NEC Registration Examination

Computer Engineering (ACtE)

Chapter 7: Data Structures & Algorithm, Database System, and Operating System (ACtE07)

Full Sorting/Searching Tables • Normalization Steps • ACID & Scheduling Tables • Memory Management Diagrams • Practice MCQs

Computer Engineering (ACtE)
Electronics and Communication
Telecommunication
Software Engineering
Information Technology

COMPLETE NOTES WITH SOLVED MCQ AND PYQS

EASY EXPLANATION



EASY EXPLANATION

7.1 Data Structures: Lists, Linked Lists and Trees

Data Types, Data Structures and ADTs

Term	Meaning	Example
Primitive data type	Built-in basic type provided by the language	`int, char, float, boolean`
Data structure	A way of organizing and storing data for efficient use	`Array, linked list, stack, tree, graph`
Abstract Data Type (ADT)	A data type defined by its BEHAVIOUR (operations), independent of implementation	`Stack ADT: push/pop, regardless of array or linked-list implementation`

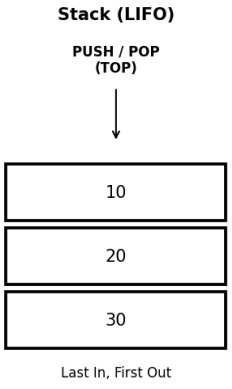
Asymptotic Notations — Time/Space Complexity

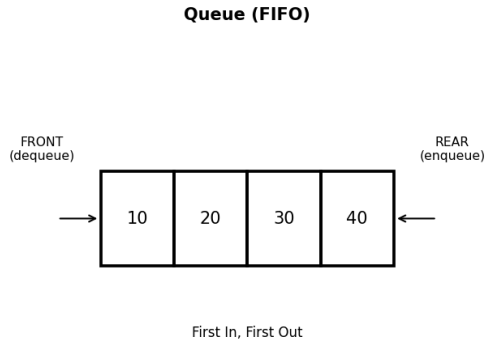
Notation	Name	Represents	Meaning
O (Big-O)	Upper bound	WORST case	`Algorithm never takes MORE than this`
Ω (Big-Omega)	Lower bound	BEST case	`Algorithm never takes LESS than this`
Θ (Big-Theta)	Tight bound	AVERAGE / exact case	`Algorithm's growth is bounded both above and below by this`

Complexity	Name	Example algorithm
`O(1)`	Constant	Array index access
`O(log n)`	Logarithmic	Binary search
`O(n)`	Linear	Linear search, single loop
`O(n log n)`	Linearithmic	Merge sort, Heap sort
$O(n^2)$	Quadratic	Bubble sort, Insertion sort, Selection sort
$O(2^n)$	Exponential	Recursive Fibonacci (naive), subset generation
`O(n!)`	Factorial	Brute-force Travelling Salesman

> **Easy Explanation**
>
> *Memory trick: O = 'at mOst' (upper limit, worst case); Ω = 'at least' (lower limit, best case); Θ = 'exactly/tightly' (both bounds — typically used for average case). When an algorithm's best and worst case coincide, O, Ω and Θ are all the same.*

Stack — Linear Data Structure (LIFO)





Operation	Description	Time complexity
push(x)	Insert x at the TOP	`O(1)`
pop()	Remove and return the TOP element	`O(1)`
peek()/top()	View the top element without removing it	`O(1)`
isEmpty()	Check if the stack has no elements	`O(1)`

Applications: `function call management (call stack), expression evaluation, undo operations, backtracking (maze/DFS), syntax parsing/bracket matching`

Stack Application — Infix, Prefix and Postfix

Notation	Operator position	Example (for A+B)
Infix	BETWEEN the two operands	A + B
Prefix (Polish)	BEFORE the two operands	+ A B
Postfix (Reverse Polish)	AFTER the two operands	A B +

Infix to Postfix conversion algorithm (using a stack):

Step	Rule
Operand encountered	Send DIRECTLY to output
'(' encountered	PUSH onto stack
')' encountered	POP and output until '(' is found; discard both parentheses
Operator encountered	POP and output all stack operators with GREATER OR EQUAL precedence, then push the current operator
End of expression	POP and output all remaining stack operators

Worked example: convert A+B*C to postfix.

Symbol read	Stack (top→right)	Output so far
A	(empty)	A
+	+	A
B	+	A B
*	+ *	A B
C	+ *	A B C
end	(pop all)	A B C * +

Evaluating a postfix expression: scan left→right; push OPERANDS; on an OPERATOR, **pop 2 operands, apply the operator, push the result back**.

Worked example: postfix evaluation

```
Evaluate postfix: 5 3 4 + *
Read 5  → push        stack: [5]
Read 3  → push        stack: [5, 3]
Read 4  → push        stack: [5, 3, 4]
Read +  → pop 4,3 → 3+4=7 → push   stack: [5, 7]
Read *  → pop 7,5 → 5*7=35 → push  stack: [35]
Result = 35
```

Queue — Linear Data Structure (FIFO)

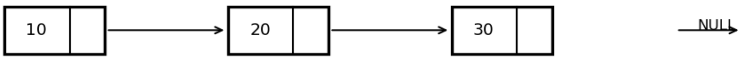
Queue type	Description
Simple Queue	Insert at REAR, delete at FRONT
Circular Queue	Rear wraps around to the beginning, reusing freed space
Priority Queue	Each element has a priority; highest priority served first (often implemented with a heap)
Deque (Double-Ended Queue)	Insertion and deletion allowed at BOTH ends

Applications: CPU/disk scheduling, BFS traversal, print spooling, handling requests in order

