Solver: Jesslyn Chew

1)

a)

i) **True**. This is possible as table tuples can have variable format and made possible with the use of record headers to indicate the length and type of record.

ii) **False**. The entire block must be read even if only 1 record is required, so the I/O cost will still be 1 if only reading od 1 record is required.

iii) **False**. There can be multiple secondary / unclustered index, which are dense index.

iv) **True**. There can only be 1 primary index that is also a clustered index. Only clustered index can be sparse, hence there can only be 1 sparse index on a database table.

v) **True**. A projection operation still needs to access all the records, whereby it selects particular attributes of the records instead of all the attributes. Hence, it does not reduce disk I/O cost and in fact increases I/O cost due to the overheads from accessing the index file.

b) Clustered index stores the tuples in the data file in the same order as the entries in the index, whereas unclustered index stores tuples in the data file that are not sorted in the same order as the entries in the index. Clustered index can use sparse and dense index for the 1st level, while the unclustered index can only use dense index for the 2nd level. Clustered index are better for range queries than unclustered index, but unclustered index is faster with updates and insertion than clustered index.

c) Step 1: Find the total number of blocks to store database table S

Record size = # of tuples * tuple size
= 1000 * 17 bytes
= 17,000 bytes

Here we assume that 1KB = 1000 bytes and not $2^{10}$

# of blocks = $\lceil 17{,}000\ bytes \div 4KB \rceil$
= $\lceil 17{,}000 \div 4{,}000 \rceil$
= $\lceil 4.25 \rceil$
= 5

Step 2: Calculate the I/O Cost

Since prefetching algorithm is used,

I/O cost = R + P + (# of blocks – 1) * max (R, P)
= R + P + 4P [since P>R]
= **R + 5P**

2)

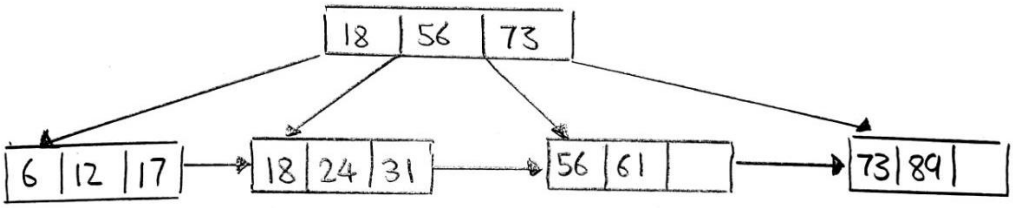
a)

i) I/O cost = # of levels, let # of levels be n
On the leaf nodes, need to have a pointer for each tuple.
# of keys per node = 255 * 0.8 = 204
$(204 + 1)^{n-1} * 204 \geq 1{,}000{,}000$
➔ n = 3
I/O cost = 3

ii) Hashing is the idea of taking the search key as an input and computing an integer using that input and a modulus function. The output indicates which bucket it must the tuple be

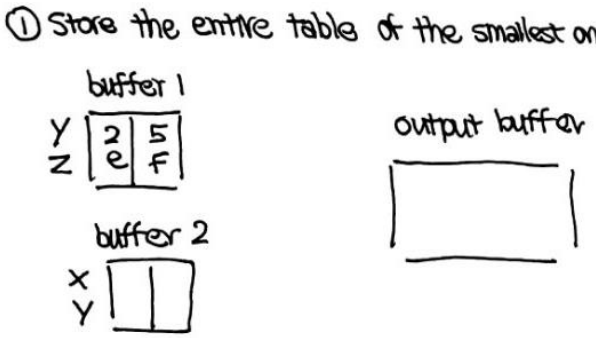
inserted in. Static hashing has a fixed number of buckets and the size of the bucket may vary. Whereas dynamic hashing allows the number of buckets to vary but the bucket size is fixed.

iii) Extensible hash index – varied number of buckets, where each bucket is 1 block
Take note that for hashing, the bucket contains the actual record rather than the pointers
# of blocks for records = 1,000,000 ÷ (11*0.8) ≈ 113,637
Size of directory (array of bucket pointers) = 113,637 ÷ (255*0.8) ≈ **558**
*[Not very certain of answer]*
If the directory must be accessed sequentially and cannot jump immediately to the correct hash value, the I/O cost is size of directory + 1 = 559.
If the database can jump to the correct hash value, the I/O cost is 2.

