CSE-4411(A) Test #2

Sur / Last Name:
Given / First Name:
Student ID:

- **Instructor:** Parke Godfrey
- **Exam Duration:** 75 minutes
- **Term:** Fall 2005

The exam is open-book and open-notes. Answer the following questions to the best of your knowledge. Your answers may be brief, but be precise and be careful. Make clear any assumptions that you need to make along with your answers, whenever necessary.

There are four major questions. Points for each question and sub-question are as indicated. In total, the exam is out of 50 points.

If you need additional space for an answer, just indicate clearly where you are continuing.

Regrade Policy

- You should not write anything on the exam paper after the exam period, if it is to be considered for regrading. Write any subsequent notes on separate paper.
- Regrading should only be requested in writing. Write what you would like to be reconsidered. Note, however, that an exam accepted for regrading will be reviewed and regraded in entirety (all questions).

Grading Box	
1.	/10
2.	/15
3.	/15
4.	/10
Total	/50

1. (10 points) **Join Algorithms.** *A sinful join.* [analysis]

   Your new boss, Dr. Datta Bas, has invented a new join algorithm which is a hybrid of general sort-merge join (MJ)—so not the two-pass version—and index-nested-loop (INL) join. He has nick-named it SIN (sort-index-nested) join. For SIN join, the outer "table" (stream) should be sorted first on the join key. Then the inner table is probed via an index as is done in the INL join.

   a. (5 points) Under what circumstances—sizes of the outer and inner, the number of distinct values over the join column(s), the type of probe index, etc.—if any, would Dr. Bas's SIN join be more efficient than regular INL join?

      Either explain *why* SIN join is *not* more efficient than INL join under any circumstances, or explain *how* it is more efficient under certain circumstances.

Call a set of records that share the same value(s) on the join column(s) a group. *When there are large groups in the outer stream that match records in the inner table, SIN is better than INL.*
An example scenario of this is if we were joining over a foreign key with the "child" table as the outer and the "parent" table as the inner. Many outer records can match one inner record.
INL must probe for each *record of the outer stream. SIN could buffer the probe matches for an outer record and reuse them when the next outer record has the same join value. Thus, SIN effectively probes per outer* group *rather than per outer record. So SIN could reduce greatly the number of probes, hence the number of I/O's. The improvement is even more pronounced if the probe index is unclustered.*
For INL, a group is scattered across the stream, so each probe for each record of the group is likely to have to fetch the pages into the buffer pool again, even though these are the same pages in each case. Note that if we just sorted the outer (as in SIN) but did not otherwise "re-implement" INL at all (such as to buffer *the probe results), this would likely still be quite advantageous. The results of the previous probe would likely still be in the buffer pool at the time of the next probe (assuming a LRU replacement strategy). So, for each outer group, we likely only pay I/O's for the first probe.*
Some folks observed that SIN could help reduce the cost of a query plan overall (compared with using INL instead) since it could provide an "interesting order". This is a worthwhile observation. However, this by itself does not answer the question asked, to explain how it *(SIN)* is more efficient *(than INL)* under certain circumstances.

b. (5 points) Under what circumstances, if any, would Dr. Bas's SIN join be more efficient than general sort-merge join (MJ)?

Either explain *why* SIN join is *not* more efficient than MJ under any circumstances, or explain *how* it is more efficient under certain circumstances.

INL typically will be better than MJ only if the outer probes the inner relatively little. The sum of the number of pages of the inner hit by probes should be far less than the size of the inner in total. Otherwise, MJ will be more efficient. (MJ has no probe I/O's.)

SIN would be more efficient than MJ if few of the outer probes have matches in the inner, and if there are relatively few probes. In the discussion for Question 1a, we noted SIN would work better than INL if there were large groups in the outer. This effectively reduces the number of probes. So SIN might beat MJ times when INL would not if there are big groups in the outer,

An example scenario of this is if we were joining over a foreign key with the "child" table as the outer and the "parent" table as the inner, and furthermore, few parent (inner) records match any child (outer) records. This is a fairly unusual situation.

2. (15 points) **General Optimization.** *Plan your plan.* [multiple choice]

   For each of the following, choose *one* best answer.

---

a. *Pipelining* in query plans

A. reduces CPU overhead.
[✓ **B.**] eliminates the need to write temporary results to disk.
C. is a type of access path.
D. can never reduce the cost of executing the query, so is avoided whenever possible.
E. is typically used with the inner stream of a join.

