Collaboration Statement: Andrew Koulogeorge and Eric Richardson

Problem 1: Heavy Hitters.

Solution.

procedure HEAVYHITTERS($\delta, \varepsilon, \varphi$)
- $m \leq \hat{m} \leq 10m$ (by assumption)
- Initialize $CountMin(\varepsilon, \frac{\delta}{\hat{m}})$ (Data structure from lecture)
- Initialize $MinHeap$ (Keep track of HH candidates)
- $\hat{M} = 0$ (Keep track of elements in stream so far)
- **for** $x \in stream$ **do**
  - $\hat{M} += 1$
  - Increment all counters in $CountMin$
  - $\hat{f}_x = Count(x)$ (Get estimate of frequency of x from $CountMin$)
  - Pop from $MinHeap$ if $|MinHeap| > \frac{1}{\varphi - \varepsilon}$
  - **if** $\hat{f}_x \geq M\varphi$ **then**
    - Push $(\hat{f}_x, x)$ onto $MinHeap$
- Assert $\hat{M} = m$ after the last element in $stream$ goes by
- $R = \{\}$
- **for** $x \in MinHeap$ **do**
  - $\hat{f}_x = Count(x)$
  - **if** $\hat{f}_x \geq \hat{M}\varphi$ **then**
    - Add to R
- Return R

Probability Guarantee Consider a bad element y such that $f_y \leq (\varphi - \varepsilon)m$ but $\hat{f}_y \geq m\varphi$. Recall the theorem from lecture that states for a $MinCount$ data structure with $\ln \frac{m}{\delta}$ hash functions we have that:

$$\mathbb{P}\left[f_y + m\varepsilon < \hat{f}_y\right] < \frac{\delta}{m}$$

$$\implies \mathbb{P}\left[m\varphi \leq \hat{f}_y\right] \leq \mathbb{P}\left[f_y + m\varepsilon < \hat{f}_y\right] < \frac{\delta}{m}$$

Where the second implication follows from $f_y \leq (\varphi - \varepsilon)m \implies f_y + m\varepsilon \leq m\varphi$. In short, the probability that a bad element y fools our $MinCount$ into thinking it is a heavy hitter is at most $\frac{\delta}{m}$ for a fixed y. Applying union bound over all m elements in the stream, we get the probability of a any bad element fooling us to be at most δ. □

For the rest of the analysis assume we are in this nice case that no bad elements y exist. Now we need to show that in this case that occurs at least $1-\delta$ of the time, we meet the various requirements.

Space Requirement + Correctness: We keep a $CountMin$ data structure with $O(\frac{1}{\varepsilon}\log\frac{m}{\delta})$ words since $m \leq \hat{m} \leq 10m$. We also keep a $MinHeap$ to store heavy hitter candidates. To ensure that our algorithm is correct, we must ensure that all heavy hitters in the heap after the stream passes. Assume the size of the heap is some constant c and we will derive a necessary lower bound.

To begin, fix a heavy hitter z and consider the last time z appears in the stream. Since $CountMin$ always overestimates the true frequency of an element, z is a heavy hitter and $m \geq M$ at all points in the stream:

$$\hat{f}_z \geq f_z \geq m\varphi \geq M\varphi$$

This implies for every heavy hitter z, it will be in the heap after the last time it passes in the stream. Now we need to show that its impossible for this heavy hitter z to be evicted after its last instance has passed in the stream. Suppose for contradiction that z was popped from $MinHeap$ after it passed the stream for the last time. This implies that it has the smallest frequency in $MinHeap$:

$$\forall\, j \in MinHeap:\ \hat{f}_j \geq \hat{f}_z \geq f_z \geq m\varphi \geq M\varphi$$

