An Optimal Algorithm for ℓ_1-Heavy Hitters in Insertion Streams and Related Problems

Arnab Bhattacharyya
Indian Institute of Science, Bangalore
arnabb@csa.iisc.ernet.in

Palash Dey
Indian Institute of Science, Bangalore
palash@csa.iisc.ernet.in

David P. Woodruff
IBM Research, Almaden
dpwoodru@us.ibm.com

ABSTRACT

We give the first optimal bounds for returning the ℓ_1-heavy hitters in a data stream of insertions, together with their approximate frequencies, closing a long line of work on this problem. For a stream of m items in $\{1, 2, \ldots, n\}$ and parameters $0 < \varepsilon < \varphi \leqslant 1$, let f_i denote the frequency of item i, i.e., the number of times item i occurs in the stream. With arbitrarily large constant probability, our algorithm returns all items i for which $f_i \geqslant \varphi m$, returns no items j for which $f_j \leqslant (\varphi - \varepsilon)m$, and returns approximations $\tilde{f}_i$ with $|\tilde{f}_i - f_i| \leqslant \varepsilon m$ for each item i that it returns. Our algorithm uses $O(\varepsilon^{-1} \log \varphi^{-1} + \varphi^{-1} \log n + \log \log m)$ bits of space, processes each stream update in $O(1)$ worst-case time, and can report its output in time linear in the output size. We also prove a lower bound, which implies that our algorithm is optimal up to a constant factor in its space complexity. A modification of our algorithm can be used to estimate the maximum frequency up to an additive εm error in the above amount of space, resolving Question 3 in the IITK 2006 Workshop on Algorithms for Data Streams for the case of ℓ_1-heavy hitters. We also introduce several variants of the heavy hitters and maximum frequency problems, inspired by rank aggregation and voting schemes, and show how our techniques can be applied in such settings. Unlike the traditional heavy hitters problem, some of these variants look at comparisons between items rather than numerical values to determine the frequency of an item.

Keywords

Heavy hitters, frequent items, data streams, algorithms

1. INTRODUCTION

The data stream model has emerged as a standard model for processing massive data sets. Because of the sheer size of the data, traditional algorithms are no longer feasible, e.g., it may be hard or impossible to store the entire input, and algorithms need to run in linear or even sublinear time.

PODS'16, June 26-July 01, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4191-2/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2902251.2902284

Such algorithms typically need to be both randomized and approximate. Moreover, the data may not physically reside on any device, e.g., if it is internet traffic, and so if the data is not stored by the algorithm, it may be impossible to recover it. Hence, many algorithms must work given only a single pass over the data. Applications of data streams include data warehousing [6, 28, 31, 34], network measurements [3, 20, 23, 27], sensor networks [8, 65], and compressed sensing [12, 30]. We refer the reader to recent surveys on the data stream model [18, 59, 60].

One of the oldest and most fundamental problems in the area of data streams is the problem of finding the ℓ_1-heavy hitters (or simply, "heavy hitters"), also known as the top-k, most popular items, frequent items, elephants, or iceberg queries. Such algorithms can be used as subroutines in network flow identification at IP routers [27], association rules and frequent itemsets [2, 32, 35, 64, 68], iceberg queries and iceberg datacubes [6, 28, 31]. The survey [19] presents an overview of the state-of-the-art for this problem, from both theoretical and practical standpoints.

We now formally define the heavy hitters problem that we focus on in this paper:

Definition 1. **((ε, φ)-Heavy Hitters Problem)** *In the (ε, φ)-Heavy Hitters Problem, we are given parameters $0 < \varepsilon < \varphi \leqslant 1$ and a stream $a_1, \ldots, a_m$ of items $a_j \in \{1, 2, \ldots, n\}$. Let f_i denote the number of occurrences of item i, i.e., its frequency. The algorithm should make one pass over the stream and at the end of the stream output a set $S \subseteq \{1, 2, \ldots, n\}$ for which if $f_i \geqslant \varphi m$, then $i \in S$, while if $f_i \leqslant (\varphi - \varepsilon)m$, then $i \notin S$. Further, for each item $i \in S$, the algorithm should output an estimate $\tilde{f}_i$ of the frequency f_i which satisfies $|f_i - \tilde{f}_i| \leqslant \varepsilon m$.*

Note that other natural definitions of heavy hitters are possible and sometimes used. For example, *ℓ_2-heavy hitters* are those items i for which $f_i^2 \geqslant \varphi^2 \sum_{j=1}^n f_j^2$, and more generally, *$\ell_p$-heavy hitters* are those items i for which $f_i^p \geqslant \varphi^p \sum_{j=1}^n f_j^p$. It is in this sense that Definition 1 corresponds to ℓ_1-heavy hitters.

We are interested in algorithms which use as little space in bits as possible to solve the (ε, φ)-**Heavy Hitters Problem**. Further, we are also interested in minimizing the *update time* and *reporting time* of such algorithms. Here, the update time is defined to be the time the algorithm needs to update its data structure when processing a stream insertion. The reporting time is the time the algorithm needs to report the answer after having processed the stream. We allow the algorithm to be randomized and to succeed with probability at least $1 - \delta$ for $0 < \delta < 1$. We do not make any assumption on the ordering of the stream $a_1, \ldots, a_m$. This is desirable as often in applications one cannot assume a best-case or even a random order. We are also interested in the case when the length m of the stream is not known in advance, and give algorithms in this more general setting.

The first algorithm for the (ε, φ)-**Heavy Hitters Problem** was given by Misra and Gries [54], who achieved $O(\varepsilon^{-1}(\log n + \log m))$ bits of space for any $\varphi > \varepsilon$. This algorithm was rediscovered by Demaine et al. [23], and again by Karp et al. [38]. Other than these algorithms, which are deterministic, there are also a number of randomized algorithms, such as the CountSketch [16], Count-Min sketch [21], sticky sampling [48], lossy counting [48], space-saving [52], sample and hold [27], multi-stage bloom filters [15], and sketch-guided sampling [41]. Berinde et al. [5] show that using $O(k\varepsilon^{-1}\log(mn))$ bits of space, one can achieve the stronger guarantee of reporting, for each item $i \in S$, $\tilde{f}_i$ with $|\tilde{f}_i - f_i| \leqslant \varepsilon/k F_1^{res(k)}$, where $F_1^{res(k)} < m$ denotes the sum of frequencies of items in $\{1, 2, \ldots, n\}$ excluding the frequencies of the k most frequent items.

We emphasize that prior to our work the best known algorithms for the (ε, φ)-**Heavy Hitters Problem** used $O(\varepsilon^{-1}(\log n + \log m))$ bits of space. Two previous lower bounds were known. The first is a lower bound of $\log(\binom{n}{1/\varphi}) = \Omega(\varphi^{-1}\log(\varphi n))$ bits, which comes from the fact that the output set S can contain φ^{-1} items and it takes this many bits to encode them. The second lower bound is $\Omega(\varepsilon^{-1})$ which follows from a folklore reduction from the randomized communication complexity of the Index problem. In this problem, there are two players, Alice and Bob. Alice has a bit string x of length $(2\varepsilon)^{-1}$, while Bob has an index i. Alice creates a stream of length $(2\varepsilon)^{-1}$ consisting of one copy of each j for which $x_j = 1$ and copies of a dummy item to fill the rest of the stream. She runs the heavy hitters streaming algorithm on her stream and sends the state of the algorithm to Bob. Bob appends $(2\varepsilon)^{-1}$ copies of the item i to the stream and continues the execution. For $\varphi = 1/2$, it holds that $i \in S$. Moreover, f_i differs by an additive εm factor depending on whether $x_i = 1$ or $x_i = 0$. Therefore by the randomized communication complexity of the Index problem [40], the $(\varepsilon, 1/2)$-heavy hitters problem requires $\Omega(\varepsilon^{-1})$ bits of space. Although this proof was for $\varphi = 1/2$, no better lower bound is known for any $\varphi > \varepsilon$.

So, while the upper bound for the (ε, φ)-**Heavy Hitters Problem** is $O(\varepsilon^{-1}(\log n + \log m))$ bits, the best known lower bound is only $\Omega(\varphi^{-1}\log n + \varepsilon^{-1})$ bits. We note that we assume $\log(\varphi n) = \Omega(\log n)$, that is, that $\varphi \geqslant 1/n^{1-\Omega(1)}$. For constant φ, and $\log n \approx \varepsilon^{-1}$, this represents a nearly quadratic gap in upper and lower bounds. Given the limited resources of devices which typically run heavy hitters algorithms, such as internet routers, this quadratic gap can be critical in applications.

A problem related to the (ε, φ)-**Heavy Hitters Problem** is estimating the *maximum frequency* in a data stream, also known as the ℓ_∞-norm. In the IITK 2006 Workshop on Algorithms for Data Streams, Open Question 3 asks for an algorithm to estimate the maximum frequency of any item up to an additive εm error using as little space as possible. The best known space bound is still $O(\varepsilon^{-1}\log n)$ bits, as stated in the original formulation of the question (note that the "m" in the question there corresponds to the "n" here). Note that, if one can find an item whose frequency is the largest, up to an additive εm error, then one can solve this problem. The latter problem is independently interesting and corresponds to finding approximate plurality election winners in voting streams [24]. We refer to this problem as the ε-**Maximum** problem.

Finally, we note that there are many other variants of the (ε, φ)-**Heavy Hitters Problem** that one can consider. One simple variant of the above is to output an item of frequency within εm of the *minimum frequency* of any item in the universe. Note that an item of frequency 0 is considered an item of minimum frequency in our definition (see Section 2). We refer to this as the ε-**Minimum** problem. This only makes sense for small universes, as otherwise outputting a random item typically works. This is useful when one wants to count the "number of dislikes", or in anomaly detection; see more motivation below. In other settings, one may not have numerical scores associated with the items, but rather, each stream update consists of a "ranking" or "total ordering" of all stream items. This may be the case in ranking aggregation on the web (see, e.g., [47, 51]) or in voting streams (see, e.g., [13, 17, 24, 73]). One may consider a variety of aggregation measures, such as the Borda score of an item i, which asks for the sum, over rankings, of the number of items $j \neq i$ for which i is ranked ahead of j in the ranking. Alternatively, one may consider the Maximin score of an item i, which asks for the minimum, over items $j \neq i$, of the number of rankings for which i is ranked ahead of j. For these aggregation measures, one may be interested in finding an item whose score is an approximate maximum. This is the analogue of the ε-**Maximum** problem above. Or, one may be interested in listing all items whose score is above a threshold, which is the analogue of the (ε, φ)-**Heavy Hitters Problem**.

We give more motivation of these variants of heavy hitters in this section below, and precise definitions in Section 2.

1.1 Our Contributions

Our results are summarized in Table 1. We note that independently of this work and nearly parallelly, there have been improvements to the space complexity of the ℓ_2-heavy hitters problem in insertion streams [10, 11] and to the time complexity of the ℓ_p-heavy hitters problem in turnstile[1] streams [43] for all $p \in (0, 2]$. These works use very different techniques. See also [72] for a survey of some recent new algorithms for heavy hitters.

Our first contribution is an optimal algorithm and lower bound for the (ε, φ)-**Heavy Hitters Problem**. Namely, we show that there is a randomized algorithm with constant probability of success which solves this problem using

$$O(\varepsilon^{-1}\log\varphi^{-1} + \varphi^{-1}\log n + \log\log m)$$

bits of space, and we prove a lower bound matching up to constant factors. In the unit-cost RAM model with $O(\log n)$ bit words, our algorithm has $O(1)$ update time and reporting time linear in the output size, under the standard assumptions that the length of the stream and universe size are at least $\mathrm{poly}(\varepsilon^{-1}\log(1/\varphi))$. Furthermore, we can achieve nearly the optimal space complexity even when the length

[1]In a turnstile stream, updates modify an underlying n-dimensional vector x initialized at the zero vector; each update is of the form $x \leftarrow x + e_i$ or $x \leftarrow x - e_i$ where e_i is the i'th standard unit vector. In an insertion stream, only updates of the form $x \leftarrow x + e_i$ are allowed.

Problem	Space complexity	
	Upper bound	Lower bound
(ε, φ)-Heavy Hitters	$O\left(\varepsilon^{-1}\log\varphi^{-1} + \varphi^{-1}\log n + \log\log m\right)$ [Theorem 2 and 7]	$\Omega\left(\varepsilon^{-1}\log\varphi^{-1} + \varphi^{-1}\log n + \log\log m\right)$ [Theorem 9 and 14]
ε-Maximum and ℓ_∞-approximation	$O\left(\varepsilon^{-1}\log\varepsilon^{-1} + \log n + \log\log m\right)$ [Theorem 1 and 7]	$\Omega\left(\varepsilon^{-1}\log\varepsilon^{-1} + \log n + \log\log m\right)$ [Theorem 9 and 14]
ε-Minimum	$O\left(\varepsilon^{-1}\log\log\varepsilon^{-1} + \log\log m\right)$ [Theorem 4 and 8]	$\Omega\left(\varepsilon^{-1} + \log\log m\right)$ [Theorem 11 and 14]
ε-Borda	$O\left(n(\log\varepsilon^{-1} + \log n) + \log\log m\right)$ [Theorem 5 and 8]	$\Omega\left(n(\log\varepsilon^{-1} + \log n) + \log\log m\right)$ [Theorem 12 and 14]
ε-Maximin	$O\left(n\varepsilon^{-2}\log^2 n + \log\log m\right)$ [Theorem 6 and 8]	$\Omega\left(n(\varepsilon^{-2} + \log n) + \log\log m\right)$ [Theorem 13]

Table 1: The bounds hold for constant success probability algorithms and for n sufficiently large in terms of ε. For the (ε, φ)-Heavy Hitters problem and the ε-Maximum problem, we also achieve $O(1)$ update time and reporting time which is linear in the size of the output. The upper bound for ε-Borda (resp. ε-Maximin) is for returning every item's Borda score (resp. Maximin score) up to an additive εmn (resp. additive εm), while the lower bound for ε-Borda (resp. ε-Maximin) is for returning only the approximate Borda score (resp. Maximin score) of an approximate maximum.

m of the stream is *not known in advance*. Although the results of [5] achieve stronger error bounds in terms of the tail, which are useful for skewed streams, here we focus on the original formulation of the problem.

