CS369G: Algorithmic Techniques for Big Data	Spring 2015-2016
Problem Set 2	
Prof. Moses Charikar	*Due: May 10, 2016,* ***1:29pm***

Policy: You are permitted to discuss and collaborate on the homework but you must write up your solutions on your own. Furthermore, you need to cite your collaborators and/or any sources that you consulted. All homework submissions are subject to the Stanford Honor Code.

Submission: homework is due before class on Tuesday May 10, 2016. Email your solution to `cs369g.stanford@gmail.com` using the subject, `[PS2_2016]SUNetID` (e.g `[PS2_2016]jdoe`). Late submissions will not be accepted.

Length of submissions: include as much of the calculations that show that you understand everything that is going through the answer. As a rule of thumb after you have solved the problem, try to identify what are the main steps taken and critical points of a proof and include them. Unnecessary long answers to questions will be penalized. The points next to each question are indicative of the hardness/length of the proof.

1 Estimating join size through sketches [10 points]

To a scientist interested in Databases, the second frequency moment F_2 can be motivated by observing that it is the size of a self-join: the join of a relation in a database with itself. In fact, one can design a sketch that can scan a relation in one pass (i.e., in streaming fashion) such that, based on the sketches of two different relations, we can estimate the size of their join. Explain how.

Recall that for two relations (i.e., tables in a database) $r(A, B)$ and $s(A, C)$, with a common attribute (i.e., column) A, we define the join $r \bowtie s$ to be a relation consisting of all tuples (a, b, c) such that $(a, b) \in r$ and $(a, c) \in s$. Therefore, if $f_j^{(r)}$ and $f_j^{(s)}$ denote the frequencies of j in the first columns (i.e., "A"-columns) of r and s, respectively, and j can take values in $[n]$, then the size of the join is $\sum_{j=1}^{n} f_j^{(r)} f_j^{(s)}$.

2 Revisiting Moment Estimation through Max-Stability [40 points]

Recall the algorithm covered in class for p norm estimation, $p > 2$. We stored the sketch $y = \Pi D x$ for D a certain $n \times n$ diagonal matrix, and Π is the $m \times n$ matrix from Problem Set 1, Problem 3. That is, there are hash functions $h : [n] \to [m]$ and $: [n] \to 1, 1$, and

$$\Pi_{i,j} = \begin{cases} \sigma(j), & \text{if } i = h_j \\ 0, & \text{otherwise} \end{cases}$$

For parts (a)-(c) below, we are content with an O(1)-approximation to $\|x\|_p$, as in lecture.

(a) **[8 points]** Suppose D has totally independent entries, as in the analysis from class. In class, we also assumed that h, σ were both totally random hash functions. What level of

independence is required from the hash families that h, σ are drawn from to ensure that the proof from class is valid? You may assume that for completely random hash functions h, σ and any bucket $j \in [m]$ the contribution $Z_j^{'}$ (as in the lecture notes) of the small elements satisfies the following tail bound $\mathbb{E}[|Z_j^{'}|^k] \leq \frac{L}{c^k}\|x\|_p^k$,for $k \geq 1$, where $L > 0$ is an absolute constant and $c > 0$ is the constant used to define large elements. Express the minimum amount of independence needed to achieve $\mathbb{P}(|Z_j^{'}| > \alpha M) \leq \frac{\delta}{m}$ in terms of the constants α, m, c, δ, L.

(b) **[2 points]** Suppose you are given a uniform random variable $r \in (0, 1)$. How can you use r to generate an exponential random variable?

(c) **[20 points]** Show how to use Nisan's pseudorandom generator to derandomize the generation of the matrix D. You may assume that $O(\log n)$ uniform random bits suffice to generate the r from part (b), in order to obtain an exponential random variable with sufficient precision for use in this algorithm (that is, we do not require you to perform any precision analyses in this homework problem).

(d) **[10 points]** How would you modify the algorithm and analysis from class to achieve not just a constant factor approximation to $\|x\|_p$, but a $(1+\epsilon)$-approximation with 2/3 probability? Your space bound should be $n^{1-2/p} \cdot \text{poly}(\epsilon^{-1} \log n)$ machine words

3 Approximate Frequency Estimation [30 points]

Algorithm 1

Input: stream $i_1, i_2, \ldots, i_m \in [n]$, number of bins k.
initialize each bin $b \in [k]$ with an element $e_b \leftarrow \emptyset$ (initially null) and a counter $c_b \leftarrow 0$.
for each element i_ℓ in the stream **do**
  if i_ℓ is in a bin b **then**
    increment b's counter $c_b \leftarrow c_b + 1$
  else
    find the bucket b_ℓ with the smallest counter value (breaking ties arbitrarily)
    replace the current element $e_{b_\ell} \leftarrow i_\ell$
    increment its counter $c_{b_\ell} \leftarrow c_{b_\ell} + 1$.
Output: for each $i \in [n]$ output $\hat{f}_i = c_b$ if $e_b = i$ and 0 otherwise.

Provide an error guarantee about the estimates returned by this algorithm.

4 Approximating the Median [20 points]

You are given an insertion only stream, i.e. a vector $x \in \mathbb{R}^n$ starts as the 0 vector and a stream $i_1 i_2 \cdots i_m$ with each $i \in [n]$ causes the change $x_i \leftarrow x_i + 1$. Design a one-pass streaming algorithm which uses little space and can process each update i in the stream quickly in order to answer the

following approximate median query: output some $j \in [n]$ such that

$$\sum_{i=1}^{j} x_i \geq m/2 - \epsilon m \qquad \text{and} \qquad \sum_{i=1}^{j-1} x_i \leq m/2 + \epsilon m.$$

Your solution should quickly answer queries and also use little space.

(a) **[12 points]** Give an deterministic algorithm solve this problem with space $O(\epsilon^{-1} \log^2 n)$ machine words, and update time $O(\epsilon^{-1} \log^2 n)$.

(b) **[8 points]** How can you decrease the update time to $O(\log(n))$ in expectation by allowing randomization?