Nearly Optimal Distinct Elements and Heavy Hitters on Sliding Windows

Vladimir Braverman
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA
vova@cs.jhu.edu

Elena Grigorescu
Department of Computer Science, Purdue University, West Lafayette, IN, USA
elena-g@purdue.edu

Harry Lang
Department of Mathematics, Johns Hopkins University, Baltimore, MD, USA
hlang8@jhu.edu

David P. Woodruff
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
dwoodruf@cs.cmu.edu

Samson Zhou
Department of Computer Science, Purdue University, West Lafayette, IN, USA
samsonzhou@gmail.com

Abstract
We study the distinct elements and \(\ell_p\)-heavy hitters problems in the sliding window model, where only the most recent \(n\) elements in the data stream form the underlying set. We first introduce the composable histogram, a simple twist on the exponential (Datar et al., SODA 2002) and smooth histograms (Braverman and Ostrovsky, FOCS 2007) that may be of independent interest. We then show that the composable histogram along with a careful combination of existing techniques to track either the identity or frequency of a few specific items suffices to obtain algorithms for both distinct elements and \(\ell_p\)-heavy hitters that are nearly optimal in both \(n\) and \(\epsilon\).

Applying our new composable histogram framework, we provide an algorithm that outputs a \((1 + \epsilon)\)-approximation to the number of distinct elements in the sliding window model and uses \(O\left(\frac{1}{\epsilon^2} \log n \log \log n + \frac{1}{\epsilon^2} \log^2 n\right)\) bits of space. For \(\ell_p\)-heavy hitters, we provide an algorithm using space \(O\left(\frac{1}{\epsilon^p} \log^p n (\log^2 \log n + \log \frac{1}{\epsilon})\right)\) for \(0 < p \leq 2\), improving upon the best-known algorithm for \(\ell_2\)-heavy hitters (Braverman et al., COCOON 2014), which has space complexity \(O\left(\frac{1}{\epsilon^2} \log^3 n\right)\). We also show complementing nearly optimal lower bounds of \(\Omega\left(\frac{1}{\epsilon^2} \log^2 n + \frac{1}{\epsilon} \log n\right)\) for distinct elements and \(\Omega\left(\frac{1}{\epsilon^p} \log^p n\right)\) for \(\ell_p\)-heavy hitters, both tight up to \(O\left(\log \log n\right)\) and \(O\left(\log \frac{1}{\epsilon}\right)\) factors.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Streaming algorithms, sliding windows, heavy hitters, distinct elements
Introduction

The streaming model has emerged as a popular computational model to describe large data sets that arrive sequentially. In the streaming model, each element of the input arrives one-by-one and algorithms can only access each element once. This implies that any element that is not explicitly stored by the algorithm is lost forever. While the streaming model is broadly useful, it does not fully capture the situation in domains where data is time-sensitive such as network monitoring [29, 30, 33] and event detection in social media [61]. In these domains, elements of the stream appearing more recently are considered more relevant than older elements. The sliding window model was developed to capture this situation [35]. In this model, the goal is to maintain computation on only the most recent $n$ elements of the stream, rather than on the stream in its entirety. We call the most recent $n$ elements active and the remaining elements expired. Any query is performed over the set of active items (referred to as the current window) while ignoring all expired elements.

The problem of identifying the number of distinct elements, is one of the foundational problems in the streaming model.

**Problem 1** (Distinct elements). Given an input $S$ of elements in $[m]$, output the number of items $i$ whose frequency $f_i$ satisfies $f_i > 0$.

The objective of identifying heavy hitters, also known as frequent items, is also one of the most well-studied and fundamental problems.

**Problem 2** ($\ell_p$-heavy hitters). Given parameters $0 < \phi < \epsilon < 1$ and an input $S$ of elements in $[m]$, output all items $i$ whose frequency $f_i$ satisfies $f_i \geq \epsilon (F_p)^{1/p}$ and no item $i$ for which $f_i \leq (\epsilon - \phi) (F_p)^{1/p}$, where $F_p = \sum_{i \in [m]} f_i^p$. (The parameter $\phi$ is typically assumed to be at least $c\epsilon$ for some fixed constant $0 < c < 1$.)

In this paper, we study the distinct elements and heavy hitters problems in the sliding window model. We show almost tight results for both problems, using several clean tweaks to existing algorithms. In particular, we introduce the composable histogram, a modification to the exponential histogram [35] and smooth histogram [19], that may be of independent interest. We detail our results and techniques in the following section, but defer complete proofs to the full version of the paper [16].

1.1 Our Contributions

Distinct elements.

An algorithm storing $O(\frac{1}{\epsilon^2} \log n \log \frac{1}{\epsilon} (\log \frac{1}{\epsilon} + \log \log n))$ bits in the insertion-only model was previously provided [53]. Plugging the algorithm into the smooth histogram framework of [19] yields a space complexity of $O(\frac{1}{\epsilon^3} \log^4 n (\log \frac{1}{\epsilon} + \log \log n))$ bits. We improve this significantly as detailed in the following theorem.

**Theorem 1.** Given $\epsilon > 0$, there exists an algorithm that, with probability at least $\frac{2}{3}$, provides a $(1 + \epsilon)$-approximation to the number of distinct elements in the sliding window model, using $O(\frac{1}{\epsilon^2} \log n \log \frac{1}{\epsilon} \log \log n + \frac{1}{\epsilon} \log^2 n)$ bits of space.
We introduce a simple extension of the exponential and smooth histogram frameworks, which use several instances of an underlying streaming algorithm. In contrast with the existing frameworks where $\mathcal{O}(\log n)$ different sketches are maintained, we observe in Section 2 when the underlying algorithm has certain guarantees, then we can store these sketches more efficiently.

### 1.2 Our Techniques

We first recall in Lemma 16 a condition that allows the reduction from the problem of finding the $\ell_2$-heavy hitters for $0 < p \leq 2$ to the problem of finding the $\ell_2$-heavy hitters. An algorithm of [12] allows us to maintain an estimate of $\mathcal{F}_p^{1/p}$ time complexities by

$\mathcal{O}(\frac{1}{\epsilon} \log^2 n + \frac{1}{\epsilon^2} \log^3 n)$

for insertion-only streams, which is also applicable to sliding windows since the model is strictly more difficult. We give a lower bound for distinct elements in the sliding window model, showing that our algorithm is nearly optimal, up to $\log \frac{1}{\epsilon}$ and $\log \log n$ factors, in both $n$ and $\epsilon$.

**Theorem 3.** Given $\epsilon > 0$ and $0 < p \leq 2$, there exists an algorithm in the sliding window model that, with probability at least $\frac{2}{3}$, outputs all indices $i \in [n]$ for which $f_i \geq \epsilon F_p^{1/p}$, and reports no indices $i \in [n]$ for which $f_i \leq \epsilon F_p^{1/p}$. The algorithm has space complexity (in bits) $\mathcal{O}(\frac{1}{\epsilon^2} \log^2 n (\log^2 \log n + \log \frac{1}{\epsilon}))$.

Finally, we obtain a lower bound for $\ell_p$-heavy hitters in the sliding window model, showing that our algorithm is nearly optimal (up to $\log \frac{1}{\epsilon}$ and $\log \log n$ factors) in both $n$ and $\epsilon$.

**Theorem 4.** Let $p > 0$ and $\epsilon, \delta \in (0, 1)$. Any one-pass streaming algorithm that returns the $\ell_p$-heavy hitters in the sliding window model with probability $1 - \delta$ requires $\Omega((1 - \delta) \epsilon^{-p} \log^2 n)$ bits of space.

More details are provided in Section 4 and Section 5.

By standard amplification techniques any result that succeeds with probability $\frac{2}{3}$ can be made to succeed with probability $1 - \delta$ while multiplying the space and time complexities by $\mathcal{O}(\log \frac{1}{\delta})$. Therefore Theorem 1 and Theorem 15 can be taken with regard to any positive probability of failure.

See Table 1 for a comparison between our results and previous work.

### 1.2 Our Techniques

<table>
<thead>
<tr>
<th>Problem</th>
<th>Previous Bound</th>
<th>New Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_2$-heavy hitters</td>
<td>$\mathcal{O}(\frac{1}{\epsilon} \log^2 n)$ [15]</td>
<td>$\mathcal{O}(\frac{1}{\epsilon^2} \log^2 n (\log^2 \log n + \log^2 \frac{1}{\epsilon}))$</td>
</tr>
<tr>
<td>Distinct elements</td>
<td>$\mathcal{O}(\frac{1}{\epsilon^2} \log^2 n + \frac{1}{\epsilon} \log^3 n)$ [53, 19]</td>
<td>$\mathcal{O}(\frac{1}{\epsilon^2} \log^2 n \log \log n + \frac{1}{\epsilon} \log^2 n)$</td>
</tr>
</tbody>
</table>

A known lower bound is $\Omega(\frac{1}{\epsilon} + \log n)$ bits [1, 50] for insertion-only streams, which is also applicable to sliding windows since the model is strictly more difficult. We give a lower bound for distinct elements in the sliding window model, showing that our algorithm is nearly optimal, up to $\log \frac{1}{\epsilon}$ and $\log \log n$ factors, in both $n$ and $\epsilon$.

**Theorem 2.** Let $0 < \epsilon \leq \frac{1}{\sqrt{n}}$. Any one-pass streaming algorithm that returns a $(1 + \epsilon)$-approximation to the number of distinct elements in the sliding window model with probability $\frac{2}{3}$ requires $\Omega(\frac{1}{\epsilon^2} \log^2 n + \frac{1}{\epsilon^2} \log n)$ bits of space.
Figure 1 Each horizontal bar represents an instance of the insertion-only algorithm. The red instance represents the sliding window. Storing an instance beginning at each possible start point would ensure that the exact window is always available, but this requires linear space. To achieve polylogarithmic space, the histogram stores a strategically chosen set of $O(\log n)$ instances (shaded grey) so that the value of $f$ on any window can be $(1+\epsilon)$-approximated by its value on an adjacent window.

