ are time steps with $t_{i+1} - t_i \leq n/(k2^{k+6})$, then there is an embedded rectangle $R \subseteq f^{-1}(1)$ with $m(R) = m \geq n/2^{k+1}$ and $\alpha(R) \geq 2^{-12(k+1)m-2} \cdot |\mathcal{T}|^{-1} \cdot |f^{-1}(1)|/|D|^n$ where $\mathcal{T}$ is the set of traces of P associated with time steps $t_1, \ldots, t_\ell$.*

Corollary 7.2. *Let P be a D-way branching program of length T computing a function $f : D^n \to \{0,1\}$. If $T \leq kn$ for $k \geq 4$ and $n \geq k^2 2^{k+8}$, then there is an embedded rectangle $R \subseteq f^{-1}(1)$ with $m(R) = m \geq n/2^{k+1}$ and $\alpha(R) \geq 2^{-12(k+2)m - k\cdot 2^{k+9}\cdot CM(P)/n - 2} \cdot |f^{-1}(1)|/|D^n|$.*

Proof. Fix a branching program P of length $T \leq kn$ computing f. We can extend P to length exactly kn by adding a chain of nodes to the root. This does not impact the cumulative memory bound of P – a single node per level is 0 space – so we assume that $T = kn$ without loss of generality. Let $\ell = k^2 2^{k+8}$. We apply the same basic idea for the choice of time steps $0 = t_1, t_1, \ldots, t_{\ell+1} = T$ used in the simple general method for multi-output functions: Namely, we break P into ℓ time segments of length either $h = \lfloor kn/\ell \rfloor$ or $\lceil kn/\ell \rceil$. We define $t_1 = 0$ and define t_i for $1 < i \leq \ell$ to be the time step during the next segment at which the set $|L_{t_i}|$ is minimized. Write $S_i = \log_2 |L_{t_i}|$. Then the cumulative memory complexity used by P satisfies

$$CM(P) \geq \sum_{i=1}^{\ell} S_i \cdot h = h \cdot \log_2 |\mathcal{T}|,$$

since $|\mathcal{T}| = \prod_{i=1}^{t} |L_{t_i}|$.

Clearly each $t_{i+1} - t_i$ is at most $2\lceil kn/\ell \rceil \leq n/(k2^{k+6})$ by definition, since their difference is at most the length of two consecutive time segments. Therefore, the conditions of Proposition 7.1 apply and we obtain that there is an embedded rectangle $R \subseteq f^{-1}(1)$ with $m(R) \geq n/2^{k+1}$ and

$$\begin{aligned} \alpha(R) &\geq 2^{-12(k+2)m-2} \cdot |\mathcal{T}|^{-1} \cdot |f^{-1}(1)|/|D^n| \\ &\geq 2^{-12(k+2)m-2-CM(P)/h} \cdot |f^{-1}(1)|/|D^n| \\ &\geq 2^{-12(k+2)m-k\cdot 2^{k+9}\cdot CM(P)/n-2} \cdot |f^{-1}(1)|/|D^n|. \end{aligned}$$

□

An example of a natural problem that we can apply this to is the Hamming Closeness problem $HAM_{1/8,n,N} : [N]^n \to \{0,1\}$ which outputs 1 iff there is a pair of input coordinates $x_i, x_j \in [N]$ such that the Hamming distance between the binary representations of x_i and x_j is at most $\frac{1}{8}\log_2 N$.

Proposition 7.3 ([BSSV03]). *For $f(x) = 1 - HAM_{1/8,n,N}(x)$, and $N \geq n^{4.39}$ we have*

- *(Proposition 6.15) $|f^{-1}(1)| \geq N^n/2$, and*
- *(Lemma 6.17) there is a constant $\beta > 0$ such that any embedded rectangle $R \subseteq f^{-1}(1)$ has $\alpha(R) \leq N^{-\beta m(R)}$.*

[BSSV03] apply the above to prove that any $[N]$-way branching program computing $HAM_{1/8,n,N}$ for $N \geq n^{4.39}$ in time T and space S requires T that is $\Omega(n \log\left(\frac{n \log n}{S}\right))$.

Theorem 7.4. *For $N \geq n^{4.39}$ any $[N]$-way branching program computing $HAM_{1/8,n,N}$ in time T that is $o(n \log n)$ requires cumulative memory $(n^2 \log n)/2^{O(T/n)}$ which is $n^{2-o(1)}$.*

Proof. Let P be an $[N]$-way branching program computing $HAM_{1/8,n,N}$ in time T that is $o(n \log n)$. We can swap the sink nodes to obtain a branching program P' computing $f = 1 - HAM_{1/8,n,N}$. Write $k = T/n$ and assume wlog that $k \geq 4$. Therefore k is $o(\log n)$ and hence $k^2 2^{k+8}$ is $n^{o(1)}$ and hence $\leq n$. Therefore by Corollary 7.2, there is an embedded rectangle $R \subseteq f^{-1}(1)$ such that $m(R) = m \geq n/2^{k+1}$ and

$$\alpha(R) \geq 2^{-12(k+2)m - k \cdot 2^{k+9} \cdot CM(P')/n - 2} \cdot |f^{-1}(1)|/N^n.$$

Therefore by Proposition 7.3, for some constant $\beta > 0$ we have

$$N^{-\beta m} \geq \alpha(R) \geq 2^{-12(k+2)m - k \cdot 2^{k+9} \cdot CM(P')/n - 3}.$$

Since $CM(P) = CM(P')$, solving we obtain

$$k \cdot 2^{k+9} \cdot CM(P) \geq \beta nm \log_2 N - 12(k+2)mn - 3n.$$

Since $k+2$ is $o(\log N)$ we obtain that $k \cdot 2^{k+9} \cdot CM(P) \geq \delta nm \log_2 N$ for some constant $\delta > 0$. Therefore, plugging in the value of T/n for k, we see that $CM(P)$ is $(n^2 \log n)/2^{O(T/n)}$. This is $n^{2-o(1)}$ by the bound on T. □

Similar bounds can also be shown by related means for various problems involving computation of quadratic forms, parity-check matrices of codes and others. For some problems the following stronger lower bound method is required.

