Private Computation of the Longest Increasing Subsequence in Data Streams

Luca Bonomi
Dept. of Mathematics and Computer Science
Emory University
Atlanta, GA
lbonomi@emory.edu

Li Xiong
Dept. of Mathematics and Computer Science
Emory University
Atlanta, GA
lxiong@emory.edu

ABSTRACT

In this paper, we study the problem of privately computing ordered statistics with the goal of monitoring sequential data streams. Despite the broad series of techniques for time-series monitoring, only few works provide provable privacy guarantees employing the formal notion of differential privacy. While these solutions are well established, their focus is mostly limited to count based statistics (e.g. number of distinct elements, heavy hitters). In this paper, we consider a more general problem of privately computing the length of the longest increasing subsequence (LIS) in the data stream model. This important statistic can be used to detect trends in time-series data (e.g. finance) and perform approximate string matching in computational biology domains. Our proposed approaches employ the differential privacy notion which provides strong and provable privacy guarantees. Our solutions estimate the length of the LIS using block decomposition and local approximation techniques. We provide a rigorous analysis to bound the approximation error of our algorithms in terms of privacy level and length of the stream.

1. INTRODUCTION

Sequential data are central in a broad range of domains and applications, such as biomedical, financial and health-care setting where data are continuously collected for monitoring purpose or for mining behavioral patterns. For example, individual household power consumption data may be collected by smart meters to provide billing information or for monitoring purpose. Despite the importance of these tasks, the release of the real data value may disclose sensitive user information. Therefore privacy preserving solutions are employed to compute the required statistics while providing privacy for users. Among them, the popular notion of *differential privacy* [6] is used to construct privacy preserving algorithms. The privacy is achieved by bounding the adversary inference ability in determining the presence of any event in the data stream [7, 10, 9, 4]. Despite the strength of such a privacy model, the current solutions are limited to count based statistics.

In this paper, we study the problem of privately computing ordered statistics with the goal of enabling applications to monitor sequential data streams. Consider for example, a user who may wish to be advised in his/her financial decisions without incurring the risk of disclosing his/her financial information. In such a setting, it is crucial to design effective solutions that enable third-party to detect user's financial trends while preserving the sensitive information. To address this problem, we propose to study the privacy preserving computation of *longest increasing subsequence* (LIS) in the stream model.

The computation of the LIS provides useful information about the sortedness of the data stream and it can be used to detect trends in time-series data. In general, the task of computing the sortedness of a data stream is receiving considerable attention from the computer science community [16, 12, 1, 13, 5, 19, 11]. The sortedness of data stream has important implications from both practical and theoretical perspectives. Many applications rely on ranked data and the massive amount of information dynamically generated cannot be processed in an off-line fashion requiring solutions to have small update time and memory requirements.

The computation of the LIS in the data stream model rises new challenges compared to the traditional privacy setting. First of all, privacy requirements in protecting sensitive information for this ordered statistic have a greater impact on the final utility. Count based statistic over a stream are typically computed by decomposition which leads to a considerable reduction of perturbation noise required by the privacy mechanism. However, ordered statistics are generally not easy to approximate via decomposition since they require a global view of the entire stream. Second, the LIS has higher memory requirements compared to standard counting based statistics (e.g. counts, heavy hitters). In fact, it has been shown in [12] that there exists a space lower bound of $\Omega(T)$ for any randomized algorithm that computes the LIS exactly over a stream of length T. This strong separation between count based functions and LIS impacts both efficiency and utility of the solutions for this problem.

To address these challenges, we propose a series of solutions for privately computing the LIS while minimizing the error introduced by the perturbation and approximation. The detailed contributions are reported below.

Our Contributions. In this paper, we study the problem of privately computing the LIS in a time-bounded stream of length T. 1) Our proposed solutions compute the length of the LIS providing strong and provable privacy guarantees based on the notion of differential privacy. 2) We propose a decomposition framework for approximating the length of the LIS using local information in the stream. This technique allows us to reduce the error due to perturbation noise from the privacy mechanism. Using the Patience Sorting algorithm [15] as a black box for locally computing the exact length of the LIS, we provide an error bound for our framework. 3) We conduct an output-sensitive utility analysis for two cases based on the length of the output LIS. In particular, we as-

Algorithm 1 Patience Sorting

procedure PATIENCE SORTING(σ)
Input: event stream σ
Output: $LIS(\sigma)$ length of the longest increasing subsequence

$P(j) \leftarrow \emptyset$ for $j = 0, 1, \ldots, m-1$
for (any new element $\sigma(i)$) **do**
  Find the largest $P(j)$ such that $P(j) \leq \sigma(i)$
  $P(j+1) = \sigma(i)$
  Output the largest j such that $P(j) \neq \emptyset$
end for
end procedure

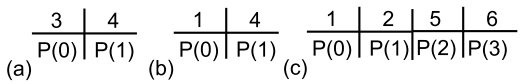

Figure 1: Running example of the Patience Sorting algorithm over the stream $\sigma = 3, 4, 1, 2, 5, 7, 6$.

sume $LIS(\sigma) = \sqrt{T}/\beta$, where T is the length of the input stream σ, and β is a parameter in the range $[1/\sqrt{T}, \sqrt{T}]$. For each solution, we bound the approximation error in the case of long and short LIS respectively depending on the value of β. 4) We propose a new streaming approach which computes the LIS using a hierarchy structure of the stream. Our algorithm achieves a $(1 - \frac{T-b}{T+b})$-approximation to the length of the LIS in the worst case, where the parameter b controls both the perturbation noise to achieve the desired level of privacy and the accuracy. 5) We provide a discussion about possible extensions of our solutions to address time-series stream monitoring and string matching problems. To the best of our knowledge, we are the first to investigate the problem of privately computing the longest increasing subsequence.

The rest of the paper is organized as follows. Section 2 provides the problem definition and presents the privacy model. Section 3 illustrates our decomposition schema and Section 4 describes our hierarchy solution. In Section 5, we provide a summary of our results and also describe some possible extensions. Finally, Section 6 concludes the paper.

