

Nearly Optimal Distinct Elements and Heavy Hitters on Sliding Windows

Vladimir Braverman¹

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

Elena Grigorescu²

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

Harry Lang³

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

David P. Woodruff⁴

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

Samson Zhou²

Department of Computer Science, Purdue University, West Lafayette, Indiana, USA
samsonzhou@gmail.com

Abstract

We study the *distinct elements* and ℓ_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 ℓ_p -heavy hitters that are nearly optimal in both n and ϵ .

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 $\mathcal{O}\left(\frac{1}{\epsilon^2} \log n \log \frac{1}{\epsilon} \log \log n + \frac{1}{\epsilon} \log^2 n\right)$ bits of space. For ℓ_p -heavy hitters, we provide an algorithm using space $\mathcal{O}\left(\frac{1}{\epsilon^p} \log^2 n (\log^2 \log n + \log \frac{1}{\epsilon})\right)$ for $0 < p \leq 2$, improving upon the best-known algorithm for ℓ_2 -heavy hitters (Braverman *et al.*, COCOON 2014), which has space complexity $\mathcal{O}\left(\frac{1}{\epsilon^4} \log^3 n\right)$. We also show complementing nearly optimal lower bounds of $\Omega\left(\frac{1}{\epsilon} \log^2 n + \frac{1}{\epsilon^2} \log n\right)$ for distinct elements and $\Omega\left(\frac{1}{\epsilon^p} \log^2 n\right)$ for ℓ_p -heavy hitters, both tight up to $\mathcal{O}(\log \log n)$ and $\mathcal{O}(\log \frac{1}{\epsilon})$ factors.

2012 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

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

¹This material is based upon work supported in part by the National Science Foundation under Grants No. 1447639, 1650041, and 1652257, Cisco faculty award, and by the ONR Award N00014-18-1-2364.

²Research supported in part by NSF CCF-1649515.

³This material is based upon work supported by the Franco-American Fulbright Commission. The author thanks INRIA (l'Institut national de recherche en informatique et en automatique) for hosting him during the writing of this paper.

⁴D. Woodruff would like to acknowledge the support by the National Science Foundation under Grant No. CCF-1815840.



36 Digital Object Identifier [10.4230/LIPIcs.APPROX/RANDOM.2018.7](https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2018.7)

37 1 Introduction

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

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

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

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

56 ► **Problem 2 (ℓ_p -heavy hitters).** Given parameters $0 < \phi < \epsilon < 1$ and an input S of elements
 57 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
 58 $f_i \leq (\epsilon - \phi)(F_p)^{1/p}$, where $F_p = \sum_{i \in [m]} f_i^p$. (The parameter ϕ is typically assumed to be at
 59 least $c\epsilon$ for some fixed constant $0 < c < 1$.)

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

66 1.1 Our Contributions

67 Distinct elements.

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

72 ► **Theorem 1.** *Given $\epsilon > 0$, there exists an algorithm that, with probability at least $\frac{2}{3}$,
 73 provides a $(1 + \epsilon)$ -approximation to the number of distinct elements in the sliding window
 74 model, using $\mathcal{O}\left(\frac{1}{\epsilon^2} \log n \log \frac{1}{\epsilon} \log \log n + \frac{1}{\epsilon} \log^2 n\right)$ bits of space.*

75 A known lower bound is $\Omega\left(\frac{1}{\epsilon^2} + \log n\right)$ bits [1, 50] for insertion-only streams, which is also
 76 applicable to sliding windows since the model is strictly more difficult. We give a lower
 77 bound for distinct elements in the sliding window model, showing that our algorithm is
 78 nearly optimal, up to $\log \frac{1}{\epsilon}$ and $\log \log n$ factors, in both n and ϵ .

79 ► **Theorem 2.** Let $0 < \epsilon \leq \frac{1}{\sqrt{n}}$. Any one-pass streaming algorithm that returns a $(1 + \epsilon)$ -
 80 approximation to the number of distinct elements in the sliding window model with probability
 81 $\frac{2}{3}$ requires $\Omega\left(\frac{1}{\epsilon} \log^2 n + \frac{1}{\epsilon^2} \log n\right)$ bits of space.

82 ℓ_p -heavy hitters.

83 We first recall in [Lemma 16](#) a condition that allows the reduction from the problem of
 84 finding the ℓ_p -heavy hitters for $0 < p \leq 2$ to the problem of finding the ℓ_2 -heavy hitters. An
 85 algorithm of [\[12\]](#) allows us to maintain an estimate of F_2 . However, observe in [Problem 2](#)
 86 that an estimate for F_2 is only part of the problem. We must also identify which elements are
 87 heavy. First, we show how to use tools from [\[13\]](#) to find a superset of the heavy hitters. This
 88 alone is not enough since we may return false-positives (elements such that $f_i < (\epsilon - \phi)\sqrt{F_2}$).
 89 By keeping a careful count of the elements (shown in [Section 4](#)), we are able to remove these
 90 false-positives and obtain the following result, where we have set $\phi = \frac{11}{12}\epsilon$:

91 ► **Theorem 3.** Given $\epsilon > 0$ and $0 < p \leq 2$, there exists an algorithm in the sliding window
 92 model that, with probability at least $\frac{2}{3}$, outputs all indices $i \in [m]$ for which $f_i \geq \epsilon F_p^{1/p}$, and
 93 reports no indices $i \in [m]$ for which $f_i \leq \frac{\epsilon}{12} F_p^{1/p}$. The algorithm has space complexity (in
 94 bits) $\mathcal{O}\left(\frac{1}{\epsilon^p} \log^2 n (\log^2 \log n + \log \frac{1}{\epsilon})\right)$.

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

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

100 More details are provided in [Section 4](#) and [Section 5](#).

101 By standard amplification techniques any result that succeeds with probability $\frac{2}{3}$ can be
 102 made to succeed with probability $1 - \delta$ while multiplying the space and time complexities by
 103 $\mathcal{O}(\log \frac{1}{\delta})$. Therefore [Theorem 1](#) and [Theorem 15](#) can be taken with regard to any positive
 104 probability of failure.

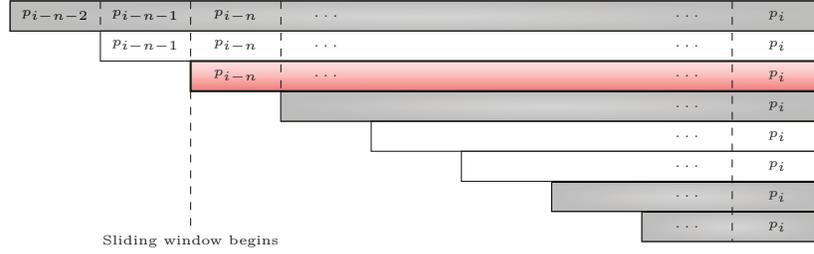
105 See [Table 1](#) for a comparison between our results and previous work.

Problem	Previous Bound	New Bound
ℓ_2 -heavy hitters	$\mathcal{O}\left(\frac{1}{\epsilon^4} \log^3 n\right)$ [15]	$\mathcal{O}\left(\frac{1}{\epsilon^2} \log^2 n (\log^2 \log n + \log^2 \frac{1}{\epsilon})\right)$
Distinct elements	$\mathcal{O}\left(\frac{1}{\epsilon^3} \log^2 n + \frac{1}{\epsilon} \log^3 n\right)$ [53, 19]	$\mathcal{O}\left(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon} \log n \log \log n + \frac{1}{\epsilon} \log^2 n\right)$

■ **Table 1** Our improvements for ℓ_2 -heavy hitters and distinct elements in the sliding window model.

106 1.2 Our Techniques

107 We introduce a simple extension of the exponential and smooth histogram frameworks, which
 108 use several instances of an underlying streaming algorithm. In contrast with the existing
 109 frameworks where $\mathcal{O}(\log n)$ different sketches are maintained, we observe in [Section 2](#) when
 110 the underlying algorithm has certain guarantees, then we can store these sketches more
 111 efficiently.



■ **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 $\mathcal{O}(\log n)$ instances (shaded grey) so that the value of f on any window can be $(1 + \epsilon)$ -approximated by its value on an adjacent window.

