CS Exit Ethiopia

Revision Outlines

Computer Science Exit Examination

1 selected course | 18 November 2026

Selected courses

01. Design and Analysis of Algorithms

https://csexitethiopia.com

abelasnake2003@gmail.com | Telegram: @Abechu7

Course contents

Each course begins on a new page. PDF bookmarks provide question, study-note section and revision-topic navigation.

Design and Analysis of Algorithms 3

Design and Analysis of Algorithms

Revision Outlines

Problem specification and correctness | Exam focus

- Problem specification and algorithm properties
  - Inputs, output, constraints
  - Empty-solution policy
  - Finite, definite, executable methods
  - Source-language independence
  - Feasible: satisfies constraints
  - Optimal: best feasible
  - Conditional branches: legitimate control flow
- Correctness proof obligations
  - Invariant initialization
  - Invariant preservation
  - Invariant at termination
  - Recursion: base cases
  - Recursion: inductive reduction
  - Testing finds failures
  - Finite tests ≠ universal correctness proof

Cost models and input size | Exam focus

- Cost models and input size
  - Array length n
  - Graph vertices V, edges E
  - Integer bit length
  - RAM: bounded-word arithmetic/access constant
  - Arbitrary-precision arithmetic costs
  - Long-output printing costs
  - Elapsed time: hardware, compiler, cache
  - I/O and competing load effects
- Analysis methods and resource metrics

- Experimental/dynamic: actual runs
- Asymptotic: abstract growth
- Probabilistic: expectations
- No universal taxonomy exclusion
- Graph/dynamic analysis labels not inherently invalid
- Auxiliary space excludes input
- I/O complexity: transfers
- Representation affects costs
- Data type complexity: nonstandard metric name

Input cases versus bounds | Exam focus

• Input-case definitions
- Fixed input size n
- Best: minimum cost
- Worst: maximum cost
- Average: specified input distribution
- Randomized expectation: internal choices
- One fixed input possible
- Smallest input ≠ best case

• Bounds independent of cases
- O/Ω/Θ: function bounds
- Independent of case or resource
- Worst upper bound: all size-n inputs
- O ≠ worst case
- Constant best case not universal
- Quadratic worst case not universal

Formal asymptotic bounds | Exam focus

• Formal asymptotic relations
- Eventually nonnegative f,g
- O: $f(n) \leq c \cdot g(n)$, $n \geq n_0$
- Fixed $c>0$
- Ω: reversed inequality

- Θ: both bounds
- o: f/g→0
- ω: f/g→∞
- Ratio existence condition
- Θ: growth up to constant factors
- No exact-seconds/equal-case implication

• Asymptotic proof scope
- Whole tail inequality
- Selected values insufficient
- Transitive upper-bound chains
- Nonnegative sums: largest growth dominates

• Worked example Bound witnesses and incorrect inequality
- $10n+5 \leq 15n$, $n \geq 1$
- $c=15, n_0=1$
- $3n^2+4n+1 \leq 8n^2$, $n \geq 1$
- $7n^2+3n+5 \leq 8n^2$: false at $n=1$
- Correct witness: $c=15, n_0=1$

Growth ordering and loose bounds | Exam focus

• Representative growth ordering
- $1<\log n<n<n \log n$
- $n \log n<n^2<n^3<\ldots$
- Fixed $b>1$: polynomial$<b^n<n!$
- Log bases: constant-factor difference
- $n \log n, n^2$: superlinear
- $\log n$: sublinear
- Polynomial: fixed input-size exponent

• Loose bounds and finite-input speed
- $O(n^2)$ implies $O(n^3)$
- Upper bounds needn't be tight
- Multiple valid offered bounds
- Growth ordering alone: insufficient finite-speed prediction
- Big-O alone: insufficient finite-speed prediction

- Worked example Ratio comparison example
  - $f=4n^2$; $g=8n$
  - $f/g=n/2\to\infty$
  - $f\in\Omega(g)$
  - $f\in\omega(g)$
  - $f\notin O(g)$
  - $f\notin\Theta(g)$

Loop counts and additive costs | Exam focus

- Loop-count arithmetic
  - Sequential costs add
  - Independent nested counts multiply
  - Dependent loops require sums
  - Constant body × n: $\Theta(n)$
  - $\Sigma_{i=1}^{n} i=n(n+1)/2=\Theta(n^2)$
  - Doubling until $i\geq n$: $\lceil \log_2 n \rceil$
  - Doubling formula condition: $n>1$
  - 10n loop: linear
  - Constant-cost output assumption
- Expected total and exact counts
  - Fixed size: m operations
  - Each expected cost f(n)
  - Linearity: expected total=m f(n)
  - Independence unnecessary
  - Changing sizes: $\Sigma f(n_i)$
  - Tests/increments/arithmetic/assignments affect coefficients
- Worked example Exact loop-count convention
  - Two initializations
  - n+1 tests
  - n increments
  - 2n body operations
  - One return
  - Total: 4n+4

