STUDENT NAME: _____ **STUDENT ID:** _____

CS 222/122C – Fall 2017, Midterm Exam

Principles of Data Management
Department of Computer Science, UC Irvine
Prof. Chen Li
(Max. Points: 100)

Instructions:

- **This exam has six (6) questions.**
- **This exam is closed book. However, you can use one cheat sheet (A4 size).**
- **The total time for the exam is 75 minutes, so budget your time accordingly.**
- **Be sure to answer each part of each question after reading the whole question carefully.**
- **If you don't understand something, ask for clarification.**
- **If you still find ambiguities in a question, write down the interpretation you are taking and then answer the question based on that interpretation.**

QUESTION	POINTS	SCORE
1	12	
2	12	
3	21	
4	16	
5	18	
6	21	
TOTAL	100	

Question 1 Short questions (12 points)

a) (3 pts) Explain on a hard disk, why sequential IOs are more efficient than random IOs.

Sequential IOs can transfer multiple pages for each seek and rotation, why random disk IOs need to pay this cost for each page.

b) (3 pts) Explain the meaning of pin/unpin operations in a buffer manager.

PIN: when a process needs to consume a frame in the buffer, it needs to pin the frame to tell the manager, which will increment the count by 1. After the process is done with using the data, it calls UNPIN to tell the manager, which will decrement the count by 1. The manager can replace this page only if the count is 0, meaning no process is accessing this page.

c) (3 pts) Explain how the dirty bit of a frame is used by a buffer manager.

A writer of a frame needs to set the dirty bit of a page to 1, indicating that the page has been changed. If the dirty bit is 1, the buffer manager needs to flush the page to the disk before replacing it. If the bit is 0, the flush is not needed.

d) (3 pts) Schema versioning allows a table's schema to be changed without physically modifying those existing records. Explain how it works.

A table has multiple schema versions stored in the catalog. Each record has its only version so that we can use the version ID to retrieve the schema and decide to how to parse the bytes.

Question 2: OLAP/OLTP (12 points)

Consider a table

Sales(salesId, productId, customerId, time, storeId, price)

with information about what customers bought what products at what stores at what time. Assume the table is large with 300 million records in it.

Consider the following two queries:

Q1: Select * from sales where salesId = 19381;
Q2: Select SUM(price) from sales group by storeId;

a) (4 pts) If the DBMS has a choice between a column store and a row store, which option is good for Q1 and which option is good for Q2? Why?

Column store is good for OLAP queries since they tend to access a few columns of many records, and column store can be IO efficient by doing compression. Row store is good for OLTP queries since they tend to access a small number of records with many columns. So a row store is good for Q1, and a column store is good for Q2.

b) (2 pts) Which query is an OLTP query? Which query is an OLAP query?

Q1: OLTP, Q2: OLAP

c) (6 pts) Use these two queries to explain at least two differences between OLTP and OLAP queries.

OLTP: accessing a small number of records, low response time, many queries at the same time; OLAP: access many records, often using aggregation, group by, expensive joins, could be time consuming, fewer queries.

Question 3: Record Manager (21 points)

Suppose we have a table with the following schema:

Students(sid INT, name VARCHAR(30), address VARCHAR(100), gpa FLOAT)

We store the table as a heap file of variable-length records. Each record is stored as a sequence of bytes with a directory of pointers. Assume that:
(1) The directory uses a 2-byte offset to store an ending position for each field.
(2) The offset starts from 0, which is the beginning of the directory.
(3) The schema is known in all record operations so there is no need to store the number of fields in each record;
(4) Integers and floating point numbers are 4 bytes.
(5) Each NULL value is represented using a special value -1 in the corresponding pointer in the directory.

Consider this record:

(100, "Peter", NULL, 3.8)

a) (4 pts) Draw a byte array and explain how this record is stored using the format stated above. Clearly indicate the number of bytes in each segment and all their values.

11	16	-1	20	100	P	e	t	e	r	3.8
2	2	2	2	4				5		4

OR

12	17	-1	21	100	P	e	t	e	r	3.8
2	2	2	2	4				5		4

b) (4 pts) Draw a byte array using the API data format from the course projects, i.e., the format for *"*data"* in functions *"insertRecord()"* and *"updateRecord()"* for RBFM. For the null indicator, show the values of the 0/1 bits. Clearly indicate the number of bytes in each segment and all their values.

00100000	100	5	P	e	t	e	r	3.8
1	4	2 or 4			5			4

c) (3 pts) Give one advantage of the data storage format in a) compared to the API format in b).

