# Time Lower Bounds for Nonadaptive Turnstile Streaming Algorithms

Kasper Green Larsen[*]
Aarhus University
larsen@cs.au.dk

Jelani Nelson[†]
Harvard University
minilek@seas.harvard.edu

Huy L. Nguyễn
Simons Institute
hlnguyen@cs.princeton.edu

## ABSTRACT

We say a turnstile streaming algorithm is *non-adaptive* if, during updates, the memory cells written and read depend only on the index being updated and random coins tossed at the beginning of the stream (and not on the memory contents of the algorithm). Memory cells read during *queries* may be decided upon adaptively. All known turnstile streaming algorithms in the literature, except a single recent example for a particular *promise problem* [7], are non-adaptive. In fact, even more specifically, they are all linear sketches.

We prove the first non-trivial update time lower bounds for both randomized and deterministic turnstile streaming algorithms, which hold when the algorithms are non-adaptive. While there has been abundant success in proving space lower bounds, there have been no non-trivial turnstile update time lower bounds. Our lower bounds hold against classically studied problems such as heavy hitters, point query, entropy estimation, and moment estimation. In some cases of deterministic algorithms, our lower bounds nearly match known upper bounds.

## Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General

## General Terms

Theory, Algorithms

## Keywords

streaming; time lower bounds; cell probe model; heavy hitters; point query

[*]Supported by Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation, grant DNRF84.

[†]Supported by NSF grant IIS-1447471 and NSF CAREER award CCF-1350670, ONR grant N00014-14-1-0632, and a Google Faculty Research Award.

## 1. INTRODUCTION

In the turnstile streaming model of computation [36] there is some vector $v \in \mathbb{R}^n$ initialized to $\vec{0}$, and we must provide a data structure that processes coordinate-wise updates to $v$. An update of the form $(i, \Delta)$ causes the change $v_i \leftarrow v_i + \Delta$, where $\Delta \in \{-M, \ldots, M\}$. Occasionally our data structure must answer queries for some function of $v$. In many applications $n$ is extremely large, and thus it is desirable to provide a data structure with space consumption much less than $n$, e.g. polylogarithmic in $n$. For example, $n$ may be the number of valid IP addresses ($n = 2^{128}$ in IPv6), and $v_i$ may be the number of packets sent from source IP address $i$ on a particular link. A query may then ask for the support size of $v$ (the "distinct elements" problem [14]), which was used for example to estimate the spread of the Code Red worm after filtering the packet stream based on the worm's signature [13, 35]. Another query may be the $\ell_2$ norm of $v$ [2], which was used by AT&T as part of a packet monitoring system [31, 42]. In some examples there is more than one possible query to be asked; in "point query" problems a query is some $i \in [n]$ and the data structure must output $v_i$ up to some additive error, e.g. $\varepsilon\|v\|_p$ [6, 12]. Such point query data structures are used as subroutines in the heavy hitters problem, where informally the goal is to output all $i$ such that $v_i$ is "large". If the data structure is linearly composable (meaning that data structures for $v$ and $v'$ can be combined to form a data structure for $v - v'$), heavy hitters data structures can be used for example to detect trending topics in search engine query streams [20, 21, 6]. In fact the point query structure [6] has been implemented in the log analysis language Sawzall at Google [41].

Coupled with the great success in providing small-space data structures for various turnstile streaming problems has been a great amount of progress in proving space lower bounds, i.e. theorems which state that *any* data structure for some particular turnstile streaming problem must use space (in bits) above some lower bound. For example, tight or nearly tight space lower bounds are known for the distinct elements problem [2, 43, 44, 25], $\ell_p$ norm estimation [2, 3, 5, 43, 29, 22, 24, 26, 44, 25], heavy hitters [27], entropy estimation [4, 29], and several other problems.

While there has been much previous work on understanding the space required for solving various streaming problems, much less progress has been made regarding time complexity: the time it takes to process an update in the stream and the time it takes to answer a query about the stream. This is despite strong motivation, since in several applications the data stream may be updated at an extremely fast rate, so that in fact update time is often arguably *more important* than space consumption; for example, [42] reported in their application for $\ell_2$-norm estimation that their system was constrained to spend only 130 nanoseconds per packet to keep up with high network speeds. If for example $n$ is the number of IP addresses $2^{128}$, then certainly the (computer connected to the) router has much more than $\lg n$ bits of memory or even cache, which is a regime much previous streaming literature has focused on. Arguably this primary focus on space was not because of the greater *practical* interest in space complexity, but simply because up until this point there was a conspicuous absence of understanding concerning the tradeoff between update time and memory consumption (especially from the perspective of lower bounds). In short, many previous research targets were defined by what our tools allowed us to prove. In fact, in high-speed applications such as in networking, one could easily imagine preferring a solution using $n^{\epsilon}$ memory (as long as it fit in cache) with a very fast update time than, say, polylogarithmic memory with worse update time.

Of course, without any space constraint achieving fast update and query times is trivial: store $v$ and its norm in memory explicitly and spend constant time to add $\Delta$ to $v_i$ and change the stored norm after each update. Thus an interesting data structural issues arises: how can we simultaneously achieve small space and low time for turnstile streaming data structures? As mentioned, surprisingly very little is understood about this question for any turnstile streaming problem. For some problems we have very fast algorithms (e.g. constant update time for distinct elements [30], and also for $\ell_2$ estimation [42]), whereas for others we do not (e.g. super-constant time for $\ell_p$ estimation for $p \neq 2$ [28] and heavy hitters problems [6, 12]), and we do not have proofs precluding the possibility that a fast algorithm exists. Indeed, the only previous time lower bounds for streaming problems are those of Clifford and Jalsenius [8] and Clifford, Jalsenius, and Sach [9, 10] for streaming multiplication, streaming edit distance and streaming Hamming distance computation. These problems are significantly harder than e.g. $\ell_p$ estimation (the lower bounds proved apply even for super-linear space usage) and there appears to be no way of extending their approach to obtain lower bounds for the problems above. Importantly, the problems considered in [8, 9, 10] are not in the turnstile model.

A natural model for upper bounds in the streaming literature (and also for data structures in general), is the word RAM model: basic arithmetic operations on machine words of $w$ bits each cost one unit of time. In the data structure literature, one of the strongest lower bounds that can be proven is in the cell probe model [34, 15, 45], where one assumes that the data structure is broken up into $S$ words each of size $w$ bits and cost is only incurred when the user reads a memory cell from the data structure ($S$ is the space). Lower bounds proven in this model hold against any algorithm operating in the word RAM model. In recent years there has been much success in proving data structure lower bounds in the cell probe model (see for example [39, 33]) using techniques from communication complexity and information theory. Can these techniques be imported to obtain time lower bounds for clasically studied turnstile streaming problems?

Question 1. *Can we use techniques from cell probe lower bound proofs to lower bound the time complexity for classical streaming problems?*

Indeed the lower bounds for streaming multiplication and Hamming distance computation were proved using the *information transfer* technique of Pătraşcu and Demaine [40], but this approach does not seem effective against the turnstile streaming problems above.

There are a couple of obvious problems to attack for Question 1. For example, for many streaming problems one can move from, say, 2/3 success probability to $1-\delta$ success probability by running $\Theta(\lg(1/\delta))$ instantiations of the algorithm in parallel then outputting the median answer. This is possible whenever the output of the algorithm is numerical, such as for example distinct elements, moment estimation, entropy estimation, point query, or several other problems. Note that doing so increases both the space and update time complexity of the resulting streaming algorithm by a $\Theta(\lg(1/\delta))$ factor. Is this necessary? It was shown that the blowup in *space* is necessary for several problems by Jayram and Woodruff [26], but absolutely nothing is known about whether the blowup in time is required. Thus, any nontrivial lower bound in terms of $\delta$ would be novel.

Another problem to try for Question 1 is the heavy hitters problem. Consider the $\ell_1$ heavy hitters problem: a vector $v$ receives turnstile updates, and during a query we must report all $i$ such that $|v_i| \geq \varepsilon\|v\|_1$. Our list is allowed to contain some false positives, but not "too false", in the sense that any $i$ output should at least satisfy $|v_i| \geq (\varepsilon/2)\|v\|_1$. Algorithms known for this problem using non-trivially small space, both randomized [12] and deterministic [19, 37], require $\tilde{\Omega}(\lg n)$ update time. Can we prove a matching lower bound?