Next, we turn to the problem of estimating the maximum frequency in a data stream up to an additive εm. We give an algorithm using

$$O(\varepsilon^{-1}\log\varepsilon^{-1} + \log n + \log\log m)$$

bits of space, improving the previous best algorithms which required space at least $\Omega(\varepsilon^{-1}\log n)$ bits, and show that our bound is tight. As an example setting of parameters, if $\varepsilon^{-1} = \Theta(\log n)$ and $\log\log m = O(\log n)$, our space complexity is $O(\log n \log\log n)$ bits, improving the previous $\Omega(\log^2 n)$ bits of space algorithm. We also prove a lower bound showing our algorithm is optimal up to constant factors. This resolves Open Question 3 from the IITK 2006 Workshop on Algorithms for Data Streams in the case of insertion streams, for the case of "ℓ_1-heavy hitters". Our algorithm also returns the the item with the approximate maximum frequency, solving the **ε-Maximum** problem.

We then focus on a number of variants of these problems. We first give nearly tight bounds for finding an item whose frequency is within εm of the minimum possible frequency. While this can be solved using our new algorithm for the **$(\varepsilon, \varepsilon)$-Heavy Hitters Problem**, this would incur $\Omega(\varepsilon^{-1}\log\varepsilon^{-1} + \log\log m)$ bits of space, whereas we give an algorithm using only $O(\varepsilon^{-1}\log\log(\varepsilon^{-1}) + \log\log m)$ bits of space. We also show a nearly matching $\Omega(\varepsilon^{-1} + \log\log m)$ bits of space lower bound. We note that for this problem, a dependence on n is not necessary since if the number of possible items is sufficiently large, then outputting the identity of a random item among the first say, $10\varepsilon^{-1}$ items, is a correct solution with large constant probability.

Finally, we study variants of heavy hitter problems that are ranking-based. In this setting, each stream update consists of a total ordering of the n universe items. For the **ε-Borda** problem, we give an algorithm using $O(n(\log\varepsilon^{-1} + \log\log n) + \log\log m)$ bits of space to report the Borda score of every item up to an additive εmn. We also show this is nearly optimal by proving an $\Omega(n\log\varepsilon^{-1} + \log\log m)$ bit lower bound for the problem, even in the case when one is only interested in outputting an item maximum Borda score up to an additive εmn. For the **ε-Maximin** problem, we give an algorithm using $O(n\varepsilon^{-2}\log^2 n + \log\log m)$ bits of space to report the maximin score of every item up to an additive εm, and prove an $\Omega(n\varepsilon^{-2} + \log\log m)$ bits of space lower bound even in the case when one is only interested in outputting the maximum maximin score up to an additive εm. This shows that finding heavy hitters with respect to the maximin score is significantly more expensive than with respect to the Borda score.

1.2 Motivations for Variants of Heavy Hitters

While the **(ε, φ)-Heavy Hitters** and **ε-Maximum** problem are very well-studied in the data stream literature, the other variants introduced are not. We provide additional motivation for them here.

For the **ε-Minimum** problem, in our formulation, an item with frequency zero, i.e., one that does not occur in the stream, is a valid solution to the problem. In certain scenarios, this might not make sense, e.g., if a stream containing only a small fraction of IP addresses. However, in other scenarios we argue this is a natural problem. For instance, consider an online portal where users register complaints about products. Here, minimum frequency items correspond to the "best" items. That is, such frequencies arise in the context of voting or more generally making a choice: in cases for which one does not have a strong preference for an item, but definitely does not like certain items, this problem applies, since the frequencies correspond to "number of dislikes".

The **ε-Minimum** problem may also be useful for anomaly detection. Suppose one has a known set of sensors broadcasting information and one observes the "From:" field in the broadcasted packets. Sensors which send a small number of packets may be down or defective, and an algorithm for the **ε-Minimum** problem could find such sensors.

Finding items with maximum and minimum frequencies in a stream correspond to finding winners under plurality and veto voting rules respectively in the context of voting[2] [9].

[2]In fact, the first work [56] to formally pose the heavy hitters problem couched it in the context of voting.

The streaming aspect of voting could be crucial in applications like online polling [39], recommender systems [1, 33, 63] where the voters are providing their votes in a streaming fashion and at every point in time, we would like to know the popular items. While in some elections, such as for political positions, the scale of the election may not be large enough to require a streaming algorithm, one key aspect of these latter voting-based problems is that they are rank-based which is useful when numerical scores are not available. Orderings naturally arise in several applications - for instance, if a website has multiple parts, the order in which a user visits the parts given by its clickstream defines a voting, and for data mining and recommendation purposes the website owner may be interested in aggregating the orderings across users. Motivated by this connection, we define similar problems for two other important voting rules, namely Borda and maximin. The Borda scoring method finds its applications in a wide range of areas of artificial intelligence, for example, machine learning [14, 36, 62, 70], image processing [46, 55], information retrieval [4, 45, 61], etc. The Maximin score is often used when the spread between the best and worst outcome is very large (see, e.g., p. 373 of [58]). The maximin scoring method also has been used frequently in machine learning [37, 71], human computation [49, 50], etc.

2. PRELIMINARIES

We denote the disjoint union of sets by $\sqcup$. We denote the set of all permutations of a set $\mathcal{U}$ by $\mathcal{L}(\mathcal{U})$. For a positive integer ℓ, we denote $\{1, \ldots, \ell\}$ by $[\ell]$. In most places, we ignore floors and ceilings for the sake of notational simplicity.

2.1 Model of Input Data

The input data is an insertion-only stream of elements from some universe $\mathcal{U}$. In the context of voting, the input data is an insertion-only stream over the universe of all possible rankings (permutations).

2.2 Communication Complexity

We will use lower bounds on *communication complexity* of certain functions to prove space complexity lower bounds for our problems. The communication complexity of a function measures the number of bits that need to be exchanged between two players to compute a function whose input is split among those two players [74]. In a more restrictive *one-way communication model*, Alice, the first player, sends only one message to Bob, the second player, and Bob outputs the result. A protocol is a method that the players follow to compute certain functions of their input. Also the protocols can be randomized; in that case, the protocol needs to output correctly with probability at least $1 - \delta$, for $\delta \in (0, 1)$ (the probability is taken over the random coin tosses of the protocol). The randomized one-way communication complexity of a function f with error probability δ is denoted by $\mathcal{R}_{\delta}^{\text{1-way}}(f)$. [42] is a standard reference for communication complexity.

2.3 Model of Computation

Our model of computation is the unit-cost RAM model on words of size $O(\log n)$, capable of generating uniformly random words and of performing arithmetic operations in $\{+, -, \log_2\}$ in one unit of time. We note that this model of computation has been used before [25]. We store an integer C using a variable length array of [7] which allows us to read and update C in $O(1)$ time and $O(\log C)$ bits of space.

2.4 Universal Family of Hash Functions

Definition 2. *(Universal family of hash functions)* *A family of functions $\mathcal{H} = \{h | h : A \to B\}$ is called a* universal family of hash functions *if for all $a \neq b \in A$, $\Pr\{h(a) = h(b)\} = 1/|B|$, where h is picked uniformly at random from $\mathcal{H}$.*

We know that there exists a universal family of hash functions $\mathcal{H}$ from $[k]$ to $[\ell]$ for every positive integer ℓ and every prime k [44]. Moreover, $|\mathcal{H}|$, the size of $\mathcal{H}$, is $O(k^2)$.

2.5 Problem Definitions

We now formally define the problems we study here. Suppose we have $0 < \varepsilon < \varphi < 1$.

Definition 3. (ε, φ)-List heavy hitters
Given an insertion-only stream of length m over a universe $\mathcal{U}$ of size n, find all items in $\mathcal{U}$ with frequency more than φm, along with their frequencies up to an additive error of εm, and report no items with frequency less than $(\varphi - \varepsilon)m$.

Definition 4. ε-Maximum
Given an insertion-only stream of length m over a universe $\mathcal{U}$ of size n, find the maximum frequency up to an additive error of εm.

Next we define the *minimum* problem for $0 < \varepsilon < 1$.

Definition 5. ε-Minimum
Given an insertion-only stream of length m over a universe $\mathcal{U}$ of size n, find the minimum frequency up to an additive error of εm.

Next we define related heavy hitters problems in the context of rank aggregation. The input is a stream of rankings (permutations) over an item set $\mathcal{U}$ for the problems below. The *Borda score* of an item i is the sum, over all rankings, of the number of items $j \neq i$ for which i is ranked ahead of j in the ranking.

Definition 6. (ε, φ)-List borda
Given an insertion-only stream over a universe $\mathcal{L}(\mathcal{U})$ where $|\mathcal{U}| = n$, find all items with Borda score more than φmn, along with their Borda score up to an additive error of εmn, and report no items with Borda score less than $(\varphi - \varepsilon)mn$.

Definition 7. ε-Borda
Given an insertion-only stream over a universe $\mathcal{L}(\mathcal{U})$ where $|\mathcal{U}| = n$, find the maximum Borda score up to an additive error of εmn.

The *maximin score* of an item i is the minimum, over all items $j \neq i$, of the number of rankings for which i is ranked ahead of j.

Definition 8. (ε, φ)-List maximin
Given an insertion-only stream over a universe $\mathcal{L}(\mathcal{U})$ where $|\mathcal{U}| = n$, find all items with maximin score more than φm along with their maximin score up to an additive error of εm, and report no items with maximin score less than $(\varphi - \varepsilon)m$.

Definition 9. ε-MAXIMIN
Given an insertion-only stream over a universe $\mathcal{L}(\mathcal{U})$ where $|\mathcal{U}| = n$, find the maximum maximin score up to an additive error of εm.

Notice that the maximum possible Borda score of an item is $m(n-1) = \Theta(mn)$ and the maximum possible maximin score of an item is m. This justifies the approximation factors in Definition 6 to 9. We note that finding an item with maximum Borda score within additive εmn or maximum maximin score within additive εm corresponds to finding an approximate winner of an election (more precisely, what is known as an ε-winner) [24].

3. ALGORITHMS

In this section, we present our upper bound results. All omitted proofs are in Appendix B. Before describing specific algorithms, we record some claims for later use. Lemma 1 follows by checking whether we get all heads in $\log m$ tosses of a fair coin.

Lemma 1. *Suppose m is a power of two*[3]. *Then there is an algorithm $\mathcal{A}$ for choosing an item with probability $1/m$ that has space complexity of $O(\log\log m)$ bits and time complexity of $O(1)$ in the unit-cost RAM model.*

Our second claim is a standard result for universal families of hash functions.

Lemma 2. *For $S \subseteq A$, $\delta \in (0,1)$, and universal family of hash functions $\mathcal{H} = \{h | h : A \to [\lceil |S|^2/\delta \rceil]\}$:*

$$\Pr_{h \in_{unif} \mathcal{H}}[\exists i \neq j \in S, h(i) = h(j)] \leqslant \delta$$

Our third claim is folklore and also follows from the celebrated DKW inequality [26]. We provide a simple proof here that works for constant δ.

We assume δ is a constant throughout this paper. This constant can be made arbitrarily small by independent repetition (in general, the complexity grows by a multiplicative factor of $\log(1/\delta)$), finding the heavy items that occur in a large constant fraction of the repetitions, and using the median of their estimated frequencies.

Lemma 3. *Let f_i and $\hat{f}_i$ be the frequencies of an item i in a stream $\mathcal{S}$ and in a random sample $\mathcal{T}$ of size r from $\mathcal{S}$, respectively. Then for $r \geqslant 2\varepsilon^{-2}\log(2\delta^{-1})$, with probability $1-\delta$, for every universe item i simultaneously,*

$$\left|\frac{\hat{f}_i}{r} - \frac{f_i}{m}\right| \leqslant \varepsilon.$$

PROOF FOR CONSTANT δ. This follows by Chebyshev's inequality and a union bound. Indeed, consider a given $i \in [n]$ with frequency f_i and suppose we sample each of its occurrences pairwise-independently with probability r/m, for a parameter r. Then the expected number $\mathbf{E}[\hat{f}_i]$ of sampled occurrences is $f_i \cdot r/m$ and the variance $\mathbf{Var}[\hat{f}_i]$ is $f_i \cdot r/m(1-r/m) \leqslant f_i r/m$. Applying Chebyshev's inequality,

$$\Pr\left[\left|\hat{f}_i - \mathbf{E}[\hat{f}_i]\right| \geqslant \frac{r\varepsilon}{2}\right] \leqslant \frac{\mathbf{Var}[\hat{f}_i]}{(r\varepsilon/2)^2} \leqslant \frac{4f_i r}{mr^2\varepsilon^2}.$$

Setting $r = \frac{C}{\varepsilon^2}$ for a constant $C > 0$ makes this probability at most $\frac{4f_i}{Cm}$. By the union bound, if we sample each element in the stream independently with probability $\frac{r}{m}$, then the probability there exists an i for which $|\hat{f}_i - \mathbf{E}[\hat{f}_i]| \geqslant \frac{r\varepsilon}{2}$ is at most $\sum_{i=1}^{n} \frac{4f_i}{Cm} \leqslant \frac{4}{C}$, which for $C \geqslant 400$ is at most $\frac{1}{100}$, as desired. $\square$

For now, assume that the length of the stream is known in advance; we show in Section 3.5 how to remove this assumption.

3.1 List Heavy Hitters

For the (ε, φ)-LIST HEAVY HITTERS problem, we present two algorithms. The first is slightly suboptimal, but simple conceptually and already constitutes a very large improvement in the space complexity over known algorithms. We expect that this algorithm could be useful in practice as well. The second algorithm is more complicated, building on ideas from the first algorithm, and achieves the optimal space complexity upto constant factors.

We note that both algorithms proceed by sampling $O(\varepsilon^{-2}\ln(1/\delta))$ stream items and updating a data structure as the stream progresses. In both cases, the time to update the data structure is bounded by $O(1/\varepsilon)$, and so, under the standard assumption that the length of the stream is at least $\mathrm{poly}(\ln(1/\delta)\varepsilon)$, the time to perform this update can be spread out across the next $O(1/\varepsilon)$ stream updates, since with large probability there will be no items sampled among these next $O(1/\varepsilon)$ stream updates. Therefore, we achieve worst-case[4] update time of $O(1)$.

