CS 186 Spring 2021 Introduction to Database Systems

MIDTERM 1

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- ◯ You must choose either this option
- ◯ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- ☐ You could select this choice.
- ☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

1 CS W186 Spring 2021 Midterm 1

`Do not share this exam until solutions are released.`

1.0.1 Contents:

- The midterm has 5 questions, each with multiple parts, and worth a total of 100 points.

1.0.2 Taking the exam:

- You have 110 minutes to complete the midterm.
- You may print this exam to work on it.
- For each question, submit only your final answer on examtool.
- For numerical answers, do not input any suffixes (i.e. if your answer is 5 I/Os, only input 5 and not 5 I/Os or 5 IOs).
- Make sure to save your answers in examtool at the end of the exam, although the website should autosave your answers as well.

1.0.3 Aids:

- You may use 1 page (double sided) of handwritten notes as well as a calculator.
- You must work individually on this exam.

1.0.4 Grading Notes:

- All I/Os must be written as integers. There is no such thing as 1.02 I/Os – that is actually 2 I/Os.
- 1 KB = 1024 bytes. We will be using powers of 2, not powers of 10
- Unsimplified answers, like those left in log format, will receive a point penalty.

(a) What is your full name?

(b) What is your student ID number?

1. (23 points) 3 SQL Birds

(a) (5 points) Monotonic Queries

A query Q is *monotonic* if whenever we add tuples to one or more of the input relations $R_1, \ldots, R_n$ to Q, the answer to Q will include all the original tuples outputted by running the query on the original relations $R_1, \ldots, R_n$.

For each of the queries below on relation `R(INT id, INT age, INT year, STRING major)`, indicate whether it is monotonic or not. You get 1 point if your answer is correct, -0.5 points if your answer is incorrect, and 0 if no answer. The minimum you can get for this problem is 0 points.

i. (1 pt) $Q =$ `SELECT * FROM R WHERE R.age >= 21;`

- ● Q is monotonic.
- ○ Q is not monotonic.

ii. (1 pt) $Q =$ `SELECT COUNT(*) FROM R WHERE R.year > 2 GROUP BY R.id;`

- ○ Q is monotonic.
- ● Q is not monotonic.

COUNT is not monotonic since adding a tuple to the input relation might change the count, and hence the previous tuple returned is not retained.

iii. (1 pt) $Q =$ `SELECT * FROM R as r1, R as r2 WHERE r1.id = r2.id;`

- ● Q is monotonic.
- ○ Q is not monotonic.

iv. (1 pt) $Q =$ `SELECT * FROM R ORDER BY R.age LIMIT 1;`

- ○ Q is monotonic.
- ● Q is not monotonic.

This query computes the person with the min age, and thus its answer can change if a new person with a lower min age appears.

v. (1 pt) $Q =$ `SELECT * FROM R GROUP BY id, age, year, major;`

- ● Q is monotonic.
- ○ Q is not monotonic.

(b) (18 points) $QL

One of your friends just became interested in trading on the stock market after recent hype and needs your help to do analysis of companies and trades. Consider the following tables:

```
/* This table is a list of companies and their names and stock symbols. */
CREATE TABLE Companies (
    company_id      INTEGER PRIMARY KEY,
    name            VARCHAR(20) NOT NULL,
    stock_symbol    VARCHAR(4) NOT NULL
);

/* This table contains share prices of companies with a date specified by
3 columns (year, month, day) and a timestamp specified by number of seconds
since Jan 1, 1970. */
CREATE TABLE Prices (
    price_id    INTEGER PRIMARY KEY,
    company_id  INTEGER REFERENCES Companies,
    share_price FLOAT NOT NULL,
    year        INTEGER NOT NULL,   /* 1975 - 2021 */
    month       INTEGER NOT NULL,   /* 1 - 12 */
    day         INTEGER NOT NULL,   /* 1 - 31 */
    time_in_sec INTEGER NOT NULL    /* Total number of seconds since Jan 1, 1970 */
);

/* This table contains info about stock trades: the price_id, the trade type
('BUY' or 'SELL') and the number of shares traded. For those unfamiliar with the
stock market, if 3 shares of a company were bought at $4 per share, this trade
(which is a 'BUY') would be $12 total. */
CREATE TABLE Trades (
    trade_id        INTEGER PRIMARY KEY,
    price_id        INTEGER REFERENCES Prices,
    trade_type      VARCHAR(4) NOT NULL,
    shares_traded   FLOAT NOT NULL
);
```

i. (1 pt) Does the following query return the total number of 'BUY' trades on January 28, 2021?

```
SELECT COUNT(*)
FROM Prices INNER JOIN Trades
    ON Prices.price_id = Trades.price_id
WHERE year = 2021 AND month = 1 AND day = 28 AND trade_type = 'BUY';
```

● Yes

○ No

The following SQL Fiddle link will help with all the SQL questions: http://sqlfiddle.com/#!17/05d30/16

ii. **(1 pt)** Same question but instead for this query:

```
SELECT COUNT(*)
FROM Prices INNER JOIN Trades
    ON Prices.price_id = Trades.price_id
GROUP BY year, month, day
HAVING year = 2021 AND month = 1 AND day = 28 AND trade_type = 'BUY';
```

◯ Yes

● No

`trade_type` can't be in the `HAVING` clause.

iii. **(1 pt)** Also same question but instead for this query:

```
SELECT SUM(ones)
FROM (
    SELECT 1 AS ones
    FROM Prices INNER JOIN Trades
        ON Prices.price_id = Trades.price_id
    WHERE year = 2021 AND month = 1 AND day = 28 AND trade_type = 'BUY'
);
```

