Solver: Jesslyn Chew

1)

a)

i) **True**. Disk access time is affected by seek time, rotational delay, and transfer time. Transfer time is the time taken to transfer the data of a block. With the block size increased, transfer time increases which in turn also increase disk access time.

ii) **False**. The length of records may not always be the same since tuples can different attribute values, so as long as the record tracks the attribute size and type in the record header.

iii) **False**. This is not true for sequential flooding. For example, let there be 2 frames and the pages are accessed in the following order: 1,2,3,1,2,3. During the 3rd access, frames contain page 1 and 2 and needs to access page 3. By LRU, page 1 is evicted but it is required for the next access. Hence, the least recently used page is not the least important and should not be evicted.

iv) **True**. The dense index helps to store the information (key and pointer to actual data location) on the records that can fit into the main memory. Since the first level already stores this information, having a 2nd level or higher dense index that has a 1-to-1 pointer to the lower level is proved pointless. Not only does it not help to lower search cost, it increases the search cost due to the overhead of accessing the additional dense index.

v) **True**. If n is the maximum number of keys, the number of pointers is (n+1). The maximum number of child nodes the root node can have is the same as the number of pointers. It can also have no child nodes. Therefore, the root node may have 0, 1, …, n+1 child nodes.

b)

i) block size = 4096 bytes, block header has 10 4-byte integer
Each record has a 4-byte real, a 25-byte character string, a 4-byte integer, a 4-byte date
Each record header has 4 4-byte pointers and a 1-byte character
Need to round up to closest multiple of 8.
Record length = 8 + 32 + 8 + 8 + 4*8 + 8 = **96 bytes**

ii) block size without header = 4096 – 10*8 = 4016 bytes
# of records = 4016 ÷ 96 = 41.833 ≈ **41** (round down)

c)

i) Yes.
IO Cost = B(R) + B(S) = 100 + 200 = 300

ii) Recall that I/O cost = ⌈B(X) / (M-1)⌉ * B(Y) + B(X), if X is the outerloop
If R is the outerloop,
I/O cost = ⌈100 / (150-1)⌉ * 200 + 100 = **300 Disk reads**
If S is the outerloop,

I/O cost = ⌈200 / (150-1)⌉ * 100 + 200 = **400 Disk reads**

2)

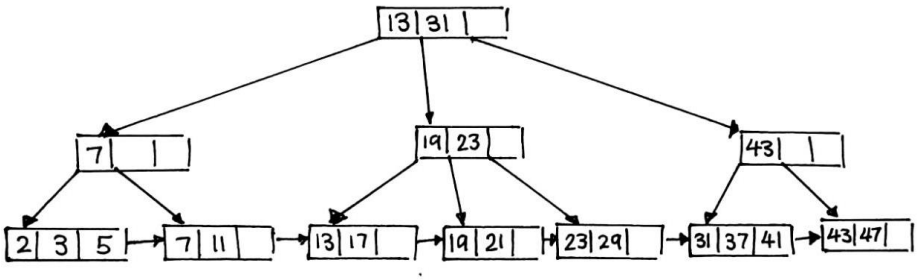
a)

i) Maximum number of keys in a node is **3**.

ii) Since a dense index is used, the number of records is the total number of keys in the leaf nodes, which is 3+2+3+2+3+2 = **15 records**.

iii) Since B+ tree is stored on the disk. Number of I/O is the number of levels is **3** if the keys are assumed to be unique. If the key is not unique, the I/O reads is **4** as the adjacent node needs to be checked as well.

b)

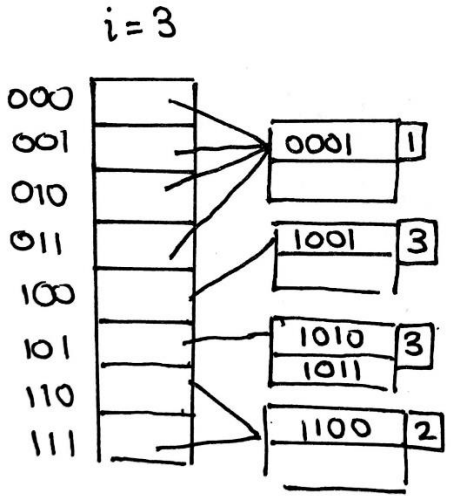


c)

i) block size = 4096 bytes
Let k be the number of keys
$4k + 8*(k+1) \leq 4096$
$k \leq 340.67$
$k = 340$
Ans: **The maximum number of keys is 340, while the maximum number of pointers is 341**

ii) For the B+ tree, the leaf nodes must have 1,000,000 pointers (excluding sequential pointers) at least. Dense index is used, which implies that 1 pointer points to 1 record.
$1000000 \leq k(k+1)^{n-1} = 340(341)^{n-1}$
$\lg(2941.18) \leq (n-1)*\lg(341)$
$n-1 \geq \frac{\lg(2941.18)}{\lg(341)}$
$n \geq 2.3695$
$n = 3$, hence the B+ tree has **3 levels**.

