COMP 3311
DATABASE MANAGEMENT SYSTEMS

FINAL REVIEW

EXTERNAL SORTING

Consider the relation Sailor(sailorId, name, rating, age), which is not sorted. Assume each attribute is 25 bytes, the page size is 1,000 bytes and there are 11,000 tuples. For the following questions, apply external sorting using a buffer of 11 pages.

A. How many sorted runs will be produced in pass 0?

- **a) 10**
- **b) 11**
- **c) 100**
- **d) 110**
- **e) None of the above.**

tuple size: 100 bytes
blocking factor: 10 tuples/page
file size: 1100 pages
memory size: 11 pages

In pass zero, 11 pages at a time are sorted in memory.

Therefore, in pass zero 1100/11 = 100 sorted runs are produced.

EXTERNAL SORTING (CONT'D)

B. What is the **total number of passes** required to sort the result completely (including pass 0)?

a) 1

b) 2

c) $\boxed{3}$

d) 4

e) 5

tuple size: 100 bytes
blocking factor: 10 tuples/page
file size: 1100 pages
memory size: 11 pages

After pass zero there are 100 sorted runs.

In pass 1, 10 runs at a time are merged, producing 10 sorted runs.

In pass 2, these 10 sorted runs are merged to produce the final output.

Therefore, the total number of passes = 3.

EXTERNAL SORTING (CONT'D)

C. What is the total page I/O cost of sorting?

- **a) 4,400**
- **b) 5,500**
- **c) 6,600**
- **d) 7,770**
- **e) None of the above.**

tuple size: 100 bytes
blocking factor: 10 tuples/page
file size: 1100 pages
memory size: 11 pages

There are a total of three passes.

In each pass the whole relation (1100 pages) is read and written.

Therefore, the total cost is 3 * 2 * 1100 = 6,600 page I/Os.

QUERY PROCESSING

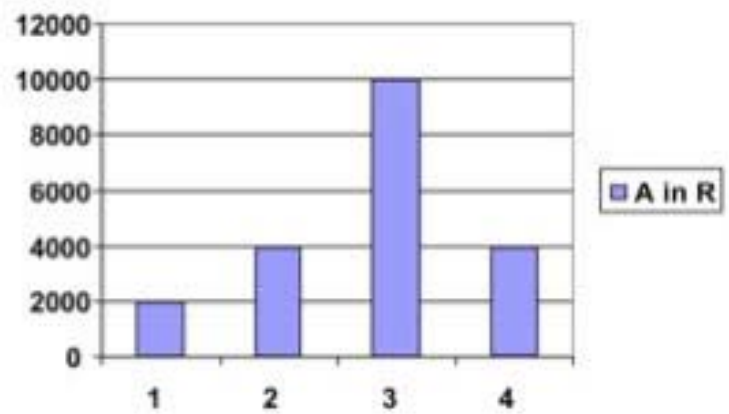
Consider the two relations R(A, *B, C*) and S(A, B, Y).

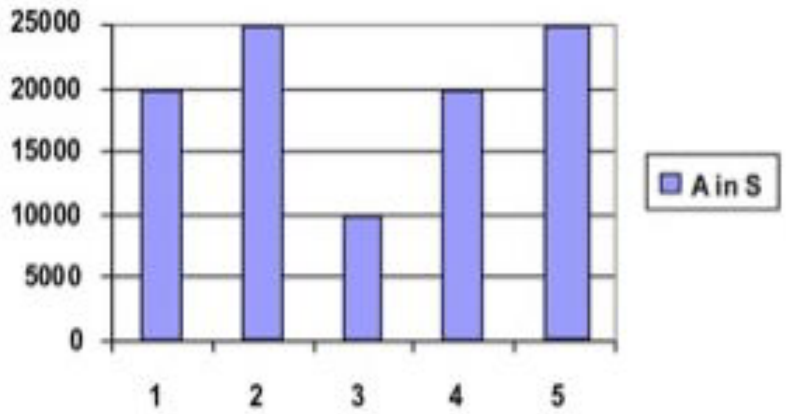
R contains 20,000 tuples and S contains 100,000 tuples.

