

Name

CWID

# Quiz 2

**Due November 26th, 2015**

## CS525 - Advanced Database Organization Solutions

---

*Please leave this empty!*

1  2  3  4  5  6  7

Sum

# Instructions

- Multiple choice questions are graded in the following way: You get points for correct answers and points subtracted for wrong answers. The minimum points for each questions is **0**. For example, assume there is a multiple choice question with 6 answers - each may be correct or incorrect - and each answer gives 1 point. If you answer 3 questions correct and 3 incorrect you get 0 points. If you answer 4 questions correct and 2 incorrect you get 2 points. . . .
- For your convenience the number of points for each part and questions are shown in parenthesis.
- There are 3 parts in this quiz
  1. Disk Organization and Buffering
  2. Index Structures
  3. Result Size Estimations
  4. I/O Cost Estimation
  5. Schedules
  6. ARIES (Optional)
  7. Physical Optimization (Optional)

## Part 1 Disk Organization and Buffering (Total: 15 Points)

### Question 1.1 Page Replacement Clock (15 Points)

Consider a buffer pool with 3 pages using the **Clock** page replacement strategy. Initially the buffer pool is in the state shown below. We use the following notation  $^{flag}[page]_{fix}^{dirty}$  to denote the state of each buffer frame.  $page$  is the number of the page in the frame,  $fix$  is its fix count,  $dirty$  is indicating with an Asterix that the page is dirty, and  $flag$  is the reference bit used by the Clock algorithm. E.g.,  $^1[5]_2^*$  denotes that the frame stores page 5 with a fix count 2, that the page is dirty, and that the reference bit is set to 1. Recall that Clock uses a pointer  $S$  that points to the current page frame (the one to be checked for replacement next). The page frame  $S$  is pointing to is shown in bold. In your solution draw an arrow to the page frame that  $S$  is pointing to.

#### Current Buffer State

$$^0[\mathbf{5}]_0 \ ^1[2]_2^* \ ^0[3]_1$$

Execute the following requests and write down state of the buffer pool after each request.

- $p$  stands for pin
- $u$  for unpin
- $d$  for marking a page as dirty

$$p(9), d(3), u(2), u(3), p(6), u(9), p(4)$$

#### Solution

p(9)

$$^1[9]_1 \ ^1[\mathbf{2}]_2^* \ ^0[3]_1$$

d(3)

$$^1[9]_1 \ ^1[\mathbf{2}]_2^* \ ^0[3]_1^*$$

u(2)

$$^1[9]_1 \ ^1[\mathbf{2}]_1^* \ ^0[3]_1^*$$

u(3)

$$^1[9]_1 \ ^1[\mathbf{2}]_1^* \ ^0[3]_0^*$$

p(6)

$$^1[\mathbf{9}]_1 \ ^0[2]_1^* \ ^1[6]_1$$

u(9)

$$^1[\mathbf{9}]_0 \ ^0[2]_1^* \ ^1[6]_1$$

p(4)

$$^1[4]_1 \ ^0[\mathbf{2}]_1^* \ ^0[6]_1$$

## Part 2 Index Structures (Total: 25 Points)

Assume that you have the following table:

Item		
name	age	salary
Peter Petersen	34	30,000
Gert Gertsen	24	90,000
Heinz Heinzen	20	10,000
Gertrud Gertsen	20	15,000
Astrid Lundgren	20	100,000
Pferdegert	43	0
Heinz Bert	43	23,000