● Yes

◯ No

iv. **(1 pt)** Does the following query return the total number of shares traded for the company whose stock symbol is 'GME'?

```
SELECT SUM(shares_traded)
FROM Companies, Prices, Trades
WHERE Companies.company_id = Prices.company_id AND
    Prices.price_id = Trades.price_id AND
    stock_symbol = 'GME';
```

● Yes

◯ No

v. **(1 pt)** Same question but instead for this query:

```
SELECT SUM(shares_traded)
FROM Companies, Prices, Trades
WHERE Companies.company_id = Prices.company_id AND
    Prices.price_id = Trades.price_id
GROUP BY stock_symbol
HAVING stock_symbol = 'GME';
```

● Yes

◯ No

vi. (2 pt) Also same question but instead for this query:

```
SELECT SUM(shares_traded) FROM (
    SELECT Companies.company_id, shares_traded
    FROM Companies, Prices, Trades
    WHERE Companies.company_id = Prices.company_id AND
        Prices.price_id = Trades.price_id
    EXCEPT
    SELECT Companies.company_id, shares_traded
    FROM Companies, Prices, Trades
    WHERE Companies.company_id = Prices.company_id AND
        Prices.price_id = Trades.price_id AND
        Companies.stock_symbol != 'GME'
);
```

○ Yes

● No

We must use `EXCEPT ALL` instead because we need to keep duplicates. If there are multiple duplicated trades with the same `company_id` and `shares_traded`, they will be eliminated with `EXCEPT`.

vii. **(2 pt)** We want to find the top 5 companies in terms of percent growth of their share price in the year 2020. This can be expressed as `(latest share price in 2020 - earliest share price in 2020) / earliest share price in 2020`. Return the company ids and percent growth values. You can assume all of a company's price timestamps are unique and that no two companies will have the same percent growth.

```
WITH
    earliest_share_prices AS (
        SELECT company_id, share_price
        FROM Prices p1
        WHERE year = 2020 AND _____(1)_____ <= ALL(
            SELECT _____(2)_____
            FROM Prices p2
            WHERE year = 2020 AND _____(3)_____
        )
    ),
    latest_share_prices AS (
        SELECT company_id, share_price
        FROM Prices p1
        WHERE year = 2020 AND _____(1)_____ >= ALL(
            SELECT _____(2)_____
            FROM Prices p2
            WHERE year = 2020 AND _____(3)_____
        )
    ),
    percent_growths AS (
        SELECT esp.company_id, _____(4)_____ AS percent_growth
        FROM earliest_share_prices esp, latest_share_prices lsp
        WHERE _____(5)_____
    )
SELECT company_id, percent_growth
FROM percent_growths
ORDER BY percent_growth
LIMIT 5;
```

What belongs in all the blanks labeled `(1)`?

> `time_in_sec` **or** `p1.time_in_sec`

viii. **(2 pt)** What belongs in all the blanks labeled `(2)`?

> `time_in_sec` **or** `p2.time_in_sec`

ix. **(2 pt)** What belongs in all the blanks labeled `(3)`?

> `p1.company_id = p2.company_id`

x. **(1 pt)** What belongs in the blank labeled `(4)`?

> `(lsp.share_price - esp.share_price) / esp.share_price`

xi. (1 pt) What belongs in the blank labeled `(5)`?

```
esp.company_id = lsp.company_id
```

xii. (1 pt) For the following relational algebra questions, refer to `Companies` as C, `Prices` as P, and `Trades` as T.

Again, here is the database schema:

```
/* This table is a list of companies and their names and stock symbols. */
CREATE TABLE Companies (
    company_id      INTEGER PRIMARY KEY,
    name            VARCHAR(20) NOT NULL,
    stock_symbol    VARCHAR(4) NOT NULL
);

/* This table contains share prices of companies with a date specified by
3 columns (year, month, day) and a timestamp specified by number of seconds
since Jan 1, 1970. */
CREATE TABLE Prices (
    price_id    INTEGER PRIMARY KEY,
    company_id  INTEGER REFERENCES Companies,
    share_price FLOAT NOT NULL,
    year        INTEGER NOT NULL,   /* 1975 - 2021 */
    month       INTEGER NOT NULL,   /* 1 - 12 */
    day         INTEGER NOT NULL,   /* 1 - 31 */
    time_in_sec INTEGER NOT NULL    /* Total number of seconds since Jan 1, 1970 */
);

/* This table contains info about stock trades: the price_id, the trade type
('BUY' or 'SELL') and the number of shares traded. For those unfamiliar with the
stock market, if 3 shares of a company were bought at $4 per share, this trade
(which is a 'BUY') would be $12 total. */
CREATE TABLE Trades (
    trade_id        INTEGER PRIMARY KEY,
    price_id        INTEGER REFERENCES Prices,
    trade_type      VARCHAR(4) NOT NULL,
    shares_traded   FLOAT NOT NULL
);
```

Which of the following returns the stock symbols of the companies which, at any time in 2020, had a share price greater than \$150?

- ● $\pi_{\text{stock\_symbol}}(\sigma_{\text{year}=2020 \wedge \text{price}>150}(C \bowtie P))$
- ○ $\pi_{\text{stock\_symbol}}(\sigma_{\text{year}=2020}(C \bowtie P)) \cup \pi_{\text{stock\_symbol}}(\sigma_{\text{price}>150}(C \bowtie P))$
- ○ $\pi_{\text{stock\_symbol}}(\sigma_{\text{year}=2020}(C \bowtie P)) - \pi_{\text{stock\_symbol}}(\sigma_{\text{price}>150}(C \bowtie P))$

xiii. (2 pt) Which of the following returns the stock symbols of the companies which have greater than 10,000 shares ever traded on it?

