CS 186 Fall 2020 — Sample Exam.

Midterm 2

INSTRUCTIONS

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

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

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

○ You must choose either this option

○ Or this one, but not both!

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

☐ You could select this choice.

☐ You could select this one too!

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

Preliminaries

1 CS 186 Fall 2020 Midterm 2 (Online)

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

1.0.1 Contents:

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

1.0.2 Taking the exam:

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

1.0.3 Aids:

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

1.0.4 Grading Notes:

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

(a) What is your full name?

(b) What is your student ID number?

(c) Who do you believe will win the election?

1. (21 points) Iterators/Joins

Poll volunteer Jerry is trying to register eligible voters for the election. He has a list of all residents in his county who have registered (Table X) which takes up 25 pages and a list of eligible voters (Table Y) which takes up 30 pages.

For all parts of this question, do not include the units in your answer. For example, if the answer is 5 I/Os, just write “5” or if the answer is 4 passes, just write “4.”

(a) (2 pt) Jerry wishes to join table X (25 pages) and table Y (30 pages) on social security number. His computer has 10 buffer pages. How many IOs will a Page Nested Loops Join of these two tables cost?

775

From our PNLJ formula, we first calculate the cost with X as the outer relation and Y as the inner relation.

$$[X] + [X][Y] = 25 + 25 * 30 = 775$$

We then try Y as the outer relation and X as the inner relation.

$$[Y] + [Y][X] = 30 + 30 * 25 = 780$$

We take the fewer IO count. Partial credit of 1 point was awarded answers of 780.

(b) (1 pt) Poll volunteer Saurav tells Jerry that Block Nested Loops Join is better than Page Nested Loops Join as it will always reduce IO cost. Jerry knows this is true only for a certain range of buffer sizes B.

With our fixed table sizes of 25 and 30 pages, give a range for B in which the IOs for BNLJ is less than the IOs for PNLJ. You should express your answer as “L<=B<=U” (no spaces) where L and U correspond to the lower and upper bounds for which the statement is true. You may use INF to express infinity in your solution if needed.

4<=B<=INF

The lower bound is 4 because at 3 buffers BNLJ becomes PNLJ and offers no IO reduction. The upper bound is INF because PNLJ is limited to using 3 buffers, but BNLJ will be able to take advantage of the extra buffers and read in the outer relation as blocks.

(c) (9 points)

Jerry wishes for his output to be ordered, and he knows he can use Sort Merge Join refinements and optimizations to potentially reduce IO counts.

For all of the sort merge join questions, apply any SMJ optimizations/refinements possible. You may also assume that there are no duplicates in either of the two tables.

i. (2 pt) How many IOs will Sort Merge Join of X (25 pages) and Y (30 pages) with 10 buffer pages take? You should not assume that these tables are already sorted.

165

Because there are 25 pages of X and our buffer size is 10, the first pass of sorting will produce 3 runs of table X. Similarly, because there are 30 pages of X and 10 buffer pages, the first pass of sorting will produce 3 runs of Y. We notice here that we can use the SMJ optimization because the sum of the runs (3+3=6) is less than or equal to B-1. Thus, we can save one pass of writes and one pass of reads.

$$SMJFormula = Sort(X) + Sort(Y) + [X] + [Y] - 2([X] + [Y])$$

$$Sort(X) + Sort(Y) - [X] - [Y]$$

$$Sort(X) + Sort(Y) - 55$$

We now execute our external sorting formulas to calculate the costs to sort X and Y.

$$passes = 1 + \lceil \log_{B-1} \lceil X/B \rceil \rceil = 2$$

$$Sort(X) = 2 * passes * [X] = 100$$

$$passes = 1 + \lceil \log_{B-1} \lceil Y/B \rceil \rceil = 2$$

$$Sort(Y) = 2 * passes * [Y] = 120$$

Therefore, our total IO cost is 100 + 120 - 55 = 165

ii. (1 pt) Express your answer to the previous question in terms of M and N which represents the number of IOs needed to sort Table X and Table Y respectively. Please do not insert spaces in between (example answer: “M+N+100”).

M+N-55

iii. (2 pt) Just before he starts, Jerry's computer dies. He turns to his phone to do the join.

How many IOs will Sort Merge Join of X (25 pages) and Y (30 pages) with 3 buffer pages take? You should not assume that these tables are already sorted.

605

Our buffer is too small to use any optimizations. We need one page for pages of X, one page for pages of Y, and one page for the output.

$$SMJFormula = Sort(X) + Sort(Y) + [X] + [Y]$$
$$Sort(X) + Sort(Y) + 55$$

We now execute our external sorting formulas to calculate the costs to sort X and Y. We know that both tables can be sorted in 2 passes of external sorting.

$$passes = 1 + \lceil \log_{B-1} \lceil X/B \rceil \rceil = 5$$
$$Sort(X) = 2 * passes * [X] = 250$$
$$passes = 1 + \lceil \log_{B-1} \lceil Y/B \rceil \rceil = 5$$
$$Sort(Y) = 2 * passes * [Y] = 300$$

Therefore, our total IO cost is 250 + 300 + 55 = 605

iv. (1 pt) Express your answer to the previous question in terms of M and N which represents the number of IOs needed to sort Table X and Table Y respectively. Please do not insert spaces in between (example answer: "M+N+100").

M+N+55

v. (2 pt) Soon after, Jerry gets access to a large supercomputer!

How many IOs will Sort Merge Join of X (25 pages) and Y (30 pages) with 100 buffer pages take? You should not assume that these tables are already sorted.

55

Our buffer is large enough such that we don't need to do any external sorting on either table. Both fit in memory, so we can join them as well. Our only IO costs are from reading them into memory originally.

vi. (1 pt) Express your answer to the previous question in terms of M and N which represents the number of IOs needed to sort Table X and Table Y respectively. Please do not insert spaces in between (example answer: "M+N+100").

55

Half credit was awarded for answers of M+N+55. In this case, M and N are both zero because you can sort in memory.

(d) Poll manager Alvin finds that Jerry is doing such a good job that he gives him the information for all adjacent counties as well. Jerry feels that Sort Merge Join might be too slow, so he decides to execute Grace Hash Join instead.

Now, Table X has 90 pages and Table Y has 100 pages. Assume we have 11 buffers. You may assume a perfectly uniform hash function in all passes of Grace Hash Join and no key skew.

i. (1 pt) After the first pass of Grace Hash Join, how many partitions of X have been created?

10

We partition pages into B-1 partitions.

ii. (1 pt) After the first pass of Grace Hash Join, how many pages are in each partition of X?

9

We partition pages into B-1 partitions. There are 90 pages of X, so each partition is 9 pages assuming a uniform hash function and no key skew.

iii. (1 pt) How many IOs will this partitioning of X take?

