Homework 1

Out: *Sep 21* Due: *Sep 30*

You can collaborate with your classmates, but be sure to list your collaborators with your answer. If you get help from a published source (book, paper etc.), cite that. The answer must be written by you and you should not be looking at any other source while writing it. Also, limit your answers to one page, preferably less —you just need to give enough detail to convince the grader.

Typeset your answer in latex. If you don't know latex please write by hand clearly and scan it into pdf using your smartphone or the scanner in the mail room.

§1 The simplest model for a *random graph* consists of n vertices, and tossing a fair coin for each pair $\{i, j\}$ to decide whether this edge should be present in the graph. Call this $G(n, 1/2)$. A k-clique is a set of k vertices with an edge between every pair. What is the expected number of k cliques? What is the variance? Try to use the Chebyshev inequality to show that the number is concentrated around the expectation and give an expression for the exact decay in probability. Can you guess the size of the largest clique in a random graph from the above estimates?

§2 In class we saw a hash to estimate the size of a set. Change it to estimate frequencies. Thus there is a stream of packets each containing a *key* and you wish to maintain a data structure which allows us to give an estimate at the end of the *number of times* each key appeared in the stream. The size of the data structure should not depend upon the number of distinct keys in the stream but can depend upon the success probability, approximation error etc. Just shoot for the following kind of approximation: if a_k is the true number of times that key k appeared in the stream then your estimate should be $a_k \pm \epsilon(\sum_k a_k)$. In other words, the estimate is going to be accurate only for keys that appear frequently ("heavy hitters") in the stream. (This is useful in detecting anomalies or malicious attacks.) Hint: Think in terms of maintaining $m_1 \times m_2$ counts using as many independent hash functions, where each key updates m_2 of them.

§3 Show that given n numbers in $[0, 1]$ it is impossible to estimate the *value* of the median within say 1.1 factor with $o(n)$ samples. (Hint: to show an impossibility result you show two different sets of n numbers that have very different medians but which generate —whp—identical samples of size $o(n)$.)

Now calculate the sample size needed (as a function of t) so that the following is true: with high probability, the median of the sample has at least $n/2 - t$ numbers less than it and at least $n/2 - t$ numbers more than it.

§4 Consider the following process for matching n jobs to n processors. In each step, every job picks a processor at random. The jobs that have no contention on the processors they picked get executed, and all the other jobs *back off* and then try again. Jobs only take one round of time to execute, so in every round all the processors are available. Show that all the jobs finish executing whp after $O(\log\log n)$ steps.

§5 In the class, we saw the Karger-Stein algorithm to find a min-cut in a graph which can be implemented in $O(n^2 \log^2(n))$ time (you can assume this bound on the running time in your solutions). Modify the algorithm to find *all* min-cuts in a graph. (Hint: Run Karger-Stein procedure a few times and keep track of all the min-cuts produced.)

§6 A cut is said to be a B*-approximate min cut* if the number of edges in it is at most B times that of the minimum cut. Show that a graph has at most $(2n)^{2B}$ cuts that are B-approximate. (Hint: Run Karger's algorithm until it has $2B+1$ supernodes. What is the chance that a particular B-approximate cut is still available? How many possible cuts does this collapsed graph have?)

§7 In Matlab or another suitable programming environment implement a pairwise independent hash function and use it to map $\{100, 200, 300, ..., 100n\}$ to a set of size around n. (Use $n = 10^5$ for starters.) Report the largest bucket size you noticed. Then make up a hash function of your own design (could involve crazy stuff like taking XOR of bits, etc.) and repeat the experiment with it and report the largest bucket size. Include your code with your answer and brief description of any design decisions.

§8 (extra credit; may need selfstudy) The *chromatic number* of a graph is defined to be the smallest number of colors required to color a graph. That is, the smallest size of the set of labels such that every vertex can be assigned a label with no two adjacent vertices being assigned the same label. In the graph $G(n, 1/2)$ show that the chromatic number is about $n/2\log n$ with high probability.