## *Our Results.*

In this paper we present the first update time lower bounds for a range of classical turnstile streaming problems. The lower bounds apply to a restricted class of randomized streaming algorithms that we refer to as randomized *non-adaptive* algorithms. We say that a randomized streaming algorithm is *non-adaptive* if:

- Before processing any elements of the stream, the algorithm may toss some random coins.
- The memory words read/written to (henceforth jointly referred to as probed) on any update operation $(i, \Delta)$ are completely determined from the index $i$ and the initially tossed random coins.

Thus a non-adaptive algorithm may not decide which memory words to probe based on the current contents of the memory words. Note that in this model of non-adaptivity, the random coins can encode e.g. a hash function chosen (independent of the stream) from a desirable family of hash functions and the update algorithm can choose what to probe based on the hash function. It is only the input-specific contents of the probed words that the algorithm may not use to decide which words to probe. To the best of our knowledge, all the known algorithms for classical turnstile streaming problems are indeed *non-adaptive*, in particular *linear sketches* (i.e. maintaining $\Pi v$ in memory for some $r \times n$ matrix $\Pi$, $r \ll n$). One exception we are aware of is an adaptive algorithm given for a *promise problem* in [7] (it is also worth mentioning that for the non-promise version of the problem, the algorithm given in the same work is again non-adaptive). We further remark that our lower bounds only require the update procedure to be non-adaptive and still apply if adaptivity is used to answer queries.

Remark 2. One common technique in turnstile streaming (e.g. see [28]) is to batch some number of updates then process them all more time-efficiently once the batch is large enough. In all examples we are aware of using this technique, a time savings is gained solely because batching allows for faster evaluation of the hash functions involved (e.g. using fast multipoint evaluation of polynomials). Thus, such techniques have only ever found application to decreasing word RAM time complexity, but have thus far never been effective in decreasing cell probe complexity, which is what our current work here lower bounds. Thus, in all these cases, there is a non-adaptive algorithm achieving the best known cell probe complexity.

For the remainder of the paper, for ease of presentation we assume that the update increments are (possibly negative) integers bounded by some $M \le \mathrm{poly}(n)$ in magnitude, and that the number of updates in the stream is also at most $\mathrm{poly}(n)$. We further assume the trans-dichotomous model [16, 17], i.e. that the machine word size $w$ is $\Theta(\lg n)$ bits. This is a natural assumption, since typically the streaming literature assumes that basic arithmetic operations on a value $|v_i|$, the index of the current position in the stream, or an index $i$ into $v$ can be performed in constant time.

We prove lower bounds for the following types of queries in turnstile streams. For each problem listed, the query function takes no input (other than point query, which takes an input $i \in [n]$). Each query below is accompanied by a description of what the data structure should output.

- $\ell_1$ **heavy hitter:** return an index $i$ such that $|v_i| \ge \|v\|_\infty - \|v\|_1/2$.
- **Point query:** given $i$ at query time, returns $v_i \pm \|v\|_1/2$.
- $\ell_p/\ell_q$ **norm estimation** $(1 \le q \le p \le \infty)$**:** returns $\|v\|_p \pm \|v\|_q/2$.
- **Entropy estimation.** returns a 2-approximation of the entropy of the distribution which assigns probability $|v_i|/\|v\|_1$ to $i$ for each $i \in [n]$.

The lower bounds we prove for non-adaptive streaming algorithms are as follows ($n^{-O(1)} \le \delta \le 1/2 - \Omega(1)$ is the failure probability of a query):

- Any randomized non-adaptive streaming algorithm for point query, $\ell_p/\ell_q$ estimation with $1 \le q \le p \le \infty$, and entropy estimation, must have worst case update time $t_u = \Omega(\frac{\lg(1/\delta)}{\sqrt{\lg n \lg(eS/t_u)}})$.

  We also show that any *deterministic* non-adaptive streaming algorithm for the same problems must have worst case update time $t_u = \Omega(\lg n/\lg(eS/t_u))$.
- Any randomized non-adaptive streaming algorithm for $\ell_1$ heavy hitters, must have worst case update time $t_u = \Omega(\min\{\sqrt{\frac{\lg(1/\delta)}{\lg(eS/t_u)}}, \frac{\lg(1/\delta)}{\sqrt{\lg t_u \cdot \lg(eS/t_u)}}\})$. Any deterministic and non-adaptive streaming algorithm for $\ell_1$ heavy hitters must have worst case update time $t_u = \Omega(\frac{\lg n}{\lg(eS/t_u)})$.

Remark 3. The deterministic lower bound above for point query matches two previous upper bounds for point query [37], which use error-correcting codes to yield deterministic point query data structures. Specifically, for space $S = O(\lg n)$, our lower bound implies $t_u = \Omega(\lg n)$, matching an upper bound based on random codes. For space $O((\lg n/\lg\lg n)^2)$, our lower bound is $t_u = \Omega(\lg n/\lg\lg n)$, matching an upper bound based on Reed-Solomon codes. Similarly, the deterministic bound above for deterministic $\ell_2/\ell_1$ norm estimation matches the previous upper bound for this problem [37], showing that for the optimal space $S = \Theta(\lg n)$, the fastest query time of non-adaptive algorithms is $t_u = \Theta(\lg n)$.

These deterministic upper bounds are also in the cell probe model. In particular, the point query data structure based on random codes and the norm estimation data structure require access to combinatorial objects that are shown to exist via the probabilistic method, but for which we do not have explicit constructions. The point query structure based on Reed-Solomon codes can be implemented in the word RAM model with $t_u = \tilde{O}(\lg n)$ using fast multipoint evaluation of polynomials. This is because performing an update, in addition to accessing $O(\lg n/\lg\lg n)$ memory cells of the data structure, requires evaluating a degree-$O(\lg n/\lg\lg n)$ polynomial on $O(\lg n/\lg\lg n)$ points to determine which memory cells to access (see [37] for details).

Remark 4. The best known randomized upper bounds are $S = t_u = O(\lg(1/\delta))$ for point query [12] and $\ell_p/\ell_p$ estimation for $p \le 2$ [42, 28]. For entropy estimation the best upper bound has $S = t_u = \tilde{O}((\lg n)^2 \lg(1/\delta))$ [23]. For $\ell_1$ heavy hitters the best known upper bound (in terms of $S$ and $t_u$) has $S = t_u = O(\lg(n/\delta))$.

In addition to being the first non-trivial turnstile update time lower bounds, we also managed to show that the update time has to increase polylogarithmically in $1/\delta$ as the error probability $\delta$ decreases, which is achieved with the typical reduction using $\lg(1/\delta)$ independent copies of a data structure with constant error probability.

Our lower bounds can also be viewed in another light. If one is to obtain constant update time algorithms for the above problems, then one has to design algorithms that are *adaptive*. Since all known upper bounds have non-adaptive updates, this would require a completely new strategy to designing turnstile streaming algorithms. Note that for randomized algorithms, our lower bounds do not rule out constant update time with constant error probability. If however the error probability is upper bounded by $2^{-\omega(\sqrt{\lg n})}$, then adaptivity is necessary to achieve constant update time.

We also show a new cell probe *upper bound* for $\ell_1$ point query which outperforms the CountMin sketch [12] in terms of *query time*, while matching it in both space and update time (see Section 4.2). Our new algorithm is inspired by the solution to the hard instance for our above lower bounds and we believe this upper bound provides evidence that despite the importance of the problems, the effort on understanding the time complexity of them has been insufficient and perhaps better algorithms are achievable. In fact, our upper bound even demonstrates the $\lg(1/\delta)/\sqrt{\lg n}$ behaviour of our lower bounds, further supporting the hypothesis that faster algorithms exist. Our upper bound is randomized, non-adaptive and in the cell probe model, meaning that we assume computation is free of charge and that we have access to input independent truly random hash functions that can be evaluated free of charge. Clearly our lower bounds apply to this setting. It would be interesting to find a fast implementation of our algorithm in the word-RAM model.