d)



3)

a)

i) T(R) = 5000, B(R) = 5000 ÷ 100 = 50, T(S) = 2000, B(S) = 2000 ÷ u
M = 10
Space requirement for two pass refined sort-merge join: B(R) + B(S) ≤ M^2

$$50 + \frac{2000}{u} \leq 10^2$$

$$u \geq \frac{2000}{50} = \mathbf{40}$$ **tuples per block**

ii) B(S) = 2000 ÷ 40 = 50
I/O Cost of 2 pass refined sort-merge join = 3(B(R)+B(S)) = 3(50+50) = **300**

iii) I/O Cost of block-based nested-loop join

$$= \left\lceil \frac{B(R)}{M-1} \right\rceil * B(S) + B(R)$$
$$= \left\lceil \frac{50}{10-1} \right\rceil * 50 + 50$$
= **350**

iv) I/O Cost of grace hash join = 3(B(R)+B(S)) = 3(50+50) = **300**

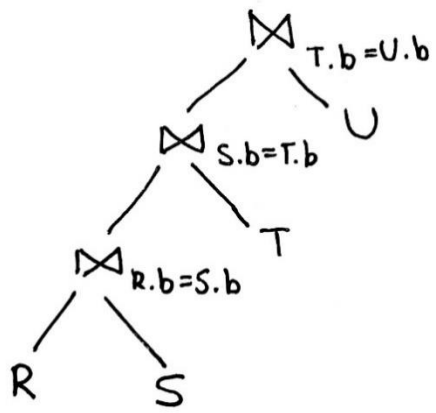
v) Based only on I/O cost, I would choose either two pass refined sort-merge join or grace hash join since both have the same cost. Other considerations may be size of main memory. For two pass refined sort-merge join, the space requirement is B(R) + B(S) ≤ M^2. Whereas grace hash join space requirement is min(B(R), B(S)) ≤ M^2. Since the main memory space requirement for grace hash join is lesser, grace hash join is preferred. Another consideration is the query. If the query requires the results to be sorted (e.g. ORDER BY, or GROUP BY), then sort-merge join is preferred.

b) [Not too confident in this answer]
B(Student) = 1000, T(Student) = 10000
B(Book) = 5000, T(Book) = 50000
B(Checkout) = 15000, T(Checkout) = 300000
Simple nested loop
I/O cost = B(Student) + B(Student)B(Checkout) [Let the smaller relation be the outer block]
= 1000 + 1000*15000
= 15 001 000
Intermediate size = T(Checkout) = 300 000
[Foreign key idea, since Student(sid, …) and Checkout(sid, …)]
Tuple-based nested loop
[Assume that intermediate results is in memory due to pipeline model. For each tuple in Book scanned, it will be processed and joined with the tuples in memory]
I/O cost = T(Book)
= 50 000
Intermediate size = T(Checkout) = 300 000

Selection
Note that for "on the fly" does not incur I/O cost.
Intermediate size = 300 000 * (7/18) / 500
= 233.33
≈ **234**
Projection
Does not affect number of tuples.

Cardinality of result = 234
I/O cost = 15 001 000 + 50 000 = **15 051 000**

c)



4)
a)

i) Before 14 LogID. [In UNDO, the output is written to the disk latest before the commit statement]

ii) B = 7

iii) Start checkpoint between LogID 9 and 10. The log entry should look like <START CKPT (T1, T2)>
There is no earliest time in the log record to end the checkpoint as <T2 COMMIT> needs to be present. The checkpoint can only end after <T2 COMMIT>. The log entry should look like <END CKPT>

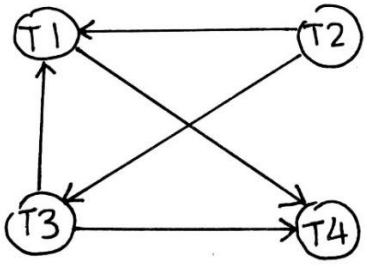
iv) Transactions T1, T2 and T4 needs to be undone as transactions that are yet to be committed after the crash needs to be undone. The actions undone are of the following order: LogID 13, 12, 10, 8, 6, 2.

v) It indicates that the transaction T4 has modified the variable J such that the new value is 13.

vi) The system needs to redo committed transactions, which are T1. The actions redone are of the following order: LogID 2, 13.

b)

i) It is serializable as the precedence graph is acyclic.



ii) T2 → T3 → T1 → T4

iii) The schedule is not 2PL. In transaction T4, “Unlock(A)” is after “S-lock(D)”. In every transaction, all lock actions precede all unlock actions for 2PL. Therefore, since an unlock action precede lock actions, it is not 2PL.

--End of Answers--