112 Sketching Algorithms

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

Algorithm 1 Input: a stream of elements p_1, p_2, \dots 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:       Delete  $t_j$ ; update indices;  $T \leftarrow T - 1$ 
9:   if  $t_2 < i - n$  then
10:    Delete  $t_1$ ; update indices;  $T \leftarrow T - 1$ 
11:    $i \leftarrow i + 1$ 

```

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

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

128 For ℓ_2 -heavy hitters (Section 4), we show that the ℓ_2 norm algorithm of [12] also satisfies

129 the sketching requirement. Thus, plugging this into [Algorithm 1](#) yields a method to maintain
 130 an estimate of ℓ_2 . [Algorithm 2](#) uses this subroutine to return the identities of the heavy
 131 hitters. However, we would still require that all n instances succeed since even $\mathcal{O}(1)$ instances
 132 that fail adversarially could render the entire structure invalid by tricking the histogram into
 133 deleting the wrong information (see [\[19\]](#) for details). We show that the ℓ_2 norm algorithm
 134 of [\[12\]](#) actually contains additional structure that only requires the correctness of $\text{polylog}(n)$
 135 instances, thus improving our space usage.

136 1.3 Lower Bounds

137 Distinct elements.

138 To show a lower bound of $\Omega\left(\frac{1}{\epsilon} \log^2 n + \frac{1}{\epsilon^2} \log n\right)$ for the distinct elements problems, we
 139 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
 140 bound of $\Omega\left(\frac{1}{\epsilon^2} \log n\right)$. We first obtain a lower bound of $\Omega\left(\frac{1}{\epsilon} \log^2 n\right)$ by a reduction from
 141 the `IndexGreater` problem, where Alice is given a string $S = x_1 x_2 \cdots x_m$ and each x_i has n
 142 bits so that S has mn bits in total. Bob is given integers $i \in [m]$ and $j \in [2^n]$ and must
 143 determine whether $x_i > j$ or $x_i \leq j$.

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

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

161 Heavy hitters.

162 To show a lower bound on the problem of finding ℓ_p -heavy hitters in the sliding window
 163 model, we give a reduction from the `AugmentedIndex` problem. Recall that in the `Augmente-`
 164 `dIndex` problem, Alice is given a length- n string $S \in \{1, 2, \dots, k\}^n$ (which we write as $[k]^n$)
 165 while Bob is given an index $i \in [n]$, as well as $S[1, i - 1]$, and must output the i^{th} symbol of
 166 the string, $S[i]$. To encode $S[i]$ for $S \in [k]^n$, Alice creates a data stream $a_1 \circ a_2 \circ \dots \circ a_b$ with
 167 the invariant that the heavy hitters in the suffix $a_i \circ a_{i+1} \circ \dots \circ a_b$ encode $S[i]$. Specifically,
 168 the heavy hitters in the suffix will be concentrated in the substream a_i and the identities
 169 of each heavy hitter in a_i gives a bit of information about the value of $S[i]$. To determine
 170 $S[i]$, Bob expires the elements a_1, a_2, \dots, a_{i-1} so all that remains in the sliding window is
 171 $a_i \circ a_{i+1} \circ \dots \circ a_b$, whose heavy hitters encode $S[i]$.

172 **1.4 Related Work**

173 The study of the distinct elements problem in the streaming model was initiated by Flajolet
 174 and Martin [44] and developed by a long line of work [1, 45, 6, 38, 43]. Kane, Nelson, and
 175 Woodruff [53] give an optimal algorithm, using $\mathcal{O}(\frac{1}{\epsilon^2} + \log n)$ bits of space, for providing a
 176 $(1 + \epsilon)$ -approximation to the number of distinct elements in a data stream, with constant
 177 probability. Blasiok [9] shows that to boost this probability up to $1 - \delta$ for a given $0 < \delta < 1$,
 178 the standard approach of running $\mathcal{O}(\log \frac{1}{\delta})$ independent instances is actually sub-optimal
 179 and gives an optimal algorithm that uses $\mathcal{O}(\frac{\log \delta^{-1}}{\epsilon^2} + \log n)$ bits of space.

180 The ℓ_1 -heavy hitters problem was first solved by Misra and Gries, who give a determinis-
 181 tic streaming algorithm using $\mathcal{O}(\frac{1}{\epsilon} \log n)$ space [59]. Other techniques include the CountMin
 182 sketch [32], sticky sampling [57], lossy counting [57], sample and hold [40], multi-stage bloom
 183 filters [21], sketch-guided sampling [54], and CountSketch [26]. Among the numerous appli-
 184 cations of the ℓ_p -heavy hitters problem are network monitoring [37, 62], denial of service
 185 prevention [40, 4, 31], moment estimation [51], ℓ_p -sampling [60], finding duplicates [47],
 186 iceberg queries [41], and entropy estimation [22, 48].

187 A stronger notion of “heavy hitters” is the ℓ_2 -heavy hitters. This is stronger than the
 188 ℓ_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
 189 algorithm that finds the ℓ_2 -heavy hitters will also find all items satisfying the ℓ_1 -guarantee.
 190 In contrast, consider a stream that has $f_i = \sqrt{m}$ for some i and $f_j = 1$ for all other elements
 191 j in the universe. Then the ℓ_2 -heavy hitters algorithm will successfully identify i for some
 192 constant ϵ , whereas an algorithm that only provides the ℓ_1 -guarantee requires $\epsilon = \frac{1}{\sqrt{n}}$, and
 193 therefore $\Omega(\sqrt{n} \log n)$ space for identifying i . Moreover, the ℓ_2 -guarantee is the best we can
 194 do in polylogarithmic space, since for $p > 2$ it has been shown that identifying ℓ_p -heavy
 195 hitters requires $\Omega(n^{1-2/p})$ bits of space [23, 5].

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

205 Table 2 shows prior work for ℓ_2 -heavy hitters in the various streaming models. A retuning
 206 of CountSketch in [63] solves the problem of ℓ_2 -heavy hitters in $\mathcal{O}(\log^2 n)$ bits of space.
 207 More recently, [13] presents an ℓ_2 -heavy hitters algorithm using $\mathcal{O}(\log n \log \log n)$ space.
 208 This algorithm is further improved to an $\mathcal{O}(\log n)$ space algorithm in [12], which is optimal.

209 In the insertion-deletion model, CountSketch is space optimal [26, 52], but the update
 210 time per arriving element is improved by [55]. Thus in some sense, the ℓ_2 -heavy hitters
 211 problem is completely understood in all regimes except the sliding window model. We
 212 provide a nearly optimal algorithm for this setting, as shown in Table 2.

213 We now turn our attention to the sliding window model. The pioneering work by Datar
 214 *et al.* [35] introduced the exponential histogram as a framework for estimating statistics
 215 in the sliding window model. Among the applications of the exponential histogram are
 216 quantities such as count, sum of positive integers, average, and ℓ_p norms. Numerous other
 217 significant works include improvements to count and sum [46], frequent itemsets [28], fre-
 218 quency counts and quantiles [2, 56], rarity and similarity [36], variance and k -medians [3] and

Model	Upper Bound	Lower Bound
Insertion-Only	$\mathcal{O}(\epsilon^{-2} \log n)$ [12]	$\Omega(\epsilon^{-2} \log n)$ [Folklore]
Insertion-Deletion	$\mathcal{O}(\epsilon^{-2} \log^2 n)$ [26]	$\Omega(\epsilon^{-2} \log^2 n)$ [52]
Sliding Windows	$\mathcal{O}(\epsilon^{-2} \log^2 n (\log \epsilon^{-1} + \log \log n))$ [Theorem 15]	$\Omega(\epsilon^{-2} \log^2 n)$ [Theorem 4]

■ **Table 2** Space complexity in bits of computing ℓ_2 -heavy hitters in various streaming models. We write $n = |S|$ and to simplify bounds we assume $\log n = \mathcal{O}(\log m)$.

219 other geometric problems [42, 25]. Braverman and Ostrovsky [19] introduced the smooth
 220 histogram as a framework that extends to smooth functions. [19] also provides sliding win-
 221 dows algorithms for frequency moments, geometric mean and longest increasing subsequence.
 222 The ideas presented by [19] also led to a number of other results in the sliding window model
 223 [34, 17, 20, 18, 27, 39, 14]. In particular, Braverman *et al.* [15] provide an algorithm that
 224 finds the ℓ_2 -heavy hitters in the sliding window model with $\phi = c\epsilon$ for some constant $c > 0$,
 225 using $\mathcal{O}(\frac{1}{\epsilon^4} \log^3 n)$ bits of space, improving on results by [49]. [7] also implements and pro-
 226 vides empirical analysis of algorithms finding heavy hitters in the sliding window model.
 227 Significantly, these data structures consider insertion-only data streams for the sliding win-
 228 dows model; once an element arrives in the data stream, it remains until it expires. It remains
 229 a challenge to provide a general framework for data streams that might contain elements
 230 “negative” in magnitude, or even strict turnstile models. For a survey on sliding window
 231 algorithms, we refer the reader to [11].

232 2 Composable Histogram Data Structure Framework

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