- Different primitives change exact count
- Growth remains $\Theta(n)$

Conditional loops and malformed code | Exam focus

- Conditional nested-loop summation
  - $i=0..n-1$; $j=i..i^2-1$
  - $k=0..j-1$ only when $j\%i==0$
  - $i=0,1$: empty j ranges
  - No remainder by zero evaluated
  - $i\geq 2$: $j=i,2i,\ldots,(i-1)i$
  - Inner work=$i\Sigma_{m=1}^{i-1}m$
  - $i^2(i-1)/2=\Theta(i^3)$
  - Sum over i: $\Theta(n^4)$
  - Condition checks: $\Theta(n^3)$
- Loop assumptions and malformed-code traps
  - Offered n^5,n^7: valid loose upper bounds
  - Neither bound tight
  - Mathematical integers
  - Constant-cost output
  - Signed C++ i*i overflow: undefined
  - Mixed-language syntax: repair before runtime claim
  - Parameter n redeclaration: syntax defect
  - return @;3: syntax defect
  - Syntax defects ≠ logarithmic algorithms

Peak storage and recursion | Exam focus

- Peak storage accounting
  - Simultaneously live storage
  - Cumulative allocations ≠ peak space
  - Requested input/auxiliary/stack convention
  - n active constant frames: $\Theta(n)$
  - log n constant frames: $\Theta(\log n)$
  - Total calls ≠ peak recursion depth

- Hidden storage and time-space tradeoffs
  - Array slicing/copying overhead
  - Index-based pseudocode may omit copies
  - Tail-call elimination: language/runtime-dependent
  - Ordinary C++/Java: no assumed elimination
  - Cached subproblem results save time
  - Caching consumes memory

Recurrences and substitution | Exam focus

- Recurrence construction and expansion
  - Every recursive call
  - Nonrecursive work
  - Base case
  - $T(n)=T(n-1)+\Theta(1)\rightarrow\Theta(n)$
  - $T(n)=T(n/2)+\Theta(1)\rightarrow\Theta(\log n)$
  - $T(n)=2T(n/2)+\Theta(n)$
  - Logarithmic levels × $\Theta(n)$
  - Result: $\Theta(n \log n)$
- Substitution proof method
  - Guess bound
  - Induction with constants
  - Base-case verification
  - Lower-order correction may close induction
  - Unequal/decrementing recurrences: separate analysis

Master theorem and recursion trees | Exam focus

- Master theorem and recursion tree
  - $T(n)=aT(n/b)+f(n)$
  - $a\geq1$; $b>1$
  - $p=\log_b a$
  - Level i: a^i nodes
  - Node size: n/b^i
  - Polynomially smaller f: $\Theta(n^p)$

- f=Θ(n^p): Θ(n^p log n)
- Polynomially larger f: Θ(f(n))
- Case 3 regularity: a f(n/b)≤c f(n)
- Fixed c<1

• Master theorem limitations
- Log-factor boundary: extension or summation
- State base work
- Include leaf costs
- T(n)=T(n−1)+n: outside this version
- Unequal-split quicksort: outside this version

• Worked example Master theorem worked cases
- $4T(n/2)+n \to \Theta(n^2)$, case 1
- $2T(n/2)+n \to \Theta(n \log n)$, case 2
- $T(n/2)+n^2 \to \Theta(n^2)$, case 3
- $4T(n/2)+n^2 \to \Theta(n^2 \log n)$, case 2

Comparison-sorting lower bounds | Exam focus

• Comparison-sort decision-tree lower bound
- Distinct-key orderings: n! leaves
- Height h: at most 2^h leaves
- $h \geq \log_2(n!) = \Omega(n \log n)$
- Mergesort matches worst-case bound
- Array indexing allowed

• Lower-bound model boundaries
- Counting/radix: structured keys
- Outside comparison-only model
- Recurrences: particular algorithm analysis
- Decision tree: entire comparison class
- Supplied inappropriate-technique justifications: all unsound

Choosing algorithm design strategies | Exam focus

• Design strategies from structural conditions
- Divide-and-conquer: smaller largely independent instances

- Combine subproblem results
- DP: reuse defined subproblem states
- Greedy: feasible safe local commitment
- Backtracking: grow/retract when constraints fail
- Branch-and-bound: best possible objective test
- Prune when incumbent cannot be beaten

• Ambiguous strategy descriptions
- Subproblems/choices/incremental/optimization: several strategies
- Exponential runtime insufficient for method choice
- Specify overlap
- Specify local-choice safety
- Specify revisability and exactness
- Specify usable bounds
- Module selection-sort classification: erroneous
- Ordinary selection: repeatedly choose one minimum

