CS 761: Randomized Algorithms	Fall 2019
Problem Set 2	
Prof. Gautam Kamath	*Deadline: 11:59 PM on November 3, 2019*

You are allowed to discuss the problems in small groups (2-4 people). List your collaborators for each problem. Every person must write up and submit their own solutions.

1. **Sampling With and Without Replacement.** Suppose we are given a set of k items sampled uniformly at random *without replacement*, from an unknown set of size n, where n is known. Show how to use this to construct a set of k items sampled uniformly at random *with replacement* from the same (unknown) set.

2. **k-wise Hash Function Families.** This problem is due to Eric Price. Consider a strongly k-universal family of hash functions $\mathcal{H}$ from $[m]$ to $[n]$, where $k = O(1)$. For a set of items $S \subset [m]$, let X be the random variable which is the load in the first bin, $X = |\{i \in S \mid h(i) = 1\}|$, where the randomness is over the uniform selection of h from $\mathcal{H}$.

   (a) For any $t \geq 1$ and set S with $|S| = n$, show that $\Pr[X \geq t] \leq O(1) \cdot 1/t^k$. Hint: Bound $E[X^k]$.

   (b) Use the previous part to show that, in the same setting, the expected maximum load of any bin is $\leq O(1) \cdot n^{1/k}$.

   (c) Based on these results, what value of k would you predict is needed to attain a max load of $O(\log n / \log \log n)$, as in the ideal case? Note that this is not formal, since we assumed $k = O(1)$, but should give you roughly the right answer.

   (d) (Optional) In the special case where $m = n$ and $S = [n]$, show that there exists a universal hash family $\mathcal{H}$ such that the expected maximum load is $\Omega(\sqrt{n})$.

3. **Python Hashing is Broken.** This problem is due to Eric Price. The Python programming language uses hash tables (or "dictionaries") internally in many places. Until 2012, however, the hash function was not randomized: keys that collided in one Python program would do so for every other program. To avoid denial of service attacks, Python implemented hash randomization – but there was an issue with the initial implementation. Also, in Python 2, hash randomization is still not the default: one must enable it with the `-R` flag.

   (a) First, lets look at the behavior of `hash("a")-hash("b")` over n = 2000 different initializations. If hash were pairwise independent over the range (64-bit integers, on a 64-bit machine), how many times should we see the same value appear?

   (b) How many times do we see the same value appear, for three different instantiations of python: (I) no randomization (`python2`), (II) Python 2's hash randomization (`python2 -R`), and (III) Python 3's hash randomization (`python3`)? If you have trouble coding this on your own, the following snippet lets you get the answer:

```
for i in `seq 1 2000`; do
python2 -R -c 'print((hash("a")-hash("b")))';
done | sort | uniq -c | awk '{print $1}' | sort -n | uniq -c
```

(c) What might be going on here? Roughly how many different hash functions does this suggest that each version has?

(d) The above suggests that Python 2's hash randomization is broken, but does not yet demonstrate a practical issue. Lets show that large collision probabilities happen. Observe that the strings "8177111679642921702" and "6826764379386829346" hash to the same value in non-randomized python 2. Check how often those two keys hash to the same value under `python2 -R`. What fraction of runs do they collide? Run it enough times to estimate the fraction to within 20% multiplicative error, with good probability. How could an attacker use this behavior to denial of service attack a website?

(e) (Optional) Find other pairs of inputs that collide.

4. **Count-min sketch with Tug-of-War.** In this problem, we will combine ideas from Count-min sketch for finding heavy-hitters with the Alon-Matias-Szegedy algorithm for estimating the ℓ_2 frequency moment of a stream. This will allow us to estimate heavy hitters of a stream with a tighter guarantee in certain cases.

Recall that in Count-Min Sketch, we maintained d hash functions $h_1, \ldots, h_d$, corresponding to d hash tables, each of size w. For the datum that appears at time t, (i_t, c_t) where i_t is the identifier, and c_t is a count, for each $j \in [d]$, we increment a counter C_j in entry $h_j(i_t)$ of the jth hash table by c_t. At the end of the stream, for a given identifier i, we can return $\hat{f}_i = \min_{j \in [d]} C_j(h_j(i))$ to get an estimate of $f_i = \sum_{t: i_t = i} c_t$. In particular, setting $w = O(1/\varepsilon)$ and $d = O(\log(1/\delta))$, with probability at least $1 - \delta$, this will give an estimate $|\hat{f}_i - f_i| \leq \varepsilon F_1$, where $F_1 = \sum_i f_i$ (we assume that $f_i \geq 0$ for all i).

Consider making the following changes to the algorithm. Instead of storing just d hash functions, we instead store $2d$ hash functions. The second set of hash functions, $g_1, \ldots, g_d$ maps to the range $\{\pm 1\}$. The modification to counter C_j at time t is still at entry $h_j(i_t)$, but now we increment it by $g_j(i_t)c_t$. Finally, our estimate $\hat{f}_i$ is now $\text{median}_{j \in [d]}\, g_j(i) C_j(h_j(i))$. We will obtain a guarantee which is in terms of $\sqrt{F_2}$, where $F_2 = \sum_i f_i^2$. Let $\hat{f}_{ij} = g_j(i) C_j(h_j(i))$.