Algorithm 1 Input: a stream of elements $p_1, p_2, \ldots$ from $[m]$, a window length $n \geq 1$, error $\epsilon \in (0, 1)$.

1: $T \leftarrow 0$
2: $i \leftarrow 1$
3: loop
4: Get $p_i$ from stream
5: $T \leftarrow T + 1$; $t_T \leftarrow i$; Compute $D(t_T)$, where $\hat{f}(D)$ is a $(1 \pm \frac{\epsilon}{4})$-approximation of $f$.
6: for all $1 < j < T$ do
7: if $\hat{f}(D(t_{j-1} : t_T)) < (1 - \frac{\epsilon}{4}) \hat{f}(D(t_{j+1} : t_T))$ then
8: Remove $t_j$; update indices; $T \leftarrow T - 1$
9: if $t_2 < i - n$ then
10: Delete $t_1$; update indices; $T \leftarrow T - 1$
11: $i \leftarrow i + 1$

Sketching Algorithms

Consider the sliding window model, where elements eventually expire. A very simple (but wasteful) algorithm is to simply begin a new instance of the insertion-only algorithm upon the arrival of each new element (Figure 1). The smooth histogram of [19], summarized in Algorithm 1, shows that storing only $O(\log n)$ instances suffices.

Algorithm 1 may delete indices for either of two reasons. The first (Lines 9-10) is that the index simply expires from the sliding window. The second (Lines 7-8) is that the indices immediately before ($t_{j-1}$) and after ($t_{j+1}$) are so close that they can be used to approximate $t_j$.

For the distinct elements problem (Section 3), we first claim that a well-known streaming algorithm [6] provides a $(1+\epsilon)$-approximation to the number of distinct elements at all points in the stream. Although this algorithm is suboptimal for insertion-only streams, we show that it is amenable to the conditions of a composable histogram (Theorem 6). Namely, we show there is a sketch of this algorithm that is monotonic over suffixes of the stream, and thus there exists an efficient encoding that efficiently stores $D(t_i : t_{i+1})$ for each $1 \leq i < T$, which allows us to reduce the space overhead for the distinct elements problem.

For $\ell_2$-heavy hitters (Section 4), we show that the $\ell_2$ norm algorithm of [12] also satisfies the sketching requirement. Thus, plugging this into Algorithm 1 yields a method to maintain

Sketching Algorithms

Consider the sliding window model, where elements eventually expire. A very simple (but wasteful) algorithm is to simply begin a new instance of the insertion-only algorithm upon the arrival of each new element (Figure 1). The smooth histogram of [19], summarized in Algorithm 1, shows that storing only $O(\log n)$ instances suffices.

Algorithm 1 may delete indices for either of two reasons. The first (Lines 9-10) is that the index simply expires from the sliding window. The second (Lines 7-8) is that the indices immediately before ($t_{j-1}$) and after ($t_{j+1}$) are so close that they can be used to approximate $t_j$.

For the distinct elements problem (Section 3), we first claim that a well-known streaming algorithm [6] provides a $(1+\epsilon)$-approximation to the number of distinct elements at all points in the stream. Although this algorithm is suboptimal for insertion-only streams, we show that it is amenable to the conditions of a composable histogram (Theorem 6). Namely, we show there is a sketch of this algorithm that is monotonic over suffixes of the stream, and thus there exists an efficient encoding that efficiently stores $D(t_i : t_{i+1})$ for each $1 \leq i < T$, which allows us to reduce the space overhead for the distinct elements problem.

For $\ell_2$-heavy hitters (Section 4), we show that the $\ell_2$ norm algorithm of [12] also satisfies the sketching requirement. Thus, plugging this into Algorithm 1 yields a method to maintain
an estimate of \( \ell_2 \). Algorithm 2 uses this subroutine to return the identities of the heavy hitters. However, we would still require that all \( n \) instances succeed since even \( \mathcal{O}(1) \) instances that fail adversarially could render the entire structure invalid by tricking the histogram into deleting the wrong information (see [19] for details). We show that the \( \ell_2 \) norm algorithm of [12] actually contains additional structure that only requires the correctness of polylog\( (n) \) instances, thus improving our space usage.

1.3 Lower Bounds

Distinct elements.

To show a lower bound of \( \Omega \left( \frac{1}{\epsilon^2} \log^2 n + \frac{1}{\epsilon} \log n \right) \) for the distinct elements problems, we show in Theorem 19 a lower bound of \( \Omega \left( \frac{1}{\epsilon^2} \log^2 n \right) \) and we show in Theorem 22 a lower bound of \( \Omega \left( \frac{1}{\epsilon} \log n \right) \). We first obtain a lower bound of \( \Omega \left( \frac{1}{\epsilon^2} \log^2 n \right) \) by a reduction from the IndexGreater problem, where Alice is given a string \( S = x_1x_2 \ldots x_m \) and each \( x_i \) has \( n \) bits so that \( S \) has \( mn \) bits in total. Bob is given integers \( i \in [m] \) and \( j \in [2^n] \) and must determine whether \( x_i > j \) or \( x_i \leq j \).

Given an instance of the IndexGreater problem, Alice splits the data stream into blocks of size \( \mathcal{O} \left( \frac{1}{\epsilon^2} \log n \right) \) and further splits each block into \( \sqrt{n} \) pieces of length \( (1 + 2\epsilon)^k \), padding the remainder of each block with zeros if necessary. For each \( i \in [m] \), Alice encodes \( x_i \) by inserting the elements \( \{0, 1, \ldots, (1 + 2\epsilon)^k - 1\} \) into piece \( x_i \) of block \((\ell - i + 1)\). Thus, the number of distinct elements in each block is much larger than the sum of the number of distinct elements in the subsequent blocks. Furthermore, the location of the distinct elements in block \((\ell - i + 1)\) encodes \( x_i \), so that Bob can recover \( x_i \) and compare it with \( j \).

We then obtain a lower bound of \( \Omega \left( \frac{1}{\epsilon^2} \log n \right) \) by a reduction from the GapHamming problem. In this problem, Alice and Bob receive length-\( n \) bitstrings \( x \) and \( y \), which have Hamming distance either at least \( \frac{n}{2} + \epsilon \sqrt{n} \) or at most \( \frac{n}{2} - \epsilon \sqrt{n} \), and must decide whether the Hamming distance between \( x \) and \( y \) is at least \( \frac{n}{2} \). Recall that for \( \epsilon \leq \frac{1}{\sqrt{n}} \), a \((1 + \epsilon)\)-approximation can differentiate between at least \( \frac{n}{2} + \epsilon \sqrt{n} \) and at most \( \frac{n}{2} - \epsilon \sqrt{n} \). We use this idea to show a lower bound of \( \Omega \left( \frac{1}{\epsilon^2} \log n \right) \) by embedding \( \Omega(\log n) \) instances of GapHamming into the stream. As in the previous case, the number of distinct elements corresponding to each instance is much larger than the sum of the number of distinct elements for the remaining instances, so that a \((1 + \epsilon)\)-approximation to the number of distinct elements in the sliding window solves the GapHamming problem for each instance.

Heavy hitters.

To show a lower bound on the problem of finding \( \ell_p \)-heavy hitters in the sliding window model, we give a reduction from the AugmentedIndex problem. Recall that in the AugmentedIndex problem, Alice is given a length-\( n \) string \( S \in \{1, 2, \ldots, k\}^n \) (which we write as \([k]^n\)) while Bob is given an index \( i \in [n] \), as well as \( S[1, i - 1] \), and must output the \( i^{th} \) symbol of the string, \( S[i] \). To encode \( S[i] \) for \( S \in [k]^n \), Alice creates a data stream \( a_1 \circ a_2 \circ \ldots \circ a_b \) with the invariant that the heavy hitters in the suffix \( a_1 \circ a_{i+1} \circ \ldots \circ a_b \) encode \( S[i] \). Specifically, the heavy hitters in the suffix will be concentrated in the substream \( a_i \) and the identities of each heavy hitter in \( a_i \) gives a bit of information about the value of \( S[i] \). To determine \( S[i] \), Bob expires the elements \( a_1, a_2, \ldots, a_{i-1} \) so all that remains in the sliding window is \( a_i \circ a_{i+1} \circ \ldots \circ a_b \), whose heavy hitters encode \( S[i] \).
1.4 Related Work

The study of the distinct elements problem in the streaming model was initiated by Flajolet and Martin [44] and developed by a long line of work [1, 45, 6, 38, 43]. Kane, Nelson, and Woodruff [53] give an optimal algorithm, using $O\left(\frac{1}{\epsilon^2} \log n\right)$ bits of space, for providing a $(1 + \epsilon)$-approximation to the number of distinct elements in a data stream, with constant probability. Blasiok [9] shows that to boost this probability up to $1 - \delta$ for a given $0 < \delta < 1$, the standard approach of running $O\left(\log \frac{1}{\epsilon}\right)$ independent instances is actually sub-optimal and gives an optimal algorithm that uses $O\left(\frac{\log^{d-1}}{\epsilon}\log n\right)$ bits of space.

The $\ell_1$-heavy hitters problem was first solved by Misra and Gries, who give a deterministic streaming algorithm using $O\left(\frac{1}{\epsilon^2} \log n\right)$ space [59]. Other techniques include the CountMin sketch [32], sticky sampling [57], lossy counting [57], sample and hold [40], multi-stage bloom filters [21], sketch-guided sampling [54], and CountSketch [26]. Among the numerous applications of the $\ell_p$-heavy hitters problem are network monitoring [37, 62], denial of service prevention [40, 4, 31], moment estimation [51], $\ell_p$-sampling [60], finding duplicates [47], iceberg queries [41], and entropy estimation [22, 48].