Binary search and paired extrema | Exam focus

• Binary-search invariant and costs
- Sorted random-access data
- Inclusive low/high bounds
- mid=low+(high−low)//2
- high=mid−1 or low=mid+1
- Stop: found or low>high
- $T(n)=T(n/2)+\Theta(1)$
- Worst: $\Theta(\log n)$; best: $\Theta(1)$
- Iterative auxiliary space: $\Theta(1)$
- Recursive auxiliary space: $\Theta(\log n)$

• Simultaneous minimum/maximum comparisons
- Two unconditional comparisons per element
- Scan comparisons: $2n-2$, every order
- Pair elements: compare within pair
- Smaller→minimum; larger→maximum
- Even n: $3n/2-2$ comparisons

- Balanced recursion: $2T(n/2)+\Theta(1)=\Theta(n)$
- Two-if scan: no $n-1$ best case

Merge sort | Exam focus

- Merge-sort invariant and procedure
  - Split by indices
  - Recursively sort halves
  - Temporary result: smaller remaining front
  - Output prefix sorted
  - Prefix contains smallest consumed elements
  - Append unexhausted suffix
  - Stable ties: left half first
- Ordinary array merge-sort analysis
  - $T(n)=2T(n/2)+\Theta(n)=\Theta(n \log n)$
  - All input orders
  - Auxiliary buffer: $\Theta(n)$
  - Recursion depth: $\Theta(\log n)$
  - $+O(n)$ alone: upper bound only
  - Tightness requires linear merge work
  - Base cases: empty/singleton subarrays
- Worked example Merge-sort worked trace
  - $[4,1,3,2] \rightarrow [4,1]$ and $[3,2]$
  - Sorted halves: $[1,4]$ and $[2,3]$
  - Select 1,2,3; append 4
  - Original sortedness unnecessary

Quicksort variants and costs | Exam focus

- Quicksort partition variants
  - Choose pivot, partition, recurse
  - Two pointers: advance i below pivot
  - Retreat j above pivot
  - $i \le j$: swap, advance both
  - Bound scans

- Equal values: ensure progress
- Recurse on resulting ranges
- Hoare boundaries: pivot not necessarily final
- Lomuto: pivot commonly final
- Keep return conventions distinct

• Quicksort costs and refinements
- Balanced splits: $\Theta(n \log n)$
- Highly unbalanced splits: $\Theta(n^2)$
- Random pivots: expected $\Theta(n \log n)$
- Expectation ≠ deterministic guarantee
- Median-of-three retains adversarial cases
- Three-way partition: repeated keys
- Standard quicksort: unstable
- Expected stack $\Theta(\log n)$; worst $\Theta(n)$
- Recurse smaller side, iterate larger
- Refined stack bound: $O(\log n)$

Sorting choices and costs

• Elementary sorting choices
- Insertion: larger sorted-prefix keys shift
- Insertion: strict comparisons preserve stability
- Optimized bubble: adjacent inversion swaps
- Bubble stop: no-swap pass
- Insertion/bubble: $\Theta(n)$ best, $\Theta(n^2)$ worst
- Selection: repeated suffix minimum
- Selection: $\Theta(n^2)$ comparisons every order
- Selection: normally unstable
- All three: $\Theta(1)$ auxiliary array space

• Heap, merge, quicksort tradeoffs
- Max-heap construction: $\Theta(n)$
- Extract/restore: $O(\log n)$ per item
- Heapsort total: $O(n \log n)$
- Heapsort: $\Theta(1)$ auxiliary space

- Heapsort: ordinarily unstable
- Heapsort ≠ standard divide-and-conquer
- Mergesort: stability/worst-case guarantees
- Quicksort practical advantage: implementation/workload-dependent

Strassen matrix multiplication | Exam focus

- Matrix multiplication divide/combine costs
  - Ordinary n×n: n products/sums per entry
  - Ordinary multiplication: $\Theta(n^3)$
  - Eight products: $8T(n/2)+\Theta(n^2)=\Theta(n^3)$
  - Strassen: seven block products
  - Block sums and differences
  - $7T(n/2)+\Theta(n^2)=\Theta(n^{\log_2 7})$
  - Strassen exponent: approximately 2.807
- Strassen assumptions and tradeoffs
  - Zero-padding preserves required product
  - Smaller exponent versus extra additions
  - Temporary storage overhead
  - Possible numerical issues
  - Ordinary base case: small matrices
  - Specify exact versus floating-point arithmetic

Maximum contiguous subarray | Exam focus

- Maximum subarray by divide-and-conquer
  - Contiguous subarray ≠ arbitrary subsequence
  - Compare best left/right/crossing
  - Crossing: left suffix+right prefix
  - Best crossing found in $\Theta(n)$
  - $2T(n/2)+\Theta(n)=\Theta(n \log n)$
- Kadane dynamic programming
  - $end[i]=max(A[i],end[i-1]+A[i])$
  - Keep largest ending sum
  - Time: $\Theta(n)$

- Extra space: Θ(1)
- Nonempty initialization: A[0], not zero
- All-negative result: largest element
- Empty allowed: zero possible
- Unqualified strategy question: D&C and DP

