1. [30 points] Locking.
   
   a. [10 points] Draw the compatibility matrix for the locks proposed in the “granularity of locking” paper.
   
   *This one is exactly out of the paper or your notes.*
   
   b. [10 points] Define “write skew” and give an example. What are two degrees of consistency separated by the presence or absence of this anomaly?
   
   *The anomaly definition is:*
   
   A5B: r1[x]…r2[y]…w1[y]…w2[x]…(c1 and c2)
   
   *This can cause violations of constraints. For example, if we had the constraint x+y>0, it is possible to find reads and writes in the above anomaly that would violate the constraint (the issue is that neither t1 nor t2 “sees” the updates from the other transaction. Write skew separates snapshot isolation and degree 3 consistency.*
   
   c. [10 points] Give an example of an anomaly and a strict vs. broad interpretation of the anomaly.
   
   *One example: dirty reads:*
   
   w1[x]…r2[x]…(a1 and c2 in either order) [strict]
   w1[x]…r2[x]…((c1 or a1) and (c2 or a2) in any order [broad]

2. [10 points] Optimistic Concurrency Control
   
   a. [5 points] How is the first validation condition (T_i finishes its write phase before T_j starts its read phase) checked by the concurrent validation technique?
   
   *In the validation code, the first for loop does not even check transactions that finished before the current transaction started, so they are implicitly “passed” by the validation technique.*
b. [5 points] Optimistic concurrency control only allows serializable schedules. Is it possible when using the serial validation technique for transaction t1 to have a lower transaction number (TN) than a transaction t2, but t2 precedes t1 in the equivalent serial schedule? Why or why not?

The serial validation technique serializes validations, and transaction numbers are given in the validation phase, so if t1 gets a lower transaction number than t2 then it must have validated entirely before t2 validated. Thus it is consistent to assume that t1 preceded t2 in an equivalent serial schedule.

However, as some of you pointed out, there need not be only one equivalent serial schedule – in particular, if the read and write sets of t1 and t2 are disjoint the equivalent serial order could have them in either order.

It is not possible for t1 and t2 to “pass each other” in the serial validation phase – this can only happen in the concurrent validation approach.

3. [15 points] Two phase commit.

In the presumed-abort two-phase commit protocol, the coordinator does not force the “end” record.

a. [8 points] Why is this OK? (That is, why does it not need to be forced?)

Forcing the end record only affects whether the record is found or not when the coordinator crashes. If the coordinator crashes after writing the end record but before it is forced, then during recovery the coordinator will find the preceding commit record and will know that the transaction committed, and can just send out “commit” messages and wait for responses to guarantee that all subordinates know the decision.

b. [7 points] Discuss the impact on performance if we required the “end” record to be forced. (Consider both normal operation and recovery.)

Forcing the end record could have at least two impacts on performance. First, during “normal” operation there could be a slowdown due to the extra writes from forcing the end records. Second, during recovery, forcing an end record could actually speed things up as it guarantees that the coordinator will find an end record and it won’t have to go through the overhead of making sure that every subordinate knows that the transaction has committed (since the end record indicates that they already know.)

4. [10 points] B-link tree locking. Give a sequence of updates that could cause an error if the B-link protocol is implemented holding only one lock at a time during the “move_right()” procedure.
The basic idea is to show that with only one lock, at some point the transaction executing the “move right” holds no locks. During that time another transaction can “slip past” the first transaction and cause trouble, for example, by inserting a new node.

5. [15 points.] Parallel Database Systems: Draw a picture of the operators and their connections for a partitioned parallelism, dataflow evaluation of the query “σ(R) Join σ(S)” on two processors.

The idea here is to show the select (or scan) operators for R and S, and the join operators, running on both processors. Also, we need the split operators and merge operators to redistribute the rows to these operators.
6. ARIES [10 points]

ARIES writes CLR records during the UNDO phase, but does not write any log records during the REDO phase. Why? (Why doesn’t it write log records during the REDO, and why does it have to write them during the UNDO?)
7. DBMin [10 points]

Would DBMin be a good candidate for a buffer replacement algorithm for a general purpose file system? Why or why not?