Q 1. [20pts] For the following questions, clearly circle True or False.

1. The hash join algorithm always has fewer page I/Os compared to the block nested loop join algorithm.
   
   FALSE

2. Data shuffling among worker nodes is a key component of parallel query processing in MapReduce/Hadoop, Spark, and parallel DBMSs.
   
   TRUE

3. Using the double buffering technique typically reduces the total number of passes needed for an external merge sort.
   
   FALSE

4. All four SQL isolation levels guarantee that there will be no WW conflicts during a concurrent execution of transactions.
   
   TRUE

5. It is typically possible to speed up the processing of the aggregation $\gamma_{\text{COUNT}(\ast)}(R)$ by using a B+ tree index on R.
   
   TRUE/FALSE. Unfortunately, this question seems to have been too ambiguous. If the index is on all attributes of R or if it follows AltRecord, it cannot speed up this query; otherwise, it can.

6. An avoiding-cascading-aborts (ACA) schedule is always guaranteed to be a recoverable schedule.
   
   TRUE

7. Apart from schema information, the RDBMS catalog also stores statistics about both relation instances and indexes.
   
   TRUE
8. A hash index is typically efficient for answering selection queries with $\neq$ predicates (denoted $<>$ in SQL).

   **FALSE**

9. Spark’s API mostly subsumes the MapReduce programming abstraction.

   **TRUE**

10. The clock algorithm is an approximation of the MRU buffer replacement policy.

    **FALSE**

**Q 2. [10pts] Hash Join with non-uniform partitioning.** (This question is a small tweak on a quiz question!) We are joining two tables R and S, which have $4BN_R$ and $12BN_S$ pages respectively, using a hash join. We are given that $4BN_R \gg 12BN_S$ and that the number of available buffer pages is $4B + 1$. The buffer pool is initially empty. We are also given that $2FN_S = 4B - 1$, where $F$ is the hash table fudge factor.

   The distribution of the join attribute values in R and S are such that after the first hash partitioning phase, we get exactly $4B$ partitions each of R and S. Each partition of R is $N_R$ pages long, but the partitions of S have multiple lengths. Suppose that S gets partitioned as follows: $B$ partitions have length $N_S$ pages each, $2B$ partitions of length $3N_S$ pages each, and $B$ partitions of length $5N_S$ pages each.

   What is the I/O cost of the above join using the regular hash join algorithm discussed in class? Exclude the cost of writing the output. Assume that perfect uniform splitting occurs during the recursive repartitioning and that we do not need to recurse more than once. Briefly explain and show all of your calculations clearly.

   **(Hint: The answer is of the following form: $xBN_R + yBN_S$, where $x \in \{10, 12, 14, 16, 18, 20\}$ and $y \in \{50, 54, 58, 62, 66, 70\}$.)**

   **Answer:** The total I/O cost is $18BN_R + 58BN_S$. Clearly, the hash table is built on S, since it is the smaller relation. The lower bound is $3 \times (4BN_R + 12BN_S) = 12BN_R + 36BN_S$ (regular hash join cost). But we need to recursivly repartition the $2B + B$ partitions of S that are longer than $2N_S$ pages each (since we only have enough buffer memory to fit a hash table on $2N_S$ pages at a time), which means we also need to repartition the corresponding $3B$ partitions of R that are of length $N_R$ pages each. Thus, the I/O cost goes up by $2 \times (2B \times 3N_S + B \times 5N_S + 3B \times N_R) = 6BN_R + 22BN_S$.

**Q 3. [25pts] Query processing and optimization.** We are given a relational database schema with the following three relations, wherein $A$ and $B$ are discrete (string) attributes
and C, D, and E are numeric attributes.

\[ R(A, B, C), S(A, B), \text{and } T(B, D, E) \]

Q 3.1 [10pts] For each of the following questions, clearly circle either Yes or No for whether the given queries \( Q_1 \) and \( Q_2 \) are equivalent.

1. \( Q_1: \sigma_{A=\text{"a"}} \land B=\text{"b"} \left( (R \bowtie S) \bowtie T \right) \) and \( Q_2: (\sigma_{A=\text{"a"}} \land B=\text{"b"}(R \bowtie S)) \bowtie T \)

   Yes

2. \( Q_1: \pi_E(\sigma_{D=1}(R \bowtie T)) \) and \( Q_2: \pi_E(T) \bowtie \sigma_{D=1}(R \bowtie T) \)

   No. Trick question! They do not even have the same result schema. :)

3. \( Q_1: \pi_B(R \bowtie S) \) and \( Q_2: (\pi_B(R)) \bowtie (\pi_B(S)) \)

   No. \( Q_1 \) joins on A and B but \( Q_2 \) loses A before the join.