Assume that for both relations, 10 tuples fit per page (i.e., the size of R is 2,000 pages and that of S is 10,000 pages).

The possible values of attribute A in R are {1, 2, 3, 4}, whereas the possible values of A in S are {1, 2, 3, 4, 5}.

The following histograms present statistical information about the occurrences of values for A in R and S (e.g., there are 2,000 tuples with A=1 in R and 20,000 tuples with A=1 in S).





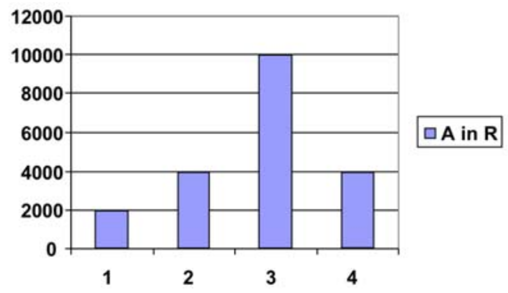
R(A, B, C)
S(A, B, Y)

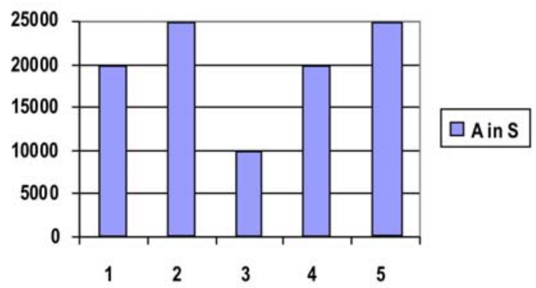
QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

A. How many tuples are there in the result of the query (R $\text{JOIN}_{R.A=S.A}$ S) (i.e., what is the cardinality of the output)?

a) 20,000
b) 100,000
c) 120,000
d) 320,000
e) 320,000,000





According to the histograms, each tuple of R with value R.A=1 (of which there are 2,000) can join with 20,000 tuples of S with S.A=1. That is, the join result will contain 2,000 * 20,000 = 40 * 10^6 tuples where R.A=S.A=1.

Performing the same computation for A = 2, 3, 4 and 5 the final result will contain (40 + 100 + 100 + 80 + 0) * 10^6 = 320,000,000 tuples.

R(A, B, C)
S(A, B, Y)

QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

B. What is the minimum cost (in terms of page I/Os) for computing (R JOIN$_{R.A=S.A}$ S) using block nested-loop join and how many main memory pages are needed?

a) **Minimum cost is 12,000 and I need 2,000 main memory pages.**

b) **Minimum cost is 12,000 and I need 2,002 main memory pages.**

c) **Minimum cost is 12,000 and I need 12,000 main memory pages.**

d) **Minimum cost is 320,000 and I need 2,002 main memory pages.**

e) **Minimum cost is 320,000 and I need 12,000 main memory pages.**

Since there is no index, at least both relations have to be read (i.e., the minimum cost is 12,000 page I/Os, *independent of the algorithm*.

For block nested-loop join, the smaller relation R (2,000 pages) needs to be able to be kept in memory, so that S is scanned only once.

Since 1 page is needed for reading S and 1 page for holding the output, a total of 2,002 main memory pages are needed.

R(A, B, C)
S(A, B, Y)

QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

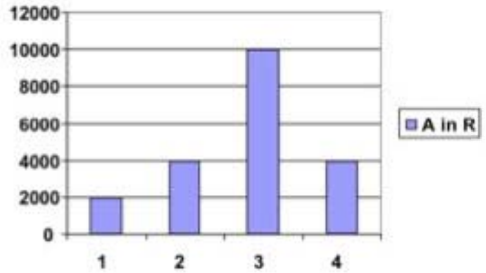
C. We want to compute (R JOIN$_{R.A=S.A}$ S) using block nested-loop join with *R as the outer relation*. What is the minimum number of main memory pages needed in order to achieve a cost of 42,000 page I/Os?

a) 502
b) 736
c) 1,000
d) 1,002
e) 2,000

Since R (2,000 pages) is the outer relation and the total cost is 42,000 page I/Os, S needs to be scanned (42,000 - 2,000) / 10,000 = 4 times.

