1. SOLUTION:

1.1 SOLUTION :The earliest LSN number that the recovery manager reads is LSN3 i.e <START T2> since when reading backwards during the recovery it is noticed that T2 is an incomplete transaction and hence it has to be restored to its old value.

It is important to note that T1 works with database elements A and B. T2 works with only database element A. T3 works with database elements B and C.

The following is the process that occurs during the recovery:

a. It is discovered that the crash occurred during the current checkpointing.

b. When we encounter <START CKPT(T2,T3)>, we know that T2 and T3 are the incomplete transactions whose values have to be restored to their earlier values.

c. 'A' value is restored to 13 and 'B' value is restored to 15 respectively.

d. C is restored to 11 and <START T3> is encountered and hence restoring the values of T3 transaction is done.

e. Now the recovery manager proceeds to read till <START T2> is encountered since it is an incomplete transaction.

f. A is restored to 9 from transaction T2 but B is not restored to 7 from transaction T1 because <COMMIT T1> log record has already been encountered. Thus there is no need to undo the changes made by T1.

g. The earliest LSN that the recovery manager reads is LSN3 till <START T2> is encountered.

1.2 SOLUTION: The actions of the recovery manager during recovery are as follows:

a. Read from the end.

b. Restore B to 15 since T3 did not commit and is an incomplete transaction.

c. Restore A to 13 since T2 is also an incomplete transaction.

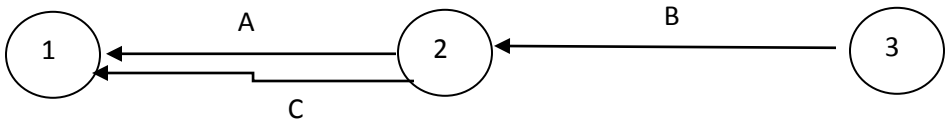
d. Restore C to 11 since T3 is an incomplete transaction.

e. Restore A to 9 since T2 is an incomplete transaction which did not commit.

f. Do not undo the changes made by T1 since it has committed.

2. SOLUTION:

a. 1. The precedence graph for the schedule is as follows:



2. Yes, the schedule is conflict serializable. The equivalent serial schedule can be obtained by swapping adjacent actions of T1, T2 and T3 in the following manner:

r1(A); r2(A); r3(B); w1(A); r2(C); r2(B); w2(B); w1(C);

r3(B); r1(A); r2(A); w1(A); r2(C); r2(B); w2(B); w1(C);

r3(B); r2(A); r1(A); w1(A); r2(C); r2(B); w2(B); w1(C);

r3(B); r2(A); r1(A); r2(C); w1(A); r2(B); w2(B); w1(C);

r3(B); r2(A); r2(C); r1(A); w1(A); r2(B); w2(B); w1(C);

r3(B); r2(A); r2(C); r1(A); r2(B); w1(A); w2(B); w1(C);

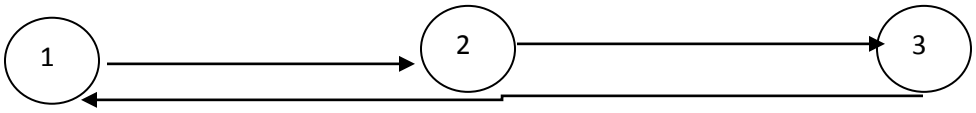
r3(B); r2(A); r2(C); r2(B); r1(A); w1(A); w2(B); w1(C);

r3(B); r2(A); r2(C); r2(B); w2(B); r1(A); w1(A); w1(C);

The final equivalent serial schedule is as follows:

r3(B); r2(A) ; r2(C); r2(B); w2(B); r1(A); w1(A); w1(C);

b. 1. The precedence graph is as follows:



2. No, the schedule conflict is not serializable since there is a loop in the graph. This means that actions of T1 precede T2 and T2 precedes T3 but then T3 precedes T1 as well, which forms a loop. Hence, there is no equivalent serial schedule.

3 SOLUTION:

There are two legal schedules of all the read and write actions of these transactions without any deadlocks. These schedules are conflict-serializable i.e. they are conflict equivalent to a serial schedule.

The serial schedule is as follows:

T1	T2

l1(A); r1(A);
A:=A+2;
w1(A);
l1(B); r1(B);
B:=B*3;
w1(B);
u1(A); u1(B);	

	l2(B); r2(B) ;
	B:=B*2;
	w2(B);
	l2(A);r2(A);
	A:=A+3;
	w2(A);
	u2(B); u2(A);

The two legal schedules are as follows:

Schedule 1:

T1	T2

l1(A); l1(B); r1(A);
A:=A+2;
w1(A);
u1(A);	

	l2(A);r2(A);
	A:=A+3;
	w2(A);
	l2(B);DENIED

r1(B);
B:=B*3;
w1(B);
u1(B);	

	l2(B); r2(B) ;
	B:=B*2;
	w2(B);
	u2(B); u2(A);

Schedule 2:

T1	T2

l1(A); l1(B);
r1(B);
B:=B*3;
w1(B);
u1(B);	

	l2(B); r2(B) ; B:=B*2; w2(B); l2(A);DENIED

r1(A);
A:=A+2;
w1(A);
u1(A);	

	l2(A);r2(A);
	A:=A+3;
	w2(A);
	u2(B); u2(A);

4 SOLUTION:

Given that there are three transactions running on the DBMS and strict 2PL concurrency policy is in effect. According to strict 2PL, all locks are released only when a transaction is completed.

The lock manager is simulated in the following table:

Here, X represents Exclusive lock, U represents Unlock, Ul represents Update lock and S represents shared lock.

Timestamp/actions	A(granted)	A(wait)	B(granted)	B(wait)
t1(T1 gets S lock on A)	S(T1)			
t2(T2 gets Ul lock on B)	S(T1)		Ul(T2)	
t3(T3 waits for T2 since no more locks permitted after Ul lock on B)	S(T1)		Ul(T2)	S(T3)
t4(T2 shares A with T1)	S(T1), S(T2)		Ul(T2)	S(T3)
t5(T3 gets Ul on A)	S(T1), S(T2),Ul(T3)		Ul(T2)	S(T3)
t6(T1 waits for T3 on A)	S(T1),S(T2),Ul(T3)		Ul(T2)	S(T3)
t7(T2 gets X lock on B)	S(T1),S(T2),Ul(T3)		X(T2)	S(T3)
t8(all the locks remain)	S(T1),S(T2),Ul(T3)		X(T2)	S(T3)
t9(T1,T3 wait on T2 for B)	S(T1),S(T2),Ul(T3)		X(T2)	S(T3), S(T1)
t10(T1 commits)	U(T1),S(T2),Ul(T3)		X(T2)	S(T3)
t11(T2 unlocks S from A and T3 gets X lock on A)	U(T2), X(T3)		X(T2)	S(T3)
t12(T3 uses U and unlocks A, T3 commits)	U(T3)(Commit)		X(T2)	
t13(T2 uses U and unlocks B, T2 commits)			U(T2)(Commit)	

REFERENCES:

1. The complete book – Ullman.
2. Lecture Slides.