CS 186 Fall 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 Fall 2021 Midterm 1

`You have 110 minutes to complete the midterm. Do not share this exam until solutions are released.`

1.0.1 Contents:

- The midterm has 6 questions, each with multiple parts, and worth a total of 100 points.

1.0.2 Aids:

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

1.0.3 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.

1.0.4 Taking the exam remotely:

- 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.

(a) What is your full name?

(b) What is your student ID number?

(c) SID of the person to your left (Put None if you are taking the exam remotely):

(d) SID of the person to your right (Put None if you are taking the exam remotely):

1. (12 points) Buffer Management

(a) (4 pt) Which of the following are true? :

A. If a page is dirty then it must have at least one pin.

B. When using the LRU policy, the reference bits are used to give pages a "second chance" to stay in memory before they are replaced.

C. A sequential scan of a file will have the same hit rate for MRU and LRU when the number of pages in a file is less than the number of frames in the buffer pool.

D. A buffer pool has 50 frames. Suppose there are 100 read accesses made. 51 of those reads are the same page and the remaining 49 reads are each to distinct, unique pages. In this case, MRU has a better hit rate than LRU.

E. None of the above

☐ A

☐ B

■ C

☐ D

☐ E

i. False, a frame can be dirtied but left unpinned after the requester is finished using it.
ii. False, reference bits are used for clock policy
iii. True
iv. False, the 51 accesses to the same page will take up one spot in the buffer pool and the remaining 49 pages will each take up one spot in the buffer pool. Since there are 50 frames in the buffer pool, both MRU and LRU will have 0 misses.

(b) (2 pt) Suppose the buffer has 4 frames. Pages are loaded into the buffer in the following order:

A Z M N K A Z M N K A Z

Which buffer eviction policy would yield the best hit rate? Briefly explain your reasoning.

MRU

This is sequential flooding. Since the # of buffer frames < # unique page access, MRU will be the best policy. It is not enough to say that MRU will do better simply because this is a sequential scan.

(c) (6 points)

Assume the buffer pool from above has been cleared for all 4 frames. Now consider the following access pattern:

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

Calculate the hit rate (in the form 'X/14') using...

i. (2 pt) LRU

5/14

Cache hits on B (2nd), B (3rd), C (2nd), F (2nd), G (2nd)

ii. **(2 pt)** MRU

3/14

Cache hits on B (2nd), C (2nd), D (2nd)

iii. **(2 pt)** Clock Policy

6/14

Cache hits on B (2nd), B (3rd), C, (2nd), E (3rd), F (2nd), G (2nd)

2. (20 points) OK, BUT FIRST COFFEE

Looks like Airbears is down AGAIN, time to find a coffee shop to work with friends! Consider the following tables:

```
CREATE TABLE coffee_shops(
    name            VARCHAR PRIMARY KEY,
    max_capacity    INT NOT NULL,
    num_outlets     INT NOT NULL,
    has_wifi        BOOLEAN NOT NULL
);

CREATE TABLE yelp_reviews (
    review_id   INT PRIMARY KEY
    name        VARCHAR REFERENCES coffee_shops
    yelp_id     INT NOT NULL, /* User's yelp id */
    rating      FLOAT,
    review_time INT /* Total number of seconds since Jan 1, 1970 */
);

CREATE TABLE coffee_lovers (
    cid     INT PRIMARY KEY,
    cname   VARCHAR NOT NULL,
    yelp_id INT REFERENCES yelp_reviews
);

CREATE TABLE visits (
    visit_id    INT PRIMARY KEY,
    name        VARCHAR REFERENCES coffee_shops,
    cid         INT REFERENCES coffee_lovers,
    order_time  INT NOT NULL, /* Total number of seconds since Jan 1, 1970 */
    pickup_time INT NOT NULL, /* Total number of seconds since Jan 1, 1970 */
    date        VARCHAR NOT NULL /* Eg. '09/12/2021' */
);
```

(a) (2 points) Find Your Favorite Coffee Shop

After a year online, you couldn't recall the name of your favorite coffee shop. All you know is that it contains "blue" in the name, and it has plenty of outlets. Specifically, the ratio between the number of outlets and the maximum capacity is greater than 0.2. You write the following query to find coffee shops that satisfy these conditions.

```
SELECT *
FROM coffee_shops
WHERE ____(1)____
AND ____(2)____;
```

i. (1 pt) Select all possible options for blank (1), which finds coffee shops with "blue" in its name.

- ☐ A. `name LIKE 'blue'`
- ■ B. `name LIKE '%blue%'`
- ☐ C. `name LIKE '%_blue_%'`
- ☐ D. `name LIKE '_blue_'`
- ☐ E. `None of the above`

Please see slide 24 in discussion 1.

ii. (1 pt) Select all possible options for blank (2), which finds coffee shops with sufficient outlets.

- ☐ A. `num_outlets / max_capacity > 0.2`
- ■ B. `CAST(num_outlets AS FLOAT) / CAST(max_capacity AS FLOAT) > 0.2`
- ■ C. `CAST(num_outlets AS FLOAT) / max_capacity > 0.2`
- ☐ D. `CAST((num_outlets/ max_capacity) AS FLOAT)  > 0.2`
- ☐ E. `None of the above`

The left hand side of A,D can both be rounded down to 0 as num_outlets and max_capacity are both integers.

