1. Given the following 2 different transactions. List all potential schedules for T1 and T2 and determine which schedules are conflict serializable and which are not. Assume at each time interval a command is issued by one of the transactions. Remember a schedule will vary from another schedule if by chance a transaction accesses a data object in a different order than the other schedule. Also note, different schedules may generate the same precedence graph.

Transaction 1	Transaction 2
Start transaction	Start transaction
READ(X)	READ(X)
X = X -N	X = X +M
WRITE(X)	WRITE(X)
Commit	Commit

Ans. We have 3 operations from T1 and 3 from T2, making 6 total operations to be interleaved while preserving each transaction's internal order. The number of valid interleavings is = 20.

There are a total of 20 schedules possible
Here are the total possible listed below:
Conflict Serialzable
R1(X), W1(X), C1, R2(X), W2(X), C2
R2(X), W2(X), C2, R1(X), W1(X), C1
R1(X), W1(X), R2(X), C1, W2(X), C2
R2(X), W2(X), R1(X), C2, W1(X), C1
R1(X), R2(X), W1(X), C1, W2(X), C2
R2(X), R1(X), W2(X), C2, W1(X), C1

Non - Serilizable Schedules
R2(X), R1(X), W2(X), W1(X), C2, C1
R1(X), W1(X), R2(X), W2(X), C1, C2
R2(X), W2(X), R1(X), W1(X), C2, C1
R1(X), R2(X), W2(X), W1(X), C2, C1
R2(X), R1(X), W1(X), W2(X), C1, C2
R1(X), R2(X), C1, W1(X), W2(X), C2
R2(X), R1(X), C2, W2(X), W1(X), C1
R1(X), R2(X), C1, W2(X), W1(X), C2
R2(X), R1(X), C2, W1(X), W2(X), C1
R1(X), W1(X), R2(X), C2, W2(X), C1
R2(X), W2(X), R1(X), C1, W1(X), C2
R1(X), R2(X), W1(X), C2, W2(X), C1
R2(X), R1(X), W2(X), C1, W1(X), C2

2. Consider the following schedules consisting of different transactions. For each schedule, identify if the schedule is conflict serializable, recoverable, and if it is cascade recoverable. Create a 3 by 3 table for your submission, where the columns are the schedule classifications and the schedules are the rows. Also for each of the properties, specify a brief reason for your answer. For Example : Schedule is Conflict Serializable because the precedence graph has no cycles. (21 POINTS)

Schedule 1	Transaction 1	Transaction 2
t0	Start transaction	Start transaction
t1		Write(X)
t2	Read(X)	
t3	X = X + 5	
t4	Write(X)	
t5		Rollback

Schedule 2	Transaction 1	Transaction 2	Transaction 3
t0	Start transaction		
t1	Read(X)		
t2		Start transaction	
t3		Read(X)	
t4			Start transaction
t5			Write(X)
t6		Read(X)	
t8	Read(Y)		
t9	Commit		
t10		Commit	
t11			Commit

Schedule3	T1	T2	T3
t0	Start transaction		
t1	Read(X)		
t2	Read(Y)		
t3	Write(X)	Start transaction	
t4		Read(Y)	Start transaction
t5			Write(Y)
t6	Write(X)		
t8		Read(Y)	
t9		commit	
t10			commit

	Conflict Serializable?	Recoverable?	Cascade Recoverable?
Schedule 1	**Yes** (No cycle in precedence graph; only T2 -> T1)	**Yes** (T1 doesn't commit before T2 aborts)	**No** (T1 reads X from T2 before T2 commits, i.e. dirty read, so not cascade-free)
Schedule 2	**Yes** (No cycle; T3 -> T1 is the only edge)	**No** (T1 commits before T3, but T1 read T3's uncommitted write)	**No** (Not even recoverable, so cannot be cascade recoverable)
Schedule 3	**Yes** (No cycle; T3 -> T2 is the only edge)	**Yes** (As long as T2 commits after T3, T2 doesn't commit before the writer)	**No** (T2 performs a dirty read from T3, so it is not cascade-free even though commit order makes it recoverable)

Explanation:
1. Conflict Serializability:

Determined by whether the precedence graph has a cycle.
All three schedules have only a single directed edge in their graph (e.g., T2 -> T1,T3 -> T1, or T3 -> T2), so none contain a cycle.
Therefore, all three are conflict-serializable.