180

We need to read in 90 pages and write out 10 partitions of 9 pages for a total of 180 IOs.

iv. (1 pt) After the first pass of Grace Hash Join, how many partitions of Y have been created?

10

We partition pages into B-1 partitions.

v. (1 pt) After the first pass of Grace Hash Join, how many pages are in each partition of Y?

10

We partition pages into B-1 partitions. There are 100 pages of Y, so each partition is 10 pages assuming a uniform hash function and no key skew.

vi. (1 pt) How many IOs will this partitioning of Y take?

200

We need to read in 100 pages and write out 10 partitions of 10 pages for a total of 200 IOs.

vii. (1 pt) Is recursive partitioning necessary? If yes, which table will be repartitioned?

No

For recursive partitioning to be necessary, only ONE of the two tables needs to have a number of partitions that is <= B-2. In this case, Table A satisfies this with 9 partitions.

viii. (2 pt) After executing the rest of the Grace Hash Join (including recursive partitioning, if necessary) how many IOs in total will Grace Hash Join take?

570

We have 200+180 IOs from making our partitions. Now we need to perform naive hash join on each pair of partitions. Each pair will take 9+10 partitions and we have 10 such pairs. Thus, this takes 10*(9+10) = 190 IOs for a total of 200+180+190 = 570 IOs.

2. (21 points) Query Optimization

For this question, let's look at tables R, S, and T with the following properties. Assume that each column's values are uniformly distributed, and that all of the column distributions are independent of each other.

Schema	Pages	Indexes	Table Stats
CREATE TABLE **R**(a INTEGER, b FLOAT, c INTEGER)	50 pages with 100 records each	**R.b:** Alt 1 index with h = 3 and 50 leaf pages and **R.c:** Alt 3, clustered index with h = 2 and 20 leaf pages	**R.b:** min = 0, max = 20, 75 unique values and **R.c:** min = 10, max = 50, 20 unique values
CREATE TABLE **S**(a INTEGER, b FLOAT, c INTEGER)	100 pages with 100 records each	**S.b:** Alt 2, clustered index with h = 2 and 10 leaf pages	**S.b:** min = 1, max = 50, 50 unique values
CREATE TABLE **T**(a INTEGER, b FLOAT, c INTEGER)	25 pages with 100 records each	**T.c:** Alt 2, unclustered index with h = 2 and 10 leaf pages	**T.c:** min = 1, max = 100, 100 unique values

(a) (4 points) Selectivity Estimation

Estimate the **number of pages of output** each of the following queries will produce.

i. (1 pt) `SELECT * FROM R WHERE R.b <= 10`

25

- Sel(R.b <= 10) = (10-0)/(20-0) = 1/2
- Number of records in output = 1/2 * 50 * 100 = 2500 tuples since R has 50 pages with 100 records each
- Number of pages in output = 2500/100 = 25 pages since each page will be able to hold 100 records

ii. (1 pt) `SELECT * FROM T WHERE T.c <= 20`

5

- Sel(T.c <= 20) = (20-1)/(100-1+1) + 1/100 = 20/100 = 1/5
- Number of tuples in output = 1/5 * 100 * 100 = 500 tuples since T has 25 pages with 100 records each
- Number of pages in output = 500/100 = 5 pages since each page will be able to hold 100 records

iii. (2 pt) `SELECT R.a, S.b, S.c  FROM R INNER JOIN S on R.a = S.a WHERE R.b <= 10`

25000

We assume that all of the columns are independent of each other, so: Sel(R.b <= 10 and R.a = S.a) = Sel(R.b <= 10) * Sel(R.a = S.a)

From 1.1.1, we know that Sel(R.b <= 10) = 1/2. Also, Sel(R.a = S.a) = 1/10 since we don't have any information about the R.a or S.a columns.

Thus, Sel(R.b <= 10 and R.a = S.a) = 1/2 * 1/10 = 1/20

Number of tuples in output = 1/20 * (50 * 100) * (100 *100) = 2,500,000 tuples

Number of pages in output = 2,500,000 / 100 = 25,000 pages since each page will be able to hold 100 records

iv. (0 pt) Feel free to show your work for the previous question below for partial credit. You may leave this blank if you'd like.

(b) **(5 points) System R Query Optimizer - Pass 1**

For the next sections, we will optimize the following query using the System R (aka Selinger) query optimizer.

```
SELECT R.a, S.b, T.c
    FROM R INNER JOIN S ON R.a = S.a
           INNER JOIN T ON R.b = T.b
    WHERE R.b <= 10 AND T.c <= 20
    GROUP BY S.b
```

i. (3 pt) What is the estimated I/O cost of performing an index scan on T using the index on T.c?

504

I/O cost = 504 I/Os Breakdown:

- 2 I/Os (cost to read inner nodes on path from root to leftmost leaf)
- 1/5 * 10 I/Os (cost to read in leaf nodes)
- 1/5 * 25 * 100 I/Os (cost to read in records on data pages: use 1 I/O per matching record because index is unclustered)

ii. (0 pt) Feel free to show your work for the previous question below for partial credit. You may leave this blank if you'd like.

iii. (2 pt) Ignore your answer for the previous problem and **assume the cost of an index scan on T.c is 30 I/Os** (as indicated in the table below). Assume that we have the following I/O costs for accessing single tables:

Option	Access Plan	I/O Cost
a	R: Full scan	50 I/Os
b	R: Index scan on R.b	28 I/Os
c	R: Index scan on R.c	72 I/Os
d	S: Full scan	100 I/Os
e	S: Index scan on S.b	102 I/Os
f	T: Full scan	25 I/Os
g	T: Index scan on T.c	30 I/Os

Which of the following plans will be kept at the end of Pass 1?

☐ Option a

■ Option b

☐ Option c

■ Option d

■ Option e

■ Option f

☐ Option g

☐ None of the above

- Option b: min cost to access table R + interesting order (used in downstream joins with T)
- Option d: min cost to access table S
- Option e: interesting order (used in GROUP BY)
- Option f: min cost to access table T

(c) (12 points) System R Query Optimizer - Pass 2/3

Let's now look at passes 2 and 3 of the System R Query Optimizer. Assume that:

- We have **B = 5** buffer pages.
- Your earlier answer to the number of pages of R after applying the `R.b <= 10` predicate is **10 pages**
- Your answer to the number of pages of T after applying the `T.c <= 20` predicate is **20 pages**

For your convenience, here is a copy of the query we are optimizing:

```
SELECT R.a, S.b, T.c
    FROM R INNER JOIN S ON R.a = S.a
           INNER JOIN T ON R.b = T.b
    WHERE R.b <= 10 AND T.c <= 20
    GROUP BY S.b
```

i. (3 pt) What is the estimated I/O cost of R BNLJ S? Use the full scan on R and the full scan on S as the single table access plans for R and S (as given in the table for Pass 1).