A stronger notion of “heavy hitters” is the $\ell_2$-heavy hitters. This is stronger than the $\ell_1$-guarantee since if $f_i \geq \epsilon F_1$ then $f_i^2 \geq \epsilon^2 F_1^2 \geq \epsilon^2 F_2$ (and so $f_i \geq \epsilon \sqrt{F_2}$). Thus any algorithm that finds the $\ell_2$-heavy hitters will also find all items satisfying the $\ell_1$-guarantee. In contrast, consider a stream that has $f_i = \sqrt{n}$ for some $i$ and $f_j = 1$ for all other elements $j$ in the universe. Then the $\ell_2$-heavy hitters algorithm will successfully identify $i$ for some constant $\epsilon$, whereas an algorithm that only provides the $\ell_1$-guarantee requires $\epsilon = \frac{1}{\sqrt{n}}$, and therefore $\Omega(\sqrt{n} \log n)$ space for identifying $i$. Moreover, the $\ell_2$-guarantee is the best we can do in polylogarithmic space, since for $p > 2$ it has been shown that identifying $\ell_p$-heavy hitters requires $\Omega(n^{1-2/p})$ bits of space [23, 5].

The most fundamental data stream setting is the insertion-only model where elements arrive one-by-one. In the insertion-deletion model, a previously inserted element can be deleted (each stream element is assigned +1 or −1, generalizing the insertion-only model where only +1 is used). Finally, in the sliding window model, a length $n$ is given and the stream consists only of insertions; points expire after $n$ insertions, meaning that (unlike the insertion-deletion model) the deletions are implicit. Letting $S = s_1, s_2, \ldots$ be the stream, at time $t$ the frequency vector is built from the window $W = \{s_t-(n-1), \ldots, s_t\}$ as the active elements, whereas items $\{s_1, \ldots, s_t-(n-1)\}$ are expired. The objective is to identify and report the “heavy hitters”, namely, the items $i$ for which $f_i$ is large with respect to $W$.

Table 2 shows prior work for $\ell_2$-heavy hitters in the various streaming models. A retuning of CountSketch in [63] solves the problem of $\ell_2$-heavy hitters in $O(\log^2 n)$ bits of space. More recently, [13] presents an $\ell_2$-heavy hitters algorithm using $O(\log n \log \log n)$ space. This algorithm is further improved to an $O(\log n)$ space algorithm in [12], which is optimal.

In the insertion-deletion model, CountSketch is space optimal [26, 52], but the update time per arriving element is improved by [55]. Thus in some sense, the $\ell_2$-heavy hitters problem is completely understood in all regimes except the sliding window model. We provide a nearly optimal algorithm for this setting, as shown in Table 2.

We now turn our attention to the sliding window model. The pioneering work by Datar et al. [35] introduced the exponential histogram as a framework for estimating statistics in the sliding window model. Among the applications of the exponential histogram are quantities such as count, sum of positive integers, average, and $\ell_p$ norms. Numerous other significant works include improvements to count and sum [46], frequent itemsets [28], frequency counts and quantiles [2, 56], rarity and similarity [36], variance and $k$-medians [3]
Table 2 Space complexity in bits of computing \( \ell_2 \)-heavy hitters in various streaming models. We write \( n = |S| \) and to simplify bounds we assume \( \log n = \Omega(\log m) \).

<table>
<thead>
<tr>
<th>Model</th>
<th>Upper Bound</th>
<th>Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion-Only</td>
<td>( \Omega(\varepsilon^2 \log n) ) [12]</td>
<td>( \Omega(\varepsilon^2 \log n) ) [Folklore]</td>
</tr>
<tr>
<td>Insertion-Deletion</td>
<td>( \Omega(\varepsilon^2 \log^2 n) ) [26]</td>
<td>( \Omega(\varepsilon^2 \log^2 n) ) [52]</td>
</tr>
<tr>
<td>Sliding Windows</td>
<td>( \Omega(\varepsilon^2 \log^2 n \log \varepsilon^{-1} + \log \log n) ) [Theorem 15]</td>
<td>( \Omega(\varepsilon^2 \log^2 n) ) [Theorem 4]</td>
</tr>
</tbody>
</table>

and other geometric problems [42, 25]. Braverman and Ostrovsky [19] introduced the smooth histogram as a framework that extends to smooth functions. [19] also provides sliding window algorithms for frequency moments, geometric mean and longest increasing subsequence. The ideas presented by [19] also led to a number of other results in the sliding window model [34, 17, 20, 18, 27, 39, 14]. In particular, Braverman et al. [15] provide an algorithm that finds the \( \ell_2 \)-heavy hitters in the sliding window model with \( \phi = ce \) for some constant \( c > 0 \), using \( \Omega\left(\frac{1}{\varepsilon^2} \log^2 n\right) \) bits of space, improving on results by [49]. [7] also implements and provides empirical analysis of algorithms finding heavy hitters in the sliding window model. Significantly, these data structures consider insertion-only data streams for the sliding window model; once an element arrives in the data stream, it remains until it expires. It remains a challenge to provide a general framework for data streams that might contain elements “negative” in magnitude, or even strict turnstile models. For a survey on sliding window algorithms, we refer the reader to [11].

2 Composable Histogram Data Structure Framework

We first describe a data structure which improves upon smooth histograms for the estimation of functions with a certain class of algorithms. This data structure provides the intuition for the space bounds in Theorem 1. Before describing the data structure, we need the definition a smooth function.

**Definition 5** ([19]). A function \( f \geq 1 \) is \((\alpha, \beta)\)-smooth if it has the following properties:
- **Monotonicity** \( f(A) \geq f(B) \) for \( B \subseteq A \) (\( B \) is a suffix of \( A \))
- **Polynomial boundedness** There exists \( c > 0 \) such that \( f(A) \leq n^c \).
- **Smoothness** For any \( \varepsilon \in (0, 1) \), there exists \( \alpha \in (0, 1) \), \( \beta \in (0, \alpha] \) so that if \( B \subseteq A \) and \( (1 - \beta)f(A) \leq f(B) \), then \( (1 - \alpha)f(A \cup C) \leq f(B \cup C) \) for any adjacent \( C \).

We emphasize a crucial observation made in [19]. Namely, for \( p > 1 \), \( \ell_p \) is a \((\varepsilon, \varepsilon^p)\)-smooth function while for \( p \leq 1 \), \( \ell_p \) is a \((\varepsilon, \varepsilon)\)-smooth function.

Given a data stream \( S = p_1, p_2, \ldots, p_n \) and a function \( f \), let \( f(t_1, t_2) \) represent \( f \) applied to the substream \( p_{t_1}, p_{t_1+1}, \ldots, p_{t_2} \). Furthermore, let \( D(t_1 : t_2) \) represent the data structure used to approximate \( f(t_1, t_2) \).

**Theorem 6.** Let \( f \) be an \((\alpha, \beta)\)-smooth function so that \( f = \Omega(n^c) \) for some constant \( c \). Suppose that for all \( \varepsilon, \delta > 0 \):
1. There exists an algorithm \( \mathcal{A} \) that maintains at each time \( t \) a data structure \( D(1 : t) \) which allows it to output a value \( \hat{f}(1, t) \) so that
   \[
   \Pr \left[ |\hat{f}(1, t) - f(1, t)| \leq \varepsilon \frac{1}{2} f(1, t), \text{ for all } 0 \leq t \leq n \right] \geq 1 - \delta.
   \]
2. There exists an algorithm \( \mathcal{B} \) which, given \( D(t_1 : t_i) \) and \( D(t_i + 1 : t_{i+1}) \), can compute \( D(t_i : t_{i+1}) \). Moreover, suppose storing \( D(t_i : t_{i+1}) \) uses \( O(\varepsilon^{-c}) \) bits of space.
Then there exists an algorithm that provides a $(1 + \epsilon)$-approximation to $f$ on the sliding window, using $\mathcal{O}\left(\frac{1}{\beta} \log^2 n + \sum_{i=1}^{\frac{2}{\epsilon}} \log n, g_i \left(\epsilon, \frac{\delta}{n}\right)\right)$ bits of space.

We remark that the first condition of Theorem 6 is called “strong tracking” and well-motivated by [10].

### 3 Distinct Elements

We first show that a well-known streaming algorithm that provides a $(1 + \epsilon)$-approximation to the number of distinct elements actually also provides strong tracking. Although this algorithm uses $\mathcal{O}\left(\frac{1}{\epsilon}\log n\right)$ bits of space and is suboptimal for insertion-only streams, we show that it is amenable to the conditions of Theorem 6. Thus, we describe a few modifications to this algorithm to provide a $(1 + \epsilon)$-approximation to the number of distinct elements in the sliding window model.

Define $\text{lsb}(x)$ to be the 0-based index of least significant bit of a non-negative integer $x$ in binary representation. For example, $\text{lsb}(10) = 1$ and $\text{lsb}(0) := \log(m)$ where we assume $\log(m) = \mathcal{O}(\log(n))$. Let $S \subseteq [m]$ and $h : [m] \rightarrow \{0, 1\}^{\log m}$ be a random hash function. Let $S_k := \{s \in S : \text{lsb}(h(s)) \geq k\}$ so that $2^k|S_k|$ is an unbiased estimator for $|S|$. Moreover, for $k$ such that $\mathbb{E}[|S_k|] = \Theta\left(\frac{1}{\epsilon}\right)$, the standard deviation of $2^k|S_k|$ is $\mathcal{O}(\epsilon|S|)$. Let $h_2 : [m] \rightarrow [B]$ be a pairwise independent random hash function with $B = \frac{\log m}{\epsilon \omega}$. Let $\Phi_B(m)$ be the expected number of non-empty bins after $m$ balls are thrown at random into $B$ bins so that $\mathbb{E}[|h_2(S_k)|] = \Phi_B(|S_k|)$.

\textbf{Fact 7.} $\Phi_m(t) = t \left(1 - \left(1 - \frac{1}{\epsilon}\right)^m\right)$

Blasiok provides an optimal algorithm for a constant factor approximation to the number of distinct elements with strong tracking.

