Question 1 [20 points] Evaluation of selections

Consider a relation with this schema:

Employees(eid: integer, ename: string, sal: integer, title: string, age: integer)

Suppose that the following indexes, all using Alternative (2) for data entries, exist:

a) a B+ tree index on sal,

b) a clustered B+ tree index on <age, sal>.

Each Employees record is 100 bytes long, and you can assume that each index data entry is 20 bytes long. The Employees relation contains 10,000 pages and each page holds 20 data records.

Consider each of the following selection conditions. Assume that the reduction factor (RF) for each term that matches an index is 0.1, except those terms on the primary key. Compute the cost of the most selective access path, including the file scan and various index scans, for retrieving all Employees tuples that satisfy the condition:

(1) [6 Points] sal > 100

(2) [6 Points] age = 25

(3) [8 Points] sal > 200 and age > 20

Question 2 [28 Points] Costs of join algorithms

Consider the join $R \bowtie S$, given the following information about the relations to be joined. The cost metric is the number of page I/Os unless otherwise noted, and the cost of writing out the result should be uniformly ignored.

Relation R contains 10,000 tuples and has 10 tuples per page.
Relation S contains 4,000 tuples and also has 10 tuples per page.
Attribute b of relation S is the primary key for S.
Both relations are stored as simple heap files.
Neither relation has any indexes built on it.
41 buffer pages are available.

(1) [6 Points] What is the cost of joining R and S using a block nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged?
(2) [5 Points] What is the cost of joining R and S using a sort-merge join?

(3) [7 Points] What is the cost of joining R and S using a two-phase hash join? What is the minimum number of buffer pages required for this cost to remain unchanged?

(4) [10 Points] Hybrid Hash Join. Given B = 41 pages of memory available as our buffer, we observe that there is more than enough memory to run the two-phase hash join algorithm, so we try to keep one of the hash buckets of the smaller relation in memory to avoid writing it during the partitioning phase and then re-reading it during the probing phase. We also assume that all buckets have the same size.

Question 3 [20 points] Complexity of Query Optimization

Analyze the complexity of System R-style query optimization:

(1) [15 points] What is the complexity of dynamic programming for finding an optimal plan for an n-way join? Here, consider the total number of plans that the query optimizer enumerates and denote the complexity using the big-O notation.

(2) [5 points] What is the maximum number of plans stored in an intermediate pass?

Hint: consider the star join graph as shown below and analyze the complexity related to this graph. Consider System R-style query optimization, as used in PostgreSQL, for a query involving n tables and the following join relationships, where an edge denotes a join predicate between two tables:

Recall that System R-style query optimizer has the following features:

- It considers only left-deep plans.
- It avoids cross products whenever possible (i.e., considers first joins with predicates and then those without join predicates).
- It uses dynamic programming to find optimal subplans involving 1 table, 2 tables, …, up to n tables (which are referred to as pass 1, pass 2, …, pass n of the dynamic programming procedure).
**Question 4 [32 points]: Query Optimization in PostgreSQL**

In this exercise, we continue to consider the DBLP dataset that you have loaded into your PostgreSQL database in the previous assignment. Let us first recall the schema and some useful commands below.

**Schema.** This database consists of four tables: (1) an authors table, containing the names of authors, (2) the venue table, containing information about conferences or journals where papers are published, (3) the papers table, describing the papers themselves, (4) the paperauths table which indicates which authors wrote which papers. The schema is the following, where the underscore indicates the primary key of a relation.

```sql
authors (id: INTEGER, name: VARCHAR(200))

venue (id: INTEGER, name: VARCHAR(200) NOT NULL, year: INTEGER NOT NULL, school: VARCHAR(200), volume: VARCHAR(50), number: VARCHAR(50), type: INTEGER NOT NULL)

papers (id: INTEGER, name: VARCHAR NOT NULL, venue: INTEGER REFERENCES VENUE(id), pages: VARCHAR(50), url: VARCHAR);

paperauths (paperid: INTEGER, authid: INTEGER)
```

**Useful commands in postgresql:**

- `postgresql: \d`
- `postgresql: \di`
- `postgresql: \d table`
- `postgresql: \d+ table`
- `postgresql: ANALYZE table`

**Linux commands:**

Sometimes your commands may lead to long running queries (processes). To check or kill a background process, you can use the following Linux commands:

a) you can use command "ps" to see your process.

b) use command "kill" to kill your processes after you locate the process you want to kill.

Please see more details on [http://linfo.org/ps.html](http://linfo.org/ps.html) and [http://linfo.org/kill.html](http://linfo.org/kill.html).

