BCS (□□□□□□□□□ □□□□□□□ □ □□□□□□□□) □□□□□ □□□□□□□ □□□□□□□ □□□□□□□□□□□□ □□□□□□ □ □□□□□□□□□ □□□□□

□□ □□□□□□□□□□ □□□□□□□□□□ □□□□□□ □□□— (a) Algorithm, (b) Operating System, (c) Database Management System, (d) Software Engineering — □□□□ □□□□□□□□□ □□□□□□□□□□□□□ □□□□□□□□□□□ □□□ □□□□□□□□□ □□□□□□□□□□ □□□□□□ □□□□□□ □□□□□□□

(a) Algorithm

1) □□□□□□□□□□□ □□? □□□□□□□□□□ □ □□□□□□□□□ □□□□□□□□□ □□□□

□□□□□□: □□□□□□□□□□□ □□□ □□□□□ □□□□□□□□□□ □□□□□□□□ □□□□□□□□□ □□□□□ □□□□□-□□□□□ □□□□□□□ □□□□□□□ □□□□□□□□□□□□□□ (finite, unambiguous steps) □□□□□□□□ □□□□□□□□□□□: (i) □□□□□□/□□□□□□□ □□□□□□□□□□□, (ii) □□□□□□□□ □□□□ □□□□□□□□ □□□□, (iii) □□□□□□□□ □□□□□□□□□, (iv) □□□□□□□□□□□□□ □□□□□□□□□□□□□□ □□□□□□□: □□□□□ □□□□□□□□ □□□□□□: □□□□–□: □□□□□□ a,b; □□□□–□: s=a+b; □□□□–□: s □□□□□□□; □□□□–□: □□□□□□□□

2) Asymptotic Notation (O, Ω, Θ) □□? □□□□□□□□□□ □□□□□□□□□□ □□□□

□□□□□□: □□□□□□□□□□□□□□ □□□□□□ □□□□□/□□□□□□□ n □□□□ □□□□ □□□□□ □□□□□□ □□ □□□□□□□ Asymptotic □□□□□□□ □□□□□□□□ □□□□ □□□□□ Big-O (O): □□□□□□ □□□□□ □□ Worst-Case□ □□□□□□□: Bubble Sort ⇒ $O(n^2)$ □ Omega (Ω): □□□□□□ □□□□□ □□ Best-Case□ □□□□□□□: Binary Search □□ Best ⇒ Ω(1) (□□□□□□□□ □□□□□□□□□)□ Theta (Θ): □□□□□ □□□□□□□, □□□□ □□□□□□ □□□□□□□ □□□□□□□: Merge Sort ⇒ Θ(n log n) □

3) □□□□□□□□□□□ □ Master Theorem □□□□□□□□□□□ □□□□□□□□ □□□□□

□□□□□□: Merge Sort □□ □□□□□□□□□□□: T(n)=2T(n/2)+Θ(n)□ Master Theorem □□□□□□□□□□ a=2, b=2, f(n)=n□ n^{log_b a}=n^{log_2 2}=n□ □□□□□□□□ f(n)=Θ(n), □□□□ T(n)=Θ(n log n)□

4) Divide & Conquer □□□□□ □□□□□□□□□□ □□ □□□□ Merge Sort □□□□□□□ □□□□□

□□□□□□: □□□□□□□□ → □□□□ □□-□□□□□□□□□ □□□□ (Divide), □□□□□□□□□□□□□□ □□□□□□□ (Conquer), □□□□ □□□□□□□□ (Combine) □□□□ □□□□□ Merge Sort □□□□: (i) □□□□□□□ □□□□□□□□□ □□□□, (ii) □□□□□□□□□ □□□□□□□□□□□□ sort, (iii) Merge□ Complexity: Θ(n log n); Space: Θ(n)□