Proposition 7.5 (Implicit in Corollary 5.4 of [BSSV03])**.** *Let P be a D-way branching program of length T computing a function $f : D^n \to \{0,1\}$. Suppose that $T \leq (k-2)n$ for $k \geq 8$ and $n \geq \ell \geq 2q^{5k^2}$ for $q \geq 2^{40}k^8$. If $0 = t_1 < t_2 < \cdots < t_{\ell+1} = T$ are time steps with $t_{i+1} - t_i \leq kn/q^{5k^2}$, then there is an embedded rectangle $R \subseteq f^{-1}(1)$ with $m(R) = m \geq q^{-2k^2} n/2$ and $\alpha(R) \geq 2^{-q^{-1/2}m} \cdot |\mathcal{T}|^{-1} \cdot |f^{-1}(1)|/|D|^n$ where $\mathcal{T}$ is the set of traces of P associated with time steps $t_1, \ldots, t_\ell$.*

Corollary 7.6. *Let P be a branching program of length T computing a function $f : D^n \to \{0,1\}$. If $T \leq (k-2)n$ for $k \geq 8$ and $n \geq 2q^{5k^2}$ for $q = 2^{40}k^8$, then there is an embedded rectangle $R \subseteq f^{-1}(1)$ with $m(R) = m \geq q^{-2k^2} n/2$ and $\alpha(R) \geq 2^{-q^{-1/2}m - q^{5k^2} CM(P)/n} \cdot |f^{-1}(1)|/|D^n|$.*

Proof Sketch. The proof is the analog of that of Corollary 7.2 using Proposition 7.5 in place of Proposition 7.1. □

Define the Element Distinctness function $ED_{n,N}$ on $[N]^n$ to be the Boolean function that is 1 iff all values in the input are distinct.

Proposition 7.7 ([BSSV03])**.** *For $N \geq n^2$,*

- *(Proposition 6.11)* $|ED^{-1}_{n,N}(1)| \geq N^n/e$, *and*

- *(Lemma 6.12) Every embedded rectangle* R *in* $ED^{-1}_{n,N}(1)$ *has* $\alpha(R) \leq 2^{-m(R)}$.

[BSSV03] used this to prove that the time T and space S for computing ED_{n,n^2} must satisfy $T = \Omega(n\sqrt{\log(n/S)/\log\log(n/S)})$. We strengthen this to the following theorem using Corollary 7.6.

Theorem 7.8. *Any* $[n^2]$*-way branching program computing* ED_{n,n^2} *in time* T *that is* $o(n\sqrt{\log n/\log\log n})$ *requires cumulative memory* $n^2/(T/n)^{O(T^2/n^2)}$ *which is* $n^{2-o(1)}$.

Proof. Let P compute ED_{n,n^2} in time T that is $o(n\sqrt{\log n/\log\log n})$. Write $k = T/n + 2$ so that $T \leq (k-2)/n$ and assume wlog that $k \geq 8$. Write $q = 2^{40}k^8$. Since T is $o(n\sqrt{\log n/\log\log n})$, k is $o(\sqrt{\log n/\log\log n})$ and $2q^{5k^2}$ which is $k^{O(k^2)}$ and hence $n^{o(1)}$ and therefore $\leq n$. We can then apply Corollary 7.6 to say that there is a rectangle $R \subseteq ED^{-1}_{n,n^2}(1)$ with $m(R) = m \geq q^{-2k^2}n/2$ and $\alpha(R) \geq 2^{-q^{-1/2}m - q^{5k^2}CM(P)/n} \cdot |ED^{-1}_{n,n^2}(1)|/|D^n|$. By Proposition 7.7, we have

$$2^{-m} \geq \alpha(R) \geq 2^{-q^{-1/2}m - q^{5k^2}CM(P)/n}/e.$$

Solving, we obtain that

$$q^{5k^2}CM(P) \geq n \cdot m(1 - 1/q^{1/2}) - 2n.$$

Therefore, since $m \geq q^{-2k^2}n/2$, we have constant c such that $CM(P) \geq n^2/q^{ck^2}$. As noted above, q^{ck^2} is $n^{o(1)}$. More precisely, the bound we obtain is

$$CM(P) \geq n^2/(T/n)^{O(T^2/n^2)}.$$

□

8 Extending the general method to quantum lower bounds

Quantum circuit time-space lower bounds have the same general structure as their classical branching program counterparts.

The standard method for quantum time-space tradeoff lower bounds

Let $f : D^n \to R^m$ be a multi-output function. For simplicity, we will assume that the output of f is always m elements in R. To obtain a time-space tradeoff lower bound on f, we must prove a lemma of the following form for some $m', h(k,n), \mu, r(n)$ and constant C:

Lemma 8.1 (Quantum generic property)**.** *For all* $k \leq m$ *and any quantum circuit* $\mathcal{C}$ *with at most* $h(k,n)$ *layers, there exists a distribution* μ *such that when* $x \sim \mu$*, the probability that* $\mathcal{C}$ *produces at least* k *correct output values of* $f(x)$ *is at most* $C \cdot |R|^{-k/r(n)}$.

Such lemmas have historically been proving using direct product theorems [KŠdW07, AŠdW09] and, more recently, using the recording query technique [HM21]. This is the quantum version of condition (B) for the classical general method. In the classical setting, the lemma could be extended to account for the 2^S boundary nodes between layers by using a union bound over 2^S possible branching programs. However in the quantum setting it is not as obvious how to use a lemma that does not account for initial state. Aaronson showed in [Aar05] how to do exactly this using the following proposition, which we previously used in Section 4.

Proposition 4.4 ([Aar05]). *Let $\mathcal{C}$ be a quantum circuit, ρ be any S qubit (possibly mixed) state, and I be the S qubit maximally mixed state. If $\mathcal{C}$ with initial state ρ produces some output $\mathcal{O}$ with probability p, then $\mathcal{C}$ with initial state I produces $\mathcal{O}$ with probability at least $p/2^{2S}$.*

Thus for any problem where we can prove something similar to Lemma 8.1, we can bound the probability of circuits with S qubits of input-dependent state producing k correct outputs as being at most $2^{2S} \cdot C \cdot |R|^{-k/r(n)}$. This idea has been applied in [KŠdW07, AŠdW09, HM21] to bound the probability that blocks produce correct outputs, even when they are given initial state from previous blocks.