• Worked example Maximum-subarray examples
- [−2,3,−1,4,−5] ending: −2,3,2,6,1
- Optimum 6: [3,−1,4]
- [−4,−2]: nonempty answer −2

Greedy proofs and counterexamples | Exam focus

• Greedy safety and exchange proofs
- Locally attractive feasible choice
- Commit without later undoing
- Optimal substructure alone insufficient
- Exchange greedy choice into optimum
- No worsening of cost
- Recurse after exchange
- MST cut proof: exchange argument

• Greedy limits and valid applications
- Simple decisions, low overhead
- Irrevocable errors
- Problem-specific applicability
- Greedy heuristic ≠ approximation guarantee
- Coins {1,3,4}; target 6
- Largest-first: 4+1+1; optimum: 3+3
- Activity selection: earliest finish
- Maximize count: nonoverlapping unweighted intervals
- Exchange first activity for earlier finish
- At least as much remaining room

Fractional versus indivisible knapsack | Exam focus

• Fractional knapsack greedy rule

- Positive weights/profits
- Divisible items
- Decreasing value/weight order
- Whole fitting items, then fitting fraction
- Exchange lower-density for higher-density weight
- Value cannot decrease
- Sorting O(n log n); scan linear
- Zero-weight beneficial items: handle separately
- Avoid division by zero

• Indivisibility breaks ratio proof
- Capacity 10
- Items (6,12),(5,8),(5,8)
- Ratio-greedy value: 12
- 0/1 optimum: 8+8=16
- Specify fractional/0/1/unbounded knapsack
- Feasibility ≠ optimality

• Worked example Fractional knapsack worked trace
- Weights 15,10,18; values 24,15,25
- Capacity 20
- Take weight 15 item
- Take half weight 10 item
- Value=24+7.5=31.5
- Initial capacity 20 ≠ item count

Unit-time job sequencing | Exam focus

• Unit-job scheduling assumptions and rule
- One machine
- Initially available jobs
- Unit durations; integer deadlines
- Profit only for on-time completion
- Nonnegative-profit jobs: decreasing profit
- Latest free slot finishing by deadline
- No feasible slot: reject

- Latest placement preserves tighter-deadline slots
- Profit-first exchange for feasible unit-job sets

• Job sequencing complexity and limitations
- Sorting plus slot scans
- O(n log n+nD)
- Cap D at n
- Equal-profit ordering arbitrary
- Earlier deadline tie rule optional
- Variable durations invalidate rule
- Release times invalidate rule

• Worked example Job sequencing worked certificate
- Slots 1…5: J_2,J_4,J_3,J_5,J_1
- Profits: 180+300+190+120+200=990
- All deadlines hold
- J_6 cannot finish by 2
- Five largest profits: 990 upper bound
- Matching bound certifies optimum

Optimal merging and Huffman coding | Exam focus

• Optimal merge pattern
- Combine two smallest weights
- Add sum to total
- Reinsert sum
- Min-heap: O(n log n) time
- Space: O(n)
- p+q: moved-record cost
- Nonempty lists: p+q−1 worst comparisons
- Actual comparisons may be fewer

• Huffman prefix coding and proof
- Frequencies as weights
- 0/1 branches to leaf symbols
- Prefix-free code
- Cost=Σfrequency×depth

- Two least frequencies: deepest siblings
- Exchange, contract, induct
- Ties change codes, preserve optimum cost
- Header overhead may outweigh tiny-file savings

• Worked example Merge/Huffman worked weights
- Weights 2,3,5,6
- 2+3=5
- 5+5=10
- 10+6=16
- Total weighted merge cost: 31
- Same construction: minimum weighted prefix-code length

MST algorithms and cut proofs | Exam focus

• MST properties and uniqueness
- Connected weighted undirected graph
- Spans vertices without cycles
- Minimum total edge weight
- V−1 edges
- Negative edges allowed
- Equal weights: possible multiple MSTs
- Distinct weights suffice for uniqueness
- A tree is a graph

• Prim, Kruskal, cut-exchange proof
- Prim: cheapest tree-to-outside crossing edge
- Kruskal: globally sorted edges
- Union-find: accept different-component endpoints
- Light cut edge exchange into optimum
- No increased total weight
- Matrix Prim: $O(V^2)$
- Binary-heap Prim: O(E log V)
- Kruskal: O(E log E)
- Dense-graph comparison: implementation-dependent
- Disconnected input: minimum spanning forest

- Worked example MST triangle examples
  - AB=1,BC=2,AC=4
  - Kruskal accepts AB,BC
  - Total cost: 3
  - All edges 1: any two optimal
  - Equal-weight example: uniqueness fails

