Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.830 Database Systems: Fall 2016 Quiz I

There are 14 questions and 15 pages in this quiz booklet. To receive credit for a question, answer it according to the instructions given. *You can receive partial credit on questions.* You have **80 minutes** to answer the questions.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

Some questions may be harder than others. Attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.
LAPTOPS MAY BE USED; NO PHONES OR INTERNET ALLOWED.

Do not write in the boxes below

1-3 (xx/26)	4-7 (xx/36)	8-10 (xx/19)	11-14 (xx/19)	Total (xx/100)

Name: Solutions

I SQL Query

Consider the following schema for a courses database:

- department(did, dname, location)
- student(sid, sname, did, age)
- course(cid, cname, time, room)
- enrolled(sid, cid)

1. [8 points]: Which of the following SQL queries will count the number of departments with no students taking the course 'Databases'.

(Circle all that apply.)

A.
```
SELECT COUNT(d.did)
FROM department d
WHERE d.did NOT IN (
    SELECT s.did
    FROM student s
    WHERE s.sid IN (
        SELECT e.sid
        FROM enrolled e, course c
        WHERE e.cid = c.cid AND c.cname = 'Databases'
    )
);
```

B.
```
SELECT COUNT(DISTINCT s.did)
FROM student s
WHERE s.sid NOT IN (
    SELECT e.sid
    FROM enrolled e, course c
    WHERE e.cid = c.cid AND c.cname = 'Databases'
);
```

C.
```
SELECT COUNT(DISTINCT d.did)
FROM department d
WHERE d.did NOT IN (
    SELECT s.did
    FROM enrolled e, course c, student s
    WHERE e.cid = c.cid AND c.cname = 'Databases' AND e.sid = s.sid
);
```

D.
```
SELECT COUNT(d.did)
FROM department d, student s, course c, enrolled e
WHERE e.cid = c.cid AND c.cname='Database' AND
      e.sid != s.sid AND d.did = s.did;
```

Answer: *A and C. B is incorrect because it counts the number of departments with students not taking the course 'Databases'.*

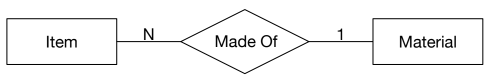
Name:

II Entity and Schema Design

Suppose you are designing a schema to record information about novels (the Internet Novel Database, or INDb). Your database needs to record the following information:

- For each novel, its title, publication date, number of pages, abstract.
- For each writer, his/her name, date of birth, short biography. A writer may write multiple novels. Each novel is written by exactly one writer.
- For each user, his/her username, password, and avatar. A user may rate multiple novels, and an novel may be rated by multiple users. Each rating has a score of 0 to 100.

