CSE-4411A

Assignment #2 & #3

1. (10 points) **Indexes, Access Paths, & Algorithms.** *You take the low road...* [Exercise]

Table **R** has an unclustered tree index on A, B, C, D. The index pages contain 99 *index records* (encompassing 99 pointers, plus a $100th$ pointer), on average;[1] data-entry pages contain 50 data entries each, on average; and data-record pages contain 20 data records each, on average.

A's values range over 1..10,000; B's over 1..1,000; C's over 1..100; and D's over 1..10.

a. (1 point) Estimate the I/O cost of

```
select * from R where A > 3000 and B > 500;
```

using the index as the access path.

b. (2 points) Estimate the I/O cost of

```
select * from R where A > 9000 and C > 80;
```

using the index as the access path. Assume a smart query optimizer and processor.

c. (2 points) Is it possible to have a query of the form

```
select * from R where ... order by C, D;
```

that accesses **R** via the index *and* does the order by on the fly? If yes, show an example query. Otherwise, explain why not.

[1] This accounts for the fill factor. A page could hold more than 99 index records.

d. (2 points) Consider that you have two tables to join, **L** and **S**, by an equality condition on their mutual column A. **L** is exceedingly large, while **S** is reasonably small. Both, however, are larger than the buffer pool. Given a buffer pool allocation of B frames, assume $B^2 > |\mathbf{S}|$, where $|\mathbf{S}|$ is the number of pages of table **S**.

Which would you favour? A (two-pass) sort-merge join (SMJ), or a (two-pass) hash join (HJ)? Explain.

e. (3 points) If you could have an index to have a better join plan for the join in Question 1d, what would it be?

Under what conditions would it allow for a better join than the SMJ and HJ (again assuming the buffer pool is large enough that at least one of them is possible as two-pass).

And what would the join plan be?

2. (10 points) **Algorithms for Relational Operators.** *Trash the hash.* [Analysis]

a. (4 points) Consider a merge join of **R** and **S** on the mutual attribute J. Let **S** be the inner stream and be sorted already on J, K.

Under which conditions is the output sorted on J, K?

Under what conditions is the output *not* sorted on J, K?

b. (3 points) Could the basic 2-pass hash join benefit from sequential reads and/or writes? If not, explain why it cannot.

Otherwise, describe what part of the algorithm benefits (e.g., writing the partitions for the outer in pass 0, etc.).

c. (3 points) Dr. Bas recently discovered how great *index intersection* is. He realized that the same idea can be applied to a *single*-index access path when the index is unclustered:

- collect together the matching data entries using the index;
- *sort* the data entries by their *rid*'s (record identifiers); and
- then retrieve the data records in order.

He claims that this is then just like using a clustered index; the only extra cost is sorting the data entries.

When he ran an experiment, however, he found this performed no better than using the unclustered index directly! It still seemed to cost one I/O per data record retrieved.

Why is Dr. Bas wrong about this being *just* like using a clustered index (except for the extra sorting step)? Are there situations when it is advantageous?

3. (10 points) **System R.** *Forest for the trees.* [Short Answer]

   Consider a nine-relation SELECT query (so involving eight joins). Say that **level 1** of the Selinger algorithm determines the single-relation access paths, **level 2** determines the two-relation plans, and so forth.

   a. (2 points) How many left-deep trees are possible?

   b. (2 points) At least how many plans are carried forward from **level 4** (so that join four of the relations)?

   c. (2 points) At least how many plans are explored in **level 4**?

   d. (2 points) What is often an advantage of left-deep trees over other trees that is useful in query plans?

   e. (2 points) What can be a disadvantage of left-deep trees in finding a "best" query plan?

4. (10 points) **Query Planning & Optimization.** *This is the last time I enrol!* [Exercise]

Schema:

Student(sid, sname, startdate, major, advisor)
FK (advisor) refs **Prof** (pid)
Class(cid, dept, number, section, term, year, room, time, pid, ta)
FK (pid) refs **Prof**
FK (ta) refs **Student** (sid)
Enrol(sid, cid, date, grade)
FK (sid) refs **Student**
FK (cid) refs **Class**
Prof(pid, pname, pdept, office)

Assume no attribute is nullable. The attribute pid in **Class** refers to the the professor / instructor for the class. The attribute ta in **Class** refers to the teaching assistant for the class. The attribute advisor in **Student** refers to the student's academic advisor.

Statistics:

- **Student**: 50,000 records on 1,000 pages
  - advisor: 2,500 distinct values
- **Enrol**: 2,000,000 records on 20,000 pages
  - sid: 50,000 distinct values
  - cid: 80,000 distinct values
- **Class**: 80,000 records on 1,600 pages
  - pid: 4,000 distinct values
  - ta: 5,000 distinct values
- **Prof**: 4,000 records on 40 pages

Indexes:

- **Student**:
  - clustered tree index on sid (200 data entries per page)
- **Enrol**:
  - clustered tree index on cid, sid (167 data entries per page)
  - unclustered tree index on sid, cid (167 data entries per page)
- **Class**:
  - clustered tree index on cid (200 data entries per page)
- **Prof**:
  - clustered tree index on pid (200 data entries per page)

All indexes are of alternative #2. For each tree index, the index pages are 3 deep, except for the index on **Prof**.pid which is 2 deep.

Consider the query

```
select S.sname, E.cid
    from Student S, Enrol E
    where S.sid = E.sid
      and E.grade = 'A'
      and S.major = 'CS';
```

Assume %10 of students are CS majors, and that there are five possible grades: *A*, *B*, *C*, *D*, and *E*.

Consider the join algorithms we have discussed in class: block nested loops (BNL), index nested loops (INL), (two-pass) hash join (HJ), two-pass sort-merge (SMJ), and the general sort-then-merge join (MJ).

---

a. (6 points) Find the best query plan (by estimated cost) for the query. Show the query tree, *fully* annotated with the chosen algorithms and access paths.

   You have an allocation of 50 buffer-pool frames.

b. (1 point) Estimate the cardinality (number of rows produced) of this query.

c. (3 points) Assume that you additionally had an unclustered tree index on **Enrol** on sid, grade, cid (133 data entries per page).

Would this allow you to have a less expensive query plan than your plan for Question 4a?