Sketching Algorithms

Jelani Nelson

October 1, 2020

Contents

Chapter 1

Introduction

Sketching and streaming. A *sketch* $C(X)$ of some data set X with respect to some function f is a *compression* of X that allows us to compute, or approximately compute, $f(X)$ given access only to $C(X)$. Sometimes f has 2 (or multiple) arguments, and for data X and Y, we want to compute $f(X, Y)$ given $C(X), C(Y)$. For example, if two servers on a network want to compute some similarity or distance measure on their data, one can simply send the sketch to another (or each to a third party), which reduces network bandwith compared to sending the entirety of X, Y.

As a trivial example, consider the case that Alice has a data set X which is a set of integers, and Bob has a similar data set Y. They want to compute $f(X, Y) = \sum_{z \in X \cup Y} z$. Then each party can let the sketch of their data simply be the sum of all elements in their data set.

When designing *streaming* algorithms, we want to maintain a sketch $C(X)$ on the fly as X is updated. In the previous example, if say Alice's data set is being inserted into on the fly then she can of course maintain a sketch by keeping a running sum. The streaming setting appears in many scenarios, such as for example an Internet router monitoring network traffic, or a search engine monitoring a query stream.

1.1 Probability Review

We will mainly be dealing with discrete random variables; we consider random variables taking values in some countable subset $S \subset \mathbb{R}$. Recall the expectation of X is defined to be

$$\mathbb{E}\, X = \sum_{j \in S} j \cdot \mathbb{P}(X = j).$$

We now state a few basic lemmas and facts without proof.

Lemma 1.1.1 (Linearity of expectation)**.**

$$\mathbb{E}(X + Y) = \mathbb{E}\, X + \mathbb{E}\, Y \tag{1.1}$$

Lemma 1.1.2 (Markov)**.** *If X is a nonnegative random variable, then*

$$\forall \lambda > 0,\ \mathbb{P}(X > \lambda) < \frac{\mathbb{E}\, X}{\lambda}$$

Lemma 1.1.3 (Chebyshev)**.**

$$\forall \lambda > 0, \mathbb{P}(|X - \mathbb{E}\, X| > \lambda) < \frac{\mathbb{E}(X - \mathbb{E}\, X)^2}{\lambda^2} \tag{1.2}$$

Proof. $\mathbb{P}(|X - \mathbb{E}\,X| > \lambda) = \mathbb{P}((X - \mathbb{E}\,X)^2 > \lambda^2)$, and thus the claim follows by Markov's inequality. □

Rather than the second moment, one can also consider larger moments to obtain:

$$\forall p \geq 1, \forall \lambda > 0,\ \mathbb{P}(|X - \mathbb{E}\,X| > \lambda) < \frac{\mathbb{E}\,|X - \mathbb{E}\,X|^p}{\lambda^p}. \tag{1.3}$$

By a calculation and picking p optimally (or by Markov's inequality on the moment-generating function e^{tX} and appropriately picking t), one can also obtain the following "Chernoff bound".

Theorem 1.1.4 (Chernoff bound)**.** *Suppose $X_1, \ldots, X_n$ are independent random variables with $X_i \in [0, 1]$. Let $X = \sum_i X_i$ and write $\mu := \mathbb{E}\,X$. Then*

$$\forall \lambda > 0,\ \mathbb{P}(X > (1 + \lambda)\mu) < \left(\frac{e^\lambda}{(1+\lambda)^{1+\lambda}}\right)^\mu \quad \textit{(upper tail)} \tag{1.4}$$

and also

$$\forall \lambda > 0,\ \mathbb{P}(X < (1 - \lambda)\mu) < \left(\frac{e^{-\lambda}}{(1-\lambda)^{1-\lambda}}\right)^\mu \quad \textit{(lower tail)} \tag{1.5}$$

Proof. We give the standard proof via bounding the moment-generating function (MGF), and only for the upper tail and only when each X_i is a Bernoulli random variable with parameter p_i (equal to 1 with probability p_i and 0 otherwise); standard references provide proofs of the most general form. The proof for the lower tail is similar. First,

$$\mathbb{P}(X > (1 + \lambda)\mu) = \mathbb{P}(e^{tX} > e^{t(1+\lambda)\mu})$$

for any $t \in \mathbb{R}$ since the map $x \mapsto e^{tx}$ is strictly increasing. As e^{tX} is a nonnegative random variable, we can apply Markov's inequality.

$$\begin{aligned}
\mathbb{P}(e^{tX} > e^{t(1+\lambda)\mu}) &< e^{-t(1+\lambda)\mu} \cdot \mathbb{E}\,e^{tX} && \text{(Markov)} \\
&= e^{-t(1+\lambda)\mu} \cdot \mathbb{E}\,e^{t\sum_i X_i} \\
&= e^{-t(1+\lambda)\mu} \cdot \mathbb{E}\left[\prod_i e^{tX_i}\right] \\
&= e^{-t(1+\lambda)\mu} \cdot \prod_i \mathbb{E}\,e^{tX_i} && \text{(independence of the } X_i\text{)} \\
&= e^{-t(1+\lambda)\mu} \cdot \prod_i \left(p_i e^t + (1 - p_i)\right) \\
&= e^{-t(1+\lambda)\mu} \cdot \prod_i \left(1 + p_i(e^t - 1)\right) \\
&\leq e^{-t(1+\lambda)\mu} \cdot \prod_i e^{p_i(e^t-1)} && (1 + a \leq e^a) \\
&= e^{-t(1+\lambda)\mu} \cdot e^{(e^t-1)\sum_i p_i} \\
&= e^{(e^t-1-t-\lambda t)\mu} \\
&= \left(\frac{e^\lambda}{(1+\lambda)^{1+\lambda}}\right)^\mu && \text{(set } t = \ln(1+\lambda)\text{)}
\end{aligned} \tag{1.6}$$

The lower tail is similar though one writes $\mathbb{P}(X < (1 - \lambda)\mu) = \mathbb{P}(-X > -(1 - \lambda)\mu) = \mathbb{P}(e^{-tX} > e^{-t(1-\lambda)\mu}$ then does similar calculations and optimizes choice of t. □

Remark 1.1.5. The upper tail has two regimes of interest for λ: $\lambda \ll 1$ and $\lambda \gg 1$. Note when $\lambda < 1$, we have

$$\begin{aligned}\frac{e^\lambda}{(1+\lambda)^{1+\lambda}} &= \frac{e^\lambda}{e^{(1+\lambda)\ln(1+\lambda)}} \\ &= \frac{e^\lambda}{e^{(1+\lambda)(\lambda-\lambda^2/2+O(\lambda^3))}} && \text{(Taylor's theorem)} \\ &= e^{-\lambda^2\mu/2+O(\lambda^3\mu)}.\end{aligned}$$

Rather than use an $O(\lambda^3)$ bound from Taylor's theorem, one can show a more precise inequality $\ln(1+\lambda) \geq 2\lambda/(2+\lambda)$ for any $\lambda \geq 0$ and use it achieve the final upper bound $e^{-\lambda^2\mu/3}$.

The second regime of interest is large λ, i.e. $\lambda \gg 1$ (specifically $\lambda > 2e-1$). In this case, $e/(1+\lambda)$ is less than $1/2$ and is also $O(1/\lambda)$ so that the overall upper tail bound is

$$\lambda^{-\Omega(\lambda\mu)}. \tag{1.7}$$

For the lower tail, as we are only ever interested in $\lambda \leq 1$ (as $X < 0$ trivially has probability 0 of occurring), logarithmic approximations as for the upper tail can be used to obtain the lower tail bound $e^{-\lambda^2\mu/2}$. By combining the upper and lower tails via a union bound,

$$\mathbb{P}(|X-\mu| > \lambda\mu) < e^{-\lambda^2\mu/3} + e^{-\lambda^2\mu/2} < 2e^{-\lambda^2\mu/3}. \tag{1.8}$$

We also have the following, similar "Hoeffding bound", which can also be proven via Markov's inequality applied to the MGF and optimizing choice of t.

Theorem 1.1.6 (Hoeffding bound)**.** *Suppose $X_1, \ldots, X_n$ are i.i.d.* Bernoulli(p) *random variables for some $p \in (0,1)$. Then for any $\epsilon \in (0,1)$,*

$$\mathbb{P}\left(\sum_{i=1}^n X_i > (p+\epsilon)n\right) < e^{-2\epsilon^2 n},$$

and

$$\mathbb{P}\left(\sum_{i=1}^n X_i < (p-\epsilon)n\right) < e^{-2\epsilon^2 n}.$$

Another inequality we will make use of is Khintchine's inequality. It essentially says that when σ_i are i.i.d. Rademachers and x_1 are scalars, $\sum_i \sigma_i x_i$ is what is known as "subgaussian" (i.e. decays at least as fast as a gaussian of some variance).

Theorem 1.1.7 (Khintchine)**.** *Let $\sigma_1, \ldots, \sigma_n$ be Rademacher random variables (i.e. uniform in $\{-1,1\}$) and independent, and $x \in \mathbb{R}^n$ is fixed. Then $\forall \lambda > 0$, $\mathbb{P}(|\langle \sigma, x\rangle| > \lambda) \leq 2e^{-\lambda^2/2\|x\|_2^2}$.*

Proof. Define $X = \sum_i \sigma_i x_i$. Then

$$\begin{aligned}\mathbb{E}\, e^{tX} &= \prod_i \mathbb{E}\, e^{t\sigma_i x_i} \\ &= \prod_i \frac{1}{2}(e^{-tx_i} + e^{tx_i}) \\ &= \prod_i \left(1 + \frac{(tx_i)^2}{2!} + \frac{(tx_i)^4}{4!} + \frac{(tx_i)^6}{6!} + \ldots\right) && \text{(Taylor expansion)}\end{aligned}$$

$$\leq \prod_i \left(1 + \frac{(tx_i)^2}{1! \cdot 2} + \frac{(tx_i)^4}{2! \cdot 2^2} + \frac{(tx_i)^6}{3! \cdot 2^3} + \dots\right)$$
$$= \prod_i e^{t^2 x_i^2/2}$$
$$= e^{t^2 \|x\|_2^2/2} \tag{1.9}$$

Eq. (1.9) is exactly the MGF of a gaussian random variable g with mean zero and variance $\|x\|_2^2$. To see that this implies the desired tail bound, by Markov's inequality we then have $\mathbb{P}(X > \lambda) = \mathbb{P}(e^{tX} > e^{t\lambda}) < e^{-t\lambda + t^2\|x\|_2^2/2}$ by Markov's inequality and Eq. (1.9). This is equal to $e^{-\lambda^2/(2\|x\|_2^2)}$ by setting $t = \lambda/\|x\|_2^2$. Since we are looking at the event $|X| > \lambda$, we must also bound $\mathbb{P}(-X > \lambda) = \mathbb{P}(e^{-tX} > e^{t\lambda}) < e^{-t\lambda + (-t)^2\|x\|_2/2}$, which is the same as above. Thus the claim holds, since $\mathbb{P}(|X| > \lambda) = \mathbb{P}(X > \lambda) + \mathbb{P}(-X > \lambda)$ as these are disjoint events (or alternatively it is good enough to say the right hand side is an upper bound, which holds via the union bound). □

We also make use of the following inequality.

Theorem 1.1.8 (Jensen's inequality)**.** *Let X be a random variable supported in $\mathbb{R}$, and suppose $\varphi : \mathbb{R} \to \mathbb{R}$ is convex. Then $\varphi(\mathbb{E}\, X) \leq \mathbb{E}\, \varphi(X)$.*

The following is a corollary of Jensen's inequality that we frequently use, where $\|X\|_p$ denotes $(\mathbb{E}\,|X|^p)^{1/p}$ is a norm for $p \geq 1$.

Lemma 1.1.9. *For $1 \leq p \leq q$, $\|X\|_p \leq \|X\|_q$.*

Proof. Define $\varphi(x) = |x|^{q/p}$, which is convex. Then by Theorem 1.1.8 applied to the random variable $|X|^p$,

$$(\mathbb{E}\,|X|^p)^{q/p} = \varphi(\mathbb{E}\,|X|^p) \leq \mathbb{E}\,\varphi(|X|^p) = \mathbb{E}\,|X|^q.$$

Raising both sides to the $1/q$th power yields the result. □

Lemma 1.1.10 (Symmetrization / Desymmetrization)**.** *Let $Z_1, \dots, Z_n$ be independent random variables. Let $r_1, \dots, r_n$ be independent Rademachers. Then*

$$\|\sum_i Z_i - \mathbb{E}\sum_i Z_i\|_p \leq 2 \cdot \|\sum_i r_i Z_i\|_p \text{ (symmetrization inequality)}$$

and

$$(1/2) \cdot \|\sum_i r_i (Z_i - \mathbb{E}\, Z_i)\|_p \leq \|\sum_i Z_i\|_p \text{ (desymmetrization inequality).}$$

Proof. For the first inequality, let $Y_1, \dots, Y_n$ be independent of the Z_i but identically distributed to them. Then

$$\begin{aligned}
\|\sum_i Z_i - \mathbb{E}\sum_i Z_i\|_p &= \|\sum_i Z_i - \mathop{\mathbb{E}}_Y \sum_i Y_i\|_p \\
&\leq \|\sum_i (Z_i - Y_i)\|_p && \text{(Jensen)} \\
&= \|\sum_i r_i (Z_i - Y_i)\|_p && (1.10) \\
&\leq 2 \cdot \|\sum_i r_i X_i\|_p && \text{(triangle inequality)}
\end{aligned}$$

Eq. (1.10) follows since the $X_i - Y_i$ are independent across i and symmetric.

For the second inequality, let Y_i be as before. Then

$$\begin{aligned}\|\sum_i r_i(Z_i - \mathbb{E}\, Z_i)\|_p &= \|\mathop{\mathbb{E}}_Y \sum_i r_i(Z_i - Y_i)\|_p \\ &\le \|\sum_i r_i(Z_i - Y_i)\|_p && \text{(Jensen)} \\ &= \|\sum_i (Z_i - Y_i)\|_p \\ &\le 2 \cdot \|\sum_i Z_i\|_p && \text{(triangle inequality)}\end{aligned}$$

□

Lemma 1.1.11 (Decoupling [dlPnG99])**.** *Let $x_1, \dots, x_n$ be independent and mean zero, and $x'_1, \dots, x'_n$ identically distributed as the x_i and independent of them. Then for any $(a_{i,j})$ and for all $p \ge 1$*

$$\|\sum_{i \ne j} a_{i,j} x_i x_j\|_p \le 4 \|\sum_{i,j} a_{i,j} x_i x'_j\|_p$$

Proof. Let $\eta_1, \dots, \eta_n$ be independent Bernoulli random variables each of expectation $1/2$. Then

$$\begin{aligned}\|\sum_{i \ne j} a_{i,j} x_i x_j\|_p &= 4 \cdot \|\mathop{\mathbb{E}}_\eta \sum_{i \ne j} a_{i,j} x_i x_j |\eta_i||1 - \eta_j|\|_p \\ &\le 4 \cdot \|\sum_{i \ne j} a_{i,j} x_i x_j \eta_i (1 - \eta_j)\|_p && \text{(Jensen)} \qquad (1.11)\end{aligned}$$

Hence there must be some fixed vector $\eta' \in \{0,1\}^n$ which achieves

$$\|\sum_{i \ne j} a_{i,j} x_i x_j \eta_i (1 - \eta_j)\|_p \le \|\sum_{i \in S} \sum_{j \notin S} a_{i,j} x_i x_j\|_p$$

where $S = \{i : \eta'_i = 1\}$. Let x_S denote the $|S|$-dimensional vector corresponding to the x_i for $i \in S$. Then

$$\begin{aligned}\|\sum_{i \in S} \sum_{j \notin S} a_{i,j} x_i x_j\|_p &= \|\sum_{i \in S} \sum_{j \notin S} a_{i,j} x_i x'_j\|_p \\ &= \|\mathop{\mathbb{E}}_{x_S} \mathop{\mathbb{E}}_{x'_{\bar{S}}} \sum_{i,j} a_{i,j} x_i x'_j\|_p \ (\mathbb{E}\, x_i = \mathbb{E}\, x'_j = 0) \\ &\le \|\sum_{i,j} a_{i,j} x_i x'_j\|_p && \text{(Jensen)}\end{aligned}$$

□

The following proof of the Hanson-Wright was shared to me by Sjoerd Dirksen (personal communication). A newer proof which we do not cover here, using more modern tools that allow extension to subgaussian variables and not just Rademachers, is given in [RV13]. Recall the Frobenius norm is defined by $\|A\|_F := (\sum_{i,j} A_{i,j}^2)^{1/2}$, and the operator norm by $\|A\| = \sup_{\|x\|_2 = \|y\|_2 = 1} x^\top A y$.

Theorem 1.1.12 (Hanson-Wright inequality [HW71])**.** *For $\sigma_1, \dots, \sigma_n$ independent Rademachers and $A \in \mathbb{R}^{n \times n}$, for all $p \geq 1$*

$$\|\sigma^T A \sigma - \mathbb{E}\, \sigma^T A \sigma\|_p \lesssim \sqrt{p} \cdot \|A\|_F + p \cdot \|A\|.$$

Proof. Without loss of generality we assume in this proof that $p \geq 2$ (so that $p/2 \geq 1$). Then

$$\begin{aligned}
\|\sigma^T A \sigma - \mathbb{E}\, \sigma^T A \sigma\|_p &\lesssim \|\sigma^T A \sigma'\|_p && \text{(Lemma 1.1.11)} && (1.12)\\
&\lesssim \sqrt{p} \cdot \| \|Ax\|_2 \|_p && \text{(Khintchine)} && (1.13)\\
&= \sqrt{p} \cdot \| \|Ax\|_2^2 \|_{p/2}^{1/2} && && (1.14)\\
&\leq \sqrt{p} \cdot \| \|Ax\|_2^2 \|_p^{1/2} && &&\\
&\leq \sqrt{p} \cdot (\|A\|_F^2 + \| \|Ax\|_2^2 - \mathbb{E}\, \|Ax\|_2^2 \|_p)^{1/2} && \text{(triangle inequality)} &&\\
&\leq \sqrt{p} \cdot \|A\|_F + \sqrt{p} \cdot \| \|Ax\|_2^2 - \mathbb{E}\, \|Ax\|_2^2 \|_p^{1/2} && &&\\
&\lesssim \sqrt{p} \cdot \|A\|_F + \sqrt{p} \cdot \|x^T A^T A x'\|_p^{1/2} && \text{(Lemma 1.1.11)} &&\\
&\lesssim \sqrt{p} \cdot \|A\|_F + p^{3/4} \cdot \| \|A^T A x\|_2 \|_p^{1/2} && \text{(Khintchine)} &&\\
&\lesssim \sqrt{p} \cdot \|A\|_F + p^{3/4} \cdot \|A\|^{1/2} \cdot \| \|Ax\|_2 \|_p^{1/2} && && (1.15)
\end{aligned}$$

Writing $E = \| \|Ax\|_2 \|_p^{1/2}$ and comparing Eq. (1.13) and Eq. (1.15), we see that for some constant $C > 0$,

$$E^2 - C p^{1/4} \|A\|^{1/2} E - C \|A\|_F \leq 0.$$

Thus E must be smaller than the larger root of the above quadratic equation, implying our desired upper bound on E^2. □

Remark 1.1.13. The "square root trick" in the proof of the Hanson-Wright inequality above is quite handy and can be used to prove several moment inequalities (for example, it can be used to prove Bernstein inequality). As far as I am aware, the trick was first used in a work of Rudelson [Rud99] on operator norms of certain random matrices.

Remark 1.1.14. We could have upper bounded Eq. (1.14) by

$$\sqrt{p} \cdot \|A\|_F + \sqrt{p} \cdot \| \|Ax\|_2^2 - \mathbb{E}\, \|Ax\|_2^2 \|_{p/2}^{1/2}$$

by the triangle inequality. Now notice we have bounded the pth central moment of a symmetric quadratic form Eq. (1.12) by the $p/2$th moment also of a symmetric quadratic form. Writing $p = 2^k$, this observation leads to a proof by induction on k, which was the approach used in [DKN10].

We have stated a moment version of the Hanson-Wright inequality, but we show now this is equivalent to a tail bound. Below we prove a lemma which lets us freely obtain tail bounds from moment bounds and vice versa (often we prove a moment bound and later invoke a tail bound, or vice versa, without even mentioning any justification).

Lemma 1.1.15. *Let Z be a scalar random variable. Consider the following statements:*

(1a) There exists $\sigma > 0$ s.t. $\forall p \geq 1$, $\|Z\|_p \leq C_1 \sigma \sqrt{p}$.

(1b) There exists $\sigma > 0$ s.t. $\forall \lambda > 0$, $\mathbb{P}(|Z| > \lambda) \leq C_2 e^{-C_2' \lambda^2 / \sigma^2}$.

(2a) There exists $K > 0$ s.t. $\forall p \geq 1$, $\|Z\|_p \leq C_3 K p$.

(2b) There exists $K > 0$ *s.t.* $\forall\ \lambda > 0$, $\mathbb{P}(|Z| > \lambda) \leq C_4 e^{-C_4'\lambda/K}$.

(3a) There exist $\sigma, K > 0$ *s.t.* $\forall\ p \geq 1$, $\|Z\|_p \leq C_5(\sigma\sqrt{p} + Kp)$.

(3b) There exist $\sigma, K > 0$ *s.t.* $\forall\ \lambda > 0$, $\mathbb{P}(|Z| > \lambda) \leq C_6(e^{-C_6'\lambda^2/\sigma^2} + e^{-C_6'\lambda/K})$.

Then 1a is equivalent to 1b, 2a is equivalent to 2b, and 3a is equivalent to 3b, where the constants C_i, C_i' *in each case change by at most some absolute constant factor.*

Proof. We will show only that 1a is equivalent to 1b; the other cases are argued identically.

To show that 1a implies 1b, by Markov's inequality

$$\mathbb{P}(Z > \lambda) \leq \lambda^{-p} \cdot \mathbb{E}\,|Z|^p \leq \left(\frac{C_1^2\sigma^2}{\lambda^2 p}\right)^{p/2}.$$

Statement 1b follows by choosing $p = \max\{1, 2C_1^2\lambda^2/\sigma^2\}$.

To show that 1b implies 1a, by integration by parts we have

$$\mathbb{E}\,|Z|^p = \int_0^\infty px^{p-1}\,\mathbb{P}(|Z| > \lambda)d\lambda \leq 2C_2 p \cdot \int_0^\infty px^{p-1} \cdot e^{-C_2'\lambda^2/\sigma^2}d\lambda.$$

The integral on the right hand side is exactly the pth moment of a gaussian random variable with mean zero and variance $\sigma'^2 = \sigma^2/(2C_2')$. Statement 1a then follows since such a gaussian has p-norm $\Theta(\sigma'\sqrt{p})$. □

Corollary 1.1.16. *For* $\sigma_1, \ldots, \sigma_n$ *independent Rademachers and* $A \in \mathbb{R}^{n\times n}$, *for all* $\lambda > 0$

$$\mathop{\mathbb{P}}_{\sigma}(|\sigma^T A\sigma - \mathbb{E}\,\sigma^T A\sigma| > \lambda) \lesssim e^{-C\lambda^2/\|A\|_F^2)} + e^{-C\lambda/\|A\|}.$$

Proof. Combine Theorem 1.1.12 with item (3b) of Lemma 1.1.15. □

Chapter 2

Counting Problems

2.1 Approximate counting

In the following, we discuss a problem first studied in [Mor78].

Problem. Our algorithm must monitor a sequence of events, then at any given time output (an estimate of) the number of events thus far. More formally, this is a data structure maintaining a single integer n and supporting the following two operations:

- **update():** increment n by 1
- **query():** output (an estimate of) n

Before any operations are performed, it is assumed that n starts at 0. Of course a trivial algorithm maintains n using $O(\log n)$ bits of memory (a counter). Our goal is to use much less space than this. It is not too hard to prove that it is impossible to solve this problem exactly using $o(\log n)$ bits of space. Thus we would like to answer **query()** with some estimate $\tilde{n}$ of n satisfying

$$\mathbb{P}(|\tilde{n} - n| > \varepsilon n) < \delta, \tag{2.1}$$

for some $0 < \varepsilon, \delta < 1$ that are given to the algorithm up front.

The algorithm of Morris provides such an estimator for some ε, δ that we will analyze shortly. The algorithm works as follows:

1. Initialize $X \leftarrow 0$.
2. For each update, increment X with probability $\frac{1}{2^X}$.
3. For a query, output $\tilde{n} = 2^X - 1$.

Intuitively, the variable X is attempting to store a value that is $\approx \log_2 n$. Before giving a rigorous analysis in Subsection 2.1.1, we first give a probability review.

2.1.1 Analysis of Morris' algorithm

Let X_n denote X in Morris' algorithm after n updates.

Claim 2.1.1.

$$\mathbb{E}\, 2^{X_n} = n + 1.$$

Proof. We prove by induction on n. The base case is clear, so we now show the inductive step.

