Estimating the Sortedness of a Data Stream

Parikshit Gopalan*
University of Texas at Austin
parik@cs.utexas.edu

T. S. Jayram
IBM Almaden
jayram@almaden.ibm.com

Robert Krauthgamer
IBM Almaden
robi@almaden.ibm.com

Ravi Kumar†
Yahoo! Research
ravikumar@yahoo-inc.com

Abstract

The *distance to monotonicity* of a sequence is the minimum number of edit operations required to transform the sequence into an increasing order; this measure is complementary to the length of the *longest increasing subsequence (LIS)*. We address the question of estimating these quantities in the one-pass data stream model and present the first sub-linear space algorithms for both problems.

We first present $O(\sqrt{n})$-space deterministic algorithms that approximate the distance to monotonicity and the LIS to within a factor that is arbitrarily close to 1. We also show a lower bound of $\Omega(n)$ on the space required by any randomized algorithm to compute the LIS (or alternatively the distance from monotonicity) *exactly*, demonstrating that approximation is necessary for sub-linear space computation; this bound improves upon the existing lower bound of $\Omega(\sqrt{n})$ [LNVZ06].

Our main result is a randomized algorithm that uses only $O(\log^2 n)$ space and approximates the distance to monotonicity to within a factor that is arbitrarily close to 4. In contrast, we believe that any significant reduction in the space complexity for approximating the length of the LIS is considerably hard. We conjecture that any deterministic $(1+\epsilon)$ approximation algorithm for LIS requires $\Omega(\sqrt{n})$ space, and as a step towards this conjecture, prove a space lower bound of $\Omega(\sqrt{n})$ for a restricted yet natural class of deterministic algorithms.

1 Introduction

The data stream model of computation, designed for analyzing massive data sets, has been intensively studied in the last few years; see the monograph by Muthukrishnan [Mut05] and the survey by Babcock et al. [BBD+02] for an overview of the area. Much of this research has focused on revisiting basic algorithmic problems and designing algorithms in the data stream model that are highly efficient with regard to storage space and update time. A problem that has received much attention in this model is estimating how close an input sequence is to being sorted [EKK+00, CMS01, AJKS02, GZ03, LNVZ06, ZLX+06]. There are several natural scenarios where this problem arises, especially in databases, information retrieval, and the web. For instance, consider a list of billions of dynamically changing web pages that are stored in ranked order according to some query-independent scoring function, like page-rank. If the contents of the web pages change, then applying the scoring function to the web pages in the list yields a new sequence of scores that may not be sorted. Since sorting is an expensive operation, an efficient procedure that estimates the deviation can be used to determine whether to re-sort the list.

The above example demonstrates that the problem can be quantified as the *distance* $D(\sigma, S(\sigma))$ between a sequence σ and a sequence $S(\sigma)$ obtained by sorting σ in non-decreasing order, where D is a metric defined on sequences. Efficient algorithms are known for estimating $D(\sigma, S(\sigma))$, the *distance to monotonicity*, in the data stream model for various metrics. Ajtai, Jayram, Kumar, and Sivakumar [AJKS02] and Gupta and Zane [GZ03] obtain algorithms for the Kendall distance (number of inversions); Cormode, Muthukrishnan, and Sahinalp [CMS01] obtain algorithms for transposition and inversion distances. However, a classical metric that has proved harder for algorithms is the *edit distance* [Gus97, Pev03, MP80, CM02]. In fact, edit distance is challenging even for related computational models such as property testing, sketching, and embedding [BEK+03, BYJKK04, ADG+03, OR05, KN05, KR06, CK06]. In this paper we consider the problem of estimating the distance to monotonicity under the edit

*Work done while the author was at Georgia Tech and IBM Almaden.

†Work done in part while the author was at IBM Almaden.

distance metric.

The edit distance between σ and $S(\sigma)$, denoted $\mathrm{ed}(\sigma)$, is the minimum number of edit operations that must be performed to transform σ into $S(\sigma)$ where each edit operation involves deleting a single element of the sequence and inserting it in a new place. If σ is a permutation, this is equivalent to the *Ulam distance* between σ and the identity permutation. The edit-distance metric is particularly suited for quantifying distance to monotonicity since it directly measures how many elements are out-of-place when taking a global view of the input. But it is precisely this property that seems to make it hard to compute on a data stream. The problem of estimating the distance to monotonicity under the edit distance on a data stream was raised in [AJKS02, CMS01]. Henceforth, distance to monotonicity will refer to the edit distance metric.

There is an alternative characterization of $\mathrm{ed}(\sigma)$ using the well-known concept of longest increasing sequence. Given a sequence σ of length n over an ordered alphabet, an *increasing sequence* in σ is a subsequence $i_1 < \cdots < i_k$ such that $\sigma(i_1) \leq \cdots \leq \sigma(i_k)$. Let $\mathrm{lis}(\sigma)$ denote the length of the *longest increasing sequence (LIS)* in σ. It can be shown that $\mathrm{ed}(\sigma) = n - \mathrm{lis}(\sigma)$, since the best way to sort the input is to identify an LIS and move around the elements not in this subsequence. One could require that an increasing sequence be strictly increasing ($\sigma(i_1) < \cdots < \sigma(i_k)$), all our results apply even with this definition. The LIS is an important and well-studied quantity in its own right; see the books by Gusfield [Gus97] and Pevzner [Pev03] for applications in bioinformatics and the survey by Aldous and Diaconis [AD99]. There is a classical algorithm for this problem, known as *Patience Sorting*, which can be viewed as a one-pass $O(n)$ space data stream algorithm. A natural question is if there is a more space-efficient data stream algorithm for this problem. Liben-Nowell, Vee, and Zhu [LNVZ06] studied this problem and showed an $\Omega(\sqrt{n})$ space lower-bound for computing $\mathrm{lis}(\sigma)$ exactly.

Thus prior to our work, there were no sub-linear space algorithms known for approximating either $\mathrm{lis}(\sigma)$ or $\mathrm{ed}(\sigma)$ in the data stream model; whereas the known lower bound even for exact computation was $\Omega(\sqrt{n})$.

Summary of results. We first consider the problem of estimating the distance to monotonicity in the one-pass data stream model. Our first result is, for arbitrary $\epsilon > 0$, a $(1+\epsilon)$-approximation deterministic algorithm for this problem using $O(\sqrt{n/\epsilon})$ space (Theorem 2.3). We also prove a lower bound of $\Omega(n)$ on the space required to *exactly* compute the distance to monotonicity, even if the algorithm is randomized (Theorem 4.2); this demonstrates that Patience Sorting is optimal for exact computation and that approximation is essential to achieve sub-linear space.

Our main result is a randomized algorithm that obtains a $(4+\epsilon)$-approximation, for any $\epsilon > 0$ using only $O(\epsilon^{-2} \log^2 n)$ space (Theorem 3.9).