2. PRELIMINARIES

Given a sequence σ of elements $\sigma(i) = a_i$ defined over a finite alphabet $\Sigma = \{0, 1, \ldots, N-1\}$, an increasing sequence of length k in σ is a subsequence $\{i_0, i_1, \ldots, i_{k-1}\}$ such that $i_0 < i_1 < \cdots < i_{k-1}$ and $a_{i_0} \leq a_{i_1} \leq \cdots \leq a_{i_{k-1}}$. Furthermore, let $\sigma[i, j]$ denote the contiguous sequence $\sigma(i)\sigma(i+1)\ldots\sigma(j)$ in the stream σ, and let $LIS(\sigma)$ be the length of the longest increasing subsequence in σ.

The problem of computing the LIS has received much attention in the streaming setting (see [2] for a survey of results), where the sequence σ is given an element at a time. In such model, data arrive continuously and at every time i algorithmic solutions are required to report $LIS(\sigma[0, i])$ by using a small amount of memory and performing only few passes over the stream. In the rest of the section, we briefly summarize the non-private techniques present in literature by categorizing them as *exact* and *approximate* solutions.

Exact Solution. The study of LIS in the streaming setting was initiated by Liben-Nowell et al. in [16], where the authors developed an exact one pass algorithm that requires $O(k)$ space for deciding if the length of longest increasing subsequence is at least k. In addition to this technique, the classical algorithm for computing the LIS is based on the Patience Sorting procedure [15]. This approach can be interpreted as a one pass streaming algorithm for computing the exact LIS in $O(T)$ space and it requires $O(\log LIS(\sigma))$ update time. Since we use this approach to build our solutions, we briefly describe this algorithm here.

In the Patience Sorting procedure, the length of the longest increasing subsequence is computed using a set of sorted *piles* $P(0) < P(1) < \cdots < P(m)$ each storing an element of the stream σ. For any new element $\sigma(i)$ that appears in the stream, the algorithm places $\sigma(i)$ in the leftmost pile $P(j)$ such that $P(j) > \sigma(i)$. The number of non empty piles represents the length of the LIS at any time point. An overview of the Patience Sorting algorithm is illustrated in Algorithm 1. Below, we describe a running example of this algorithm.

EXAMPLE 1. *Let $\sigma = 3, 4, 1, 2, 5, 7, 6$ be a stream in input. The algorithm starts with a set of empty piles $P(j)$ for $j = 0, \ldots, m-1$. When the first element arrives in the stream it is placed in the first pile $P(0)$. After the arrival of the second element, the situation in the piles is illustrated in Figure 1 (a). The number of piles denotes the length of the longest increasing subsequence at each time. Therefore, in this case the length of the LIS is two. When the third element $\sigma(2) = 1$ arrives in the stream, the algorithm places this element in $P(0)$, as shown in Figure 1 (b). Following the steps of the algorithm, the final set of piles is reported in Figure 1 (c). At the end of the stream the length of the longest increasing subsequence is four.*

Despite the simplicity of this procedure, the Patience Sorting algorithm is optimal from the space complexity perspective. In fact, Gopalan et al. [12] showed a space lower bound of $\Omega(n)$ for any randomized algorithm that computes the LIS exactly.

Approximate Solution. In [12] the authors proposed a $(1+\epsilon)$-approximation for the LIS computation using $O(\sqrt{T/\epsilon})$ space. A series of works have been developed to estimate the length of the LIS using the number of inverted elements in the stream. In this direction, Ajtai et al. [1] proposed a $(1+\epsilon)$-approximation which requires $O(\frac{1}{\epsilon}\log\log T)$ space to estimate the number of inverted pairs. Later this result has been improved by Gupta and Zane [13]. Cormode et al. [5] proposed a series of algorithmic solutions based on distance preserving embeddings. Recently in [19], the authors investigated the problem of computing the LIS in asymmetric edit distance setting.

2.1 Differential Privacy

Differential privacy [6] is a recent notion of privacy that aims to protect the disclosure of information when data statistics are released. In the streaming setting, due to the dynamics of the data, the classical differential privacy notion has been redefined such that the privacy is guaranteed at *event-level* [7, 10, 9, 4]. In other words, the privacy goal is to protect the presence or absence of any single event in the stream. The formal definition of the differential privacy notion adopted in our work is reported below.

DEFINITION 1 (DIFFERENTIAL PRIVACY [4, 9]). *Two streams σ and σ' of the same length are neighboring streams if they differ exactly in one element at time t. A privacy mechanism M gives α-differential privacy if for any two neighboring streams σ, σ', and for any set of outcomes $S \subseteq Range(M)$,*

$$Pr[M(\sigma) \in S] \leq e^{\alpha} \times Pr[M(\sigma') \in S] \quad (1)$$

The parameter α is called the *privacy parameter* and it defines the privacy level of the mechanism. Higher values of α lead to lower level of privacy, while smaller values pose a stronger privacy guarantee. Intuitively, a mechanism is differentially private if an adversary is unable to determine whether an event of interest took place or not by observing the output of the mechanism over the stream.

Our goal consists in designing a mechanism that, at any time t in the stream, reports the length of the longest increasing subsequence while achieving differential privacy. In addition, we would like the mechanism to be useful, that is, its output well approximates the real length of the LIS. To evaluate our solutions, we introduce the following utility notion.

DEFINITION 2 ((ϵ, δ)-USEFUL). *A streaming algorithm $\mathcal{A}$ is (ϵ, δ)-useful, if for any input stream σ and query q, with probability at least $1-\delta$, the relative distance between the approximate answer from $\mathcal{A}$ and the real answer of q is within ϵ, formally* $P\left[\frac{\|\mathcal{A}(\sigma)-q(\sigma)\|}{q(\sigma)} < \epsilon\right] \geq 1-\delta$

To achieve differential privacy, one well established technique is the *Laplace Mechanism* [8]. Dwork et al. [8] showed that to obtain a α-differentially private solution it is sufficient to perturb the real output of the function by adding a random variable (noise) from a Laplace distribution with probability density function $pdf(x|\lambda) = \frac{1}{2\lambda}e^{-|x|/\lambda}$, where the parameter λ is determined by α and the *sensitivity* of the function to compute. The sensitivity measures the contribution of any single element on the final output. We will use the Laplace mechanism and some other statistical tools to design our privacy preserving solutions.

Statistical tools. In our approaches, we make use of the *Laplace Mechanism* to achieve differential privacy and *sequential composition* property of differential privacy. Furthermore, in our construction the noise may not come from a single Laplace distribution, but rather is composed by a sum of independent Laplace distributions. Therefore, here we state two useful results for sum of independent Laplace distributions.

