Design and Analysis of Algorithms

Short notes for B.Tech CSE / IT · BCA · MCA · GATE CS

Uncodemy Handwritten Notes · free for students

1. Introduction

- **Algorithm** = a finite set of clear steps that solves a problem.
- Properties: input, output, definiteness, finiteness, effectiveness.
- **Analysis** = finding how much time and memory an algorithm needs as input size n grows.
- Cases: **best**, **average**, **worst**. Exams mostly ask the worst case.

➤ Asymptotic notations

Notation	Meaning	Definition (for n >= n0)
Big-O	Upper bound	f(n) <= c·g(n)
Big-Omega	Lower bound	f(n) >= c·g(n)
Theta	Tight bound	c1·g(n) <= f(n) <= c2·g(n)
Little-o	Strict upper	f(n) / g(n) → 0
Little-omega	Strict lower	f(n) / g(n) → infinity

- Order: 1 < log log n < log n < √n < n < n log n < n^2 < n^3 < 2^n < n! < n^n

TIP To compare two functions, take log of both. Example: n^(log n) vs 2^n → (log n)^2 vs n, so 2^n grows faster.

2. Recurrence Relations

- **Substitution**: guess and prove by induction.
- **Recursion tree**: draw cost at each level and add.
- **Master theorem** for T(n) = a·T(n/b) + f(n), a >= 1, b > 1.

FORMULA Compare f(n) with n^(log_b a). Case 1: f smaller (polynomially) → Θ(n^(log_b a)). Case 2: equal → Θ(n^(log_b a) · log n). Case 3: f larger (and regularity holds) → Θ(f(n)).

➤ Solved examples

Recurrence	a, b, n^(log_b a)	Result
T(n) = 2T(n/2) + n	2, 2, n	Θ(n log n)
T(n) = T(n/2) + 1	1, 2, 1	Θ(log n)
T(n) = 8T(n/2) + n^2	8, 2, n^3	Θ(n^3)
T(n) = 2T(n/2) + n^2	2, 2, n	Θ(n^2)
T(n) = 7T(n/2) + n^2	7, 2, n^2.81	Θ(n^2.81)

- Not master form: T(n) = T(n − 1) + n gives Θ(n^2). T(n) = 2T(n − 1) + 1 gives Θ(2^n).

EXAM POINT Master theorem does not apply to T(n) = 2T(n/2) + n/log n (gap is not polynomial). Its answer is Θ(n log log n).

3. Divide and Conquer

- Steps: **divide** the problem, **conquer** sub-problems recursively, **combine** answers.

Algorithm	Recurrence	Time
Binary search	T(n) = T(n/2) + 1	O(log n)
Merge sort	T(n) = 2T(n/2) + n	O(n log n)
Quick sort (best/avg)	T(n) = 2T(n/2) + n	O(n log n)
Quick sort (worst)	T(n) = T(n−1) + n	O(n^2)
Strassen matrix multiply	T(n) = 7T(n/2) + n^2	O(n^2.81)
Max and min of array	T(n) = 2T(n/2) + 2	3n/2 − 2 comparisons

➤ Quick sort

- Pick a pivot, put smaller elements left and larger right (partition), recurse.
- Worst case when the array is already sorted and the first/last element is the pivot.
- Randomised pivot gives expected O(n log n).

Merge sort

- Stable, needs O(n) extra space. Merging two sorted lists of size m and n takes at most m + n − 1 comparisons.

REMEMBER Any comparison-based sorting needs Ω(n log n) comparisons in the worst case.

4. Greedy Method

- Make the locally best choice at each step and never go back.
- Works when the problem has the **greedy choice property** and **optimal substructure**.

Fractional knapsack

- Sort items by profit / weight ratio, take the highest first, take a fraction of the last one.

EXAMPLE Capacity W = 50. Items (profit, weight): A (60, 10), B (100, 20), C (120, 30). Ratios 6, 5, 4. Take A fully (10), B fully (30 total), then 20/30 of C = 80. Profit = 60 + 100 + 80 = 240.

Job sequencing with deadlines

- Sort jobs by profit (high to low). Put each job in the latest free slot before its deadline.

Huffman coding

- Repeatedly join the two smallest frequencies. Gives an optimal prefix code.
- Example frequencies a:5, b:9, c:12, d:13. Join 5 + 9 = 14, then 12 + 13 = 25, then 14 + 25 = 39.
- Codes length: a, b, c, d all get 2 bits here. Total bits = (5 + 9 + 12 + 13) x 2 = 78.