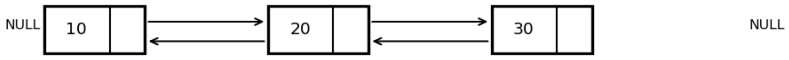
Linked Lists — Detailed Comparison

EASY PLANATION

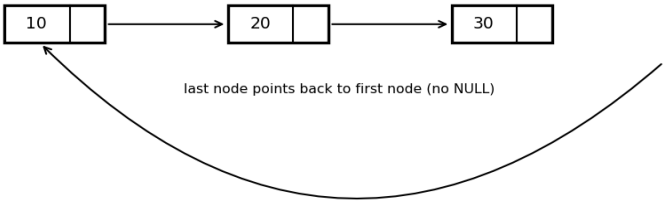
Singly Linked List



Doubly Linked List



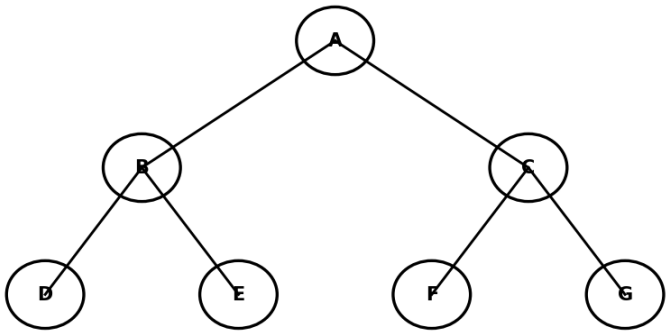
Circular Linked List



Feature	Array	Singly Linked List	Doubly Linked List	Circular Linked List
Memory	Contiguous	Non-contiguous (scattered)	Non-contiguous	Non-contiguous
Access	Random, O(1)	Sequential, O(n)	Sequential, O(n)	Sequential, O(n)
Insertion/Deletion at start	O(n) (shift needed)	O(1)	O(1)	O(1)
Extra memory per node	None	1 pointer (next)	2 pointers (next, prev)	1 pointer (next)
Traversal direction	Either (by index)	FORWARD only	BOTH directions	Forward, and WRAPS around
Size	FIXED (static array) or costly to grow	DYNAMIC — grows/shrinks easily	DYNAMIC	DYNAMIC

Binary Trees and Traversals

Binary Tree Traversals



Pre-order (Root-L-R): A B D E C F G

In-order (L-Root-R): D B E A F C G

Post-order (L-R-Root): D E B F G C A

Traversal	Order	Typical use
Pre-order	Root → Left → Right	`Copying/cloning a tree; prefix expression`
In-order	Left → Root → Right	`Gives SORTED output for a Binary Search Tree`
Post-order	Left → Right → Root	`Deleting a tree; postfix expression`

Traversal	Order	Typical use
Level-order	Level by level (uses a QUEUE)	Shortest path in an unweighted tree; BFS on trees

Tree type	Special property
Binary Search Tree (BST)	Left subtree < node < right subtree, for EVERY node
AVL Tree	Self-balancing BST — balance factor (height difference of subtrees) is -1, 0, or +1 for every node
Complete Binary Tree	All levels full except possibly the last, which fills LEFT to RIGHT
Full/Strict Binary Tree	Every node has 0 or 2 children (never exactly 1)
Heap (Max/Min)	Complete binary tree where every parent is ≥ (or ≤) its children

AVL rotations to rebalance: LL, RR, LR, RL

Max nodes at level i (root=level 0): 2^i | Max nodes in a tree of height h: 2^(h+1) – 1

Probable MCQs: Lists, Linked Lists and Trees

1. Which notation represents the worst-case running time of an algorithm?

a) Big-Omega (Ω)
b) Big-O (O)
c) Big-Theta (Θ)
d) None
✓Answer: b) Big-O (O)
Why: Big-O gives the upper bound on growth — the worst case.

2. The postfix form of A+B*C is:

a) AB+C*
b) ABC*+
c) A+BC*
d) *+ABC
✓Answer: b) ABC*+
Why: Multiplication binds tighter, so B and C combine first: ABC+.*

3. Evaluating the postfix expression '5 3 4 + *' gives:

a) 27
b) 35
c) 15
d) 12
✓Answer: b) 35
Why: 3+4=7, then 5×7=35.

4. Which linked list allows traversal in both directions?

a) Singly linked list
b) Doubly linked list
c) Circular singly linked list
d) Array
✓Answer: b) Doubly linked list
Why: A doubly linked list has both next and previous pointers.

5. In-order traversal of a BST produces elements in:

a) Random order

EASY EXPLANATION

b) Sorted (ascending) order

c) Reverse sorted order

d) Level order

✓Answer: b) Sorted (ascending) order

Why: In-order (Left-Root-Right) traversal of a BST always yields sorted output.

6. In an AVL tree, the balance factor of any node must be within:

a) -1, 0, +1

b) -2 to +2

c) 0 to 5

d) Unlimited

✓Answer: a) -1, 0, +1

Why: AVL trees maintain a balance factor of -1, 0, or +1 for every node.

7. Which traversal uses a queue rather than a stack/recursion?

a) Pre-order

b) In-order

c) Post-order

d) Level-order

✓Answer: d) Level-order

Why: Level-order (BFS-style) traversal processes nodes level by level using a queue.

8. A stack is the natural data structure for evaluating which expression form?

a) Infix only

b) Postfix expressions

c) English sentences

d) SQL queries

✓Answer: b) Postfix expressions

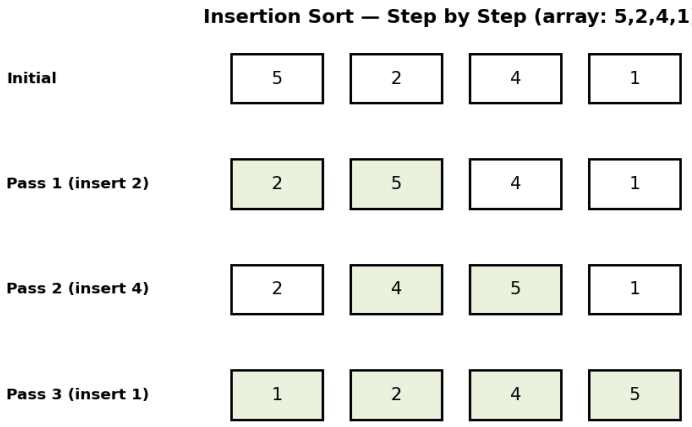
Why: Postfix evaluation pushes operands and pops them for operators — a stack-based process.

EASY EXPLANATION

7.2 Sorting, Searching and Graphs

Sorting Algorithms — Complete Comparison Table

Algorithm	Best case	Average case	Worst case	Space	Stable?	Method
Bubble Sort	O(n)	O(n²)	O(n²)	O(1)	Yes	Exchange (swap adjacent)
Selection Sort	O(n²)	O(n²)	O(n²)	O(1)	No	Selection (find min each pass)
Insertion Sort	O(n)	O(n²)	O(n²)	O(1)	Yes	Insertion into sorted part
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	Yes	Divide and Conquer
Quick Sort	O(n log n)	O(n log n)	O(n²)	O(log n)	No	Divide and Conquer (pivot)
Heap Sort	O(n log n)	O(n log n)	O(n log n)	O(1)	No	Selection via max-heap
Shell Sort	O(n log n)	O(n^1.25) approx	O(n²)	O(1)	No	Gap-based insertion
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)	Yes	Digit-by-digit bucket sort



Easy Explanation

Merge Sort and Heap Sort GUARANTEE O(n log n) even in the worst case, making them safe choices when worst-case performance matters. Quick Sort is usually FASTEST in practice (small constants, in-place) but degrades to O(n²) on already-sorted input unless the pivot is chosen randomly or as median-of-three.