From here we take any circuit $\mathcal{C}$ with T layers and S qubits and split it into sub-circuits $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ with $h = h(k, n)$ layers each. This makes $\ell = \lceil T/h \rceil$. While Lemma 8.1 gives us that $\mathcal{C}_1$ produces at least k correct outputs with probability at most $|R|^{-k/r(n)}$, sub-circuits $\mathcal{C}_2, \ldots, \mathcal{C}_\ell$ start with some initial state $\rho_2, \ldots, \rho_\ell$ that can depend on the input. Since ρ_i has at most S qubits, the probability that $\mathcal{C}_i$ produces at least k correct outputs is at most

$$2^{2S} \cdot C \cdot |R|^{-k/r(n)}$$

Assume that:

$$2S \leq \frac{\log_2 |R|}{r(n)} \cdot m' - \log_2(2C) \tag{9}$$

Then we can set $k = [2S + \log_2(2C)] \cdot r(n)/\log_2 |R|$ and get that the probability of producing K correct outputs is at most $1/2$. There must be some block that produces at least $m \cdot h(S, n)/T$ correct outputs, so we must have that

$$m \cdot h(S, n)/T > k = [2S + \log_2(2C)] \cdot r(N)/\log_2 |R|$$

This gives us that

$$TS \text{ is } \Omega\left(\frac{mh(S, n)\log |R|}{r(n)}\right).$$

In the event that (9) is not satisfied, we can instead use the bounded-error quantum query complexity of f (denoted $Q(f)$) instead of the decision tree complexity to obtain that TS is

$$\Omega\left(\frac{Q(f) \cdot m' \log_2 |R|}{r(n)}\right).$$

Generic quantum cumulative complexity

Quantum sorting In Section 4 we are able to exploit some specific structure that leads to a cleaner proof than we can obtain in general. Specifically for quantum sorting we have $h_0(s) = \sqrt{s}$. Since h_0 is not a constant function, we cannot apply arguments like Theorem 3.3 or Theorem 5.3. However h_0 is a polynomial function. This means that a block with at least $s/4$ qubits per layer for at least

$h(s/4,n)/2$ layers has a constant fraction of the cumulative complexity of a block with $h(s,n)$ layers that each have s qubits. This means that we can use Lemma 4.3 to upper bound the number of outputs for a block with $h(4s,n)$ layers and $4s$ initial qubits while obtaining a lower bound on the cumulative complexity of such a block that is within a constant factor of the TS complexity of that block. To obtain such a bound on the cumulative complexity, we can start with a segment of length $h(s,n)$ when $s=1$ that ends at the start of the next block and then repeatedly multiply s by four until we find a block of length $h(s,n)$ where one of the first $h(s,n)/2$ layers has less than s space. Since this is the first such segment, we know that there must be $h(s/4,n)/2$ layers that each have $s/4$ qubits, which gives us the asymptotically tight cumulative complexity lower bound for the block. The argument we used in our quantum sorting proof can be applied to other classical and quantum time-space tradeoffs where $h_0(s)$ is a polynomial function.

The generic completion In general, $h_0(s)$ may not be a polynomial function. When this is not the case, we can observe that Theorem 5.6 does not exploit any structure of branching programs that cannot be applied to quantum circuits. It depends only on the existence of a lemma that bounds the number of outputs for short computation and a way to apply that lemma to computation with input dependent initial state, which are given by our generic Lemma 8.1 and Proposition 4.4. We state the quantum versions of Theorem 5.6 and Remark 5.8 when $\alpha=1$ here for completeness. Note that since Proposition 4.4 gives us a bound of $p/2^{2S}$ rather than $p/2^S$, the cumulative memory bounds we obtain in the quantum setting are half of those from Theorem 5.6.

Corollary 8.2. *Let $c>0$. Suppose that function f defined on D^n satisfies generic Lemma 8.1 with m' that is $\omega(\log_2 n)$. For $s>0$, let $h(s,n)=h'(s\cdot r(n)/\log_2|R|,n)$. Let $h(s,n)=h_0(s)h_1(n)$ where $h_0(1)=1$ and h_0 is a differentiable function where $s/h_0(s)$ is increasing and concave. Let S^* be defined by:*

$$\frac{S^*}{h_0(S^*)}=\frac{m\cdot h_1(n)\cdot\log_2|R|}{12r(n)T}$$

Then either

$$T\log_2(2CT^{c+1})>\frac{m\cdot h_1(n)\cdot\log_2|R|}{12r(n)}$$

Which implies that T is $\Omega(\frac{m\cdot h_1(n)\cdot\log|R|}{r(n)\log n})$ or the cumulative memory used by a quantum circuit that computes f in time T with error $\varepsilon\le(1-1/(2T^c))$ is at least

$$\mathcal{L}_{h_0}(n\log_2|D|)\cdot\min\left(m\cdot h(S^*,n),\ 3m'\cdot h'(m'/2,n)\right)\cdot\frac{\log_2|R|}{12r(n)}.$$

Additionally if the quantum circuit uses $o(\frac{m'\log|R|}{r(n)})$ qubits, then the cumulative memory bound instead is

$$\mathcal{L}_{h_0}(n\log_2|D|)\cdot m\cdot h(S^*,n)\cdot\frac{\log_2|R|}{12r(n)}.$$

9 Quantum applications of the generic method

Disjoint Collision Pairs Finding

In [HM21] the authors considered the problem of finding k disjoint collisions in a random function $f : [m] \to [n]$, and were able to prove a time-space tradeoff that $T^3 S$ is $\Omega(k^3 n)$ for circuits that solve the problem with success probability $2/3$. Specifically, they consider circuits that must output triples $(x_{j_{2i}}, x_{j_{2i+1}}, y_{j_i})$ where $f(x_{j_{2i}}) = f(x_{j_{2i+1}}) = y_{j_i}$. To obtain this result, they prove the following theorem using the recording query technique:

Proposition 9.1 (Theorem 4.6 in [HM21])**.** *For all $1 \le k \le n/8$ and any quantum circuit $\mathcal{C}$ with at most T quantum queries to a random function $f : [m] \to [n]$, the probability that $\mathcal{C}$ produces at least k disjoint collisions in f is at most $O(T^3/(k^2 n))^{k/2} + 2^{-k}$.*

The above theorem can be extended to a lemma matching Lemma 8.1 by choosing a sufficiently small constant α and setting $T = \alpha k^2 n$ to obtain a probability of at most 2^{S+1-k}. This is sufficient to obtain a matching lower bound on the cumulative memory complexity using Corollary 8.2.