This is, all elements j currently in the heap have the property that $\hat{f}_j \geq m\varphi$. Since we are reasoning in the case where $\nexists\, y\ s.t\ \hat{f}_y \geq m\varphi$ and $f_y \leq m(\varphi-\varepsilon) \implies f_j > m(\varphi-\varepsilon)\ \forall\, j \in MinHeap$. This implies the total number of elements we have seen in the stream so far $\geq cf_j > cm(\varphi-\varepsilon)$. If $c \geq \dfrac{1}{\varphi-\varepsilon} \implies$ elements seen in the stream so far $> m$, a contradiction. Thus, as long as the size of the heap is at least as large as $\dfrac{1}{\varphi-\varepsilon}$, $MinHeap$ will contain all the heavy hitters with probability at least $1-\delta$. If $\varepsilon < \dfrac{\varphi}{2} \implies 2\varepsilon < \varphi \implies \varepsilon < \varphi - \varepsilon \implies \dfrac{1}{\varepsilon} > \dfrac{1}{\varphi-\varepsilon} \implies$ heap size is $O(\frac{1}{\varepsilon})$ □

Thus, the total space of the algorithm is dominated by the $MinCount$ table: $O(\frac{1}{\varepsilon}\log\frac{m}{\delta})$

During Stream Time Requirement: When an element x passed by in the stream, we update all the counters in the $CountMin$ data structure. There are $\log\frac{m}{\delta}$ counters to update and we assume we can compute each hash function in constant time. we then compute the current frequency estimate of x which takes the min across all $\log\frac{m}{\delta}$ counters. If x is estimated to be a heavy hitter so far, we push it onto the heap which takes $O(\log\frac{1}{\varepsilon})$ time since we argued that the heap size is $O(\frac{1}{\varepsilon})$. Removing candidates when the heap becomes

full also takes $O(\log \frac{1}{\varepsilon})$. Since we only evict from the heap when it fills up, we do at most 2 heap operations for each element in the stream. Thus, the total time taken for a single element in the stream $= O(\log \frac{m}{\delta}) + O(\log \frac{1}{\varepsilon}) = O(\log \frac{m}{\delta})$. (Assuming $\frac{m}{\delta} > \frac{1}{\varepsilon}$)

Post Stream Time Requirement: After all the elements in the stream pass, we loop over each element in the heap, compute the estimate for its frequency, and place it in R if our estimate tells us its a heavy hitter. Since our heap is of size $O(\frac{1}{\varepsilon})$ and each query to $MinCount$ takes $O(\log \frac{m}{\delta})$ time $\implies$ Post Stream Time Requirement = $O(\frac{1}{\varepsilon} \log \frac{m}{\delta})$

Problem 2: A Better Analysis of COUNT-SKETCH.

Solution.
In the COUNT-SKETCH framework, suppose we define H to be the set of elements i with the $\lceil \frac{1}{\varepsilon^2} \rceil$-largest $\mathbf{f}_i$'s. Let $L := [n] \backslash H$ represent the long tail of low frequency elements. Then, define

$$||\mathbf{f}||_{tail} = \sqrt{\sum_{i \in L} \mathbf{f}_i^2}$$

We wish to improve the bound given in class by showing that

$$\mathbf{Pr}[|\hat{\mathbf{f}}_i - \mathbf{f}_i| \geq \varepsilon ||\mathbf{f}||_{tail}] \leq \frac{1}{3}$$

To begin, suppose we fix $i \in [n]$. Then, by the analysis given in class, we have that the entry in the counter table C at index $h(i)$, where h is drawn independently from a UHF with domain size $[n]$ and range k, is given by

$$C[h(i)] = g(i) \cdot \mathbf{f}_i + \sum_{j \neq i : h(j) = h(i)} g(j) \cdot \mathbf{f}_j$$

Also note that g is drawn from a strongly independent UHF with domain size $[n]$ and range $\{-1, 1\}$. Then, since our estimate $\hat{\mathbf{f}}_i$ is equivalent to $C[h(i)] \cdot g(i)$, we have

$$\begin{aligned} \hat{\mathbf{f}}_i &= g(i) \cdot (g(i) \cdot \mathbf{f}_i + \sum_{j \neq i : h(j) = h(i)} g(j) \cdot \mathbf{f}_j) \\ &= \mathbf{f}_i + \sum_{j \neq i : h(j) = h(i)} g(i) g(j) \cdot \mathbf{f}_j \end{aligned}$$