Sorting category	Description	Example
Internal sorting	All data fits in MAIN MEMORY	Quick sort, heap sort
External sorting	Data too large for main memory, uses SECONDARY STORAGE	External merge sort

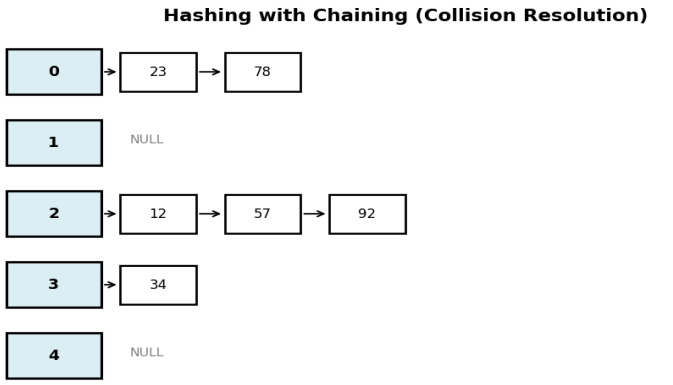
Searching Algorithms

Algorithm	Pre-condition	Time complexity	Approach
Sequential (Linear) Search	None – works on unsorted data	O(n)	Check each element one by one
Binary Search	Array MUST BE SORTED	O(log n)	Repeatedly halve the search range

Algorithm	Pre-condition	Time complexity	Approach
Interpolation Search	Sorted AND roughly uniformly distributed	O(log log n) average	Estimates position using value interpolation
Tree search (BST)	Data organized as a BST	O(log n) average, O(n) worst	Traverse left/right based on comparison

Binary search: compare with mid = (low+high)/2;
if key < arr[mid] search LEFT half, else search RIGHT half

Hashing

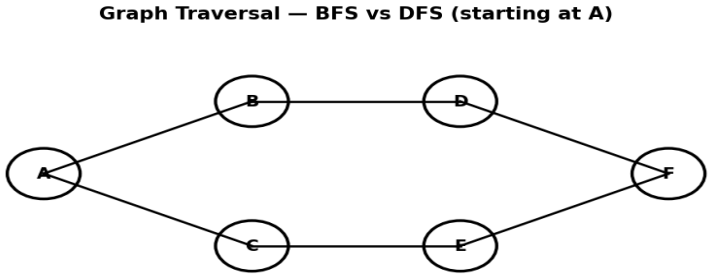


Concept	Description
Hash function	Maps a key to an array INDEX; e.g. h(key) = key MOD table_size
Collision	Two different keys hash to the SAME index
Load factor (α)	α = (number of stored elements) / (table size) – indicates how full the table is

Collision resolution technique	How it works
Chaining	Each bucket holds a LINKED LIST of all keys that hash there
Linear Probing	On collision, check the NEXT slot (i+1, i+2, ...) — but suffers 'primary clustering'
Quadratic Probing	On collision, check slots at increasing SQUARED intervals (i+1², i+2², ...)
Double Hashing	On collision, use a SECOND hash function to compute the step size

Average search time in a well-designed hash table: O(1) — the single biggest advantage of hashing over trees or sorted arrays.

Graph Representation and Traversal

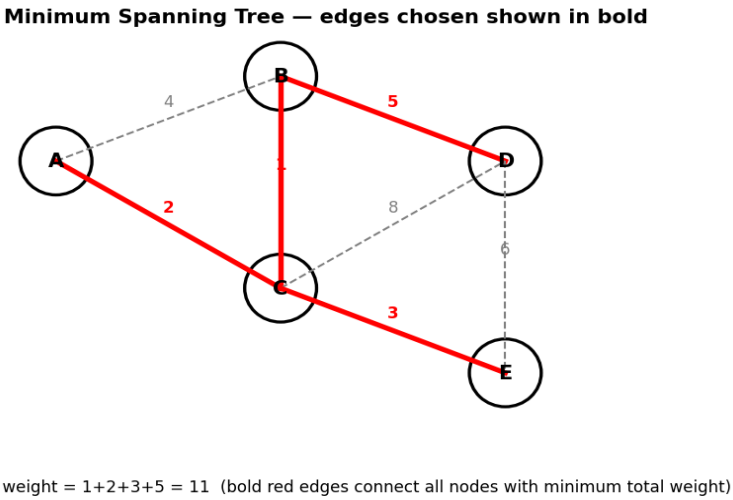


Representation	Space	Edge lookup	Best for
Adjacency Matrix	O(V²)	O(1)	DENSE graphs (many edges)
Adjacency List	O(V+E)	O(degree of vertex)	SPARSE graphs (few edges) — most common

Feature	BFS	DFS
Data structure used	Queue	Stack (or recursion)
Exploration style	Level by level (breadth first)	As deep as possible, then backtrack
Shortest path (unweighted)	YES — guaranteed shortest path	NOT guaranteed
Space complexity	Can be higher (stores a whole level)	Generally lower
Typical use	Shortest path, level-order, social network 'friends within N hops'	Topological sort, cycle detection, maze solving, connected components

Other Important Graph Algorithms

Algorithm	Purpose	Approach	Complexity
Warshall's Algorithm	Transitive closure (reachability)	Dynamic programming over all vertex triples	O(V³)
Topological Sort	Linear ordering of a DAG so every edge u→v has u before v	DFS-based or Kahn's algorithm (BFS with in-degree)	O(V+E)
Prim's Algorithm	Minimum Spanning Tree	Grow the tree one vertex at a time, always picking the cheapest connecting edge	O(E log V)
Kruskal's Algorithm	Minimum Spanning Tree	Sort ALL edges; add if it doesn't form a cycle (Union-Find)	O(E log E)
Dijkstra's Algorithm	Single-source shortest path	Greedy — NON-negative weights only	O(E log V) with a heap
Bellman-Ford Algorithm	Single-source shortest path	Works with NEGATIVE weights; detects negative cycles	O(V·E)
Floyd-Warshall Algorithm	All-pairs shortest path	Dynamic programming	O(V³)



? Easy Explanation

Prim's grows ONE tree outward from a starting vertex (like ice crystallizing); Kruskal's looks at ALL edges globally, sorted cheapest-first, and adds any edge that doesn't create a cycle. Both give a correct (and for a given graph, usually the same-weight) MST, but they build it in very different orders. Dijkstra's FAILS with negative weights; Bellman-Ford handles negative weights and can even detect negative cycles.

? Probable MCQs: Sorting, Searching and Graphs

1. Which sorting algorithm guarantees O(n log n) time in the WORST case?

a) Quick Sort

b) Merge Sort

c) Insertion Sort

d) Bubble Sort

✓Answer: b) Merge Sort

Why: Merge Sort always divides and merges in O(n log n), unlike Quick Sort's O(n²) worst case.

