Question 1.  [12 marks]

Short Answer Question

Part (a)  [2 marks] what are different types of data independence? explain each in one line

Logical data independence: Protection from changes in logical structure of data.
Physical data independence: Protection from changes in physical structure of data.

Part (b)  [3 marks] what is atomicity? explain an algorithm that insures atomicity

DBMS ensures atomicity (all-or-nothing property) even if system crashes in the middle of a Xact.  Idea: Keep a log (history) of all actions carried out by the DBMS while executing a set of Xacts: Before a change is made to the database, the corresponding log entry is forced to a safe location. (WAL protocol; OS support for this is often inadequate.) Write Ahead Log (WAL), if log entry wasnt saved before the crash, corresponding change was not applied to database! After a crash, the effects of partially executed transactions are undone using the log. Write Ahead Log (WAL), if log entry wasnt saved before the crash, corresponding change was not applied to database!
**Part (c)** [2 marks] what’s the difference between static and dynamic hashing? what’s the difference between linear and extendible hashing?

static: Long overflow chains can develop and degrade performance. dynamic: Extendible and Linear Hashing: Dynamic techniques to fix this problem.

check slides but mainly: LH handles the problem of long overflow chains without using a directory, and handles duplicates.

**Part (d)** [2 marks] Briefly explain how Index Nested Loops work?

basically foreach tuple r in R do foreach tuple s in S where ri == sj do add \( r, s \) to result

**Part (e)** [3 marks] What is the cost of hash join for 2 relations with M and N pages? explain briefly

3(M+N) check slides
Question 2.  [12 marks]

Cost Calculation

Fill in the blanks in the following table

- B: The number of data pages
- R: Number of records per page
- D: (Average) time to read or write disk page
- Hash: No overflow buckets, 80% page occupancy
- Tree: 67% occupancy

<table>
<thead>
<tr>
<th></th>
<th>Scan</th>
<th>Equality</th>
<th>Range</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unclustered Tree index</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unclustered Hash index</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 3. [14 marks]

Hash following data entries use both linear hashing and extendible hashing techniques with least significant bits: 8, 16, 24, 32, 40, 48, 56, 64, 128, 7, 15, 31, 63, 127, 1, 10, 4

![Diagram showing linear and extendible hashing](image.png)
**Question 4.** [12 marks]

**Join Algorithms**

The nearest-neighbor join is a variant of the relational equi-join operator. The nearest-neighbor join of R and S on attribute B returns each tuple of R (r) paired with all tuples of S (s) such that \( r[B] \geq s[B] \) and \( r[B] - s[B] \) is the smallest. Formally, if \((r,s)\) is in output of the nearest-neighbor join of R and S on B, then \( r[B] \geq s[B] \) and there does not exist a tuple \( s' \) in S where \( r[B] - s[B] > r[B] - s'[B] \).

Hence, for a tuple of R, if there are tuples of S with the same B value, they will be returned (this is the same as in an equi-join). For a tuple of R r where there are no tuple(s) of S with the same B value, we return the set of tuples in S whose B values are as close as possible to \( r[B] \) and lower than \( r[B] \).

Consider Reserves(sid,bid) and Boat(bid,bname) where bid is not a key for Boat. Assume we have three boats (5,'African Queen'), (10, 'Lebarge'), (12, 'Bluenose'), (12, 'Canadian Dime') and five reservation (s1, 1), (s10, 5), (s2, 9), (s3, 9), (s4, 13). Then Reserves nearest-neighbor join Boat on bid is:

- (s10, 5, 5, 'African Queen')
- (s2, 9, 5, 'African Queen')
- (s3, 9, 5, 'African Queen')
- (s4, 13, 12, 'Bluenose')
- (s4, 13, 12, 'Canadian Dime')

Of course, a tuple of R joins with more than one tuple of S iff all the S tuples have the same value of B. So in our example, (s4, 13) joins with multiple tuples, but they all have the same bid value (12). Notice that (s1,1) is not in the result since there is no boat with bid \( \leq 1 \).

**Part (a) [5 marks]**

**Sort Merge Join** Can you adapt the sort merge join to produce the nearest-neighbor join? Describe essential modifications to the data structures and algorithm. Be precise. If it is not possible to do so, explain why. If it is possible, state whether the I/O cost will be higher or lower than the cost of an equi-join sort merge.

**Solution:** Sort phase is unchanged.

In merge phase maintain two pointers on S, modify inner loop to still check for equality but if check fails back up one entry in S and return r with s-1.

The cost should be the same as the sort merge.

They may have (correctly) commented that size of output is same or larger so if that is taken into account (which we don’t) cost is larger.

**Part (b) [5 marks]**

**Index Nested Loop Join** Can you adapt the index nested loop join algorithm to process nearest neighbor joins? Describe essential modifications to the data structures and algorithm. Be precise. Does the type of index matter? If it is not possible to do so, explain why. Does it matter which relation is outer? Will the I/O cost be different from an index nested loop equi-join? State whether it will be higher or lower and whether this depends on the relative sizes of R and S.

Give the I/O cost of your implementation for Reserves nearest-neighbor join Boats on bid (using sizes, buffer pages, and indices available in previous question – Question 4).
Solution: Permit answer with outjoin (note that left outer join is easy with INL – finds tuples of R that don’t join with S. But to find tuples of S that don’t join with R this is harder, you either have to do a R INL S followed by S INL R or some other trick).

Answer for NNJ

NN join is not symmetric so in R nn S, R must be outer and index must be a B+tree on S. Hash indices can’t be used.

Algorithm is same except when probe fails, we scan backward in index to next closest (lower) value.

Reserves read 100 pages
for each tuple of Reserves 10,000 probe index cost is
height 2 + data entry page + 1 (since bid key)
cost 100 + 10,000 (4) = 40,100

cost is same (might be a bit higher than equi-join since when we have to “back up one” to find next lowest bid, we might occasionally incur an extra I/O.) But it is fine if they didn’t notice this.