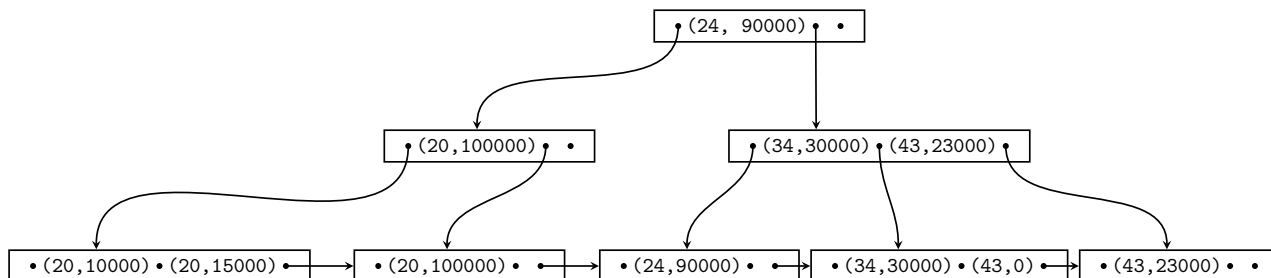
### Question 2.1 Construction (9 Points)

Create a B+-tree for table *Item* over attributes *age* and *salary* with  $n = 2$  (up to two keys per node). You should start with an empty B+-tree and insert the keys in the order shown in the table above. Write down the resulting B+-tree after each step.

When splitting or merging nodes follow these conventions:

- **Leaf Split:** In case a leaf node needs to be split during insertion and  $n$  is even, the left node should get the extra key. E.g, if  $n = 2$  and we insert a key 4 into a node  $[1,5]$ , then the resulting nodes should be  $[1,4]$  and  $[5]$ . For odd values of  $n$  we can always evenly split the keys between the two nodes. In both cases the value inserted into the parent is the smallest value of the right node.
- **Non-Leaf Split:** In case a non-leaf node needs to be split and  $n$  is odd, we cannot split the node evenly (one of the new nodes will have one more key). In this case the “middle” value inserted into the parent should be taken from the right node. E.g., if  $n = 3$  and we have to split a non-leaf node  $[1,3,4,5]$ , the resulting nodes would be  $[1,3]$  and  $[5]$ . The value inserted into the parent would be 4.
- **Node Underflow:** In case of a node underflow you should first try to redistribute values from a sibling and only if this fails merge the node with one of its siblings. Both approaches should prefer the left sibling. E.g., if we can borrow values from both the left and right sibling, you should borrow from the left one.