2. Binary search requires the input array to be:

a) Sorted

b) Unsorted

c) A linked list

d) Circular

✓Answer: a) Sorted

Why: Binary search relies on comparing against the middle element, valid only on sorted data.

3. The average time complexity of a well-designed hash table search is:

a) O(n)

b) O(log n)

c) O(1)

d) O(n²)

✓Answer: c) O(1)

Why: With a good hash function and low load factor, search is essentially constant time.

4. Which collision resolution technique stores colliding keys in a linked list per bucket?

a) Linear probing

b) Chaining

c) Quadratic probing

d) Double hashing

✓Answer: b) Chaining

Why: Chaining attaches a list to each bucket to hold all keys that hash there.

5. Which traversal technique uses a Queue data structure?

a) DFS

b) BFS

c) In-order

d) Post-order

✓Answer: b) BFS

Why: BFS explores level by level, using a queue to track the next vertices to visit.

6. Warshall's algorithm is used to compute the:

a) Minimum spanning tree

b) Transitive closure of a graph

c) Shortest path only

d) Topological order

✓Answer: b) Transitive closure of a graph

Why: Warshall's algorithm determines reachability between all pairs of vertices.

7. Dijkstra's algorithm fails to work correctly when the graph has:

a) Only positive weights

b) Negative edge weights

EASY EXPLANATION

c) No cycles

d) One vertex only

✓**Answer: b) Negative edge weights**

Why: Dijkstra's greedy approach assumes non-negative weights; use Bellman-Ford otherwise.

8. Adjacency list representation is more space-efficient for:

a) Dense graphs

b) Sparse graphs

c) Complete graphs only

d) Trees only

✓**Answer: b) Sparse graphs**

Why: Adjacency lists use O(V+E) space, ideal when E is much smaller than V².

9. Kruskal's algorithm builds the MST by:

a) Growing one tree from a start vertex

b) Sorting all edges and adding non-cycle-forming ones

c) Running BFS on the graph

d) Always picking the highest weight edge

✓**Answer: b) Sorting all edges and adding non-cycle-forming ones**

Why: Kruskal's is a global, edge-sorted, cycle-avoiding approach using Union-Find.

EASY EXPLANATION

7.3 Data Models, Normalization and SQL

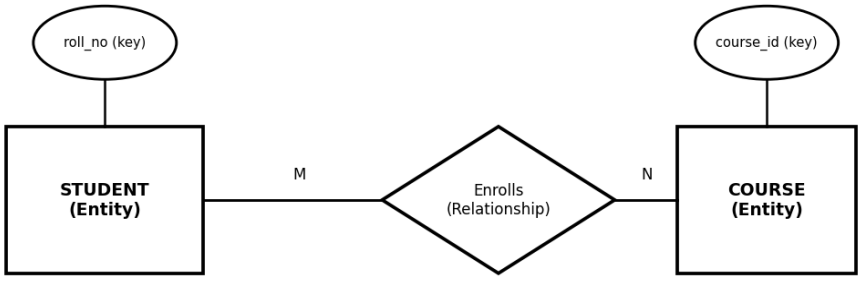
Levels of Data Abstraction

Level	Also called	Describes
Physical level	Internal level	HOW data is actually stored on disk (file organization, indexing)
Logical level	Conceptual level	WHAT data is stored and the RELATIONSHIPS between them (schema)
View level	External level	HOW individual users see a PART of the database (customized views)

Data independence type	Meaning
Physical data independence	Changing the physical storage does NOT require changing the logical schema
Logical data independence	Changing the logical schema does NOT require changing application programs/views

E-R Model — Complete Vocabulary

Simple E-R Diagram: Student — Enrolls — Course



Rectangle=Entity, Diamond=Relationship, Ellipse=Attribute, M:N = many-to-many cardinality

Term	Symbol in E-R diagram	Meaning
Entity	Rectangle	A real-world object with independent existence (e.g. Student)
Attribute	Ellipse/Oval	A property of an entity (e.g. name, roll_no)
Relationship	Diamond	An association BETWEEN two or more entities (e.g. Enrolls)
Key attribute	Underlined ellipse	Uniquely identifies an entity instance
Multi-valued attribute	Double ellipse	An attribute that can have MULTIPLE values (e.g. phone numbers)
Derived attribute	Dashed ellipse	Computed from other attributes (e.g. age from date of birth)

Entity type	Property
Strong entity	Has its OWN primary key; drawn as a single rectangle
Weak entity	Has NO primary key of its own; depends on a strong (owner) entity; drawn as a DOUBLE rectangle, identified by a partial key + owner's key

Cardinality (mapping)	Meaning	Example
One-to-One (1:1)	One entity instance relates to exactly one of another	Person – Passport
One-to-Many (1:N)	One instance relates to MANY of another	Department – Employees
Many-to-Many (M:N)	Many instances relate to many others	Student – Course (enrollment)

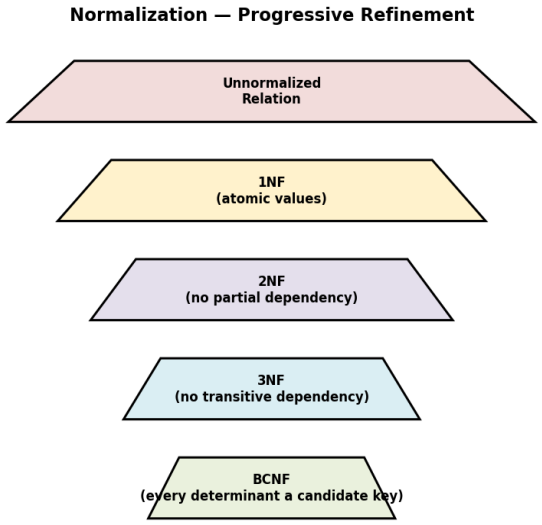
Functional Dependency

X → Y means 'X functionally determines Y'

Given the same value of X, Y is ALWAYS the same

FD type	Description
Full functional dependency	Y depends on the WHOLE of a composite key X, not just part of it
Partial dependency	Y depends on only PART of a composite key
Transitive dependency	X→Y and Y→Z implies X→Z, where Y is NOT a key

Normalization — Step by Step



Normal Form	Rule	Removes
1NF	Every attribute holds a single ATOMIC value; no repeating groups/arrays	Multi-valued / composite attributes
2NF	Must be in 1NF, AND no non-key attribute depends on only PART of a composite primary key	Partial dependency
3NF	Must be in 2NF, AND no non-key attribute depends on another NON-KEY attribute	Transitive dependency
BCNF (Boyce-Codd NF)	Must be in 3NF, AND for every functional dependency X→Y, X must be a SUPER KEY	Remaining anomalies 3NF misses
4NF	Must be in BCNF, AND no MULTI-VALUED dependency remains	Multi-valued dependency

Goal of normalization: eliminate **data redundancy** and prevent **insertion, deletion, and update anomalies**.

