Problem 1: Serializability

- Schedule S1:
  - Schedule S1 is not conflict-serializable since the dependency graph has a cycle.
  - Schedule S1 is not view serializable.

  ![Dependency Graph S1]

- Schedule S2:
  - Schedule S2 is conflict-serializable since the dependency graph is acyclic. The order T2-T3-T1 is the only equivalent serial order.
  - Schedule S2 is trivially view serializable since it is conflict-serializable. The only serial order allowed is still T2-T3-T1.

  ![Dependency Graph S2]

Problem 2: Two-phase locking

1. This will be permitted by 2PL, but not by Strict 2PL. This is because when T1 tries to request an exclusive lock to write Y, T2 has not committed yet, and therefore has not released its exclusive lock on Y. It will be permitted by 2PL because then T2 can release the lock on Y immediately after it acquires a shared lock for X (prior to reading X). Then T1 will be able to acquire its exclusive lock on Y.
2. This will be permitted by 2Pl, but not by strict 2PL. It will not be permitted by strict 2PL because when T2 tries acquire an x-lock prior to writing Y, it will not be able to get this lock since T3 has a shared lock on Y. Since T2 makes the request before T3 commits, T2 will not be able to get the lock. However, in 2PL, after reading Z, T3 can immediately acquire an x-lock for x and release its lock for Y. So the given schedule will follow 2PL.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(X)</td>
<td>R(Y)</td>
<td>R(Y)</td>
</tr>
<tr>
<td>W(Y)</td>
<td>R(Z)</td>
<td>R(Z)</td>
</tr>
<tr>
<td>W(Y)</td>
<td>W(X)</td>
<td>W(X)</td>
</tr>
</tbody>
</table>

**Problem 3:**

- Suppose T1 follows uncommitted read and T2 follows strict 2PL. This would result in both read-write conflicts and write-read conflicts. Suppose T1 reads X. Since T1 does not request a s-lock on X, T2 will be able to request and get a x-lock on X and then change the value of X while T1 is still in progress. Alternatively, suppose T2 writes X and then T1 tries to read X. T1 will be able to read X before T2 has committed since T1 does not request any locks. This would result in a write-read conflict. A write-write conflict would not occur both transations use x-locks for writes and hold them until the commit.

- Suppose T1 follows committed read and T2 follows strict 2PL. This would only result in read-write conflicts. Suppose T1 reads X. It releases its lock immediately after the read. Then T2 could write X before T1 is completed resulting in a read-write conflict. A write-read conflict will not occur. This is because if T2 writes X, it
will get an x-lock which will not be released until it commits, so T1 will not be able to read X until T2 has committed. A write-write conflict will not occur because both transactions are still requesting the standard x-locks for writes.

- Suppose T1 has short term S and X locks, and T2 follows strict 2PL. In this case we can have write-write conflicts. Because both transactions can write over what the other has done since locks are released immediately after a write is done. This can lead to inconsistent data (we will also have RW and WR conflicts). However, deadlock is impossible because locks are released immediately after reads and writes are completed.

**Problem 4: Hierarchical locking**

- IS[D], IS[E], IS[E600], S[E600:4], U[E600:4], U[E600], U[E], U[D]
- IS[D], IS[S], S[S20]...S[S25], IS[S26], S[S26:0]...S[S26:20], U[S26:20]...U[S26:0], U[S26]...U[S20], U[S], U[D]
- IS[D], S[S], U[S], U[D]
- IS[D], IS[E], S[E10]...S[E80], U[E80]...U[E10], U[E], U[D]
- IS[D], IS[S], S[S10]...S[S80], U[S80]...U[S10], U[S], U[D]
- SIX[D], SIX[S], X[10 pages], U[10 pages], U[S], U[D]
- IX[D], IX[S], IX[S10], X[S10:40], U[S10:40], U[S10], U[S], U[D]
- IX[D], X[E], U[E], U[D]

**Problem 5: Deadlocks**

**Problem 5.1: Deadlock Detection**

T1:S(A), T1:R(A), T2:Blocks[X(A)], T3:X(B), T3:W(B), T1:Blocks[X(B)], T3:Commits[Rel(B)], T1:X(B), T1:W(B), T1:Commit[Rel(A),Rel(B)], T2:X(A), T2:W(A), T2:X(B), T2:W(B), T2:Commit[Rel(A), Rel(B)]

No deadlock occurs.

Note: Depending on the assumptions that you make, you may also get deadlock. Full marks were given for both the answers.

**Problem 5.2: Deadlock Prevention**

**Wound-wait:** Schedules 1, 5 and 6 are possible. **Wait-die:** Schedules 4 and 7 are possible.
Problem 6: Z-ordering