(b) (4 pt) Which of the following queries will find the number of people who visited each shop on 08/25/2021? Do not include shops nobody visited on that date in your output. **There could be zero, one, or more than one correct queries. In the case that there are zero correct queries, mark E None of the above.**

A.
```
SELECT coffee_shops.name, count(*)
FROM coffee_lovers c INNER JOIN
(coffee_shops NATURAL JOIN visits)
ON c.cid = visits.cid
WHERE visits.date = '08/25/2021'
GROUP BY coffee_shops.name;
```

B.
```
SELECT coffee_shops.name, count(*)
FROM coffee_lovers c INNER JOIN
(coffee_shops INNER JOIN visits ON coffee_shops.name = visits.name)
ON c.cid = visits.cid
WHERE visits.date = '08/25/2021'
GROUP BY coffee_shops.name;
```

C.
```
SELECT s.name, count(*)
FROM coffee_lovers c, coffee_shops s, visits v
WHERE c.cid = visits.cid AND s.name = v.name AND v.date = '08/25/2021'
GROUP BY c.cname
HAVING count(c.cname) > 0;
```

D.
```
SELECT coffee_shops.name, count(c.cid)
FROM coffee_lovers c INNER JOIN
(coffee_shops LEFT JOIN visits ON coffee_shops.name = visits.name)
ON c.cid = visits.cid
WHERE visits.date = '08/25/2021'
GROUP BY coffee_shops.name;
```

E. None of the above

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

C should not group by the customer/coffee_lover's name.

(c) (4 pt) Which of the following queries will find out ids and names of the coffee lovers who have visited any coffee shops on 09/12/2021? **There could be zero, one, or more than one correct queries. In the case that there are zero correct queries, mark E None of the above.**

A.
```
SELECT c.cid, c.cname
FROM coffee_lovers c INNER JOIN visits v
ON c.cid = v.cid
WHERE v.date = '09/12/2021';
```

B.
```
SELECT c.cid, c.cname
FROM coffee_lovers c INNER JOIN visits v
ON c.cid = v.cid
GROUP BY c.cid, c.cname
HAVING v.date = '09/12/2021';
```

C.
```
SELECT DISTINCT c.cid, c.cname
FROM coffee_lovers c, visits v
WHERE c.cid = v.cid AND v.date = '09/12/2021';
```

D.
```
SELECT c.cid, c.cname
FROM coffee_lovers c INNER JOIN visits v
ON c.cid = v.cid AND v.date = '09/12/2021'
GROUP BY c.cid, c.cname;
```

E. None of the above

☐ A

☐ B

■ C

■ D

☐ E

A is not correct because a coffee lover can visit multiple coffee shops on '09/12/2021' (partial credit given for choosing A). B is not correct because v.date = '09/12/2021' cannot be in a Having clause.

(d) (4 pt) You want to find out how a coffee shop's average customer rating across all of its yelp_reviews is related to its average waiting time across all of its visits, where the waiting time for one visit is defined as (pickup_time - order_time).

For each coffee shop, find the average waiting time and average rating from customers, where outliers with a waiting time above 20 minutes (1200 secs) are not included as part of the average. If a coffee shop has not been rated before, still include its average waiting time in the output and set average rating to NULL.

Which of the following queries give the correct output? **There could be zero, one, or more than one correct queries. In the case that there are zero correct queries, mark E None of the above.**

A.
```
SELECT v.name, avg(v.pickup_time - v.order_time) AS AW,
avg(r.rating) AS AR
FROM yelp_reviews r RIGHT OUTER JOIN visits v
ON v.name = r.name
WHERE v.pickup_time - v.order_time <= 1200
GROUP BY v.name;
```

B.
```
SELECT v.name, avg(v.pickup_time - v.order_time) AS AW,
avg(r.rating) AS AR
FROM yelp_reviews r RIGHT OUTER JOIN visits v
ON v.name = r.name
GROUP BY v.name
HAVING max(v.pickup_time - v.order_time) <= 1200;
```

C.
```
WITH
    avg_w AS (SELECT v.name AS coffee_shop,
                     avg(v.pickup_time - v.order_time) AS AW
              FROM visits v
              WHERE v.pickup_time - v.order_time <= 1200
              GROUP BY v.name),
    avg_r AS (SELECT r.name AS coffee_shop,
                      avg(r.rating) AS AR
              FROM yelp_reviews r
              GROUP BY r.name)
SELECT avg_w.coffee_shop, avg_w.AW, avg_r.AR
FROM avg_w LEFT OUTER JOIN avg_r
ON avg_w.coffee_shop = avg_r.coffee_shop;
```

D.
```
SELECT v.name AS coffee_shop,
avg(v.pickup_time - v.order_time) AS AW
FROM visits v
WHERE v.pickup_time - v.order_time <= 1200
GROUP BY v.name

UNION ALL

SELECT r.name AS coffee_shop, avg(r.rating) AS AR
FROM yelp_reviews r
GROUP BY r.name;
```

E. None of the above

■ A

☐ B

■ C

☐ D

☐ E

(e) **(4 pt)** Find all coffee lovers who have visited all coffee shops that have wifi. Fill out the blank in the following query to give the correct output.

```
SELECT c.cname
    FROM coffee_lovers c
    WHERE NOT EXISTS
        (SELECT *
            FROM coffee_shops s
            WHERE NOT EXISTS
                (SELECT *
                    FROM visits v
                    WHERE _____________ ));
```

```
s.name = v.name AND v.cid = c.cid OR s.has_wifi = false
```

There doesn't exist a coffee shop with wifi that this coffee lover hasn't visited. We gave full credit to everyone for this question because the above query will output everyone's name if no coffee shops have wifi. Should add "assume there is at least one coffee shop with wifi".