Anomaly type	Description
Insertion anomaly	Cannot insert a fact until other unrelated facts are also known

Anomaly type	Description
Deletion anomaly	Deleting one fact accidentally deletes other unrelated, still-needed facts
Update anomaly	The same fact is stored redundantly, so an update must be repeated in multiple places (risk of inconsistency)

Easy Explanation

2NF vs 3NF in one line: 2NF removes PARTIAL dependency (only meaningful when the primary key has MULTIPLE columns); 3NF removes TRANSITIVE dependency (A→B→C, where C depends on B, a non-key attribute, rather than directly on the key A).

Relational Algebra Operators

Operator	Symbol	Purpose	Example
Selection	σ (sigma)	Filters ROWS matching a condition	σ(age>20)(Student)
Projection	π (pi)	Selects specific COLUMNS	π(name,age)(Student)
Union	U	Combines rows from two UNION-COMPATIBLE relations	R ∪ S
Set Difference	–	Rows in R but NOT in S	R − S
Cartesian Product	×	All possible combinations of rows from R and S	R × S
Join	⋈	Combines related rows from two relations based on a common attribute	R ⋈ S
Rename	ρ (rho)	Renames a relation or its attributes	ρ(NewName)(R)

Join type	Description
Inner Join	Returns only MATCHING rows from both tables
Left Outer Join	All rows from the LEFT table, matched rows from the right (NULL if no match)
Right Outer Join	All rows from the RIGHT table, matched rows from the left
Full Outer Join	ALL rows from both tables, matched where possible
Natural Join	Inner join that automatically matches columns with the SAME name

SQL Command Categories

Category	Full form	Commands	Purpose
DDL	Data Definition Language	CREATE, ALTER, DROP, TRUNCATE	Defines/modifies database STRUCTURE
DML	Data Manipulation Language	SELECT, INSERT, UPDATE, DELETE	Manipulates the DATA itself
DCL	Data Control Language	GRANT, REVOKE	Controls ACCESS PERMISSIONS
TCL	Transaction Control Language	COMMIT, ROLLBACK, SAVEPOINT	Manages TRANSACTIONS

Common SQL commands

```
-- DDL
CREATE TABLE Student (roll_no INT PRIMARY KEY, name VARCHAR(30), age INT);

-- DML
INSERT INTO Student VALUES (1, 'Ram', 20);
SELECT name FROM Student WHERE age > 18;
UPDATE Student SET age = 21 WHERE roll_no = 1;

-- JOIN example
SELECT s.name, c.course_name
FROM Student s JOIN Enrolls e ON s.roll_no = e.roll_no
               JOIN Course c  ON e.course_id = c.course_id;
```

View: a **virtual table** defined by a stored query — holds no data of its own, always reflects the current state of its underlying base tables.

EXPLANATION

? Probable MCQs: Data Models, Normalization and SQL

1. A relation is in 1NF if:

a) It has a primary key
b) All attribute values are atomic (indivisible)
c) It has no foreign keys
d) It has fewer than 5 columns
✓Answer: b) All attribute values are atomic (indivisible)
Why: 1NF requires atomic, single values in every attribute — no repeating groups.

2. A weak entity is characterized by:

a) Having its own unique primary key
b) Depending on a strong entity for identification
c) Never used in a database
d) Having no attributes
✓Answer: b) Depending on a strong entity for identification
Why: Weak entities lack a full primary key, identified via a partial key plus the owner's key.

3. Removing partial dependency from a relation achieves which normal form?

a) 1NF
b) 2NF
c) 3NF
d) BCNF
✓Answer: b) 2NF
Why: 2NF specifically eliminates partial dependency on a composite key.

4. Removing transitive dependency achieves which normal form?

a) 1NF
b) 2NF
c) 3NF
d) 0NF
✓Answer: c) 3NF
Why: 3NF removes dependency of a non-key attribute on another non-key attribute.

5. In relational algebra, σ represents:

a) Projection
b) Selection
c) Join
d) Union
✓Answer: b) Selection
Why: σ (sigma) filters rows based on a condition.

6. In relational algebra, π represents:

a) Selection
b) Projection (choosing columns)
c) Renaming
d) Cartesian product
✓Answer: b) Projection (choosing columns)
Why: π (pi) selects specific columns/attributes from a relation.

7. A database VIEW is best described as:

a) A physically stored duplicate table

EASY EXPLANATION

b) A virtual table derived from a query

c) An index on a table

d) A type of primary key

✓Answer: b) A virtual table derived from a query

Why: A view holds no data itself; it's a saved query presented as a virtual table.

8. CREATE, ALTER and DROP belong to which SQL command category?

a) DML

b) DDL

c) DCL

d) TCL

✓Answer: b) DDL

Why: These commands define/modify database structure — Data Definition Language.

9. A Natural Join automatically matches columns that:

a) Have the same data type only

b) Have the same name in both tables

c) Are primary keys only

d) Are numeric

✓Answer: b) Have the same name in both tables

Why: Natural join joins on all identically-named columns automatically.

EASY EXPLANATION

7.4 Transaction Processing, Concurrency Control and Crash Recovery

ACID Properties — Detailed

Property	Meaning	Violated by (example)
Atomicity	Transaction is ALL-OR-NOTHING	Power failure mid-transaction, partially applied
Consistency	Moves the DB from one VALID state to another valid state	A transaction violates an integrity constraint
Isolation	Concurrent transactions do NOT interfere with each other's intermediate results	Dirty read, lost update
Durability	Once COMMITTED, changes survive even a crash	Committed data lost after a crash (no proper logging)

Transaction States

State	Description
Active	The transaction is currently EXECUTING
Partially committed	Final statement executed, but not yet permanently saved
Committed	Transaction completed SUCCESSFULLY, changes made PERMANENT
Failed	Cannot proceed with normal execution
Aborted	Transaction ROLLED BACK, database restored to its state before the transaction

Concurrency Problems (Without Proper Control)

Problem	Description
Lost Update	Two transactions update the same item; one update is OVERWRITTEN and lost
Dirty Read	A transaction reads data written by another UNCOMMITTED transaction, which later rolls back
Unrepeatable Read	A transaction reads the same row twice and gets DIFFERENT values because another transaction modified it in between
Phantom Read	A transaction re-executes a query and finds NEW rows that another transaction inserted in between

Serializability and Locking

- **Serializability:** a concurrent schedule is **correct** if it produces the **same result as SOME serial (one-at-a-time) execution** of the same transactions.

