CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (SPRING 2025)
PROF. JIGNESH PATEL

Homework #4 (by Chris and Yuchen) – Solutions
Due: **Sunday, March 23, 2025 @ 11:59pm**

IMPORTANT:
- Enter all of your answers into **Gradescope by 11:59pm on Sunday, March 23, 2025**.
- **Plagiarism**: Homework may be discussed with other students, but all homework is to be completed **individually**.

For your information:
- Graded out of **100** points; **3** questions total
- Rough time estimate: $\approx$ 2 - 3 hours (0.5 - 1 hours for each question)

Question	Points	Score
Sorting Algorithms	32	
Join Algorithms	44	
Query Execution, Planning, and Optimization	24	
Total:	100	

Question 1: Sorting Algorithms .... [32 points]
   Graded by:

We have a database file with 4 million pages (N = 4,000,000 pages), and we want to sort it using external merge sort. Assume that the DBMS is not using double buffering or blocked I/O, and that it uses quicksort for in-memory sorting. Let B denote the number of buffers.

(a) **[4 points]** Assume that the DBMS has <u>50</u> buffers. How many sorted runs are generated? Note that the final sorted file does not count towards the sorted run count.
   □ 34   □ 1,633   □ 80,000   ■ **81,667**   □ 81,670   □ 81,671

   Solution:
   $$\left\lceil \frac{4,000,000}{50} \right\rceil + \left\lceil \frac{80,000}{49} \right\rceil + \left\lceil \frac{1,633}{49} \right\rceil = 81,667$$

(b) **[4 points]** Again, assuming that the DBMS has <u>50</u> buffers. How many passes does the DBMS need to perform in order to sort the file?
   □ 2   □ 3   ■ **4**   □ 5   □ 6

   Solution:
   $$1 + \left\lceil \log_{B-1}\left(\left\lceil \frac{N}{B} \right\rceil\right) \right\rceil = 1 + \lceil \log_{49}\left(\lceil 4,000,000/50 \rceil\right)\rceil$$
   $$= 1 + 3 = 4$$

(c) **[4 points]** Again, assuming that the DBMS has <u>50</u> buffers. How many pages does each sorted run have after the third pass (i.e. Note: this is Pass #2 if you start counting from Pass #0)?
   □ 49   □ 50   □ 2,450   □ 2,500   □ 117,649   ■ **120,050**   □ 125,000

   Solution: On the first pass, B buffer pages will be used to create the sorted runs. From the second pass onward, B-1 runs will be sorted through a K-way merge.
   First pass: 50 pages for each sorted run. Third pass: $50 * 49 * 49 = 120,050$ pages for each sorted run.

(d) **[4 points]** Again, assuming that the DBMS has <u>50</u> buffers. What is the total I/O cost to sort the file?
   □ 4,000,000   □ 16,000,000   ■ **32,000,000**   □ 64,000,000   □ 96,000,000

   Solution: $Cost = 2N \times \#passes = 2 \times 4,000,000 \times 4 = 32,000,000$

(e) **[4 points]** Suppose the DBMS has <u>67</u> buffers. What is the largest database file (expressed in terms of the number of pages) that can be sorted with external merge sort

---

using <u>three</u> passes?
☐ 11,342   ☐ 120,050   ☐ 278,852   ■ **291,852**   ☐ 300,763

Solution: We want the largest integer N such that $N \leq B \times (B - 1)^2$. The largest such value is $B \times (B - 1)^2$ itself, which is $67 \times 66^2 = 291,852$

(f) **[4 points]** What is the smallest number of buffers B such that the DBMS can sort the target file using only <u>three</u> passes?
☐ 157   ☐ 158   ☐ 159   ■ **160**   ☐ 161

Solution: We want the smallest integer B such that $N \leq B \times (B - 1)^2$. If $B = 160$, then $4,000,000 \leq 160 \times 159^2 = 4,044,960$; any smaller value for B would fail.

(g) For each of the following statements about sorting, pick True or False.

    i. **[4 points]** The DBMS receives a query that requires sorting. Assume that the sort order is a <u>prefix</u> of the index key. Under which scenario(s) will using an unclustered B+Tree index have comparable performance to a clustered B+Tree index:
☐ The sort order exactly matches the index key.
■ **Query contains a LIMIT 1, and the first tuple answers the query.**
■ **All attributes accessed by the query are contained in the index.**
☐ Using unclustered index will always perform worse than using clustered index.
☐ None of the above.