\textbf{Theorem 8 ([9]).} There is a streaming algorithm that, with probability $1 - \delta$, reports a $(1 + \epsilon)$-approximation to the number of distinct elements in the stream after every update and uses $\mathcal{O}\left(\frac{\log n + \log \delta^{-1}}{\epsilon^2} + \log n\right)$ bits of space.

Thus we define an algorithm \textbf{Oracle} that provides a 2-approximation to the number of distinct elements in the stream after every update, using $\mathcal{O}(\log n)$ bits of space.

Since we can specifically track up to $\mathcal{O}\left(\frac{1}{\epsilon}\right)$ distinct elements, let us consider the case where the number of distinct elements is $\omega\left(\frac{1}{\epsilon}\right)$. Given access to \textbf{Oracle} to output an estimate $K$, which is a 2-approximation to the number of distinct elements, we can determine an integer $k > 0$ for which $\frac{K}{2^k} = \mathcal{O}\left(\frac{1}{\epsilon}\right)$. Then the quantity $2^k \Phi_B^{-1}(|h_2(S_k)|)$ provides both strong tracking as well as a $(1 + \epsilon)$-approximation to the number of distinct elements:

\textbf{Lemma 9 ([9]).} The median of $\mathcal{O}(\log \log n)$ estimators $2^k \Phi_B^{-1}(|h_2(S_k)|)$ is a $(1 + \epsilon)$-approximation at all times for which the number of distinct elements is $\Theta\left(\frac{x^2}{\epsilon^2}\right)$, with constant probability.

Hence, it suffices to maintain $h_2(S_i)$ for each $1 \leq i \leq \log m$, provided access to \textbf{Oracle} to find $k$, and $\mathcal{O}(\log \log n)$ parallel repetitions are sufficient to decrease the variance.

Indeed, a well-known algorithm for maintaining $h_2(S_i)$ simply keeps a $\log m \times O\left(\frac{1}{\epsilon}\right)$ table $T$ of bits. For $0 \leq i \leq \log n$, row $i$ of the table corresponds to $h_2(S_i)$. Specifically, the bit in entry $(i, j)$ of $T$ corresponds to $0$ if $h_2(s) \neq j$ for all $s \in S_i$ and corresponds to $1$ if
there exists some \( s \in S_i \) such that \( h_2(s) = j \). Therefore, the table maintains \( h_2(S_i) \), so then Lemma 9 implies that the table also gives a \((1+\epsilon)\)-approximation to the number of distinct elements at all times, using \( \mathcal{O}\left(\epsilon^{-1}\log n\log \log n\right) \) bits of space and access to Oracle. Then the total space is \( \mathcal{O}\left(\frac{1}{\epsilon} \log n \log \log n\right) \) after again using \( \mathcal{O}\left(\log \log n\right) \) parallel repetitions to decrease the variance.

Naively using this algorithm in the sliding window model would give a space usage dependency of \( \mathcal{O}\left(\frac{1}{\epsilon^2} \log^2 n \log \log n\right) \). To improve upon this space usage, consider maintaining tables for substreams \((t_1, t), (t_2, t), (t_3, t), \ldots \) where \( t_1 < t_2 < t_3 < \ldots < t \). Let \( T_i \) represent the table corresponding to substream \((t_i, t)\). Since \((t_{i+1}, t)\) is a suffix of \((t_i, t)\), then the support of the table representing \((t_{i+1}, t)\) is a subset of the support of the table representing \((t_i, t)\). That is, if the entry \((a, b)\) of \( T_{i+1} \) is one, then the entry \((a, b)\) of \( T_i \) is one, and similarly for each \( j < i \). Thus, instead of maintaining \(\frac{1}{\epsilon} \log n \) tables of bits corresponding to each of the \((t_i, t)\), it suffices to maintain a single table \( T \) where each entry represents the ID of the last table containing a bit of one in the entry. For example, if the entry \((a, b)\) of \( T_3 \) is zero but the entry \((a, b)\) of \( T_8 \) is one, then the entry \((a, b)\) for \( T \) is 8. Hence, \( T \) is a table of size \( \log m \times \mathcal{O}\left(\frac{1}{\epsilon}\right) \), with each entry having size \( \mathcal{O}\left(\log \frac{1}{\epsilon} + \log \log n\right) \) bits, for a total space of \( \mathcal{O}\left(\frac{1}{\epsilon} \log n \left(\log \frac{1}{\epsilon} + \log \log n\right)\right) \) bits. Finally, we need \( \mathcal{O}\left(\frac{1}{\epsilon} \log^2 n\right) \) bits to maintain the starting index \( t_i \) for each of the \( \frac{1}{\epsilon} \log n \) tables represented by \( T \). Again using a number of repetitions, the space usage is \( \mathcal{O}\left(\frac{1}{\epsilon} \log n \left(\log \frac{1}{\epsilon} + \log \log n\right) \log \log n + \frac{1}{\epsilon} \log^2 n\right) \).

Since this table is simply a clever encoding of the \( \mathcal{O}\left(\frac{1}{\epsilon} \log n\right) \) tables used in the smooth histogram data structure, correctness immediately follows. We emphasize that the improvement in space follows from the idea of Theorem 6. That is, instead of storing a separate table for each instance of the algorithm in the smooth histogram, we instead simply keep the difference between each instance.

Finally, observe that each column in \( T \) is monotonically decreasing. This is because \( S_k := \{ s \in S : \text{lsb}(h(s)) \geq k \} \) is a subset of \( S_{k-1} \). Alternatively, if an item has been sampled to level \( k \), it must have also been sampled to level \( k-1 \). Instead of using \( \mathcal{O}\left(\log \frac{1}{\epsilon} + \log \log n\right) \) bits per entry, we can efficiently encode the entries for each column in \( T \) with the observation that each column is monotonically decreasing.

**Proof of Theorem 1.** Since the largest index of \( T_i \) is \( i = \frac{1}{\epsilon} \log n \) and \( T \) has \( \log m \) rows, the number of possible columns is \( \left(\frac{1}{\epsilon} \log n + \log m - 1\right) \), which can be encoded using \( \mathcal{O}\left(\log n \log \frac{1}{\epsilon}\right) \) bits. Correctness follows immediately from Lemma 9 and the fact that the estimator is monotonic. Again we use \( \mathcal{O}\left(\frac{1}{\epsilon} \log^2 n\right) \) bits to maintain the starting index \( t_i \) for each of the \( \frac{1}{\epsilon} \log n \) tables represented by \( T \). As \( T \) has \( \mathcal{O}\left(\frac{1}{\epsilon}\right) \) columns and accounting again for the \( \mathcal{O}\left(\log \log n\right) \) repetitions to decrease the variance, the total space usage is \( \mathcal{O}\left(\frac{1}{\epsilon} \log n \log \frac{1}{\epsilon} \log n + \frac{1}{\epsilon} \log^2 n\right) \) bits.

**4 \( \ell_p \) Heavy Hitters**

Subsequent analysis by Berinde et al. [8] proved that many of the classic \( \ell_2 \)-heavy hitter algorithms not only revealed the identity of the heavy hitters, but also provided estimates of their frequencies. Let \( f_{tail(k)} \) be the vector \( f \) whose largest \( k \) entries are instead set to zero. Then an algorithm that, for each heavy hitter \( i \), outputs a quantity \( \hat{f}_i \) such that \( |\hat{f}_i - f_i| \leq \epsilon |f_{tail(k)}| \) is said to satisfy the \((\epsilon, k)\)-tail guarantee. Jowhari et al. [52] show an algorithm that finds the \( \ell_2 \)-heavy hitters and satisfies the tail guarantee can also find the \( \ell_p \)-heavy hitters. Thus, we first show results for \( \ell_2 \)-heavy hitters and then use this property to prove results for \( \ell_p \)-heavy hitters.
To meet the space guarantees of Theorem 15, we describe an algorithm, Algorithm 2, that only uses the framework of Algorithm 1 to provide a 2-approximation of the $\ell_2$ norm of the sliding window. We detail the other aspects of Algorithm 2 in the remainder of the section.

Recall that Algorithm 1 partitions the stream into a series of “jump-points” where $f$ increases by a constant multiplicative factor. The oldest jump point is before the sliding window and initiates the active window, while the remaining jump points are within the sliding window. Therefore, it is possible for some items to be reported as heavy hitters after the first jump point, even though they do not appear in the sliding window at all! For example, if the active window has $\ell_2$ norm $2\lambda$, and the sliding window has $\ell_2$ norm $(1 + \epsilon)\lambda$, all $2\epsilon\lambda$ instances of a heavy hitter in the active window can appear before the sliding window even begins. Thus, we must prune the list containing all heavy hitters to avoid the elements with low frequency in the sliding window.

To account for this, we begin a counter for each element immediately after the element is reported as a potential heavy hitter. However, the counter must be sensitive to the sliding window, and so we attempt to use a smooth-histogram to count the frequency of each element reported as a potential heavy hitter. Even though the count function is $(\epsilon, \epsilon)$ smooth, the necessity to track up to $\Omega\left(\frac{1}{\epsilon^2}\right)$ heavy hitters prevents us from being able to $(1 + \epsilon)$-approximate the count of each element. Fortunately, a constant approximation of the frequency of each element suffices to reject the elements whose frequency is less than $\frac{1}{8} \ell_2$. This additional data structure improves the space dependency to $O\left(\frac{1}{\epsilon^2}\right)$.