237 ► **Definition 5.** [19] A function $f \geq 1$ is (α, β) -smooth if it has the following properties:

238 **Monotonicity** $f(A) \geq f(B)$ for $B \subseteq A$ (B is a suffix of A)

239 **Polynomial boundedness** There exists $c > 0$ such that $f(A) \leq n^c$.

240 **Smoothness** For any $\epsilon \in (0, 1)$, there exists $\alpha \in (0, 1)$, $\beta \in (0, \alpha]$ so that if $B \subseteq A$ and
 241 $(1 - \beta)f(A) \leq f(B)$, then $(1 - \alpha)f(A \cup C) \leq f(B \cup C)$ for any adjacent C .

242 We emphasize a crucial observation made in [19]. Namely, for $p > 1$, ℓ_p is a $(\epsilon, \frac{\epsilon^p}{p})$ -smooth
 243 function while for $p \leq 1$, ℓ_p is a (ϵ, ϵ) -smooth function.

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

247 ► **Theorem 6.** Let f be an (α, β) -smooth function so that $f = \mathcal{O}(n^c)$ for some constant c .
 248 Suppose that for all $\epsilon, \delta > 0$:

249 (1) There exists an algorithm \mathcal{A} that maintains at each time t a data structure $D(1 : t)$
 250 which allows it to output a value $\hat{f}(1, t)$ so that

$$251 \Pr \left[|\hat{f}(1, t) - f(1, t)| \leq \frac{\epsilon}{2} f(1, t), \text{ for all } 0 \leq t \leq n \right] \geq 1 - \delta.$$

252 (2) There exists an algorithm \mathcal{B} which, given $D(t_1 : t_i)$ and $D(t_i + 1 : t_{i+1})$, can compute
 253 $D(t_i : t_{i+1})$. Moreover, suppose storing $D(t_i : t_{i+1})$ uses $\mathcal{O}(g_i(\epsilon, \delta))$ bits of space.

254 Then there exists an algorithm that provides a $(1 + \epsilon)$ -approximation to f on the sliding
 255 window, using $\mathcal{O}\left(\frac{1}{\beta} \log^2 n + \sum_{i=1}^{\frac{4}{\beta} \log n} g_i\left(\epsilon, \frac{\delta}{n}\right)\right)$ bits of space.

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

258 3 Distinct Elements

259 We first show that a well-known streaming algorithm that provides a $(1 + \epsilon)$ -approximation
 260 to the number of distinct elements actually also provides strong tracking. Although this al-
 261 gorithm uses $\mathcal{O}\left(\frac{1}{\epsilon^2} \log n\right)$ bits of space and is suboptimal for insertion-only streams, we show
 262 that it is amenable to the conditions of [Theorem 6](#). Thus, we describe a few modifications
 263 to this algorithm to provide a $(1 + \epsilon)$ -approximation to the number of distinct elements in
 264 the sliding window model.

265 Define $\text{lsb}(x)$ to be the 0-based index of least significant bit of a non-negative integer x
 266 in binary representation. For example, $\text{lsb}(10) = 1$ and $\text{lsb}(0) := \log(m)$ where we assume
 267 $\log(m) = \mathcal{O}(\log n)$. Let $S \subset [m]$ and $h : [m] \rightarrow \{0, 1\}^{\log m}$ be a random hash function. Let
 268 $S_k := \{s \in S : \text{lsb}(h(s)) \geq k\}$ so that $2^k |S_k|$ is an unbiased estimator for $|S|$. Moreover, for
 269 k such that $\mathbf{E}[|S_k|] = \Theta\left(\frac{1}{\epsilon^2}\right)$, the standard deviation of $2^k |S_k|$ is $\mathcal{O}(\epsilon |S|)$. Let $h_2 : [m] \rightarrow$
 270 $[B]$ be a pairwise independent random hash function with $B = \frac{100}{\epsilon^2}$. Let $\Phi_B(m)$ be the
 271 expected number of non-empty bins after m balls are thrown at random into B bins so that
 272 $\mathbf{E}[|h_2(S_k)|] = \Phi_B(|S_k|)$.

273 **► Fact 7.** $\Phi_m(t) = t \left(1 - \left(1 - \frac{1}{t}\right)^m\right)$

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

276 **► Theorem 8.** [\[9\]](#) *There is a streaming algorithm that, with probability $1 - \delta$, reports a*
 277 *$(1 + \epsilon)$ -approximation to the number of distinct elements in the stream after every update*
 278 *and uses $\mathcal{O}\left(\frac{\log \log n + \log \delta^{-1}}{\epsilon^2} + \log n\right)$ bits of space.*

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

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

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

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

291 Indeed, a well-known algorithm for maintaining $h_2(S_i)$ simply keeps a $\log m \times \mathcal{O}\left(\frac{1}{\epsilon^2}\right)$
 292 table T of bits. For $0 \leq i \leq \log n$, row i of the table corresponds to $h_2(S_i)$. Specifically, the
 293 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
 294 there exists some $s \in S_i$ such that $h_2(s) = j$. Therefore, the table maintains $h_2(S_i)$, so then

295 **Lemma 9** implies that the table also gives a $(1 + \epsilon)$ -approximation to the number of distinct
 296 elements at all times, using $\mathcal{O}(\frac{1}{\epsilon^2} \log n)$ bits of space and access to Oracle. Then the total
 297 space is $\mathcal{O}(\frac{1}{\epsilon^2} \log n \log \log n)$ after again using $\mathcal{O}(\log \log n)$ parallel repetitions to decrease
 298 the variance.

299 Naïvely using this algorithm in the sliding window model would give a space usage de-
 300 pendency of $\mathcal{O}(\frac{1}{\epsilon^3} \log^2 n \log \log n)$. To improve upon this space usage, consider maintaining
 301 tables for substreams $(t_1, t), (t_2, t), (t_3, t), \dots$ where $t_1 < t_2 < t_3 < \dots < t$. Let T_i represent
 302 the table corresponding to substream (t_i, t) . Since (t_{i+1}, t) is a suffix of (t_i, t) , then the
 303 support of the table representing (t_{i+1}, t) is a subset of the support of the table representing
 304 (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
 305 similarly for each $j < i$. Thus, instead of maintaining $\frac{1}{\epsilon} \log n$ tables of bits corresponding
 306 to each of the (t_i, t) , it suffices to maintain a single table T where each entry represents the
 307 ID of the *last* table containing a bit of one in the entry. For example, if the entry (a, b) of
 308 T_9 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
 309 table of size $\log m \times \mathcal{O}(\frac{1}{\epsilon^2})$, with each entry having size $\mathcal{O}(\log \frac{1}{\epsilon} + \log \log n)$ bits, for a total
 310 space of $\mathcal{O}(\frac{1}{\epsilon^2} \log n (\log \frac{1}{\epsilon} + \log \log n))$ bits. Finally, we need $\mathcal{O}(\frac{1}{\epsilon} \log^2 n)$ bits to maintain
 311 the starting index t_i for each of the $\frac{1}{\epsilon} \log n$ tables represented by T . Again using a number
 312 of repetitions, the space usage is $\mathcal{O}(\frac{1}{\epsilon^2} \log n (\log \frac{1}{\epsilon} + \log \log n) \log \log n + \frac{1}{\epsilon} \log^2 n)$.

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

318 Finally, observe that each column in T is monotonically decreasing. This is because
 319 $S_k := \{s \in S : \text{lsb}(h(s)) \geq k\}$ is a subset of S_{k-1} . Alternatively, if an item has been sampled
 320 to level k , it must have also been sampled to level $k-1$. Instead of using $\mathcal{O}(\log \frac{1}{\epsilon} + \log \log n)$
 321 bits per entry, we can efficiently encode the entries for each column in T with the observation
 322 that each column is monotonically decreasing.

323 **Proof of Theorem 1:** Since the largest index of T_i is $i = \frac{1}{\epsilon} \log n$ and T has $\log m$
 324 rows, the number of possible columns is $\binom{\frac{1}{\epsilon} \log n + \log m - 1}{\log m}$, which can be encoded using
 325 $\mathcal{O}(\log n \log \frac{1}{\epsilon})$ bits. Correctness follows immediately from **Lemma 9** and the fact that the
 326 estimator is monotonic. Again we use $\mathcal{O}(\frac{1}{\epsilon} \log^2 n)$ bits to maintain the starting index t_i
 327 for each of the $\frac{1}{\epsilon} \log n$ tables represented by T . As T has $\mathcal{O}(\frac{1}{\epsilon^2})$ columns and account-
 328 ing again for the $\mathcal{O}(\log \log n)$ repetitions to decrease the variance, the total space usage is
 329 $\mathcal{O}(\frac{1}{\epsilon^2} \log n \log \frac{1}{\epsilon} \log \log n + \frac{1}{\epsilon} \log^2 n)$ bits. \square

330 **4** ℓ_p Heavy Hitters

331 Subsequent analysis by Berinde *et al.* [8] proved that many of the classic ℓ_2 -heavy hitter
 332 algorithms not only revealed the identity of the heavy hitters, but also provided estimates
 333 of their frequencies. Let $f_{\text{tail}(k)}$ be the vector f whose largest k entries are instead set
 334 to zero. Then an algorithm that, for each heavy hitter i , outputs a quantity \hat{f}_i such that
 335 $|\hat{f}_i - f_i| \leq \epsilon \|f_{\text{tail}(k)}\|_1 \leq \epsilon \|f\|_1$ is said to satisfy the (ϵ, k) -tail guarantee. Jowhari *et al.* [52]
 336 show an algorithm that finds the ℓ_2 -heavy hitters and satisfies the tail guarantee can also
 337 find the ℓ_p -heavy hitters. Thus, we first show results for ℓ_2 -heavy hitters and then use this
 338 property to prove results for ℓ_p -heavy hitters.

339 To meet the space guarantees of **Theorem 15**, we describe an algorithm, **Algorithm 2**,

340 that only uses the framework of [Algorithm 1](#) to provide a 2-approximation of the ℓ_2 norm
