Z5007: Algorithms & Data Structures Master Reference

Programming and Data Structures (Weeks 1–17) · Odd Semester 2026-27 · innocent@iitmz.ac.in

1. Asymptotic Bounds & Master Theorem

- **Big-O (O):** $f(n) \le c\,g(n)$ for $n \ge n_0$ (upper bound).
- **Big-Omega (Ω):** $f(n) \ge c\,g(n)$ for $n \ge n_0$ (lower bound).
- **Big-Theta (Θ):** Tight bound (O and Ω).
- **Growth Order:** $1 < \log\log n < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < 2^n < n!$.

Master Theorem for $T(n) = aT(n/b) + \Theta(n^d)$ $(a \ge 1, b > 1)$:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } d < \log_b a \\ \Theta(n^d \log n) & \text{if } d = \log_b a \\ \Theta(n^d) \& \text{if } d > \log_b a \end{cases}$$

Amortized Dynamic Array Doubling: Resizing capacity $1 \to 2 \to 4 \to \cdots \to n$ copies $< 2n$ items total, yielding $O(1)$ amortized `append()`.

2. Python Built-In Operation Complexities

Structure	Operation	Cost
`list`	Index `a[i]`, `append`, `pop()`	$O(1)$ amort.
`list`	`pop(0)`, `insert(0,x)`, `x in a`	$O(n)$
`deque`	`appendleft`, `popleft`, `append`	$O(1)$ worst
`dict/set`	Lookup, insert, delete	$O(1)$ avg / $O(n)$
`heapq`	`heappush`, `heappop`; `heapify`	$O(\log n)$; $O(n)$

3. Sorting Algorithms & Lower Bounds

Algorithm	Best	Avg	Worst	Space / Stable
Insertion	n	n^2	n^2	$O(1)$ / Yes
Selection	n^2	n^2	n^2	$O(1)$ / No
Merge Sort	$n\log n$	$n\log n$	$n\log n$	$O(n)$ / Yes
Quick Sort	$n\log n$	$n\log n$	n^2	$O(\log n)$ / No
Heap Sort	$n\log n$	$n\log n$	$n\log n$	$O(1)$ / No
Counting	$n+k$	$n+k$	$n+k$	$O(k)$ / Yes
Radix (d)	$d(n+k)$	$d(n+k)$	$d(n+k)$	$O(n+k)$ / Yes

Comparison Lower Bound: Any comparison sort requires $\Omega(\log_2(n!)) = \Omega(n\log n)$ comparisons in the worst case.

4. Trees, AVL Rotations & Binary Heaps

- **Traversals:** Inorder (L-N-R, sorted on BST), Preorder (N-L-R), Postorder (L-R-N), Level-order (BFS).
- **AVL Balance Factor:** $bf(u) = h(u.\text{left}) - h(u.\text{right}) \in \{-1, 0, 1\}$. Guaranteed height $h \le 1.44\log_2(n+2)$.
- **Rotations:** **LL** → right rotate; **RR** → left rotate; **LR** → left child then right root; **RL** → right child then left root.
- **Binary Max-Heap (0-indexed):** Parent $(i-1)//2$, Left $2i+1$, Right $2i+2$. Floyd `heapify` builds in $O(n)$ via $\sum_{h=0}^{\lfloor \log n \rfloor} \frac{n}{2^{h+1}} h = O(n)$.

5. Hash Tables & Disjoint Set Union (DSU)

- **Load Factor:** $\alpha = n/m$. Chaining expected probe cost: $1+\alpha$ (miss), $1+\alpha/2$ (hit).
- **Open Addressing (Linear Probing):** $(h(k)+i) \bmod m$; expected miss $\approx \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$; requires tombstones on delete.
- **DSU:** Path compression + union by rank gives $O(\alpha(n))$ amortized per `find`/`union`, where $\alpha(n) \le 4$.

6. Graph Algorithms ($|V| = V, |E| = E$)

Algorithm	Problem / Requirement	Time
BFS	Unweighted shortest path	$O(V+E)$
DFS	Cycles, SCC, Topo sort	$O(V+E)$
Kahn's	DAG topological order	$O(V+E)$
Dijkstra	Non-negative weights ($w \ge 0$)	$O((V+E)\log V)$
Bellman-Ford	Negative weights, cycle check	$O(VE)$
Kruskal / Prim	Minimum Spanning Tree (MST)	$O(E\log V)$

Cut Property (MST): For any cut $(S, V \setminus S)$, the minimum-weight edge crossing the cut belongs to an MST (basis of Kruskal and Prim).

7. Greedy & Dynamic Programming Templates

- **Activity Selection:** Sort by earliest finish time f_i; pick next compatible start $s_j \ge f_{\text{last}}$. Time $O(n\log n)$.
- **0/1 Knapsack (n items, capacity W):** $dp[w] = \max(dp[w], v_i + dp[w-w_i])$ iterating $w = W \ldots w_i$ **backwards**. Time $O(nW)$, space $O(W)$.
- **Unbounded Knapsack / Coin Change:** Iterate $w = w_i \ldots W$ **forwards**.
- **LCS ($X_{1..m}, Y_{1..n}$):** If $X_i = Y_j$: $1 + dp[i-1, j-1]$; else $\max(dp[i-1, j], dp[i, j-1])$. Time $O(mn)$.
- **Edit Distance (Levenshtein):** Match costs 0; replace/insert/delete cost $1 + \min(dp[i-1, j-1], dp[i, j-1], dp[i-1, j])$.
- **LIS:** $O(n^2)$ DP or $O(n\log n)$ patience-sorting with `bisect_left`.

8. String Matching, Tries & Complexity Classes

- **KMP Prefix Table $\pi[i]$:** Length of longest proper prefix of $P[0..i]$ matching a suffix. Builds in $O(m)$, searches text T in $O(n+m)$ without backtracking in T.
- **Trie (Prefix Tree):** Lookup/insert word of length L in $O(L)$ independent of dictionary size n.
- **Rabin-Karp:** Rolling polynomial hash gives $O(n+m)$ expected, $O(nm)$ worst case.
- **P vs. NP:** Decision problems verifiable in poly-time are in **NP**. $L \in$ **NPC** iff $L \in$ **NP** and every $A \in$ **NP** reduces $A \le_p L$.
- **Classic Reductions:** 3-SAT $\le_p$ Clique $\le_p$ Vertex Cover $\le_p$ Set Cover.

9. Quick Proof & Exam Checklist

- **Loop Invariant:** State Initialization, Maintenance, and Termination explicitly before concluding correctness.

- **Greedy Exchange Argument:** Assume an optimal schedule differs from greedy at earliest step k; swap to show equal or better cost without breaking feasibility.
- **DP Design Steps:** (1) Subproblem definition, (2) Recurrence, (3) Base cases, (4) Iteration order, (5) Complexity & parent-pointer reconstruction.
- **Negative-Weight Trap:** Never run Dijkstra on negative edges; run Bellman-Ford ($V - 1$ relaxations + 1 cycle check pass).