IMPORTANT:
- Enter all of your answers into Gradescope by 11:59pm on Sunday Oct 9, 2022.
- 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: ≈ 1 - 2 hours (0.5 - 1 hours for each question)

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorting Algorithms</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Join Algorithms</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Query Execution</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

Revision: 2022/09/29 14:28
Question 1: Sorting Algorithms .......................... [35 points]

We have a database file with fourteen million pages ($N = 14,000,000$ 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) [8 points] Assume that the DBMS has eight buffers. How many passes does the DBMS
need to perform in order to sort the file?
☐ 8  ☐ 9  ☐ 10  ☐ 11  ☐ 12

(b) [9 points] Again, assuming that the DBMS has eight buffers. What is the total I/O cost
to sort the file?
☐ 224,000,000  ☐ 252,000,000  ☐ 280,000,000  ☐ 308,000,000  ☐ 336,000,000

(c) [9 points] What is the smallest number of buffers $B$ that the DBMS can sort the target
file using only eight passes?
☐ 7  ☐ 8  ☐ 9  ☐ 2,450  ☐ 2,451  ☐ 2,452  ☐ 3,742  ☐ 3,743

(d) [9 points] Suppose the DBMS has forty-two buffers. What is the largest database file
(expressed in terms of $N$, the number of pages) that can be sorted with external merge
sort using four passes?
☐ 2,894,682  ☐ 3,111,696  ☐ 3,185,784  ☐ 118,681,962  ☐ 130,691,232
☐ 133,802,928  ☐ 4,865,960,442  ☐ 5,489,031,744  ☐ 5,619,722,976

Homework #3 continues…
Question 2: Join Algorithms .............................. [35 points]

Consider relations R(a, b, c), S(a, d), and T(a, e, 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 \( B = 445 \) pages in the buffer
- Table R spans \( M = 1,500 \) pages with 80 tuples per page
- Table S spans \( N = 4,500 \) pages with 150 tuples per page
- Table T spans \( O = 200 \) pages with 250 tuples per page

Answer the following questions on computing the I/O costs for the joins. You can assume the simplest cost model where pages are read and written one at a time. You can also assume that you will need one buffer block to hold the evolving output block and one input block to hold the current input block of the inner relation. You may ignore the cost of the writing of the final results.

(a) [4 points] Block nested loop join with \( S \) as the outer relation and \( R \) as the inner relation:
\[
\begin{array}{cccccccc}
17,500 & 18,000 & 18,500 & 19,000 & 19,500 & 20,500 & 21,000 \\
\end{array}
\]

(b) [4 points] Block nested loop join with \( R \) as the outer relation and \( S \) as the inner relation:
\[
\begin{array}{cccccccc}
17,500 & 18,000 & 18,500 & 19,000 & 19,500 & 20,500 & 21,000 \\
\end{array}
\]

(c) Sort-merge join with \( S \) as the outer relation and \( R \) as the inner relation:

i. [3 points] What is the cost of sorting the tuples in \( R \) on attribute \( a \)?
\[
\begin{array}{cccccccc}
3,000 & 6,000 & 9,000 & 12,000 & 15,000 & 18,000 \\
\end{array}
\]

ii. [3 points] What is the cost of sorting the tuples in \( S \) on attribute \( a \)?
\[
\begin{array}{cccccccc}
3,000 & 6,000 & 9,000 & 12,000 & 15,000 & 18,000 \\
\end{array}
\]

iii. [3 points] What is the cost of the merge phase in the worst-case scenario?
\[
\begin{array}{cccccccc}
1,500 & 3,000 & 4,500 & 6,000 & 12,000 & 2,250,000 \\
6,750,000 & 20,250,000 & 1,080,000 \\
\end{array}
\]

iv. [3 points] What is the cost of the merge phase assuming there are no duplicates in the join attribute?
\[
\begin{array}{cccccccc}
1,500 & 3,000 & 4,500 & 6,000 & 12,000 & 2,250,000 \\
6,750,000 & 20,250,000 \\
\end{array}
\]

v. [3 points] Now consider joining \( R \), \( S \) and then joining the result with \( T \). Suppose the cost of the final merge phase is 800 and assume that there are no duplicates in the join attribute. How many pages did the join result of \( R \) and \( S \) span?
\[
\begin{array}{cccccccc}
2 & 4 & 6 & 8 & 200 & 400 & 600 & 800 \\
\end{array}
\]

(d) Hash join with \( S \) as the outer relation and \( R \) as the inner relation. You may ignore recursive partitioning and partially filled blocks.

i. [4 points] What is the cost of the probe phase?
\[
\begin{array}{cccccccc}
1,500 & 3,000 & 4,500 & 6,000 & 9,000 & 12,000 & 18,050 \\
\end{array}
\]

Question 2 continues...
ii. [4 points] What is the cost of the partition phase?
- 1,500
- 3,000
- 4,500
- 6,000
- 9,000
- 12,000
- 18,050

(e) [4 points] Assume that the tables do not fit in main memory and that a high cardinality of distinct values hash to the same bucket using your hash function \( h_1 \). Which of the following approaches works the best?
- Create hashtables for the inner and outer relation using \( h_1 \) and rehash into an embedded hash table using \( h_2 \neq h_1 \) for large buckets
- Create hashtables for the inner and outer relation using \( h_1 \) and rehash into an embedded hash table using \( h_1 \) for large buckets
- Use linear probing for collisions and page in and out parts of the hashtable needed at a given time
- Create 2 hashtables half the size of the original one, run the same hash join algorithm on the tables, and then merge the hashtables together

Homework #3 continues...
Question 3: Query Execution ........................................... [30 points]

(a) [6 points] Which processing model has on average the smallest working buffer per operator invocation? Ignore optimizations like projection pushdown. Select only one answer.

- □ Iterator
- □ Materialization
- □ Vectorization

(b) [6 points] In the iterator processing model, the logic of an operator is independent of its children and parents. (i.e., the code does not case on what type of iterator the children or parents are)

- □ True
- □ False

(c) [6 points] In the vectorized processing model, each operator that receives input from multiple children requires multi-threaded execution to generate the Next() output tuples from each child.

- □ True
- □ False

(d) [6 points] The iterator processing model often leads to good code locality (in the instruction cache sense).

- □ True
- □ False

(e) [6 points] An index scan is always better (fewer I/O operations, faster run-time) than a sequential scan, regardless of the processing model.

- □ True
- □ False