CS 245 Midterm Test
Winter 2021

- Please read all instructions carefully. In the case of any ambiguity, state any assumptions you made in your answer. You may also ask clarifying questions in a **private** question on Piazza, but note that the course staff may take several hours to answer these.

- There are five problems, some with multiple parts, for a total of 72 points. You have until 11:59 PM Pacific on February 16th to work on the test.

- The test is open-book, but you are not allowed to communicate with other people to do it. This includes asking public questions on Piazza -- please only ask private questions about the test or any content covered in it. You may also use the Internet during the test, but keep in mind that many online resources might use terms differently from our course, and that directly copying an online resource is considered plagiarism and is not allowed. We don't think you will need resources other than the course materials.

- You may complete this test digitally (e.g. using the Annotate tools in Acrobat Reader), or print it, handwrite your answers legibly, and scan it using a scanner or a mobile app such as GeniusScan. In either case, ensure that the file you upload to Gradescope exactly matches the format of this document and is sharp, legible, and aligned.

- Solutions will be graded on correctness and clarity. For the long-answer problems, please show your intermediate work. Each problem has a relatively simple to explain solution, and we may deduct points if your solution is much more complex than necessary. Partial solutions will be graded for partial credit.

NAME: ____

SUID: ____

In accordance with both the letter and spirit of the Stanford Honor Code, I have neither given nor received assistance on this test. A typed signature is fine.

SIGNATURE: ____

Problem 1: System R (14 points)

a) (4 points) From Phase 0 to Phase 1, System R changed its storage design from XRM "inversions" to B-trees. For a table with 1000 tuples each with 10 fields, which of the following metrics improved in Phase 1? *(Check all that apply.)*

- ❑ Time for a range query including ~500 rows
- ❑ Number of I/O operations in a range query including ~500 rows
- ❑ Number of I/O operations in a query for a single row
- ❑ Table storage space

1, 2, 3. B-trees reduce the number of I/Os in large range query (2) and also time (1). Even though (3) has an inversion in Phase 0, there are still more I/Os to follow pointers for each field than the B-tree depth. B-tree data layout increases (4) since it doesn't use pointers to data.

b) (6 points) For each of the following recovery features of System R, describe a failure in which the feature is ***NOT*** required to return the system to a consistent state. Use each failure type (disk/media, system, transaction) once. *Briefly explain* how the system would recover instead.

System failure - "For recovery from system failures, System R uses the change log mentioned above plus something called 'shadow pages.'"
Media failure - "The database is restored using the latest image dump and the recovery process reapplies all database changes as specified on the log for completed transactions."
Transaction failure - "System R simply processes the change log backwards removing all changes made by the transaction."

i. Table on the backup disk.

System failure OR transaction failure OR media failure of backup.

ii. Change log on the main disk.

Media failure of main disk OR system failure of main memory OR transaction failure that uses in-memory of backup change log.

iii. Shadow pages.

Transaction failure OR media failure. We also accepted system failures with a note on how the change log makes shadow pages redundant.

c) (4 points) David and Patrick recently opened a small apothecary. They set up a System R database to track their items and customers' purchases. The database has the following tables:

Items

id	**name**	**price**
1	earl grey	4.00
2	oolong	5.00
3	green	3.50

Purchases

item_id	**sale_price**	**timestamp**
2	5.00	2021-02-11 08:43:20
3	4.50	2021-02-11 08:50:59
...		

i. David and Patrick often run the following SQL query to record purchases:

```
INSERT INTO Purchases (item_id, sale_price, timestamp) VALUES (<item_id>,
(SELECT price FROM Items WHERE id = <item_id>), CURRENT_TIMESTAMP);
```

In System R's hierarchical locking scheme, what is the minimum set of objects that need to be locked to execute this query?

Lock on Items row and Purchases table.

ii. Give an example of an analytical SQL query (just a description is fine). How could this information be used in a future business decision? *(1-2 sentences)*

Open-ended: Find the least popular tea based on the number of items sold to put on sale. Find the tea that generated the most revenue in the month of January to know what to restock next year.

Problem 2: Row and Column Stores (14 points)

Consider a table with 4 columns and 100,000 rows. The columns are an integer, variable length string, integer, and integer *in that order* (all integers are **8 bytes**). The average string length is 20 bytes. We assume that the values in columns 0, 2, and 3 are uniformly distributed between [0, 1024), and that cache lines are 64 bytes. For each question, answer assuming the average case.

Please read what the question is asking for carefully. Show your work for partial credit.

We will use the following queries.

Query 1:
SELECT SUM(col0) + SUM(col2) FROM table

Query 2:
SELECT SUM(col0) + SUM(col2) FROM table
WHERE col0 < 256

a) **(4 points)** Suppose the table is stored in memory as a **row store** without padding. How many *cache lines* must be accessed to compute query 1?