We have

$$\begin{aligned}
\mathbb{E}\, 2^{X_{n+1}} &= \sum_{j=0}^{\infty} \mathbb{P}(X_n = j) \cdot \mathbb{E}(2^{X_{n+1}} | X_n = j) \\
&= \sum_{j=0}^{\infty} \mathbb{P}(X_n = j) \cdot (2^j(1 - \frac{1}{2^j}) + \frac{1}{2^j} \cdot 2^{j+1}) \\
&= \sum_{j=0}^{\infty} \mathbb{P}(X_n = j) 2^j + \sum_j \mathbb{P}(X_n = j) \\
&= \mathbb{E}\, 2^{X_n} + 1 \\
&= (n+1) + 1
\end{aligned} \tag{2.2}$$

□

It is now clear why we output our estimate of n as $\tilde{n} = 2^X - 1$: it is an unbiased estimator of n. In order to show Eq. (2.1) however, we will also control on the variance of our estimator. This is because, by Chebyshev's inequality,

$$\mathbb{P}(|\tilde{n} - n| > \varepsilon n) < \frac{1}{\varepsilon^2 n^2} \cdot \mathbb{E}(\tilde{n} - n)^2 = \frac{1}{\varepsilon^2 n^2} \mathbb{E}(2^X - 1 - n)^2. \tag{2.3}$$

When we expand the above square, we find that we need to control $\mathbb{E}\, 2^{2X_n}$. The proof of the following claim is by induction, similar to that of Claim 2.1.1.

Claim 2.1.2.

$$\mathbb{E}(2^{2X_n}) = \frac{3}{2}n^2 + \frac{3}{2}n + 1. \tag{2.4}$$

This implies $\mathbb{E}(\tilde{n} - n)^2 = (1/2)n^2 - (1/2)n - 1 < (1/2)n^2$, and thus

$$\mathbb{P}(|\tilde{n} - n| > \varepsilon n) < \frac{1}{\varepsilon^2 n^2} \cdot \frac{n^2}{2} = \frac{1}{2\varepsilon^2}, \tag{2.5}$$

which is not particularly meaningful since the right hand side is only better than $1/2$ failure probability when $\varepsilon \geq 1$ (which means the estimator may very well always be 0!).

2.1.2 Morris+

To decrease the failure probability of Morris' basic algorithm, we instantiate s independent copies of Morris' algorithm and average their outputs. That is, we obtain independent estimators $\tilde{n}_1, \ldots, \tilde{n}_s$ from independent instantiations of Morris' algorithm, and our output to a query is

$$\tilde{n} = \frac{1}{s} \sum_{i=1}^{s} \tilde{n}_i$$

Since each $\tilde{n}_i$ is an unbiased estimator of n, so is their average. Furthermore, since variances of independent random variables add, and multiplying a random variable by some constant $c = 1/s$ causes the variance to be multiplied by c^2, the right hand side of (Eq. (2.5)) becomes

$$\mathbb{P}(|\tilde{n} - n| > \varepsilon n) < \frac{1}{2s\varepsilon^2} < \delta$$

for $s > 1/(2\varepsilon^2\delta) = \Theta(1/(\varepsilon^2\delta))$.

2.1.3 Morris++

It turns out there is a simple technique (which we will see often) to reduce the dependence on the failure probability δ from $1/\delta$ to $\log(1/\delta)$. The technique is as follows.

We run t instantiations of Morris+, each with failure probability $\frac{1}{3}$. Thus, for each one, $s = \Theta(1/\varepsilon^2)$. We then output the median estimate from all the s Morris+ instantiations. Note that the expected number of Morris+ instantiations that succeed is at least $2t/3$. For the median to be a bad estimate, at most half the Morris+ instantiations can succeed, implying the number of succeeding instantiations deviated from its expectation by at least $t/6$. Define

$$Y_i = \begin{cases} 1, & \text{if the } i\text{th Morris+ instantiation succeeds.} \\ 0, & \text{otherwise.} \end{cases} \tag{2.6}$$

Then by the Hoeffding bound,

$$\mathbb{P}\left(\sum_i Y_i \le \frac{t}{2}\right) \le \mathbb{P}\left(\sum_i Y_i - \mathbb{E}\sum_i Y_i| < -t/6\right) < e^{-2(1/6)^2 t} \le \delta \tag{2.7}$$

for $t \ge \lceil 18 \ln(1/\delta) \rceil$.

Overall space complexity. When we unravel Morris++, it is running a total of $st = \Theta(\lg(1/\delta)/\varepsilon^2)$ instantiations of the basic Morris algorithm. Now note that once any given Morris counter X reaches the value $\lg(stn/\delta)$, the probabilitye that it is incremented at any given moment is at most $\delta/(nst)$. Thus the probability that it is incremented at all in the next n increments is at most $\delta/(st)$. Thus by a union bound, with probability $1-\delta$ none of the st basic Morris instantiations ever stores a value larger than $\lg(stn/\delta)$, which takes $O(\lg\lg(stn/\delta))$ bits. Thus the total space complexity is, with probability $1-\delta$, at most

$$O(\varepsilon^{-2}\lg(1/\delta)(\lg\lg(n/(\varepsilon\delta)))).$$

In particular, for constant ε, δ (say each $1/100$), the total space complexity is $O(\lg\lg n)$ with constant probability. This is exponentially better than the $\log n$ space achieved by storing a counter!

Remark 2.1.3. As we will continue to see, designing some random process and associated unbiased estimator with bounded variance for some desired statistic that can be maintained and computed in small space is a common strategy. One can then take the median of means of several copies of this basic structure to then obtain $1+\varepsilon$-approximation with success probability $1-\delta$. In the particular case of approximate counting, this led to a space blow-up of $\Theta(\log(1/\delta)/\varepsilon^2)$ though it is possible to do better by a more tailored approach. Specifically, Morris instead suggests adjusting the increment probability from $1/2^X$ to $1/(1+\alpha)^X$ for small α then estimating n as $\tilde{n} := ((1+\alpha)^X - 1)/\alpha$. This makes intuitive sense: if we increment with probability 1.0^X, then we have a simple deterministic counter, which has zero variance but poor memory performance. Meanwhile if we increment with probability 0.5^X then the memory usage is reduced at the cost of higher variance. One may then intuit that if incrementing with probability $1/(1+\alpha)^X$, space usage increases while variance decreases as $\alpha \downarrow 0$, which is indeed the case. By bounding variance and applying Chebyshev's inequality, it is possible to show that $\alpha = \Theta(\varepsilon^2\delta)$ leads to $(1+\varepsilon)$-approximation with probability $1-\delta$ while using space $O(\log(1/\varepsilon) + \log\log n + \log(1/\delta))$ with high probability [Mor78, Fla85]. In fact, it is even possible to improve the analysis to show $O(\log(1/\varepsilon) + \log\log n + \log\log(1/\delta))$ space suffices via a different argument, and to show that this bound is optimal [NY20].

2.2 Distinct elements

We next consider another counting problem: the *count distinct* or *distinct elements problem*, also known as the F_0 problem, defined as follows. We are given a stream of integers $i_1, \ldots, i_m \in [n]$ where $[n]$ denotes the set $\{1, 2, \ldots, n\}$. We would like to output the number of *distinct* elements seen in the stream. As with Morris' approximate counting algorithm, our goal will be to minimize our space consumption.[1]

There are two straightforward solutions as follows:

1. **Solution 1:** keep a bit array of length n, initialized to all zeroes. Set the ith bit to 1 whenever i is seen in the stream (n bits of memory).
2. **Solution 2:** Store the whole stream in memory explicitly ($m \lceil \log_2 n \rceil$ bits of memory).

We can thus solve the problem exactly using $\min\{n, m \lceil \log_2 n \rceil\}$ bits of memory. In order to the reduce the space, we will instead settle for computing some value $\tilde{t}$ s.t. $\mathbb{P}(|t - \tilde{t}| > \varepsilon t) < \delta$, where t denotes the number of distinct elements in the stream. The first work to show that this is possible using small memory (assuming oracle access to certain random hash functions) is that of Flajolet and Martin (FM) [FM85].

2.2.1 Idealized FM algorithm: freely stored randomness

We first discuss the following idealized algorithm:

1. Pick a random function $h : [n] \to [0, 1]$
2. Maintain counter $X = \min_{i \in stream} h(i)$
3. Output $1/X - 1$

Note this algorithm really is idealized, since we cannot afford to store a truly random such function h in $o(n)$ bits (first, because there are n independent random variables $(h(i))_{i=1}^n$, and second because its outputs are real numbers).

Intuition. The value X stored by the algorithm is a random variable that is the minimum of t i.i.d $Unif(0, 1)$ random variables. One can then compute the expectation of what the minimum is, as a function of t, and invert that function in hopes of obtaining (a good approximation to) t.

Claim 2.2.1. $\mathbb{E}\, X = \frac{1}{t+1}$.

Proof.

$$\begin{aligned}
\mathbb{E}\, X &= \int_0^\infty \mathbb{P}(X > \lambda) d\lambda \\
&= \int_0^\infty \mathbb{P}(\forall i \in stream, h(i) > \lambda) d\lambda \\
&= \int_0^\infty \prod_{r=1}^{t} \mathbb{P}(h(i_r) > \lambda) d\lambda
\end{aligned}$$

[1] The reason for the name "F_0" is that if one defines a histogram $x \in \mathbb{R}^n$ with x_i being the number of occurrences of i in the stream, then one can define the *pth moment* F_p as $\sum_i x_i^p$. The number of distinct elements is then the limit of the pth moment for $p \to 0$.

$$= \int_0^1 (1-\lambda)^t d\lambda$$
$$= \frac{1}{t+1}$$

□

We will also need the following claim in order to execute Chebyshev's inequality to bound the failure probability in our final algorithm.

Claim 2.2.2. $\mathbb{E}\, X^2 = \frac{2}{(t+1)(t+2)}$

Proof.

$$\begin{aligned}
\mathbb{E}\, X^2 &= \int_0^1 \mathbb{P}(X^2 > \lambda) d\lambda \\
&= \int_0^1 \mathbb{P}(X > \sqrt{\lambda}) d\lambda \\
&= \int_0^1 (1-\sqrt{\lambda})^t d\lambda \\
&= 2\int_0^1 u^t (1-u) du && \text{(substitution } u = 1-\sqrt{\lambda}\text{)} \\
&= \frac{2}{(t+1)(t+2)}
\end{aligned}$$

□

This gives $Var[X] = \mathbb{E}\, X^2 - (\mathbb{E}\, X)^2 = \frac{t}{(t+1)^2(t+2)}$, or the simpler $Var[X] < (\mathbb{E}\, X)^2 = \frac{1}{(t+1)^2}$.

FM+. To obtain an algorithm providing a randomized approximate guarantee, just as with Morris+ we form an algorithm FM+ which averages together the outputs from s independent instantiations of the basic FM algorithm.

1. Instantiate $s = \lceil 1/(\epsilon^2\delta) \rceil$ FMs independently, FM_1,...,FM_s.
2. Let X_i be the output of FM_i.
3. Upon a query, output $1/Z - 1$, where $Z = \frac{1}{s}\sum_i X_i$.

We have that $\mathbb{E}(Z) = \frac{1}{t+1}$, and $Var(Z) = \frac{1}{s}\frac{t}{(t+1)^2(t+2)} < \frac{1}{s(t+1)^2}$.

Claim 2.2.3. $\mathbb{P}(|Z - \frac{1}{t+1}| > \frac{\epsilon}{t+1}) < \delta$

Proof. We apply Chebyshev's inequality.

$$\mathbb{P}(|Z - \frac{1}{t+1}| > \frac{\varepsilon}{t+1}) < \frac{(t+1)^2}{\varepsilon^2}\frac{1}{s(t+1)^2} = \eta$$

□

Claim 2.2.4. $\mathbb{P}(|(\frac{1}{Z} - 1) - t| > O(\varepsilon)t) < \eta$

Proof. If $t = 0$ (the empty stream), the claim is trivial. Otherwise, by the previous claim, with probability $1 - \eta$ we have

$$\frac{1}{(1 \pm \epsilon)\frac{1}{t+1}} - 1 = (1 \pm O(\varepsilon))(t+1) - 1 = (1 \pm O(\varepsilon))t \pm O(\varepsilon) = (1 \pm O(\varepsilon))t$$

□

FM++. To obtain our final algorithm, again as with Morris++ we take the median output from multiple independent instantiations of FM+.

1. Instantiate $q = \lceil 18 \ln(1/\delta) \rceil$ independent copies of FM+ with $\eta = 1/3$.
2. Output the median $\hat{t}$ of $\{1/Z_j - 1\}_{j=1}^{q}$ where Z_j is the output of the jth copy of FM+.

Claim 2.2.5. $\mathbb{P}(|\widehat{t} - t| > \epsilon t) < \delta$

Proof. Let

$$Y_j = \begin{cases} 1 & \text{if } |(1/Z_j - 1) - t| \leq \epsilon t \\ 0 & \text{else} \end{cases}$$

and put $Y = \sum_{j=1}^{q} Y_j$. We have $\mathbb{E}\, Y > 2q/3$ by our choice of η. The probability we seek to bound is equivalent to the probability that the median fails, i.e. at least half of the FM+ estimates have $Y_j = 0$. In other words,

$$\sum_{j=1}^{q} Y_j \leq q/2$$

We then get that

$$\mathbb{P}(\sum Y_j < q/2) \leq \mathbb{P}(Y - \mathbb{E}\, Y < -q/6) \tag{2.8}$$

Then by a Hoeffding bound, the above is at most

$$e^{-2(\frac{1}{6})^2 q} \leq \delta$$

as desired. □

The final space required, ignoring h, is that required to store $O(\frac{\lg(1/\delta)}{\epsilon^2})$ real numbers since $O(\lg(1/\delta))$ copies of FM+ are instantiated, each averaging $O(1/\epsilon^2)$ copies of FM. Each single FM just stores a single number (the minimum hash ever seen).

2.2.2 A non-idealized algorithm: KMV

We next describe a modified algorithm for F_0-estimation which does not assume access to a truly random hash function. Before we continue, we first discuss k-wise independent hash families.

k-wise independent hash families.

Definition 2.2.6. A family $\mathcal{H}$ of functions mapping $[a]$ to $[b]$ is k-wise independent if $\forall j_1, \ldots, j_k \in [b]$ and $\forall$ distinct $i_1, \ldots, i_k \in [a]$,

$$\mathbb{P}_{h \in \mathcal{H}}(h(i_1) = j_1 \wedge \ldots \wedge h(i_k) = j_k) = 1/b^k$$

Example. The set $\mathcal{H}$ of all functions $[a] \to [b]$ is k-wise independent for every k. $|\mathcal{H}| = b^a$ so $h \in \mathcal{H}$ is representable in $a \lg b$ bits.

Example. Let $a = b = q$ for $q = p^r$ a prime power and define $\mathcal{H}_{poly}$ to be the set of all degree $\leq k-1$ polynomials with coefficients in $\mathbb{F}_q$, the finite field of order q. $|\mathcal{H}_{poly}| = q^k$ so $h \in \mathcal{H}$ is representable in $k \lg p = k \lg a$ bits.

Claim 2.2.7 ([WC79])**.** $\mathcal{H}_{poly}$ is k-wise independent.

Proof. Given $((i_r, j_r))_{r=1}^k$, there is exactly one degree at most $k-1$ polynomial h over $\mathbb{F}_q$ with $h(i_r) = j_r$ for $r = 1, \ldots, k$, via interpolation. Thus the probability

$$\mathop{\mathbb{P}}_{h \in \mathcal{H}}(h(i_1) = j_1 \wedge \ldots \wedge h(i_k) = j_k)$$

exactly equals $1/|\mathcal{H}_{poly}| = 1/p^k = 1/b^k$. □

We now present an algorithm of [BYJK+02], later known as the "KMV algorithm" (for "k minimum values"). We will assume $1/\varepsilon^2 < n$, since we will be shooting for a space bound that is at least $1/\varepsilon^2$, and there is always a trivial solution for exact F_0 computation using n bits. We also assume, without loss of generality, that $\varepsilon < 1/2$.

The algorithm is quite similar to the idealized FM algorithm, but rather than maintain only the *smallest* hash evaluation, we maintain the k smallest hash evaluations for some appropriately chosen $k \in \Theta(1/\varepsilon^2)$. The intuition is that the smallest hash evaluation leads us to an estimator with high variance (all it takes is for one item to hash to something really tiny, which will throw off our estimator). Meanwhile, if we consider the kth smallest hash evaluation for some large k, then the variance should be smaller since a single item (or small number of items) cannot throw off our statistic by having a wildly small or big hash value. This makes some intuitive sense since the median ($k = t/2$) is the most robust order statistic to outliers. The tradeoff of course is that to keep track of the kth smallest hash value we will use space that grows with k, as we will keep track of all k bottom hash values seen.

Specifically, the KMV algorithm chooses a hash function $h : [n] \to [M]$ for $M = n^3$ from a 2-wise independent hash family (the idea here is to discretize $[0, 1]$ into multiples of $1/M$). We pick $k = \lceil 24/\varepsilon^2 \rceil$. We keep track in memory of the k smallest hash evaluations. If at the time of the query we have seen less than k distinct hash values, then we just output the number of distinct hash values seen. Otherwise, if X is the kth smallest then we output our estimate of $t = F_0$ as $\tilde{t} = kM/X$.

For some intuition: note if we had $t \geq k$ independent hash values in $[0, 1]$, we expect the kth smallest value v to be $k/(t+1)$ (namely, we expect k equally spaced values between 0 and 1). Thus a reasonable estimate for t would be $k/v - 1$. If $t \leq k$ then our output is exactly correct as long as the distinct elements are perfectly hashed, and indeed this happens with high probability since all of $[n]$ is perfectly hashed with large probability by our large choice of M. Otherwise, for $t \geq k \gg 1/\varepsilon^2$, we expect $k/v > 1/\varepsilon^2$, and thus neglecting to subtract the 1 and simply outputting k/v gives the same answer up to a $1 + \varepsilon$ factor. Since in our actual algorithm we discretize $[0, 1]$ into multiples of $1/M$, our value X is actually representing Mv, and thus we output kM/X.

We now provide a more formal analysis. Note that our hash function h is only 2-wise independent! We would like to say that with good probability,

$$(1 - \varepsilon)t \leq \tilde{t} \leq (1 + \varepsilon)t.$$

We consider the two bad events that $\tilde{t}$ is too big or too small, and show that each happens with probability at most 1/6, and thus the probability that either happens is at most 1/3 by the union bound.

First let us consider the case that $\tilde{t} > (1+\varepsilon)t$. Since $\tilde{t} = kM/X$, i.e. $X = kM/\tilde{t}$, this can only happen if at least k distinct indices in the stream hashed to a value smaller than $kM/((1+\varepsilon)t)$. Let Y_i be an indicator random variable for the event that the ith distinct integer in the stream hashed to a value below $kM/((1+\varepsilon)t)$, and let Y denote $\sum_{i=1}^{t} Y_i$. Then $\mathbb{E}\, Y_i < k/((1+\varepsilon)t)$ and thus

$$\mathbb{E}\, Y < k/(1+\varepsilon).$$

We also have $Var[Y_i] < \mathbb{E}\, Y_i^2 = \mathbb{E}\, Y_i < k/((1+\varepsilon)t)$, and thus

$$Var[Y] < k/(1+\varepsilon)$$

as well (note $Var[Y] = \sum_i Var[Y_i]$ since the Y_i are pairwise independent). Thus by Chebyshev's inequality,

$$\mathbb{P}(Y \geq k) \leq \mathbb{P}(|Y - \mathbb{E}\, Y| > k(1 - 1/(1+\varepsilon)) < \frac{k}{1+\varepsilon} \cdot \frac{(1+\varepsilon)^2}{k^2\varepsilon^2} = \frac{1+\varepsilon}{\varepsilon^2 k} < 1/6.$$

We can similarly bound the probablity that $\tilde{t} < (1-\varepsilon)t$. This can only happen if there weren't *enough* distinct indices in the stream that hashed to small values under h. Specifically, let Z_i be an indicator random variable for the event that the ith distinct integer in the stream hashed to a value below $kM/((1-\varepsilon)t)$, and define $Z = \sum_i Z_i$. Then $\tilde{t} < (1-\varepsilon)t$ can only occur if $Z < k$. But we have $k/((1-\varepsilon)t) \geq \mathbb{E}\, Z_i > k/((1-\varepsilon)t) - 1/M \geq (1+\varepsilon)k/t - 1/M$ (the $1/M$ is due to rounding). We note $1/M < \varepsilon k/(4t)$ since $\varepsilon > 1/\sqrt{n}$ and $t < n$. Thus $k/((1-\varepsilon)t) \geq \mathbb{E}\, Z_i > (1 + 3\varepsilon/4)k/t$, implying

$$k/(1-\varepsilon) \geq \mathbb{E}\, Z > (1 + 3\varepsilon/4)k$$

and also since Z is a sum of pairwise independent Bernoullis, again

$$Var[Z] \leq \mathbb{E}\, Z \leq k/(1-\varepsilon).$$

Thus by Chebyshev's inequality,

$$\mathbb{P}(Z < k) < \mathbb{P}(|Z - \mathbb{E}\, Z| > (3/4)\varepsilon k) < \frac{k}{1-\varepsilon} \cdot \frac{16}{9\varepsilon^2 k^2} < \frac{16}{9(1-\varepsilon)} \cdot \frac{1}{\varepsilon^2 k} < 1/6,$$

as desired.

Other comments. It is known, via a different algorithm, that for constant failure probability space $O(1/\varepsilon^2 + \log n)$ is achievable [KNW10b], and furthermore this is optimal [AMS99, Woo04] (also see [TSJ08]). An algorithm that is more commonly used in practice is HyperLogLog [FEFGM07]. Although it assumes access to a truly random hash function, and asymptotically uses a factor $\lg\lg n$ more space than the algorithm of [KNW10b], its performance in practice is superior. For a historical discussion of the development of HyperLogLog and other distinct elements and approximate counting algorithms (e.g. Morris), see [Lum18]. For arbitrary, potentially subconstant failure probability δ, the optimal bound is $\Theta(\varepsilon^{-2}\log(1/\delta) + \log n)$; the upper bound was shown in [Bła20], and the lower bound in [AMS99, JW13].

2.2.3 Another algorithm via geometric sampling

We now describe another approach to designing a small-space algorithm for the distinct elements problem. This algorithm will use more memory than the KMV algorithm, but it serves the pedagogical value of introducing a technique that is common in the design of streaming algorithms: *geometric sampling*. The main idea of geometric sampling is to pick a hash function $h : [n] \to \{0, \dots, \log n\}$ from a 2-wise independent family with elements in the hash function's range having geometrically decreasing probabilities: $\mathbb{P}(h(i) = j) = 2^{-(j+1)}$ for $j < \log n$ (and $\mathbb{P}(h(i) = \log n) = 1/n$)[2]. This hash function h then naturally partitions the stream into $\log n + 1$ different "substreams" $S_0, \dots, S_{\log n}$, where S_r is the stream restricted to $\{i : h(i) = r\}$. Each S_r is then fed into some separate data structure D_r. In the specific case of distinct elements, D_r is a data structure that solves the following promise problem (for some parameter k): if $t \le k$, then the number of distinct elements must be output exactly; otherwise, the data structure may fail in an arbitrary way. This problem has a trivial deterministic solution using $O(k \log n)$ bits of memory: write down the indices of the first k distinct elements seen (and if a $(k+1)$st element is seen, simply write Error). We first show that such an algorithm (with $k = k_1$ a constant) can give a constant-factor approximation to $t := F_0$ with good probability. We then run the same algorithm in parallel with some larger $k = k_2 = C/\varepsilon^2$. Let then r^* be such that $t/2^{r^*} = \Theta(1/\varepsilon^2)$, which we can calculate based on our constant-factor approximation to t (in the case that $t \ll 1/\varepsilon^2$ there is no such r^*, but this case can be solved exactly by keeping track of an extra data structure D' with $k = k_2$ that ignores h and processes all stream updates). In expectation, the number of distinct elements $F_0(r^*)$ at level r^* is $t/2^{r^*}$, and the variance is upper bounded by this quantity as well. Thus by Chebyshev's inequality, $F_0(r^*) = t/2^{r^*} \pm O(\sqrt{t/2^{r^*}}) = (1 \pm \varepsilon)t/2^{r^*}$ with large constant probability. By choosing C large enough, we can also guarantee that $(1+\varepsilon)t/2^{r^*} \le k_2$ so that we typically output $F_0(2^{r^*})$ correctly, exactly. Then multiplying the output of D_{r^*} by 2^{r^*} gives $(1 \pm \varepsilon)t$ with large probability.

A constant-factor approximation. The data structure is as mentioned above with $k_1 = 1$. Given an integer i in the stream, we feed it to the data structure $D_{h(i)}$. Then to obtain a constant-factor approximation, we return 2^ℓ for ℓ being the largest value of $r \in \{0, \dots, \log n\}$ such that $D_r.\mathsf{query}() \neq 0$ (even if D_r reports an error, we count that as not returning 0). Note that ℓ is a random variable, as it depends on h.

