1)

a)

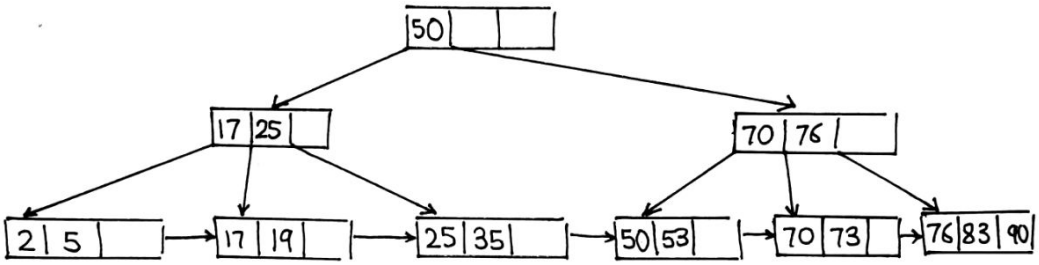
i) Sequential I/O reads data blocks in a consecutive manner, where the next data block to be read is the one stored physically adjacent to it. Whereas the random I/O reads data block without some form of order.

ii) 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. A table can have only one clustered index, but it can have multiple unclustered index.

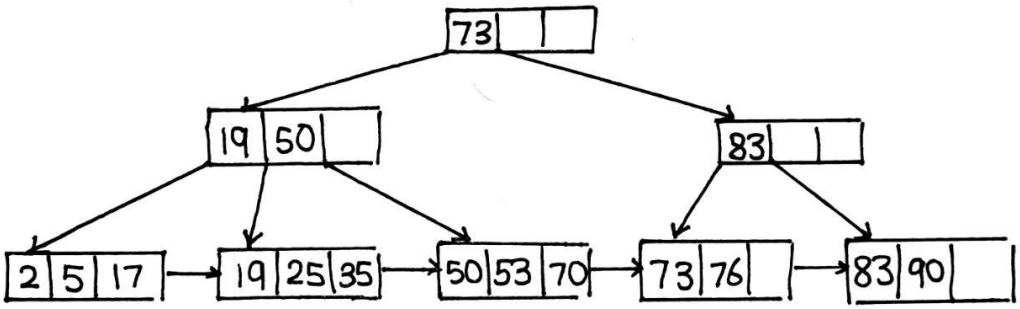
b) Since prefetching algorithm is used,
I/O cost = R + P + (# of blocks – 1) * max (R, P)
= **10R + P** [since R > P]

c)

i)



ii)



d)

i) # of blocks for dense index = 100,000 ÷ 100 = **1,000 blocks**
# of blocks for sparse index = 100,000 ÷ 20 ÷ 100 = **50 blocks**

ii) Single level spare index → 50 index blocks
To find a record using binary search: $\log_2 50 = 6\ disk\ I/O$
To retrieve the record = 1 disk I/O

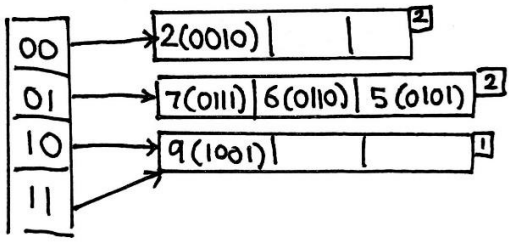
Total = 6 + 1 = **7 disk I/O**

iii) Since we have 100,000 tuples, the last level needs to have 100,000 pointers (excluding incremental pointers)
# of keys per node = 100
# of pointers per node = 101
$101^{n-1}(100) \geq 100{,}000$
$101^{n-1} \geq 1000$
$n \geq 2.49 \approx 3$
There are 3 layers
# of blocks on level 1 = 1
# of blocks on level 2 = 101
# of blocks on level 3 = 100,000 ÷ 100 = 1000
Total number of blocks = 1 + 101 + 1000 = **1102 blocks**