450

50 + ceil(10/5-2) * 100 = 450 I/Os

ii. (0 pt) Feel free to show your work for the previous question below for partial credit. You may leave this blank if you'd like.

iii. (1 pt) Will R BNLJ S produce an interesting order?

○ Yes

[✓ No]

BNLJ never produces any interesting orders.

iv. (3 pt) Jerry is trying to estimate the I/O cost for R SMJ T. He thinks it's 300 I/Os (the cost of sorting R) + 150 I/Os (the cost of sorting T) + 75 I/Os (cost of merging) = 525 I/Os.

However, Jennifer disagrees and thinks we can compute R SMJ T in fewer I/Os.

List 2 optimizations we can make that will lower the I/O cost of R SMJ T.

Solutions that listed any 2 of the following optimizations received full credit:

A. Access R using index scan on R.b so that we do not have to run external sorting on R

B. Apply selection for R.b <= 10 on the fly

C. Apply the SMJ optimization

v. (1 pt) Will R SMJ T produce an interesting order?

○ Yes

[✓ No]

The output of R SMJ T will be sorted on the b column of R and T, but orderings involving R.b or T.b will be useful in any downstream joins or in the GROUP BY/ORDER BY clauses.

vi. (2 pt) Assume that we have the following I/O costs for joining 2 tables together:

Option	Access Plan	Sorted on	I/O Cost
a	$R \bowtie S$	N/A	800 I/Os
b	$S \bowtie R$	S.b	750 I/Os
c	$S \bowtie T$	N/A	700 I/Os
d	$T \bowtie S$	S.b	650 I/Os
e	$R \bowtie T$	N/A	150 I/Os
f	$R \bowtie T$	R.b	160 I/Os
g	$T \bowtie R$	T.c	170 I/Os

Using just this information, along with the query, which of the following plans will be kept at the end of Pass 2?

☐ Option a

[✓ Option b]

☐ Option c

☐ Option d

[✓ Option e]

☐ Option f

☐ Option g

☐ None of the above

- Option b: min cost for joining together R and S + interesting order (GROUP BY S.b)
- Option e: min cost for joining R and T

vii. (2 pt) Assume that the access plans retained at the end of Pass 2 were S ⋈ R, S ⋈ T, and R ⋈ T. Which of the following access plans will be considered in Pass 3 of the System R query optimizer?

☐ (R ⋈ S) ⋈ T

[✓] (S ⋈ R) ⋈ T

[✓] (R ⋈ T) ⋈ S

☐ T ⋈ (S ⋈ R)

☐ R ⋈ (S ⋈ T)

☐ S ⋈ (R ⋈ T)

☐ None of the above

- (R ⋈ S) ⋈ T is not considered because R ⋈ S was not retained at the end of Pass 2
- T ⋈ (S ⋈ R) is not considered because this plan is not left deep
- R ⋈ (S ⋈ T) is not considered because this plan is not left deep
- S ⋈ (R ⋈ T) is not considered because this plan is not left deep

3. (20 points) Concurrency or Procurrency

(a) Tripping (up) on ACID

For the following three questions, select **ALL** ACID properties that could be violated by the database, if any.

i. (2 pt) You have a special database that currently has one table `students` with a primary key `sid`. You run three transactions and the resulting `students` table (after the transactions commit) consists of every row in the previous table duplicated three times. You decide you don't like this new table and restart your computer. Luckily, when you reboot the database, the `students` table has returned to the way it was (only one copy of each row).

- ☐ Atomicity
- ■ Consistency
- ☐ Isolation
- ■ Durability
- ☐ None

Duplicating a primary key is a clear violation of PK consistency. This database also isn't durable because the committed changes are not saved even in the event of a hard reset.

ii. (2 pt) You have a special database that commits all active transactions prematurely and flushes their updates as soon as it detects any deadlock. The database uses strict 2PL.

- ■ Atomicity
- ■ Consistency
- ■ Isolation
- ☐ Durability
- ☐ None

Atomicity is defined as "all actions in the transaction should happen or none should." In this example, some actions may commit while the rest won't (if a deadlock occurs mid-schedule) and this violates atomicity.

The DB could be an inconsistent state at the time of deadlock. For example, in a bank transaction, if A wants to send $100 to B and the deadlock occurs after A has $100 deducted from their bank account but before B has received it, the transaction would prematurely commit and consistency would be violated.

Further, If a transaction prematurely commits, it exposes some of its intermediate operations to other transactions, which means other transactions could read those changes, so they're no longer isolated.

iii. (2 pt) Instead of conflict serializability, your database only accepts schedules that are write-write equivalent to some serial schedule. Two schedules are write-write equivalent if they involve the same actions of the transactions in the same order, and every pair of writes across two different transactions on the same resource is ordered the same way.

☐ Atomicity

☐ Consistency

[✓ Isolation]

☐ Durability

☐ None

This type of equivalency does not account for the case of conflicting write-reads: one transaction attempts to write to a resource while another attempts to read to it. If one transaction reads some stale data before a write is committed, the isolation barrier would be broken. Notice how conflict serializability accounts for this case while "write-write" serializability doesn't.

(b) Don't Bank on It

Tara is about to participate in a 3 way transfer at her bank. She is supposed to receive $50 each from her friends Utne and Valdo. Though she is only supposed to receive $100, she knows that the bank transactions are **not atomic** and she knows the ordering of operations ahead of time. Therefore, she knows that she can possibly make a profit here. The transfer is done in **one transaction** as follows (T is Tara's bank account, U is Utne's, and V is Valdo's). Keep in mind that R stands for read and W stands for write.

Timestamp:	1	2	3	4	5	6	7	8
T1	R(U)	R(V)	U=U-100	W(U)	R(T)	V=V-100	T=T+200	W(V)

Timestamp:	9	10	11	12	13	14	15	16
T1	T=T-50	V=V+50	W(V)	W(T)	U=U+50	W(U)	T=T-50	W(T)

i. (2 pt) What is the maximum profit Tara could make? (don't include the dollar sign)

150

We know the transaction is non-atomic. If the transaction commits anywhere between timestamps 12 and 15 inclusive, Tara's bank account will see $150 dollars added to it.

ii. (2 pt) Select all possible values that Utne can see in his bank account after the transfer is done.