3.1.1 *A simpler, near-optimal algorithm*

Theorem 1. *Assume the stream length is known beforehand, and $\delta > 0$ is any constant. Then there is a randomized one-pass algorithm $\mathcal{A}$ for the (ε, φ)-LIST HEAVY HITTERS problem which succeeds with probability at least $1-\delta$ using $O\left(\varepsilon^{-1}\log 1/\varepsilon + (1/\varphi)\log n + \log\log m\right)$ bits of space. Moreover, $\mathcal{A}$ has an update time of $O(1)$ and reporting time linear in its output size.*

Overview. The overall idea is as follows. We sample $\ell = O(\varepsilon^{-2})$ many items from the stream uniformly at random as well as hash the id's (the word "id" is short for identifier) of the sampled elements into a space of size $O(\varepsilon^{-4})$. Now, both the stream length as well as the universe size are $\mathrm{poly}(\varepsilon^{-1})$. From Lemma 3, it suffices to solve the heavy hitters problem on the sampled stream. From Lemma 2, because the hash function is chosen from a universal family, the sampled elements have distinct hashed id's. We can then feed these elements into a standard Misra-Gries data structure with ε^{-1} counters, incurring space $O(\varepsilon^{-1}\log\varepsilon^{-1})$. Because we want to return the unhashed element id's for the heavy hitters, we also use $\log n$ space for recording the φ^{-1} top items according to the Misra-Gries data structure and output these when asked to report.

[3]In all our algorithms, whenever we pick an item with probability $p > 0$, we can assume, without loss of generality, that $1/p$ is a power of two. If not, then we replace p with p' where $1/p'$ is the largest power of two less than $1/p$. This does not affect correctness and performance of our algorithms.

[4]We emphasize that this is stronger than an amortized guarantee, as on every insertion, the cost will be $O(1)$.

Algorithm 1 for (ε, φ)-List heavy hitters

```
Input: A stream $\mathcal{S}$ of length $m$ over $\mathcal{U} = [n]$; let $f(x)$ be
    the frequency of $x \in \mathcal{U}$ in $\mathcal{S}$
Output: A set $X \subseteq \mathcal{U}$ and a function $\hat{f} : X \to \mathbb{N}$ such that
    if $f(x) \geqslant \varphi m$, then $x \in X$ and $f(x) - \varepsilon m \leqslant \hat{f}(x) \leqslant$
    $f(x) + \varepsilon m$ and if $f(y) \leqslant (\varphi - \varepsilon)m$, then $y \notin X$ for every
    $x, y \in \mathcal{U}$
 1: Initialize:
 2:     $\ell \leftarrow 6\log(6/\delta)/\varepsilon^2$
 3:     Hash function $h$ uniformly at random from a
        universal family $\mathcal{H} \subseteq \{h : [n] \to \lceil 4\ell^2/\delta \rceil\}$.
 4:     An empty table $\mathcal{T}_1$ of (key, value) pairs of length
        $\varepsilon^{-1}$. Each key entry of $\mathcal{T}_1$ can store an integer in
        $[0, \lceil 4\ell^2/\delta \rceil]$ and each value entry can store an
        integer in $[0, 11\ell]$.  ▷ The table $\mathcal{T}_1$ will be in sorted
        order by value throughout.
 5:     An empty table $\mathcal{T}_2$ of length $1/\varphi$. Each entry of $\mathcal{T}_2$
        can store an integer in $[0, n]$.  ▷ The entries of $\mathcal{T}_2$
        will correspond to ids of the keys in $\mathcal{T}_1$ of the
        highest $1/\varphi$ values
 6:
 7: procedure Insert(x)
 8:     With probability $p = 6\ell/m$, continue. Otherwise, return .
 9:     Perform Misra-Gries update using $h(x)$ maintaining
        $\mathcal{T}_1$ sorted by values.
10:     if The value of $h(x)$ is among the highest $1/\varphi$ valued
        items in $\mathcal{T}_1$ then
11:         if $x_i$ is not in $\mathcal{T}_2$ then
12:             if $\mathcal{T}_2$ currently contains $1/\varphi$ many items then
13:                 For $y$ in $\mathcal{T}_2$ such that $h(y)$ is not among
                    the highest $1/\varphi$ valued items in $\mathcal{T}_1$, replace $y$ with $x$.
14:             else
15:                 We put $x$ in $\mathcal{T}_2$.
16:         Ensure that elements in $\mathcal{T}_2$ are ordered according
            to corresponding values in $\mathcal{T}_1$.
17:
18: procedure Report( )
19:     return items in $\mathcal{T}_2$ along with their corresponding
        values in $\mathcal{T}_1$
```

Proof of Theorem 1. The pseudocode of our (ε, φ)-List heavy hitters algorithm is in Algorithm 1. By Lemma 3, if we select a subset $\mathcal{S}$ of size at least $\ell = 6\varepsilon^{-2}\log(6\delta^{-1})$ uniformly at random from the stream, then $\Pr[\forall i \in \mathcal{U}, |(\hat{f}_i/|\mathcal{S}|) - (f_i/n)| \leqslant \varepsilon] \geqslant 1 - \delta/3$, where f_i and $\hat{f}_i$ are the frequencies of item i in the input stream and $\mathcal{S}$ respectively. First we show that with the choice of p in line 10 in Algorithm 1, the number of items sampled is at least ℓ and at most 11ℓ with probability at least $(1 - \delta/3)$. Let X_i be the indicator random variable of the event that the item x_i is sampled for $i \in [m]$. Then the total number of items sampled $X = \sum_{i=1}^{m} X_i$. We have $\mathbb{E}[X] = 6\ell$ since $p = 6\ell/m$. Now we have the following.

$$\Pr[X \leqslant \ell \text{ or } X \geqslant 11\ell] \leqslant \Pr[|X - \mathbb{E}[X]| \geqslant 5\ell] \leqslant \delta/3$$

The inequality follows from the Chernoff bound and the value of ℓ. From here onwards we assume that the number of items sampled is in $[\ell, 11\ell]$.

We use (a modified version of) the Misra-Gries algorithm [54] to estimate the frequencies of items in $\mathcal{S}$. The length of the table in the Misra-Gries algorithm is ε^{-1}. We pick a hash function h uniformly at random from a universal family $\mathcal{H} = \{h|h : [n] \to \lceil 4\ell^2/\delta \rceil\}$ of hash functions of size $|\mathcal{H}| = O(n^2)$. Note that picking a hash function h uniformly at random from $\mathcal{H}$ can be done using $O(\log n + \log(1/\varepsilon) + \log(1/\delta))$ bits of space, which is $O(\log n)$ bits for constant δ and assuming that $\varepsilon \geqslant 1/n$ (as otherwise, there is an $\Omega(n)$ lower bound for the problem). Lemma 2 shows that there are no collisions in $\mathcal{S}$ under this hash function h with probability at least $1 - \delta/3$. From here onwards we assume that there is no collision among the ids of the sampled items under the hash function h.

We modify the Misra-Gries algorithm as follows. Instead of storing the id of any item x in the Misra-Gries table (table $\mathcal{T}_1$ in line 5 in Algorithm 1) we only store the hash $h(x)$ of the id x. We also store the ids (not the hash of the id) of the items with highest $1/\varphi$ values in $\mathcal{T}_1$ in another table $\mathcal{T}_2$. Moreover, we always maintain the table $\mathcal{T}_2$ consistent with the table $\mathcal{T}_1$ in the sense that the i^{th} highest valued key in $\mathcal{T}_1$ is the hash of the i^{th} id in $\mathcal{T}_2$.

Upon picking an item x with probability p, we create an entry corresponding to $h(x)$ in $\mathcal{T}_1$ and make its value one if there is space available in $\mathcal{T}_1$; decrement the value of every item in $\mathcal{T}_1$ by one if the table is already full; increment the entry in the table corresponding to $h(x)$ if $h(x)$ is already present in the table. When we decrement the value of every item in $\mathcal{T}_1$, the table $\mathcal{T}_2$ remains consistent and we do not need to do anything else. Otherwise there are three cases to consider. Case 1: $h(x)$ is not among the $1/\varphi$ highest valued items in $\mathcal{T}_1$. In this case, we do not need to do anything else. Case 2: $h(x)$ was not among the $1/\varphi$ highest valued items in $\mathcal{T}_1$ but now it is among the $1/\varphi$ highest valued items in $\mathcal{T}_1$. In this case the last item y in $\mathcal{T}_2$ is no longer among the $1/\varphi$ highest valued items in $\mathcal{T}_1$. We replace y with x in $\mathcal{T}_2$. Case 3: $h(x)$ was among the $1/\varphi$ highest valued items in $\mathcal{T}_1$. When the stream finishes, we output the ids of all the items in table $\mathcal{T}_2$ along with the values corresponding to them in table $\mathcal{T}_1$. Correctness follows from the correctness of the Misra-Gries algorithm and the fact that there is no collision among the ids of the sampled items. □

3.1.2 An optimal algorithm

Theorem 2. *Assume the stream length is known beforehand. Then there is a randomized one-pass algorithm $\mathcal{A}$ for the (ε, φ)-*List heavy hitters *problem which succeeds with constant probability using $O\left(\varepsilon^{-1}\log\varphi^{-1} + \varphi^{-1}\log n + \log\log m\right)$ bits of space. Moreover, $\mathcal{A}$ has an update time of $O(1)$ and reporting time linear in its output size.*

Overview. As in the simpler algorithm, we sample $\ell = O(\varepsilon^{-2})$ many stream elements and solve the $(\varepsilon/2, \varphi)$-List heavy hitters problem on this sampled stream. Also, the Misra-Gries algorithm for $(\varphi/2, \varphi)$-List heavy hitters returns a candidate set of $O(\varphi^{-1})$ items containing all items of frequency at least $\varphi\ell$. It remains to count the frequencies of these $O(\varphi^{-1})$ items with upto $\varepsilon\ell/2 = O(\varepsilon^{-1})$ *additive* error, so that we can remove those whose frequency is less than $(\varphi - \varepsilon/2)\ell$.

Fix some item $i \in [n]$, and let f_i be i's count in the sampled stream. A natural approach to count f_i approximately is to increment a counter probabilistically, instead of deterministically, at every occurrence of i. Suppose that we increment a counter with probability $0 \leqslant p_i \leqslant 1$ whenever item i

Algorithm 2 for (ε, φ)-List heavy hitters

```
Input: A stream $\mathcal{S}$ of length $m$ over universe $\mathcal{U} = [n]$; let
    $f(x)$ be the frequency of $x \in \mathcal{U}$ in $\mathcal{S}$
Output: A set $X \subseteq \mathcal{U}$ and a function $\hat{f} : X \to \mathbb{N}$ such that
    if $f(x) \geqslant \varphi m$, then $x \in X$ and $f(x) - \varepsilon m \leqslant \hat{f}(x) \leqslant$
    $f(x) + \varepsilon m$ and if $f(y) \leqslant (\varphi - \varepsilon)m$, then $y \notin X$ for every
    $x, y \in \mathcal{U}$
 1: Initialize:
     2: $\ell \leftarrow 10^5 \varepsilon^{-2}$
     3: $s \leftarrow 0$
     4: Hash functions $h_1, \ldots, h_{200\log(12\varphi^{-1})}$ uniformly at
        random from a universal family
        $\mathcal{H} \subseteq \{h : [n] \to [^{100}/_{\varepsilon}]\}$.
     5: An empty table $\mathcal{T}_1$ of (key, value) pairs of length
        $2\varphi^{-1}$. Each key entry of $\mathcal{T}_1$ can store an element
        of $[n]$ and each value entry can store an integer in
        $[0, 10\ell]$.
     6: An empty table $\mathcal{T}_2$ with $100\varepsilon^{-1}$ rows and
        $200\log(12\varphi^{-1})$ columns. Each entry of $\mathcal{T}_2$ can
        store an integer in $[0, 100\varepsilon\ell]$.
     7: An empty 3-dimensional table $\mathcal{T}_3$ of size at most
        $100\varepsilon^{-1} \times 200\log(12\varphi^{-1}) \times 4\log(\varepsilon^{-1})$. Each entry
        of $\mathcal{T}_3$ can store an integer in $[0, 10\ell]$.   ▷ These are
        upper bounds; not all the allowed cells will
        actually be used.
 8:
 9: procedure Insert($x$)
10:     With probability $^{\ell}/_{m}$, increment $s$ and continue.
    Else, return
11:     Perform Misra-Gries update on $\mathcal{T}_1$ with $x$.
12:     for $j \leftarrow 1$ to $200\log(12\varphi^{-1})$ do
13:         $i \leftarrow h_j(x)$
14:         With probability $\varepsilon$, increment $\mathcal{T}_2[i, j]$
15:         $t \leftarrow \lfloor \log(10^{-6}\mathcal{T}_2[i, j]^2) \rfloor$ and $p \leftarrow \min(\varepsilon \cdot 2^t, 1)$
16:         if $t \geqslant 0$ then
17:             With probability $p$, increment $\mathcal{T}_3[i, j, t]$
18:
19: procedure Report( )
20:     $X \leftarrow \emptyset$
21:     for each key $x$ with nonzero value in $\mathcal{T}_1$ do
22:         for $j \leftarrow 1$ to $200\log(12\varphi^{-1})$ do
23:             $\hat{f}_j(x) \leftarrow \sum_{t=0}^{4\log(\varepsilon^{-1})} \mathcal{T}_3[h(x), j, t] / \min(\varepsilon 2^t, 1)$
24:         $\hat{f}(x) \leftarrow \text{median}(\hat{f}_1, \ldots, \hat{f}_{10\log\varphi^{-1}})$
25:         if $\hat{f}(x) \geqslant (\varphi - \varepsilon/2)s$ then
26:             $X \leftarrow X \cup \{x\}$
27:     return $X, \hat{f}$
```

arrives in the stream. Let the value of the counter be $\hat{c}_i$, and let $\hat{f}_i = \hat{c}_i/p_i$. We see that $\mathbb{E}\left[\hat{f}_i\right] = f_i$ and $\mathsf{Var}[\hat{f}_i] \leqslant f_i/p_i$. It follows that if $p_i = \Theta(\varepsilon^2 f_i)$, then $\mathsf{Var}[\hat{f}_i] = O(\varepsilon^{-2})$, and hence, $\hat{f}_i$ is an unbiased estimator of f_i with additive error $O(\varepsilon^{-1})$ with constant probability. We call such a counter an *accelerated counter* as the probability of incrementing accelerates with increasing counts. For each i, we can maintain $O(\log\varphi^{-1})$ accelerated counters independently and take their median to drive the probability of deviating by more than $O(\varepsilon^{-1})$ down to $O(\varphi)$. So, with constant probability, the frequency for the $O(\varphi^{-1})$ items in the Misra-Gries data structure is estimated within $O(\varepsilon^{-1})$ error, as desired.