Each row fits in a cache line (on average) so every row must be accessed. The size in memory is:
(8 * 3 + 20) * 100000 = 4400000 bytes
Which corresponds to 68750 cache lines.

66,667 is not an acceptable answer. Although we can ignore the 4 bytes that split a cache line, the next cache line must be fetched to read the next row.

b) **(2 points)** Suppose the table is stored in memory as a **column store** without compression. How many *cache lines* must be accessed to compute query 1?

Only columns 0 and 2 must be accessed for query 1. The engine can simply scan columns 0 and 2:
2 * (8 * 100000 / 64) = 25000 cache lines

c) **(4 points)** Suppose the table is stored in memory as a **column store** without compression and the primary key is column 0 (assume the table is sorted by the primary key). How many *cache lines* must be accessed to compute query 2? **State your assumptions when answering this question.**

The DB knows that column 0 is sorted, so the DB can scan columns 0 and 2 until 256 is achieved. Because we assume that the data is uniformly distributed, this corresponds to 1/4 of the rows. This corresponds to
((8 * 100000) / 4) / 64 = 3125 cache lines
Per column, so a total of
3125 * 2 = 6250 cache lines

We will accept off-by-one depending on the assumptions.

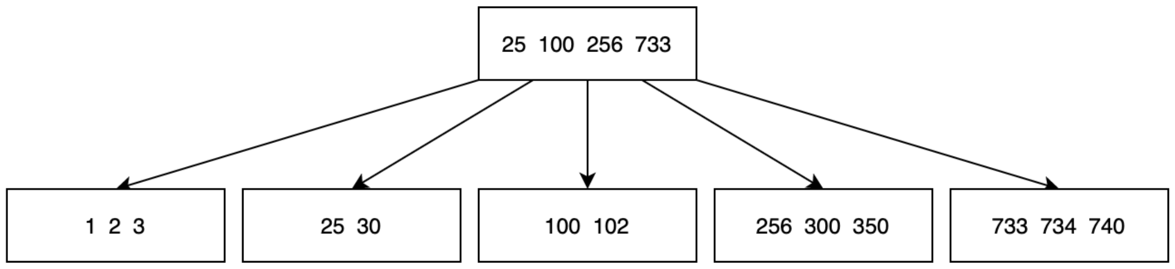
d) **(4 points)** Suppose the table is stored *on disk* as a **row store** where each block is 1024 bytes and the data is stored without padding. Suppose the primary key is column 0, the table is sorted by the primary key, and we have an in-memory index on column 0. How many *disk blocks* must be accessed to answer query 2?

The index is stored in memory, so we can ignore accessing the index for the purpose of the question. As with part c, we only need to access 1/4 of the table:
(8 * 3 + 20) * 100000 / 4 = 1100000 bytes
Converting this to blocks gives:
ceil(1100000 / 1024) = 1075 blocks

If stated, we also accepted answers where records cannot be split across blocks (23 records / block) => 1087 blocks

Problem 3: Indexes (14 points)

a) (6 points) Considering the following B+ tree:



The order *n* of the B+ tree is 4. Follow the same rules taught in class of the maximum and minimum pointers or keys of the tree.

Insert elements in the following order:

INSERT(1000), INSERT(301), INSERT(735)

During all the operations, there are ________ leaf overflows and ________ non-leaf overflows. After all the operations, the root node contains _____ keys.

Answer: 1 1 1

Explanation: The first and second insertion add one key in the leftmost and second-leftmost leaf node. No overflow happens till now. The third insertion tries to insert into the leftmost leaf node once again. The number of keys of that node will be 5 > 4, and one leaf overflow happens. The current root node should insert a key and a pointer to this newly split leaf node, but the keys would be 5 > 4, so one non-leaf overflow happens. A new root with 1 key and 2 pointers will be created.

b) (2 points) Assume there are M records for B+ tree to index. In the first case, there are N separate exact search queries on this index. In the second case, there is a range query on this index with results spanning N leaf nodes. How many pointers in the B+ tree will these two cases traverse separately to retrieve the results? The first case: __b__, the second case: __c__.
(Fill in one of a, b, c or d for each case).