- [✓ U]
- [✓ U - 50]
- [✓ U - 100]
- ☐ U - 150
- ☐ U - 200
- ☐ None of the above

Utne will either see U (transaction commits between timestamps 0-3 inclusive), U - 100 (timestamps 4-13 inclusive), or U-50 (14 onwards).

iii. (1 pt) Assume that the bank upgrades their infrastructure and now separates the transfer into three separate transactions. This time we have a four way transfer between four bank accounts (T, U, V, and A). Once again, R stands for read and W stands for write. The schedule is as follows:

	1	2	3	4	5	6	7	8	9	10
T1	R(T)				R(V)			R(A)		W(U)
T2			W(V)				R(T)		R(A)	
T3		R(U)		W(A)		W(T)				

Is this schedule conflict serializable?

○ Yes

[✓ No]

The dependency graph contains a cycle between T1, T2, and T3 (along with a cycle between T1 and T3).

iv. (2 pt) If you marked no in the previous answer, please provide the smallest set of timestamps of the operations which, once removed, will make this schedule conflict serializable.

If you marked yes, write 0.

For example, if you believe the minimum number of operations you need to remove to make the schedule conflict serializable is 3 and removing T1's R(T) and W(U) and T3's W(A) will satisfy this constraint, provide as your answer 1,4,10.

1 or 6

Removing either of these operations individually will result in a graph with edges consisting of T3 pointing to T2, T3 pointing to T1, and T2 pointing to T1. Note that this graph is acyclic.

(c) Pop, Lock, and Drop It (with a heavy emphasis on lock it)

Refer to the following schedule of transactions for the next 3 questions. These transactions use shared and exclusive locks on the resources A, B, C, and D. **Assume our lock manager's wait queue policy is always first-in-first-out**.

	1	2	3	4	5	6	7	8	9	10
T1	S(C)									X(A)
T2		S(B)						X(B)		
T3			S(A)		X(C)				X(D)	
T4				S(A)		S(C)	X(C)			

i. (2 pt) Directly after timestep 6 but before timestep 7, what transactions are in the wait queue for resource C?

☐ T1

☐ T2

[✓ T3]

[✓ T4]

☐ None of the above

Transaction 1 acquires Resource C at Timestamp 1. Transaction 3 attempts to acquire an X lock in timestamp 5 but must wait in the wait queue. In timestamp 6, Transaction 4 attempts to acquire an S lock. Even though the granted set consists entirely of S locks; there is already a request in the wait queue and therefore, we must append Transaction 4 to the waiting queue.

ii. (1 pt) You are contemplating between using wound-wait and wait-die for deadlock avoidance for this schedule. You would like to maximize the number of transactions aborted. Both wound-wait and wait-die are ordered on lock acquisition. This means that a transaction held the lock first gets priority. Assume aborted transactions don't restart until long after timestamp 10. Also assume that waiting transactions do not try to acquire other locks. Between wound-wait and wait-die, which strategy will result in the most number of aborted transactions?

[✓ Wait-die]

○ Wound-wait

○ They both result in the same number.

Wait-die results in 2 aborted transactions (T3 at t=5 because it has lower priority than T1 for resource C and T4 at t=7 because it has lower priority than T1 for resource C). Wound-wait results in 0 transactions aborted due to our priority scheme: all contested lock operations are initiated by the transaction that requested the lock second so they must patiently wait, but they are not aborted.

iii. (2 pt) Which transactions are aborted in the maximum case?

For example, if you answered wait-die previously and if you believe wait-die aborts 3 transactions; T1, T2, and T4; your answer would be T1,T2,T4 or any permutation thereof. If you answered 'They both would result in the same number,' provide the transactions that would be aborted by Wait-die.

T3,T4

Refer to the previous solution.

(d) Project SIX

In this subpart, we will explore 2 potential transactions and concurrency design choices.

i. (2 pt) In Project 4, upgrading a lock has priority over other queued requests. Your friend believes this is unfair. She proposes that the currently held lock is released and the request for the upgraded lock is added to the queue. To put it more concretely, let's assume T1 holds an S lock on resource A and T2 is waiting for an X lock on resource A. If T1 then requests an upgrade from S to X for the held lock, then, in normal circumstances, T1's upgrade would have priority and T1 would immediately be granted the X lock on A. However, your friend proposes that the lock is instead released, the new request gets added to the wait queue, and T2 receives their X lock on A. Your friend believes this will promote fairness across transaction requests so transactions won't get stalled for too long. Why might this not be a good idea? Answer in 1-2 sentences.

Any answer that mentions 2PL will get credit. With your friend's strategy, transactions can potentially acquire locks, release them, and then acquire them again, clearly violating 2PL.

ii. (2 pt) We know from lecture that a schedule is conflict serializable if and only if its dependency graph is acyclic. Your friend suggests the following process to identify the conflict equivalent serial schedule for a schedule with an **acyclic** dependency graph: find a node with only outgoing edges, then perform depth-first search on it. He states that this will always return one such conflict equivalent serial schedule. Is your friend right? If yes, explain why. If not, propose another way of finding a conflict equivalent serial schedule. Answer in 1-2 sentences.

Depth-first search seems like a good approach until you realize that there may be multiple nodes with exclusively outgoing edges. In this case, these nodes will never be included in the serial schedule. Therefore, an approach that works similarly but includes all nodes is performing a topological sort on the nodes.

4. (14 points) Logging and Recovery

(a) (4 points) True or False

Choose True or False for each of the statements below.

i. (1 pt) Undo logging relies on flushing changes made by each transaction to the disk before it commits.

[✓ True]

False

In undo logging we write a flush record to the log before commit. Since it is a force-based recovery algorithm the disk changes from flush must have been persisted before the commit record is written to the log.

ii. (1 pt) Redo logging relies on flushing changes made by each transaction to the disk before it commits.

True

[✓ False]

In redo logging we write a commit record to the log before flush. Therefore the disk changes might not have persisted to the disk yet by the time the commit record is written to the log.

iii. (1 pt) For redo logging, we only need to store the new value and not the previous value in each update record in the log.

[✓ True]

False

Redo logging re-applies changes to the pages. Therefore we only need the new value that is going to be written.

iv. (1 pt) Write-ahead logging means updates to in-memory pages are written to the disk before the corresponding log record is written to the disk.

True

[✓ False]

Write-ahead logging means the log record must be written before the corresponding data updates are made.

(b) (10 points) Recovery

Consider the recovery log below:

```
1:  <START T1>
2:  <T1 X 5>
3:  <START T2>
4:  <T1 Y 8>
5:  <T2 X 9>
6:  <START T3>
7:  <T3 Z 112>
8:  <COMMIT T1>
9: <T2 X 13>
10: <ABORT T2>
11: <T3 Y 17>
[DB CRASHED]
```

i. (3 pt) Under write-ahead logging, and assuming that the recovery manager is aware of the status of all transactions before it starts recovery (e.g., by storing a separate "transaction status table" that flushed to the disk each time when the log is flushed).

