

COSC-4411: Assignment #3 & #4

Due: Friday 26 November 2004

1. (5 points) **External Sorting.** *A better sort of sort?*

Dr. Datta Bas has developed an “improved” version of the standard external sort routine. External sorting is usually quite efficient as few passes over the records are required. However, when the buffer pool is small and/or the file to sort is huge, a more significant number of passes may be needed, requiring a read and write of every page every pass.

Dr. Bas has made the observation that we might improve performance as follows: we do not need to keep *all* the fields of the records during the sort, but just the fields that are part of the sort key (and the *rid*)!

For Dr. Bas’s external sort routine, pass 0 (the initial sorting pass) and the final (merge) pass are modified. Pass 0 is modified to “project” the records, removing the unneeded fields. The final (merge) pass—call it pass f —is modified to “join” back the removed fields. Merge passes $1 \dots f - 1$ proceed just as in the regular external sort routine. The only difference is that the number of pages in the “file” for these runs is much smaller because we have removed the fields unneeded for the sorting.

Assume the buffer pool allocation is B . Pass 0 uses one buffer frame as *input* to read sequentially each page of the file to sort. It projects each record in the input frame to a record with just *search key* + *page#* + *slot#* and places it sequentially in the array of the remaining $B - 1$ frames. The *page#* + *slot#* here represent the record’s *rid* in the original file. When the $B - 1$ frames allocated for sorting become full, the projected records are then sorted, and written out as a $B - 1$ page sorted run.

Pass f must have only $B - 2$ runs remaining to merge. One frame is reserved for *output* as before. $B - 2$ frames are used as input to merge the (potentially) $B - 2$ runs. The last frame is reserved to be used to fetch the page with the original record (from the original file) for each projected record in the merge stream, in order to retrieve the missing fields and add them back. (Recall that each projected record includes the *rid* of the original record for this purpose.)

Assume that you have file **F** to sort. File **F** is 400 pages and 5 records fit per page. The “projected” records for Dr. Bas’s routine fit 50 records to a page, so after pass 0 of his routine, the “file” fits in 40 pages. Your buffer pool allocation (B) for the sort is 6 frames.

- a. (2 points) First, calculate the I/O cost of sorting **F** using the basic external sort routine. (Assume that pass 0 produces runs of size B , so 6 in this case.)
- b. (2 points) Now calculate the I/O cost of sorting **F** using Dr. Bas’s external sort routine. (Note that Bas’s pass 0 produces runs of size $B - 1$, so 5 in this case.)
- c. (1 point) Under what conditions, if any, is Dr. Bas’s sort routine advantageous? If none, briefly explain why not.

2. (5 points) **Few Sort.** *The few, the proud, the sort routine.*

Dr. Bas has noted that for many queries for which the query optimizer uses sorting in the query plan the input table is large, but the number of distinct values over the sort key is small.

For instance, consider that table **T** has one million records (10^6) over 10,000 pages, we want to sort it on **T.A**, but there are only 70 distinct values of **A** in **T**. Given eleven buffer frames for the job, running the basic external sort routine on it would take four passes.

Describe an external sorting routine, *few sort*, that could sort a large table with few distinct values more efficiently than the regular external sort. You can assume you have some auxiliary memory (in main memory) beyond the buffer pool for bookkeeping purposes.

Cost out how expensive your routine would be sorting **T** given eleven buffer frames.

3. (5 points) **Sorting.** *Over doing it.*

We want to re-implement the sort-merge join algorithm to handle *over-sorting*.

Consider table **R** with attributes **A** and **B**, table **S** with attributes **B** and **C**, and $\mathbf{R} \bowtie_{\mathbf{B}} \mathbf{S}$. Normally for the sort-merge join, both **R** and **S** would be sorted on **B**. However, we could sort **R** on **B, A**—a nested sort over both attributes, as in order by **B, A**—and the merge could still work.

Call this *over-sorting* when we sort one or the other table, or both, on more attributes than just the join attributes.

What changes would need to be made to the basic sort-merge join algorithm to accommodate *over-sorting*, and what restrictions, if any, would we need to place on what *over-sorting* is permitted?

Would a sort-merge join implementation that accommodates *over-sorting* be potentially useful? Why or why not?

4. (5 points) **Evaluating the relational operators.**

Selecting the join and joining the Selection.

Consider tables **R** with attributes A and B and **S** with attributes B and C. Column B is unique in **S**. Values of B in **R** are the same as the values of B in **S**. Assume that there are no indexes available.

Consider the sort-merge join to be the efficient *two-pass* version discussed in the textbook (in which the sort-merge's merge steps are integrated with the external sort's merge steps) and not the more general version. (In the more general version, the external sort is *entirely* done before the merge-join. Furthermore, assume that the sort-merge join algorithm produces runs of length $2B$ in "pass 0" of sorting.) Consider that there may be problems with skew.