4. \( Q_1: \pi_{B,C,E}(R \bowtie (\sigma_{E=1}(T))) \) and \( Q_2: \sigma_{E=1}(\pi_{B,C,E}(R \bowtie T)) \)

   Yes

5. \( Q_1: R \bowtie S \bowtie T \) and \( Q_2: (R \bowtie S) \bowtie (S \bowtie T) \bowtie (T \bowtie R) \)

   Yes

Q 3.2 [9pts] We are given an instance of the relation \( T \) whose heap file has a size of 10 million pages. We are also given a B+ tree index on \( T \) with the index key \((B, D)\) that follows the alternative of storing RID lists in the leaf pages. The index has 5 million leaf pages. Assume that each \( B \) value is 8 B in size and that there are a million unique values of \( B \) in the given instance of \( T \). A page is 8 KB in size.

Describe an efficient physical query plan with I/O cost less than 9 million pages for processing the following SQL query, given 4 GB of buffer memory and an initially empty buffer pool. Mention if each physical operator is pipelined or materialized and compute the total I/O cost of your plan (in number of pages) rounded to the nearest million.

\[ \text{SELECT } B, \text{ AVG(D) FROM } T \text{ GROUP BY } B \]
**Answer:** This is a simple group by aggregate query. The union of the grouping list \((B)\) and the aggregation attribute \((D)\) is a subset of the index key \((B, D)\); thus we only need to read the leaf nodes. Furthermore, since the grouping list is a prefix of the index key, the leaf entries are already pre-sorted on \(B\), which means all entries of a group occur consecutively in the leaf pages. Finally, \(\text{AVG}\) can be computed incrementally for each group. Thus, overall, the physical query plan is simply a sequential scan of the leaf level of the tree. The I/O cost is only about 5 million pages.

**Q 3.3 [6pts]** Name three indexes on \(R\) (including at least one hash index) that match the predicate in the following SQL query and briefly explain why each index matches.

```sql
SELECT * FROM R
WHERE (B = "b" OR A <> "a") AND NOT (C <= 20 OR A <> "a")
```

**Answer:** First, rewrite the predicate to CNF:

\[(B = "b" \text{ OR } A <> "a") \text{ AND } (C > 20) \text{ AND } (A = "a"\)]

Now, matching indexes become obvious (check class notes). For example, B+ tree indexes that match include one on \(A\), on \(C\), on \((C, A)\), on \((A, B)\), and so on. A hash index on \(A\) also matches. At first glance, it seems that no index on \(B\) would match. Since the conjunct with \(B\) has an OR on another attribute, a hash index on \(B\) or on \\{\(B, A\)\} will not match that conjunct. Similarly, any B+ tree index with \(B\) as a prefix of the index key will not match that conjunct. But looking deeper, if we rewrite the CNF further using the basic rules of logic, we find something interesting:

\[
\Leftrightarrow (B = "b" \text{ OR } A <> "a") \text{ AND } (A = "a" \text{ AND } C > 20)
\]

\[
\Leftrightarrow (B = "b" \text{ AND } A = "a" \text{ AND } C > 20) \text{ OR } (A <> "a" \text{ AND } A = "a" \text{ AND } C > 20)
\]

\[
\Leftrightarrow (B = "b" \text{ AND } C > 20 \text{ AND } A = "a") \text{ OR } (\text{FALSE AND } C > 20)
\]

\[
\Leftrightarrow (B = "b" \text{ AND } C > 20 \text{ AND } A = "a") \text{ OR } \text{FALSE}
\]

\[
\Leftrightarrow B = "b" \text{ AND } C > 20 \text{ AND } A = "a"
\]

Now, we see that a hash index on \(B\) matches this new first conjunct! In fact, a hash index on \\{\(A, B\)\} is even better, since it matches two conjuncts put together! Similarly, a B+ tree index on \(B,\) on \(B, C\), on \(B, A\), and so on will now match at least the first conjunct. In fact, all 15 B+ tree indexes possible on \(R\) will now match this query!