2. Recoverability

A schedule is recoverable if no transaction commits before another transaction whose uncommitted data it used.
In Schedule 1, T1 never commits before T2 (which eventually aborts), so it's recoverable.
In Schedule 2, T1 commits first but had read T3's uncommitted write, making it non-recoverable.
In Schedule 3, T2 reads T3's uncommitted data but does not commit before T3 if we assume the correct commit order, so it is recoverable.

3. Cascade Recoverability

A schedule is cascadeless or cascade recoverable if no transaction ever reads data from another uncommitted transaction (i.e. no dirty reads).
Schedule 1 and 3 have a dirty read scenario, so No.
Schedule 2 also has T1 reading from uncommitted T3, and is not even recoverable, so also No.

3. Consider the following lock table and perform the following tasks: (15 POINTS)

Transaction	Items locked	Items Waiting on
T1	A	C
T2	C	A
T3	D	B
T4	B	
T5	E	A, D

a. Create a waits-for graph for the lock table. (5 points)
b. Briefly explain the scenario of a deadlock. Identify if there is a deadlock in the system. If so, which transactions are involved. (5 points)
c. Suggest any one deadlock resolution strategy to break the deadlock. (5 points)

a. T1 is waiting on C, which is locked by T2.
   i. Edge: T1 -> T2
b. T2 is waiting on A, which is locked by T1.
   i. Edge: T2 -> T1
c. T3 is waiting on B, which is locked by T4.
   i. Edge: T3 -> T4
d. T4 is not waiting on anything (no outgoing edges).
e. T5 is waiting on A (held by T1) and D (held by T3).
   i. So edges: T5 -> T1 and T5 -> T3.

Hence the Waits-For Graph edges are:

- T1 <-> T2 (a cycle, T1 -> T2 and T2 -> T1)

- T3 -> T4
- T5 -> T1 and T5 -> T3

B. Yes. T1 and T2 form a circular wait: T1 wants C from T2, T2 wants A from T1. So T1 <-> T2 is a deadlock cycle.

C. Wait-Die: If an older transaction must wait for a younger transaction, it waits otherwise it aborts the younger transaction.

Wound Wait: If a younger transaction blocks an older, the older “wounds” the younger (forces it to abort). Otherwise the older waits.

Deadlock Detection with a Waits-For Graph and subsequent rollback of a transaction in the cycle.

4. Apply the basic timestamping algorithm to the following schedule using the values of the read and write timestamps for the accessed data objects V, X, Y, and Z. Determine the read and write timestamps for the objects after the schedule is run. Assume that the transaction id issued to the latest transaction is 20 (transaction 20 is the latest transaction before transactions A, B, AND C are started). Determine if the schedule can be performed as is or if a transaction(s) will need to be restarted. If so, what transactions will need to be restarted given the basic timestamping ordering algorithm ? After each operation, also print the objects and their Read and Write timestamps (5 * 3 table as below) (20 POINTS)

Objects and their timestamps

Object	Read timestamp	Write timestamp
V	12	13
X	15	12
Y	17	13
Z	14	14

Transactions to complete and the timing between the operations

TIME	Transaction A	Transaction B	Transaction C
11		READ(Z)	
12		READ(Y)	
13		WRITE(Y)	
14			READ(Y)
15			READ(Z)

16	READ(X)		
17	WRITE(X)		
18			WRITE(Y)
19			WRITE(Z)
20	READ(V)		
21	READ(Y)		
22	WRITE(Y)		
23		WRITE(V)	

Time	Operation	Condition Check	Updated Timestamps	Result
11	B: READ(Z)	TS(B)=22 >= WTS(Z)=14 ? Yes	R-TS(Z)=max(14,22)=22	Allowed
12	B: READ(Y)	TS(B)=22 >= WTS(Y)=13 ? Yes	R-TS(Y)=max(17,22)=22	Allowed
13	B: WRITE(Y)	TS(B)=22 >= R-TS(Y)=22 and TS(B)=22 >= WTS(Y)=13 ? Yes	W-TS(Y)=22	Allowed
14	C: READ(Y)	TS(C)=23 >= WTS(Y)=22 ? Yes	R-TS(Y)=max(22,23)=23	Allowed
15	C: READ(Z)	TS(C)=23 >= WTS(Z)=14 ? Yes	R-TS(Z)=max(22,23)=23	Allowed
16	A: READ(X)	TS(A)=21 >= WTS(X)=12 ? Yes	R-TS(X)=max(15,21)=21	Allowed
17	A: WRITE(X)	TS(A)=21 >= R-TS(X)=21 and TS(A)=21 >= WTS(X)=12 ? Yes	W-TS(X)=21	Allowed
19	C: WRITE(Y)	TS(C)=23 >= R-TS(Y)=23 and TS(C)=23 >= WTS(Y)=22 ? Yes	W-TS(Y)=23	Allowed
20	A: READ(V)	TS(A)=21 >= WTS(V)=13 ? Yes	R-TS(V)=max(12,21)=21	Allowed
21	A: READ(Y)	TS(A)=21 >= WTS(Y)=23 ? No (21 < 23)	No update	Not Allowed