(a) Θ(MN) (b) Θ(N log M) (c) Θ(N + log M) (d) Θ(M + log N)

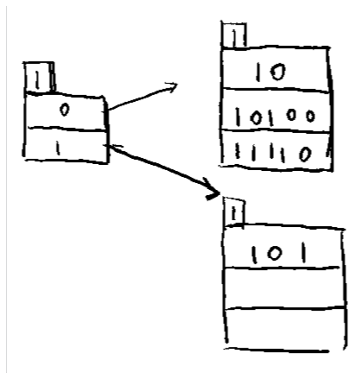
c) (6 points) Consider an extendible hashing index that uses the **last i bits** (instead of the first i bits taught in class) of b bits output by hash function to map it to a bucket. Assume each bucket can hold at most 3 keys and the original directory is empty.
Insert hashed keys in the following order:

2, 5, 20, 30, 18, 10

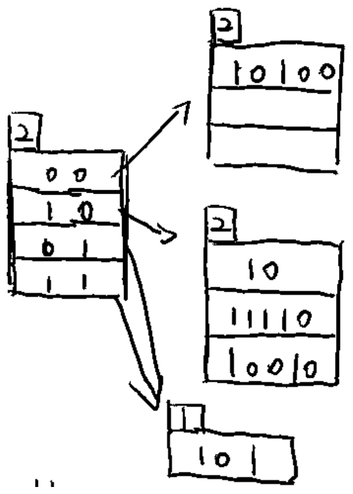
After the insertion,

i. The global depth of the directory is ___.
ii. The local depth of the bucket containing 5 is ___.
iii. The local depth of the bucket containing 20 is ___.
iv. The local depth of the bucket containing 10 is ___.

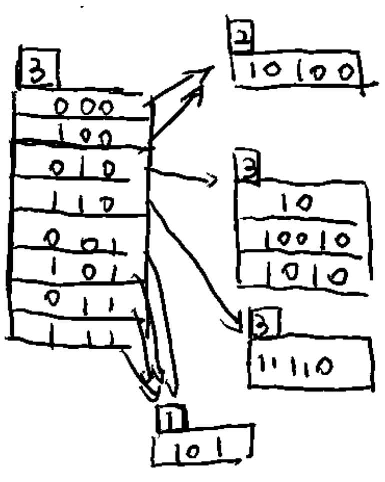
Answer: 3, 1, 2, 3
Explanation:
The binary of these numbers:
2 (10), 5 (101), 20 (10100), 30 (11110), 18 (10010), 10 (1010)
After inserting 2, 5, 20, 30, the directory looks like (a). After inserting 18, it looks like (b).
Finally, after inserting 10, it looks like (c).



(a)



(b)



(c)

Problem 4: Compression and Query Execution (14 points)

a) (3 points) For each of these following statements about compression, circle whether it is true or false. If it is false, briefly explain why.

i. Data must be decompressed to perform natural joins even if the data and the predicate are compressed in the same way:

TRUE [✓ FALSE]

In Section 2 of the c-store compression paper, they say, "exact-match comparisons and natural joins can be performed directly on compressed data if the constant portion of the predicate is compressed in the same way as the data." Intuitively, you can hash the predicate value and compare that against the compressed values in the database.

ii. Null suppression techniques apply to both row-store and column-store databases:

[✓ TRUE] FALSE

iii. Run-length encoding is not useful for row-store databases:

TRUE [✓ FALSE]

RLE can be useful for row-stores to compress large string attributes that have many blanks or repeated characters, as mentioned in section 4.3 of the c-store compression paper.

b) (3 points) Suppose we are applying dictionary encoding to a column with 64 unique values. How many values can fit into the following number of bytes?

i. 2 bytes = ___2___ values

ii. 3 bytes = ___4___ values

iii. 6 bytes = ___8___ values

c) (8 points) Coleman Media Inc. has decided on the following table to record TV viewership. They plan to store the data in C-store, but they need your expertise on compression schemes!

Title	**Channel**	**StartTime**	**Viewers**	**Date**	**DayOfWeek**
Log & Order	245	9:00 PM	2,718,281	3/14/2020	Saturday
Breaking DAG	246	7:00 PM	16,180,340	3/14/2020	Saturday

The table currently has 1,000,000 rows. There are 1,000 channels running 24 hours a day for 7 days a week. Each channel can only play 1 TV program at a time. Each program is an hour-long and shows only start at the beginning of each hour. The number of viewers varies significantly with a peak during prime time hours like the rows above to as little as a few hundred people for the early morning hours.

i. Between **null suppression** and **bit-vector encoding**, which compression scheme would be a better fit for the **DayOfWeek** column? *Please explain your reasoning and any assumptions.*

Bit-vector encoding. There are only 7 distinct values and they are all similar lengths

ii. Between **dictionary encoding** and **null suppression**, which compression scheme would be a better fit for the **Viewers** column? *Please explain your reasoning and any assumptions.*

Null suppression. There are an unknown number of unique values and the values are heavily skewed. While the prime time shows will have a large number of views, the majority of rows will have small viewership values, so null suppression can remove a lot of leading zeros.

iii. Between **dictionary encoding** and **run-length encoding**, which compression scheme would be a better fit for the **StartTime** column in a projection **sorted by Date and StartTime**? *Please explain your reasoning and any assumptions.*

Run-length encoding. The projection is sorted by StartTime so there will be 1,000 rows for a given StartTime value, one for each channel.

iv. Between **dictionary encoding** and **run-length encoding**, which compression scheme would be a better fit for the **StartTime** column in a projection **sorted by Date, Channel, and StartTime**? *Please explain your reasoning and any assumptions.*

Dictionary encoding. Unlike the question above, this projection is sorted by channel before StartTime so each row will have a distinct StartTime value because shows don't run concurrently on the same channel. Also, there are only 24 unique values for this column because shows only start at the beginning of the hour, so we only need 5 bits per value.

