CS W186 Spring 2020 Final (Online)

Do not share this exam until solutions are released.

Contents:

- The final has *11 questions*, each with multiple parts, and worth a total of *174.5 points*.

Taking the exam:

- You have *180 minutes* to complete the final.
- You may print this exam or download it onto an electronic device to work on it.
- For each question, submit only your *final answer* on Gradescope.
- 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) and do not use LaTeX.
- Questions tagged with [**EXPLANATION**] require you to show work and not doing so will result in **no credit**. You can do this by inputting an explanation in the text field or submitting a file (photo, PDF) of your work. The text field supports LaTeX by using $$insert expression here$$.
- Make sure to submit on Gradescope at the end of the exam.

Aids:

- This exam is open-book, open-calculator, and open-Internet.
- **You must work individually on this exam.**

Grading Notes:

- All I/Os must be wcoverritten 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 Pre-Exam (1 point)

1. (1 point) Please read and sign the Honor Code Statement on Gradescope.

2 Separated Quarantine Life (19.5 points)

You did a great job setting up the student roster with Octograder in the beginning of the semester, so the CS W186 TAs find you again, this time to help generate project grades. They hand you the following schema:

```
1  CREATE TABLE students (
2    name VARCHAR,
3    sid INTEGER UNIQUE,
4    edx_id INTEGER UNIQUE,
5    email VARCHAR UNIQUE
6  );
7
8  CREATE TABLE assignments (
9    assignment_name VARCHAR,
10   profile VARCHAR,
11   assignment_group VARCHAR,
12   weight FLOAT,
13   due_date TIMESTAMP,
14   PRIMARY KEY(assignment_name, profile)
15 );
16
17 CREATE TABLE results (
18   submission_id INTEGER PRIMARY KEY,
19   edx_id INTEGER REFERENCES students,
20   assignment_name VARCHAR REFERENCES assignments,
21   profile VARCHAR,
22   submission_time TIMESTAMP,
23   score FLOAT
24 );
25
```

Throughout this question, assume the following:

- `students` contains all enrolled students. Each student has unique `sid`, `edx_id`, and `email`, but the database uses `edx_id` as the unique identifier for students.
- `assignment_group` represents the 5 projects in this class: it is one of proj1, proj2, proj3, proj4, proj5.
- `assignment_name` is the parameter that you have to enter when running the `turn_in.py` script: it is one of proj1, proj2, proj3_part1, proj3_part2, proj4_part1, proj4_part2, proj4_part3, proj5.
- `profile` is either public or hidden.
- `results` contains two rows for every student submission: one with public profile and one with hidden profile for that `assignment_name`.
- `score` and `weight` are both between 0 and 1, inclusive. The `weight`s of all (`assignment_name`, `profile`) pairs for an `assignment_group` add up to 1. Given the submissions of a project from a student, the score is the sum of `score` * `weight` for all (`assignment_name`, `profile`) pairs for that `assignment_group`, which gives a score out of 1.
- We only deal with the raw `score`s; ignore the effects of late submissions, academic dishonesty, etc.

1. (4 points) [**EXPLANATION**] You try to build the schema, but you get a warning saying `Schema Error`. Fix the schema so that there are no errors. Your fix should stick to the assumptions from the previous page.

   Make sure to clearly state which line(s) you are changing, some examples of valid answers: "add ... after line 4", "change line 17 to ...", "delete ... from line 24".

   (Note: there are two schema errors and three changes to make. You need all of them to get full credit.)

   Solution: Add `Primary Key` to line 4. Delete `REFERENCES assignments` from line 20 and add `FOREIGN KEY (assignment_name, profile) REFERENCES assignments` after line 23.

The schema is fixed, and you are ready to generate project grades. Due to the difficulties in getting help in OH because of the special circumstances of remote learning this semester, the TAs try to come up with ways to help student grades. Currently there are four proposals in how to generate the total project grades:

Proposal 1: Stick with the current policy: for every student, use the latest submission for every assignment and sum them up to get a score out of 5.

Proposal 2: For every student, use the highest score from every (`assignment_name`, `profile`) pair and sum them up to get a score out of 5.

Proposal 3: For every student, use the latest submission for every assignment to get 5 scores for the 5 projects, then drop the lowest one, to get a score out of 4.

Proposal 4: Grade on completion: for every student, give 1 if they submitted anything for a project and 0 otherwise, and sum them up to get a score out of 5.

For questions 2 to 8, select a proposal if the query outputs result for that proposal. The output should have two columns, `edx_id` and total score, and it should have one row and only one row for anyone who has made at least one submission for any assignment. The students who didn't submit anything (i.e. `results` doesn't have any row for them) don't need to be in the output.

Select "Error" if the query errors, or select "Incorrect Result" if the query outputs some results but don't match any of the proposals.

2. (1.5 points) Query:

```
SELECT edx_id, SUM(score * weight)
FROM results R1, assignments A
WHERE R1.assignment_name = A.assignment_name and R1.profile = A.profile
AND submission_time >=
(
  SELECT MAX(submission_time)
  FROM results R2
  WHERE R1.assignment_name = R2.assignment_name AND R1.profile = R2.profile
)
GROUP BY edx_id;
```

A. Proposal 1: latest submission
B. Proposal 2: highest score from every (`assignment_name`, `profile`) pair
C. Proposal 3: latest submission and drop lowest
D. Proposal 4: completion
E. Error
F. Incorrect Result

Solution: F. Inner query doesn't filter `edx_id`, so the `submission_time` is compared to other students' time too, so the result might be missing rows.

3. (1.5 points) Query:

```
SELECT edx_id, SUM(score)
FROM
(
  SELECT edx_id, assignment_name, 1 as score
  FROM results R, assignments A
  WHERE R.assignment_name = A.assignment_name AND R.profile = A.profile
  GROUP BY edx_id, assignment_name
) S
GROUP BY edx_id;
```

A. Proposal 1: latest submission
B. Proposal 2: highest score from every (`assignment_name`, `profile`) pair
C. Proposal 3: latest submission and drop lowest
D. Proposal 4: completion
E. Error
F. Incorrect Result

Solution: E. Query Error: column reference `assignment_name` is ambiguous.

4. (1.5 points) Query:

```
SELECT edx_id, SUM(MAX(score*weight))
FROM assignments A, results R
WHERE A.assignment_name = R.assignment_name AND A.profile = R.profile
GROUP BY edx_id, assignment_name, profile;
```

A. Proposal 1: latest submission
B. Proposal 2: highest score from every (`assignment_name`, `profile`) pair
C. Proposal 3: latest submission and drop lowest
D. Proposal 4: completion
E. Error
F. Incorrect Result

Solution: E. Query Error: aggregate function calls cannot be nested.

5. (1.5 points) Query:

```
WITH temp AS (
  SELECT edx_id, assignment_name, profile, score
  FROM results R1
  WHERE submission_time >= ALL
  (
    SELECT submission_time
    FROM results R2
    WHERE R1.edx_id = R2.edx_id AND R1.assignment_name = R2.assignment_name
    AND R1.profile = R2.profile
  )
)
SELECT edx_id, SUM(score * weight)
FROM temp T, assignments A
WHERE T.assignment_name = A.assignment_name AND T.profile = A.profile
GROUP BY edx_id;
```

A. Proposal 1: latest submission
B. Proposal 2: highest score from every (`assignment_name`, `profile`) pair
C. Proposal 3: latest submission and drop lowest
D. Proposal 4: completion
E. Error
F. Incorrect Result

Solution: A.

6. (1.5 points) Query:

```
SELECT edx_id, SUM(weighted_score)
FROM
(
  SELECT R.edx_id, R.assignment_name, R.profile, MAX(score * weight) AS weighted_score
  FROM assignments A, results R
  WHERE A.assignment_name = R.assignment_name AND A.profile = R.profile
  GROUP BY R.edx_id, R.assignment_name, R.profile
) S
GROUP BY edx_id;
```

A. Proposal 1: latest submission
B. Proposal 2: highest score from every (`assignment_name`, `profile`) pair
C. Proposal 3: latest submission and drop lowest
D. Proposal 4: completion
E. Error
F. Incorrect Result

Solution: B.

7. (1.5 points) Query:

```
WITH temp(edx_id, assignment_group, weighted_score) AS (
  SELECT edx_id, assignment_group, SUM(score * weight)
  FROM results R1, assignments A
  WHERE R1.assignment_name = A.assignment_name AND R1.profile = A.profile AND submission_time >=
  (
    SELECT MAX(submission_time)
    FROM results R2
    WHERE R1.edx_id = R2.edx_id AND R1.assignment_name = R2.assignment_name
    AND R1.profile = R2.profile
  )
  GROUP BY edx_id
)
SELECT edx_id, SUM(weighted_score)
FROM temp t1
WHERE t1.weighted_score > ANY
(
  SELECT t2.weighted_score
  FROM temp t2
  WHERE t1.edx_id = t2.edx_id
)
GROUP BY edx_id;
```

A. Proposal 1: latest submission
B. Proposal 2: highest score from every (`assignment_name`, `profile`) pair
C. Proposal 3: latest submission and drop lowest
D. Proposal 4: completion
E. Error
F. Incorrect Result

Solution: E. Query Error: column `a.assignment_group` must appear in the GROUP BY clause or be used in an aggregate function.

8. (1.5 points) Query:

```
SELECT edx_id, SUM(weighted_score) FROM
(
  SELECT R.edx_id, R.assignment_name, R.profile, score * weight AS weighted_score
  FROM assignments A, results R
  WHERE A.assignment_name = R.assignment_name AND A.profile = R.profile

  EXCEPT

  SELECT R.edx_id, R.assignment_name, R.profile, score * weight AS weighted_score
  FROM assignments A, results R
  WHERE A.assignment_name = R.assignment_name AND A.profile = R.profile AND EXISTS
  (
    SELECT *
    FROM results R2
    WHERE R2.edx_id = R.edx_id AND R2.assignment_name = R.assignment_name AND R2.profile =
R.profile AND R2.submission_time > R.submission_time
  )
) S
GROUP BY edx_id;
```

   A. Proposal 1: latest submission
   B. Proposal 2: highest score from every (`assignment_name`, `profile`) pair
   C. Proposal 3: latest submission and drop lowest
   D. Proposal 4: completion
   E. Error
   F. Incorrect Result

Solution: A.