However, there are two immediate issues with this approach. The first problem is that we may need to keep counts for $\Omega(\ell) = \Omega(\varepsilon^{-2})$ distinct items, which is too costly for our purposes. To get around this, we use a hash function from a universal family to hash the universe to a space of size $u = \Theta(\varepsilon^{-1})$, and we work throughout with the hashed id's. We can then show that the space complexity for each iteration is $O(\varepsilon^{-1})$. Also, the accelerated counters now estimate frequencies of hashed id's instead of actual items, but because of universality, the expected frequency of any hashed id is $\ell/u = O(\varepsilon^{-1})$, our desired error bound.

The second issue is that we need a constant factor approximation of f_i, so that we can set p_i to $\Theta(\varepsilon^2 f_i)$. But because the algorithm needs to be one-pass, we cannot first compute p_i in one pass and then run the accelerated counter in another. So, we divide the stream into *epochs* in which f_i stays within a factor of 2, and use a different p_i for each epoch. In particular, set $p_i^t = \varepsilon \cdot 2^t$ for $0 \leqslant t \leqslant \log(p_i/\varepsilon)$. We want to keep a running estimate of i's count to within a factor of 2 to know if the current epoch should be incremented. For this, we subsample each element of the (already sampled) stream with probability ε independently and maintain exact counts for the observed hashed id's. It is easy to see that this requires only $O(\varepsilon^{-1})$ bits in expectation. Consider any $i \in [u]$ and the prefix of the stream upto $b \leqslant \ell$, and let $f_i(b)$ be i's frequency in the prefix, let $\bar{c}_i(b)$ be i's frequency among the samples in the prefix, and $\bar{f}_i(b) = \frac{\bar{c}_i(b)}{\varepsilon}$. We see that $\mathbb{E}\left[\bar{f}_i(b)\right] = f_i(b)$, Moreover, we show that for any $b \in [\ell]$, $\bar{f}_i(b)$ is a 4-factor approximation of $f_i(b)$ with constant probability. By repeating $O(\log\varphi^{-1})$ times independently and taking the median, the error probability can be driven down to $O(\varphi)$.

Now, for every hashed id $i \in [u]$, we need not one accelerated counter but $O(\log(\varepsilon f_i))$ many, one corresponding to each epoch t. When an element with hash id i arrives at position b, we decide, based on $\bar{f}_i(b)$, the epoch t it belongs to and then increment the t'th accelerated counter with probability p_i^t. The storage cost over all i is still $O(1/\varepsilon)$. Also, we iterate the whole set of accelerated counters $O(\log\varphi^{-1})$ times, making the total storage cost $O(\varepsilon^{-1}\log\varphi^{-1})$.

Let $\hat{c}_{i,t}$ be the count in the accelerated counter for hash id i and epoch t. Then, let $\hat{f}_i = \sum_t \hat{c}_{i,t}/p_i^t$. Clearly, $\mathbb{E}\left[\hat{f}_i\right] = f_i$. The variance is $O(\varepsilon^{-2})$ in each epoch, and so, $\mathsf{Var}[\hat{f}_i] = O(\varepsilon^{-2}\log\varepsilon^{-1})$, not $O(\varepsilon^{-2})$ which we wanted. This issue is fixed by a change in how the sampling probabilities are defined, effectively making the sum a geometric series. We now go on to the formal proof.

Pseudocode appears in Algorithm 2 and the full proof in the appendix. Note that the numerical constants are chosen for convenience of analysis and have not been optimized. Also, for the sake of simplicity, the pseudocode does not have the optimal reporting time, but it can be modified to achieve this; see the proof for details.

3.2 Maximum

By tweaking Algorithm 1 slightly, we get the following result for the ε-Maximum problem.

Theorem 3. *Assume the length of the stream is known beforehand. Then there is a randomized one-pass algorithm $\mathcal{A}$ for the ε-Maximum problem which succeeds with probability at least $1 - \delta$ using $O\left(\min\{^1/_\varepsilon, n\}(\log {^1/_\varepsilon} + \log\log {^1/_\delta}) + \log n + \log\log m\right)$ bits* *of space. Moreover, the algorithm $\mathcal{A}$ has an update time of $O(1)$.*

Proof. Instead of maintaining the table $\mathcal{T}_2$ in Algorithm 1, we just store the actual id of the item with maximum frequency in the sampled items. □

3.3 Minimum

Theorem 4. *Assume the length of the stream is known beforehand. Then there is a randomized one-pass algorithm $\mathcal{A}$ for the ε-Minimum problem which succeeds with probability at least $1-\delta$ using $O\left(({}^1/_\varepsilon)\log\log({}^1/_{\varepsilon\delta})+\log\log m\right)$ bits of space. Moreover, the algorithm $\mathcal{A}$ has an update time of $O(1)$.*

Overview. Pseudocode is provided in Algorithm 3. The idea behind our ε-Minimum problem is as follows. It is most easily explained by looking at the REPORT(x) procedure starting in line 13. In lines 14-15 we ask, is the universe size $|U|$ significantly larger than $1/\varepsilon$? Note that if it is, then outputting a random item from $|U|$ is likely to be a solution. Otherwise $|U|$ is $O(1/\varepsilon)$.

The next point is that if the number of distinct elements in the stream were smaller than $1/(\varepsilon\log(1/\varepsilon))$, then we could just store all the items together with their frequencies with $O(1/\varepsilon)$ bits of space. Indeed, we can first sample $O(1/\varepsilon^2)$ stream elements so that all relative frequencies are preserved up to additive ε, thereby ensuring each frequency can be stored with $O(\log(1/\varepsilon))$ bits. Also, since the universe size is $O(1/\varepsilon)$, the item identifiers can also be stored with $O(\log(1/\varepsilon))$ bits. So if this part of the algorithm starts taking up too much space, we stop, and we know the number of distinct elements is at least $1/(\varepsilon\log(1/\varepsilon))$, which means that the minimum frequency is at most $O(m\varepsilon\log(1/\varepsilon))$. This is what is being implemented in steps 9-10 and 18-19 in the algorithm.

We can also ensure the minimum frequency is at least $\Omega(m\varepsilon/\log(1/\varepsilon))$. Indeed, by randomly sampling $O((\log(1/\varepsilon)/\varepsilon)$ stream elements, and maintaining a bit vector for whether or not each item in the universe occurs - which we can with $O(1/\varepsilon)$ bits of space since $|U| = O(1/\varepsilon)$ - any item with frequency at least $\Omega(\varepsilon m/\log(1/\varepsilon))$ will be sampled and so if there is an entry in the bit vector which is empty, then we can just output that as our solution. This is what is being implemented in steps 8 and 16-17 of the algorithm.

Finally, we now know that the minimum frequency is at least $\Omega(m\varepsilon/\log(1/\varepsilon))$ and at most $O(m\varepsilon\log(1/\varepsilon))$. At this point if we randomly sample $O((\log^6 1/\varepsilon)/\varepsilon)$ stream elements, then by Chernoff bounds all item frequencies are preserved up to a relative error factor of $(1\pm 1/\log^2(1/\varepsilon))$, and in particular the relative minimum frequency is guaranteed to be preserved up to an additive ε. At this point we just maintain the exact counts in the sampled stream but truncate them once they exceed poly$(\log(1/\varepsilon))$) bits, since we know such counts do not correspond to the minimum. Thus we only need $O(\log\log(1/\varepsilon))$ bits to represent their counts. This is implemented in step 11 and step 20 of the algorithm.

3.4 Borda and Maximin

We defer describing the algorithms for Borda and Maximin to the appendix.

Algorithm 3 for ε-Minimum

```
Input: A stream $\mathcal{S}=(x_i)_{i\in[m]}\in\mathcal{U}^m$ of length $m$ over $\mathcal{U}$;
    let $f(x)$ be the frequency of $x\in\mathcal{U}$ in $\mathcal{S}$
Output: An item $x\in\mathcal{U}$ such that $f(x)\leqslant f(y)+\varepsilon m$ for
    every $y\in\mathcal{U}$
 1: Initialize:
 2:   $\ell_1 \leftarrow {}^{\log(6/\varepsilon\delta)}/_\varepsilon$, $\ell_2 \leftarrow {}^{\log(6/\delta)}/_{\varepsilon^2}$, $\ell_3 \leftarrow {}^{\log^6(6/\delta\varepsilon)}/_\varepsilon$
 3:   $p_1 \leftarrow {}^{6\ell_1}/_m$, $p_2 \leftarrow {}^{6\ell_2}/_m$, $p_3 \leftarrow {}^{6\ell_3}/_m$
 4:   $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3 \leftarrow \emptyset$
 5:   $\mathcal{B}_1 \leftarrow$ the bit vector for $\mathcal{S}_1$
 6:
 7: procedure Insert(x)
 8:     Put $x$ in $\mathcal{S}_1$ with probability $p_1$ by updating the bit
        vector $\mathcal{B}_1$
 9:     if the number of distinct items in the stream so far
        is at most ${}^1/_{(\varepsilon\log(1/\varepsilon))}$ then
10:         Pick $x$ with probability $p_2$ and put the id of $x$ in
            $\mathcal{S}_2$ and initialize the corresponding counter to 1 if $x\notin\mathcal{S}_2$
            and increment the counter corresponding to $x$ by 1.
11:     Pick $x$ with probability $p_3$, put the id of $x$ in $\mathcal{S}_3$
        and initialize the corresponding counter to 1 if $x_i\notin\mathcal{S}_3$
        and increment the counter corresponding to $x_i$ by 1.
        Truncate counters of $\mathcal{S}_3$ at $2\log^7({}^2/_{\varepsilon\delta})$.
12:
13: procedure Report( )
14:     if $|\mathcal{U}| \geqslant {}^1/_{((1-\delta)\varepsilon)}$ then
15:         return an item $x$ from the first ${}^1/_{((1-\delta)\varepsilon)}$ items
            in $\mathcal{U}$ (ordered arbitrarily) uniformly at random
16:     if $\mathcal{S}_1 \neq \mathcal{U}$ then
17:         return any item from $\mathcal{U}\setminus\mathcal{S}_1$
18:     if the number of distinct items in the stream is at
        most ${}^1/_{(\varepsilon\log(1/\varepsilon))}$ then
19:         return an item in $\mathcal{S}_2$ with minimum counter
            value in $\mathcal{S}_2$
20:     return the item with minimum frequency in $\mathcal{S}_3$
```

Theorem 5. *Assume the length of the stream is known beforehand. Then there is a randomized one-pass algorithm $\mathcal{A}$ for (ε,φ)-List Borda problem which succeeds with probability at least $1-\delta$ using $O\left(n\left(\log n+\log\frac{1}{\varepsilon}+\log\log\frac{1}{\delta}\right)+\log\log m\right)$ bits of space.*

Theorem 6. *Assume the length of the stream is known beforehand. Then there is a randomized one-pass algorithm $\mathcal{A}$ for (ε,φ)-List maximin problem which succeeds with probability at least $1-\delta$ using $O\left(n\varepsilon^{-2}\log^2 n+n\varepsilon^{-2}\log n\log\delta^{-1}+\log\log m\right)$ bits of space.*

3.5 Unknown stream length

Now we consider the case when the length of the stream is not known beforehand. We present below an algorithm for (ε,φ)-List heavy hitters and ε-Maximum problems in this setting.

Theorem 7. *There is a randomized one-pass algorithm for (ε,φ)-List heavy hitters and ε-Maximum problems with space complexity $O\left(\varepsilon^{-1}\log\varepsilon^{-1}+\varphi^{-1}\log n+\log\log m\right)$ bits and update time $O(1)$ even when the length of the stream is not known beforehand.*

PROOF. We describe below a randomized one-pass algorithm for the $(8\varepsilon, \varphi)$-LIST HEAVY HITTERS problem. We may assume that the length of the stream is at least $1/\varepsilon^2$; otherwise, we use the algorithm in Theorem 1 and get the result. Now we guess the length of the stream to be $1/\varepsilon^2$, but run an instance $\mathcal{I}_1$ of Algorithm 1 with $\ell = \log(6/\delta)/\varepsilon^3$ at line 2. By the choice of the size of the sample (which is $\Theta(\log(1/\delta)/\varepsilon^3)$), $\mathcal{I}_1$ outputs correctly with probability at least $(1-\delta)$, if the length of the stream is in $[1/\varepsilon^2, 1/\varepsilon^3]$. If the length of the stream exceeds $1/\varepsilon^2$, we run another instance $\mathcal{I}_2$ of Algorithm 1 with $\ell = \log(6/\delta)/\varepsilon^3$ at line 2. Again by the choice of the size of the sample, $\mathcal{I}_2$ outputs correctly with probability at least $(1-\delta)$, if the length of the stream is in $[1/\varepsilon^3, 1/\varepsilon^4]$. If the stream length exceeds $1/\varepsilon^3$, we discard $\mathcal{I}_1$, free the space it uses, and run an instance $\mathcal{I}_3$ of Algorithm 1 with $\ell = \log(6/\delta)/\varepsilon^3$ at line 2 and so on. At any point of time, we have at most two instances of Algorithm 1 running. When the stream ends, we return the output of the older of the instances we are currently running. We use the approximate counting method of Morris [57] to approximately count the length of the stream. We know that the Morris counter outputs correctly with probability $(1-2^{-k/2})$ using $O(\log\log m + k)$ bits of space at any point in time [29]. Also, since the Morris counter increases only when an item is read, it outputs correctly up to a factor of four at every position if it outputs correctly at positions $1, 2, 4, \dots, 2^{\lfloor \log_2 m \rfloor}$; call this event E. Then we have $\Pr(E) \geqslant 1-\delta$ by choosing $k = 2\log_2(\log_2 m/\delta)$ and applying a union bound over the positions $1, 2, 4, \dots, 2^{\lfloor \log_2 m \rfloor}$. The correctness of the algorithm follows from the correctness of Algorithm 1 and the fact that we are discarding at most εm many items in the stream (by discarding a run of an instance of Algorithm 1). The space complexity and the $O(1)$ update time of the algorithm follow from Theorem 1, the choice of k above, and the fact that we have at most two instances of Algorithm 1 currently running at any point of time.

