Problem 1:

Solution provided.

Output:

```
cd "/Users/nabid/Documents/MS TUK/Winter_19-20/Database
System/Exercise/Exercise 10/exec" ;
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/ja
va -
agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:
54566 -Dfile.encoding=UTF-8 -cp "/Users/nabid/Library/Application
Support/Code/User/workspaceStorage/6ff3f04169006a1c4a445d34afb09355/red
hat.java/jdt_ws/exec_67744d7c/bin" Recovery
--------- Test 1 ---------
--- Input log: ---
[#1, T1, BOT, , , ]
[#2, T2, BOT, , , ]
[#3, T1, A, A-=50, A+=50, #1]
[#4, T2, C, C+=100, C-=100, #2]
[#5, T1, B, B+=50, B-=50, #3]
[#6, T1, COMMIT, , , #5]
[#7, T2, A, A-=100, A+=100, #4]
--- Tests: ---
Loser Transactions: [T2]
Expected Loser Transactions: [T2]
Test 1 successful
```

Question 2:

a) Rollback is performed on T2 due to ABORT as followed:
   - Log entries that belong to this transaction are processed in reverse order.
   - This can be done using the log buffer, since we can see here that the main memory is still intact (Abort happened before crash).
   - Using PrevLSN we can traverse backward and execute undo operations.
   - Before the execution of the undo we need to, write a log entry using CLRs.

b)

<#7'	T2	PA	U(A2)	-	#7	#5>
<#5'	T2	PC	U(C)	-	#7'	#2>
<#2'	T2	BOT	-	-	#5'	0>

Time	Operation	DB Buffer		DB Entry		Log Entry in Log Buffer							Log File
		(Page,	LSN)	(Page,	LSN)	[LSN,	TA,	PageID,	Redo,	Undo,	PrevLSN,	UndoNxtLSN]	LSNs
10	b_1					#1	T1	BOT	-	-	-	-	
20	b_2					#2	T2	BOT	-	-	-	-	
30	$w_1(B)$	PB	#3			#3	T1	PB	R(B)	U(B)	#1	-	
40	$w_1(A_1)$	PA	#4			#4	T1	PA	R(A1)	U(A1)	#3	-	
50	$w_2(C)$	PC	#5			#5	T2	PC	R(C)	U(C)	#2	-	
60	c_1					#6	T1	COMMIT	-	-	#4	-	#1, #3, #4, #6
70	$w_2(A_2)$	PA	#7			#7	T2	PA	R(A2)	U(A2)	#5	-	
75	flush(P_A)			PA	#8								
80	a_2					#9	T2	ABORT	-	-	#7	-	
81						<#7'	T2	PA	U(A2)	-	#7	#5>	
82						<#5'	T2	PC	U(C)	-	#7'	#2>	
85	r_2					<#2'	T2	BOT	-	-	#5'	0>	
100	b_3					#13	T3	BOT	-	-	-	-	
110	$w_3(B)$	PB	#14			#14	T3	P(B)	R(B)	U(B)	#13	-	
120	flush(P_A)			PA	#15								
						Crash							

T1

Problem 1:

Solution provided.

Output:

```
cd "/Users/nabid/Documents/MS TUK/Winter_19-20/Database
System/Exercise/Exercise 10/exec" ;
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/ja
va -
agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:
54566 -Dfile.encoding=UTF-8 -cp "/Users/nabid/Library/Application
Support/Code/User/workspaceStorage/6ff3f04169006a1c4a445d34afb09355/red
hat.java/jdt_ws/exec_67744d7c/bin" Recovery
--------- Test 1 ---------
--- Input log: ---
[#1, T1, BOT, , , ]
[#2, T2, BOT, , , ]
[#3, T1, A, A-=50, A+=50, #1]
[#4, T2, C, C+=100, C-=100, #2]
[#5, T1, B, B+=50, B-=50, #3]
[#6, T1, COMMIT, , , #5]
[#7, T2, A, A-=100, A+=100, #4]
--- Tests: ---
Loser Transactions: [T2]
Expected Loser Transactions: [T2]
Test 1 successful
```