If S is scanned 4 times, this means that 4 "blocks" of R need to be read, and each "block" should be at least 2,000 / 4 = 500 pages.

Since, 1 page is needed for reading S and 1 page for the output, in total 502 main memory pages are needed.



QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

D. We want to compute (R JOIN$_{R.A=S.A}$ S) **using hash join with *R as the build input*. How many buckets should be used for partitioning and what is the minimum main memory requirement?**

a) 4 buckets and at least 202 main memory pages.

b) 4 buckets and at least 1,002 main memory pages.

c) 10 buckets and at least 202 main memory pages.

d) 10 buckets and at least 502 main memory pages.

e) 20 buckets and at least 102 main memory pages.

The join condition is R.A=S.A and there are only four values of A in R. Therefore, 4 buckets should be used.

According to the histograms, half of the tuples (10,000 tuples) of the build input A, have value R.A=3. Thus, the bucket size should be 1,000 pages so that each bucket of R can fit into main memory.

In addition, we need 1 page for reading S and 1 for the output.

Therefore, 1,002 main memory pages are needed.

R(A, B, C)
S(A, B, Y)

QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

E. Given that R.B is a NOT NULL foreign key referencing S.B, how many tuples are in the result of the query (R $JOIN_{R.B=S.B}$ S)?

- **a) 20,000**
- **b) 100,000**
- **c) 120,000**
- **d) 320,000**
- **e) 320,000,000**

Each tuple of R joins with exactly one tuple in S.

Therefore, the size of the join result is the same as the size of R, namely, 20,000 tuples.

R(A, B, C)
S(A, B, Y)

QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

F. We want to compute (R $\text{JOIN}_{R.B=S.B}$ S) using indexed nested-loop join with *R as the outer relation*. Assume that there is a hash index on S.B with no overflow buckets (i.e., finding an index entry has cost 1). What is the total page I/O cost of the join?

a) 6,000

b) 22,000

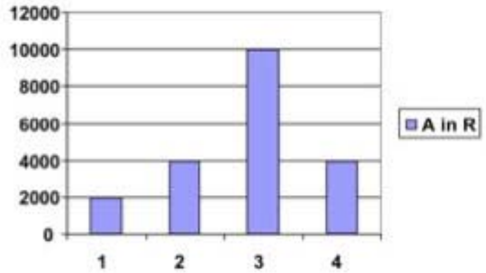
c) 40,000

d) 42,000

e) None of the above.

For each tuple of R, the index entry with the corresponding value of B is found and the pointer followed to the tuple of S for a cost of 2 page I/Os per tuple. This is repeated for all 20,000 tuples of R, for a total cost of 20,000 * 2 = 40,000 page I/Os.

In addition, the tuples of R have to be read for a total cost of 40,000 + 2,000 = 42,000 page I/Os.



QUERY PROCESSING (CONTD)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

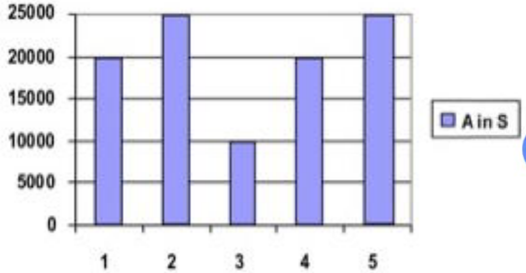
G. **How many tuples are expected in the result of $((\sigma_{A=1}R)\ \text{JOIN}_{R.B=S.B}\ S)$ and what is the minimum page I/O cost of processing the query using indexed nested-loop join with *R as the outer relation*. Assume that the only index is a hash index on S.B with no overflow buckets.**