O(1) field access

d) (10 pts) When inserting a new record, we always first check the last page. If the last page doesn't have enough space, we need to find an existing page that can hold the record, or create a new page if we can't find it.

Suppose we have two ways to implement it:
(1) Scan each page from the beginning of the file until we find a suitable page.
(2) Use a **single** header page to store the free-space information of each page and look up the header page.

Assume that:
(1) After each insert, all the modified pages should be immediately flushed to disk.
(2) Initially the file doesn't have any pages, even the header page in implementation (2).
(3) We know where the last page is without any disk IO.

Initially the table is empty. We are going to insert 10 records one by one, and each page can store exactly 3 records. Fill in the following two tables for the number of reads and writes for each insertion.

When inserting a record requires creating a new page, there should be 0 Read and 1 Write. Because you can just write the page with the record in it.
However, we also accept answers that assume 1 Read and 1 Write in this situation, as long as the answer itself is consistent.

For the header page implementation. You can choose to 1) always check the last page first, or 2) always check the header page first.
1) will have 2 Reads when inserting record 4, 7, and 10.
2) will have 1 Read when inserting record 4, 7 and 10.
Both are correct.

When creating a new page, there are 0 Read and 1 Write.

Implementation 1 (scan)		
Inserted Record	Number of Reads	Number of Writes
1	0	1
2	1	1
3	1	1
4	1	1
5	1	1
6	1	1
7	2	1
8	1	1
9	1	1
10	3	1

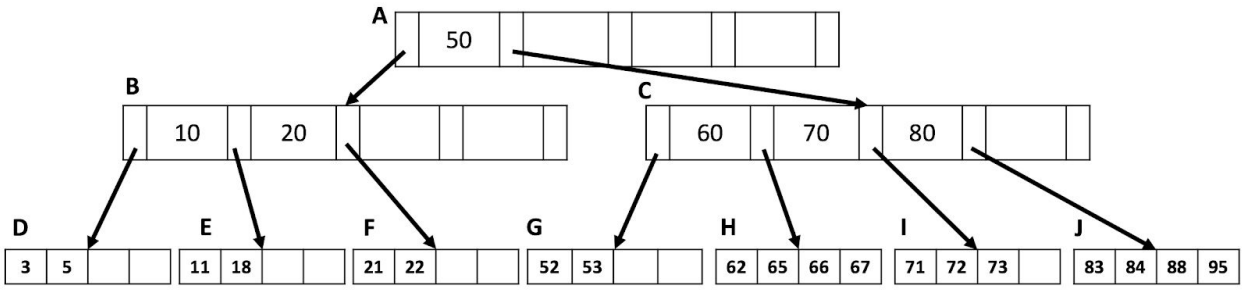
Implementation 2 (header page)		
Inserted Record	Number of Reads	Number of Writes
1	0	2
2	2	2
3	2	2
4	2 (or 1)	2
5	2	2
6	2	2
7	2 (or 1)	2
8	2	2
9	2	2
10	2 (or 1)	2

1 Read and 1 Write when creating a new page:

Implementation 1 (scan)		
Inserted Record	Number of Reads	Number of Writes
1	1	1
2	1	1
3	1	1
4	2	1
5	1	1
6	1	1
7	3	1
8	1	1
9	1	1
10	4	1

Implementation 2 (header page)		
Inserted Record	Number of Reads	Number of Writes
1	2	2
2	2	2
3	2	2
4	2	2
5	2	2
6	2	2
7	2	2
8	2	2
9	2	2
10	2	2

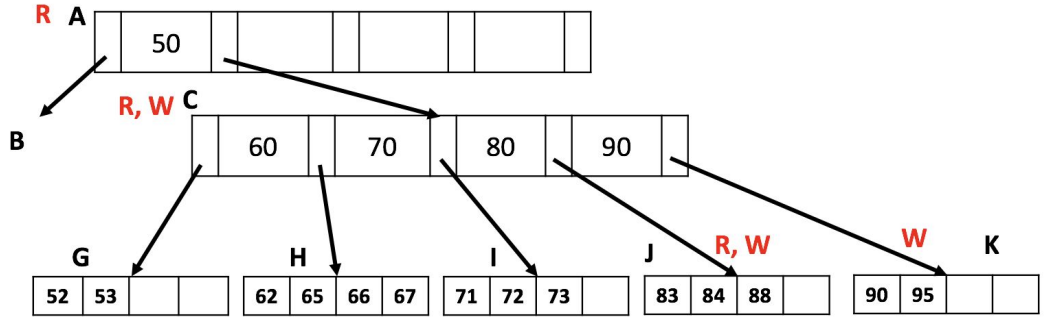
Question 4: B+ Tree (16 points)
Consider the following B+ tree