### *Technique.*

As suggested by Question 1, we prove our lower bounds using recent ideas in cell probe lower bound proofs. More specifically, we use ideas from the technique now formally known as *cell sampling* [18, 38, 32]. This technique derives lower bounds based on one key observation: if a data structure/streaming algorithm probes $t$ memory words on an update, then there is a set $C$ of $t$ memory words such that at least $m/S^t$ updates probe only memory words in $C$, where $m$ is the number of distinct updates in the problem (for data structure lower bound proofs, we typically consider queries rather than updates, and we obtain tighter lower bounds by forcing $C$ to have near-linear size).

We use this observation in combination with the standard one-way communication games typically used to prove streaming space lower bounds. In these games, Alice receives updates to a streaming problem and Bob receives a query. Alice runs her updates through a streaming algorithm for the corresponding streaming problem and sends the resulting $Sw$ bit memory footprint to Bob. Bob then answers his query using the memory footprint received from Alice. By proving communication lower bounds for *any* communication protocol solving the one-way communication game, one obtains space lower bounds for the corresponding streaming problem.

At a high level, we use the cell sampling idea in combination with the one-way communication game as follows: if Alice's non-adaptive streaming algorithm happens to "hash" her updates such that they all probe the same $t$ memory cells, then she only needs to send Bob the contents of those $t$ cells. If $t < S$, this gives a communication saving over the standard reduction above. We formalize this as a general sketch-compression theorem, allowing us to compress the memory footprint of any non-adaptive streaming algorithm at the cost of increasing the error probability. This general theorem has the advantage of allowing us to re-use previous space lower bounds that have been proved using the standard reduction to one-way communication games, this time however obtaining lower bounds on the update time. We demonstrate these ideas in Section 2 and also give a more formal definition of the classic one-way communication game.

## 2. SKETCH COMPRESSION

In the following, we present a general theorem for compressing non-adaptive sketches. Consider a streaming problem in which we are to maintain an $n$-dimensional vector $v$. Let $U$ be the *update domain*, where each element of $U$ is a pair $(i, \Delta) \in [n] \times \{-M, \dots, M\}$ for some $M = \text{poly}(n)$. We interpret an update $(i, \Delta) \in U$ as having the effect $v[i] \leftarrow v[i] + \Delta$. Initially all entries of $v$ are 0. We also define the *query domain* $Q = \{q_1, \dots, q_r\}$, where each $q_i \in Q$ is a function $q_i : \mathbb{Z}^n \to \mathbb{R}$. With one-way communication games in mind, we define the input to a streaming problem as consisting of two parts. More specifically, the *pre-domain* $D_{pre} \subseteq U^a$ consists of sequences of $a$ update operations. The *post-domain* $D_{post} \subseteq \{U \cup Q\}^b$ consists of sequences of $b$ updates and/or queries. Finally, the *input domain* $D \subseteq D_{pre} \times D_{post}$ denotes the possible pairings of $a$ initial updates followed by $b$ intermixed queries and updates. The set $D$ defines a streaming problem $P_D$.

We say that a randomized streaming algorithm for a problem $P_D$ uses $S$ words of space if the maximum number of memory words used when processing any $d \in D$ is $S$. Here a memory word consists of $w = \Theta(\lg n)$ bits. The worst case update time $t_u$ is the maximum number of memory words read/written to upon processing an update operation for any $d \in D$. The error probability $\delta$ is defined as the maximum probability over all $d \in D$ and queries $q_i \in d \cap Q$, of returning an incorrect results on query $q_i$ after the updates preceding it in the sequence $d$.

A streaming problem $P_D$ of the above form naturally defines a one-way communication game: on an input $(d_1, d_2) \in D$, Alice receives $d_1$ (the first $a$ updates) and Bob receives $d_2$ (the last $b$ updates and/or queries). Alice may now send a message to Bob based on her input and Bob must answer all queries in his input as if streaming through the concatenated sequence of operations $d_1 \circ d_2$. The error probability of a communication protocol is defined as the maximum over all $d \in D$ and $q_i \in \{d \cap Q\}$, of returning an incorrect results on $q_i$ when receiving $d$ as input.

Traditionally, the following reduction is used:

THEOREM 5. *If there is a randomized streaming algorithm for $P_D$ with space usage $S$ and error probability $\delta$, then there is a private coin protocol for the corresponding one-way communication game in which Alice sends $Sw$ bits to Bob and the error probability is $\delta$.*

**Proof.** Alice simply runs the streaming algorithm on her input and sends the memory image to Bob. Bob continues the streaming algorithm and outputs the answers. ■

Recall from Section 1 that a randomized streaming algorithm is *non-adaptive* if:

- Before processing any elements of the stream, the algorithm may toss some random coins.
- The memory words read/written to (henceforth jointly referred to as probed) on any update operation $(i, \Delta)$ is completely determined from $i$ and the initially tossed random coins.

We show that for non-adaptive algorithms, one can efficiently reduce the communication by increasing the error probability. We require some additional properties of the problem however: we say that a streaming problem $P_D$ is *permutation invariant* if for any *permutation* $\pi : [n] \to [n]$, it holds that $\pi(q_i(v)) = (\pi(q_i)(\pi(v)))$ for all $q_i \in Q$. Here $\pi(v)$ is the $n$-dimensional vector with value $v[i]$ in entry $\pi[i]$, $\pi(q_i)$ maps all indices (if any) in the definition of the query $q_i$ wrt. $\pi$ and $\pi(q_i(v))$ maps all indices in the answer $q_i(v)$ (if any) wrt. $\pi$.

Observe that point query, $\ell_p$ estimation, entropy estimation and heavy hitters all are permutation invariant problems. For point query, we have $\pi(q_i(v)) = q_i(v)$ since answers contain no indices, but $\pi(q_i)$ might differ from $q_i$

since queries are defined from indices. For $\ell_p$ estimation and entropy estimation, we simply have $\pi(q_i(v)) = q_i(v)$ and $\pi(q_i) = q_i$ since neither queries or answers involve indices. For heavy hitters we have $\pi(q_i) = q_i$ (there is only one query), but we might have that $\pi(q_i(v)) \neq q_i(v)$ since the answer to the one query is an index. We now have the following:

THEOREM 6. *If there is a randomized non-adaptive streaming algorithm for a permutation invariant problem $P_D$ with $a \leq \sqrt{n}$, having space usage $S$, error probability $\delta$, and worst case update time $t_u \leq (1/2)(\lg n/\lg(eS/t_u))$, then there is a private coin protocol for the corresponding one-way communication game in which Alice sends at most $a \lg e + t_u a \lg(eS/t_u) + \lg a + \lg\lg(en/a) + t_u w + 1$ bits to Bob and the error probability is $2e^a \cdot (eS/t_u)^{t_u a}\delta$.*

Before giving the proof, we present the two main ideas. First observe that once the random choices of a non-adaptive streaming algorithm have been made, there must be a large collection of indices $I \subseteq [n]$ for which all updates $(i, \Delta)$, where $i \in I$, probe the same small set of memory words (there are at most $\binom{S}{t_u}$ distinct sets of $t_u$ words to probe). If all of Alice's updates probed only the same set of $t_u$ words, then Alice could simply send those words to Bob and we would have reduced the communication to $t_u w$ bits. To handle the case where Alice's updates probe different sets of words, we make use of the permutation invariant property. More specifically, we show that Alice and Bob can agree on a collection of $k$ permutations of the input indices, such that one of these permutes all of Alice's updates to a new set of indices that probe the same $t_u$ memory cells. Alice can then send this permutation to Bob and they can both alter their input based on the permutation. Therefore Alice and Bob can solve the communication game with $\lg k + t_u w$ bits of communication. The permutation of indices unfortunately increases the error probability as we shall see below. With these ideas in mind, we now give the proof of Theorem 6.

**Proof** (of Theorem 6). Alice and Bob will permute the indices in their communication problem and use the randomized non-adaptive streaming algorithm on this transformed instance to obtain an efficient protocol for the original problem.