- **a) The result has 2,000 tuples and the cost is 6,000 page I/Os.**
- **b) The result has 2,000 tuples and the cost is 22,000 page I/Os.**
- **c) The result has 20,000 tuples and the cost is 40,000 page I/Os.**
- **d) The result has 20,000 tuples and the cost is 42,000 page I/Os.**
- **e) None of the above.**

According to the histograms, there are only 2,000 tuples in R with R.A=1. Each of these tuples, matches exactly 1 tuple in the join with S (R.B=S.B). Thus, the result contains 2,000 tuples.

Finding the matching tuple in S, has cost 2 page I/Os (see previous question) and so finding all matches for the 2,000 tuples has cost 4,000 page I/Os.

Adding the cost of reading R, the total cost is 6,000 page I/Os.



QUERY PROCESSING (CONT'D)

Relation	R	S
# tuples	20,000	100,000
# pages	2,000	10,000

H. How many tuples are expected in the result of $((\sigma_{A=1}R)$ JOIN$_{R.B=S.B}$ $(\sigma_{A=3}S))$ and what is the minimum page I/O cost of processing the query using index nested-loop join with *R as the outer relation*. Assume that the only index is a hash index on S.B with no overflow buckets.

a) **Expected number of tuples is 200 with cost 600 page I/Os.**

b) **Expected number of tuples is 200 with cost 6,000 page I/Os.**

c) **Expected number of tuples is 2,000 with cost 6,000 page I/Os.**

d) **Expected number of tuples is 2,000 with cost 42,000 page I/Os.**

e) **None of the above.**

According to the histogram of S, among the 2,000 tuples in the result of $((\sigma_{A=1}R)$ JOIN$_{R.B=S.B}$ S), only 10,000/100,000=0.1 (i.e.,10%) are expected to satisfy the condition S.A=3. Thus, the result is expected to contain (2,000 * 0.1) = 200 tuples.

The query processing is the same as in the previous question with cost 6,000 page I/Os.

TRANSACTION MANAGEMENT

3. Consider the following schedules of two transactions T_1 and T_2. Indicate for each whether it is serial, (conflict) serializable or not serializable. r denotes a READ and w a WRITE operation.

   a) Schedule: $r_1(A)\ w_1(A)\ r_2(A)\ w_2(B)$

   Serial: T_1, T_2

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(B)

3. **Consider the following schedules of two transactions T_1 and T_2. Indicate for each whether it is serial, (conflict) serializable or not serializable. r denotes a READ and w a WRITE operation.**

   b) **Schedule: $r_1(A)$ $r_2(A)$ $w_1(A)$ $w_2(B)$**

   Serializable: T_2, T_1

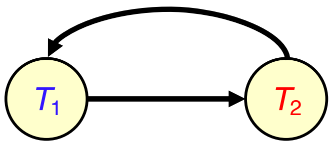
T_1	T_2
read(A)	
	read(A)
write(A)	
	write(B)

TRANSACTION MANAGEMENT (CONT'D)

3. **Consider the following schedules of two transactions T_1 and T_2. Indicate for each whether it is serial, (conflict) serializable or not serializable. r denotes a READ and w a WRITE operation.**

   c) **Schedule: $r_1(A)\ r_2(A)\ w_2(A)\ w_1(A)$**

   Not Serializable



T_1	T_2
read(A)	
	read(A)
	write(A)
write(A)	

TRANSACTION MANAGEMENT (CONT'D)

4. Consider the schedule $r_1(A)$ $w_1(A)$ $r_2(A)$ $w_2(B)$ c_1 c_2 (where c_1 and c_2 indicate the commit statements).

   a) Is the schedule recoverable? Why?

   It is recoverable (T_2 reads database item A written by T_1, but T_1 commits before T_2).

   Is the schedule cascadeless? Why?

   It is not cascadeless (T_2 read database item A before T_1 commits).