- ○ $\pi_{\text{stock\_symbol}}(\gamma_{\text{C.company\_id}}(C \bowtie P \bowtie T)) - \pi_{\text{stock\_symbol}}(\gamma_{\text{C.company\_id,SUM(shares\_traded)>10000}}(C \bowtie P \bowtie T))$
- ● $\pi_{\text{stock\_symbol}}(\gamma_{\text{C.company\_id,SUM(shares\_traded)>10000}}(C \bowtie P \bowtie T))$
- ○ $\pi_{\text{stock\_symbol}}(\gamma_{\text{company\_id,COUNT(shares\_traded)>10000}}(C \bowtie P \bowtie T))$

We decided to award everyone who answered this question full points since all three are incorrect. We cannot project out a column that is not in the group by operator.

2. (22 points) Put Your Records On (Tell Me Your Favorite Song)

(a) (4 points) True or False

For the next 4 questions, indicate whether the statement is True or False.

i. (1 pt) For heap files, a delete operation on a packed page will always cost the same number of I/Os as a delete operation on an unpacked page.

◯ True

● False

Packed pages involve updating recordID pointers, which can incur additional I/Os if they are on other pages, such as an index. Unpacked pages avoid rearranging records, so this cost is avoided.

ii. (1 pt) Reading a 1024KB record from disk will always take exactly twice as long as reading a 512KB record from disk.

◯ True

● False

Consider a page size of 1024KB, each read will cost 1 I/O.

iii. (1 pt) A bitmap in the page header is not necessary to track fixed length records stored in a packed page layout.

● True

◯ False

Since the pages are packed and records are fixed length, a bitmap is not necessary to track which "slots" are available. Instead, the recordID can be used to compute the byte offset where a record begins, since all records are of fixed size.

iv. (1 pt) The free space pointer in a slotted page layout will never point to the end of a deleted record.

◯ True

● False

If the last record on the page is deleted, the corresponding slot will be reclaimed, but the actual defragmentation process to fill in gaps may not happen until later.

(b) (10 points) Extra, Extra (IO)! Read (or Write) all about it!

Consider the following table.

```
CREATE TABLE Publications (
    article_id INTEGER PRIMARY KEY,
    journalist_id INTEGER,
    publisher VARCHAR[26] NOT NULL,
    topic CHAR[20],
    off_record BOOLEAN
);
```

For the next 5 questions, assume the following:

- *Publications* is stored in a Heap File Page Directory implementation. There are 30 data pages and each page directory header has 8 entries.
- Data pages follow the slotted page layout presented in class. The footer on each data page reserves 8 bytes total for the slot count and free space pointer. Each slot takes 8 bytes.
- Records are stored as variable length records.
- The record header contains a bitmap to track all fields that can be NULL. The bitmap is as small as possible, rounded up to the nearest byte.
- Integers and pointers are 4 bytes. Booleans are 1 byte.
- Each page is 2KB (2048 bytes).

i. (2 pt) What is the minimum size of a *Publications* record in bytes?

34

The minimum record size is 4 (article_id) + 4 (journalist_id) + 4 (pointer to publisher) + 20 (topic) + 1 (off_record) + 1 (null bitmap) = 34 bytes. This is when the publisher field is an empty string.

A common mistake students made is forgetting that just because a field can be NULL does not mean we do not need to allocate space for it. For fixed length fields we always allocate the same amount of space even if the field can be NULL, i.e. topic CHAR[20] will always take 20 bytes.

ii. (2 pt) What is the minimum number of *Publications* records it takes to fill up a data page? A data page is full when no more records can be inserted.

30

The maximum size for a Publications record is 34 + 26 = 60 bytes. Subtracting 8 bytes from the page size of 2048 bytes for the slot count and pointer to free space leaves 2040 bytes for slots and records. Thus, it takes a minimum of

$$\frac{2040}{60+8} = 30$$

Publications records to fill up a data page.

iii. (2 pt) Write an explanation for your answer to the previous 2 questions.

iv. (4 points)

For the next 2 questions, assume the queries are independent of each other; i.e. the 2nd query is run on a copy of the file that has never had the 1st query run on it. There is always at least 1 data page with enough free space to insert records. The buffer is large enough to hold all data and header pages, and starts empty **for each question**.

Provide the **worst** case cost in I/Os to execute the following queries.

A. (2 pt) `SELECT * FROM Publications WHERE topic = 'New Building' and off_record = TRUE;`

34

Any of the records could match the WHERE predicate, so a full table scan must occur. This involves reading all header pages, of which there are 4 since it takes 4 header pages to store entries for 30 data pages if each page directory header can store 8 entries. All 30 data pages also need to be read so the total I/O cost is 34.