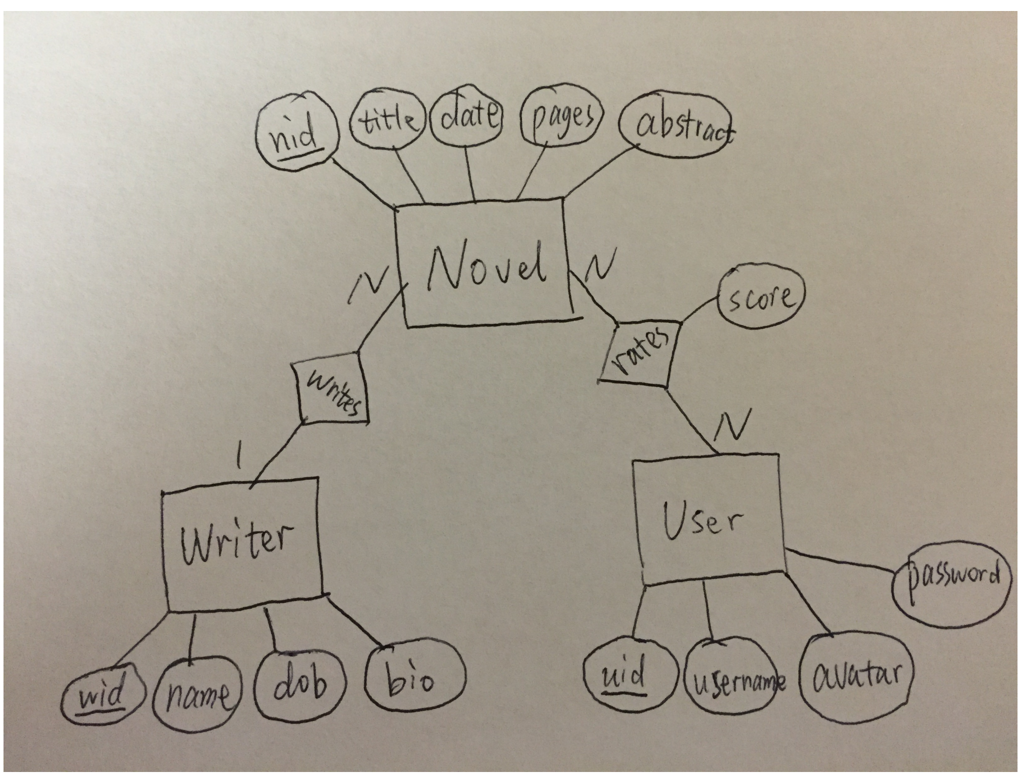
2. [10 points]: Draw an entity relationship diagram for this database. Please draw entities as squares, attributes as ovals, and denote relationships as diamonds between pairs of entities. Label each edge with a "1" or an "N" to indicate whether the entity on the other side of the relationship connects to 1 or N entities on this side of the relationship. For example, the following would indicate that each item is made of a single material (e.g. wood, metal), and multiple items can be made of the same material:



Give each entity, relationship, and attribute a name.

(Draw your diagram in the space below)

.



Name:

3. [8 points]: Use your ER diagram to determine a relational schema for this database. For each table, use the form:

TablenameN (field1-name, ..., fieldn-name)

to denote its schema. If you wish, you can create an additional field for each table to serve as a unique identifier. Underline the primary keys and use the "... references ..." syntax for foreign keys in your schema.

(Write your answer in the space below.)

Answer: *novel(<u>nid</u>, title, date, pages, abstract, wid references writer.wid)*

writer(<u>wid</u>, name, dob, bio)

user(<u>uid</u>, username, password, avatar)

rates(<u>uid</u> references user.uid, <u>wid</u> references writer.wid, score)

III Indexing

4. [6 points]: You are building an index selection tool that measures a numeric benefit for a set of indexes on a query workload W. Larger benefits are better. Suppose that index 1 covers attributes (a,b), and has benefit 10 on W, index 2 covers attributes (a,b,c) and has benefit 12 on W, and index 3 covers attribute (a,c) and has benefit 7 on W.

If you have sufficient storage to create two indexes, which would you recommend, and why?
(Choose one of the following.)

A. 1,2

B. 2,3

C. 1,3

Justify your answer: **Answer:** *B is the correct choice. Index 2 has the most most overall benefit, so it makes sense to choose it. Adding Index 1 to this index is unlikely to add much additional benefit, since Index 2 completely covers Index 1 (i.e., any query that can benefit from Index 1 can also benefit from Index 2.) Adding Index 3 to Index 2 can help queries that have predicates on a and c but not b.*

IV Joins and Optimization

5. [12 points]:

Consider the problem of joining three tables, A, B, and C via the query:

```
SELECT *
FROM   A,B,C
WHERE  A.a = B.b
AND    C.c = A.b
AND    B.d = C.e
```

Suppose that $|A| > |B| > |C|$, there are no indexes, none of the tables fit in memory, and $|A.a \bowtie B.b| > |A.b \bowtie C.c| > |B.d \bowtie C.e|$ (here $|A|$ refers to number of records in $|A|$, and each record of each table is the same size). Each join is a key-foreign key equality join.

If you know nothing about the size of the final join, which of the following statements about these joins are true? For "bushy" plans like $(A \bowtie C) \bowtie (B \bowtie C)$, assume that the right hand plan is materialized as a table on disk before being used in the top-most join.

(Circle 'T' or 'F' for each choice.)

