New York University
Computer Science and Engineering

# CS-GY 6763: Homework 1.
Due Wednesday, Feb. 4th, 2026, 11:59pm ET.

*Collaboration is allowed on this problem set, but solutions must be written-up individually. Please list collaborators for each problem separately, or write "No Collaborators" if you worked alone.*

## Problem 1: Short(er) problems. (10 pts)

1. (2pts) Prove Chebyshev's inequality using Markov's Inequality.

2. (4pts) Consider inserting $m$ keys into a hash table of size $n = 5m^2$ using a uniformly random hash function. By the mark-and-recapture analysis from class, we know that the expected number of collisions in the table is $\frac{m\cdot(m-1)}{2\cdot 5m^2} < 1/10$. So, by Markov's inequality, with probability $> 9/10$, the table has no collisions ($< 1$ collisions). Thus, we can look up items from the table in worst-case $O(1)$ time.

   Give an alternative proof of the fact that we have no collisions with $> 9/10$ probability in a table of size $cm^2$ for some sufficiently large constant $c$. Specifically, to have no collisions, we must have the following events all happen in sequence: the second item inserted into the hash table doesn't collide with an existing item, the third item inserted doesn't collide with an existing item, ..., the $m^{\text{th}}$ item inserted doesn't collide with an existing item. Analyze the probability these events all happen. **Hint:** You might want to use the fact that $\frac{1}{2e} \leq (1 - \frac{1}{n})^n \leq \frac{1}{e}$ for any positive integer $n \geq 2$.

3. (4pts) A monkey types on a 26-letter keyboard that contains lowercase letters only. Each letter is chosen independently and uniformly at random from the alphabet. If the monkey types 100,000 letters, what is the expected number of times the sequence 'nyu' appears? Give an upper bound on the probability that the sequence 'nyu' appears at least ten times.

## Problem 2: Why does Count-Min work so well in practice? (12 pts)

We showed that Count-Min can estimate the frequency of any item in a stream of $n$ items up to additive error $\frac{1}{m}n$ using $O(m)$ space. In practice it is often observed that this bound is pessimistic: the algorithm performs better than expected. In this problem, you will establish one reason why.

For any positive integer $m$, let $f_1, \ldots, f_m$ be the frequencies of the $m$ most frequent items in our stream. Let $C = n - \sum_{i=1}^{m} f_i$. In general, we can have that $C \ll n$. For example, it has been observed that up to 95% of YouTube video views come from just 1% of videos. Prove that using $O(m)$ space, Count-Min actually returns an estimate $\tilde{f}$ to $f(v)$ for any item $v$ satisfying:

$$f(v) \leq \tilde{f} \leq f(v) + \frac{1}{m}C$$

with 9/10 probability. This is strictly better than the $\frac{1}{m}n$ error bound shown in class.

## Problem 3: Randomized methods for efficient disease identification group testing. (12 pts)

One of the most important factors in controlling diseases like bird flu or, a few years ago, COVID-19, is testing. However, testing often requires processing in a lab, so can be expensive and slow. One way to make it cheaper is to test patients/livestock/etc. in *groups*. The biological samples from multiple individuals (e.g., multiple nose swabs) are combined into a single test tube and tested for the disease all at once. If the test comes back negative, we know everyone in the group is negative. If the test comes back positive, we do not know which patients in the group actually have the disease, so further testing would be necessary. There's a trade-off here, but it turns out that, overall, group testing can save on the total number of tests run.

1. (6pts) Consider the following deterministic "two-level" testing scheme. We divide a population of $n$ individuals to be tested into $C$ groups of the same size. We then test each of these groups. For any group that comes back positive, we retest all members of the group individually. Show that there is a

choice for $C$ such that, if $k$ individuals in the population have a disease (would test positive), we can find all of those individuals with $\leq 2\sqrt{nk}$ tests. You can assume $k$ is known in advance (often it can be estimated accurately from the positive rate of prior tests). This is already an improvement on the naive $n$ tests when $k < 25\% \cdot n$.