Under UNDO logging, what is the earliest line the recovery manager must read to ensure full recovery of the database? Write down the line number below. No explanation needed. For instance, answering N means the recovery manager must read from the end of the log up to and including line N for full recovery.

6

T3 is the transaction that we need to unroll since it didn't complete when the DB crashed. Therefore, during recovery, we won't need to read earlier than line 6 in the log where T3 started.

ii. (3 pt) With the same assumptions as above, under REDO logging, what is the latest line the recovery manager must read to ensure full recovery of the database? Write down the line number below. No explanation needed. For instance, answering N means the recovery manager must read from the beginning of the log up to and including line N for full recovery.

8

T1 is the only transaction that committed. Therefore, during recovery, we only need to reapply the changes made by T1 and won't need to read past line 8 in the log where T1 committed.

iii. (2 pt) Under UNDO logging, what will be the value of X after recovery? Write down the answer below. No explanation needed.

9

Recall that each "write" entry in the undo log stores the previous value that was overwritten. X was written by T1 and T2, but since T2 did not commit X will hold the last value written by T1. And line 5 tells us that T1 wrote 9 to X.

iv. (2 pt) Under REDO logging, what will be the value of X after recovery? Write down the answer below. No explanation needed.

5

Recall that each "write" entry in the redo log stores the new value that was written. X was written by T1 and T2, but since T2 did not commit X will hold the last value written by T1. And line 2 tells us that T1 wrote 5 to X.

5. (20 points) The Electoral College of Database Design

We have the following tables in our database:

`Candidates`**(id, name, homeStateID, partyID, age)**

`States`**(sid, name, population, prevWinnerID, prevMargin)**

`Counties`**(cid, sid, name, population, prevWinnerID, prevMargin)**

`Polls`**(pid, sid, cid, projectedMargin, projectedWinnerID)**

For the entire question, you may assume the following:

- External sorting will always take exactly 2 passes.
- The SMJ optimization is not used.
- External hashing will never involve recursive partitioning.
- For external hashing, the number of pages written will always equal the number of pages read.
- The size of each relation, **including joined relations**, is at least 100 times the number of buffer pages.
- There are no indexes on any column.
- The database uses the **System R** query optimizer.

(a) Joins

Hint: For the next 6 questions, assume selections and projections do not change the size of the relation and do not worry about the selectivity of the join(s).

i. (1 pt) Which of the following joins will the query optimizer use to execute **Query A**?

Query A

```
SELECT C.name, C.partyID, S.name, S.prevMargin
FROM Candidates AS C INNER JOIN States AS S
ON C.homeStateID != S.sid
WHERE S.population > 5000000 AND S.prevMargin < 5;
```

○ Simple Nested Loop Join

○ Page Nested Loop Join

[✓ Block Nested Loop Join]

○ Index Nested Loop Join

○ Sort Merge Join

○ Grace Hash Join

ii. (1 pt) Explain your answer for **Query A** in 2-4 sentences.

Query A is not performing an equijoin so we cannot use Grace Hash Join or Sort Merge Join. There are no indexes so we cannot use Index Nested Loop Join either. This leaves the rest of the nested loop joins and since BNLJ is always the cheapest nested loop join, that is the correct answer.

iii. (1 pt) Which of the following joins will the query optimizer use to execute **Query B**?

Query B

```
SELECT S.sid, MAX(P.projectedMargin), MIN(P.projectedMargin)
FROM Polls AS P INNER JOIN States AS S
ON P.sid = S.sid
INNER JOIN Counties AS C
ON S.sid = C.sid
WHERE S.name LIKE 'A%' AND P.cid IS NOT NULL
GROUP BY S.sid;
```

- ○ Simple Nested Loop Join
- ○ Page Nested Loop Join
- ○ Block Nested Loop Join
- ○ Index Nested Loop Join
- ○ Sort Merge Join
- [✓ Grace Hash Join]

iv. (2 pt) Explain your answer for **Query B** in 2-6 sentences.

INLJ is eliminated, because there are no indices. SNLJ and PNLJ can be eliminated, because they will not outperform BNLJ. GHJ is cheaper than SMJ even though SMJ has an interesting order for the GROUP BY. BNLJ is also eliminated because the cost is orders of magnitude more expensive than GHJ (and SMJ), due to the ratio between the relation sizes and the number of buffer pages.

Keep reading below (at your own risk) for a more in-depth solution.

INLJ is eliminated since there are no indexes, and we can also eliminate all NLJ's except for BNLJ. Let's assume the selection and projection pushdowns do not change the size of the relations as the problem states. The problem also states the size of all relations (including joined relations) is at least

$$100 * B$$

. For the remainder of the problem we will approximate the size of each relation as

$$[X] = 100 * B$$

.

Grace Hash Join for an arbitrary pair of tables R and S, will cost

$$3 * ([R] + [S])$$

. This comes from the fact that we assume external hashing will never result in recursive partitioning, so GHJ will only involve reading each of the tables and writing them back to disk after partitioning them. This results in

$$2 * ([R] + S)$$

I/Os. The final pass over the partitions is during the probing phase of GHJ, and since we do not consider the write cost of joins it is not included.

Performing GHJ twice for both joins costs:

$$3 * ([X] + [X]) = 6 * [X]$$

for the first join and then

$$2 * [X] + 3 * [X]$$

for the second join, since the left relation input to the second join will be streamed in from the first join so the initial read cost can be avoided. We also hash the final joined relation to perform the

GROUP BY for a total cost of

$$6 * [X] + 5 * [X] + 3 * [X] = 14 * [X]$$

.

Similarly, Sort Merge Join for an arbitrary pair of tables R and S, will cost

$$5 * ([R] + [S])$$

, because it takes 2 passes for external sorting each table, which involves reading and writing the entire relation. The final cost is the merge phase of SMJ which costs

$$[R] + [S]$$

.

Performing SMJ twice will cost:

$$5 * ([X] + [X]) = 10 * [X]$$

for the first join and

$$5 * [X]$$

for the second join since the left relation input to the second join will be streamed in from the first join so the read cost can be avoided. Additionally, the output of the first join is already sorted so there is no need to sort it again. Since the output is already sorted, there is no additional cost for the GROUP BY, so the total cost would be

$$15 * [X]$$

.

Another combination one may consider is a GHJ for the first join and SMJ for the second join. This results in

$$6 * [X]$$

to perform the GHJ,

$$4 * [X] + 5 * [X]$$

to perform the SMJ for a total cost of

$$15 * [X]$$

.

The final join to consider is BNLJ. Since the cost of BNLJ is

$$[X] + \lceil \frac{[X]}{B - 2} \rceil * [X]$$

and we know that each table is of equal size and that each table is at least 100 times the size of B, the number of buffer pages, we can see that a lower bound for BNLJ is

$$[X] + 100 * [X]$$

. We already see that with only just one BNLJ, the cost is greater than performing GHJ for both joins.

v. (1 pt) Which of the following joins will the query optimizer use to execute **Query C**?