 341 of the sliding window. We detail the other aspects of [Algorithm 2](#) in the remainder of the
 342 section.

343 Recall that [Algorithm 1](#) partitions the stream into a series of “jump-points” where f
 344 increases by a constant multiplicative factor. The oldest jump point is before the sliding
 345 window and initiates the active window, while the remaining jump points are within the
 346 sliding window. Therefore, it is possible for some items to be reported as heavy hitters
 347 after the first jump point, even though they do not appear in the sliding window at all! For
 348 example, if the active window has ℓ_2 norm 2λ , and the sliding window has ℓ_2 norm $(1 + \epsilon)\lambda$,
 349 all $2\epsilon\lambda$ instances of a heavy hitter in the active window can appear before the sliding window
 350 even begins. Thus, we must prune the list containing all heavy hitters to avoid the elements
 351 with low frequency in the sliding window.

352 To account for this, we begin a counter for each element immediately after the element
 353 is reported as a potential heavy hitter. However, the counter must be sensitive to the
 354 sliding window, and so we attempt to use a smooth-histogram to count the frequency of
 355 each element reported as a potential heavy hitter. Even though the count function is (ϵ, ϵ)
 356 smooth, the necessity to track up to $\mathcal{O}(\frac{1}{\epsilon^2})$ heavy hitters prevents us from being able to
 357 $(1 + \epsilon)$ -approximate the count of each element. Fortunately, a constant approximation of
 358 the frequency of each element suffices to reject the elements whose frequency is less than
 359 $\frac{\epsilon}{8}\ell_2$. This additional data structure improves the space dependency to $\mathcal{O}(\frac{1}{\epsilon^2})$.

360 4.1 Background for Heavy Hitters

361 We now introduce concepts from [\[13, 12\]](#) to show the conditions of [Theorem 6](#) apply, first
 362 describing an algorithm from [\[12\]](#) that provides a good approximation of F_2 at all times.

363 ► **Theorem 10** (Remark 8 in [\[12\]](#)). *For any $\epsilon \in (0, 1)$ and $\delta \in [0, 1)$, there exists a one-pass
 364 streaming algorithm Estimator that outputs at each time t a value $\hat{F}_2^{(t)}$ so that*

$$365 \Pr \left[|\hat{F}_2^{(t)} - F_2^{(t)}| \leq \epsilon F_2^{(t)}, \text{ for all } 0 \leq t \leq n \right] \geq 1 - \delta,$$

366 and uses $\mathcal{O}(\frac{1}{\epsilon^2} \log m (\log \log m + \log \frac{1}{\epsilon}) \log \frac{1}{\delta})$ bits of space and $\mathcal{O}((\log \log m + \log \frac{1}{\epsilon}) \log \frac{1}{\delta})$
 367 update time.

368 The algorithm of [Theorem 10](#) is a modified version of the AMS estimator [\[1\]](#) as follows.
 369 Given vectors Z_j of 6-wise independent Rademacher (i.e. uniform ± 1) random variables,
 370 let $X_j(t) = \langle Z_j, f^{(t)} \rangle$, where $f^{(t)}$ is the frequency vector at time t . Then [\[12\]](#) shows that
 371 $Y_t = \frac{1}{N} \sum_{j=1}^N X_{j,t}^2$ is a reasonably good estimator for F_2 . By keeping $X_j(1, t_1), X_j(t_1 +$
 372 $1, t_2), \dots, X_j(t_i + 1, t)$, we can compute $X_{j,t}$ from these sketches. Hence, the conditions of
 373 [Theorem 6](#) are satisfied for Estimator, so [Algorithm 1](#) can be applied to estimate the ℓ_2
 374 norm. One caveat is that naïvely, we still require the probability of failure for each instance
 375 of Estimator to be at most $\frac{\delta}{\log n}$ for the data structure to succeed with probability at least
 376 $1 - \delta$. We show in [Appendix A](#) that it suffices to only require the probability of failure for
 377 each instance of Estimator to be at most $\frac{\delta}{\text{polylog } n}$, thus incurring only $\mathcal{O}(\log \log n)$ additional
 378 space rather than $\mathcal{O}(\log n)$. We now refer to a heavy hitter algorithm from [\[12\]](#) that is space
 379 optimal up to $\log \frac{1}{\epsilon}$ factors.

380 ► **Theorem 11** (Theorem 11 in [\[12\]](#)). *For any $\epsilon > 0$ and $\delta \in [0, 1)$, there exists a one-
 381 pass streaming algorithm, denoted (ϵ, δ) – BPTree, that with probability at least $(1 - \delta)$,
 382 returns a set of $\frac{\epsilon}{2}$ -heavy hitters containing every ϵ -heavy hitter and an approximate fre-
 383 quency for every item returned satisfying the $(\epsilon, 1/\epsilon^2)$ -tail guarantee. The algorithm uses*

384 $\mathcal{O}\left(\frac{1}{\epsilon^2} \left(\log \frac{1}{\delta\epsilon}\right) (\log n + \log m)\right)$ bits of space and has $\mathcal{O}\left(\log \frac{1}{\delta\epsilon}\right)$ update time and $\mathcal{O}\left(\frac{1}{\epsilon^2} \log \frac{1}{\delta\epsilon}\right)$
 385 retrieval time.

386 Observe that [Theorem 10](#) combined with [Theorem 6](#) already yields a prohibitively expensive $\frac{1}{\epsilon^3}$ dependency on ϵ . Thus, we can only afford to set ϵ to some constant in [Theorem 10](#)
 387 and have a constant approximation to F_2 in the sliding window.
 388

389 At the conclusion of the stream, the data structure of [Theorem 6](#) has another dilemma:
 390 either it reports the heavy hitters for a set of elements \mathcal{S}_1 that is a superset of the sliding
 391 window or it reports the heavy hitters for a set of elements \mathcal{S}_2 that is a subset of the sliding
 392 window. In the former case, we can report a number of unacceptable false positives, elements
 393 that are heavy hitters for \mathcal{S}_1 but may not appear at all in the sliding window. In the latter
 394 case, we may entirely miss a number of heavy hitters, elements that are heavy hitters for
 395 the sliding window but arrive before \mathcal{S}_2 begins. Therefore, we require a separate smooth
 396 histogram to track the counter of specific elements.

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

400 The algorithm follows directly from [Theorem 6](#) and the observation that ℓ_1 is (ϵ, ϵ) -smooth.

401 4.2 ℓ_2 -Heavy Hitters Algorithm

402 We now prove [Theorem 15](#) using [Algorithm 2](#). We detail our ℓ_2 -heavy hitters algorithm
 403 in full, using $\ell_2 = \sqrt{F_2}$ and ϵ -heavy hitters to refer to the ℓ_2 -heavy hitters problem with
 parameter ϵ .

Algorithm 2 ϵ -approximation to the ℓ_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 $f_j < \frac{\epsilon}{12}\ell_2$.

- 1: Maintain sketches $D(p_{t_1} : p_{t_2}), D(p_{t_2} + 1 : p_{t_3}), \dots, D(p_{t_{k-1}} + 1 : p_{t_k})$ to estimate the ℓ_2 norm.
 ▷ Use Estimator and [Algorithm 1](#) with parameters $(\frac{1}{2}, \frac{\delta}{2})$ here.
 - 2: Let A_i be the merged sketch $D(p_{t_i} + 1 : p_{t_k})$.
 - 3: For each merged sketch A_i , find a superset H_i of the $\frac{\epsilon}{16}$ -heavy hitters.
 ▷ Use $(\frac{\epsilon}{16}, \frac{\delta}{2})$ – BPTree here. ([Theorem 11](#))
 - 4: For each element in H_1 , create a counter.
 ▷ Instantiate a 2 – SmoothCounter for each of the $\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$ elements reported in H_1 .
 - 5: Let $\hat{\ell}_2$ be the estimated ℓ_2 norm of A_1 .
 ▷ 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 .
 ▷ Output by 2 – SmoothCounter. ([Theorem 12](#))
 - 7: Output any element i with $\hat{f}_i \geq \frac{1}{4}\epsilon\hat{\ell}_2$.
-

404 **► Lemma 13.** *Any element i with frequency $f_i > \epsilon\ell_2$ is output by [Algorithm 2](#).*

406 **► Lemma 14.** *No element i with frequency $f_i < \frac{\epsilon}{12}\ell_2(W)$ is output by [Algorithm 2](#).*

407 **► Theorem 15.** *Given $\epsilon, \delta > 0$, there exists an algorithm in the sliding window model
 408 ([Algorithm 2](#)) that with probability at least $1 - \delta$ outputs all indices $i \in [m]$ for which
 409 $f_i \geq \epsilon\sqrt{F_2}$, and reports no indices $i \in [m]$ for which $f_i \leq \frac{\epsilon}{12}\sqrt{F_2}$. The algorithm has space
 410 complexity (in bits) $\mathcal{O}\left(\frac{1}{\epsilon^2} \log^2 n (\log^2 \log n + \log \frac{1}{\epsilon})\right)$.*

411 **4.3 Extension to ℓ_p norms for $0 < p < 2$**

412 To output a superset of the ℓ_p -heavy hitters rather than the ℓ_2 -heavy hitters, recall that an
 413 algorithm provides the (ϵ, k) -tail guarantee if the frequency estimate \hat{f}_i for each heavy hitter
 414 $i \in [m]$ satisfies $|\hat{f}_i - f_i| \leq \epsilon \cdot \|f_{tail(k)}\|_1$, where $f_{tail(k)}$ is the frequency vector f in which
 415 the k most frequent entries have been replaced by zero. Jowhari *et al.* [52] show the impact
 416 of ℓ_2 -heavy hitter algorithms that satisfy the tail guarantee.

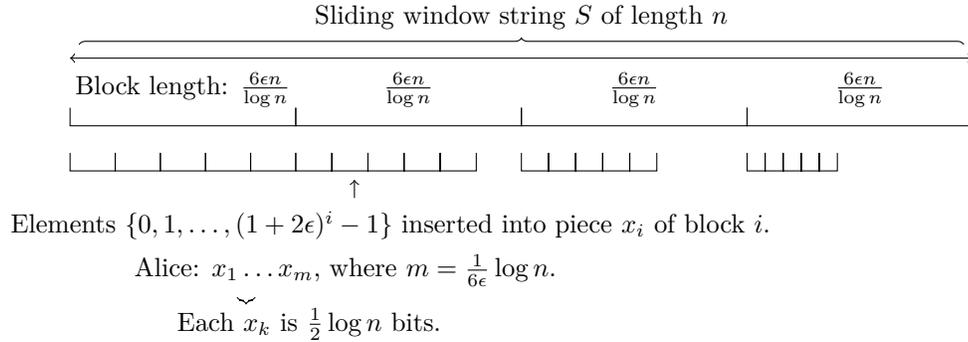
417 **► Lemma 16.** [52] *For any $p \in (0, 2]$, any algorithm that returns the $\epsilon^{p/2}$ -heavy hitters for*
 418 *ℓ_2 satisfying the tail guarantee also finds the ϵ -heavy hitters for ℓ_p .*

419 The correctness of [Theorem 3](#) immediately follows from [Lemma 16](#) and [Theorem 15](#).