Theorem 9.2. *Finding $\omega(\log_2 n) \le k \le n/8$ disjoint collisions in a random function $f : [m] \to [n]$ with probability at least $2/3$ requires time T is $\Omega(kn^{1/3}/\log n)$ or cumulative memory $\Omega(k^3 n/T^2)$.*

Proof. Proposition 9.1 lets us apply Corollary 8.2 with $m = m' = k, h'(k,n) = \alpha k^{2/3} n^{1/3}, |R| = m^2 n - mn$, and $r(n) = \log_2 |R|$. Thus we have $h(s,n) = h'(s,n)$ and h_0 is a differentiable function where $s/h_0(s)$ is an increasing and concave function. With these parameters, we have:

$$S^* \text{ is } \Omega\left(\frac{k^3 n}{T^3}\right)$$

By Corollary 8.2 with the observation that the loss is constant we get that:

$$T \text{ is } \Omega\left(\frac{kn^{1/3}}{\log n}\right)$$

or the quantum cumulative memory is:

$$\Omega\left(\min\left(\frac{k^3 n}{T^2}, k^{5/3} n^{1/3}\right)\right).$$

By Proposition 9.1 we know that any quantum circuit with at most $T' = \alpha k^{2/3} n^{1/3}$ layers can produce k disjoint collisions with probability at most 2^{1-k}. Thus we know that $T > T'$ and our cumulative memory bound becomes $\Omega(k^3 n/T^2)$. □

On Tradeoffs for Linear Inequalities and Boolean Linear Algebra

In this section we consider problems in Boolean linear algebra where we write $A \bullet x$ for Boolean (i.e. and-or) matrix-vector product and $A \bullet B$ for Boolean matrix multiplication. In [KŠdW07] the authors prove the following time-space tradeoff for Boolean matrix vector products:

Proposition 9.3 (Theorem 23 in [KŠdW07]). *For every S in $o(n/\log n)$, there is an $n \times n$ Boolean matrix A_S such that every bounded-error quantum circuit with space at most S that computes Boolean matrix vector product $A_S \bullet x$ in T queries requires that T is $\Omega(\sqrt{n^3/S})$.*

This result is weaker than a standard time-space tradeoff since the function involved is not independent of the circuits that might compute it. In particular, [KŠdW07] does not find a single function that is hard for all space bounds, as the matrix A that they use changes depending on the value of S. For example, a circuit using space $S' \gg S$ could potentially compute $A_S \bullet x$ using $o(n^{3/2}/(S')^{1/2})$ queries. This means that an extension of their bound to cumulative memory complexity does not follow from our Corollary 8.2, as blocks with distinct numbers of initial qubits would be computing outputs for different functions. In [AŠdW09] the authors use the same space-dependent matrices to prove a result for systems of linear inequalities.

Proposition 9.4 (Theorem 19 in [AŠdW09]). *Let S be in $\min(O(n/t), o(n/\log n))$ and $\vec{t}$ be the all-t vector. There is an $n \times n$ Boolean matrix A_S such that every bounded error quantum circuit using space S for evaluating the system $A_S x \geq \vec{t}$ using T queries requires T that is $\Omega(\sqrt{(tn^3/S)})$.*

Again this result is not a general time-space tradeoff and hence is not compatible with obtaining a true cumulative memory bound[8]. While neither of the above results is a time-space tradeoff for a fixed function, [KŠdW07] leverages the ideas for Proposition 9.3 to compute a true time-space tradeoff lower bound for computing Boolean matrix multiplication.

Proposition 9.5 (Theorem 25 in [KŠdW07]). *If a quantum circuit computes the Boolean matrix product $A \bullet B$ with bounded error using T queries and S space, then TS is $\Omega(n^5/T)$.*

In Proposition 9.5, unlike in Proposition 9.3 and Proposition 9.4, both A and B are inputs to the problem. This allows the lower bound argument to use the properties of the circuit to find matrices A and B for which the circuit will be particularly challenged. More precisely, to prove the above result, the authors use a lemma matching the form of Lemma 8.1 that we extract from their lower bound argument.

Proposition 9.6 (from Theorem 25 in [KŠdW07]). *Let $R \subseteq [n] \times [n]$ be any fixed set of $k \in o(n)$ outputs to the function $f(A, B) = A \bullet B$. Then there are constants $\alpha, \gamma > 0$ such that for any quantum circuit $\mathcal{C}$ with at most $\alpha\sqrt{kn}$ layers, there is a distribution $\mu_{\mathcal{C}}$ over pairs of matrices such that when $(A, B) \sim \mu_{\mathcal{C}}$, the probability that $\mathcal{C}$ produces the correct values for R is at most $2^{-\gamma k}$.*

Note that, though there are $\Omega(n^2)$ total output values, Proposition 9.6 only works when k — the number of output values in a block — is sublinear in n. This is not a problem in the time-space tradeoff lower bound. Proposition 9.6 upper bounds the value of k for a block as $O(S)$. Since the time T must be $\Omega(n^2)$ simply to read the input, the bound $T^2S = \Omega(n^5)$ trivially holds when S is $\Omega(n)$. Thus the time-space tradeoff proof only needs to apply Proposition 9.6 when S (and therefore k) is sublinear in n.

We cannot apply such an argument when considering cumulative memory complexity, as a circuit can use $\Omega(n)$ qubits for a small number of layers without having an asymptotic effect on the

[8]The analogous cumulative complexity result would require the matrix A to depend extensively on the structural properties of the circuit, including the number of qubits after each layer and the locations of each fixed output gate. It is unclear whether the TS results also may need the matrix A_S to depend on the locations of the output gates.

cumulative memory complexity. However, if we consider $o(n)$ space bounded computation, we can get a matching bound on the cumulative memory complexity.

Theorem 9.7. *Any quantum circuit that computes the boolean matrix product $A \bullet B$ requires $\Omega(n)$ ancilla qubits, time T that is $\Omega(n^{5/2}/\log n)$, or cumulative memory that is $\Omega(n^5/T)$.*

Proof. Proposition 9.6 lets us apply Corollary 8.2 with m' being $o(n)$, $m = n^2$, $h'(k,n) = \alpha\sqrt{kn}$, $|R| = 2$, and $r(n) = 1/\gamma$. Thus we have $h(s,n) = h'(s/\gamma, n) = \alpha\sqrt{sn/\gamma}$ and $h_0(s) = \sqrt{s}$. Therefore we define S^* to be

$$S^* = \frac{\gamma\alpha^2 n^5}{36T^2}$$

Thus by Corollary 8.2 we get that T is $\Omega(n^{5/2}/\log n)$ or since the space bound is $o(n)$ and the loss function is constant, the cumulative memory is $\Omega(n^5/T)$. □

Though this is somewhat limited in its range of applicability, it still yields a strict generalization of the time-space tradeoff lower bound of Proposition 9.5 when S is $o(n)$ and T is $o(n^{5/2}/\log n)$.

10 Acknowledgements

Many thanks to David Soloveichik for his guidance and contributions to our initial results. Thanks also to Scott Aaronson for encouraging us to consider cumulative memory complexity in the context of quantum computation.