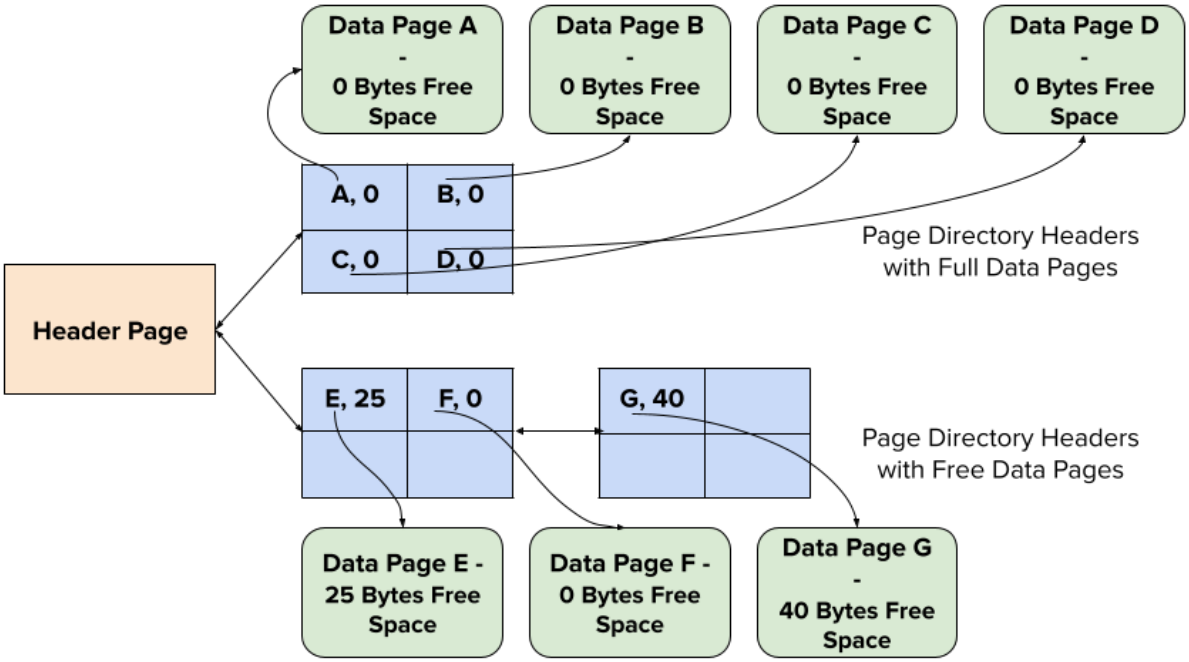
B. (2 pt) `INSERT INTO Publications(186, 15, 'Daily Cal', 'Student Life', FALSE);`

36

First, all of the records need to be checked to ensure that no other record has `article_id` of 186, since it is a PRIMARY KEY and PRIMARY KEYs must be unique. This is simply the cost of a full table scan, which is 34. Then, it takes 1 I/O to write the data page where the record was inserted and 1 I/O to write the corresponding page directory header after updating the page entry.

(c) (8 points) Inside Scoop on Heap File Design

Consider the design of a new heap file implementation called the Linked List of Page Directories (LLPD). The LLPD consists of a regular linked list with a sublist for free pages and a sublist for full pages as usual. However, the sublists contain page directory header pages, instead of data pages. Each page directory header page is implemented as shown in lecture. If all the entries in a page directory header page point to data pages that are full, the header page will be placed in the full pages sublist; otherwise, it will be placed in the free pages sublist. This means that if a page directory header is in the full list, then all of its entries must be occupied and each of those data pages have no free space. An example of the LLPD is shown below.



Sample Linked List of Page Directories

For the next 4 questions, assume the following:

- There are 25 page directory headers with full data pages and 30 page directory headers with free data pages.
- Each page directory header has 10 entries.
- There are a total of 450 data pages.
- Queries are independent of each other; i.e. the 2nd query is run on a copy of the file that has never had the 1st query run on it.
- There is always at least 1 data page with enough free space to insert records.
- The buffer is large enough to hold all data and header pages, and starts empty for each question.

Consider the following *Journalists* table, which is stored in the LLPD file described above.

```
CREATE TABLE Journalists (
    journalist_id INTEGER NOT NULL,
    first_name VARCHAR[45] NOT NULL,
    last_name VARCHAR[45] NOT NULL,
    num_articles INTEGER
);
```

i. **(2 pt)** Given the LLPD heap file for the *Journalists* table, what is the worst case cost in I/Os for checking if there is enough space to insert a record?

31

This requires reading the header page and reading all page directory headers in the free data pages list to check if any data page has enough free space.

ii. **(2 pt)** What is the worst case cost in I/Os to execute the following query on the LLPD heap file for the *Journalists* table? `INSERT INTO Journalists(286, 'Oski', 'Undercover', 50);`

39

The worst case is when the 2nd to last page directory header in the free data pages sublist has a data page with just enough free space, such that the data page becomes full and all other entries are full, which means the page directory header needs to be moved to the full data pages sublist. This involves 1 I/O to read the header pages, 30 I/Os to read all the page directory headers with free data pages, 1 I/O to read the relevant data page, 1 I/O to write the relevant data page, 2 I/Os to write the appropriate next and prev pointers for the neighbors of the page directory header page being moved to the other list, 2 I/Os to read and write the original head of page directory headers with full data pages sublist after updating its prev pointer, 1 I/O to update the next pointer of the header page, and finally 1 I/O to write the page directory header which has an entry for the data page where the record was inserted. The total cost is 39 I/Os.

iii. **(2 pt)** Write an explanation for your answer to the previous question.

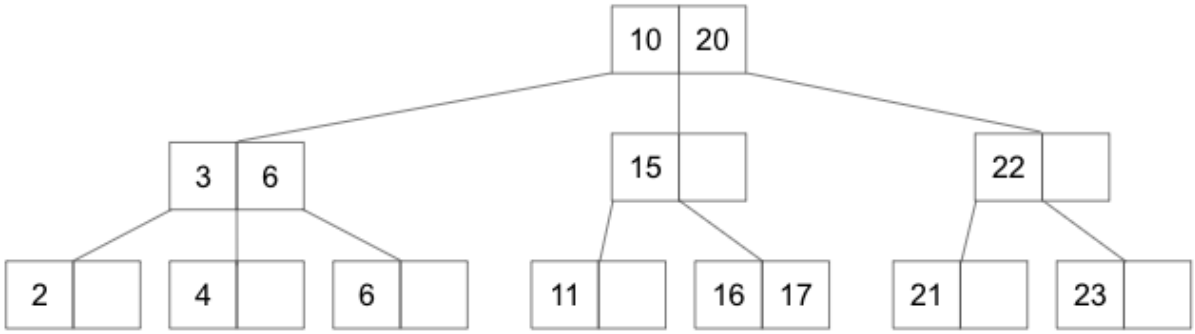
iv. (2 pt) State 1 change you could make to this design to further minimize the worst case I/O cost for checking if there is enough space to insert a record and explain. Your solution may not use indices or make any assumptions about the buffer manager. Limit your answers to 4 sentences.