After an hour of meeting over Zoom, the TAs still cannot decide on a proposal. The professors decide to be nice and take the maximum score of using proposals 1, 2, and 3, after scaling them to be out of 1 (unfortunately, proposal 4 was eliminated from consideration). The scores from proposals 1, 2, and 3 as described above are already generated. The scores from proposals 1 and 2 are divided by 5, and scores from proposal 3 are divided by 4, so that they are all average project scores out of 1. They are stored in relations `P1(edx_id, weighted_score)`, `P2(edx_id, weighted_score)`, and `P3(edx_id, weighted_score)` for proposals 1, 2, 3, respectively.

9. (5 points) **[EXPLANATION]** Select the queries that output the name, email, and the maximum weighted score from proposals 1, 2, and 3. The output should have exactly one row for every enrolled student. For students who didn't submit anything (i.e. they are not in `P1`, `P2`, or `P3`), they should have a score of 0 in the output.

   For the queries that are not selected, briefly explain what is wrong (i.e. what error it throws or how is the output different from the expected output). No need to have explanation for the correct queries.

   There may be zero, one, or multiple correct answers.

A.

```
SELECT name, email, GREATEST(P1.weighted_score, P2.weighted_score, P3.weighted_score)
FROM students S
LEFT OUTER JOIN P1 ON S.edx_id = P1.edx_id
LEFT OUTER JOIN P2 ON S.edx_id = P2.edx_id
LEFT OUTER JOIN P3 ON S.edx_id = P3.edx_id;
```

B.

```
SELECT name, email, GREATEST(P1.weighted_score, P2.weighted_score, P3.weighted_score)
FROM students S
FULL OUTER JOIN P1 ON S.edx_id = P1.edx_id
FULL OUTER JOIN P2 ON S.edx_id = P2.edx_id
FULL OUTER JOIN P3 ON S.edx_id = P3.edx_id

UNION

SELECT name, email, 0
FROM students S
LEFT OUTER JOIN P1 ON S.edx_id = P1.edx_id
WHERE P1.weighted_score IS NULL;
```

C.

```
SELECT name, email, GREATEST(P1.weighted_score, P2.weighted_score, P3.weighted_score)
FROM students S, P1, P2, P3
WHERE S.edx_id = P1.edx_id AND S.edx_id = P2.edx_id AND S.edx_id = P3.edx_id

UNION

SELECT name, email, 0
FROM students
WHERE edx_id NOT IN
(
  SELECT edx_id
  FROM P1
);
```

Solution: C.

A has `NULL` for students who didn't submit anything. B also has rows with `NULL` in addition to rows with 0 for students who didn't submit anything.

3 Maintaining (at Least 6 ft. of) Free Space (14 points)

On Netflick's new spin-off reality show, *Too Sick to Handle*, contestants compete to see if they can hone their apocalypse survival skills. $\$10,000,000$ is in the jackpot, as long as contestants obey one rule: during the duration of the month-long "Quarantine", they must stay at least 6 feet apart from other people at all times. To make sure the contestants don't break the rules, the producers of *Too Sick to Handle* employ an all-knowing AI, Dana, to watch over them, and have asked you to help program her.

In Dana's database of contestants, we start with the following schema.

```
CREATE TABLE Contestants (
    cid INTEGER PRIMARY KEY,
    firstName CHAR[15],
    lastName CHAR[15]
);
```

Throughout this question, assume integers are 4 bytes each and pages are 1 KiB (1024 B) each.

1. (1 point) How many contestants records can fit on each page? Page headers consist of a bitmap and an integer to store the bitmap's length.

   Solution: 29 records. Each record takes up 32 + 15*8 + 15*8 + 1 = 273 bits for the 3 fields and it's corresponding bit in the bitmap. The page is 1 KiB = 1024 B, so accounting for the page header, we have $\lfloor (1024\text{B} - 4\text{B}) * 8/273 \rfloor = 29$ records.

2. (2 points) Assume now that each contestant record is 12B, and we have unpacked page with bits [0, 1, 2, 3, 4, 5, 6, 9] in the bitmap on (signalling presence of a record) and an integer to store the bitmap's length. Rounded down to the nearest byte, how much free space, not including the bitmap, is left on the page in bytes?

   Solution: 913B. We start by calculating the number of records on each page, which is just $\lfloor (1024\text{B} - 4\text{B}) * 8/(12 * 8 + 1) \rfloor = 84$ records. We thus have a 4 byte integer in the header, 84 bit bitmap, and 12*8 bytes of records. Subtracting that from the total size, we have $(1024\text{B} - 4\text{B}) * 8 - 84 - (12\text{B} * 8 \text{ records}) * 8$ bits, which, rounded down, is equal to 913 B.

There should be a deduction from the prize pool whenever the AI Dana catches contestants incurring an infraction. You decide to add some more info for each contestant, which is as follows:

```
CREATE TABLE Contestants (
    cid INTEGER PRIMARY KEY,
    firstName CHAR[15],
    lastName CHAR[15],
    occupation VARCHAR,
    address VARCHAR,
    numInfractions INTEGER
);
```

For problems 3-7, assume that each unpacked page's footer contains an 10-byte area for the record count and pointer to free space, as well as a slot directory where it takes 4 bytes per record to store the pointer/offset (in bytes) to the record in the page and another 4 bytes per record to store the length (in bytes) of that record.

3. (1 point) Assume each record header uses 1 byte pointers. What is the minimum possible record size in bytes?

   Solution: 40 bytes. There are 2 bytes in the record header for the variable length fields, so 2 + 4 + 15 + 15 + 4 = 40 bytes.

4. (2 points) **[EXPLANATION]** How much total free space does the following page have? Assume we have a slot directory independent of the schema above that contains of the following entries: [0, 100], [100, 100], [200, 100], [300, 100], [400, 100], [500, 20], [544, 256], [800, 108].

   Solution: 66 bytes. To calculate the amount of free space at the end, we have 1024 (page size) - 10 (slot directory's record count and pointer to free space) - 8 (records total) * (4 + 4 per record in the slot directory) - 908 (where the last record ends) = 42 Bytes. Adding the 24 bytes of free space between the sixth and seventh records, there are 66 bytes of free space total.

5. (1 point) True or False: Using the slot directory in question 4, we can insert a record of total length 44B in the page without any reorganization.

   Solution: False. There is enough total space on the page, but there is fragmentation, so we can't insert the record without repacking the page.

6. (1 point) True or False: Using the slot directory in question 4, we can insert a record of total length 38B in the page without any reorganization.

   Solution: False. A record of 38 bytes will fit in after the last current record in the page, but it requires 4 + 4 more bytes in the slot directory, which brings it to 38 + 4 + 4 = 46 B > 42 B (largest consecutive free space).

7. (1 point) What is the minimum number of records we would have to delete from the page to have enough space to insert a record of length 384B?

   Solution: 2 records. Delete the records of length 256 and 108 to get a consecutive free space of 24 + 256 + 108 + 42 = 430B.

In designing the AI Dana, you also consider the pros and cons of file organizations by comparing a page directory with a linked list implementation.

8. (3 points) [**EXPLANATION**] Suppose that you want to insert a record into your table, and in your page directory implementation, the first page with free space for the record is referenced in the 10th header page. In order for a linked list implementation to have a faster insert (in terms of I/O's) even with the **worst** case number of I/Os, which page (e.g. 1st/2nd/3rd) should be the **first** page in the linked list that has enough free space for the record? Assume buffer memory is not an issue and full pages in the linked list implementation are added to the beginning of the list for efficiency. Write your answer as a number (i.e. if it's the 3rd page, write 3).

Solution: The 4th page. Inserting a page in the page directory page takes 13 I/Os:

10 (to read header pages) + 2 (to read write data on the page) + 1 (to update the header page).

In the worst case, inserting a page in the linked list will require finding the correct page, writing the record, adjusting the pointers in the free pages linked list, and inserting the page at beginning of the full pages linked list:

1 (read the header page) + X (read the free pages linked list) + 1 (write the record, update pointers on the page) + 3 (adjust the previous and next pointers in the free page list) + 1 (update pointer in the header) + 2 (update the first page in the full pages linked list) = X + 8 I/Os.

If the 4th page has just enough space for the inserted record, this comes out to 12 I/Os for the linked list implementation.

The producers tell you that the more contestants you have, the more interesting the "Quarantine" will be. You start looking at file and index layouts to make efficient queries to the `Contestants` table, as they expect thousands of contestants to join the show.

9. (2 points) As contestants balance maintaining distance with maintaining their normal routines, there likely will be many updates per contestant (who are identified by their cid) to their infraction count. The producers also tell you that they will be holding special challenges for contestants within certain ranges of infraction numbers incurred, of whom you will need to help the producers identify.

   If you can only have one index, which combination of choices **(one letter per each column)** would optimize the performance in this scenario?

File Layout	**Index**
(a) Sorted File on cid	(d) Unclustered Index on cid
(b) Sorted File on numInfractions	(e) Unclustered Index on numInfractions
(c) Heap File	(f) Clustered Index on cid
	(g) Clustered Index on numInfractions

Solution: The scenario requires many `cid` lookups to update the corresponding `numInfractions`. Since the challenges apply to certain ranges of contestants, you will also need range scans of the `numInfractions`.