(f) **(2 pt)** Select all of the following answers that return the **name** (coffee shops) of all the yelp reviews that have rating greater than 4.5.

☐ A. $\pi\_\text{name}(\text{yelp\_reviews}) - \pi\_\text{name}(\sigma\_\text{rating} \leq 4.5(\text{yelp\_reviews}))$

☐ B. $\sigma\_\text{rating} > 4.5(\pi\_\text{name}(\text{yelp\_reviews}))$

■ C. $\pi\_\text{name}(\sigma\_\text{rating} > 4.5(\text{yelp\_reviews}))$

☐ D. $\pi\_\text{name, rating}(\text{yelp\_reviews}) - \sigma\_\text{rating} \leq 4.5(\pi\_\text{name}(\text{yelp\_reviews}))$

☐ E. None of the above.

A is incorrect because it excludes coffee shops with ratings both > 4.5 and $<= 4.5$. B, D are incorrect because the "rating" column is projected away so the selection operator will not be able to access it.

3. (17 points) Disks and Files

In his quest to become the world's best Pokemon Master, Ash Ketchum has been storing relevant information about other trainers through his battles and travels. Ash is interested in the different ways the data he has collected can be stored, read, and updated. To better understand these aspects, he reached out to CS 186 students to navigate the different implementation decisions!

For the rest of the questions consider the following schema:

```
CREATE TABLE Trainer(
    tid INTEGER PRIMARY KEY,
    age INTEGER NOT NULL,
    name VARCHAR(30),
    pokemon_count INTEGER NOT NULL,
    rank INTEGER NOT NULL,
    active BOOLEAN NOT NULL
);
```

For all parts of this question dealing with this table, assume the following information:

- Integers are 4 bytes and pointers are 4 bytes.
- **Record headers contain a bitmap.** Bitmaps should be as small as possible, rounded up to the nearest byte.
- Each data page is 1 KB (1 KB = 1024 B).
- The `Trainer` table is stored as a heap file using the page directory implementation.
  - Each header page has 6 entries. There are 100 data pages.
- If a question requires the use of slotted pages, the implementation of slotted pages will be the same as seen in lecture, discussion, and course notes.

(a) (2 pt) What is the maximum size of a record from the `Trainer` table?

52

The maximum size of a record in bytes is 52 bytes.

The breakdown for the calculation is as follows (1 + 4 + 4 + 4 + 30 + 4 + 4 + 1 = 52 bytes): 1 byte for the bitmap 4 bytes for the pointer to the end of `name` 4 bytes for `tid` 4 bytes for `age` 30 bytes for `name` 4 bytes for `pokemon_count` 4 bytes for `rank` 1 byte for `active`

(b) (2 pt) What is the maximum number of records that can fit onto one data page if variable length records are used?

33

In order to determine the maximum number of records that can fit onto a data page using variable length records, the ideal record size would be when the variable length fields are empty (the variable length fields take up 0 bytes).

Therefore, the size of a record will be 22 bytes (1 byte for the bitmap, 4 bytes for the variable length field's pointer, 4 bytes for `tid`, 4 bytes for `age`, 0 bytes for `name` (it is an empty string), 4 bytes for `pokemon_count`, 4 bytes for `rank`, 1 byte for `active`).

Recall from discussion when we are storing variable length records on a page, we use a slotted page approach. In discussion, slotted pages were implemented such that 4 bytes are used for a free space pointer, 4 bytes are used for the slot count, and then each record would have a [record length, record pointer] (8 bytes).

Therefore, the number of records that can be stored on one page is as follows: Floor((1024 - 8) / (22 + 8)) = 33 records.

(c) (3 pt) For the remaining questions, assume the queries are run independent of each other and that the records are stored as variable length records. Also, the buffer is large enough to hold all data and header pages, and starts empty for each question.

What is the worst case cost in I/Os for the following query?

```
UPDATE Trainer SET rank = rank + 1 WHERE active = True;
```

217

In the worst case scenario, all the records will have `active` toggled to true.

As a result, the header pages must be read in and the data pages must be read in entirety and then updated as well. There are ceil(100/6) = 17 header pages and there are 100 data pages in total. Therefore, the I/O cost will be 100*2 + 17 = 217 I/Os.

(d) (3 pt) What is the best case cost in I/Os for the following query?

```
SELECT name FROM Trainer WHERE tid > 186 AND tid <= 286;
```

5

The following query is performing a range search on the primary key of the `Trainer` table. Given that `tid` will be unique for every entry, it follows that 100 records will fall into this range.

In the previous question, we found that 33 records can fit onto a page therefore, Ceil(100/33) = 4 pages is how many pages will be needed to store all 100 records. The best case is when all the records are in the first 4 pages. This incurs 1 I/O to read the header page and then 4 I/Os to read in the data pages. Therefore, 5 I/Os.

(e) (3 pt) What is the worst case cost in I/Os for the following query?

```
INSERT INTO Trainers VALUES (186, 23, ‘‘Bartholomew’’, 5, 10, True);
```

119

While this question asks for the worst case, it is important to realize that inserting a new record will require checking that the `tid` is unique (remember `tid` is the primary key of the table). As a result, all data pages must be checked to ensure there is no duplicate for the primary key value provided.

There are ceil(100/6) = 17 header pages and there are 100 data pages in total. Therefore the I/O cost will be 17 I/Os to read in the header pages, 100 I/Os to read in the data pages, 1 I/O to write to a data page with free space, and 1 I/O to update the page header with the relevant metadata. Therefore, the total cost will be 119 I/Os.