Solution: If the query contains a LIMIT 1 and the first tuple answers the query, then both unclustered/clustered will make a single heap I/O. Hence, this is correct.

When all attributes accessed by the query are contained in the index, the DBMS can perform an index-only or covering scan and avoid accessing the heap. Hence, this is also correct.

    ii. **[2 points]** Sort aggregation cannot handle <u>**can not**</u> be used for all aggregates discussed in lecture (i.e., "DISTINCT", "GROUP BY"). For some of them, we have to use hash aggregation.
☐ True   ■ **False**

Solution: All aggregates can be implemented as some form of (1) sort to form groups and (2) additional processing to then produce the answer.

    iii. **[2 points]** It is always more efficient for a DBMS to use a hash aggregate than a sort aggregate.
☐ True   ■ **False**

Solution: If the aggregation query then has a further "ORDER BY", the DBMS may benefit from using a sort aggregate instead.

Question 2: Join Algorithms .... [44 points]
Graded by:

Consider relations X(a, b), Y(a, c, e), and Z(a, d, f) to be joined on the common attribute a. Assume that there are no indexes available on the tables to speed up the join algorithms.

- There are $B = 450$ pages in the buffer
- Table X spans $M = 1,500$ pages with 200 tuples per page
- Table Y spans $N = 250$ pages with 450 tuples per page
- Table Z spans $O = 2,000$ pages with 140 tuples per page
- The join result of Y and Z spans $P = 170$ pages

For the following questions, assume a simple cost model where pages are read and written one at a time. Also assume that one buffer block is needed for the evolving output block and one input block is needed for the current input block of the inner relation. You may ignore the cost of the writing of the final results.

(a) **[2 points]** What is the I/O cost of a simple nested loop join with Y as the outer relation and X as the inner relation?
- ☐ 300,250
- ☐ 375,250
- ☐ 675,250
- ☐ 27,050,000
- ☐ 60,750,250
- ■ **168,750,250**

Solution: $N + n \times M = 250 + 250 \times 450 \times 1,500 = 168,750,250$

(b) **[2 points]** What is the I/O cost of a block nested loop join with Y as the outer relation and Z as the inner relation?
- ☐ 1,650
- ☐ 1,900
- ☐ 2,100
- ■ **2,250**
- ☐ 2,700
- ☐ 3,250
- ☐ 3,450

Solution: $N + \lceil \frac{N}{B-2} \rceil \times O = 250 + \lceil \frac{250}{448} \rceil \times 2,000 = 250 + 2,000 = 2,250$

(c) **[2 points]** What is the I/O cost of a block nested loop join with Z as the outer relation and Y as the inner relation?

☐ 1,650
☐ 1,900
☐ 2,100
☐ 2,250
☐ 2,700
■ **3,250**
☐ 3,450

Solution: $O + \lceil \frac{O}{B-2} \rceil \times N = 2,000 + \lceil \frac{2,000}{448} \rceil \times 250 = 2,000 + 1250 = 3,250$

(d) For a sort-merge join with Z as the outer relation and X as the inner relation:

    i. **[3 points]** What is the cost of sorting the tuples in X on attribute a?
      ☐ 1,500
      ☐ 3,000
      ■ **6,000**
      ☐ 8,000
      ☐ 12,000

      Solution: $passes = 1 + \lceil \log_{B-1}(\lceil \frac{M}{B} \rceil) \rceil = 1 + \lceil \log_{449}(\lceil \frac{1,500}{450} \rceil) \rceil = 1 + 1 = 2$
      $2M \times passes = 2 \times 1,500 \times 2 = 6,000$

    ii. **[3 points]** What is the cost of sorting the tuples in Z on attribute a?
      ☐ 2,000
      ☐ 4,000
      ☐ 6,000
      ■ **8,000**
      ☐ 16,000

      Solution: $passes = 1 + \lceil \log_{B-1}(\lceil \frac{O}{B} \rceil) \rceil = 1 + \lceil \log_{449}(\lceil \frac{2,000}{450} \rceil) \rceil = 1 + 1 = 2$
      $2O \times passes = 2 \times 2,000 \times 2 = 8,000$

    iii. **[3 points]** What is the cost of the merge phase in the worst-case scenario?
      ☐ 340
      ☐ 1,640
      ☐ 3,500
      ☐ 28,000
      ☐ 210,000
      ☐ 2,800,000
      ■ **3,000,000**
      ☐ 3,250,000

      Solution: $O \times M = 2,000 \times 1,500 = 3,000,000$

    iv. **[3 points]** What is the cost of the merge phase assuming there are no duplicates in

the join attribute?