In terms of file layout, we want our file to be sorted by `numInfractions` so that we can efficiently scan the contestants who are within a certain range, so we pick (b). We should pick an unclustered index cid (d) to make infraction updates efficient.

4 B+ Trees Get Degrees (15 points)

For questions 1-3 we have a clustered alternative 2 B+ tree of height=2 (remember the height of the root is 0). Assume all matches for this B+ tree occur on the same data page. Also assume we have 10 buffer pages, use LRU as our eviction policy, and the buffer is empty at the start of each question. How many I/Os do each of the following operations require?

1. (2 points) `get(key)` where `key` does not appear in the table

   Solution:
   3 - need to reach the leaf page, but can stop there

2. (2 points) [**EXPLANATION**] `get(key)` where `key` appears in the table 5 times (assume order d=3)

   Solution:
   4 - need to reach the leaf page, and one data page because it is clustered

3. (2 points) [**EXPLANATION**] `put(key, value)` where no splits happen

   Solution:
   6 - 3 to reach the leaf, 1 to read data page, 1 to write data page, 1 to write the leaf page.

For questions 4-6, we want to see how modifications to the B+ tree affect the I/O count. For each modification, how much does the I/O count for the operation in question 2 (get(key) where key occurs 5 times) change by? Use +/- signs if the change is non-zero. If there is no change, put 0. For example, if the answer to question 2 is 10 I/Os, and you think after a modification it will only take 8 I/Os, your answer should be -2. These questions are independent of each other, so each question is only making one modification to the B+ tree presented at the start of the problem.

4. (2 points) How does the I/O count change if we make the tree an alternative 1 tree? Remember that we are assuming all records occur on the same page.

   Solution: -1
   You no longer need to go to the data page level, so you can stop when you reach the leaf.

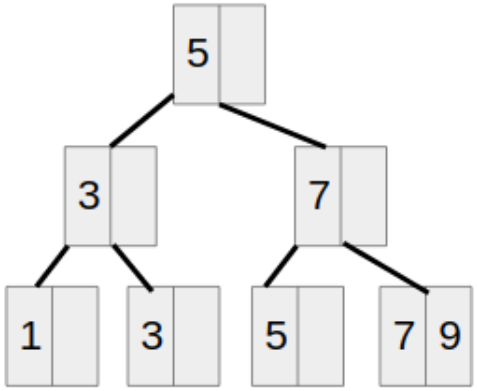
5. (2 points) How does the I/O count change if we make the tree an alternative 3 tree?

   Solution: 0
   Alternative 3 doesn't change the I/Os, just how the leaves are organized.

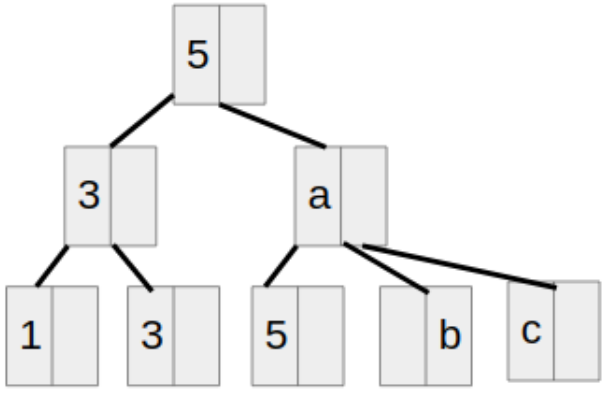
6. (2 points) How does the I/O count change if we make the tree unclustered (assume all 5 matches occur on different data pages)?

Solution: +4
Instead of having to read the one data page, we now need to read 5 different data pages.

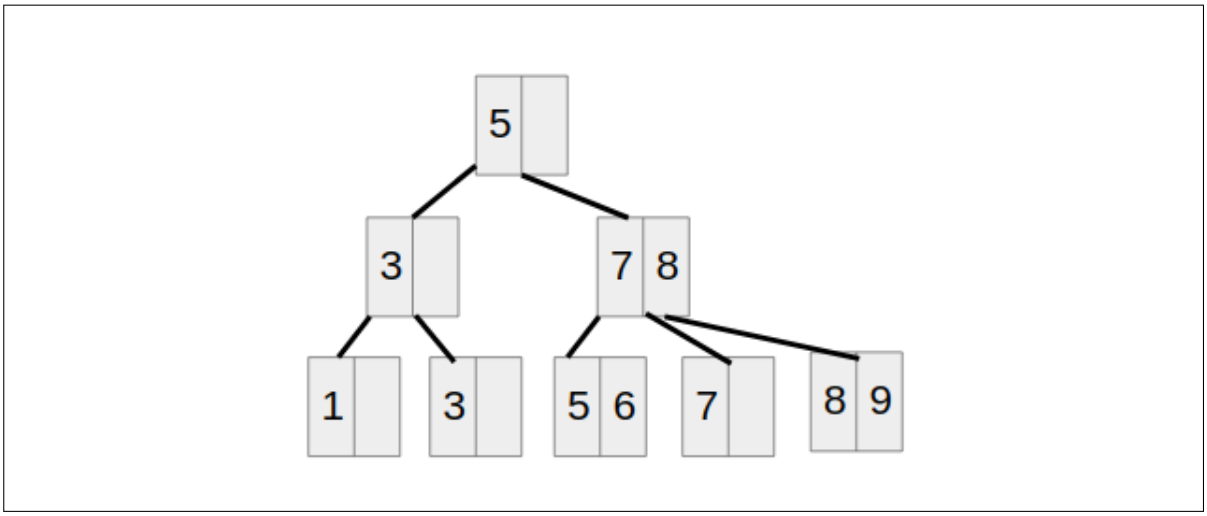
7. (3 points) We will now start with the following B+ tree:



We now insert 8 and then 6. The skeleton of the B+ tree after those insertions is below. Some keys are provided, some slots have letters in them, and some slots are empty even though a key should be there. Fill out the answer sheet with what keys belong in the slots that have letters in them. If a letter is in a place where there is no key, just write NONE. If a node only has one key, it belongs in the leftmost slot.



Solution:
a. 7
b. NONE
c. 8
The answers come directly from this diagram:


5
3
7 8
1
3
5 6
7
8 9

5 Hash Headaches, Stanford Sorts (17 points)

Jealous of the success of Postgres (which began as a research project at Berkeley), students at Stanford decide to develop their own database: Regres. Worried that they might injure themselves without adult supervision you decide to advise them with your knowledge from CS W186!

One of the students is experimenting with different hash functions for external hash on a set of random uniformly distributed unsigned 32-bit integers. We say a hash function causes data skew if we expect more keys to hash to certain values than others. For example, consider a hash function that returns 1 if the key is a multiple of 10 and 0 otherwise. This function causes data skew as roughly 10% of the possible keys would have a hash value of 1, while about 90% would hash to 0.

1. (2 points) The student asks you to consider the following hash function: the hash value is computed as the number of digits in the base 10 representation of the integer. Possible values are in the range of [1, 10] inclusive. 0 is considered to be 1 digit. Does this hash function cause significant data skew? And if so, which hash value has the most keys hash to it? Also include the percentage of keys which hash to it. In the example above, an acceptable answer would be: "Yes this function would cause data skew, roughly 90% of keys hash to the value of 0." You may round to the percentage to the nearest multiple of 5.

   Hint: The smallest unsigned 32-bit integer is 0, the largest is $2^{32}-1 = 4,294,967,295$ (roughly 4 billion).

   Solution: Yes, roughly 75% of keys hash to the value of 10.

   The set of 32-bit integers with 10 digits in decimal comprised of the range from 1,000,000,000 to 4,294,967,295 inclusive. This means roughly $\frac{3294967296}{4294967295} \approx 76.7\% \approx 75\%$ hash to the value of 10. Approximating with 3 billion out of 4 billion would have also yielded the correct answer. Full credit is awarded to any value between 75% and 80% inclusive. Leaving in fraction form is also acceptable.

2. (2 points) The student decides that writing good hash functions based only on digits is tricky and instead creates a program that returns a random number in the range [1, 186] uniformly as a hash value, regardless of what the key passed in is. On average, would this approach cause significant data skew? What might go wrong if they used this approach in external hashing when their goal is to group together duplicate values?

   Solution: No, this wouldn't cause data skew as on average we would expect that $\frac{1}{186}$ of keys would hash to the each possible value. However, this method of obtaining hash values isn't a proper hash function (hash functions and mathematical functions in general must be deterministic for a given input). In external hashing this could be problematic for removing duplicates, since we might send two keys with same value to different partitions when we need them in the same partition to spot duplicates.

   Just stating that the hash functions need to be independent doesn't warrant full credit.

3. (3 points) You decide to give the students a working implementation of one of the perfectly uniform hash functions frequently mentioned in CS 186 exams. The students test it out on a small input and it seems to work great. The next day the students call you to complain that there's a problem with the hash function you gave them: if they try to use it to group duplicates from a large input their code never stops running! Close examination reveals that the only hash function the students use for the entire

external hashing procedure is the perfectly uniform one you gave them, and that the most frequently repeated key can easily have all duplicates fit in memory at the same time. Explain why their external hashing worked for small inputs, but not for large ones. Your answer should be around 2 to 4 sentences of explanation.

Solution: For small enough inputs external hashing can partition the data into sufficiently small partitions in a single pass, and so only a single hash function is needed. If the input is large enough, then external hashing will need to use recursive partitioning. Recall that recursive partitioning needs multiple **independent** hash functions to work properly. Since the students only used the hash function you give them, if any single partition from the first partitioning phase was too large for memory then recursive partitioning would continuously send those same keys to the same partition on future passes without separating them up any further, causing recursive partitioning to run again and the student code to run indefinitely.