T F $(A \bowtie B) \bowtie C$ will definitely not be faster than $(B \bowtie C) \bowtie A$

Answer: *True. Given that tables do not fit into memory and there are no indexes, it's reasonable to assume that the joins will be done using grace hash. The cost of* $(A \bowtie B)$ *is* $3(|A| + |B|)$*; the cost of joining this to* $|C|$ *via grace hash is* $2|A \bowtie B|$ *(since we have to write it out and read it in)* + $3|C|$*, for a total cost of* $3(|A| + |B|) + 2|A \bowtie B| + 3|C|$.

Compare this to the cost of $(B \bowtie C) \bowtie A$*. The cost of this is* $3(|B| + |C|) + 2|B \bowtie C| + 3|A|$.

Subtracting these two from each other, we are left with $2|A \bowtie B|$ *in the first case and* $2|B \bowtie C|$ *in the second. Since* $|A.a \bowtie B.b| > |B.d \bowtie C.e|$*, there is no way that* $(A \bowtie B) \bowtie C$ *can be faster.*

It's possible that the 2nd join could be done via an in-memory hash (if the intermediate results fit into memory), but since $|A.a \bowtie B.b| > |B.d \bowtie C.e|$*, there's no way for* $|A.a \bowtie B.b|$ *to fit into memory and* $|B.d \bowtie C.e|$ *to not fit.*

T F $(A \bowtie C) \bowtie B$ will definitely not be faster than $(B \bowtie C) \bowtie A$

Answer: *True. A similar analysis as in the previous problem shows that* $(A \bowtie C) \bowtie B$ *can't be faster.*

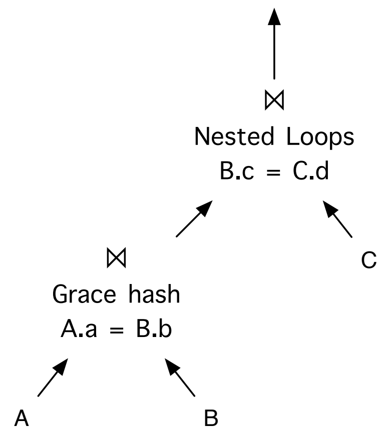
T F $(A \bowtie C) \bowtie (B \bowtie C)$ will definitely be faster than $(A \bowtie B) \bowtie C$

Answer: *False. We can't reason about the relative cost of the materialization step and the cost of an additional pass over C in the bushy plan vs the fact that the join is over two smaller tables.*

T F $(A \bowtie C) \bowtie (B \bowtie C)$ could be faster than $(B \bowtie C) \bowtie A$

Answer: *True. It could be that the join* $(A \bowtie C)$ *has many fewer tuples than A, which could mean that it is cheaper to perform the bushy join.*

6. [10 points]: Consider the following physical query plan generated by a query optimizer with support for nested loops, index nested loops, grace hash, and blocked hash join.



Given what you know about join algorithms and optimizers, which of the following statements are most likely true, assuming the optimizer is able to accurately estimate statistics (e.g., cardinalities and selectivities of tables and intermediate results.)

(Circle 'T' or 'F' for each choice.)

T F Neither A nor B fits into memory

Answer: *True. If A or B fit into memory a Grace hash would not make sense.*

T F C does not fit into memory

Answer: *False. An NL join would be a poor choice if C were large.*

T F The cardinality of (A join B) is equal to or smaller than the cardinality of (B join C)

Answer: *True. Generally it is better to order joins from lowest to highest cardinality. It may seem contradictory that C is small and the cardinality of B join C is large, but this could occur if each tuple in B joins with several tuples in C.*

T F The cardinality of C is very large

Answer: *False. An NL join would be a poor choice if C were large.*

T F None of the tables have indexes on them, since index nested loops joins weren't used

Answer: *False. Index nested loops joins are often not the fastest way to compute a join, especially if indexes are not clustered.*

7. [8 points]: Suppose the Selinger optimizer chooses to join four tables A,B,C, and D in the order

```
((A sort-merge B) sort-merge C) sort-merge D)
```

Here the query contains join predicates between all pairs of tables (so that the optimizer might have generated any left-deep ordering of these predicates.) Assume sort-merge can spill to disk (i.e., doesn't require the tables to fit into memory).