2. (6pts) We can use randomness to do better. Consider the following scheme: Collect $q = O(\log n)$ biological samples from each individual (in reality, divide one sample into $q$ parts). Then, repeat the following process $q$ times: randomly partition our set of $n$ individuals into $C$ groups, and test each group in aggregate. Once this process is complete, report that an individual "is positive" if the group they were part of tested positive all $q$ times. Report that an individual "is negative" if *any* of the groups they were part of tested negative. Prove that for $C = O(k)$, with probability 9/10, this scheme finds all truly positive patients and reports no false positives. Thus, we only require $O(k \log n)$ tests!

## Problem 4: Try out mark-and-recapture! (12 pts)

1. (6pts) Prove the following claim Lecture 1: Specifically, if we collect $O(\sqrt{n}/\epsilon)$ samples uniformly from a set of unknown size $n$, then we can return an estimation $\tilde{n}$ which, with probability 9/10 satisfies:

$$(1-\epsilon)n \leq \tilde{n} \leq (1+\epsilon)n.$$

2. (6pts) Consider estimating the number of unique articles on Wikipedia. Wikipedia provides a way to access a random article by following the link https://en.wikipedia.org/wiki/Special:Random. You might notice that Wikipedia's random article generator does not return *truly uniform* random articles. As discussed here, Wikipedia assigns each article $i$ a random id $r_i$, which we can model as a random real number in $[0, 1]$. Then, to pick a random article, a number is sampled uniformly from $[0, 1]$ article $i$ is returned if that number lies in the range $[r_i, r_{i+1}]$. Since these intervals themselves are random, the probability distribution won't be *perfectly* uniform.

   Prove that, when the interval lengths are not perfectly uniform, the mark-and-recapture will systematically underestimate the number of articles. I.e., it will return an underestimate even as the number of samples $m \to \infty$.

## Problem 5: Distributed Importance Sampling. (15 pts)

In many machine learning training pipelines, not all data points are created equal. We often want to perform *Weighted Reservoir Sampling*: given a stream of items $x_1, x_2, \ldots$ with associated positive weights $w_1, w_2, \ldots$, we want to maintain a sample of size $k$. The goal is that, at any point in time, the probability that item $x_i$ is included in the sample is proportional to its weight relative to the other items seen so far.

A naive approach is difficult because standard reservoir sampling relies on uniform probabilities. However, in 2006, Efraimidis and Spirakis proposed a specialized algorithm suitable for distributed systems.

**The Algorithm:**

- For each item $x_i$ arriving in the stream with weight $w_i$, generate a random number $u_i$ uniformly from $(0, 1]$.
- Compute a "key" for the item: $k_i = u_i^{1/w_i}$.
- Maintain the $k$ items with the **largest** keys $k_i$ seen so far.

(a) (5pts) **The Single Item Case ($k = 1$).** Consider just two items $x_1, x_2$ with weights $w_1, w_2$. Prove that the probability that $x_1$ has the larger key (and is thus selected) is exactly $\frac{w_1}{w_1+w_2}$.

*Hint: You are comparing two random variables $K_1 = U_1^{1/w_1}$ and $K_2 = U_2^{1/w_2}$. Set up the integral $\int_0^1 \Pr[U_1^{1/w_1} > y] f_{K_2}(y) dy$ or a similar double integral.*

(b) (5pts) **Generalizing to Stream.** Using your result from part (1), argue why this method works for a stream of $n$ items for $k = 1$. Specifically, show that for any item $i$, the probability it has the maximum key is $\frac{w_i}{\sum_{j=1}^{n} w_j}$.

(c) (5pts) **Distributed Merging.** One of the massive advantages of this specific algorithm is that it is "mergeable." Suppose we have two servers, $A$ and $B$. Server $A$ observes a stream of $n_A$ items and maintains a weighted reservoir sample $S_A$ of size $k$ using the algorithm above. Server $B$ observes a distinct stream of $n_B$ items and maintains $S_B$, also of size $k$.

Describe a procedure to merge $S_A$ and $S_B$ into a single global reservoir sample $S_{global}$ of size $k$ that represents the weighted sample of the union of both streams. Prove that your merged sample has the same distribution as if a single server had processed all $n_A + n_B$ items sequentially.