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(B)
commit	
	commit

TRANSACTION MANAGEMENT (CONT'D)

4. **Consider the schedule: $r_1(A)$ $w_1(A)$ $r_2(A)$ $w_2(B)$ c_1 c_2 (where c_1 and c_2 indicate the commit statements).**

   b) **Change the time of the commits (c_1, c_2) in the schedule in a) so that it becomes a cascadeless schedule.**

   $r_1(A)$ $w_1(A)$ c_1 $r_2(A)$ $w_2(B)$ c_2

   Transaction *T_2* reads the value of A written by *T_1*, after *T_1* commits.

T_1	T_2
read(A)	
write(A)	
commit	
	read(A)
	write(B)
	commit

TRANSACTION MANAGEMENT (CONT'D)

4. Consider the schedule: $r_1(A)\ w_1(A)\ r_2(A)\ w_2(B)\ c_1\ c_2$ (where c_1 and c_2 indicate the commit statements).

   (c) Is the schedule $r_2(A)\ r_1(A)\ w_1(A)\ w_2(B)\ c_2\ c_1$ recoverable and cascadeless?

   Both recoverable and cascadeless – no transaction reads items after they have been written by the other.

T_1	T_2
	read(A)
read(A)	
write(A)	
	write(B)
	commit
commit	

TRANSACTION MANAGEMENT (CONT'D)

5. Rewrite the schedule $r_2(A)$ $r_1(A)$ $w_1(A)$ $w_2(B)$ according to the 2PL protocol (i.e., add lock-S, lock-X, unlock statements below). Explain briefly whether the schedule is allowed by 2PL.

   The schedule is allowed by 2PL . T_1 has to wait until T_2 unlocks A.

T_1	T_2
	lock-S(A)
	read(A)
lock-S(A)	
read(A)	
lock-X(A) $\Longrightarrow$ **wait**	
write(A)	
	lock-X(B)
	write(B)

6. **Rewrite the schedule $r_2(A)$ $r_1(A)$ $w_1(A)$ $w_2(B)$ according to timestamp-ordering protocol (i.e., add RTS (read timestamp) and WTS (write timestamp) statements). Assume that the timestamps of T_1 and T_2 are 2 and 1, respectively. The initial read and write timestamps of A and B are both 0.**

T_1 [TS=2]	T_2 [TS=1]
	read(A) RTS(A)=1; WTS(A)=0
read(A) RTS(A)=2; WTS(A)=0	
write(A) RTS(A)=2; WTS(A)=2	
	write(B) RTS(B)=0; WTS(B)=1

Read
If TS(T_i) < WTS(Q) **rollback**
If TS(T_i) ≥ WTS(Q)
RTS(Q) = max(TS(T_i), RTS(Q))

Write
If TS(T_i) < RTS(Q) **rollback**
If TS(T_i) < WTS(Q) **ignore**
Otherwise WTS(Q) = TS(T_i)

7. Rewrite the schedule $r_2(A)\ r_1(A)\ w_1(A)\ w_2(B)$ according to the multi-version timestamp-ordering protocol (i.e., add RTS (read timestamp) and WTS (write timestamp) statements and specify the versions of the items). Assume that the timestamps of T_1 and T_2 are 1 and 2, respectively and that the initial versions of items are A_0 and B_0. Complete the correct version numbers (e.g., read(A_0) instead of read(A)).

T_1 [TS=1]	T_2 [TS=2]
	read(A_0) RTS(A_0)=2; WTS(A_0)=0
read(A_0) RTS(A_0)=2; WTS(A_0)=0	
write(A) TS(T_1)=1 < RTS(A_0)=2 ⟹ **rollback**	
	write(B)

Read

Reads always succeed

set RTS(Q) = TS(T_i)

Write

If TS(T_i) < RTS(Q) **rollback**

If TS(T_i) = WTS(Q)

overwrite contents

If TS(T_i) > WTS(Q)

create new version

set R/WTS(Q')=TS(T_i)