For the problem of estimating the length of the LIS, we note that the deterministic data stream algorithm (Theorem 2.3) for estimating distance to monotonicity can be adapted to estimating the LIS, with identical guarantees. Since $\mathrm{lis}(\sigma) = n - \mathrm{ed}(\sigma)$, the linear space lower bound of Theorem 4.2 for exact computation, applies to the LIS problem as well. The main question is whether there exists a (possibly randomized) data stream algorithm that approximates the length of the LIS to within some constant factor using only polylogarithmic space. We are unable to resolve this question but conjecture that every such *deterministic* algorithm requires $\Omega(\sqrt{n})$ space. As a step towards this conjecture, we show that the lower bound holds for a restricted yet large natural classes of deterministic algorithms (Theorem 4.6); in particular, this class includes all the algorithms in this paper.

Techniques. Our randomized algorithm builds on a body of work in property testing of monotonicity [EKK+00, GGL+00, FLN+02]. A first approach to computing the distance to monotonicity might be to relate it to the number of inversions; unfortunately, these quantities can be far apart. Ailon, Chazelle, Comandur, and Liu [ACCL04], building upon earlier work by Ergun, Kannan, Kumar, Rubinfeld, and Viswanathan [EKK+00], show that a variation of this idea can be used to give a 2-approximation to the distance to monotonicity; they consider the set of indices that are endpoints of some interval where the majority of the input elements within that interval are inverted with respect to the endpoint. This algorithm can be realized in the data stream model, however, it uses $O(n)$ space. The obvious bottleneck is that to decide if an index is a left endpoint requires knowledge of the elements that will come in the future.

We show that it actually suffices to consider only those indices that are right endpoints for a slight loss in the approximation, namely, the number of such right endpoints is a 4-approximation to the distance to monotonicity. We then devise a sampling scheme to test whether a given index is a *right endpoint* of such an interval. This scheme is similar in spirit to reservoir sampling [Vit85], which solves the following problem: given access to a set of elements arriving in streaming fashion, produce at any point in time random elements from the set of all elements seen so far, using small memory. Our scheme is more involved since we need produce samples from different subsets of all the elements seen so far, namely from the last k elements

seen for numerous different values of $k > 0$.

Our $O(\sqrt{n})$-space deterministic algorithms for the distance to monotonicity and the LIS problem can be viewed as a bounded-space version of Patience Sorting. To motivate these algorithms, we formulate a t-player one-way communication problem, where each player is given a sequence and they wish to approximate the distance to monotonicity (alternatively, the LIS) of the concatenation of their sequences. Our protocol for this problem keeps track of only a few *candidate increasing sequences* in the input and the correctness is argued using a a careful induction. Our deterministic data stream algorithm can be viewed as a simulation of this protocol with $t = \sqrt{n}$ players. While this algorithm needs to know the length of the data stream in advance, we also present a modification that uses slightly more space, but does not need to know the length in advance.

An interesting aspect of the above protocol is that the maximum message size sent by any player *increases* with the number of players. This points to the difficulty in proving a space lower bound for the LIS problem because, in contrast, for most communication complexity problems such as multi-player set-disjointness for which tight lower bounds have been shown [BYJKS04, CKS03], the maximum communication *decreases* with more players. Indeed, the only other problem we are aware of that shares this behavior is the problem of approximately counting the number of 1's in a data stream of length n. Ajtai [Unpublished manuscript, 2002] has shown a lower bound of $\Omega(\log n)$ for deterministic algorithms for this problem; however his proof is combinatorial and does not address the multi-player setting directly. We formulate a problem in communication complexity such that strong lower bounds for this problem will imply the conjectured lower bound of $\Omega(\sqrt{n})$ for the LIS. We use this approach to establish the conjecture for a natural class of algorithms using new combinatorial techniques.

For lack of space, some results and proofs are omitted from the current version. In another paper that appears in these proceedings, Sun and Woodruff [SW07] studied the problems of computing the LIS and LCS (longest common subsequence) in the data-stream model. However, their goal is to obtain algorithms whose space complexity is optimal in terms of the length of the LIS, as opposed to the length of the entire input sequence as in our work. While the main results of the two papers are different, there is some overlap; Sun and Woodruff also analyze the two-player protocol for approximating the LIS (Lemma 2.1) and give an $\Omega(n)$ lower bound for exact computation of the LIS (Lemma 4.1).

2 Deterministic approximation algorithms

We obtain a deterministic $O(\sqrt{n})$-space algorithm to approximate LIS. We first describe Patience Sorting, followed by a communication protocol to solve a multi-party communication version of the LIS problem. Our algorithm can be viewed as a simulation of this protocol with $O(\sqrt{n})$ players.

Patience Sorting. Let $\{1, \ldots, m\}$ be the alphabet. In analyzing the space complexity of our algorithms, we will suppress the dependency on m, and state the space used in terms of the number of input symbols stored. The number of bits stored is bounded by this quantity multiplied by $\log m$. Given a sequence σ, for $i \in \{1, \ldots, n\}$ let $P_\sigma(i)$ be the smallest letter a in the alphabet such that there is an increasing sequence of length i in σ ending at a. For $i > \mathrm{lis}(\sigma)$ there is no increasing sequence in σ of length i, so we will set $P_\sigma(i) = \infty$. Note that P_σ is an array of letters and it is increasing. If σ is clear from the context, we will just use $P(i)$. PatienceSort is based on computing $P(i)$; see [AD99] for more details.

Algorithm PatienceSort(σ)
1. Set $P(i) = \infty$ for all i
2. For $j = 1, \ldots, n$, read $\sigma(j)$ and find the largest i so that $P(i) \leq \sigma(j)$ and set $P(i+1) = \sigma(j)$
3. Output the largest i so that $P(i) \neq \infty$

A multi-player protocol to approximate LIS. Consider the following two-player communication complexity problem. Alice is given a string σ_1, Bob is given a string σ_2 and they wish to compute a $(1-\epsilon)$-approximation to $\mathrm{lis}(\sigma)$ where $\sigma = \sigma_1 \circ \sigma_2$. We will give a one-way protocol for this problem that uses $\epsilon^{-1}(\log m + \log n)$ bits of communication. Alice first runs PatienceSort on σ_1, computes $k_1 = \mathrm{lis}(\sigma_1)$, and sends $\langle i, P_{\sigma_1}(i)\rangle$ for all $i \in \{\epsilon k_1, 2\epsilon k_1, \ldots, k_1\}$. Using these tuples, Bob then computes the best extension of these sequences, in addition to the empty sequence, by σ_2 and outputs k_2 which is the length of the longest sequence.

Lemma 2.1. *The value k_2 output by Bob satisfies* $\mathrm{lis}(\sigma) \geq k_2 > (1-\epsilon)\,\mathrm{lis}(\sigma)$.

