2. (20 pts) Selection Cost Estimates

In terms of $t_s$, $t_T$, $n$ (number of records fetched), $b$ (number of blocks containing fetched records), and $h_i$, give estimated cost.

We will only grade answers completely in the boxes.

(a) Primary $B^+$-tree index, equality on key:

**answer:** $(h_i + 1) * (t_T + t_s)$

(b) Primary $B^+$-tree index, equality on non-key:

**answer:** $h_i * (t_T + t_s) + b * t_T$

(c) Secondary $B^+$-tree index, equality on key:

**answer:** $(h_i + 1) * (t_T + t_s)$

(d) Secondary $B^+$-tree index, equality on non-key:

**answer:** $(h_i + n) * (t_T + t_s)$

3. (10 pts) Write **yes** or **no** in the box according to whether the corresponding lock types are compatible. You will not receive points for anything besides “yes” or “no”.

(a) IS, SIX

**answer:** yes

(b) SIX, S

**answer:** no

(c) IX, IX

**answer:** yes

(d) S, X

**answer:** no

(e) SIX, X

**answer:** no
4. (20 pts) **Concept matching.** Write the closest match from the above table of phrases into the corresponding box. No concept should be used more than once.

(a) helps to avoid deadlocks  
**answer:** known lock acquire ordering

(b) ensures conflict serializability  
**answer:** two-phase locking

(c) guarantees no dirty reads  
**answer:** strict two-phase locking

(d) can be found with a waits-for graph  
**answer:** deadlocks

(e) is when a transaction never commits  
**answer:** starvation

(f) transactions are favored by wound wait  
**answer:** older

(g) calibrates privacy.  
**answer:** k-anonymity

(h) is the all or nothing nature of transactions  
**answer:** atomicity

(i) supports roll-up and drill-down  
**answer:** data cube

(j) used when search key matches file sort key  
**answer:** primary index
5. (10 pts) Assume timestamp-based concurrency control in the above schedule. Annotate each transaction by striking out (writing a horizontal line through) the instruction that causes that transaction to abort, if there is one. Example: “The fifth word has been struck out”.

answer:

6. (15 pts) Snapshot Isolation. Assume the following schedule of transactions under snapshot isolation (first committer wins). Which would abort, and why?

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
</tr>
</thead>
<tbody>
<tr>
<td>read(Y)</td>
<td>read(X)</td>
<td>write(Y)</td>
<td>write(X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>write(X)</td>
<td>write(Y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>read(Z)</td>
<td>write(Z)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TS(T_5) < TS(T_4) < TS(T_3) < TS(T_2) < TS(T_1)

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>start trans</td>
<td>start trans</td>
</tr>
<tr>
<td>write(A)100</td>
<td>write(B)100</td>
</tr>
<tr>
<td>read(B)?</td>
<td>read(A)?</td>
</tr>
<tr>
<td>commit trans</td>
<td>commit trans</td>
</tr>
</tbody>
</table>
**answer:** T1 would abort because it’s writeset overlaps w/ T2’s, which committed during T1’s lifetime.
7. (15 pts) Classify the above schedules as serializable, not serializable, conflict serializable or view serializable. Select the most specific applicable label. More than one could have the same answer.

(a) (A)  
\textbf{answer: serializable}

(b) (B)  
\textbf{answer: conflict serializable}

(c) (C)  
\textbf{answer: not serializable}
8. (15 pts) Precedence graph

<table>
<thead>
<tr>
<th></th>
<th>T₁</th>
<th>T₂</th>
<th>T₃</th>
<th>T₄</th>
<th>T₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>read(U)</td>
<td>read(Y)</td>
<td>read(X)</td>
<td>read(U)</td>
<td>read(V)</td>
<td>read(W)</td>
</tr>
<tr>
<td>read(U)</td>
<td>read(Z)</td>
<td>write(Y)</td>
<td>write(Z)</td>
<td>read(Y)</td>
<td>read(W)</td>
</tr>
<tr>
<td>write(U)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) (10 pts) Build the precedence graph for the above schedule.
answer:

(b) (5 pts) Is it conflict serializable?
answer: yes
9. (15 pts) Waits-For graph

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S(V)</td>
<td></td>
<td>X(V)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S(W)</td>
<td></td>
<td></td>
<td>S(V)</td>
<td></td>
</tr>
<tr>
<td>S(Z)</td>
<td></td>
<td></td>
<td></td>
<td>S(Z)</td>
</tr>
</tbody>
</table>

(a) (10 pts) Build the waits-for graph for the above schedule.
answer:

(b) (5 pts) Is there a deadlock?
answer: yes

10. (10 pts) Consider the cost of a sort-merge of 12 tuples. Assume each tuple is an entire block, there are only 4 blocks of memory, that only one block is read at a time, and any other common assumption.

(a) How many block transfers would be required?

answer: \[2 \times 12 + 1 \times 12 = 36\]

(b) How many disk seeks would be required?

answer: \[2 \times 3 + 12 \times (2 - 1) = 18\]
11. (25 pts) Short answer.

(a) Why do we usually use physical log records rather than logical log records?
   **answer:** Logical log records are not idempotent, have to know whether something makes it to disk or not in order to undo, or redo.

(b) What is a recoverable schedule?
   **answer:** No (1) dirty reads from a (2) transaction that aborts.

(c) What does “NO FORCE” mean in the context of buffer management?
   **answer:** Tuples not forced to disk before transaction commits.

(d) Why, in detail, is snapshot isolation potentially much faster than serializability?
   **answer:** No locks are held, so no contention.

(e) What is Thomas’s Write Rule, and what is it good for?
   **answer:** Ignore obsolete writes. In timestamp-ordering, ignore a transaction’s final write if it would be overwritten in equivalent serial order. Allows more transactions to commit even if not conflict-serializable.