The students tell you they've been working for 12 months optimizing the in-memory sorting and merging functions used for external sort. Their optimized versions of in-memory sort (used in pass 0) and in-memory merge (used in the remaining passes) are 12× faster than the default implementations! For example, if the default implementation of either algorithm takes 12 seconds to do something, the optimized implementation will take 1 second.

You decide to test out a small example to see how much this would improve the overall speed of external sort. You find that disk I/Os will take roughly 15 milliseconds to complete, the default implementation of in-memory sort on 4 pages of records takes 4 milliseconds to complete, and the default implementation of in-memory merge takes 1 millisecond **for each** page being merged to complete (merging 10 pages takes 10 milliseconds). For the following questions consider running external sort on 12 pages of records with access to 4 memory buffers.

Assume the students are using the same implementation of external sort presented in class, and that disk operations are blocking (multiple disk I/Os can't be run in parallel). The times for in-memory sort and in-memory merge assume that the pages have already been loaded into memory and don't include the time to write pages back to disk.

4. (2 points) How many I/Os will it take to run external sort **and** how long in milliseconds will it take to complete these I/Os using the default implementation?

   Solution: The formula for the number of passes needed for external sort is $1 + \log_{B-1}\lceil N/B \rceil = 1 + \log_3\lceil 12/4 \rceil = 2$ passes total. Each pass incurs $2N$ I/Os, for a total of $2 \times 2N = 48$ I/Os. Each I/O takes 15 milliseconds, totalling to $48 \times 15 = 720$ milliseconds.

5. (2 points) How many separate in-memory sorts will take place **and** how many milliseconds in total will be spent performing in-memory sorts? Two in-memory sorts are considered separate from each other if they take place over different pages of records.

   Solution: All in-memory sorts in external sort take place during pass 0. Pass 0 consists of 3 separate in-memory sorts of 4 pages of records each. So there are 3 in-memory sorts taking $3 \times 4 = 12$ milliseconds total.

6. (1 point) How many milliseconds will be spent performing in-memory merge using the default implementation?

   Solution: After pass 0 there will be 3 sorted runs of 4 pages on disk. To merge these 12 pages it will take $12 \times 1 = 12$ milliseconds.

7. (2 points) [**EXPLANATION**] Sum together the times from the previous three questions to get the total run-time of external sort with the default implementations of in-memory sort and in-memory merge. Then compute the total run-time (in milliseconds) of external sort with the optimized (12× faster) versions of in-memory sort and in-memory merge. How many times faster is external sort with the optimizations (compute the default runtime divided by the optimized runtime)? If you don't have access to a calculator feel free to leave things in fractional form. Otherwise round your answer to two decimal digits.

Solution: Summing together the results from the previous parts should give a total of 720+12+12 = 744 milliseconds. The optimized version would reduce the cost of in-memory and in-memory sort to 1 millisecond each, resulting in $720 + 1 + 1 = 722$ milliseconds. Dividing the first by the second reveals the students optimization resulted in $\frac{744}{722} = 1.03\times$ speed up, much less than the 12× from just the in-memory portions!

8. (3 points) The students suspect that in another twenty-four months they can produce custom hardware to perform the in-memory sort and in-memory merge over 1000000x faster! Based on your answer to question 7, is it worthwhile for the students to focus on this approach if their goal is to increase the overall speed of their external sorts?

   (There are multiple correct ways to answer this, any answer that draws a rational conclusion about improvements to the speed of external sort based on the results of question 5.7 is eligible for full credit. Your answer should have around 2-4 sentences of justification).

Solution: In the previous question the optimizations on the in-memory algorithms only had a very small effect on the overall runtime of the algorithm. If we managed to do away with the cost of the in memory sort/merge entirely we would still get only $\frac{744}{720} \approx 1.03\times$ speedup. So for this reason it may not be the best use of the students time to spend another two years on this optimization.

If you recall Amdahl's law from 61c, this is not unlike how the limiting factor in optimizing work is the non-parallelizable portion: it doesn't matter how much you speed up the little things, you'll be bounded by the large time cost of disk speeds. If we found a way to avoid 1 write and 1 read (30 ms) we could achieve a greater speed up than the 1000000x optimization on the in-memory algorithms. If you've ever wondered why this class places emphasis on I/Os, this is one of the reasons.

Justifications for saying the students should pursue the hyperoptimization route include but are not limited to: our benchmark was only tested on a small input, and since the asymptotic cost of in-memory sorts/merge's grow super-linearly it may be worthwhile to optimize them further to get better speedups across a large number of buffers; there may be cases where even the marginal speedup is critical (real-time trading).

Any answer that applies the speedup to the disk I/Os missed the point of this question, and isn't eligible for any credit. The speedups were explicitly about in-memory operations only.

6 Join Me for Virtual Commencement (22 points)

Due to COVID-19, UC Berkeley, the number one public university, has moved commencement online! You have been asked by the chancellor to match diplomas and graduating students in Berkeley's database.

You are given the Students(sid, name, major, address) table (aliased as s) and the Diplomas(sid, name, major) table (aliased as d). Your goal throughout this problem is to join Students and Diplomas on the join condition `s.sid = d.sid`.

- The `Students` table has 500 pages with 50 records per page.
- The `Diplomas` table has 600 pages with 50 records per page.
- There are no indices
- Remember that the we do not include the final write.
- Assume all hash functions are perfect and evenly distributes the data.
- For every question, choose the join order that minimizes the I/O cost.
- **Only write the number in the answer box. Exclude units such as I/Os**

For question 1 and 2, we have B = 12 buffer pages.

1. (2 points) How many I/Os will Index Nested Loop Join cost? If it is not possible to join S and D, write `N/A`.

   Solution: Since there is no index, INLJ becomes SNLJ. 500 + 500 * 50 * 600 = 15000500 IOs.

2. (5 points) **[EXPLANATION]** How many I/Os will the **cheapest** join cost?

   Solution: We will be doing OPTIMIZED SMJ. To sort Students, we start with 500 pages and produce 42 runs of 12 pages after Pass 0. In Pass 1 1, we merge together 11 runs at a time and end up with 4 sorted runs. The I/O cost of this is 2*500*2 = 2000 I/Os.

   To sort Diplomas, we start with 600 pages and produce 50 runs of 12 pages after Pass 0. In Pass 1, we merge together 11 runs at a time and end up with 5 sorted runs. The I/O cost of this is 2*600*2 = 2400 I/Os.

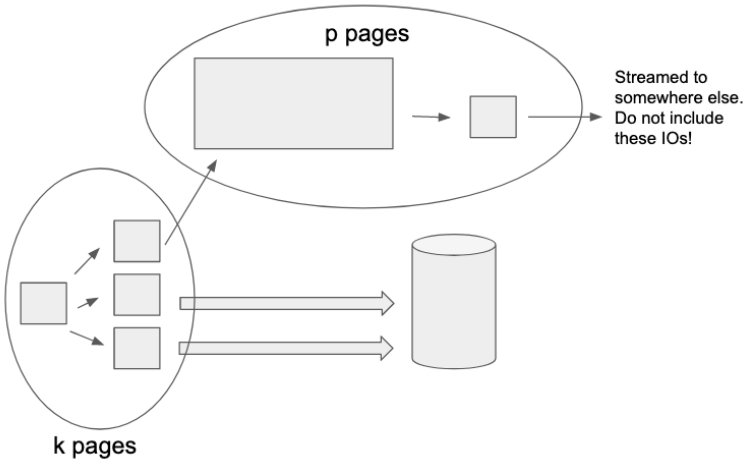
   Because we can do optimization (5 + 4 < 11), we can read all pages one last time (500 + 600) for a grand total of 2000 + 2400 + 1100 = 5500 I/Os.

With the knowledge from CS W186, you decide to explore an optimization to Grace Hash Join in hopes of finding a cheaper way to join Students and Diplomas on the same join condition above.

The optimization works like this:
We first split the total number of buffer pages B into two groups, k and p. The partitioning phase of our optimized Grace Hash Join (including recursive partitioning), hashes into k - 1 partitions, not B - 1. In our optimized GHJ, we don't write the first partition back to disk. The first partition output buffer becomes an input buffer to directly build an in memory hash table with p - 1 pages (the last page in p will be our output buffer)! We write the other partitions (k - 2) to disk like normal.

When we partition the second relation, the first partition will actually be directly probing our in memory hash table. **Do not include the final joined write IOs like how we exclude it in the final build and probe phase in normal Grace Hash Join**. The other k - 2 partitions of the second relation will be written back to disk like normal. During any partitioning phase, if the first partition has more than p - 1 pages, we write back all partitions back to disk just like normal Grace Hash Join. There is no difference in the final build and probe phase other than having less partitions than normal Grace Hash Join (**and we have access to all B buffers for this phase**).

Below is a diagram of the partitioning phase of the optimized Grace Hash Join:



3. (3 points) How many IOs will regular Grace Hash Join take with B = 50?

Solution: We first partition Students into 49 partitions of 11 pages each. Then we partition Diplomas into 49 partitions of 13 pages each. We do not need to recursively partition, therefore we simply go straight to build and probe. The grand total is 500 + 49(11) + 600 + 49(13) + 49(11) + 49(13) = 3452 IOs

4. (4 points) **[EXPLANATION]** How many IOs will our optimized Grace Hash Join take with B = 50, k = 26, p = 24?

Solution: We will read 500 pages of Students, and then each partition (25 partitions) will have 20 pages each. Since we only write back 24 partitions, we write 24 * 20 pages which will go into the true build and probe phase of GHJ. For Diplomas, we will have 25 partitions of 24 pages each. Again, we only write back 24 partitions. Then we will do a final read for the final build and probe phase. Grand total is 500 + 24(20) + 600 + 24(24) + 24(20) + 24(24) = 3212 IOs which is less than normal Grace Hash Join! This optimization is called Hybrid Grace Hash Join.