420 **5 Lower Bounds**

421 **5.1 Distinct Elements**

422 To show a lower bound of $\Omega\left(\frac{1}{\epsilon} \log^2 n + \frac{1}{\epsilon^2} \log n\right)$ for the distinct elements problem, we show
 423 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
 424 of $\Omega\left(\frac{1}{\epsilon^2} \log n\right)$. We first obtain a lower bound of $\Omega\left(\frac{1}{\epsilon} \log^2 n\right)$ by a reduction from the
 IndexGreater problem.



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

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

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

437 **► Lemma 18.** *The one-way communication complexity of IndexGreater is $\Omega(nm)$ bits.*

438 ► **Theorem 19.** *Let $p > 0$ and $\epsilon, \delta \in (0, 1)$. Any one-pass streaming algorithm that returns*
 439 *a $(1 + \epsilon)$ -approximation to the number of distinct elements in the sliding window model with*
 440 *probability $\frac{2}{3}$ requires $\Omega\left(\frac{1}{\epsilon} \log^2 n\right)$ space.*

441 To obtain a lower bound of $\Omega\left(\frac{1}{\epsilon^2} \log n\right)$, we give a reduction from the GapHamming problem.

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

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

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

448 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
 449 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
 450 bound of $\Omega\left(\frac{1}{\epsilon^2} \log n\right)$ by embedding $\Omega(\log n)$ instances of GapHamming into the stream.

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

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

455 5.2 ℓ_p -Heavy Hitters

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

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

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

470 References

- 471 1 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating
 472 the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. A preliminary version
 473 appeared in the Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory
 474 of Computing (STOC), 1996.
- 475 2 Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding
 476 windows. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium*
 477 *on Principles of Database Systems*, pages 286–296, 2004.

- 478 **3** Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O’Callaghan. Maintaining
479 variance and k-medians over data stream windows. In *Proceedings of the Twenty-Second*
480 *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*,
481 pages 234–243, 2003.
- 482 **4** Nagender Bandi, Divyakant Agrawal, and Amr El Abbadi. Fast algorithms for heavy
483 distinct hitters using associative memories. In *27th IEEE International Conference on*
484 *Distributed Computing Systems (ICDCS)*, page 6, 2007.
- 485 **5** Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics
486 approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–
487 732, 2004. A preliminary version appeared in the Proceedings of the 43rd Symposium on
488 Foundations of Computer Science (FOCS), 2002.
- 489 **6** Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting
490 distinct elements in a data stream. In *Randomization and Approximation Techniques, 6th*
491 *International Workshop, RANDOM, Proceedings*, pages 1–10, 2002.
- 492 **7** Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Heavy hitters in streams
493 and sliding windows. In *35th Annual IEEE International Conference on Computer Com-*
494 *munications, INFOCOM*, pages 1–9, 2016.
- 495 **8** Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J. Strauss. Space-optimal heavy
496 hitters with strong error bounds. *ACM Trans. Database Syst.*, 35(4):26:1–26:28, 2010. A
497 preliminary version appeared in the Proceedings of the Twenty-Eighth ACM SIGMOD-
498 SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009.
- 499 **9** Jaroslaw Blasiok. Optimal streaming and tracking distinct elements with high proba-
500 bility. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete*
501 *Algorithms, SODA*, pages 2432–2448, 2018.
- 502 **10** Jaroslaw Blasiok, Jian Ding, and Jelani Nelson. Continuous monitoring of ℓ_p norms in data
503 streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms*
504 *and Techniques, APPROX/RANDOM*, pages 32:1–32:13, 2017.
- 505 **11** Vladimir Braverman. Sliding window algorithms, 2016.
- 506 **12** Vladimir Braverman, Stephen R. Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang,
507 and David P. Woodruff. Bptree: An ℓ_2 heavy hitters algorithm using constant memory.
508 In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of*
509 *Database Systems, PODS*, pages 361–376, 2017.
- 510 **13** Vladimir Braverman, Stephen R. Chestnut, Nikita Ivkin, and David P. Woodruff. Beating
511 counts sketch for heavy hitters in insertion streams. In *Proceedings of the 48th Annual ACM*
512 *SIGACT Symposium on Theory of Computing, STOC*, pages 740–753, 2016.
- 513 **14** Vladimir Braverman, Petros Drineas, Jalaj Upadhyay, and Samson Zhou. Numerical linear
514 algebra in the sliding window model. *CoRR*, abs/1805.03765, 2018. URL: <http://arxiv.org/abs/1805.03765>, [arXiv:1805.03765](https://arxiv.org/abs/1805.03765).
- 515 <http://arxiv.org/abs/1805.03765>, [arXiv:1805.03765](https://arxiv.org/abs/1805.03765).
- 516 **15** Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky. How to catch ℓ_2 -heavy-hitters on
517 sliding windows. *Theor. Comput. Sci.*, 554:82–94, 2014. A preliminary version appeared
518 in the Proceedings of Computing and Combinatorics, 19th International Conference (CO-
519 COON), 2013.
- 520 **16** Vladimir Braverman, Elena Grigorescu, Harry Lang, David P. Woodruff, and Samson
521 Zhou. Nearly optimal distinct elements and heavy hitters on sliding windows. *CoRR*,
522 abs/1805.00212, 2018. URL: <http://arxiv.org/abs/1805.00212>, [arXiv:1805.00212](https://arxiv.org/abs/1805.00212).
- 523 **17** Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering on
524 sliding windows in polylogarithmic space. In *35th IARCS Annual Conference on Foundation*
525 *of Software Technology and Theoretical Computer Science, FSTTCS*, pages 350–364, 2015.

- 526 **18** Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering
527 problems on sliding windows. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM*
528 *Symposium on Discrete Algorithms, SODA*, pages 1374–1390, 2016.
- 529 **19** Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In
530 *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS) Proceedings*,
531 pages 283–293, 2007.
- 532 **20** Vladimir Braverman, Rafail Ostrovsky, and Alan Roytman. Zero-one laws for sliding win-
533 dows and universal sketches. In *Approximation, Randomization, and Combinatorial Opti-*
534 *mization. Algorithms and Techniques, APPROX/RANDOM*, pages 573–590, 2015.
- 535 **21** Yousra Chabchoub, Christine Fricker, and Hanene Mohamed. Analysis of a bloom filter
536 algorithm via the supermarket model. In *21st International Teletraffic Congress, ITC*,
537 pages 1–8, 2009.
- 538 **22** Amit Chakrabarti, Graham Cormode, and Andrew McGregor. A near-optimal algorithm
539 for estimating the entropy of a stream. *ACM Trans. Algorithms*, 6(3):51:1–51:21, 2010.
- 540 **23** Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the
541 multi-party communication complexity of set disjointness. In *18th Annual IEEE Conference*
542 *on Computational Complexity*, pages 107–117, 2003.
- 543 **24** Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication com-
544 plexity of gap-hamming-distance. *SIAM J. Comput.*, 41(5):1299–1317, 2012. A preliminary
545 version appeared in the Proceedings of the 43rd ACM Symposium on Theory of Computing,
546 STOC 2011.
- 547 **25** Timothy M. Chan and Bashir S. Sadjad. Geometric optimization problems over sliding
548 windows. *Int. J. Comput. Geometry Appl.*, 16(2-3):145–158, 2006. A preliminary version
549 appeared in the Proceedings of Algorithms and Computation, 15th International Symposi-
550 um (ISAAC), 2004.
- 551 **26** Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data
552 streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004. A preliminary version appeared in the
553 Proceedings of the Automata, Languages and Programming, 29th International Colloquium
554 (ICALP), 2002.
- 555 **27** Jiecao Chen, Huy L. Nguyen, and Qin Zhang. Submodular maximization over sliding
556 windows. *CoRR*, abs/1611.00129, 2016.
- 557 **28** Yun Chi, Haixun Wang, Philip S. Yu, and Richard R. Muntz. Catch the moment: main-
558 taining closed frequent itemsets over a data stream sliding window. *Knowl. Inf. Syst.*,
559 10(3):265–294, 2006. A preliminary version appeared in the Proceedings of the 4th IEEE
560 International Conference on Data Mining (ICDM), 2004.
