1. The cost of join algorithms (20 points)

Consider the join $R \bowtie_{R.a = S.b} S$, given the following information about the relations to be joined. The cost metric is the number of page I/Os unless otherwise noted, and the cost of writing out the result should be uniformly ignored.

- Relation $R$ contains 200,000 tuples and has 20 tuples per page.
- Relation $S$ contains 4,000,000 tuples and also has 20 tuples per page.
- Attribute $a$ of relation $R$ is the primary key for $R$.
- Each tuple of $R$ joins with exactly 20 tuples of $S$.
- 1,002 buffer pages are available.

(a) What is the cost of joining $R$ and $S$ using a block nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged?

(b) What is the cost of joining $R$ and $S$ using a sort-merge join? What is the minimum number of buffer pages required for this cost to remain unchanged?

(c) What is the cost of joining $R$ and $S$ using a hash join? What is the minimum number of buffer pages required for this cost to remain unchanged?

(d) What would be the lowest possible I/O cost for joining $R$ and $S$ using any join algorithm, and how much buffer space would be needed to achieve this cost? Explain briefly.

(e) How many tuples does the join of $R$ and $S$ produce, at most, and how many pages are required to store the result of the join back on disk?
2. Query Optimization (20 points)

Consider the following relational schema and SQL query scenario:

Suppliers(sid, sname, city)
Supply(sid, pid)
Parts(pid, pname, price)

Now consider the following query:

SELECT S.sname, P.pname
FROM Suppliers S, Parts P, Supply Y
WHERE S.sid = Y.sid AND Y.pid = P.pid AND
S.city = 'Amherst' AND P.price \leq 1000

(a) What information about these relations does the query optimizer need to select a good query execution plan for the given query?

(b) How many different join orders, assuming that cross-products are disallowed, does a System R style query optimizer consider when deciding how to process the given query? List each of these join orders.

(c) What indexes might be of help in processing this query? Explain briefly.

(d) How does adding DISTINCT to the SELECT clause affect the plans produced?

(e) How does adding ORDER BY sname to the query affect the plans produced?

(f) How does adding GROUP BY sname to the query affect the plans produced?

3. Analyzing concurrent schedules (30 points)

For each of the following schedules, state which of the following properties hold: conflict serializable, recoverable, cascadeless, or strict. A schedule is strict if a value written by a transaction T is not read or overwritten by other transactions until T either aborts or commits.

Please display your answer as a table with one row per schedule, and one column per property. Indicate the satisfied properties with a check mark.

(a) T1:R(X), T2:W(X), T1:W(X), T2:Abort, T1:Commit
(b) T1:R(X), T2:R(X), T1:W(X), T1:Commit, T2:W(X), T2:Commit
(c) T1:R(X), T2:W(X), T1:W(X), T2:Commit, T1:Commit
(d) T1:W(X), T2:R(X), T1:W(X), T2:Commit, T1:Abort
(e) T1:R(X), T2:R(X), T1:Commit, T2:W(X), T2:Commit
(f) T1:W(X), T2:R(Y), T1:R(Y), T2:R(X), T1:Commit, T2:Commit
4. Locking Protocols (20 points)

Consider the following two transactions:

\[ T_1 = R(A) \ W(A) \ R(B) \ W(B) \]
\[ T_2 = R(B) \ W(A) \ R(A) \ W(B) \]

Assume that exclusive lock (X) and unlock (U) actions are inserted by the scheduler, resulting in the following annotated transactions:

\[ T_1 = X(A) \ R(A) \ W(A) \ X(B) \ R(B) \ W(B) \], AFTER COMMIT: U(A) \ U(B) \\
\[ T_2 = X(B) \ R(B) \ X(A) \ R(A) \ W(A) \ W(B) \], AFTER COMMIT: U(A) \ U(B) \\

Consider concurrent execution of these two transactions and answer the following questions:

(a) Is conflict serializability guaranteed? Why or why not?
(b) Is deadlock possible? If so, then assuming that T1 starts first, which transaction(s) would be rolled back (aborted) under the wait-die deadlock prevention scheme?
(c) Is cascading rollback possible? If not, explain why not. If so, show a scenario that results in cascading rollback.

Now, say that lock and unlock actions are inserted in the following way instead:

\[ T_1 = X(A) \ R(A) \ W(A) \ X(B) \ U(A) \ R(B) \ W(B) \ U(B) \]
\[ T_2 = X(A) \ X(B) \ R(B) \ R(A) \ W(A) \ U(A) \ W(B) \ U(B) \]

(d) Is conflict serializability guaranteed? Why or why not?
(e) Is deadlock possible? If so, then assuming that T1 starts first, which transaction(s) would be rolled back (aborted) under the wound-wait deadlock prevention scheme?
(f) Is cascading rollback possible? If not, explain why not. If so, show a schedule that results in cascading rollback.