COS 597D, Fall 2013
Example done on board 11/13/13

Problem:
Consider the evaluation of query (R ◊◊ S ◊◊ T) for relations R, S, and T, where ◊◊ denotes natural join. Relations R, S and T share a single attribute, A, and no pair of relations shares any other attributes.

R contains 50,000 tuples and has 5 tuples per block; S contains 100,000 tuples and has 10 tuples per block; T contains 1000 tuples and has 25 tuples per block.

Forty-five (45) buffer blocks are available in main memory for the evaluation of the query. (Note in class I wrote 43 buffers, which is why I kept being off by 2 from my notes.

Note that T can fit in the main memory buffer with 5 pages to spare. Given this, the *block nested loop* algorithm becomes a more desirable algorithm.

Question:
If *block nested loop* is used for both joins, what is the best order to do the joins in terms of minimizing disk I/O? Do calculations of the disk I/O cost of possible plans.

Part a: For block nested loop, we want to minimize the number of times we re-read the inner-loop relation by bringing large chunks of the outer relation into the buffer at one time. For this reason, it is desirable to have T as the outer relation when it is one of the join arguments. We also need to consider how large the intermediate relation (result of the first join) is. Assuming pages of R, S, and T on disk are full:

R requires 10,000 pages,
S requires 10,000 pages and
T requires 40 pages.

The only information we have to help us estimate the size of any intermediate result is that A is the primary key of S. Therefore the join of S with R or T on A will have no more tuples than R or T has, respectively (at most one tuple of S will match each tuple of R or T on attribute A). Of course the tuples of the intermediate result will have more attributes. Therefore, the size of the records for these tuples is larger, and less records fit in a page. Our only estimate for the size of R ◊◊T is based on the product of the number of tuples in each.

Let us prune our search for a good plan by eliminating R ◊◊ T as a possible first join due to the large estimated size of the intermediate result (50 million tuples with less than 5 tuples per page).

Consider two possibilities:

1. *Compute (T ◊◊ S) = I first*, with T the outer relation. The cost is |T| + |S| = **10,040 disk page I/Os** because T fits in the buffer. The intermediate result is estimated to be 1000 tuples with about 7 tuples per page (0.04 pages per tuple of T + 0.1 pages per tuple of S = 0.14 pages per tuple of the intermediate result, giving 7.14 tuples per page). So the intermediate result requires 143 pages, assuming records cannot be split across pages. *Then compute I ◊◊ R*. I uses the smaller number of pages and we would normally put it as the outer relation. If we materialize I, paying a write of 143 pages, we have the full buffer to do the second join and block nested loop join can be done in cost (|I| + |I| + ⌈|I|/43⌉ ∗ |R|) = **40,286 disk page I/Os**. If we don't materialize, computing (T ◊◊ S) is taking 42 buffer pages, one of which is the output page collecting I. We have three additional buffer pages but need only 2 for R and the final output. Therefore, we can use two to collect I. Then, the block nested loop is executed in cost ⌈|I|/2⌉ * |R| = 720,000 disk page I/Os, with I again as the outer relation. Clearly, materializing is better.
   TOTAL COST = 50,326 disk page I/Os

   Note that we could give up fitting all of T in the buffer when computing T◊◊S and use more of the buffer pages for pipelined output to the second join calculation. However note that any reduction below 40 in the buffer pages allocated to T increases the cost of T◊◊S by at least |S|=10,000 because T will be read in pieces. Materializing only costs a write and read of I, which is 286.

2. *Compute (R ◊◊ S) = I first*. Since they are both the same size in pages, arbitrarily choose R as the outer relation. Use 43 buffer pages for R. The cost is |R|+ ⌈|R|/43⌉ ∗|S|) = 2,340,000. We need go no further – it is already clear that our first possibility is substantially cheaper.

Conclusion: compute T◊◊S= I first, with T the outer relation; then compute I◊◊R with I as the outer relation. I is materialized.
TOTAL COST = 50,326 disk page I/Os