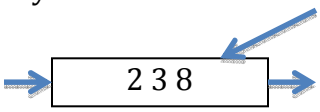
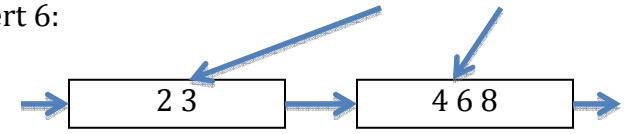
Solutions to 2004 final

1 Multiple Choice on Indexing and Query Processing(24)

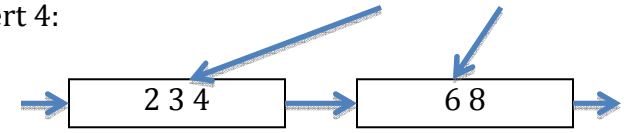
1.True. Different orders of insertion could result in different splitting. Consider this leaf in a B+ tree where n=3. Say we want to insert 4 and 6.



Insert 4, then insert 6:



Insert 6, then insert 4:



2. False. Not true for queries that select a certain value. B+ tree works better than hash with range queries.

3. False. B is primary key of S, so $V(S,B) = T(S) = 3000$.

$$T(R \bowtie S) = \frac{T(R)\times T(S)}{\max(V(R,B),V(S,B))} = \frac{5000\times 3000}{\max(V(R,B),3000)} \leq \frac{5000\times 3000}{3000} = 5000$$

4.True. According to the definition of foreign key and primary key, for each tuple r in R, there is a unique tuple s in S such that r.A=s.A. So the size of natural join is the size of R.

5. False. If the relation is not stored in an order on attribute A, then a sparse first level index for A is useless. So two separate sparse first level indexes on different keys do not make sense unless the storage preserves order for both attributes.

6. True. The relation could be stored without preserving order of either attribute.

7. False. The first level should be dense in case the relation is not stored in order of the attribute. Also a dense second level index is useless.