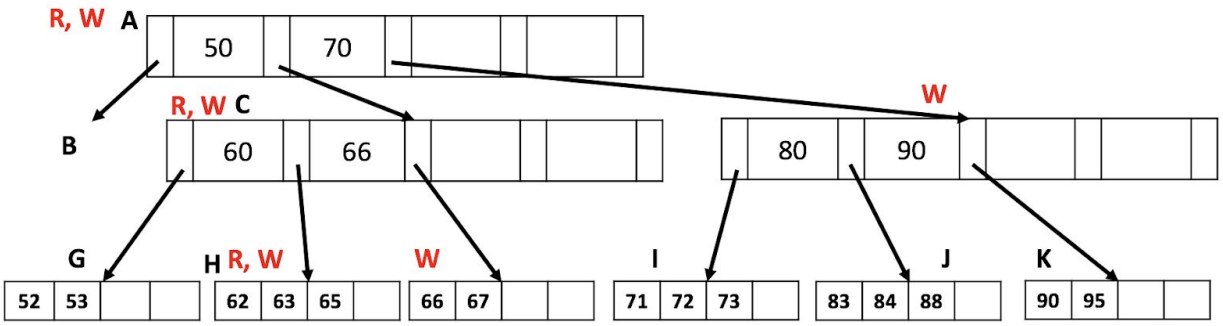
For the following questions, draw the updated B+ tree after each operation. If there is no ambiguity, you can just draw the part that is changed. For each operation, clearly add an "R" (for read) and "W" (for write) on all the affected pages, as well as the original A/B/C/.../J labels.

Assume:
(1) Each node can hold up to 4 key entries.
(2) For delete operations, you can borrow from the **right** sibling.
(3) For insert operations, you need to do split, not rotation.

a) (8 pts) Insert 90 on the **original** tree.



b) (8 pts) Insert 63 **after operation in question a).**



Question 5: Dynamic Hashing (18 points)

a) (4 pts) Briefly describe a main drawback of static hashing in terms of its performance.

Long overflow chain

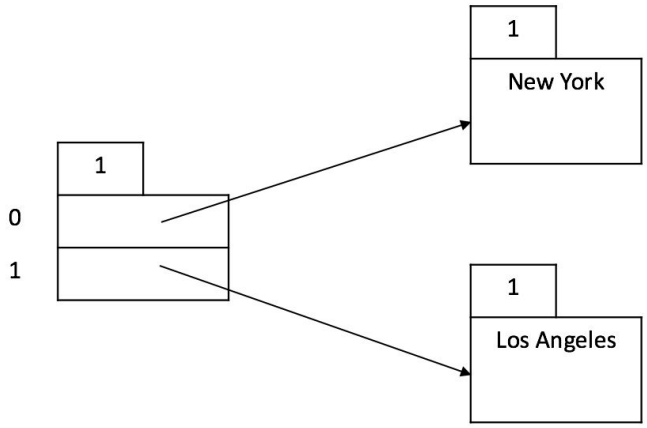
b) (4 pts) Briefly describe how global depth and local depth in extendible hashing are used.

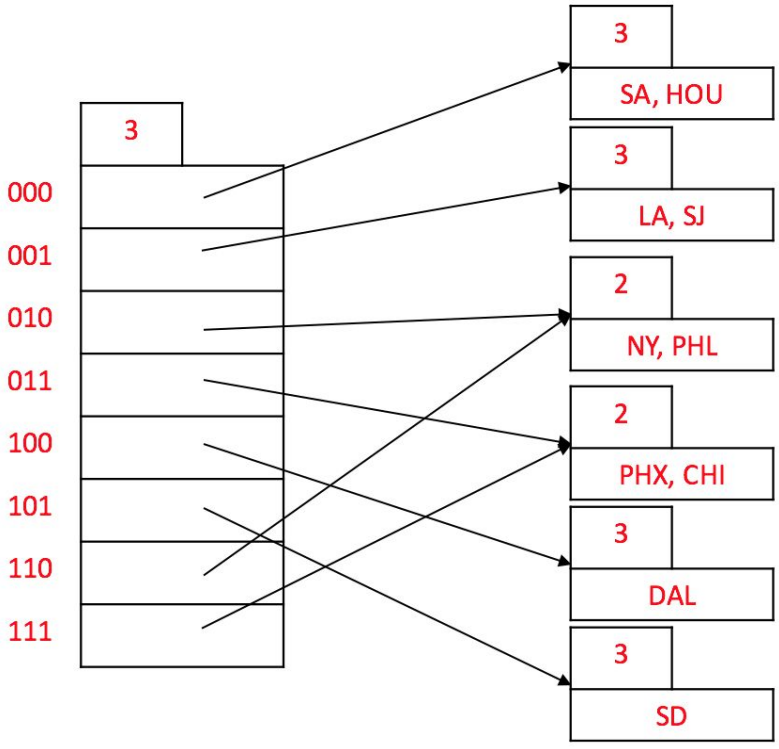
Global depth to figure out which bucket an entry belongs to
Local depth for splitting