In the analysis below, we define $L := \log_2 t$.

Lemma 2.2.8. $\mathbb{P}(\ell > L + 4) \le 1/8$.

Proof. For any r we have $\mathbb{E}_h F_0(r) = t/2^r$. Thus

$$\begin{aligned}
\mathbb{P}(\ell > L+4) &= \mathbb{P}(\exists r > L+4 : F_0(r) > 0) \\
&\le \sum_{r=\lceil L+4 \rceil}^{\log n} \mathbb{P}(F_0(r) > 0) \text{ (union bound)} \\
&= \sum_{r=\lceil L+4 \rceil}^{\log n} \mathbb{P}(F_0(r) \ge 1) \text{ ($F_0(r)$ is an integer)} \\
&= \sum_{r=\lceil L+4 \rceil}^{\log n} \mathbb{P}(F_0(r) \ge \frac{2^r}{t} \cdot \mathbb{E}\, F_0(r))
\end{aligned}$$

[2]One way to implement such h is to select $h' : [n] \to [n]$ from a pairwise independent family and defining $h(i)$ to be the index of the least significant bit of $h'(i)$.

$$\leq \sum_{r=\lceil L+4 \rceil}^{\log n} \frac{t}{2^r}$$
$$< \sum_{q=4}^{\infty} 2^{-q}$$
$$= \frac{1}{8}.$$

□

Lemma 2.2.9. $\mathbb{P}(F_0(\lceil L-4 \rceil) = 0) < 1/8$.

Proof. Write $q = F_0(\lceil L-4 \rceil)$ so that $\mathbb{E}\, q = t/2^{\lceil L-4 \rceil}$, and $Var[q] < \mathbb{E}\, q$ by pairwise independence of h. Then

$$\begin{aligned}\mathbb{P}(q=0) &\leq \mathbb{P}(|q - \mathbb{E}\, q| \geq \mathbb{E}\, q) \\ &< \mathbb{P}(|q - \mathbb{E}\, q| \geq \sqrt{\mathbb{E}\, q}\sqrt{Var[q]} \\ &\leq \frac{1}{\mathbb{E}\, q} \text{ (Chebyshev's inequality)},\end{aligned}$$

which is less than $1/8$ by the definition of L. □

Lemma 2.2.8 tells us that it is unlikely that $\ell > \log_2 + 4$ (and thus unlikely that our estimator is bigger than $16t$). Lemma 2.2.9 implies it is unlikely that $\ell < \lceil L-4 \rceil$. Thus by a union bound, we have the following result.

Lemma 2.2.10. $2^\ell \in [\frac{1}{16}t, 16t]$ *with probability at least* $3/4$.

Refining to $(1+\varepsilon)$-approximation. We run the constant-factor approximation algorithm in parallel, so that at any point we can query it to obtain $\hat{t} := 2^\ell$. We henceforth condition on the good event that $\hat{t} \in [\frac{1}{16}t, 16t]$.

Define $k_2 = \lceil 512/\varepsilon^2 + 68/\varepsilon \rceil$. We first check whether $t \leq k_2$ by checking D'; if so, we return $D'.\mathsf{query}()$, which is guaranteed to equal t exactly with probability 1. Otherwise, as mentioned in Subsection 2.2.3 we define r^* such that $t/2^{r^*} = \Theta(1/\varepsilon^2)$; specifically we set $r^* = \lceil \log_2(\varepsilon^2 \hat{t}/32) \rceil$. Then due to rounding from the ceiling function and also by our guarantees on $\hat{t}$, we have $1/\varepsilon^2 < t/2^{r^*} \leq 512/\varepsilon^2$. We also have $Var_h[F_0(r^*)] \leq t/2^{r^*}$, and thus by Chebyshev's inequality $F_0(r^*) = t/2^{r^*} \pm 3\sqrt{512}/\varepsilon = t/2^{r^*} \pm 68/\varepsilon = (1 \pm 68\varepsilon)t/2^{r^*}$ with probability at least $8/9$. In this case, (1) scaling up $F_0(r^*)$ by 2^{r^*} gives a $1 + O(\varepsilon)$ approximation as desired (which can be made $1+\varepsilon$ by running this algorithm with $\varepsilon' = \varepsilon/68$, and (2) we can know $F_0(r^*)$ exactly (in order to scale it up) since $F_0(r^*) \leq k_2$ by our choice of k_2. In summary we have:

Theorem 2.2.11. *There is an algorithm for $(1+\varepsilon)$-approximation of F_0 with probability $1-\delta$ using $O(\varepsilon^{-2} \log^2 n \log(1/\delta))$ bits of memory.*

Proof. The constant factor approximation suceeds with probability at least $3/4$. Conditioned on it succeeding, our refinement to $1+\varepsilon$ approximation succeeds with probabliity at least $8/9$. Thus our overall success probablity is at least $3/4 \cdot 8/9 = 2/3$, which can be amplified to $1-\delta$ in the usual manner by returning the median of $\Theta(\log(1/\delta))$ copies of this basic data structure.

We now analyze the memory to obtain success probability $2/3$; for probability $1-\delta$, this increases by a multiplicative $\log(1/\delta)$ factor. Recall we run two algorithms in parallel, with $k_1 = 1$

and $k_2 = \Theta(1/\varepsilon)^2$; we focus on the latter since its memory consumption is strictly more. Storing the hash function h requires only $O(\log n)$ bits. D' takes an additional $O(k_2 \log n) = O(\varepsilon^{-2} \log n)$ bits of memory. Similarly, each D_i takes $O(k_2 \log n)$ bits of memory and there are $\log_2 n + 1$ such structure, yielding the desired space bound in total. □

Remark 2.2.12. One advantage of this algorithm over the KMV algorithm is that its update and query times are faster; in fact in the so-called "transdichotomous word RAM model" (where the word size is at least $\log n$ bits) [FW93], these times are $O(1)$ (for success probability 2/3), whereas the natural implementation of KMV would use a heap and have $\Theta(\log(1/\varepsilon))$ update time. To see this, we can store the distinct elements in D' and each D_r using a dynamic dictionary data structure such as hashing with chaining, which supports $O(1)$ worst case expected update time. We also know which D_r to update since we can compute $h(i)$ for each i in constant time. For the constant-factor approximation algorithm, since $k_1 = 1$ we need not store the identity of the seen item in D_r; we can simply mark a bit 0 or 1 as to whether any item has ever been seen. As there are $\log n$ such bits, we can store them in a single machine word x. We can find the largest non-empty D_r by computing the index of the least significant set bit of x, which can be done in $O(1)$ time in the word RAM model [Bro93]. For the parallel data structure with larger $k_2 = \Theta(1/\varepsilon^2)$, updates are similarly fast; for queries, we only query the size of a single D_r (namely D_{r^*}), which takes constant time. Thus overall, queries are worst case $O(1)$ time, and updates are $O(1)$ expected time (the bottleneck being the use of dynamic dictionary in each D_r).

2.3 Quantiles

In this problem we see a stream of comparable items $y_1, y_2, \ldots, y_n$, in some arbitrary order. For example, x_i could be the response time to serve request i in some system. We will use $x_1, \ldots, x_n$ throughout this section to denote the y_i in sorted order: $x_1 < x_2 < \cdots < x_n$. Without loss of generality we can assume no two items are equal, since any algorithm can interpret y_i in the stream as (y_i, i) and use lexicographic ordering. We are also given up front some $\varepsilon \in [0, 1)$ at the beginning of the stream. At query time we would then like to support three (really two) types of queries:

- rank(x): if r is such that $x_r < x \leq x_{r+1}$ (i.e. it is the true rank of r), then return $r \pm \varepsilon n$.
- select(r): return any item whose rank in the sorted order is $r \pm \varepsilon n$.
- quantile(ϕ): this is syntactic sugar for select(ϕn).

Going back to the example, a typical use of such a data structure is to answer quantile queries for $\phi = 0.999$ say, when monitoring the performance of a system to ensure that 99.9% of user queries are responded to quickly. We remark that the assumption that the y_i are distinct is not necessary but just simplifies the remaining discussion in this section; without that assumption, one could leave the requirement of rank(x) unchanged (any r satisfying the given inequalities is acceptable), and for select(r), if $x_a = x_{a+1} = \cdots = x_b = x$, then it is acceptable to return x as long as $[a, b] \cap [r - \varepsilon n, r + \varepsilon n] \neq \emptyset$.

The state-of-the-art algorithms for this problem are as follows. By a "comparison-based" sketch, we mean a sketch that works in the model where items come from an infinite universe in which between any two values $x < y$ there exists a z such that $x < z < y$, and the memory of the algorithm can be divided into M_1 and M_2 (see [CV20]). Here M_1 is a subset of items seen, and M_2 is some extra storage that is only allowed to keep track of results of comparisons between stream items and those items in M_1. When processing a new stream update x, we are allowed to compare

x with all elements of M_1, possibly store some of those results in M_2, then also possibly include x in M_1 and remove some current items from M_1. All decisions made are determined only by the results of comparisons.

- **Deterministic, not comparison-based.** If the input values are known to come from a finite universe $\{0, \ldots, U-1\}$, the `q-digest` sketch [SBAS04] solves quantiles sing $O(\varepsilon^{-1} \log U)$ words of memory.
- **Deterministic, comparison-based.** The GK sketch [GK01] uses $O(\varepsilon^{-1} \log(\varepsilon n))$ words of memory, which was recently shown to be best possible for any deterministic comparison-based algorithm [CV20].
- **Randomized, comparison-based.** The KLL sketch [KLL16] when given parameter δ at the beginning of the stream uses $O(\varepsilon^{-1} \log\log(1/\delta))$ words of memory and succeeds with probability $1 - \delta$. That is

  $$\forall \text{ queries } q, \mathbb{P}(\mathsf{KLL} \text{ answers } q \text{ correctly}) \geq 1 - \delta$$

  If one wants the "all-quantiles" guarantee, i.e.

  $$\mathbb{P}(\forall \text{ queries } q,\ \mathsf{KLL} \text{ answers } q \text{ correctly}) \geq 1 - \delta,$$

  then this is accomplished using $O(\varepsilon^{-1} \log\log(1/(\varepsilon\delta)))$ words of memory. For achieving the first guarantee, they also prove that their bound is optimal up to a constant factor (see also [CV20] for a strengthening of the lower bound statement).

Open: Is `q-digest` optimal?
Open: Can one do better than the KLL sketch by not being comparison-based?
Open: The GK analysis is complex; can one match its performance via a simpler approach?

For more details of these and several other sketches, see the survey by Greenwald and Khanna [GK16]. As that survey was written in 2016 though, some of the more recent results on quantiles are not covered, e.g. the KLL sketch [KLL16] and a faster version using the same memory [ILL+19], an optimal lower bound for deterministic comparison-based sketches [CV20], and some recent works on relative-error quantile sketches (see [CKL+20] and the references therein). Some of these more recent developments appear in Chapter 4 of the upcoming book of Cormode and Yi [CY20].

In the following sections, we will cover the following sketches: (1) the `q-digest` sketch, (2) the MRL sketch [MRL98], which is deterministic and comparison-based but achieves the worse memory bound $O(\varepsilon^{-1} \log^2(\varepsilon n))$ when compared to the GK sketch, and (3) KLL.

2.3.1 q-digest

In this setting, where items have values in $\{0, \ldots, U-1\}$, we will allow for items of the stream to have the same value and not do the reduction $y_i \mapsto (y_i, i)$ mentioned in Section 2.3. The main idea of this data structure is to conceptually imagine a perfect binary tree whose leaves correspond to the values $0, 1, \ldots, U-1$. For every node `v` in this tree, we have a counter `c[v]`. Each node represents an interval of values, from its leftmost leaf descendant to its rightmost. The `q-digest` structure will store the names of all `v` with `c[v]` $\neq 0$, together with their `c[]` values. Ideally we would like that `c[v]` is zero for all internal nodes of the tree, and equals the number of occurrences of `v` in the stream for every leaf node `v`. Doing so is equivalent to simply storing a histogram: for each $x \in \{0, \ldots, U-1\}$, we keep track of the number of stream elements equal to x. An exact histogram

would of course allow us to answer all rank/select queries exactly (with $\varepsilon = 0$), but unfortunately could be too memory-intensive if too many distinct values are seen in the stream (which could be as large as $\min\{n, U\}$ values). To remedy this, `q-digest` occasionally merges two sibling nodes into their parent, by zeroing out their `c[]` values after adding them to their common parent. This naturally saves memory by zeroing out nodes, at the expense of worsening accuracy: for counts stored at an internal nodes `v`, we cannot be sure what precise values the items contributing to those counts had within `v`'s subtree.

In what follows we assume U is a power of 2, and we write $L := \log_2 U$. This is without loss of generality, as if not we could simply round U upward and increase it by at most a factor of 2. In the perfect binary tree, we refer to the sibling of a non-root node `v` as `s[v]`, and its parent as `p[v]`. For a non-leaf `v`, we use `left[v]`, `right[v]` to denote its left and right children, respectively.

`q-digest` sketch:

initialization // $\varepsilon \in (0,1)$ is given

1. $n \leftarrow 0$
2. $S \leftarrow \emptyset$ // set of (v, c[v]) pairs with c[v] $\neq 0$; note c[v] $= 0$ for v not tracked by S
3. $K := \lceil 6L/\varepsilon \rceil$

insert(i) // see item i in the stream

1. $n \leftarrow n + 1$
2. **if** leaf node i is in S:
3. `c[`i`]` $\leftarrow$ `c[`i`]` $+ 1$
4. **else**:
5. $S \leftarrow S \cup \{(i, 1)\}$
6. **if** $|S| > K$:
7. compress()

compress() // the root has depth 0, and leaves are at depth L

1. **for** $\ell = L, L-1, \ldots, 1$:
2. **for** each node `v` in S at level ℓ:
3. **if** `c[p[v]] + c[v] + c[s[v]]` $\leq \varepsilon n/L$:
4. `c[p[v]]` $\leftarrow$ `c[p[v]] + c[v] + c[s[v]]`
5. `c[v]` $\leftarrow 0$
6. `c[s[v]]` $\leftarrow 0$
7. adjust S as needed (remove `v` and `s[v]`, and associate `p[v]` with its new `c[p[v]]` value)

rank(x) // let t[v] denote the the sum of all c[u] values of nodes u in v's subtree

1. // below we express $x = \sum_{i=0}^{L-1} x_i 2^i$ in binary
2. `v` $\leftarrow$ `root`
3. `ans` $\leftarrow$ `c[v]`
4. **for** $i = L-1, L-2, \ldots, 0$:
5. **if** $x_i = 1$:
6. `ans` $\leftarrow$ `ans` $+$ `t[left[v]]`
7. `v` $\leftarrow$ `right[v]`
8. **else**:
9. `v` $\leftarrow$ `left[v]`
10. `ans` $\leftarrow$ `ans` $+$ `c[v]`
11. **return** `ans`

select(r)

1. $A \leftarrow$ `c[root]`, `v` $\leftarrow$ `root`, `x` $\leftarrow 0$
2. **for** $i = L-1, L-2, \ldots, 0$:
3. **if** $A +$ `t[left[v]]` $\geq r - 1$:
4. `v` $\leftarrow$ `left[v]`
5. **else**:
6. $A \leftarrow A +$ `c[left[v]]`
7. `v` $\leftarrow$ `right[v]`
8. `x` $\leftarrow$ `x` $+ 2^i$
9. $A \leftarrow A +$ `c[v]`
10. **return** `x`

Figure 2.1: Pseudocode for `q-digest`; bottom-up implementation.

Fig. 2.1 provides a bottom-up implementation of `q-digest` (bottom-up since updates directly affect the leaves, and only percolate upward later when compress is called). S can be implemented as any dynamic dictionary data structure. A top-down recursive implementation of both insert and compress is also possible; see [CY20, Chapter 4] for details.

To answer queries, we let `t[v]` denote the sum of all `c[]` values of nodes in its subtree, including its own `c[v]` value. When answering rank(x), we return the sum of all `c[]` values of leaves to the left of and including x, as well as their ancestors. We can accomplish this by traversing the unique root to x path in the tree and keeping a running sum (see Fig. 2.1). To answer select(r), we would like to find the unique value $x \in \{0, \dots, U-1\}$ such that the sum of all `c[]` values of leaves to the left of and including x, and their ancestors, is at least $r-1$, whereas the same computation for $x-1$ would be strictly less than $r-1$. We find such x via a depth-first search (DFS) from the root. We keep a running sum A that approximates the number of stream elements that are strictly less than the entire range covered by the current node `v`, and we return the leaf that our DFS lands; see Fig. 2.1 for details.

Lemma 2.3.1. *`q-digest` answers both rank and select correctly, i.e. with error $\pm\varepsilon n$ for each.*

Proof. The key observation is that any element x in the stream contributes to the `c[]` value of exactly one node: either x's leaf node, or one of its ancestors. For the calculation of rank(x), note that any $z \le x$ either increments the counter of some ancestor of x, or of some node in the left subtree of an ancestor of x, and thus it is counted. For $z > x$, it either increments the counter of some node in the right subtree of an ancestor of x (in which case it is not counted when we calculate rank(x)), or it increments the counter of an ancestor of x (in which case it is counted). But x has only L ancestors, and each one has a counter value of at most $\varepsilon n/L$ by compress, so the total error introduced by larger z is at most $L \cdot (\varepsilon n/L) = \varepsilon n$. Similar reasoning implies the correctness of select, which we leave as an exercise to the reader. □

The following lemma motivates our choice of K in the pseudocode, as it means we only have to call compress after every at least roughly $K/2$ updates. Thus the amortized runtime to compress is improved compared with running it after every insertion (which would still be correct and improve the space by a factor of two).

Lemma 2.3.2. *After any call to compress, the resulting S will have size at most $3L/\varepsilon + 1$.*

Proof. Consider S after compression. Then for any node `v` tracked by S which is not the root,

$$\texttt{c[p[v]]} + \texttt{c[v]} + \texttt{c[s[v]]} > \varepsilon n/L.$$

Summing over such `v`,

$$3n \ge \sum_{\substack{\texttt{v} \in S \\ \texttt{v not the root}}} \texttt{c[p[v]]} + \texttt{c[v]} + \texttt{c[s[v]]} > (|S| - 1)\varepsilon n/L, \tag{2.9}$$

where the LHS holds since each stream element contributes to exactly one $\texttt{c}[v]$ counter, and each `c[v]` appears at most three times in the above sum (it is the sibling of at most one node in S, and the parent of at most one node in S). Rearranging gives $|S| \le 3L/\varepsilon + 1$, as desired. □

Remark 2.3.3. There are several valid decision choices in an implementation of `q-digest`. For example, it is correct to call compress after every insertion, though that would increase runtime unnecessarily. Alternatively, one could percolate up merges into the parent starting from node i until reaching either the root or a parent whose `c[]` value would be too large to allow the merge.

Doing so helps deamortize the data structure to prevent frequent calls to compress (every $\approx K/2$ updates), but at the same time, note that n is changing: it increases after every insertion. Thus capacities of nodes to hold more in their counters is continuously increasing: previous parents which rejected merges may later allow them due to increased capacity. To ensure that $|S|$ always stays small, one could then call compress every time n doubles, say. The RHS of Eq. (2.9) in between calls to compress would still be at least $(|S|-1)\varepsilon n/(2L)$, implying that the data structure always has size at most $6L/\varepsilon + 1$ even between calls to compress.

2.3.2 MRL

MRL sketch:

initialization // $\varepsilon \in (0,1)$ and the final stream length n are given

1. $k := \varepsilon^{-1}\lceil \log(\varepsilon n)\rceil + 1$, rounded up to the nearest even integer
2. $L := \lceil \log(n/k)\rceil$
3. // the below $\mathtt{A}_j$ arrays are 1-indexed
4. initialize empty "compactor" arrays $\mathtt{A}_0, \ldots, \mathtt{A}_L$ each of size k, initialized to have all null entries

insert(i) // see item i in the stream

1. insert i into $\mathtt{A}_0$ // i.e. replace the first non-null entry in $\mathtt{A}_0$ with i
2. $j \leftarrow 0$
3. **while** $\mathtt{A}_j$ has size k: // i.e. no non-null entries
4. sort($\mathtt{A}_j$)
5. insert $\mathtt{A}_j$[1], $\mathtt{A}_j$[3], ..., $\mathtt{A}_j$[n-1] into $\mathtt{A}_{j+1}$
6. set all entries of $\mathtt{A}_j$ to null
7. $j \leftarrow j + 1$

rank(x)

1. ans $\leftarrow 0$
2. **for** $j = 0, 1, \ldots, L$:
3. **for each** item z in $\mathtt{A}_j$:
4. **if** z $<$ x:
5. ans $\leftarrow$ ans $+ 2^j$
6. **return** ans

select(r)

1. B $\leftarrow$ array containing $(\mathtt{z}, 2^j)$ for every $\mathtt{z} \in \mathtt{A}_j$, over all $j \in \{0, \ldots, L\}$
2. sort B lexicographically
3. $i \leftarrow$ smallest index such that $\sum_{j=1}^{i}$ B[j].second $\geq r$
4. **return** B[i].first

Figure 2.2: Pseudocode for MRL sketch.

The MRL data structure of [MRL98] is also deterministic, but is comparison-based and therefore does not need to assume knowledge of where the universe comes from. It does however need to know the length n of the final stream in advance[3]. The space complexity will be $O(\varepsilon^{-1}\log^2(\varepsilon n))$. Though we have not yet spoken about *mergeability* of sketches, we briefly do so now to state an open problem related to this concept. Roughly speaking, a sketching algorithm is *fully mergeable* if given two sketches S_1 and S_2 created from inputs D_1 and D_2, a sketch S of $D := D_1 \sqcup D_2$ can be created with no degradation in quality of error or failure probability, and satisfying the same efficiency constraints as S_1, S_2. Furthermore, this mergeability condition should hold in an arbitrary merge tree of several sketches (i.e. it should be possible to merge a previous merger of sketches with another merger of sketches, etc.). The MRL sketch unfortunately does not satisfy this property because, as we will soon see, the data structure sets an internal parameter k based on

[3]The more space-efficient GK sketch does not need to know n in advance. Also, for the MRL sketch it suffices to only know an upper bound N on the final stream length, and the memory bounds will then have n replaced with N.

n at the beginning of the sketching procedure. Thus if k_1 is set based on $n_1 := |D_1|$ and k_2 based on $n_2 := |D_2|$, it is not clear how to combine the skeches with different k parameters into a new sketch with $k_3 = n_1 + n_2$ without increasing the error ε guaranteed by the original two sketches.

Open: Does there exist a deterministic, comparison-based quantiles sketch using $o(n)$ memory which is fully mergeable? It is conjectured in [ACH+13] that the answer is no.

The main component in the MRL sketch is a *compactor*, which has a single parameter k that is a positive even integer. A compactor stores anywhere between 0 and k items. When it is full, i.e. has k items, it "compacts": this amounts to sorting its elements into $x_1 < x_2 < \cdots < x_k$ then outputting the odd-indexed elements $x_1, x_3, x_5, \ldots, x_{k-1}$. In the MRL sketch compactors are chained together, so that the output of one compactor is inserted into the next compactor, possibly causing a chain reaction; see Fig. 2.2.

Figure 2.3: Illustrative figure for understanding error introduced during compaction.

A picture to keep in mind for how compaction introduces error into rank queries is drawn in Fig. 2.3. Imagine the query element is x (the red dot), and let the black dots be the elements of the stream currently living in one of the compactors with $k = 8$, arranged in sorted order. In truth, 3 of these elements are less than x. But if they are compacted and moved into the next level, we will think that 4 of these elements were less than x since x is in between two elements that were merged together. Being off by one in this count introduces an error of 2^j for compactions performed at level j. Note also that if x had even rank in the sorted order amongst these elements, no error would be introduced at all. Thus the error introduced for any x during a compaction at level j can be expressed as $\eta_x 2^j$, where $\eta_x \in \{0, 1\}$ depends on the parity of x's rank amongst the items that were compacted. Thus for any x, querying the rank of x has additive error

$$E(x) := \sum_{j=0}^{L-1} \sum_{i=1}^{m_j} \eta_{\mathsf{x},j,i} 2^j, \tag{2.10}$$

where m_j is the number of compactions performed by $\mathtt{A}_j$. Since there are only n items in the stream, there are at most $n/2^j$ items that are ever inserted into $\mathtt{A}_j$. A compaction clears out k elements at a time, and thus $m_j \le n/(k2^j)$. Therefore

$$\begin{aligned} E(\mathsf{x}) &\le \sum_{j=0}^{L-1} \sum_{i=1}^{m_j} 2^j \\ &\le \sum_{j=0}^{L-1} \frac{n}{k2^j} 2^j \\ &= \frac{n}{k} \cdot L \end{aligned} \tag{2.11}$$

To ensure Eq. (2.11) is at most εn, we must set k so that $L/k = \lceil \log(n/k) \rceil / k \le \varepsilon$. That is, $k \ge \varepsilon^{-1} \lceil \log(n/k) \rceil$. This inequality is satisfied by our choice of k.

