Comp115 Spring 2017, HW4 (bonus)

Due April 11th, 2017

Submit via provide using:

```
$ provide comp115 115HW4-QEval <single_file.pdf>
```
or online at: `https://www.cs.tufts.edu/comp/115/provide.cgi`

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. The cost of writing out the result should be uniformly ignored, but bear in mind that one output buffer should be used for producing the result.

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.
Both relations are stored as simple heap files.
Neither relation has any indexes built on it.
1002 buffer pages are available.

ANSWER: Let M = 10,000 be the number of pages in R, N = 200,000 be the number of pages in S, and B = 1002 be the number of buffer pages available.

1. What is the cost of joining R and S using a page-oriented simple nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged? **[10 pts]**

   ANSWER: The basic idea is to read each page of the outer relation, and for each page scan the inner relation for matching tuples. Total cost would be:

   $pagesinouter + (pagesinouter \cdot pagesininner)$

   which is minimized by having the smaller relation be the outer relation.

   $TotalCost = M + (M \cdot N) = 2,000,010,000$

   The minimum number of buffer pages for this cost is 3.

2. 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? **[10 pts]**

   ANSWER: This time read the outer relation in blocks, and for each block scan the inner relation for matching tuples. So the outer relation is still read once, but the inner relation is scanned only once for each outer block, of which there are $\lceil \frac{pagesinouter}{B-2} \rceil$.

   $TotalCost = M + N \cdot \lceil \frac{M}{B-2} \rceil = 2,010,000$

   The minimum number of buffer pages for this cost is 1002.

3. 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? **[20 pts]**

ANSWER:

Since $B > \sqrt{N} > \sqrt{M}$ we can use the refinement to Sort-Merge discussed on pages 254-255 in the text.

$TotalCost = 3 \cdot (M + N) = 630,000$

The best external sort algorithm can create runs with length $2 * B$ using B buffers. So, we want B such that:

$\lceil \frac{M}{2B} \rceil + \lceil \frac{N}{2B} \rceil \leq B$, by simplifying we get: $\frac{M}{2B} + \frac{N}{2B} \leq B$, so, $\frac{M+N}{2B} \leq B$, so $B^2 \geq \frac{M+N}{2}$, so $B^2 \geq \frac{210000}{2} = 105000$, so $B \geq \sqrt{105000} = 324.04$, so $B \geq 325$

***NOTE:** if R.a were not a key, then the merging phase could require more than one pass over one of the relations, making the cost of merging M*N I/Os in the worst case. The minimum number of buffer pages required is 325. With 325 buffer pages, the initial sorting pass will split R into 16 runs of size 650 and split S into 308 runs of size 650 (approximately). These 324 runs can then be merged in one pass, with one page left over to be used as an output buffer. With fewer than 325 buffer pages the number of runs produced by the first pass over both relations would exceed the number of available pages, making a one-pass merge impossible.*

4. 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? **[20 pts]**

   ANSWER: The cost of Hash Join is $3 * (M + N)$ if $B > \sqrt{M}$ M is the number of pages in the smaller relation, S (more details in page 258). Since $\sqrt{M} = 100$, we can assume that this condition is met. We will also assume uniform partitioning from our hash function.

   $TotalCost = 3 \cdot (M + N) = 630,000$

   A good guess is that we need $B > \sqrt{M}$.

   ***Note:** An answer that takes into account the fudge factor f is also accepted. Then the memory requirements would be $B > \sqrt{f \cdot M}$*

5. 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. **[15 pts]**

   ANSWER: The optimal cost would be achieved if each relation was only read once. We could do such a join by storing the entire smaller relation in memory, reading in the larger relation page-by-page, and for each tuple in the larger relation we search the smaller relation (which exists entirely in memory) for matching tuples. The buffer pool would have to hold the entire smaller relation, one page for reading in the larger relation, and one page to serve as an output buffer.

   $TotalCost = M + N = 210,000$

   The minimum number of buffer pages for this cost is $M + 1 + 1 = 10,002$.

6. 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? **[15 pts]**

   ANSWER: Any tuple in S can match at most one tuple in R because R.a is a primary key (which means the R.a field contains no duplicates). So the maximum number of tuples in the result is equal to the number of tuples in S, which is 4,000,000. The size of a tuple in the

result could be as large as the size of an R tuple plus the size of an S tuple (minus the size of the shared attribute). This may allow only 10 tuples to be stored on a page. Storing 4,000,000 tuples at 10 per page would require 400,000 pages in the result.

7. Would your answers to any of the previous questions in this exercise change if you were told that S.b is the primary key for S? **[10 pts]**

   ANSWER: If b is PK for S, and a is PK for R then, this contradicts the statement that each R tuple joins with exactly 20 S tuples.