For this question, you can create, load, clean the dblp data and create indexes on them using the following command after logging into your account:

```
bash /courses/cs600/cs645/cs645/dblp_hw2_sol/main.sh
```

**Note:** Read the SQL scripts to create tables / load tables / clean tables / build indexes which the above bash command executes. Please understand them before you execute them as they drop your previous tables for DBLP.

**A. System Catalog and Statistics**
As we learned in class, the query optimizer needs to estimate the number of rows retrieved by a query in order to make good choices of query plans.

One important type of the statistics is the total number of entries in each table and index, as well as the number of disk blocks occupied by each table and index. This information is kept in the table `pg_class`, in the columns `reltuples` and `relpages`. We can look at it with queries similar to this one:

```sql
select relname, relkind, reltuples, relpages
from pg_class
where relname LIKE 'author%';
```

When a table is large, the number of tuples in the `pg_class` may differ slightly from the actual number of tuples (which you can compute using "Select count(*) From table_name"). For large tables, `ANALYZE` takes a random sample of the table contents, rather than examining every row. This allows even very large tables to be analyzed in a small amount of time.

The view `pg_stats` provides additional information about each attribute. Of particular importance are:
- `avg_width`: the average width of an attribute in bytes, which is useful for variable length attributes.
- `n_distinct`: the number of distinct values in an attribute. If greater than zero, it is the estimated number of distinct values in the column. If it is -1, then this attribute is declared to be unique so it has a distinct value in each row.
- `histogram_bounds`: the histogram bounds that divide the column's values into groups of approximately equal population.

```sql
select tablename, attname, avg_width, n_distinct, histogram_bounds
from pg_stats
where tablename like 'author%';
```

If your `pg_stats` table is empty, please run “`analyze;`” command in PostgreSQL, which will ask the optimizer to collect statistics from the source tables.

**B. Query Plans**

We use the `explain` command to see the query plan used by PostgreSQL:

```sql
explain select * from authors where name = 'David J. DeWitt';
```

```
QUERY PLAN
-----------------------------------------------------------------------------
Index Scan using authors_name on authors  (cost=0.43..8.45 rows=1 width=19)
  Index Cond: ((name)::text = 'David J. DeWitt '::text)
```

This query plan uses an index scan based on authors.name. Its estimated costs are in units of disk page fetches, between 0.43 (time expended before output scan can start) and 8.45 (total cost). Furthermore, the output is expected to contain 1 answer of width 19 in bytes.

```sql
explain
select paperid, authid
from paperauths, authors
where authors.id=authid;
```

```
Hash Join  (cost=55249.15..396917.25 rows=8997964 width=8)
  Hash Cond: (paperauths.authid = authors.id)
    ->  Seq Scan on paperauths  (cost=0.00..129793.64 rows=8997964 width=8)
```

This query plan uses an hash join based on paperauths.authid and authors.id. Its estimated costs are in units of disk page fetches, between 0.00 (time expended before output scan can start) and 129793.64 (total cost). Furthermore, the output is expected to contain 8,997,964 answers of width 8 in bytes.
C. Explaining query Plans

In this exercise, you are asked to extract and explain the query plans for the following queries.

(1) [10 Points] Consider the following query:

```sql
select * from authors
where name < 'David J. DeWitt';
```

(a) What is the query plan that PostgreSQL uses for this query?

(b) What is the estimated number of rows? 

What is the actual number of rows returned by the query?

Please use "explain analyze" command, which shows for each operator, the estimated cost and output size of the optimizer, followed by the actual cost and output size by executing the query.

```sql
explain analyze
select * from authors
where name < 'David J. DeWitt';
```

(c) If the estimated number of rows differs from the actual number of rows returned, briefly explain, using a simple formula and within 5 lines of text, how the query optimizer estimated the number of output rows.

Hint: Use pg_class and pg_stats to look for the statistics about the authors relation and the name attribute. In particular, pg_stats returns the histogram of each column.

(2) [10 Points] Now let us consider the original, simpler query:

```sql
select * from authors
where name = 'David J. DeWitt';
```
(a) What is the query plan that PostgreSQL uses for this query?

(b) Briefly explain, within 5 lines of text, why the plan chosen from PostgreSQL for this query differs from that for the previous query.

(3) [12 Points] Consider the next query:

```
SELECT *
FROM papers p, authors a, paperauths pa
WHERE pa.paperid = p.id AND pa.authid = a.id
   AND a.name = 'David J. DeWitt';
```

(a) What is the query plan that PostgreSQL uses for this query?

(b) List the operators used in the query plan, from bottom up. For each join operation, name the algorithm (hash, scan, index scan) used on each input relation.

(c) What is the estimated number of returned rows? __________________________________________

What is the actual number of rows returned by the query? ____________________________________

(d) Briefly explain, with a simple formula and within 5 lines of text, why the estimated number of rows differs from the actual number of rows returned.