Intuitively, this represents our ground truth value $\mathbf{f}_i$ in addition to some error terms that may occur due to hash collisions. Since H and L are disjoint, we can split this summation

over collisions into those that occur in H and those that occur in L:

$$\hat{\mathbf{f}}_i = \mathbf{f}_i + \sum_{j \neq i \in H : h(j) = h(i)} g(i)g(j)\mathbf{f}_j + \sum_{k \neq i \in L : h(k) = h(i)} g(i)g(k)\mathbf{f}_k$$

$$\implies \hat{\mathbf{f}}_i - \mathbf{f}_i = \sum_{j \neq i \in H : h(j) = h(i)} g(i)g(j)\mathbf{f}_j + \sum_{k \neq i \in L : h(k) = h(i)} g(i)g(k)\mathbf{f}_k$$

Let A be a random variable that takes the value of the summation over H, and B a random variable that takes the value of the summation over L. First, note that $\textbf{Exp}[A] = \textbf{Exp}[B] = 0$ due to the fact that $\textbf{Exp}[g(i)g(j)] = 0$ for any $j \neq i$. We now calculate the variance of B:

$$\begin{aligned}
\textbf{Var}[B] &= \textbf{Exp}[B^2] - \textbf{Exp}[B]^2 \\
&= \textbf{Exp}[B^2] \\
&= \sum_{j \neq i \in H} \textbf{Pr}[h(j) = h(i)] \cdot g(i)^2 g(j)^2 \mathbf{f}_j^2 \\
&\leq \sum_{j \neq i \in H} \frac{1}{k} \cdot \mathbf{f}_j^2 \\
&= \frac{||f||_{tail}^2}{k}
\end{aligned}$$

Then, applying Chebyshev with $t = \varepsilon ||\mathbf{f}||_{tail}$, we have

$$\begin{aligned}
\textbf{Pr}[|B| \geq \varepsilon ||\mathbf{f}||_{tail}] &\leq \frac{||\mathbf{f}||_{tail}^2}{k\varepsilon^2 ||\mathbf{f}||_{tail}^2} \\
&= \frac{1}{10}
\end{aligned}$$

where the last step follows with $k = \dfrac{10}{\varepsilon^2}$. Thus, we have bounded the probability that the magnitude of error accumulated by collisions from L exceeds $\varepsilon ||\mathbf{f}||_{tail}$. Now, consider A. Trivially, the probability that $A \geq \varepsilon ||\mathbf{f}||_{tail}$ can be no greater than the probability that some element in H collides with i. More generally, we have that

$$\textbf{Pr}[|A| \geq \varepsilon ||\mathbf{f}||_{tail}] \leq \textbf{Pr}[A \neq 0]$$

Since the only case in which A is nonzero is when there exists some element $j \in H$ such that $h(j) = h(i)$, we can apply a union bound over the probability that a hash collision occurs in H:

$$\begin{aligned}
\textbf{Pr}[A \neq 0] &\leq \textbf{Pr}[\exists j \in H : h(j) = h(i)] \\
&\leq \sum_{j \in H} \frac{1}{k} \\
&= \frac{1}{k\varepsilon^2} \\
&= \frac{1}{10}
\end{aligned}$$

Therefore, we have that $\mathbf{Pr}[|A| \geq \varepsilon||\mathbf{f}||_{tail}]$ and $\mathbf{Pr}[|B| \geq \varepsilon||\mathbf{f}||_{tail}]$ are both at most $\frac{1}{10}$. Now, let $\mathcal{E}$ denote the event that $|\hat{\mathbf{f}}_i - \mathbf{f}_i| \geq \varepsilon||\mathbf{f}||_{tail}$. Suppose $A = 0$, then the only case in which $\mathcal{E}$ occurs is when $|B| \geq \varepsilon||\mathbf{f}||_{tail}$. Similarly, suppose $|B| < \varepsilon||\mathbf{f}||_{tail}$. Then, it is impossible for $\mathcal{E}$ to occur unless $A \neq 0$. Thus, it follows that

$$\begin{aligned}
\mathbf{Pr}[\mathcal{E}] &\leq \mathbf{Pr}[A \neq 0 \vee |B| \geq \varepsilon||\mathbf{f}||_{tail}] \\
&\leq \mathbf{Pr}[A \neq 0] + \mathbf{Pr}[|B| \geq \varepsilon||\mathbf{f}||_{tail}] \\
&= \frac{1}{10} + \frac{1}{10} \\
&= \frac{1}{5}
\end{aligned}$$

Hence, COUNT-SKETCH with $k \leq \frac{10}{\varepsilon^2}$ satisfies

$$\mathbf{Pr}[|\hat{\mathbf{f}}_i - \mathbf{f}_i| \geq \varepsilon||\mathbf{f}||_{tail}] \leq \frac{1}{3}$$

for every $i \in [n]$.