5) Greedy □□□□□ □□? Kruskal's Algorithm □□□□□□□ □□ □□□□□□□□□□ □□□□

□□□□□□: Greedy-□□ □□□□□□□□ □□□□□ □□□□□□□□□□□□□□□ □□□□□ □□□□□□□□□□□ □□□□□□□□ □□□□□ Kruskal: (i) □□ edge □□□□ □□□□□□□□□ □□□□□□□□□□□□□□ □□□□□□□, (ii) □□□□□□□□ □□ □□□□□ □□□□ □□□□□□ edge □□□□ (Union-Find □□□□□ □□□□□□□□□), (iii) n−1 □□ edge □□□□ □□□□□□ □□□□□□□□: □□□□ □□□□□□□□□□□□ (A,B,C,D) □ □□□: AB=1, AC=5, BC=2, BD=4, CD=3 ⇒ Kruskal □□□□□

AB(1), BC(2), CD(3) ⇒ [illegible] 6[illegible]

6) Dynamic Programming (DP) [illegible]? Fibonacci [illegible] 0/1 Knapsack [illegible]

[illegible]: DP-[illegible] [illegible]-[illegible] [illegible] (Overlapping subproblems + Optimal substructure) [illegible] Fibonacci: F[0]=0, F[1]=1; for i=2..n: F[i]=F[i−1]+F[i−2] ⇒ Θ(n) [illegible], Θ(n) [illegible] ([illegible] O(1) [illegible])[illegible] 0/1 Knapsack: dp[i][w]=i-[illegible] w [illegible]: max(dp[i−1][w], value_i + dp[i−1][w−weight_i]) [illegible] weight_i ≤ w[illegible]

7) Branch & Bound [illegible]? [illegible] TSP/Knapsack [illegible]

[illegible]: [illegible]/[illegible] (prune) [illegible] TSP: Lower bound [illegible] [illegible]-[illegible] MST [illegible]; [illegible] LB ≥ [illegible], [illegible]

8) String Matching: Naive, KMP [illegible] Rabin-Karp [illegible]

[illegible]: Naive: [illegible] ⇒ O(nm)[illegible] KMP: Prefix-function (LPS) [illegible] ⇒ O(n+m)[illegible] Rabin-Karp: Rolling-hash [illegible] ⇒ [illegible] O(n+m)[illegible] [illegible]: [illegible]=ABABCABAB, [illegible]=ABAB ⇒ KMP LPS=[0,0,1,2]; [illegible] 0 [illegible] 5[illegible]

9) Computational Geometry: Convex Hull (Graham Scan) [illegible]?

[illegible]: (i) [illegible] Y ([illegible] X) [illegible], (ii) [illegible]-[illegible] sort, (iii) [illegible]; [illegible]; [illegible] hull[illegible] Complexity: O(n log n)[illegible]

10) BFS/DFS, Dijkstra, Bellman-Ford, Floyd-Warshall [illegible]

[illegible]: BFS: Unweighted shortest path; DFS: [illegible] sort/[illegible]; Dijkstra: [illegible]-[illegible]; Bellman-Ford: [illegible]/[illegible]; Floyd-Warshall: [illegible]-[illegible]

11) Minimum Spanning Tree (MST): Prim vs Kruskal [illegible]

[illegible]: Kruskal: Edge-centric, sort + Union-Find; Sparse [illegible] Prim: Vertex-centric, [illegible]; Dense [illegible] adjacency matrix-[illegible] O(E log V)[illegible]

12) Max-Flow (Ford-Fulkerson/Edmonds-Karp) [illegible]? [illegible]

□□□□□: □□□□□→□□□□□□ □□□□□□□□ □□□□□□; Residual □□□□□□ augmenting path □□□□□ □□□□ □□□□□□□ □□□□ Edmonds-Karp: BFS-□ □□□□□□□ □□□ (edge-count) □□ □□□□ ⇒ $O(VE^2)$□ □□□□□□: s→a(3), s→b(2), a→b(1), a→t(2), b→t(3) ⇒ □□□□□□□□ □□□□=4□

13) Binary Search □□□□□□ □□□ □□□? □□□□□□□□ □□□□□□

□□□□□: □orted □□□□□□□□ □□□□□□ □□□□□□□, □□□/□□□□ □□□□□; □□□□□□□□ □□□□□□□□□ □□□□□□□ □□□□□□ ⇒ $T(n)=T(n/2)+O(1)$ ⇒ $O(\log n)$□