---

b. Which of the following is *true* about the join algorithms?

A. The larger of the two tables should always be designated the outer table.
B. The smaller of the two tables should always be designated the outer table.
C. In any situation (2-pass) hash join (HJ) could be used, it would also be *possible* to use 2-pass sort-merge join (SMJ).
[✓ **D.**] In any situation (2-pass) SMJ could be used, it would also be *possible* to use HJ.
E. If an appropriate index is available for an index-nested loops join (INL), then INL is guaranteed to be less expensive than SMJ or HJ.

---

c. The index-nested-loops join

A. can only use a *clustered* index to probe the inner table.
B. cannot be pipelined.
[✓ **C.**] probes the inner table for *each* record of the outer table / stream.
D. could be used for any join in a *bushy-tree* plan.
E. is generally more efficient when the outer table / stream is large and the inner table is small than vice-versa.

The following information is available on tables **Sailors** and **Reserves**.

- **Reserves**: 10,000 records
  - bid: 50 values (1..50)
- **Sailors**: 1000 records
  - level: 10 values (1..10)
  - fav_colour: 5 values
- **Boats**: 50 records
  - colour: 5 values

The primary key of **Sailors** is sid, of **Reserves** is sid + bid + day, and of **Boats** is bid. Table **Reserves** reserves has a foreign key on sid referencing **Sailors** (on sid), and a foreign key on bid referencing **Boats** (on bid). All columns are *not null*.

For each of the following queries, estimate the selectivity of the query as the number of tuples it likely returns, as System R would.[1]

d.
```
select S.sid, R.day
    from Sailor S, Reserves R
    where S.sid = R.sid and
          R.bid = 7;
```

A. 2
B. 5
C. 20
D. 50
[✓ **E.**] 200
F. 500

e.
```
select S.sid, S.sname, R.bid, B.bname, R.day
    from Sailor S, Reserves R, Boats B
    where S.sid = R.sid and R.bid = B.bid and
          R.bid = 47 and S.level <= 7 and S.level > 3;
```

A. 1
B. 5
C. 20
[✓ **D.**] 80
E. 100
F. 10,000

f.
```
select S.sid, S.sname, B.bid, B.bname
    from Sailor S, Boats B
    where S.fav_colour = B.colour;
```

A. 5
B. 50
C. 1,000
D. 5,000
[✓ **E.**] 10,000
F. 50,000

[1] The estimation techniques discussed in Chapter 12 of the textbook are from System R.

g.
```
select part_no from Product
    where (price > 1000 and weight < 20)
        or (price > 5000 and quantity < 200);
```

There is an index on weight and on quantity. There is not an index on price.

A. Evaluation of the query above requires a table scan, so the indexes do not help.
B. An access path that employs just the index on weight is possible.
C. An access path that employs just the index on quantity is possible.
$\boxed{\textbf{D.}}$ No access path with just one index is possible, but an access path that employs both the indexes on weight and quantity is possible.
E. There is not enough information provided to answer this question.

---

h. Restricting focus to left-linear join trees is beneficial for all *except* which of the following reasons?

A. The inner "relation" for every join is a base table—or the evaluation of a single relation sub-query—so a more accurate size estimation of the output can be obtained.
B. The inner "relation" for every join is a base table—or the evaluation of a single relation sub-query—so an index-nested-loops join remains possible.
C. This helps prune the search search space of all possible join trees dramatically.
$\boxed{\textbf{D.}}$ For any query plan based on a join tree that is not left linear, there is guaranteed to be a query plan based on left-linear join tree that is less expensive.
E. Left-linear join trees typically enable pipelining along the outer "relations".

---

i. *Double buffering* is useful in the external sort algorithm because it

A. increases fan-in during merges so that fewer passes may be necessary.
B. results in initial runs that are twice as long (than if double buffering were not used).
$\boxed{\textbf{C.}}$ likely ensures that merge processing does not need to wait while pages are being read and written.
D. allows for sequential reads and writes.
E. speeds up I/O operations.