- 561 **29** Graham Cormode. The continuous distributed monitoring model. *SIGMOD Record*,
562 42(1):5–14, 2013.
- 563 **30** Graham Cormode and Minos N. Garofalakis. Streaming in a connected world: querying
564 and tracking distributed data streams. In *EDBT*, page 745, 2008.
- 565 **31** Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. Finding hierar-
566 chical heavy hitters in streaming data. *TKDD*, 1(4):2:1–2:48, 2008.
- 567 **32** Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-
568 min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. A preliminary version
569 appeared in the Proceedings of the 6th Latin American Symposium (LATIN), 2004.
- 570 **33** Graham Cormode and S. Muthukrishnan. What’s new: finding significant differences in
571 network data streams. *IEEE/ACM Transactions on Networking*, 13(6):1219–1232, 2005.
- 572 **34** Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic graphs in the sliding-
573 window model. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Proceedings*,
574 pages 337–348, 2013.

- 575 **35** Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream
576 statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002. A preliminary
577 version appeared in the Proceedings of the Thirteenth Annual ACM-SIAM Symposium on
578 Discrete Algorithms (SODA), 2002.
- 579 **36** Mayur Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream
580 windows. In *Algorithms - ESA 2002, 10th Annual European Symposium, Proceedings*,
581 pages 323–334, 2002.
- 582 **37** Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of
583 internet packet streams with limited space. In *Algorithms - ESA, 10th Annual European
584 Symposium, Proceedings*, pages 348–360, 2002.
- 585 **38** Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended
586 abstract). In *Algorithms - ESA, 11th Annual European Symposium, Proceedings*, pages
587 605–617, 2003.
- 588 **39** Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam.
589 Submodular optimization over sliding windows. In *Proceedings of the 26th International
590 Conference on World Wide Web, WWW*, pages 421–430, 2017.
- 591 **40** Cristian Estan and George Varghese. New directions in traffic measurement and accounting:
592 Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313,
593 2003.
- 594 **41** Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D.
595 Ullman. Computing iceberg queries efficiently. In *VLDB’98, Proceedings of 24rd Interna-
596 tional Conference on Very Large Data Bases*, pages 299–310, 1998.
- 597 **42** Joan Feigenbaum, Sampath Kannan, and Jian Zhang. Computing diameter in the stream-
598 ing and sliding-window models. *Algorithmica*, 41(1):25–41, 2005.
- 599 **43** Philippe Flajolet, Eric Fusy, Olivier Gandouet, and Frederic Meunier. Hyperloglog: the
600 analysis of a near-optimal cardinality estimation algorithm. In *AofA: Analysis of Algo-
601 rithms*, page 137–156, 2007.
- 602 **44** Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *24th Annual Symposium
603 on Foundations of Computer Science*, pages 76–82, 1983.
- 604 **45** Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of
605 data streams. In *SPAA*, pages 281–291, 2001.
- 606 **46** Phillip B. Gibbons and Srikanta Tirthapura. Distributed streams algorithms for sliding
607 windows. In *SPAA*, pages 63–72, 2002.
- 608 **47** Parikshit Gopalan and Jaikumar Radhakrishnan. Finding duplicates in a data stream.
609 In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms,
610 SODA*, pages 402–411, 2009.
- 611 **48** Nicholas J. A. Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and streaming entropy
612 via approximation theory. In *49th Annual IEEE Symposium on Foundations of Computer
613 Science, FOCS*, pages 489–498, 2008.
- 614 **49** Regant Y. S. Hung and Hing-Fung Ting. Finding heavy hitters over the sliding window of a
615 weighted data stream. In *LATIN: Theoretical Informatics, 8th Latin American Symposium,
616 Proceedings*, pages 699–710, 2008.
- 617 **50** Piotr Indyk and David P. Woodruff. Tight lower bounds for the distinct elements problem.
618 In *44th Symposium on Foundations of Computer Science (FOCS)*, pages 283–288, 2003.
- 619 **51** Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of
620 data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing
621 (STOC)*, pages 202–208, 2005.
- 622 **52** Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding
623 duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-
624 SIGACT-SIGART Symposium on Principles of Database Systems*, pages 49–58, 2011.

- 625 **53** Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the
 626 distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-*
 627 *SIGART Symposium on Principles of Database Systems, PODS*, pages 41–52, 2010.
- 628 **54** Abhishek Kumar and Jun (Jim) Xu. Sketch guided sampling - using on-line estimates
 629 of flow size for adaptive data collection. In *INFOCOM 2006. 25th IEEE International*
 630 *Conference on Computer Communications, Joint Conference of the IEEE Computer and*
 631 *Communications Societies*, 2006.
- 632 **55** Kasper Green Larsen, Jelani Nelson, Huy L. Nguyen, and Mikkel Thorup. Heavy hitters
 633 via cluster-preserving clustering. In *IEEE 57th Annual Symposium on Foundations of*
 634 *Computer Science, FOCS*, pages 61–70, 2016.
- 635 **56** Lap-Kei Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding
 636 frequent items over sliding windows. In *Proceedings of the Twenty-Fifth ACM SIGACT-*
 637 *SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 290–297, 2006.
- 638 **57** Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data
 639 streams. *PVLDB*, 5(12):1699, 2012. A preliminary version appeared in the Proceedings of
 640 the 28th International Conference on Very Large Data Bases (VLDB), 2002.
- 641 **58** Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures
 642 and asymmetric communication complexity. In *Proceedings of the Twenty-Seventh Annual*
 643 *ACM Symposium on Theory of Computing*, pages 103–111, 1995.
- 644 **59** Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*,
 645 2(2):143–152, 1982.
- 646 **60** Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error ℓ_p -sampling with ap-
 647 plications. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete*
 648 *Algorithms, SODA*, pages 1143–1160, 2010.
- 649 **61** Miles Osborne, Sean Moran, Richard McCreddie, Alexander Von Lunen, Martin Sykora,
 650 Elizabeth Cano, Neil Ireson, Craig MacDonald, Iadh Ounis, Yulan He, Tom Jackson, Fabio
 651 Ciravegna, and Ann O’Brien. Real-time detection, tracking and monitoring of automati-
 652 cally discovered events in social media. In *Proceedings of the 52nd Annual Meeting of the*
 653 *Association for Computational Linguistics*, 2014.
- 654 **62** Subhabrata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks.
 655 *IEEE/ACM Trans. Netw.*, 12(2):219–232, 2004.
- 656 **63** Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications
 657 to linear probing and second moment estimation. *SIAM J. Comput.*, 41(2):293–331, 2012.

658 **A Full Version**

659 We show that the structure of the F_2 algorithm only requires the correctness of a specific
 660 $\mathcal{O}(\text{polylog } n)$ algorithms in the data structure. Given a vector $v \in \mathbb{R}^m$, let $F_2(v) = v_1^2 +$
 661 $v_2^2 + \dots + v_m^2$. Recall that the histogram creates a new algorithm each time a new element
 662 arrives in the data stream. Instead of requiring all n algorithms perform correctly, we show
 663 that it suffices to only require the correctness of a specific $\mathcal{O}(\text{polylog } n)$ of these algorithms.

664 Let F be the value of F_2 on the most recent n elements. For the purpose of analysis,
 665 we say that an algorithm is *important* if it is still maintained within the histogram when its
 666 output is at least $\frac{F}{2 \log n}$ and the algorithm never outputs anything greater than $8F \log^3 n$.

667 We first show that with high probability, all algorithms correctly maintain a $\log n$ -
 668 approximation of the value of F_2 for the corresponding frequency vector. Conditioned on
 669 each algorithm correctly maintaining a $\log n$ -approximation, we then show that $\mathcal{O}(\log^6 n)$
 670 algorithms are important. Observe that an algorithm that reports a 2-approximation to
 671 F is important. Furthermore, we show that any algorithm that is not important cannot
 672 influence the output of the histogram, conditioned on each algorithm correctly maintaining

673 a $\log n$ -approximation. Thus, it suffices to require correctness of strong tracking on these
 674 $\mathcal{O}(\log^6 n)$ important algorithms and we apply a union bound over the $\mathcal{O}(\log^6 n)$ important
 675 algorithms to ensure correctness. Hence for each algorithm, we require the probability of