Dijkstra shortest paths | Exam focus

- Dijkstra initialization and relaxation
  - dist[source]=0; others ∞
  - Predecessors initially unset
  - Extract minimum-distance unsettled u
  - dist[v]=min(dist[v],dist[u]+w(u,v))
  - Improvement: update predecessor
  - Minimum ∞: remaining vertices unreachable
  - Stop after destination finalized
  - Discovery alone insufficient for stopping
- Dijkstra proof, failure, costs
  - Nonnegative edges preserve finalized distances
  - Later unsettled paths cannot improve finalized
  - s→a=2; s→b=5; b→a=−4
  - Finalize a=2: miss later distance 1
  - Binary-heap/list: O((V+E)log V)
  - Matrix scanning: $O(V^2)$
  - MST: total connection cost
  - Dijkstra: each source-path distance

DP states and correctness | Exam focus

- Dynamic-programming specification and proof
  - State meaning and choices
  - Recurrence and bases
  - Dependency order
  - Better subsolution must improve full solution

- Failed replacement: missing state information
- Cover all legal final choices
- Generate feasible candidates
- Induct in dependency order
- Store reconstruction choices

• DP signatures and complexity
- Overlapping states motivate reuse
- Optimal substructure supports optimization
- DP also counts and decides
- Optimization not universally required
- Multiple users per state not required
- Cost=states×work per state
- DP may remain exponential

Memoization, tabulation, Fibonacci | Exam focus

• Memoization versus tabulation
- Memoization: recursive cache check
- Store solved state results
- Tabulation: explicit dependency order
- Chosen domain may exclude irrelevant states
- Cache keys: all relevant parameters
- Recursion/locality/sparsity affect cost

• Fibonacci reuse and arithmetic assumptions
- Naive branches recompute F(n−2)
- Naive call count: Θ(φ^n)
- O(2^n): valid loose bound
- F(0)=0; F(1)=1
- F(i)=F(i−1)+F(i−2)
- Memo/tabulation: Θ(n) unit-cost additions
- Two recent values: Θ(1) machine words
- Arbitrary precision: growing bit storage
- Naive overlap ≠ independent D&C ideal

DAG paths and Bellman-Ford | Exam focus

- DAG/multistage shortest-path DP
  - Reverse topological cost-to-go
  - F(t)=0
  - $F(u)=\min_{(u,v)}[w(u,v)+F(v)]$
  - Missing routes: ∞
  - Record minimizing next vertex
  - Alternative: forward source-distance propagation
  - Adjacency-list complexity: Θ(V+E)
  - Negative edges harmless without cycles
  - Module forward/backward labels inconsistent
  - Follow recurrence dependencies
- Bellman-Ford and strategy ambiguity
  - General graphs
  - Relax all edges: up to V−1 passes
  - Additional pass: reachable negative-cycle test
  - Complexity: O(VE)
  - Single-source shortest path ≠ uniquely greedy
  - Graph assumptions select algorithm
- Worked example DAG cost-to-go example
  - s→a=2,s→b=5,a→t=4,b→t=−1
  - F(a)=4; F(b)=−1
  - F(s)=min(6,4)=4
  - Optimal route via b

Floyd-Warshall and reachability | Exam focus

- Floyd-Warshall initialization and invariant
  - All-pairs distances
  - D[i,i]=0
  - D[i,j]: minimum direct-edge weight
  - Missing edges: ∞
  - Negative self-loop: min(0,weight)
  - k outermost

- D[i,j]=min(D[i,j],D[i,k]+D[k,j])
- Before phase k: earlier intermediates only
- Guard unreachable sums
- Guard numeric overflow

• Floyd-Warshall scope and reachability variant
- Time: $\Theta(V^3)$
- Storage: $\Theta(V^2)$
- Negative diagonal: negative cycle
- Pairs traversing cycle: no finite minimum
- Warshall reachability: OR/AND replace min/+
- Floyd cost ≠ universal APSP optimum
- Sparse nonnegative graph: repeated Dijkstra
- Other models permit other methods

• Worked example Floyd-Warshall improvement retention
- D[a,b]=4; D[b,c]=2
- Initial D[a,c]=11
- Phase b: D[a,c]→6
- Later phases retain improvement
- Module reprints 11: incorrect

0/1 knapsack and dominance | Exam focus

• 0/1 knapsack state and recurrence
- Integer capacity W
- K[i,w]: first-i maximum within capacity w
- Base: K[0,w]=0
- $\text{weight}_i > w$: K[i,w]=K[i−1,w]
- Otherwise $\max(K[i-1,w], \text{value}_i + K[i-1, w-\text{weight}_i])$
- Previous row: each item at most once
- Backtrack changed values: recover items

• Knapsack complexity and dominance
- Time O(nW): pseudo-polynomial
- Binary W encoding: O(log W) bits
- Storage: O(W)

- 0/1 updates: capacities downward
- Upward updates: item reuse, unbounded variant
- Sparse states: (weight,profit)
- Dominating state: no greater weight
- Dominating state: no smaller profit
- Same remaining items required

• Worked example Knapsack optimum and dominance example
- Weights 2,3,4; values 1,2,5
- Capacity W=6
- Choose weights 2+4
- Optimal value: 6
- (weight 5,profit 3) dominated
- Dominating state: (weight 4,profit 5)