Question 2:

a) Rollback is performed on T2 due to ABORT as followed:
   - Log entries that belong to this transaction are processed in reverse order.
   - This can be done using the log buffer, since we can see here that the main memory is still intact (Abort happened before crash).
   - Using PrevLSN we can traverse backward and execute undo operations.
   - Before the execution of the undo we need to, write a log entry using CLRs.

b)

<#7'	T2	PA	U(A2)	-	#7	#5>
<#5'	T2	PC	U(C)	-	#7'	#2>
<#2'	T2	BOT	-	-	#5'	0>

Sheet-10

Problem-3)

(a) Suppose, there are two transactions T_1 and T_2 which are being applied to data item D_1. If T_1 updates D_1 and then aborts, and then T_2 also updates D_1, it would be a problem in situations where crash happens such that commit of T_2 is successful but abort operation of T_1 is unsuccessful. The reason behind this is crash occured between those two operations. If rollback information were not written on log, dirty read can happen, because T_2 will be redone using the value updated by T_1.

(6) Case-1: Data persisted, cash not released

For ATM machine transaction, releasing money is the most crucial part. The transaction process of ATM can be described as below:

Begin ATM Transaction
- Customer enters card
- Customer enters pin
- Customer enters withdrawal amt.
- ATM checks whether sufficient balance available in customer's account.

BOT
- Debit from customer account
- commit
- print receipt to customer

EOT

- Dispense cash to customer
- Send notification to bank

End ATM transaction

ATM machine shouldn't release money to customer before persisting data to the database and log durably. Exception like power outage can happen during the operation that may lead to situation where money has been debited from account and chages are persisted to database, but ATM fails to release money to customer. This can be addressed by crediting money back to customer's by issuing compensation transaction after reconciliation.

Case-2: Cash released, data could n't be persisted

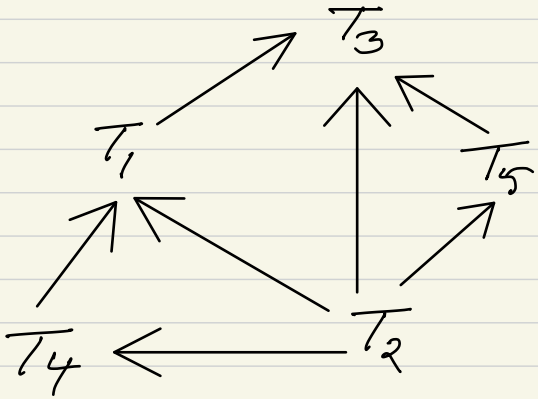
In this case, if for some reasons, if data couldn't be persisted to DB, and customer gets money, bank will loose money.

Problem-4 |

(a) $S_1 := r_1(c)\ w_2(d)\ w_2(b)\ r_4(c)\ r_5(d)\ r_3(c)\ r_5(c)\ r_4(a)\ r_4(b)\ r_1(d)\ c_5\ r_2(a)\ w_3(c)\ w_1(b)\ r_3(b)\ c_2\ r_3(c)\ w_4(a)\ c_3\ c_1\ c_4$

This is not a FSR because T_5 has not been committed. If we try to check CSR, according to the schedule conflict graph will

look as below:-



Conflict serializability states that, two operations from two transaction are in conflict schedule, if they access same data object and at least one of them is a write operation. From the operations $R_1(C) \rightarrow W_3(C)$, we can see the above rule is satisfied. So it's a CSR.