 676 failure to be at most $\mathcal{O}\left(\frac{\delta}{\log^6 n}\right)$ for the histogram to succeed with probability at least $1 - \delta$.

677 ► **Fact 24.** Given m -dimensional vectors x, y, z with non-negative entries, then $F_2(x + y +$
 678 $z) - F_2(x + y) \geq F_2(x + z) - F_2(x)$.

679 Although the number of algorithms in the histogram at any given moment is at most
 680 $\mathcal{O}(\log n)$, it may be possible that many algorithms have output at least $\frac{F}{2 \log n}$ only to be
 681 deleted at some point in time. We now show that in a window of size $2n$, there are only
 682 $\mathcal{O}(\log^6 n)$ important algorithms.

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

686 **Proof.** Let $s_1 < s_2 < \dots < s_i$ be the starting points of important algorithms A_1, A_2, \dots, A_i ,
 687 respectively, that begin within the most recent $2n$ elements. For each $1 < j < i$, let t_j be
 688 the first time that algorithm A_j outputs a value that is at least $\frac{F}{2 \log n}$. The idea is to show
 689 at the end of the stream, the elements between s_j and s_{j+1} are responsible for an increase in
 690 F_2 by at least $\frac{cF}{2 \log^2 n}$ for all j . Since an algorithm is important if it never outputs anything
 691 greater than $8F \log^3 n$, then the F_2 value of the substream represented by the algorithm is
 692 at most $8F \log^4 n$, and it follows that $i = \mathcal{O}(\log^6 n)$.

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

699 In other words, the elements from time s_{j-1} to s_j are responsible for a difference of at
 700 least $\frac{cF}{2 \log^2 n}$ between the F_2 values of the substreams represented by A_{j-1} and A_j at time
 701 t_j . Thus by [Fact 24](#), the difference between the F_2 values of the substreams represented by
 702 A_{j-1} and A_j at any time $t \geq t_j$ is at least $\frac{cF}{2 \log^2 n}$. By induction, the value of F_2 on the
 703 substream from s_1 to t_j is at least $\frac{(j-1)cF}{2 \log^2 n}$. Recall that the F_2 of the substream represented
 704 by any important algorithm is at most $8F \log^4 n$. Therefore, $i = \mathcal{O}(\log^6 n)$ and so at most
 705 $\mathcal{O}(\log^6 n)$ algorithms are important. ◀

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

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

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

711 **Proof.** Note that any algorithm A that outputs a value that is at least $8F \log^3 n$ must
 712 represent a substream whose F_2 value is at least $8F \log^2 n$ at the end of the stream, assuming
 713 a $\log n$ -approximation of all algorithms. Observe that the substream represented by an
 714 important algorithm B that provides a 2-approximation has F_2 value at most $2F$ at the
 715 end of the stream. By [Corollary 27](#), the ratio between the F_2 values of the substreams

716 represented by A and B must be at least $4 \log^2 n$ at every previous point in time. Thus, if
 717 A and B always correctly maintain a $\log n$ -approximation of the corresponding substreams,
 718 the ratio of the outputs between A and B is at least 4, so A will never cause the histogram
 719 data structure to delete B . ◀

720 Hence, it remains to show that with high probability, all algorithms correctly maintain a
 721 $\log n$ -approximation of the value of F_2 for the corresponding frequency vector. Recall that
 722 Estimator from Theorem 10 uses an AMS sketch so that the resulting frequency of each
 723 element f_i is multiplied by a Rademacher random variable R_i .

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

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

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

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

734 ▶ **Lemma 31**. *Let $z_1, z_2, \dots, z_m \in \{-1, 1\}$ be a set of $\log n$ -wise independent random vari-*
 735 *ables and $f \in \mathbb{R}^m$ be a given vector. Then $\Pr[\sum_{i=1}^m z_i f_i \geq (\log n) \|f\|_2] \leq \frac{1}{\log n \sqrt{\log n}}$.*

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

$$737 \Pr \left[\left| \sum_{i=1}^m z_i f_i \right| \geq (\log n) \|f\|_2 \right] = \Pr \left[\left| \sum_{i=1}^m z_i f_i \right|^p \geq (\log n)^p \|f\|_2^p \right].$$

738 By Markov's inequality, $\Pr[\sum_{i=1}^m z_i f_i^p \geq (\log n)^p \|f\|_2^p] \leq \frac{\mathbf{E}[(\sum_{i=1}^m z_i f_i)^p]}{(\log n)^p \|f\|_2^p}$. By Corol-
 739 lary 30, it follows that $\frac{\mathbf{E}[(\sum_{i=1}^m z_i f_i)^p]}{(\log n)^p \|f\|_2^p} \leq \frac{\sqrt{p}^p \|f\|_2^p}{(\log n)^p \|f\|_2^p} = \frac{1}{\log n \sqrt{\log n}}$. ◀

740 Therefore, with high probability, all algorithms correctly maintain a $\log n$ -approximation of
 741 the value of F_2 for the corresponding frequency vector.

742 B Supplementary Proofs

743 **Proof of Lemma 13:** Since the ℓ_2 norm is a smooth function, and so there exists a
 744 smooth-histogram which is an $(\frac{1}{2}, \frac{\delta}{2})$ -estimation of the ℓ_2 norm of the sliding window by
 745 Theorem 6. Thus, $\frac{1}{2} \hat{\ell}_2(A_1) \leq \ell_2(W) \leq \frac{3}{2} \hat{\ell}_2(A_1)$. With probability $1 - \frac{\delta}{2}$, any element i
 746 whose frequency satisfies $f_i(W) \geq \epsilon \ell_2(W)$ must have $f_i(W) \geq \epsilon \ell_2(W) \geq \frac{1}{2} \epsilon \hat{\ell}_2(A_1)$ and is
 747 reported by $(\frac{\epsilon}{16}, \frac{\delta}{2})$ -BPTree in Step 3.

748 Since BPTree is instantiated along with A_1 , the sliding window may begin either before
 749 or after BPTree reports each heavy hitter. If the sliding window begins after the heavy hitter
 750 is reported, then all $f_i(W)$ instances are counted by SmoothCounter. Thus, the count of f_i
 751 estimated by SmoothCounter is at least $f_i(W) \geq \epsilon \ell_2(W) \geq \frac{1}{2} \epsilon \hat{\ell}_2(A_1)$, and so Step 7 will
 752 output i .

753 On the other hand, the sliding window may begin before the heavy hitter is reported.
 754 Recall that the BPTree algorithm identifies and reports an element when it becomes an
 755 $\frac{\epsilon}{16}$ -heavy hitter with respect to the estimate of ℓ_2 . Hence, there are at most $2 \cdot \frac{\epsilon}{16} \hat{\ell}_2(A_1) \leq$
 756 $\frac{1}{8} \epsilon \hat{\ell}_2(A_1)$ instances of an element appearing in the active window before it is reported by
 757 BPTree. Since $f_i(W) \geq \epsilon \ell_2(W) \geq \frac{1}{2} \epsilon \hat{\ell}_2(A_1)$, any element i whose frequency satisfies $f_i(W) \geq$
 758 $\epsilon \ell_2(W)$ must have $f_i(W) \geq \frac{\epsilon}{2} \hat{\ell}_2(A_1)$ and therefore must have at least $(\frac{1}{2} - \frac{1}{8}) \epsilon \hat{\ell}_2(A_1) \geq$
 759 $\frac{1}{4} \epsilon \hat{\ell}_2(A_1)$ instances appearing in the stream after it is reported by BPTree. Thus, the count
 760 of f_i estimated by SmoothCounter is at least $\frac{1}{4} \epsilon \hat{\ell}_2(A_1)$, and so Step 7 will output i . \square

761 **Proof of Lemma 14:** If i is output by Step 7, then $\hat{f}_i \geq \frac{1}{4} \epsilon \hat{\ell}_2(A_1)$. By the properties of
 762 SmoothCounter and Estimator, $f_i(W) \geq \frac{\hat{f}_i}{2} \geq \frac{1}{8} \epsilon \hat{\ell}_2(A_1) \geq \frac{1}{12} \ell_2(W)$, where the last inequality
 763 comes from the fact that $\ell_2(W) \leq \frac{3}{2} \hat{\ell}_2(A_1)$. \square

764 **Proof of Theorem 15:** By Lemma 13 and Lemma 14, Algorithm 2 outputs all elements
 765 with frequency at least $\epsilon \ell_2(W)$ and no elements with frequency less than $\frac{\epsilon}{12} \ell_2(W)$. We
 766 now proceed to analyze the space complexity of the algorithm. Step 1 uses Algorithm 1
 767 in conjunction with the Estimator routine to maintain a $\frac{1}{2}$ -approximation to the ℓ_2 -norm
 768 of the sliding window. By requiring the probability of failure to be $\mathcal{O}\left(\frac{\delta}{\text{polylog} n}\right)$ in Theo-
 769 rem 10 and observing that $\beta = \mathcal{O}(1)$ in Theorem 6 suffices for a $\frac{1}{2}$ -approximation, it follows
 770 that Step 1 uses $\mathcal{O}(\log n(\log n + \log m \log^2 \log m))$ bits of space. Since Step 3 runs an in-
 771 stance of BPTree for each of the at most $\mathcal{O}(\log n)$ buckets, then by Theorem 11, it uses
 772 $\mathcal{O}\left(\frac{1}{\epsilon^2} \left(\log \frac{1}{\delta \epsilon}\right) \log n(\log n + \log m)\right)$ bits of space.

773 Notice that BPTree returns a list of $\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$ elements, by Theorem 11. By running
 774 SmoothCounter for each of these, Step 7 provides a 2-approximation to the frequency of
 775 each element after being returned by BPTree. By Theorem 12, Step 7 has space complex-
 776 ity (in bits) $\mathcal{O}\left(\frac{1}{\epsilon^2} (\log n + \log m) \log n\right)$. Assuming $\log m = \mathcal{O}(\log n)$, the algorithm uses