The algorithm for the ε-MAXIMUM problem is same as the algorithm above except we use the algorithm in Theorem 3 instead of Algorithm 1. □

Note that this proof technique does not seem to apply to our optimal Algorithm 2. Similarly to Theorem 7, we get the following result for the other problems.

Theorem 8. *There are randomized one-pass algorithms for ε-MINIMUM, (ε, φ)-BORDA, and (ε, φ)-MAXIMIN problems with space complexity $O\left((1/\varepsilon)\log\log(1/\varepsilon\delta) + \log\log m\right)$, $O\left(n\left(\log\log n + \log\frac{1}{\varepsilon} + \log\log\frac{1}{\delta}\right) + \log\log m\right)$, and $O\left(n\log^2 n\log(1/\delta)/\varepsilon^2 + \log\log m\right)$ bits respectively even when the length of the stream is not known beforehand. Moreover, the update time for ε-MINIMUM is $O(1)$.*

4. HARDNESS

In this section, we prove space complexity lower bounds for the ε-HEAVY HITTERS, ε-MINIMUM, ε-BORDA, and ε-MAXIMIN problems. We present reductions from certain communication problems for proving space complexity lower bounds. Let us first introduce those communication problems with necessary results.

4.1 Communication Complexity

Definition 10. *(INDEXING$_{m,t}$)*
Let t and m be positive integers. Alice is given a string $x = (x_1, \cdots, x_t) \in [m]^t$. Bob is given an index $i \in [t]$. Bob has to output x_i.

The following is a well known result [42].

Lemma 4. *$\mathcal{R}_\delta^{\text{1-way}}(\text{INDEXING}_{m,t}) = \Omega(t\log m)$ for constant $\delta \in (0,1)$.*

[67] defines a communication problem called PERM, which we generalize to ε-PERM as follows.

Definition 11. *(ε-PERM)*
Alice is given a permutation σ over $[n]$ which is partitioned into $1/\varepsilon$ many contiguous blocks. Bob is given an index $i \in [n]$ and has to output the block in σ where i belongs.

Our lower bound for ε-PERM matches the lower bound for PERM in Lemma 1 in [67] when $\varepsilon = 1/n$.

Lemma 5. *$\mathcal{R}_\delta^{\text{1-way}}(\varepsilon - \text{PERM}) = \Omega(n\log(1/\varepsilon))$, for any constant $\delta < 1/10$.*

Finally, we consider the GREATER-THAN problem.

Definition 12. *(GREATER-THAN$_n$)*
Alice is given an integer $x \in [n]$ and Bob is given an integer $y \in [n], y \neq x$. Bob has to output 1 if $x > y$ and 0 otherwise.

The following result is due to [53, 66]. We provide a simple proof of it in the appendix for completeness[5].

Lemma 6. *$\mathcal{R}_\delta^{\text{1-way}}(\text{GREATER-THAN}_n) = \Omega(\log n)$, for every $\delta < 1/4$.*

4.2 Reductions

We observe that a trivial $\Omega((1/\varphi)\log n)$ bits lower bound for (ε, φ)-LIST HEAVY HITTERS, (ε, φ)-LIST BORDA, (ε, φ)-LIST MAXIMIN follows from the fact that any algorithm may need to output $1/\varphi$ many items from the universe. Also, there is a trivial $\Omega(n\log n)$ lower bound for (ε, φ)-LIST BORDA and (ε, φ)-LIST MAXIMIN because each stream item is a permutation on $[n]$, hence requiring $\Omega(n\log n)$ bits to read.

We show now a space complexity lower bound of $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varphi})$ bits for the ε-HEAVY HITTERS problem.

Theorem 9. *Suppose the size of universe n is at least $1/4\varepsilon(\varphi-\varepsilon)$. Any randomized one pass (ε, φ)-HEAVY HITTERS algorithm with success probability at least $(1-\delta)$ must use $\Omega((1/\varepsilon)\log 1/\varphi)$ bits of space, for constant $\delta \in (0,1)$.*

PROOF. Consider the INDEXING$_{1/2(\varphi-\varepsilon), 1/2\varepsilon}$ problem where Alice is given a string $x = (x_1, x_2, \cdots, x_{1/\varepsilon}) \in [1/2(\varphi-\varepsilon)]^{1/2\varepsilon}$ and Bob is given an index $i \in [1/2\varepsilon]$. We assume $\varphi > 2\varepsilon$. The stream we generate is over $[1/2(\varphi-\varepsilon)] \times [1/2\varepsilon] \subseteq \mathcal{U}$ (this is possible since $|\mathcal{U}| \geqslant 1/(4\varepsilon(\varphi-\varepsilon))$). Alice generates a stream of length $1/2\varepsilon$ by inserting one copy of (x_j, j) for each $j \in [1/2\varepsilon]$. Alice now sends the memory content of the algorithm to Bob. Bob resumes the run of the algorithm by generating another stream of length $1/2\varepsilon$ by inserting $\varphi/\varepsilon - 1$ copies of (j, i) for each $j \in [1/\varphi-\varepsilon]$. The length of the stream is ε^{-1}, the frequency of the item (x_i, i) is φ/ε, while the frequency of every other item is $\varphi/\varepsilon - 1$ or 1. Hence from the output of the (ε, φ)-HEAVY HITTERS algorithm, Bob knows i with probability at least $(1-\delta)$. Now the result follows from Lemma 4, since $\frac{1}{\varepsilon}\log\frac{1}{\varphi-\varepsilon} > \frac{1}{\varepsilon}\log\frac{1}{\varphi}$. □

[5]A similar proof appears in [40] but theirs gives a slightly weaker bound.

We now use the same idea as in the proof of Theorem 9 to prove an $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ space complexity lower bound for the ε-Maximum problem.

Theorem 10. *Suppose the size of universe n is at least $\frac{1}{\varepsilon^2}$. Any randomized one pass ε-*Maximum *algorithm with success probability at least $(1-\delta)$ must use $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ bits of space, for constant $\delta \in (0,1)$.*

Proof. Consider the Indexing$_{1/\varepsilon,1/\varepsilon}$ problem where Alice is given a string $x = (x_1, x_2, \cdots, x_{1/\varepsilon}) \in [1/\varepsilon]^{1/\varepsilon}$ and Bob is given an index $i \in [1/\varepsilon]$. The stream we generate is over $[1/\varepsilon] \times [1/\varepsilon] \subseteq \mathcal{U}$ (this is possible since $|\mathcal{U}| \geqslant \frac{1}{\varepsilon^2}$). Alice generates a stream of length $m/2$ in such a way that the frequency of every item in $\{(x_j, j) : j \in [1/\varepsilon]\}$ is at least $\lfloor \varepsilon m/2 \rfloor$ and the frequency of any other item is 0. Alice now sends the memory content of the algorithm to Bob. Bob resumes the run of the algorithm by generating another stream of length $m/2$ in such a way that the frequency of every item in $\{(j, i) : j \in [1/\varepsilon]\}$ is at least $\lfloor \varepsilon m/2 \rfloor$ and the frequency of any other item is 0. The frequency of the item (x_i, i) is at least $\lfloor \varepsilon m \rfloor$ where as the frequency of every other item is at most $\lfloor \varepsilon m/2 \rfloor$. Hence the $\varepsilon/5$-Maximum algorithm must output (x_i, i) with probability at least $(1-\delta)$. Now the result follows from Lemma 4. $\square$

For ε-Minimum, we prove a space complexity lower bound of $\Omega(1/\varepsilon)$ bits.

Theorem 11. *Suppose the universe size n is at least $1/\varepsilon$. Then any randomized one pass ε-*Minimum *algorithm must use $\Omega(1/\varepsilon)$ bits of space.*

Proof. We reduce from Indexing$_{2,5/\varepsilon}$ to ε-Minimum thereby proving the result. Let the inputs to Alice and Bob in Indexing$_{2,5/\varepsilon}$ be $(x_1, \ldots, x_{5/\varepsilon}) \in \{0,1\}^{5/\varepsilon}$ and an index $i \in [5/\varepsilon]$ respectively. Alice and Bob generate a stream $\mathcal{S}$ over the universe $[(5/\varepsilon)+1]$. Alice puts two copies of item j in $\mathcal{S}$ for every $j \in \mathcal{U}$ with $x_j = 1$ and runs the ε-Minimum algorithm. Alice now sends the memory content of the algorithm to Bob. Bob resumes the run of the algorithm by putting two copies of every item in $\mathcal{U} \setminus \{i, (5/\varepsilon)+1\}$ in the stream $\mathcal{S}$. Bob also puts one copy of $(5/\varepsilon)+1$ in $\mathcal{S}$. Suppose the size of the support of $(x_1, \ldots, x_{5/\varepsilon})$ be ℓ. Since $1/(2\ell+(2/\varepsilon)-1) > \varepsilon/5$, we have the following. If $x_i = 0$, then the ε-Minimum algorithm must output i with probability at least $(1-\delta)$. If $x_i = 1$, then the ε-Minimum algorithm must output $(5/\varepsilon)+1$ with probability at least $(1-\delta)$. Now the result follows from Lemma 4. $\square$

We show next a $\Omega(n\log(1/\varepsilon))$ bits space complexity lower bound for ε-Borda.

Theorem 12. *Any one pass algorithm for ε-*Borda *must use $\Omega(n\log(1/\varepsilon))$ bits of space.*

Proof. We reduce ε-Perm to ε-Borda. Suppose Alice has a permutation σ over $[n]$ and Bob has an index $i \in [n]$. The item set of our reduced election is $\mathcal{U} = [n] \sqcup \mathcal{D}$, where $\mathcal{D} = \{d_1, d_2, \ldots, d_{2n}\}$. Alice generates a vote $\mathfrak{v}$ over the item set $\mathcal{U}$ from σ as follows. The vote $\mathfrak{v}$ is $\mathcal{B}_1 \succ \mathcal{B}_2 \succ \cdots \succ \mathcal{B}_{1/\varepsilon}$ where $\mathcal{B}_j$ for $j = 1, \ldots, 1/\varepsilon$ is defined as follows.

$$\begin{aligned}\mathcal{B}_j = {} & d_{(j-1)2\varepsilon n+1} \succ d_{(j-1)2\varepsilon n+2} \succ \cdots \succ d_{(2j-1)\varepsilon n} \\ & \succ \sigma_{j\varepsilon n+1} \succ \cdots \succ \sigma_{(j+1)\varepsilon n} \succ d_{(2j-1)\varepsilon+1} \succ \cdots \succ d_{2j\varepsilon n}\end{aligned}$$

Alice runs the ε-Borda algorithm with the vote $\mathfrak{v}$ and sends the memory content to Bob. Let $\mathcal{D}_{-i} = \mathcal{D} \setminus \{i\}$, $\overrightarrow{\mathcal{D}_{-i}}$ be an arbitrary but fixed ordering of the items in $\mathcal{D}_{-i}$, and $\overleftarrow{\mathcal{D}_{-i}}$ be the reverse ordering of $\overrightarrow{\mathcal{D}_{-i}}$. Bob resumes the algorithm by generating two votes each of the form $i \succ \overrightarrow{\mathcal{D}_{-i}}$ and $i \succ \overleftarrow{\mathcal{D}_{-i}}$. Let us call the resulting election $\mathcal{E}$. The number of votes m in $\mathcal{E}$ is 5. The Borda score of the item i is at least $12n$. The Borda score of every item $x \in \mathcal{U}$ is at most $9n$. Hence for $\varepsilon < 1/15$, the ε-Borda algorithm must output the item i. Moreover, it follows from the construction of $\mathfrak{v}$ that an εmn additive approximation of the Borda score of the item i reveals the block where i belongs in the ε-Perm instance. $\square$

We next give a nearly-tight lower bound for the ε-maximin problem.

Theorem 13. *Any one-pass algorithm for ε-*maximin *requires $\Omega(n/\varepsilon^2)$ memory bits of storage.*

Proof. We reduce from Indexing. Let $\gamma = 1/\varepsilon^2$. Suppose Alice has a string y of length $(n-\gamma)\cdot\gamma$, partitioned into $n-\gamma$ blocks of length γ each. Bob has an index $\ell = i + (j-\gamma-1)\cdot\gamma$ where $i \in [\gamma], j \in \{\gamma+1, \ldots, n\}$. The Indexing problem is to return y_ℓ for which there is a $\Omega(|y|) = \Omega(n/\varepsilon^2)$ lower bound (Lemma 4).

The initial part of the reduction follows the construction in the proof of Theorem 6 in [69], which we encapsulate in the following lemma.

Lemma 7 (Theorem 6 in [69]). *Given y, Alice can construct a matrix $P \in \{0,1\}^{n\times\gamma}$ using public randomness, such that if P^i and P^j are the i'th and j'th rows of P respectively, then with probability at least $2/3$, $\Delta(P^i, P^j) \geqslant \frac{\gamma}{2} + \sqrt{\gamma}$ if $y_\ell = 1$ and $\Delta(a,b) \leqslant \frac{\gamma}{2} - \sqrt{\gamma}$ if $y_\ell = 0$.*

Let Alice construct P according to Lemma 7 and then adjoin the bitwise complement of the matrix P below P to form the matrix $P' \in \{0,1\}^{2n\times\gamma}$; note that each column of P' has exactly n 1's and n 0's. Now, we interpret each row of P as a candidate and each column of P as a vote in the following way: for each $v \in [\gamma]$, vote v has the candidates in $\{c : P'_{c,v} = 1\}$ in ascending order in the top n positions and the rest of the candidates in ascending order in the bottom n positions. Alice inserts these γ votes into the stream and sends the state of the ε-Maximin algorithm to Bob as well as the Hamming weight of each row in P'. Bob inserts γ more votes, in each of which candidate i comes first, candidate j comes second, and the rest of the $2n-2$ candidates are in arbitrary order.