(f) (4 pt) Select all the statements below that are true. **There may be zero, one, or more than one correct answer.**

A. Suppose there exists an index on some column A and it happens to be the primary key of the table. It will always be slower to perform equality search on column A on a heap file representation as opposed to a sorted file.

B. A packed page layout with fixed length records requires a bitmap to keep track of the records being stored in the page.

C. In the worst case, the linked list implementation of a heap file incurs less cost to find a page with enough free space than the page directory implementation.

D. Heap files are always faster to insert a new record into when compared to a sorted file.

E. None of the above

☐ A

☐ B

☐ C

☐ D

■ E

A. False; it is not necessarily always slower. Consider the scenario when the record in question is contained on the first page.

B. False; a packed page layout may use a bitmap, but it is not required. With a packed page layout with fixed length records, a record can always be found using the size of the fixed length records and the record id. Given that the file size and the record length would be known, we would also know exactly how many records can fit on this page beforehand.

C. False; the page directory implementation allows you to read in header pages and identify which pages have enough free space, whereas a linked linked implementation would require traversing all the data pages labeled as containing free space. Given the assumption, we know that there in general we will have fewer header pages than data pages.

D. False; Inserting into an empty heap file is not necessarily faster than inserting into a sorted file.

4. (28 points) B+ Trees

Let's build indices! Please keep in mind that some questions have multiple parts so budget your time accordingly.

(a) (2 pt) Which of the following are true? Select all that apply. There may be more than one correct answer.

A. B+ Trees will speed up queries by reducing the big O runtime of searching for records on a page in memory.

B. B+ Trees can possibly increase the I/O cost of a query.

C. B+ Trees can speed up queries by reducing the number of data pages read into memory.

D. B+ Trees can only be built on columns with type INTEGER or FLOAT.

E. None of the above.

☐ A

■ B

■ C

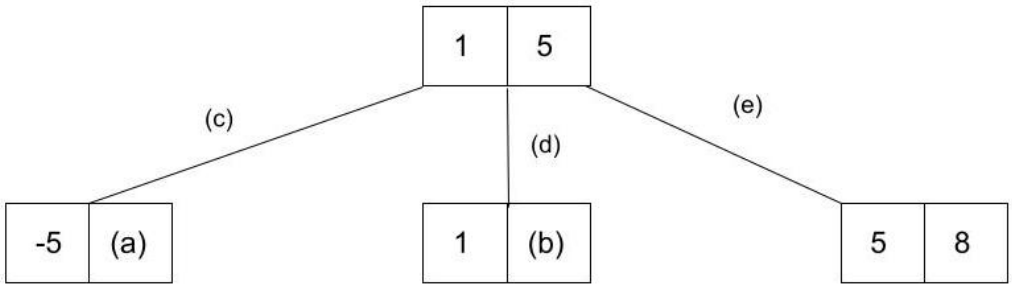
☐ D

☐ E

A - B+ Trees can improve the number of I/Os incurred but don't affect the in-memory big O runtime. B - One example: if we're searching for a single record and it's on the very first page, reading in nodes in the B+ Tree will be more costly. In general, indices will help but not always! C - Instead of searching through all the pages in the heapfile for a record or range of records, we can use a B+ Tree to find the corresponding page more quickly, saving I/Os. D - We can build indices on any type of column.

(b) (3 points) B+ Tree Searches and Inserts

Given the tree below



Answer the following questions.

i. (1 pt) We want to search for key 3. Which of the following edges would we traverse?

○ c

● d

○ e

○ None of the above

Key 3 is between 1 and 5 so we search the middle child.

ii. (1 pt) We want to search for key 10. Which of the following edges would we traverse?

- ○ c
- ○ d
- ● e
- ○ None of the above

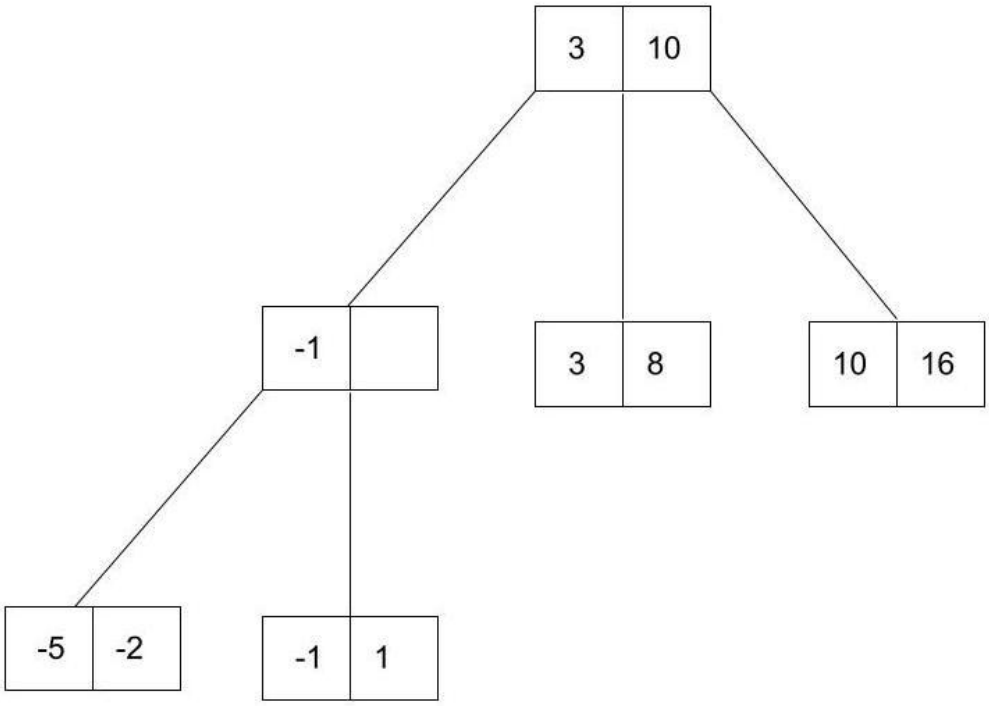
Key 10 is larger than 10 so we search the rightmost child.

iii. (1 pt) We want to insert key 4. Where would it be inserted into?