For each of the following, choose which of the following join algorithms is likely to be best. In each case, there are 250 buffer frames allocated for the join.

- A. A block nested loops join with **R** as the outer and with **S** as the inner.
- B. A block nested loops join with **S** as the outer and with **R** as the inner.
- C. A (2-pass) sort-merge join with **R** as the outer and with **S** as the inner.
- D. A (2-pass) sort-merge join with **S** as the outer and with **R** as the inner.
- E. A hash join with **R** as the outer and with **S** as the inner.
- F. A hash join with **S** as the outer and with **R** as the inner.

Table **R** is 3,000,000 pages with 80 records a page, and table **S** is 40,000 pages with 100 records a page. Which would be the most cost effective method to evaluate $\mathbf{R} \bowtie_{\mathbf{B}} \mathbf{S}$?

5. (5 points) **Reduction Factors and Access Paths.** *The path less worn.*

Consider table **T** with attributes A, B, C, D, & E. Table **T** has 1,000,000 records and **T** has no key.

- Column A has 10 values, 1...10.
- Column B has 100 values, 1...100.
- Column C has 1,000 values, 1...1,000.
- Column D has 10,000 values, 1...10,000.
- Column E has 100,000 values, 1...100,000.

Assume independence of the attributes, and a uniform distribution for each.

The following are possible indexes for **T**.

- 1) B+ tree unclustered index on D + C + B.
- 2) B+ tree clustered index on D + B + C.
- 3) B+ tree clustered index on C + A + B + D.
- 4) hash unclustered index on A + B + D + E.
- 5) hash clustered index on A + B + E.
- 6) hash clustered index on D + A + B + E.

Assume that the depth of each B+ tree above is four: so it costs four I/O's from the root to the *data entry* page. Assume the hash indexes are extendible hashes, so one I/O to get the directory page and one I/O to get the data entry page, so two I/O's.

We are interested to retrieve records from **T** that satisfy

where $A = 7$ and $B = 53$ and $D \leq 1,000$ and $E \geq 40,000$ and $E \leq 41,000$

Estimate how many records will be retrieved.

Which index provides the best access path? What is its I/O cost?

Assuming that many tuples are returned, which index provides the best access path *and* preserves an *interesting order* on D? What is its I/O cost?

6. (5 points) **Query Planning.** *Yo! Optimize this!*

Consider the schema

```

Sailors(sid, sname, rating, age)
Reserves(sid, bid, day)
    FK (sid) refs Sailors
    FK (bid) refs Boats
Boats(bid, bname, color, size, maint)
    FK (maint) refs Sailors (sid)

```

This is the same as the textbook's usual "Yacht Club" database, but with one addition: there is a foreign key from **Boats** referencing **Sailors** to indicate which sailor is in charge of maintaining that boat.

The underlined attributes indicate the primary key for each table, and the FK's the foreign keys. Assume no columns are nullable.

The following catalog information is available.

- **Sailors**: 1000 records on 50 pages
 - **Sailors.rating**: 10 values (1..10)
- **Reserves**: 10,000 records on 250 pages
 - **Reserves.bid**: 100 values
 - **Reserves.day**: 500 values
- **Boat** has 100 records on 10 pages.
 - **Boat.size**: 10 values (3..12 meters)

Available indexes are as follows. Each follows alternative #2, with data entries.

- A unique, hash index on **sid** for **Sailors** (10 data-entry pages / buckets). Assume this is a linear hash.
- A unique, hash index on **bid** for **Boat** (2 data-entry pages / buckets). Assume this is a linear hash.
- A unique, clustered B+ tree index on **sid + bid + day** (in that order) for **Reserves** (75 data-entry pages / leaves). Assume two levels of index pages. At level three are the data entry pages.
- An unclustered hash index on **day** for **Reserves** (25 data-entry pages / buckets). Assume this is a linear hash.

Consider the following query.

```
select S.sid, S.sname, B.bname, R.day
  from Sailor S, Reserves R, Boat B
 where S.sid = R.sid and R.bid = B.bid and
        B.maint = S.sid;
```

- a. (1 point) Using the assumptions of uniform distributions and column independence that System R—the type of optimizer the textbook presents—makes, estimate how many tuples this query returns.
- b. (4 points) Design a query plan that System R might find. (So do not do anything outside of System R's search space. You do not have to mimic System R's steps, but you should stay within the plan space it explores.)

Ideally, your query plan's estimated cost should be under 300 I/O's. The query plan has 10 buffer-pool frames available.

In your answer, show your query plan clearly as an annotated tree (as in the textbook examples). Calculate the cost estimation of your query step-by-step clearly.