Note that because of Bob's votes, the maximin score of j is the number of votes among the ones casted by Alice in which j defeats i. Since $i < j$, in those columns v where $P_{i,v} = P_{j,v}$, candidate i beats candidate j. Thus, the set of votes in which j defeats i is $\{v \mid P_{i,v} = 0, P_{j,v} = 1\}$. The size of this set is $\frac{1}{2}\left(\Delta(P^i, P^j) + |P^j| - |P^i|\right)$. Therefore, if Bob can estimate the maximin score of j upto $\sqrt{\gamma}/4$ additive error, he can find $\Delta(P^i, P^j)$ upto $\sqrt{\gamma}/2$ additive error as Bob knows $|P^i|$ and $|P^j|$. This is enough, by Lemma 7, to solve the Indexing problem with probability at least $2/3$. $\square$

Finally, we show a space complexity lower bound that depends on the length of the stream m.

Theorem 14. *Any one pass algorithm for ε-*Heavy hitters*, ε-*Minimum*, ε-*Borda*, and ε-*maximin *must use* $\Omega(\log\log m)$ *memory bits, even if the stream is over a universe of size* 2*, for every* $\varepsilon < \frac{1}{4}$.

Proof. It is enough to prove the result only for ε-Heavy hitters since the other three problems reduce to ε-Heavy hitters for a universe of size 2. Suppose we have a randomized one pass ε-Heavy hitters algorithm which uses $s(m)$ bits of space. Using this algorithm, we will show a communication protocol for the Greater-than$_m$ problem whose communication complexity is $s(2^m)$ thereby proving the statement. The universal set is $\mathcal{U} = \{0, 1\}$. Alice generates a stream of 2^x many copies of the item 1. Alice now sends the memory content of the algorithm. Bob resumes the run of the algorithm by generating a stream of 2^y many copies of the item 0. If $x > y$, then the item 1 is the only ε-winner; whereas if $x < y$, then the item 0 is the only ε-winner. □

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.

[3] A. Arasu, S. Babu, and J. Widom. CQL: A language for continuous queries over streams and relations. In *Proc. 9th International Workshop on Database Programming Languages DBPL*, pages 1–19, 2003.

[4] J. A. Aslam and M. Montague. Models for metasearch. In *Proc. 24th Annual international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 276–284. ACM, 2001.

[5] R. Berinde, P. Indyk, G. Cormode, and M. J. Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Trans. Database Syst*, 35(4):26, 2010.

[6] K. S. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 359–370, 1999.

[7] D. K. Blandford and G. E. Blelloch. Compact dictionaries for variable-length keys and data with applications. *ACM Trans. Algor.*, 4(2):17, 2008.

[8] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proc. 2nd International Conference on Mobile Data Management, MDM*, pages 3–14, 2001.

[9] F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. Procaccia. Handbook of computational social choice, 2015.

[10] V. Braverman, S. R. Chestnut, N. Ivkin, J. Nelson, Z. Wang, and D. P. Woodruff. BPTree: an ℓ_2 heavy hitters algorithm using constant memory. 2016. arXiv:1603.00759.

[11] V. Braverman, S. R. Chestnut, N. Ivkin, and D. P. Woodruff. Beating countsketch for heavy hitters in insertion streams. Technical report, 2016. arXiv:1511.00661. To appear in STOC '16.

[12] E. J. Candes, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Commun. Pur. Appl. Math.*, 59:1207–1223, 2006.

[13] I. Caragiannis and A. D. Procaccia. Voting almost maximizes social welfare despite limited communication. *Artif. Intell.*, 175(9–10):1655 – 1671, 2011.

[14] B. Carterette and D. Petkova. Learning a ranking from pairwise preferences. In *Proc. 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 629–630. ACM, 2006.

[15] Y. Chabchoub, C. Fricker, and H. Mohamed. Analysis of a bloom filter algorithm via the supermarket model. In *Proc. 21st International Teletraffic Congress, ITC*, pages 1–8, 2009.

[16] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

[17] V. Conitzer and T. Sandholm. Communication complexity of common voting rules. In *Proc. 6th ACM Conference on Electronic Commerce*, pages 78–87. ACM, 2005.

[18] G. Cormode. Sketch techniques for massive data. In G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, editors, *Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches*, volume 4 of *Foundations and Trends in Databases*, pages 1–294. Now Publishers Inc., Hanover, MA, USA, Jan. 2012.

[19] G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.

[20] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding hierarchical heavy hitters in streaming data. *ACM Trans. Knowl. Discov. Data*, 1(4), 2008.

[21] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[22] T. M. Cover and J. A. Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

[23] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *Proc. 10th Annual European Symposium on Algorithms*, pages 348–360. Springer, 2002.

[24] P. Dey and A. Bhattacharyya. Sample complexity for winner prediction in elections. In *Proc. 14th International Conference on Autonomous Systems and Multiagent Systems (AAMAS-15)*, 2015.

[25] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A reliable randomized algorithm for the closest-pair problem. *J. Algorithms*, 25(1):19–51, 1997.

[26] A. Dvoretzky, J. Kiefer, and J. Wolfowitz. Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator. *Ann. Math. Stat.*, 27(3):642 – 669, 1956.

[27] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *Theor. Comput. Syst.*, 21(3):270–313, 2003.

[28] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *Proc. 24rd International Conference on Very Large Data Bases*, pages 299–310, 1998.

[29] P. Flajolet. Approximate counting: a detailed analysis. *BIT Numerical Mathematics*, 25(1):113–134, 1985.

[30] A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin. One sketch for all: fast algorithms for compressed sensing. In *Proc. 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 237–246, 2007.

[31] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures. In *Proc. 2001 ACM SIGMOD International Conference on Management of Data,*, pages 1–12, 2001.

[32] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2000.

[33] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.

[34] J. Hershberger, N. Shrivastava, S. Suri, and C. D. Tóth. Space complexity of hierarchical heavy hitters in multi-dimensional data streams. In *Proc. Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 338–347, 2005.

[35] C. Hidber. Online association rule mining. In *SIGMOD 1999, Proc. ACM SIGMOD International Conference on Management of Data*, pages 145–156, 1999.

[36] T. K. Ho, J. J. Hull, and S. N. Srihari. Decision combination in multiple classifier systems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(1):66–75, 1994.

[37] A. Jiang, L. S. Marcolino, A. D. Procaccia, T. Sandholm, N. Shah, and M. Tambe. Diverse randomized agents vote to win. In *Proc. Annual Conference on Neural Information Processing Systems*, pages 2573–2581, 2014.

[38] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28(1):51–55, 2003.

[39] P. Kellner, J. Twyman, and A. Wells. Polling voting intentions. In *Political Communication in Britain*, pages 94–108. Springer, 2011.

[40] I. Kremer, N. Nisan, and D. Ron. On randomized one-round communication complexity. *Comput. Complex.*, 8(1):21–49, 1999.

[41] A. Kumar and J. J. Xu. Sketch guided sampling - using on-line estimates of flow size for adaptive data collection. In *Proc. 25th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, 2006.

[42] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, New York, NY, USA, 1997.

[43] K. G. Larsen, J. Nelson, H. L. Nguyen, and M. Thorup. Heavy hitters via cluster-preserving clustering. arXiv:1604.01357, Apr. 2016.

[44] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen. Introduction to algorithms. *The MIT press*, 5:2–2, 2001.

[45] H. Li. Learning to rank for information retrieval and natural language processing. In *Synthesis Lectures on Human Language Technologies*, volume 7, pages 1–121. Morgan & Claypool Publishers, 2014.

[46] A. Lumini and L. Nanni. Detector of image orientation based on borda count. *Pattern Recogn. Lett.*, 27(3):180–186, 2006.

[47] N. Mamoulis, M. L. Yiu, K. H. Cheng, and D. W. Cheung. Efficient top-k aggregation of ranked inputs. *ACM Trans. Database Syst*, 32(3):19, 2007.

[48] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. 28th International Conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.

[49] A. Mao, A. D. Procaccia, and Y. Chen. Social Choice for Human Computation. In *Proc. Fourth Workshop on Human Computation (HCOMP-12)*, 2012.

[50] A. Mao, A. D. Procaccia, and Y. Chen. Better Human Computation Through Principled Voting. In *Proc. 27th Conference on Artificial Intelligence (AAAI'13)*, 2013.

[51] A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2):319–362, 2004.

[52] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proc. 10th International Conference on Database Theory*, ICDT'05, pages 398–412, Berlin, Heidelberg, 2005. Springer-Verlag.

[53] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *J. Comput. Syst. Sci.*, 57(1):37–49, 1998.

[54] J. Misra and D. Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.

[55] M. M. Monwar and M. L. Gavrilova. Multimodal biometric system using rank-level fusion approach. *IEEE Trans. Syst. Man. Cybern. B, Cybern.*, 39(4):867–878, 2009.

[56] J. S. Moore. J. Algorithm, June 1981. p. 208–209.

[57] R. Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, 1978.

[58] D. Mullen and B. Roth. *Decision making: Its logic and practice.* Savage, MD: Rowman and Littlefield Publishers, Inc., 1991.

[59] S. Muthukrishnan. *Data streams: Algorithms and applications.* Now Publishers Inc, 2005.

[60] J. Nelson. Sketching and streaming algorithms for processing massive data. *XRDS: Crossroads, The ACM Magazine for Students*, 19(1):14–19, 2012.

[61] R. Nuray and F. Can. Automatic ranking of information retrieval systems using data fusion. *Inf. Process Manag.*, 42(3):595–614, 2006.

[62] A. Prasad, H. Pareek, and P. Ravikumar. Distributional rank aggregation, and an axiomatic analysis. In *Proc. 32nd International Conference on Machine Learning (ICML-15)*, pages 2104–2112, 2015.

[63] P. Resnick and H. R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, 1997.

[64] A. Savasere, E. Omiecinski, and S. B. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. 21th International Conference on Very Large Data Bases*, pages 432–444, 1995.

[65] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proc. 2nd International Conference on Embedded Networked Sensor Systems*, pages 239–249, 2004.

[66] D. Smirnov. Shannon's information methods for lower bounds for probabilistic communication complexity. Master's thesis, Moscow University, 1988.

[67] X. Sun and D. P. Woodruff. Tight bounds for graph problems in insertion streams. In *Proc. 18th. International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2015)*, pages 435–448, 2015.

[68] H. Toivonen. Sampling large databases for association rules. In *Proc. 22th International Conference on Very Large Data Bases*, pages 134–145, 1996.

[69] D. Van Gucht, R. Williams, D. P. Woodruff, and Q. Zhang. The communication complexity of distributed set-joins with applications to matrix multiplication. In *Proc. 34th ACM Symposium on Principles of Database Systems*, pages 199–212. ACM, 2015.

[70] M. N. Volkovs and R. S. Zemel. New learning methods for supervised and unsupervised preference aggregation. *J. Mach. Learn. Res.*, 15(1):1135–1176, 2014.

[71] G. Wang and F. H. Lochovsky. Feature selection with conditional mutual information maximin in text categorization. In *Proc. 13th ACM International Conference on Information and Knowledge Management*, pages 342–349. ACM, 2004.

[72] D. P. Woodruff. New Algorithms for Heavy Hitters in Data Streams. 2016. arXiv 1603.01733. Appeared in ICDT '16.

[73] L. Xia. Computing the margin of victory for various voting rules. In *Proc. 13th ACM Conference on Electronic Commerce*, pages 982–999. ACM, 2012.

[74] A. C.-C. Yao. Some complexity questions related to distributive computing (preliminary report). In *Proc. 11th annual ACM Symposium on Theory of computing*, pages 209–213. ACM, 1979.

APPENDIX

Appendix A Information Theory Facts

For a discrete random variable X with possible values $\{x_1, x_2, \ldots, x_n\}$, the Shannon entropy of X is defined as $H(X) = -\sum_{i=1}^{n} \Pr(X = x_i) \log_2 \Pr(X = x_i)$. Let $H_b(p) = -p \log_2 p - (1-p)\log_2(1-p)$ denote the binary entropy function when $p \in (0,1)$. For two random variables X and Y with possible values $\{x_1, x_2, \ldots, x_n\}$ and $\{y_1, y_2, \ldots, y_m\}$, respectively, the conditional entropy of X given Y is defined as $H(X \mid Y) = \sum_{i,j} \Pr(X = x_i, Y = y_j) \log_2 \frac{\Pr(Y=y_j)}{\Pr(X=x_i, Y=y_j)}$. Let $I(X;Y) = H(X) - H(X \mid Y) = H(Y) - H(Y \mid X)$ denote the mutual information between two random variables X, Y. Let $I(X;Y \mid Z)$ denote the mutual information between two random variables X, Y conditioned on Z, i.e., $I(X;Y \mid Z) = H(X \mid Z) - H(X \mid Y, Z)$. The following summarizes several basic properties of entropy and mutual information.

Proposition 1. *Let X, Y, Z, W be random variables.*

1. *If X takes value in $\{1, 2, \ldots, m\}$, then $H(X) \in [0, \log m]$.*
2. *$H(X) \geqslant H(X \mid Y)$ and $I(X;Y) = H(X) - H(X \mid Y) \geqslant 0$.*
3. *If X and Z are independent, then we have $I(X;Y \mid Z) \geqslant I(X;Y)$. Similarly, if X, Z are independent given W, then $I(X;Y \mid Z, W) \geqslant I(X;Y \mid W)$.*
4. *(Chain rule of mutual information) $I(X,Y;Z) = I(X;Z) + I(Y;Z \mid X)$. More generally, for any random variables $X_1, X_2, \ldots, X_n, Y$, $I(X_1, \ldots, X_n; Y) = \sum_{i=1}^{n} I(X_i; Y \mid X_1, \ldots, X_{i-1})$. Thus, $I(X,Y;Z \mid W) \geqslant I(X;Z \mid W)$.*
5. *(Fano's inequality) Let X be a random variable chosen from domain $\mathcal{X}$ according to distribution μ_X, and Y be a random variable chosen from domain $\mathcal{Y}$ according to distribution μ_Y. For any reconstruction function $g : \mathcal{Y} \to \mathcal{X}$ with error δ_g,*
$$H_b(\delta_g) + \delta_g \log(|\mathcal{X}| - 1) \geqslant H(X \mid Y).$$

We refer readers to [22] for a nice introduction to information theory.