(a) For some given i and j, compute $E[\hat{f}_{ij}]$.

(b) For some given i and j, upper bound $Var[\hat{f}_{ij}]$.

(c) Given these two quantities, choose values of d and w, upper-bounding the probability that $|\hat{f}_{ij} - f_i| \geq \varepsilon\sqrt{F_2}$ by a constant, and (in turn) upper-bounding the probability that $|\hat{f}_i - f_i| \geq \varepsilon\sqrt{F_2}$ by δ.

(d) Compare this type of guarantee with that of Count-Min Sketch. When is each guarantee better? Give a set of frequencies (i.e., a set of f_i's) illustrating where one might be better than the other.

5. **ℓ_1-dimension reduction.** This problem is due to Ilya Razenshteyn. In class, we explored dimension reduction in ℓ_2. We will now consider dimension reduction in ℓ_1. While the natural adaptation of Johnson-Lindenstrauss does not work, we will consider an alternative via sketching.

Consider the following map $f(x)$ for $x \in \mathbb{R}^d$. Define the vector $y_i = x_i/u_i$, where the u_i's are chosen as i.i.d. random variables from the exponential distribution $Exp(1)$, where $Exp(\lambda)$ has density function $p(t) = \lambda e^{-\lambda t}$. The sketch $f(x)$ is the algorithm in the previous problem ("Count-min sketch with Tug-of-War") applied to the vector y (we will set the parameters w and d in the steps of this problem).

The estimator uses said algorithm to extract the heavy hitters of y and outputs the largest one (in absolute value). Note that the resulting sketch is linear: specifically, to estimate $\|x - x'\|_1$ given sketches $f(x)$ and $f(x')$, it suffices to run the estimator on $f(x) - f(x')$.

(a) Let $X_1, \ldots, X_n$ be independent samples from $Exp(\lambda_1), \ldots Exp(\lambda_n)$, respectively. Show that $\min\{X_1, \ldots, X_n\}$ is distributed as $Exp(\lambda_1 + \cdots + \lambda_n)$.

(b) Prove that the largest (absolute value of a) coordinate of y, termed q, is within a constant factor of $\sum_{i=1}^{d} |x_i|$, with probability at least 0.95.

(c) Prove that q is a ϕ-heavy hitter (in the vector y) with probability at least 0.9, where $\phi = c/\log d$ for some small positive constant $c > 0$. In this context, we say an element q is a ϕ-heavy hitter if $q \geq \phi\|y\|_1$.

Hint: Prove that all $u_i \geq 1/(\lambda d^2)$ with probability $1 - \Omega(1/d)$. Conditioned on that, prove a convenient bound on the expectation of $\|y\|_1$ and use Markov's inequality.

Another hint: You may use the fact that exponential random variables are memoryless, without proof. Namely, if $X \sim Exp(\lambda)$, $\Pr(X > s + t | X > s) = \Pr(X > t)$ for all $s, t \geq 0$. Optionally, you may prove this as well.

Yet another hint: I recommend not looking at this hint until you find yourself wondering how to further simplify an unfamiliar mathematical expression – it is far more satisfying if you arrive at that point yourself. Regardless, it's here.

(d) Prove that, for ϕ from part c) with constant c sufficiently small, the estimator is a constant factor approximation to $\sum |x_i|$ with probability at least 0.6.

(e) Conclude that the space of the overall ℓ_1 sketch is $O(\log^{O(1)} d)$.

(f) (Optional) Instead of a constant-factor approximation, show how to modify the algorithm to obtain a $(1 \pm \varepsilon)$ approximation at a small increase in the amount of space.

6. **A Game of Coins.** Suppose two people are playing the following game, which starts with k coins on the number 0, and the game is played on the number line $\{0, 1, \ldots, n\}$. On Player 1's turn, they select two disjoint subsets of the coins A and B (note that $A \cup B$ does not need to contain all the coins). On Player 2's turn, they remove all the coins from either A or B, while the coins in the other set move one space forward to the next number. Player 1 wins if a coin reaches n. Player 2 wins if there is only one coin remaining, that has not reached n.

   (a) Construct a winning strategy for Player 1 if $k \geq 2^n$.

   (b) Show that there exists a winning strategy for Player 2 using the probabilistic method if $k < 2^n$.

   (c) "Derandomize" your argument for the previous part to give a (deterministic) winning strategy for Player 2 if $k < 2^n$.

7. **Graph Colouring.** We are given a graph $G = (V, E)$, along with a set C_v of $8r$ colours for each $v \in V$ and r is a positive integer. There are at most r neighbors u of v containing colour c, for any node $v \in V$ and colour $c \in C_v$. Recall that a proper colouring of graph requires that every two nodes connected by an edge are distinct colours. Show that such a colouring exists.