You look at the optimizer and find that it estimates that the number of records output by A join B is more than the number of records output by A join C. Give two reasons why it might have produced the plan above:

(Write your answers below.)

Reason 1: **Answer:** *A sort-merge B could produce results in an "interesting order" for the join of C, or (A sort-merge B) sort-merge C could produce results in an interesting order for the join with D.*

Reason 2: **Answer:** *If C fits into memory and A and B do not, then A sort-merge C would need to spill to disk, and (A sort-merge C) sort-merge B would need to spill to disk. However, although A sort-merge B would also need to spill, it is possible (if A-sort-merge B produces few results) that (A sort-merge B) sort-merge C would not need to spill.*

Another possible answer is that, just because the size of the output of A join B is bigger than A join C, it doesn't mean that A join B actually does more work. In particular, if A and B fit into memory and C does not, but A join B produces a lot of results, it could be cheaper to first do an in-memory join of A and B, and then do a join with C that spills to disk.

Finally, it is possible that there are clustered indexes on the join attributes of A and B, which would allow it to avoid sorting for this first join.

V Column-Stores

Consider three tables T1, T2, and T3, each of which has 10 columns (a through j), each of which is 4 bytes wide. T1.a, T2.a, and T3.a are the primary keys of each table, respectively. You are running the query:

```
SELECT AVG(T1.e)
FROM T1, T2, T3
WHERE T1.b = T2.a
AND T1.c = T3.a
AND T2.f > 15
AND T3.g > 50
```

Here attributes T1.e, T2.f, and T3.g are uniformly distributed integers between 0 and 100, foreign keys values (T1.b, T1.c) are selected uniformly and at random, each table is 100 GB on disk (2.5 billion records), and you are running in a row-store with 10 GB of memory.

8. [4 points]: In what order should these joins be run:

(Circle the best ordering)

A. T1.b = T2.a then T1.c = T3.a

B. T1.c = T3.a then T1.b = T2.a

Answer: *B is a better choice, because it allows the second join to process fewer tuples (since the result of the first join is smaller due to the more selective filter).*

9. [6 points]: For each join, considering your chosen join order, and choosing amongst grace hash, simple hash, in-memory hash, or nested loops, which join algorithm would you recommend for each join:

(Write the join algorithm for each join.)

A. T1.b = T2.a Algorithm: ___________________________

Answer: *Simple hash. See Figure 1 – there is not quite enough space for the needed columns to fit into memory, but they very nearly do, so a simple hash is the best answer.*

B. T1.c = T3.a Algorithm: ___________________________

Answer: *In memory hash. See Figure 1 – T3.a can fit into an in-memory hash table, which can be probed with the records of T1. The key observation is that projections can be pushed down into the plan, and that the columns that are needed can fit into memory.*

Name:

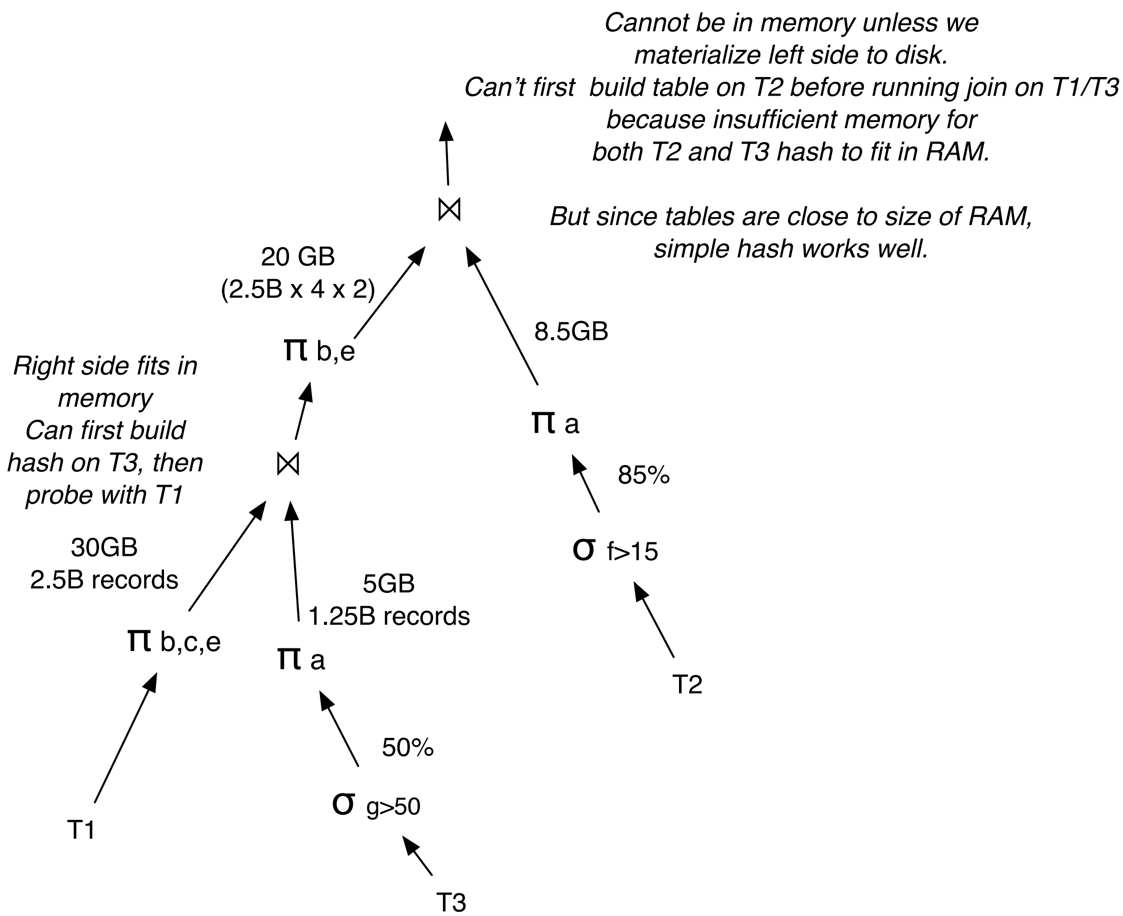


Figure 1: Diagram indicating costs and cardinalities for previous problem.

Name:

10. [9 points]: Now suppose you switch to a column-oriented database like C-Store. If the query on the previous page is the only query being run, and you can choose to sort each table's columns on a set of attributes (e.g., a then b then c), which sort order would you recommend for each table, and why? Assume that system is employing late-materialization, and can compress columns using either Lempel-Ziv (e.g., zip) or run-length encoding.

(Choose a sort order for each table.)

A. Sort order for T1: **Answer:** *e*

B. Sort order for T2: **Answer:** *f*

C. Sort order for T3: **Answer:** *g*

Justification for choice of above orders?

Answer: *The key observation is that by sorting on e, f, and g, and RLE encoding these columns, they can be made dramatically smaller (800 bytes vs 10 GB.) This large savings in I/O is almost certainly the best thing that can be done here.*

If you assume sort-merge join is available (which it was not in the previous question, but this was somewhat ambiguous), you can get some performance benefit the joins, i.e., by sorting T1 on c and T3 on a, both sorts can be avoided, at the cost of losing the I/O reduction from RLE (since there is no benefit to RLE encoding e/f/g when these are secondary sort orders, because there are very few repeated values of a/c). Similarly, by sorting T2 on a, you can avoid the sort cost for T2, again losing the I/O benefit (but will still have to sort the output of the T1/T2 join).

Which of these approaches is better depends on the specific hardware configuration, so credit was given for both; to receive full credit answers needed to make clear that both alternatives were considered.