Theorem 2.3.4. *The MRL sketch uses space* $O(k \log(n/k)) = O(\varepsilon^{-1} \log^2(\varepsilon n))$.

2.3.3 KLL

KLL is a randomized sketch that, for any fixed rank query, succeeds with probability $1-\delta$. Its space usage is $O(\varepsilon^{-1}\log\log(1/\delta))$ words. As mentioned in Section 2.3, it can also provide the "all-quantiles" guarantee using $O(\varepsilon^{-1}\log\log(1/(\varepsilon\delta))$, in which case select can be implemented by performing rank queries on every stored item.

The starting point of KLL is a randomized improvement to the MRL sketch given in [ACH+13]. This randomized MRL sketch is identical to the MRL sketch, except that rather than a compactor always output the odd-indexed elements, it uniformly at random decides to either output the odd- or even-indexed elements in the sorted order. Thus, again looking at Fig. 2.3, we see that the error introduced when estimating $\mathsf{rank}(\mathsf{x})$ by any compaction at level j can be expressed as $\eta\sigma 2^j$, where $\eta \in \{0,1\}$, and $\sigma \in \{-1,1\}$ is uniformly random. Then we can rewrite the error term in our estimate for $\mathsf{rank}(\mathsf{x})$ as

$$E(\mathsf{x}) := \sum_{j=0}^{L-1}\sum_{i=1}^{m_j} \eta_{\mathsf{x},j,i}\sigma_{j,i}2^j, \tag{2.12}$$

This error random variable precisely fits the setup of Khintchine's inequality (see Theorem 1.1.7), where the vector "x" in the statement of the inequality has entries $(\eta_{\mathsf{x},j,i}2^j)_{j,i}$. Then we see that

$$\|x\|_2^2 \le \sum_{j=0}^{L-1} m_j 2^{2j} \le \frac{n}{k}\cdot 2^L \le 2\left(\frac{n}{k}\right)^2. \tag{2.13}$$

Plugging into Khintchine's inequality, we therefore have

$$\mathbb{P}(|E(\mathsf{x})| > \varepsilon n) \le 2e^{-\varepsilon^2 n^2\cdot\frac{k^2}{2n^2}}, \tag{2.14}$$

which is at most δ for $k = \lceil \varepsilon^{-1}\sqrt{2\log(1/\delta)}\rceil$. Note also a further benefit: since k does not depend on n, we do not need to know n in advance. Although the number of compactors does depend on n, we can simply start off with only one compactor and allocate new compactors as they become needed. We thus have the following theorem.

Theorem 2.3.5. *There is a randomized, comparison-based sketch for quantiles with additive error $\pm\varepsilon n$ and failure probability at most δ using $O(\varepsilon^{-1}\sqrt{\log(1/\delta)}\log(\varepsilon n/\sqrt{\log(1/\delta)}))$ words of memory. This algorithm does not need to know the stream length n in advance.*

KLL then makes further optimizations to improve the space complexity. The first optimization is to not make all the compactors have equal size. Rather, the last compactor $\mathtt{A}_L$ will have (the largest) size k. Then compactor $\mathtt{A}_{L-j}$ will have size equal to the maximum of 2 and $\approx (2/3)^j k$. Since we do not know n a priori and thus do not know how many compactors we will need, this is achieved by making our compactors have dynamically changing sizes. For example, initially we will only have one compactor $\mathtt{A}_0$ with capacity k. Then when a compaction finally happens, we will create $\mathtt{A}_1$ with capacity k and shrink the capacity of $\mathtt{A}_0$ to its new, smaller capacity. It can be shown that the number of compactors at any given time is still $\log(n/k)+O(1)$. Furthermore, the number of compactions m_j at level j is still at most $n/(k2^j)$. Similar computations to Eqs. (2.13) and (2.14) then imply $k = O(\varepsilon^{-1}\sqrt{\log(1/\delta)})$ still suffices, but since our compactor sizes are geometrically decreasing (down to a minimum size of 2), our space is improved. In particular, we have the following theorem, which improves Theorem 2.3.5 by making the $\log n$ term additive instead of multiplicative.

Theorem 2.3.6. *There is a randomized, comparison-based sketch for quantiles with additive error $\pm\varepsilon n$ and failure probability at most δ using $O(\varepsilon^{-1}\sqrt{\log(1/\delta)}+\log(\varepsilon n/\sqrt{\log(1/\delta)}))$ words of memory. This algorithm does not need to know the stream length n in advance.*

We only sketch the remaining improvements to the KLL sketch. The first is to observe that since we enforce that all capacities must be at least 2 and otherwise compactor capacities decay geometrically, only the top $O(\log k)$ compactors have size more than 2, and the bottom $T = L - O(\log k)$ compactors all have size exactly 2. One can view the action of these bottom compactors then in unison: from the input stream, elements come in, and the output then of compactor $\mathtt{A}_{T-1}$ is a uniformly random element amongst 2^T input stream elements. Thus, rather than spend $O(T)$ space implementing these T elements, we can implement a sampler using $O(1)$ words of memory. This improves the overall space to $O(k)$ down from $O(k + \log(n/k))$. The next idea is to make all the top s compactors have size k, and only compactors $\mathtt{A}_{L-j}$ for $j > s$ have size $\approx (2/3)^j k$. Then the space is $O(sk)$, but re-examining the error term:

$$\begin{aligned}
|E(\mathsf{x})| &= \left|\sum_{j=0}^{L-1}\sum_{i=1}^{m_j}\eta_{\mathsf{x},j,i}\sigma_{j,i}2^j\right| \\
&\le \left|\sum_{j=0}^{L-s}\sum_{i=1}^{m_j}\eta_{\mathsf{x},j,i}\sigma_{j,i}2^j\right| + \left|\sum_{j=L-s+1}^{L-1} m_j 2^j\right| \\
&\le \left|\underbrace{\sum_{j=0}^{L-s}\sum_{i=1}^{m_j}\eta_{\mathsf{x},j,i}\sigma_{j,i}2^j}_{\alpha}\right| + \underbrace{\frac{ns}{k}}_{\beta}
\end{aligned}$$

Then for the α term, a simpler computation to Eq. (2.13) yields a bound on $\|x\|_2^2$ of $2(n/k)^2 \cdot 2^{-s}$. We can then apply Khintchine as above to obtain failure probability

$$\mathbb{P}(|\alpha| > \varepsilon n/2) \le 2e^{-\varepsilon^2 n^2 \cdot \frac{k^2}{8n^2}\cdot 2^s}.$$

If $s = \lceil \ln\ln(1/\delta))\rceil$, then the above is at most δ as long as $k \ge \sqrt{8}/\varepsilon$. On the other hand, we must also ensure $\beta \le \varepsilon n/2$, which requires $k > s/\varepsilon$. Thus overall, we can afford to set $k = O(\varepsilon^{-1}\log\log(1/\delta))$, which leads to a space bound of $ks = O(\varepsilon^{-1}(\log\log(1/\delta))^2)$.

The final optimization to achieve the improved space bound is to observe that the bottleneck above comes from the top s compactors. We can instead replace them by an optimal deterministic sketch, such as the GK sketch. Thus outputs from $\mathtt{A}_{L-s}$ are fed into the GK sketch and can be viewed as all having the same weight 2^{L-s+1}. When we add in the GK rank query output into our rank query outputs, we thus multiply its output by 2^{L-s+1} first. We thus overall have the following theorem.

Theorem 2.3.7. *There is a randomized, comparison-based sketch for quantiles with additive error $\pm\varepsilon n$ and failure probability at most δ using $O(\varepsilon^{-1}\log\log(1/\delta))$ words of memory. This algorithm does not need to know the stream length n in advance.*

Remark 2.3.8. Although Theorem 2.3.7 gives an optimal memory bound for this problem amongst comparison-based solutions, it is not known to be fully mergeable (see the definition sketch in Subsection 2.3.2). This is because the GK sketch it uses in the last optimization is not known to be fully mergeable. Without this optimization though, KLL is fully mergeable. Thus, the best fully mergeable sketch we have uses space $O(\varepsilon^{-1}(\log\log(1/\delta))^2)$; see [KLL16] for details.

Chapter 3

Lower Bounds

In this chapter we discuss common techniques for proving lower bounds for the size of a sketch to solve some problem, or for the memory used by a streaming algorithm to solve some task. There are two main techniques common in this area: compression-based arguments, and communication complexity.[1] We discuss these both in the below sections, together with example applications.

3.1 Compression-based arguments

3.1.1 Distinct elements

Recall in Section 2.2 we discussed computing the number of distinct elements in a data stream. In that problem we saw a stream $x_1, \dots, x_m$ of (possibly non-distinct) elements in a stream, each in $\{1, \dots, n\}$. We developed randomized, approximate algorithm: that is, if the true number of distinct elements is t, the algorithms we discussed achieving non-trivially low memory used randomness to output a number $\tilde{t}$ satisfying

$$\mathbb{P}(|t - \tilde{t}| > \varepsilon t) < \delta.$$

for some $\varepsilon, \delta \in (0, 10$. One could ask: were *both* randomization and approximation necessary? Could we have a randomized exact algorithm that fails with some small probability δ? Or a deterministic approximate algorithm? It turns out that the answer to both of these questions is no [AMS99]. Before proving so, we first show that the strongest possible guarantee, i.e. an algorithm that is both exact *and* deterministic, requires $\Omega(n)$ bits of memory. The proof we give will be the first example of a *compression argument*. Applied to streaming algorithms, essentially such proofs fit the following template: if there exists an algorithm that uses very little memory (say S bits), then there exists an injection (we often refer to as the *encoding*) $\text{Enc} : A \to \{0,1\}^{g(S)}$. Therefore $g(S) \geq \log_2 A$, which depending on g, may imply some lower bound on S in terms of $\log A$. Such arguments are called compression arguments because they show that a streaming algorithm existed that was too good to be true (uses too little memory), then we would be able to compress a big set, i.e. A, into a smaller one.

Theorem 3.1.1. *Suppose $\mathcal{A}$ is a deterministic streaming algorithm that computes the number of distinct elements exactly. Then $\mathcal{A}$ uses at least n bits of memory.*

Proof. We show that the existence of $\mathcal{A}$ implies the existence of an injection $\text{Enc} : \{0,1\}^n \to \{0,1\}^S$. Therefore $S \geq n$. The injection is defined as follows. Given some $x \in \{0,1\}^n$, we artificially create

[1] These techniques are not actually disjoint; some communication complexity bounds are ultimately proven via compression-based arguments.

a stream that contains in some fixed (say sorted) order all i such that $x_i = 1$. We then run the algorithm $\mathcal{A}$ on it, then define $\text{Enc}(x)$ to be the memory contents of $\mathcal{A}$.

To show that $\text{Enc}(x)$ is an injection, we show that we can invert it (or "Decode") via some inverse function Dec. If M is an S bit string, then $\text{Dec}(M)$ is defined as follows. We initialize $\mathcal{A}$ with M as its memory. We then execute the following:

```
s ← A.query()  // support size of x, i.e. |{i : x_i ≠ 0}|
x ← (0, 0, ..., 0)
for i = 1 ... n:
    A.update(i) // append i to the stream
    r ← A.query() // will either be s or s + 1
    if r = s:  // Encoder must have included i, so it wasn't a new distinct element
        x_i ← 1
    s ← r
return x
```

□

We next show that no deterministic exact algorithm exists using $o(n)$ bits of memory. Before doing so though, we introduce the concept of an error-correcting code.

Definition 3.1.2. An *error-correcting code* is a set $\mathcal{C} \subset [q]^t$. Here q is referred to as the *alphabet size* and ℓ as the *block length*. The *Hamming distance* between two elements of $[q]^t$ is the number of entries where they differ, i.e. $\Delta(x, y) := |\{i : x_i \neq y_i\}|$. The *relative Hamming distance* is $\delta(x, y) = \Delta(x, y)/\ell$. The *distance* of the code is $\min_{c,c' \in \mathcal{C}} \Delta(c, c')$ and the *relative distance* of the code is $\min_{c,c' \in \mathcal{C}} \delta(c, c')$.

There are entire books and courses devoted to error-correcting codes (and more generally, the topic of "coding theory"), so we do not attempt to do the entire field justice here. One of the main reasons codes are useful objects of study is the following observation, which follows by the triangle inequality.

Observation 3.1.3. If $\mathcal{C}$ has distance D, then the open Hamming balls of radii $D/2$ about all $c \in C$ are disjoint.

In other words, if an adversary takes a codeword $c \in C$ then changes fewer than $D/2$ of its entries to produce a new $\tilde{c} \in [q]^\ell$, then another party can later uniquely "decode" $\tilde{c}$ to obtain c. That is, there is a unique $c \in \mathcal{C}$ that could have been modified in this way to obtain $\tilde{c}$. This observation is why such $\mathcal{C}$ are called error-correcting codes: one can imagine that a user wishes to transmit an m-bit message M for $m = \lfloor \log_2 |\mathcal{C}| \rfloor$ to some other user(s). If all users agree ahead of time on some injection $\text{Enc} : \{0,1\}^m \to \mathcal{C}$, then the sender can transmit $\text{Enc}(M)$. The channel of transmission may then have noise that corrupts what was sent (e.g. scratches on a CD, or static on a phone line), but as long as the channel corrupts fewer than $D/2$ entries of the encoded message, the recipients can uniquely decode to recover M precisely.

One other parameter typically of interest when discussing codes is the *rate* of the code. Essentially the rate measures the following: if the original messages and the codewords are over the same alphabet $[q]$, then by what multiplicative factor do messages blow up after encoding. More precisely, rate is the inverse of this quantity: $\mathcal{C}$ has enough codewords to support encoding length-$\log_q |\mathcal{C}|$ messages with alphabet $[q]$. Meanwhile, these messages are being blown up to length t. Thus the rate is defined as $(\log_q |\mathcal{C}|)/\ell$. Rates are numbers between 0 and 1 and are a form of measuring the

efficiency of the code from a redundancy perspective: the closer the rate is to 1, the more efficient it is.

For our purposes in proving sketching lower bounds in this chapter, we do not need to explicitly describe any code, but rather just show that good codes exist. We will do so via the *probabilistic method*.[2]

Theorem 3.1.4. *For any integers $q, n > 1$, there exists a code $\mathcal{C}$ with $|\mathcal{C}| = n$ and block length $\ell = O(q \log n)$ with relative distance $1 - 6/q$.*

Proof. We pick $c_i \in [q]^\ell$ uniformly at random, and do so independently for $i = 1, 2, \ldots, n$. We then wish to show that $\mathcal{C} = \{c_i\}_{i=1}^n$ has the desired property with positive probability. Look at some particular pair $c \neq c' \in \mathcal{C}$. Let Y_k for $k = 1, 2, \ldots, t$ be an indicator random variable for the event $c_k = c'_k$. Then the Y_k are independent Bernoulli with parameter $1/q$. Then $Y = \sum_{k=1}^{\ell} Y_k$ equals $\ell - \Delta(c, c')$ and has $\mathbb{E}\, Y = \ell/q$. By the upper tail of the Chernoff bound in the regime $\lambda > 2e - 1$ (see Eq. (1.4) and Remark 1.1.5),

$$\mathbb{P}(Y > 6\ell/q) < \exp(-\Omega(\ell/q)),$$

which is less than $1/n^2$ by choice of t. Thus by a union bound over all choices of $c \neq c' \in \mathcal{C}$, the probability that there *exists* such a pair with $\ell - \Delta(c, c') > 6\ell/q$ is strictly less than 1. Thus, the desired code exists. □

Corollary 3.1.5. *For any integer $n > 0$ and any integers $\ell, q > 1$ such that $n = q\ell$, there exists a subset $\mathcal{B}_{q,\ell}$ of $\{0,1\}^n$ satisfying the following properties:*

1. *Every $c \in \mathcal{B}_{q,\ell}$ has support size ℓ, i.e. $|\{i : c_i \neq 0\}| = \ell$.*
2. *For $c \neq c' \in \mathcal{B}_{q,\ell}$, $|\{i : c_i = c'_i\}| \leq 6\ell/q$.*
3. *$|\mathcal{B}_{q,\ell}| = \exp(\Omega(\ell/q))$.*

Proof. The corollary follows from Theorem 3.1.4 by writing codewords in unary. That is, for each $c \in \mathcal{C} \subset [q]^\ell$, we convert it into an element of $\{0,1\}^{q\ell}$ by replacing each entry c_i in c with an element of $\{0,1\}^q$ (by putting a 1 in the c_ith position and 0's in the other $q - 1$ positions). □

Now we are ready to present a lower bound against deterministic but approximate algorithms for the distinct elements problem.

Theorem 3.1.6. *Suppose $\mathcal{A}$ is a deterministic streaming algorithm that always outputs a value $\tilde{t}$ when queried such that $t \leq \tilde{t} \leq 1.9t$, where t is the number of distinct elements. Then $\mathcal{A}$ uses at least cn bits of memory for some constant $c > 0$.*

Proof. We show that the existence of $\mathcal{A}$ implies the existence of an injection $\mathrm{Enc} : \mathcal{B}_{q,\ell} \to \{0,1\}^S$ for $q = 100$ and $\ell = n/q$, where $\mathcal{B}_{q,\ell}$ is as in Corollary 3.1.5. Therefore $S \geq \log |\mathcal{B}_{q,\ell}| = \Omega(n)$. The injection is defined as follows. Given some $x \in \mathcal{B}_{q,\ell}$, we artificially create a stream that contains in some fixed (say sorted) order all i such that $x_i = 1$. We then run the algorithm $\mathcal{A}$ on it, then define $\mathrm{Enc}(x)$ to be the memory contents of $\mathcal{A}$.

To show that $\mathrm{Enc}(x)$ is an injection, we show that we can invert it (or "Decode") as in the proof of Theorem 3.1.1. If M is an S bit string, then $\mathrm{Dec}(M)$ is defined as follows:

[2] Briefly, this method works as follows: if one wants to show that an object with some property P exists, then pick an object randomly from some distribution and show that the picked object has property P with nonzero probability.

```
for c ∈ B_{q,t}:
    A.init(M) // initialize A's memory to M
    for i = 1, 2, ..., n:
        if c_i = 1:
            A.update(i)
    if A.query() ≤ 1.9ℓ
        return x
```

Each time through the for loop, $\mathcal{A}$'s memory is reset to the contents immediately after the encoder processed x. Thus the exact number of distinct elements is ℓ, so the reported value to a query will be at most 1.9ℓ if $c = x$. Meanwhile if $c \neq x$, thinking of c, x as sets (containing the elements corresponding to 1 bits), item (2) of Corollary 3.1.5 implies the true number of distinct elements is $|c \cup x| = |c| + |x| - |c \cap x| > 2\ell - 6\ell/q = 1.94\ell$. Thus the output of query will not be at most 1.9ℓ.

□

The 1.1-approximation in Theorem 3.1.6 can be changed to any approximation factor strictly less than 2 by adjusting q. It is even possible to show that α-approximation to the number of distinct elements for any constant $\alpha > 1$ requires $\Omega(n/\alpha)$ bits of memory deterministically [CK16], though we will not present that argument here.

We next show that exact, randomized algorithms must also use $\Omega(n)$ bits of memory.

Theorem 3.1.7. *Suppose $\mathcal{A}$ is a randomized streaming algorithm that outputs the exact number of distinct elements with success probability at least 2/3 for the last query in any fixed sequence of stream updates and queries. Then $\mathcal{A}$ uses at least cn bits of memory for some constant $c > 0$.*

Proof. We first remark that the existence of $\mathcal{A}$ using space S and succeeding with probability 2/3 implies the existence of $\mathcal{A}'$ that outputs the exact right answer with failure probability at most 10^{-6} and uses space $O(S)$: $\mathcal{A}'$ simply runs $O(1)$ copies of $\mathcal{A}$ in parallel with independent randomness and returns the median query result across all parallel runs as its own query response. The claimed failure probability holds via the Chernoff-Hoeffding bound. We thus assume that $\mathcal{A}$ in fact fails with probability at most 10^{-6}.

Consider again the set $\mathcal{B}_{q,\ell}$ from Corollary 3.1.5 for $q = 100$. We will show the existence of such $\mathcal{A}$ implies the existence of an injection $\text{Enc} : \mathcal{B} \to \{0,1\}^S$ for some $\mathcal{B} \subseteq \mathcal{B}_{q,\ell}$ with $|\mathcal{B}| \geq |\mathcal{B}_{q,\ell}|/2$. Thus $S \geq \log|\mathcal{B}| = \log|\mathcal{B}_{q,\ell}| - 1 = \Omega(n)$.

We define Enc' identically as Enc in the proof of Theorem 3.1.1, and Dec' is similar, though instead of returning x we return the closest element (in Hamming) distance from $\mathcal{B}_{q,\ell}$ to x. This is not yet the desired injection and inverse because they are randomized procedures and not actual functions, since $\mathcal{A}$ is a randomized algorithm. For fixed $x \in \mathcal{B}_{q,\ell}$, let x' be the (random) vector recovered by Dec'. Let $Y_{x,i}$ be an indicator random variable for the event that the ith bit was recovered incorrectly, i.e. $x'_i \neq x_i$. Let $Y_x = \sum_{i=1}^n Y_{x,i}$. Then $\mathbb{E}\, Y_x \leq n/10^6$ by linearity of expectation. Let Z_x be an indicator random variable for the event $Y_x > 2n/10^6$. Then $\mathbb{P}(Z_x = 1) < 1/2$ by Markov's inequality. Thus by linearity of expectation,

$$\mathbb{E} \sum_{x \in \mathcal{B}_{q,\ell}} Z_x \geq \frac{1}{2} \cdot \mathcal{B}_{q,\ell}. \tag{3.1}$$

This expectation is over the randomness of $\mathcal{A}$. One can view any randomized algorithm $\mathcal{A}(x)$ as simply a deterministic algorithm $\mathcal{A}(x, r)$, where r is the (random) string that sources $\mathcal{A}$ with

randomness. Eq. (3.1) implies that there *exists* a fixed string r^* such that $\text{Dec}'(\text{Enc}'(x))$ returns x for at least half the $x \in \mathcal{B}_{q,\ell}$ when using r^* as the source of randomness for $\mathcal{A}$. We denote this set of x where the scheme suceeds as $\mathcal{B}$. We then define Enc, Dec as Enc', Dec' but using r^* as the random string. □

3.1.2 Quantiles

Recall in the quantiles problem, we would like to answer ε-approximate rank/select queries after seeing a stream of comparable items. As mentioned, Cormode and Yi showed that any deterministic, comparison-basedf algorithm for this problem requires $\Omega(\varepsilon^{-1}\log(\varepsilon n))$ words of memory, showing that the GK sketch is optimal. We will not show their proof here, but rather a much simpler lower bound.

Theorem 3.1.8. *Suppose $\varepsilon \in (10/n, 1)$. Then any deterministic, comparison-based algorithm for ε-approximate* rank/select *requires at least $\Omega(1/\varepsilon)$ words of memory.*

Proof. Suppose the algorithm stores S elements of the stream. Including the smallest and largest elements of the stream would increase the space to at most $S+2$. If $S+2 \geq n/2$, then are done. Otherwise, call these elements $x_1 < x_2 < \cdots x_{S+2}$. Now any other element in the stream other than those stored is in the interval (x_i, x_{i+1}) for some unique i; there are at most $S-1$ intervals of this form. Thus these at most $S-1$ intervals contain at least $n-S-2$ items of the stream, so at least one interval must contain at least $(n-S-2)/(S-1) \geq 2n/S$ stream items. Every item in this interval has identical comparisons with all the stored elements and thus all have the same response to a rank query. Meanwhile, the smallest and largest elements in this interval have a difference of at least $2n/S-1$ in rank, which can only be at most $2\varepsilon n$ if $S = \Omega(1/\varepsilon)$. Noting that for two numbers (their ranks) differing by more than ℓ there is integer within $\ell/2$ of both finishes the proof. □

We next show a stronger lower bound using a compression-based argument similar to that for distinct elements, based on error-correcting codes. The following lower bounds shows an $\Omega(1/\varepsilon)$ lower bound even for randomized algorithms, and even when the elements come from a bounded universe $\{1, \ldots, U\}$.

Theorem 3.1.9. *In what follows, $C, c > 0$ are some universal constants. Let $\varepsilon \in (0,1)$ be given, and suppose $n, U > 1$ are given integers with $n > C/\varepsilon$ and $U > Cn/\varepsilon$. Suppose $\mathcal{A}$ is a randomized streaming algorithm for $c\varepsilon$-approximate* rank/select *with success probability at least $2/3$ for the last query in any fixed sequence of stream updates and queries, over streams of length n where elements are in the universe $\{1, \ldots, U\}$. Then $\mathcal{A}$ uses at least $\Omega(\varepsilon^{-1}\log(\varepsilon U))$ bits of memory.*

Proof. As in the proof of Theorem 3.1.6, we can assume wlog that $\mathcal{A}$ actually succeeds with probability at least $7/8$ while only increasing its space usage S by at most a constant factor.