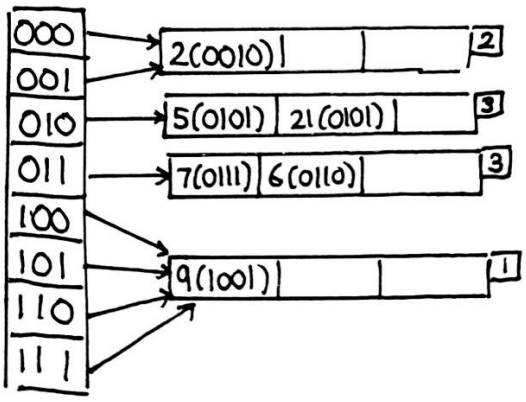
2)

a)

i)



ii)



b) 1. Index on (name, salary), unclustered, B+ tree index. Salary is a range query, using B+ tree would speed up the query process.
2. Index on (zipcode), unclustered, hash-based index. Zipcode is a value-base query, hash based index would speed up the query process.

c) Number of leaf nodes = 1,000,000 / 50 = 20,000
Number of keys in each index node = 200 * 0.8 = 160
Number of levels in the tree = $\log_{160} 20{,}000 = 2$

To retrieve a tuple with a given key value = 2 + 1 = 3 disk I/)
(Assume that the root node is not in memory)

d)

i) For each data block, use 1 memory block to read it into memory
    For each tuple in the block
        If the tuple belongs to one of the exiting groups
            Add the tuple to the memory block of the same group
        Else
            Add the tuple to another memory block
        End-If
    End-For
End-For

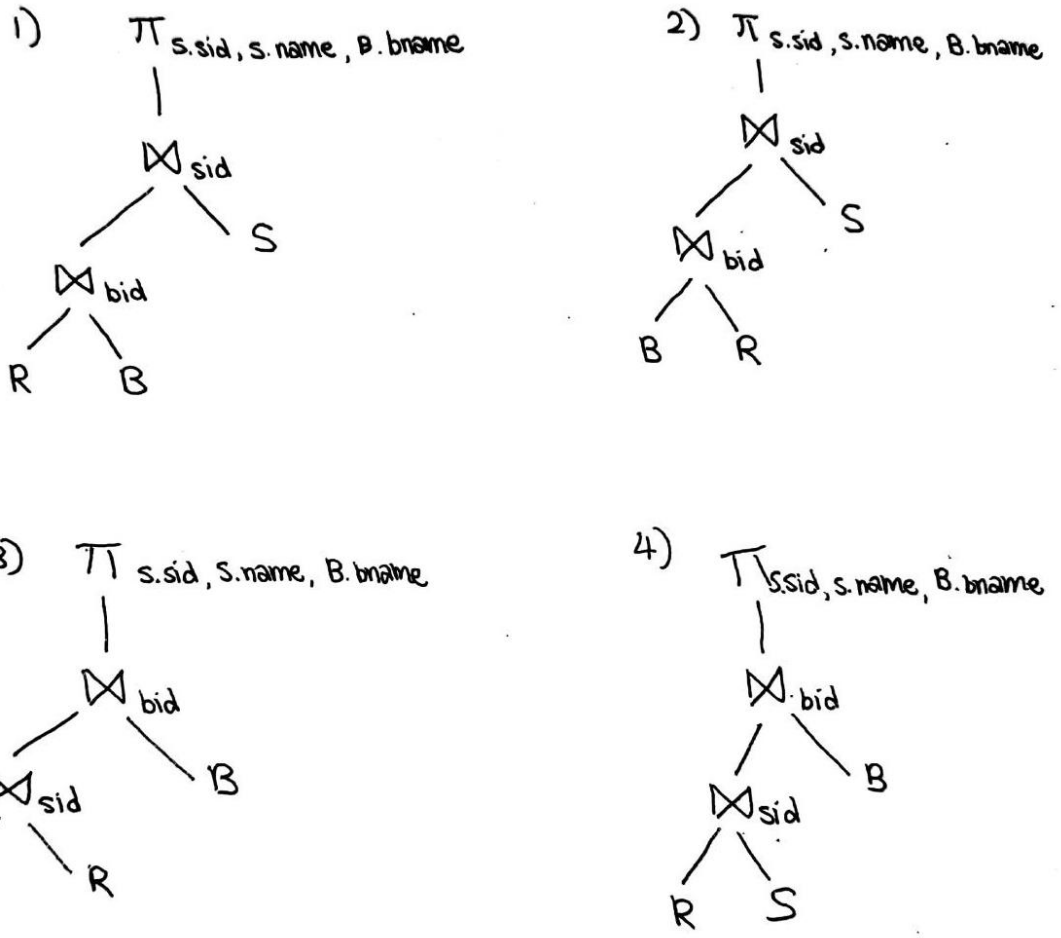
ii) Block-based nested loop join algorithm:
Assume there are m blocks in memory
For each block in R, use 1 memory block to read it into memory
    For m-1 blocks in S, use m-1 memory blocks to read them
        For each r in R
            For each s in S
                If r join s, add the join to the output block
                End-If
                If the output block is full, write it onto the disk
                End-If
            End-For
        End-For
    End-For
End-For
Difference: one pass algorithm only reads the data once and it requires one of the table to be able to fit into the memory. However, block-based nested loop join algorithm reads one table once but the other table for multiple times and it does not have any requirement on the sizes of the tables.