4.1 Background for Heavy Hitters

We now introduce concepts from [13, 12] to show the conditions of Theorem 6 apply, first we can compute $X_i$, where $X_i$ is the frequency vector at time $t$. Then [12] shows that $Y_t = \frac{1}{\delta} \sum_{j=1}^N X_{j,t}$ is a reasonably good estimator for $F_2$. By keeping $X_j(t_0, t_1, X_j(t_1 + 1, t_2), \ldots, X_j(t_1 + 1, t)$, we can compute $X_{j,t}$ from these sketches. Hence, the conditions of Theorem 6 are satisfied for Estimator, so Algorithm 1 can be applied to estimate the $\ell_2$ norm. One caveat is that naively, we still require the probability of failure for each instance of Estimator to be at most $\frac{\delta}{\log n}$ for the data structure to succeed with probability at least $1 - \delta$. We show in Appendix A that it suffices to only require the probability of failure for each instance of Estimator to be at most $\frac{\delta}{\text{polylog} n}$, thus incurring only $O(\log n)$ additional space rather than $O(\log n)$. We now refer to a heavy hitter algorithm from [12] that is space optimal up to $\log \frac{1}{\epsilon}$ factors.

**Theorem 11** (Theorem 11 in [12]). For any $\epsilon > 0$ and $\delta \in \mathbb{R}$, there exists a one-pass streaming algorithm, denoted $(\epsilon, \delta) – \text{BPTree}$, that with probability at least $(1 - \delta)$, returns a set of $\frac{\epsilon}{2}$-heavy hitters containing every $\epsilon$-heavy hitter and an approximate frequency for every item returned satisfying the $(\epsilon, 1/\epsilon^2)$-tail guarantee. The algorithm uses $O\left(\frac{1}{\epsilon^2} \left(\log \frac{1}{\epsilon}\right) (\log n + \log m)\right)$ bits of space and has $O\left(\log \frac{1}{\epsilon^2}\right)$ update time and $O\left(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon^2}\right)$ retrieval time.
Algorithm 2 \(\epsilon\)-approximation to the \(\ell_2\)-heavy hitters in a sliding window.

**Input:** A stream \(S\) of updates \(p_i\) for an underlying vector \(v\) and a window size \(n\).

**Output:** A list including all elements \(i\) with \(f_i \geq \epsilon \ell_2\) and no elements \(j\) with \(f_j < \frac{1}{12}\ell_2\).

1. Maintain sketches \(D(p_{t_1} : p_{t_2}), D(p_{t_2} + 1 : p_{t_3}), \ldots, D(p_{t_k} + 1 : p_{t_l})\) to estimate the \(\ell_2\) norm.
   \(\triangleright\) Use Estimator and Algorithm 1 with parameters \(\left(\frac{1}{2}, \frac{4}{7}\right)\) here.
2. Let \(A_i\) be the merged sketch \(D(p_{t_1} + 1 : p_{t_k})\).
3. For each merged sketch \(A_i\), find a superset \(H_1\) of the \(\frac{1}{m}\)-heavy hitters.
   \(\triangleright\) Use \(\left(\frac{1}{12}, \frac{2}{7}\right)\) - BPTree here. (Theorem 11)
4. For each element in \(H_1\), create a counter.
   \(\triangleright\) Instantiate a \(2 - \text{SmoothCounter}\) for each of the \(O\left(\frac{1}{\epsilon}\right)\) elements reported in \(H_1\).
5. Let \(\hat{\ell}_2\) be the estimated \(\ell_2\) norm of \(A_1\).
   \(\triangleright\) Output of Estimator on \(A_1\). (Theorem 10)
6. For element \(i \in H_1\), let \(\hat{f}_i\) be the estimated frequency of \(i\).
   \(\triangleright\) Output by \(2 - \text{SmoothCounter}\). (Theorem 12)
7. Output any element \(i\) with \(\hat{f}_i \geq \frac{1}{4} \epsilon \hat{\ell}_2\).

Observe that Theorem 10 combined with Theorem 6 already yields a prohibitively expensive \(\frac{1}{\epsilon}\) dependency on \(\epsilon\). Thus, we can only afford to set \(\epsilon\) to some constant in Theorem 10 and have a constant approximation to \(F_2\) in the sliding window.

At the conclusion of the stream, the data structure of Theorem 6 has another dilemma: either it reports the heavy hitters for a set of elements \(S_1\) that is a superset of the sliding window or it reports the heavy hitters for a set of elements \(S_2\) that is a subset of the sliding window. In the former case, we can report a number of unacceptable false positives, elements that are heavy hitters for \(S_1\) but may not appear at all in the sliding window. In the latter case, we may entirely miss a number of heavy hitters, elements that are heavy hitters for the sliding window but arrive before \(S_2\) begins. Therefore, we require a separate smooth histogram to track the counter of specific elements.

**Theorem 12.** For any \(\epsilon > 0\), there exists an algorithm, denoted \((1 + \epsilon) - \text{SmoothCounter}\), that outputs a \((1 + \epsilon)\)-approximation to the frequency of a given element in the sliding window model, using \(O\left(\frac{1}{\epsilon}(\log n + \log m)\log n\right)\) bits of space.

The algorithm follows directly from Theorem 6 and the observation that \(\ell_1\) is \((\epsilon, \epsilon)\)-smooth.

### 4.2 \(\ell_2\)-Heavy Hitters Algorithm

We now prove Theorem 15 using Algorithm 2. We detail our \(\ell_2\)-heavy hitters algorithm in full, using \(\ell_2 = \sqrt{F_2}\) and \(\epsilon\)-heavy hitters to refer to the \(\ell_2\)-heavy hitters problem with parameter \(\epsilon\).

**Lemma 13.** Any element \(i\) with frequency \(f_i > \epsilon \ell_2\) is output by Algorithm 2.

**Lemma 14.** No element \(i\) with frequency \(f_i < \frac{1}{12}\ell_2(W)\) is output by Algorithm 2.

**Theorem 15.** Given \(\epsilon, \delta > 0\), there exists an algorithm in the sliding window model (Algorithm 2) that with probability at least \(1 - \delta\) outputs all indices \(i \in [m]\) for which \(f_i \geq \epsilon \sqrt{F_2}\) and reports no indices \(i \in [m]\) for which \(f_i \leq \frac{1}{12}\sqrt{F_2}\). The algorithm has space complexity (in bits) \(O\left(\frac{1}{\epsilon^2}(\log^2 n + \log n + \log \frac{1}{\epsilon})\right)\).
7:12 Nearly Optimal Distinct Elements and Heavy Hitters on Sliding Windows

Sliding window string $S$ of length $n$

<table>
<thead>
<tr>
<th>Block length: $6\epsilon n$ log $n$</th>
<th>$6\epsilon n$ log $n$</th>
<th>$6\epsilon n$ log $n$</th>
<th>$6\epsilon n$ log $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements ${0, 1, \ldots, (1 + 2\epsilon)^i - 1}$ inserted into piece $x_i$ of block $i$.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alice: $x_1 \ldots x_m$, where $m = \frac{1}{6\epsilon} \log n$.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Each $x_k$ is $\frac{1}{2} \log n$ bits.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2 Construction of distinct elements instance by Alice. Pieces of block $i$ have length $(1 + 2\epsilon)^i - 1$.

4.3 Extension to $\ell_p$ norms for $0 < p < 2$

To output a superset of the $\ell_p$-heavy hitters rather than the $\ell_2$-heavy hitters, recall that an algorithm provides the $(\epsilon, k)$-tail guarantee if the frequency estimate $\hat{f}_i$ for each heavy hitter $i \in [m]$ satisfies $|\hat{f}_i - f_i| \leq \epsilon \cdot ||f_{\text{tail}(k)}||_1$, where $f_{\text{tail}(k)}$ is the frequency vector $f$ in which the $k$ most frequent entries have been replaced by zero. Jowhari et al. [52] show the impact of $\ell_2$-heavy hitter algorithms that satisfy the tail guarantee.

▶ Lemma 16 ([52]). For any $p \in (0, 2]$, any algorithm that returns the $\epsilon^{p/2}$-heavy hitters for $\ell_2$ satisfying the tail guarantee also finds the $\epsilon$-heavy hitters for $\ell_p$.

The correctness of Theorem 3 immediately follows from Lemma 16 and Theorem 15.

5 Lower Bounds

5.1 Distinct Elements

To show a lower bound of $\Omega \left( \frac{1}{\epsilon^2} \log^2 n + \frac{1}{2\epsilon} \log n \right)$ for the distinct elements problem, we show in Theorem 19 a lower bound of $\Omega \left( \frac{1}{\epsilon} \log^2 n \right)$ and we show in Theorem 22 a lower bound of $\Omega \left( \frac{1}{\epsilon} \log n \right)$. We first obtain a lower bound of $\Omega \left( \frac{1}{\epsilon} \log^2 n \right)$ by a reduction from the IndexGreater problem.

▶ Definition 17. In the IndexGreater problem, Alice is given a string $S = x_1 x_2 \cdots x_m$ of length $mn$, and thus each $x_i$ has $n$ bits. Bob is given integers $i \in [m]$ and $j \in [2^n]$. Alice is allowed to send a message to Bob, who must then determine whether $x_i > j$ or $x_i \leq j$.

Given an instance of the IndexGreater problem, Alice first splits the data stream into blocks of size $O \left( \frac{m}{\epsilon \log n} \right)$. She further splits each block into $\sqrt{n}$ pieces of length $(1 + 2\epsilon)^k$, before padding the remainder of block $(\ell - k + 1)$ with zeros. To encode $x_i$ for each $i \in [m]$, Alice inserts the elements $\{0, 1, \ldots, (1 + 2\epsilon)^k - 1\}$ into piece $x_i$ of block $(\ell - i + 1)$, before padding the remainder of block $(\ell - k + 1)$ with zeros. In this manner, the number of distinct elements in each block dominates the number of distinct elements in the subsequent blocks. Moreover, the location of the distinct elements in block $(\ell - i + 1)$ encodes $x_i$, so that Bob can compare $x_i$ to $j$. We formalize this argument in Appendix B.

▶ Lemma 18. The one-way communication complexity of IndexGreater is $\Omega(nm)$ bits.
Theorem 19. Let $p > 0$ and $\epsilon, \delta \in (0, 1)$. Any one-pass streaming algorithm that returns a $(1 + \epsilon)$-approximation to the number of distinct elements in the sliding window model with probability $\frac{2}{3}$ requires $\Omega \left( \frac{1}{\epsilon} \log^2 n \right)$ space.