LCS and reconstruction | Exam focus

• LCS prefix-state recurrence
- Subsequence: order preserved, gaps allowed
- L[i,j]: prefix lengths i,j
- Empty-prefix rows/columns: 0
- Compare X[i−1] with Y[j−1]
- Match: L[i,j]=1+L[i−1,j−1]
- Mismatch: max(L[i−1,j],L[i,j−1])
- Short-note correction: prefix lengths versus indices

• LCS evaluation and reconstruction
- Increasing rows/columns
- Matching characters: diagonal reconstruction
- Otherwise follow equal-optimum neighbour
- Ties: different valid LCSs
- Time: Θ(mn)
- Full table: Θ(mn) storage
- Value-only two rows: O(min(m,n))
- Reconstruction: retained decisions or another technique

• Worked example LCS versus longest common substring

- X=ABC; Y=AC
- LCS=AC; length=2
- Substring matching: different recurrence
- Substring mismatch resets to zero

Matrix-chain and optimal-BST interval DP | Exam focus

• Matrix-chain interval DP
- A_i dimensions: p[i−1]×p[i]
- Base: M[i,i]=0
- M[i,j]=min_{i≤k<j}(M[i,k]+M[k+1,j]+p[i−1]p[k]p[j])
- Increasing interval length
- Store optimal split
- 10×30,30×5,5×60
- (AB)C: 4500; A(BC): 27000
- Time $O(n^3)$; space $O(n^2)$
- Fixed multiplication order; variable parenthesization

• Optimal-BST interval DP
- Sorted-key successes p_i; gap failures q_i
- W(i,j)=Σ_{k=i}^j p_k+Σ_{k=i−1}^j q_k
- Comparison-count base: C(i,i−1)=0
- C(i,j)=W(i,j)+min_r(C(i,r−1)+C(r+1,j))
- Store optimal root r
- CLRS dummy-depth empty cost: q_{i-1}
- Final-value difference: fixed total gap weight
- Basic time $O(n^3)$; space $O(n^2)$

Reliability design under budget | Exam focus

• Reliability redundancy model
- Independent device failures
- m_i parallel copies; device reliability r_i
- Stage $\varphi_i(m_i)=1-(1-r_i)^{m_i}$
- Series total: $\prod\varphi_i$
- Integer copies $m_i \geq 1$

- Budget: $\Sigma m_i c_i \leq B$
- Extra copies: reliability versus budget
- Correlated failures invalidate independence formula

• Reliability budget DP
- R[i,b]: first-i best within budget b
- Base: R[0,b]=1
- R[i,b]=max_m R[i−1,b−mc$_i$]φ$_i$(m)
- Feasible m≥1
- Empty candidate set: 0
- Time O(stages×budget×maximum copies)
- Value storage: O(budget)
- Retain copy counts for reconstruction

• Worked example Reliability allocation comparison
- Costs 30,15,20; reliabilities 0.9,0.8,0.5
- Budget 105
- Copies (1,2,2): cost 100
- Reliability: 0.9×0.96×0.75=0.648
- Copies (1,1,3): cost 105
- Reliability only 0.63

Exact TSP subset DP | Exam focus

• Exact TSP subset-state recurrence
- Minimum tour: every vertex once, return
- Shortest path needn't cover/return
- Fix start s
- F(S,j): minimum from s through S
- Exactly S; ending j∈S
- Base: F({s},s)=0
- F(S,j)=min_{i∈S−{j}}[F(S−{j},i)+w(i,j)]
- Finish: min_{j≠s}[F(V,j)+w(j,s)]
- Missing edges: ∞

• Held-Karp costs and alternatives
- Store predecessors for tour reconstruction

- Time: $O(V^2 2^V)$
- Space: O(V 2^V)
- Improves factorial enumeration
- Still exponential
- Nearest-neighbour: heuristic, no general exactness
- Backtracking/branch-and-bound: exact TSP alternatives
- Generic method selection needs details

• Held-Karp boundary conditions
- S must contain start s
- Initialize F({s},s)=0
- Impossible states: ∞
- Larger sets: endpoints j≠s
- Avoid nonexistent predecessor states

BFS, DFS, graph representations

• BFS versus DFS discovery
- BFS: mark on enqueue
- FIFO neighbour exploration
- Record predecessor and level
- Levels: minimum edge counts
- Unweighted/equal-weight shortest paths
- DFS: mark on entry
- Recursive or iterative branch completion
- Unvisited neighbour before parent resumption
- Resume parent after branch

• Graph traversal conditions and costs
- Disconnected: restart each unvisited vertex
- Adjacency-list time: O(V+E)
- Auxiliary visited/queue/stack: O(V)
- Matrix scanning: $O(V^2)$
- Exact order requires neighbour order
- Reverse DFS finish order: DAG-only topological
- Printed under DP ≠ DP optimization

