Problem 1

1. a) We create the precedence graphs first and then we check for serializability.

   a. Now, there is a cycle (T1,T2,T4) so the schedule is not conflict serializable.

   b. The schedule is conflict serializable since there is no cycle in the graph. An equivalent serial schedule is T1, T2, T3, T4.
2. a. An application of the 2PL protocol on the schedule is the following. We request locks before each operation. Notice that some transactions may have to block before they get the lock. Thus the order of the operations will change.

\[ S1(A);R1(A);S2(B);R2(B);S3(C);R3(C);X1(B);X2(C);X3(D);W3(D);C3; \]
\[ U3(C);U3(D);W2(C);C2;U2(B);U2(C);W1(B);C1;U1(A);U1(B) \]

b. If we apply the 2PL protocol where before each read we request a shared lock and before a write we request an exclusive lock, then we will end up in a deadlock:

\[ S1(A);R1(A);S2(B);R2(B);S3(C);R3(C);S1(B);R1(B);S2(C);R2(C); \]
\[ S3(A);R3(A);X1(A);X2(B);X3(C); \] ! deadlock.

In that case we have to abort one of the transactions in order to break the deadlock. We pick T3 (you can pick any other one). Then T3 aborts and releases its locks. This will allow the other two transactions to continue. After the execution of the other two transactions we restart T3.

So after the abort we have:

\[ W1(A);C1;U1(A);U1(B);W2(B);C2;U2(B);U2(C);W3(B);C3;U3(C);U3(B); \]

Problem 2.

We apply the ARIES algorithm. We need to reconstruct the transaction table and the dirty pages table at the time of the crash and then we apply the redo and the undo phases.

1. Analysis phase: We start at the last checkpoint and we recover the two tables TT and DPT at the start of the checkpoint. Since this is the start of the system the two tables are empty.

Then, we go and we read all the log records from the checkpoint until the crash. Each time we check if we need to update the TT or/and the DPT table. At the end of this phase we have the following tables:

<table>
<thead>
<tr>
<th>Xact</th>
<th>lastLSN</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>90</td>
<td>R</td>
</tr>
<tr>
<td>T3</td>
<td>70</td>
<td>R</td>
</tr>
</tbody>
</table>
It is clear that we need to undo (roll back) transactions T2 and T3 from the TT table.

2. During the Redo phase we start from the smallest recLSN in DPT and we perform all the operations again one by one. Therefore, we need to perform the operations on pages P1, P2, P3, and P4 (unless the changes have been saved and we do not need to do them again). We start from LSN 20 and move forward doing the Redo operations.

When we redo an action on a page, we need to update the pageLSN for this page. However, we do not need to write anything on the log.

3. In the Undo phase, first we create toUndo list. In our case, toUndo = { 70,90 }.

We first undo the log record with LSN = 90. We write a CLR record for undoing operation in LSN 90. The toUndoNextLSN is 40 and we update toUndo = {40,70}.

Next, we undo the log record with LSN = 70. We write a CLR record for undoing operation in LSN 70. The toUndoNextLSN is 50 and we update toUndo = {40,50}.

Next, we undo LSN =50, write a CLR record for this with toUndoNextLSN = null, and since this is the last operation of T3 to undo we write a new record T3 End and we remove T3 from TT. We have toUndo={40}.

Next, we undo LSN =40, write a CLR record for this with toUndoNextLSN = null, and since this is the last operation of T2 to undo we write a new record T2 End and we remove T2 from TT. Now, we finish and we are ready to restart.

4. The log after the crash is:

```
100 | CLR: Undo T2 LSN 90: UndoNextLSN = 40
110 | CLR: Undo T3 LSN 70: UndoNextLSN = 50
120 | CLR: Undo T3 LSN 50: UndoNextLSN = null
130 | T3 End
140 | CLR: Undo T2 LSN 40: UndoNextLSN = null
150 | T2 End
```