CS W186 Fall 2019 Midterm 1

Do not turn this page until instructed to start the exam.

Contents:

- You should receive one *single-sided answer sheet* and a 20-page *exam packet*.
- The midterm has *6 questions*, each with multiple parts, and worth a total of *79 points*.

Taking the exam:

- You have *110 minutes* to complete the midterm.
- All answers should be written on the answer sheet. The exam packet will be collected but not graded.
- For each question, place only your *final answer* on the answer sheet; *do not show work*.
- For multiple choice questions, please *fill in the bubble or box completely* as shown on the left below. ***Answers marking the bubble or box with an X or checkmark may receive a point penalty***.

- A blank page is provided at the end of the exam packet for use as scratch paper.

Aids:

- You are allowed **one handwritten** 8.5" × 11" double-sided pages of notes.
- *No electronic devices are allowed on this exam.* No calculators, tablets, phones, smartwatches, etc.

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.

0 Pre-Exam Questions (0 points)

1. (0 points) How many grams of sugar are in an average boba drink?

Solution: ~36-38g in a 16 oz. drink (regular/medium size at a lot of places).

2. (0 points) How many TA(s) does CS W186 have this semester?

Solution: 18! Do you know all of them?

1 SQL & Relational Algebra (18 points)

You have just graduated with a Computer Science degree from UC Berkeley, a prestigious institution whose alumni have received more Turing Awards than any other university in the world. Your friends are all excited to begin their nascent careers (and make big bucks).

Wary of the tech bubble, however, and sensing that there might be more money in the boba craze than in browsing Stack Overflow eight hours a day, you decide not to join your friends in South Bay, and instead set up a bubble tea shop on Bancroft Avenue.

You hire Joe Hellerstein to design your database. He builds a database for you with the following schemas:

```
-- This table lists all of your sales, along with foreign keys for the drink sold and the
    customer who bought it.
CREATE TABLE Sales (
    sale_id INTEGER PRIMARY KEY,
    drink_id INTEGER REFERENCES Drinks,
    customer_id INTEGER REFERENCES Customers);

-- This table lists all of the drinks you have for sale.
CREATE TABLE Drinks (
    drink_id INTEGER PRIMARY KEY,
    name VARCHAR(20) UNIQUE NOT NULL,
    price FLOAT NOT NULL);

-- This table stores customer information.
CREATE TABLE Customers (
    customer_id INTEGER PRIMARY KEY,
    name VARCHAR(50) NOT NULL);
```

You also hire Joe to run your database for the first few weeks. However, his consulting rate of $250 an hour proves too expensive, and you sadly have to let him go. Alas! You have to run the database yourself now. Let's hope you remember everything you learned in CS 186.

1. (4 points) A customer walks in and asks what your most popular drink is. Embarrassingly, you have no idea.

   What query will give you the name of the most sold drink(s)?

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

   A.
   ```
   SELECT Drinks.name
   FROM Sales, Drinks
   WHERE Sales.drink_id = Drinks.name
   GROUP BY Drinks.name
   ORDER BY COUNT(*) DESC;
   ```

   B.
   ```
   SELECT Drinks.name
   FROM Sales INNER JOIN Drinks
   ON Sales.drink_id = Drinks.drink_id
   WHERE COUNT(Sales.drink_id) >= ALL (
       SELECT COUNT(*)
       FROM Sales
       GROUP BY Sales.drink_id);
   ```

   C.
   ```
   SELECT Drinks.name
   FROM Sales, Drinks
   WHERE Sales.drink_id = Drinks.drink_id
   GROUP BY Drinks.name
   HAVING COUNT(*) = (
       SELECT COUNT(*)
       FROM Sales
       GROUP BY Sales.drink_id
       ORDER BY COUNT(*)
       LIMIT 1);
   ```

   Solution: The first query has two errors. First, the joining key is incorrect. Second, as no limit is imposed, the query would return all drinks, not just the most popular ones.

   The second query does aggregation in `WHERE`, which is not allowed.

   The third query is wrong because it gets the least sold drink(s).

2. (4 points) Unfortunately, your bubble tea shop is not doing so well, and you need to cut some items from the menu. You wonder if there are any drinks that have never been ordered.

   What query will get you the names of drinks that have never been ordered?

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

   A.
   ```
   SELECT Drinks.name
   FROM Drinks
   WHERE Drinks.drink_id NOT IN (
       SELECT Sales.drink_id
       FROM Sales);
   ```

   B.
   ```
   SELECT Drinks.name
   FROM Sales LEFT OUTER JOIN Drinks
   ON Sales.drink_id = Drinks.drink_id
   WHERE Sales.drink_id IS NULL;
   ```

C.
```
SELECT Drinks.name
FROM Sales RIGHT OUTER JOIN Drinks
ON Sales.drink_id = Drinks.drink_id
GROUP BY Drinks.name
HAVING COUNT(Sales.drink_id) = 0;
```

Solution: The first query is correct.

The second query will not return anything. After the `LEFT OUTER JOIN`, it only has drinks that have actually been sold.

The third query is correct, and requires remembering 1. how outer joins create `NULL` values, and 2. how aggregate functions work on columns with null values. It's tricky, but it was explicitly covered in lecture, with examples as well.

After fixing your menu, your shop is much more efficient, and business is booming. You decide to reward your customers by buying each of them their favourite drink. However, you only want to reward your most loyal customers - those with 5 or more purchases.

To find your loyal customers and their favourite drinks, you execute the following query:

```
WITH LoyalCustomers(customer_id) AS (
    SELECT Customers.customer_id
    FROM Sales, Customers
    WHERE Sales.customer_id = Customers.customer_id
    GROUP BY Customers.customer_id
    HAVING COUNT(*) >= 5
)
SELECT Drinks.name, Customers.name
FROM Drinks, Customers, LoyalCustomers
WHERE Customers.customer_id = LoyalCustomers.customer_id
  AND Drinks.drink_id IN (
    SELECT Sales.drink_id
    FROM Sales
    WHERE Sales.customer_id = LoyalCustomers.customer_id
    GROUP BY Sales.drink_id
    ORDER BY COUNT(*) DESC
    LIMIT 1
);
```

3. (1 point) Can there be any `NULL` values anywhere in this result?
4. (1 point) Can there be any duplicate rows in this result?
5. (1 point) If you execute this query twice in a row, are you guaranteed to get the same result?
6. (1 point) The query above contains the following subquery:

   ```
   SELECT Sales.drink_id
   FROM Sales
   WHERE Sales.customer_id = LoyalCustomers.customer_id
   GROUP BY Sales.drink_id
   ORDER BY COUNT(*) DESC
   LIMIT 1
   ```

   Is this a correlated subquery? Or is it an uncorrelated subquery?

Solution: No, there can't be any `NULL` values anywhere. Both the result columns, `Drinks.name` and `Customers.name`, have `NOT NULL` constraints.

Yes, there can be duplicate results (e.g. two customers with the same name and same favourite drink).

No, we cannot expect to get the same result. The subquery selects a favourite drink, but if there is a tie, it nondeterministically picks one.

The subquery is correlated (on `LoyalCustomers.customer_id`.

For the following questions, we refer to the Sales table as S, Drinks table as D, and Customers table as C.

7. (3 points) One day, Josh, the owner of a boba tea shop across the street from you, comes into your shop. Jealous of the popularity of your shop, Josh says customers only buy the cheap drinks from your shop. Irritated by his statement, you want to find all the sale_id and price for drinks with price above 4.2 dollars.

   For the following relational algebra expressions, mark True if it finds the sale_id and price of drinks sold with price above 4.2 dollars, False otherwise.

   A. $\pi_{\textbf{sale\_id, price}}((\sigma_{\textbf{price}>4.2}(D)) \bowtie S)$

   B. $\pi_{\text{sale\_id, price}}(S \bowtie (\sigma_{\text{price}>4.2}(\pi_{\text{drink\_id}}(D))))$

   C. $\pi_{\textbf{sale\_id, price}}(\sigma_{\textbf{price}>4.2}(D \bowtie \pi_{\textbf{drink\_id, sale\_id}}(S)))$

   Solution: A. True.

   B. False - the selection on price is invalid because we don't have the price column after the inner projection.

   C. True.

8. (3 points) To further convince Josh, you want to find the customer_id of all the customers who have bought both drinks of price above 4.2 dollars and drinks of price not above 4.2 dollars.

   For the following relational algebra expressions, mark True if it finds the customer_id of customers who have bought both drinks with price > 4.2 and drinks with price ≤ 4.2, False otherwise.

   A. $\pi_{\text{customer\_id}}(\sigma_{\text{price}>4.2 \text{ AND } \text{price}<=4.2}(S \bowtie D))$

   B. $\pi_{\textbf{customer\_id}}(\sigma_{\textbf{price}>4.2}(S \bowtie D)) \bigcap \pi_{\textbf{customer\_id}}(\sigma_{\textbf{price}<=4.2}(S \bowtie D))$

   C. $\pi_{\text{customer\_id}}(\sigma_{\text{price}>4.2}(S \bowtie D)) - (\pi_{\text{customer\_id}}(\sigma_{\text{price}<=4.2}(S \bowtie D)) - \pi_{\text{customer\_id}}(\sigma_{\text{price}>4.2}(S \bowtie D)))$

   Solution: A. False - this returns nothing.

   B. True.

   C. False - $S_1 \bigcap S_2 = S_1 - (S_1 - S_2) \neq S_1 - (S_2 - S_1)$.

2 Disks & Files (15 points)

1. (5 points) Which of the following statements are True? **There may be zero, one, or more than one correct answer.**
   A. Slot directory footers can only be used for pages storing variable length records.
   B. Modifying a page currently in memory counts as 1 I/O.
   C. Insertion and deletion are faster for packed pages than unpacked pages.
   D. I/O cost of performing a full scan on a sorted file is the same as scanning a heap file, assuming both are packed.
   E. Binary search allows for fast lookups on sorted files implemented as linked lists.

> **Solution:** A - False, slot directory footers can also be used for pages storing fixed length records but a bitmap is better so we don't use it
> B - False, only reading pages from disk and writing pages to disk count as I/Os
> C - False, deletion involves reordering records
> D - True, scanning involves reading all pages regardless if the file is sorted or not
> E - False, binary search can't be performed on a linked list

Consider a Heap File that takes up 201 MB, including any metadata, storing **variable length records** where half of the data pages are full and the other half contain some free space. The file is implemented as a **Linked List** and each page is 1 MB (for this question, assume 1 MB = 1024 KB).

Assume, for the following two questions, that any insertion does not change the status of a page from having free space to being full.

2. (2.5 points) What is the I/O cost to insert a record into the heap file in **the best case**?

> **Solution:** In the best case, the first free data page contains enough space to insert the variable length record.
>
> 1 (read header page) + 1 (read 1st data page) + 1 (write 1st data page) = 3 I/Os

3. (2.5 points) In **the worst case**?

> **Solution:** Assuming no data pages can be added, in the worst case, only the last free data page contains enough space to store the new variable length record (record size could be very large). Since there are 201 pages in total, 200 are data pages and 100 are free data pages.
>
> 1 (read header page) + 100 (read all free data pages) + 1 (write last data page) = 102 I/Os
>
> Assuming a new data page can be added, in the worst, none of the free data pages contain enough space so a new data page needs to be allocated. The prev last data page also needs to be written

to disk because it contains an updated pointer to the new last data page.

1 + 100 + 1 (read new data page) + 1 (write new data page) + 1 (update pointer on prev last data page) = 104 I/Os

Consider a 200 MB Heap file storing **fixed length records** where at least one data page contains some free space. The file is implemented as a **Page Directory** where each header page can hold 19 entries and each page is 1 MB. On each page, the page header is 24 KB and the size of each record is 10 KB. You have 5GB of memory available to you, and it is entirely empty to begin with. The file size includes any relevant metadata.

4. (2.5 points) What is the I/O cost to insert a record into the heap file in **the best case**?

> **Solution:**
> 1 (read header page) + 1 (read data page) + 1 (write updated data page) + 1 (write updated header page) = 4 I/Os.

5. (2.5 points) In **the worst case**?

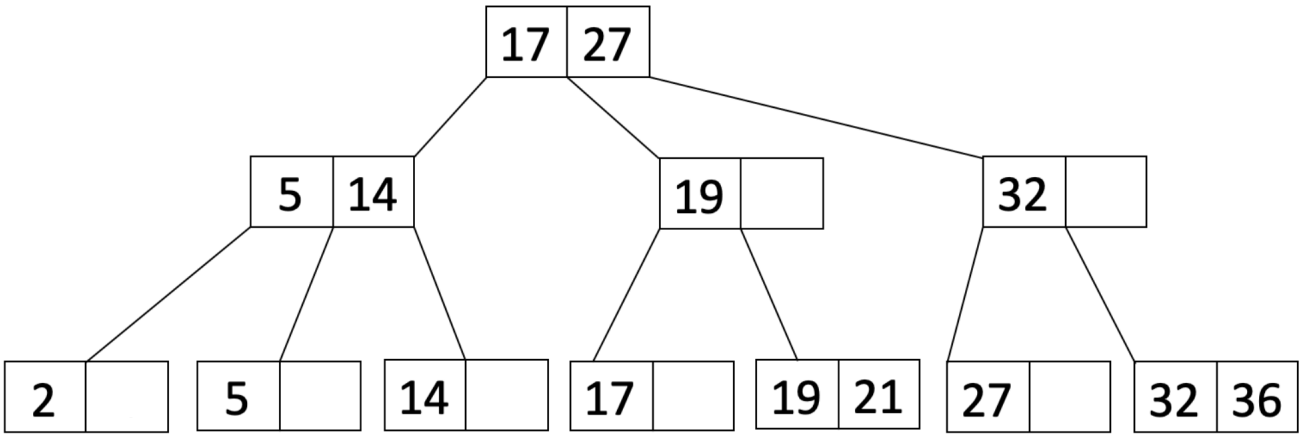
> **Solution:** First, we calculate the number of pages in the heap file – 200MB/1MB = 200 pages. Then, we calculate the number of data pages. This can be algebraically found – x/19 + x = 200, so x = 190 data pages, making there 10 header pages. In the worst case, we need to read all 10 header pages because the free page that is guaranteed to contain free space can exist at the end of the file. Since fixed length records are used and all records can perfectly fit in a data page, a record can be inserted as long as there is space.
>
> 10 (read page directory pages) + 1 (read data page) + 1 (write updated data page) + 1 (write updated header page) = 13 I/Os

3 B+ Trees (15 points)

1. (5 points) Which of the following statements are true? **There may be zero, one, or more than one correct answer.**
   A. The maximum number of entries a node can hold in an order d tree is $2d + 1$.
   B. Clustered B+ trees are always more efficient (in terms of I/O's) than unclustered B+ trees for equality search queries on the key the B+ tree was constructed.
   C. When an insertion into any B+ tree causes an inner node (non-leaf node) to split, the height of the tree always increases.
   D. For at least one type of B+ tree, it is possible to have repeated keys in inner nodes (non-leaf nodes).
   E. When bulk loading B+ trees, we only need to keep the path from the root to the rightmost leaf in memory for insertions and can disregard the rest of the tree.

Solution: Choice A is false because the maximum number of entries a node can hold in an order d tree is $2d$. Choice B is false because the I/O cost is the same if there is only one record with the corresponding key in the equality search. Choice C is false because an insertion into a B+ tree will cause the height to increase if the root node splits (not just any inner node). Choice D is true (ex: inserting 500 different records all with key 1 into an order 1, alternative 1 B+ will definitely create repeated keys in inner nodes). Choice E is true because we only need to keep the rightmost path from root to leaf in memory because the left side of the tree does not change when bulk loading.

Consider the following order $d = 1$ B+ tree where the keys represent ages in a people table:

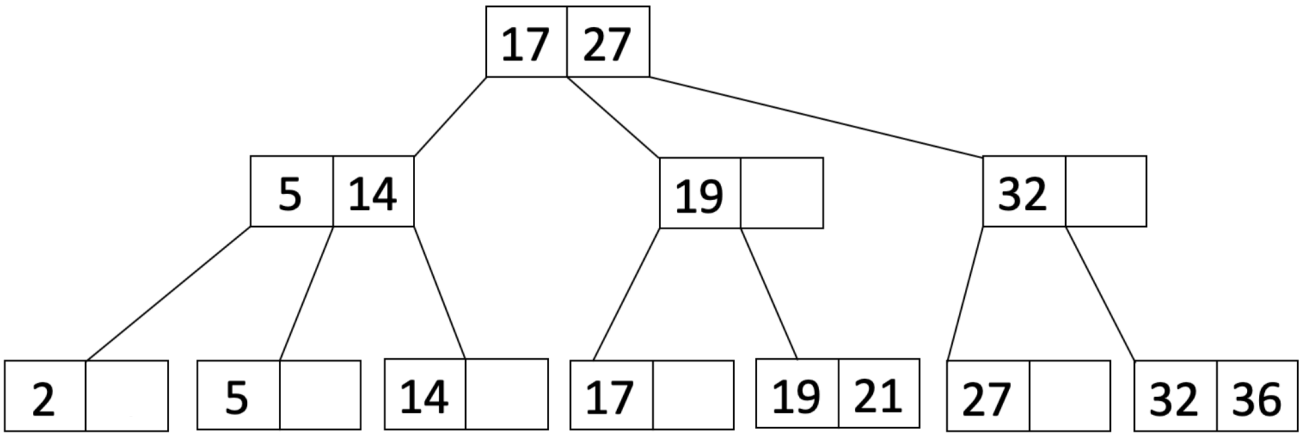


2. (4 points) For this question, let this B+ tree be Alternative 3 and unclustered. We want to insert Jenny who is age 21 into the table if she is not in the table already. If there are at most 10 people of the same age before the insertion, what is the I/O cost for inserting Jenny into the table in **the worst case**? Assume that the data pages in the table are all 2/3 full and that we can hold at least two pages in memory at any given time.

Solution: The answer is 15 I/Os. We need 3 I/Os to traverse the tree to get to the correct leaf node. In addition, we need to get all people that are age 21 to make sure none of them are Jenny which will take 10 I/Os in the worst case. If none of them are Jenny, we must write Jenny to some data page and add a reference to this record in the leaf page in the B+ tree - this is 2 writes aka 2 I/Os. Therefore, $3 + 10 + 2 = 15$ I/Os.

A clarification was made and announced during the exam to reflect the fact that there were at most 10 people of the same age before the insertion.

Consider the same order $d = 1$ B+ tree but now Alternative 2 and clustered:

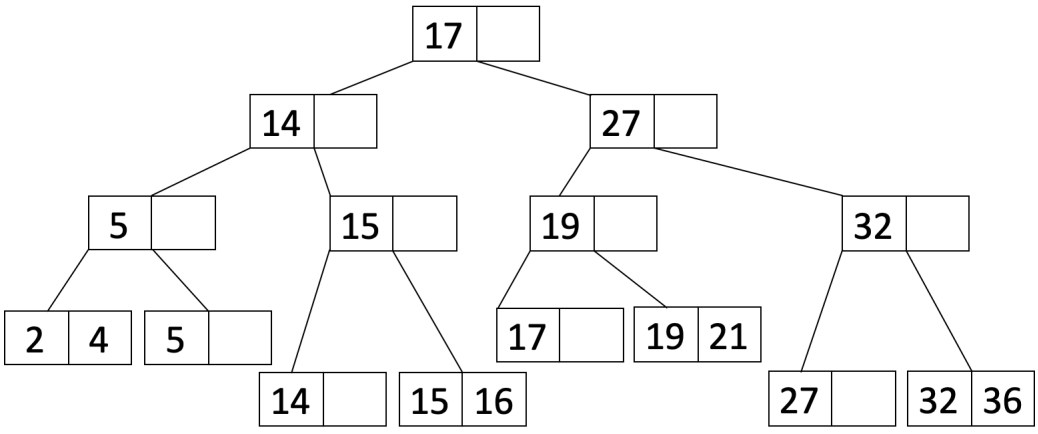


3. (1 point) What is the minimum number of inserts we can perform to change the height of the tree?

Solution: The answer is 2 inserts. For example, we can insert 3 and 4 to increase the height of the tree.

4. (2 points) If we inserted (in order) 4, 15, and 16 into this tree, how many nodes split?

Solution: The answer is 3 splits, we will cause the nodes with $(14, 15)$, $(5, 14)$, and $(17, 27)$ to split. This will result in the following tree:



5. (3 points) Using the resulting tree from the last question (after the insertions), what is the maximum number of inserts we can do without changing the height of that tree?

Solution: The answer is 42 inserts. An order $d = 1$ tree with a height of 3 can hold a maximum of 54 entries. We currently have 12 entries so $54 - 12 = 42$ inserts.

4 Buffer Management (13 points)

1. (5 points) Which of the following statements are true? There may be zero, one, or more than one correct statement.

   A. LRU always performs at least as well as the Clock Policy, but the Clock Policy is easier to compute.

   B. Clock Policy always performs better than LRU when the page is not in the cache and the cache is full.

   C. The dirty bit is used by the Clock Policy to determine if a page can be replaced.

   D. The pin count of a page can only be decremented by the buffer manager.

   E. MRU will often have a better hit rate than LRU if LRU is suffering from sequential flooding.

Solution: a. False - There are access patterns in which clock performs better
b. False - Clock may require iterating through every page in the cache twice to evict a page.
c. False - The dirty bit just tells us whether or not we have to write back to disk when the page is replaced.
d. False - The pin count is decremented by whoever requested the page to signify that they are done with it.
e. True - This is the major use case for MRU

For the following parts, we are given an initially empty buffer pool with **4** buffer frames. For the access pattern:

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

Answer the following questions:

2. (1 point) What is the hit rate for MRU?
3. (1 point) What page is not in the buffer pool when MRU completes?
4. (1 point) What is the hit rate for LRU?
5. (0.5 points) Was the last B a hit for LRU?
6. (0.5 points) Was the last C a hit for LRU?
7. (0.5 points) Was the last D a hit for LRU?
8. (0.5 points) Was the last E a hit for LRU?

Solution:

2. 7/13
3. C
4. 3/13
5. No
6. No
7. No
8. No

The solutions come directly from the following diagram:

MRU

A					X		X	X				
	B								X			
		C								X	D	
			D	E		X						X

LRU

A				E		X					D	
	B				A		X	X				E
		C							B			
			D							C		

For the following parts, we are again given an initially empty buffer pool with **4** buffer frames. For the access pattern:

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

Answer the following questions about the **CLOCK** policy:

9. (1 point) How many hits were there?
10. (1 point) What page is not in the buffer pool when CLOCK completes?
11. (1 point) What page is in frame 1 (the frame that started with A in it) at the end of the algorithm?

Solution:

9. 3
10. E
11. D

The solutions come directly from the following diagram:

Clock

A				E				X			D
	B					A					
		C					X		B		
			D		X					C	

Just for fun!

12. (0.0001 points) Oranges are hybrids of mandarins and what other fruit?

Solution: Pomelos!

5 Sorting & Hashing (18 points)

For parts 1 and 2, you may assume the following:

- The file size is 1000 pages
- There are 10 pages of memory available

1. (2 points) How many passes are required to merge the runs after they are sorted?

   Solution: We have $\lceil log_{B-1}\lceil \frac{N}{B} \rceil \rceil = \lceil log_9 \lceil \frac{1000}{10} \rceil \rceil = \lceil log_9 100 \rceil = 3$ passes. Note that we are asking for the number of passes required to merge the already sorted runs.

2. (1 point) *Assuming the answer to question 1 is **5***, from start to finish, what is the total number of I/Os, in pages, required to sort this file?

   Solution: $2N(1+5) = 2000(1+5) = 12000$

3. (2 points) Suppose we are given a new file to sort every 3 minutes. It takes about 1 minute to complete a pass (i.e. to scan, perform sorting/merging computation, and to write, where all buffers are in use). If we want to keep up with the rate at which we receive the files, what is the largest file size we are able to receive? Assume B=10, as in parts 1 and 2.

   Solution: $B(B-1)^2 = 10 \cdot (9)^2 = 810$

4. (1 point) True/False: If we double the buffer pool size, then we can halve the number of I/Os in pass 0.

   Solution: False, we still need to read and write the same number of pages into and out of the buffers.

For parts 5 to 8, you may assume the following:

- The file size is 200 pages
- There are 6 pages of memory available

A newly hired TA, Jenny, wrote a suboptimal hash function HP, which she used in the first pass of external hash on a set of data. Consequently, she ended up with partitions of size 20, 30, 30, 40, and 80 pages. Due to the lack of time, she cannot rerun the first pass with a better hash function; she will have to work with the current partitions. Not afraid to give up and take shortcuts, she reaches out to Lakshya, who, after emitting a long and audible sigh, provides her with a variety of perfect hash functions $H_{P1}, H_{P2}, ...$ that can evenly distribute values. She decides to use these for the remaining partitions needed to complete the hashing.

5. (2 points) Including the first pass over the data using HP, what is the most number of times a single record is read from disk?

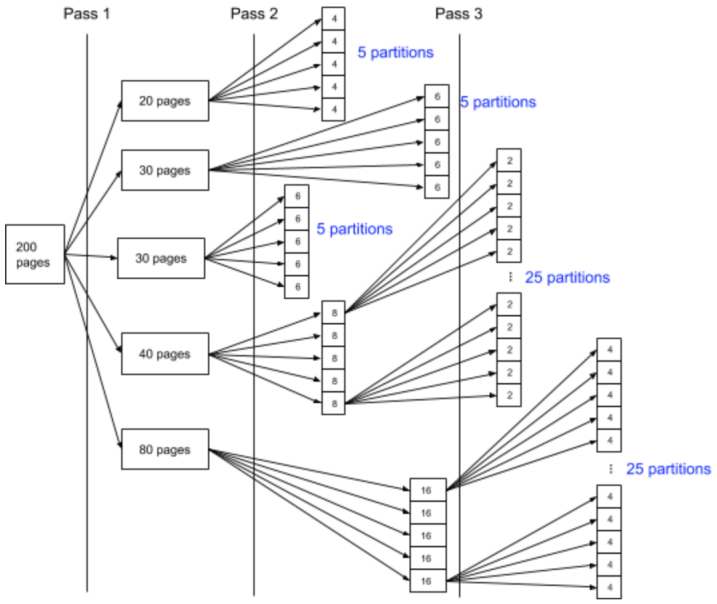
Solution: 3. We need to get the partition sizes to be less than or equal to $B = 6$ pages. This means after pass 1, we need to recursively partition the $B - 1$ partitions. The first three partitions only need to be partition one more time, while the other two partitions need to be partitioned two more times. Therefore, records in the latter two partitions must be read 3 times.

In the exam, we specified that this was to only consider the divide phase. However, as the wording on the initial test was ambiguous and could have been interpreted to mean the divide and conquer phase, we accepted either 3 or 4 as an acceptable answer.

A diagram is included in the solution of the next section.

6. (3 points) Calculate the total number of hash partitions at the end of the divide phase.

Solution: There are 65 partitions. To see this, consider the following diagram:



By the end of pass 2, we have 15 partitions that is less than the memory size (6 pages). These do not need to be hashed any further. By the end of pass 3, the 10 partitions (remaining from pass 2) that are further hashed, giving us 50 partitions.

7. (4.5 points) Calculate the number of I/Os for the divide phase. **Do not include the I/Os for the conquer phase**

Solution: 1070 I/Os.

For each pass, we have to read and write the file. In the first pass, we will read and write out 200 pages, so we have 400 I/Os for pass 1.

Following this, we hash the partition of 20 pages into 5 partitions of 4 pages each – this would be 20 + 5*4 = 40 I/Os. We hash each partition of 30 pages 5 partitions of 6 pages each. This will be 60 I/Os for each of the two partitions of 30 pages.

The partition of 40 pages will first hash into 5 partitions of 8 pages each (40 I/Os to read, 40 I/Os to write). Then, for the third partitioning phase, we hash *each of these partitions of 8 pages* into 5 partitions of **2 pages each** (because 8/5 = 1.60, and there is no such thing as 1.6 pages, we must round up – $\lceil \frac{8}{5} \rceil = 2$). Now, we'll get, in total, 25 partitions of 2 pages each. We'll still read in 40 pages worth of data in the third partitioning phase, but we'll write out 50 pages worth of data – for the partition of 40 pages, we've just done 40 + 40 + 40 + 50 = 170 I/Os.

80 pages of the file are partitioned into 5 partitions of 16 pages each (80 I/Os to read, 80 to write). Each of those 5 partitions are partitioned into 5 partitions of 4 pages *each* ($\lceil \frac{16}{5} \rceil = 4$ – we must do this as 16 does not go evenly into 5, and we cannot just have 3.2 pages worth of data written to disk; that is actually 4 pages). In total, we end up with 25 partitions of 4 pages each. So for the partition of 80 pages, we have performed 80 + 80 + 80 + 25 * 4 = 340 I/Os.

So, the divide phase takes 400 + 40 + 60 + 60 + 170 + 340 = 1070 I/Os.

Inspired by the clean and effective code that the head TA wrote, the new TA decides to write and use a perfect granular hash function HR for the conquer phase of external hash.

8. (2.5 points) How many I/Os are performed in this phase of external hash?

Solution: Notice that in the divide phase, we ended up with 5 partitions of 4 pages each, 10 partitions of 6 pages each, 25 partitions of 2 pages each, and 25 partitions of 4 pages each. In total, as we must read and write each partition to conquer, we have to perform 2*(5*4 + 10*6 + 25*2 + 25*4) = 460 I/Os.