1. Consider transactions whose operations have been recorded in the log as follows: \(<T, \text{START}>, <T, A, 10, 11>, <U, \text{START}>, <U, B, 20, 21>, <T, C, 30, 31>, <U, D, 40, 41>, <U, \text{COMMIT}>, <T, E, 50, 51>, <T, \text{COMMIT}\. Explain the recovery actions performed after a system failure, if the last log record found on the disk is

(a) \(<U, B, 20, 21>\).
(b) \(<U, \text{COMMIT}\.\)

2. Consider transactions \(T_1\) and \(T_2\), which update database elements \(A\) and \(B\) as follows:

\(T_1: \text{Read}(A,t); t:= t+2; \text{Write}(A,t); \text{Read}(B,t); t:= t*3; \text{Write}(B,t)\);

\(T_2: \text{Read}(B,s); s:= s*2; \text{Write}(B,s); \text{Read}(A,s); s:= s+3; \text{Write}(A,s)\);

Assume that actions of different transactions may interleave, but the mutual order of actions within each transaction is fixed.

(a) What are the serial schedules of the above 12 actions? Show that all of them are equivalent.
(b) Give examples of (i) a non-serial but serializable schedule and (ii) a nonserializable schedule of the 12 actions.
(c) How many serializable schedules of the 12 actions are there? (Hint: Consider possible cases where one of the transactions is the first one to read one or the both of the elements.)
(d) How many conflict-serializable schedules of the 12 actions are there?

3. Construct the precedence graph for each of the below schedules to check whether it is conflict-serializable. Show the serial schedules that are conflict-equivalent to the conflict-serializable ones.
(a) $r_1(A); r_2(A); r_3(B); w_1(A); r_2(C); r_2(B); w_2(B); w_1(C);
(b) $w_1(A); r_3(A); w_3(A); r_2(B); w_2(C); r_3(C);
(c) $r_1(A); r_2(A); r_1(B); r_2(B); r_3(A); r_4(B); w_1(A); w_2(B);

4. We saw that the two-phase locking (2PL) protocol guarantees conflict serializability of schedules. Does 2PL permit all possible conflict serializable schedules? Either prove this or give an example of a conflict serializable schedule prohibited by 2PL. (Hint: more than two transactions may participate in a schedule.)

5. Assume that the scheduler inserts a shared-lock request in front of each read operation and an exclusive-lock request in front of each write operation issued by transactions (if the transaction is not already holding a sufficient lock for the operation). Also assume that the scheduler releases the locks held by a transaction only at the end (commit/abort) of the transaction. (This protocol is known as rigorous locking.) Show how schedules (a), (b) and (c) of Exercise 5.3 would proceed under rigorous locking: Which lock requests would be delayed and when would the requested locks be granted?