Articulation, bridges, biconnected components

- Articulation discovery and low values
  - Removal increases connected-component count
  - Undirected DFS
  - disc[u]: discovery time
  - low[u]: smallest reachable discovery number
  - Reachability from u's subtree
  - Downward tree edges; at most one back-edge
  - Update from children/nonparent back edges
  - Root articulation: ≥2 DFS children
  - Nonroot articulation: some low[v]≥disc[u]
- Articulation, bridges, biconnected blocks
  - Child cannot bypass u to ancestor
  - Bridge: low[v]>disc[u]
  - Block trigger: low[v]≥disc[u]
  - Edge stack: pop through (u,v)
  - Adjacency-list time: O(V+E)
  - Vertex-biconnectivity ≠ directed strong connectivity
- Biconnected-component DFS procedure
  - Simple undirected graph
  - On entry: disc[u]=low[u]=++time
  - Unvisited v: push (u,v), DFS(v)
  - Then low[u]=min(low[u],low[v])
  - Visited nonparent ancestor: disc[v]<disc[u]
  - Push back edge once
  - Back-edge update: low[u]=min(low[u],disc[v])
  - low[v]≥disc[u]: pop through (u,v)
  - Separate root-child articulation rule
  - Bridge test: low[v]>disc[u]
  - Restart each unvisited vertex
  - Multigraph: skip parent edge by identity
  - Do not skip every parent-directed edge
- Worked example Two-triangle low-link check

- Triangles A-B-C-A and C-D-E-C
- DFS order: A,B,C,D,E
- disc=(1,2,3,4,5)
- low=(1,1,1,3,3)
- Only articulation: C
- No bridges
- Blocks: {A-B,B-C,C-A}; {C-D,D-E,E-C}
- low[D]=disc[C]
- Articulation ≥ versus bridge >

Backtracking and safe pruning | Exam focus

- Backtracking candidate template
  - Partial assignment x[1..k]
  - For each allowed choice: assign
  - Test possible extension
  - Promising: recurse
  - Return: undo assignment
  - Report complete feasible candidates
  - Stop after one: existence/single-answer request
  - Explicit constraints: variable domains
  - Implicit constraints: relations among choices
- Backtracking completeness and pruning
  - Consider every feasible extension
  - Prune only impossible branches
  - Disliked partial objective insufficient
  - DFS space proportional to depth
  - Total search may remain exponential
  - Equivalent-state memoization reduces work
  - Stronger constraints reduce work
  - Branch count: actual choices/tests

N-queens | Exam focus

- N-queens representation and conflict tests

- x[r]=column; one queen per row
- Earlier row i: conflict comparison
- Reject column conflict: x[r]=x[i]
- Reject diagonal: $|x[r]-x[i]|=|r-i|$
- Distinct rows built into representation
- Legal placement: recurse, then remove
- Column and r−column/r+column sets
- Set-based conflict tests: constant time

• N-queens bounds and conventions
- Distinct columns: at most n! permutations
- Additional partial-node work
- Pruning ≠ polynomial-time guarantee
- Placement/conflict storage: O(n)
- n=2,3: no solutions
- State empty-board convention

• Worked example N-queens placement examples
- n=4: (2,4,1,3) valid
- Rows 1,2; columns 2,3
- Row difference=column difference=1
- Diagonal conflict: reject partial branch

Subset sum | Exam focus

• Subset-sum include/exclude pruning
- Track current sum s
- Track remaining sum R
- Nonnegative weights assumption
- Prune s>target
- Prune s+R<target
- Equality: report subset
- All subsets: consider zero-weight extensions
- Sorted positives strengthen tests
- Sorting cannot force next-item inclusion

• Subset-sum boundary assumptions and bounds

- Negative values: overshoot may recover
- Use valid attainable-range bounds
- At most 2^n subsets
- Recursion depth: O(n)
- Duplicate values: define index-distinct counting

• Worked example Subset-sum feasibility bounds example
- {3,4,5,6}; target 13
- Solution: {3,4,6}
- s=9,R=3: 12<13, failure
- s=9,R=4: equality still possible

Graph coloring | Exam focus

• Fixed-m graph coloring search
- Assign colours 1…m
- Reject already-coloured neighbour's colour
- All colours fail: backtrack
- At most m^V complete assignments
- Additional conflict-checking cost
- Simple test: O(V)
- Exponential bound
- Module polynomial-looking expression: garbled
- Self-loop: proper coloring impossible

• Coloring decision versus optimization
- Decision: do m colours suffice?
- Chromatic number: minimum colours
- Increasing m with complete decision search
- Greedy: vertex-order-dependent colour choice
- Greedy may use too many
- No general greedy optimum proof
- Odd cycle: 3 colours
- Edge-containing bipartite graph: 2 colours
- Edgeless nonempty graph: 1 colour