14) Amortized Analysis: Dynamic Array append □□□□□□□

□□□□□: □□□□ □□□□ □□□□□□□ (□x) □□□□ $O(n)$ □□□□ □□□□; □□□□□□ n □□□ □□□□□□□□□ □□□□ □□□ □□□ ≤ 2n ⇒ □□□□□ □□□□□□□□ □□□ $O(1)$ □□□□□□□□□

15) Approximation Algorithm: Vertex Cover 2-approx□

□□□□□: □□□□□□□□□□ □□□□ □□ (u,v) □□□; □□□□ u,v □□□□-□ □□□□□; □□□□ incident □□ □□□□ □□□□ □□□□□□ □□□□□□ □□□ □□□□ □□□□□□□ □□□□ ≤ 2 × OPT□

16) Parallel Algorithm: Parallel Sum/MapReduce □□□□□□

□□□□□: Divide-and-conquer□ n □□□□□□□□ □□□□□□□ □□□□□□□ □□□ □□□ log n □□□□ □ n/2 □□□□□□□ □□□□□ $O(\log n)$ □□□□□ □□□□□□□ MapReduce-□ Map □□□□□□□□ □□□□□ □□, Reduce-□ □□□□□□□

(b) Operating System

1) □□□□□□□□□ □□□□□□□□□□ □□□□□□□ □□□□ □ Process/Thread □□□□□□□□□□

□□□□□□: OS □□□□: □□□□□□□□□□□, □□□□□□□, I/O, □□□□□, □□□□□□□□□□ □□□□□□□□□□□□□□ Process: □□□□□□□ □□□□□□□□□□□-□□□□□□; Thread: □□□ □□□□□□□□□□□□□ □□□□□□ □□□□□□ □□□□□□□□□□□ □□□□□□, □□□□/□□□□□/□□□□ □□□□□□□ □□□□□

2) Process State □ PCB □□?

□□□□□□: State: New→Ready→Running→Waiting/Blocked→Terminated□ PCB (Process Control Block): PID, □□□□□□□□□□, PC, □□□□□□, □□□□□□□□□□□□□, □□□□□□□□□□□-□□□□□□□, □□□□□-□□□□□ □□□□□□□□ □□□□□□□□ □□□□□

3) CPU Scheduling: FCFS, SJF, Priority, RR □□□□□□□□□□; □□□□□ □□□□ □□□□□□□□□

□□□□□□: FCFS: □□□□ □□□□□□□; SJF: □□□□ □□□□ □□□□; Priority: □□□□□□□□□□□□□-□□□□□□□□□; RR: □□□□□-□□□□□□□□□□□□ □□ □□□□□-□□□□□□ □□□□□□□□ (Burst ms): P1=6, P2=2, P3=4 (□□□□□ □□□□□ 0): FCFS □ □□□□ □□□□□□□□□=(0+6+8)/3=4.67ms; RR(q=2) □□□□ □□□□ □□□□□□□□□ □□□□□

4) Critical Section Problem □ □□□□□□□□□□□□□

□□□□□□: □□□□□□□□□ □□□□□ □□□□□□□□□□□ Mutual exclusion, Progress, Bounded waiting □□□□□□□□ □□□□□ □□□□□ □□□□□□□: Peterson’s algorithm (2-process), Bakery algorithm (N-process), □□□□□□□□□□□□□ □□□□□□□□□□□ (Test&Set;, Compare&Swap;), □□□□□□□□□□□ Semaphore/Monitor□

5) Semaphore □□? P/V □□□□□□□□□ □ □□□□□□□□□

□□□□□□: □□□□□□□□□ S □□□□□ □□□□□□□□□□□□□□□ P(S): while S≤0 wait; S=S−1□ V(S): S=S+1; □□□□□□□□□□□ waiting queue □□□□□ □□□□□□□□□ Producer-Consumer: empty, full, mutex □□□□□□□□ □□□□□□ □□□□□□ □□□□□□□□□□□□□□