By Yao's principle, we have that the randomized complexity of a private coin one-way communication game with error probability $\delta$ equals the complexity of the best deterministic algorithm with error probability $\delta$ over the worst distribution. Hence we show that for any distribution $\mu$ on $D \subseteq D_{pre} \times D_{post}$, there exists a deterministic one-way communication protocol with $t_u a \lg S + \lg a + \lg\lg n + t_u w$ bits of communication and error probability $2e^a \cdot (eS/t_u)^{t_u a}\delta$. We let $\mu_1$ denote the marginal distribution over $D_{pre}$ and $\mu_2$ the marginal distribution over $D_{post}$.

Let $\mu = (\mu_1, \mu_2)$ be a distribution on $D$. Define a new distribution $\gamma = (\gamma_1, \gamma_2)$: pick a uniform random permutation $\pi$ of $[n]$. Now draw an input $d$ from $\mu$ and permutate all indices of updates (and queries if defined for such) using the permutation $\pi$. The resulting sequence $\pi(d)$ is given to Alice and Bob as before, which defines the new distribution $\gamma = (\gamma_1, \gamma_2)$. We use $A \sim \mu_1$ to denote the r.v. providing Alice's input drawn from distribution $\mu_1$ and $\pi(A) \sim \gamma_1$ denotes the random variable providing Alice's transformed input. We define $B \sim \mu_2$ and $\pi(B) \sim \gamma_2$ symmetrically.

Recall we want to solve the one-way communication game on $A$ and $B$. To do this, first observe that by fixing the random coins, the randomized non-adaptive streaming algorithm gives a non-adaptive and deterministic streaming algorithm that has error probability $\delta$ and space usage $S$ under distribution $\gamma$. Before starting the communication protocol on $A$ and $B$, Alice and Bob both examine the algorithm (it is known to both of them). Since it is non-adaptive and deterministic, they can find a set of $t_u$ memory words $C$, such that at least $n/\binom{S}{t_u} > n/(eS/t_u)^{t_u} \geq \sqrt{n} \geq a$ indices $i \in [n]$ satisfy that any update $(i, \Delta)$ probes only memory words in $C$. We let $I^C$ denote the set of all such indices (again, $I^C$ is known to both Alice and Bob). Alice and Bob also agree on a set of permutations $\{\rho_1, \ldots, \rho_k\}$ (we determine a value for $k$ later), such that for any set of at most $a$ indices, $I'$, that can occur in Alice's updates, there is at least one permutation $\rho_i$ where:

- $\rho_i(j) \in I^C$ for all $j \in I'$
- Let $I(A)$ denote the (random) indices of the updates in $A$. Then the probability that the non-adaptive and deterministic protocol errs on input $\rho_i(A)$, conditioned on $I(A) = I'$, is at most $2e^a \cdot (eS/t_u)^{t_u a} \cdot \varepsilon_{I(A)=I'}$ where $\varepsilon_{I(A)=I'}$ is the error probability of the deterministic and non-adaptive streaming algorithm on distribution $\gamma$, conditioned on $I(A) = I'$.

Again, this set of permutations is known to both players. The protocol is now simple: upon receiving $A$, Alice finds the index $i$ of a permutation $\rho_i$ satisfying the above for the indices $I(A)$. She then sends this index to Bob and runs the deterministic and non-adaptive algorithm on $\rho_i(A)$. She forwards the addresses and contents of all memory words in $C$ as well. This costs a total of $\lg k + |C| \cdot w \leq \lg k + t_u(n)w$ bits. Note that no words outside $C$ are updated during Alice's updates. Bob now remaps his input $B$ according to $\rho_i$ and runs the deterministic and non-adaptive streaming algorithm on his updates and queries. Observe that for each query $q_j \in B$, Bob will get the answer $\rho_i(q_j)(\rho_i(v))$ if the algorithm does not err, where $v$ is the "non-permuted" vector after processing all of Alice's updates $A$ and all updates in $B$ preceeding the query $q_j$. For each such answer, he computes $\rho_i^{-1}(\rho_i(q_j)(\rho_i(v)))$. Since $P_D$ is permutation invariant, we have $\rho_i^{-1}(\rho_i(q_j)(\rho_i(v))) = \rho_i^{-1}(\rho_i(q_j(v))) = q_j(v)$. The final error probability (over $\mu$) is hence at most $2e^a \cdot (eS/t_u)^{t_u a} \cdot \delta$ since $\mathbb{E}_{I'}\, \varepsilon_{I(A)=I'} = \delta$.

We only need a bound on $k$. For this, fix one set $I'$ of at most $a$ indices in $[n]$ and consider drawing $k$ uniform random permutations. For each such random permutation $\Gamma$, note that $\Gamma(I')$ is distributed as $I(\pi(A))$ conditioned on $I(A) = I'$. Hence, the expected error probability when using the map $\Gamma$ (expectation over choice of $\Gamma$) is precisely $\varepsilon_{I(A)=I'}$. We also have

$$\mathbb{P}(\Gamma(I') \subseteq I^C) = \frac{\binom{|I^C|}{|I'|}}{\binom{n}{|I'|}} \geq \left(\frac{|I^C|}{en}\right)^{|I'|} \geq e^{-a} \cdot \left(\frac{eS}{t_u}\right)^{-t_u a}$$

By Markov's inequality and a union bound, we have both $\Gamma(I') \subseteq I^C$ and error probability at most $2e^a \cdot (eS/t_u)^{t_u a} \cdot \varepsilon_{I(A)=I'}$ with probability at least $e^{-a} \cdot (eS/t_u)^{-t_u a}/2 \stackrel{\text{def}}{=} p$ over the choice of $\Gamma$. Thus if we pick $k = (1/p)a\lg(en/a) > (1/p) \cdot \lg\binom{n}{a}$ and use that $1 + x \leq e^x$ for all real $x$, then setting the probability that all permutations $\Gamma$ chosen fail to have the desired failure probability and $\Gamma(I') \subseteq I^C$ is at most $(1-p)^k < 1/\binom{n}{a}$. Thus by a union bound over all $\binom{n}{a}$ size-$a$ subsets of indices, we can conclude that the desired set of permutations exists. ∎

## 3. IMPLICATIONS

In this section, we present the (almost) immediate implications of Theorem 6. All these results follow by reusing the one-way communication game lower bounds originally proved to obtain space lower bounds of heavy hitters [27]. Consider the generic protocol in Figure 1. For different streaming problems, we will show the different ways to implement the function $Check(t, j)$ using the updates and queries of the problems, where $Check(t, j)$ is supposed to be able to tell if $t$ is equal to $i_j$ with failure probability $\delta$. We show first that if this is the case, we obtain a lower bound on the update time $t_u$. The lower bounds then follow from the implementation of $Check(t, j)$.

```
 1: Alice chooses $a$ indices $i_1, \ldots, i_a$ randomly without replacement in $[n]$.
 2: Alice performs updates $v[i_j] \leftarrow v[i_j] + C^j$ for $j = 1, \ldots, a$, where $C$ is a large constant.
 3: Alice sends her memory state to Bob.
 4: for $j$ from $a$ down to 1 do        ▷ Bob decodes $i_a, \ldots, i_1$ from Alice's message.
 5:     for all $t \in [n]$ do
 6:         if Check(t,j) then
 7:             Bob declares $i_j = t$.
 8:             Bob performs the update $v[i_j] \leftarrow v[i_j] - C^j$
 9:         end if
10:     end for
11: end for
```

**Figure 1: The communication protocol for Alice to send $a$ random numbers in $[n]$ to Bob.**

The proofs of the following two theorems follow easily from Theorem 6:

Theorem 7. *Any randomized non-adaptive streaming algorithm that can implement all the $Check(t, j)$ calls using no more than $k \le n^{O(1)}$ queries with failure probability $n^{-\Omega(1)} \le \delta \le 1/2 - \Omega(1)$ each, must have worst case update time $t_u = \Omega\left(\min\left\{\sqrt{\frac{\lg 1/\delta}{\lg(eS/t_u)}}, \frac{\lg 1/\delta}{\sqrt{\lg k \lg(eS/t_u)}}\right\}\right)$.*