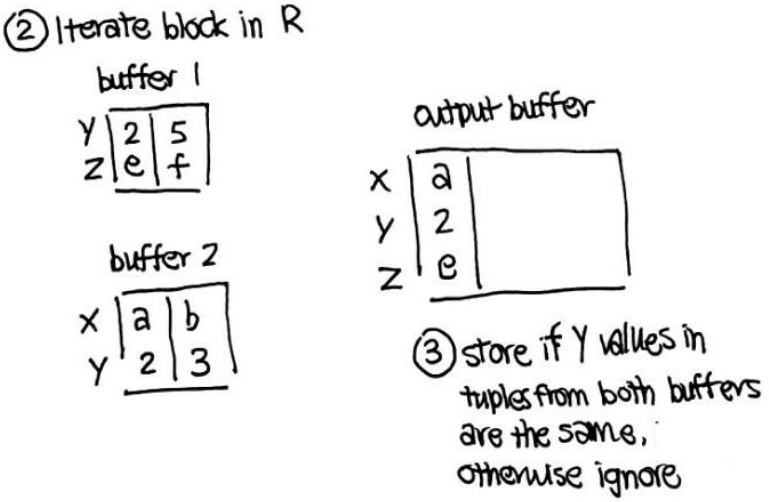


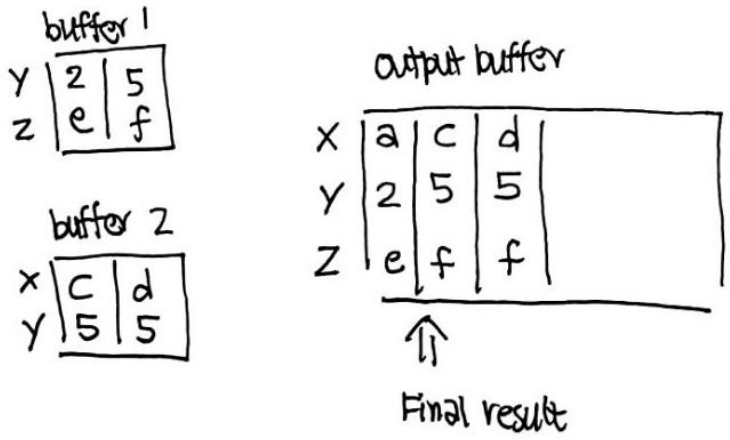

b)

Ensure that it is sorted and that the B+ tree does not violate the minimum key rule.




c)






3)

a)

i) Cost of refined sort merge join = 3(B(R) + B(S))
= 3(1000 + 500)
= **4500**

b) For each of the relation, sort the tuples using sort-merge. This is done reading every M pages of the table into memory and sorting them using memory-based sorting algorithm. Each sorted set of pages (run) is written back to disk. Afterwards, each input block is allocated to each run and sorted again. When the sorting is done for both relations, the join is executed. Step through the 2 sorted tables in parallel and output matching tuples of the same x value.

Step 1: Find cost for sort-merge = 3(B(R) + B(S))

B(R) = 30,000 ÷ 30 = 1,000

B(S) = 9,000 ÷ 10 = 900

Cost for sort-merge = 3(1000 + 900) = 5700 I/O Costs

Step 2: Find cost for join

Cost for $x_1 = \left(\frac{3000}{30}\right) + \left(\frac{200}{10}\right) = 120$

Cost for $x_2 = \left(\frac{6000}{30}\right) + \left(\frac{3000}{10}\right) = 500$

Cost for $x_3 = \left(\frac{9000}{30}\right) + \left(\frac{3000}{10}\right) = 600$

Cost for $x_4 = \left(\frac{1800}{30}\right) + \left(\frac{200}{10}\right) = 80$

Cost for $x_5 = \left(\frac{1200}{30}\right) + \left(\frac{600}{10}\right) = 100$