6) Inter-Process Communication (IPC) □□□□□□□□□

□□□□□□: Pipes/FIFOs, Message queue, Shared memory (+Semaphore), Sockets, Signals□ □□□□□□□□: Shared memory □□□□□□ □□□□ □□□□□ □□□□□□ □□□□□□□□; □□□□□□□□□□□□□□□□□□□ □□□□□□□□□□

7) Deadlock: □□□□□, □□□□□□□□□□/□□□□□□□□/□□□□□□□□ □ □□□□□□□□□□□□□□

□□□□□□: □□□□□□□□□□□ □□□□ □□□□□: Mutual exclusion, Hold-and-wait, No preemption, Circular wait□ Prevention: □□□□□ □□□□□ □□□□□ (□□□□□ circular wait □□□□□ □□□□□□□□□ □□□□□□□)□ Avoidance: Banker's algorithm□ Detection & Recovery: Wait-for graph; □□□□□□□ □□□□ □□□□□□□□□□□□ kill/rollback□

8) [illegible] [illegible]: Paging, Segmentation, Page Replacement[illegible]

[illegible]: Paging: [illegible] [illegible] [illegible]/[illegible]; [illegible]-[illegible] [illegible] TLB[illegible] Segmentation: [illegible] [illegible]; [illegible] [illegible] [illegible] [illegible] Replacement: FIFO, LRU, Optimal (Belady’s anomaly: FIFO-[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]) [illegible] [illegible]: [illegible] [illegible] 7 0 1 2 0 3 0 4; [illegible]=3 ⇒ FIFO [illegible] 6 [illegible], LRU [illegible] 6 [illegible] ([illegible] [illegible])[illegible]

9) Secondary Storage: Disk Scheduling [illegible]

[illegible]: FCFS, SSTF, SCAN, C-SCAN, LOOK[illegible] [illegible]: [illegible]=50; [illegible]: 82,170,43,140,24,16,190 ⇒ SCAN ([illegible]) [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

10) File System: [illegible], FCB/Inode, Space allocation, Directory[illegible]

[illegible]: [illegible]: Boot block, Super block, inode/FCB [illegible], Data blocks[illegible] FCB/Inode: [illegible] [illegible]— size, owner, permissions, block-pointers[illegible] Allocation: Contiguous, Linked, Indexed ([illegible] inode-[illegible] direct/indirect pointers) [illegible] Directory: [illegible]/[illegible] [illegible] [illegible]; [illegible] /home/user/file.txt[illegible]

11) Protection & Security: Access Control[illegible]

[illegible]: Authentication ([illegible]/[illegible]), Authorization (ACL/Capability), Auditing/Logging[illegible] Access control matrix-[illegible] [illegible] × [illegible] [illegible] [illegible]

(c) Database Management System (DBMS)

1) DBMS □□? □□□, □□□□□□□/□□□□□□□□□□

□□□□□: □□□□□□□□ □□□□□/□□□□□□□□□□ □□□□□□□□□□□□ □□□: Hierarchical, Network, Relational (RDBMS), NoSQL (Key-Value, Document, Column, Graph) □ □□□□□□: □□ Redundancy, Consistency, □□□□□□□□□□, □□□□□□□□□□□□ □□□□□□□: □□□/□□□□□□, □□□□□□□□□□□□□ □□□□□□□□□□

2) ER-□□□□: Entity/Attribute/Relationship, Cardinality □□□□□□□□

□□□□□: Entity=Student(id,name), Course(cid,title); Relationship=Enroll(Student↔Course) (Many-to-Many) □ Cardinality: 1:1, 1:N, M:N; Participation: Total/Partial□ ER-□□□□□□□□□□□□□ □□□□□□ □□□□□□□□□ □□□□ M:N □□□□□□□□□ □□□□ □□□□□ □□□□□ Enroll(student_id, course_id, grade)□

3) Relational Model: Keys □ Integrity Constraints□

□□□□□: Keys: Super/Candidate/Primary/Alternate/Foreign□ Integrity: Entity (PK NULL □□□), Referential (FK→PK), Domain (□□□□-□□□□/□□□□□□), Unique, Check□

