1. Lecture 9, Transaction processing. Consider a Snapshot Isolation transaction processing system (Slides 12-15), designed as follows:

- The database supports MVCC, but is **unaware of transactions**. Every write to the database instantly becomes durable.
- A dedicated Transaction Oracle (TO) service allocates transaction timestamps. The Begin timestamp serves as read snapshot, whereas the Commit timestamp marks the transaction’s writes. Upon commit, the TO atomically validates the absence of WW conflicts between transactions.

a) What state must be stored at the TO, to discover the WW conflicts? Design a mechanism to reduce this state. (It is allowed to falsely abort a small fraction of transactions).

b) Design an efficient mechanism to keep the transactions’ writes invisible to the reads of concurrent transactions, and make them take effect upon commit. The mechanism must guarantee that if a transaction does not generate WW conflicts, it will become durable even if the client crashes upon commit. Describe the commit protocol in detail.

c) Complement the above mechanism, to guarantee that the read snapshot property always holds. That is, if a committed transaction T1 received a commit timestamp T1.WS, then transaction T2 that received a begin timestamp T2.RS > T1.WS will see all T1’s writes.

d) Consider that non-transactional atomic reads and writes access the database concurrently with transactions. A read retrieves the latest version of an item, whereas a write unconditionally creates a new version. Can consistency problems emerge under this mixed workload? (Either explain why no problem exists, or demonstrate a case that violates consistency).

2. Lecture 10, reservoir sampling: assume you are given a stream of elements, and a parameter R. The length of the stream is not known in advance. Elements at even positions of the stream will be called even-position elements, whereas elements at odd positions of the stream will be called odd-position elements.

a. Devise a sampling scheme that, at any moment after consuming an even number of stream elements, is able to sample a single element of the stream satisfying the following:
   i. All even-position elements have the same probability of being sampled.
   ii. All odd-position elements have the same probability of being sampled.
   iii. Any even-position element has a probability that is R times greater of being sampled than any odd-position element.

b. Devise a sampling scheme that, at any moment after consuming an even number of stream elements that is at least 2k, is able to sample k elements of the stream satisfying the following:
   i. All even-position elements have the same probability of being sampled into the k-element sample.
ii. All odd-position elements have the same probability of being sampled into the k-element sample.

iii. Any even-position element has a probability that is \( R \) times greater of being sampled than any odd-position element.

In both sub-questions try to be as efficient as possible in handling the stream and any data structures required by your sampling scheme. In particular, any stream element can only be read once.

3. Lecture 10, sliding window computations (slide 15): given a binary stream and a window of size \( N \), prove formally that any deterministic streaming (one-pass) algorithm that maintains the sum of the last \( N \) elements of the stream with absolute accuracy must use at least \( N \) bits of memory. In other words, any deterministic one-pass sliding window algorithm over binary streams that uses less than \( N \) bits of memory cannot guarantee to always return the correct sum of the last \( N \) elements.

Note: the claim is also true for randomized (non-deterministic) one-pass algorithms, but the proof is easier in the deterministic case.

4. Lecture 10, frequent item algorithm: one of the drawbacks of the Karp-Papadimitriou-Shanker algorithm as shown on slide 10 is that it may output false positives of arbitrary frequency in the stream. As an improvement, consider the following modification to the algorithm so that each false positive will have frequency greater than \( (1-\varepsilon)\theta|S| \), for a parameter \( \varepsilon \in (0,1) \) of choice. Assume, for simplicity, that \( 1/\varepsilon \theta \) is an integer.
   - First, we use a larger PF buffer of size \( 1/\varepsilon \theta \), i.e. we decrease counters once \( |PF|=1/\varepsilon \theta \) (line 8).
   - Second, we count the number of times we decrease counters, i.e. the number of times the (revised) condition on line 8 is true; denote this counter by \( C \).
   - Finally, we only output surviving elements \( e \in PF \) such that \( PF.value(e)+C>|S|\theta \) (rather than all elements who survive in PF).

Prove the following:

a. Any element whose frequency is greater than \( |S|\theta \) will be in the output of this modified algorithm.

b. Any element whose frequency \( \leq (1-\varepsilon)\theta|S| \) will not be in the output of the modified algorithm.

5. Lecture 11, Winnow algorithm: using the Winnow algorithm as a black box, devise an algorithm for learning the class of disjunctions over \( n \) Boolean variables (i.e., remove the restriction of the disjunctions being monotone). Prove an error bound.

Good Luck!