1 Computation Paths

Streaming algorithms can be made robust by making failure probability low enough, namely:

$$\delta' = \delta \cdot n^{-O(\lambda_\varepsilon(f))}$$

1. Only need to change output $\lambda_\varepsilon(f)$ times
2. Stream is $\text{poly}(n)$-length and output is $O(\log n)$ bits so $n^{O(\lambda_\varepsilon(f))}$ computation paths between algorithm and adversary
3. Union bound over all of them!

Should think of the adversarially created stream as a tree of depth $\lambda_\varepsilon(f)$ and branching factor $\text{poly}(n)$, which gives $n^{O(\lambda_\varepsilon(f))}$ computation paths. This is because the adversary may choose $\lambda_\varepsilon(f)$ many points among $\text{poly}(n)$ choices of value and index at which to adjust stream.

2 Polynomially Bounded Adversaries

Our non-robust $F_0$-estimation algorithm was:

- Pick a hash function $h : [n] \to [m]$ where $m = O(n^2)$ (Birthday Paradox)
- Maintain smallest $t = \frac{100}{\varepsilon^2}$ values $h(i)$ from stream

Note that the state of the algorithm doesn’t change if you insert the same item twice, so breaking it requires breaking the hash function. This motivates using a cryptographic assumption on our hash function:

Assumption: For any $c > 0$ there is a $d > 0$ and a family of $n^d$ hash functions that can be evaluated in $O(\log n)$ memory such that any $n^c$-time Adversary cannot break this.

An example would be exponentially secure pseudorandom functions (AES or SHA256). This makes the algorithm robust against polynomially bounded adversaries.
3 Advanced Robust Algorithms

3.1 Differential Privacy

The general methods we have been using achieve a space complexity proportional to \( \lambda_\epsilon(f) \cdot (\text{space of non-robust sketch}) \). Using differential privacy techniques [Hassidim, Kaplan, Mansour, Matias, Stemmer] we can improve the dependence to \( \sqrt{\lambda_\epsilon(f)} \).

High-level idea is to compute the sketches \( S^1x, ..., S^{\sqrt{\lambda_\epsilon(f)}x} \) and take the median of the values \( R \). This doesn’t quite work, instead we want a “private median” which is roughly a random element in the middle quantile. This is useful for say, \( \ell_2 \)-norm with insertion and deletion, where the flip number is in the worst case \( \Theta(n) \) allowing us to achieve \( O(\sqrt{n} \cdot \frac{\log n}{\epsilon}) \). \( \ell - 2 \) with insertion and deletion is still wide open more outside of this more or less.

3.2 Difference Estimators

Instead of introducing a new sketch after every flip, which is necessary if the data changes greatly, if the difference is small, our estimator does not have to be \((1 \pm \epsilon)\) but can be a constant factor. For instance let’s say our previous value is \( f(x^1) = 10 \) and our updated value is \( f(x^2) = 10 + 7\epsilon \). We can reuse our estimate for \( f(x^1) \) and give a rough estimate of the difference \( f(x^2) - f(x^1) \) to get an estimate of \( f(x^2) \).

In the \( \ell_2 \)-norm case this might look like estimating \( |x^2|_2^2 - |x^1|_2^2 \) to \( O(\epsilon)|x^1|_2^2 \).

\[
O(\epsilon)|x^1|_2^2 = O(\epsilon)f(x^1) = |x^2|_2^2 - |x^1|_2^2 = |x^1 - x^2|_2^2 + 2\langle x^1, x^1 - x^2 \rangle
\]

Both quantities just need to be sketched to a constant factor. The first can be done with a CountS-sketch with just \( O(\frac{1}{\epsilon}) \) rows. The dot product can be approximated with an approximate matrix product using a \( O\left(\frac{\log n}{\epsilon^2}\right) \) space sketch. This is \( O\left(\frac{\log n}{\epsilon^2}\right) \) space in total which is an improvement to resketching which would take \( O(\log n/\epsilon^2) \).

We could sequentially sketch all the differences but overall the error would snowball. Instead we will sketch differences in a binary tree. To estimate \( f(x^3) \) we could approximate \( f(x^2) - f(x) \) and \( f(x^3) - f(x^2) \) and combine. The difference between \( f(x^3) \) and \( f(x^2) \) is a single \( \epsilon \) factor, which is as we saw above. In the \( f(x^2) - f(x) \) case, we know the difference is a \( 2\epsilon \) factor apart, so we need to estimate to \( \frac{1}{2} \) the relative error.

In general we will use specific intervals in a binary tree fashion. For the \( \ell_2 \)-norm problem, on the bottom level of the tree we use \( O\left(\frac{1}{\epsilon}\right) \) sketches up to \( O\left(\frac{1}{\epsilon}\right) \) accuracy and on the top we do \( O(\log n) \) flips and \( O\left(\frac{1}{\epsilon^2}\right) \) memory.