Possible answers include:

- Keep the free sublist page directory entries sorted in decreasing order of free space.
- Replace the linked list with another page directory header that points to the original page directory headers. In the main header, each entry can consist of a pointer to the page directory header and the maximum free space an entry in that header has.

3. (20 points) Sapphire and B+ Trees

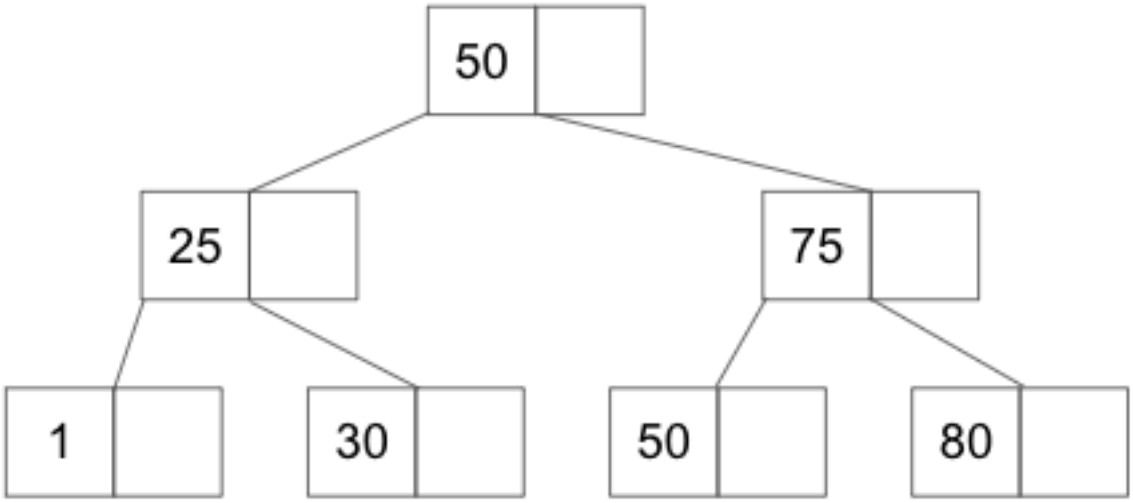
(a) (2 pt) What is the minimum number of keys you could insert to change the height of this tree?



2

One possible sequence of insertions is inserting 7 and 8.

(b) (2 pt) What is the maximum number of keys you could insert without changing the height of this tree?



14

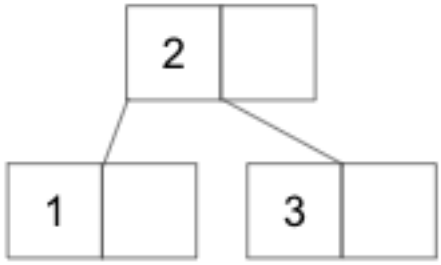
A tree with height 2 and order 1 can hold at most 2 * 3^2 = 18 entries. Currently, the tree stores 4 entries, so we can insert a maximum of 18 - 4 = 14 more entries without increasing the height.

Make sure when finding the number of entries currently in the tree to only count the values in leaf nodes. Remember that only leaf nodes store data (or pointers to data for Alt. 2 and 3 indexes), and the values in inner nodes are only used for searching.

(c) (2 pt) What sequence of operations (i.e. get, put, remove) can be used to obtain the following B+ tree?

Put **"N/A"** as your answer if you believe it is not possible to obtain the following B+ tree through a sequence of operations.

Otherwise, provide your answer as a **comma-separated list** of **at most 5 operations**, i.e. put 7, get 4, put 10, etc.



put 1, put 2, put 3, remove 2

In order for 2 to be in the inner node, a leaf node must have split where 2 was the split key/middle value. Thus, we inserted 1, 2, and 3 into the tree. (Notably, a sequence of operations that would not work is put 1, put 2, remove 2, put 3). We implement the remove operation without rebalancing after deletion, so removing 2 will only delete the value from the leaf node.

(d) (3 points)

In this problem, we want to bulk load a B+ tree with order d = 3 with entries 1, 2, . . . , 32 with a fill factor of f = 2/3.

i. (1 pt) How many leaf nodes are there in the resulting B+ tree?

8

Since d = 3 and f = 2/3, each leaf node can hold 2 * 3 * 2/3 = 4 entries. We have 32 entries, so they will be stored in 32/4 = 8 leaf nodes.

ii. (1 pt) How many inner nodes are there in the resulting B+ tree?

3

Recall that the fill factor only applies to leaf nodes, and not to inner nodes. Also, make sure to count the root node as an inner node (for a tree with height >= 1) - any non-leaf node is an inner node.

- When we bulk load the entries 1, . . . , 28, we can accomplish this with a tree of a height 1 and a single inner node (which will contain 5, 9, 13, 17, 21, 25). Then, when we add 29, the rightmost inner node splits and 29 is added to the inner node.
- Now that the inner node has > 2d = 6 entries, it will split into 2 inner nodes (which will contain 5, 9, 13 and 21, 25, 29), with 17 being pushed up to a parent inner node.
- We can add the remaining entries (30, 31, 32), into the same leaf node without any splitting, so overall our B+ tree will have 3 inner nodes and height h = 2.

iii. (1 pt) What is the height of the resulting B+ tree?