THEOREM 1 (LAPLACE MECHANISM [8]). *For a function $f : D^T \rightarrow \mathbb{R}^d$, let $GS(f)$ be the sensitivity of f defined as*

$$GS(f) = \max_{D,D'} \|f(D) - f(D')\|_1 \tag{2}$$

where D' and D are neighbouring, then the algorithm that outputs: $\tilde{f}(D) = f(D) + Lap(GS(f)/\alpha)^d$ satisfies α-differential privacy.

THEOREM 2 (SEQUENTIAL COMPOSITION [17]). *Let M_i be a non-interactive privacy mechanism which provides α_i-differential privacy. Then, a sequence of $M_i(D)$ over the database D provides $(\sum_i \alpha_i)$-differential privacy.*

LEMMA 1 (SUM OF LAPLACE DISTRIBUTIONS [4]). *Let $Y = \sum_{i=1}^{n} l_i$ be the sum of $l_1, \ldots, l_n$ independent Laplace random variables with zero mean and parameter b_i for $i = 1, \ldots, n$, and $b_{max} = \max\{b_i\}$. Let $\nu \geq \sqrt{\sum_{i=1}^{n} b_i^2}$, and $0 < \lambda < \frac{2\nu^2}{b_{max}}$. Then $Pr[Y > \lambda] \leq exp\{-\frac{\lambda^2}{8\nu^2}\}$*

COROLLARY 1 (MEASURE CONCENTRATION [4]). *Let Y, $\{b_i\}_i$, λ and b_{max} defined as in Lemma 1. Suppose $0 < \delta < 1$ and $\nu > \max\{\sqrt{\sum_i b_i^2}, b_{max}\sqrt{2\ln\frac{2}{\delta}}\}$. Then $Pr[|Y| > \nu\sqrt{8\ln\frac{2}{\delta}}] \leq \delta$*

2.2 Differentially Private Computation of the LIS - A Baseline Approach

In the rest of the paper, we present our solutions for computing the length of the LIS in the stream. Our approaches require the stream to be *time-bounded*, we assume in fact that the length of the stream is T and it is given a priori.

Here we consider a baseline approach that solves the problem of privately computing the length LIS by perturbing directly its real value at every time point. In particular, for every new element $\sigma(i) = a_i$ in the stream, the algorithm first computes the real $LIS(\sigma[0,i])$ (e.g. using any non-private solution, Patience Sorting in this case) and then it adds a perturbation noise η_i. Given the privacy parameter α, due to the composition property of differential privacy, to obtain an overall mechanism of α-differential privacy, the baseline approach applies the Laplace mechanism at each time point with parameter $\alpha' = \alpha/T$. For each new incoming element, it samples a Laplace variable $\eta_i \sim Lap(1/\alpha')$ which will be used to perturb the real value of $LIS(\sigma[0,i])$. Therefore, at every time i, the algorithm will answer the LIS query by returning $\tilde{l}(\sigma[0,i]) = LIS(\sigma[0,i]) + \eta_i$. We can observe that the sensitivity for the LIS function is 1, since replacing an element from the stream may change the length of the longest increasing subsequence by at most 1. Therefore, perturbing the real value of $LIS(\sigma[0,i])$ with η_i is sufficient to achieve privacy. The utility of this approach is reported in the following theorem.

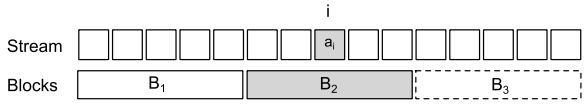

Figure 2: Block Decomposition example at time i: expired blocks (solid lines), active blocks (gray) and the future blocks (dashed lines).

THEOREM 3 (BASELINE UTILITY). *The baseline algorithm is $(\frac{\beta\sqrt{T}}{\alpha}\ln\frac{1}{\delta}, \delta)$-useful for computing the longest increasing subsequence.*

PROOF. *The released length of the LIS at each time i is obtained by perturbing the real length of the LIS with Laplace noise. Therefore, at every time step in the stream we have that the additive error from the noise can be bounded as follows:*

$$Pr[|\eta_i| > \gamma] \leq 2\int_{\gamma}^{\infty} \frac{\alpha}{2T} e^{-x\alpha/T} dx = e^{-\gamma\alpha/T} \tag{3}$$

Hence, with probability at most δ the additive error is at least $\frac{T}{\alpha}\ln\frac{1}{\delta}$. The final result follows by normalizing the error by the $LIS(\sigma) = \sqrt{T}/\beta$. □

Space and Time Analysis. The memory and time complexity for this approach are the same as the non-private algorithm used to compute the real length of the LIS. Therefore, using the Patience Sorting algorithm for example, the space and update time required are $O(LIS(\sigma))$ and $O(\log LIS(\sigma))$ respectively.

3. DECOMPOSITION FRAMEWORK

The baseline approach introduces an additive error that grows linearly with the length of the stream. Therefore, for small LIS this error could dramatically degenerate the utility of this solution. The reason for this large perturbation noise is due to the fact that each individual element in the stream could affect all the possible outputs of the algorithm over the entire stream. This phenomenon could also occur for more sophisticated streaming algorithms that compute the LIS by using a small sketch of stream ([12, 19] for example). Although such solutions could reduce the space requirements, the use of a sketch does not directly reduce the error due to the perturbation noise since an element of the stream could still affect a large number of outputs (e.g. linear with the length of stream).

To overcome this problem, we decompose the computation of the LIS over segments of the stream. This intuition follows the

idea proposed by Chan et al. [4] where a linear and binary decomposition frameworks are employed to privately compute the number of non-zero elements in a binary stream. Despite the similarity in these decompositions, the computation of the longest increasing subsequence is harder to achieve than the simple count function. For this reason, we study the utility loss in approximating the LIS inflicted by using the local information of the stream. Due to space limitation, we consider only an extension of the binary decomposition since it provides better utility with respect to the linear decomposition proposed in the original paper [4].

In our work, we investigate the implications of decomposing the LIS computation over blocks (i.e. stream segments) both from the utility and complexity perspective. It is important to note that the nature of the decomposition should be data-independent to avoid additional privacy cost. In principle, any algorithm $\mathcal{A}_{LIS}$ that computes the LIS (either exact or approximate way) can be used as a building block to compute the LIS on each stream segment so that the perturbation noise required by the privacy mechanism can be reduced with respect to the direct use of $\mathcal{A}_{LIS}$. On the other hand, by limiting our computation on segments we introduce an approximation error.