8. False. Counter example:

Consider these two tables

E1	
S	A
3	1
2	2

E2	
S	A
3	2
2	2

$PROJECT_S E1$
S
3
2

$PROJECT_S E2$
S
3
2

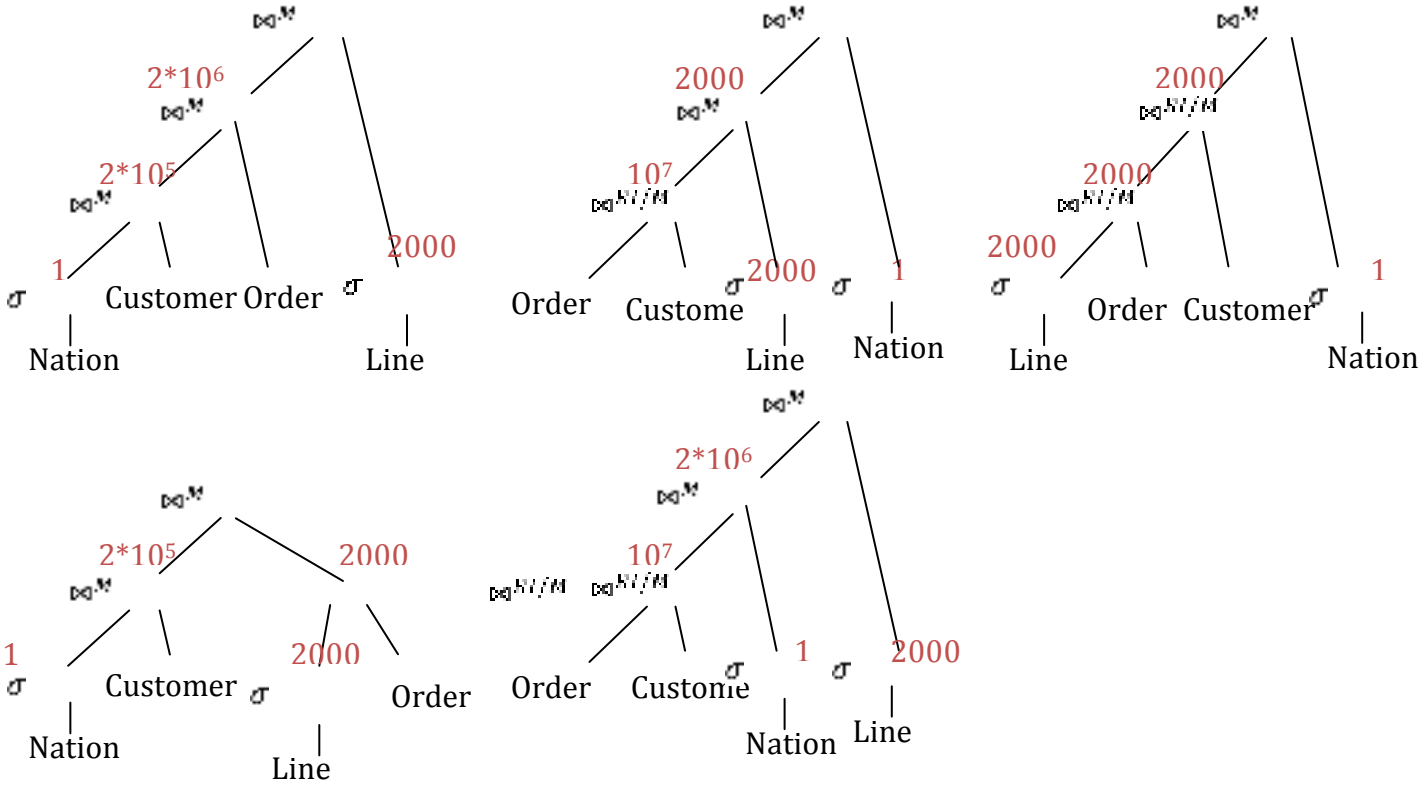
$PROJECT_S(E1 - E2)$
S
3

$\neq$

$(PROJECT_S E1) - (PROJECT_S E2)$
S

2 Midterm problem revisited (80 points)

1.2. 3.(To simplify, here we omit the equivalent plans of exchanging the join order of the two lowest level relations)



4.
select Nation: B(Nation)=1
select Line: B(Line)= $2*10^7*20/2000=2*10^5$

Plan3 is looking most promising:
. join Order, RI: T(.)*(1+1)=4000
. join Customer, RI: T(.)*(1+1) =4000
. join Nation, M: 2B(.)=4000*52/2000=104

Plan1,4 exceed Plan1 because of sorting Customer.
Plan 2,5 exceed Plan 1 because of reading Order.

3 Reducing SQL Queries to Bag Algebra, Part I(20 points)

1.SELECT department, SUM(salary) FROM Employee GROUPBY department

$$\gamma_{department;SUM(salary)}Employee$$

or

$$\gamma_{department;SUM(salary)}$$
$$|$$
$$Employee$$

2. SELECT department, SUM(salary) FROM Employee GROUPBY department HAVING AVERAGE(salary)>100

$$\pi_{department,SS}\sigma_{AS>100}\left(\gamma_{department;SUM(salary)\rightarrow SS,AVG(salary)\rightarrow AS}Employee\right)$$

or

$$\pi_{department,SS}$$
$$|$$
$$\sigma_{AS>100}$$
$$|$$
$$\gamma_{department;SUM(salary)\rightarrow SS,AVG(salary)\rightarrow AS}$$
$$|$$
$$Employee$$

3. SELECT department, SUM(salary) FROM Employee GROUPBY department
HAVING AVERAGE(salary)>100 AND COUNT(*) > 10

$$\pi_{department,Ss}\sigma_{AS>100AAC>10}\left(\gamma_{department;SUM(salary)\rightarrow SSAVG(salary)\rightarrow AS,COUNT(*)\rightarrow AC}Employee\right)$$

$\pi_{department,Ss}$

|

$\sigma_{AS>100AAC>10}$

|

$\gamma_{department;SUM(salary)\rightarrow SSAVG(salary)\rightarrow AS,COUNT(*)\rightarrow AC}$

|

$Employee$

4 Concurrency Control and Recovery (18 points)

1. Write a 2PL schedule that is not recoverable

T_2 and T_3 read a, b, c, which T_1 writes. Therefore to create a not recoverable schedule, we could simply put T_1 before T_2T_3 and put $Commit_1$ after $Commit_2Commit_3$.

$T_1T_2T_3Commit_2Commit_3Commit_1$

a	b	c	commit
R1			
	R1		
W1			
	W1		
		W1	
		R2	
	R2		
	W2		
		R3	
R3			
		W3	
			Commit2
			Commit3
			Commit1

2. Write a 2PL and recoverable schedule that does not avoid cascading rollback.

We can make a small change to the previous schedule by moving $Commit_1$ before $Commit_2Commit_3$. So T_2T_3 commit after T_1 commits (recoverable) yet T_2T_3 read values written by T_1, which is not committed yet (not ACR).

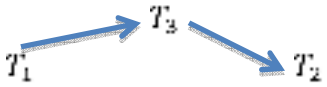
$T_1T_2T_3Commit_1Commit_2Commit_3$

a	b	c	commit
R1			
	R1		
W1			
	W1		
		W1	
		R2	
	R2		
	W2		
		R3	
R3			
		W3	
			Commit1
			Commit2
			Commit3

3. Write a strict 2PL schedule that is not serial.

$r_1(a), T_2, \text{Commit}_2, T_1, \text{Commit}_1, T_3, \text{Commit}_3$

5 Concurrency Control (10 points)
In every serial schedule that is conflict equivalent to S, T2 precedes T1 =>



Therefore we need to insert an action of T3 in or before T1 (T1->T3) and an action of T3 in or after T2 (T3->T2) without generating other arrows:

$w_3(A)r_1(A)r_1(C)w_1(A)r_2(C)r_2(B)w_2(B)w_3(B)$

(Other options are replacing $w_3(B)$ with $r_3(B)$ or inserting $r_3(A)$ anywhere before $w_1(A)$ instead of inserting $w_3(A)$)