**Proof.** We first prove the lower bound for the case where $\sqrt{\frac{\lg 1/\delta}{\lg(eS/t_u)}}$ is the smaller of the two terms. This is the case at least from $n^{-\Omega(1)} \le \delta \le k^{-3}$. Assume for contradiction that such a randomized non-adaptive algorithm exists with $t_u = o(\sqrt{\lg(1/\delta)/\lg(eS/t_u)})$. Set $a = c_0 t_u$ for a sufficiently large constant $c_0$. We invoke Theorem 6 to conclude that Alice can send her memory state to Bob using $a \lg e + t_u a \lg(eS/t_u) + \lg a + \lg\lg(en/a) + t_u w + 1$ bits while increasing the failure probability of Bob's $k$ queries to

$$2e^a \cdot (eS/t_u)^{t_u a}\delta \le (eS/t_u)^{c_0 t_u^2}\delta^{1-o(1)} \le \delta^{1-o(1)} \le k^{-2}$$

each. By a union bound, the answer to all Bob's queries are correct with probability at least $1 - 1/k \ge 9/10$, so Bob can recover $i_1, \ldots, i_a$ with probability 9/10. By Fano's inequality, Alice must send Bob at least $\Omega(H(i_1, \ldots, i_a)) = \Omega(a \lg(n/a)) \ge c_0 c_1 t_u \lg n$ bits for some constant $c_1$ (where $c_1$ does not depend on $c_0$). But the size of her message was

$$a \lg e + t_u a \lg(eS/t_u) + \lg a + \lg\lg(en/a) + t_u w + 1 \le c_0 t_u^2 \lg(eS/t_u) + t_u w + o(t_u \lg n).$$

We assumed

$$t_u = o\left(\sqrt{\lg(1/\delta)/\lg(eS/t_u)}\right) = o\left(\sqrt{\lg n/\lg(eS/t_u)}\right)$$

and thus the above is bounded by $t_u w + o(t_u \lg n)$. Setting $c_0$ high enough, we get that $w \ge (c_0 c_1 \lg n)/2$ and thus we have reached a contradiction.

For the case $k^{-3} < \delta < 1/2 - \Omega(1)$, observe that we can decrease the failure probability to $k^{-3}$ by increasing the space, update time and query time by a factor $\alpha = \Theta(\lg_{1/\delta} k)$: simply run $\alpha$ independent copies of the streaming algorithm and return the majority answer on a query. Hence the lower bound becomes

$$\Omega\left(\frac{\sqrt{\frac{\lg k}{\lg(eS\alpha/(t_u\alpha))}}}{\alpha}\right) = \Omega\left(\frac{\lg 1/\delta}{\sqrt{\lg k \lg(eS/t_u)}}\right).$$

∎

Theorem 8. *Any deterministic non-adaptive streaming algorithm that can implement $Check(t, j)$ must have worst case update time $t_u = \Omega(\frac{\lg n}{\lg(eS/t_u)})$.*

**Proof.** The proof is similar to that of Theorem 7. Assume for contradiction there is a deterministic non-adaptive streaming algorithm with $t_u = o(\lg n/\lg(eS/t_u))$. Choose $a = c_0 t_u$ for sufficiently large constant $c_0$. By Theorem 6, Alice can send her memory state to Bob using

$$\begin{aligned} a \lg e + t_u a \lg(eS/t_u) + \lg a + \lg\lg(en/a) + t_u w + 1 &\le \\ c_0 t_u^2 \lg(eS/t_u) + t_u w + o(t_u \lg n) &\le \\ t_u w + o(t_u \lg n) & \end{aligned}$$

bits and Bob can still answer every query correctly. Since Bob can recover $i_1, \ldots, i_a$, Alice must send Bob at least $H(i_1, \ldots, i_a) = \Omega(a \lg(n/a)) = c_0 c_1 t_u \lg n$ for some constant $c_1$ (independent of $c_0$). Setting $c_0$ large enough gives $w \ge (c_0 c_1 \lg n)/2$, thus deriving the contradiction. ∎

### 3.1 Applications to specific problems

*Point query.*

We can implement each $Check(t, j)$ by simply querying for the value of $v[t]$ and check if the returned value is at least $C^j/3$. By the guarantee of the algorithm, if it does not fail, the returned value is within a factor 3 from the right value and thus, $Check(t, j)$ can correctly tell if $t = v_j$. Thus we run a total of $na = n^{O(1)}$ queries to implement all $Check(t, j)$'s.

***$\ell_p/\ell_q$ estimation.***

We can implement $Check(t, j)$ as follows.

- $v[t] \leftarrow v[t] - C^j$
- Check if $\|v\|_p$ is at most $C^j/3$
- $v[t] \leftarrow v[t] + C^j$

By the guarantee of the algorithm, if it does not fail, the estimated $\ell_p/\ell_q$ norm is at most $3C^{j-1} < C^j/3$ if $t = i_j$ and it is greater than $C^j/3$ if $t \neq i_j$. Thus, $Check(t, j)$ can correctly tell if $t = v_j$. Again we used $n^{O(1)}$ queries in total.

We can also implement $Check(t, j)$ for entropy estimation.

*Entropy estimation.*

We implement $Check(t, j)$ as follows.

- $v[t] \leftarrow v[t] - C^j$
- Check if the entropy is at most 1/3
- $v[t] \leftarrow v[t] + C^j$

Consider the case the algorithm does not fail. First, if $t = i_j$ then the entropy is at most

$$\begin{aligned}
&\sum_{i=1}^{j} \frac{C^i(C-1)}{C^{j+1}-C} \lg \frac{C^{j+1}-C}{C^i(C-1)} \\
&\leq \sum_{i=1}^{j-1} \frac{C^i(C-1)}{C^{j+1}-C} \lg 2C^{j-i} + \frac{C^j(C-1)}{C^{j+1}-C} \lg \frac{C^{j+1}-C}{C^j(C-1)} \\
&\leq O\left(\frac{\lg C}{C-1}\right) + \frac{C^j(C-1)}{C^{j+1}-C} \cdot \frac{C^j-C}{C^j(C-1)} \\
&\leq O\left(\frac{\lg C}{C-1}\right) \\
&\leq 1/10.
\end{aligned}$$

On the other hand, if $t \neq i_j$ then after the first operation, $\|v\|_1 \leq \frac{2C^{j+1}-C^j-C}{C-1} < 3C^j$ so the entropy is at least the binary entropy of whether the index $i_j$ is picked, which is greater than $H(1/3) > 0.9$. Thus, by the guarantee of the algorithm, $Check(t, j)$ correctly tells if $t = v_j$ using $n^{O(1)}$ queries.

Corollary 9. *Any randomized non-adaptive streaming algorithm for $\ell_1$ heavy hitters with failure probability $n^{-O(1)} \leq \delta \leq 1/2 - \Omega(1)$ must have worst case update time $t_u = \Omega(\min\{\sqrt{\frac{\lg(1/\delta)}{\lg(eS/t_u)}}, \frac{\lg(1/\delta)}{\sqrt{\lg t_u \cdot \lg(eS/t_u)}}\})$. Any deterministic non-adaptive streaming algorithm must have worst case update time $t_u = \Omega\left(\frac{\lg n}{\lg(eS/t_u)}\right)$.*

**Proof.** We use a slightly different decoding procedure for Bob. Instead of running $Check(t, j)$ all indices $t \in [n]$, in iteration $j$, Bob can simply query for the heavy hitter to find $i_j$. Note that in iteration $j$, we have $\|v\|_1 = \frac{C^{j+1}-C}{C-1} < 2C^j = 2v[i_j]$ so if the algorithm is correct, Bob will find the index $i_j$. We have thus implemented all the $Check(t, j)$'s using only $a$ queries. Recall from the proof of Theorem 7 that $a = O(t_u)$. ■

## 4. UPPER BOUNDS

In this section we first present an upper bound for the concrete hard instance used in Section 3 to derive our update time lower bounds. The upper bound nearly matches our lower bound and in particular exhibits the $\lg(1/\delta)/\sqrt{\lg n}$ behaviour with optimal space usage. In Section 4.2 we present an algorithm for $\ell_1$ point query which outperforms the Count-Min sketch in terms of *query time*, while matching it in both space and update time. Our new algorithm is inspired by the solution to the hard instance from Section 3 and we believe this upper bound provides evidence that it might actually be the known upper bounds and not the lower bounds we present that are sub-optimal. Both our upper bounds are randomized, non-adaptive and in the cell probe model, meaning that we assume computation is free of charge and that we have access to truly random hash functions that can be evaluated free of charge. Clearly our lower bounds apply to this setting. Obtaining similar upper bounds in the word-RAM seems to require more ideas. We discuss this in further detail in Section 4.1 and Section 4.2.