Proof. Assume the LIS is of the form $\pi_1 \circ \pi_2$ where π_1 is a substring of σ_1 and π_2 is a substring of σ_2. Assume that $|\pi_1| = \ell_1$ and $|\pi_2| = \ell_2$ so that $\mathrm{lis}(\sigma) = \ell_1 + \ell_2$.

Let $\pi_1(\ell_1) = a$ and $\pi_2(1) = b$ so that $a \leq b$. Choose ℓ_1' to be a multiple of ϵk_1 such that $\ell_1 - \epsilon k \leq \ell_1' \leq \ell_1$. Since π_1 is an increasing sequence, $\pi_1(\ell_1') \leq a$. Let $P_{\sigma_1}(\ell_1') = a'$. From the definition of P_{σ_1}, $a' \leq \pi_1(\ell_1')$. Hence we have $a' \leq \pi_1(\ell_1') \leq a \leq b$. Thus Alice's message tells Bob that σ_1 contains an increasing sequence of length ℓ_1' ending at $a' \leq a$. Bob can extend

this sequence by π_2 to get an increasing sequence of length $\ell_1' + \ell_2$. Thus $k_2 \geq \ell_1' + \ell_2 \geq \ell_1 - \epsilon k_1 + \ell_2 \geq \mathrm{lis}(\sigma) - \epsilon k_1 \geq (1-\epsilon)\,\mathrm{lis}(\sigma)$, where the last inequality holds since $\mathrm{lis}(\sigma) \geq k_1$. □

A stronger statement is true: Bob can estimate the length of the longest sequence ending at or before a for every $a \in \{1, \ldots, m\}$ within an additive error of ϵk_1.

The above protocol can be generalized to $t > 2$ players; we give a brief overview of the t-player one-way protocol. The players now compute an array Q_σ which is an approximation to P_σ, based on the messages received from the other players and their own inputs. If k_j is the longest increasing sequence detected by the j-th player, then he sends the values $Q_\sigma(i)$ for all $i \in \{\frac{\epsilon}{t-1}k_j, \frac{2\epsilon}{t-1}k_j, \ldots, k_j\}$. Based on this information and his own input, player P_{j+1} will update the values of Q_σ. One can prove that each player introduces an additive error of at most $\frac{\epsilon}{t-1}\,\mathrm{lis}(\sigma)$, thus the overall additive error is bounded by $\epsilon\,\mathrm{lis}(\sigma)$. The max communication complexity of this protocol is bounded by $\frac{t}{\epsilon}$. Since each player's input consists of $\frac{n}{t}$ numbers, simulating this protocol using a data stream algorithm will require space $\max(\frac{n}{t}, \frac{t}{\epsilon})$. Thus the optimal setting is to take $t = \sqrt{\epsilon n}$.

2.1 The deterministic algorithm We present our algorithm directly as a data stream algorithm, rather than as a simulation of a communication protocol. The algorithm can be viewed as a variant of PatienceSort with only bounded space available. We compute values $Q(i)$ that are meant to approximate $P(i)$. However unlike in PatienceSort, we ensure that the number of indices i for which $Q(i)$ is stored is never more than $O(\sqrt{n})$. The algorithm proceeds similar to PatienceSort, but if the number of stored values exceeds this bound, then a cleanup operation is invoked, where only the values of Q for $O(\sqrt{n})$ evenly spaced indices i are retained.

Algorithm ApproximateLIS(σ)
1. Set $S = \{0\}, Q(0) = 0$
2. For $j = 1, \ldots, n$
3. Read $\sigma(j)$ and find the largest $i \in S$ so that $Q(i) \leq \sigma(j)$ and set $Q(i+1) = \sigma(j)$; if $i+1 \notin S$, add it to S
4. If $|S| > 2\sqrt{n/\epsilon}$, let $k = \max\{i \in S\}$, $S = \{\lceil\sqrt{\epsilon/n}k\rceil, \lceil 2\sqrt{\epsilon/n}k\rceil, \ldots, k\}$, and store $Q(i)$ only for $i \in S$
5. Output $k = \max\{i \in S\}$

Note that we store Q only for the indices in S. For the purpose of analysis, consider the following "interpolated" function P' on the interval $[1, k]$, where k is the largest value in S, using the following definition: for $i \in S$ set $P'(i) = Q(i)$ and for $i \notin S$ set $P'(i) = Q(j)$ for the smallest $j > i$ that lies in S. The motivation for this definition is that $P'(i)$ is the smallest letter a so that the algorithm detects an increasing sequence of length i ending at a. If $P'(j) = a$ and $i < j$ then certainly we have $P'(i) \leq a$.

We refer to each execution of step 4 as a cleanup operation. Between two consecutive cleanup operations, the set S grows by $\sqrt{n/\epsilon}$, hence the value of j increases by at least this amount. Since the stream is of length n, in total there are no more than $\sqrt{\epsilon n}$ cleanup operations. Based on when the cleanups occur, we can break the string σ as $\sigma_1 \circ \cdots \circ \sigma_{\sqrt{\epsilon n}}$ (the last few might be empty). We bound the performance of the algorithm by comparing it to PatienceSort. This is done by comparing the values of P' and P prior to every cleanup operation. Let P_t' denote the function P' after processing $\sigma_1 \circ \cdots \circ \sigma_t$ and prior to the t-th cleanup. Let us denote $P_{\sigma_1 \circ \cdots \circ \sigma_t}$ by P_t. Let k_t be the largest value in S at this point. We have $k_1 \leq \cdots \leq k_t$ since the algorithm never discards the largest element in S. Also $k_t \leq \mathrm{lis}(\sigma)$ since our algorithm detects an increasing sequence of length k_t. The intuition for the analysis is that at every step, the algorithm maintains an additive approximation to $L(a)$. Each new cleanup operation causes the error to increase, but it can be bounded by $t\sqrt{\epsilon/n}k_{t-1}$ after t cleanup operations.

Lemma 2.2. *For $t \leq \sqrt{\epsilon n}$,*

$$P_t'\left(i - (t-1)\sqrt{\epsilon/n}k_{t-1}\right) \leq P_t(i).$$

Proof. The base case when $t = 1$ is trivial, since in this case $P_1'(i) = P_1(i)$. Assume by induction that the lemma holds for $t-1$. For simplicity, we will ignore the rounding operator in step 4.

Consider what happens to P and P' after processing the string σ_t. After the $(t-1)$-st cleanup, the memory contains the value of P_{t-1}' for all multiples of $\sqrt{\epsilon/n}k_{t-1}$. P_t' is obtained by computing the best possible extensions of these sequences using σ_t.