To obtain a lower bound of $\Omega \left( \frac{1}{\epsilon} \log n \right)$, we give a reduction from the GapHamming problem.

Definition 20 ([50]). In the GapHamming problem, Alice and Bob receive $n$ bit strings $x$ and $y$, which have Hamming distance either at least $\frac{n}{2} + \sqrt{n}$ or at most $\frac{n}{2} - \sqrt{n}$. Then Alice and Bob must decide which of these instances is true.

Chakrabarti and Regev show an optimal lower bound on the communication complexity of GapHamming.

Lemma 21 ([24]). The communication complexity of GapHamming is $\Omega(n)$.

Observe that a $(1 + \epsilon) \frac{n}{2} \leq \frac{n}{2} + \sqrt{n}$ for $\epsilon \leq \frac{2}{\sqrt{n}}$ and thus a $(1 + \epsilon)$-approximation can differentiate between at least $\frac{n}{2} + \sqrt{n}$ and at most $\frac{n}{2} - \sqrt{n}$. We use this idea to show a lower bound of $\Omega \left( \frac{1}{\epsilon} \log n \right)$ by embedding $\Omega(\log n)$ instances of GapHamming into the stream.

Theorem 22. Let $p > 0$ and $\epsilon, \delta \in (0, 1)$. Any one-pass streaming algorithm that returns a $(1 + \epsilon)$-approximation to the number of distinct elements in the sliding window model with probability $\frac{2}{3}$ requires $\Omega \left( \frac{1}{\epsilon} \log n \right)$ space for $\epsilon \leq \frac{1}{\sqrt{n}}$.

Hence, Theorem 2 follows from Theorem 19 and Theorem 22.

5.2 $\ell_p$-Heavy Hitters

To show a lower bound for the $\ell_p$-heavy hitters problem in the sliding window model, we consider the following variant of the AugmentedIndex problem. Let $k$ and $n$ be positive integers and $\delta \in [0, 1)$. Suppose the first player Alice is given a string $S \in [k]^n$, while the second player Bob is given an index $i \in [n]$, as well as $S[1, i - 1]$. Alice sends a message to Bob, and Bob must output $S[i]$ with probability at least $1 - \delta$.

Lemma 23 ([58]). Even if Alice and Bob have access to a source of shared randomness, Alice must send a message of size $\Omega((1 - \delta)n \log k)$ in a one-way communication protocol for the AugmentedIndex problem.

We reduce the AugmentedIndex problem to finding the $\ell_p$-heavy hitters in the sliding window model. To encode $S[i]$ for $S \in [k]^n$, Alice creates a data stream $a_1 \circ a_2 \circ \ldots \circ a_b$ with the invariant that the heavy hitters in the suffix $a_1 \circ a_{i+1} \circ \ldots \circ a_b$ encodes $S[i]$. Thus to determine $S[i]$, Bob just needs to run the algorithm for finding heavy hitters on sliding windows and expire the elements $a_1, a_2, \ldots, a_{i-1}$ so that remains in the sliding window is $a_i \circ a_{i+1} \circ \ldots \circ a_b$. We formally prove Theorem 4 in Appendix B.

References


Nearly Optimal Distinct Elements and Heavy Hitters on Sliding Windows


APPROX/RANDOM 2018


A Full Version

We show that the structure of the $F_2$ algorithm only requires the correctness of a specific $O(\text{polylog } n)$ algorithms in the data structure. Given a vector $v \in \mathbb{R}^m$, let $F_2(v) = v_1^2 + v_2^2 + \ldots + v_m^2$. Recall that the histogram creates a new algorithm each time a new element arrives in the data stream. Instead of requiring all $n$ algorithms perform correctly, we show that it suffices to only require the correctness of a specific $O(\text{polylog } n)$ of these algorithms.

Let $F$ be the value of $F_2$ on the most recent $n$ elements. For the purpose of analysis, we say that an algorithm is important if it is still maintained within the histogram when its output is at least $F / \log n$ and the algorithm never outputs anything greater than $8F \log \frac{3}{n}$.

We first show that with high probability, all algorithms correctly maintain a $\log n$-approximation of the value of $F_2$ for the corresponding frequency vector. Conditioned on each algorithm correctly maintaining a $\log n$-approximation, we then show that $O(\log^6 n)$ algorithms are important. Observe that an algorithm that reports a 2-approximation to $F$ is important. Furthermore, we show that any algorithm that is not important cannot influence the output of the histogram, conditioned on each algorithm correctly maintaining a $\log n$-approximation. Thus, it suffices to require correctness of strong tracking on these $O(\log^6 n)$ important algorithms and we apply a union bound over the $O(\log^6 n)$ important algorithms to ensure correctness. Hence for each algorithm, we require the probability of failure to be at most $O\left(\frac{\delta}{\log n}\right)$ for the histogram to succeed with probability at least $1 - \delta$. 
Fact 24. Given $m$-dimensional vectors $x, y, z$ with non-negative entries, then $F_2(x + y + z) - F_2(x + y) \geq F_2(x + z) - F_2(x)$.

Although the number of algorithms in the histogram at any given moment is at most $O(\log n)$, it may be possible that many algorithms have output at least $\frac{F_2}{2 \log n}$ only to be deleted at some point in time. We now show that in a window of size $2n$, there are only $O(\log^6 n)$ important algorithms.

Lemma 25. Conditioned on all algorithms in the stream correctly providing a $\log n$-approximation, then there are at most $O(\log^6 n)$ important algorithms that begin in the most recent $2n$ elements.

Proof. Let $s_1 < s_2 < \ldots < s_j$ be the starting points of important algorithms $A_1, A_2, \ldots, A_i$, respectively, that begin within the most recent $2n$ elements. For each $1 < j < i$, let $t_j$ be the first time that algorithm $A_j$ outputs a value that is at least $\frac{F_2}{2 \log n}$. The idea is to show at the end of the stream, the elements between $s_j$ and $s_{j+1}$ are responsible for an increase in $F_2$ by at least $\frac{cF_2}{\log n}$ for all $j$. Since an algorithm is important if it never outputs anything greater than $8F_2 \log^3 n$, then the $F_2$ value of the substream represented by the algorithm is at most $8F_2 \log^3 n$, and it follows that $i = O(\log^6 n)$.

Recall that to maintain the histogram, there exists a constant $c$ such that whenever two adjacent algorithms have output within a factor of $c$, then we delete one of these algorithms. Hence, $A_{j-1}$ must output a value that is at least $\frac{cF_2}{2 \log n}$ at time $t_j$. Otherwise, the histogram would have deleted algorithm $A_j$ before $t_j$, preventing $A_j$ from being important. Conditioning on correctness of a $\log n$-approximation of all algorithms, the value of $F_2$ on the frequency vector from $s_{j-1}$ to $t_j$ is at least $\frac{cF_2}{2 \log n}$.

In other words, the elements from time $s_{j-1}$ to $s_j$ are responsible for a difference of at least $\frac{cF_2}{2 \log n}$ between the $F_2$ values of the substreams represented by $A_{j-1}$ and $A_j$ at time $t_j$. Thus by Fact 24, the difference between the $F_2$ values of the substreams represented by $A_{j-1}$ and $A_j$ at any time $t \geq t_j$ is at least $\frac{cF_2}{2 \log n}$. By induction, the value of $F_2$ on the substream from $s_1$ to $t_j$ is at least $\frac{(j-1)cF_2}{2 \log n}$. Recall that the $F_2$ of the substream represented by any important algorithm is at most $8F_2 \log^3 n$. Therefore, $i = O(\log^6 n)$ and so at most $O(\log^6 n)$ algorithms are important.

Fact 26. For $x > 0$ and $a, b \geq 0$, $\frac{(x+a)^2}{x^2} \geq \frac{(x+a+b)^2}{(x+b)^2}$.

Corollary 27. For $a_i, b_i, x_i \geq 0$ where $\sum x_i^2 > 0$, $\sum \frac{(x_i+a_i)^2}{x_i^2} \geq \sum \frac{(x_i+a_i+b_i)^2}{(x_i+b_i)^2}$.

Lemma 28. Conditioned on all algorithms in the stream correctly providing a $\log n$-approximation, then any algorithm that outputs a value that is at least $8F_2 \log^3 n$ cannot delete an important algorithm that provides a $2$-approximation to $F$.

Proof. Note that any algorithm $A$ that outputs a value that is at least $8F_2 \log^3 n$ must represent a substream whose $F_2$ value is at least $8F_2 \log^2 n$ at the end of the stream, assuming a $\log n$-approximation of all algorithms. Observe that the substream represented by an important algorithm $B$ that provides a $2$-approximation has $F_2$ value at most $2F_2$ at the end of the stream. By Corollary 27, the ratio between the $F_2$ values of the substreams represented by $A$ and $B$ must be at least $4 \log^2 n$ at every previous point in time. Thus, if $A$ and $B$ always correctly maintain a $\log n$-approximation of the corresponding substreams, the ratio of the outputs between $A$ and $B$ is at least 4, so $A$ will never cause the histogram data structure to delete $B$.
Hence, it remains to show that with high probability, all algorithms correctly maintain a log $n$-approximation of the value of $F_2$ for the corresponding frequency vector. Recall that Estimator from Theorem 10 uses an AMS sketch so that the resulting frequency of each element $f_i$ is multiplied by a Rademacher random variable $R_i$.

**Theorem 29** (Khintchine’s inequality). Let $R \in \{-1, 1\}^n$ be chosen uniformly at random and $f \in \mathbb{R}^n$ be a given vector. Then for any even integer $p$, $\mathbb{E} \left[ \left( \sum_{i=1}^{n} R_i f_i \right)^p \right] \leq \sqrt{p} \| f \|_2^p$.