- ○ a
- ● b
- ○ None of the above

Key 4 is between 1 and 5 so we insert into the middle child.

(c) (1 pt) Is the following B+ Tree possible assuming no deletions and no rebalancing?



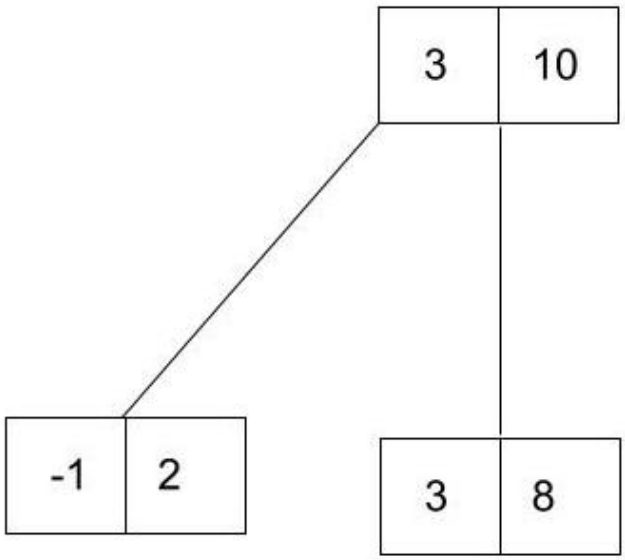
○ Yes

● No

(d) (1 pt) Briefly justify your answer above in 1 sentence.

BPlus trees are balanced and grow from the root.

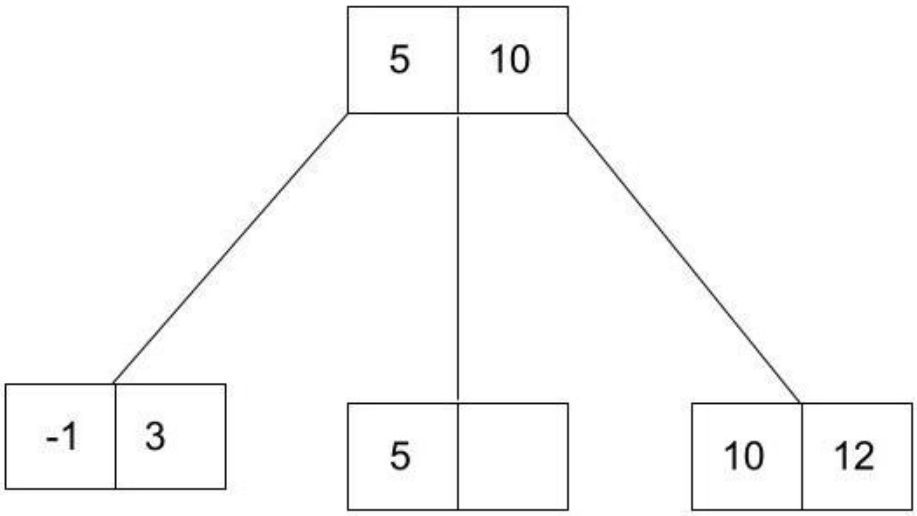
(e) (1 pt) Is the following B+ Tree possible assuming no deletions and no rebalancing?



- ○ Yes
- ● No

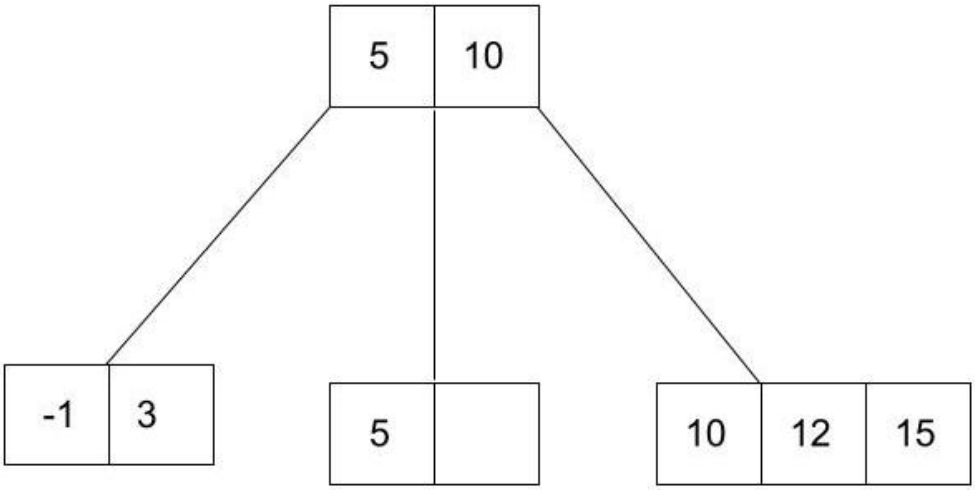
(f) (1 pt) Briefly justify your answer above in 1 sentence.

Key 10 could not have ended up in the inner/root node unless it was copied up from a leaf node.