VI Query Planning

The following tables in Postgres are from TPC-H, a decision support benchmark.

```
               Table "public.lineitem"
     Column      |         Type          | Modifiers
-----------------+-----------------------+-----------
 l_orderkey      | bigint                | not null
 l_partkey       | bigint                | not null
 l_suppkey       | bigint                | not null
 l_linenumber    | integer               | not null
 l_quantity      | numeric               |
 l_extendedprice | numeric               |
 l_discount      | numeric               |
 l_tax           | numeric               |
 l_returnflag    | character(1)          |
 l_linestatus    | character(1)          |
 l_shipdate      | date                  |
 l_commitdate    | date                  |
 l_receiptdate   | date                  |
 l_shipinstruct  | character(25)         |
 l_shipmode      | character(10)         |
 l_comment       | character varying(44) |
Indexes:
    "lineitem_pkey" PRIMARY KEY, btree (l_orderkey, l_linenumber)
    "idx_lineitem_commitdate" btree (l_commitdate)
    "idx_lineitem_partkey" btree (l_partkey)
    "idx_lineitem_receiptdate" btree (l_receiptdate)
    "idx_lineitem_shipdate" btree (l_shipdate)
    "idx_lineitem_suppkey" btree (l_suppkey)

               Table "public.orders"
     Column      |         Type          | Modifiers
-----------------+-----------------------+-----------
 o_orderkey      | integer               | not null
 o_custkey       | bigint                | not null
 o_orderstatus   | character(1)          |
 o_totalprice    | numeric               |
 o_orderdate     | date                  |
 o_orderpriority | character(15)         |
 o_clerk         | character(15)         |
 o_shippriority  | integer               |
 o_comment       | character varying(79) |
Indexes:
    "orders_pkey" PRIMARY KEY, btree (o_orderkey)
    "idx_orders_custkey" btree (o_custkey)
```

We have run VACUUM FULL ANALYZE on all of the tables in the database, which means that all of the statistics used by PostgreSQL server should be up to date.

Name:

Consider the following query and two query plans produced by the EXPLAIN command for it:

```
SELECT l_shipmode,
       SUM(CASE
             WHEN o_orderpriority = '1-URGENT'
                   OR o_orderpriority = '2-HIGH' THEN 1
             ELSE 0
           END) AS high_line_count,
       SUM(CASE
             WHEN o_orderpriority <> '1-URGENT'
                  AND o_orderpriority <> '2-HIGH' THEN 1
             ELSE 0
           END) AS low_line_count
FROM   orders,
       lineitem
WHERE  o_orderkey = l_orderkey
       AND l_shipmode IN ( 'TRUCK', 'AIR' )
       AND l_commitdate < l_receiptdate
       AND l_shipdate < l_commitdate
       AND l_receiptdate >= DATE '1995-01-01'
       AND l_receiptdate < DATE '1995-01-01' + interval '1' month
GROUP  BY l_shipmode
ORDER  BY l_shipmode;

                                   QUERY PLAN 1
----------------------------------------------------------------------------------------------
 Sort  (cost=266445.78..266445.78 rows=1 width=27)
   Sort Key: lineitem.l_shipmode
   ->  HashAggregate  (cost=266445.76..266445.77 rows=1 width=27)
         Group Key: lineitem.l_shipmode
         ->  Nested Loop  (cost=0.86..266400.34 rows=2595 width=27)
               ->  Index Scan using idx_lineitem_receiptdate on lineitem
                         (cost=0.43..247401.13 rows=2595 width=19)
                     Index Cond: ((l_receiptdate >= '1995-01-01'::date) AND
                                  (l_receiptdate < '1995-02-01 00:00:00'::timestamp without time zone))
                     Filter: ((l_shipmode = ANY ('{TRUCK,AIR}'::bpchar[])) AND
                              (l_commitdate < l_receiptdate) AND (l_shipdate < l_commitdate))
               ->  Index Scan using orders_pkey on orders
                         (cost=0.43..7.31 rows=1 width=20)
                     Index Cond: (o_orderkey = lineitem.l_orderkey)

                                   QUERY PLAN 2
----------------------------------------------------------------------------------------------
 Sort  (cost=317988.67..317988.67 rows=1 width=27)
   Sort Key: lineitem.l_shipmode
   ->  HashAggregate  (cost=317988.65..317988.66 rows=1 width=27)
         Group Key: lineitem.l_shipmode
         ->  Hash Join  (cost=61377.00..317943.24 rows=2595 width=27)
               Hash Cond: (lineitem.l_orderkey = orders.o_orderkey)
               ->  Seq Scan on lineitem  (cost=0.00..256514.34 rows=2595 width=19)
                     Filter: ((l_shipmode = ANY ('{TRUCK,AIR}'::bpchar[])) AND
                              (l_commitdate < l_receiptdate) AND
                              (l_shipdate < l_commitdate) AND
                              (l_receiptdate >= '1995-01-01'::date) AND
                              (l_receiptdate < '1995-02-01 00:00:00'::timestamp without time zone))
               ->  Hash  (cost=42627.00..42627.00 rows=1500000 width=20)
                     ->  Seq Scan on orders  (cost=0.00..42627.00 rows=1500000 width=20)
```