☐ 340
☐ 1,640
■ **3,500**
☐ 28,000
☐ 210,000
☐ 2,800,000
☐ 3,000,000
☐ 3,250,000

Solution: $O + M = 2,000 + 1,500 = 3,500$

v. **[3 points]** Now consider joining Y, Z and then joining the result with X. What is the cost of the final merge phase assuming there are no duplicates in the join attribute?

☐ 420
■ **1,670**
☐ 1,920
☐ 2,170
☐ 28,000
☐ 63,000
☐ 90,000

Solution: $P + X = 170 + 1,500 = 1,670$

(e) **[2 points]** Consider a hash join with Y as the outer relation and X as the inner relation. You may ignore recursive partitioning and partially filled blocks. What is the cost of the combined probe and partition phases?

☐ 2,250
☐ 3,500
☐ 5,000
■ **5,250**
☐ 6,750
☐ 10,500

Solution: $3(N + M) = 5,250$

(f) **[3 points]** Assume that the tables do not fit in main memory and that a large number of distinct values hash to the same bucket using hash function h_1. Which of the following approaches works the best?

☐ Create two hashtables half the size of the original one, run the same hash join algorithm on the tables, and then merge the hashtables together.

■ **Create hashtables for the inner and outer relation using h_1 and rehash into an embedded hash table using h_2 != h_1 for large buckets.**

☐ Use linear probing for collisions and page in and out parts of the hashtable needed at

a given time.

☐ Create hashtables for the inner and outer relation using h_1 and rehash into an embedded hash table using h_1 for large buckets.

> **Solution:** Use Grace hash join with recursive partitioning, which is what the correct option describes.

(g) For each of the following statements about joins, pick True or False.

   i. **[2 points]** If both tables in a simple nested loop join fit entirely in memory, the order of inner and outer tables does not significantly affect I/O costs.
      ■ **True**   ☐ False

   > **Solution:** If both tables fit entirely in memory, then they can be read just once and therefore the order would not be very important.

   ii. **[2 points]** If neither table fits entirely in memory, I/O costs would be lower if we process both tables on a per-block basis rather than per-tuple basis.
      ■ **True**   ☐ False

   > **Solution:** A block nested loop join has fewer disk accesses when compared to a simple nested loop join.

   iii. **[3 points]** A sort-merge join is faster than a hash join on all circumstances.
      ☐ True   ■ **False**

   > **Solution:** Sort merge join can be just as fast as hash join under specific circumstances. For example, if the sort merge is performed on already-sorted data (i.e. sort cost is 0 and overall cost is M+N), and the hash join is perform on data can fit entirely in memory where overall cost is M+N.

   iv. **[3 points]** An index nested loop join requires an index on the outer- and inner- tables.
      ☐ True   ■ **False**

   > **Solution:** An index nested loop join only requires the inner- table to have an index.

   v. **[3 points]** For a hash join to work, the inner table (or its partitions) need to fit into memory.
      ☐ True   ■ **False**

   > **Solution:** The inner table can be any size. Only outer table (or its partitions) need to fit in memory.

   vi. **[5 points]** A nested loop join can output a sorted stream of tuples under the following condition:
      ☐ All nested loop joins can output a sorted stream.

☐ No intra-operator parallelism.
☐ Outer- table (or data) is already sorted.
■ **Outer- table (or data) is sorted and no intra-operator parallelism.**
☐ Inner- table (or data) is already sorted.
☐ Inner- table (or data) is sorted and no intra-operator parallelism.

> **Solution:** With no intra-operator parallelism, a nested loop join sequentially iterates through the outer- table to find matching tuples from the inner- table. If the outer- table is already sorted, the nested loop join will preserve its order.
> *Grading Note:* Due to ambiguity and the question being underspecified, all students were given credit for this question.

Question 3: Query Execution, Planning, and Optimization . . . . . [24 points]
Graded by:

(a) **[2 points]** The iterator model allows tuples to continously flow through the entire sequence of operators in the execution plan before retrieving the next tuple.
☐ True　■ **False**

> **Solution:** False. The statement is true only for a single pipeline, but a query can have multiple pipelines. If an operator is a pipeline breaker (e.g. build-side of hash join, subqueries, order-by), it cannot emit tuples until all its children emit all their tuples.