### Solution

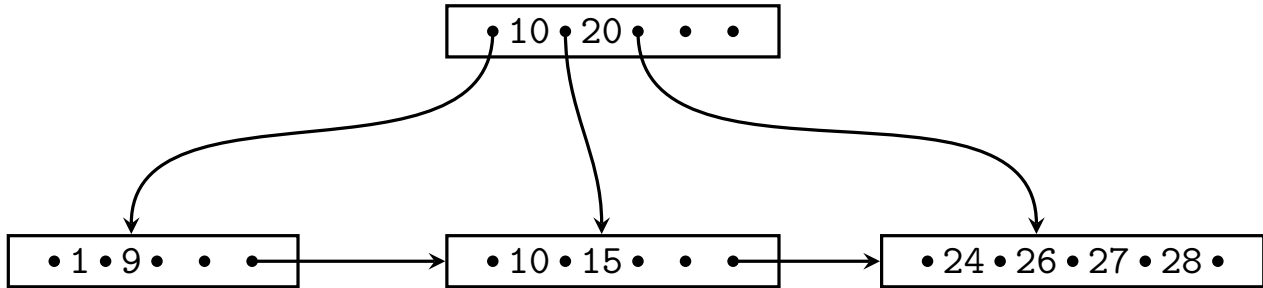


## Question 2.2 Operations (9 Points)

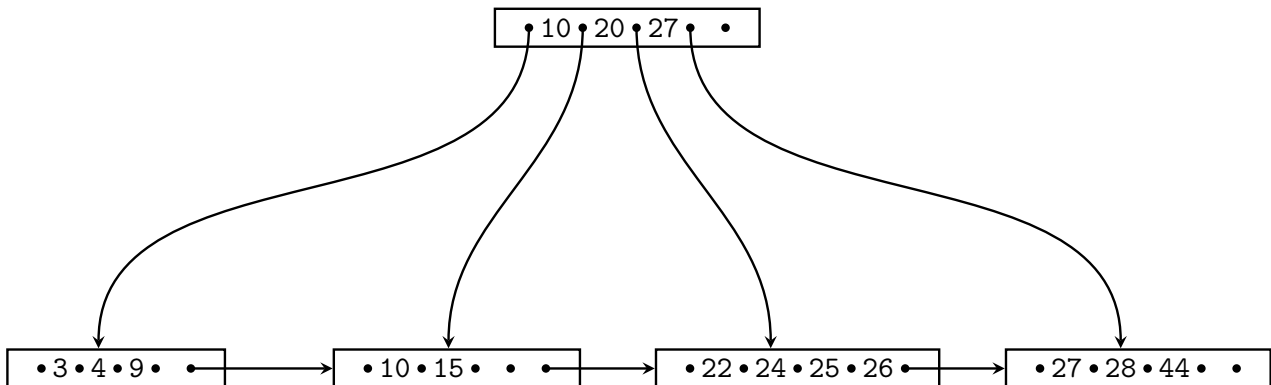
Given is the B+-tree shown below ( $n = 4$ ). Execute the following operations and write down the resulting B+-tree after each operation:

`delete(1)`, `insert(3)`, `insert(4)`, `insert(25)`, `insert(22)`, `insert(44)`

Use the conventions for splitting and merging introduced in the previous question.



Solution





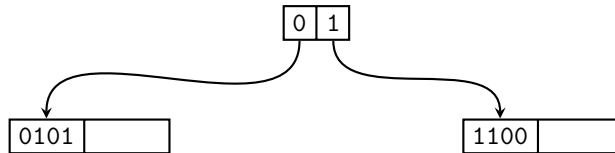
### Question 2.3 Extensible Hashing (7 Points)

Consider the extensible Hash index shown below that is the result of inserting values 4 and 6. Each page holds two keys. Execute the following operations

`insert(0), insert(2), insert(5), insert(7), insert(1)`

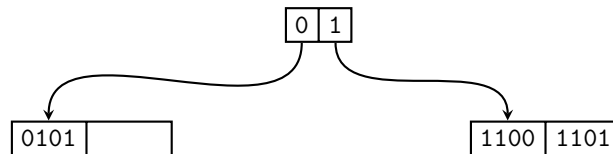
and write down the resulting index after each operation. Assume the hash function is defined as:

x	h(x)
0	1101
1	1001
2	1110
3	1000
4	0101
5	0000
6	1100
7	0111
8	0100

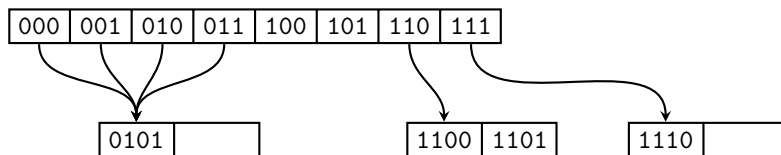


#### Solution

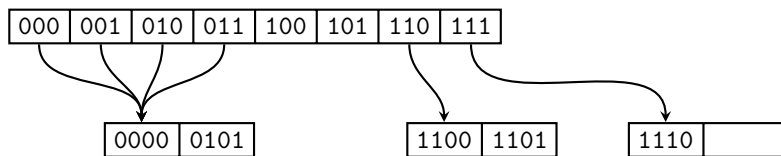
`insert(0)`



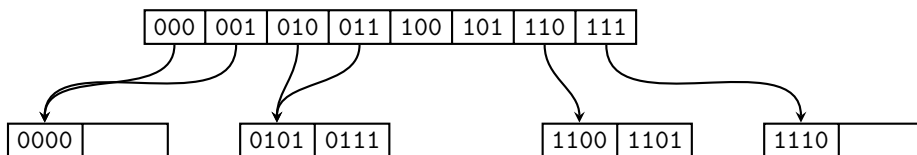
`insert(2)`



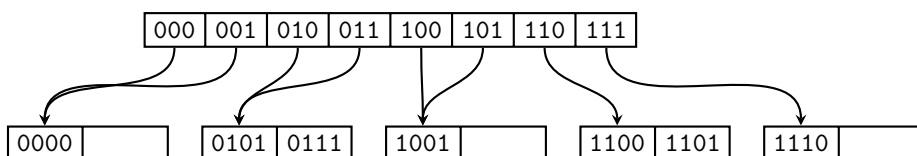
`insert(5)`



`insert(7)`



`insert(1)`









### Part 3 Result Size Estimations (Total: 20 Points)

Consider a table *student* with attributes *CWID*, *name*, *major*, *credits*, a table *course* with *title*, *instructor*, *credits*, and a table *registered* with attributes *student* and *course*. *registered.student* is a foreign key to student *CWID*. Attribute *course* of relation *registered* is a foreign key to attribute *title* of relation *course*. Given are the following statistics:

$$\begin{array}{lll} T(student) = 30,000 & T(course) = 80 & T(registered) = 10,000 \\ V(student, CWID) = 30,000 & V(course, title) = 80 & V(registered, student) = 3,000 \\ V(student, name) = 29,500 & V(course, instructor) = 50 & V(registered, course) = 30 \\ V(student, major) = 20 & V(course, credits) = 6 & \\ V(student, credits) = 32 & & \end{array}$$

The min and max values for some of the columns are:

$$\begin{array}{ll} \min(course, credits) = 0 \text{ (should be 1)} & \max(course, credits) = 36 \text{ (should be 6)} \\ \min(student, credits) = 0 & \max(student, credits) = 36 \end{array}$$

#### Question 3.1 Estimate Result Size (4 Points)

Estimate the number of result tuples for the query  $q = \sigma_{instructor=Peter}(course)$  using the first assumption presented in class (values used in queries are uniformly distributed within the active domain).

#### Solution

$$T(q) = \frac{T(course)}{V(course, instructor)} = \frac{80}{50} = 1.6$$

#### Question 3.2 Estimate Result Size (5 Points)

Estimate the number of result tuples for the query  $q = \sigma_{major=CS \wedge credits < 16}(student)$  using the first assumption presented in class. Assume that the minimal and maximal values in the *credits* attribute are 1 and 32, respectively.

### Solution

Because the disjunctive condition uses the same attributes, we can estimate it by summing up the probabilities.

$$\begin{aligned} T(q) &= \frac{T(student)}{V(student, major)} \cdot \frac{(15 - 1 + 1)}{\max(student, credits) - \min(student, credits) + 1} \\ &= \frac{30,000}{20} \cdot \frac{15 - 1 + 1}{36 - 0 + 1} = 1,500 \cdot \frac{15}{37} \approx 608 \end{aligned}$$

### Question 3.3 Estimate Result Size (5 Points)

Estimate the number of result tuples for the query  $q = \sigma_{credits \geq 4 \wedge credits \leq 6}(course)$  using the first assumption presented in class. Assume that the minimal and maximal values in the `credits` attribute are 1 and 6,

### Solution

$$T(q) = \frac{(6 - 4 + 1) \cdot T(course)}{\max(course, credits) - \min(course, credits) + 1} = \frac{3 \cdot 80}{6 - 1 + 1} = \frac{80 \cdot 3}{6} = 40$$

if done using the min and max values given

$$T(q) = \frac{(6 - 4 + 1) \cdot T(course)}{\max(course, credits) - \min(course, credits) + 1} = \frac{3 \cdot 80}{36 - 0 + 1} = \frac{80 \cdot 3}{37} \approx 6$$

### Question 3.4 Estimate Result Size (6 Points)

Estimate the number of result tuples for the query

$q = \sigma_{course=DatabaseIntroduction \vee course=AdvancedDatabase}(registered) \bowtie_{course=title} course$

using the first assumption presented in class.