In the rest of the section, we use the Patience Sorting algorithm [15] as a simple building block. We prove the reduction in the perturbation noise and the approximation error of our solution. We focus on this particular algorithm because it allows us to have an internal procedure that computes the exact length of the LIS over segments of the stream. In this way, we can directly measure how our decomposition impacts the exact solution. Since the original Patience Sorting algorithm computes not only the length of the LIS but also the elements forming the sequence, we use a modified version that only keeps the top element of the piles in the data structure as illustrated in the Algorithm1. In this way, we can compute the length of the LIS but using only $O(LIS(\sigma))$ space.

Before presenting our technique, we illustrate some concepts that will be useful in explaining our algorithm. A block $B = \sigma[j, j + b - 1]$ of size b represents a continuous segment of b symbols in the stream σ. Due to the dynamics of the data in the stream, a block assumes three different states over stream depending on the current time. At time i, the block B can be in one of the following states: **expired** hence the new arrival does not affect the block B (i.e. $j + b - 1 < i$), **active** when the new arrival is contained in the block B (i.e. $j \leq i \leq j + b - 1$) and **future** hence B contains only upcoming elements (i.e. $j > i$). An example of block decomposition of the stream is illustrated in Figure 2. The life cycle of a block B consists of starting as a future block, becoming active, and finally the block expiration.

3.1 Binary Decomposition

We start observing that in general the decomposition of the LIS over blocks may incur large approximation error. In fact, by simply dividing the stream into blocks and combining the length of their LIS as a answer could lead to an approximation error that is proportional to the number of blocks used in the decomposition. To reduce this error, we develop a decomposition using variable length blocks, where the number of blocks in the stream decomposition is $O(\log T)$. We organize the blocks in a binary tree where at time i the tree has $\log i$ levels. Each level $l = 0, \ldots, \log i$ in the tree partitions the stream into disjoint blocks of length $i/2^l$. Figure 3 illustrates an example of binary decomposition of the stream.

Using this representation, each node k in the tree is associated with a block B_k and it stores the perturbed value of the $LIS(B_k)$. At any time i the algorithm updates the noisy LIS of the active blocks in the binary tree, and it answers the query $LIS(\sigma[0, i])$ as

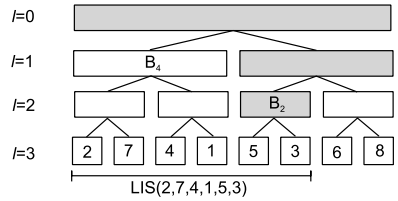

Figure 3: Binary Decomposition example. At time 5 (six symbols), the algorithm updates the active blocks (in gray). It computes the answer to the LIS query by summing the contributions of B_2 and B_4 containing the 2 and 4 most recent symbols respectively.

Algorithm 2 Binary Decomposition

1: **procedure** BINARY DECOMPOSITION(T, α, σ)
Input: upper bound on the stream length T; privacy parameter α; event stream σ
Output: $\tilde{l}(\sigma)$ released longest increasing subsequence
2: **for** $(i = 0, 1, \ldots, T-1)$ **do**
3: **for** (every active B at time i) **do**
4: UPDATE PILES($B, \sigma(i)$)
5: **end for**
6: **for** (every block B that will expire at time $i+1$) **do**
7: $LIS(B) \leftarrow$ number of piles for the block B
8: $\tilde{l}(B) \leftarrow LIS(B) + Lap(2 \log T/\alpha)$
9: **end for**
10: Let $i_1 < i_2 < \cdots < i_m$ be the positions of non-zeros in the binary representation of $i+1$
11: $\tilde{l}(\sigma) \leftarrow 0$
12: $k \leftarrow i$
13: **for** $(j = i_1, i_2, \ldots, i_m)$ **do**
14: $B \leftarrow \sigma(k - 2^j + 1) \cdots \sigma(k)$ ▷ Retrieve the block to reconstruct the LIS
15: $k \leftarrow k - 2^j$
16: $\tilde{l}(\sigma) \leftarrow \tilde{l}(\sigma) + \tilde{l}(B)$ ▷ Sum the noisy contributions of the expired block B
17: **end for**
18: **Output** $\tilde{l}(\sigma)$
19: **end for**
20: **end procedure**

illustrated in Algorithm 2.

Algorithm Description. In the loop at lines 3-5, the algorithm updates the piles for the active blocks associated with the time i. In particular, the procedure `Update Piles` implements the Patience Sorting algorithm as in Algorithm 1, where in this case the update is performed independently on each active block B for any new coming element $\sigma(i)$. At lines 6-9, the noisy length of the LIS for each block that will expire is computed. At line 10, we compute the binary representation of $i+1$ and let $i_1 < i_2 < \cdots < i_m$ be the positions of non-zeros in such representation. Then the answer for $LIS(\sigma[0, i])$ is computed by summing up the length of the LIS for the blocks containing the most recent $2^{i_1}, 2^{i_2}, \ldots, 2^{i_m}$ elements respectively. Therefore at each time i, the output result is obtained by adding the contributions of at most $\Theta(\log i)$ blocks in the loop at lines 13-17.

Privacy Analysis. We can observe that each element affects at most $\log T$ blocks; therefore, perturbing the LIS of each block with a random variable from $Lap(\log T/\alpha)$ is sufficient to satisfy the privacy requirement.

THEOREM 4 (BINARY DECOMPOSITION PRIVACY). *The Binary Decomposition achieves α-differential privacy.*

PROOF. *In this decomposition, each element $\sigma(i)$ participates in the LIS of at most $\log T$ active blocks. Therefore, for any two neighboring streams the difference in L_1-norm of their outputs can*

be bounded by $\log T$. *Therefore using Theorem 1, it is sufficient to add to each LIS of each block a random variable from a Laplace distribution with parameter* $\log T/\alpha$ *to satisfy the privacy requirement.* □

Utility Analysis. This decomposition with variable length blocks allows us to reduce the perturbation error due to the privacy mechanism. However, in this way we introduce an approximation error that depends on the number of blocks. We can observe that at most $O(\log T)$ blocks of variable length are needed to answer a LIS query. The utility results for this decomposition are reported below.