**Q 4. [21pts]** Transaction management and concurrency control. We are given a database with three distinct data objects A, B, and C. We are also given the following three trans-
actions that arrive concurrently.

\[ T_1 : R(A), W(A), R(B), W(B), \text{Commit} \]
\[ T_2 : R(B), W(B), R(C), W(C), \text{Commit} \]
\[ T_3 : R(C), W(C), R(A), W(A), \text{Commit} \]

**Q 4.1 [2pts]** Give a clear example of a *serial* schedule of the three transactions.

**Answer:** $T_2 \rightarrow T_1 \rightarrow T_3$; any other ordering is fine too.

**Q 4.2 [5pts]** Give a clear example of a *serializable* schedule of the three transactions that is not serial but is equivalent to the serial schedule in your above answer.

**Answer:** The following interleaved schedule is equivalent to the above serial schedule:

\[ R_{T_1}(A), R_{T_2}(B), W_{T_1}(A), W_{T_2}(B), R_{T_2}(C), W_{T_2}(C), \text{Commit}_{T_2}, \]
\[ R_{T_1}(B), R_{T_3}(C), W_{T_1}(B), W_{T_3}(C), \text{Commit}_{T_1}, R_{T_3}(A), W_{T_3}(A), \text{Commit}_{T_3} \]

**Q 4.3 [14pts]** Consider the following interleaved schedule. For each of the questions that follow, clearly circle yes or no.

\[ R_{T_1}(A), R_{T_2}(B), R_{T_3}(C), W_{T_1}(A), W_{T_2}(B), W_{T_3}(C), \]
\[ R_{T_1}(B), W_{T_1}(B), \text{Commit}_{T_1}, R_{T_2}(C), W_{T_2}(C), \text{Commit}_{T_2}, R_{T_3}(A), W_{T_3}(A), \text{Commit}_{T_3} \]

1. Is the schedule serializable?
   - No

2. Is the schedule recoverable?
   - No

3. Does the schedule have a WW conflict between any pair of transactions?
   - Yes

4. Does the schedule have a WR conflict between any pair of transactions?
   - Yes
5. Does the schedule have a RW(R) conflict between any pair of transactions?
   No

6. Suppose we use the READ UNCOMMITTED isolation level of SQL. Will it lead to a deadlock with the given schedule?
   Yes

7. Suppose we use the SERIALIZABLE isolation level of SQL. Will it lead to a deadlock with the given schedule?
   Yes

Q 5. [24pts] For the following questions, clearly circle the right answer (only one option is correct).

1. Which of the following symbols does not represent a relational operator from the extended relational algebra?
   (a) γ       (b) ∪       (c) µ       (d) π       (e) ×
   ANSWER: (c)

2. Which of the following relational operators do not preserve the schema of (at least one of) their inputs?
   (a) Set union  (b) Set intersection  (c) Set difference  (d) Select  (e) Project
   ANSWER: (e)

3. Which of the following relational operators can be processed using a regular (unmodified) hash join implementation?
   (a) Set union  (b) Set intersection  (c) Set difference  (d) Select  (e) Project
   ANSWER: (b)

4. Which of the following SQL aggregates require a shuffle among worker nodes in a parallel DBMS when the GROUP BY list is empty?
(a) SUM   (b) AVG   (c) VARIANCE   (d) MEDIAN   (e) MAX
ANSWER: (d). Recall from class that MEDIAN is not an algebraic aggregate.

5. Which file organization is typically the most efficient for inserting new records?

(a) Heap file   (b) Sorted file   (c) B+ tree index with AltRecord
ANSWER: (a)

6. We are given this join query: \( R \bowtie S \bowtie T \bowtie U \). Recall that some query optimizers only consider left-deep join trees for join order enumeration. How many different left-deep join trees exist for this query? (Hint: Swapping the left and right input of a \( \bowtie \) in a given tree yields a different tree.)

(a) 4   (b) 6   (c) 15   (d) 16   (e) 24
ANSWER: (e). The left-deep tree has 4 spots; so, the number of trees is the number of permutations of those 4 spots, i.e., \( 4! = 24 \).

7. Which is the dominant parallelism paradigm that is used in parallel DBMSs, MapReduce/Hadoop, and Spark?

(a) Shared-nothing   (b) Shared-memory   (c) Shared-Disk
ANSWER: (a)

8. Given the following bags \( A = \{a, b, c, a, a, x, y, x, z\} \) and \( B = \{e, a, b, d, x, y, a, y\} \), what is the result of the bag intersection of \( A \) and \( B \) (INTERSECT ALL in SQL)?