5. (4 points) How many IOs will our optimized Grace Hash Join take with B = 50, k = 11, p = 39? Note: This question will be graded independently from question 4 and will be all or nothing, so please check your work. However, you may opt out of doing this problem and receive 1 point.

   ☐ I opt out of this problem.

Solution: When we first partition Students, we see that we get 10 partitions of 50 pages each. Since 50 is greater than p - 1, we cannot apply our optimization here, and it becomes a normal partitioning pass of Grace Hash Join. After the first partitioning pass, we will have 10 partitions of Students with 50 pages each and 10 partitions of Diplomas with 60 pages each. Since 50 is greater than B - 2, we have to recursively partition. When we recursively partition Students, we end up with 100 partitions of 5 pages each and recursively partitioning Diplomas will give 100 partitions of 6 pages each. Since 5 is less than p - 1, we can apply our optimization in this partitioning pass! One partition of Students and one partition of Diplomas will go directly to the build and probe phase, so we only write 10*9 = 90 partitions of 5 pages of Students and 10*9 = 90 partitions of 6 pages of Diplomas for the second partitioning pass. Finally, we need to just read for the build and probe phase. The final total number of IOs will be 500 + 10(50) + 600 + 10(60) + 10(50) + 90(5) + 10(60) + 90(6) + 90(5) + 90(6) = 5280 IOs. Notice how choosing a bad k value will give us a worse result than normal Grace Hash Join!

6. (1 point) What happens to the I/O cost when k is closer to 1 (Assume B = 50)?

   A. Greater than normal Grace Hash Join

   B. Less than normal Grace Hash Join

   C. Same as normal Grace Hash Join

Solution: Greater. Essentially as k becomes closer to 1, we will have to recursively partition many times since we will have small number of partitions with large number of pages in each.

7. (3 points) What is the minimum k that we can have for B = 50 such that our optimized Grace Hash Join costs less than regular Grace Hash Join and can be done without recursive partitioning?

Solution: We can solve this with a system of equations. The two major equations we need are k + p = 50 and 500/(k-1) $\leq$ p - 1 because we need to make sure the first partition can fit into p - 1 pages of memory. We find that a valid k can be anywhere from 17 to 33, and since we want to find the minimum k, the answer would be 17.

7 Parallel Puffles (16 points)

For questions 1-5, assume we have m machines, b buffer pages per machine, p pages all starting on 1 machine, and k KB per page. For questions 1-3, write your answers as expressions of m, b, p, and k (some answers may not require all variables).

1. (2 points) **[EXPLANATION]** If we uniformly hash partition the p pages to the other machines, what would be the network cost in KB?

   Solution: $\frac{kp(m-1)}{m}$
   We send a fraction ($\frac{m-1}{m}$) of the p pages, each with k KB, across the network.

2. (2 points) In the previous problem, if we had instead used a non-uniform hash function, what is the difference between the highest possible and lowest possible network cost in KB?

   Solution: kp
   In the highest possible case, the hash function partitions all records to machines other than the original machine will all the data, resulting in a network cost of kp. In the lowest possible case, the hash function partitions all records to the original machine with all the data, resulting in 0 network cost.

3. (2 points) If we uniformly hash partition the p pages to the other machines and then perform parallel hashing with uniform hash functions, what is the maximum p which would guarantee at most 2 total passes in each machine?

   Solution: $mb(b-1)$
   Each machine can hash at most $b(b-1)$ pages in 2 total passes with uniform hash functions. We have m machines resulting in $p = mb(b-1)$ in total.

4. (1 point) Let p_{max} be the correct answer to the previous question. How many I/Os across all machines would it take to perform parallel hashing? Assume records off the network are streamed directly to disk on machines which does not incur I/Os.

   A. p_{max}
   B. $2p_{max}$
   C. $4p_{max}$
   D. $6p_{max}$

   Solution: There are 2 passes across all the data so we have $2 \cdot 2 \cdot p_{max} = 4p_{max}$ total I/Os.

5. (1 point) Suppose we evenly range-partitioned the records across all machines so that each machine has an equal number of pages on disk. How many I/Os across all machines would it take to then parallel sort the records?

   A. $\frac{2p}{m}(1 + \lceil \log_{b-1} \lceil \frac{p}{mb} \rceil \rceil)$

   B. $2p(1 + \lceil \log_{b-1} \lceil \frac{p}{mb} \rceil \rceil)$

   C. $\frac{2p}{m}(1 + \lceil \log_{b-1} \lceil \frac{p}{b} \rceil \rceil)$

   D. $2p(1 + \lceil \log_{b-1} \lceil \frac{p}{b} \rceil \rceil)$

> **Solution:** Each machine receives $\frac{p}{m}$ pages so it requires $1 + \lceil \log_{b-1} \lceil \frac{p}{mb} \rceil \rceil$ passes. Multiply by $\frac{2p}{m}$ to get the number of I/Os per machine and then multiply by m to get $2p(1 + \lceil \log_{b-1} \lceil \frac{p}{mb} \rceil \rceil)$ I/Os across all machines.

Waddle waddle! Welcome to Club Penguin! As you begin to explore this Antarctic open-world realm, you want a companion to join you, so you decide to adopt a Puffle! However, the Pet Shop is currently having database issues and needs your CS W186 expertise to help them. They currently have 10 machines and the following tables:

`Penguins(penguin_id PRIMARY KEY, num_coins, current_location, days_joined)`: 100 pages
`Puffles(puffle_id PRIMARY KEY, owner_id, color, cost, special_ability)`: 1000 pages

Assume each page is 4KB and is full with records. Also note that `Penguins` is range-partitioned on `days_joined` and `Puffles` is hash-partitioned on `color` in the following way:

Machine	[Penguins]	Penguins.days_joined	[Puffles]	Puffles.color
M1	10	[0, 10)	135	Blue
M2	10	[10, 20)	130	Pink, Black
M3	10	[20, 30)	120	Green
M4	10	[30, 40)	135	Yellow
M5	10	[40, 50)	100	Orange, Brown
M6	10	[50, 60)	115	Rainbow
M7	10	[60, 70)	60	Purple
M8	10	[70, 80)	70	Red
M9	10	[80, 90)	130	White
M10	10	[90, 100)	5	Gold

6. (1 point) We want to add a newly-joined penguin (you!) to the `Penguins` table. Assuming (for this question only) that a record in the `Penguins` table is 20 bytes and a master machine elsewhere is sending new records to the Pet Shop's 10 machines, what is the network cost in bytes associated with adding this new record?

> **Solution:** 200 bytes since we have to send the record to all machines to check for any duplicate `penguin_id` (primary key). Partial credit of 0.5 points was awarded for 20 bytes (assumed we only have to hash to M1).

7. (2 points) We want to do a parallel Grace Hash Join of `Penguins` and `Puffles` on `Penguins.penguin_id = Puffles.puffle_id`. Assuming we use a uniform hash function that partitions data evenly across machines for both tables, what is the network cost in KB?

Solution: 3960 KB. $\frac{9}{10}$ of both tables is sent across the network, resulting in $4 \cdot \frac{9}{10} \cdot (1000 + 100) = 3960$

8. (2 points) Continuing from the previous problem, assume we used that uniform hash function to partition both tables and now the records are on disk for all machines. If each machine has 6 buffer pages, how many ms would the parallel Grace Hash Join take if an I/O takes 1ms.

Solution: 330ms. Grace Hash Join with [`Puffles`]=100, [`Penguins`]=10, and B=6 takes 330 I/Os which is 330ms.

9. (3 points) **[EXPLANATION]** Going back to the original partitions in the table, what is the lowest network cost for executing the following query where we want to find all Puffles who are currently unadopted (`Puffles.owner_id IS NULL`) that all owners with 20 days joined or less can afford? Ignore the network cost of concatenating results.

```
SELECT * FROM Penguins, Puffles
WHERE Penguins.days_joined < 20
    AND Puffles.cost < Penguins.num_coins
    AND Puffles.owner_id IS NULL;
```

Solution: 720 or 680 KB. This is a broadcast join of the `Penguins` records on M1 and M2 across the other machines. M1 needs to send its records to M2-M10 and M2 needs to send its records to M1, M3-M10 in order to perform this parallel join. This results in $2 \cdot 9 \cdot 4 \cdot 10 = 720$ KB.

However, this is not the lowest possible cost since instead of M1 and M2 sending its records to M10, the 5 pages of `Puffles` on M10 can be sent to both M1 and M2 to also perform the parallel join. This saves $2 \cdot 5 = 10$ pages sent across the network which in total saves $4 \cdot 10 = 40$ KB. $720 - 40 = 680$ KB. However, this reduces the amount of machines we use (9 instead of 10) in the parallel join which may be less efficient.

8 Query-ntine Optimization: Work From Home (17 points)

Some of the CS W186 staff are retiring and therefore need to look for new employment in this post-COVID-19 apocalyptic realm (job market). Help them sort through the data and figure out which company's recruiters to contact (while maintaining appropriate social distancing protocols.) Assume that each column's values are uniformly distributed, and that all of the column distributions are independent of each other.

Table Schema	Records	Pages	Additional Notes

`CREATE TABLE Companies (`
`company_id INTEGER PRIMARY KEY,`
`company_name VARCHAR,`
`company_address VARCHAR,`
`);`
	200	10	• Unclustered alternative 3 index of height 1 and 5 leaf pages on `company_id`.

`CREATE TABLE Employees (`
`employee_id INTEGER PRIMARY KEY,`
`employee_name VARCHAR,`
`field VARCHAR,`
`salary INTEGER,`
`cid REFERENCES Companies(company_id),`
`job_id REFERENCES Jobs(job_id),`
`);`
	100,000	1,000	• Clustered alternative 2 index of height 3 and 500 leaf pages on `employee_id`.
			• Clustered alternative 2 index of height 3 and 800 leaf pages on `salary`.
			• `salary` ranges from \$1 to \$1 million.
			• `field` has 20 possible values.