LEMMA 2 (BINARY BLOCK ERROR BOUND). *Let* σ *be a stream of* T *symbols, and let* $LIS(\sigma) = \frac{\sqrt{T}}{\beta}$, *where* β *is positive. Without loss of generality we assume* $T = 2^t - 1$, *and we consider a partition of the stream* σ *into* $B_0, B_1, \ldots, B_{t-1}$ *non-overlapping blocks, where each block* B_k *is of size* 2^k. *Then in reporting the sum of the longest increasing subsequence in each block,* $lis(\sigma) = \sum_{k=0}^{t-1} LIS(B_k)$, *we incur the following approximation error.*

$$LIS(\sigma) \leq lis(\sigma) \leq \begin{cases} \log T \cdot LIS(\sigma) & \beta \geq 1 \\ (1 + \log \beta\sqrt{T}) \cdot LIS(\sigma) & \beta \in [1/\sqrt{T}, 1) \end{cases} \tag{4}$$

PROOF. *First, we start noticing the following lower-bound* $lis(\sigma) \geq LIS(\sigma)$. *In fact, the part of the real longest increasing subsequence which is contained in each block is at most the length of the longest increasing subsequence in the stream segment represented by the block. Second, we prove the two cases separately. For short value of* $LIS(\sigma)$ ($\beta \geq 1$), *we consider the case where each segment in each block is monotonic but none of them can be concatenated to form an increasing sequence in the entire stream. Then, we have that* $LIS(\sigma) \geq LIS(B_k)$, *for* $k = 0, \ldots, t-1$, *which leads to* $\log T \cdot LIS(\sigma) \geq \sum_{k=0}^{t-1} LIS(B_k) = lis(\sigma)$. *For the case of long value of LIS (*$\beta \in [1/\sqrt{T}, 1)$*), we proceed as follows. Let* j *be a positive integer such that* $2^{j-1} < \sqrt{T}/\beta \leq 2^j$. *Therefore, for all the blocks* B_k *with* $k \geq j$ *we have that* $LIS(B_k) \leq \sqrt{T}/\beta$, *otherwise there exists a monotonic sequence which is longer than the longest increasing subsequence, hence we have a contradiction. Furthermore, due to the binary tree decomposition the sum of the length of the LIS for the blocks* B_k *with* $k = 0, \ldots, j-1$ *can be bounded as follows.*

$$\sum_{k=0}^{j-1} LIS(B_k) \leq \sum_{k=0}^{j-1} 2^k = 2^j - 1 \leq 2\sqrt{T}/\beta \tag{5}$$

Therefore, the reported $lis(\sigma)$ *can be upper bounded with the value below.*

$$\begin{aligned} lis(\sigma) = \sum_{k=0}^{t-1} LIS(B_k) &\leq \sum_{k=0}^{j-1} LIS(B_k) + \sum_{k=j}^{t-1} LIS(B_k) \\ &\leq 2\sqrt{T}/\beta + (t-j)\sqrt{T}/\beta \\ &\approx \sqrt{T}/\beta(1 + \log \beta\sqrt{T}) \end{aligned} \tag{6}$$

This concludes the proof of the Lemma. □

THEOREM 5 (BINARY DECOMPOSITION UTILITY). *The binary decomposition algorithm for computing the length of the longest increasing subsequence achieves the following utility results.*

$$\begin{cases} ((\log T - 1) + \frac{\beta \log^{3/2} T}{\alpha\sqrt{T}} \ln \frac{1}{\delta}, \delta)\text{-useful} & \beta \geq 1 \\ (\log \beta\sqrt{T} + \frac{\beta \log^{3/2} T}{\alpha\sqrt{T}} \ln \frac{1}{\delta}, \delta)\text{-useful} & \beta \in [1/\sqrt{T}, 1) \end{cases}$$

PROOF. *This decomposition has the advantage that the number of blocks combined in estimating the length of the LIS is only logarithmic which leads to an approximation error as shown in Lemma 2. This decomposition introduces a perturbation noise which is a sum of at most* $O(\log T)$ *i.i.d. Laplace random variables with parameter* $O(\log T/\alpha)$. *Let* $\xi = \sum_k \eta_k$ *denote the error due to the sum of the Laplace random variables, we can use the result in Corollary 1 to bound this quantity. Choosing* $\nu = \sqrt{\sum_k \frac{\log T}{\alpha}}\sqrt{2 \ln \frac{2}{\delta}}$ *with probability at least* $1-\delta$, *the quantity* ξ *is at most* $O(\frac{\log^{3/2} T}{\alpha} \ln \frac{2}{\delta})$. *Therefore, the final utility follows using the results from Lemma 2 and by normalizing this value by the* $LIS(\sigma)$. □

Space Analysis. The space requirement for this approach is related to the number of active blocks that need to be updated and to the space complexity of the internal procedure. Due to the nature of the binary decomposition at any time i there are $\Theta(\log T)$ blocks that are active. Using a similar argument as in Theorem 5, we can show that the space complexity is $O(LIS(\sigma) \ln(\beta^2 LIS(\sigma)))$.

THEOREM 6 (BINARY DECOMPOSITION SPACE COMP.). *Let* $LIS(\sigma)$ *be the length of the longest increasing subsequence in the stream* σ, *then the Binary decomposition framework has space complexity* $O(LIS(\sigma) \ln(\beta^2 LIS(\sigma)))$.

PROOF. *We begin by recalling that the internal procedure for computing the length of the* LIS *is the Patience Sort algorithm, where we keep only the top of the piles. At any time* i *in the stream,* $\log T$ *blocks are active, one in each level of the tree structure. Furthermore, let* j *be a positive integer such that* $2^{j-1} < LIS(\sigma) \leq 2^j$. *Therefore for the blocks in any level* $i > j$ *in the tree, we can upper bound their space requirements with* $LIS(\sigma)(\log T - j + 1)$, *since* $LIS(\sigma)$ *is the current length of the longest increasing subsequence. On the other hand, due to the nature of the binary tree the space required by the blocks below the level* i *is* $2^j - 1$. *Therefore the space complexity for this approach is* $O(LIS(\sigma)(\log T - j + 1))$. *Using the notion that* $LIS(\sigma) = \sqrt{T}/\beta$ *and* $j = \log(LIS(\sigma))$, *the previous requirements can be rewritten as* $O(LIS(\sigma) \ln(\beta^2 LIS(\sigma)))$. □

Time Analysis. The total update time for this solution is related to the updates of the active blocks. Since at every time i there are $\Theta(\log T)$ active blocks, the update time is $O(\log T \log LIS(\sigma))$ using Patience Sorting algorithm.