## Solution

To estimate the selection result size  $q_1 = \sigma_{course=DatabaseIntroduction \vee course=AdvancedDatabase}(registered)$ :

$$\begin{aligned} T(q_1) &= (1 - [(1 - \frac{1}{V(registered, course)}) \cdot (1 - \frac{1}{V(registered, course)})]) \cdot T(registered) \\ &= (1 - (1 - \frac{1}{30}) \cdot (1 - \frac{1}{30})) \cdot 10,000 \approx 656 \end{aligned}$$

Recognizing that both predicates are over the same attribute alternatively this can be computed as

$$\begin{aligned} T(q_1) &= \frac{2}{V(registered, course)} \cdot T(registered) \\ &= \frac{2}{30} \cdot 10,000 \approx 667 \end{aligned}$$

$$\begin{aligned} T(q) &= \frac{T(q_1) \cdot T(course)}{\max(V(q_1, course), V(course, title))} \\ &= \frac{656 \cdot 80}{\max(2, 80)} = 656 \end{aligned}$$

## Part 4 I/O Cost Estimation (Total: 20 Points)

### Question 4.1 External Sorting (4 Points)

You have  $M = 21$  memory pages available and should sort a relation  $R$  with  $B(R) = 16,000,000$  blocks. Compute the number of I/Os necessary to sort  $R$  using the external merge sort algorithm introduced in class.

#### Solution

$$\begin{aligned} IO &= 2 \cdot B(R) \cdot (1 + \lceil \log_{M-1}(\frac{B(R)}{M}) \rceil) \\ &= 2 \cdot 16,000,000 \cdot (1 + 5) \\ &= 192,000,000 \end{aligned}$$

### Question 4.2 External Sorting (4 Points)

You have  $M = 801$  memory pages available and should sort a relation  $R$  with  $B(R) = 10,000$  blocks. Compute the number of I/Os necessary to sort  $R$  using the external merge sort algorithm introduced in class.

#### Solution

$$\begin{aligned} IO &= 2 \cdot B(R) \cdot (1 + \lceil \log_{M-1}(\frac{B(R)}{M}) \rceil) \\ &= 2 \cdot 10,000 \cdot (1 + 1) \\ &= 40,000 \end{aligned}$$

### Question 4.3 I/O Cost Estimation (6 = 2+2+2 Points)

Consider two relations  $R$  and  $S$  with  $B(R) = 70,000$  and  $B(S) = 15,000$ . You have  $M = 101$  memory pages available. Estimate the minimum number of I/O operations needed to join these two relations using **block-nested-loop join**, **merge-join** (the inputs are not sorted), and **hash-join**. You can assume that the hash function evenly distributes keys across buckets. Justify your result by showing the I/O cost estimation for each join method.

## Solution

- **BNL**:  $S$  is smaller, thus, keep chunks of  $S$  in memory  
 $\lceil \frac{B(S)}{M-1} \rceil \cdot [B(R) + \min(B(S), (M-1))] = 150 \cdot [70,000 + 100] = 10,515,000$  I/Os
- **MJ**: We can generate sorted runs of size 100 that means the number of sorted runs from  $R$  and  $S$  is low enough to keep one page from each run of both  $R$  and  $S$  in memory. We need 2 merge pass for the sort for  $R$  and  $S$ , but can execute the merge phase and join in one pass.  $5 \cdot (B(R) + B(S)) = 5 \cdot (70,000 + 15,000) = 425,000$  I/Os.
- **HJ**: We need 2 partitioning pass, because we can create 100 buckets and the bucket sizes of  $R$  and  $S$  will be 700 and 150 after one pass. Thus, we can not fit one bucket from  $S$  (the smaller relation) into memory and a second partitioning phase is needed. Cost is  $(4+1) \cdot (B(R) + B(S)) = 5 \cdot (70,000 + 15,000) = 425,000$  I/Os.