 777 $\mathcal{O}\left(\frac{1}{\epsilon^2} \log^2 n (\log^2 \log n + \log \frac{1}{\epsilon})\right)$ bits of space. \square

778 **Proof of Theorem 3:** By Theorem 11, BPTree satisfies the tail guarantee. Therefore
 779 by Lemma 16, it suffices to analyze the space complexity of finding the $\epsilon^{p/2}$ -heavy hitters
 780 for ℓ_2 . By Theorem 15, there exists an algorithm that uses $\mathcal{O}\left(\frac{1}{\epsilon^2} \log^2 n (\log^2 \log n + \log \frac{1}{\epsilon})\right)$
 781 bits of space to find the ϵ -heavy hitters for ℓ_2 . Hence, there exists an algorithm that uses
 782 $\mathcal{O}\left(\frac{1}{\epsilon^p} \log^2 n (\log^2 \log n + \log \frac{1}{\epsilon})\right)$ bits of space to find the ϵ -heavy hitters for ℓ_p , where $0 <$
 783 $p \leq 2$. \square

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

788 Observe that Alice can form the string $T = x_1 x_2 \dots x_m$ of length mn , where each x_k
 789 has n bits of S . Alice can then use the IndexGreater protocol and communicate to Bob a
 790 message that will solve the IndexGreater problem. Let $j = \lfloor \frac{i}{n} \rfloor$ so that the symbol $S[i]$
 791 is a bit inside x_{j+1} . Then Bob constructs the string w by first concatenating the bits
 792 $S[jn+1], S[jn+2], \dots, S[i-1]$, which he is given from the AugmentedIndex problem. Bob
 793 then appends a zero to w , and pads w with ones at the end, until w reaches n bits:

$$794 \quad w = S[jn+1] \circ S[jn+2] \circ \dots \circ S[i-1] \circ 0 \circ \underbrace{1 \circ 1 \circ \dots \circ 1}_{\text{until } w \text{ has } n \text{ bits}} .$$

795 Bob takes the message from Alice and runs the `IndexGreater` protocol to determine whether
 796 $x_j > w$. Observe that by construction $x_j > w$ if and only if $S[i] = 1$. Thus, if the `Index-`
 797 `Greater` protocol succeeds, then Bob will have solved the `AugmentedIndex` problem, which
 798 requires communication complexity $\Omega(nm)$ bits. Hence, the communication complexity of
 799 `IndexGreater` follows. \square

800 **Proof of Theorem 19:** We reduce a one-way communication protocol for `IndexGreater`
 801 to finding a $(1 + \epsilon)$ -approximation to the number of distinct elements in the sliding window
 802 model.

803 Let n be the length of the sliding window and suppose Alice receives a string $S =$
 804 $x_1 x_2 \dots x_\ell \in \{0, 1\}^\ell$, where $\ell = \frac{1}{6\epsilon} \log n$ and each x_k has $\frac{1}{2} \log n$ bits. Bob receives an index
 805 $i \in [\ell]$ and an integer $j \in [\sqrt{n}]$. Suppose Alice partitions the sliding window into ℓ blocks,
 806 each of length $\frac{n}{\ell} = \frac{6\epsilon n}{\log n}$. For each $1 \leq k \leq \frac{1}{6\epsilon} \log n$, she further splits block $(\ell - k + 1)$ into
 807 \sqrt{n} pieces of length $(1 + 2\epsilon)^k$, before padding the remainder of block $(\ell - k + 1)$ with zeros.
 808 Moreover, for piece x_k of block $(\ell - k + 1)$, Alice inserts the elements $\{0, 1, \dots, (1 + 2\epsilon)^k - 1\}$,
 809 before padding the remainder of block $(\ell - k + 1)$ with zeros. Hence, the sliding window
 810 contains all zeros, with the exception of the elements $\{0, 1, \dots, (1 + 2\epsilon)^k - 1\}$ appearing in
 811 piece x_k of block $(\ell - k + 1)$ for all $1 \leq k \leq \ell = \frac{1}{6\epsilon} \log n$. Note that $(1 + 2\epsilon)^k \leq \sqrt[3]{n}$ and
 812 $x_k \leq \sqrt{n}$ for all k , so all the elements fit within each block, which has length $\frac{6\epsilon n}{\log n}$. Finally,
 813 Alice runs the $(1 + \epsilon)$ -approximation distinct elements sliding window algorithm and passes
 814 the state to Bob. See Figure 2 for an example of Alice's construction.

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

825 **Proof of Theorem 22:** We reduce a one-way communication protocol for the `GapHam-`
 826 `ming` problem to finding a $(1 + \epsilon)$ -approximation to the number of distinct elements in
 827 the sliding window model. For each $\frac{\log \frac{1}{\epsilon}}{2} \leq i \leq \frac{\log n - 1}{2}$, let $j = 2i$ and x_j and y_j each
 828 have length 2^j and (x_j, y_j) be drawn from a distribution such that with probability $\frac{1}{2}$,
 829 $\text{HAM}(x_j, y_j) = (1 + 4\epsilon)2^{j-1}$ and otherwise (with probability $\frac{1}{2}$), $\text{HAM}(x_j, y_j) = (1 - 4\epsilon)2^{j-1}$.
 830 Then Alice is given $\{x_j\}$ while Bob is given $\{y_j\}$ and needs to output $\text{HAM}(x_j, y_j)$. For
 831 $\epsilon \leq \frac{1}{\sqrt{n}}$, this is precisely the hard distribution in the communication complexity of `GapHam-`
 832 `ming` given by [24].

833 Let $a = \frac{\log \frac{1}{\epsilon}}{2}$ and $b = \frac{\log n - 1}{2}$. Let $w_{2k} = x_{2k}$ and let w_{2k-1} be a string of length 2^{2k-1} ,
 834 all consisting of zeros. Suppose Alice forms the concatenated string $S = w_{2b} \circ w_{2b-1} \circ \dots \circ$
 835 $w_{2a+1} \circ w_{2a}$. Note that $\sum_{k=2a}^{2b} 2^k \leq n$, so S has length less than n . Alice then forms a data
 836 stream by the following process. She initializes $k = 1$ and continuously increments k until
 837 $k = n$. At each step, if $S[k] = 0$ or k is longer than the length of S , Alice inserts a 0 into the
 838 data stream. Otherwise, if $S[k] = 1$, then Alice inserts k into the data stream. Meanwhile,
 839 Alice runs the $(1 + \epsilon)$ -approximation distinct elements sliding window algorithm and passes
 840 the state of the algorithm to Bob.

841 To find $\text{HAM}(x_{2i}, y_{2i})$, Bob first expires $(\sum_{k=2^{i+1}}^{2^b} 2^k) - 2^{2i}$ elements by inserting zeros
 842 into the data stream. Similar to Alice, Bob initializes $k = 1$ and continuously increments k
 843 until $k = 2^{2i}$. At each step, if $y_{2i}[k] = 0$ (that is, the k^{th} bit of y_{2i} is zero), then Bob inserts a 0
 844 into the data stream. Otherwise, if $y_{2i}[k] = 1$, then Bob inserts k into the data stream. At the
 845 end of this procedure, the sliding window contains all zeros, nonzero values corresponding to
 846 the nonzero indices of the string $x_{2i} \circ w_{2^{i-1}} \circ x_{2^{i-2}} \circ \dots \circ x_{2^{a+2}} \circ w_{2^{a+1}} \circ x_{2^a}$, and nonzero values
 847 corresponding to the nonzero indices of y_{2i} . Observe that each w_j solely consists of zeros
 848 and $\sum_{k=a}^{i-1} 2^{2k} < 2^{2i-1}$. Therefore, $\text{HAM}(x_{2i}, y_{2i})$ is at least $(1 - 4\epsilon)2^{2i-1}$ while the number
 849 of distinct elements in the sliding window is at most $(1 + 4\epsilon)2^{2i}$ while the number of distinct
 850 elements in the suffix $x_{2^{i-2}} \circ x_{2^{i-3}} \dots$ is at most $(1 + \epsilon)2^{2i-2}$. Thus, a $(1 + \epsilon)$ -approximation
 851 to the number of distinct elements differentiates between $\text{HAM}(x_{2i}, y_{2i}) = (1 + 4\epsilon)2^{2i-1}$ and
 852 $\text{HAM}(x_{2i}, y_{2i}) = (1 - 4\epsilon)2^{2i-1}$.

853 Since the sliding window algorithm succeeds with probability $\frac{2}{3}$, then the GapHamming
 854 distance problem succeeds with probability $\frac{2}{3}$ across the $\Omega(\log n)$ values of i . Therefore, any
 855 $(1 + \epsilon)$ -approximation sliding window algorithm for the number of distinct elements that
 856 succeeds with probability $\frac{2}{3}$ requires $\Omega(\frac{1}{\epsilon^2} \log n)$ space for $\epsilon \leq \frac{1}{\sqrt{n}}$. \square

857 **Proof of Theorem 4:** We reduce a one-way communication protocol for the AugmentedIn-
 858 dex problem to finding the ℓ_p heavy hitters in the sliding window model. Let $a = \frac{1}{2^p \epsilon^p} \log \sqrt{n}$
 859 and $b = \log n$. Suppose Alice receives $S = [2^a]^b$ and Bob receives $i \in [b]$ and $S[1, i - 1]$. Ob-
 860 serve that each $S[i]$ is $\frac{1}{2^p \epsilon^p} \log \sqrt{n}$ bits and so $S[i]$ can be rewritten as $S[i] = w_1 \circ w_2 \circ \dots \circ w_t$,
 861 where each $t = \frac{1}{2^p \epsilon^p}$ and so each w_i is $\log \sqrt{n}$ bits.

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

868 Bob expires all elements of the stream before a_i , runs the sliding window heavy hitters
 869 algorithm on the resulting vector, and then computes the heavy hitters. We claim that
 870 the algorithm will output t heavy hitters and by concatenating the last $\log \sqrt{n}$ bits of the
 871 binary representation of each of these heavy hitters, Bob will recover exactly $S[i]$. Ob-
 872 serve that the ℓ_p norm of the underlying vector represented by $a_i \circ a_{i+1} \circ \dots \circ a_b$ is exactly
 873 $(\frac{1}{2^p \epsilon^p} (1^p + 2^p + 4^p + \dots + 2^{p(b-i)}))^{1/p} \leq \frac{1}{2\epsilon} 2^{b-i+1} = \frac{1}{\epsilon} 2^{b-i}$. Let u_1, u_2, \dots, u_t be the coordi-
 874 nates of the frequency vector incremented by Alice as part of a_i . Each coordinate u_j has
 875 frequency $2^{b-i} \geq \epsilon (\frac{1}{\epsilon} 2^{b-i})$, so that u_j is an ℓ_p -heavy hitter.

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