Problem 5: Cost-Based Optimization (16 points)

Suppose we have relations X(A, B), Y(B, C), and Z(C, D). Recall that the notation X(A, B) means that relation X has attributes A and B. We have collected the following statistics about our relations, where T(R) is the number of tuples in a relation and V(R, A) is the number of distinct values of attribute A in a relation. Note that A is a primary key of X, B is a primary key of Y, and C is a primary key of Z.

T(X) = 400	T(Y) = 1000	T(Z) = 1000
V(X, A) = 400	V(Y, B) = 1000	V(Z, C) = 1000
V(X, B) = 200	V(Y, C) = 5	V(Z, D) = 900

We wish to optimize the query $\sigma_{B<100 \text{ AND } C<500}(X \bowtie Y \bowtie Z)$.

a) (4 points) Cost-based optimization involves choosing the most efficient plan out of those logically equivalent to our query. Write four different relational algebra expressions that are logically equivalent to our query. None need be optimal.

Many acceptable answers. Examples: $\sigma_{B<100 \text{ AND } C<500}((X \bowtie Y) \bowtie Z)$, $\sigma_{B<100 \text{ AND } C<500}(X \bowtie (Y \bowtie Z))$, $\sigma_{B<100} X \bowtie (\sigma_{B<100 \text{ AND } C<500} Y \bowtie {}_{C<500} Z))$
$\sigma_{B<100} (X \bowtie \sigma_{C<500} Y \bowtie {}_{C<500} Z)$

b) (8 points) Size estimation is an important part of cost-based optimization. For each of the following tables, provide both its expected size and the expected number of unique values for each of its attributes (rounded to the nearest integer if appropriate), assuming that values of attribute T in relation R are distributed uniformly between 0 and V(R, T).

i. $X \bowtie Y$

$T(X \bowtie Y)$: ____400____
$V(X \bowtie Y, A)$: ____400____
$V(X \bowtie Y, B)$: ____200____
$V(X \bowtie Y, C)$: ___5_____

ii. $\sigma_{B<100}(X)$

$T(\sigma_{B<100}(X))$: ___200_____
$V(\sigma_{B<100}(X), A)$: ____200____
$V(\sigma_{B<100}(X), B)$: ____100____

iii. $\sigma_{B<100 \text{ AND } C<500}(X \bowtie Y)$

$T(\sigma_{B<100 \text{ AND } C<500}(X \bowtie Y))$: ___200_____
$V(\sigma_{B<100 \text{ AND } C<500}(X \bowtie Y), A)$: ___200_____
$V(\sigma_{B<100 \text{ AND } C<500}(X \bowtie Y), B)$: ___100_____
$V(\sigma_{B<100 \text{ AND } C<500}(X \bowtie Y), C)$: ___5_____

iv. $\sigma_{B<100 \text{ AND } C<500}(Y \bowtie Z)$

$T(\sigma_{B<100 \text{ AND } C<500}(Y \bowtie Z))$: ____100____
$V(\sigma_{B<100 \text{ AND } C<500}(Y \bowtie Z), B)$: ___100_____
$V(\sigma_{B<100 \text{ AND } C<500}(Y \bowtie Z), C)$: ____5____
$V(\sigma_{B<100 \text{ AND } C<500}(Y \bowtie Z), D)$: ___5_____

c) (4 points) Assume that all joins are executed as hash joins in memory after applying pushed-down selections. The hash join is executed by hashing each row of the table for which the join column is the primary key, then performing lookups with each row of the other table. For each of these query plans, how many values must be hashed? How many hash table lookups are necessary?

i. $(\sigma_{B<100}(X) \bowtie \sigma_{B<100 \text{ AND } C<500}(Y)) \bowtie {}_{C<500}(Z)$

Interior: 100 hashed, 200 lookups, output size 200.
Exterior: 500 hashed, 200 lookups.
Total: 600 hashed, 400 lookups.

ii. $\sigma_{B<100}(X) \bowtie (\sigma_{B<100 \text{ AND } C<500}(Y) \bowtie \sigma_{C<500}(Z))$

Interior: 500 hashed, 100 lookups, output size 100.
Exterior: 100 hashed, 200 lookups.
Total: 600 hashed, 300 lookups.