Although we would like to apply Khintchine’s inequality directly, the Rademacher random variables $R_i$ used in Estimator are log $n$-wise independent. Nevertheless, we can use independence to consider the log $n$-th moment of the resulting expression.

**Corollary 30.** Let $z_1, z_2, \ldots, z_m \in \{-1, 1\}$ be a set of log $n$-wise independent random variables and $f \in \mathbb{R}^m$ be a given vector. Then for any even integer $p \leq \log n$, $\mathbb{E} \left[ \left( \sum_{i=1}^{m} z_i f_i \right)^p \right] \leq \sqrt{p} \| f \|_2^p$.

We now show that each algorithm fails to maintain a log $n$-approximation of the value of $F_2$ for the corresponding frequency vector only with negligible probability.

**Lemma 31.** Let $z_1, z_2, \ldots, z_m \in \{-1, 1\}$ be a set of log $n$-wise independent random variables and $f \in \mathbb{R}^m$ be a given vector. Then $\mathbb{P} \left[ \left| \sum_{i=1}^{m} z_i f_i \right| \geq (\log n) \| f \|_2 \right] \leq \frac{1}{\log n \log n}$.

**Proof.** For the ease of notation, let $p = \log n$ be an even integer. Observe that

$$\mathbb{P} \left[ \left| \sum_{i=1}^{m} z_i f_i \right| \geq (\log n) \| f \|_2 \right] = \mathbb{P} \left[ \left| \sum_{i=1}^{m} z_i f_i \right|^p \geq (\log n)^p \| f \|_2^p \right].$$

By Markov’s inequality, $\mathbb{P} \left[ \left| \sum_{i=1}^{m} z_i f_i \right|^p \geq (\log n)^p \| f \|_2^p \right] \leq \frac{\mathbb{E} \left[ \left( \sum_{i=1}^{m} z_i f_i \right)^p \right]}{(\log n)^p \| f \|_2^p}$. By Corollary 30, it follows that $\frac{\mathbb{E} \left[ \left( \sum_{i=1}^{m} z_i f_i \right)^p \right]}{(\log n)^p \| f \|_2^p} \leq \frac{\sqrt{p}}{(\log n)^p \| f \|_2^p} = \frac{1}{\log n \log n}$. Therefore, with high probability, all algorithms correctly maintain a log $n$-approximation of the value of $F_2$ for the corresponding frequency vector.

**B Supplementary Proofs**

**Proof of Lemma 13.** Since the $\ell_2$ norm is a smooth function, and so there exists a smooth-histogram which is an $\left( \frac{1}{2}, \frac{1}{2} \right)$-estimation of the $\ell_2$ norm of the sliding window by Theorem 6. Thus, $\frac{1}{2} \ell_2(A_1) \leq \ell_2(W) \leq \frac{3}{2} \ell_2(A_1)$. With probability $1 - \frac{\epsilon}{2}$, any element $i$ whose frequency satisfies $f_i(W) \geq \epsilon \ell_2(W)$ must have $f_i(W) \geq \epsilon \ell_2(W) \geq \frac{1}{2} \ell_2(A_1)$ and is reported by $(\frac{1}{10}, \frac{1}{10})$-BPTree in Step 3.

Since BPTree is instantiated along with $A_1$, the sliding window may begin either before or after BPTree reports each heavy hitter. If the sliding window begins after the heavy hitter is reported, then all $f_i(W)$ instances are counted by SmoothCounter. Thus, the count of $f_i$ estimated by SmoothCounter is at least $f_i(W) \geq \epsilon \ell_2(W) \geq \frac{1}{2} \ell_2(A_1)$, and so Step 7 will output $i$.

On the other hand, the sliding window may begin before the heavy hitter is reported. Recall that the BPTree algorithm identifies and reports an element when it becomes an $\frac{\epsilon}{2}$-heavy hitter with respect to the estimate of $\ell_2$. Hence, there are at most $2 \cdot \frac{\epsilon}{\ell_2(A_1)} \leq \frac{2 \epsilon}{\ell_2(A_1)}$ instances of an element appearing in the active window before it is reported by BPTree. Since $f_i(W) \geq \epsilon \ell_2(W) \geq \frac{1}{2} \ell_2(A_1)$, any element $i$ whose frequency satisfies $f_i(W) \geq \epsilon \ell_2(W) \geq \frac{1}{2} \ell_2(A_1)$ is reported by BPTree and thus Step 7 will output $i$.
\( \ell_2(W) \) must have \( f_i(W) \geq \frac{5}{4} \hat{\ell}_2(A_1) \) and therefore must have at least \( (\frac{1}{2} - \frac{1}{4}) \ell_2(A_1) \) instances appearing in the stream after it is reported by BPTree. Thus, the count of \( f_i \) estimated by SmoothCounter is at least \( \frac{1}{4} \hat{\ell}_2(A_1) \), and so Step 7 will output \( i \).

**Proof of Lemma 14.** If \( i \) is output by Step 7, then \( \hat{f}_i \geq \frac{1}{4} \hat{\ell}_2(A_1) \). By the properties of SmoothCounter and Estimator, \( f_i(W) \geq \frac{5}{4} \geq \frac{1}{4} \hat{\ell}_2(A_1) \geq \frac{1}{12} \hat{\ell}_2(W) \), where the last inequality comes from the fact that \( \ell_2(W) \leq \frac{5}{4} \hat{\ell}_2(A_1) \).

**Proof of Theorem 15.** By Lemma 13 and Lemma 14, Algorithm 2 outputs all elements with frequency at least \( \ell_2(W) \) and no elements with frequency less than \( \frac{\ell_2(W)}{2} \). We now proceed to analyze the space complexity of the algorithm. Step 1 uses Algorithm 1 in conjunction with the AugmentedIndex problem, which requires \( O(\text{polylogn}) \) in Theorem 10 and observing that \( \beta = O(1) \) in Theorem 6 suffices for a \( \ell_2 \)-approximation, it follows that Step 1 uses \( O(\log n(\log n + \log m \log^2 \log m)) \) bits of space. Since Step 3 runs an instance of BPTree for each of the at most \( O(\log n) \) buckets, then by Theorem 11, it uses \( O\left(\frac{1}{\epsilon^2} \log 2 \log n \log (\log n + \log m)\right) \) bits of space.

Notice that BPTree returns a list of \( O\left(\frac{1}{\epsilon^2}\right) \) elements, by Theorem 11. By running SmoothCounter for each of these, Step 7 provides a 2-approximation to the frequency of each element after being returned by BPTree. By Theorem 12, Step 7 has space complexity (in bits) \( O\left(\frac{1}{\epsilon^2} \log 2 \log (\log n + \log m) \right) \). Assuming \( \log m = O(\log n) \), the algorithm uses \( O\left(\frac{1}{\epsilon^2} \log^2 n \log (\log^2 n + \log \frac{1}{\epsilon})\right) \) bits of space.

**Proof of Theorem 3.** By Theorem 11, BPTree satisfies the tail guarantee. Therefore by Lemma 16, it suffices to analyze the space complexity of finding the \( \epsilon/\sqrt{2} \)-heavy hitters for \( \ell_2 \). By Theorem 15, there exists an algorithm that uses \( O\left(\frac{1}{\epsilon^2} \log n \log (\log n + \log \frac{1}{\epsilon})\right) \) bits of space to find the \( \epsilon \)-heavy hitters for \( \ell_2 \). Hence, there exists an algorithm that uses \( O\left(\frac{1}{\epsilon^2} \log^2 n \log (\log^2 n + \log \frac{1}{\epsilon})\right) \) bits of space to find the \( \epsilon \)-heavy hitters for \( \ell_p \), where \( 0 < p < 2 \).

**Proof of Lemma 18.** We show the communication complexity of IndexGreater through a reduction from the AugmentedIndex problem. Suppose Alice is given a string \( S \in \{0,1\}^{nm} \) and Bob is given an index \( i \) along with the bits \( S[1], S[2], \ldots, S[i-1] \). Then Bob’s task in the AugmentedIndex problem is to determine \( S[i] \).

Observe that Alice can form the string \( T = x_1 x_2 \cdots x_m \) of length \( mn \), where each \( x_k \) has \( n \) bits of \( S \). Alice can then use the IndexGreater protocol and communicate to Bob a message that will solve the IndexGreater problem. Let \( j = \lfloor \frac{i}{n} \rfloor \) so that the symbol \( S[i] \) is a bit inside \( x_{j+1} \). Then Bob constructs the string \( w \) by first concatenating the bits \( S[jn+1], S[jn+2], \ldots, S[i-1] \), which he is given from the AugmentedIndex problem. Bob then appends a zero to \( w \) and pads \( w \) with ones at the end, until \( w \) reaches \( n \) bits:

\[
w = S[jn+1] \circ S[jn+2] \circ \cdots \circ S[i-1] \circ 0 \circ 1 \circ 1 \circ \cdots \circ 1 .
\]

Bob takes the message from Alice and runs the IndexGreater protocol to determine whether \( x_j > w \). Observe that by construction \( x_j > w \) if and only if \( S[i] = 1 \). Thus, if the IndexGreater protocol succeeds, then Bob will have solved the AugmentedIndex problem, which requires communication complexity \( \Omega(nm) \) bits. Hence, the communication complexity of IndexGreater follows.
Proof of Theorem 19. We reduce a one-way communication protocol for IndexGreater to finding a \((1 + \epsilon)\)-approximation to the number of distinct elements in the sliding window model.