Consider all sequences S of $1/\varepsilon$ elements of $[n]$. Given some sequence S, we treat it as a length-$(1/\varepsilon)$ string over the alphabet $[n]$ and encode it with a constant-rate error-correcting code with encoding function Enc that can recover from a $1/4$-fraction of errors (so the relative distance is more than $1/2$); the length of $\text{Enc}(S)$ is $1/\varepsilon' = \Theta(1/\varepsilon)$. The existence of such a code is guaranteed by Theorem 3.1.4 by picking q to be a sufficiently large constant. Now take the ith (0-indexed) symbol, call it x, of $\text{Enc}(S)$ and change it to $i \cdot (n/\varepsilon) + x$ for all i, to get a new string S' over alphabet $[U]$ with $U = n/\varepsilon$. Now run A on S' with error parameter $\varepsilon'/3$ after replicating each letter of S' $\varepsilon' n$ times (so the stream length is n) to get a quantiles data structure. Given the length of S' and our choice of error parameter, over $\mathcal{A}$'s randomness we can recover each symbol of S'

(and thus Enc(S)) with probability 7/8 since S' is a sorted string. Thus, the probability we recover fewer than 3/4 of the letters is at most 1/2. Thus, there exists a fixed random string to source $\mathcal{A}$'s randomness which allows $\mathcal{A}$ to correctly recover a 3/4-fraction of the letters of Enc(S) (and thus decode to recover S) for at least half the possible sequences S. In other words, if the space (in bits) $\mathcal{A}$ uses is s, we have an injection from a collection of size at least $(1/2)n^{1/\varepsilon}$ to bitsrings of length s, implying $s \geq \varepsilon^{-1} \log n - 1 = \Omega(\varepsilon^{-1} \log(\varepsilon U))$. □

3.2 Communication Complexity

Another common method to prove lower bounds is via reductions from problems in communication complexity (though the methods to prove that the communication problems being reduced from are themselves often compression-based). One can imagine representing solving a a streaming problem with ℓ updates as a communication game, in which there are ℓ players $A_1, \ldots, A_\ell$ where player A_i holds update i. One can imagine that if the players had a streaming algorithm, player A_1 could run $\mathcal{A}$ on the first update, send $\mathcal{A}$'s memory contents as a message to player A_2, who can then run $\mathcal{A}$ initialized with that memory on the second update, etc., until player A_ℓ can finally run $\mathcal{A}$ on their update before calling $\mathcal{A}.\mathsf{query}()$. In this way, the memory complexity of solving some streaming problem is captured by the maximum message length required to solve this corresponding communication problem. This translation is not perfect however, since in the communication model we allow players to send arbitrary functions of their input and received message whereas a streaming algorithm must be an algorithm, i.e. it cannot compute uncomputable functions to determine how to update its memory state. For example, nothing prohibits a player in a communication game from deciding what message to send the next player based on the solution to some instance of the halting problem, or some other undecidable problem; its decision on what to send (or player A_ℓ's decision on what to output) is allowed to be the result of an arbitrarily complex function. As we will see in this section, it is often the case that we can prove optimal memory lower bounds for streaming problems by considering only 2 players, call them Alice and Bob. For many problems we can imagine that Alice's input corresponds to say the first half of stream updates, and Bob holds the second half, and their task is to compute some function on the concatenated stream. Even though Alice and Bob have quite a lot of power (they each know half the entire stream!), it turns out that for many streaming problems solving this associated 2-player communication problem is just as hard.

Various models of communication games. Before we continue, we describe a few different types of communication games studied. We describe all these models in the 2-player setting where Alice has some $x \in \mathcal{X}$, Bob has some input $y \in \mathcal{Y}$, and they would like to send messages back and forth to compute $f(x, y)$. Specifically Alice sends a message, to which Bob responds, to which Alice may respond, etc., until one player declares they know the answer and outputs $f(x, y)$. We note that this back-and-forth communication corresponds to multiple passes over the input for a streaming algorithm: for example if Alice talks, then Bob talks, then Alice talks again, this corresponds to the algorithm making two passes over the input stream. For a communication protocol Π and inputs x, y, we define $\Pi(x, y)$ to be the transcript of all messages sent, and $|\Pi(x, y)|$ as the total number of bits of all messages in this transcript.

- **Deterministic complexity.** As the name suggests, Alice and Bob act completely deterministically. The complexity of a protocol is then $\max_{x,y} |\Pi(x, y)|$. We then define $D(f)$ as the minimum over all correct communication protocols Π of $\max_{x,y} |\Pi(x, y)|$.

- **Randomized complexity.** This is again a worst-case notion of complexity (where x, y are worst-case inputs), but where Alice and Bob act randomly and must only fail with probability at most δ. We define $R_\delta^{pub}(f)$ as the minimum over all protocols that are correct with probability at least $2/3$ on any input of $\max_{x,y} |\Pi(x,y)|$. The *pub* here denotes that the source of randomness is an infinitely long public random string in the sky that Alice and Bob both have read access to. $R_\delta^{priv}(f)$ is similarly defined but where Alice and Bob each have their own private source of randomness, which the other player does not know (unless it is communicated, which incurs cost). Clearly $R_\delta^{priv}(f) \geq R_\delta^{pub}(f)$, since in the public coin model we can simulate a private coin protocol by having Alice pretend the even-indexed public bits are her private bits, and Bob pretending the odd-indexed public bits are his.

- **Distributional complexity.** In this model μ is some distribution over $(\mathcal{X}, \mathcal{Y})$, and we assume the inputs are not worst case but rather $(x, y) \sim \mu$. Then $D_\delta^\mu(f)$ denotes the minimum complexity of a protocol that succeeds with probability at least $1 - \delta$ on (x, y) drawn from μ. Without loss of generality we can assume the protocol is deterministic (since for any randomized protocol, its probability of correctness is the average probability of correctness over its source of randomness, and therefore there must be a fixed random string to source it with that performs at least as well as this average).

We also sometimes consider the "one-way model" in which Alice only sends a single message to Bob, and Bob must then output the answer (Bob never speaks to Alice). For one way complexities, we use the notation $D^{\to}(f), R_\delta^{priv,\to}(f), R_\delta^{pub,\to}(f), D_\delta^{\mu,\to}(f)$.

We now state a few theorems known in the communication complexity literature without proof. The interested reader can find proofs in the textbook of Kushilevitz and Nisan [KN97].

Theorem 3.2.1 (Yao's minimax principle)**.** *For any f and any $\delta \in (0,1)$, $R_\delta^{pub}(f) = \sup_\mu D_\delta^\mu(f)$.*

Theorem 3.2.2. *For any f and fixed constant $\delta \in (0, 1/2)$, $R_\delta^{priv} = \Omega(\log(D(f)))$.*

Theorem 3.2.3 (Newman's theorem)**.** *For any $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ and $\epsilon, \delta \in (0,1)$, $R_{\epsilon+\delta}^{priv} \leq R_\delta^{pub}(f) + O(\log n + \log(1/\epsilon))$.*

3.2.1 Equality

The equality function $\textsc{Equality}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is defined by $\textsc{Equality}_n(x, y) = 1$ iff $x = y$. The following known theorem states that amongst deterministic communication protocols, the trivial protocol of Alice sending Bob her entire input is best possible.

Theorem 3.2.4. *For all $n \geq 1$, $D(\textsc{Equality}_n) \geq n$.*

The above theorem implies Theorem 3.1.1 as a corollary.

Corollary 3.2.5. *Suppose $\mathcal{A}$ is a deterministic streaming algorithm that computes the number of distinct elements exactly. Then $\mathcal{A}$ uses at least n bits of memory.*

Proof. Given such an $\mathcal{A}$ using S bits of memory, we define a communication protocol using S bits of communication for $\textsc{Equality}$. Define $X = \{i : x_i = 1\}$ and similarly for Y. Alice runs $\mathcal{A}$ on the stream consisting of all $i \in X$ then sends $\mathcal{A}$'s memory contents to Bob as a message. Bob then runs $\mathcal{A}.\mathsf{query}()$ and immediately outputs 0 if the result is not equal to $|Y|$. If he does not reject, he continues running $\mathcal{A}$ from the memory state sent on all i in Y then runs $A.\mathsf{query}()$ again. He then outputs 1 iff this second query equals $|Y|$. Note the output of this second query is $|X \cup Y|$, which is strictly larger than $|Y|$ iff $Y \neq X$ since X, Y are the same size. □

It is interesting to note that EQUALITY_n is extremal for both Theorems 3.2.2 and 3.2.3.

Theorem 3.2.6. $R^{priv}_{1/3}(\text{EQUALITY}_n) = O(\log n)$ *and* $R^{pub}_{1/3}(\text{EQUALITY}_n) = O(1)$

Proof. For the private coin model, Alice and Bob each treat their inputs as binary representations of integers in the range $\{0, \ldots, 2^{n-1}\}$. Alice then uses private randomness to pick a uniformly random prime p in the interval $[1, 10n^3]$ and sends to Bob the two numbers p and $x \mod p$. Bob then outputs 1 iff $x \equiv y \mod p$. The total communication is $O(\log n)$ bits since p and $x \mod p$ are each at most n^3 and thus requires $3 \log_2 n$ bits each to transmit. As for correctness, this protocol clearly is correct if $x = y$. If they are different, then Bob errs iff p divides $|x-y|$. But $|x-y| < 2^n$ and thus has at most n prime divisors (note if N has prime divisors $p_1, \ldots, p_k$ then $N \geq \prod_{j=1}^{k} p_j \geq 2^k$, implying $k \leq \log_2 N$). Meanwhile, the interval $[1, 10n^3]$ has at least $(1-o(1))10n^3/\ln(10n^3)$ primes by the prime number theorem, and thus the probability that the chosen p is one of these divisors is at most $n/((1-o(1))10n^3/\ln(10n^3)) < 1/3$.

For the public coin model, Alice and Bob use public randomness to agree on two independent uniformly random strings $r, r' \in \{0,1\}^n$. Alice then sends Bob $\langle r, x\rangle \mod 2$ and $\langle r', x\rangle \mod 2$ (two bits). Bob then outputs 1 iff $\langle r, x\rangle = \langle r, y\rangle \mod 2$ and $\langle r', x\rangle = \langle r', y\rangle \mod 2$. Again, this protocol succeeds with probability 1 if $x = y$. If they are not equal, then they succeed iff $\langle x - y, r\rangle = 0 \mod 2$. But since they are not equal, $x - y$ has at least one non-zero entry $(x-y)_i$. Then $\langle x - y, r\angle = (x_i - y_i)r_i + \sum_{j \neq i}(x_j - y_j)r_j$. For any conditioning of this sum, $(x-y)r_i$ either flips the sum or keeps it the same modulo 2 with equal probability. Thus $\mathbb{P}(\langle x - y, r\angle = 0) = 1/2$. Thus Bob fails with probability exactly $(1/2)^2 = 1/4 < 1/3$. □

3.2.2 Disjointness

The disjointness problem is defined by $\text{DISJ}_n(S, T) = 1$ if $S \cap T = \emptyset$ and 0 otherwise, where $S, T \subseteq [n]$. The following theorem is known.

Theorem 3.2.7. *[KS92, Raz92]* $R^{pub}_{1/3}(\text{DISJ}_n) = \Omega(n)$.

Now consider the ℓ_∞-norm approximation problem in the streaming model: we see a sequence of integers $i_1, \ldots, i_\ell \in \{1, \ldots, n\}$ in a stream, which define a histogram $x \in \mathbb{R}^n$ where x_i represents the number of times i appeared in the stream. Upon a query, we should output some $\tilde{z}$ such that $\|x\|_\infty \leq \tilde{z} \leq (1+\varepsilon)\|x\|_\infty$ with probability at least $2/3$. Recall $\|x\|_\infty = \max_i |x_i|$.

Theorem 3.2.8. *Suppose $\mathcal{A}$ is a randomized streaming algorithm for the ℓ_∞-norm approximation problem which outputs $\tilde{z}$ such that $\|x\|_\infty \leq \tilde{z} \leq 1.1\|x\|_\infty$ with probability at least $2/3$. Then $\mathcal{A}$ uses $\Omega(n)$ bits of memory.*

Proof. □

One might ask: what about approximation factors $\alpha \gg 1$? That is, we should output $\tilde{z}$ such that $\|x\|_\infty \leq \tilde{z} \leq \alpha\|x\|_\infty$. For this case, the work [BJKS04] considered a t-player generalization $\text{DISJ}_{n,t}$. There are players $A_1, \ldots, A_t$ where A_i receives as input some $S_i \subseteq [n]$. They are promised that either (1) $A_i \cap A_j = \emptyset$ for all $i \neq j$, or (2) $A_i \cap A_j = \{x\}$ for some $x \in [n]$ for all $i \neq j$. They must decide which case they are in. That work considered the one-way complexity of this problem in which A_1 sends a message to A_2, who then sends a message to A_3, etc., until A_t finally has to output the answer. The complexity measures are for the total amount of bits sent by all players combined.

Theorem 3.2.9. *[BJKS04]* $R^{pub}_{1/3}(\textsc{Disj}_{n,t}) = \Omega(n/t)$. *In particular, some player must send a message of length at least* $\Omega(n/t^2)$ *bits.*

Corollary 3.2.10. *Suppose* $\mathcal{A}$ *is a randomized streaming algorithm that provides a better than* α*-approximation for the* ℓ_∞*-norm approximation problem for some* $\alpha \geq 2$*, with probability at least* $2/3$*. Then* $\mathcal{A}$ *uses* $\Omega(n/\alpha^2)$ *bits of memory.*

Proof. The proof is identical to Theorem 3.2.8 but where we reduce from $\textsc{Disj}_{n,\lceil\alpha\rceil}$. In the case that all sets are pairwise disjoint, we have $\|x\|_\infty \leq 1$ and so the algorithm will report an answer less than α. Meanwhile in case (2), $\|x\|_\infty \geq t = \lceil\alpha\rceil$ and thus will output an answer that is at least $\lceil\alpha\rceil$. Thus the two cases are distinguishable by the streaming algorithm, implying some player sends a message of length at least $\Omega(n/\alpha^2)$ bits. Since each player's message length is exactly the memory usage of $\mathcal{A}$, the claim follows. □

One can also apply the above reasoning to obtain a lower bound for approximating the "pth moment" $\|x\|_p^p$ for any $p \geq 2$, where $\|x\|_p^p := \sum_i |x_i|^p$.

Corollary 3.2.11. *Suppose* $\mathcal{A}$ *is a randomized streaming algorithm that provides a* 2*-approximation for the* p*th moment approximation problem, with probability at least* $2/3$*. Then* $\mathcal{A}$ *uses* $\Omega(n^{1-2/p})$ *bits of memory.*

Proof. We reduce from $\textsc{Disj}_{n,t}$ for $t = \lceil(3n)^{1/p}\rceil$. In the case the sets are disjoint, we have $\|x\|_p^p \leq n$. Meanwhile if some element is in all the pairwise intersections then the pth moment is at least $t^p = 3n$. Thus a 2-approximation can distinguish these cases, solving disjointness. □

It is known that this $n^{1-2/p}$ is optimal up to $\log n$ factors [IW05].

3.2.3 Indexing, GapHamming, and Distinct Elements

We wrap up this chapter by showing a chain of reductions that imply $(1+\varepsilon)$-approximation of the number of distinct elements in a data stream requires $\Omega(1/\varepsilon^2)$ bits of memory (as long as $\varepsilon > 1/\sqrt{n}$). We first introduce the communication problems $\textsc{Index}_n$ and $\textsc{GapHamming}_n$ then show that the former reduces to the latter, which then reduces to distinct elements in the streaming model.

In the $\textsc{Index}_n$ problem Alice gets $x \in \{0,1\}^n$ and Bob gets $j \in [n]$, and Bob must output x_j. We consider one-way model, in which Bob must output the answer based on a single message to Bob. Intuitively this problem is hard, since Alice does not know j and thus seemingly must tell Bob her entire string for him to succeed with good probability. Our goal is to show $R^{pub,\rightarrow}_{1/3}(\textsc{Index}_n) = \Omega(n)$. We will use Theorem 3.2.1, which states that it suffices to lower bound $D^{\mu,\rightarrow}_{1/3}(\textsc{Index}_n)$ for some μ that we may choose freely. We consider μ being the uniform distribution over (x, j). Since these are now random variables, we henceforth refer to them as (X, J).

Before continuing, we state a few standard results from information theory.

Lemma 3.2.12 (Chain rule)**.** $H(Y|X) = H(X,Y) - H(X)$.

Lemma 3.2.13 (Fano's inequality)**.** *Suppose* X *is a random variable finitely supported in* $\mathcal{X}$*. Let* $\hat{X} \stackrel{\text{def}}{=} g(Y)$ *be the predicted value of* X *for* g *a deterministic function also taking values in* $\mathcal{X}$*. Then if* $\mathbb{P}(\hat{X} \neq X) = \delta$*,*

$$H(X|Y) \leq H(X|\hat{X}) \leq H_2(\delta) + \delta \log_2(|\mathcal{X}| - 1),$$

where $H_2(\delta) \stackrel{\text{def}}{=} -\delta \log_2 \delta - (1-\delta)\log_2(1-\delta)$.

Proof. Let E be an indicator r.v. for $X \neq \hat{X}$. Then by the chain rule,

$$H(E, X|\hat{X}) = H(X|\hat{X}) + H(E|X, \hat{X})$$

but also

$$H(E, X|\hat{X}) = H(E|\hat{X}) + H(X|E, \hat{X}) \leq H(E) + H(X|E, \hat{X}) = H_2(\delta) + H(X|E, \hat{X}).$$

Also $H(E|X, \hat{X}) = 0$. Thus $H(X|\hat{X}) \leq H_2(\delta) + H(X|E, \hat{X})$. But also

$$\begin{aligned} H(X|E, \hat{X}) &= (1-\delta) \cdot H(X|E=0, \hat{X}) + \delta \cdot H(X|E=1, \hat{X}) \\ &= \delta \cdot H(X|E=1, \hat{X}) \\ &\leq \delta \cdot \log_2(|\mathcal{X}| - 1). \end{aligned}$$

The last inequality is because if $E = 1$ then $X \neq \hat{X}$, so X can take on one of at most $|\mathcal{X}| - 1$ values. □

Theorem 3.2.14. *For any $n \geq 1$, $D_\delta^{uniform,\rightarrow}(\textsc{Index}_n) \geq (1 - H_2(\delta))n$.*

Proof. Let $\Pi(X)$ be the message Alice sends to Bob on input X. Then given $\Pi(X), J$, Bob outputs $\hat{X}_J$ s.t. $\mathbb{P}(\hat{X}_J \neq X_J) \leq \delta$. Thus by Fano's inequality, $H(X_J|\Pi(X), J) \leq H_2(\delta)$. By definition of conditional entropy and the chain rule,

$$\begin{aligned} H(X_J|\Pi(X), J) &= \sum_{j=1}^{n} \mathbb{P}(J=j) \cdot H(X_J|\Pi(X), J=j) \\ &= \frac{1}{n} \sum_{j=1}^{n} H(X_j|\Pi(X)) \\ &\geq \frac{1}{n} \sum_{j=1}^{n} H(X_j|\Pi(X), X_1, \ldots, X_{j-1}) \\ &= \frac{1}{n} \sum_{j=1}^{n} H(X_1, \ldots, X_j, \Pi(X)) - H(X_1, \ldots, X_{j-1}, \Pi(X)) \\ &= 1 - \frac{1}{n} H(\Pi(X)) \\ &\geq 1 - \frac{1}{n}|\Pi|. \end{aligned}$$

The theorem follows by rearranging the inequality $1 - |\Pi|/n \leq H_2(\delta)$. □

In $\textsc{GapHam}_n$, Alice and Bob receive $x, y \in \{0,1\}^n$, respectively. Recall the Hamming distance $\Delta(x, y) := |\{i : x_i \neq y_i\}|$. In Gap Hamming, Alice and Bob are promised that $\Delta(x, y)$ is *either* at least $n/2 + \sqrt{n}$ *or* at most $n/2 - \sqrt{n}$ and must decide which.

The following theorem is originally due to [Woo04], but the simpler proof we present here via reduction form Index was given later in [TSJ08]. The lower bound being restricted to one-way protocols was removed later in [CR12] (see also later simpler proofs [She12, Vid12]).

Theorem 3.2.15. $R_{1/3}^{pub,\rightarrow}(\textsc{GapHam}_N) = \Omega(N)$.

Proof. We reduce from $\textsc{Index}_n$ to $\textsc{GapHam}_N$ for some odd integer $n = \beta N$ (β a constant to be determined later). Recall in the indexing problem Alice receives $x \in \{0,1\}^n$ and Bob receives $i \in [n]$ and would like to compute x_i after one single message from Alice. First, Alice forms a vector $x' \in \{-1,1\}^n$ where $x'_i = -1$ if $x_i = 1$, and $x'_i = 1$ if $x_i = 0$. The two players will use public randomness to agree upon N independent, uniformly random vectors $r^1, \ldots, r^N \in \{-1,1\}^n$. Alice will then create a vector $a \in \{-1,1\}^n$ where $a_k = sign(\langle x', r^k \rangle)$, and Bob creates $b \in \{-1,1\}^n$ with $b_k = r^k_i$. Note the argument to $sign$ can never be 0 since n is odd. Alice and Bob then let their answer be the answer to $\textsc{GapHam}_N$ for their inputs a, b.

We now must argue correctness. We write $a_k = r^k_i x'_i + (\sum_{j \neq i} r^k_j x'_j) = r^k_i x'_i + \alpha$. Note if $\alpha \neq 0$ then $|\alpha| \geq 2$ since it has an even number of summands, each of which is odd. Thus when $\alpha \neq 0$, $sign(a_k) = sign(\alpha)$, which is uniformly random in $\{-1,1\}$ since α is a symmetric random variable. Meanwhile if $\alpha = 0$, then $r^k_i x'_i$ equals r^k_i iff $x_i = 0$. Thus, letting $[[\mathcal{E}]]$ denote 1 if $\mathcal{E}$ is true and 0 otherwise,

$$\begin{aligned}\mathbb{P}(a_k \neq b_k) &= \mathbb{P}(\alpha \neq 0) \cdot \mathbb{P}(a_k \neq b_k | \alpha \neq 0) + \mathbb{P}(\alpha = 0) \cdot \mathbb{P}(a_k \neq b_k | \alpha = 0) \\ &= \left(1 - \frac{c}{\sqrt{n}}\right) \cdot \frac{1}{2} + \frac{c}{\sqrt{n}}[[x_i = 0]], \qquad \text{(Stirling approximation)}\end{aligned}$$

which is either $1/2 - c/\sqrt{n}$ or $1/2 + c/\sqrt{n}$ depending on whether $x_i = 0$. Thus $\mathbb{E}\,\Delta(a,b)$ is either $N/2 - cN/\sqrt{n}$ or $N/2 + cN/\sqrt{n}$. By setting $n = N/(100c^2)$, $\mathbb{E}\,\Delta(a,b)$ is either $N/2 + 10\sqrt{N}$ or $N/2 - 10\sqrt{N}$. Thus by a Chernoff bound, it is either at least $N/2 + \sqrt{N}$ or at most $N/2 - \sqrt{N}$ with large constant probability, which is then decided by the Gap Hamming protocol. □

Corollary 3.2.16. *For any $\varepsilon \in (1/\sqrt{n}, 1)$, any algorithm $\mathcal{A}$ providing a $(1+\varepsilon)$-approximation of the number of distinct elements in a stream with probability at least $2/3$ must use $\Omega(1/\varepsilon^2)$ bits of memory.*

Proof. We reduce from $\textsc{GapHam}_N$ for $N = 1/(5\varepsilon^2)$. Alice runs $\mathcal{A}$ on the i such that $x_i = 1$ and sends the memory contents of $\mathcal{A}$ to Bob, as well as $|\,support(x)|$, i.e. $|\{i : x_i \neq 0\}|$ (the same as $|x|$ if one treats x as a set). Bob then continues running the algorithm on i such that $y_i = 1$. Note the players can do this since although the distinct elements universe is $[n]$, we have $N \leq n$ by the definition of N and restriction $\varepsilon > 1/\sqrt{n}$. Treating x, y as sets, we have that the number of distinct elements is $|x \cup y| = \Delta(x,y) + |x \cap y| = \Delta(x,y) + |x| + |y| - |x \cup y|$. Rearranging, $\Delta(x,y) = 2|x \cup y| - |x| - |y|$. Bob is told $|x|$, and he knows $|y|$. He also can run $\mathcal{A}.\mathsf{query}()$ to obtain $2(1 \pm \varepsilon)|x \cup y|$, which is equal to $2|x \cup y|$ with an additive error of at most $2\varepsilon N = 1/\varepsilon < \sqrt{N}$. Thus this additive error is enough to decide Hamming distance at most $N/2 - \sqrt{N}$ or at least $N/2 = \sqrt{N}$ for Gap Hamming. □

Chapter 4

Linear Sketching

The focus of this chapter will is *linear sketching*. This is a general technique for sketching a high-dimensional vector $x \in \mathbb{R}^n$ where we store Πx in memory for $\Pi \in \mathbb{R}^{m \times n}$ for some $m \ll n$. If Π has a succinct representation (so that we spend $\ll mn$ space to store it but there is a low-memory algorithm to compute $\Pi_{i,j}$ given i, j), then our sketch reduces the representation of x from n units of memory to $m \ll n$. This technique is not only commonly used in the design of streaming algorithms, but also when designing distributed algorithms as well as we will discuss later in this chapter.

