CSE 521: Design and Analysis of Algorithms **Fall 2019**

Problem Set 2

Deadline: Oct 27 (at 6:00 PM) in Canvas

Instructions

- You should think about each problem by yourself for at least an hour before choosing to collaborate with others.
- You are allowed to collaborate with fellow students taking the class in solving the problems (in groups of at most 2 people for each problem). But you **must** write your solution on your own.
- You are not allowed to search for answers or hints on the web. You are encouraged to contact the instructor or the TAs for a possible hint.
- You cannot collaborate on Extra credit problems
- Solutions typeset in LATEX are preferred.
- Feel free to use the Discussion Board or email the instructor or the TA if you have any questions or would like any clarifications about the problems.
- Please upload your solutions to Canvas. The solution to each problem **must** be uploaded separately.

---

1) In this problem we see how to use pairwise independent hash functions for de-randomization. Say A is a randomized algorithm that uses m random bits and will output the optimum solution of a minimization problem with probability $1/2$. In the first lecture we argued that we can improve the success probability to $1-1/2^k$ by simply running k independent copies of A and return the minimum outputted solution. But that needs $O(km)$ random bits. Prove that for any $r \leq 2^m$, we can improve the success probability to $1-1/r$ using $O(m)$ random bits by running A only $O(r)$ many times. Note that the number of random bits is independent of r.

2) a) **Optional [0-points]:** Let $X_1, \ldots, X_n$ be independent random variables uniformly distributed in $[0,1]$ and let $Y = \min\{X_1, \ldots, X_n\}$. Show that $\mathbb{E}[Y] = \frac{1}{n+1}$ and $\mathrm{Var}(Y) \leq \frac{1}{(n+1)^2}$.

   Consider the following algorithm for estimating F_0, the number of unique elements in a sequence $x_1, \ldots, x_m$ in the set $\{0, 1, \ldots, n-1\}$. Let $h : \{0, 1, \ldots, n-1\} \to [0,1]$ s.t., $h(i)$ is chosen uniformly and independently at random in $[0,1]$ for each i. We start with $Y = 1$. After reading each element x_i in the sequence we let $Y = \min\{Y, h(x_i)\}$.

   b) Show that by the end of the stream $\frac{1}{\mathbb{E}[Y]} - 1$ is equal to F_0.

   c) Use the above idea to design a streaming algorithm to estimate the number of distinct elements in the sequence with multiplicative error $1 \pm \epsilon$. For the analysis you can assume that you have access to k independent hash functions as described above. Show that $k \leq O(1/\epsilon^2)$ many such hash functions is enough to estimate the number of distinct elements within $1+\epsilon$ factor with probability at least $9/10$.

3) Say we have a sequence of number $X_1, \ldots, X_n \in \{0, \ldots, n-1\}$; also let $f_i = \sum_{j=0}^{n-1} \mathbb{I}[X_j = i]$ for all $0 \leq i \leq n-1$. Given an $\epsilon > 0$, we want to output all indices i such that $f_i \geq \epsilon n$ (with high probability). You can use memory at most $O(\frac{1}{\epsilon^2} \log^C(n))$ for any constant $C > 0$, i.e., it is ok if your algorithm uses $1000 \log^{100} n/\epsilon^2$ amount of memory. The running time of your algorithm is not limited and it can depend on n. Note that this is a streaming problem and you get the read the input only once. With probability $1-1/n$ your algorithm should

(a) output all i such that $f_i \geq \epsilon n$, and

(b) any i in the output of your algorithm should satisfy $f_i \geq \epsilon n/2$.

Hint: First use a single hash table with a test where any i with $f_i \geq \epsilon n$ passes the test with probability $9/10$ and any i where $f_i < \epsilon n/2$ fails the test with probability $9/10$. Then use the median trick (and multiple hash tables) to boost these these probabilities to $1 - 1/n^2$. Finally use a union bound.

4) In this problem you are supposed to implement the NNS algorithm for the hamming distance. You are given n points $P \subseteq \{0,1\}^d$ that you are supposed to preprocess and store based on the algorithm that we discussed in class. Then, you will be given t query points; for each query point you need to find a point at distance no more than twice the closest point.

In the input files lsh-1.in,lsh-2.in,lsh-3.in you are given n, d, t in this order. The input is followed by points of P, the $i+1$-st row of the input contains the i-th point of P. Then, the input is followed by query points (so the $n+1+i$-th row of the input has the i-th query point). In the i-th line of the output, write the index of the point P that is closest to the i-th query point. Please submit your code together with the output to Canvas.

5) **Extra Credit:** Solve problem 1 using $O(m)$ random bits with running A only $O(\log^C r)$ many times where $C > 0$ is a constant.