Consider table R with attributes A and B, table S with attributes B and C, and table T with attributes C and D. All joins below are natural joins; that is, the join columns are those columns named the same between the two tables.

j. Consider the following relational-algebra expressions:

I. $(\mathsf{R} \bowtie \mathsf{S}) \bowtie \mathsf{T}$
II. $(\mathsf{R} \bowtie \mathsf{T}) \bowtie \mathsf{S}$
III. $\mathsf{R} \bowtie (\mathsf{S} \bowtie \mathsf{T})$

Which of the above relational algebra expressions necessarily evaluate to the same result?

[✓ **A.**] All three must evaluate to the same result.
B. They each may evaluate to a different result.
C. **I** & **II**
D. **I** & **III**
E. **II** & **III**
F. Not enough information is provided to determine this.

k. Consider the following relational-algebra expressions:

I. $\pi_{\mathsf{A,D}}((\mathsf{R} \bowtie \mathsf{S}) \bowtie (\mathsf{S} \bowtie \mathsf{T}))$
II. $\pi_{\mathsf{A,D}}(\pi_{\mathsf{A,B,D}}(\mathsf{R} \bowtie \mathsf{S}) \bowtie \pi_{\mathsf{A,B,D}}(\mathsf{S} \bowtie \mathsf{T}))$
III. $\pi_{\mathsf{A,D}}(\pi_{\mathsf{A,C,D}}(\mathsf{R} \bowtie \mathsf{S}) \bowtie \pi_{\mathsf{A,C,D}}(\mathsf{S} \bowtie \mathsf{T}))$

Which of the above relational algebra expressions necessarily evaluate to the same result?

A. All three must evaluate to the same result.
[✓ **B.**] They each may evaluate to a different result.
C. **I** & **II**
D. **I** & **III**
E. **II** & **III**
F. Not enough information is provided to determine this.

*I also accepted **F** since the above R.A. in the original was ill-formed. (Column D was not projected in the inner projections in II and III, making for nonsensical expressions.) Assuming the correction (as made above), **B** is the answer.*

l. Consider the following relational-algebra expressions:

I. $\pi_{\mathsf{B}}(\mathsf{R} \bowtie (\mathsf{S} \bowtie \mathsf{T}))$
II. $\pi_{\mathsf{B}}((\pi_{\mathsf{B}}(\mathsf{R}) \bowtie \mathsf{S}) \bowtie \mathsf{T})$
III. $\pi_{\mathsf{B}}((\pi_{\mathsf{B}}(\mathsf{R} \bowtie \mathsf{S})) \bowtie \mathsf{T})$

Which of the above relational algebra expressions necessarily evaluate to the same result?

A. All three must evaluate to the same result.
B. They each may evaluate to a different result.
[✓ **C.**] **I** & **II**
D. **I** & **III**
E. **II** & **III**
F. Not enough information is provided to determine this.

The following questions are about System R, and how it works.

Consider a query that joins tables **R**, **S**, **T**, **U**, **V**, **X**, and **Y**. Assume any pair of tables (e.g., **S** & **V**) have a join condition between them.

---

m. How many left-join trees are possible?
- **A.** 1
- **B.** 7
- **C.** 49
- **D.** 128
- $\boxed{\textbf{E.}}$ 5,040

---

n. In phase 4 of join enumeration, (when query plans of four tables are generated), *at least* how many four-table plans are retained?
- **A.** 1
- **B.** 16
- **C.** 24
- $\boxed{\textbf{D.}}$ 35
- **E.** 2,401

*I also accepted **A** as Some may have read the question as* how many *plans* per *four-table selection are kept. whereas the question is intended to ask how many four-table plans (at least) are retained.*

---

o. In each phase of join enumeration, additional plans may be retained than just the plans with the best estimated cost. These additional plans are ones that
- **A.** are estimated to return *fewer* output records than the best plan is estimated to return.
- $\boxed{\textbf{B.}}$ generate some order on the output stream.
- **C.** use hash join, which is nice.
- **D.** each make it possible to eliminate one, or more, of the remaining joins.
- **E.** are randomly chosen.

(Scratch space.)

3. (15 points) **Joins.** *Join up free, today only!* [exercise / short answer]

Consider the query

```
select *
    from R, S
    where R.A = S.A;
      and S.D is between x and y
      and R.E = z;
```

The schema are **R**(<u>A</u>, B, E) and **S**(<u>C</u>, A, D). The underlined attributes designate the primary key. **S** has a foreign key cast on **R** through A, and **S**.A is not nullable.