2

(e) (11 points)

For this problem, suppose we have the following `students` table and the following indexes built on the `students` table. Assume each question is independent and starts off with an empty buffer pool.

```
CREATE TABLE students (
    sid INTEGER PRIMARY KEY,
    name TEXT,
    age INTEGER,
    score INTEGER
);
```

Index A:

- Alternative 1 B+ Tree on `(score, name)` with h = 2, d = 4
- Each leaf node is 1/2 full

Index B:

- Alternative 2, clustered B+ Tree on `age` with h = 2
- Contains 25 leaf nodes
- The first entry with `age = 20` appears on the 23rd leaf node

Index C:

- Alternative 3, unclustered B+ Tree on `sid` with h = 2

Additional assumptions:

- The `students` table has 81 data pages.
- There are 5 records with `score = 50`.
- There are 12 records with `age >= 20`.
- Each data page stores 8 records.

i. (2 pt) Estimate the minimum I/O cost of the following query:

```
SELECT * FROM students WHERE name = 'Alice';
```

81

Index A is built on (score, name), but name is only used as a tiebreaker for record with equal scores and we cannot use this index to efficiently look up records based on name. Thus, we need to do a full scan on students and read in all of the data pages, which costs 81 I/Os.

Note that we do not need to traverse any indexes since they will not help us efficiently look up records (a common answer was 84 I/Os). Doing so will incur extra I/O cost, and we can directly start reading from the start of the file instead.

ii. (2 pt) Estimate the minimum I/O cost of the following query:

```
SELECT * FROM students WHERE score = 50;
```

4

We will use Index A for this query since it is built on (score, name).

- Since the B+ tree has a height of 2, it takes 2 I/Os to read inner nodes from the root to the level above the leaf nodes.
- Each leaf node can hold a maximum of 8 entries since d = 4, and they are 1/2 full, meaning they currently hold 4 entries. There are 5 records with score = 50, so these records lie on 2 leaf nodes. Thus, it will take 2 I/Os to read these leaf nodes and read the relevant records.
- Overall, the I/O cost is 2 + 2 = 4 I/Os.

iii. (3 pt) Estimate the minimum I/O cost of the following query:

```
SELECT * FROM students WHERE age >= 20;
```

7

We will use Index B for this query since it is built on age.

- Since the B+ tree has a height of 2, it takes 2 I/Os to read inner nodes from the root to the level above the leaf nodes.
- Since the first entry with age = 20 appears on the 23rd leaf node and we are searching for records with age > 20, we need to read the 23rd leaf node and all leaf nodes to the right. Since there are 25 leaf nodes in total, we will need to read in 3 leaf nodes, a process that incurs 3 I/Os.
- Since the B+ tree is Alternative 2, we need to follow the pointers the leaf nodes store to data pages and read in the corresponding records. With a clustered index, we incur 1 I/O per page of matching records. We have 12 records with age >= 20, and since we can store 8 records on each page, reading in the data pages will cost ceil(12/8) = 2 I/Os.
- Overall, the I/O cost is 2 + 3 + 2 = 7 I/Os.

iv. (2 pt) For this problem only, assume the following costs: `SELECT * FROM students WHERE score = 50;` costs 10 I/Os. `SELECT * FROM students WHERE age >= 20;` costs 20 I/Os.

Using this information, estimate the minimum I/O cost of the following query:

```
SELECT * FROM students WHERE score = 50 AND age >= 20;
```

10

Since we don't have an index built on both score and age, we can either use Index A and scan through all records with score = 50, picking those with age >= 20 or use Index B and scan through all records with age >= 20, picking those with score = 50. The first option takes 10 I/Os and the second takes 20 I/Os, so we choose the former.

v. (2 pt) Estimate the minimum I/O cost of the following query when using Index C. Ignore the costs of maintaining the other indexes.

```
UPDATE students SET score = score + 1 WHERE sid = 1;
```

5

We will use Index C for this query since it is built on sid.

- Note that since sid is the primary key, we will be updating only one record.
- Since the B+ tree has a height of 2, it takes 3 I/Os to read from the root to the leaf node.
- Since the B+ tree is Alternative 3, we need to read in the data page
- the record lies on and write back the modified data. This costs 2 I/Os.
- Overall, the I/O cost is 3 + 2 = 5 I/Os.

4. (17 points) Buffer Management

(a) (5 points) True or False

i. (1 pt) In the MRU eviction policy, the page that gets evicted is **always** the page most recently added to the buffer pool.

○ True

● False

The page that gets replaced is actually the page most recently used, which may differ from the one most recently *added* to the buffer pool.

ii. (1 pt) In the LRU eviction policy, the page that gets evicted is **always** the page least recently added to the buffer pool.

○ True

● False

The page that gets replaced is the page least recently *accessed*, not *added*.

iii. (1 pt) Pages in the buffer pool can only be unpinned by the buffer manager after being selected by the eviction policy.

○ True

● False

Pages are unpinned by their requester, not by the buffer manager.

iv. (1 pt) For identical access patterns, the Clock Policy will never outperform the hit rate of LRU since it is only an approximation.

○ True

● False

False. There are some access patterns where Clock Policy will perform equal or better than LRU.

v. (1 pt) Pinned pages can only be evicted if the page has not been dirtied.

○ True

● False

Pinned pages can never be evicted.

(b) (3 points)

Consider the MRU eviction policy with 4 buffer frames and the following access pattern:

A B C D E F A B C D E F

i. (1 pt) Which pages are in the buffer pool at the end of execution?

- [x] A
- [x] B
- [ ] C
- [ ] D
- [x] E
- [x] F

Buffer pool (in order of recency) at each step:

A. A
B. B, A
C. C, B, A
D. D, C, B, A
E. E, C, B, A
F. F, C, B, A
G. A, F, C, B
H. B, A, F, C
I. C, B, A, F
J. D, B, A, F
K. E, B, A, F
L. F, E, B, A

ii. (1 pt) How many hits occurred?