Let \(n\) be the length of the sliding window and suppose Alice receives a string \(S = x_1 x_2 \ldots x_\ell \in \{0, 1\}^\ell\), where \(\ell = \frac{1}{\epsilon^2} \log n\) and each \(x_k\) has \(\frac{1}{2} \log n\) bits. Bob receives an index \(i \in [\ell]\) and a nonnegative integer \(j \in [\sqrt{n}]\). Suppose Alice partitions the sliding window into \(\ell\) blocks, each of length \(\ell^2 = \frac{\epsilon n}{\log n}\). For each \(1 \leq k \leq \frac{1}{\epsilon^2} \log n\), she further splits block \((\ell - k + 1)\) into \(\sqrt{n}\) pieces of length \((1 + 2\epsilon)k\), before padding the remainder of block \((\ell - k + 1)\) with zeros. Moreover, for piece \(x_k\) of block \((\ell - k + 1)\), Alice inserts the \(\{0, 1, \ldots, (1 + 2\epsilon)k - 1\}\), before padding the remainder of block \((\ell - k + 1)\) with zeros. Hence, the sliding window contains all zeros, with the exception of the elements \(\{0, 1, \ldots, (1 + 2\epsilon)k - 1\}\) appearing in piece \(x_k\) of block \((\ell - k + 1)\) for all \(1 \leq k \leq \ell = \frac{1}{\epsilon^2} \log n\). Note that \((1 + 2\epsilon)k \leq \sqrt{n}\) and \(x_k \leq \sqrt{n}\) for all \(k\), so all the elements fit within each block, which has length \(\frac{\epsilon n}{\log n}\). Finally, Alice runs the \((1 + \epsilon)\)-approximation distinct elements sliding window algorithm and passes the state to Bob. See Figure 2 for an example of Alice’s construction.

Given integers \(i \in [\ell]\) and \(j \in [\sqrt{n}]\), Bob must determine if \(x_i > j\). Thus, Bob is interested in \(x_i\), so he takes the state of the sliding window algorithm, and inserts a number of zeros to expire each block before block \(i\). Note that since Alice reversed the stream in her final step, Bob can do this by inserting \((\ell - i)\left(\frac{1}{\epsilon^2} \log n\right)\) number of zeros. Bob then inserts \((j - 1)(1 + 2\epsilon)^i\) additional zeros, to arrive at piece \(j\) in block \(i\). Since piece \(x_j\) contains \((1 + 2\epsilon)^i\) distinct elements and the remainder of the stream contains \((1 + 2\epsilon)^{\ell - 1}\) distinct elements, then the output of the algorithm will decrease below \(\frac{1 + 2\epsilon}{1 + \epsilon}\) during piece \(x_j\). Hence, if the output is less than \(\frac{1 + 2\epsilon}{1 + \epsilon}\) after Bob arrives at piece \(j\), then \(x_i \leq j\). Otherwise, if the output is at least \(\frac{1 + 2\epsilon}{1 + \epsilon}\), then \(x_i > j\). By the communication complexity of IndexGreater (Lemma 18), this requires space \(\Omega\left(\frac{1}{\epsilon^2} \log^2 n\right)\).

Proof of Theorem 22. We reduce a one-way communication protocol for the GapHamming problem to finding a \((1 + \epsilon)\)-approximation to the number of distinct elements in the sliding window model. For each \(\frac{\log \frac{1}{\epsilon}}{2} \leq i \leq \log \frac{n - 1}{2}\), let \(j = 2i\) and \(x_j\) and \(y_j\) each have length \(2^i\) and \((x_j, y_j)\) be drawn from a distribution such that with probability \(\frac{1}{2}\), \(\text{HAM}(x_j, y_j) = (1 + 4\epsilon)2^{2i - 1}\) and otherwise (with probability \(\frac{1}{2}\)), \(\text{HAM}(x_j, y_j) = (1 - 4\epsilon)2^{2i - 1}\). Then Alice is given \(\{x_j\}\) while Bob is given \(\{y_j\}\) and needs to output \(\text{HAM}(x_j, y_j)\). For \(\epsilon \leq \frac{1}{\sqrt{n}}\), this is precisely the hard distribution in the communication complexity of GapHamming given by [24].

Let \(a = \frac{\log \frac{1}{\epsilon}}{2}\) and \(b = \frac{\log \frac{n - 1}{2}}{2}\). Let \(w_{2k} = x_{2k}\) and let \(w_{2k - 1}\) be a string of length \(2^{2k - 1}\), all consisting of zeros. Suppose Alice forms the concatenated string \(S = w_{2k} \circ w_{2k - 1} \circ \cdots \circ w_{2a + 1} \circ w_{2a}\). Note that \(\sum_{k=2a}^{2b} 2^k \leq n\), so \(S\) has length less than \(n\). Alice then forms a data stream by the following process. She initializes \(k = 1\) and continuously increments \(k\) until \(k = n\). At each step, if \(S[k] = 0\) or \(k\) is longer than the length of \(S\), Alice inserts a 0 into the data stream. Otherwise, if \(S[k] = 1\), then Alice inserts \(k\) into the data stream. Meanwhile, Alice runs the \((1 + \epsilon)\)-approximation distinct elements sliding window algorithm and passes the state of the algorithm to Bob.

To find \(\text{HAM}(x_{2i}, y_{2i})\), Bob first expires \(\left(\sum_{k=2i+1}^{2b} 2^k\right) - 2^{2i}\) elements by inserting zeros into the data stream. Similar to Alice, Bob initializes \(k = 1\) and continuously increments \(k\) until \(k = 2^{2i}\). At each step, if \(y_{2i}[k] = 0\) (that is, the \(k^n\) bit of \(y_{2i}\) is zero), then Bob inserts a 0 into the data stream. Otherwise, if \(y_{2i}[k] = 1\), then Bob inserts \(k\) into the data stream. At the end of this procedure, the sliding window contains all zeros, nonzero values corresponding to the nonzero indices of the string \(x_{2i} \circ w_{2i-1} \circ x_{2i-2} \circ \cdots \circ x_{2b+2} \circ w_{2a+1} \circ x_{2a}\), and nonzero values...
corresponding to the nonzero indices of \( y_{2i} \). Observe that each \( w_j \) solely consists of zeros and \( \sum_{k=1}^{t-1} 2^{2k} < 2^{2t-1} \). Therefore, \( \text{HAM}(x_{2i}, y_{2i}) \) is at least \((1 - 4\epsilon)2^{2i-1}\) while the number of distinct elements in the sliding window is at most \((1 + 4\epsilon)2^{2i} \) while the number of distinct elements in the suffix \( x_{2i-2} \circ x_{2i-3} \cdots \) is at most \((1 + \epsilon)2^{2i-2}\). Thus, a \((1 + \epsilon)\)-approximation to the number of distinct elements differentiates between \( \text{HAM}(x_{2i}, y_{2i}) = (1 + 4\epsilon)2^{2i-1} \) and \( \text{HAM}(x_{2i}, y_{2i}) = (1 - 4\epsilon)2^{2i-1} \).

Since the sliding window algorithm succeeds with probability \( \frac{2}{3} \), then the GapHamming distance problem succeeds with probability \( \frac{2}{3} \) across the \( \Omega(\log n) \) values of \( i \). Therefore, any \((1 + \epsilon)\)-approximation sliding window algorithm for the number of distinct elements that succeeds with probability \( \frac{2}{3} \) requires \( \Omega\left(\frac{1}{\epsilon^2} \log n\right) \) space for \( \epsilon \leq \frac{1}{\sqrt{n}} \).

**Proof of Theorem 4.** We reduce a one-way communication protocol for the AugmentedIndex problem to finding the \( \ell_p \) heavy hitters in the sliding window model. Let \( a = \frac{\sqrt{n}}{\log n} \log \sqrt{n} \) and \( b = \log n \). Suppose Alice receives \( S = [2^n]^b \) and Bob receives \( i \in [b] \) and \( S[1, i-1] \). Observe that each \( S[i] \) is \( \frac{1}{\sqrt{n}} \log \sqrt{n} \) bits and so \( S[i] \) can be rewritten as \( S[i] = w_1 \circ w_2 \circ \ldots \circ w_t \), where each \( t = \frac{1}{\sqrt{n}} \) and so each \( w_i \) is \( \log \sqrt{n} \) bits.

To recover \( S[i] \), Alice and Bob run the following algorithm. First, Alice constructs data stream \( A = a_1 \circ a_2 \circ \ldots \circ a_b \), which can be viewed as updates to an underlying frequency vector in \( \mathbb{R}^n \). Each \( a_k \) consists of \( t \) updates, adding \( 2^{p(b-k)} \) to coordinates \( v_1, v_2, \ldots, v_t \) of the frequency vector, where the binary representation of each \( v_j \in [n] \) is the concatenation of the binary representation of \( j \) with the \( \log \sqrt{n} \) bit string \( w_j \). She then runs the sliding window heavy hitters algorithm and passes the state of the algorithm to Bob.

Bob expires all elements of the stream before \( a_t \), runs the sliding window heavy hitters algorithm on the resulting vector, and then computes the heavy hitters. We claim that the algorithm will output \( t \) heavy hitters and by concatenating the last \( \log \sqrt{n} \) bits of the binary representation of each of these heavy hitters, Bob will recover exactly \( S[i] \). Observe that the \( \ell_p \) norm of the underlying vector represented by \( a_1 \circ a_{i+1} \circ \ldots \circ a_b \) is exactly \( \left( \frac{1}{\sqrt{n}} \left( 1^p + 2^p + 4^p + \ldots + 2^{p(b-1)} \right) \right)^{1/p} \leq \frac{1}{\sqrt{n}} 2^{b-1} = \frac{1}{2} 2^{b-i} \). Let \( u_1, u_2, \ldots, u_t \) be the coordinates of the frequency vector incremented by Alice as part of \( a_t \). Each coordinate \( u_j \) has frequency \( 2^{b-i} \geq \epsilon \left( \frac{1}{2} 2^{b-i} \right) \), so that \( u_j \) is an \( \ell_p \)-heavy hitter.

Moreover, the first \( \log t \) bits of \( u_j \) encode \( j \in [t] \) while the next \( \log \sqrt{n} \) bits encode \( w_j \). Thus, Bob identifies each heavy hitter and finds the corresponding \( j \in [t] \) so that he can concatenate \( S[i] = w_1 \circ w_2 \circ \ldots \circ w_t \).