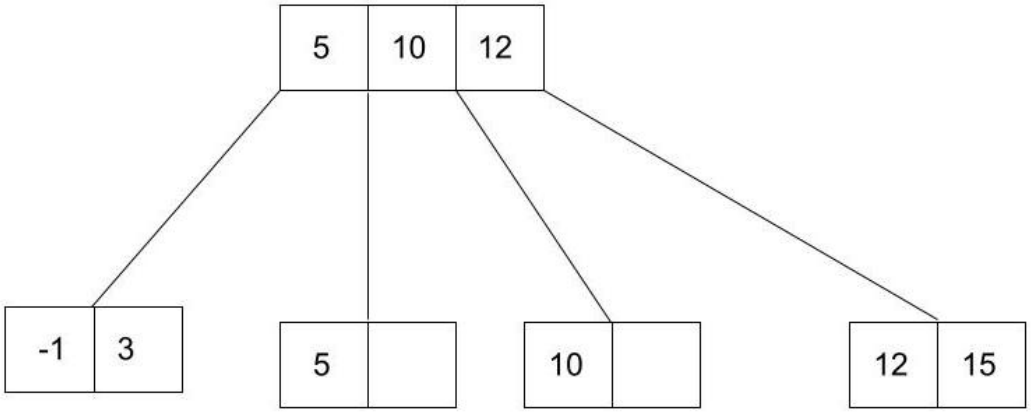


(g) (1 pt)

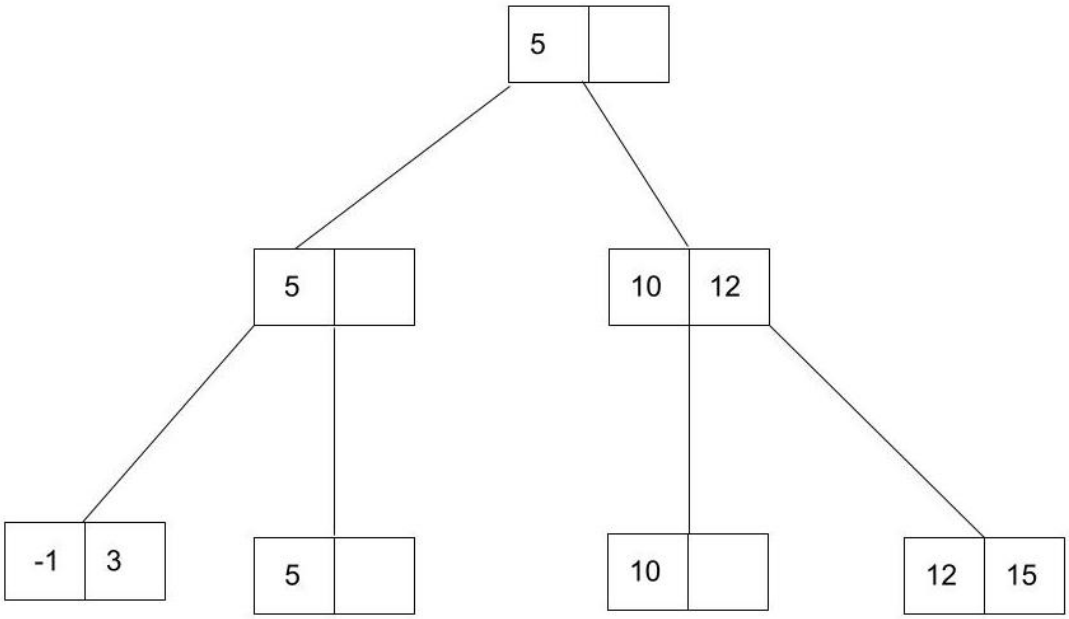
Given the above order 1 B+ Tree, what is the result after inserting 15?



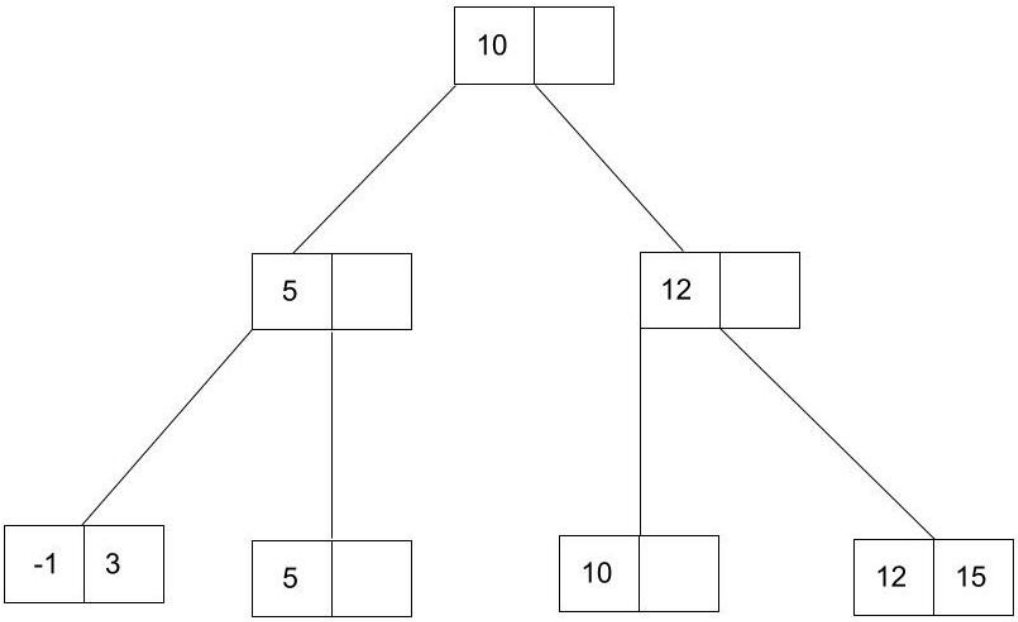
A.