4. HIERARCHY MECHANISM

In the previous section, we showed that the binary decomposition considerably reduces the perturbation noise in the final output compared to the baseline approach. However, such technique suffers from the fact that the computation of the LIS is generally hard to be decomposed in blocks leading in some cases to a large approximation error. To overcome this problem, we propose a new algorithm which computes the LIS over the stream by simulating the behavior of the Patience Sorting algorithm. In contrast to our previous approaches, this solution computes the length of the LIS by smoothing the impact of each element with the purpose of reducing the perturbation noise while achieving a good approximation ratio.

The main idea is to reduce the impact of those elements that stay too long in the LIS so that the total noise required by the privacy mechanism is decreased. Given an integer $b > 0$, we construct a series of $m = \Theta(\ln \frac{T}{b})$ layers $l_0, l_1, \ldots, l_{m-1}$ with b buckets each, where at layer i each bucket contains 2^i elements. Given the series of elements with index $\{1, 2, \ldots, T\}$ in the stream, each layer

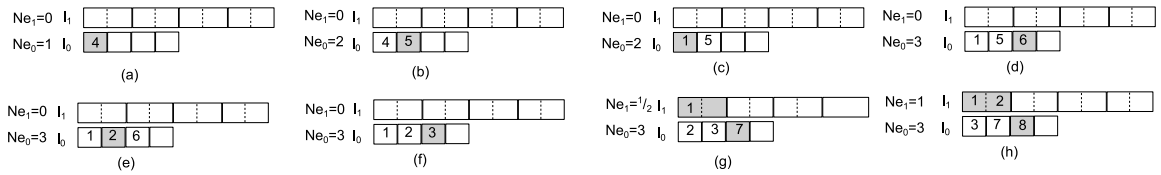

Figure 4: Running example of the Hierarchy mechanism on the input stream $4, 5, 1, 6, 2, 3, 7, 8$, $b = 4$ and $m = 2$.

simulates the behavior of the Patience Sorting algorithm where in this case the original piles are replaced with buckets that can contain multiple elements. In fact, at layer i the elements in the range $[(j-1)2^i + 1, j2^i]$ can be placed into the same bucket j. Intuitively, each layer has a different granularity, in fact l_0 keeps the exact top elements in the most recent b piles in the Patience Sorting algorithm, while l_1 keeps an approximation of the next $2b$ piles and so forth for the other layers. As the original algorithm, our procedure computes the length of the LIS by counting the number on non-empty buckets. In our case multiple elements may fall in the same bucket; therefore, we use a scaling factor equal to the length of the bucket to compute the contribution of each layer. Furthermore, in addition to insertion and replacement moves allowed in the Patience Sorting algorithm, we introduce an expiration move that forces elements that stay in a bucket at layer l_i for more than $2^i b$ iterations to be moved up to layer l_{i+1}. The algorithm computes the length of the LIS in the stream by adding the contribution at each layer. The code for this procedure is reported in Algorithm 3.

Algorithm Description. The algorithm starts initializing a set of m layers containing b buckets each, at lines 2-3. Within a layer i, each bucket is denoted with $P_i(j)$, for $j = 1, \ldots, b$ and it has size $2^i b$. In the main loop, lines 4-21, each new element coming in the stream is inserted in the first layer using the the same rule as the Patience Sorting algorithm, lines 5-7. In the inner loop at lines 9-13, the algorithm checks layer by layer to find the expired elements. When an expired element p in a pile $P_i(j)$ is found, the algorithm removes p and inserts it in the next layer. At line 14, the number of non-empty buckets for each layer is computed by normalizing the number of elements within each bucket with the corresponding bucket's size. In the loop at lines 16-19, the perturbation noise is applied to each count and finally the length of the LIS is returned.

We illustrate our hierarchy mechanism in the example below.

EXAMPLE 2. *Consider the situation in Figure 4. When the first element arrives in the stream it is placed in the first bucket at l_0 as shown in (a). The second element that arrives is 5, since it is larger than 4 it is placed in the next bucket (b). The third element in the stream is 1. Since the insertion of the elements in the buckets follows the same rules as the Patience Sorting algorithm, we find the bucket that contains the smallest element larger than 1 and insert this element in that bucket. Therefore, in our case, 1 overwrites 4 in the first bucket (c). At this point the length of the LIS is 2, as represented by the number of non empty buckets in l_0. The algorithm proceeds in a similar manner of the next three incoming elements (d),(e) and (f). After these new elements, the element 1 in l_0 is moved up to l_1 since it has been present in l_0 for more than b steps and the new incoming element 7 is inserted in l_0 (g). In the next step, the element 2 is moved up, and it is inserted in the same bucket with the element 1. At the same time the new element 8 is inserted in l_0 (e). The reported length of the LIS is obtained by summing the contribution of each layer. Layer l_0 contributes with $Ne_0 = 3$ and l_1 contributes with $Ne_1 = 1$. Hence the algorithm reports a length of the LIS of 4 while the exact length is 5.*

Algorithm 3 Hierarchy Mechanism

1: **procedure** HIERARCHY MECHANISM(T, α, σ, b)
   Input: upper bound on the stream length T; privacy parameter α; event stream σ; accuracy parameter b
   Output: $\tilde{l}(\sigma)$ released longest increasing subsequence
2: $m = \Theta(\ln \frac{T}{b})$
3: Initialize each layer $l_i = [P_i(1), \ldots, P_i(b)]$ $i = 0, \ldots, m-1$ with b empty buckets
4: **for** $(i = 0, 1, \ldots, T-1)$ **do**
5: Insert $\sigma(i)$ in l_1
6: Find the largest $P_1(j)$ such that $P_1(j) \leq \sigma(i)$
7: $P_1(j+1) = \sigma(i)$
8: **for** $(i = 0, \ldots, m-1)$ **do**
9: Let p be the element that expires at l_i
10: Remove p from l_i and insert it in l_{i+1}
11: Find the largest element in $P_{i+1}(j)$ such that $P_{i+1}(j) \leq p$
12: $P_{i+1}(j+1) = p$
13: **end for**
14: Let Ne_i be the number of non-empty buckets at layer l_i
15: $\tilde{l}(\sigma) \leftarrow 0$
16: **for** $(i = 0, 1, \ldots, m-1)$ **do**
17: $\widehat{Ne_i} \leftarrow Ne_i + Lap(mb/\alpha)$
18: $\tilde{l}(\sigma) \leftarrow \tilde{l}(\sigma) + \widehat{Ne_i}$ ▷ Sum the noisy contribution of each layer
19: **end for**
20: **Output** $\tilde{l}(\sigma)$
21: **end for**
22: **end procedure**

Privacy Analysis. In this algorithm the contribution of each element on the LIS is progressively decreased according to the layer in which the element appears. The privacy result for our hierarchy mechanism is reported in the following theorem.