Let $P_t(i) = c$. Consider the increasing sequence of length i ending at c. Split this sequence into $\pi_1 \circ \pi_2$ where π_1 lies in $\sigma_1 \circ \cdots \circ \sigma_{t-1}$ and π_2 lies in σ_t. Assume that they have length ℓ_1 and ℓ_2 respectively so that $\ell_1 + \ell_2 = i$. Let b be the first letter of π_2 and a the last letter of π_1 so that $a \leq b \leq c$. Assume that $P_{t-1}(\ell_1) = a$, since if there is an increasing sequence of length ℓ_1 that ends earlier than a, we could replace π_1 by it. Applying the induction hypothesis to $P_{t-1}'(\ell_1)$,

$$P_{t-1}'\left(\ell_1 - (t-2)\sqrt{\frac{\epsilon}{n}}k_{t-2}\right) \leq P_{t-1}(\ell_1) = a. \tag{2.1}$$

We can find ℓ_1' which is a multiple of $\sqrt{\epsilon/n}k_{t-1}$ satisfy-

ing

(2.2)

$$\ell_1-(t-2)\sqrt{\frac{\epsilon}{n}}k_{t-2}-\sqrt{\frac{\epsilon}{n}}k_{t-1} \;\le\; \ell'_1 \;\le\; \ell_1-(t-2)\sqrt{\frac{\epsilon}{n}}k_{t-2}.$$

Since $k_{t-1} \ge k_{t-2}$, we can lower bound ℓ'_1 by $\ell'_1 \ge \ell_1-(t-1)\sqrt{\frac{\epsilon}{n}}k_{t-1}$. Since P'_{t-1} is a monotone function on $i \in [1, k_{t-1}]$, from (2.1) and (2.2) we get

$$P'_{t-1}(\ell'_1) \;\le\; P'_{t-1}\left(\ell_1-(t-2)\sqrt{\frac{\epsilon}{n}}k_{t-2}\right) \;\le\; a.$$

Since ℓ'_1 is a multiple of $\sqrt{\epsilon/n}k_{t-1}$, this value is stored in the memory even after the $(t-1)$-st cleanup. Extending this sequence using π_2, we get an increasing sequence ending at a of length

$$\ell'_1+\ell_2 \;\ge\; \ell_1-(t-1)\sqrt{\frac{\epsilon}{n}}k_{t-1}+\ell_2 \;\ge\; i-(t-1)\sqrt{\frac{\epsilon}{n}}k_{t-1}$$

Hence $P'_t(i-(t-1)\sqrt{\epsilon/n}k_{t-1}) \;\le\; a \;= P_t(i)$, which completes the induction. □

THEOREM 2.3. *There is a one-pass deterministic algorithm that computes a $(1-\epsilon)$-approximation to the LIS using space $O(\sqrt{n/\epsilon})$ for any $\epsilon > 0$.*

Proof. Assume that $\mathrm{lis}(\sigma) = k$ and the LIS ends at a so that $P(k) = a$. Assume that the total number of cleanups is $t \le \sqrt{\epsilon n}$. Applying Lemma 2.2, we get $P(k - t\sqrt{\epsilon/n}k_{t-1}) \le a$. Hence the algorithm detects an increasing sequence of length k_t where $k_t \;\ge\; k - t\sqrt{\epsilon/n}k_{t-1} \;\ge\; k - \epsilon k_{t-1} \;\ge\; (1-\epsilon)k$, where the last inequality uses $k_{t-1} \le k$. □

The above algorithm needs to know n in advance. We can derive an algorithm that does not need to know n in advance, yet works in space $O(n^{1/2+\delta})$, where δ is any constant; we omit the details in this version.

Our algorithm can be modified to approximate the distance to monotonicity instead of the LIS. Define Q_σ to be an array of letters with $Q_\sigma(i) = a$ if a is the smallest letter so that one can get a monotone sequence of length i ending at a by deleting i elements of σ, and $Q_\sigma(i) = \infty$ if $i < \mathrm{ed}(\sigma)$. Note that $Q_\sigma(i) = P_\sigma(n-i)$. The approximation algorithm computes an approximation Q' to Q and stores the values of Q' for $O(\sqrt{n})$ indices. We omit the details in this version.

3 Randomized algorithm for distance to monotonicity

We present a randomized $(4+\epsilon)$-approximation algorithm for distance to monotonicity in $O(\epsilon^{-2}\log^2 n)$ space (Theorem 3.9). We obtain a characterization of distance to monotonicity via inversions, and use it to obtain a randomized data stream algorithm.

3.1 Characterization via inversions

A pair of indices (i,j) is said to be *inverted* in σ if either $i < j$ and $\sigma(i) > \sigma(j)$, or $i > j$ and $\sigma(i) < \sigma(j)$. For an index i, let $\mathrm{inv}(i)$ denote the set of indices j that are inverted with respect to i. For $\delta \le 1/2$, define a set R_δ containing all indices i that are the right endpoints of an interval where more than a δ-fraction of the elements lie in $\mathrm{inv}(i)$, i.e., the set of i such that there is a j for which more than δ-fraction of indices in $[j, i-1]$ belong to $\mathrm{inv}(i)$. Let $R = R_{1/2}$, i.e., R contains all indices i that are the right endpoints of an interval, a strict majority of whose elements lie in $\mathrm{inv}(i)$. Notice that if $(i-1, i)$ is an inversion then $i \in R$.

LEMMA 3.1. *There is a procedure that deletes at most $2|R|$ indices in σ so as to obtain an increasing sequence.*

Proof. Assume without loss of generality that $\sigma(n+1) = m$ and so $n+1 \notin R$. The procedure is as follows. Scan the string from right to left, starting at $i = n+1$. At every iteration, if $i-1 \notin R$, proceed to $i-1$. Otherwise, skip to the next index j (namely, largest $j < i$) that is not in $\mathrm{inv}(i) \cup R$, deleting all the skipped characters (namely, indices in $[j+1, i-1]$). The procedure stops once the sequence is exhausted (i.e., index 1 is reached).

We claim that a majority of the indices that are deleted at any step lie in R. To see this, consider a single iteration. Clearly, $i \notin R$ and let j be the largest index such that $j < i$ and j does not belong to $\mathrm{inv}(i) \cup R$. Every index in $[j+1, i-1]$ lies in $\mathrm{inv}(i)$ or in R. But since $i \notin R$, at least half the indices from $[j+1, i-1]$ are not inverted with respect to i, and thus lie in R.

This procedure returns a subsequence $(i_1, \ldots, i_k)$ such that every two successive indices $(i_{\ell-1}, i_\ell)$ is not an inversion, and thus in increasing order. It follows that the entire sequence is increasing. □

Fix a set $D \subseteq [n]$ of indices of size $\mathrm{ed}(\sigma)$ such that deleting D leaves an increasing sequence. Note that D need not be unique. Define a set S_δ containing all indices $i \notin D$ that are the right endpoints of an interval where more than a δ-fraction of the elements lie in D, i.e., the set of i such that there is a j for which more than δ-fraction of indices in $[j, i-1]$ lie in D. An index j satisfying this last condition will be called a *witness* for the membership of i in S_δ.