Now, consider storing a set of city code records in a table, which has a dynamic hash table. Suppose we want to insert the following 10 entries. We are using a hash function called ***h,*** and the binary hashed value for each entry is given below. Assume that the bits of h(k) are used in least-to-most order of significance and that only 2 records can fit into a bucket/page. For example, for a binary value "1101", we use the two rightmost bits "01" to find the bucket.

City code	h(city)
NY	1010
LA	0001
CHI	0111
HOU	0000
PHX	1011
PHL	1010
SA	1000
SD	1101
DAL	1100
SJ	1001

c) (10 pts) Suppose the index is based on the extendible hashing scheme. Below is the state of dynamic hashed index after inserting first two data entries. Draw a diagram representing final state of the dynamic hash index after inserting all the 10 given data entries.





Question 6: Indexing Performance (21 points)

Suppose you are in charge of managing the database for a bookstore. Consider a table

Books(bookId, category, price, title, author)

containing 10,000,000 records organized as a heap file, and each page contains 100 records. We have the following assumptions.

- "bookId" is an integer value distributed uniformly over the range [0, 99,999].
- "category" is an integer value distributed uniformly over the range [0, 999]
- "price" is a double value distributed over the range [0, 1,000]
- "title" and "author" are variable-length strings with a not-null constraint and a maximum length of 100.
- "rating" is a double value uniformly distributed over the range [0, 5]
- The value distributions of multiple attributes are independent.
- Each plan can only access a single index, possibly followed by accessing the heap file.
- Each index page contains 1,000 key entries (in case of composite keys, this number is divided by the number of keys in the composite key) and non-leaf pages of a B+ tree are always cached by the buffer manager.
- RIDs are sorted before accessing the heap file.

You are given a task to set up proper secondary indexes for each type of workload, if applicable. For questions a) - c), describe **which strategy would work the best for the given workload**. **Briefly explain the reason, describe whether an index-only plan is applicable or not**, and **report the I/O cost of using the method that you choose**.

1) Heap-file scan
2) Unclustered B+ tree index on "title";
3) Unclustered B+ tree index on "author";
4) Unclustered B+ tree index on "price";
5) Unclustered B+ tree index on "category";
6) Unclustered B+ tree index on "rating";
7) Unclustered B+ tree index on composite key "<category, price>";
8) Unclustered B+ tree index on composite key "<price, category>";
9) Unclustered B+ tree index on composite key "<title, author>";

a) (7 pts) SELECT title, author FROM Books WHERE category=10;

5), not applicable
#records = 10M/1000 = 10K
#index pages = 10
#data pages =10K
total cost=10K+10 = 10,010

b) (7 pts) SELECT * FROM Books WHERE price>=10;

1), not applicable
total cost = 10M/100=100K

c) (7 pts) SELECT COUNT(*), AVG(price) FROM Books
WHERE category BETWEEN 21 AND 23 AND PRICE <= 500;

7), applicable
We have two approaches to do index search.

a) perform separate scans with category = 21, 22 or 23, and price <=500.
For each sub query, #records = 10M/1000/2=5K, #index pages = 5K/1000=5.
Thus, total cost = 5*3=15

b) perform one index scan, with search lower bound being [21, 0] and upper bound being [23, 500].
#records = 10M * (1/1000+1/1000 + 1/1000*1/2)= 10M/400 = 25K, #index pages = 25K/ 1000 = 25
Thus, total cost = 25.

c) perform separate scans with category = 21, 22 or 23, and price <=500.
For each sub query, #records = 10M/1000/2=5K, #index pages = 5K/500=10.
Thus, total cost = 10*3=30

d) perform one index scan, with search lower bound being [21, 0] and upper bound being [23, 500].

#records = 10M * (1/1000+1/1000 + 1/1000*1/2)= 10M/400 = 25K, #index pages = 25K/ 500 = 50

Thus, total cost = 50.