Lock type	Also called	Can be held by	Permits
Shared lock	Read lock (S)	MULTIPLE transactions simultaneously	Only READING
Exclusive lock	Write lock (X)	Only ONE transaction	READING and WRITING

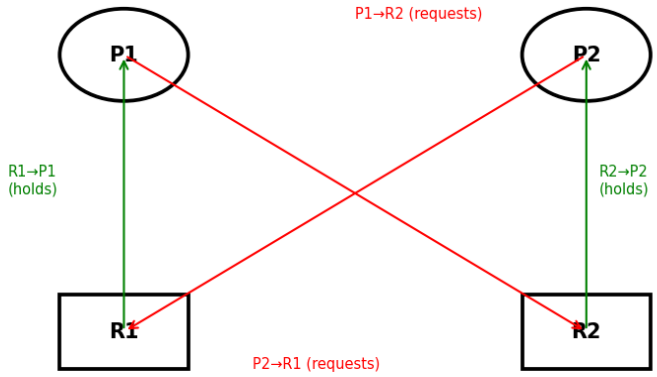
Two-Phase Locking (2PL) phase	Rule
Growing phase	Transaction may ACQUIRE locks, but never release any
Shrinking phase	Transaction may RELEASE locks, but never acquire any more

2PL guarantees serializability, but can still suffer **deadlock**. A stricter variant, **Strict 2PL**, holds all exclusive locks until COMMIT/ABORT, which also prevents cascading rollbacks.

Deadlock — Complete Coverage

FAST EXPLANATION

Deadlock — Circular Wait Condition



Circular chain P1→R2→P2→R1→P1 = DEADLOCK (Coffman's 4 conditions all hold)

Coffman's 4 NECESSARY conditions (ALL must hold simultaneously):
Mutual Exclusion + Hold & Wait + No Preemption + Circular Wait

Strategy	Approach	Trade-off
Prevention	Deny at least ONE of the 4 Coffman conditions structurally	Can reduce resource utilization/concurrency
Avoidance	Grant a request only if the resulting state is 'SAFE' (Banker's Algorithm)	Needs advance knowledge of maximum resource needs
Detection & Recovery	Allow deadlock to occur, detect it (wait-for graph), then recover (kill/rollback a process)	Overhead of periodic detection
Ignorance (Ostrich algorithm)	Do nothing — assume deadlocks are rare enough to not matter	Used by many general-purpose OSes (e.g. most of Windows/Linux)

Easy Explanation

EASY EXPLANATION

ALL FOUR Coffman conditions must hold AT THE SAME TIME for deadlock — breaking even ONE (e.g. allowing preemption) prevents it. This is exactly why each prevention strategy targets just one condition (e.g. 'request all resources at once' breaks Hold & Wait; 'impose a strict resource ordering' breaks Circular Wait).

Failure Classification and Recovery

Failure type	Description
Transaction failure	Logical error (e.g. bad input, overflow) or system-detected error causing the transaction to abort
System crash	Hardware/software failure — CONTENTS OF VOLATILE MEMORY (including buffers) are LOST
Disk failure	Loss of stored data on NON-VOLATILE storage — needs backups/replication to recover

Recovery concept	Description
Write-Ahead Logging (WAL)	Every change is logged BEFORE it is applied to the database, so recovery can replay/undo from the log
REDO	Reapplies changes of COMMITTED transactions that may not have reached disk before the crash
UNDO	Reverses changes of UNCOMMITTED transactions that were partially applied before the crash
Checkpoint	A saved snapshot marking that all earlier transactions are safely on disk — speeds up recovery by limiting how far back the log must be scanned

? Probable MCQs: Transactions, Concurrency and Recovery

1. Which ACID property ensures a transaction is either fully completed or fully rolled back?

a) Consistency
b) Atomicity
c) Isolation
d) Durability
✓Answer: b) Atomicity
Why: Atomicity guarantees all-or-nothing execution.

2. Reading data written by an uncommitted transaction that later rolls back is called a:

a) Lost update
b) Dirty read
c) Phantom read
d) Deadlock
✓Answer: b) Dirty read
Why: A dirty read occurs when uncommitted, possibly-invalid data is read by another transaction.

3. In Two-Phase Locking, once a transaction releases a lock, it:

a) Can acquire new locks freely
b) Cannot acquire any more locks (shrinking phase)
c) Must acquire all locks again
d) Is immediately aborted
✓Answer: b) Cannot acquire any more locks (shrinking phase)
Why: After entering the shrinking phase, only releases are allowed, never new acquisitions.

4. Which of the following is NOT one of Coffman's 4 conditions for deadlock?

a) Mutual exclusion
b) Hold and wait
c) Preemption allowed
d) Circular wait

EASY EXPLANATION

✓**Answer: c) Preemption allowed**

Why: The actual condition is 'NO preemption' — allowing preemption would prevent deadlock.

5. The Banker's Algorithm is used for deadlock:

a) Detection

b) Avoidance

c) Prevention by denying mutual exclusion

d) Ignoring

✓**Answer: b) Avoidance**

Why: Banker's Algorithm grants requests only if the resulting state remains 'safe' — avoidance.

6. Write-Ahead Logging requires that:

a) Data is written to disk before the log

b) Log entries are written before the actual database change

c) No logs are needed for recovery

d) Only committed transactions are logged

✓**Answer: b) Log entries are written before the actual database change**

Why: WAL mandates the log record precedes the corresponding database modification.

7. During recovery, REDO is used to:

a) Reverse uncommitted changes

b) Reapply changes of committed transactions

c) Delete all logs

d) Restart the database from scratch

✓**Answer: b) Reapply changes of committed transactions**

Why: REDO ensures committed transactions' effects survive a crash by reapplying them from the log.

8. A shared (S) lock permits:

a) Only one transaction to read and write

b) Multiple transactions to read simultaneously

c) Any transaction to write

d) No access at all

✓**Answer: b) Multiple transactions to read simultaneously**

Why: Shared locks allow concurrent reads but not writes.