### 4.1 Solving the Hard Instance

Recall from Section 3 that the hard instance used to derive the lower bounds has the following form: We have $k$ distinct indices $i_1, \ldots, i_k$ and $k$ values $\Delta_1, \ldots, \Delta_k \in V$ where $V = \{1, \ldots, n^{O(1)}\}$. We perform the updates $v_{i_j} \leftarrow v_{i_j} + \Delta_j$ for $j = 1, \ldots, k$. Following that, we perform the updates $v_{i_j} \leftarrow v_{i_j} - \Delta_j$ for $j = k, \ldots, 1$ possibly intermixed with queries asking whether a given index $i$ satisfies $v_i = \Gamma_i$ for a query value $\Gamma_i \in V$. We present a randomized and non-adaptive algorithm for this problem, where we assume the availability of truly random hash functions. Our solution uses optimal space $\Theta(k \lg(1/\delta)/\lg n)$ words, each word of $w = \Theta(\lg n)$ bits. The query time and update time are both $\Theta(\lg(1/\delta)/\sqrt{\lg n})$ for any $\delta \leq \exp(-\Theta(\sqrt{\lg n}))$.

*Algorithm.*

Construct a table $T$ with $t$ rows and $r$ columns. We denote the rows by $T_1, \ldots, T_t$. Every entry of the table stores a $w$ bit word, which is initialized to 0. We have $t$ truly random hash functions $h_1, \ldots, h_t : [n] \to [r]$ mapping indices to uniform random table cells. We also have $t$ truly random hash functions $\sigma_1, \ldots, \sigma_t : [n] \times V \to \binom{w}{\sqrt{w}}$ mapping indices and a value to uniform random length $w$ bit strings with precisely $\sqrt{w}$ 1-bits. Upon an update $v_i \leftarrow v_i + \Delta$ or $v_i \leftarrow v_i - \Delta$ for some $\Delta \in V$, we do a bitwise XOR of $\sigma_j(i, \Delta)$ onto the word stored in entry $T_j[h_j(i)]$ for $j = 1, \ldots, t$.

To answer whether an index $i$ satisfies $v_i = \Gamma$ for some $\Gamma \in V$, we compute the standard inner product $\langle T_j[h_j(i)], \sigma_j(i, \Gamma)\rangle$ for all $j = 1, \ldots, t$ (interpreting the words as $\{0, 1\}$-vectors in $\mathbb{R}^w$). Note that such an inner product is simply a count of how many 1's $T_j[h_j(i)]$ and $\sigma_j(i, \Gamma)$ have in common. Since $\sigma_j(i, \Gamma)$ has only $\sqrt{w}$ 1's, this number of always bounded by $\sqrt{w}$. If the majority of the $T_j[h_j(i)]$'s have

$$\langle T_j[h_j(i)], \sigma_j(i, \Gamma)\rangle \geq \sqrt{w}/2,$$

we return that $v_i$ equals $\Gamma$ and otherwise that $v_i \neq \Gamma$.

*Analysis.*

In the cell probe model, we measure query time and update time only as the number of memory words read/updated. Thus the query time and update time is $t$ and the space usage is $tr$ words. Given a desired error probability $\delta \leq \exp(-\Theta(\sqrt{\lg n}))$, we set $t = \Theta(\lg(1/\delta)/\sqrt{\lg n})$ and let $r = ck/\sqrt{\lg n}$ for some constant $c > 0$. This gives optimal space and an update and query time of $O(\lg(1/\delta)/\sqrt{\lg n})$. What remains is to show that the error probability is bounded by $\delta$. For this, consider a query asking whether $v_i = \Gamma$ for some $\Gamma \in V$. Since the hash functions used for the different rows are independent, we restrict our attention to one row $T_j$. Let $K_j(i, \Gamma)$ be set of pairs $(i', \Delta_{i'})$ such that $\Delta_{i'} \in V$, $v_{i'} = \Delta_{i'}$ and either $i' \neq i$ or $\Delta_{i'} \neq \Gamma$. Note that $\Delta_{i'} \in V$

implies $\Delta_{i'} \neq 0$ and thus $|K_j(i,\Gamma)| \in \{k-1,k\}$ depending on whether $\Gamma = v_i$ or not. From $K_j(i,\Gamma)$ we also define $C_j(i,\Gamma) \subseteq K_j(i,\Gamma)$ as the pairs $(i',\Delta_{i'}) \in K_j(i,\Gamma)$ satisfying $h_j(i') = h_j(i)$, i.e. $C_j(i,\Gamma)$ is the *conflict list* of pairs hashing to the same word as index $i$ in table $T_j$.

Let $\Sigma$ denote the bitwise XOR of $\sigma_j(i',\Delta_{i'})$ for all $(i',\Delta_{i'})$ in $C_j(i,\Gamma)$. Then if $v_i = \Gamma$ we have $T_j[h_j(i)] = (\Sigma \text{ XOR } \sigma_j(i,\Gamma))$ and if $v_i \neq \Gamma$ we have $T_j[h_j(i)] = \Sigma$. It follows that if $\langle \Sigma, \sigma_j(i,\Gamma)\rangle < \sqrt{w}/2$, then $\langle T_j[h_j(i)], \sigma_j(i,\Gamma)\rangle > \sqrt{w}/2$ iff $v_i = \Gamma$. Hence we let $E_{\text{err}}$ denote the event that $\langle \Sigma, \sigma_j(i,\Gamma)\rangle \geq \sqrt{w}/2$. We wish to bound $\mathbb{P}(E_{\text{err}})$.

For this, let $E_{\text{few}}$ denote the event $|C_j(i,\Gamma)| \leq \sqrt{w}/4$. Now observe that conditioned on $E_{\text{few}}$, there are no more than $w/4$ bits in $\Sigma$ that are 1. Since $\sigma_j$ is truly random and $(i,\Gamma) \notin C_j(i,\Gamma)$, we have that $\sigma_j(i,\Gamma)$ is independent of these set bits and

$$\mathbb{E}[\langle \Sigma, \sigma_j(i,\Gamma)\rangle] \leq \sqrt{w}/4.$$

It follows that

$$\mathbb{P}(E_{\text{err}} \mid E_{\text{few}}) \leq \exp(-\Omega(\sqrt{w})) = \exp(-\Omega(\sqrt{\lg n})).$$

Next observe that

$$\mathbb{E}[|C_j(i,\Gamma)|] \leq k/r = \sqrt{\lg n}/c.$$

For big enough constant $c$, this is less than $\sqrt{w}/8$ and we get from a Chernoff bound that

$$\mathbb{P}(\neg E_{\text{few}}) = \mathbb{P}(|C_j(i,\Gamma)| > \sqrt{w}/4) < \exp(-\Omega(\sqrt{w})) = \exp(-\Omega(\sqrt{\lg n})).$$

We conclude

$$\begin{aligned}\mathbb{P}(E_{\text{err}}) &= \\ \mathbb{P}(E_{\text{err}} \mid E_{\text{few}})\,\mathbb{P}(E_{\text{few}}) + \mathbb{P}(E_{\text{err}} \mid \neg E_{\text{few}})\,\mathbb{P}(\neg E_{\text{few}}) &= \\ \exp(-\Omega(\sqrt{\lg n})).\end{aligned}$$

The probability that we fail in most of the $t$ rows is thus bounded by $\exp(-\Omega(t\sqrt{\lg n})) \leq \delta$ which completes the analysis.