References

[Aar05] Scott Aaronson. Limitations of quantum advice and one-way communication. *Theory of Computing*, 1(1):1–28, 2005.

[AB16] Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 241–271, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[ABB21] Mohammad Hassan Ameri, Alexander R. Block, and Jeremiah Blocki. Memory-hard puzzles in the standard model with applications to memory-hard functions and resource-bounded locally decodable codes. Cryptology ePrint Archive, Paper 2021/801, 2021. https://eprint.iacr.org/2021/801.

[ABP17] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 3–32, Cham, 2017. Springer International Publishing.

[Abr87] Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987.

[Abr90] Karl R. Abrahamson. A time-space tradeoff for Boolean matrix multiplication. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 412–419. IEEE Computer Society, 1990.

[Abr91] Karl R. Abrahamson. Time-space tradeoffs for algebraic problems on general sequential machines. *J. Comput. Syst. Sci.*, 43(2):269–289, 1991.

[ACK+16] Joël Alwen, Binyi Chen, Chethan Kamath, Vladimir Kolmogorov, Krzysztof Pietrzak, and Stefano Tessaro. On the complexity of Scrypt and proofs of space in the parallel random oracle model. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 358–387. Springer, 2016.

[ACP+17] Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 33–62, 2017.

[AdRNV17] Joël Alwen, Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals. Cumulative space in black-white pebbling and resolution. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:21, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Ajt02] Miklós Ajtai. Determinism versus nondeterminism for linear time RAMs with memory restrictions. *J. Comput. Syst. Sci.*, 65(1):2–37, 2002.

[Ajt05] Miklós Ajtai. A non-linear time lower bound for Boolean branching programs. *Theory Comput.*, 1(1):149–176, 2005.

[Amb02] Andris Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002.

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, Stoc '15, page 595–603, New York, NY, USA, 2015. Association for Computing Machinery.

[AŠdW09] Andris Ambainis, Robert Špalek, and Ronald de Wolf. A new quantum lower bound method, with applications to direct product theorems and time-space tradeoffs. *Algorithmica*, 55(3):422–461, Nov 2009.

[BBB+21] Andrew Baird, Bryant Bost, Stefano Buliani, Vyom Nagrani, Ajay Nair, Rahul Popat, and Brajendra Singh. AWS serverless multi-tier architectures with Amazon API Gateway and AWS Lambda, Oct 2021.

[BBBV97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997.

[BBC+01] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. Acm*, 48(4):778–797, jul 2001.

[BC82] Allan Borodin and Stephen A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.*, 11(2):287–297, 1982.

[BCGS16] Dan Boneh, Henry Corrigan-Gibbs, and Stuart Schechter. Balloon hashing: A memory-hard function providing provable protection against sequential attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 220–248, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BDK16] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 292–302, 2016.

[Bea91] Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20(2):270–277, 1991.

[BFK+81] Allan Borodin, Michael J. Fischer, David G. Kirkpatrick, Nancy A. Lynch, and Martin Tompa. A time-space tradeoff for sorting on non-oblivious machines. *J. Comput. Syst. Sci.*, 22(3):351–364, 1981.

[BJS01] Paul Beame, T. S. Jayram, and Michael E. Saks. Time-space tradeoffs for branching programs. *J. Comput. Syst. Sci.*, 63(4):542–572, 2001.

[BRS93] Allan Borodin, Alexander A. Razborov, and Roman Smolensky. On lower bounds for read-k-times branching programs. *Comput. Complex.*, 3:1–18, 1993.

[BSSV03] Paul Beame, Michael E. Saks, Xiaodong Sun, and Erik Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *J. ACM*, 50(2):154–195, 2003.

[BZ17] Jeremiah Blocki and Samson Zhou. On the depth-robustness and cumulative pebbling cost of Argon2i. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 445–465, Cham, 2017. Springer International Publishing.

[CFL83] Ashok K. Chandra, Merrick L. Furst, and Richard J. Lipton. Multi-party protocols. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, page 94–99, New York, NY, USA, 1983. Association for Computing Machinery.

[CT19] Binyi Chen and Stefano Tessaro. Memory-hard functions from cryptographic primitives. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 543–572, Cham, 2019. Springer International Publishing.

[DNW05] Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, pages 37–54, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[HM21] Yassine Hamoudi and Frédéric Magniez. Quantum time-space tradeoff for finding multiple collision pairs. In Min-Hsiu Hsieh, editor, *16th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2021)*, volume 197 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:21, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Juk09] Stasys Jukna. A nondeterministic space-time tradeoff for linear codes. *Inf. Process. Lett.*, 109(5):286–289, 2009.

[KŠdW07] Hartmut Klauck, Robert Špalek, and Ronald de Wolf. Quantum and classical strong direct product theorems and optimal time-space tradeoffs. *SIAM Journal on Computing*, 36(5):1472–1493, 2007.

[LL20] Tan N. Le and Zhenhua Liu. Flex: Closing the gaps between usage and allocation. In *e-Energy '20: The Eleventh ACM International Conference on Future Energy Systems, Virtual Event, Australia, June 22-26, 2020*, pages 404–405. ACM, 2020.

[MNT93] Yishay Mansour, Noam Nisan, and Prasoon Tiwari. The computational complexity of universal hashing. *Theor. Comput. Sci.*, 107(1):121–133, 1993.

[Oko93] E. Okol'nishnikova. On lower bounds for branching programs. *Siberian Advances in Mathematics*, 3(1):152–166, 1993.

[Pip79] Nicholas Pippenger. On simultaneous resource bounds (preliminary version). In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 307–311. IEEE Computer Society, 1979.

[RD16] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In *Proceedings, Part I, of the 14th International Conference on Theory of Cryptography - Volume 9985*, page 262–285, Berlin, Heidelberg, 2016. Springer-Verlag.

[Sha03] Ronen Shaltiel. Towards proving strong direct product theorems. *computational complexity*, 12(1):1–22, Jun 2003.

[She11] Alexander A. Sherstov. Strong direct product theorems for quantum communication and query complexity. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, Stoc '11, page 41–50, New York, NY, USA, 2011. Association for Computing Machinery.

[ŠS05] Robert Špalek and Mario Szegedy. All quantum adversary methods are equivalent. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, pages 1299–1311, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[SSGW11] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, Socc '11, New York, NY, USA, 2011. Association for Computing Machinery.

[SW03] Martin Sauerhoff and Philipp Woelfel. Time-space tradeoff lower bounds for integer multiplication and graphs of arithmetic functions. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 186–195. ACM, 2003.