## Question 4.4 I/O Cost Estimation (6 = 2+2+2 Points)

Consider two relations  $R$  and  $S$  with  $B(R) = 5,000,000$  and  $B(S) = 6,000,000$ . You have  $M = 11$  memory pages available. Compute the minimum number of I/O operations needed to join these two relations using **block-nested-loop join**, **merge-join** (the inputs are not sorted), and **hash-join**. You can assume that the hash function evenly distributes keys across buckets. Justify your result by showing the I/O cost estimation for each join method.

## Solution

- **BNL**:  $R$  is smaller, thus, keep chunks of  $R$  in memory  
 $\lceil \frac{B(R)}{M-1} \rceil \cdot [B(S) + \min(B(R), (M-1))] = 500,000 \cdot [6,000,000 + 100] = 3,000,050,000$  I/Os
- **MJ**: We can generate sorted runs of size 10 that means the number of sorted runs from  $R$  and  $S$  is low enough after 6 merge passes to keep one page from each run of both  $R$  and  $S$  in memory (5 runs for  $R$  and 6 runs for  $S$ ). We need 6 merge passes for the sort, but can execute the last merge phase and join in one pass.  $(12 + 1) \cdot (B(R) + B(S)) = 13 \cdot (5,000,000 + 6,000,000) = 143,000,000$  I/Os.
- **HJ**: We need 6 partitioning passes, because we can create 10 buckets. The bucket sizes of  $R$  and  $S$  after the sixth partitioning step will be 6 and 5. Thus, we can fit one bucket from  $R$  into memory to join with  $S$ . Cost is  $(12 + 1) \cdot (B(R) + B(S)) = 13 \cdot (5,000,000 + 6,000,000) = 143,000,000$  I/Os.

## Part 5 Schedules (Total: 20 Points)

### Question 5.1 Schedule Classes (20 Points)

Indicate which of the following schedules belong to which class. Recall transaction operations are modelled as follows:

$w_1(A)$  transaction 1 wrote item  $A$   
 $r_1(A)$  transaction 1 read item  $A$   
 $c_1$  transaction 1 commits  
 $a_1$  transaction 1 aborts

$S_1 = w_1(A), r_2(B), r_3(E), w_1(C), c_1, r_4(E), w_2(A), w_2(B), c_2, r_3(D), w_3(C), w_3(D), c_3, r_4(D), c_4$

$S_2 = r_3(B), w_3(B), r_3(A), w_2(A), r_1(B), w_1(B), c_1, r_2(B), c_2, r_3(A), c_3$

$S_3 = r_1(A), r_2(B), r_1(B), r_2(A), w_2(B), c_2, r_1(B), w_1(A), c_1$

$S_4 = w_2(B), r_1(B), c_1, w_3(B), c_3, r_2(A), c_2$

- $S_1$  is recoverable
- $S_1$  is cascade-less
- $S_1$  is strict
- $S_1$  is conflict-serializable
- $S_1$  is 2PL
  
- $S_2$  is recoverable
- $S_2$  is cascade-less
- $S_2$  is strict
- $S_2$  is conflict-serializable
- $S_2$  is 2PL
  
- $S_3$  is recoverable
- $S_3$  is cascade-less
- $S_3$  is strict
- $S_3$  is conflict-serializable
- $S_3$  is 2PL
  
- $S_4$  is recoverable
- $S_4$  is cascade-less
- $S_4$  is strict
- $S_4$  is conflict-serializable
- $S_4$  is 2PL





## Part 6 Optional: ARIES (Total: 10 Optional Points)

### Question 6.1 Recovery (10 Points)

Consider the state of the log and pages on disk shown below. For simplicity we do not show the actual undo/redo actions for updates, but instead show only the affected page. Assume a crash occurred after the last log entry. Answer the following questions:

1. **Analysis:** Write down the result of the analysis phase (RedoLSN, Transaction Table, Dirty Page Table)
2. **Redo:** Which pages will be loaded from disk during redo? Which pages will be modified during redo?
3. **Undo:** Write down the additional log entries that will be written during undo.