### *Discussion.*

There are two places in the above algorithm where we make use of free computation in the cell probe model: Computing $\langle T_j[h_j(i)], \sigma_j(i,\Gamma)\rangle$ and in the efficient truly random hash functions. There seems to be hope of getting around the true randomness by resorting to $\Theta(\lg n)$-wise independent hash functions and batching several evaluations to exploit algorithms such as fast multipoint evaluation of polynomials to obtain close-to-constant amortized evaluation time, see e.g. [28]. Hashing only to bit strings with exactly $\sqrt{w}$ 1's in near-constant time might need some additional ideas.

## 4.2 Faster $\ell_1$ Point Query in the Cell Probe Model

Below we present our improved algorithm for $\ell_1$ point query. In this problem we must support updates $v_i \leftarrow v_i + \Delta$ for $\Delta \in \{-n^{O(1)},\dots,n^{O(1)}\}$. Given a query index $i$, we must return $v_i$ up to an additive error of $\varepsilon\|v\|_1$ with probability at least $1-\delta$. For the reader familiar with the classic CountMin sketch, our basic idea is to use the word-packing approach from Section 4.1 to pack roughly $\lg n$ counters in one word. The difficulty here is that $\Delta$ requires $\Theta(\lg n)$ bits and not just one bit as in Section 4.1, thus no more than one counter fits in a word. We get around this by storing a *summary word* for groups of roughly $\lg n$ counters. The summary words store $(1+\varepsilon)$-approximations of the values of the counters and hence several approximations can be packed in one word. When answering queries, it suffices to use the approximate counters and thus we have effectively reduced the number of words that must be read. Our final result, in the cell probe model, is $O(\varepsilon^{-1}\lg(1/\delta))$ words of space, update time $O(\lg(1/\delta))$, and query time $O(1+\varepsilon\lg(1/\delta) + \frac{\lg(1/\delta)}{\sqrt{\lg n}} \cdot \sqrt{\lg\lg n + \lg(1/\varepsilon)})$. The space and update time match the CountMin sketch, whereas the query time strictly outperforms it for $1/n^{o(1)} \leq \varepsilon \leq o(1)$ and $\delta = o(1)$. The details are as follows.

### *Algorithm.*

Let $k = \Theta(\min\{\lg^{-2}(1/\delta), \varepsilon^{-2}, \lg n/\lg\lg_{1+\varepsilon} n\})$. We store a table $T$ with $t$ rows denoted $T_1,\dots,T_t$. Each row $T_j$ is partitioned into $r$ blocks of $k$ entries each. We use $T_j[h,\ell]$ to denote the $\ell$'th entry of the $h$'th block in $T_j$ for any $(h,\ell) \in [r]\times[k]$. Each of the $trk$ entries stores a $\Theta(\lg n)$-bit counter which is initialized to 0. In addition to $T$, we store a table $A$ with $t$ rows and $r$ columns. The rows of $A$ are denoted $A_1,\dots,A_t$ and entry $A_j[h]$ stores a $(1+\varepsilon/4)$-approximation of $T_j[h,\ell]$ for each $\ell \in [k]$ and thus an entry of $A$ takes $O(k\lg\lg_{1+\varepsilon} n) \leq O(\lg n)$ bits.

We have $t$ truly random hash functions $h_1,\dots,h_t : [n] \to [r]$ mapping indices to blocks and $t$ truly random hash functions $\sigma_1,\dots,\sigma_t : [n] \to \binom{k}{\sqrt{k}}$ mapping indices to a subset of $\sqrt{k}$ counters inside a block.

Upon an update $v_i \leftarrow v_i + \Delta$, we add $\Delta$ to all counters $T_j[h_j(i),\ell]$ for $j = 1,\dots,t$ and $\ell \in \sigma_j(i)$. We also update the corresponding $(1+\varepsilon/4)$-approximations of the counters. These are stored in $A_j[h_j(i)]$ for $j = 1,\dots,t$.

To answer a query for index $i$, we read $A_j[h_j(i)]$ for $j = 1,\dots,t$. For each $j$, we extract from $A_j[h_j(i)]$ a $(1+\varepsilon/4)$-approximations of $T_j[h_j(i),\ell]$ for each $\ell \in \sigma_j(i)$. We finally return the median of these approximations as our estimate of $v_i$.

### *Analysis.*

We first analyse the resource requirements. The space usage is $O(trk)$ words. In the cell probe model, the query time and update time is defined as the number of words read/updated and thus the algorithm has update time $O(t\sqrt{k})$ and query time $O(t)$.

We fix the parameters in the following. Given a desired error probability $\delta$, we set $t = 1 + O(\lg(1/\delta)/\sqrt{k})$. To obtain asymptotically optimal space, we set $r = c(\varepsilon^{-1}\lg(1/\delta))/(tk)$ for a large constant $c > 0$. Note that by our choice of $k$, we have $r \geq c(\varepsilon^{-1}\lg(1/\delta))/(k + O(\lg(1/\delta)\sqrt{k})) \geq 1$ if $c$ is sufficiently large. What remains is to show that this gives the desired error probability on a query for index $i$. What makes the analysis more cumbersome than for the standard CountMin sketch is mainly the dependencies introduced by the blocking.

For the analysis, first observe that if we consider a counter $T_j[h_j(i),\ell]$ for an $\ell \in \sigma_j(i)$ and let $\beta$ denote the sum of absolute values of all other indices contributing to $T_j[h_j(i),\ell]$, i.e. $\beta = \sum_{i' \neq i : h_j(i) = h_j(i') \wedge \ell \in \sigma_j(i')} |v_{i'}|$, then if $\beta \leq \varepsilon\|v\|_1/2$ we have

$$(v_i - \varepsilon\|v\|_1/2)/(1+\varepsilon/4) \leq A_j[h_j(i)] \leq (v_i + \varepsilon\|v\|_1/2)(1+\varepsilon/4).$$

Since $v_i \le \|v\|_1$, this implies

$$\begin{aligned} A_j[h_j(i)] &\ge (1-\varepsilon/4)(v_i - \varepsilon\|v\|_1/2) \\ &\ge v_i - \varepsilon\|v\|_1/2 - \varepsilon\|v\|_1/4 - \varepsilon\|v\|_1/8 \\ &\ge v_i - \varepsilon\|v\|_1. \end{aligned}$$

and

$$\begin{aligned} A_j[h_j(i)] &\le (v_i + \varepsilon\|v\|_1/2)(1+\varepsilon/4) \\ &\le v_i + \varepsilon\|v\|_1/2 + \varepsilon\|v\|_1/4 + \varepsilon\|v\|_1/8 \\ &\le v_i + \varepsilon\|v\|_1. \end{aligned}$$

It follows that our median estimate is correct if more than half of the counters $T_j[h_j(i), \ell]$ satisfy

$$\sum_{i' \ne i : h_j(i) = h_j(i') \wedge \ell \in \sigma_j(i')} |v_{i'}| \le \varepsilon\|v\|_1/2.$$

To bound the probability this happens, let $C_j$ denote the subset of indices $i' \ne i$ such that $h_j(i') = h_j(i)$. We say that the event $E_{\text{err}}$ happens if there are more than $\sqrt{k}/4$ indices $\ell \in \sigma_j(i)$ for which $\sum_{i' \in C_j : \ell \in \sigma_j(i')} |v_{i'}| > \varepsilon\|v\|_1/2$. We wish to bound $\mathbb{P}(E_{\text{err}})$. For this, partition $C_j$ into two sets $H_j$ and $L_j$. The set $H_j$ contains the *heavy* indices $i' \in C_j$ such that $|v_{i'}| \ge \varepsilon\|v\|_1/2$. $L_j$ contains the remaining *light* indices. We define two auxiliary events. Let $E_{\text{errH}}$ be the event $|H_j| \ge \sqrt{k}/16$ and let $E_{\text{errL}}$ be the event $\sum_{i' \in L_j} |v_{i'}| \ge \varepsilon\|v\|_1\sqrt{k}/16$. If we condition on $\neg E_{\text{errH}}$ and $\neg E_{\text{errL}}$, then there can be no more than $k/16 + k/8 < k/5$ indices $\ell \in [k]$ for which $\sum_{i' \in C_j : \ell \in \sigma_j(i')} |v_{i'}| > \varepsilon\|v\|_1/2$. From this it follows from a Chernoff bound that $\mathbb{P}(E_{\text{err}} \mid \neg E_{\text{errH}} \wedge \neg E_{\text{errL}}) = \exp(-\Omega(\sqrt{k}))$.