The σ_{D} has a reduction factor of $\frac{1}{10}$. Likewise, the σ_{E} has a reduction factor of $\frac{1}{10}$. There are no indexes on **R**.E or **S**.D, however.

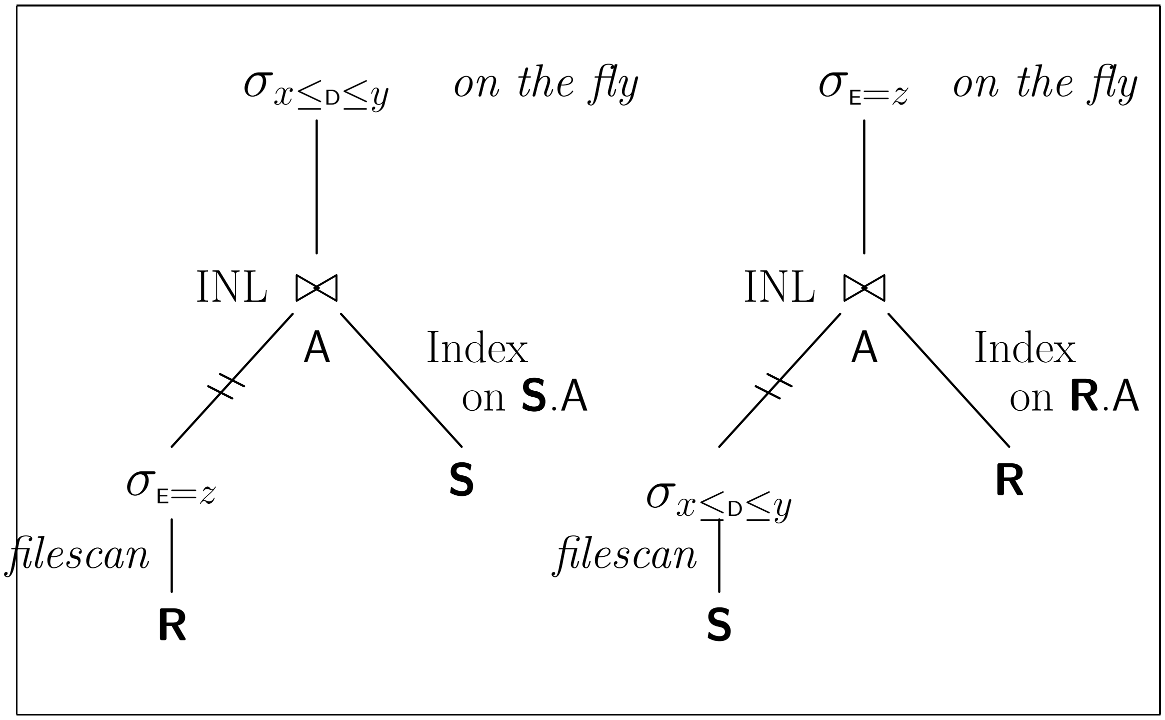
There is a clustered tree index on **R**.A and an unclustered tree index on **S**.A. Each index is of alternative #2 and has two layers of index pages. So the third layer in each case consists of the data-entry pages.

Let N_{T} generically denote the number of records in table **T**. Let P_{T} denote the number of pages of **T**. And let $V_{\mathsf{T.C}}$ denote the number of distinct values found in column C of table **T**.

$N_{\mathsf{R}} \ll N_{\mathsf{S}}$ and $P_{\mathsf{R}} \ll P_{\mathsf{S}}$. That is, the number of records in table **R** is *much* less than the number of records in table **S**; and likewise, the number of pages of table **R** is *much* less than the number of pages of table **S**.

For the following questions, push selections as far down in the tree as possible.

a. (4 points) Show the annotated query plans (trees) both for using an index nested loops join with **R** as the outer and for using an index nested loops join with **S** as the outer. Be complete. Do not leave any operators out.



b. (6 points) Which would be the better plan? The index nested loops (INL) join with **R** as the outer and using the unclustered index on **S**.A to probe, or the INL join with **S** as the outer and using the clustered index on **R**.A to probe?
Justify your claim. Use N_{R}, N_{S}, P_{R}, P_{R}, $V_{\mathsf{R.A}}$, etc. in your argument.