Regarding streaming algorithms, up until this point we have focused on streaming algorithms in the so-called *insertion-only model*. Specifically, consider the scenario of a vector $x \in \mathbb{R}^n$ with n large, and x starting as the 0 vector. Then we see a sequence of updates (i, Δ) each causing the change $x_i \leftarrow x_i + \Delta$. For example consider $\Delta = 1$ for every update, so that x is simply a histogram tracking the number of occurrences of each item from some size-n universe in the stream

In the above setup, there are three popularly studied models:

1. **insertion-only:** Each update has $\Delta = 1$ (the assumption in previous chapters).

2. **strict turnstile:** Some updates Δ can be negative, but we are given the promise that $\forall i$, $x_i \geq 0$ at all times. This makes sense for example in graph streaming, in which case $n = \binom{|V|}{2}$ for vertex set V, and x_e represents the multiplicity of edge e. In a dynamic graph edges may be inserted and deleted, but the multiplicity of an edge would never be negative.

3. **general turnstile:** Anything goes. Updates can be negative, and entries in x can be negative as well. For example, we may wish to process one time period's updates with $\Delta = -1$ and another period with $\Delta = +1$ so that we can later query x being the difference vector of the two time periods.

In this chapter, when discussing streaming we focus on algorithms for the strict and general turnstile models. Note that if we store $y = \Pi x$ in memory, the update (i, Δ) causes the change $x \leftarrow x + \Delta \cdot e_i$, and thus $y \leftarrow y + \Delta \cdot \Pi^i$, where Π^i is the ith column of Π (since generally $Az = \sum_i z_i A^i$ for any matrix-vector product Az). All known algorithms for both strict and general turnstile are actually linear sketches. It is in fact known [LNW14, AHLW16] that *any* algorithm in these two models can be converted into a linear sketch with only a logarithmic factor loss in space complexity, if the algorithm is required to be correct on very long streams[1].

[1] When this assumption does not hold, in certain cases one can do better than linear sketching [KP20].

4.1 Heavy hitters

In this section we discuss two types of (related) problems, referred to as *heavy hitters* and *point query*. For both of these problems, we consider x being updated in the turnstile streaming model. There is also an integer parameter $k > 0$ given at the beginning of the stream.

In the ℓ_1 *point query* problem with parameter k (we sometimes say (k, ℓ_1)-*point query*), we must answer queries of the form query(i), for $i \in [n]$, with a value in the range $x_i \pm \|x\|_1/k$. Here k is a parameter known at the beginning of the stream.

In ℓ_1 *heavy hitters* with parameter k (or (k, ℓ_1)-*heavy hitters*), there is only one query, and we must answer it with a set $L \subset [n]$ such that

1. $|L| = O(k)$
2. $|x_i| > \|x\|_1/k \implies i \in L$

The indices i satisfying the second criterion are called k-*heavy hitters* or k-*frequent* (sometimes we drop the k if clear from context).

Note $\|x\|_1/k = (1/k)\sum_{i=1}^{n} |x_i|$. The number of i which can be strictly larger than an α-fraction of this sum must be strictly less than $1/\alpha$, and thus there can be at most $k-1$ heavy hitters. We are thus requiring that the list L being returned not be more than a constant factor larger than the absolute biggest it need to be to contain all the heavy hitters.

Remark 4.1.1. The tail versions of these problems are also commonly studied. Define $x_{tail(k)}$ to be the vector x but with the top k entries in magnitude all zeroed out. Then in the ℓ_1-tail point query problem, we would like error $\|x_{tail(k)}\|_1/k$ instead of $\|x\|_1/k$. Similarly for heavy hitters, we require that $|x_i| > \|x_{tail(k)}\|_1/k \implies i \in L$. Note under this definition, there can be at most $2k-1$ heavy hitters. Specifically, ignoring the k "head" indices (those not in the tail), there are fewer than k tail indices that can contribute strictly more than a $1/k$ fraction of the total tail mass.

The following lemma shows why point query and heavy hitters are related.

Lemma 4.1.2. *Suppose there is an algorithm $\mathcal{A}$ solving $(3k, \ell_1)$-point query with failure probability at most δ/n and using S words of memory. Then there is an algorithm $\mathcal{A}'$ solving (k, ℓ_1)-heavy hitters with failure probability at most δ and using space at most $S + 3k$.*

Proof. Algorithm $\mathcal{A}'$ uses $\mathcal{A}$ to process the stream. Then to answer a heavy hitters query, it loops through all $i \in [n]$ and point queries each one, remembering the $4k$ indices i with the point query values (breaking ties arbitrarily). Henceforth condition on the event that all n point queries return correct values, which happens with failure probability at most δ by the union bound. Note then that any k-frequent index i will have a point query value strictly larger than $\|x\|_1/k - \|x\|_1/(3k) = (2/3)\|x\|_1/k$. Meanwhile, any index i with $|x_i| < \|x\|_1/(3k)$ will have a point query value strictly less than $(2/3)\|x\|_1/k$, and thus will not appear larger than any actual k-frequent item. Since there are at most $3k$ indices satisfying $|x_i| \geq \|x\|_1/(3k)$, our return list is thus guaranteed to contain all k-frequent indices. □

4.1.1 CountMin sketch

We here describe the CountMin sketch [CM05], which solves ℓ_1 point query in the general turnstile model. We will describe it here in the strict turnstile model. We now describe the operation of the algorithm:

1. We store hash functions $h_1, ...h_L : [n] \to [t]$, each chosen independently from a 2-wise independent family.

2. We store counters $C_{a,b}$ for $a \in [L]$, $b \in [B]$ with $B = 2k$, $L = \lceil \log_2(1/\delta) \rceil$.

3. Upon an update (i, Δ), we add Δ to all counters $C_{a,h_a(i)}$ for $a = 1, \ldots, L$.

4. To answer $query(i)$, we output $\min_{1 \le a \le L} C_{a,h_a(i)}$.

Note that our total memory consumption is $O(L)$ to store the seeds that specify all L hash functions, as well as $O(BL)$ words to store the counters $C_{a,b}$. Thus the total memory consumption is $O(BL)$ words.

Lemma 4.1.3. CountMin.query(i) *returns* $x_i \pm \|x\|_1/k$ *w.p.* $\ge 1 - \delta$.

Proof. Fix i, let $Z_j = 1$ if $h_r(j) = h_r(i)$, $Z_j = 0$ otherwise. Now note that for any $r \in [L]$, $C_{r,h_r(i)} = x_i + \sum_{j \ne i} x_j Z_j := x_i + E$. We have $\mathbb{E}\, E = \sum_{j \ne i} |x_j| \cdot \mathbb{E} Z_j = \sum_{j \ne i} |x_j|/B \le \|x\|_1/(2k)$. Thus by Markov's inequality, $\mathbb{P}(E > \|x\|_1/k) < 1/2$. Thus by independence of the L rows of the CountMin sketch, $\mathbb{P}(\min_r C_{r,h_r(i)} > x_i + \|x\|_1/k) < 1/2^L \le \delta$. □

Thus we easily obtain the following theorem via Lemma 4.1.2.

Theorem 4.1.4. *There is an algorithm solving the ℓ_1 k-heavy hitter problem in the strict turnstile model with failure probability δ, space $O(k \log(n/\delta))$, update time $O(\log(n/\delta))$, and query time $O(n \log(n/\delta))$.*

It is also possible to obtain the tail guarantee for point query from the CountMin sketch with the same memory up to a constant factor.

Lemma 4.1.5. *The CountMin sketch as above but with $B \ge 4k$ guarantees that for any i,* query(i) $= x_i \pm \|x_{tail(k)}\|_1/k$ *w.p.* $\ge 1 - \delta$.

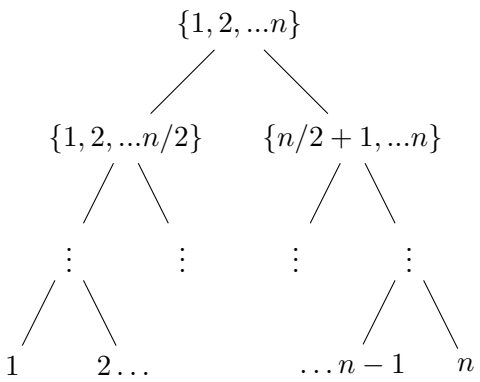
Proof. We let $H \subset [n]$ denote the indices of the top k entries in magnitude in x, and $T := [n] \backslash H$ be the remaining indices (the support of the tail). For fixed $r \in [L]$ we write $C_{r,h_r(i)} - x_i = \sum_{j \in H \backslash \{i\}} |x_j| Z_j + \sum_{j \in T \backslash \{i\}} |x_j| Z_j := E_1 + E_2$ where Z_j is an indicator random variable for the event $h_r(i) = h_r(j)$. Consider the event $\mathcal{F}_r$ that $h_r(i) \notin h_r(H \backslash \{i\})$. This happens with probability at least $1 - |H|/B \ge 3/4$ by a union bound over all $j \in H \backslash \{i\}$ (recall $|H| = k$). When $\mathcal{F}_r$ occurs, $E_1 = 0$. Now consider the event $\mathcal{F}'_r$ that $|E_2| \le \|x_T\|_1/k = \|x_{tail(k)}\|_1/k$, where x_S denotes the projection of x onto $S \subset [n]$ (i.e. zero out all entries that are not in S). By an analysis similar to the proof of Lemma 4.1.3, $\mathbb{P}(\mathcal{F}'_r) \ge 3/4$. Thus by a union bound $\mathbb{P}(\mathcal{F}_r \wedge \mathcal{F}'_r) \ge 1/2$, i.e. for each r $C_{r,h_r(i)}$ provides at most the desired error with probability at least $1/2$. Since each $C_{r,h_r(i)}$ is either equal to x_i or an overestimate, $\min_r C_{r,h_r(i)}$ fails to give the desired error guarantee iff each $C_{r,h_r(i)}$, which happens with probability at most $2^{-L} \le \delta$. □

Remark 4.1.6. Similar guarantees for point query and heavy hitters are obtained in the general turnstile model, but with a slightly larger B and L (by constant factors) and with the estimator returned as the median of the $C_{r,h_r(i)}$ instead of the minimum. For example, if we pick $B = 3k$ then we have $\mathbb{P}(|C_{r,h_r(i)} - x_i| > \|x\|_1/k) < 1/3$. The Chernoff bound thus implies that the median estimator succeeds with probability $1 - \delta$ for $L = C \log(1/\delta)$ for sufficiently large constant $C > 0$.

Speeding up query time

While the above algorithm gives *some* correct heavy hitter algorithm with small space, the query time is quite slow. Here we discuss the *dyadic trick* technique of [CM05] to speed up query.

Consider a perfect binary tree whose leaves are in correpondence with $[n]$.



There are $1 + \lg n$ levels of the tree. We imagine level j of the tree (with the root being level 0) corresponds to a 2^j-dimensional vector $x(j)$ being updated. Each node in the tree is a coordinate of the vector at the corresponding level, and the value at that coordinate is the sum of the two values of the children (with ith leaf simply having value x_i). Then when we see an update (i, Δ), we imagine that this update happens to all coordinates that are ancestors of the ith leaf. Then what we actually store in memory is $1 + \lg n$ CM sketches, one per level. Then upon an update, we feed that update to the appropriate coordinate at the CM sketch at every level.

Each CM sketch is instantiated to solve the $4k$-heavy hitters problem with failure probability $\eta = \delta/(4k \log n)$. Thus the total space is $O(k \log(1/\eta) \log n) = O(k \log((k \log n)/\delta) \log n)$.

To answer a query, the key insight is that in the strict turnstile model *the value at any ancestor of a node is at least as big as the value at that node*, and furthermore the ℓ_1 norm of the implicit vector at each level of the tree is exactly the same. Therefore, if i is a heavy hitter for the vector x at the lowest level of the tree, then *every* ancestor of i is a heavy hitter at its level as well. Since there can only be at most $2k$ indices that are $2k$-heavy hitters, this suggests the following depth first search tree. We move down the tree starting from the root (the root vertex is certainly a heavy hitter for its 1-dimensional vector). At each level j of the tree, we keep track of a list L_j of heavy hitters at that level (L_j should contain all k-heavy hitters of the vector at its level and not contain any item that is not at least a $2k$-heavy hitter). Then, for each of the two children of an index in L_j, we point query that child using the CM sketch at level $j + 1$. If a child has point query output at least $(3/4)\|x\|_1/k$, we include it in L_{j+1} (note that computing $\|x\|_1$ exactly in the strict turnstile model is trivial: maintain a counter). Finally, our final output list L is simply the list corresponding to the bottom-most level of the tree.

Correctness. Note that since each L_j has size at most $2k$, we point query at most $4k$ children on the next level. Thus the total number of queries is at most $4k \log n$ (if it is ever the case that we find ourselves querying more, we can simply output Fail). Thus since we only do at most $Q \leq (4 \lg n)/\alpha$ queries, since each CM sketch has failure probability at most δ/Q, by a union bound the probability that any point query we ever make fails is at most δ. Conditioned on such failure not occurring, if some heavy hitter leaf i is not included in the final L, look at the highest ancestor i' of i on some level j which was not included in L_j. i' cannot be the root, since it is included in L_0. Thus the

parent of i' was included in L_{j-1}, which implies i' was point queried; this is a contradiction to it not being included, since we conditioned on all point queries succeeding.

Complexity. The space used is $O(k \lg(1/\eta) \log n) = O(k \log((k \log n)/\delta) \log n)$ (in words) as mentioned. The query time is the same. The update time is $O(\log n \lg(1/\eta)) = O(\log n \lg((k \log n)/\delta))$.

Though we will not discuss it here, currently the best known algorithm for ℓ_1 heavy hitters in the turnstile model is the ExpanderSketch of [LNNT16]. It achieves $O(k \log(n/\delta))$ words of space, $O(\log(n/\delta))$ update time, and $O(k \log(n/\delta)\, poly(\lg n))$ query time to achieve failure probability δ.

4.1.2 CountSketch

The CountSketch [CCF04] algorithm is similar to the CountMin sketch, except it provides the guarantee that point query returns an estimate equal to $x_i \pm \|x\|_2/\sqrt{k}$ (called "ℓ_2 point query"). It also solves ℓ_2 *heavy hitters*, which requires returning a list of size L containing all i such that $|x_i| > \|x\|_2/\sqrt{k}$. One could also study the tail version of this problem, in which one wishes the point query error to be at most $\|x_{tail(k)}\|/\sqrt{k}$, and where ℓ_2 tail heavy hitters are defined to be the i such that $|x_i| > \|x_{tail(k)}\|_2/\sqrt{k}$.

We first show that obtaining the ℓ_q tail guarantee is always at least as good as obtaining the ℓ_p tail guarantee for $q > p$, up to potentially changing the k by a factor of 2. Thus an ℓ_2 tail guarantee is better than an ℓ_1 tail guarantee. For example, consider the vector $x = (\sqrt{n}, 1, 1, \ldots, 1)$. Then index $i = 1$ is a 1-heavy hitter in ℓ_2 even in the non-tail version of the problem, whereas it is only $O(1/\sqrt{n})$-heavy in ℓ_1.

Lemma 4.1.7. *For any $1 \le p < q$, $\|x_{tail(2k)}\|_q/k^{1/q} \le \|x_{tail(k)}\|_p/k^{1/p}$.*

Proof. For simplicity assume x is infinite-dimensional (pad it with zeroes). Let $S_1 \subset [n]$ of size k be the set of i with the largest $|x_i|$ values (break ties arbitrarily). Let S_2 be the indices of the next k largest entries, etc. Then

$$\begin{aligned}
\frac{1}{k^{1/q}}\|x_{tail(2k)}\|_q &= \frac{1}{k^{1/q}}\left(\sum_{j=3}^{\infty}\|x_{S_j}\|_q^q\right)^{1/q} \\
&\le \frac{1}{k^{1/q}}\left(\sum_{j=3}^{\infty} k \cdot \|x_{S_j}\|_\infty^q\right)^{1/q} \\
&\le \frac{1}{k^{1/q}}\left(\sum_{j=3}^{\infty} k \cdot \frac{\|x_{S_{j-1}}\|_p^q}{k^{q/p}}\right)^{1/q} && (\forall i \in S_j,\ |x_i| \le (\|x_{S_{j-1}}\|_p^p/k)^{1/p}) \\
&= \frac{1}{k^{1/p}}\left(\sum_{j=3}^{\infty}\|x_{S_{j-1}}\|_p^q\right)^{1/q} \\
&= \frac{1}{k^{1/p}}\left(\sum_{j=3}^{\infty}\|x_{S_{j-1}}\|_p^p\right)^{1/p} && (\forall v,\ \|v\|_q \le \|v\|_p) \\
&= \frac{1}{k^{1/p}}\|x_{tail(k)}\|_p
\end{aligned}$$

□

The CountSketch works similarly to the CountMin sketch, but differs slightly in the following way: we in addition select L hash functions $h_1, \dots, h_L : [n] \to \{-1, 1\}$ independently from a 2-wise independent family. Then to process an update (i, Δ), we add $\sigma_r(i)\Delta$ to $C_{r,h_r(i)}$ for $r = 1, \dots, L$ (instead of only adding Δ as in the CountMin sketch). For point query, we pick $B = 9k$ and $L = C \log(1/\delta)$ for some constant $C > 0$. To answer a query, we output the median of $\sigma_r(i)C_{r,h_r(i)}$ over all $r \in [L]$.

Lemma 4.1.8. CountSketch.query(i) *returns* $x_i \pm \|x\|_2/\sqrt{k}$ *w.p.* $\geq 1 - \delta$.

Proof. Fix r and let Z_j be an indicator random variable for the event $h_r(i) = h_r(j)$. Write the error random variable $E_r := \sigma_r(i)C_{r,h_r(i)} - x_i = \sum_{j \neq i} \sigma_r(i)\sigma_r(j)Z_j x_j$. Then the lemma is equivalent to showing that the median over r of $|E_r|$ is at most $\|x\|_2/\sqrt{k}$ with probability at least $1 - \delta$. For this to hold, by the Chernoff bound and choice of L it suffices to show that for any $r \in [L]$, $\mathbb{P}(|E_r| > \|x\|_2/\sqrt{k}) < 1/3$.

$$
\begin{aligned}
\mathbb{E}\,|E_r| &\leq \sqrt{\mathbb{E}\,E_r^2} && \text{(Lemma 1.1.9)} \\
&= \left(\mathbb{E} \left[\sum_{j,j' \neq i} \sigma_r(j)\sigma_r(j')Z_j Z_j' x_j x_{j'} \right] \right)^{1/2} \\
&= \left(\mathbb{E} \left[\sum_{j \neq i} Z_j x_j^2 + \sum_{\substack{j,j' \neq i \\ j \neq j'}} \sigma_r(j)\sigma_r(j')Z_j Z_j' x_j x_{j'} \right] \right)^{1/2} \\
&= \left(\sum_{j \neq i} (\mathbb{E}\,Z_j) x_j^2 + \sum_{\substack{j,j' \neq i \\ j \neq j'}} (\mathbb{E}\,\sigma_r(j)\sigma_r(j'))(\mathbb{E}\,Z_j Z_j') x_j x_{j'} \right)^{1/2} && \text{(linearity of expectation)} \\
&= \left(\sum_{j \neq i} (\mathbb{E}\,Z_j) x_j^2 + \sum_{\substack{j,j' \neq i \\ j \neq j'}} (\mathbb{E}\,\sigma_r(j))(\mathbb{E}\,\sigma_r(j'))(\mathbb{E}\,Z_j Z_j') x_j x_{j'} \right)^{1/2} && \text{(2-wise independence)} \\
&= \left(\sum_{j \neq i} (\mathbb{E}\,Z_j) x_j^2 \right)^{1/2} \\
&\leq \frac{\|x\|_2}{\sqrt{B}} \\
&= \frac{1}{3} \cdot \frac{\|x\|_2}{\sqrt{k}}
\end{aligned}
$$

Thus by Markov's inequality, $\mathbb{P}(|E_r| > \|x\|_2/\sqrt{k}) < 1/3$, as desired. □

The following corollary then holds by combining Lemma 4.1.8 with Lemma 4.1.2.

Corollary 4.1.9. *The CountSketch solves ℓ_2 k-heavy hitters in general turnstile streams with failure probability δ using $O(k \log(n/\delta))$ words of memory.*

The following lemma also holds by combining Lemma 4.1.8 with an analysis almost identical to that of Lemma 4.1.5.

Lemma 4.1.10. *The CountSketch sketch as above but with $B \geq 15k$ guarantees that for any i,* query(i) *returns a value $x_i \pm \|x_{tail(k)}\|_2/\sqrt{k}$ w.p. $\geq 1 - \delta$.*

4.2 Graph sketching

In this section we show the perhaps surprising result that linear sketching can be used to solve combinatorial problems on graphs. Specifically, we consider a dynamic (multi)graph on n vertices in which vertices can be both inserted and deleted. This model can be faithfully represented in the strict turnstile model, where x has dimension $\binom{n}{2}$; x_e specifies the presence (or number of copies) of edge e in the graph. An insertion of edge e then corresponds to the turnstile update $(e, +1)$, whereas an edge deletion corresponds to the update $(e, -1)$. Solving graph problems naively would require remembering x exactly, i.e. $\Omega(n^2)$ memory in the worst case. In this section we describe the "AGM sketch" [AGM12], which shows that dynamic spanning forest can be solved using $O(n \log^c n)$ bits of memory with success probability $1 - 1/poly(n)$. The AGM sketch achieves the exponent with $c = 3$, though in these notes we describe a simpler version achieving $c = 4$. One can also obtain bounds based on the failure probability as a separate parameter δ; see [NY19]. Note that being able to query the dynamic spanning forest allows for solving many other problems, such as finding an s-t path, global connectivity, or s-t connectivity. It is furthermore known that the $\Omega(n \log^3 n)$ bound is tight; any algorithm that reports a spanning forest with probability at least even 1% must use this much memory [NY19]. Yu recently showed a stronger lower bound in the distributed sketching model, in which all vertices and a "referee" share public randomness each vertex knows only its own neighborhood and must send a short message to the referee; he showed that even if the query is not required to output an entire spanning forest but simply a single bit indicating whether the graph is connected, the average message length must be $\Omega(\log^3 n)$ bits [Yu20] (the AGM sketch provides a matching upper bound for spanning forest in this model as well). After the paper [AGM12], linear sketching has been proven useful for a wide variety of dynamic graph problems; see [McG14] for a survey.

The AGM sketch uses certain other data structures as subroutines, which we describe first. Both these data structures operate in the general turnstile model.

4.2.1 k-sparse recovery

Recall $support(x) \subseteq [n]$ denotes the indices i such that $x_i \neq 0$. The parameter k is given at the beginning of the stream, and there is a single query. The answer to a query is as follows: if $|support(x)| \leq k$, the query simply returns x exactly (the indices in the support together with their values); otherwise, the query response can be arbitrary. We show that this problem can be solved deterministically using $O(k \log(nM))$ bits of memory if we are promised that all update amounts are integers and no entry of x is ever larger than M in magnitude. Note with this restriction it suffices to solve the problem over $\mathbb{F}_p$ for any prime $p > M$, as any x_i will equal $x_i \mod p$. The solution we discuss will also require $p \geq n$, and thus we will work over any prime $p > \max\{M, n\}$. The total space will be at most $2k\lceil \log p \rceil$ bits.

Recall in linear sketching we maintain Πx in memory for some $\Pi \in \mathbb{R}^{m \times n}$ (in this case $\Pi \in \mathbb{F}_p^{m \times n}$)). k-sparse recovery is possible iff $\Pi x \neq \Pi y$ for x, y distinct k-sparse vectors, which is equivalent to $\Pi(x - y) \neq 0$. Noting $x - y$ is $2k$-sparse, this requirement is thus equivalent to $\Pi z \neq 0$ for any $2k$-sparse vector z. If S denotes the support of z, then $\Pi z = \Pi_S z$, where Π_S is the $m \times |S|$ submatrix of Π keeping only the columns indexed by S. Thus our requirement for Π is that all of its $m \times 2k$ submatrices have full column rank. We will pick $m = 2k$, so these are square submatrices; thus having full column rank is equivalent to $\det(\Pi_S) \neq 0$ for all $2k \times 2k$ submatrices

of Π. We will specifically pick Π to be the transpose of a Vandermonde matrix. Specifically, pick $x_1 \neq x_2 \neq \cdots \neq x_n \in \mathbb{F}_p$ (this is why we require $p \geq n$, to guarantee at least n distinct elements in $\mathbb{F}_p$) and set $\Pi_{i,j} = x_j^{i-1} \mod p$ for $i, j \in [n]$. For concreteness, we could pick $x_j = j$. The following known fact, which we will not prove here, implies that any $2k \times 2k$ submatrix of Π has nonzero determinant.