LEMMA 3.2. *For all $\delta \le 1/2$,*

$$|R_\delta \setminus D| \le |S_\delta| \le (1-\delta)/\delta \cdot |D|.$$

Proof. [Sketch] For the first inequality, notice that the subsequence of σ induced by the indices not in D forms an increasing sequence, and thus for every $i \notin D$ we

have $\text{inv}(i) \subseteq D$. It follows that $i \in R_\delta \setminus D$ implies $i \in S_\delta$, and thus $R_\delta \setminus D \subseteq S_\delta$.

For the second inequality, we give a procedure to compute the set S_δ. Scan the sequence left to right, starting at the smallest index $j \in D$. At every iteration, find the next index k (i.e., smallest index $k > j$) such that at most δ-fraction of $[j, k-1]$ lies in D, and add the indices in $[j, k-1] \setminus D$ to S_δ; then let ℓ be the smallest index greater than k that lies in D, and set $j = \ell$ and proceed to the next iteration. The procedure stops once the sequence is exhausted (i.e., index n is reached).

One can show that S_δ is computed correctly via simple but tedious case analysis, which we defer to the full version. Since for every t indices added to S_δ at an iteration, at least $(1-\delta)(t/\delta)$ indices of D are skipped over, we have $|S_\delta| \leq (1-\delta)/\delta \cdot |D|$. □

THEOREM 3.3. *For all $\delta \leq 1/2$,*

$$\text{ed}(\sigma)/2 \;\leq\; |R| \;\leq\; |R_\delta| \;\leq\; \text{ed}(\sigma)/\delta \tag{3.3}$$

Proof. Lemma 3.1 shows that $\text{ed}(\sigma) \leq 2|R|$. By definition, we have $R \subseteq R_\delta$. Using Lemma 3.2 and recalling that $|D| = \text{ed}(\sigma)$, we obtain $|R_\delta| = |R_\delta \cap D| + |R_\delta \setminus D| \leq |D| + (1-\delta)/\delta \cdot |D| = \text{ed}(\sigma)/\delta$. □

The first inequality in Equation 3.3 is new, whereas the last inequality is implied by results of Ailon et al. [ACCL04] via a different proof. Both inequalities are tight. For the first one, let $k < n/4$ and take $\pi = k+1, \ldots, n/2, n, \ldots, n-k+1, 1, \ldots, k, n/2+1, \cdots, n-k$. Here $\text{ed}(\pi) = 2k$ whereas $|R| = k$. For the last inequality, let $k < \delta n$ and take $\sigma = n, \ldots, n-k+1, 1, \ldots, n-k$. Here $\text{ed}(\sigma) = k$ whereas $|R_\delta| = k/\delta - 2$.

3.2 The randomized algorithm Theorem 3.3 immediately suggests a natural algorithm for estimating $\text{ed}(\sigma)$: at every time i, check whether there exists an interval $I = [j, i-1]$ in which a majority of the elements are inverted with respect to i. Observe that the decision at time i depends only on elements already seen at that time, which is very suitable for a data stream implementation. Unfortunately, this naive algorithm uses linear space and hence undesirable. Below, we improve upon this idea to obtain an algorithm that uses only polylogarithmic space. Here is the basic idea. First, the "slack" between R and R_δ in Theorem 3.3 effectively allows us to replace the majority decision for an interval I with an estimate based on a sample of $O(\log i)$ elements from I. At first cut, it appears that at every time i we need a random sample from every interval $[j, i-1]$, which is clearly prohibitive. But the crucial observation is that the samples obtained for different intervals *need not be independent*. We exploit this fact by maintaining, at each time i, only $O(\log^2 i)$ samples from $\sigma(1), \ldots, \sigma(i-1)$.

Let ϵ be such that $0 < \epsilon < 1/6$. We give below an algorithm that achieves $4+O(\epsilon)$ approximation. Theorem 3.9 would then follow immediately by scaling ϵ appropriately together with a slightly improved implementation. Let $C = C(\epsilon)$ be a parameter to be determined later so that $C\epsilon^2$ is a sufficiently large constant.

A bucket of samples. The algorithm maintains a sample of the already seen elements of σ in a *bucket* B. The fact that $\sigma(j)$ is retained in the bucket is denoted by $j \in B$; note that the algorithm actually maintains in B a record of the tuple $\langle j, \sigma(j) \rangle$. For $j < i$ define

$$p(j,i) = \min\left(1, \frac{C \cdot \log(2i)}{(i-j)}\right)$$

and let $p(i,i) = 1$. The bucket B will be maintained so that at each time i it has the following distribution: for all $j \leq i$,

$$\Pr[j \in B \text{ at time } i\,] = p(j,i)$$

Furthermore, for every time i, the events $\{j \in B \text{ at time } i\,\}$ for $j \in [i]$ are mutually independent of each other. However, there is no such requirement for events corresponding to different times i, and in fact the bucket B at time i will be quite similar to that at time $i-1$.

Note that for every $j \leq i$, we have $p(j,i) \geq p(j, i+1)$. Thus, if the bucket has the right distribution at time i, we can obtain the right distribution at time $i+1$ by retaining each $j < i$ already in the bucket independently with probability $p(j,i)/p(j,i+1)$ and then adding i to the bucket. The following Lemma upper bounds the number of samples retained in the bucket; it immediately implies a similar bound on the space (storage requirement) of the algorithm.

LEMMA 3.4. *Fix an i and let B be the bucket at time i. Then, $\mathbb{E}[|B|] \leq C \log^2(2i)$ and $|B| = O(C \log^2(2i))$ with probability at least $1 - (2i)^{-\Omega(C \log i)}$.*

Estimating majority in one interval. We next describe procedure TESTMAJORITY(j, i) that uses only the bucket B to test whether a near-majority of elements in $I_j = [j, i-1]$ are inverted with respect to i. More precisely, it distinguishes between the case where the fraction of inversions is more than $1/2$ and where it is at most $1/2 - 3\epsilon$. The procedure, described below, operates as follows: it first generates a random sample S of elements from the interval I_j by an appropriate selection from the bucket B (line 2), and then uses the fraction of elements in S that are inverted with respect

to i as an estimate for the corresponding fraction in I_j (lines 4-5).

Procedure TESTMAJORITY(j,i)
1. Initialize the set S to be empty
2. For each $k \in B$, if $j \le k \le i-1$, add k to S with probability $p(j,i)/p(k,i)$.
4. If at least $(\frac{1}{2}-\epsilon)$ fraction of elements $k \in S$ satisfy $\sigma(k) > \sigma(i)$ then return true
5. return false

Now we analyze this procedure. Recall that each element $k \in I_j$ is retained in the bucket independently with probability $p(k,i)$. Thus, a random sample that is uniform over I_j can be generated by picking each element $k \in B \cap I_j$ independently with probability inversely proportional to $p(k,i)$, namely, with probability $p(j,i)/p(k,i)$. We thus obtain.