Appendix B

We now present the missing proofs.

Lemma 1. *Suppose m is a power of two*[6]*. Then there is an algorithm $\mathcal{A}$ for choosing an item with probability $1/m$ that has space complexity of $O(\log\log m)$ bits and time complexity of $O(1)$ in the unit-cost RAM model.*

Proof. We generate a $\log_2 m$ bits integer C uniformly at random and choose the item, say x, only if $C = 0$. □

We remark that the algorithm in Lemma 1 has optimal space complexity as shown in Proposition 2 below which may be of independent interest.

Proposition 2. *[⋆] Any algorithm that chooses an item from a set of size n with probability p for $0 < p \leqslant \frac{1}{n}$, in unit cost RAM model must use $\Omega(\log\log m)$ bits of memory.*

Proof. The algorithm generates t bits uniformly at random (the number of bits it generates uniformly at random may also depend on the outcome of the previous random bits) and finally picks an item from the say x. Consider a run $\mathcal{R}$ of the algorithm where it chooses the item x with *smallest number of random bits getting generated*; say it generates t random bits in this run $\mathcal{R}$. This means that in any other run of the algorithm where the item x is chosen, the algorithm must generate at least t many random bits. Let the random bits generated in $\mathcal{R}$ be $r_1, \cdots, r_t$. Let s_i be the memory content of the algorithm immediately after it generates i^{th} random bit, for $i \in [t]$, in the run $\mathcal{R}$. First notice that if $t < \log_2 n$, then the probability with which the item x is chosen is more than $\frac{1}{n}$, which would be a contradiction. Hence, $t \geqslant \log_2 n$. Now we claim that all the s_i's must be different. Indeed otherwise, let us assume $s_i = s_j$ for some $i < j$. Then the algorithm chooses the item x after generating $t - (j - i)$ many random bits (which is strictly less than t) when the random bits being generated are $r_1, \cdots, r_i, r_{j+1}, \cdots, r_t$. This contradicts the assumption that the run $\mathcal{R}$ we started with chooses the item x with smallest number of random bits generated. □

[6]In all our algorithms, whenever we pick an item with probability $p > 0$, we can assume, without loss of generality, that $1/p$ is a power of two. If not, then we replace p with p' where $1/p'$ is the largest power of two less than $1/p$. This does not affect correctness and performance of our algorithms.

Lemma 2. *For $S \subseteq A$, $\delta \in (0,1)$, and universal family of hash functions $\mathcal{H} = \{h | h : A \to [\lceil |S|^2/\delta \rceil]\}$:*

$$\Pr_{h \in_{unif} \mathcal{H}} [\exists i \neq j \in S, h(i) = h(j)] \leqslant \delta$$

Proof. For every $i \neq j \in S$, since $\mathcal{H}$ is a universal family of hash functions, we have

$$\Pr_{h \text{ picked uniformly at random from } \mathcal{H}} \{h(i) = h(j)\} \leqslant 1/\lceil |S|^2/\delta \rceil$$

Now we apply a union bound to get

$$\Pr_{h \text{ picked uniformly at random from } \mathcal{H}} \{\exists i \neq j \in S, h(i) = h(j)\}$$
$$\leqslant \quad |S|^2/\lceil |S|^2/\delta \rceil$$
$$\leqslant \quad \delta$$

□

Proof of Theorem 2. Pseudocode appears in Algorithm 2. Note that the numerical constants are chosen for convenience of analysis and have not been optimized. Also, for the sake of simplicity, the pseudocode does not have the optimal reporting time, but it can be modified to achieve this; see the end of this proof for details.

By standard Chernoff bounds, with probability at least 99/100, the length of the sampled stream $\ell/10 \leqslant s \leqslant 10\ell$. For $x \in [n]$, let $f_{\mathrm{samp}}(x)$ be the frequency of x in the sampled stream. By Lemma 3, with probability at least 9/10, for all $x \in [n]$:

$$\left| \frac{f_{\mathrm{samp}}(x)}{s} - \frac{f(x)}{m} \right| \leqslant \frac{\varepsilon}{4}$$

Now, fix $j \in [10 \log \varphi^{-1}]$ and $x \in [n]$. Let $i = h_j(x)$ and $f_i = \sum_{y: h_j(y) = h_j(x)} f_{\mathrm{samp}}(y)$. Then, for a random $h_j \in \mathcal{H}$, the expected value of $\frac{f_i}{s} - \frac{f_{\mathrm{samp}}(x)}{s}$ is $\frac{\varepsilon}{100}$, since $\mathcal{H}$ is a universal mapping to a space of size $100\varepsilon^{-1}$. Hence, using Markov's inequality and the above:

$$\begin{aligned} &\Pr\left[\left|\frac{f(x)}{m} - \frac{f_i}{s}\right| \geqslant \frac{\varepsilon}{2}\right] \\ &\leqslant \Pr\left[\left|\frac{f(x)}{m} - \frac{f_{\mathrm{samp}}}{s}\right| \geqslant \frac{\varepsilon}{4}\right] + \Pr\left[\left|\frac{f_{\mathrm{samp}}(x)}{m} - \frac{f_i}{s}\right| \geqslant \frac{\varepsilon}{4}\right] \\ &< \frac{1}{10} + \frac{1}{25} < \frac{3}{20} \end{aligned} \tag{1}$$

In Lemma 8 below, we show that for each $j \in [200 \log(12\varphi^{-1})]$, with error probability at most 3/10, $\hat{f}_j(x)$ (in line 23) estimates f_i with additive error at most $5000\varepsilon^{-1}$, hence estimating $\frac{f_i}{s}$ with additive error at most $\frac{\varepsilon}{2}$. Taking the median over $200 \log(12\varphi^{-1})$ repetitions (line 24) makes the error probability go down to $\frac{\varphi}{6}$ using standard Chernoff bounds. Hence, by the union bound, with probability at least 2/3, for each of the $2/\varphi$ keys x with nonzero values in $\mathcal{T}_1$, we have an estimate of $\frac{f(x)}{m}$ within additive error ε, thus showing correctness.

Lemma 8. *Fix $x \in [n]$ and $j \in [200 \log 12\varphi^{-1}]$, and let $i = h_j(x)$. Then, $\Pr[|\hat{f}_j(x) - f_i| > 5000\varepsilon^{-1}] \leqslant 3/10$, where $\hat{f}_j$ is the quantity computed in line 23.*

Proof. Index the sampled stream elements $1, 2, \ldots, s$, and for $b \in [s]$, let $f_i(b)$ be the frequency of items with hash id i restricted to the first b elements of the sampled stream. Let $\bar{f}_i(b)$ denote the value of $\mathcal{T}_2[i,j] \cdot \varepsilon^{-1}$ after the procedure Insert has been called for the first b items of the sampled stream.

Claim 1. *With probability at least 9/10, for all $b \in [s]$ such that $f_i(b) \geqslant 100\varepsilon^{-1}$, $\bar{f}_i(b)$ is within a factor of 4 of $f_i(b)$.*

Proof. Fix $b \in [s]$. Note that $\mathbb{E}\left[\bar{f}_i(b)\right] = f_i(b)$ as $\mathcal{T}_2$ is incremented with rate ε. $\mathsf{Var}[\bar{f}_i(b)] \leqslant f_i/\varepsilon$, and so by Chebyshev's inequality:

$$\Pr[|\bar{f}_i(b) - f_i(b)| > f_i(b)/2] < \frac{4}{f_i(b)\varepsilon}$$

We now break the stream into chunks, apply this inequality to each chunk and then take a union bound to conclude. Namely, for any integer $t \geqslant 0$, define b_t to be the first b such that $100\varepsilon^{-1}2^t \leqslant f_i(b) < 100\varepsilon^{-1}2^{t+1}$ if such a b exists. Then:

$$\begin{aligned} \Pr[\exists t \geqslant 0 : |\bar{f}_i(b_t) - f_i(b_t)| > f_i(b_t)/2] &< \sum_t \frac{4}{100 \cdot 2^{t-1}} \\ &< \frac{1}{10} \end{aligned}$$

So, with probability at least 9/10, every $\bar{f}_i(b_t)$ and $f_i(b_t)$ are within a factor of 2 of each other. Since for every $b \geqslant b_0$, $f_i(b)$ is within a factor of 2 from some $f_i(b_t)$, the claim follows. □

Assume the event in Claim 1 henceforth. Now, we are ready to analyze $\mathcal{T}_3$ and in particular, $\hat{f}_j(x)$. First of all, observe that if $t < 0$ in line 15, at some position b in the stream, then $\mathcal{T}_2[i,j]$ at that time must be at most 1000, and so by standard Markov and Chernoff bounds, with probability at least 0.85,

$$f_i(b) \begin{cases} < 4000\varepsilon^{-1}, & \text{if } t < 0 \\ > 100\varepsilon^{-1}, & \text{if } t \geqslant 0 \end{cases} \tag{2}$$

Assume this event. Then, $f_i - 4000\varepsilon^{-1} \leqslant \mathbb{E}\left[\hat{f}_j(x)\right] \leqslant f_i$.

Claim 2.

$$\mathsf{Var}(\hat{f}_j(x)) \leqslant 20000\varepsilon^{-2}$$

Proof. If the stream element at position b causes an increment in $\mathcal{T}_3$ with probability $\varepsilon 2^t$ (in line 17), then $1000 \cdot 2^{t/2} \leqslant \mathcal{T}_2[i,j] \leqslant 1000 \cdot 2^{(t+1)/2}$, and so, $\bar{f}_i(b) \leqslant 1000\varepsilon^{-1}2^{(t+1)/2}$. This must be the case for the highest $b = \bar{b}_t$ at which the count for i in $\mathcal{T}_3$ increments at the t'th slot. The number of such occurrences of i is at most $f_i(\bar{b}_t) \leqslant 4\bar{f}_i(\bar{b}_t) \leqslant 4000\varepsilon^{-1}2^{(t+1)/2}$ by Claim 1 (which can be applied since $f_i(b) > 100\varepsilon^{-1}$ by Equation 2). So:

$$\mathsf{Var}[\hat{f}_j(x)] \leqslant \sum_{t \geqslant 0} \frac{f_i(\bar{b}_t)}{\varepsilon 2^t} \leqslant \sum_{t \geqslant 0} \frac{4000}{\varepsilon^2} 2^{-t/3} \leqslant 20000\varepsilon^{-2}$$

Elements inserted with probability 1 obviously do not contribute to the variance. □

So, conditioning on the events mentioned, the probability that $\hat{f}_j(x)$ deviates from f_i by more than $5000\varepsilon^{-1}$ is at most 1/50. Removing all the conditioning yields what we wanted:

$$\Pr[|\hat{f}_j(x) - f_i| > 5000\varepsilon^{-1}] \leqslant \frac{1}{50} + \frac{3}{20} + \frac{1}{10} \leqslant 0.3$$

We next bound the space complexity.

Claim 3. *With probability at least $2/3$, Algorithm 2 uses $O(\varepsilon^{-1}\log\varphi^{-1}+\varphi^{-1}\log n+\log\log m)$ bits of storage, if $n=\omega(\varepsilon^{-1})$.*

Proof. The expected length of the sampled stream is $\ell = O(\varepsilon^{-2})$. So, the number of bits stored in $\mathcal{T}_1$ is $O(\varphi^{-1}\log n)$. For $\mathcal{T}_2$, note that in lines 13-15, for any given j, $\mathcal{T}_2$ is storing a total of $\varepsilon\ell = O(\varepsilon^{-1})$ elements in expectation. So, for $k \geqslant 0$, there can be at most $O((\varepsilon 2^k)^{-1})$ hashed id's with counts between 2^k and 2^{k+1}. Summing over all k's and accounting for the empty cells gives $O(\varepsilon^{-1})$ bits of storage, and so the total space requirement of $\mathcal{T}_2$ is $O(\varepsilon^{-1}\log\varphi^{-1})$.
.

The probability that a hashed id i gets counted in table $\mathcal{T}_3$ is at most $10^{-6}\varepsilon^3\bar{f}_i^2(s)$ from line 15 and our definition of $\bar{f}_i$ above. Moreover, from Claim 1, we have that this is at most $16\cdot 10^{-6}\varepsilon^3 f_i^2(s)$ if $f_i > 100\varepsilon^{-1}$. Therefore, if $f_i = 2^k\cdot 100\varepsilon^{-1}$ with $k \geqslant 0$, then the expected value of a cell in $\mathcal{T}_3$ with first coordinate i is at most $1600\cdot 2^{2k}\varepsilon = 2^{O(k)}$. Taking into account that there are at most $O((\varepsilon 2^k)^{-1})$ many such id's i and that the number of epochs t associated with such an i is at most $\log(16\cdot 10^{-6}\varepsilon^2 f_i^2) = O(\log(\varepsilon f_i)) = O(k)$ (from line 15), we get that the total space required for $\mathcal{T}_3$ is:

$$\sum_{j=1}^{O(\log\varphi^{-1})}\left(O(\varepsilon^{-1})+\sum_{k=0}^{\infty}O((\varepsilon 2^k)^{-1})\cdot O(k)\cdot O(k)\right)$$
$$= O(\varepsilon^{-1}\log\varphi^{-1})$$

where the first $O(\varepsilon^{-1})$ term inside the summation is for the i's with $f_i < 100\varepsilon^{-1}$. Since we have an expected space bound, we obtain a worst-case space bound with error probability $1/3$ by a Markov bound.

The space required for sampling is an additional $O(\log\log m)$, using Lemma 1. □

We note that the space bound can be made worst case by aborting the algorithm if it tries to use more space.

The only remaining aspect of Theorem 2 is the time complexity. As observed in Section 3.1, the update time can be made $O(1)$ per insertion under the standard assumption of the stream being sufficiently long. The reporting time can also be made linear in the output by changing the bookkeeping a bit. Instead of computing $\hat{f}_j$ and $\hat{f}$ at reporting time, we can maintain them after every insertion. Although this apparently makes INSERT costlier, this is not true in fact because we can spread the cost over future stream insertions. The space complexity grows by a constant factor.