Cost for $x_6 = \left(\frac{9000}{30}\right) + \left(\frac{2000}{10}\right) = 500$

Total Cost = 120 + 500 + 600 + 80 + 100 + 500 + 5700 = **7600 I/O Costs**

c)

i) T(R) ÷ selectivity factor = 100 ÷ 3 ÷ 10
= 3.333
= **4**

ii) $T(R) * \left(1 - \left(1 - \frac{m_1}{T(R)}\right) * \left(1 - \frac{m_2}{T(R)}\right)\right) = 100 * \left(1 - \left(1 - \frac{1}{10}\right) * \left(1 - \frac{1}{50}\right)\right)$

$= 11.8$

$\approx \mathbf{12}$

iii) T(R) * T(S) ÷ max (V(R,d), V(S,d)) ÷ max (V(R,b), V(S,e))
= 100 * 500 ÷ max (50,30) ÷ max (10,100)
= 100 * 500 ÷ 50 ÷ 100
= **10**

d)

i) # of tuples = (10*30 + 40*100 + 100*200) ÷ 30 = **810**

ii) # of tuples = T(R) * T(S) ÷ max(V(R,b), V(S,b))
= 300 * 600 ÷ 30
= **6000**

4)

a)

i) For UNDO, output(X) must be performed before COMMIT
Hence, for T1 the latest time is before logID 8 (7). For T2, the latest time is before logID 10 (9).

ii) Checkpoint after logID 6: <START CKPT(T1, T2, T3)>
End checkpoint after logID 14: <END CKPT>

iii) For UNDO, need to undo the transactions that are not committed in the reverse order as it is from bottom-up.
After logID 13, active and not committed transactions are T3 and T4.
**Ans: LogID 13, 12, 9 – <T4, A, 40>, <T3, D, 35>, <T3, C, 30>**

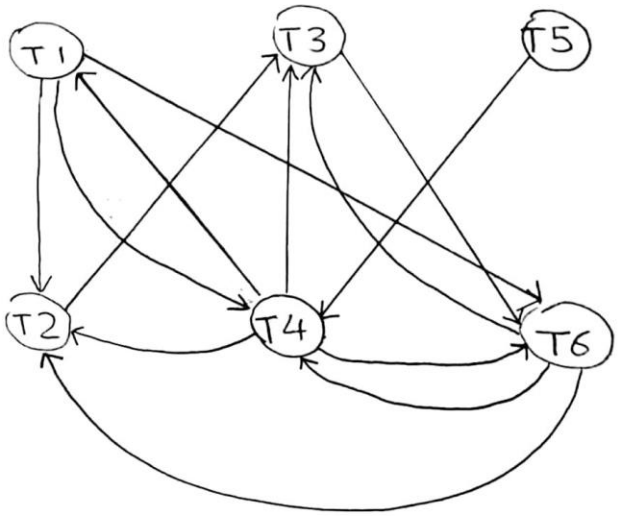
iv) For REDO, output(X) is performed after COMMIT
Hence, the earliest time for T3 is after logID 14 (15). For T4, the earliest time is after logID 16 (17).

v) For REDO, the transactions that are committed after the checkpoint are redone.
Transactions committed after checkpoint: T2, T3
**Ans: LogID 4, 5, 9, 12 – <T2, B, 15>, <T2, C, 20>, <T3, C, 30>, <T3, D, 35>**

b) Deadlock occur if 2-phase locking protocol is used, since 2 2-phase locking don't allow releasing of locks before obtaining any locks. Each transaction needs the lock that is held by the other transaction and due to 2PL, it cannot release those locks.

| **T1** | **T2** |
|---|---|
| $l_1(X)$; $r_1(X)$ | |
| | $l_2(Y)$; $r_2(Y)$ |
| | $l_2(X)$ DENIED |
| $l_1(X)$ DENIED | |

c) S is not serializable as it contains a few cycles. Some cycles are T1→T4→T1, T2→T3→T6→T2.




--End of Answers--