THEOREM 7 (HIERARCHY MECHANISM PRIVACY). *The Hierarchy Mechanism achieves α-differential privacy.*

PROOF. *Given any two neighboring streams, we can observe that each element can affect at most m layers over the entire stream. In particular, at l_0 an element contributes to the LIS with a factor 1 for b times, at l_1 contributes with factor 1/2 for $2b$ and at the general level l_i contributes with factor $1/2^i$ for $2^i b$ times. Let $\mathbf{Ne}$ be the vector of contributions for each layer for the input stream $\sigma = a_1, \ldots, a_i \ldots, a_T$. Then, $\forall i \in [1, T]$ and $\sigma' = a_1, \ldots, a'_i \ldots, a_T$ we have that*

$$\|\mathbf{Ne}(\sigma) - \mathbf{Ne}(\sigma')\| \leq mb \tag{7}$$

Then adding a random Laplace noise with parameter mb/α to the contribution of each layer i, is sufficient to satisfies α-differential privacy. Furthermore, using Corollary 1 we can see that the additive error introduced by noise is only $O(\frac{b}{\alpha} \log^{3/2}(\frac{T}{b}) \log(\frac{2}{\delta}))$. □

Approximation Error. Our algorithm smooths the contribution of each element in the stream according to its layer leading to an underestimated value for the length of the LIS. The following Theorem summarizes the approximation ratio in the worst case.

THEOREM 8 (HIERARCHY APPROXIMATION ERROR). *Let σ be a stream of length T, and b be the number of buckets in each*

Table 1: Summary of results for LIS query over entire stream.

Method	Error	Memory	Update Time
Baseline	$O(\frac{\beta\sqrt{T}}{\alpha}\ln\frac{1}{\delta})$	$O(LIS(\sigma))$	$O(\log\frac{\sqrt{T}}{\beta})$
Binary	$O((\log T-1)+\frac{\beta\log^{3/2}T}{\alpha\sqrt{T}}\ln\frac{1}{\delta})$ where $\beta\geq 1$ $O(\log\beta\sqrt{T}+\frac{\beta\log^{3/2}T}{\alpha\sqrt{T}}\ln\frac{1}{\delta})$ where $\beta\in[1/\sqrt{T},1)$	$O(LIS(\sigma)\ln(\beta^2 LIS(\sigma)))$	$O(\log T\log\frac{\sqrt{T}}{\beta})$
Hierarchy	$O((1-\frac{T-b}{T+b})+\frac{b\beta}{\sqrt{T}\alpha}\log^{3/2}(\frac{T}{b})\log(\frac{2}{\delta}))$	$O(LIS(\sigma))$	$O(\log b\log\frac{T}{b})$

layer of our algorithm. Then, the hierarchy mechanism returns a $(1-\frac{T-b}{T+b})$-approximation of the length of LIS.

PROOF. *Let k be the length of the LIS over the entire stream. We begin by showing that this algorithm never overestimates the length of the LIS and then proceed by showing the error in the underestimate. To understand why this algorithm always reports a length of the LIS less or equal to the real length we consider the following case. Let us assume that there exists an element $\sigma(j)$ in a bucket at level $i>0$ in our algorithm that differs from the Patience Sort. Since this element is extra in our algorithm it means that there is an element $\sigma(j')$, $j'>j$ that replaces $\sigma(j)$ in the exact Patience Sort. Since $\sigma(j')<\sigma(j)$, we have that in our structure $\sigma(j')$ has replaced another element $\sigma(j'')$. Due to the nature of our algorithm this operation could only occur in a layer $i'<i$, hence in replacing $\sigma(j'')$ with $\sigma(j')$ in our algorithm we have a larger loss of contribution than replacing $\sigma(j)$. Therefore we cannot have an overestimate length of the LIS.*

Now, we examine the error in underestimating the length of the LIS. Consider a worst case scenario where only the first k symbols in σ contribute to the LIS, while the rest of the stream does not increase the length of the LIS. In this situation, as the stream proceeds the elements of the LIS that initially are in layer 0 are progressively moved up introducing a small additive error. Below, we quantify this error. Let $m=\log(\frac{T}{b}+1)-1$ be the number of layers in our structure, then the maximum additive errors on the LIS is achieved when all the elements forming the LIS are in layer m. This quantity is computed as follows.

$$\sum_{i=1}^{m}\frac{k}{2^i}=k\left(\frac{T-b}{T+b}\right) \tag{8}$$

Hence the returned value from our algorithm is lower bounded by $LIS(\sigma)(1-\frac{T-b}{T+b})$. This shows that our returned length $\tilde{l}(\sigma)$ satisfies the following inequality.

$$LIS(\sigma)\left(1-\frac{T-b}{T+b}\right)\leq\tilde{l}(\sigma)\leq LIS(\sigma) \tag{9}$$

Therefore, our algorithm provides a $(1-\frac{T-b}{T+b})$-approximation of the length of LIS. □

Space Analysis. Since this algorithm simulates the Patience Sorting algorithm by keeping only the top of the piles forming the LIS, it follows that the space complexity is linear with the length of the LIS in the stream $O(LIS(\sigma))$.

Time Analysis. For any new incoming element in the stream, the total running time is given by the cost required for updating each pile. There are at most $m-1$ buckets, one for each level, that need update, where each operation requires $O(\log b)$ time. Since $m=\Theta(\log\frac{T}{b})$, the update time is $O(\log b\log\frac{T}{b})$.