[Tom80] Martin Tompa. Time-space tradeoffs for computing functions, using connectivity properties of their circuits. *J. Comput. Syst. Sci.*, 20(2):118–132, 1980.

[Yao77] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 222–227. IEEE Computer Society, 1977.

[Yes84] Yaacov Yesha. Time-space tradeoffs for matrix multiplication and the discrete Fourier transform on any general sequential random-access computer. *Journal of Computer and System Sciences*, 29(2):183–197, 1984.

[Zha19] Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 239–268, Cham, 2019. Springer International Publishing.

A Optimizations

In this section we prove general optimization lemmas that allow us to derive worst-case properties of the allocation of branching program layers into blocks.

The first special case is relevant for our analysis of quantum sorting algorithms.

Lemma A.1. *If* $\sum_i x_i \leq \sum_i x_i^2$ *then* $\sum_i x_i^3 \geq \sum_i x_i^2$.

Proof. Without loss generality we remove all x_i that are 0 or 1 since they contribute the same amount to each of $\sum_i x_i$, $\sum_i x_i^2$, and $\sum_i x_i^3$. Therefore every x_i satisfies $0 < x_i < 1$ or it satisfies $x_i > 1$. For simplicity we rename those x_i with $0 < x_i < 1$ by y_i and those x_i with $x_i > 1$ by z_j.

Then $\sum_i x_i \leq \sum_i x_i^2$ can be rewritten as

$$\sum_i y_i(1-y_i) \leq \sum_j z_j(z_j - 1),$$

and both quantities are positive. Let y^* be the largest value < 1 and z^* be the smallest value > 1. Therefore we have

$$\begin{aligned}
\sum_i (y_i^2 - y_i^3) &= \sum_i y_i^2(1-y_i) \\
&\leq \sum_i y^* y_i(1-y_i) \\
&= y^* \sum_i y_i(1-y_i) \\
&\leq y^* \sum_j z_j(z_j-1) \\
&< z^* \sum_j z_j(z_j-1) \\
&= \sum_j z^* z_j(z_j-1) \\
&\leq \sum_j z_j^2(z_j-1) \\
&= \sum_j (z_j^3 - z_j^2).
\end{aligned}$$

Rewriting we have $\sum_i y_i^2 + \sum_j z_j^2 < \sum_i y_i^3 + \sum_j z_j^3$, or equivalently $\sum_i x_i^3 > \sum_i x_i^2$, as required. □

The following is a generalization of the above to all differentiable functions $p : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ such that $s/p(s)$ is a concave increasing function of s.

Lemma 5.7. *Let* $p : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ *be a differentiable function such that* $q(x) = x/p(x)$ *is a concave increasing function of* x. *For* $x_1, x_2, \ldots \in \mathbb{R}^{\geq 0}$, *if* $\sum_i x_i \geq K$ *and* $\sum_i p(x_i) \leq L$ *then* $\sum_i x_i p(x_i) \geq q^{-1}(K/L) \cdot L$.

Proof. By hypothesis,

$$\sum_i (x_i - Kp(x_i)/L) \geq 0. \tag{10}$$

Observe that $s - Kp(s)/L$ is an increasing function of s since $s/p(s)$ is an increasing function of s that is 0 precisely when $s = q^{-1}(K/L)$. Since all x_i with $x_i = q^{-1}(K/L)$ evaluate to 0 in Equation (10), we can rewrite it as

$$\sum_{x_i > q^{-1}(K/L)} (x_i - Kp(x_i)/L) \geq \sum_{x_i < q^{-1}(K/L)} (Kp(x_i)/L - x_i)\,, \tag{11}$$

where each of the summed terms on the two sides of the inequality is positive. For $x_i \neq q^{-1}(K/L)$, define

$$f(x_i) = x_i \cdot \frac{p(x_i) - q^{-1}(K/L) \cdot L/K}{x_i - Kp(x_i)/L}.$$

Observe that for $x_i = q^{-1}(K/L)$ the denominator is 0 and the numerator equals $p(x_i) - x_i \cdot L/K$ which is also 0. For $x_i > q^{-1}(K/L)$ both the numerator and denominator are positive and for $x_i < q^{-1}(K/L)$ both the numerator and denominator are negative. Hence $f(x_i)$ is non-negative for every $x_i \neq q^{-1}(K/L)$.

CLAIM: If q is a convex differentiable function, we can complete f to a (non-decreasing) continuous function of x with $f'(x) \geq 0$ for all x with $0 < x \neq q^{-1}(K/L)$.

Proof of Claim. Write $a = q^{-1}(K/L)$. Then since $p(x) > 0$ and $q(a) > 0$, we have

$$\begin{aligned} f(x) &= \frac{x \cdot p(x) - x \cdot a/q(a)}{x - q(a) \cdot p(x)} = \frac{x - (x/p(x)) \cdot a/q(a)}{x/p(x) - q(a)} \\ &= \frac{x - q(x) \cdot a/q(a)}{q(x) - q(a)} = \frac{1}{q(a)} \cdot \frac{q(a) \cdot x - a \cdot q(x)}{q(x) - q(a)}. \end{aligned}$$

Therefore

$$\begin{aligned} f'(x) &= \frac{1}{q(a)} \cdot \frac{(q(a) - a \cdot q'(x))(q(x) - q(a)) - (q(a) \cdot x - a \cdot q(x)) \cdot q'(x)}{(q(x) - q(a))^2} \\ &= \frac{q(x) - q(a) + (a - x) \cdot q'(x)}{(q(x) - q(a))^2}. \end{aligned}$$

Since the denominator is a square and q is increasing, to prove that $f'(x) \geq 0$ for $x \neq a$ it suffices to prove that the numerator is non-negative.

Suppose first that $x < a$, Then $a - x > 0$ and the numerator $q(x) - q(a) + (a - x) \cdot q'(x) \geq 0$ if and only if $q'(x) \geq \frac{q(a) - q(x)}{a - x}$, which is equivalent to the slope of the tangent to q at x being at least that of the chord from x to a. This is certainly true since q is a concave function.

Suppose now that $x > a$. Then $a - x < 0$ and the numerator $q(x) - q(a) + (a - x) \cdot q'(x) \geq 0$ if and only if $q'(x) \leq \frac{q(x) - q(a)}{x - a}$. Again, this is true since q is a concave function.