B.



C.

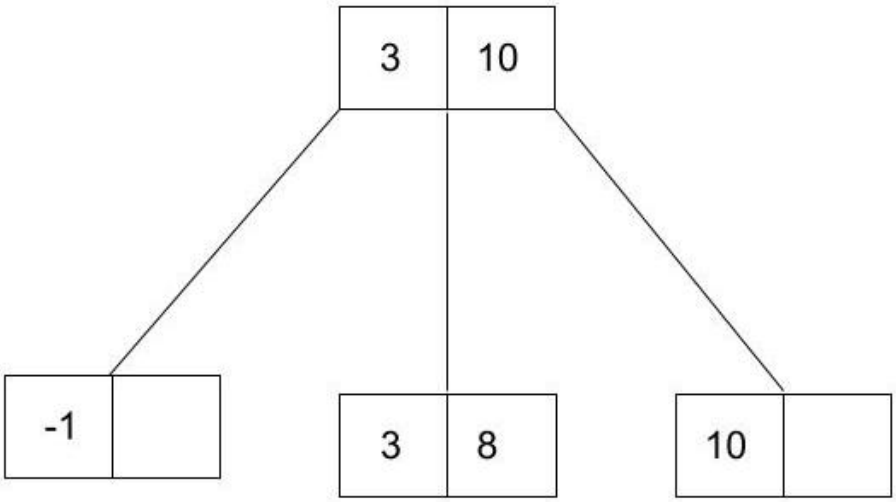


D.

- ○ A
- ○ B
- ○ C
- ● D

15 is inserted into the rightmost child, causing it to overflow and split. 12, the middle key, is then copied up to the root node which also overflows and splits. 10, the middle key of the root, is pushed up to be the new root.

(h) (1 pt) Give an example of a key we insert to increase the height of the order 1 tree below? Write N/A if it is not possible to increase the height of the tree in one insertion. Assume that no duplicates are allowed in this B+ Tree.



Any key in [4, 9] will cause the middle child to split, causing a key to be copied up to the root. This causes the root to also split.

(i) (15 points) Lakshya's Jokes

Lakshya does stand up comedy in his free time and has a table to store all of his jokes.

```
CREATE TABLE jokes (
    joke_id INTEGER PRIMARY KEY,
    topic TEXT,
    description TEXT,
    rating INTEGER
    num_laughs INTEGER
);
```

Since his table is getting very large, he decides to build the two following indices to speed up his queries.

Index A:

- Alternative 1 B+ Tree on column `topic`
- Order 3 (d=3)
- Height 4

Index B:

- Alternative 2 unclustered B+ Tree on column `rating`
- Order 2 (d=2)
- Height 3

Assumptions:

- There are 5 records where topic='parrots'

- There are 15 records where rating > 80
- There are 8 records where topic='databases'
- The root is NOT cached in memory.
- The buffer manager starts empty at the start of each query.
- The table contains a total of 100 data pages.
- One data page can store up to 10 records.
- Assume the leaf nodes are full.

Answer the following questions. Please pay attention to whether we're looking for **best case (minimum I/O cost)** or **worst case**.

i. (1 pt) Do we need to duplicate the data pages between Index A and Index B?

◯ Yes

● No

No, Index A needs to be sorted, but B does not need to be sorted because it's unclustered.

ii. (2 pt) What is the **minimum** I/O cost for the following query?

```
SELECT * FROM jokes;
```

100

Read all data pages. Credit also given for 104

iii. (2 pt) What is the **minimum** I/O cost for the following query?

```
SELECT * FROM jokes WHERE topic='parrots';
```

5

4 I/Os to read the inner nodes + 1 I/O to read the leaf node which contains the data in an Alt 1 tree. In the best/minimum cost case, all the data will fit onto one leaf page since we assume 5 matches.

iv. (1 pt) Is the above query sped up by either index?

● Yes

◯ No

v. (2 pt) What is the **worst case** I/O cost for the following query?

```
SELECT * FROM jokes WHERE rating > 80;
```

23

3 I/Os to read the inner nodes. In the worst case, the 15 matches will be split across 5 leaf pages so 5 I/Os to read the leaf nodes. Since it's an unclustered index, 1 I/O per matching record. In the worst case, the 15 matching records will each be on a different page.

vi. (1 pt) Is the above query sped up by either index?

● Yes

◯ No

vii. **(2 pt)** What is the **worst case** I/O cost for the following query?

```
SELECT * FROM jokes WHERE num_laughs > 10 AND topic='databases';
```

6

4 I/Os to read the inner nodes. In the worst case, the 8 matching records will be split across 2 data pages so 2 I/Os to read these two leaf pages.

viii. **(1 pt)** Is the above query sped up by either index?

- (●) Yes
- ( ) No

ix. **(2 pt)** What is the **minimum** I/O cost for the following query?

```
SELECT * FROM jokes WHERE num_laughs > 10 OR topic='databases';
```

100

We can't use Index A to help us with this query. We need to check all records since the predicate contains an OR. Read all data pages. Credit also given for 104

x. **(1 pt)** Is the above query sped up by either index?

- ( ) Yes
- (●) No

(j) **(2 pt)** Suppose we want a tree of height 2. How many records would we need to insert when bulk loading with a fill factor of 3/4 given d=4?