Problem 3: Single Element Recovery.

Solution.
Given a binary vector $x \in \{0,1\}^n$, we wish to find an index $i \in [n]$ such that $x_i = 1$ for some $x_i \in x$. A sum query is the sum of a subset of elements from x denoted by $x(A_k) = \sum_{j \in A_k} x_j$ where A_k is a subset of index's from $[n]$.

(a) Suppose that we knew that exactly 1 entry $x_i \in x$ was 1 and the rest were zero $\implies \sum_{i=1}^{n} x_i = 1$. We wish to identify a deterministic algorithm such that $\forall\ x \in \{0,1\}^n\ s.t\ \sum_{i=1}^{n} x_i = 1$, our algorithm computes at most $\log_2(n)$ sum queries and returns the index $i \in [n]\ s.t\ x_i = 1$.

We will compute which index $i \in [n]$ is 1 in x by constructing the binary representation of i using each value of $x(A_k)$ as a single bit. Consider the binary string formed by the concatenation of all of the results of $x(A_k)$ in decreasing order:

$$(x(A_{\log_2 n}), x(A_{\log_2 n - 1}), ..., x(A_2), x(A_1))$$

We wish to construct these subsets such that when each query sum is computed the above tuple equals $i - 1$ in binary. For the construction, consider each index $i = 1 \to [n]$. For each i, we compute the binary representation of $i - 1$. Note that this binary representation has at most $\log_2(n)$ bits. For each subset A_k, add i to the subset A_k if the kth digit in the binary representation of $i - 1$ is 1. Based on this

construction, when the ith element in x is set to 1 the tuple of query sums above will spell out the index i in binary minus 1 (in the case where n is a power of 2, all $1s \implies i = n$ and all $0s \implies i = 1$. □

(b) Now suppose that $\sum_{i=1}^{n} x_i = d$. We still wish to recover an any $i \in [n]$ where $x_i = 1$. Define a "nonzero index" as an index i such that $x_i = 1$. Also recall that we need to ask all of our questions about query sums at once. Consider taking T subsets from x: $R_1, R_2, ..., R_T$ where each element of x is sampled into R_k independently with probability $\frac{1}{d}$. Note that each of these subsets are i.i.d.

Procedure: For each R_k, we can construct a bit vector x_k which contains the elements of x indexed by the elements of R_k. We can then apply the algorithm from part (a) on this collection of bit vectors AND compute the sum of all the entries of x_k. This requires at most $\log_2(n)T + T$ sum queries since the length of x_t is at most n. First, we check if one of these subsets has exactly 1 nonzero index by using the sum query over all the index's of the subset. If we sampled a subset R with this property, we will be able to deterministically find a nonzero index by applying the algorithm from part (a). Thus, all that is left to show is that the probability that all of the subsets do not contain exactly one nonzero index is small. Let Q_i be the number of nonzero index's in the subset R_i:

$$\mathbb{P}[Fail] = \mathbb{P}[Q_1 \neq 1 \cap Q_2 \neq 1 \cap ... \cap Q_T \neq 1] = \prod_{i=1}^{T} \mathbb{P}[Q_i \neq 1]$$