#### Log

LSN	Type	TID	PrevLSN	UndoNxtLSN	Data
1	begin	1	-	-	-
2	update	1	1	-	Page 3
3	begin	2	-	-	-
4	update	2	3	-	Page 3
5	begin	3	-	-	-
6	update	2	4	-	Page 5
7	begin_cp	-	-	-	-
8	update	3	5	-	Page 5
9	update	1	2	-	Page 1
10	update	3	8	-	Page 1
11	end_cp	-	-	-	Transaction Table: $\langle T_1, u, 9, - \rangle, \langle T_2, u, 6, - \rangle, \langle T_3, u, 10, - \rangle$ , Dirty Page Table: $\langle 1, 8 \rangle$
12	commit	3	10	-	-
13	update	2	6	-	Page 6
14	commit	2	13	-	-

#### Disk

PageID	PageLSN
1	0
3	4
5	8
6	0
15	0

### Solution

(1):

RedoLSN: 8

Transaction Table:  $\langle T_1, u, 9, - \rangle$

Dirty Page Table:  $\langle 1, 8 \rangle, \langle 5, 8 \rangle, \langle 6, 13 \rangle$

(2):

All pages (1,5,6) have to be loaded from disk.

Only page 1 and 6 will be modified based on redo info from log entry 9, 10 and 13.

(3):

Transaction  $T_1$  will be rolled back. The CLR's written during undoing this update is shown below.

LSN	Type	TID	PrevLSN	UndoNxtLSN	Data
15	CLR	1	-	2	Page 1
16	CLR	1	-	1	Page 3



## Part 7 Bonus: Physical Optimization (Total: 10 Bonus Points)

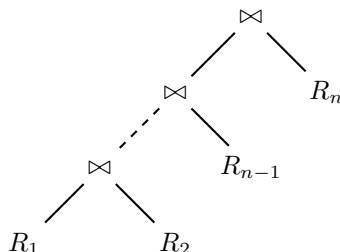
Consider the following relations  $R(A, B)$ ,  $S(C, D, E)$ ,  $T(F, G)$  with  $S = \frac{1}{10}$  (10 tuples fit on each page). The sizes and value distributions are:

$$\begin{array}{lll}
 N(R) = 60 & V(R, A) = 60 & V(R, B) = 10 \\
 N(S) = 5 & V(S, C) = 1 & V(S, D) = 2 & V(S, E) = 5 \\
 N(T) = 50,000 & V(T, F) = 50,000 & V(T, G) = 5
 \end{array}$$

### Question 7.1 Greedy Join Enumeration (10 Points)

Use the greedy join enumeration algorithm to find the cheapest plan for the join  $R \bowtie_{B=C} S \bowtie_{D=F} T$ . Assume that **nested-loop** (not the block based version) is the only available join implementation with the left input being the “outer” (for each tuple from the outer we have to scan the whole inner relation). Furthermore, there are no indices defined on any of the relations (that is you have to use **sequential scan** for each of the relations). As a cost model consider the **total number of I/O operations**. For example, if you join two relations with 5,000 and 10,000 tuples with  $S = \frac{1}{10}$ , where the 5,000 tuple relation is the outer, then the cost would be 5,000,000 (scan the inner 5000 times) + 500 to scan the other once. The total cost is then 5,000,500 I/Os. Assume that the system supports pipelining for the outer input of a join. That is if you join the result of a join with a relation where the join result is the outer, then there is no I/O cost for scanning the outer. Also under these assumptions you never have to store join results to disk. **Hint: You will have to estimate the size of intermediate results. Use the estimation based on the number of values and not the one based on the size of the domain. Use the assumption that the number of values in a join attribute of a join result is the minimum of the number of values in the join attribute of each input.**

Write down the state after each iteration of the algorithm using the following notation. Write  $((R_1, R_2), \dots, R_{n-1}), R_n)^{C, S}$  to denote a plan as shown below with I/O cost  $C$  and result size  $S$ . Alternatively you are also allowed to draw join trees as shown below.