Next observe that there can be no more than $2\varepsilon^{-1}$ indices $i'$ for which $|v_{i'}| \ge \varepsilon\|v\|_1/2$. For each such index $i'$, we have $\mathbb{P}(i' \in H_j) = 1/r$ and we get $\mathbb{E}[|H_j|] \le 2\varepsilon^{-1}/r = 2tk/(c\lg(1/\delta)) = 2k/(c\lg(1/\delta)) + O(\sqrt{k}/c) = O(\sqrt{k}/c)$. Setting $c$ high enough, this is no more than $\sqrt{k}/32$. It follows from a Chernoff bound that $\mathbb{P}(E_{\text{errH}}) = \exp(-\Omega(\sqrt{k}))$.

For analysing $\mathbb{P}(E_{\text{errL}})$, observe that

$$\begin{aligned} \mathbb{E}[\sum_{i' \in L_j} |v_{i'}|] &\le \|v\|_1/r \\ &= \varepsilon\|v\|_1 tk/(c\lg(1/\delta)) \\ &= O(\varepsilon\|v\|_1\sqrt{k}/c). \end{aligned}$$

For large enough $c$, this is bounded by $\varepsilon\|v\|_1\sqrt{k}/32$ and thus $E_{\text{errL}}$ is an event in which $\sum_{i' \in L_j} |v_{i'}|$ exceeds its expectation by at least a factor 2. Since all $i'$ considered have $|v_{i'}| \le \varepsilon\|v\|_1/2$, it follows that the probability of $E_{\text{errL}}$ is maximized when the mass (of $\|v\|_1$) is distributed evenly on $2\varepsilon^{-1}$ coordinates. Thus $E_{\text{errL}}$ becomes the event that at least $\sqrt{k}/8$ of these coordinates fall in $L_j$. By the arguments we used to bound $\mathbb{P}(E_{\text{errH}})$, we conclude $\mathbb{P}(E_{\text{errL}}) = \exp(-\Omega(\sqrt{k}))$. Combining the pieces, we conclude $\mathbb{P}(E_{\text{err}}) = \exp(-\Omega(\sqrt{k}))$.

Finally observe that for the median estimate to be incorrect, the event $E_{\text{err}}$ must happen for $\Omega(t)$ rows and the hash functions for these rows are independent. It finally follows by a Chernoff bound that the error probability is bounded by $\exp(-\Omega(t\sqrt{k})) \le \delta$. By our choice of parameters, this gives optimal space of $O(\varepsilon^{-1}\lg(1/\delta))$ words, update time $O(t\sqrt{k}) = O(\sqrt{k} + \lg(1/\delta)) = O(\lg(1/\delta))$ and query time $O(t) = O(1 + \lg(1/\delta)/\sqrt{k})$ which is bounded by

$$O\left(1 + \varepsilon\lg(1/\delta) + \frac{\lg(1/\delta)}{\sqrt{\lg n}} \cdot \sqrt{\lg\lg n + \lg(1/\varepsilon)}\right)$$

This strictly outperforms the CountMin sketch for any $\varepsilon$ and $\delta$ satisfying $1/n^{o(1)} \le \varepsilon \le o(1)$ and $\delta = o(1)$.

### *Discussion.*

In addition to the true randomness, the median computation also prevents the above algorithm from being efficiently implemented in word RAM. If one finds an efficient way of extracting the relevant $\sqrt{k}$ approximations from each $A_j[h_j(i)]$, one can most likely use the standard randomized median selection algorithm [11] to find the median efficiently using word-level parallelism (basically running an external memory model [1] median selection algorithm).

## 5. REFERENCES

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31:1116–1127, 1988.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[3] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.

[4] A. Chakrabarti, G. Cormode, and A. McGregor. A near-optimal algorithm for estimating the entropy of a stream. *ACM Transactions on Algorithms*, 6(3), 2010.

[5] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *IEEE Conference on Computational Complexity*, pages 107–117, 2003.

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

[7] R. H. Chitnis, G. Cormode, M. T. Hajiaghayi, and M. Monemizadeh. Parameterized streaming algorithms for vertex cover. *CoRR*, abs/1405.0093, 2014.

[8] R. Clifford and M. Jalsenius. Lower bounds for online integer multiplication and convolution in the cell-probe model. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 593–604, 2011.

[9] R. Clifford, M. Jalsenius, and B. Sach. Tight cell-probe bounds for online hamming distance computation. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 664–674, 2013.

[10] R. Clifford, M. Jalsenius, and B. Sach. Cell-probe bounds for online edit distance and other pattern matching problems. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015. To appear.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Mass., 1990.

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

[13] C. Estan, G. Varghese, and M. E. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw.*, 14(5):925–937, 2006.

[14] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[15] M. L. Fredman. Observations on the complexity of generating quasi-Gray codes. *SIAM J. Comput.*, 7:134–146, 1978.

[16] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.

[17] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.

[18] A. Gál and P. B. Miltersen. The cell probe complexity of succinct data structures. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming*, pages 332–344, 2003.

[19] S. Ganguly and A. Majumder. CR-precis: A deterministic summary structure for update data streams. In *ESCAPE*, pages 48–59, 2007.

[20] Google. Google Trends. `http://www.google.com/trends`.

[21] Google. Google Zeitgeist. `http://www.google.com/press/zeitgeist.html`.

[22] A. Gronemeier. Asymptotically optimal lower bounds on the NIH-multi-party information complexity of the AND-function and disjointness. In *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 505–516, 2009.

[23] N. J. A. Harvey, J. Nelson, and K. Onak. Sketching and streaming entropy via approximation theory. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 489–498, 2008.

[24] T. S. Jayram. Hellinger strikes back: A note on the multi-party information complexity of AND. In *Proceedings of the 12th International Workshop on Randomization and Computation (RANDOM)*, pages 562–573, 2009.

[25] T. S. Jayram, R. Kumar, and D. Sivakumar. The one-way communication complexity of Hamming distance. *Theory of Computing*, 4(1):129–135, 2008.

[26] T. S. Jayram and D. P. Woodruff. Optimal bounds for Johnson-Lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms*, 9(3):26, 2013.

[27] H. Jowhari, M. Saglam, and G. Tardos. Tight bounds for Lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 49–58, 2011.

[28] D. M. Kane, J. Nelson, E. Porat, and D. P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 745–754, 2011.

[29] D. M. Kane, J. Nelson, and D. P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1161–1178, 2010.

[30] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 41–52, 2010.

[31] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: methods, evaluation, and applications. In *Internet Measurement Comference*, pages 234–247, 2003.

[32] K. G. Larsen. Higher cell probe lower bounds for evaluating polynomials. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 293–301, 2012.

[33] K. G. Larsen. *Models and Techniques for Proving Data Structure Lower Bounds*. PhD thesis, Aarhus University, 2013.

[34] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1969.

[35] D. Moore, 2001. `http://www.caida.org/research/security/code-red/`.

[36] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[37] J. Nelson, H. L. Nguyễn, and D. P. Woodruff. On deterministic sketching and streaming for sparse recovery and norm estimation. *Lin. Alg. Appl.*, 441:152–167, Jan. 2014. Preliminary version in RANDOM 2012.

[38] R. Panigrahy, K. Talwar, and U. Wieder. Lower bounds on near neighbor search via metric expansion. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 805–814, 2010.

[39] M. Pătraşcu. *Lower bound techniques for data structures*. PhD thesis, Massachusetts Institute of Technology, 2008.

[40] M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.

[41] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[42] M. Thorup and Y. Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM J. Comput.*, 41(2):293–331, 2012.

[43] D. P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 167–175, 2004.

[44] D. P. Woodruff. *Efficient and Private Distance Approximation in the Communication and Streaming Models*. PhD thesis, Massachusetts Institute of Technology, 2007.

[45] A. C.-C. Yao. Should tables be sorted? *J. ACM*, 28:615–628, 1981.