It remains to show that we can complete f to a continuous function by giving it a finite value at $a = q^{-1}(K/L)$. By l'Hôpital's rule, the limit of $q(a) \cdot f(x)$ as x approaches a is

$$\frac{q(a) - a \cdot q'(a)}{q'(a)}$$

if the denominator is non-zero, which it is, since q is an increasing differentiable function at a. □

We now have the tools we need. Let x_-^* be the largest $x_i < q^{-1}(K/L)$ and x_+^* be the smallest $x_i > q^{-1}(K/L)$. Then we have $f(x_+^*) \geq f(x_-^*)$ and

$$\begin{aligned}
&\sum_{x_i > q^{-1}(K/L)} \Big(x_i \; p(x_i) - q^{-1}(K/L) \cdot L/K \cdot x_i \Big) \\
&= \sum_{x_i > q^{-1}(K/L)} x_i \cdot \frac{p(x_i) - q^{-1}(K/L) \cdot L/K}{x_i - Kp(x_i)/L} \cdot (x_i - Kp(x_i)/L) \\
&= \sum_{x_i > q^{-1}(K/L)} f(x_i) \cdot (x_i - Kp(x_i)/L) \\
&\geq \sum_{x_i > q^{-1}(K/L)} f(x_+^*) \cdot (x_i - Kp(x_i)/L) \\
&= f(x_+^*) \sum_{x_i > q^{-1}(K/L)} (x_i - Kp(x_i)/L) \\
&\geq f(x_-^*) \sum_{x_i > q^{-1}(K/L)} (x_i - Kp(x_i)/L) \\
&\geq f(x_-^*) \sum_{x_i < q^{-1}(K/L)} (Kp(x_i)/L - x_i) \qquad \text{by Equation (11)} \\
&= \sum_{x_i < q^{-1}(K/L)} f(x_-^*) \cdot (Kp(x_i)/L - x_i) \\
&\geq \sum_{x_i < q^{-1}(K/L)} f(x_i) \cdot (Kp(x_i)/L - x_i) \\
&= \sum_{x_i < q^{-1}(K/L)} x_i \cdot \frac{q^{-1}(K/L) \cdot L/K - p(x_i)}{Kp(x_i)/L - x_i} \cdot (Kp(x_i)/L - x_i) \\
&= \sum_{x_i < q^{-1}(K/L)} \Big(q^{-1}(K/L) \cdot L/K \cdot x_i - x_i \; p(x_i) \Big).
\end{aligned}$$

Adding back the terms where $x_i = q^{-1}(K/L)$, which have value 0, and rewriting we obtain that

$$\sum_i \Big(x_i \; p(x_i) - q^{-1}(K/L) \cdot L/K \cdot x_i \Big) \geq 0.$$

Therefore we have

$$\sum_i x_i \; p(x_i) \geq q^{-1}(K/L) \cdot L/K \cdot \sum_i x_i \geq q^{-1}(K/L) \cdot (L/K) \cdot K = q^{-1}(K/L) \cdot L$$

as required. □

B Proof of Lemma 5.5

Lemma 5.5. *The following hold for every non-decreasing function $p : \mathbb{R} \to \mathbb{R}$ with $p(1) = 1$:*

(a) $1/p(n) \leq \mathcal{L}_p(n) \leq 1$.

(b) If p is a polynomial function $p(s) = s^{1/c}$ then $\mathcal{L}_p(n) > 1/2^{c+1}$.

(c) For any $c > 1$, $\mathcal{L}_p(n) \geq \min_{1 \leq s \leq n} \frac{p(s) - p(s/c)}{cp(s)}$.

(d) We say that p is nice *if it is differentiable and there is an integer $c > 1$ such that for all x, $p'(cx) \geq p'(x)/c$. If p is nice then $\mathcal{L}_p(n)$ is $\Omega(1/\log_2 n)$. This is tight for p with $p(s) = 1 + \log_2 s$.*

Proof. Since p is non-decreasing, $1 \leq p^{-1}(j) \leq p^{-1}(k)$ for $1 \leq j \leq k$ and hence

$$\frac{1}{k} \leq \frac{\sum_{j=1}^{k} p^{-1}(j)}{k \cdot p^{-1}(k)} \leq 1 \tag{12}$$

since $p^{-1}(k)$ is included in the numerator. $\mathcal{L}_p(n)$ is the minimum over all integers $k \in [1, p(n)]$ of $\frac{\sum_{j=1}^{k} p^{-1}(j)}{k \cdot p^{-1}(k)}$ and p is non-decreasing so we have $1/p(n) \leq \mathcal{L}_p(n) \leq 1$, which proves part (a)

When $p(s) = s^{1/c}$ we have

$$\sum_{j=1}^{k} p^{-1}(j) \geq \sum_{j=\lceil (k+1)/2 \rceil}^{k} j^c > \lceil k/2 \rceil (k/2)^c \geq (k/2)^{c+1} = k \cdot p^{-1}(k)/2^{c+1}$$

so each term in the definition of $\mathcal{L}_p(n)$ is larger than $1/2^{c+1}$ which proves part (b). (More precise bounds can be shown but we are not focused on the specific constant.)

Let $1 \leq k \leq p(n)$ be an integer. Then $1 \leq s = p^{-1}(k) \leq n$. Observe that there are at least $p(s) - p(s/c)$ integers $j \leq k$ with $p^{-1}(j) \geq s/c$. Therefore

$$\frac{\sum_{j=1}^{k} p^{-1}(j)}{k \cdot p^{-1}(k)} \geq \frac{(p(s) - p(s/c)) \cdot s/c}{kp^{-1}(k)} = \frac{p(s) - p(s/c)}{ck} = \frac{p(s) - p(s/c)}{cp(s)}. \tag{13}$$

The minimum over all $k \in [1, p(n)]$ is equivalent to the minimum over all $s \in [1, n]$, which proves part (c).

Now suppose that p is nice. Since p is differentiable, for any s,

$$\begin{aligned} p(cs) - p(s) &= \int_{s}^{cs} p'(y)\, dy \\ &= \int_{s/c}^{c} p'(cx) c\, dx \quad \text{by substitution } y = cx \\ &\geq \int_{s/c}^{c} p'(x)\, dx \quad \text{since } p \text{ is nice} \\ &= p(s) - p(s/c). \end{aligned}$$

Then by induction we have that for every positive integer $i \le \log_c s$, $p(s) - p(s/c) \ge p(s/c^{i-1}) - p(s/c^i)$. Write $\ell = \lfloor \log_c s \rfloor$. Then $s/c^\ell < c$ and

$$p(s) - p(s/c^\ell) = \sum_{i=1}^{\ell} [p(s/c^{i-1}) - p(s/c^i)] \le \ell \cdot [p(s) - p(s/c)],$$

or equivalently that $p(s) - p(s/c) \ge (p(s) - p(s/c^\ell))/\ell$ and hence

$$p(s) - p(s/c) \ge (p(s) - p(c))/\log_c s$$

since p is a non-decreasing function. Applying the lower bound from Equation (12) when $k = p(s) < 2p(c)$ and the lower bound from Equation (13) when $p(s) \ge 2p(c)$ we obtain

$$\mathcal{L}_p(n) \ge \min\left(\frac{1}{2p(c)}, \min_{1 \le s \le n : p(s) \ge 2p(c)} (1 - p(c)/p(s))/(c \log_c s)\right).$$

Since c is a constant, we obtain that $\mathcal{L}_p(n)$ is $\Omega(1/\log n)$.