### Solution

---

**Calculate Result Sizes:**

Using the formula from class the estimated result sizes are:

$$T(R \bowtie S) = \frac{T(R) \cdot T(S)}{\max(V(R, B), V(S, C))} = \frac{60 \cdot 5}{\max(10, 1)} = 30$$

$$T(S \bowtie T) = \frac{T(S) \cdot T(T)}{\max(V(S, D), V(T, F))} = \frac{5 \cdot 50,000}{\max(2, 50,000)} = 5$$

$$T(R \bowtie T) = T(R) \cdot T(T) = 60 \cdot 50,000 = 300,000$$

$$R(R \bowtie S \bowtie T) = \frac{T(R) \cdot T(S) \cdot T(T)}{\max(V(R, B), V(S, C)) \cdot \max(V(S, D), V(T, F))} = \frac{60 \cdot 5 \cdot 50,000}{10 \cdot 50,000} = 30$$


---

**Initialization:**

$$(R)^{6;60}, (S)^{1;5}, (T)^{5,000;50,000}$$

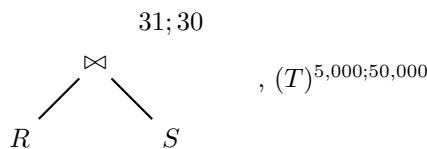

---

**n = 1:**

Here we have 6 different options how to join two of the plans from the initialization:

$$(R \bowtie S)^{66;30}, (R \bowtie T)^{300,006;300,000}, (S \bowtie R)^{31;30}, (S \bowtie T)^{25,001;5}, (T \bowtie R)^{300,500;300,000}, (T \bowtie S)^{55,001;5}$$

As an example take the join  $(R \bowtie S)$ . Here  $R$  is the outer and  $S$  is the inner. The cost is computed as: For each tuple from  $R$  ( $T(R)$ ) we have to scan  $S$  once (1 I/O). Thus, the cost is  $B(R) + T(R) \cdot B(S) = 6 + 60 \cdot 1 = 66$  I/Os. Greedy join enumeration chooses the plan with the lowest cost  $(S \bowtie R)$ :



**n = 2:**

Now we need to consider two join options.

For  $((S \bowtie R) \bowtie T)$  we pipeline the result of  $(S \bowtie R)$  so the cost is:

$$Cost(S \bowtie R) + T(S \bowtie R) \cdot B(T) = 31 + 30 \cdot 5,000 = 150,031$$

Recall that the assumption is that only the outer input of the join can be pipelined. For  $(T \bowtie (S \bowtie R))$ , the result of the join between  $(S \bowtie R)$  is the "inner", so we have to store the result of  $S \bowtie R$  on disk resulting in  $B(S \bowtie R)$  additional I/O. Since  $(S \bowtie R)$  has 30 result tuples and  $S(S \bowtie R) = S(S) + S(R) = 1/10 + 1/10 = 1/5$  it follows that  $B(S \bowtie R) = 6$ . Thus, the total cost is

$$Cost(S \bowtie R) + B(S \bowtie R) + B(T) + T(T) \cdot B(S \bowtie R) = 31 + 6 + 5,000 + 50,000 \cdot 6 = 305,037$$

$$(S \bowtie R) \bowtie T)^{150,031;30}, (T \bowtie (S \bowtie R))^{305,037;30}$$