This solution points out a strong connection between the approximation ratio and the noise required to achieve privacy. We can see that increasing b has a beneficial effect on the approximation ratio but on the other hand increases the privacy cost. In fact, as an extreme case using $b=T$ the algorithm returns the exact length of the LIS but incurs a large perturbation noise. Compared with our decomposition framework, this algorithm provides the user with a way to balance the approximation ratio and the noise due to the privacy mechanism.

5. SUMMARY OF RESULTS

Table 1 summarizes the utility results of our proposed solutions. We can see that both our strategies outperform the baseline approach in many perspectives. We notice that the baseline approach incurs a large perturbation error which could dramatically compromise the utility. Specifically, the additive error in the baseline strategy grows linearly with the length of the stream. For the binary decomposition instead, we provide output-sensitive utility results showing the benefits of this technique for different lengths of LIS. Due to the use of disjoint blocks, this approach incurs a considerably smaller perturbation error with respect to the baseline solution. In fact, the dependency of the error with respect to the perturbation noise is only polylogarithmic in this case. Furthermore, we can observe that the decomposition framework has small space requirements and update time. In principle, the space and time complexity of this solution could be further improved by using more sophisticated algorithms (e.g. [12, 19]) as internal procedure instead of relying on the Patience Sorting. For count based statistic the binary decomposition has been shown very effective; however due to the nature of the LIS, this strategy incurs an approximation error. Our hierarchy approach specifically addresses the LIS problem by directly simulating the Patience Sorting algorithm. This procedure incurs a smaller computational time and it has small memory requirements. Comparing the worst case performance of this technique with the binary decomposition, we can observe that the decomposition framework is still superior leading to a smaller additive error with the same approximation ratio. This result is due to the fact that the hierarchy strategy suffers when the LIS constitutes the initial part of the stream. In fact, as the execution proceeds the elements in the sketch are moved in higher level increasing the approximation error over the stream. However, we can notice that in real scenarios such situation is unlikely to occur because in many applications we can assume that the stream presents trends over time.

5.1 Extensions

In this section, we describe how to employ our developed techniques to solve real world problems.

Detecting trends in time-series data. Our proposed techniques can be extended to effectively detect trends in time-series data by restricting the computation of the LIS over windows in the stream. In fact, in monitoring applications, recent data is more important than distant data; therefore, using a sliding window W, we limit the computation of the LIS on the most W recent data. For example, a sudden increase of price in financial data will lead to an increment

in the length of the LIS in the current window. Constraining the computation of the LIS on a sliding window of length W is beneficial both from the utility and complexity perspective. In fact, it has been shown in [3] that for the binary mechanism the use of a sliding window reduces the impact of the privacy to a factor that is independent from the length of the stream but it is only related to the size of the window W. A similar result can be also derived for the hierarchy mechanism, where in this case, the number of layers in the data structure depends only on the length of W rather than the entire stream.

Approximate String Matching. The problem of computing the LIS is a classical string matching problem that has been extensively studied in computational biology [14]. However, only few solutions have been proposed to privately match biological sequences. Generally, these approaches provide privacy and security in matching strings by applying cryptographic techniques [18]. However, due to their high complexity these approaches may not be effective in real scenarios. In this setting, we believe that our solutions can be very promising by providing formal privacy guarantee and incurring a small computational overhead. Since the problem structure of the LIS is similar to other popular problems for computing string similarity measures (e.g. edit distance), we believe that our hierarchy approach could be a first step toward the design of efficient privacy preserving algorithms for matching strings.

6. CONCLUSIONS

In this paper, we considered the problem of privately detecting trends in stream data. Specifically, we addressed the problem of computing the length of the LIS while protecting the presence of single event in the stream. We developed two different solutions that provide formal guarantee of privacy. The first approach approximates the length of the LIS by assembling local information computed on segments of the stream. The second approach constructs a small sketch of the stream by exploiting the structure of the problem. Using a rigorous analysis, we showed that these strategies provided significant benefits over the baseline approach.

For the future, we consider to investigate two possible research directions. First, we plan to further develop our extensions and turn our theoretical results into concrete algorithms to be applied to solve time-series monitoring and string matching problems. Second, our proposed solutions provide important insights about the privacy implications for computing complex ordered statistics. Therefore, we plan to better understand what kind of privacy sketching algorithms can benefit in this setting.

7. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1117763.

8. REFERENCES

[1] Miklós Ajtai, T. S. Jayram, Ravi Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 370–379. ACM, 2002.

[2] David Aldous and Persi Diaconis. Longest increasing subsequences: From patience sorting to the baik-deift-johansson theorem. *Bull. Amer. Math. Soc*, 36:413–432, 1999.

[3] Jean Bolot, Nadia Fawaz, S. Muthukrishnan, Aleksandar Nikolov, and Nina Taft. Private decayed predicate sums on streams. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 284–295, New York, NY, USA, 2013. ACM.

[4] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3), November 2011.

[5] Graham Cormode, S. Muthukrishnan, and Süleyman Cenk Sahinalp. Permutation editing and matching via embeddings. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming,*, ICALP '01, pages 481–492, 2001.

[6] Cynthia Dwork. Differential privacy. In *ICALP*, 2006.

[7] Cynthia Dwork. Differential privacy in new settings. In *SODA*, pages 174–183, 2010.

[8] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC 2006*.

[9] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 715–724, New York, NY, USA, 2010. ACM.

[10] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N. Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *ICS*, pages 66–80, 2010.

[11] Funda Ergun and Hossein Jowhari. On distance to monotonicity and longest increasing subsequence of a data stream. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 730–736, 2008.

[12] Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 318–327, 2007.

[13] Anupam Gupta and Francis X. Zane. Counting inversions in lists. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '03, pages 253–254, 2003.

[14] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997.

[15] J. M. Hammersley. A few seedlings of research. In *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability*, pages 345–394, Berkeley, Calif., 1972. University of California Press.

[16] David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. In *Proceedings of the 11th annual international conference on Computing and Combinatorics*, COCOON'05, pages 263–272, 2005.

[17] Frank D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD '09*.

[18] S. Rane and Wei Sun. Privacy preserving string comparisons based on levenshtein distance. In *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, pages 1–6, Dec 2010.

[19] Michael Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *SODA*, pages 1698–1709, 2013.