`CREATE TABLE Jobs (`
`job_id INTEGER PRIMARY KEY,`
`position_name VARCHAR,`
`);`
	100	10	• None

1. (1 point) What is the selectivity of `salary > 50,000` from the `Employees` table?

Solution: (1,000,000 - 50,000)/(1,000,000) = (950,000)/(1,000,000) = 95/100 = 0.95

Questions 2-5 will refer to the following query:

```
SELECT *
FROM Companies as C, Employees as E, Jobs as J
WHERE C.company_id = E.cid AND E.job_id = J.job_id
    AND (E.salary > 750,000 OR E.field = "Computer␣Science")
ORDER BY C.company_id, E.salary
```

Additional information you may need: there are 7 buffer pages available.

2. (3 points) **[EXPLANATION]** What is the selectivity of this query?

Solution:
(1/200) * (1/100) * [(1/4) + (1/20) - (1/4)(1/20)]
= (1/20000) * [(6/20) - (1/80)]
= (1/20000) * (23/80)
= 23/1600000
= 0.000014375

3. (6 points) **[EXPLANATION]** Select all of the following single table access plans that will advance to the next pass. You must explain the reasoning behind **each** choice.

   A. Full scan on `Companies`

   B. Index scan on `Companies.company_id`

   C. Full scan on `Employees`

   D. Index scan on `Employees.salary`

   E. Index scan on `Employees.employee_id`

   F. Full scan on `Jobs`

Solution:
The cost for each individual scan is:

A: 10 I/Os (scan all pages)
B: 1 (scan root) + 5 (scan leaf pages) + 200 (access all records since it's unclustered) = 206 I/Os
C: 1,000 I/Os (scan all pages)
D: 3 (scan root and index pages) + 800 (scan leaf pages and access records) + 1000 (1 I/O per page of matching records) = 1803 I/Os.
Note that although we have E.salary > 750,000 in the WHERE clause, it does not suffice to only looking at the records E.salary > 750,000 because it is OR'd with E.field = `Computer Science` (we do not want to exclude records with salary <= 750,000 where field = `Computer Science`).
E: 3 (scan root and index pages) + 500 (scan leaf pages and access records) + 1000 (1 I/O per page of matching records) = 1503 I/Os
F: 10 I/Os (scan all pages)

A, C, and F are included for being the lowest cost scans for each individual table.

B and D are included for producing an "interesting order" on, respectively, `Companies.company_id` and `Employees.salary`.

4. (2 points) Assume the join types listed below are the only options for joining relations `C` and `E`. No other join types are considered. What will be kept from Pass 2?

A. Block Nested Loop Join (BNLJ)

B. Index Nested Loop Join (INLJ)

C. Sort-Merge Join

D. None of the above

Solution:
Note that we have the predicate `(E.salary > 750,000 OR E.field =` Computer Science`)`, which has selectivity 23/80.
The number of records of E remaining after applying this selection is 23/80 * 100,000 = 28750 records, or 288 pages.

The cost for each join type is:

E BNLJ C: $1000 + \lceil 288/(7-2) \rceil * 10 = 1580$ I/Os
We have an initial cost of 1000 to read in all the pages of E, but since we push down the selection on E, only 288 pages will be passed along to the BNLJ operator. This means the rest of the join will cost $\lceil 288/(7-2) \rceil$ * 10, since we split these 288 pages of E into chunks of 5 pages and scan through all of C for each chunk of E.

C BNLJ E: $10 + \lceil 10/7 - 2 \rceil * 1000 = 2010$ I/Os
We assume by default that we push down the selection on E but do not materialize the output after applying the selection. For each chunk of C, we need to scan through all the tuples of E (that satisfy the predicate), but since we don't materialize the results after the selection we will end up scanning through all 1000 pages of E.

E INLJ C: [E] + (23/80)$|E|$ * (cost of find matching tuples in C)
= 1000 + (23/80)(100000) * 3 (1 to read in the root, 1 to read in the leaf node, 1 to read in the data page)
= 87250 I/Os

C INLJ E: [C] + $|C|$ * (cost to find matching tuples in E)
= 10 + 200 * 1000 = 200010 I/Os
Note that we do not have an index on E.cid, so this devolves into an SNLJ. In order to identify the matching tuples in E, we will scan through all 1000 pages of E, as we don't materialize the results after performing the selection on E.

E SMJ C (same cost as C SMJ E):
1) Cost to sort E: (1000 + 288) + (288*2)*3 = 3016 I/Os
In Pass 0, we read in all 1000 pages of C, but we push down the selection so that only 288 pages of records need to be written into sorted runs. Each sorted run (except the last one) will have B = 7 pages, so we will have $\lceil \frac{288}{7} \rceil = 42$ sorted runs at the end of Pass 0.

In Pass 1, we merge together 6 runs at a time to produce 7 runs. After Pass 2, we produce 2 sorted runs, and after pass 3, we produce 1 sorted run. In each of these 3 passes, the IO cost is 288*2 because we read in 288 pages and write out 288 pages.

2) Cost to sort C: (10*2) * 2 = 40 I/Os
In Pass 0, we form 1 run with 7 pages and 1 run with the remaining 3 pages, and in Pass 1, we merge those 2 runs together.

3) Cost of merging pass (in the average case) = 288 + 10 = 298 I/Os
Note that for E, we only need to consider the 288 pages that remained after the selection and were sorted in Step 1.
However, since we have 2 runs for E and 2 runs for C in the 2nd-to-last sorting pass, and all of these can fit in memory with room for 1 output buffer ($2 + 2 \leq 7\text{-}1$), we can apply the SMJ optimization, saving 2*(288+10) I/Os. Thus, the total I/O cost with the optimization = 3016 + 40 + 298 - 596 = 2758 I/Os.

Note that after calculating the I/O cost for sorting C, you may realize SMJ is already significantly more expensive than BNLJ. The optimization can only save 2*(288+10) I/Os at most which would still be greater than the cost of BNLJ, so we can preemptively conclude that SMJ is therefore not the cheapest option. On an exam when you're under a time limit, you may not need to proceed with calculating the full I/O cost of SMJ to determine which joins will be kept by the System R query optimizer.

We keep E BNLJ C because this is the lowest cost option. We also keep E SMJ C because it produces an interesting order: the output will be sorted on C.company_id, which is used in a downstream operation (see the **`ORDER BY`** clause).

5. (2 points) Assume the join types listed below are the only options for joining relations `C` and `J`. No other join types are considered. What will be kept from Pass 2?

   A. Block Nested Loop Join

   B. Grace Hash Join

   C. Index Nested Loop Join

   D. Sort-Merge Join

   E. None of the above

Solution: $E \bowtie J$ is a cross join since there is no join predicate between the two tables. We avoid cartesian products in the algorithm so this join is never considered.

6. (3 points) Select all of the following join orders that will be considered for Pass 3 in the System R algorithm.

   A. $C \bowtie (E \bowtie J)$

   B. $(E \bowtie C) \bowtie J$

   C. $(J \bowtie C) \bowtie E$

   D. $E \bowtie (C \bowtie J)$

   E. $(C \bowtie J) \bowtie E$

   F. $(J \bowtie E) \bowtie C$

Solution: B and F will be considered because they're both left-deep plans without cross-joins.

A and D are not considered because neither are left-deep plans.

C, D, and E are not considered because they all involve a cross-join between the tables `Companies` and `Jobs`.

9 Tran$actions & ConCURRENCY (18 points)

True or False:

1. (1 point) For both schedule dependency graphs and deadlock detection graphs, the nodes represent different transactions being executed concurrently.

   Solution: True

2. (1 point) Two schedules are considered view equivalent if they have the same sets of initial reads, dependent reads, and winning writes.

   Solution: True

3. (1 point) Although view serializability allows for more schedules than conflict serializability, in practice, it is not used because it is too difficult to enforce efficiently.

   Solution: True

4. (1 point) A schedule S is serializable if and only if it is also conflict equivalent to a serial schedule.

   Solution: False, conflict serializable schedules are also serializable, but not the other way around. Some serializable schedules are not conflict serializable

5. (1 point) Two phase locking is a concurrency control protocol that prevents cascading aborts by guaranteeing conflict serializability.

   Solution: False, two phase locking guarantees conflict serializability, but doesn't prevent cascading aborts.

6. (1 point) Deadlock scenarios are always a product of unavoidable situations such as multiple lock upgrades.

   Solution: False, a bad implementation of lock upgrading (i.e. without appropriate prioritization) can lead to avoidable deadlocks.

7. (1 point) Implementations of multi-granularity locking involve locks being both acquired and released in a top to bottom order, as opposed to a bottom to top order.

Solution: False, neither mentioned implementation is correct. Locks should be acquired from top to bottom, and released from bottom to top.

8. (1 point) Dependency graphs are used to check whether two transactions are equivalent.

Solution: False, they are used to check whether two transactions are specifically conflict equivalent, not in the general sense.

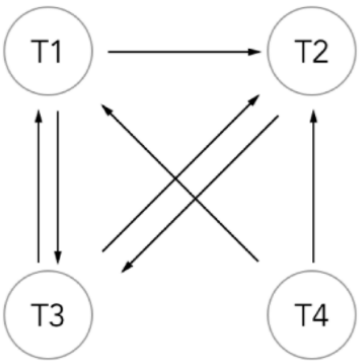
Consider the following schedule. For the questions in this section, unless specified, you are allowed to mark more than one options as correct. If no options are correct, select None.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T1				w(b)					r(a)			com		
T2		w(c)			r(c)		r(a)	r(b)						com
T3	r(b)					w(b)				r(c)	com			
T4			w(a)										com	

9. (1 point) Which transactions have edges pointing towards T1 in the conflict dependency graph?

A. T1
B. T2
C. T3
D. T4
E. None

Solution:



10. (1 point) Which transactions does T2 point to in the conflict dependency graph?

    A. T1
    B. T2
    C. T3
    D. T4
    E. None

> **Solution:** See part A solution diagram

11. (1 point) Which transactions have edges pointing towards T4 in the conflict dependency graph?

    A. T1
    B. T2
    C. T3
    D. T4
    E. None

> **Solution:** None of the above.

12. (1 point) What is the minimum number of operations that, if removed, the resulting dependency graph would be conflict serializable?

> **Solution:** 2. The edges from T2 to T3, T3 to T1

13. (1 point) True or False: Conflict dependency graphs to detect whether a schedule is view serializable should have self looping transitions to indicate conflicts for when a single transaction has two write operations on the same resource.

> **Solution:** False. Self looping transitions are not an entity in any sort of dependency graph within the context of this course.

14. (1 point) If we were to derive a schedule that is conflict equivalent to the one presented above, which of the following assumption(s) would be correct?

    A. T3 must be the last transaction since there are outgoing edges from it in the dependency graph
    B. T4 should be the first transaction since there are no incoming edges.
    C. Neither T1 or T2 should be the first transaction because there are incoming edges.
    D. It does not matter whether T2 or T3 is the last transaction since they have the same number of outgoing edges from it.
    E. None

Solution: A transaction can be selected as the last one in a schedule if there are no outgoing edges.

The CS 186 Office Hours queue sometimes leads to hold ups. Therefore, in order to make the flow of office hours more intuitive, the CS 186 staff has decided to implement a new database system allowing students to access and edit TA resources during office hours from TAs teaching specific topics.

The office hours work as follows: students, like transactions, can place a lock on a specific instructor, like a resource. At a particular moment in time, the office hours queue has four teaching assistants online: David, Ethan, Brian and Angela, who will be denoted as D, E, B, and A

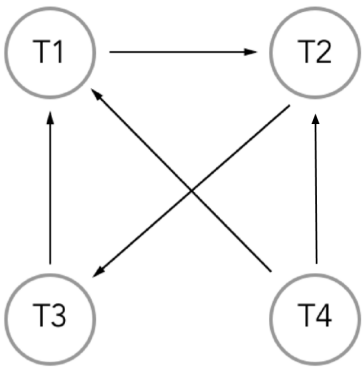
Consider the following schedule with four students online, denoted as T1 through T4. S means Read, and X means Write.

	1	2	3	4	5	6	7	8	9
T1			X(B)	X(A)				S(D)	
T2		S(E)					X(D)		
T3	X(E)					S(B)			
T4					X(A)				S(D)

15. (1 point) Identify all transactions involved in deadlock(s), if there are any.

A. T1
B. T2
C. T3
D. T4
E. None

Solution:



For the following two questions, assume the following transaction priorities: {T1: 1, T2: 2, T3: 3, T4: 4}

16. (1 point) Assume we use the "wound-wait" deadlock avoidance strategy. For the above schedule, which transaction, if any, would wait?

A. T1

B. T2

C. T3

D. T4

E. None

Solution:

- T2: At timestep 2, T2 wants an S lock on resource E that is locked by T3, which has higher priority, so T2 will wait.

17. (1 point) Assume we use the "wait-die" deadlock avoidance strategy. For the above schedule, which transaction, if any, would wait?

A. T1

B. T2

C. T3

D. T4

E. None

Solution:

- T4: At timestep 5, T4 wants an X lock on resource A that is locked by T1, which has lower priority, so T4 will wait.
- T3: At timestep 6, T3 wants an S lock on resource B that is locked (X lock) by T1, which has lower priority, so T3 will wait.

18. (1 point) A transaction that aborts in a "wait-die" policy would wait if the "wound-wait" policy were used instead

Solution: True

10 Recovering from Spring 2020 (17 points)

David's getting tired of everything crashing around him. His semester is a disaster, reality is a social construct, and his personal database keeps struggling to recover (just like him). His database, which he's nicknamed BoostedEgg and is on version 2020.5, has been having issues. Can you help him figure out what's wrong?

David's database just crashed, so he gets the log up to the crash and sees the following log.

LSN	Record	prevLSN
0	master	-
10	update: T1 writes P1	0
20	update: T2 writes P2	0
30	update: T3 writes P1	0
40	Begin Checkpoint	-
50	update: T1 writes P2	10
60	update: T2 writes P2	20
70	commit: T2	60
80	end: T2	70
90	End Checkpoint	-
	C R A S H	

1. (1 point) **[EXPLANATION]** Is the above schedule possible with 2PL?

   A. Yes

   B. No

   Solution: Once T2 releases its lock for P2, T1 would be able to make its update on P2 at LSN 50. However, T2 would not be able to re-acquire the lock for the update at LSN 60 once it has started the release phase of 2PL.

2. (1 point) Is the above schedule possible with strict 2PL?

   A. Yes

   B. No

   Solution: If a schedule is not possible under 2PL, it is not possible under strict 2PL.

David is tinkering with BoostedEgg v2020.5 to release a v2020.6, when it crashes again. He gets the log up to the crash and sees the following log.

LSN	Record	prevLSN
0	master	-
10	update: T1 writes P1	0
20	update: T3 writes P1	0
30	Begin Checkpoint	-
40	update: T1 writes P2	10
50	update: T2 writes P2	0
60	commit: T2	50
70	end: T2	60
80	End Checkpoint	-
90	update: T1 writes P3	40
	C R A S H	

3. (1 point) [**EXPLANATION**] Is the above schedule possible with 2PL?

   A. Yes

   B. No

> **Solution:** T1 could have acquired the locks for P1, P2, and P3 at the start of its transaction. It then could have released the lock for P1 after updating it at LSN 10 before T3 acquires the lock for P1 before T3's update at LSN 20. T1 could have also released the lock for P2 after updating it at LSN 40 before T2 acquires the lock for P2 before T2's update at LSN 50.

4. (1 point) Is the above schedule possible with strict 2PL?

   A. Yes

   B. No

> **Solution:** With strict 2PL, all locks would be released at the same time when the transaction commits. Under strict 2PL, T3 would not have been able to update P1 (LSN 20) and T2 would not have been able to update P2 (LSN 50) before T1 released its locks on P1 and P2. Since T1 still has the update on P3 (LSN 90), T1 would not have released its locks on P1 and P2 before then.

BoostedEgg v2020.5 is managing transactions and pages and logs and oof...just crashed. Again. David gets the log up to the crash and sees the following:

LSN	Record	prevLSN
0	master: checkpoint at LSN 40	-
10	update: T1 writes P1	null
20	update: T2 writes P2	null
30	update: T3 writes P1	null
40	Begin Checkpoint	-
50	update: T2 writes P2	20
60	update: T1 writes P2	10
70	commit: T2	50
80	end: T2	70
90	End Checkpoint	-
100	update: T3 writes P3	30
110	update: T4 writes P4	null
120	update: T3 writes P2	100
130	update: T1 writes P1	60
140	commit: T4	110
	C R A S H	

The end checkpoint also contains the following tables:

TID	Status	lastLSN
T1	Running	10
T2	Committing	70
T3	Running	30

PID	recLSN
P1	10
P2	60

5. (3 points) BoostedEgg v2020.5 just finished its analysis phase of ARIES. Committed transactions have been ended and running transactions have been aborted. Were additional records written? If so, fill in any additional records written.
   If no additional records were written, bubble or type in `N/A` for all sections.
   If additional records were written, fill in the table accordingly. For just this question, fill out the table numerically by transaction, then by chronological order for each transaction. You may not need all the rows provided in the table. Bubble or type in `N/A` for all sections for the extra rows.

Row	Record Type (Commit, Abort, End)	Transaction	prevLSN
1			
2			
3			
4			
5			

Solution:

Row	Record Type (Commit, Abort, End)	Transaction	prevLSN
1	Abort	T1	130
2	Abort	T3	120
3	End	T4	140
4			
5			

We know we need to abort T1 (lastLSN = 130), and T3 (lastLSN = 120). Since T4 is committing at LSN 140, we write an END record and remove it from the transaction table after.

6. (4 points) Fill in the following dirty page table and transaction table as they would appear right before the redo phase starts. Committed transactions have been ended and running transactions have been aborted. If an entry should not appear in the table then type `N/A` for all sections for the corresponding entry.

TID	Status	lastLSN
T1		
T2		
T3		
T4		

PID	recLSN
P1	
P2	
P3	
P4	

Solution:

TID	Status	lastLSN
T1	Aborting	150
T3	Aborting	160

PID	recLSN
P1	10
P2	60
P3	100
P4	110

We were given the LSN of the most recent checkpoint in the master record, so we can take the tables we got from the checkpoint and start analysis from LSN 40.
We see that the pages modified since the checkpoint are P1, P2, P3, and P4, but because P1 and P2 are already in the DPT, we don't need to make any changes to those. However, we do need to add P3 and P4 into the DPT with the appropriate LSNs corresponding to the first updates on those pages.
We see that T1, T2, T3, and T4 had log records emitted since the checkpoint, so we update the transaction table for all four transactions to reflect those log records. From the previous question, we know that T1's lastLSN is now 150 and T3's lastLSN is now 160, and both their statuses are aborting. Since there is an End record at LSN 80 for T2, we removed T2 from the transaction table altogether. We also see that T4 is committing at LSN 140, so we wrote the END record and removed it from the transaction table after.

Partial credit rewarded if student put lastLSN for T1 was 130 and lastLSN for T3 was 120.

7. (2 points) Were any pages flushed to disk at any point during the log? If so, what pages?
If no pages were flushed to disk, bubble in `No` and do not mark any of the pages. If pages were flushed to disk, bubble in `Yes` and mark the pages accordingly. No credit will be given if the answer bubble is not filled out, or if no pages are marked if `Yes` was bubbled.

A. Yes

B. No

What pages were flushed to disk?

A. P1

B. P2

C. P3

D. P4

Solution: P2 was flushed to disk between LSN 20 and LSN 60.

8. (4 points) Assume that we've already completed the redo phase. Complete the undo phase. Specifically, what log records are emitted?
Reminder that the undoNextLSN is `null` for CLRs that don't have a next record to undo. For non-CLR records, bubble `N/A` for all irrelevant sections.
You may not need all the rows provided in the table. Bubble or type in `N/A` for all sections for the extra rows.

Row	Record Type	Transaction	LSN of record being undone	undoNextLSN
1				
2				
3				
4				
5				
6				
7				
8				

Solution:

Row	Record Type	Transaction	LSN of record being undone	undoNextLSN
1	CLR	T1	130	60
2	CLR	T3	120	100
3	CLR	T3	100	30
4	CLR	T1	60	10
5	CLR	T3	30	null
6	End	T3	N/A	N/A
7	CLR	T1	10	null
8	End	T1	N/A	N/A

We add the two aborting transactions at the end of analysis (T1 and T3) along with their lastLSN (as the weight) into a priority queue. We take the highest LSN in the priority queue and process it. The order of processing is as follows:

- LSN 160 (T3): This is the abort record written at the end of the Analysis phase, so we get the prevLSN and add this back to the priority queue (add LSN 120, T3 back into the priority queue).
- LSN 150 (T1): This is the abort record written at the end of the Analysis phase, so we get the prevLSN and add this back to the priority queue (add LSN 130, T1 back into the priority queue).
- LSN 130 (T1): We can undo this record at LSN 130, giving us the CLR with LSN 180. We get the undoNextLSN using the prevLSN (LSN 50) of this update record at LSN 90. We get the prevLSN through the lastLSN of T1 in the transaction table, and update the lastLSN in the transaction accordingly. We then requeue this transaction back into the PQ with a weight of 60.
- LSN 120 (T3): We undo this record at LSN 120, giving us the CLR with LSN 190 with the prevLSN as 160 and undoNextLSN as 100. Requeue T3 with weight 100.
- LSN 100 (T3): We undo this record at LSN 100, giving us the CLR with LSN 200 with the prevLSN as 190 and undoNextLSN as 30. Requeue T3 with weight 30.
- LSN 60 (T1): We undo this record at LSN 60, giving us the CLR with LSN 210 with the prevLSN as 180 and undoNextLSN as 10. Requeue T3 with weight 10.
- LSN 30 (T3): We undo this record at LSN 30, giving us the CLR with LSN 220 with prevLSN as 140 and undoNextLSN as `null`. At this point we are done with the rollback process so we write an end record with prevLSN as 220.
- LSN 10 (T1): We undo this record at LSN 10, giving us the CLR with LSN 230 with prev LSN as 210 and undoNextLSN as `null`. At this point we are done with the rollback process so we write an end record with prevLSN as 230.

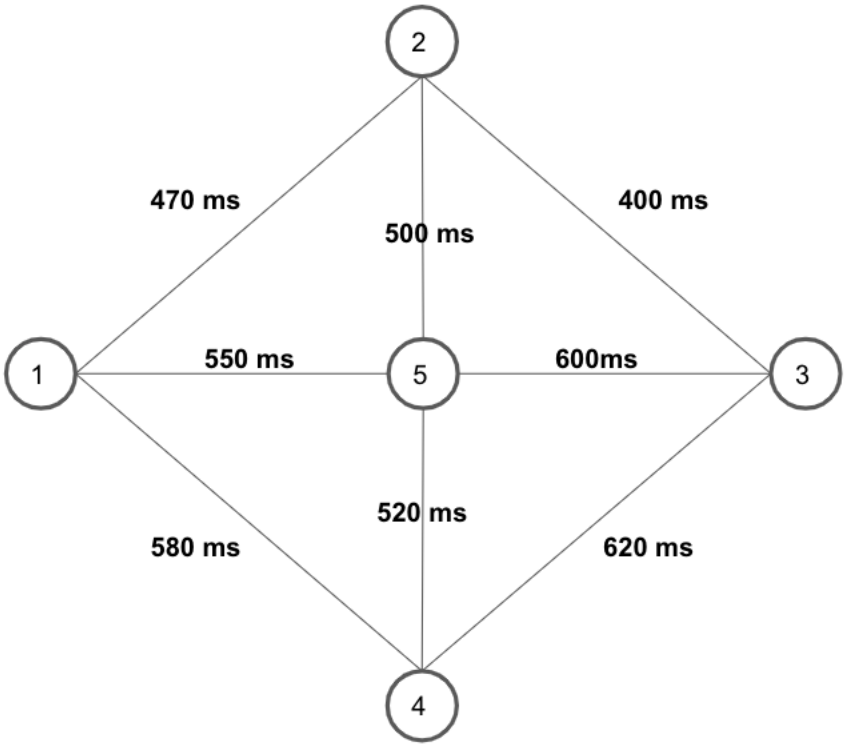
Note: LSN of the record being emitted and prevLSN were not asked for in the question.

11 Distributed Semester (18 points)

Due to an influx of online activity, Berkeley's Database was partitioned across 5 Machines to better handle the increased workload.

The machines are configured in the below format and use 2PC to reach consensus along with **presumed abort**. Assume that everything is instantaneous except for the **sending of messages across the network (send times defined by edges in the graph)** and **flushing of log records, which takes 20ms**.

When performing 2PC for any transaction, the participants are the coordinator's adjacent nodes and there is no need to re-flush records if they already exist. All participants will vote yes and **timeouts occur only after 10000 ms.**



Assume that **Machine 5 is the coordinator** and starts 2PC for the transaction at 0ms. Use the following series of events to answer problems 2-7.

- **1800 ms**: Machine 1 crashes
- **1900 ms**: Machine 3 crashes
- **2000 ms**: Machine 1 recovers
- **2200 ms**: Machine 3 recovers
- **3687 ms**: Machine 5 crashes
- **3700 ms**: Machine 4 crashes
- **3900 ms**: Machine 5 recovers
- **4590 ms**: Machine 4 recovers

1. (2 points) What is the 2nd message Machine 5 sends?
    A. VOTE YES
    B. VOTE NO
    C. PREPARE
    D. COMMIT
    E. ABORT
    F. ACK
    G. STATUS INQUIRY

> **Solution:** Machine 5 receives all yes votes and sends a commit message at 1240 ms (600 + 20 + 600 + 20) after flushing a commit record.

2. (2 points) What is the 2nd message Machine 1 sends?
    A. VOTE YES
    B. VOTE NO
    C. PREPARE
    D. COMMIT
    E. ABORT
    F. ACK
    G. STATUS INQUIRY

> **Solution:** Machine 1 receives the commit message at 1790 ms (1240 + 550). Since flushing the commit record at the participant takes 20 ms, machine 1 crashes before the record is flushed and recovers with only a prepare log record. Upon finishing recovery, it sends an inquiry to get the status of the transaction.

3. (2 points) What is the 3rd message Machine 3 sends?
    A. VOTE YES
    B. VOTE NO
    C. PREPARE
    D. COMMIT
    E. ABORT
    F. ACK
    G. STATUS INQUIRY

> **Solution:** Machine 3 sends an ACK after it recovers since it flushed a commit record at 1860 ms (1240 + 600 + 20), which is before it crashed at 1900 ms.

4. (3.5 points) **[EXPLANATION]** What is the earliest time at which Machine 5 receives an ACK from all participants?

Solution: After sending out commit messages at 1240 ms, machine 5 waits for ACKs from all participants. Only machine 1 crashes before it can send an ACK back so the coordinator waits for its ACK. Upon recovering at 2000 ms, machine 1 sends a status inquiry (550), receives a commit msg (550), flushes a commit record (20) and sends back and ACK (550). Therefore, the time at which machine 5 receives all ACKs for the first time is 2000 + 550 + 550 + 20 + 550 = **3670 ms**.

5. (2 points) What is the first message Machine 5 sends after it recovers?
   A. VOTE YES
   B. VOTE NO
   C. PREPARE
   D. COMMIT
   E. ABORT
   F. ACK
   G. STATUS INQUIRY
   H. No Msg - Xact has ended

Solution: Machine 5 receives all ACKs at 3670 but crashes at 3687 before it could possibly flush an end record. When it recovers, it only finds a flushed commit record and reruns phase 2 of 2PC by sending out a commit message to all participants.

6. (2 points) What is the 3rd message Machine 2 sends?
   A. VOTE YES
   B. VOTE NO
   C. PREPARE
   D. COMMIT
   E. ABORT
   F. ACK
   G. STATUS INQUIRY

Solution: After machine 5 recovers, it reruns phase 2 so upon receiving a commit message, machine 2 will resend an ACK.

7. (3.5 points) **[EXPLANATION]** What is the earliest time at which Machine 5 can flush an END record to its log?

Solution: After machine 5 recovers at 3900 ms, it reruns phase 2 and waits for all ACKs before writing an end record. Machine 4 recovers at 4590 ms and sends an ACK since it has a flushed commit record. Once machine 4's ACK arrives at 5110 ms (4590 + 520), machine 5 can write and flush an end record at **5130 ms** since the other participant's ACKs were all received before machine 4's ACK. All other participants' ACKS were received by 5100 ms (3900 + 600 + 600).

8. (1 point) While using presumed abort, if a coordinator doesn't receive unanimous yes votes and crashes after sending out abort messages, it will recover with a flushed abort record in its log and resend abort messages.

A. True

B. False

Solution: Under presumed abort, abort records aren't flushed because we always assume a transaction is being aborted if there are no log records that tell us otherwise. Therefore, when the coordinator recovers, it won't find any flushed log records and participants will inquire after timing out if they have voted and have not yet received the status of the transaction.