Query C

```
SELECT S.sid, AVERAGE(C.population)
FROM States AS S INNER JOIN Counties AS C
ON S.sid = C.sid
GROUP BY S.sid
ORDER BY AVERAGE(C.population);
```

○ Simple Nested Loop Join

○ Page Nested Loop Join

○ Block Nested Loop Join

○ Index Nested Loop Join

[✓ Sort Merge Join]

○ Grace Hash Join

vi. (2 pt) Explain your answer for **Query C** in 2-4 sentences.

Full credit was given for the previous question for either Grace Hash Join or Sort Merge Join.

Same as above, we can eliminate INLJ, SNLJ, and PNLJ. BNLJ can also be eliminated for costing an order of magnitude more I/Os than SMJ and GHJ. (The actual math can be seen in the previous solution.)

Again, let's assume that [X] = size of all relations.

Using SMJ will cost

$$5 * ([X] + [X])$$

to perform the SMJ and

$$3 * [X]$$

to sort the joined relation to perform the ORDER BY. This comes from the fact that the output of the SMJ will be streamed into the first pass of external sorting, so the initial read cost can be avoided.

Using GHJ will cost

$$3 * ([X] + [X])$$

to perform the GHJ and

$$3 * [X]$$

to hash the joined relation to perform the GROUP BY. This comes from the fact that the output of the GHJ will be streamed into the partitioning pass, so the initial read cost can be avoided. Finally, there is another cost of

$$4 * [X]$$

sort the joined relation to perform the ORDER BY.

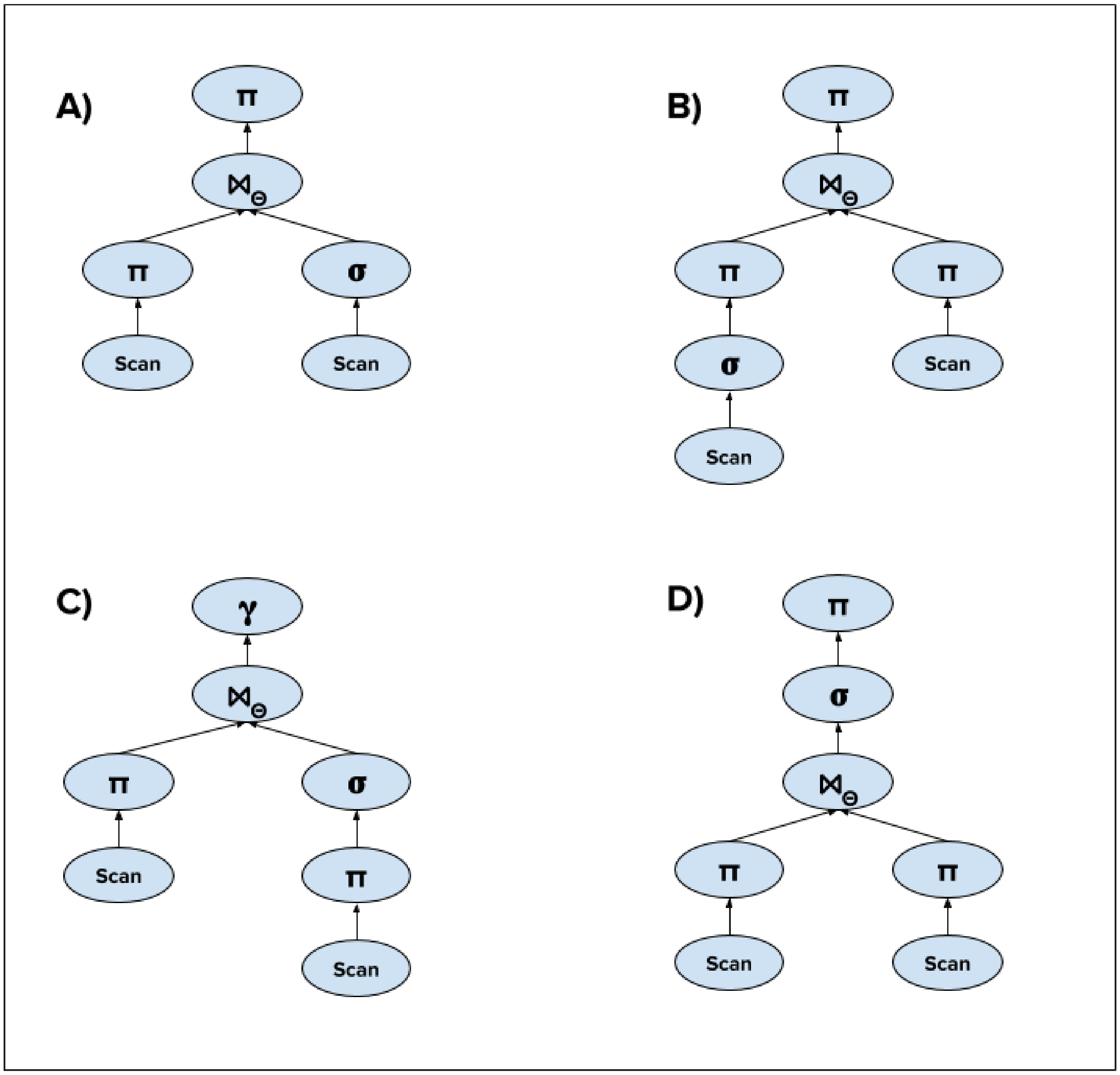
Both SMJ and GHJ will cost the same for this query.

(b) Query Plan

i. (1 pt) Which of the following query plans will System R choose in order to execute **Query A**?