LEMMA 3.5. *Fix $j < i$. Then the corresponding set S has the following (marginal) distribution:* $\Pr[k \in S] = p(j,i)$, *if $k \in I_j$ and 0 otherwise. Furthermore, the events for different k (but same i and j) are mutually independent.*

We use Lemma 3.5 to show that $|S \cap \mathrm{inv}(i)|/|S|$ is a fairly good approximation to $|I_j \cap \mathrm{inv}(i)|/|I_j|$. We bound the error probability of the test by $(2i)^{-O(1)}$ where the hidden constant depends on ϵ and C.

LEMMA 3.6. *If more than $1/2$ fraction of I_j lies in* $\mathrm{inv}(i)$, *then*

$$\Pr[\text{TESTMAJORITY}(j,i) = \text{true}] \ge 1 - (2i)^{-\Omega(\epsilon^2 C)}.$$

If less than $(1/2 - 3\epsilon)$ fraction of I_j lies in $\mathrm{inv}(i)$, *then*

$$\Pr[\text{TESTMAJORITY}(j,i) = \text{false}] \ge 1 - (2i)^{-\Omega(\epsilon^2 C)}.$$

Approximating the distance to monotonicity. We describe algorithm APPROXIMATEDIST(σ) that uses the procedure TESTMAJORITY(j,i) to estimate the distance to monotonicity of σ. The algorithm, described below, maintains a count d that estimates for how many values i, there exists at least one $j \in [i-1]$ such that TESTMAJORITY(j,i) = true. More specifically, after the i-th element is read, the algorithm updates the bucket B (line 3) so that it has the right distribution, and determines whether to increment d by 1 (line 4). A naive testing of every $j \in [i-1]$ would make the update time (per input element) linear in i. A more efficient way is to test only values $j \in B$ and argue that they are representative of all $j \in [i-1]$ (by relating the outcome of TESTMAJORITY(j',i) for $j' \notin B$ with that of TESTMAJORITY(j',i) for a nearby $j \in B$). This would require testing only $O(|B|)$ values of j, and as we saw in Lemma 3.4, $|B| \le O(C \log^2 i)$ with high probability. This can be further reduced to testing only $O(\log i)$ values of j, using an idea from Ailon et al. [ACCL04]: try only j's for which the length of the interval $[j, i-1]$ changes by a factor of $1+\epsilon_1$, where $\epsilon_1 = \epsilon/4$. Specifically, define $T(i) = \{i-1, \lfloor \frac{i-1}{1+\epsilon_1} \rfloor, \lfloor \frac{i-1}{(1+\epsilon_1)^2} \rfloor, \ldots, 1\}$, and observe that $|T(i)| = O(\epsilon^{-1} \log i)$.

Algorithm APPROXIMATEDIST(σ)
1. Initialize the bucket B to be empty and set $d = 0$
2. For $i = 1, \ldots, n$
3. Remove from B each $j \in B$ independently with probability $1 - \frac{p(j,i)}{p(j,i-1)}$
4. Add i to B
5. For each $t \in T(i)$, compute TESTMAJORITY$(i-t,i)$. If at least one of them returns true, then set $d = d+1$.
6. Output d

We now analyze the correctness.

LEMMA 3.7. *Let $\hat{R}$ denote the set of indices i that cause d to increase. With probability at least* $1 - \sum_{i=1}^{n} (2i)^{1-\Omega(\epsilon^2 C)}$,

$$R \subseteq \hat{R} \subseteq R_{1/2-3\epsilon}.$$

Proof. Fix $i \in R$ and let j be a witness to this. Hence a majority of elements from $[j, i-1]$ are in $\mathrm{inv}(i)$. Consider first the simpler case where $i - j \in T(i)$. Then the algorithm runs TESTMAJORITY(j,i), which by Lemma 3.6 returns true with probability at least $1 - (2i)^{-\Omega(\epsilon^2 C)}$. In general, pick the smallest $t \in T(i)$ such that $t \ge i - j$. Observe that necessarily $t \le (1+\epsilon_1)(i-j)$, and therefore the fraction of elements from $[i-t, i-1]$ that lie in $\mathrm{inv}(i)$ is more than $\frac{(i-j)/2}{t} \ge \frac{1}{2(1+\epsilon_1)} > 1/2 - \epsilon_1/2$. Observe that the analysis of Lemma 3.6 goes through even with such a weaker assumption (namely, that $\frac{|I_j \cap \mathrm{inv}(i)|}{|I_j|} > 1/2 - \epsilon/4$), and therefore TESTMAJORITY$(i-t,i)$ returns true with probability at least $1 - (2i)^{-\Omega(\epsilon^2 C)}$. We get that $\Pr[i \notin \hat{R}] \le (2i)^{-\Omega(\epsilon^2 C)}$.

Now fix $i \notin R_{1/2-3\epsilon}$. Then for every $j < i$, fewer than $1/2 - 3\epsilon$ elements in $[j, i-1]$ belong to $\mathrm{inv}(i)$. By applying Lemma 3.6 and taking a union bound, the probability that for at least one $j \in [i-1]$ the execution of TESTMAJORITY(i,j) = true is at most $(2i)^{1-\Omega(\epsilon^2 C)}$. Recalling that we assumed $\epsilon^2 C$ is sufficiently large, we get that $\Pr[i \in \hat{R}] \le (2i)^{-\Omega(\epsilon^2 C)}$.

Finally, the lemma follows by applying a union bound over all $i \in [n]$. □

By setting $C = C_0 \epsilon^{-2} \log(1/\delta)$ for a sufficiently large constant C_0, we get the following theorem.

THEOREM 3.8. *For every $0 < \epsilon < 1/6$ and $0 < \delta < 1$, algorithm* APPROXIMATEDIST(σ) *computes a $(4+O(\epsilon))$-approximation to $\mathrm{ed}(\sigma)$ with probability at least $1-\delta$. At each time i, with probability at least $1-(2i)^{-\Omega(\epsilon^2 C)}$, the algorithm uses $O(C\log^2 i)$ space and its update time is $O(C\epsilon^{-1}\log^3 i)$.*

Proof. Recall that $\hat{R}$ denotes the set of indices i that cause d to increase. By Lemma 3.7 and the choice of C, we know that $R \subseteq \hat{R} \subseteq R_{1/2-3\epsilon}$ holds with probability at least $1-\sum_i (2i)^{-O(\epsilon^2 C)} \geq 1-\delta$. By Theorem 3.3, we have $|R| \geq \mathrm{ed}(\sigma)/2$ and $|R_{1/2-3\epsilon}| \leq \frac{2}{1-6\epsilon}\,\mathrm{ed}(\sigma)$. Hence with probability $1-\delta$ the algorithm approximates $\mathrm{ed}(\sigma)$ within factor $4+O(\epsilon)$.

