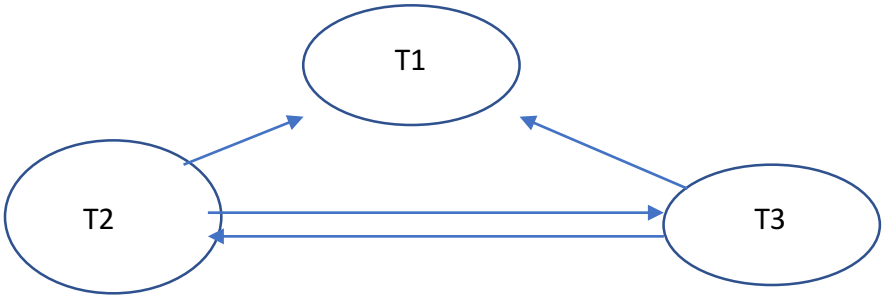
A. 1) no the precedence graph is not acyclic



2)

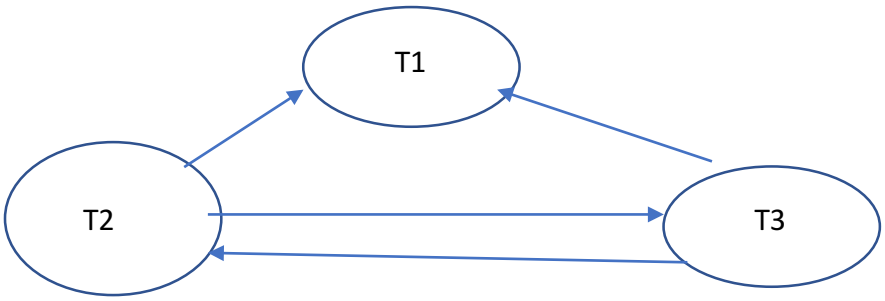
T1	T2	T3
lock-x(D)		
write D		
	lock-x(C)	
	write C	
lock-s(C) blocked		
		lock-s(C) blocked
	lock-s(A)	
	read A	
	unlock-x(C)	
	unlock-s(A)	
lock-s(C)		
read C		
		lock-s(C)
		read C
		lock-x(A)
		write A
		unlock-s(C)
		unlock-s(A)
lock-x(A)		
write A		
unlock-x(D)		
unlock-s(C)		
unlock-s(A)		

It will complete.

3) Wait-die is a deadlock prevention strategy, hence there will be no deadlocks.

T1	T2	T3
Lock-x(D)		
Write D		
	Lock-x(C)	
	Write C	
Wait for T2		
		Rollback
	Lock-s(A)	
	Read A	
	Unlock-x(C)	
	Unlock-s(A)	
Lock-s(C)		
Read C		
		Lock-s(C)
		Read C
		Lock-x(A)
		Write A
		Unlock-s(C)
		Unlock-x(A)
Lock-x(A)		
Write A		
Unlock-x(D)		
Unlock-s(C)		
Unlock-x(A)		

4) no the precedence graph is not acyclic

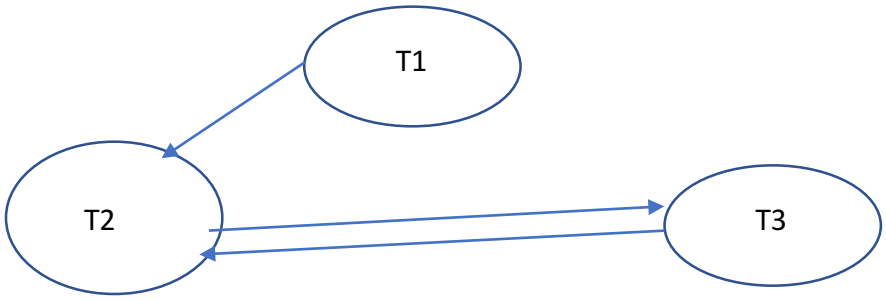


5)

T1	T2	T3
lock-x(D)		
write D		
	lock-x(C)	
	write C	
lock-s(C) blocked		
		lock-x(A)
		lock-s(C) blocked
	lock-s(A) blocked	

Here we have a deadlock, T2 is waiting for T3 to release lock on A while T3 is waiting for T2 to release lock on C.

Demonstrated by cycle in wait-for graph:



6)

T1	T2	T3
Lock-x(D)		
Write D		
	Lock-x(C)	
	Write C	
Force T2 to rollback		
Lock-s(C)		
Read C		
		Lock-x(A)
		Write A
		Lock-s(C)
		Read C
		Unlock-x(A)
		Unlock-s(C)
Lock-x(A)		
Write A		
Unlock-x(D)		
Unlock-s(C)		
Unlock-x(A)		
	Lock-x(C)	
	Write C	
	Lock-s(A)	
	Read A	
	Unlock-x(C)	
	Unlock-s(A)	

No deadlock

B)
a) No it is interleaving between transactions
b) Yes, the equivalent serial schedule is:
w3(A) c3 r1(A) w1(B) c1 r2(B) w2(C) c2 r4(B) c4
c) Yes,

r1(A) is after w3(A), c1 is after c3
r2(B) is after w1(B), c2 is after c1
r4(B) is after w1(B), c4 is after c1

d) No it is not cascadeless since r1(A) is after w3(A) but c3 is after r1(A).
It can be made cascadeless by moving c3 before r1(A).