Query A

```
SELECT C.name, C.partyID, S.name, S.prevMargin
FROM Candidates AS C INNER JOIN States AS S
ON C.homeStateID != S.sid
WHERE S.population > 5000000 AND S.prevMargin < 5;
```



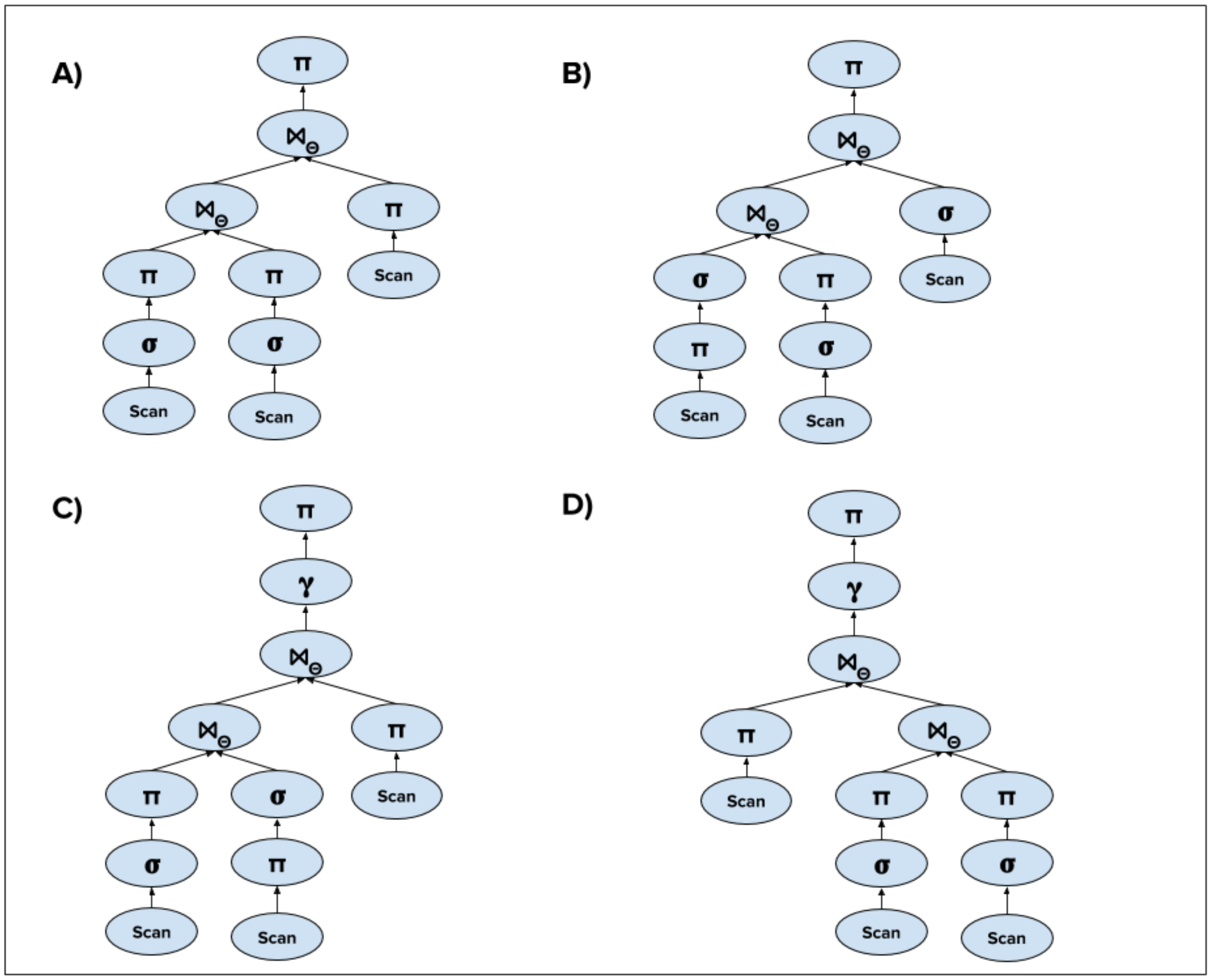
- ○ A
- ● B
- ○ C
- ○ D
- ○ None of the above

A - Wrong because both tables will have a projection applied before being passed to join. C - Wrong because missing final projection and there is no need for GROUP BY / AGG. D - Wrong because selection should be applied before the join.

ii. (1 pt) Which of the following query plans will System R choose in order to execute **Query B**?

Query B

```
SELECT S.sid, MAX(P.projectedMargin), MIN(P.projectedMargin)
FROM Polls AS P INNER JOIN States AS S
ON P.sid = S.sid
INNER JOIN Counties AS C
ON S.sid = C.sid
WHERE S.name LIKE 'A%' AND P.cid IS NOT NULL
GROUP BY S.sid;
```