It is easily seen that the algorithm's space requirement is $O(|B|)$, which by Lemma 3.4 is $O(C\log^2 i)$ with high probability. Similarly, for each element i the algorithm consists of updating the bucket B by scanning it linearly, and $|T(i)| = O(\epsilon^{-1}\log i)$ calls to procedure TESTMAJORITY, each of which requiring a linear scan of the bucket B. Using Lemma 3.4 and applying a union bound, we conclude that the update time for element i is $O(C\epsilon^{-1}\log^3 i)$, with high probability. □

If the length of the data stream n is known in advance, we can set the sampling probabilities as $p(j,i) = \min\left(1, \frac{C\log(2n)}{i-j}\right)$. Then the bounds above will hold with probability $1-n^{-\Omega(1)}$, the space used is $O(C\log^2 n)$ and the update time is $O(C\epsilon^{-1}\log^3 n)$. The full version contains an improvement to the update time per element and a more efficient sampling procedure, which gives:

THEOREM 3.9. *There is a randomized algorithm that computes a $4+\epsilon$ approximation to the distance to monotonicity for any $\epsilon > 0$ with probability arbitrarily close to 1. The space used is $O(\epsilon^{-2}\log^2 n)$ and update time per element is $O(\epsilon^{-3}\log^2 n)$ where n is the length of input.*

4 Lower bounds

4.1 Lower bounds for exact algorithms. First consider the case when σ is not a permutation. Consider the communication problem where Alice is given the first half of σ, Bob the second half and they wish to compute $\mathrm{lis}(\sigma)$. We give a reduction from the problem of computing the AND of two bits to computing the LIS when $n=2$ and $m=4$. Alice holds a bit x, Bob holds a bit y and they want to compute $x \wedge y$. Let

$$\sigma(1) = \begin{cases} 4 \text{ if } x = 0, \\ 2 \text{ if } x = 1. \end{cases} \qquad \sigma(2) = \begin{cases} 1 \text{ if } y = 0, \\ 3 \text{ if } y = 1. \end{cases}$$

Let $\sigma = \sigma(1), \sigma(2)$. Then clearly, $\mathrm{lis}(\sigma) = 2$ if $x \wedge y = 1$ and $\mathrm{lis}(\sigma) = 1$ otherwise.

We extend this to reduce set disjointness to computing $\mathrm{lis}(\sigma)$. Assume that Alice and Bob have strings $x = x_1, \ldots, x_n$ and $y = y_1, \ldots, y_n$ respectively that represent the incidence vectors of sets X, Y on $[n]$. Alice and Bob compute $\sigma(i)$ and $\sigma(n+i)$ for $1 \leq i \leq n$ as follows:

$$\sigma(i) = \begin{cases} 4(i-1)+4 \text{ if } x_i = 0, \\ 4(i-1)+2 \text{ if } x_i = 1. \end{cases}$$

$$\sigma(n+i) = \begin{cases} 4(i-1)+1 \text{ if } y_i = 0, \\ 4(i-1)+3 \text{ if } y_i = 1. \end{cases}$$

Let $\sigma = \sigma(1), \ldots, \sigma(2n)$.

LEMMA 4.1. *If $X \cap Y \neq \emptyset$, then $\mathrm{lis}(\sigma) = n+1$. If $X \cap Y = \emptyset$, then $\mathrm{lis}(\sigma) = n$.*

Proof. Assume that X and Y are disjoint. We claim that any increasing sequence can contain at most one element from each interval $[4(i-1)+1, 4(i-1)+4]$ for $1 \leq i \leq n$. This is because the string σ contains precisely two elements in this interval $\sigma(i)$ and $\sigma(n+i)$ and they are in increasing order iff $x_i \wedge y_i = 1$. Hence when $X \cap Y$ is empty, only one of them can occur in any increasing sequence and $\mathrm{lis}(\sigma) \leq n$. Equality holds since $\sigma(1), \ldots, \sigma(n)$ is an increasing sequence.

On the other hand, assume that $i \in X \cap Y$. The following is an increasing subsequence of σ: $\sigma(1), \ldots, \sigma(i), \sigma(n+i), \ldots, \sigma(2n)$. It is easy to check that in fact $\mathrm{lis}(\sigma) = n+1$. □

The resulting sequences are not permutations. However it is possible to reduce computing the AND of two bits to LIS for permutations for $n = 8$; we omit the details in this version. We use this to show:

THEOREM 4.2. *Given a permutation σ over the alphabet $[8n]$, any randomized algorithm to decide whether $\mathrm{lis}(\sigma)$ is $2n+1$ or $2n+2$ requires space $\Omega(n)$ even if multiple passes are allowed.*

4.2 Lower bounds for deterministic approximation algorithms.

CONJECTURE 4.3. *There is a constant $\epsilon > 0$ such that every deterministic one-pass data stream algorithm that computes a $(1+\epsilon)$ approximation to the LIS requires space $\Omega(\sqrt{n})$.*

It is consistent with our current knowledge that an $\Omega(\sqrt{n})$ lower bound holds even for randomized algorithms and when $(1+\epsilon)$ is replaced by some large

constant. We present a communication complexity approach to proving this conjecture. Let $\mathrm{CC}_t^{\mathrm{tot}}(f)$ and $\mathrm{CC}_t^{\max}(f)$ denote the total and maximum communication complexity respectively of t-player protocols for computing a function f. We define a *primitive* function h where each player has a single number, and the players have to decide whether the input string is a decreasing sequence or whether it contains a large increasing sequence. Formally, given a string $x = x_1, \ldots, x_t$ over the alphabet $[m]$, the promise function $h(x)$ is defined as $h(x) = 0$ if $\mathrm{lis}(x) = 1$ and $h(x) = 1$ if $\mathrm{lis}(x) > \epsilon t$. Player P_i is given the number x_i. If we ignore the alphabet-size, it is easy to show that $\mathrm{CC}_t^{\mathrm{tot}}(h) \leq (1-\epsilon)t$. We conjecture that in fact this is tight and $\mathrm{CC}_t^{\mathrm{tot}}(h) = \Omega(t)$.

Next we define a function g on t^2 numbers which is the OR of t disjoint copies of h. Consider the input numbers arranged in a $t \times t$ array $X = (x_{i,j})$ where $x_{i,j} \in [m]$. Define the function g as the OR of h applied to every column of the matrix. Formally, $g(X) = \vee_{j=1}^{t} h(x_{1,j}, \ldots, x_{t,j})$. Player P_i receives the i-th row of the matrix X as input. Computing $g(X)$ is equivalent to deciding whether the columns of X are all decreasing or whether some column contains a large increasing sequence. The naive protocol for g would be to run the protocol for h independently for every column, giving $\mathrm{CC}_t^{\mathrm{tot}}(g) = O(t^2)$ and $\mathrm{CC}_t^{\max}(g) = O(t)$. We conjecture that this is tight and that $\mathrm{CC}_t^{\max}(g) = \Omega(t)$. This has the flavor of a direct-sum result, but we unaware of any prior work that applies to this setting. An $\Omega(t)$ lower bound for $\mathrm{CC}_t^{\max}(g)$ would imply Conjecture 4.3.