Fact 4.2.1. *Let $A \in F^{r \times r}$ be such that $A_{i,j} = x_j^{i-1}$ for $i, j \in [r]$ for some field F. Then*

$$\det(A) = \prod_{1 \leq i < j \leq n} (x_i - x_j)$$

The above fact is usually written for $A^\top$, but note $\det(A) = \det(A^\top)$ for any A. Note Fact 4.2.1 implies $\det(A) \neq 0$ if the x_i are distinct.

Lemma 4.2.2. *Suppose $A \in F^{m \times n}$ is such that every $m \times 2k$ submatrix of A has full column rank, where F is a field. Then there is an algorithm running in $\binom{n}{2k} \cdot poly(n)$ to recover x given $y = Ax$ for any k-sparse x.*

Proof. We loop over all $S \subset [n]$ of size exactly $2k$ (our guess for a set containing the support of x) and compute $x' = \Pi_S^{-1} y$. If x' is k-sparse, then we form x as the n-dimensional vector x with $x_S = x'$ and $x_{[n] \setminus S} = \vec{0}$ and return x. □

Remark 4.2.3. For the particular scheme we propose, it is possible to actually recover x from Πx in $O(k^2 \, polylog(p))$ time via an algorithm called *syndrome decoding*, though we will not cover it here.

4.2.2 SupportFind

In this problem, we would like a randomized data structure which, if $x = 0$, reports `null`. Otherwise, if $x \neq 0$ it should return some $i \in support(x)$ with probability at least $1 - \delta$ (with probability δ it is allowed to behave arbitrarily). There are no promises regarding the vector x; it may or may not be sparse, yet the query algorithm should still succeed.

We describe an algorithm, the JST sketch, for this problem due to [JST11] which uses $O(\log(1/\delta) \log^2 n)$ bits of memory if all entries in x are promised to be integers which are at most $poly(n)$ in magnitude. It is known that this bound is optimal even if the entries of x are promised to always be either 0 or 1 [KNP+17].

The JST sketch uses the geometric sampling technique of Subsection 2.2.3. Specifically, we pick a hash function $h : [n] \to [\log_2 n]$ with $\mathbb{P}(h(i) = j) = 1/2^j$ (other than for $j = \log_2 n$, in which case we have $\mathbb{P}(h(i) = j) = 1/2^{j-1}$ so that the probabilities add to one). For now we assume h is a perfectly random hash function, though we discuss in Remark 4.2.5 how this can be relaxed using bounded independence. For each $j \in [\log_2 n]$, we also instantiate a k-sparse recovery data structure A_j from Subsection 4.2.1 with $k = C \log(1/\delta)$ for a sufficiently large constant $C > 0$ and with $M = poly(n)$.

To process $\mathsf{update}(i, \Delta)$, we simply call $A_{h(i)}.\mathsf{update}(i, \Delta)$. To process a query, we loop from $j = \log_2 n$ down to 1 and for each such j call $A_j.\mathsf{query}()$. For the first (i.e. largest) value of j for which the query is not the zero vector, we return any index in the support of the query response. See Subsection 4.2.2 for pseudocode.

Theorem 4.2.4. *If $x = 0$, `null` is returned with probability 1. Otherwise, the probability some $i \in support(x)$ is returned is at least $1 - \delta$.*

```
for j = log_2 n, ..., 1:
    z ← A_j.query()
    if z ≠ 0:
        return any i such that z_i ≠ 0
return null // x is the zero vector
```

Proof. The case $x = 0$ is clear, as all A_j will return the zero vector when queried. Also clear is the case $|\mathit{support}(x)| \le k := C\log(1/\delta)$, since every A_j will receive a vector with support size at most that of x (and thus will return its received vector exactly), and at least one of the A_j must receive a nonzero vector if $x \neq 0$ since every index of x is hashed to exactly one A_j.

For the remainder of the proof we thus focus on the case that $\mathit{support}(x) \ge k$. Let t denote $|\mathit{support}(x)|$. Let $x^{(j)}$ denote the vector x where we zero out all coordinates i such that $h(i) \neq j$, and define the random variable $T_j := |\mathit{support}(x^{(j)})|$. For some (large) constant c such that $1 < c < C$, let $j^* \in [\log_2 n]$ be such that $c\log(1/\delta) \le t/2^{j^*} < 2c\log(1/\delta)$. Such j^* must exist since $t > C\log(1/\delta)$. We define two events: $\mathcal{E}_1$ is the event $\max_{j \ge j^*} T_j \le k$, and $\mathcal{E}_2$ is the event $T_{j^*} \ge 1$. Note if both events occur, then our query output is guaranteed to be correct. This is because $\mathcal{E}_1$ implies $A_j.\mathsf{query}()$ will correctly return $x^{(j)}$ for all $j \ge j^*$, and $\mathcal{E}_2$ implies that at least one of these $x^{(j)}$ is nonzero (since in particular $x^{(j^*)} \neq 0$). We then show $\mathbb{P}(\neg\mathcal{E}_1 \vee \neg\mathcal{E}_2) \le \delta$, by the union bound.

We bound $\mathbb{P}(\neg\mathcal{E}_1)$ itself by a union bound over $j \ge j^*$. Note $\mathbb{E}\,T_j = t/2^j$. Then $\mathbb{P}(T_j > k) = \mathbb{P}(T_j > (k2^j/t)\cdot\mathbb{E}\,T_j) < (k2^j/t)^{-C'k}$ (see Eq. (1.7)). Summing over $j \ge j^*$ gives a geometric series dominated by its largest term, which is the term for j^*, which is $(k2^{j^*}/t)^{-C'k} \le (k/(c\log(1/\delta))^{-C'k} = (C/c)^{-C'k} < \delta/2$ for C sufficiently larger than c. $\mathbb{P}(\neg\mathcal{E}_2)$ is also bounded by the Chernoff bound. We have $\mathbb{E}\,T_{j^*} \in [c\log(1/\delta), 2c\log(1/\delta)]$. Thus $\mathbb{P}(\neg\mathcal{E}_2) = \mathbb{P}(T_{j^*} = 0)$, which is at most $\delta/2$ by the Chernoff bound. □

Remark 4.2.5. It is a useful fact to know that tail bounds imply moment bounds and vice versa. In one direction, if we have a bound on all moments $\|Z\|_p$ then we have a tail bound via Markov's inequality: $\mathbb{P}(|Z| > \lambda) < \inf_p\{\lambda^{-p}\|Z\|_p^p\}$ (recall $\|Z\|_p := (\mathbb{E}\,|Z|^p)^{1/p}$). The value $p \ge 1$ can be chosen to minimize the right hand side. In the other direction, $\|Z\|_p^p = \int_0^\infty px^{p-1}\,\mathbb{P}(|Z| > x)dx$ via integration by parts. Thus a tail bound on $|Z|$ yields moment bounds. Now, since the Chernoff bound gives strong tail bounds, one can use this correspondence to obtain the implied moment bounds on $|\sum_i X_i - \mu|$ for all $p \ge 1$, and from those moment bounds re-derive the Chernoff bound itself by choosing p optimally based on λ and μ, which is determined by p-wise independence of the X_i if p is an even integer. The punchline is that if one were to carry out this calculation exercise, one would find that whenever the Chernoff bound yields tail probability δ, it sufficed to choose $p = O(\log(1/\delta))$, so that the X_i could be $O(\log(1/\delta))$-wise independent (see also [BR94, SSS95], which take different approaches to showing this). This observation allows one to select the h in the JST sketch from an $O(\log(1/\delta))$-wise independent family, so that it only takes $O(\log(1/\delta)\log n)$ bits to represent.

4.2.3 AGM sketch

Before describing the AGM sketch, we first design a non-streaming algorithm. Imagine that we proceed in $R = \log_2 n$ rounds. We start each round with a partition that is a refinement of the partition of vertices into connected components, and in the first round each vertex is in its own partition. Now, at the beginning of each round we ask each partition (which we henceforth call a *super-vertex*) to identify an edge leaving it and entering another partition. At the end of the

vertex we then merge along the set of all identified edges: if two super-vertices are connected by an identified edge, we merge them into an even bigger super-vertex. The spanning forest we return is the set of all identified edges across the rounds. Note this algorithm is correct since the number of non-maximal components at least halves in each round, and we started with n components and at the end there are is at least 1.

We now describe how to implement the above approach in the streaming model. Imagine each vertex u keeps track of a *signed neighborhood vector* $x_u \in \mathbb{R}^{\binom{n}{2}}$. For any edge e such that either e is not actually in the graph, or e does not contain vertex u, we set $(x_u)_e = 0$. However if $e = (u, v)$ is actually in the graph, we set $(x_u)_e = 1$ if $u < v$ or $(x_u)_e = -1$ if $u > v$. The key reason for signing in this way is that if A is a partition (or super-vertex), then if we define $x_A := \sum_{u \in A} x_u$, then $support(x_A)$ is exactly the set of edges with exactly one endpoint in A and one endpoint in a different super-vertex. Recall that the data structure of Subsection 4.2.2 is a (randomized) linear sketch. We pick $R = \log n$ such linear sketching matrices independnetly $\Pi_1, \ldots, \Pi_R$ each with failure probability $\delta < 1/(n^{1+\beta}R)$ (if our desired failure probability is $1/n^\beta$). We store in memory $\Pi_r x_u$ for all vertices $u \in [n]$ and all $r \in [R]$. The total space is thus $O(nR \log^3 n) = O(n \log^4 n)$ bits. We simulate the offline algorithm by, in each round r, forming sketches for each super-vertex A as $\sum_{u \in A} \Pi_r x_u$, which by linearity is just $\Pi_r(\sum_{u \in A} x_u) = \Pi_r x_A$. We then query for each A to obtain edges leaving each super-vertex. The probability that we always obtain correct edges is the probability that no SupportFind query ever fails, which is at most $\delta nR < 1/n^b$ by the union bound.

We remark that one may be tempted to pick only a single Π and store all Πx_u in memory. Then we can reuse the same Π in each round. Doing so unfortunately is incorrect. This is because our guarantees for randomized algorithms $\mathcal{A}$ are of the following form: for all inputs x, $\mathbb{P}(\mathcal{A}$ gives the correct answer on input $x) \geq 1 - \delta$. Here the probability is over some random string α that provides $mathcal A$ with its source of randomness. But by the order of quantifiers, this means x must be fixed *before* drawing α randomly. In other words, x is not allowed to depend on α. However if we use the same Π in each round, then the fact that we merged certain vertices after round 1 is because of the (random) set of edges our SupportFind data structure happened to identify in round 1. Then we form super-vertices A based on these identified edges, so our next query in random 2, i.e. the fact that we are asking about certain A, is correlated with the randomness of Π. The AGM sketch avoids this by using fresh random Π_r in each round, independent of the linear sketches used in previous rounds. Thus the queries being asked of Π_r are uncorrelated with the randomness used to specify Π_r.

Remark 4.2.6. The version of the AGM sketch described here uses $O(n \log^4 n)$ bits of memory, though it is possible to implement it using $O(n \log^3 n)$ bits of memory. The improvement comes from slightly changing the definition of the SupportFind problem which the AGM sketch relies on. Instead, imagine defining the problem so that there are two types of failure modes, with separate failure probabilities δ_1 and δ_2. In the first failure mode, the data structure should output Fail. In the second failure mode, it may fail without warning (i.e. it simply outputs an index not actually in the support of x). Note some failures of the first form are tolerable for the AGM sketch; it simply means there are some supervertices in some rounds which fail to identify an outgoing edge. This is acceptable though, as long as most supervertices in most rounds do identify an outgoing edge (and we can increase R by a constant factor to compensate). Thus we can set δ_1 to be some small constant, e.g. $1/10$. Failures of the second type though are deadly, since even one such failure causes the entire algorithm to fail. We thus set $\delta_2 = 1/poly(n)$. It is possible to show that such a data structure, with these two failure modes where one is allowed to happen fairly often, can be implemented more memory-efficiently than the structure in Subsection 4.2.2 (though the design of the data structures are very similar), leading to the optimal implementation of the AGM sketch.

For details, see the appendix of [NY19].

4.3 Norm estimation

The problem of ℓ_p norm estimation, i.e. providing a multiplicative approximation to $\|x\|_p := (\sum_i |x_i|^p)^{1/p}$, was first investigated by Alon, Matias, and Szegedy [AMS99]. Specifically, let F_p denote the *pth moment* $\|x\|_p^p = \sum_{i=1}^p |x_i|^p$. As usual, we would like a $(1+\varepsilon)$-approximation to F_p with probability $2/3$. It turns out there is a phase transition in the space complexity of F_p estimation.

- $0 \le p \le 2$: it is known that $poly(\varepsilon^{-1} \log n)$ words of space is achievable [AMS99, Ind06]. For $p = 0$, we treat 0^0 as 0 and any nonzero element to the 0th power as 1, which is the limit of F_p for $p \downarrow 0$. Note that in insertion-only streams, F_0 is simply the distinct elements problem from Section 2.2.
- For $p > 2$ and constant ε, it is known the space complexity is $n^{1-2/p}$ up to logarithmic factors [BJKS04, IW05]. That is, there are both upper and lower bounds. Recall this fact was discussed in and below Corollary 3.2.11.

4.3.1 AMS sketch

We now look at the case $p = 2$, which is solved by the AMS sketch [AMS99]. Let $\sigma \in \{-1, 1\}^{m \times n}$ be drawn from a 4-wise independent family for some $m < n$ to be determined later. We need $O(\log(mn)) = O(\log n)$ bits, i.e. one machine word, to represent σ. Define $\Pi \in \mathbb{R}^{m \times n}$ by $\Pi_{i,j} = \sigma_{i,j}/\sqrt{m}$ so that $y = \Pi x$ satisfies $y_i = \sum_{j=1}^n \sigma_{i,j} x_j / \sqrt{m}$. We estimate $\|x\|_2^2$ by $\|\Pi x\|_2^2 = \|y\|_2^2$.

Analysis.

$$
\begin{aligned}
\mathbb{E}\, y_r^2 &= \frac{1}{m} \mathbb{E} \left(\sum_{j=1}^n \sigma_{r,j} x_j \right)^2 \\
&= \frac{1}{m} \left[\|x\|_2^2 + \mathbb{E} \sum_{j \neq j'} \sigma_{r,j} \sigma_{r,j'} x_j x_{j'} \right] \\
&= \frac{1}{m} \left[\|x\|_2^2 + \sum_{j \neq j'} (\mathbb{E}\, \sigma_{r,j} \sigma_{r,j'}) x_j x_{j'} \right] \\
&= \frac{1}{m} \left[\|x\|_2^2 + \sum_{j \neq j'} (\mathbb{E}\, \sigma_{r,j})(\mathbb{E}\, \sigma_{r,j'}) x_j x_{j'} \right] \qquad \text{(2-wise independence)} \\
&= \frac{1}{m} \|x\|_2^2,
\end{aligned}
$$

and thus $\|y\|_2^2 = \sum_{r=1}^m y_r^2$ has expectation $\|x\|_2^2$. Next we need to estimate the variance in order to apply Chebyshev's inequality. Observe that

$$
\mathbb{E}(\|y\|_2^2 - \mathbb{E}\, \|y\|_2^2)^2 = \frac{1}{m^2} \mathbb{E}(\sum_{r=1}^m \sum_{j \neq j'} \sigma_{r,j} \sigma_{r,j'} x_j x_{j'})^2
$$

$$= \frac{1}{m^2} \sum_{r_1,r_2} \sum_{\substack{j_1 \neq j_2 \\ j_3 \neq j_4}} (\mathbb{E}\, \sigma_{r_1,j_1}\sigma_{r_1,j_2}\sigma_{r_2,j_3}\sigma_{r_2,j_4}) x_{r,j_1}x_{r,j_2}x_{r,j_3}x_{r,j_4} \tag{4.1}$$

$$= \frac{2}{m} \sum_{j_1 \neq j_2} x_{j_1}^2 x_{j_2}^2 \tag{4.2}$$

$$\leq \frac{2}{m}\|x\|_2^4,$$

To see Eq. (4.2), observe that if $r_1 \neq r_2$ then the four $\sigma_{r,j}$ in Eq. (4.1) all have different indices and thus by 4-wise independence the expectation is zero. Thus we need only consider the case $r_1 = r_2 = r$. In this case, we must either have $j_1 = j_3, j_2 = j_4$ or $j_1 = j_4, j_2 = j_3$ else at least one random sign will appear with exponent one and make the expectation zero. Now for a fixed $j < j'$, the term $x_j^2 x_{j'}^2$ appears twice in the summation Eq. (4.2) (once for $j_1 = j, j_2 = j'$ and once for $j_1 = j', j_2 = j$), whereas it appears four times in Eq. (4.1). We thus multiply by the factor two to compensate.

Thus by Chebyshev's inequality, for $m \geq 6/\varepsilon^2$ the probability our estimator is outside of $[(1-\varepsilon)\|x\|_2^2, (1+\varepsilon)\|x\|_2^2]$, i.e. deviates from its expectation by more than $\varepsilon\|x\|_2^2$, is at most

$$\frac{2\|x\|_2^4}{m} \cdot \frac{1}{\varepsilon^2\|x\|_2^4} \leq 1/3.$$

Remark 4.3.1. One can also obtain the same result by letting Π be the CountSketch matrix (as shown in [TZ12]). That is, we pick random $h : [n] \to [m]$ from a 2-wise independent family and $\sigma \in \{-1, 1\}^n$ from a 4-wise independent family and define $\Pi \in \mathbb{R}^{m \times n}$ to be the matrix with exactly one nonzero per column: $\Pi_{h(j),j} = \sigma_j$ for each $j \in [n]$. Then one can show $\mathbb{E}\,\|\Pi z\|_2^2 = \|z\|_2^2$ and $Var[\|\Pi z\|_2^2] = O(1/m)\|z\|_2^4$, so that by Chebyshev's inequality for $m = O(1/\varepsilon^2)$ we have $\|\Pi z\|_2^2$ is a $(1 \pm \varepsilon)$-approximation of $\|z\|_2^2$ with probability at least $2/3$.

4.3.2 Indyk's p-stable sketch

The AMS sketch gives a memory-efficient sketch for $p = 2$, but what about other norms? In [Ind06], Indyk showed that a memory-efficient streaming algorithm exists for estimating ℓ_p norms for any $p \in (0, 2)$ (when $p \leq 1$ is not a norm, but $\|x\|_p := (\sum_i |x_i|^p)^{1/p}$ is still a well-defined function). To accomplish this, he made use of *p-stable distributions*.

Definition 4.3.2. A probability distribution $\mathcal{D}_p$ over $\mathbb{R}$ is said to be *p-stable* if for $Z, Z_1, \ldots, Z_n$ independently drawn from $\mathcal{D}_p$ and for any fixed $x \in \mathbb{R}^n$, the random variable $\sum_{i=1}^n x_i Z_i$ is equal in distribution to $\|x\|_p \cdot Z$.

Some examples are the standard normal distribution $\mathcal{N}(0, 1)$, which is 2-stable. This holds since $x_i g_i$ is a gaussian with variance x_i^2, and the sum of independent gaussians is a gaussian whose variance is the sum of the individual variances. Another less-known example is the Cauchy distribution, which is 1-stable; it has probability density function $\varphi(x) = 1/(\pi(1 + x)^2)$. It is a known theorem that such distributions exist iff $p \in (0, 2]$. Note that p-stable random variables for $p \neq 2$ cannot have bounded variance, since otherwise the sum of independent copies would have to be gaussian as a limiting distribution by the central theorem. In fact, it is known that any p-stable distribution must have tail bounds $\mathbb{P}(|Z| > \lambda) = O(1/(1 + \lambda)^p)$ for all $\lambda > 0$ [Nol10] (see also [Nel11, Theorem 42]); this implies that such distributions cannot exist for $p > 2$ (since otherwise they would have bounded variance, violating the central limit theorem). For $p < 2$ in fact the tail is precisely $\Theta(1/(1 + \lambda)^p)$, so they do not have bounded absolute qth moments for any $q \geq p$.

Though p-stable distributions do not necessarily have closed form density functions, they do have closed form characteristic functions, i.e. $\hat{\varphi}_Z(t) = \mathbb{E}\, e^{itZ}$ (i.e. the Fourier tranform of the pdf). Namely, $\hat{\varphi}_Z(t) = e^{|t|^p}$. Note then for the random variable $x_i Z_i$, $\hat{\varphi}_{x_i Z_i}(t) = \mathbb{E}\, e^{i(tx_i)Z_i} = e^{|x_i|^p|t|^p}$. Since adding two independent random variables convolves their pdfs, it pointwise multiplies their characteristic functions, so that $\sum_i x_i Z_i$ has characteristic function $e^{\|x\|_p^p \cdot |t|^p}$.

We first describe an idealized version of Indyk's p-stable sketch. Pick a matrix $\Pi \in \mathbb{R}^{m \times n}$ where the $\Pi_{i,j} = Z_{i,j}$ are i.i.d. p-stable random variables, scaled so that $\mathbb{P}(Z \in [-1, 1]) = 1/2$ (note that scaling a p-stable distribution by a fixed constant preserves p-stability, i.e. if Z follows a p-stable distribution then αZ follows a p-stable distribution as well). We maintain $y = \Pi x$ in memory, and we estimate $\|x\|_p$ as $median_{1 \le r \le m} |y_i|$.

Analysis. Let $I_S : \mathbb{R} \to \mathbb{R}$ be the indicator of the set S, i.e. $I_S(x) = 1$ if $x \in S$, and it equals zero otherwise. Then since $y_r/\|x\|_p$ is p-stable with scale factor 1, we have $\mathbb{E}\, I_{[-1,1]}(y_r/\|x\|_p) = 1/2$ for each $r \in [m]$. Because the p-stable distribution has a pdf which is both bounded and continuous (this is known; see full discussion in [KNW10a]), we also have the following two facts by linearity of expectation:

- $\mathbb{E}\left[\sum_{r=1}^m I_{[-1-\varepsilon,1+\varepsilon]}(\frac{y_r}{\|x\|_p})\right] = \frac{m}{2} + \Theta(\varepsilon m) \ge \frac{m}{2} + c_1 \varepsilon m.$
- $\mathbb{E}\left[\sum_{r=1}^m I_{[-1+\varepsilon,1-\varepsilon]}(\frac{y_r}{\|x\|_p})\right] = \frac{m}{2} - \Theta(\varepsilon m) \le \frac{m}{2} + c_2 \varepsilon m.$

Note that if $\sum_r I_{[-1-\varepsilon,1+\varepsilon]}(y_r/\|x\|_p) > m/2$ then strictly more than half the y_r satisfy $|y_r| \le (1+\varepsilon)\|x\|_p$, and similarly $\sum_r I_{[-1-\varepsilon,1+\varepsilon]}(y_r/\|x\|_p)$ implies that strictly less than half the y_r satisfy $|y_r| \le (1-\varepsilon)\|x\|_p$. Thus if both these events happen simultaneously, we indeed have that the median estimate is $(1 \pm \varepsilon)\|x\|_p$. To show that this happens with good probability, we use Chebyshev's inequality. Specifically, for any r we have $Var[I_S(y_r/\|x\|_p)] \le 1$ since the range of I_S is $[0, 1]$. Thus the variances of the above sums are each at most m by independence of the y_r. Chebyshev's inequality and a union bound thus implies that the probability that either sum deviates from its expectation by more than $3\sqrt{m}$ is at most $2/9$. We can then ensure $3\sqrt{m} < \max\{c_1 \varepsilon m, c_2 \varepsilon m\}$ by picking $m \ge 9(\min\{c_1, c_2\})^{-2}/\varepsilon^2$.

Of course, the two main issue with this idealized algorithm, as in Subsection 2.2.1, are precision and independence. The $\Pi_{i,j}$ are real numbers, but our computer can only perform finite-precision arithmetic; this can be dealt with by simply rounding the $\Pi_{i,j}$ to appropriate precision before doing computation. We will not delve into those details here as they are fairly routine, and we instead refer the reader to [KNW10a]. Regarding the independence, we used independence in two places: (1) to argue that the variance of the sum of indicators equals the sum of the variances, and (2) to know that $y_r/\|x\|_p$ is p-stable, so that we can estimate $\mathbb{E}\, I_S(y_r/\|x\|_p)$. For (1), note this holds even if the random variables summed are only 2-wise independent. Thus we can simply have that the random seeds $s_1, \ldots, s_m$ used to generate the rows of Π are not fully independent, but rather is a simply from a 2-wise independent sample space. For (2), it is known (though unfortunately quite complicated to prove so we will not do so here), that k-wise independence suffices for $k = O(1/\varepsilon^p)$.

