

# Database Tuning, ITU, Spring 2009

Rasmus Pagh

February 17, 2009

## 1 Robust sort-merge join

Suppose we have relations  $R_1(A_1, A_2, A_3)$  and  $R_2(A_2, A_4)$ , where  $A_2$  is the primary key of  $R_2$ . All attributes have fixed length. The length of  $A_1$  is 10 bytes, the length of  $A_2$  is 20 bytes, the length of  $A_3$  is 30 bytes, and the length of  $A_4$  is 40 bytes. The attribute  $A_2$  is a key for  $R_2$ . The relations reside on a disk with block size  $B=3000$  bytes. We want to perform a sort-merge join of  $R_1$  and  $R_2$ , using 101 blocks (i.e., 303000 bytes) of internal memory, of which one is reserved to be used as an output buffer.

In the following we consider a sort-merge join of  $R_1$  and  $R_2$ , as described in RG. The replacement sort method that produces longer runs on average is not used, but we use the refinement that produces the join result as the runs of  $R_1$  and  $R_2$  are read concurrently from disk.

a) For what values of  $|R_1|$  and  $|R_2|$  does the sort-merge join use two passes? (Write a condition involving  $|R_1|$  and  $|R_2|$ .)

b) Assume that  $|R_1| = 10,000$  and  $|R_2| = 30,000$ . How many I/Os are used for the sort-merge join in the worst case? Count also I/Os to write the output.

It is pointed out in RG that the sort-merge join algorithm presented there may be doing a lot of work in the “atypical” case where the set of tuples with a particular value on the join attribute(s) does not fit into internal memory. The aim of the following is to arrive at an algorithm that handles the general case efficiently. Let  $B(R)$  and  $B(S)$  denote the number of disk blocks of two relations  $R$  and  $S$ , respectively.

c) Modify the two-pass sort-merge join described in RG to get a sort-merge join that, in the worst case, uses  $3(B(R) + B(S)) + O(B(R) \bowtie S)$  I/Os to compute the sort-merge join of two relations  $R$  and  $S$ . You may assume that internal memory has space for approximately  $2\sqrt{B(R) + B(S)}$  blocks. Argue that your algorithm has the desired complexity.

## 2 Sort-merge-based join on partly sorted input

Relations are often stored on disk sorted according to some attribute(s), for example relations organized in a clustered B<sup>+</sup>-tree index. Also, results of some relational algebra operations are produced in sorted order. For simplicity we assume here that a relation is stored in a sequence of consecutive disk blocks (no “holes”).

We now consider a 2-pass sort-merge join, as described in RG section 14.4.2, in the special case where one of the relations is already sorted according to the join attribute(s). Let  $R$  denote the unsorted relation, and  $S$  denote the sorted relation. We denote by  $B(R)$  and  $B(S)$  the number of disk blocks occupied by  $R$  and  $S$ , respectively, and by  $M$  the number of disk blocks that fit in internal memory.

**a)** Which part of the 2-pass sort-merge join algorithm may be skipped in this special case? Argue that the resulting I/O complexity is  $3B(R) + B(S)$ , not counting I/Os to write the output.

**b)** Describe a modification of the 2-pass sort-merge join algorithm that uses  $3B(R) + B(S)$  I/Os, not counting I/Os to write the output, and works in the above special case if  $M > \sqrt{B(R)} + 2$ . Argue for the the I/O complexity and memory requirements of the algorithm.