1. For each of the following example executions, determine if it is serializable, assuming each active transaction ultimately commits.
   (a) Read₁(x); Write₂(x); Write₂(y); Read₃(y); Read₃(z)
   (b) Read₁(x); Write₂(x); Write₂(y); Read₃(y); Read₃(z); Write₁(z)
   (c) Read₁(x); Write₂(x); Write₂(y); Read₃(y); Write₁(z); Read₃(z)
   (d) Read₁(x); Write₂(y); Read₁(y); Write₂(z); Read₃(z); Write₃(z); Read₁(z)

2. Try to complete the proof of the serialization graph theorem by showing the final write portion.

3. Consider the two phase locking protocol with reads and writes.
   (a) Is two phase locking deadlock-free? If so, prove it. If not, illustrate a deadlock situation. That is, you must illustrate a situation in which there is a cycle in the waits-for graph.
   (b) I stated in class that most deadlocks will involve few transactions. Using an average case model in the spirit of Tay’s performance analysis, show why this might be so.

4. In class, we discussed strict two phase locking which is two phase locking with the added constraint that all locks must be released at the end of the transaction and a transaction acquires locks immediately before it first accesses a database item. Suppose you remove these added constraints (so locks can be released earlier than the end of the transaction provided no new locks are acquired afterwards and locks may be obtained way before they are needed), giving “pure” two phase locking.

   An execution is order-preserving serializable if it is serializable and the following holds: whenever all operations of some transaction $T₁$ precede all operations of some other transaction $T₂$ (i.e., whenever $T₁$ completes before $T₂$ begins), then there exists an equivalent serial execution in which $T₁$ precedes $T₂$. Prove that two phase locking produces order-preserving serializable executions.

5. Imagine an intention locking protocol in which normal locks are acquired in a two phase manner, but intention locks can be released at any time. Would such a protocol be serializable? If so, prove it. If not, show why not.

6. Recall the Kung and Robinson optimistic concurrency control algorithm as discussed in class. How would you modify that algorithm to ensure that every transaction eventually completes (i.e., no transaction is restarted forever)?
7. Show that the datacycle algorithm ensures serializability assuming that every write on a variable is preceded by a read on that variable. You may assume that every update transaction either commits or aborts in the single time interval between two broadcasts. You may also assume that every update transaction does all its reading during the previous broadcast phase and only that phase. You may also assume that every read transaction does all its reading in a single broadcast phase. Hint: find some relation among transactions that ensures acyclicity.

8. Show that the multiversion read consistency algorithm ensures serializability. That is, read-only transactions use the multiversion technique whereas read-write transactions use strict two phase locking.

9. In the Oracle database system, transactions (including read-write ones) use multiversion read consistency for any SELECT statement, but acquire exclusive row locks for SELECT FOR UPDATE statements and hold those locks until the end of the transaction (including an end-of-file lock to prevent phantoms). Suppose some transactions in an execution issue SELECT statements followed by INSERT statements. (INSERT statements also acquire exclusive locks on the data items they insert and hold those locks until the end of the transaction.) Can this lead to a non-serializable execution? If so, show one. Otherwise, say why not.

10. (AQuery) (i) Given a schema ticks(ID, date, endofdayprice) find the maximum and minimum return of each stock where return is the ratio of the price of day i compared with the price of day i-1.

   (ii) Assuming ticks were already sorted by ID, how would you process the above query if you had complete control of the optimization process?