The assignment is to be turned in before Midnight (by 11:59pm) on February 8 , 2018. You should turn in the solutions to this assignment as a pdf file through the TEACH website. The solutions should be produced using editing software programs, such as LaTeX or Word, otherwise they will not be graded.

1: Query optimization (2 points)

Consider the following relations:

Product (name, production-year, rating, company-name)
Company (name, state, employee-num)

Assume each product is produced by just one company, whose name is mentioned in the *company-name* attribute of the *Product* relation. Attributes *name* are the primary key for both relations *Product* and *Company*. Attribute *rating* shows how popular a product is and its values are between 1-5.

The following query returns the products with rating of 5 that are produced after 2000 and the states of their companies.

```
SELECT p.name, c.state
FROM Product p, Company c
WHERE p.company-name = c.name and p.production-year > 2000 and p.rating = 5
```

Suggest an optimized logical query plan for the above query.

Solution:
For the above query the logical plan would be to first select products who's rating is 5. Given that there are only five options for this value it is likely that this at least reduces the number of products by 1/5. While doing this you would only select the company name and product name for each tuple who's rating was 5. Next you would select the items from the resulting list that were made after 2000. Next you would select Company name from C. Finally you would join the results on the company name.

2: Query optimization (2 point)

For the four base relations in the following table, find the best join order according to the dynamic programming algorithm used in System-R. You should give the dynamic programming table entries for evaluate the join orders. Suppose that we are only interested in left-deep join trees and join trees without Cartesian products. Note that you must use the Selinger-style formulas to compute the size of each join output. The database system uses hash join to compute every join. Assume that there is enough main memory to perform the hash join for every pairs of relations. Each block contains at most 5 tuples of a base or joint relation. If relations are joined on multiple attributes, you can compute the selectivity factor of the full join by multiplying the selectivity factors of joining on each attribute.

R(A,B,C)	S(B,C)	W(B,D)	U(A,D)
T(R)=4000	T(S)=3000	T(W)=2000	T(U)=1000
V(R,A) =100 V(R,B) =200 V(R,C) =100	V(S,B) =100 V(S,C) = 300	V(W,B) =100 V(W,D) =50	V(U,A) =100 V(U,D) =100

Solution:
In the graphic shown below we can see that the Final cost of the Join was 378000 IO accesses and the final size of the join would be $\tilde{4}00$ tuples. To compute this I used the following formulas:

$$Cost(RS) = 3B(R) + 3B(S) = 3/5T(R) + 3/5T(S)$$

When computing something with a nested join say U(RS) I Used:

$$Cost(U(RS)) = 3B(U) + 3B(RS) = 3/5T(U) + 3/5T(RS) + Cost(RS)$$

To compute the Size i used the following formula:

$$Size(RS) = T(R) * T(S)/Max(V(S,A), V(R,A)) * Max(V(S,B), V(R,B)) * ...$$

Where we can keep going based on the number of attributes we are joining on. Shown in the graphic below are the Formulas above applied to the relations in the problem.

Join	Cost	Size	Plan
RS	4200	200	RS
RW	3600	40000	RW
RU	3000	40000	RU
SW	3000	60000	SW
WU	1800	20000	WU
RSW	41400	4000	R(SW)
RSU	28800	2000	S(RU)
RWU	16200	4000	R(WU)
SWU	15600	600000	S(WU)
RSWU	378000	400	R(SWU)

3: Concurrency control (3 points)

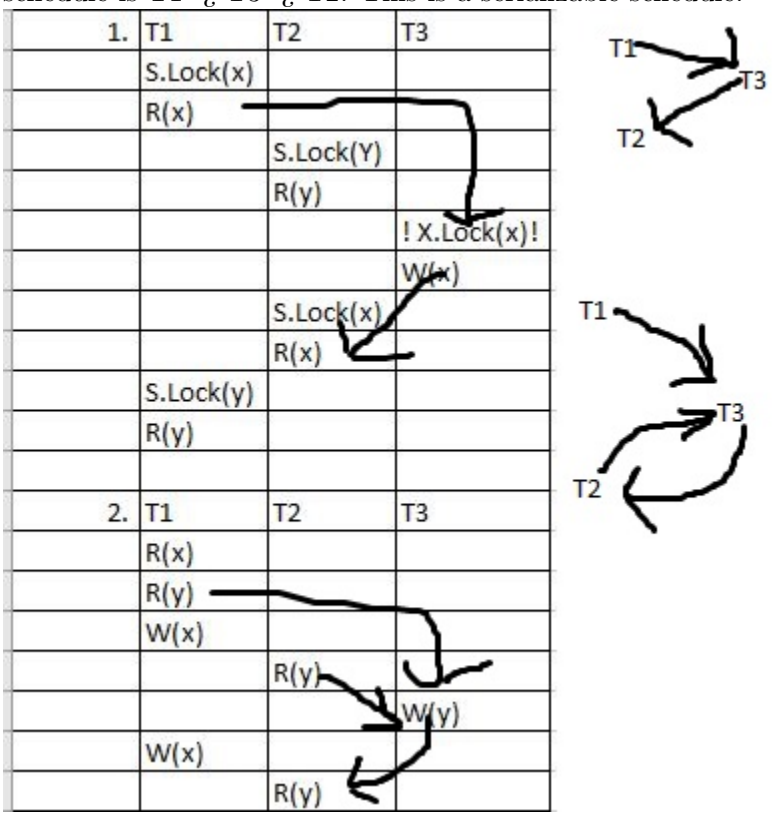
Consider the following classes of schedules: serializable and 2PL. For each of the following schedules, state which of the preceding classes it belongs to. If you cannot decide whether a schedule belongs to a certain class based on the listed actions, explain briefly your reasons.

The actions are listed in the order they are scheduled and prefixed with the transaction name. If a commit or abort is not shown, the schedule is incomplete; assume that abort or commit must follow all the listed actions.

1. T1:R(X), T2:R(Y), T3:W(X), T2:R(X), T1:R(Y)
2. T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y)

Solution:
In the graphic below you will see that I've drawn out both of these schedules in terms of when the writes and the reads happen. After doing this I drew the lines between the different operations. Finally I drew a tree representing the different transactions. If they have a cycle they are not serializable. To determine if they are 2PL I added the locks that would be needed in a 2PL implementaion and determined if the order of the operations in the transaction would be allowed. For each of these steps you will find a different graphic below.

Serliazable:
Schedule One: After creating the serialization graph you can see that no cycle is created as the schedule is T1 -¿ T3 -¿ T2. This is a serializable schedule.



Schedule Two: Showed by the graphic above you can see that this schedule is not serializable. After drawing the serialization graph you can see that a cycle is formed between T2 and T3. Specifically we can say that T2 is getting a dirty read on Y.

2PL:
Schedule One: As you can see in the graphic in section one. This graph is not PL2 because T3 will be unable to write to X as long as T1 has a shared lock on X which would not be released until after T1 reads in Y.
Schedule Two: This Schedule is not serializable and therefore isn't 2PL.