As usual, provide enough detail to support your answers. Make sure to answer all the questions. Also, if you are planning to submit your assignment handwritten, please write legibly. We encourage you to itemize your answers.

The following are some overall comments for each question:

**Question 1**

- Some of you missed to include examples for each of the anomalies.

**Question 2**

- Optimistic Concurrency Control is not a locking protocol. Contrary, Pessimistic CC is based on locking.
- Optimistic does better when there is a mix of small and large transaction. Overhead of checking (whether many conflicts or less is the same and can be emulated by using locks or creating conflict graphs). The validating transaction must check against all committed transaction while it was executing.
- Optimistic does better with low conflicts, since overhead because of rollback is minimum.
- Pessimistic perform better with high degree of conflicts.
- See more details in 21.4.1 in Textbook.
- See more details in following paper (focus on 2-2.3, 3.1, 3.2, 4.1, 4.2):
Question 3

The schedule is not serial, operations from each transaction are interleaved. The precedence graph is:

```
  T1 --> r3(A), w1(A) --> r1(C), w3(C) --> T3
   |         |           |           |
   v         v           v           v
  T2       r3(A), w2(A) r2(C), w3(C)
```

The schedule is not conflict serializable because there exist cycles in the precedence graph. If a schedule’s precedence graph is cyclical, the schedule cannot be conflict serializable.

Question 4

- Just the schedule 3 can lead to deadlocks in 2PL. However, this can only happen if you skip operations. Since we posted in Piazza that you did not need to skip operation while evaluating 2PL, the deadlock was not possible.
- Sample solution:

**Schedule 1:** \(R_1(X), W_4(X), R_2(X), W_2(X), R_3(Y), W_4(Y), C_1, R_2(Y), W_3(Y), C_2\)

  a) Yes, the schedule is conflict serializable and equivalent to a serial schedule with the following order: T1, T2.
  b) No, the schedule does not avoid cascading aborts. T2 is reading an uncommit data (x).
  c) No, is not possible under 2PL because T2 cannot acquire a shared lock on X.

**Schedule 3:** \(W_1(A), W_2(B), W_3(C), R_1(X), R_2(X), R_1(Y), W_1(X), C_1, W_2(Y), C_2, W_3(Y), C_3\)

  a) No, the schedule is not conflict serializable. Conflicting operation are shown above in colors. Here’s the precedence graph:
b) Yes, the schedule avoids cascading aborts. There are no reads to uncommitted information.

c) No, it is not possible under 2PL because T1 cannot acquire the lock in W1(x). If you skip operations, 2PL can lead to a deadlock. Here’s the wait-for graph.