Since Q_i is a binomial random variable with d trails, one for each of the nonzero index's, each having a success probability of $\frac{1}{d}$ we have:

$$\mathbb{P}[Q_i = 1] = \binom{d}{1}\frac{1}{d}(1 - \frac{1}{d})^{d-1} \geq 1 - \frac{d-1}{d} = \frac{1}{d}$$

$$\implies \mathbb{P}[Q_i \neq 1] \leq 1 - \frac{1}{d}$$

where the equality is the definition of the binomial pmf and the inequality follows from $(1-x)^y \geq 1 - xy$ for $x \geq 0$. Since each Q^i is identically distributed we can apply this upper bound for each of the T terms:

$$\mathbb{P}[Fail] = \prod_{i=1}^{T} \mathbb{P}[Q_i \neq 1] \leq (1 - \frac{1}{d})^T \leq \exp\frac{-T}{d} \leq \delta$$

$$\implies T \geq d \log\frac{1}{\delta}$$

Thus, by using $d \log\frac{1}{\delta}(\log_2 n + 1) = O(\log\frac{1}{\delta}\log_2 n)$ query sums we can find a nonzero index of x with probability at least $1 - \delta$. □

(c) Now suppose that instead of knowing the exact value of $\sum_{i=1}^{n} x_i$ beforehand, we know that:

$$d \leq \sum_{i=1}^{n} x_i \leq 2d$$

Let $y = \sum_{i=1}^{n} x_i$. I claim that we can still apply the algorithm in the previous part except now we will sample each element from x into R_k with probability $\frac{1}{2d}$. The difference is that now the Binomial random variable Q_i has y trials and has a probability of success of $\frac{1}{2d}$:

$$d \leq y \leq 2d \implies \frac{1}{2} \leq \frac{y}{2d} \leq 1$$

$$\mathbb{P}[Q_i = 1] = \binom{y}{1} \frac{1}{2d}(1-\frac{1}{2d})^{y-1} \geq \frac{y}{2d}(1-\frac{y-1}{2d}) \geq \frac{1}{2}(1-\frac{y-1}{2d}) = \frac{1}{2}(1-\frac{y}{2d}+\frac{1}{2d}) \geq \frac{1}{4d}$$

$$\implies \mathbb{P}[Q_i \neq 1] \leq 1 - \frac{1}{4d}$$

The same analysis from the previous question now holds. We get a larger constant value on d but it still requires $O(\log \frac{1}{\delta} \log_2 n)$ query sums to find a nonzero index with probability $\geq 1 - \delta$. Qualitatively, if we can construct an estimate for y within an interval $d \leq y \leq 2d$ we can find a non negative index in x w.h.p $\square$

(d) Suppose that $y = \sum_{i=1}^{n} x_i$ is completely unknown. Let $d = 2^i$ for $1 \leq i \leq \lfloor \log_2 n \rfloor$. If $2 \leq y \leq n \implies \exists\, k,\ 1 \leq k \leq \lfloor \log_2 n \rfloor\ s.t\ 2^k \leq y < 2^{k+1} \implies d \leq y < 2d$. In short, we know there exists an interval around y which is length d, but since we do not know d up front, we do not know the exact value of d such that sampling iid from $[n]$ with probability $\frac{1}{2d}$ inherits the results from part (c). If we knew d, we could apply part (c) and recover a nonzero index in $O(\log \frac{1}{\delta} \log_2 n)$ query sums.

Here is the idea to "get" d: sample $\lfloor \log_2 n \rfloor$ index subsets $R_1, R_2, ...R_{\lfloor \log_2 n \rfloor}$ from $[n]$ where R_i samples each index from $[n]$ independently with probability $\frac{1}{2^{i+1}}$. We dont know for which k y will lie in an interval $2^k \leq y < 2^{k+1}$, so exhaust all possible upper bounds by sampling $\lfloor \log_2 n \rfloor$ different subsets $\implies$ there will exist a subset R_k for which we can apply the results from part c.

The boosting is being applied on each of the subsets individually. For each $R_1, R_2, ...R_{\lfloor \log_2 n \rfloor}$ apply the algorithm from part b which takes a total of $O(log^2 n)$ queries. We showed

that one of these subsets will be sampling with a probability within a constant factor of $\frac{1}{y}$ so we can boost this process $O(log\frac{1}{\delta})$ times $\implies$ if we dont know the number of nonzero entries in x upfront, we can still recover a nonzero index i with $O(log^2 nlog\frac{1}{\delta}))$ queries with probability at least $1-\delta$ $\square$