◯ A

◯ B

● C

◯ D

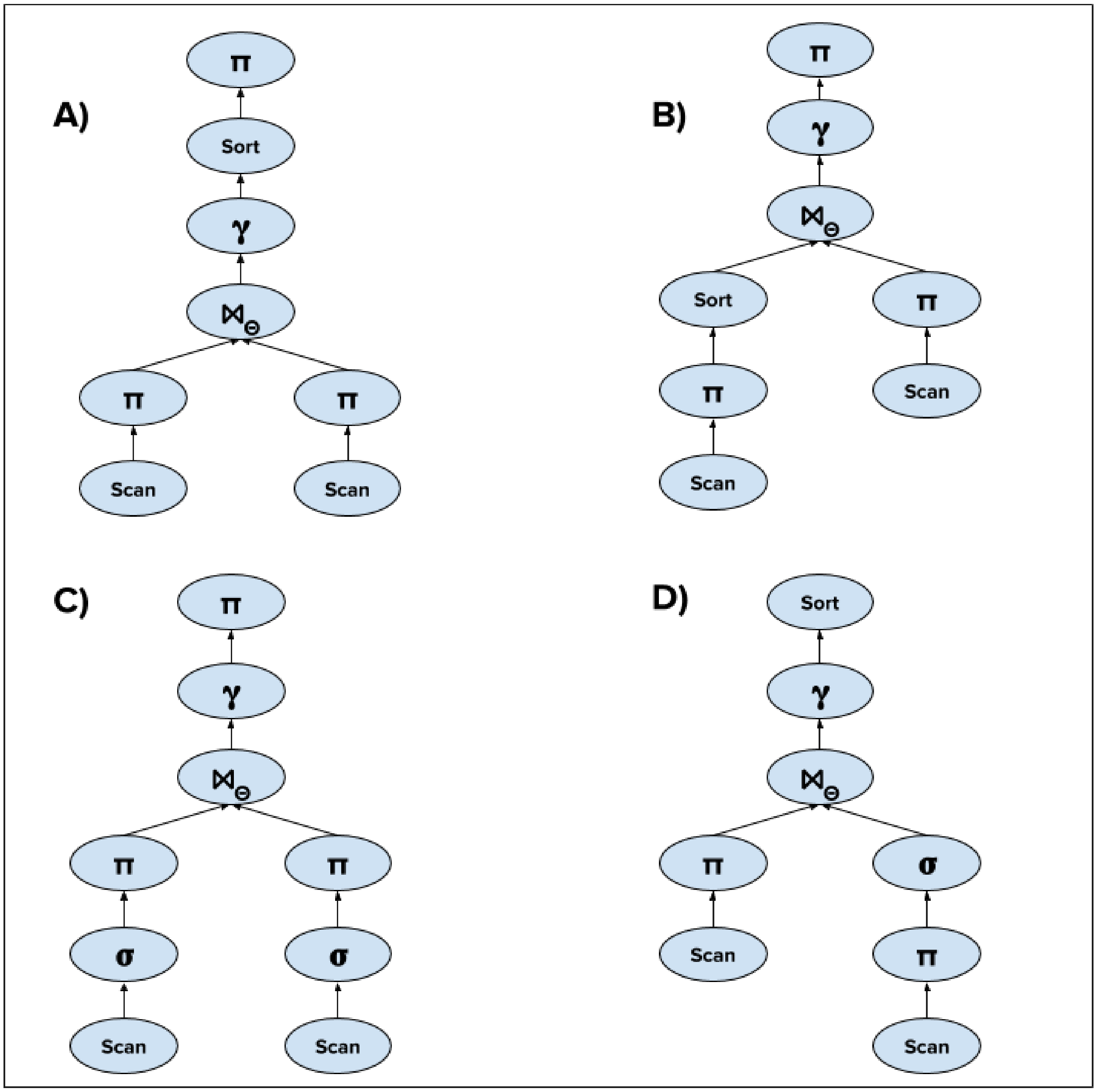
◯ None of the above

A - Wrong because missing GROUP BY operator. Even though the GHJ would save the cost of the GROUP BY, the MAX and MIN aggregates still need to be computed. B - Wrong. Same as A with projection replaced by a selection. D - Wrong because right deep.

iii. (1 pt) Which of the following query plans will System R choose in order to execute **Query C**?

Query C

```
SELECT S.sid, AVERAGE(C.population)
FROM States AS S INNER JOIN Counties AS C
ON S.sid = C.sid
GROUP BY S.sid
ORDER BY AVERAGE(C.population);
```



[✓ A]
B
C
D

◯ None of the above

B - Wrong because sorting has to happen after the GROUP BY in order to ORDER BY AVERAGE(*). C - Wrong because unnecessary selections and does not have a sort operator. D - Wrong because unnecessary selection for the right table and does not have a final projection.

(c) Concurrency

For the next 3 problems, consider the same 3 queries from above, which are copied below.

Query A

```
SELECT C.name, C.partyID, S.name, S.prevMargin
FROM Candidates AS C INNER JOIN States AS S
ON C.homeStateID != S.sid
WHERE S.population > 5000000 AND S.prevMargin < 5;
```

Query B

```
SELECT S.sid, MAX(P.projectedMargin), MIN(P.projectedMargin)
FROM Polls AS P INNER JOIN States AS S
ON P.sid = S.sid
INNER JOIN Counties AS C
ON S.sid = C.sid
WHERE S.name LIKE 'A%' AND P.cid IS NOT NULL
GROUP BY S.sid;
```

Query C

```
SELECT S.sid, AVERAGE(C.population)
FROM States AS S INNER JOIN Counties AS C
ON S.sid = C.sid
GROUP BY S.sid
ORDER BY AVERAGE(C.population);
```

i. (1 pt) Assume the **only** queries executed on the database are Queries A, B, and C. Queries are executed within a transaction, and each transaction consists of a **single** query. (i.e. Query A would be executed within its own transaction.)

Select **all** of the following that guarantee conflict serializability.

- [✓] 2PL
- [✓] Strict 2PL
- [✓] No locking
- ☐ None of the above

2PL - Yes Strict 2PL - Yes No locks - Yes, It is a read only workload so you don't need locks as there are never any conflicting operations.

ii. (1 pt) Now, consider adding **Query D** shown below. Assume the **only** queries executed on the database are Queries A, B, C, and D. Again, queries are executed within a transaction, and each transaction consists of a **single** query. (i.e. Query A would be executed within its own transaction.)

Query D

```
INSERT INTO Polls
VALUES (20, 5, 186, 4, 3);
```

Select **all** of the following that guarantee conflict serializability.

■ 2PL

■ Strict 2PL

☐ No locking

☐ None of the above

2PL - Yes Strict 2PL - Yes

There are now writes so we need locks. Any type of 2PL will guarantee conflict serializability.

iii. (1 pt) Assume the **only** queries executed on the database are Queries A, B, C, and D. Again, queries are executed within a transaction. However, there may be **multiple** queries (including repeats) within the same transaction. (i.e. Query A and B would be executed within the same transaction. Query A can also be executed twice within the same transaction.)

Select **all** of the following that guarantee conflict serializability.

☐ 2PL

☐ Strict 2PL

☐ No locking

■ None of the above

This is the definition of a Phantom Read, since we may have the same query executed multiple times within the same transaction and each time there could be different tuples in the result. 2PL cannot protect against this.

(d) Performance

For the next 4 questions, indicate how the specified performance metric changes if the given change is implemented and explain your answer.

i. (1 pt) Reduce the number of active transactions once thrashing is observed.

● Throughput Increase

○ Throughput Decrease

ii. (1 pt) Explain your previous answer in 2-3 sentences.

By definition thrashing happens when number of active transactions reaches some threshold causing throughput to go down due to an increasing level of lock contention. By reducing the number of active transactions, throughput increases again, because there will be less lock contention which reduces waiting and mitigates the effects of thrashing.

iii. (1 pt) Enforce view serializability instead of conflict serializability.

○ Throughput Increase

● Throughput Decrease

iv. (1 pt) Explain your previous answer in 2-3 sentences.

Conflict serializability is more efficient to enforce and that is why it is often chosen by many databases.

Special case: Points were also awarded for arguing that throughput will increase, because view serializability allows for more schedules than conflict serializable schedules. Points were only given for answers that mentioned that conflict serializability is a subset of view serializability and explicitly talked about more serializable schedules being allowed to execute under view serializability.

v. (1 pt) Switch from Strict 2PL to 2PL, assuming there is never an edge in the dependency graph.

○ Latency Increase

● Latency Decrease

vi. **(1 pt)** Explain your previous answer in 2-3 sentences.

In 2PL locks are no longer held until the txn is ready to commit, thus there will be less waiting. Less waiting means that transactions will finish faster. The argument that 2PL may have to deal with cascading aborts and thus result in an increase in latency is not valid, because no edges in the conflict dependency graph means a transaction will never read uncommitted data. Therefore, there will be no cascading aborts.

vii. **(1 pt)** Switch from Undo Logging to Redo Logging, given an infinite buffer size.

○ Latency Increase

[✓ Latency Decrease]

viii. **(1 pt)** Explain your previous answer in 2-3 sentences.

Redo Logging does No-Steal / No-Force and Undo Logging does Steal / Force. Redo Logging's No-Steal policy is no longer a performance bottleneck since the buffer manager has an unlimited number of buffer pages. Undo Logging's Force policy leads to greater latency due to disk IO cost for flushing data pages than Redo Logging's No-Force policy, so the overall latency will decrease.

No more questions.