EASY EXPLANATION

7.5 Operating System and Process Management

Types of Operating Systems

Type	Description
Batch OS	Jobs collected and executed in batches with NO user interaction during execution
Multiprogramming OS	MULTIPLE programs reside in memory; CPU switches to another when one waits for I/O — maximizes CPU utilization
Time-Sharing OS	Extends multiprogramming with RAPID CPU switching (time slices), giving each user the illusion of a dedicated system
Multiprocessing OS	Uses MULTIPLE CPUs working together
Distributed OS	Manages a group of physically SEPARATE, networked computers as one system
Real-Time OS (RTOS)	Guarantees response within STRICT time deadlines
Network OS	Manages data, users, and resources across a network, running on a server

Process Concepts

Term	Meaning
Process	A PROGRAM IN EXECUTION — has its own memory space, PCB, and resources
Thread	A LIGHTWEIGHT unit of execution WITHIN a process — shares the process's memory/resources
Program	A PASSIVE entity — just code stored on disk, not yet executing

PCB (Process Control Block) — the data structure holding everything the OS needs to manage a process:

PCB field	Contents
Process ID (PID)	Unique identifier
Process state	New / Ready / Running / Waiting / Terminated
Program Counter	Address of the next instruction
CPU registers	Saved register values for context switching
Memory management info	Base/limit registers, page tables
Scheduling info	Priority, pointers to scheduling queues
Open file list, I/O status	Files and devices currently allocated to the process

Multitasking / Multiprocessing	Description
Multitasking	ONE CPU rapidly switches between multiple tasks (time-sharing) — illusion of parallelism
Multiprocessing	MULTIPLE CPUs execute tasks in TRUE parallel

CPU Scheduling Algorithms — Full Comparison

Algorithm	Preemptive?	Selection basis	Key advantage	Key disadvantage
FCFS	No	Arrival order	Simple, fair by arrival	CONVOY EFFECT — short jobs stuck behind long ones
SJF (non-preemptive)	No	Shortest CPU burst	MINIMUM average waiting time (provably optimal)	Needs burst-time estimate; STARVATION of long jobs
SRTF (preemptive SJF)	Yes	Shortest REMAINING time	Even better average waiting time	More context switches; worse starvation risk
Priority Scheduling	Either	Highest priority value	Important jobs served first	STARVATION of low priority — fixed by AGING
Round Robin	Yes	Cyclic, fixed TIME QUANTUM	Fair; great response time; no starvation	Performance very sensitive to quantum size
Multilevel Queue	Either	Fixed queue per process class	Suits distinct workload types	Rigid — a process cannot move between queues
Multilevel Feedback Queue	Yes	Processes MOVE between queues by behaviour	Most adaptive and flexible	Most complex to configure

EASY EXPLANATION

Turnaround Time (TAT) = Completion Time – Arrival Time

Waiting Time (WT) = Turnaround Time – Burst Time

Response Time (RT) = Time of FIRST CPU allocation – Arrival Time

Easy Explanation

SJF gives the provably MINIMUM average waiting time but can starve long jobs; Round Robin gives the best average RESPONSE time and guarantees no starvation; FCFS is simplest but suffers the convoy effect. AGING (gradually increasing a waiting process's priority) is the standard starvation fix.

Process Synchronization

Concept	Description
Race condition	Final result depends unpredictably on the TIMING of concurrent processes accessing shared data
Critical section	The part of code where a process accesses SHARED resources – must run with mutual exclusion
Mutual exclusion	Only ONE process/thread may be in its critical section at a time

Requirement for a critical section solution	Meaning
Mutual Exclusion	Only one process in the critical section at a time
Progress	If no process is in the critical section, one of the waiting processes must be allowed in without indefinite delay
Bounded Waiting	A limit must exist on how many times other processes can enter before a waiting process gets its turn

Tool	Type	Key property
Semaphore	Integer variable with wait()/P() and signal()/V()	Binary (0/1, mutex-like) or Counting (controls a POOL of resources)
Mutex	Binary LOCK with OWNERSHIP	Only the process that locked it may unlock it
Monitor	High-level construct	Combines mutual exclusion + CONDITION VARIABLES; easier to use correctly than raw semaphores

Classical problem	Scenario
Producer-Consumer	Producers add items and consumers remove items from a SHARED BOUNDED BUFFER
Readers-Writers	Multiple READERS may access data simultaneously, but a WRITER needs exclusive access
Dining Philosophers	5 philosophers share 5 forks; illustrates DEADLOCK and STARVATION in resource sharing

Probable MCQs: Process Management and Synchronization

1. The PCB is used by the OS to store:

a) Only file data

b) All information needed to manage a process

c) Only user login details

d) Source code

✓Answer: b) All information needed to manage a process

Why: PCB holds everything needed to suspend and later resume a process correctly.

2. Which scheduling algorithm provides the provably MINIMUM average waiting time?

a) FCFS

b) SJF

c) Round Robin

FAST EXPLANATION

d) Priority

✓Answer: b) SJF

Why: Shortest Job First is optimal for average waiting time among non-preemptive algorithms.

3. The convoy effect is a drawback of which scheduling algorithm?

a) FCFS

b) SJF

c) Round Robin

d) SRTF

✓Answer: a) FCFS

Why: In FCFS, short processes queue behind a long one, inflating average waiting time.

4. A binary semaphore is functionally similar to a:

a) Counting semaphore with unlimited value

b) Mutex (lock)

c) Round-robin scheduler

d) Page table

✓Answer: b) Mutex (lock)

Why: A binary semaphore (0 or 1) behaves like a mutex, allowing one process access at a time.

5. Starvation in priority scheduling is commonly solved by:

a) Increasing the time quantum

b) Aging

c) Disabling interrupts

d) Using FCFS instead

✓Answer: b) Aging

Why: Aging gradually raises a waiting process's priority so it eventually runs.

6. The Dining Philosophers problem primarily illustrates:

a) Sorting efficiency

b) Deadlock and starvation in resource sharing

c) Memory paging

d) File allocation

✓Answer: b) Deadlock and starvation in resource sharing

Why: Shared, limited forks among philosophers models classic concurrency deadlock/starvation.

7. Which requirement of critical section solutions ensures no indefinite delay when the section is free?

a) Mutual exclusion

b) Progress

c) Bounded waiting

d) Starvation

✓Answer: b) Progress

Why: Progress guarantees the decision of who enters next isn't postponed indefinitely.

8. A Monitor differs from a semaphore because a monitor:

a) Cannot provide mutual exclusion

b) Combines mutual exclusion with condition variables in a high-level construct

c) Is only used in hardware

d) Cannot be implemented in software

✓Answer: b) Combines mutual exclusion with condition variables in a high-level construct

Why: Monitors bundle mutual exclusion and condition variables, reducing programmer error versus raw semaphores.