Theorem 4.3.3 ([KNW10a])**.** *Let $Z_1, \ldots, Z_n$ be i.i.d. from $\mathcal{D}_p$, and let $Y_1, \ldots, Y_n$ be k-wise independent from $\mathcal{D}_p$. Then for any fixed $x \in \mathbb{R}^n$,*

$$\sup_{t \in R} \left| \mathbb{E}\, I_{(-\infty,t]}(\sum_i x_i Z_i) - \mathbb{E}\, I_{(-\infty,t]}(\sum_i x_i Y_i) \right| < O(1/k^{1/p}).$$

Theorem 4.3.3 says that the CDFs of the distributions of $\sum_i x_i Z_i$ and $\sum_i y_i Z_i$ are close everywhere. Note $I_{(a,b]}$ is simply $I_{(\infty,b]} - I_{(\infty,a]}$, so Theorem 4.3.3 implies that the amount of probability mass in any interval is the same in the two distributions up to an additive $O(1/k^{1/p})$. We have only talked about k-wise independence for uniform distributions in this course, but note that any distribution can be generated from a uniform random variable in $[0,1]$ via the inverse CDF (which in our case we will discretize to finite precision, i.e. integer multiples of γ for some small γ). Specifically for p-stable random variables, it is known how to do this generation efficiently [CMS76].

4.3.3 Branching programs and pseudorandom generators

Although Indyk's p-stable sketch could be derandomized to obtain an optimal algorithm via k-wise independence, it is unfortunately quite technical to show that this is true, and it was not even known until about a decade after Indyk published his algorithm. It turns out though that there is a simple generic way to derandomize many streaming algorithms, and that is by modeling their execution as that of a *Read-Once Branching Program* (ROBP) then using a generic Pseudorandom Generator (PRG) against such objects, such as Nisan's PRG [Nis92].

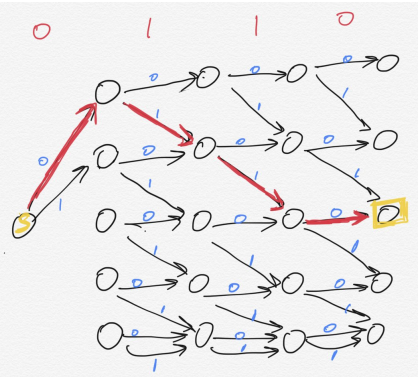


Figure 4.1: Example of an ROBP that calculates the sum of 4 input bits.

Definition 4.3.4. A *read-once branching program* is a directed, layered graph with layers $0, 1, 2, \ldots, L$. Here L is called the *length*. The 0th layer has just one vertex, called the *start vertex*, and every other layer has the same number of vertices W, called the *width*. Each vertex, except in layer L, has out-degree exactly d for some value of d, and each edge goes into a vertex in the next layer. The out-edges from a given vertex are also labeled $0, 1, \ldots, d-1$. Vertices in layer L have no out-edges. Such an ROBP can be viewed as computing a function $f : \{0, 1, \ldots, d-1\}^L \to [W]$, where on input $(x_1, x_2, \ldots, x_L) \in \{0, 1, \ldots, d-1\}^L$, the evaluation of the function is the index of the vertex one arrives at in the final layer by starting at the start vertex and in the ith step following the edge out of the current vertex with label x_i.

For example, Fig. 4.1 describes an ROBP which calculates the sum $x_1 + x_2 + x_3 + x_4$ of 4 input bits. L is 4 and the width W is 5. Essentially each layer is simply keeping track of the running sum of the input bits seen so far (which is a number in $\{0, 1, 2 \ldots, 4\}$ and hence at most 5 states are needed per layer). The left layer with a yellow S is the start vertex, and the red highlighted path shows the evaluation of f on input $(0, 1, 1, 0)$. The vertex in layer L boxed in yellow denotes the output state, which here we imagine is indexed as 2 (counting vertices from top to bottom, starting at 0). Note that one can imagine an ROBP of width W as representing a computation using only $\lceil \log_2 W \rceil$ bits of memory, since that is the memory required to remember the current state.

In the case of for example Indyk's p-stable sketch, imagine that we generate each $\Pi_{i,j}$ from a uniform random number in $[0,1]$ with only b bits of precision, i.e. an integer multiple of $1/2^b$, and we round the resulting p-stable random variable so that it only requires b' bits of precision to store. Then since the output of the algorithm is invariant under permutation of the stream (since it applies a linear sketch), the algorithm's estimate of $\|x\|_p$ would be identical if we saw all updates to $i = 1$ first, then $i = 2$, etc. Consider the following computation which sees the entries of Π in row-major order, i.e. we see the entries in the order $\Pi_{1,1}, \Pi_{1,2}, \ldots, \Pi_{1,n}, \Pi_{2,1}, \ldots, \Pi_{2,n}, \ldots, \Pi_{m,1}, \ldots, \Pi_{m,n}$ (more specifically, we see the b-bit integers that specify each of these entries).

Indyk ROBP:

$c_{low} \leftarrow 0$
$c_{high} \leftarrow 0$
for $r = 1, \ldots, m$:
  $C \leftarrow 0$
  for $j = 1, \ldots, n$:
    $C \leftarrow C + x_j \cdot \Pi_{r,j}$
  if $C \leq (1-\varepsilon)\|x\|_p$:
    $c_{low} \leftarrow c_{low} + 1$
  if $C \leq (1+\varepsilon)\|x\|_p$:
    $c_{high} \leftarrow c_{high} + 1$

One can model the evolution of the memory state of Subsection 4.3.3 as a ROBP with $d = 2^b$, $L = mn$, and the width sufficiently large to remember C, c_{low}, c_{high}. If each x_j is an integer bounded by T in magnitude, then we can store C using $b' + \log(nT)$ bits of precision. Also c_{low}, c_{high} are always integers in the set $\{0, 1, \ldots, m\}$. Thus the total space we need to identify our current state (i.e. a vertex in a layer) is $\lceil \log_2 W \rceil = O(b' + \log(nT) + \log m)$ bits. The final state reveals c_{low}, c_{high}, which is the number of r such that $|y_r| \leq (1-\varepsilon)\|x\|_p$ and $|y_r| \leq (1+\varepsilon)\|x\|_p$, respectively. As seen in Subsection 4.3.2, with probability $7/9$ these are simultaneously strictly less than $m/2$ and strictly more than $m/2$.

The goal of a PRG is to preserve the distribution over final states with good probability, so that feeding the ROBP truly random bits to generate the $\Pi_{i,j}$ versus *pseudorandom* bits results in a distribution over vertices in the final layer that is almost distinguishable.

Definition 4.3.5. The *total variation distance* between two probability distributions $\mathcal{D}, \mathcal{D}'$ is $\|\mathcal{D} - \mathcal{D}'\|_{TV} := \sup_S |\mathbb{P}_{X \sim \mathcal{D}}(X \in S) - \mathbb{P}_{X' \sim \mathcal{D}'}(X' \in S)|$.

From our perspective, we can imagine that $\mathcal{D}$ generates a sequence X of nmb independent, uniform bits that specifies the $\Pi_{i,j}$, and $\mathcal{D}'$ generates a pseudorandom sequence X' of nmb bits. We can define S to be the set of inputs x (i.e. the nmb bits specifying all of Π) such that for f being the ROBP representing Subsection 4.3.3, $f(x)$ leads to a memory state that implies Indyk's output is correct, i.e. $(1 \pm \varepsilon)\|x\|_p$. Then if $\mathcal{D}, \mathcal{D}'$ have TV-distance at most ϵ, then using the pseudorandom bits generated by $\mathcal{D}'$ must still lead to a correctness probability of at least $2/3 - \epsilon$. Nisan's PRG gives us just that.

Theorem 4.3.6. *Let $\mathcal{U}_{A,t}$ denote the uniform distribution on A^t. For any $S, L \geq 1$ there exists a function $G_{nisan} : \{0,1\}^s \to (\{0,1\}^S)^L$ for $s = O(S \log L)$ such that for any f calculated by an ROBP with width 2^S, degree $d = 2^S$, and length L,*

$$\|f(\mathcal{U}_{\{0,1\}^S, L}) - f(G_{nisan}(\mathcal{U}_{\{0,1\}, s}))\|_{TV} \leq \exp(-\Omega(S)).$$

Furthermore, for any $x \in \{0,1\}^s$ and any $j \in [L]$, $(G_{nisan}(x))_j$ can be computed in space $O(S \log L)$.

We can thus derandomize Indyk's algorithm by letting the $\Pi_{i,j}$ be specified by G_{nisan} applied to a short s-bit random string for $s = O(S \log L) = O((b' + \log(nT) + \log m) \cdot \log(mn)) = O((b' + \log(nT)) \log n)$ bits since $m \leq n$. It can be shown that one can take $b' = O(\log(nT/\varepsilon))$ [KNW10a], leading to $O(\log(nT/\varepsilon) \cdot \log n)$ bits overall to specify a sufficiently good pseudorandom matrix Π. Overall this leads to an algorithm for ℓ_p norm estimation using $O(\varepsilon^{-2} \log(nT/\varepsilon) + \log(nT/\varepsilon) \log n)$ bits of memory (the first summand is for maintaining the actual sketch, and the second is for remembering the seed to Nisan's PRG). Using k-wise independence instead of Nisan's PRG eliminates the second term, leading to a space-optimal algorithm (matching a lower bound of [KNW10a]), at the cost of requiring a much more complicated analysis.

Chapter 5

Johnson-Lindenstrauss Transforms

The following "Johnson-Lindenstrauss lemma" (JL lemma) has been highly impactful in the design of algorithms for high-dimensional data.

Theorem 5.0.1 (JL lemma [JL84])**.** *For any $\varepsilon \in (0,1)$ and any $X \subset \mathbb{R}^d$ for $|X| = n$ finite, there exists an embedding $f : X \to \mathbb{R}^m$ for $m = O(\varepsilon^{-2} \log n)$ such that*

$$\forall x, y \in X,\ (1-\varepsilon)\|x-y\|_2^2 \leq \|f(x)-f(y)\|_2^2 \leq (1+\varepsilon)\|x-y\|_2^2. \tag{5.1}$$

Note one can take the square root of all terms in Eq. (5.1) to say the ℓ_2 norm itself is preserved (and not the square), which affects ε by roughly a factor of 2; we write the squared version as it makes some later arguments less clumsy.

A common use of the JL lemma is in the design of approximate algorithms for high-dimensional computational geometry problems. The idea is that given some input X of a set of high-dimensional vectors, rather than solve the computational problem on X we can instead solve it on $f(X)$ for an embedding f as in the JL lemma. Then presumably, since $f(X)$ lives in lower dimension, the algorithm is faster.

Consider as one example the k-means clustering example. In this problem we have input $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ and a given integer parameter $k > 1$, and we would like to return $y_1, \ldots, y_k \in \mathbb{R}^d$ minimizing

$$\sum_{i=1}^{n} \min_{1 \leq j \leq k} \|x_i - y_j\|_2^2.$$

Any choice of the y_j's induce a Voronoi partition $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_k)$ on X: $\mathcal{P}_r$ is the set of all i such that $y_r = argmin_j \|x_i - y_j\|_2^2$. One can then rewrite the k-means objective as

$$\min_{k-\text{partitions } \mathcal{P}} \min_{y_1,\ldots,y_k} \sum_{i=1}^{n} \|x_i - y_{\pi_{\mathcal{P}}(i)}\|_2^2 = \min_{k-\text{partitions } \mathcal{P}} \min_{y_1,\ldots,y_k} \sum_{j=1}^{k} \sum_{i \in \mathcal{P}_j} \|x_i - y_j\|_2^2. \tag{5.2}$$

where $\pi_{\mathcal{P}}(i)$ denotes the partition j such that $i \in \mathcal{P}_j$. One can show then via calculus that for any fixed $\mathcal{P}$, the optimal choice of the centers y_j are the centroids $\mu_j := (1/|\mathcal{P}_j|) \sum_{i \in \mathcal{P}_j} x_i$. Thus Eq. (5.2) can be rewritten, and via some simple subsequent manipulations one obtains that the objective is

$$\min_{k-\text{patitions } \mathcal{P}} \sum_{j=1}^{k} \sum_{i \in \mathcal{P}_j} \|x_i - \mu_j\|_2^2 = \min_{k-\text{patitions } \mathcal{P}} \sum_{j=1}^{k} \frac{1}{|\mathcal{P}_j|} \sum_{i < i' \in \mathcal{P}_j} \|x_i - x_{i'}\|_2^2. \tag{5.3}$$

From Eq. (5.3) it is apparent that if an embedding f preserves all squares distances within X up to $1\pm\varepsilon$, then the cost of *any* clustering (i.e. any partition $\mathcal{P}$) is similarly preserved. Thus we obtain an approximately optimal clustering by mapping X down to low dimension via the JL lemma, solving k-means there, then using the discovered clustering partition even for the original high-dimensional X. Thus, for $(1+\varepsilon)$-approximation it suffices to solve k-means clustering in dimension $O(\varepsilon^{-2}\log n)$.

All known proofs of the JL lemma first prove the following "Distributional Johnson-Lindenstrauss lemma" (DJL lemma).

Lemma 5.0.2 (DJL lemma)**.** *For any $\varepsilon,\delta\in(0,1/2)$ and integer $d>1$, there exists a distribution $\mathcal{D}_{\varepsilon,\delta}$ over matrices $\Pi\in\mathbb{R}^{m\times d}$ for $m=O(\varepsilon^{-2}\log(1/\delta))$ such that for any fixed $z\in\mathbb{R}^d$ with $\|z\|_2=1$,*

$$\mathbb{P}_{\Pi\sim\mathcal{D}_{\varepsilon,\delta}}(|\|\Pi z\|_2^2-1|>\varepsilon)<\delta.$$

The JL lemma is then a corollary of the DJL lemma for the following reason: we set $\delta<1/n^2$ and pick a random Π as in the DJL lemma. Then for any $x\neq y\in X$, we set $z_{x,y}:=(x-y)/\|x-y\|_2$. The DJL lemma implies $\mathbb{P}(|\|\Pi z_{x,y}\|_2^2-1|>\varepsilon)<\delta$, which is equivalent to $\mathbb{P}(|\|\Pi(x-y)\|_2^2-\|x-y\|_2^2|>\varepsilon\|x-y\|_2^2)<\delta$. By a union bound, the probability there exists some $x\neq y\in X$ such that $\|\Pi(x-y)\|_2^2\notin[(1-\varepsilon)\|x-y\|_2^2,(1+\varepsilon)\|x-y\|_2^2]$ is at most $\binom{n}{2}\delta<1$. Thus there exists a Π^* such that $\|\Pi^*(x-y)\|_2^2\in[(1-\varepsilon)\|x-y\|_2^2,(1+\varepsilon)\|x-y\|_2^2]$ for all $x,y\in X$. We define $f(x)=\Pi^*x$.

The main task is thus to prove the DJL lemma.

5.1 Proof of the Distributional Johnson-Lindenstrauss lemma

We prove the DJL lemma using the Hanson-Wright inequality, specifically the tail version (see Corollary 1.1.16). We will let $\mathcal{D}_{\varepsilon,\delta}$ be the distribution over matrices Π with i.i.d. entries $\Pi_{r,i}=\sigma_{r,i}/\sqrt{m}$, where the $\sigma_{r,i}$ are independent Rademachers (i.e. uniform in $\{-1,1\}$). Define the matrix

$$B_z=\frac{1}{\sqrt{m}}\cdot\begin{bmatrix}-z^\top- & 0 & \cdots & 0\\ 0 & -z^\top- & \cdots & 0\\ \vdots & \vdots & & \vdots\\ 0 & 0 & \cdots & -z^\top-\end{bmatrix}. \tag{5.4}$$

Define the vector $\sigma\in\{-1,1\}^{md}$ by $\sigma=(\sigma_{1,1},\sigma_{1,2},\ldots,\sigma_{1,d},\ldots,\sigma_{m,1},\ldots,\sigma_{m,d})$. Then $\Pi z=B_z\sigma$. Thus $\|Piz\|_2^2-1=\|B_z\sigma\|_2^2-1=\sigma^\top B_z^\top B_z\sigma-\mathbb{E}\,\sigma^\top B_z^\top B_z\sigma$. Defining $A_z:=B_z^\top B_z$ and applying Corollary 1.1.16,

$$\mathbb{P}(|\sigma^\top A_z\sigma-\mathbb{E}\,\sigma^\top A_z\sigma|>\varepsilon)\lesssim e^{-C\varepsilon^2/\|A_z\|_F^2}+e^{-C\varepsilon/\|A\|}. \tag{5.5}$$

We thus need to bound $\|A_z\|_F$ and $\|A_z\|$.

We see A_z is an $md\times md$ block-diagonal matrix with m blocks, where each block equals $(1/m)zz^\top$. The squared Frobenius norm is $(1/m^2)\sum_{r=1}^m\|zz^\top\|_F^2=(1/m^2)\sum_{r=1}^m\sum_{i,j}z_i^2z_j^2=(1/m)\|z\|_2^4=1/m$. We also have $\|A_z\|$ is its largest singular value. Since A_z is real and symmetric, the spectral theorem implies all its eigenvalues are real. Thus the largest singular value is the largest magnitude of any eigenvalue. Since A_z is block-diagonal, its eigenvalues are the eigenvalues of each block. Thus we just need to bound the largest eigenvalue of $(1/m)zz^\top$. This is a rank-1 matrix whose sole eigenvector with nonzero eigenvalue is z, which has corresponding eigenvalue $(1/m)\|z\|_2^2=1/m$. Thus overall, Eq. (5.5) is bounded by

$$e^{-C\varepsilon^2/m}+e^{-C\varepsilon/m}$$

which is at most δ for $m=\Omega(\varepsilon^{-1}\log(1/\delta)+\varepsilon^{-2}\log(1/\delta))=\Omega(\varepsilon^{-2}\log(1/\delta))$, as desired.

Bibliography

[ACH+13] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Trans. Database Syst.*, 38(4):26:1–26:28, 2013. 28, 29

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467, 2012. 49

[AHLW16] Yuqing Ai, Wei Hu, Yi Li, and David P. Woodruff. New characterizations in turnstile streams with applications. In *Proceedings of the 31st Conference on Computational Complexity (CCC)*, pages 20:1–20:22, 2016. 43

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. 20, 31, 53

[BJKS04] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004. 38, 39, 53

[Bła20] Jarosław Błasiok. Optimal streaming and tracking distinct elements with high probability. *ACM Trans. Algorithms*, 16(1):3:1–3:28, 2020. 20

[BR94] Mihir Bellare and John Rompel. Randomness-efficient oblivious sampling. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–287, 1994. 51

[Bro93] Andrej Brodnik. Computation of the least significant set bit. In *ERK*, 1993. 23

[BYJK+02] Ziv Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *RANDOM*, pages 1–10, 2002. 19

[CCF04] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004. 47

[CK16] Amit Chakrabarti and Sagar Kale. Strong fooling sets for multi-player communication with applications to deterministic estimation of stream statistics. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 41–50, 2016. 34

[CKL+20] Graham Cormode, Zohar S. Karnin, Edo Liberty, Justin Thaler, and Pavel Veselý. Relative error streaming quantiles. *CoRR*, abs/2004.01668, 2020. 24

[CM05] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. 44, 46

[CMS76] John M. Chambers, Colin L. Mallows, , and B. W. Stuck. A method for simulating stable random variables. *J. Amer. Statist. Assoc.*, 71:340–344, 1976. 56

[CR12] Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. *SIAM J. Comput.*, 41(5):1299–1317, 2012. 40

[CV20] Graham Cormode and Pavel Veselý. A tight lower bound for comparison-based quantile summaries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 81–93, 2020. 23, 24

[CY20] Graham Cormode and Ke Yi. *Small Summaries for Big Data*. Cambridge University Press, 2020. To be published. Draft at `http://dimacs.rutgers.edu/~graham/ssbd.html`. 24, 26

[DKN10] Ilias Diakonikolas, Daniel M. Kane, and Jelani Nelson. Bounded independence fools degree-2 threshold functions. In *51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 11–20, 2010. 10

[dlPnG99] Victor de la Peña and Evarist Giné. *Decoupling: From dependence to independence*. Probability and its Applications. Springer-Verlag, New York, 1999. 9

[FEFGM07] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorith. In *Proceedings of the 2007 International Conference on the Analysis of Algorithms (AoFA)*, 2007. 20

[Fla85] Philippe Flajolet. Approximate counting: A detailed analysis. *BIT Comput. Sci. Sect.*, 25(1):113–134, 1985. 15

[FM85] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985. 16

[FW93] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993. 23

[GK01] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD)*, pages 58–66, 2001. 24

[GK16] Michael B. Greenwald and Sanjeev Khanna. Quantiles and equi-depth histograms over streams. In Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi, editors, *Data Stream Management - Processing High-Speed Data Streams*, Data-Centric Systems and Applications, pages 45–86. Springer, 2016. 24

[HW71] David Lee Hanson and Farroll Tim Wright. A bound on tail probabilities for quadratic forms in independent random variables. *Ann. Math. Statist.*, 42:1079–1083, 1971. 10

[ILL+19] Nikita Ivkin, Edo Liberty, Kevin J. Lang, Zohar S. Karnin, and Vladimir Braverman. Streaming quantiles algorithms with small space and update time. *CoRR*, abs/1907.00236, 2019. 24

[Ind06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006. 53, 54

[IW05] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 202–208, 2005. 39, 53

[JL84] William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984. 59

[JST11] Hossein Jowhari, Mert Saglam, and Gábor 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. 50

[JW13] T. S. Jayram and David P. Woodruff. Optimal bounds for Johnson-Lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms*, 9(3):26, 2013. 20

[KLL16] Zohar S. Karnin, Kevin J. Lang, and Edo Liberty. Optimal quantile approximation in streams. In *Proceedings of the 57th 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 71–78, 2016. 24, 30

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997. 37

[KNP+17] Michael Kapralov, Jelani Nelson, Jakub Pachocki, Zhengyu Wang, David P. Woodruff, and Mobin Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 475–486, 2017. 50

[KNW10a] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1161–1178, 2010. 55, 58

[KNW10b] Daniel M. Kane, Jelani Nelson, and David 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. 20

[KP20] John Kallaugher and Eric Price. Separations and equivalences between turnstile streaming and linear sketching. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1223–1236, 2020. 43

[KS92] Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discret. Math.*, 5(4):545–557, 1992. 38

[LNNT16] Kasper Green Larsen, Jelani Nelson, Huy L. Nguyễn, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2016. 47

[LNW14] Yi Li, Huy L. Nguyen, and David P. Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 174–183, 2014. 43

[Lum18] Jérémie O. Lumbroso. The story of HyperLogLog: How Flajolet processed streams with coin flips. *CoRR*, abs/1805.00612v2, 2018. 20

[McG14] Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, 2014. 49

[Mor78] Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, 1978. 13, 15

[MRL98] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 426–435, 1998. 24, 27

[Nel11] Jelani Nelson. *Sketching and Streaming High-Dimensional Vectors*. PhD thesis, Massachusetts Institute of Technology, June 2011. 54

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Comb.*, 12(4):449–461, 1992. 56

[Nol10] John P. Nolan. *Stable Distributions — Models for Heavy Tailed Data*. Birkhauser, 2010. 54

[NY19] Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1844–1860, 2019. 49, 53

[NY20] Jelani Nelson and Huacheng Yu. Optimal bounds for approximate counting, 2020. 15

[Raz92] Alexander A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106(2):385–390, 1992. 38

[Rud99] Mark Rudelson. Random vectors in the isotropic position. *J. Functional Analysis*, 164(1):60–72, 1999. 10

[RV13] Mark Rudelson and Roman Vershynin. Hanson-Wright inequality and sub-gaussian concentration. *arXiv*, abs/1306.2872, 2013. 9

[SBAS04] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 239–249, 2004. 24

[She12] Alexander A. Sherstov. The communication complexity of gap hamming distance. *Theory Comput.*, 8(1):197–208, 2012. 40

[SSS95] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM J. Discret. Math.*, 8(2):223–250, 1995. 51

[TSJ08] D. Sivakumar T. S. Jayram, Ravi Kumar. The one-way communication complexity of hamming distance. *Theory of Computing*, 4(1):129–135, 2008. 20, 40

[TZ12] Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM J. Comput.*, 41(2):293–331, 2012. 54

[Vid12] Thomas Vidick. A concentration inequality for the overlap of a vector on a large set, with application to the communication complexity of the gap-hamming-distance problem. *Chic. J. Theor. Comput. Sci.*, 2012. 40

[WC79] Mark N. Wegman and Larry Carter. New classes and applications of hash functions. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 175–182, 1979. 19

[Woo04] David P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 167–175, 2004. 20, 40

[Yu20] Huacheng Yu. Tight distributed sketching lower bound for connectivity. *CoRR*, abs/2007.12323, 2020. 49