(a) \( \{a, b, x, y\} \)   (b) \( \{b, a, y, x, a\} \)   (c) \( \{x, b, a, x, y\} \)   (d) \( \{a, b, c, x, y, z\} \)
ANSWER: (b)

9. In a hard disk, which of the following components of the data access time accounts for the delay caused by the radial movement of the arm?

(a) Seek time   (b) Rotational delay   (c) Transfer time
ANSWER: (a)
10. Given 1 GB of buffer memory, what is roughly the largest file size (to the nearest order of magnitude) that can technically be sorted with only 2 passes?

(a) 10 GB     (b) 1 TB     (c) 100 TB     (d) 10 PB     (e) 100 PB

**ANSWER:** Any of the above. This question was worked out in class but unfortunately, I forgot to mention the page size here. So, everyone gets these 2pts. For what it is worth, I explain the correct answer. Since buffering and blocked I/O could increase the number of passes, we ignore them. Thus, to upper bound \( N \), we need \( \log_{B-1}(\lceil N/2B \rceil) = 1 \), which means \( N \approx 2B(B - 1) \). Typical page sizes are between 4 KB and 16 KB, which means \( B \) is roughly between 64 thousand and 250 thousand. So, \( N \) is roughly between 8 billion (16 KB pages) and 125 billion (4 KB pages). So, the upper bound is roughly between 120 TB and 500 TB.

11. Suppose we are given two union-compatible relations \( R \) and \( S \) with sizes \( N_R \) and \( N_S \) (\( \ll N_R \)) pages respectively. Suppose we have \( B = 2 + F \times N_S/2 \) pages of buffer memory (\( F \) is the hash table fudge factor) and the buffer pool is initially empty. What is the minimum possible I/O cost of a **UNION ALL** of \( R \) and \( S \), excluding output write cost?

(a) \( 6N_R + 6N_S \)     (b) \( 4N_R + 4N_S \)     (c) \( 2N_R + 2N_S \)     (d) \( N_R + 2N_S \)     (e) \( N_R + N_S \)

**ANSWER:** (e). Just read and append the two files.

12. In a B+ tree index, which nodes are allowed to have duplicates of the index key?

(a) Leaf nodes     (b) Root node     (c) Non-root internal nodes

**ANSWER:** (a)

**Extra Credit Q. [5pts]** Suppose you are working for Netflix and using a Hadoop cluster. You are given a dump of a large Ratings table with the same schema from class: Ratings (RID, MovieID, UserID, Stars, RateDate). The table is stored as a distributed CSV file on HDFS that is hash partitioned on RID. You are tasked with obtaining the number of five-star ratings for each movie in the given file. Unfortunately, your Hadoop cluster only has regular MapReduce (no Hive, Pig, Spark, or any SQL engine).

How will you implement this task in MapReduce? A proper explanation of the Map and Reduce phases are enough; no need to write code in Java, Python, etc.

Tired of writing low-level MapReduce programs, you decide to convince your boss that Netflix really needs to use Hive or SparkSQL. How can you use the above task to achieve your goal?
Answer: The task can be expressed as the following query in relational algebra, where $R$ denotes the Ratings table: $\gamma_{\text{MovieID}, \text{COUNT}(\ast)}(\sigma_{\text{Stars}=5}(R))$. This is a select-aggregate query, which can be easily expressed using MapReduce. In the Map function, after parsing the record, if it satisfies Stars=5, emit a key-value pair with the key being the record’s MovieID and the value being 1. In the Reduce function, add up all the values in the iterator for each MovieID and emit (output to file) the MovieID and the sum obtained.

Of course, thanks to 190D, you must now be going: "Wait, I can do the same thing with just one line in SQL!" If not, well, you should be! :) So, install Hive or SparkSQL, define the Ratings table schema using their DDL, load the CSV file into the DBMS (you can also create an "external table" to avoid a data copy), and go:

```
SELECT MovieID, COUNT(*) FROM Ratings WHERE Stars=5 GROUP BY MovieID
```

For added wows, also build an index on Stars a priori (Hive has indexes but SparkSQL does not yet). So there you have it, higher productivity and faster performance even on the latest hot "Big Data" DBMSs thanks to ideas that are over 3 decades old—your boss will be floored. Do send me a thank you note when you get a raise. :)