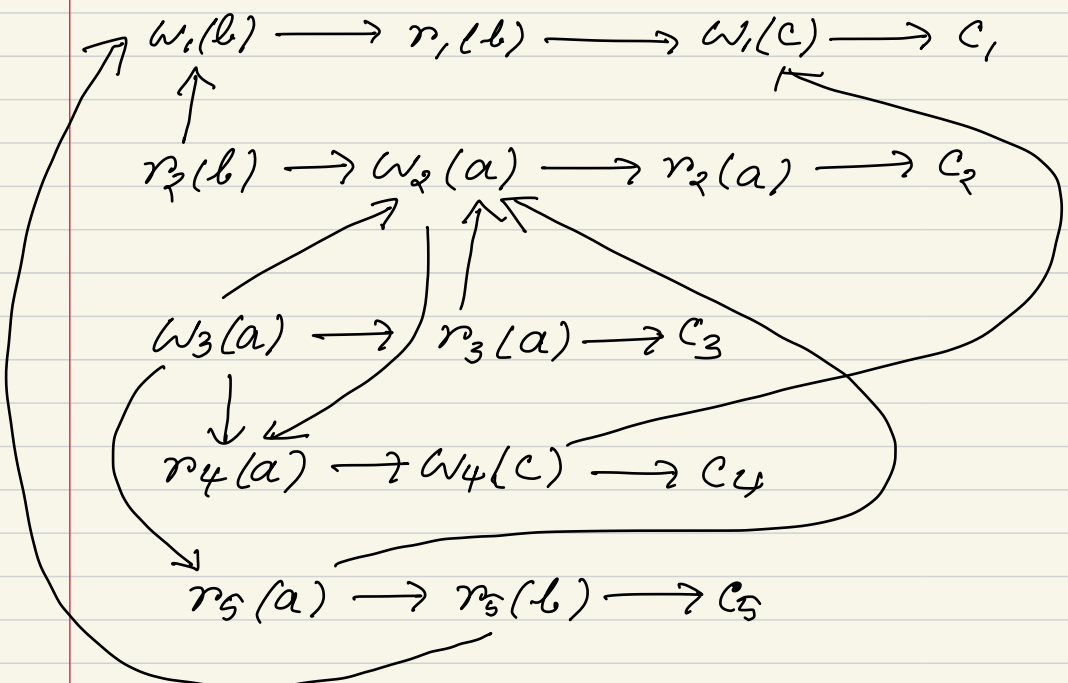
(b) If we serialize S_2 in order $t_1 t_2 t_3$ we get:-

$$S_2 = r_1(a)\, w_1(b)\, r_1(a)\, w_1(c)\, c_1\, w_2(a)\, w_2(c)\, c_2$$
$$w_3(c)\, r_3(a)\, w_3(a)\, w_3(b)\, r_3(c)\, c_3$$

which is equivalent to S_2'

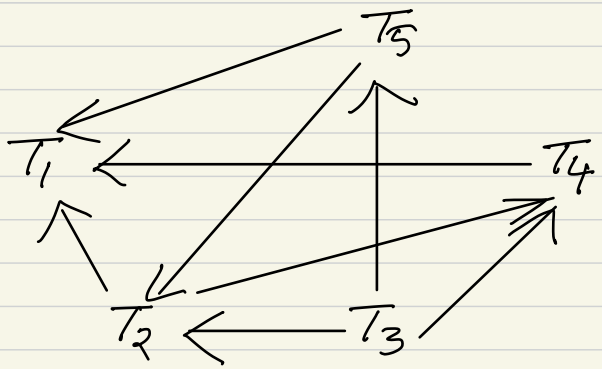
So $S_2 \approx_V S_2'$

(c) For the given schedule S_4, conflicting step graph D is below:-



and the conflict graph

$G_1 =$



As there are no cycles, it's a CSR.

If we apply commutativity rules C_1, C_2, C_3 to S_4, we get the below serial:-

$$w_3(a)\, r_3(a)\, r_5(a)\, r_5(b)\, r_2(b)\, w_2(a)$$
$$r_2(a)\, r_4(a)\, w_4(c)\, w_1(b)\, r_1(b)\, w_1(c)$$

Final schedule will be

$$T_3\, T_5\, T_2\, T_4\, T_1$$

S_4 doesn't belong to OCSR, cause in original graph, T_2 must be committed before T_5