➤ Minimum Spanning Tree (MST)

Prim's	Kruskal's
Grows one tree from a start vertex	Picks edges in increasing weight
Uses min-heap / priority queue	Uses sorting + Union-Find
O(E log V) with binary heap	O(E log E)
Good for dense graphs	Good for sparse graphs

- An MST of V vertices has V − 1 edges. If all edge weights are distinct, the MST is unique.

➤ Dijkstra's shortest path

- Single source, non-negative weights only. Time O((V + E) log V) with a binary heap.
- Relaxation: if d[u] + w(u, v) < d[v] then d[v] = d[u] + w(u, v).

5. Dynamic Programming (DP)

- Solve each sub-problem once and store the answer (**memoization** top-down or **tabulation** bottom-up).
- Needs **optimal substructure** and **overlapping sub-problems**.

Greedy	Dynamic Programming
One choice per step	Tries all choices, keeps best
Fast, may not be optimal	Always optimal (if DP applies)
Fractional knapsack	0/1 knapsack

➤ Classic DP problems

Problem	Time
0/1 knapsack	O(nW)
Longest Common Subsequence (LCS)	O(mn)
Matrix chain multiplication	O(n^3)
Floyd–Warshall (all pairs shortest path)	O(V^3)
Bellman–Ford (handles negative edges)	O(VE)
Travelling salesman (DP)	O(n^2 · 2^n)

FORMULA LCS: if x[i] == y[j], L[i][j] = L[i-1][j-1] + 1, else L[i][j] = max(L[i-1][j], L[i][j-1]).

EXAMPLE LCS of "ABCB" and "BDCAB": one LCS is "BCB", length 3.

➤ Matrix chain multiplication example

- A1 (10 x 20), A2 (20 x 30), A3 (30 x 40).
- (A1 A2) A3 = 10·20·30 + 10·30·40 = 6000 + 12000 = 18000.
- A1 (A2 A3) = 20·30·40 + 10·20·40 = 24000 + 8000 = 32000.
- Best = 18000 multiplications with (A1 A2) A3.

FORMULA m[i][j] = min over k of (m[i][k] + m[k+1][j] + p(i-1) · p(k) · p(j)).

6. Backtracking and Branch and Bound

- **Backtracking**: build a solution step by step, undo a step when it cannot lead to a solution. Uses DFS.
- Examples: N-Queens, sum of subsets, graph colouring, Hamiltonian cycle.
- **Branch and Bound**: explores a state-space tree using bounds to prune. Uses BFS / least cost. Example: 0/1 knapsack, TSP.

EXAMPLE 4-Queens has 2 solutions: (2, 4, 1, 3) and (3, 1, 4, 2) (column of queen in each row).

7. Graph Algorithms (summary)

Algorithm	Purpose	Time
BFS	Traversal, unweighted shortest path	O(V + E)
DFS	Traversal, cycle detection	O(V + E)
Topological sort	Order of a DAG	O(V + E)
Dijkstra	Single source, non-negative	O((V + E) log V)
Bellman–Ford	Single source, negative edges, detects negative cycle	O(VE)
Floyd–Warshall	All pairs	O(V^3)
Prim / Kruskal	MST	O(E log V)
Strongly connected components	Kosaraju / Tarjan	O(V + E)

- Topological sort exists only for a DAG (directed acyclic graph).
- Bellman–Ford runs V − 1 rounds of relaxing every edge.

8. Complexity Classes

- **P**: problems solvable in polynomial time.
- **NP**: solutions can be verified in polynomial time.
- **NP-hard**: every NP problem reduces to it in polynomial time.
- **NP-complete** = in NP and NP-hard.
- P is a subset of NP. Whether P = NP is still unknown.

NP-complete examples	In P examples
SAT, 3-SAT	2-SAT
Vertex cover, Clique	Shortest path
Hamiltonian cycle, TSP (decision)	MST
0/1 knapsack (decision), Subset sum	Sorting

REMEMBER To prove X is NP-complete: show X is in NP, then reduce a known NP-complete problem to X.

9. Most Asked Questions (practice)

1. Define Big-O, Omega and Theta with graphs and examples.
2. Solve recurrences using the master theorem and recursion tree.
3. Explain quick sort, its worst case and how to avoid it.
4. Solve a fractional knapsack and a job sequencing problem.
5. Find the MST using Prim's and Kruskal's algorithms.
6. Apply Dijkstra's algorithm on a given graph. Why does it fail with negative edges?
7. Solve 0/1 knapsack and LCS using DP tables.
8. Find the optimal parenthesization for matrix chain multiplication.
9. Explain N-Queens using backtracking.
10. Define P, NP, NP-hard and NP-complete with examples.

TIP GATE loves recurrence solving, MST edge questions, Dijkstra traces and DP table filling. Always write the table step by step.