4) Functional Dependency (FD) □ Minimal Cover□

□□□□□: X→Y □□□□ X □□□□ □□□ Y□ □□□□□ Armstrong's axioms: Reflexivity, Augmentation, Transitivity□ Minimal cover: (i) RHS □□□ □□□, (ii) LHS □□□□ □□□□□□□□ □□□□□□□□□□□□ □□□, (iii) □□□□□□□□ FD □□□□

5) Normalization: 1NF→BCNF □□□□□□□□□□

□□□□□: □□□□□□□ □□□□□□: Order(order_id, cust, cust_city, item, item_price, cust_city_zip) □ □□□□□□□□□: cust→cust_city,cust_city_zip; item→item_price□ 2NF: □□□□□ □□□□□□□□□ □□□ (□□□ □□□□□□□□□ □□ □□□□□)□ 3NF: □□□□□□□□□□□ □□□□□□□□□ □□□: □□□□□ □□□□ ⇒ Customer(cust, city, zip), Item(item, price), Order(order_id, cust), OrderItem(order_id, item, qty) □ BCNF: □□□□□□□□ FD-□□ determinant □□□ super key (□□□□□□□□□□ 3NF □□□□□□□)□

6) □□□□ □□□□□□□□□□□□□□ □ □□□□□□□□□□: Primary/Secondary, Clustered, B+Tree/Hash□

□□□□□: File organization: Heap, Sequential, Hash□ Index: Primary/Secondary; Clustered (□□□□-□□□□□□□□□ □□□□ □□□) □□□□ Non-clustered□ B+Tree: □□ □□ □□□-□□□□□□, □□□□□-□□□□□□□□□□ □□□□□□□□; □□□□□□□ O(log_b N) □ Hash Index: □□□□□□□□ □□□□□ O(1) □□□□□□□□□□□; □□□□□□□ □□□□□□□□□□□□

7) Query Processing/Optimization □ Cost Estimation□

□□□□□: □□□: Parse→Rewrite→Plan enumerate→Cost estimate→Choose plan□ □□□ □□□□□□: □□□□□□□□□□□□□ s (0..1); σ_{A=v}(R) ⇒ |R|×s; Nested-loop join □□□ ≈ |R| + |R|×|S|; Hash join ≈ |R|+|S|; Sort-merge ≈ sort(R)+sort(S)+|R|+|S|□

8) Transaction □ ACID; Concurrency Control (Lock-based, 2PL)□

□□□□□: ACID: Atomicity, Consistency, Isolation, Durability□ Schedules: Conflict/ View serializable□ Lock-based CC: Shared/Exclusive; Compatibility matrix; Two-Phase Locking (2PL): □□□□□□-□□□□□□□□ □□ □□□□□□, □□□□□□-□□□□□□□□ □□□□□□ Cascading abort □□□□□□ Strict 2PL□

9) Deadlock Handling DBMS-□ □ Recovery□

□□□□□: Wait-for graph □□□□□ □□□□□□; □□□□-□□□; victim □□□□□□□□ □□□ □□□□□□□□□ Recovery: Write-Ahead Logging (WAL), Checkpoint, ARIES (Analysis→Redo→Undo)□

10) SQL □□□□□□: □□□□□, □□□□□□□□□□□□, □□□□□□□, □□□□□□□, □□□□□□□□

□□□□□/□□□□□□□: CREATE TABLE Student(id INT PRIMARY KEY, name VARCHAR(40), dept VARCHAR(20)); CREATE TABLE Course(cid INT PRIMARY KEY, title VARCHAR(60)); CREATE TABLE Enroll(id INT REFERENCES Student(id), cid INT REFERENCES Course(cid), grade CHAR(2), PRIMARY KEY(id,cid)); -- □□□□□□□ CREATE INDEX idx_student_dept ON Student(dept); -- □□□□□□□ SELECT s.name, c.title FROM Student s JOIN Enroll e ON s.id=e.id JOIN Course c ON c.cid=e.cid WHERE s.dept='CSE'; -- □□□□□□□ (□□□□□) CREATE TRIGGER trg_grade_check BEFORE INSERT ON Enroll FOR EACH ROW BEGIN IF NEW.grade NOT IN ('A','B','C','D','F') THEN SIGNAL 'Invalid grade'; END IF; END;