LEMMA 4.4. *$\mathrm{CC}_t^{\max}(g)$ is a lower bound on the space used by any one-pass deterministic streaming algorithm that computes a $(1+\epsilon)$ approximation to the LIS.*

Proof. Given $X = (x_{i,j})$, let $\sigma_{i,j} = m(j-1)+x_{i,j}$ giving a $t \times t$ matrix $\sigma = (\sigma_{i,j})$. Consider the string of length t^2 obtained by ordering the elements of this array row-wise. By abuse of notation, we will also call this string σ. We claim that if $g(X) = 0$ then $\mathrm{lis}(\sigma) = t$, and if $g(X) = 1$ then $\mathrm{lis}(\sigma) \geq (1+\epsilon)t$.

It is clear that $\mathrm{lis}(\sigma)$ is at least t, since every row is an increasing sequence. If $g(X) = 0$, then every column of σ is decreasing. Hence $\mathrm{lis}(\sigma)$ contains at most 1 element per column implying that $\mathrm{lis}(\sigma) = t$. On the other hand, assume that $g(X) = 1$ and the j-th column of X has an increasing sequence of length ϵt. Then the j-th column of σ has a corresponding increasing sequence. Taking $\sigma_{1,1}, \ldots, \sigma_{1,j-1}$ followed by the increasing sequence from column j, followed by $\sigma_{t,j+1}, \ldots, \sigma_{t,t}$, we get an increasing sequence of length $(1+\epsilon)t$.

Standard arguments show that a space s deterministic one pass algorithm giving a $(1+\epsilon)$ approximation to the LIS will imply a one-way protocol for g with $\mathrm{CC}_t^{\max}(g) \leq s$. Player P_1 computes the numbers $\sigma_{1,1}, \ldots, \sigma_{1,t}$ and runs the streaming algorithm on it. He passes the contents of the memory to P_2 who runs it on $\sigma_{2,1}, \ldots, \sigma_{2,t}$ and so on, which gives the desired protocol. □

We use this approach to prove Conjecture 4.3 for a restricted yet natural class of algorithms. To motivate this class of algorithms, consider the problem of approximating the length of the LIS when the inputs are real numbers. All our algorithms work in this setting. For instance, ApproximateLIS will store the array $P'(i)$ and the set S of indices. The space used by the algorithm consist of two parts: a subset of the input sequence, namely the array P', and some auxiliary information, namely the set S. Measuring storage space in terms of bit lengths is no longer meaningful, since the inputs are real numbers. This motivates the following definition:

DEFINITION 4.5. (NATURAL ALGORITHM) *A* Natural Algorithm *for* $\mathrm{lis}(\sigma)$ *is a one-pass data stream algorithm that stores some subset S of symbols from the input string σ, as well as some auxiliary bits of information A. The space used by the algorithm is $|S| + |A|$.*

Thus we charge a cost of only 1 unit for symbols of the input string σ. However any additional storage (auxiliary variables) is charged bitwise. We do not allow the algorithm to store alphabet symbols that are not part of the input σ, since this might allow the algorithm to encode all its storage into a single number. All the algorithms mentioned in this paper fit naturally in this class, with essentially the same space bounds. In the full version, we show

THEOREM 4.6. *Any deterministic Natural Algorithm that computes a $(1+\epsilon)$-approximation to* $\mathrm{lis}(\sigma)$ *requires space $\Omega(\sqrt{n})$ for some constant $\epsilon > 0$.*

References

[ACCL04] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Estimating the distance to a monotone function. In *Proceedings of the 8th International Workshop on Randomization and Computation*, pages 229–236, 2004.

[AD99] D. Aldous and P. Diaconis. Longest increasing subsequences: From patience sorting to the Baik–Deift–Johansson theorem. *Bulletin of the American Mathematical Society*, 36:413–432, 1999.

[ADG+03] A. Andoni, M. Deza, A. Gupta, P. Indyk, and S. Raskhodnikova. Lower bounds for embedding edit distance into normed spaces. In *Proceedings of the 14th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 523–526, 2003.

[AJKS02] M. Ajtai, T.S. Jayram, R. Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 370–379, 2002.

[BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM Symposium on Principles of Databases Systems*, pages 1–16, 2002.

[BEK+03] T. Batu, F. Ergun, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 316–324, 2003.

[BYJKK04] Z. Bar-Yossef, T. S. Jayram, R. Krauthgamer, and R. Kumar. Approximating edit distance efficiently. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 550–559, 2004.

[BYJKS04] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004.

[CK06] M. Charikar and R. Krauthgamer. Embedding the Ulam metric in ℓ_1. *Theory of Computing*, 2006. To appear.

[CKS03] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multiparty communication complexity of set-disjointness. In *Proceedings of the 18th Annual IEEE Conference on Computational Complexity*, pages 107–117, 2003.

[CM02] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proceedings of the 13th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 667–676, 2002.

[CMS01] G. Cormode, S. Muthukrishnan, and S. C. Sahinalp. Permutation editing and matching via embeddings. In *Proceedings of 28th International Colloquium on Automata, Languages and Programming*, pages 481–492, 2001.

[EKK+00] F. Ergun, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *Journal of Computing and System Sciences*, 60(3):717–751, 2000.

[FLN+02] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 474–483, 2002.

[GGL+00] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.

[Gus97] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[GZ03] A. Gupta and F. Zane. Counting inversions in lists. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 253–254, 2003.

[KN05] S. Khot and A. Naor. Nonembeddability theorems via Fourier analysis. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 101–112, 2005.

[KR06] R. Krauthgamer and Y. Rabani. Improved lower bounds for embeddings into ℓ_1. In *Proceedings of the 17th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 1010–1017, 2006.

[LNVZ06] D. Liben-Nowell, E. Vee, and A. Zhu. Finding longest increasing and common subsequences in streaming data. *Journal of Combinatorial Optimization*, 11(2):155–175, 2006.

[MP80] W. J. Masek and M. S. Paterson. A faster algorithm for computing string edit distance. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[Mut05] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers Inc., 2005.

[OR05] R. Ostrovsky and Y. Rabani. Low distortion embeddings for edit distance. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, pages 218–224, 2005.

[Pev03] P. Pevzner. *Computational Molecular Biology*. Elsevier Science Ltd., 2003.

[SW07] X. Sun and D. P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proceedings of the 18th ACM-SIAM Annual Symposium on Discrete Algorithms*, 2007.

[Vit85] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[ZLX+06] Y. Zhang, X. Lin, J. Xu, F. Korn, and W. Wang. Space-efficient relative error order sketch over data streams. In *Proceedings of the 22nd International Conference on Data Engineering*, page 51, 2006.