**Sample Solution 5**

Problem 1

**Schedule 1**

1. Yes, the equivalent serial schedule is T2 -> T1 -> T3 which can be achieved through swapping. Also, the dependency graph reveals no circles:

T1

T2

T3

Dependency graph for Schedule 1

1. No. T2 has read a value of ‘B’ that has been written by T2. When T2 aborts, this value must be rolled back, but T3 has already committed, so this is not possible.
2. No. Both T1 and T3 have read A value written by T2. Failure of T2 will cause both to be rolled back (aka cascading rollback)
3. No. T3 would not be able to obtain a lock for reading the value of ‘C’, because T1 has already acquired a lock (Exclusive) on T1 and T1 cannot release any locks yet since it needs to acquire addition locks later.
4. No. The schedule is not two-phase (aka it is not cascadeless)
5. No. Same reason

**Schedule 2**

1. Yes. The equivalent serial schedule is T2 -> T1. The dependency graph does not contain cycles:

T2

T1

1. Yes. Neither of the transaction reads a value written previously by the other one.
2. Yes. Same reason as(b)
3. No. T1 cannot acquire an exclusive lock on ‘A’ due to T2 already having a shared lock on ‘A’, which it cannot release yet since it needs to acquire additional locks laer.
4. No. Due to (d)
5. No. Due to (d)

**Schedule 3**

1. No. The dependency graph reveals a cycle:

T1

T2

T3

1. Yes, only T3 reads value of ‘A’ written by another transaction T1 and T1 commits before T3
2. Yes. If we assume that the R(A) of T3 happens after the commit of T2. No, otherwise
3. No. T1 cannot acquire an exclusive lock on ‘A’ due to T2’s shared lock on ‘A’
4. No. Due to (d)
5. No. Due to (d)

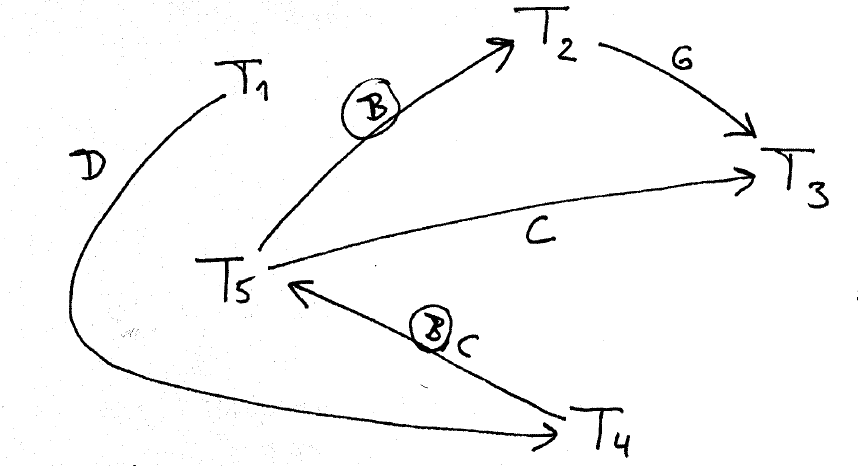
Problem 2

**Note**: In the following we assume the use of 2PL. This means that nobody can release any locks until they have acquired the last lock. The analysis also holds for strict and rigorous 2PL.

Wait-for graph:

For each wait-for edge, we show the object that is involved. For example, T1 waits for T4 on D.

**Note**: B is circled because these wait-for relationships would not exist if T2 and T5 release their locks immediately after the last operation on the schedule shown.



These is no cycle, and no deadlock. Transaction could finish in order T3, T2, T5, T4, T1.