(d) Software Engineering

1) Software Engineering [illegible] SDLC [illegible]

[illegible]: [illegible] [illegible] [illegible] [illegible]/[illegible] SDLC: Requirement→Design→Implementation→Testing→Deployment→Maintenance[illegible]

2) Requirement Engineering: [illegible] [illegible] [illegible]

[illegible]: Functional vs Non-functional (Performance, Security, Usability)[illegible] Elicitation [illegible]: Interview, Workshop, Observation, Questionnaire, Prototyping[illegible] Specification: SRS [illegible]; Validation: Review/Prototyping; Change management[illegible]

3) [illegible] [illegible]: Context, DFD, State, UML (Use-case/Sequence/Class)[illegible]

[illegible]: Context diagram[illegible] [illegible]-[illegible] Actor[illegible] DFD-[illegible] [illegible]-[illegible]/[illegible]/[illegible] State machine [illegible]-[illegible] [illegible]; UML Use-case [illegible], Sequence [illegible], Class [illegible]

4) [illegible] [illegible] [illegible] (OOD) [illegible] [illegible] [illegible]

[illegible]: SOLID, Encapsulation, Inheritance, Polymorphism[illegible] [illegible] [illegible]: Singleton, Factory, Adapter, Observer, Strategy, MVC[illegible]

5) Real-Time [illegible] Reuse-oriented Design[illegible]

[illegible]: Real-time-[illegible] [illegible] [illegible] [illegible] (Rate-Monotonic, EDF)[illegible] Reuse-oriented-[illegible] [illegible]/[illegible]/[illegible] [illegible], [illegible] [illegible] [illegible] [illegible]

6) Dependability: Fault tolerance, [illegible]

[illegible]: Redundancy, Exception handling, Recovery blocks, N-version programming; [illegible] CIA (Confidentiality-Integrity-Availability), Threat modelling[illegible]

7) Maintenance [illegible] [illegible] [illegible] [illegible]

[illegible]: Corrective, Adaptive, Perfective, Preventive[illegible] Version control (Git), Build/Release, Change control board (CCB), CI/CD[illegible]

8) Verification [illegible] Validation; [illegible] [illegible] [illegible] [illegible]

[illegible]: Verification=“[illegible] [illegible] [illegible]?”, Validation=“[illegible] [illegible] [illegible]?”[illegible] [illegible]: Unit→Integration→System→Acceptance[illegible] [illegible]: Equivalence partitioning, Boundary value, Decision table, Path/Cyclomatic complexity based[illegible]

9) □□□□□□□□□□ □□□□□□□□□: Halstead, Cyclomatic Complexity□

□□□□□: Halstead: n1(#distinct operators), n2(#distinct operands), N1, N2 ⇒ Program length $N=N1+N2$; Vocabulary $n=n1+n2$; Volume $V=N\times\log_2 n$; Difficulty $D=(n1/2)\times(N2/n2)$; Effort $E=D\times V$□ Cyclomatic complexity: $V(G)=E-N+2P$ (□□□□ □□□□□□□□ □□□□□□□□ □□□□□ □□□□□□□) ⇒ □□□□□□-□□□□□□ □□□□□□ □□□□□□

10) COCOMO □□□/□□□□ □□□□□□□; Reliability/Availability□

□□□□□: Basic COCOMO: Effort($a\times(KLOC)^b$), Development time=$c\times(Effort)^d$; Organic/Semi-detached/Embedded-□ a,b,c,d □□□□□□□ Reliability: MTBF=MTTF+MTTR; Availability: A=MTTF/(MTTF+MTTR) ≈ (1−□□□□□□□□□)□

11) Quality Assurance (QA) □□□□□□□□□□□□□

□□□□□: □□□ □□□□□□, □□□□□□□□□□ □□□□□□□□□□□□□□, □□□□□□-□□□□□□□□, □□□□□□□□□□□□□□, □□□□□□□□□□□□, □□□□, □□□□□□□□ □□□□□□□□□□□□□□□, □□□□□□ □□□□□□□□□□□□□□□□□