55

After 6 insertions, we fill up the initial leaf node to the fill factor. On the 7th insertion, it splits and we get a height 1 tree. After a total of 54 insertions, we have 9 leaf nodes filled to the fill factor. On the 55 insertion, the leaf node splits and the height increases to 2.

5. (15 points) Hashing

We have a file with N=100 pages and a machine with B=5 buffer pages, and we want to hash our file. For the first pass, you have a strange hash function. The hash function used sends half the data to the first bucket. Of the remaining data, 60% goes to the second bucket. The remaining buckets split the remainder of the data equally. Following that, you use a (different) uniform hash function for each of the remaining passes. Assume that all the data in the file is unique with no duplicates on the key being hashed on.

(a) (0.5 pt) How many I/Os are performed when you read in the data pages for the first time?

100

(b) (0.5 pt) How many partitions are created?

4

(c) (0.5 pt) How many I/Os are performed in writing these partitions to disk?

100

(d) (0.5 pt) How many pages went into the largest partition?

50

$\frac{1}{2}$, 3/10, 1/10, 1/10 is the distribution function, meaning you get [50, 30, 10, 10] as the pages per bucket.

(e) (1 pt) How many more passes will it take to hash this partition, including the final build phase?

3

50 pages get split into 4 partitions of 13. Each partition of 13 is broken into 4 partitions of 4. An in-memory hash table is constructed on each of those 4-page partitions.

(f) (3 pt) How many I/Os will it take to hash this partition after it was initially created/written to disk?

346

50 pages get split into 4 partitions of 13. Each partition of 13 is broken into 4 partitions of 4. An in-memory hash table is constructed on each of those 4-page partitions. So our calculation would be ((50) + 13 * 4) + 4 * ((13 + 4 * 4) + 4 * (4 + 4)) = 346

(g) (0.5 pt) We're done with calculating I/Os for hashing the first partition. Let's move on to the second partition. How many pages will go into this partition?

30

$\frac{1}{2}$, 3/10, 1/10, 1/10 is the distribution function, meaning you get [50, 30, 10, 10] as the pages per bucket.

(h) **(1 pt)** How many more passes does it take to hash this partition, including the final build phase?

3

30 pages get split into 4 partitions of 8. Each partition of 8 gets split into 4 partitions of 2. An in-memory hash table is constructed on each of those 2-page partitions in the third pass.

(i) **(3 pt)** How many I/Os will it take to hash this partition after it was initially created/written to disk?

190

30 pages get split into 4 partitions of 8. Each partition of 8 gets split into 4 partitions of 2. An in-memory hash table is constructed on each of those 2-page partitions in the third pass. So our calculation would be (30 + 8 * 4) + 4 * (8 + 2 * 4 + 4 * (2 + 2)) = 190

(j) **(0.5 pt)** We're done with calculating the I/Os for the second partition. Now, let's deal with the remaining partitions. How many pages go into each of the remaining partitions?

10

(k) **(1 pt)** How many passes does it take to hash one of these partitions, including the final build phase?

2

10 pages get split into 4 partitions of 3, and each of those mini-partitions has an in-memory hash table built on it.

(l) **(2 pt)** How many I/Os will it take to hash one of these partitions after it was initially created and written to disk at the end of pass 0?

46

10 pages get split into 4 partitions of 3, and each of those mini-partitions has an in-memory hash table built on it. Thus, our calculation is (10 + 3 * 4) + 4 * (3 + 3) = 46

(m) **(1 pt)** Now add them all up. How many total I/Os will it take to hash this file, from start to finish?

828

100 + 100 + 346 + 190 + 46 * 2 = 828

6. (8 points) SORTING

Now, let's try to sort the file of N=100 pages with B=5 buffer pages.

(a) (0.5 pt) How many sorted runs will we create in pass 0?

20

100/5 = 20

(b) (0.5 pt) How many pages will be in each run?

5

(c) (1 pt) How many passes will it take to sort this file?

4

1 + ceil($log_4 20$) = 4

(d) (1 pt) How many I/Os will it take to sort this file?

800

2 * N * # of passes = 2 * 100 * 4 = 800

(e) (0.5 pt) We're now given one extra buffer page, so we have B=6 buffer pages. If we were to sort an N=100 pages file with B=6 buffer pages, how many sorted runs will we create in pass 0?

17 sorted runs

ceil(100/6) = 17 sorted runs

(f) (1 pt) And how many passes would it take to sort the file?

3 passes

1 + ceil($log_5 17$) = 3

(g) (0.5 pt) How many I/Os would it take to sort the file now?

600

100 * 2 * 3 = 600

(h) **(1 pt)** What is the minimum number of buffer pages we would need in order to completely sort this file in exactly two passes?

11

We must have, at most, B-1 runs after pass 0. So, B * (B-1) must be greater than or equal to 100, which means B must be at least 11.

(i) **(1 pt)** What is the minimum number of buffer pages we would need to completely sort this file in only one pass?

100

We'd need to be able to fit all 100 pages in memory at once to sort this in one pass.

(j) **(1 pt)** If we have 5 buffer pages in pass 0, 4 buffer pages in pass 1, and 3 buffer pages in pass 2, what is the maximum file size in number of pages we can sort in exactly 3 passes?

30

5 * 3 * 2 = 30.

B * (B-1) * (B-1), where B in pass 0 is 5, B in pass 1 is 4, and B in pass 2 is 3. Thus, we have 5 * 3 * 2 = 30.

No more questions.