Theorem 4. *Assume the length of the stream is known beforehand. Then there is a randomized one-pass algorithm $\mathcal{A}$ for the ε-Minimum problem which succeeds with probability at least $1-\delta$ using $O\left((1/\varepsilon)\log\log(1/\varepsilon\delta)+\log\log m\right)$ bits of space. Moreover, the algorithm $\mathcal{A}$ has an update time of $O(1)$.*

Proof of Theorem 4. The pseudocode of our ε-Minimum algorithm is in Algorithm 3. If the size of the universe $|\mathcal{U}|$ is at least $1/((1-\delta)\varepsilon)$, then we return an item x chosen from $\mathcal{U}$ uniformly at random. Note that there can be at most $1/\varepsilon$ many items with frequency at least εm. Hence every item x among other remaining $\delta/((1-\delta)\varepsilon)$ many items has frequency less than εm and thus is a correct output of the instance. Thus the probability that we answer correctly is at least $(1-\delta)$. From here on, let us assume $|\mathcal{U}| < 1/((1-\delta)\varepsilon)$.

Now, by the value of p_j, it follows from the proof of Theorem 1 that we can assume $\ell_j < |\mathcal{S}_j| < 11\ell_j$ for $j=1,2,3$ which happens with probability at least $(1-(\delta/3))$. We first show that every item in $\mathcal{U}$ with frequency at least εm is sampled in $\mathcal{S}_1$ with probability at least $(1-(\delta/6))$. For that, let X_i^j be the indicator random variable for the event that the j^{th} sample in $\mathcal{S}_1$ is item i where $i\in\mathcal{U}$ is an item with frequency at least εm. Let $\mathcal{H}\subset\mathcal{U}$ be the set of items with frequencies at least εm. Then we have the following.

$$\Pr[X_i^j = 0] = 1-\varepsilon$$
$$\Rightarrow \Pr[X_i^j = 0\ \forall j\in\mathcal{S}_1] \leqslant (1-\varepsilon)^{\ell_1} \leqslant \exp\{-\varepsilon\ell_1\} = \varepsilon\delta/6$$

Now applying a union bound we get the following.

$$\Pr[\exists i\in\mathcal{H}, X_i^j = 0\ \forall j\in\mathcal{S}_1] \leqslant (1/\varepsilon)\varepsilon\delta/6 \leqslant \delta/6$$

Hence with probability at least $(1-(\delta/3)-(\delta/6)) \geqslant (1-\delta)$, the output at line 17 is correct. Now we show below that if the frequency of any item $x\in\mathcal{U}$ is at most $\varepsilon\ln(6/\delta)/\ln(6/\varepsilon\delta)$, then $x\in\mathcal{S}_1$ with probability at least $(1-(\delta/6))$.

$$\Pr[x\notin\mathcal{S}_1] = (1-\varepsilon\ln(6/\delta)/\ln(6/\varepsilon\delta))^{\ln(6/\varepsilon\delta)/\varepsilon} \leqslant \delta/6$$

Hence from here onwards we assume that the frequency of every item in $\mathcal{U}$ is at least $\varepsilon m\ln(6/\delta)/\ln(6/\varepsilon\delta)$.

If the number of distinct elements is at most $1/(\varepsilon\ln(1/\varepsilon))$, then line 19 outputs the minimum frequency item up to an additive factor of εm due to Chernoff bound (see [24]). Note that we need only $O(\ln(1/((1-\delta)\varepsilon)))$ bits of space for storing ids. Hence $\mathcal{S}_2$ can be stored in space $O((1/\varepsilon\ln(1/\varepsilon))\ln(1/((1-\delta)\varepsilon)\ln\ln(1/\delta)) = O(1/\varepsilon\ln\ln(1/\delta))$.

Now we can assume that the number of distinct elements is at least $1/(\varepsilon\ln(1/\varepsilon))$. Hence if $f(t)$ is the frequency of the item t with minimum frequency, then we have $m\varepsilon/\ln(1/\varepsilon) \leqslant f(t) \leqslant m\varepsilon\ln(1/\varepsilon)$.

Let f_i be the frequency of item $i\in\mathcal{U}$, e_i be the counter value of i in $\mathcal{S}_3$, and $\hat{f}_i = e_i m/\ell_3$. Now again by applying Chernoff bound we have the following for any fixed $i\in\mathcal{U}$.

$$\begin{aligned}\Pr[|f_i-\hat{f}_i| > f_i/\ln^2(1/\varepsilon)] &\leqslant 2\exp\{-\ell_3 f_i/(m\ln^4(1/\varepsilon))\}\\ &\leqslant 2\exp\{-f_i\ln^2(6/\varepsilon\delta)/(\varepsilon m)\}\\ &\leqslant \varepsilon\delta/6.\end{aligned}$$

Now applying a union bound we get the following using the fact that $|\mathcal{U}| \leqslant 1/\varepsilon(1-\delta)$.

$$\Pr[\forall i\in\mathcal{U}, |f_i-\hat{f}_i| \leqslant f_i/\ln^2(1/\varepsilon)] > 1-\delta/6$$

Again by applying Chernoff bound and union bound we get the following.

$$\Pr[\forall i\in\mathcal{U} \text{ with } f_i > 2m\varepsilon\ln(1/\varepsilon), |f_i-\hat{f}_i| \leqslant f_i/2] > 1-\delta/6$$

Hence the items with frequency more than $2m\varepsilon\ln(1/\varepsilon)$ are approximated up to a multiplicative factor of $1/2$ from below in $\mathcal{S}_3$. The counters of these items may be truncated. The other items with frequency at most $2m\varepsilon\ln(1/\varepsilon)$ are be approximated up to $(1\pm 1/\ln^2(1/\varepsilon))$ relative error and thus up to an additive error of $\varepsilon m/3$. The counters of these items would not get truncated. Hence the item with minimum counter value in $\mathcal{S}_3$ is the item with minimum frequency up to an additive εm.

We need $O(\ln(1/\varepsilon\delta))$ bits of space for the bit vector $\mathcal{B}_1$ for the set $\mathcal{S}_1$. We need $O(\ln^2(1/\varepsilon\delta))$ bits of space for the set $\mathcal{S}_2$ and $O((1/\varepsilon)\ln\ln(1/\varepsilon\delta))$ bits of space for the set $\mathcal{S}_3$ (by the choice of truncation threshold). We need an additional $O(\ln\ln m)$ bits of space for sampling using Lemma 1.

Moreover, using the data structure of Section 3.3 of [23] Algorithm 3 can be performed in $O(1)$ time. Alternatively, we may also use the strategy described in Section 3.1 of spreading update operations over several insertions to make the cost per insertion be $O(1)$. $\square$

Theorem 3. *Assume the length of the stream is known beforehand. Then there is a randomized one-pass algorithm $\mathcal{A}$ for the ε-*Maximum *problem which succeeds with probability at least $1-\delta$ using $O\left(\min\{1/\varepsilon, n\}(\log 1/\varepsilon + \log\log 1/\delta) + \log n + \log\log m\right)$ bits of space. Moreover, the algorithm $\mathcal{A}$ has an update time of $O(1)$.*

Proof. Instead of maintaining the table $\mathcal{T}_2$ in Algorithm 1, we just store the actual ID of the item with maximum frequency in the sampled items. $\square$

Theorem 5. *Assume the length of the stream is known beforehand. Then there is a randomized one-pass algorithm $\mathcal{A}$ for (ε,φ)-*List Borda *problem which succeeds with probability at least $1-\delta$ using $O\left(n\left(\log n + \log\frac{1}{\varepsilon} + \log\log\frac{1}{\delta}\right) + \log\log m\right)$ bits of space.*

Proof. Let $\ell = 6\varepsilon^{-2}\log(6n\delta^{-1})$ and $p = 6\ell/m$. On each insertion of a vote v, select v with probability p and store for every $i \in [n]$, the number of candidates that candidate i beats in the vote v. Keep these exact counts in a counter of length n.

Then it follows from the proof of Theorem 1 that $\ell \leqslant |\mathcal{S}| \leqslant 11\ell$ with probability at least $(1-\delta/3)$. Moreover, from a straightforward application of the Chernoff bound (see [24]), it follows that if $\hat{s}(i)$ denotes the Borda score of candidate i restricted to the sampled votes, then:

$$\Pr\left[\forall i \in [n], \left|\frac{m}{|\mathcal{S}|}\hat{s}(i) - s(i)\right| < \varepsilon mn\right] > 1-\delta$$

The space complexity for exactly storing the counts is $O(n\log(n\ell)) = O(n(\log n + \log\varepsilon^{-1} + \log\log\delta^{-1}))$ and the space for sampling the votes is $O(\log\log m)$ by Lemma 1. $\square$

Theorem 6. *Assume the length of the stream is known beforehand. Then there is a randomized one-pass algorithm $\mathcal{A}$ for (ε,φ)-*List maximin *problem which succeeds with probability at least $1-\delta$ using $O\left(n\varepsilon^{-2}\log^2 n + n\varepsilon^{-2}\log n\log\delta^{-1} + \log\log m\right)$ bits of space.*

Proof. Let $\ell = (8/\varepsilon^2)\ln(6n/\delta)$ and $p = 6\ell/m$. We put the current vote in a set $\mathcal{S}$ with probability p. Then it follows from the proof of Theorem 1 that $\ell \leqslant |\mathcal{S}| \leqslant 11\ell$ with probability at least $(1-\delta/3)$. Suppose $|\mathcal{S}| = \ell_1$; let $\mathcal{S} = \{v_i : i \in [\ell_1]\}$ be the set of votes sampled. Let $D_{\mathcal{S}}(x,y)$ be the weight of the edge in the weighted majority graph constructed from the votes in $\mathcal{S}$ for items $x,y \in \mathcal{U}$. Then by the choice of ℓ it follows that $|D_{\mathcal{S}}(x,y)^m/\ell_1 - D_{\mathcal{E}}(x,y)| \leqslant \varepsilon m/2$ for every item $x,y \in \mathcal{U}$ [24]. Note that each vote can be stored in $O(n\ln n)$ bits of space. Hence simply finding $D_{\mathcal{S}}(x,y)$ for every $x,y \in \mathcal{U}$ by storing $\mathcal{S}$ and returning all the items with maximin score at least $\varphi\ell_1$ in $\mathcal{S}$ requires $O\left(n\ln^2 n\ln(1/\delta)/\varepsilon^2 + \ln\ln m\right)$ bits of memory.

Rather than simply storing all the items, we can do as follows. From $\mathcal{S}$, we generate another stream $\bar{\mathcal{S}}$ of elements belonging to the universe $\mathcal{U}\times\mathcal{U}$. Let a vote $v \in \mathcal{S}$ be $c_1 \succ c_2 \succ \cdots \succ c_n$. From v we put (c_j, c_k) in $\bar{\mathcal{S}}$ for every $j < k$. We count the frequency of every item in $\mathcal{S}$ by simply keeping one counter for every items in the universe $\mathcal{U}\times\mathcal{U}$ and output all the items with maximin score at least $\varphi\ell_1$ in $\mathcal{S}$. This requires $O\left(n^2(\ln\ln n + \ln(1/\varepsilon + \ln\ln(1/\delta))\} + \ln\ln m\right)$ bits of space. The correctness of the algorithm follows from the argument above. $\square$

Lemma 5. $\mathcal{R}_\delta^{\textit{1-way}}(\varepsilon - \text{Perm}) = \Omega(n\log(1/\varepsilon))$*, for any constant $\delta < 1/10$.*

Proof. Let us assume σ, the permutation Alice has, is uniformly distributed over the set of all permutations. Let τ_j denotes the block the item j is in for $j \in [n]$, $\tau = (\tau_1,\ldots,\tau_n)$, and $\tau_{<j} = (\tau_1,\ldots,\tau_{j-1})$. Let $M(\tau)$ be Alice's message to Bob, which is a random variable depending on the randomness of σ and the private coin tosses of Alice. Then we have $\mathcal{R}^{1-way}(\varepsilon - \text{Perm}) \geqslant H(M(\tau)) \geqslant I(M(\tau);\tau)$. Hence it is enough to lower bound $I(M(\tau);\tau)$. Then we have the following by chain rule.

$$\begin{aligned} I(M(\tau);\tau) &= \sum_{j=1}^{n} I(M(\tau);\tau_j|\tau_{<j}) \\ &= \sum_{j=1}^{n} H(\tau_j|\tau_<j) - H(\tau_j|M(\tau),\tau_<j) \\ &\geqslant \sum_{j=1}^{n} H(\tau_j|\tau_<j) - \sum_{j=1}^{n} H(\tau_j|M(\tau)) \\ &= H(\tau) - \sum_{j=1}^{n} H(\tau_j|M(\tau)) \end{aligned}$$

The number of ways to partition n items into $1/\varepsilon$ blocks is $n!/((\varepsilon n)!)^{(1/\varepsilon)}$ which is $\Omega((n/e)^n/(\varepsilon n/e)^n)$. Hence we have $H(\tau) = n\log(1/\varepsilon)$. Now we consider $H(\tau_j|M(\tau))$. By the correctness of the algorithm, Fano's inequality, we have $H(\tau_j|M(\tau)) \leqslant H(\delta) + (1/10)\log_2((1/\varepsilon) - 1) \leqslant (1/2)\log(1/\varepsilon)$. Hence we have the following.

$$I(M(\tau);\tau) \geqslant (n/2)\log(1/\varepsilon)$$

$\square$

Lemma 6. $\mathcal{R}_\delta^{\textit{1-way}}(\text{Greater-than}_n) = \Omega(\log n)$*, for every $\delta < 1/4$.*

Proof. We reduce the Augmented-indexing$_{2,\lceil\log n\rceil+1}$ problem to the Greater-than$_n$ problem thereby proving the result. Alice runs the Greater-than$_n$ protocol with its input number whose representation in binary is $a = (x_1x_2\cdots x_{\lceil\log n\rceil}1)_2$. Bob participates in the Greater-than$_n$ protocol with its input number whose representation in binary is $b = (x_1x_2\cdots x_{i-1}1\underbrace{0\cdots0}_{(\lceil\log n\rceil-i+1)\ 0's})_2$. Now $x_i = 1$ if and only if $a > b$. $\square$