R *outer*

$$P_{\mathsf{R}} + \frac{1}{10}(3 + \frac{N_{\mathsf{S}}}{N_{\mathsf{R}}}) \cdot N_{\mathsf{R}} = P_{\mathsf{R}} + \frac{3N_{\mathsf{R}}}{10} + \frac{N_{\mathsf{S}}}{10}$$

S *outer*

$$P_{\mathsf{S}} + \frac{1}{10}(4N_{\mathsf{S}}) = P_{\mathsf{S}} + \frac{4N_{\mathsf{S}}}{10}$$

Note that an estimate of the number of matches per probe with **R** *as the outer is* $\frac{N_{\mathsf{S}}}{N_{\mathsf{R}}}$. *Every* **S** *record matches exactly one* **R** *record. So conversely, each* **R** *record would match around* $\frac{N_{\mathsf{S}}}{N_{\mathsf{R}}}$ **S** *records, on average, assuming uniformity. Note though, the uniformity assumption is not important here. There will be* N_{S} *I/O's spent in probing* in total, *one for each* **S** *record.*
Of course, the number of matches per probe with **S** *as the outer is exactly one.*

$$P_{\mathsf{R}} + \frac{3N_{\mathsf{R}}}{10} + \frac{N_{\mathsf{S}}}{10} \; ? \; P_{\mathsf{S}} + \frac{4N_{\mathsf{S}}}{10}$$
$$10P_{\mathsf{R}} + 3N_{\mathsf{R}} < 10P_{\mathsf{S}} + 3N_{\mathsf{S}}$$

So having **R** *as the outer is the winner, even though the probe index is unclustered.*
My apologies if the notation of this question—N_{T} *for number of records of* **T**, *and* P_{T} *for number of pages of* **T**—*messed anyone up too much. I did not intend to clash with the book's notation.*

c. (5 points) Is the better of the two INL plans likely to be better than a (two-pass) sort-merge join plan or a hash-join plan for this query? Explain.

The less expensive of the two plans in Question 3b (with **R** *as the outer) costs* $P_{\mathsf{R}} + \frac{3N_{\mathsf{R}}}{10} + \frac{N_{\mathsf{S}}}{10}$ *I/O's. This is quite bad, since it involves the* number of records *of* **S** *(and* **R***). If we have the buffer pool allocation for HJ or for SMJ, either would just cost* $3(P_{\mathsf{R}} + P_{\mathsf{S}})$. *So if seven or more* **R** *records pack per page (*$\frac{N_{\mathsf{R}}}{P_{\mathsf{R}}}$*)* and *thirty or more* **S** *records pack per page (*$\frac{N_{\mathsf{S}}}{P_{\mathsf{S}}}$*), which is quite likely, the HJ or SMJ would be better than the INJ join.*
If we were considering general merge join (MJ) here as well, we could note that there is a clustered index on **R**.*A, so* **R** *would not need to be sorted.*

4. (10 points) **Query Plans.** *Okay, here's our plan...* [exercise]

   The following information is available on tables **Purchase** and **Books**.

   - **Purchase**(cust#, book#, when, qnty)
     - 20,000,000 records on 200,000 pages
     - foreign keys:
       - on (book#) referencing **Books**
     - column cardinalities:
       - book#: 10,000 values
     - indexes:
       - unique clustered tree index on cust# + book# + when#
         depth of 3 index pages
         120 data entries per page
       - unclustered tree index on cust# + when + qnty
         depth of 3 index pages
         120 data entries per page
       - unclustered tree index on book# + when + qnty
         depth of 3 index pages
         120 data entries per page
   - **Books**(book#, title, author, year, publisher, price)
     - 10,000 records on 200 pages
     - column cardinalities:
       - author: 1,000 values
     - indexes:
       - unique clustered tree index on book#
         depth of 3 index pages
         200 data entries per page
       - unclustered tree index on title + author + year
         depth of 3 index pages
         100 data entries per page

   Underlined attributes in the schema (e.g., book#) indicate components of the primary key.