(b) **[2 points]** Assume that the DBMS zone maps are up to date. The DBMS can use these zone maps to answer specific queries without reading any actual table heap tuples:
■ **True**　☐ False

> **Solution:** True. If the zone maps are up-to-date, the DBMS could answer queries asking for the minimum value of a given attribute (amongst others) by only looking at zone maps.

(c) **[2 points]** Assuming a query with multiple OR predicates. Using a multi-index scan will always perform better than a sequential scan.
☐ True　■ **False**

> **Solution:** False. If the OR predicates cover enough of the table, it may be more performant to use a sequential scan.

(d) **[2 points]** For OLAP queries, which often involve complex operations on vast datasets, intra-query parallelism is typically not preferred to optimize performance.
☐ True　■ **False**

> **Solution:** False. OLAP queries, characterized by their complex operations on large volumes of data, can greatly benefit from intra-query parallelism. By executing the operations of a single query in parallel, it helps in significantly decreasing the latency, thus optimizing the performance of these types of queries.

(e) **[2 points]** The process per DBMS worker approach provides better fault isolation and scheduling control than the thread per DBMS worker approach.
☐ True　■ **False**

> **Solution:** False. While the process per DBMS worker approach does provide better fault isolation, thread per worker gives the DBMS finer control over scheduling.

(f) **[2 points]** In OLAP workload, the vectorized model's performance improvements come mainly from the reduction in the number of disk I/O operations.
☐ True　■ **False**

> **Solution:** False. While the Vectorized Model can reduce some I/O operations due to its batch processing, its primary advantage is from reducing CPU overhead, optimizing cache utilization, and leveraging SIMD instructions.

(g) **[2 points]** The query optimizer in a database management system always guarantees the generation of an optimal execution plan by exhaustively evaluating all possible plans to ensure the lowest cost for query execution.
☐ True   ■ **False**

> **Solution:** No, it is usually not necessary to estimate the cost of every plan for a query via a cost model. In this case, the time it would take to enumerate every plan and then filter out the plans to pick the most optimal one would introduce too high of an overhead compared to the query time itself. Usually, DBMSs will use rule-based optimizations (or heuristics) first, transforming the plan into a more simple one.

(h) **[2 points]** Predicate and projection pushdown will always improve query performance.
☐ True   ■ **False**

> **Solution:** False. If the predicate/projection being pushed down involves an expensive function call (i.e., UDF), the query may benefit from deferring it to later.

(i) **[2 points]** The execution plan with the lowest cost is guaranteed to be the most efficient among all execution plans enumerated by the query optimizer.
☐ True   ■ **False**

> **Solution:** False. The execution plan is not guaranteed to be the most efficient, as the statistics used to compute cardinality might be stale and the cost model may not accurately reflect the actual performance of the plan (e.g. assume all predicates in a conjunction is indepedent, while there might be some correlation).

(j) **[2 points]** Sampling statistics requires evaluating each tuple in the entire table.
☐ True   ■ **False**

> **Solution:** False. Sampling only needs to look at a subset (or representative sample) of the table.

(k) **[2 points]** Equi-depth histogram maintains counts for a group of values instead of each unique key to reduce memory footprint and uses the same range size for each bucket.
☐ True   ■ **False**

> **Solution:** False. Equi-depth histogram varies the range size of buckets so that the total number of occurrences for each bucket is roughly the same.

(l) A database contains a single table: University(id,name,state,city). You need to estimate the cardinality of the following query:

SELECT * FROM University WHERE state = 'PA' AND city = 'Pittsburgh'

For the following questions, assume University has 5,000 rows with 6% having state = 'PA', and 0.6% having city = 'Pittsburgh'.

   i. **[1 point]** Under uniform data assumption and independent predicates assumption, what is the estimated cardinality c of this query? Take $\lceil c \rceil$ of the result.
     ☐ 1   ■ **2**   ☐ 10   ☐ 30   ☐ 300

Solution: $\lceil 5,000 \times 0.6 \times 0.06 \rceil = \lceil 1.8 \rceil = 2$.

---

ii. **[1 point]** Is the result from previous question an overestimate or underestimate of the true cardinality?

☐ Overestimate   ■ **Underestimate**

Solution: We have an underestimate of the true cardinality. The true cardinality is 30 rows, which is significantly higher than the estimated cardinality of 2 rows. The reason is that state and city are not independent attributes - they are highly correlated. In fact, all universities in Pittsburgh are in PA.