4 hits (at the 7th, 8th, 9th, and 12th accesses)

iii. (1 pt) Would the hit rate of LRU with 4 buffer frames be better than, worse than, or equally well as MRU on the given access pattern?

- ○ Better
- ● Worse
- ○ Equal

LRU gets 0 hits on this access pattern, due to sequential flooding.

(c) (4 points)

Consider the Clock eviction policy with 4 buffer frames and the following access pattern:

A, C, D (remain pinned), B, A, E, (unpin D), C, F

Assume the clock hand starts out pointing at frame 1. You can assume all pages are unpinned immediately after being accessed unless otherwise stated. For this question, assume that pages have their reference bits set to 1 upon being unpinned.

i. (2 pt) Which pages are in the buffer pool at the end of execution?

- ☐ A
- ☐ B
- ■ C
- ■ D
- ■ E
- ■ F

Buffer pool (in order of recency) at each step:

A. A
B. A, C
C. A, C, D
D. A, C, D, B
E. A*, C, D, B
F. E, C, D, B
G. E, C*, D, B
H. E, C, D, F

ii. (1 pt) Which frames have a reference bit of 1 at the end of the access pattern?

- ■ 1
- ☐ 2
- ☐ 3
- ■ 4

Buffer pool after each step: A: Miss Clock: (A,1)(_,0)(_,0)(_,0) Hand: 2

C: Miss Clock: (A,1)(C,1)(_,0)(_,0) Hand: 3

D: Miss Clock: (A,1)(C,1)*(D,1)*(_,0) Hand: 4

B: Miss Clock: (A,1)(C,1)*(D,1)*(B,1) Hand: 1

A: Hit Clock: (A,1)(C,1)*(D,1)*(B,1) Hand: 1

E: Miss Clock: (E,1)(C,0)*(D,1)*(B,0) Hand: 2

Unpin D Clock: (E,1)(C,0)(D,1)(B,0) Hand: 2

C: Hit Clock: (E,1)(C,1)(D,1)(B,0) Hand: 2

F: Miss Clock: (E,1)(C,0)(D,0)(F,1) Hand: 1

iii. (1 pt) How many hits occurred?

2 hits (at the 5th, 7th time slots)

(d) (5 points)

i. (2 pt) You're analyzing the performance of your implementation of the LRU eviction policy and find that for a benchmark of 1000 page accesses it has a hit rate of 0.84. You also find that the LRU policy requires 20 ms of overhead time to update its internal state whenever a page is accessed regardless of whether it's a hit or a miss. Assuming that on average an I/O takes 500 ms, how many **seconds** does it take to process the access pattern using LRU?

100

LRU has 840 hits and 160 misses. The misses take 160 * 500ms = 80 seconds. There were 1000 accesses, so 1000 * 20ms = 20 seconds of additional time spent on overhead, for a total of 80 + 20 = 100 seconds.

ii. (1 pt) Write an explanation for your answers to the previous question.

iii. (2 pt) You have four buffer frames in memory and need to access the sequence of pages ABCDE repeatedly, in that order, 200 times, for a total of 1000 page accesses. Now, assume that on average an I/O takes 500 ms, on average how many **milliseconds** of overhead time would MRU need per access for this query to have an average completion time of 250 seconds?

148ms

Our miss pattern will be E-D-C-B-A (simulate out a round to see this for yourself). This would mean that after the first 5 accesses (which are all misses), we have a hit rate of 0.80, giving us 0.20 * 995 + 5 = 204 misses. You will thus take 102 seconds in total for IOs. To calculate the overhead time required, the overhead for all 1000 accesses must be completed in 148 seconds, which would give 148 seconds/1000 = 148 ms.

5. (18 points) Let's SORT/HASH this out!

For all problems, assume that your **buffer size is 10 pages**. Please input only your final answer unless asked for otherwise, and do NOT include units in your answer. For questions that ask you to consider modifications to the algorithm, you should consider each modification separately from all previous questions.

(a) (10 points) Sorting

i. (1 pt) What is the maximum number of pages in a table that can be sorted without using external sorting?

10

A table with 10 pages will fit entirely in memory and can be sorted without using any external sorting algorithm. Any table that is larger will require external sorting.

ii. (1 pt) What is the maximum number of pages in a table that can be sorted in 2 passes of external sorting?

90

In pass 0, sorted runs of at most size 10 will be created. In pass 1, 9 runs can be merged at once, since 1 page of the buffer is needed for the output. Therefore, 10(9) or 90 pages can be sorted in these two passes.

iii. (1 pt) What is the maximum number of pages in a table that can be sorted in 3 passes of external sorting?

810

In pass 0, sorted runs of at most size 10 will be created. In pass 1 and 2, 9 runs can be merged at once, since 1 page of the buffer is needed for the output. Therefore, 10(9)(9) or 810 pages can be sorted in these two passes.

iv. (1 pt) How many passes will it take to sort a table with N pages if N is in between your answers to the previous two questions? In other words, N > the maximum number of pages that can be sorted in 2 passes, but N < the maximum number of pages that can be sorted in 3 passes. You should express your answer as an integer.

3

v. (2 pt) How many I/Os will it take to sort a table with 500 pages?

