

# Algorithm Design & Analysis Cheat Sheet

## 1. Sorting Algorithms (Comparisons, Swaps & Stability)

Algorithm	Best Case Time	Worst Case Time	Key Metric & Notes
Insertion Sort	$O(n)$	$O(n^2)$	Best: $n-1$ comps, 0 swaps. Worst: $\sim n^2/2$ comps/swaps. Stable: Yes.
Merge Sort	$O(n \log n)$	$O(n \log n)$	Recurrence (Worst): $W(n) = 2W(n/2) + n - 1$ . Space: $O(n)$ . Stable: Yes.
Quick Sort	$O(n \log n)$	$O(n^2)$	Worst: Pivot is consistently Min/Max. Comparisons: $n(n-1)/2$ . Stable: No.
Heap Sort	$O(n \log n)$	$O(n \log n)$	Heapify: $O(n)$ . deleteMax: $O(\log n)$ . Total: $O(n \log n)$ . Stable: No.

## 2. Recurrence Relation Analysis (Week 8)

### A. Master Theorem: $T(n) = aT(n/b) + f(n)$

Case	Condition	Result
1 (Leaves Win)	$f(n) = O(n^{(\log_b a - \epsilon)})$	$\Theta(n^{(\log_b a)})$
2 (Tie)	$f(n) = \Theta(n^{(\log_b a)} \cdot \log^k n)$	$\Theta(n^{(\log_b a)} \cdot \log^{k+1} n)$
3 (Root Wins)	$f(n) = \Omega(n^{(\log_b a + \epsilon)})$	$\Theta(f(n))$

### B. Summation Formulas (Iteration Method)

Type	Pattern / Work	Formula / Result	Big-Theta
AP (Arithmetic)	$1 + 2 + \dots + n$	$n(n+1)/2$	$\Theta(n^2)$
GP (Decaying)	Ratio $r < 1$	Sum approx. First Term	$\Theta(\text{largest term})$
GP (Growing)	Ratio $r > 1$	Sum approx. Last Term (Leaves)	$\Theta(\text{leaves cost})$

## 3. Graph & Tree Algorithms (Weeks 4-7)

Algorithm	Time Complexity	Key Restriction / Purpose
BFS / DFS	$O(V + E)$	Traversal (visiting all nodes).
Topological Sort	$O(V + E)$	Must be a DAG. Uses DFS (reverse finish time).
Dijkstra's	$O(E \log V)$	No negative weights. Finds SSSP.
Prim's (MST)	$O(E \log V)$	Grows a tree from one starting node. Works with negative weights.
Kruskal's (MST)	$O(E \log E)$	Processes edges globally. Uses Union-Find for cycle detection.
Union-Find Ops (WQU+PC)	$O(M \log^* N) \sim O(M)$	Cycle detection for Kruskal's.

## 4. Dynamic Programming & Greedy (Weeks 9-11)

# Algorithm Design & Analysis Cheat Sheet

Problem	Approach	Time Complexity	Notes
Fractional Knapsack	Greedy	$O(n \log n)$	Sort by Value/Weight ratio. Always optimal.
Interval Scheduling	Greedy	$O(n \log n)$	Sort by Earliest Finish Time. Always optimal.
0/1 Knapsack	DP	$O(n * W)$	Pseudo-polynomial. $W$ = capacity.
LCS	DP	$O(n * m)$	$n, m$ are string lengths. Recurrence involves $\max(\text{subproblems})$ .

## 5. String Matching Algorithms (Week 12)

Algorithm	Best Case Comps	Worst Case Comps	Notes
Brute Force	$O(n)$	$O(mn)$	Worst case: $m(n-m+1)$ comps.
Rabin-Karp	$O(n+m)$	$O(mn)$	Uses hashing (mod $q$ ) and radix ( $d$ ). Worst case: spurious hits.
Boyer-Moore	$O(n/m)$	$O(mn)$	Sublinear in best/average case. Uses charJump and matchJump.

## 6. Complexity Classes & Hard Problems (Week 13)

Class / Problem	Time Complexity	Classification
P (Deterministic Poly)	$O(n^k)$	P
NP (Nondeterministic Poly)	Unknown	NP
TSP (Decision)	Unknown (Worst: $O(n!)$ )	NP-Complete
Towers of Hanoi	$O(2^n)$	Exponential Time

*Disclaimer: These tables summarize theoretical worst-case complexities. Actual runtime depends on constants and input structure.*