Name:

11. [2 points]: Based on the above EXPLAIN output, which of these two plans will Postgres choose, and why?

(Circle the best plan.)

Plan 1 Plan 2

Justification:

Answer: *Plan 1. From Postgres optimizer's perspective, the cost of Plan 1 is 266445.78. and the cost of Plan 2 is 317988.67.*

12. [2 points]: What type of join is Postgres using in Plan 1?

(Write your answer in the space below.)

Answer: *Index nested loop join.*

13. [5 points]: What is the estimated selectivity of the predicate `l_orderkey = o_orderkey`, in terms of the fraction of the orders table it selects?

(Write your answer in the space below.)

Answer: *Three possible answers were accepted:*

A. *2595/1500000. It's the ratio of the cardinality of output and orders.*

B. *1/1500000. For each tuple in lineitem, it only matches with one tuple in orders.*

C. *1. The cardinality of output is 2595 and only 2595 tuples are read in orders.*

Now consider the following query and the query plan output from the EXPLAIN command:

```
SELECT l_orderkey,
       SUM(l_extendedprice * ( 1 - l_discount )) AS revenue,
       o_orderdate,
       o_shippriority
FROM   orders,
       lineitem
WHERE
       l_orderkey = o_orderkey
       AND o_orderdate < DATE '1995-02-01'
GROUP  BY l_orderkey,
          o_orderdate,
          o_shippriority;
```

```
                                    QUERY PLAN
-----------------------------------------------------------------------------------
 HashAggregate  (cost=358998.96..393962.93 rows=2797118 width=28)
   Group Key: lineitem.l_orderkey, orders.o_orderdate, orders.o_shippriority
   ->  Hash Join  (cost=55061.22..317042.19 rows=2797118 width=28)
         Hash Cond: (lineitem.l_orderkey = orders.o_orderkey)
         ->  Seq Scan on lineitem  (cost=0.00..181499.15 rows=6001215 width=20)
         ->  Hash  (cost=46322.00..46322.00 rows=699138 width=12)
               ->  Seq Scan on orders  (cost=0.00..46322.00 rows=699138 width=12)
                     Filter: (o_orderdate < '1995-02-01'::date)
(8 rows)
```

14. [10 points]: When we create a B+Tree on `orders(o_orderdate)`, we find that Postgres still chooses a sequential scan instead of an index scan on orders. Which of these explains why this might be the case.

(Circle 'T' or 'F' for each choice.)

T F The predicate on `o_orderdate` is not selective enough.
Answer: *True. If the predicate is not selective, there will be more random I/Os to the order table, which Postgres estimates will be slower than a sequential scan.*

T F The predicate on `o_orderdate` is too selective.
Answer: *False.*

T F Using the index will require using an index nested loops join, resulting in more passes over the lineitem table.
Answer: *False. lineitem is only scanned once.*

T F Using the index will result in more random I/Os to the order table, which Postgres estimates will be slower than a sequential scan of the orders table.
Answer: *True.*

T F The index on `orders(o_orderdate)` is a clustered index.
Answer: *False. Using a clustered index scan can result in a sequential scan.*

Name: