I SQL [10 points]

It is October 2024, and recent news highlights that the Chinese A-shares market has experienced a surprising surge following a series of government policies aimed at supporting the economy. Inspired by this growth and optimism, you view this as an opportunity to buy stocks during this bullish market period! However, before making any investments, it's crucial to research promising stocks that are likely to perform well in the long term. As a ShanghaiTecher who is a beginner in the world of finance, your first step is to find trusted and credible reputable analysts to guide your investment decisions. You are provided with the following tables.

```
/* This table is a list of stocks and their stock symbol and the
company they correspond to.  */

CREATE TABLE Stocks (
    stock_id INTEGER PRIMARY KEY,
    company_id INTEGER REFERENCES Companies,
    stock_symbol VARCHAR(4) NOT NULL
);

/* This table contains info about companies:  the company name,
whether or not the company is hiring and what type of business
the company does.  */

CREATE TABLE Companies (
    company_id INTEGER PRIMARY KEY,
    company_name VARCHAR(20) NOT NULL,
    is_hiring BOOLEAN NOT NULL,
    business_type VARCHAR(20) NOT NULL
);

/* This table contains info about analysts:  the company they work for,
their first name, last name and whether or not they are reputable.  */

CREATE TABLE Analysts (
    analyst_id INTEGER PRIMARY KEY,
    company_id INTEGER REFERENCES Companies,
    first_name VARCHAR(20) NOT NULL,
    last_name VARCHAR(20) NOT NULL,
    is_reputable BOOLEAN NOT NULL
);

/* This table contains info about the analysis an analyst does,
which includes the rating an analyst gives a stock and the date the
analysis was given (year, month, day).  */

CREATE TABLE Analysis (
    analysis_id INTEGER PRIMARY KEY,
    analyst_id INTEGER NOT NULL REFERENCES Analysts,
    stock_id INTEGER NOT NULL REFERENCES Stocks,
    rating INTEGER NOT NULL,
    year INTEGER NOT NULL,
    month INTEGER NOT NULL,
    day INTEGER NOT NULL
);
```

Note: Assume there is a 1:1 correlation between `stock_id` and `company_id` (meaning every entry in the Stocks table corresponds to a unique entry in the Companies table) for all the following questions.

1. (2 points) You want to find the name of all the companies with at least one reputable analyst. If there are multiple companies with the same name with at least one reputable analyst, we want the same company name to be returned multiple times.

Does the following query achieve this? Mark T for Yes and F for False. You don't need to explain the reasons.

```
SELECT DISTINCT Companies.company_name
FROM Analysis, Stocks, Companies
WHERE Analysis.stock_id = Stocks.stock_id
AND Stocks.company_id = Companies.company_id
AND Analysis.rating <= 2;
```

2. (2 points) Same question but instead for this query:

```
SELECT DISTINCT Companies.company_name
FROM Companies INNER JOIN
(SELECT Analysts.company_id
 FROM Analysts
 WHERE Analysts.is_reputable
 GROUP BY Analysts.company_id) AS c
ON Companies.company_id = c.company_id;
```

3. (2 points) Same question but instead for this query:

```
SELECT Companies.company_name
FROM (SELECT company_id
      FROM Companies

      EXCEPT

      SELECT DISTINCT company_id
      FROM Analysts
      WHERE NOT is_reputable) AS c
INNER JOIN Companies
ON c.company_id = Companies.company_id;
```

1. F. We did not include the Analyst table in our query. We need to use the Analysts table because we need to use the `is_reputable` field.

2. F. If two companies have the same name and both of these companies have reputable analysts, this query will only output the company name once. It should be output twice.

3. F. In the case where a company has two analysts, one which is reputable and another which is not, the `company_id` will then be excluded from the inner query (`SELECT company_id FROM Companies EXCEPT (SELECT DISTINCT company_id FROM Analyst WHERE NOT is_reputable)`).

4. (4 points) Your friend tells you about a good analyst with the first name "Raymond". However, there are many analysts named "Raymond"! You now want to find the **number of companies** that had a reputable analyst with the first name "Raymond" who performed some analysis for the company in the year **2024**. If the analyst did not analyze any companies in 2024, your query should output 0 for that analyst. Queries should return the `analyst_id` and the number of companies the analyst covered. **There can be zero, one, or more than one correct queries.**

Assume there is a 1:1 correlation between `stock_id` and `company_id` (meaning every entry in the Stocks table corresponds to a unique entry in the Companies table).

A.

```
WITH Raymond AS (
    SELECT Analysts.analyst_id AS analyst_id
    FROM Analysts
    WHERE Analysts.is_reputable AND Analysts.first_name = "Raymond"),

Raymond_stocks AS (
    SELECT DISTINCT Raymond.analyst_id AS analyst_id, Analysis.stock_id AS stock_id
    FROM Raymond LEFT OUTER JOIN Analysis
    ON Analysis.year = 2024 AND Raymond.analyst_id = Analysis.analyst_id)

SELECT Raymond_stocks.analyst_id, COUNT(Raymond_stocks.stock_id)
FROM Raymond_stocks
GROUP BY Raymond_stocks.analyst_id;
```

B.

```
SELECT Raymond.analyst_id, COUNT(Analysis.stock_id)
FROM Analysis INNER JOIN
(SELECT analyst_id FROM Analysts
 WHERE first_name = "Raymond" AND is_reputable) AS Raymond
ON Analysis.analyst_id = Raymond.analyst_id AND Analysis.year = 2024
GROUP BY Raymond.analyst_id;
```

C.

```
WITH num_analyzed AS (
    SELECT Stocks.company_id AS company_id, COUNT(Analysis.analyst_id) as count
    FROM Stocks LEFT OUTER JOIN Analysis
    ON Stocks.stock_id = Analysis.stock_id AND Analysis.year = 2024
    GROUP BY Stocks.company_id
    HAVING COUNT(Analysis.analyst_id) >= 1)

SELECT Analysts.analyst_id, num_analyzed.count
FROM Analysts INNER JOIN num_analyzed
ON Analysts.company_id = num_analyzed.company_id
WHERE Analysts.first_name = "Raymond" AND Analysts.is_reputable;
```

D. None of the above.

4. A.

Query A: The Raymond table first gets the `analyst_id` of all analysts that are reputable whose first name is "Raymond". The Raymond_stocks table gets all unique `stock_ids` that an analyst from the Raymond table analyzed in 2024. The DISTINCT keyword is important here because Raymond could have had multiple analyses of the same stock in 2024. LEFT OUTER JOIN is important here because it is possible an analyst did not have any analysis in 2024. We then GROUP BY and count the number of stocks an analyst analyzed. This is guaranteed to be the total number of companies because `stock_id` and `company_id` have a 1:1 correlation.

Query B: Using an INNER JOIN is incorrect because our output will then not have analysts who analyzed 0 companies.

Query C: The num_analyzed table finds the number of times a company is analyzed in 2024 by analysts. Our output is returning an analyst and the number of times the company they work for is analyzed, which is not what we are asking for.

II Relational Algebra [10 points]

We have relations $R(A, B)$ and $S(C, D)$ where A, B, C, D are all non `NULL` integers, assume R and S are sets, and we use set semantics for relational algebra and bag (multiset) semantics for SQL. In the following we write A instead of $R.A$ etc when the context is clear.

Is each of the following relational algebra expression or SQL query equivalent, i.e., they always return the same results regardless of the contents of the input relations? Mark T for Yes and F for False. You don't need to explain the reasons.

1. (2 points) $S \bowtie_{D=A} R =$ `SELECT * FROM R, S WHERE R.A = S.D`

Equivalent?

2. (2 points) $\pi_A(R \times S) = \pi_A(R \bowtie_{A=C} S) \cup \pi_A(R \bowtie_{A \neq C} S)$

Equivalent?

3. (2 points) $R = \pi_{R.A,R.B}(R \bowtie_{R.A=T.A} \rho_{T(A,B)}(R))$

Equivalent?

4. (2 points) $\pi_{A,B}(R \bowtie_{B \neq C} S) =$
`SELECT * FROM R INTERSECT SELECT R.A, R.B FROM R, S WHERE R.B = S.C`

Equivalent?

5. (2 points) $\gamma_{C,\max(D)}(\sigma_{C=1}(S)) =$
`SELECT S.C FROM S WHERE S.C = 1 ORDER BY S.C LIMIT 1`

Equivalent?

1. T.
2. T.
3. T.
4. F.
5. F.

III Disk, Buffers and Files [10 points]

It's Friday morning, and as usual, there's no way you're making it to your 8:15 AM CS150A lecture. So, like every week, you plan to catch up later by watching the recording on your school's ecourse platform. You grab your coffee, get comfortable, open the website... But to your shock, all the lecture videos have mysteriously vanished! To resolve this issue, you contact IT Services, and they ask for your help in considering a better design. Time to put your database skills to the test! Consider the following schema.

```
CREATE TABLE ecourse (
    lecture_id INTEGER PRIMARY KEY,
    week INTEGER NOT NULL,
    topic VARCHAR(30) NOT NULL,
    num_student INTEGER,
    enable_recording BOOLEAN
);
```

- `ecourse` is stored in a Heap File Page Directory implementation. There are 25 data pages and each header page has 6 entries.
- Data pages follow the slotted page layout as presented in lecture.
- On each data page, 8 bytes are reserved for the slot count and pointer to free space. Each slot is 8 bytes.
- Records are stored as variable length records.
- The record header contains a bitmap to track all fields that can be `NULL`. The bitmap is as small as possible, rounded up to the nearest byte.
- Integers and pointers are 4 bytes. Booleans are 1 byte.
- Each page is 1 KB (1024 Bytes).
- There are no indices on any field.

1. (2 points) What is the maximum size of a `ecourse` record in bytes?

2. (2 points) What is the maximum number of `ecourse` records that can fit on a data page?

For questions 3-4, assume the queries are independent of each other; i.e. query 4 is run on a copy of the file that has never had query 3 run on it. The buffer is large enough to hold all data and header pages, and starts empty for each question.

3. (3 points) What is the **worst case** cost in I/Os for the following query?

```
DELETE FROM ecourse WHERE enable_recording = FALSE;
```

4. (3 points) What is the **best case** cost in I/Os for the following query?

```
SELECT num_student FROM ecourse WHERE lecture_id = CS150A;
```

Your answer:

1. The maximum record size is 48 bytes. 1 byte for null bitmap, 4 bytes for pointer to end of `topic`, 4 bytes each for `lecture_id`, `week`, `num_student`, 30 bytes for `topic`, and 1 byte for `enable_recording`.

2. The minimum record size is $1 + 4 + 4 + 4 + 4 + 1 = 18$ bytes, when `topic` is an empty string. This comes from 1 byte for the bitmap, 4 bytes for each of the integers, 4 bytes for the pointer in the record header to the end of `topic`, and 1 byte for the boolean. Subtracting 8 bytes from the page size of 1024 bytes for the slot count and pointer to free space, leaves 1016 bytes for slots and records. The maximum number of records that can fit on the slotted page is $\left\lfloor \frac{1016}{18+8} \right\rfloor = 39$ records.

3. 60 I/Os. In the worst case, there is a record with `enable_recording = FALSE`, on every data page. This involves reading all header and data pages, and writing all header and data pages. This is 2 * (5 + 25).

4. 2 I/Os. The best case is when the first directory entry on the first header page contains the matching record. Since `lecture_id` is a `PRIMARY KEY`, no other data pages need to be read.

IV File Organization [10 points]

In I/O (Input/Output) cost model for analysis, we define:
B: the number of data blocks in the file;
R: the number of records per block;
D: average time to read/write one disk block.
Please answer the following questions:

1. [**3 points**] Please compare the I/O cost of equality search in heap file and sorted file? Correct answer without proper explanation will earn no scores. (Use the approximate value when computing number of pages touched.)

2. [**3 points**] Please compare the I/O cost of deletion of a single record in heap file and sorted file? Correct answer without proper explanation will earn no scores.

3. [**4 points**] Is there any type of file organization to reduce the I/O cost of deletion? If yes, show the type and its I/O cost.

Your answer:

1. Heap file: 0.5 x B $\times$ D. Pages touched on average = 0.5 x B.
   Sorted file: $(log2B)D$. Pages touched in binary search = log2B

2. Heap file: $(0.5B + 1)D$. The first part $(0.5B)$ $\times$ D denotes the average cost of locating the record, and the second part $1D$ is the cost of writing the revised page back into the disk.
   Sorted file: $(log2B + B)D$. The first part $(log2B)D$ denotes the average cost of locating the record, and the second part BD is the average cost of shifting the rest pages by 1 record.

3. The indexed file with B+ tree index constructed on the sorted key. The I/O cost is $(log2B)D$.

V Indexes and B+ Trees (1) [12 points]

Given the following (degree d = 1) B+ tree:

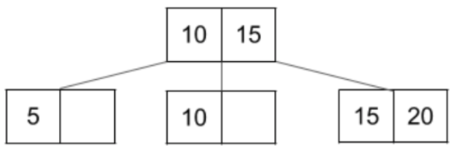

Figure 1: B+ Tree

1. [**4 points**] Draw what the tree looks like after adding 2, 6, and 12 in that order.
   For the following questions, consider the tree directly after 2, 6, and 12 have been added.
2. [**4 points**] What is the maximum number of inserts we can do without changing the height of the tree?
3. [**4 points**] What is the minimum number of inserts we can do that will change the height of the tree?

Your answer:

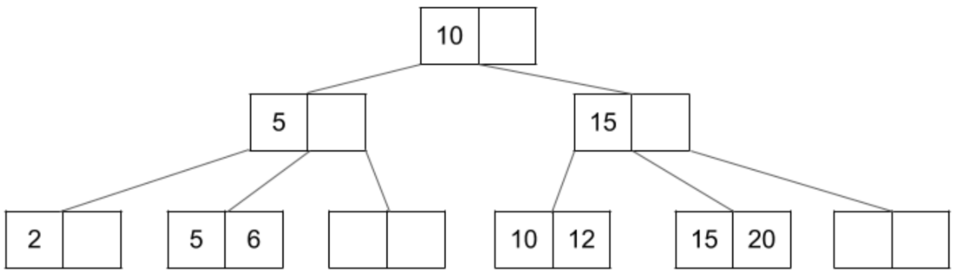

Figure 2: B+ Tree

2. Answer: 11 inserts. The maximum number of keys for a fixed height h is given by $2d \cdot (2d+1)^h$. We have $d = 1$ from the question, and $h = 2$ (we must remember h is the number of non-root layers). Plugging these numbers in gives us $2 * 3^2 = 18$. Now, we already have 7 keys in the tree, so we can insert 18-7=11 more.
3. Answer: 4 inserts (e.g. 16, 17, 18, 19). The idea is to repeatedly insert into the fullest nodes; this is hopefully apparent from understanding how and when B-trees split.

VI Indexes and B+ Trees (2) [8 points]

1. **[2 points]** Select all of the following statements which are true. There is at least one correct answer.

   A. As the height of a B+ tree increases, the lookup costs for an equality search grows linearly with respect to the height. Assume the B+ tree is alt 1.

   B. A leaf node cannot be a root node because leaf nodes cannot store the primary key of a record.

   C. When bulk loading, the fill factor of < 1 is used to reserve a percentage of space on each leaf node for future growth.

   D. When constructing a B+ tree, serial insertion of data is preferred over bulk loading as bulk loading can cause a B+ tree to be more unbalanced.

   B: False, leaf nodes can be root nodes.
   D: False, depending on the indices of the data, the order of insertion can cause the tree to be unbalanced.

Professor Cheung has an alternative 1 B+ tree of height h = 2 and order d = 7 to store all restaurants he has reviewed for his blog. Assume that the B+ tree is at its full capacity (all inner and leaf pages are full). Note: index pages include both leaf and inner node pages.

2. **[2 points]** In the worst case, what is the largest number of pages that will be read in a range search? Assume no pointers to leaf nodes exist in the header page.

   A. 194
   B. 196
   C. 225
   D. 227
   E. 343
   F. 345

   We would need to traverse to the leftmost leaf node, reading in 1 root node and 1 index node. In the worst case all $(2 * 7 + 1)^2 = 225$ leaf nodes are read. $1 + 1 + 225 = 227$.

3. **[2 points]** In the best case, what is the fewest number of pages that will be written to when updating a single review/record? Assume the record sizes are fixed and the primary key is unchanged.

   A. 0
   B. 1
   C. 2
   D. 3
   E. 4

   When updating a single record in an alternative 1 B+ Tree, we only need to write to the index page that stores the desired record.

4. **[2 points]** In the best case, what is the fewest number of pages that will be written to when inserting a new review/record? (assume the insert is successful)

   A. 3
   B. 4
   C. 6
   D. 7
   E. None of the above

Since the tree is at full capacity, it will split at every level when inserting the next review/record. This means that two pages are written at every level, along with a new page for the root node. With 2 inner node levels and 1 leaf page level, there are 3 levels in total. Therefore in the best case, the number of pages that will be written to is $2 * 3 + 1 = 7$.

VII Buffer Management [10 points]

We're given a buffer pool with 3 pages, which is empty to begin with.

1. **[4 points]** For the following statements, choose whether the statement is always true, sometimes true, or never true.

   (a) The number of page hits using Clock policy is less than or equal to the number of page hits using LRU policy for the same access pattern.

   A. Always true.

   B. Sometimes true.

   C. Never true.

   Clock results in a greater number of hits compared to LRU when sequential flooding occurs.

   (b) LRU performs poorly (large proportion of page misses) when the number of pages in a repeated sequential scan is greater than the buffer pool size.

   A. Always true.

   B. Sometimes true.

   C. Never true.

   By definition, sequential flooding occurs when a repeating sequence of $|S|$ number of pages is read, where $|S| > B$. If this occurs, LRU will miss 100% of the time.

   (c) When running the Clock algorithm, the clock hand moves when a page miss occurs.

   A. Always true.

   B. Sometimes true.

   C. Never true.

   According to the Clock algorithm, the clock hand is always advanced after inserting a page into the buffer pool.

   (d) Assuming that the buffer is empty initially, performing a full scan over a table once is an example of a workload where MRU is better than LRU in terms of I/O cost.

   A. Always true.

   B. Sometimes true.

   C. Never true.

   If each page is only loaded one time, every page access will result in a miss. Therefore, all of the buffer replacement policies will perform the same.

2. **[2 points]** Consider the following page access pattern: A , B , C , D , E , C , C , E , D
   Which policy(s) has the least amount of cache hits?

   A. LRU

   B. MRU

   C. Clock.

3. **[2 points]** Consider the following page access pattern: A, B, E, D, C, C, E, D, A
   Which policy(s) has the most amount of cache hits?

   A. LRU

   B. MRU

   C. Clock.

4. **[2 points]** What's the hit rate using the policy you selected above?
   Your answer:
   $3/9 = 1/3$

VIII Iterations and Joins [10 points]

Considering the following two tables:

- Companies: (`company_id`, industry, ipo_date)
- NYSE: (`company_id`, date, trade, quantity)

We have 20 pages of memory, and we want to join two tables Companies and NYSE on C.company_id = N.company_id. Attribute company_id is the primary key for Companies. For every tuple in Companies, assume there are 4 matching tuples in NYSE.

NYSE contains [N] = 100 pages, NYSE holds pN = 100 tuples per page.

Companies contains [C] = 50 pages, C holds pC = 50 tuples per page.

There are unclustered B+ indexes on C.company_id and N.company_id – for both indexes, assume it takes 2 I/Os to access a leaf.

1. (2 points) How many disk I/Os are needed to perform a simple nested loops join?

2. (2 points) How many disk I/Os are needed to perform a block nested loops join?

3. (2 points) How many disk I/Os are needed to perform an index nested loops join?

4. (2 points) For this part only, assume the index on NYSE.company_id is clustered. What is the cost of an index nested loops join using companies as the outer relation?

5. (2 points) How many disk I/Os are needed to perform a sort merge join? This is ordinary sort-merge, not optimized sort-merge.

Your answer:

1. [C] + pC*[C]*[N] = 50 + 50*50*100 = 250050

2. (# pages in smaller relation) + ceil((# pages in smaller relation) / (# pages in memory – 2)) * (# pages in larger relation) = 50 + ceil(50/18) * 100 = 350 I/Os

3. [C] + [C] * pC * (cost to find matching NYSE tuples) = 50 + 50 * 50 * (2 + 4) = 15,050

4. [C] + [C]*pC * (cost to find matching NYSE tuples) = 50 + 50 * 50 * (2 + ceil(4/100)) = 7,550 I/Os

5. Sorting C: 4 * (50 pages) = 200 I/Os
Sorting N: 4 * (100 pages) = 400 I/Os
Joining: [C] + [N] = 150 I/Os
Total: 200 + 400 + 150 = 750 I/Os

IX External Sorting [10 points]

Your "dream machine" allocates 64MB for the buffer (memory) for its external sorting algorithms. All input is read from disk and all output is written to disk. The I/O cost is the number of page reads/writes, where each page is 256KB large. Note that 1024KB = 1MB.

1. [**3 points**] What's the I/O cost of fully sorting a 4096 MB file with external merge sort? Explain (Just write down the equation form is also OK).

   Solution
   B = 256 pages
   N = 16384 pages
   passes = 2
   I/O = 2*2*4096 MB = 16384 MB = 65536 pages

2. [**3 points**] Suppose we reduce the size of our buffer to 32 MB. What is the largest file size (in MB) that we can externally sort in 2 passes? Explain (Just write down the equation form is also OK).

   Solution
   B = 128 pages
   N = B(B-1) = 16256 pages = 4064 MB

3. [**4 points**] Write down whether the following statements are true or false.

   (a) Deduplicating the file using hashing will have a higher IO cost than sorting the file (without deduplicating). False. Consider an extreme case, where the entire file consists of duplicates - there will only be one tuple to output (and only one pass over the input).

   (b) If external hashing recursively partitions on one partition, it will do so on all partitions. False. You only need to recursively partition oversized partitions.

Your answer:

X Hash and Joins [10 points]

Considering the following two questions concerning external hashing and grace hash join.

1. (2 points) We want to hash $N = 100$ pages using $B = 10$ buffer pages. Suppose in the initial partitioning pass, the pages are unevenly hashed into partitions of 10, 20, 20, and 50 pages. Assuming uniform hash functions are used for every partitioning pass after this pass, what is the total I/O cost for External Hashing?

2. (8 points) We have 2 tables - Catalog and Transactions. Catalog has a total of 100 pages and 20 tuples per page. Transactions has a total of 50 pages and 50 tuples per page. Assume that the distribution among the key that we are joining on is uniform for the two tables.

(a) (2 points) If we had 10 buffer pages, how many partitioning phases would we require for grace hash join? Consider which table we should build the hash table in the probing phase on.

(b) (2 points) What is the I/O cost for the grace hash join then?

(c) (2 points) For the above question, if we only had 8 buffer pages, how many number of partition phases would there be?

(d) (2 points) What will be the I/O cost?

Your answer:

1.

634 I/Os. (I/Os are in bold) **100** pages are read and hashed into partitions of size 10, 20, 20, and 50. Summing these results in **100** pages written.

The partition of size 10 can fit into memory ($10 \leq (B = 11)$) so we can build the in-memory hash table for it. This involves reading in **10** pages, building the hash table, and writing **10** pages.

Since the partitions of sizes 20 and 50 cannot fit into memory, they must be recursively partitioned. For one of the partitions of size 20, **20** pages are read, hashed into 9 partitions of size 3 ($\lceil \frac{20}{B-1} \rceil = 3$), and thus **27** pages are written (9 partitions x 3 pages/partition). Since each of these partitions of size 3 can now fit into memory, we read in a total of **27** pages, build the in-memory hash tables for each partition, and write out **27** pages. The same process after the initial partition is repeated for the other partition of size 20 which results in another $20 + 27 + 27 + 27 =$ **101** I/Os.

For the partition of size 50, **50** pages are read, hashed into 9 partitions of size 6 ($\lceil \frac{50}{B-1} \rceil = 6$), and thus **54** pages are written (9 partitions x 6 pages/partition). Since each of these partitions of size 6 can now fit into memory, we read in a total of **54** pages, build the in-memory hash tables for each partition, and write out **54** pages.

Summing all I/Os results in a total of **634** I/Os.

2.

(a) T is smaller, need partitions of T to be at most $B - 2 = 8$ pages. After 1 partitioning pass, we have partitions of size 6. $6 \leq 8$ so it's small enough meaning we still only need **1 partitioning pass**.

(b) We need 1 partitioning pass.
Partitioning phase:
$\lceil \frac{C}{B-1} \rceil = 12$ pages per partition for C, 12(9) pages in total after partitioning
$\lceil \frac{T}{B-1} \rceil = 6$ pages per partition for T, 6(9) pages in total after partitioning
Partitioning I/Os: $100 + 12(9) + 50 + 6(9) = 312$
Probing phase: $12(9) + 6(9) = 162$ I/Os
Total: $312 + 162 =$ **474 I/Os**

(c) T is smaller, need partitions of T to be at most $B - 2 = 6$ pages. After 1 partitioning pass, we have partitions of size 8, which is too big. We need a second partitioning pass. $8/7 = 1.1 \rightarrow 2$ pages, which is small enough so we need **2 passes**.

(d) Partitioning phase:
$\lceil \frac{C}{B-1} \rceil = 15$ pages per partition for C

$\lceil\frac{T}{B-1}\rceil = 8$ pages per partition for T
$\lceil\frac{C}{B-1}\rceil = 3$ pages per partition for second pass for C
$\lceil\frac{T}{B-1}\rceil = 2$ pages per partition for second pass for T
Partitioning I/Os: [100 + 50] (1st read) + [15(7) + 8(7)] (first write) + [15(7) + 8(7)] (second read) + [3(49) + 2(49)] (second write) = 717 I/Os
Build and Probe Phase: 3(49) + 2(49) = 245 I/Os
Total: 717 + 245 = **962 I/Os**