3)

a)

i) Number of tuples in selected B = 100 * 10 / 3 = 333
Number of tuples in selected R = 10,000 * 40 / 500 = 800
Number of tuples in S = 1,000 * 20 = 20,000
Number of tuples after joining tables = 333 * 800 * 20,000 / 800 / 20,000

ii)



iii) For 1), blocked-nested loop join. T(R) is too big to use hash join or sort-merge join. Nested-loop join does not have any requirements on the sizes of the 2 tables.
For 2), block-based nested loop join. B contains 100 blocks and memory buffer has 50 blocks, we can use block-based nested loop join to hold 33/34 blocks of B in memory and pass R into memory part by part. This algorithm can complete the join by passing R into memory 3 times and B only once. Total I/O cost = 10,000 * 3 + 100 = 30,100 blocks.
For 3), Hash join. T(S) = 1,000 < M * (M – 1) = 2,450. Hence, hash join can be applied here. I/O cost = 3 * (1,000 + 10,000) = 33,000.
For 4), block-nested loop join. T(R) is too big to use hash join or sort-merge join. Nested-loop join does not have any requirements on the sizes of the 2 tables.

b) two-pass multi-merge sorting algorithm:
Assume there are m blocks in memory buffer
First pass: read (m – 1) blocks to sort them in memory
Second pass: read m blocks to merge the sorted sublists
(m – 1) * m <= 25
m >= 6

memory requirement: at least 6 blocks in memory buffer
I/O cost: 25 * 4 = 100 disk I/Os (assume data are read from disk and the final sorted data are written back to disk)

4)

a)

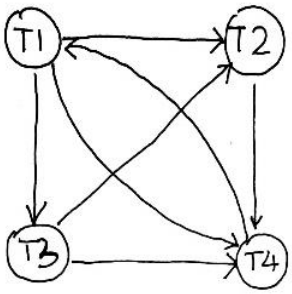
i) At system failure, changes made by T2, T4 are written to disk. Whereas changes made by T1, T3, T5, T6 may not be written to disk. This results in the following possible values:
A = 20,21
B = 41
C = 30,31,32,33
D = 50,51,52

ii) Need to undo transactions that are not committed, which are T1 and T6. logID 20, 15, 5.

iii) Need to redo transactions that are committed after checkpoint, which are T3 and T5. logID 8, 13, 17.

iv) A = 20, B = 41, C = 31, D = 52.

b) It is not conflict serializable as there is a cycle between T1 and T4.



c) wait-for graph:

5) T1	T2	T3	T4
$l_1(A)$; $r_1(A)$			
	$l_2(B)$; $r_2(B)$		
$l_1(c)$; $w_1(c)$			
	$l_2(D)$; $w_2(D)$		
		$l_3(c)$ DENIED	
$l_1(B)$ DENIED			
			$l_4(D)$ DENIED
	$l_2(A)$ DENIED		

A deadlock occurs, as T1 and T2 need to access an element that is currently locked by the other transaction.

--End of Answers--

Solver: Du Youwei