A

CSE 444: Database Internals.
Section 4: Operator Algorithms.

1. 15.3.4-B

Consider two relations R and S such that $T(R) \geq T(S)$. How would you perform a nested-loop join in the following cases so as to minimize the number of I/Os, given that the memory size is M blocks large.

[Write the optimized pseudo-code]

(1) R is unclustered but S is clustered.
B(R) = 1000 B(S) = 500 M = 101
[R outside: $B(R)/(M-1)(T(R)/B(R)(M-1)+B(S)) = T(R)+B(S)B(R)/(M-1) = T(R)+5000$]
[S outside: $B(S)/(M-1)(M-1+T(R)) = B(S)+B(S)T(R)/(M-1) = 500+5T(R)$]

(2) R is clustered but S is unclustered.
[S outside: $B(S)/(M-1)(T(S)/B(S)(M-1)+B(R)) = T(S)+B(R)B(S)/(M-1) = T(S)+5000$]
[R outside: $B(R)/(M-1)(M-1+T(S)) = B(R)+B(R)T(S)/(M-1) = 1000+10T(S)$]

2. 15.4.4-A

Consider the join of two sorted relations R and S with $B(R) = 1000$ and $B(S) = 500$ and $M = 101$. Let the join attribute by Y such that only two value of Y are present that appear equally in half the tuples of R and S. How do we compute the join? And, how many I/Os do we need?

[We read in 100 blocks of R at a time and read the relevant parts of S one at a time. So 1000 for R, and 250 each of S repeated 10 times. For a total of 3500. If we take the alternative approach: we read 400 blocks of S for which we only need to read 500 blocks from R and one 100 block of S where we need to read the whole of R. This leads to a total of $500 + 4 * 500 + 1 * 1000 = 3500$.]

3. 15.4.9

Suppose you are sorting a relation R with $B(R)$ blocks. The two-pass sort-merge algorithm proposed in the lecture takes $3B(R)$ I/Os to produce the sorted file. Can we do better by not writing some blocks onto the disk during the first pass?

[Mention that if the k runs on disk and you are left with n blocks such that $n+k \leq M$, then you need not write out the last k blocks out. This saves you $2k$ blocks in the second pass. The total cost is thus: $3B(R) - 2n$ and this works for databases of size $n + (B(R) - n)/M \leq M$.]

[Alternatively, you could always keep the first block of each sorted list in memory. The first run, thus, has $M - 1$ tuples, the second has $M - 2$ and so on. $B(R) \leq M(M + 1)/2$. The total I/O is $3B(R) - 2M$.]

4. SORTING AN ARBITRARILY LARGE FILE

Suppose you want to sort a file R with $B(R)$ blocks on a machine with M blocks of memory. Unlike the case discussed in the lecture, you can not assume any bound on the size of $B(R)$ (*i.e.*, $B(R) > M^2$). Can you suggest an algorithm to sort this file? How many disk I/Os do you need?

[You recursively merge the files creating a merge-tree with a fanout of M.]