Observe that p given by $p(s) = 1 + \log_2 s$ is nice for every constant $c > 0$ since $p'(cx) = (\ln 2)^{-1}/(cx) = p'(x)/c$. In this case we have $p^{-1}(j) = 2^{j-1}$ and $\mathcal{L}_p(n) < 2/p(n) < 2/\log_2 n$ since the largest term $p^{-1}(k)$ in each numerator is (a little) more than the sum of all smaller terms put together. Together with the lower bound, this proves part (d). $\square$

C The dependence of α on γ in Proposition 4.2

The following lemma is sufficient to prove Theorem 13 in [KŠdW07] (our Proposition 4.2). Although the authors prove a more general version of this proposition, the statement below captures what is necessary in our proof.

Proposition C.1 (Special case of Lemma 12 in [KŠdW07])**.** *Let p be a degree $2\alpha\sqrt{kn}$ univariate polynomial such that:*

- $p(i) = 0$ *when* $i \in \{0, \ldots, k-1\}$
- $p(k) = \sigma$
- $p(i) \in [0,1]$ *when* $i \in \{k+1, \ldots, n\}$

Then there exists universal positive constants a and b such that for any $\gamma > 0$ where $ke^{\gamma+1} \le n - k$:

$$\sigma \le a \cdot exp\left(\frac{b(2\alpha\sqrt{kn} - k)^2 + 4e^{\gamma/2+1/2}k\sqrt{n-k}(2\alpha\sqrt{n} - \sqrt{k})}{n - k(e^{\gamma+1}+1)} - k - \gamma k\right).$$

The σ in this bound gives the completeness bound on the k-threshold problem. So we need a choice of α such that $\sigma \le e^{-\gamma k}$. We now prove that this is possible when $\alpha \in \Omega(e^{-\gamma/2})$.

Lemma C.2. *If $\sqrt{k/n} \le \alpha \le \min(1/(16\sqrt{e^{\gamma+1}} + 1), 1/(2\sqrt{2b}))$ then $\sigma \le a \cdot e^{-\gamma k}$.*

Proof. We show this with a chain of inequalities from Proposition C.1 using our bounds on α:

$$
\begin{aligned}
\sigma &\leq a \cdot \exp\left(\frac{b(2\alpha\sqrt{kn}-k)^2 + 4e^{\gamma/2+1/2}k\sqrt{n-k}(2\alpha\sqrt{n}-\sqrt{k})}{n-k(e^{\gamma+1}+1)} - k - \gamma k\right) \\
&= a \cdot \exp\left(\frac{b(4\alpha^2 kn - 4\alpha k\sqrt{kn} + k^2) + 4e^{\gamma/2+1/2}k\sqrt{n-k}(2\alpha\sqrt{n}-\sqrt{k})}{n-k(e^{\gamma+1}+1)} - k - \gamma k\right) \\
&= a \cdot \exp\left(\frac{kb\alpha^2(4n - 4\sqrt{kn}/\alpha + k/\alpha^2) + 4e^{\gamma/2+1/2}k\sqrt{n-k}(2\alpha\sqrt{n}-\sqrt{k})}{n-k(e^{\gamma+1}+1)} - k - \gamma k\right) \\
&\leq a \cdot \exp\left(\frac{kb\alpha^2(4n - 3k/\alpha^2) + 4e^{\gamma/2+1/2}k\sqrt{n-k}(2\alpha\sqrt{n}-\sqrt{k})}{n-k(e^{\gamma+1}+1)} - k - \gamma k\right) \\
&\leq a \cdot \exp\left(\frac{kb\alpha^2(4n - 4k(e^{\gamma+1}+1)) + 4e^{\gamma/2+1/2}k\sqrt{n-k}(2\alpha\sqrt{n}-\sqrt{k})}{n-k(e^{\gamma+1}+1)} - k - \gamma k\right) \\
&= a \cdot \exp\left(\frac{4e^{\gamma/2+1/2}k\sqrt{n-k}(2\alpha\sqrt{n}-\sqrt{k})}{n-k(e^{\gamma+1}+1)} + k(4b\alpha^2 - 1 - \gamma)\right) \\
&\leq a \cdot \exp\left(\frac{4e^{\gamma/2+1/2}k(2\alpha n-\sqrt{nk})}{n-k(e^{\gamma+1}+1)} + k(4b\alpha^2 - 1 - \gamma)\right) \\
&\leq a \cdot \exp\left(\frac{4e^{\gamma/2+1/2}k\alpha(2n-\sqrt{nk}/\alpha)}{n-k(e^{\gamma+1}+1)} + k(4b\alpha^2 - 1 - \gamma)\right) \\
&\leq a \cdot \exp\left(\frac{4e^{\gamma/2+1/2}k\alpha(2n-k/\alpha^2)}{n-k(e^{\gamma+1}+1)} + k(4b\alpha^2 - 1 - \gamma)\right) \\
&\leq a \cdot \exp\left(\frac{4e^{\gamma/2+1/2}k\alpha(2n-2k(e^{\gamma+1}+1))}{n-k(e^{\gamma+1}+1)} + k(4b\alpha^2 - 1 - \gamma)\right) \\
&= a \cdot \exp\left(k(4b\alpha^2 + 8e^{\gamma/2+1/2}\alpha - 1 - \gamma)\right) \\
&\leq a \cdot \exp\left(k(4b\alpha^2 - 1/2 - \gamma)\right) \\
&\leq a \cdot \exp\left(-\gamma k\right).
\end{aligned}
$$

□

To go from this lemma to $\sigma \leq e^{-\gamma k}$, we set $\gamma' = \gamma + \ln(a)$ in Lemma C.2 and observe that we get the same asymptotic lower bound on the corresponding α'.

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il