At Time=21, Transaction A attempts to read Y, but TS(A)=21<WTS(Y)=23. Under basic timestamp ordering, a previous transaction cannot read data already written by a younger transaction. A must be rolled back and restarted with a new, larger timestamp.

Final Timestamp Before Conflict

Object	R-TS	W-TS
V	21	13
W	21	21
Y	23	23
Z	23	14

Y now has R-TS=23 and W-TS=23 set by Transaction C TS = 23.

A has TS=21 which is less than the W-TS(Y)=23, so A's read fails.

Transaction B (TS=22) and Transaction C (TS=23) continue to completion without needing to restart. A restarts.

5. Consider the following schedule involving transactions **T1**, **T2**, and **T3**: **(11 POINTS)**

Time	**Transaction T1**	**Transaction T2**	**Transaction T3**
t1	Read_lock(T1,A)		
t2	Read(A)		
t3		Write_lock(T2, B)	
t4		Write(B)	
t5	Write_lock(T1, C)		
t6	Write(C)		
t7			Read_Lock(T3, B)

t8			Read(B)
t9	Commit		
t10		Unlock(T2, B)	
t11		Read_Lock(T2, C)	
t12		Read(C)	
t13		Commit	
t14			Write_Lock(T3, D)
t15			Write(D)
t16			Commit

a. Analyze the schedule and determine if each transaction follows the Two-Phase Locking (2PL) protocol. Justify your answer. **(5 POINTS)**
   i. T1: Locks A, then C, releases at commit. This is 2PL (rigourous).
   ii. T2: Releases B at t10, then acquires lock on C at t11. This violates 2PL.
   iii. T3: Locks B, then D, releases only at commit. This is strict 2PL.

b. Classify each transaction by their variation of 2PL protocol. **(3 POINTS)**
   i. T1: Strict (also rigorous).
   ii. T2: Not 2PL.
   iii. T3: Strict (rigorous).

c. Modify the transactions to ensure each transaction follows Rigorous 2PL without changing the final outcome of the schedule. **(2 POINTS)**

   i. Remove Unlock(B) at t10. T2 must hold all locks until commit t13. Then T2 follows strict 2PL.

6. Given the schedule below, classify the schedule as recoverable or not recoverable. If classified as recoverable, then specify if it is strict-recoverable, cascadeless, cascade recoverable. Please support your answer with a valid description. **(6 POINTS)**

Time	Transaction 1	Transaction 2	Transaction 3
t_1	Start transaction		
t_2	Read(X)		Start transaction
t_3			Read(X)
t_4	Write(X)		
t_5	Commit		
t_6		Start transaction	Commit
t_7		Read(X)	
t_8		Commit	

(a) Is the schedule recoverable? Yes, It is recoverable because no transaction commits before another transaction from which it has read uncommitted data.

(b) Level (Strict/Cascadeless/etc.) - It is not strict or cascadeless. Instead, it is cascade recoverable, some transaction does read uncommitted data, but it commits only after the writer commits.

(c) Justification: A transaction T2 reads uncommitted data from T1, but T2 commits strictly after T1. Therefore, even though T2 performed a dirty read, the commit order still ensures recoverability. However, because there was a dirty read, the schedule is not cascadeless or strict. Hence it is cascade recoverable only.

7. Define a cascading rollback. Provide an example of a schedule with a cascade rollback. **(7 POINTS)**

A cascading rollback occurs when one transaction (say T2) has read some item X that was written by another transaction (T1) before T1 committed. If T1 then aborts, any transaction that read T1's uncommitted data must also abort. This can propagate if subsequent transactions read from T2, etc.

Example:
T1 :W(X) (uncommitted write on X)

T2: R(X) (reads X from T1 before T1 commits)

T1: abort -> T2 must rollback (cascade).

Here, T2 is forced to roll back because it depended on T1 uncommitted write. If T2 had passed data to another transaction T3, then T3 would need to roll back as well, further extending the cascade.