   Consider the following query.

```
select B.title, B.year, sum(P.qnty) * B.price as total
    from Purchase P, Books B
    where P.book# = B.book# and
          B.author = 'Agatha Christie'
    group by B.book#, B.title, B.year;
```

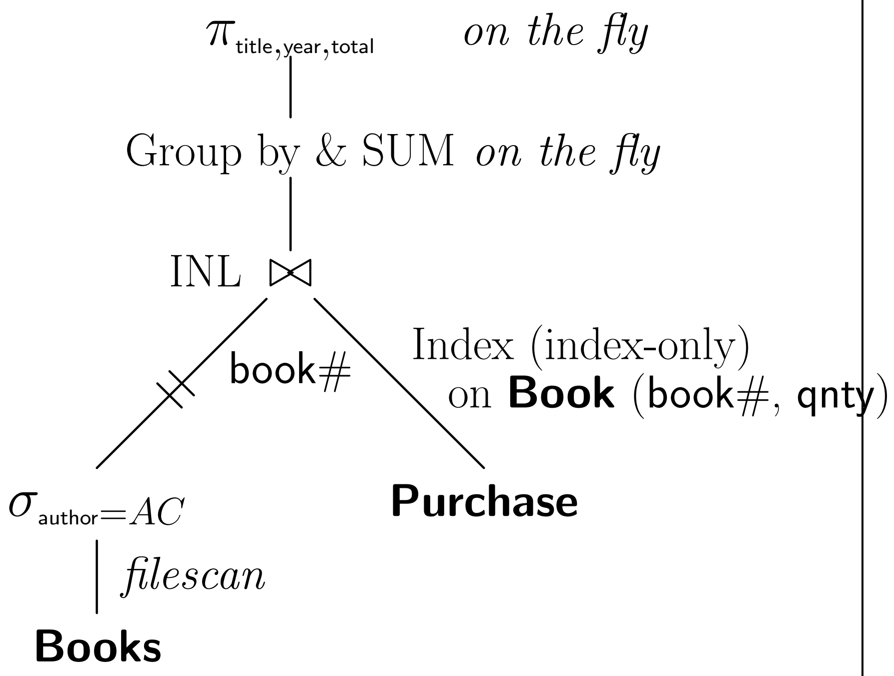
a. (2 points) Estimate the cardinality—the number of records returned—of the query *without* the group by and sum steps.

Also estimate the cardinality of the query itself (*with* the group by and sum steps).

without: $\frac{20{,}000{,}000}{1{,}000} = 20,000$
with: $\frac{10{,}000}{1{,}000} = 10$

b. (6 points) Assume that you have a quota of 20 buffer frames. Design a query plan for the query. Show the annotated query tree. Estimate the cost of your plan.

Part of the grading is based on how inexpensive your plan is. For full credit, your plan should cost less than 500 I/O's.


$\pi_{\text{title,year,total}}$ on the fly
Group by & SUM on the fly
INL $\bowtie$
book#
Index (index-only) on **Book** (book#, qnty)
$\sigma_{\text{author}=AC}$
filescan
Purchase
Books

Filescan of **Books***: 200 I/O's.*
Only 10 records survive the select author = AC*. Thus, the INL join does 10 probes. Each probe costs 3 I/O's to traverse the index pages. We estimate that each probe has 2,000 matches. Since we are index-only, though, we need the data entries (DE's) only. 120 DE's pack per DE page. This means each probe needs to read* $\lceil \frac{2000}{120} \rceil = 17$ *DE pages. So each probe costs 20 I/O's in all. In total, the probes cost 200 I/O's.*
The group-by and sum can be done on the fly. This is effectively grouping by book (for which the key is book#*). The stream comes out of the join partitioned this way already, so no sorting or partitioning operator is needed.*
The final project (without distinct) can, of course, be done on the fly.
Thus, this costs 400 I/O's in all.
All of this can be done wit 20 buffer frames (BF). No operation here is BF intensive.

c. (2 points) Agatha Christie has written *many more* books than the average author (in **Books**). Given this statistic (say, that Agatha Christie has 66 books in **Books**), would the plan you generated for Question 4b have been different? Why or why not?

In this case, we would likely keep the same solution. The reduction factor for the AC select would be less selective, but the cost of the INL join would remain fairly good.
Note that if Agatha Cristie's (AC's) books sell much better than others, on average, this would be a greater deviation from our uniformity assumptions and would lead to the INL join being much more expensive.
Generally speaking though, being quite off on a reduction-factor estimate can lead to picking a sub-optimal plan.

(Scratch space.)

(Scratch space.)

Relax. Turn in your exam. Go home.