CS143: Database Systems
Homework #7

1. Consider the relation $R(A, B, C)$ with the following properties:

   - R contains 8,000 tuples
   - The size of a disk block is 4,000 bytes
   - The size of each R tuple is 40 bytes
   - Each disk block holding R tuples is full

   (a) How many blocks are required to store the relation R?

   ANSWER:
   Each disk block holds $4000/40 = 100$ R tuples, so R is stored in $8000/100 = 80$ blocks.

   (b) We would like to sort the tuples of R on attribute A using the sort-merge algorithm in two passes. That is, after the initial sorting pass, we want to generate the completely sorted relation by one more merging pass. What is the minimum number of main memory blocks required?

   ANSWER:
   10. In order to finish sort-merge algorithm in two passes, the number of sorted runs created from the first sorting pass should be at most one less than the available memory blocks. Assuming that we have M memory blocks, the number of sorted runs created from the first sorting pass is $\lceil 80/M \rceil$. This should be less than or equal to $M - 1$. That is, $\lceil 80/M \rceil \leq M - 1$. The minimum such M is 10.

2. Consider two relations $R(A, B)$ and $S(A, C)$. Suppose relation R occupies 1,000 blocks and relation S occupies 100 blocks. When we performed $R \bowtie S$ using the hash join algorithm that we learned in the class, the algorithm incurred (approximately) 3,300 disk block I/Os (either read or write), excluding the I/Os for writing the final join result. Write the minimum number of main-memory buffers (each of the size of a disk block) that we must have had when we performed the join.

   ANSWER:
   12. The total cost of the join is 3 times the sizes of the input relations, indicating that the hash-join algorithm went through just one bucketizing step and one join step. This means that each bucket of the smaller table should fit in main memory during the join step (with two extra memory blocks left for reading the other table and for writing the output). Assuming we have M memory blocks, the size of each bucket of the smaller table S after one bucketization step is $\lceil \frac{100}{M-1} \rceil$. This number should be smaller than or equal to $M-2$. That is, That is, $\lceil \frac{100}{M-1} \rceil \leq M-2$. The minimum M that satisfies this inequality is 12.

3. Suppose you have 2 relations, R(A, B, C) and S(B, C, D, E).

   You have a clustering unique (no duplicate keys) B+-tree index on attribute A for relation R. Assume this index is kept entirely in memory (i.e., you do not need to read it from disk).

   For relation S, you have a non-clustering non-unique B+-tree index for attribute B. Furthermore, assume that the index is kept in memory (i.e. you do not need to read it from disk).

   Other relevant data:

   - 500 tuples of R are stored per block on disk.
   - $|R|$ = 750,000 (number of tuples of R).
   - 100 tuples of S are stored per block on disk.
   - $|S|$ = 250,000 (number of tuples of S).
   - For every R tuple, there are roughly 5 tuples in S with R.B = S.B.

   You want to execute the following query:

   ```
   SELECT * FROM R, S WHERE R.B=S.B AND R.C=S.C
   ```

   We present you with the following query plans:

   ```
   For every block Bi of R, retrieved using the clustered index on A for R
       For every tuple r of Bi
           Use the index on B for S to retrieve all of the tuples s of S such that s.B=r.B
               For each of these tuples s, if s.C=r.C, output r.A, r.B, r.C,
                                                              s.B, s.C, s.D, s.E
   ```

   How many disk I/Os are needed to execute this query plan?

   ANSWER:
   3,751,500. $\frac{750{,}000}{500} = 1{,}500$ blocks to read R + $750{,}000 \cdot 5 = 3{,}750{,}000$ blocks of S read.

   Now assume that we have a clustering non-unique B+-tree index for attribute C of S. For every R tuple, there are roughly 5000 tuples in S with R.C = S.C. Assume that all of the tuples of S that agree on attribute C are stored in sequentially adjacent blocks on disk (that is, if more than one block is needed to store all of the tuples with some value of C, then these blocks will be sequentially located on the disk). The index is kept in main memory.

   Using this new index we came up with a second query plan:

   ```
   For every block Bi of R, retrieved using the clustered index on A for R
       For every tuple r of Bi
           Use the index on C for S to retrieve all of the tuples s of S such that s.C=r.C
           For each of these tuples s, if s.B=r.B, output r.A, r.B, r.C,
                                                          s.B, s.C, s.D, s.E
   ```

   Now analyze each of the two plans more carefully in terms of their behavior regarding accesses to disk. Your analysis should consider the behavior of the number of I/Os and access time. Explain which of the two plans is therefore better under what circumstances. For the analysis of access time, you do not need to compute a concrete number. Just include in your analysis what accesses to disk are sequential accesses and which ones are random accesses and discuss its consequence on overall access time.

   ANSWER:
   Both plans require the same 1,500 I/O operations to read the blocks of R.

The first plan requires an additional $(750,000 \cdot 5)$ random I/Os.

The new plan requires an additional $(750,000 \cdot \frac{5000}{100})$ sequential I/Os.

Thus the new plan requires approximately 10 times more I/O (5 vs. 50 blocks of S per tuple in R). If the cost of random access is more than 10 times the cost of sequential access, then the second plan will require less time to complete.