EASY EXPLANATION

EASY EXPLANATION

7.6 Memory Management, File Systems and System Administration

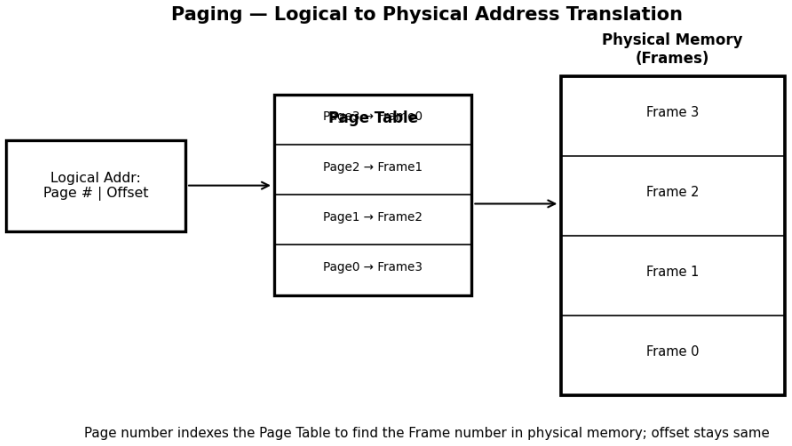
Memory Management Techniques

Technique	Description
Contiguous allocation	Each process occupies ONE continuous block of memory — simple but causes fragmentation
Paging	Memory divided into FIXED-size frames; process divided into equal-size PAGES — no external fragmentation
Segmentation	Memory divided into VARIABLE-size logical SEGMENTS (code, data, stack) matching the program's structure
Swapping	A whole process is temporarily moved OUT to disk and back to free up RAM for others
Virtual Memory	Gives each process the illusion of a LARGE, contiguous private address space, backed by disk

Fragmentation type	Occurs in	Description
Internal fragmentation	Paging / fixed partitions	Allocated memory block is LARGER than needed — wasted space INSIDE the block
External fragmentation	Contiguous/variable allocation	Enough TOTAL free memory exists, but not in one CONTIGUOUS block

Solution to external fragmentation: compaction (shuffling processes to consolidate free memory into one block) or switching to paging.

Paging — Address Translation



Page number indexes the Page Table to find the Frame number in physical memory; offset stays same

Logical Address = Page Number (p) + Page Offset (d)

Physical Address = Frame Number (f, from Page Table[p]) + Offset (d) — offset stays the same

Worked example: page size = 1KB (offset = 10 bits). Logical address 3000 in decimal → page number = 3000 div 1024 = 2, offset = 3000 mod 1024 = 952. If the page table maps page 2 → frame 5, physical address = 5×1024 + 952 = 6072.

Demand Paging and Page Replacement

Demand paging: a page is loaded into memory **only when actually referenced** — reduces memory waste but causes a **page fault** on first access.

Algorithm	Strategy	Belady's Anomaly?
FIFO	Replace the OLDEST loaded page	YES — can occur
LRU (Least Recently Used)	Replace the page UNUSED for the longest time	NO
Optimal (OPT/MIN)	Replace the page needed FURTHEST in the future (theoretical best)	NO
LFU	Replace the page with the LOWEST access count	NO (different issues)

Algorithm	Strategy	Belady's Anomaly?
Second Chance (Clock)	FIFO + a reference bit gives a page a 'second chance' before eviction	NO

Worked FIFO/Belady's Anomaly example: reference string `1,2,3,4,1,2,5,1,2,3,4,5`

Frames	Page faults (FIFO)
3 frames	`9 faults`
4 frames	`10 faults ← MORE frames, MORE faults (the anomaly!)`

- **Thrashing:** the system spends **more time paging (swapping) than executing** useful work — happens when too many processes compete for too little physical memory. Fixed by reducing the degree of multiprogramming or adding memory.

Easy Explanation

Belady's Anomaly is a classic exam trap: students assume MORE memory frames always means FEWER page faults, but FIFO can violate this. LRU and Optimal NEVER suffer from Belady's Anomaly — this distinction is frequently tested.

File Systems

File allocation method	Description	Advantage	Disadvantage
Contiguous	File occupies consecutive disk blocks	FAST sequential AND direct access	External fragmentation; hard to grow files
Linked	Each block POINTS to the next block	No external fragmentation; easy to grow	SLOW random access (must follow pointers); pointer overhead
Indexed	A separate INDEX BLOCK lists all data block addresses	Supports fast DIRECT access; no external fragmentation	Extra space for the index block; index block itself may need indirection for large files

Directory structure	Description
Single-level	`One directory for ALL users/files – simple but no organization, name conflicts`
Two-level	`Separate directory PER USER`
Tree-structured	`HIERARCHICAL – directories can contain subdirectories (used by all modern OSes)`
Acyclic-graph	`Allows SHARING via links (e.g. symbolic links), but no cycles allowed`

System Administration

Task	Description
User account management	`Creating/deleting accounts, setting permissions and quotas`
Backup and restore	`Regular data backups; tested restore procedures for disaster recovery`
Start/shutdown procedures	`Controlled boot sequence and graceful shutdown to avoid data loss`
Monitoring	`Tracking system performance, resource usage, and security logs`
Security patching	`Applying updates to fix vulnerabilities`

Probable MCQs: Memory Management and File Systems

1. Belady's Anomaly is associated with which page replacement algorithm?

a) LRU
b) Optimal
c) FIFO
d) LFU

✓Answer: c) FIFO

Why: FIFO can uniquely show MORE frames causing MORE page faults.

2. Internal fragmentation is most associated with:

EASY EXPLANATION

a) Linked allocation
b) Paging / fixed-size partitions
c) Segmentation only
d) Indexed allocation
✓Answer: b) Paging / fixed-size partitions
Why: Fixed-size frames/partitions can allocate more space than a process needs, wasting space inside the block.

3. Which file allocation method uses a separate index block listing all data blocks?
a) Contiguous allocation
b) Linked allocation
c) Indexed allocation
d) Sequential allocation
✓Answer: c) Indexed allocation
Why: Indexed allocation keeps a dedicated index block pointing to all of a file's data blocks.

4. Demand paging loads a page into memory:
a) At process start, all at once
b) Only when actually referenced
c) Never
d) Only during shutdown
✓Answer: b) Only when actually referenced
Why: Demand paging brings pages in lazily, only on first reference, saving memory.

5. Thrashing occurs when:
a) A system has excess free memory
b) The system spends more time paging than executing
c) All processes are in the Ready state
d) The CPU is idle
✓Answer: b) The system spends more time paging than executing
Why: Thrashing is caused by excessive paging overwhelming actual computation time.

6. Which directory structure is used by virtually all modern operating systems?
a) Single-level
b) Two-level
c) Tree-structured (hierarchical)
d) None — flat structure only
✓Answer: c) Tree-structured (hierarchical)
Why: Tree-structured directories allow nested subdirectories, the modern standard.

7. External fragmentation can be resolved by:
a) Increasing page size
b) Compaction
c) Adding more processes
d) Disabling virtual memory
✓Answer: b) Compaction
Why: Compaction shuffles allocated memory to consolidate free space into one contiguous block.

8. Virtual memory primarily gives each process the illusion of:
a) Unlimited CPU speed
b) A large, contiguous private address space
c) Direct access to other processes' memory

EASY EXPLANATION

d) No need for a page table

✓Answer: b) A large, contiguous private address space

Why: Virtual memory abstracts physical memory limitations behind a large private address space per process.

EASY EXPLANATION