Question 1. [30 points] Short Questions

a) **[7 points]** Illustrate one example of a query plan that the System R optimizer (Selinger'79) will not consider and is a plan that is more efficient that the plans that the System R optimizer will consider.

b) **[8 points]** Consider the B-link protocol. What is the maximum number of latches that can be acquired by a transaction? Illustrate the scenario where this maximum number of latches is acquired.

c) **[7 points]** 2PC with Presumed Commit is always better (higher performant) than 2PC with Presumed Abort.

☐ True ☐ False

Justify your answer

d) **[8 points]** Consider a data set in the Mariposa distributed database system that is accessed very rarely. For any site, the cost of storing this data set may be more than the price it fetches during query evaluation. In a true cost-based economic model, no site will store this data set, and this dataset will be "lost." Assume that we want the system to not loose such data. Propose a solution to fix this problem.

Question 2. [30 points] Concurrency Control

a) **[15 points]** In the Gray at al. '76 paper the protocol says:

1. Before requesting an S or IS lock, **ANY** path to the root must be locked in IS mode or greater, and
2. Before requesting an IX, SIX, or X lock, **ALL** paths to the root must be locked in IX mode or greater.

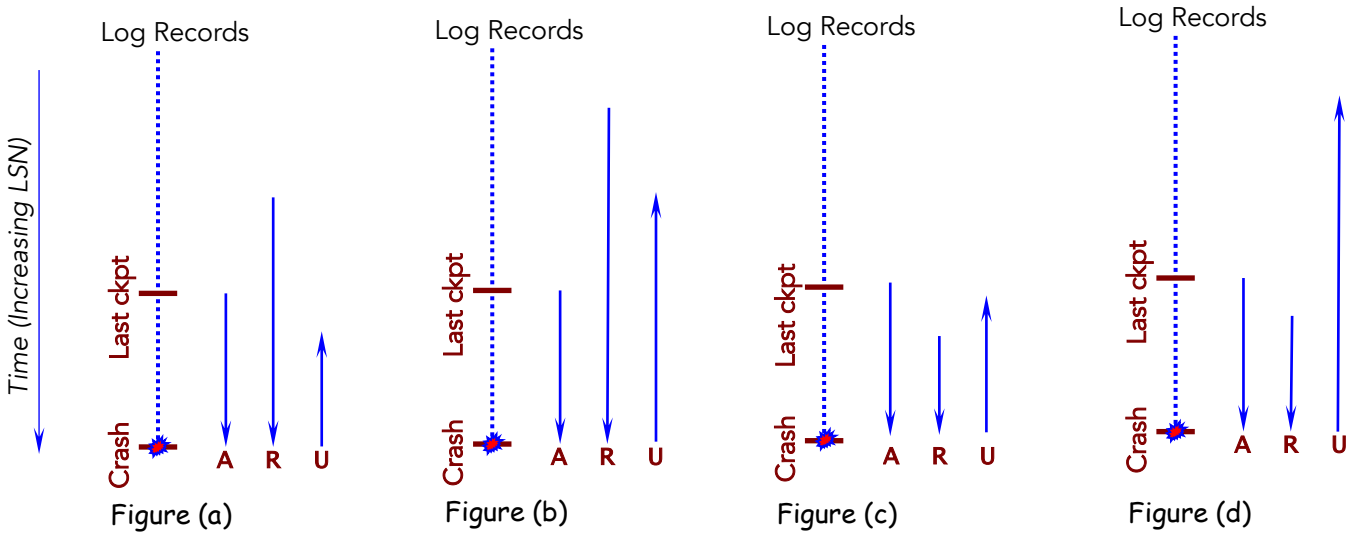
Could you flip the "ANY" and "ALL" in the protocol above? Explain.

b) **[5 points]** Consider the Optimistic Concurrency Control (CC) method by Kung and Robinson. Is their method ANSI serializable? Explain why or why not.

c) **[10 points]** In the Kung and Robinson optimistic CC protocol, what is the CC-related overhead that read-only transactions experience? Explain the overhead in each of the 3 phases of the protocol. Note that these transactions don't know that they will not incur any writes at the start of the transaction.

Question 3. [20 points] Recovery

a) **[6 points]** Consider the four figures below that indicate log records being processed by ARIES during recovery after a crash. Each dot on the timeline represents an update log record. In the log records assume that there is an update record right before and after the checkpoint record. The arrows denote the direction and the position from which the logs are scanned during the three phases of recovery. The arrow labeled A, R, and U denote the Analysis, Redo and Undo phases respectively. Which of these diagrams represent processing that can **never** occur in ARIES? Why?



Answer: _____

Why?:

e) **[14 points]** Consider disaster recovery scenarios in which there is an "operational" site where transactions are run. A snapshot of the database is stored at a "remote" location, and periodically logs are shipped to the remote site and "applied" using the redo phase of ARIES. Now, consider applying a **large** log file in which there are many more log records than records in the database. One option is to apply the logs in log sequence number, i.e. read the log file sequentially and redo the actions in the log. Can you think of a more efficient REDO algorithm in this case? Explain the performance benefits of your new method.

Question 4. [20 points] Query Processing

Traditional query processing operators are "blocking," as they need to examine all/most of the input before they produce **any** output. For example, a sort-merge join operator produces its first join output after it has examined **all** the tuples in the input relations (and produced sorted runs). There is a growing interest in non-blocking operator algorithms that produce **some** output results as soon as possible. With non-blocking operators, some results are produced fairly quickly, and the remaining results show up over time. In contrast, with a blocking operator all the result show up in a big chunk at or near the end of the operator execution.

Design a non-blocking **hash-based** join algorithm. Describe its IO complexity.

Question 5. [20 points] Parallel Databases

Parallel data platforms leverage "partitioned parallelism" to speed up individual query operations (e.g. a join) by using multiple processors to evaluate each operation. Consider the set-union operation, namely R U S, where R and S have the same schema. Recall, that in SQL UNION eliminates duplicates.

Design an ***efficient*** parallel algorithm to compute the set union operation when |R| << |S|, i.e. the cardinality of R is significantly smaller than the cardinality of S, and R is large enough that it can't fit in the aggregate memory of all the processors (Hint: you will likely need to think of some partitioning method).