3000

You can use

$$2N * (1 + \lceil \log_{B-1} \lceil N/B \rceil \rceil)$$

to reach this answer, where N = 500 and B = 10, or you can recognize that 500 falls in the range asked by the previous question and will therefore take 3 passes to sort. Each pass takes 2N I/Os.

vi. **(1 pt)** Jerry is building his own external algorithm. He decides to use only half of the buffers in Pass 0 of the external sorting algorithm, and excludes one buffer page during all other passes to use for non-related tasks. On average, will this result in a greater number, equal number, or fewer number of I/Os when compared to the external sorting algorithm taught in class?

- ● Greater Number of I/Os
- ○ Fewer Number of I/Os
- ○ Equal Number of I/Os

The sorted runs will be of smaller length and more sorted runs will be created. This means that occasionally, it will take more passes to merge the high number of sorted runs, resulting in a greater number of I/Os.

vii. **(1 pt)** Jerry decides to re-build his own external algorithm. This time, he uses the same methodology taught in CS186 for the external sorting. However, for the in-memory sort, he doesn't remember how to implement efficient sorts such as merge sort, so he uses selection sort instead. On average, will this result in a greater number, equal number, or fewer number of I/Os when compared to the external sorting algorithm taught in class?

- ○ Greater Number of I/Os
- ○ Fewer Number of I/Os
- ● Equal Number of I/Os

Even though selection sort has a slower runtime than merge sort, the choice of algorithm to sort the pages in memory will not have any effect on the number of I/Os for external sorting. As an aside, the sorts in memory may run slower, but often incurring extra I/Os will cost far more time than a slower in-memory algorithm.

viii. **(1 pt)** Jerry writes a magic algorithm that ensures every individual page of his table will be sorted at no additional I/O cost. He then wishes to sort the entire table. On average, will this result in a greater number, equal number, or fewer number of I/Os when compared to the external sorting algorithm taught in class?

- ○ Greater Number of I/Os
- ○ Fewer Number of I/Os
- ● Equal Number of I/Os

Even though the individual pages are sorted, there is no guarantee that the records are sorted across different pages in any order. Therefore, we will still need to run the entire external sorting algorithm in order to sort the table.

ix. **(1 pt)** Jerry needs to sort all the CS186 students based on the results of their midterm 1 scores. He observes the table and finds that many people received the same score (and it's a good one!). Suppose there are in fact 11 pages of students with duplicate scores. On average, will this high number of duplicates result in a greater number, equal number, or fewer number of I/Os when compared to the external sorting algorithm taught in class?

- ○ Greater Number of I/Os
- ○ Fewer Number of I/Os
- ● Equal Number of I/Os

The number of duplicates does not affect the I/Os of external sorting.

(b) (8 points) Hashing

i. (1 pt) Let's transition over to external hashing. Remember that we are still with a buffer size of 10 pages. How many partitions are created from Pass 0 during the hashing of a table with 120 pages?

9

During the partitioning phase, B-1 partitions are made.

ii. (1 pt) How many pages are in each partition after Pass 0 during the hashing of a table with 120 pages?

14

120 / 9 = 13.33, but because you cannot read/write a fraction of a page, this is rounded up to 14 pages.

iii. (1 pt) Is recursive partitioning needed during the hashing of a table with 120 pages?

- [x] Yes
- [ ] No

Because 14 is larger than our buffer size of 10, we need to recursively partition.

iv. (2 pt) What is the total number of I/Os needed to hash a table with 120 pages?

858

In pass 0, we read in 120 pages and write out 9 partitions of 14 pages each. In pass 1 (our recursive partitioning pass), for each of our 9 partitions of 14 pages, we partition them into 9 partitions of 2 pages. This results in 9(14) reads and 9(9)(2) writes. Finally in pass 2, we read in each of our 81 partitions of 2 pages to construct our in memory hash table and write that back to disk. This results in 81(2) reads and 81(2) writes.

In total, the number of I/Os = 120+9(14)+9(14)+81(2)+81(2)+81(2) = 858.

v. (1 pt) This time, Jerry needs to group all the CS186 students based on the results of their midterm 1 scores. He tries to write his own external hashing algorithm, and he decides to try some of his own hash functions.

In the first partitioning pass, he calculates the score mod 5 and assigns that tuple to the corresponding partition. On average, will this result in a greater number, equal number, or fewer number of I/Os when compared to the external hashing algorithm taught in class? For this question only, you may assume midterm scores are evenly distributed.

- [x] Greater Number of I/Os
- [ ] Fewer Number of I/Os
- [ ] Equal Number of I/Os

This procedure hashes scores into 5 partitions, but in order to make full use of our buffer, we should hash into 9 partitions. This results in partitions of larger sizes which is more likely to cause recursive partitioning and a great number of I/Os to be executed.

vi. **(1 pt)** Jerry uses an uneven hash function to construct in-memory hash tables. On average, will this result in a greater number, equal number, or fewer number of I/Os when compared to the external hashing algorithm taught in class?

- ○ Greater Number of I/Os
- ○ Fewer Number of I/Os
- ● Equal Number of I/Os

The uneven hash functions will cause a greater number of collisions during the construction of the in-memory hash tables, but it will not affect the I/O count.

vii. **(1 pt)** Suppose, once again, there are in fact 11 pages of students with duplicate scores. On average, will this high number of duplicates result in a greater number, equal number, or fewer number of I/Os when compared to the external hashing algorithm taught in class?

- ● Greater Number of I/Os
- ○ Fewer Number of I/Os
- ○ Equal Number of I/Os

We accepted all answers to this question due to lack of clarity on how duplicates are handled in External Hashing. The original explanation can be found below.

If the number of duplicates is greater than the buffer size, that partition will never fit inside the buffer and an in-memory hash table cannot be built. Thus, the I/Os will increase significantly (to infinity) as the external hash cannot be done. In practice, once a number of different hash functions has been tried, the DBMS may assume that the partition contains solely duplicates and terminate the hashing of that partition.

No more questions.