Hamiltonian cycles | Exam focus

- Hamiltonian cycle backtracking
  - Fix start: eliminate rotational duplicates
  - Extend to unused adjacent vertex
  - At V vertices: require return edge
  - Missing return edge: backtrack
  - Hamiltonian: visit vertices once
  - Euler: visit edges
  - Fixed-start candidates: up to (V−1)!
  - Polynomial checking overhead
- Hamiltonian necessary conditions and TSP
  - Simple undirected graph; V≥3
  - Degree 1 rules out Hamiltonian cycle
  - Articulation point rules out Hamiltonian cycle
  - Absence of obstructions insufficient
  - TSP: optimize weighted Hamiltonian tour
  - Existence search alone doesn't minimize cost
  - Undirected reversal equivalence for unique enumeration

Branch-and-bound states and bounds | Exam focus

- Branch-and-bound node lifecycle and selection
  - Live: generated, eligible for expansion
  - E-node: currently expanded
  - Dead: finished or pruned
  - FIFO: queue order
  - LIFO: stack order
  - Least-cost/best-bound: priority queue
  - Selection affects work/storage
  - Valid bounds preserve exactness
- Branch-and-bound bound directions
  - Minimization incumbent: feasible upper bound U
  - Partial lower bound L: every completion
  - One optimum/value: prune L≥U

- Maximization incumbent: feasible lower bound
- Optimistic completion: upper bound
- Maximization prune: upper≤incumbent
- All optima: retain equality branches

Knapsack branch and bound | Exam focus

- Knapsack fractional-relaxation bound
  - Positive weights/values
  - Sort decreasing density
  - Branch include/exclude
  - Reject overweight nodes
  - Fill remaining capacity fractionally
  - Fractional fill: remaining items only
  - Relaxed profit: upper bound
  - Whole-item completion updates incumbent
  - Fractional result ≠ legal 0/1 answer
- Knapsack branch-and-bound certificate
  - Capacity 10; (6,12),(5,8),(5,8)
  - Root bound: 12+(4/5)8=18.4
  - First included: whole-item value 12
  - First excluded: other two value 16
  - All live upper bounds≤16: optimality
  - Avoid downward floating-point bound rounding
  - Worst-case search: exponential

TSP matrix-reduction branch and bound | Exam focus

- Directed-TSP matrix-reduction bound
  - Diagonal/forbidden entries: ∞
  - Subtract finite row minima
  - Then subtract finite column minima
  - Accumulate reductions
  - Tour: one entry per row/column
  - Reduction sum: lower bound

- Relaxed assignment may contain disjoint subtours

• TSP include-edge reduction and search
  - Include i→j: remove row i/column j
  - Prevent premature subtours
  - Reduce residual matrix
  - Child bound=parent+selected reduced cost+new reductions
  - Compare with best complete tour
  - Expand promising live node
  - Matrix zeros ≠ valid complete tour
• TSP branching and subtour prevention
  - Branch on undecided i→j
  - Include: delete row i/column j
  - Include bound: parent+reduced edge+new reductions
  - Exclude: set entry ∞
  - Exclude bound: parent+new reductions
  - Forced in/out-degree at most one
  - Track forced path head h, tail t
  - Forbid t→h before all vertices included
  - Residual required row/column without choice: infeasible
  - Prune infeasible child
  - Accept one complete all-vertex cycle
  - Reject disjoint cycles
• Worked example TSP subtour procedure check
  - Forced 1→2→1 and 3→4→3
  - One incoming/outgoing each: insufficient
  - Two disjoint 2-cycles ≠ tour
  - Forced 1→2: forbid 2→1
  - Retain prohibition while path size<4
  - Existing three-city bound=25
  - Tour cost=25

P, NP, reductions, encoding | Exam focus

• P and NP definitions

- Decision problems
- P: deterministic polynomial input-bit time
- NP: polynomial-length certificates
- Polynomial-time verification
- Equivalent nondeterministic polynomial-time machine
- NP ≠ non-polynomial
- P⊆NP
- P=NP remains open

• Reductions, hardness, encoding size
- $A \leq_p B$: polynomial many-one instance transformation
- Preserve yes/no answers
- Prove B hard: known-hard A→B
- Reverse direction insufficient
- NP-complete: NP-hard and in NP
- Optimization TSP: NP-hard
- Bounded-tour decision TSP: NP-complete
- Tour/cost verification versus global-optimality certification
- O(nW): pseudo-polynomial knapsack
- Not polynomial in W's bit length

Damaged items and required clarifications

• Conditional-output damaged item
- p=10,n=5
- if(n>p) false: output skipped
- Execution terminates
- No implied feasibility/finiteness/control-flow violation
- Efficiency needs specification
- Required output needs specification

• Missing notation and syntax clarifications
- 3n?+7n+12: missing exponent
- Precise degree undetermined
- Fixed nonnegative integer exponents: polynomial
- Stray ini/inl: repair syntax

- Redeclared n: repair syntax
- Exact counts need primitive-cost model
- Marked key cannot replace missing clarification