Q3. (14 pts.) Consider two relations $R(a, b)$ and $S(b, c)$ with the following statistics:

$T(R) = 10{,}000$, $B(R) = 1{,}000$ (each block contains 10 tuples),
$V(R, b) = 200$ (number of distinct values of attribute b in R),

$T(S) = 6{,}000$, $B(S) = 1{,}500$ (each block contains 4 tuples),
$V(S, b) = 100$ (number of distinct values of attribute b in S),
$V(S, c) = 20$ (number of distinct values of attribute c in S) and for all c values, $c > 100$.
Also, we assume the number of available memory blocks is M = 102.

Answer the following questions:

(i) Estimate the number of tuples in $\sigma_{c=150}(S)$ (2 points)

$\frac{6000}{20} = 300$ tuples

(ii) Estimate the number of tuples in R $\bowtie$ $(\sigma_{c>25}(S))$. (2 points)

$6000 \times 1000 \times \frac{1}{100}$

(iii) Suppose we have a clustered B+tree index available for attribute b of S. The tree has 3 levels, and each leaf node contains 4 records. Assume each node of the index occupies one block. Estimate the cost of executing the following query: (the cost is measured by disk I/Os for accessing the index and the records). (3 points)

SELECT *
FROM S
WHERE $b = 100$ OR $b = 1000$

$\frac{60}{4} = 15$ $\frac{6000}{100} = 60$

$2 + 15 + 2 + 15 = 34$ I/Os.

(iv) Consider joining R and S using a block nested loops join (without using any index). Suppose R is used as the outer loop. Estimate the cost. (3 points)

$R \bowtie S$

$1000 + \frac{1000}{100} * 1500 = 1000 + 15000 = 16000$

(v) Now suppose we want to use sort-join. Assuming that the number of tuples with the same b value is not large, we aim to use the following sort-based join approach:

1. Created sorted sublists of size M, using b as the sort key, for both R and S.
2. Bring the first block of each sublist into the memory buffer.
3. Repeatedly find the least b-value, and output the join of all tuples from R with all tuples from S that share this common b-value. If the buffer for one of the sublist is exhausted, then replenish it from disk.

Estimate the cost (total disk I/Os) of applying this sort-join on R and S. (4 points)

$3(M+N) = 3(1000 + 1500) = 7500$ I/Os.

Q4. (17 pts.) Suppose we want to compute the following:

$\delta_b(\ R(a,b) \bowtie S(b,c) \bowtie T(c,d))$, where δ_b specifies sorting on b.

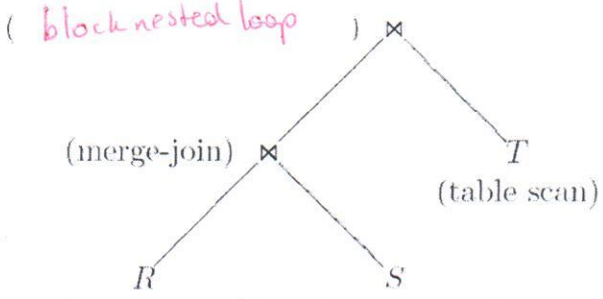
That is, we want to join three relations R, S, and T, and sort the results on attribute b. Let us make the following assumptions:

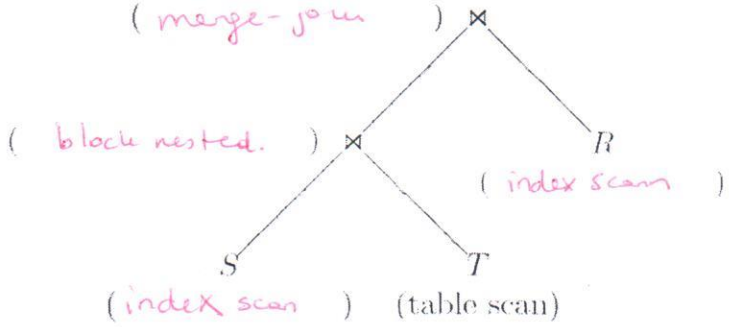
- First, for accessing each relation:
  - R can be index scanned on attribute a or b, or table scanned.
  - S can be index scanned only on b, or table scanned.
  - T can only be table scanned.

  We assume the index is based on B+tree.

- Second, for joining two relations, we have the following two choices:
  - merge-join can be used if the two relations are already sorted on the join attribute.
  - Block-nested-loop join can be used in any case.

(i) Consider the following two logical query plans, where some physical access and join methods are labeled already. Note that only the join part (and not sorting) of the query is shown.





Suppose we would like to avoid applying an additional sort on the final output relation, while still achieve the desired sorted result. Fill appropriate access and join methods into the brackets in the above two query plans. (5 points)

(ii) An interesting order holds for a join result if it is sorted in an order that will be useful for the operations in the higher part of the expression tree. For our problem, we want the final result sorted on attribute *b*. Consider the following three plans for R ⋈ S. Circle "yes" if the plan produces an interesting order, and "no" if not. (3 points)

Plan	R ⋈ S	interesting order?
Plan A	(table-scan R) nested-loop-join (table-scan S)	yes / no
Plan B	(index-scan R on *b*) merge-join (index-scan S on *b*)	yes / no
Plan C	(index-scan R on *b*) nested-loop-join (table-scan S)	yes / no

Briefly explain your choices. (3 points)

(iii) For query optimization, we use an enhanced method that improves upon the dynamic programming approach: It will keep for each subexpression the plan of lowest cost (as dynamic programming does). In addition, it will also keep the one with lowest cost from those plans that produce an interesting order.

The table below lists the estimated costs for the three plans in part (ii). Which plans would be kept when considering the subexpression R ⋈ S using this enhanced method? Circle "yes" if the plan will be kept, and "no" if it will be pruned.

Plan	estimated cost	keep?
Plan A	2000	yes / no
Plan B	3000	yes / no
Plan C	4000	yes / no

Briefly explain your choices. (3 points)

Q5. (12 pts.) Consider a database with data items {A, B, C, D}. The system has the following log records. Note that an entry <T, X, old, new> means transaction T changes the value of X from old to new. In the following log, DPT represents the Dirty Page Table and TT represents the Transaction Table.

1. <T2 start>
2. <T2, B, 10, 11>
3. <T1 start>
4. <T2 commit>
5. <T1, A, 20, 21>
6. <begin checkpoint>
   DPT = (A, 5)
   TT = (T1, running, 5)
   <end checkpoint>
7. <T3 start>
8. <T3, C, 30, 31>
9. <T4 start>
10. <T4, B, 11, 41>
11. <T4 commit>
12. <T3, D, 50, 51>
13. <T1, C, 31, 32>
14. <T5 start>
15. <T5, D, 51, 52>
16. <T3 commit>
17. <T6 start>
18. <T6, A, 21, 33>
19. <T5 commit>
    System failed

a) What is the smallest Log sequence number (LSN) accessed during the Analysis phase? (1 pt.)
Answer: 6

b) Fill in the contents of the **Dirty Page Table** and the **Transaction Table** at the end of the Analysis phase. (5 pts.)

PageID	RecLSN
A	5
C	8
B	10
D	12

XID	Status	LastLSN
T1	running	13
T3	commit	16
T4	commit	11
T5	commit	19
T6	running	18

c) List all possible values of A, B, C and D. That is, what are the possible data values on the disk at the point of failure (after action 19)? Assume 3 buffer pages are available. (3 points)

Memory A = 33, C = 32, B = 41

Disk A = 21, B = 41, C = 30, D = 50

d) What are the values of A, B, C, D after recovery? (3 points)

A = 20
B = 11
C = 31