Design and Analysis of Algorithms

Time and Space Complexity—Comparison Chart

ALGORITHM TECHNIQUES	NAME OF THE ALGORITHM	BEST CASE	AVERAGE CASE	WORST CASE	RECURRENCE RELATIONS / TIME BREAKDOWN	SPACE COMPLEXITY
DIVIDE & CONQUER	Binary Search	$\Omega(1)$	$\theta(\log n)$	$O(\log n)$	$T(n) = T(n/2) + O(1)$	$O(\log n)$ (recursive) or $O(1)$ (iterative)
	Merge Sort	$\Omega(n \log n)$	$\theta(n \log n)$	$O(n \log n)$	$T(n) = 2T(n/2) + O(n)$	$O(n)$ due to auxiliary space for merging
	Quick Sort	$\Omega (n \log n)$	$\theta(n \log n)$	$O(n^2)$	$T(n) = T(k) + T(n-k-1) + O(n)$	$O(\log n)$ (average case) to $O(n)$ (worst case) due to recursion stack
	Strassen's Matrix Multiplication	$\Omega (n^2.81)$	$\theta(n^2.81)$	$O(n^2.81)$	$T(n) = 7T(n/2) + O(n^2)$	$O(n^2)$ for storing matrices
	Convex Hull	$\Omega (n \log n)$	$\theta(n \log n)$	$O(n \log n)$	$T(n) = 2T(n/2) + O(n)$	$O(n)$ for sorting the points
GREEDY	Job Sequencing with Deadlines	$\Omega (n \log n)$	$\theta(n \log n)$	$O(n^2)$	Sorting jobs $O(n \log n)$, iterating $O(n^2)$ in worst case	$O(n)$ for job schedule
	Fractional Knapsack	$\Omega (n \log n)$	$\theta(n \log n)$	$O(n \log n)$	Sorting items $O(n \log n)$, then linear scan $O(n)$	$O(1)$ (if sorting is done in-place)
	Prim's Algorithm (using Min-Heap)	$\Omega (E \log V)$	$\theta(E \log V)$	$O(E \log V)$	Priority queue operations $O(E \log V)$	$O(V)$ for storing MST + $O(V^2)$ adjacency matrix (or $O(E + V)$ adjacency list)
	Kruskal's Algorithm	$\Omega (E \log E)$	$\theta(E \log E)$	$O(E \log E)$	Sorting edges $O(E \log E)$, Union-Find operations $O(E \log V)$	$O(E + V)$ using adjacency list and Union-Find structure
	Huffman Coding	$\Omega (n \log n)$	$\theta(n \log n)$	$O(n \log n)$	Sorting characters by frequency $O(n \log n)$, building tree $O(n \log n)$	$O(n)$ for the tree structure
	Single Source Shortest Path (Dijkstra's Algorithm)	$\Omega (V^2)$ (without Min-Heap)	$\theta(E \log V)$ (with Min-Heap)	$O(E \log V)$	Priority queue operations $O(E \log V)$	$O(V)$ for storing distances + $O(E + V)$ for graph representation
DYNAMIC PROGRAMMING	Optimal Binary Search Tree (OBST)	$\Omega (n^2)$	$\theta(n^2)$	$O(n^3)$	$T(i, j) = \min(T(i, r-1) + T(r+1, j)) + cost[i][j]$	$O(n^2)$ for DP table
	0/1 Knapsack Problem	$\Omega (nW)$	$\theta(nW)$	$O(nW)$	$T(i, w) = \max(T(i-1, w), T(i-1, w-wi) + vi)$	$O(nW)$ for DP table
	Traveling Salesman Problem (TSP)	$\Omega (n^2 * 2^n)$	$\theta(n^2 * 2^n)$	$O(n^2 * 2^n)$	$T(S, i) = \min(T(S-\{i\}, j) + dist(j, i))$	$O(n * 2^n)$ for DP table
	Ford-Fulkerson Algorithm	$\Omega(E*max\_flow)$	$\theta(E*max\_flow)$	$O(E*max\_flow)$	Augmenting path method iterating over edges	$O(V)$ (for residual graph representation)
BACKTRACKING	Eight Queens Problem	$\Omega (1)$	$\theta(n!)$	$O(n!)$	$T(n) = n * T(n-1) + O(n)$	$O(n)$ for board and recursion stack
	Sum of Subsets	$\Omega (1)$	$\theta(2^n)$	$O(2^n)$	$T(n) = 2T(n-1) + O(1)$	$O(n)$ for recursion stack
	Graph Coloring	$\Omega (1)$	$\theta(k^n)$	$O(k^n)$	$T(n) = k * T(n-1) + O(n)$	$O(n)$ for storing colors and recursion stack
	0/1 Knapsack (Backtracking)	$\Omega (1)$	$\theta(2^n)$	$O(2^n)$	$T(n) = 2T(n-1) + O(1)$	$O(n)$ for recursion stack
BRANCH & BOUND	0/1 Knapsack (Branch and Bound)	$\Omega (1)$	$\theta(2^n)$	$O(2^n)$	$T(n) = 2T(n-1) + O(1)$	$O(n)$ for recursion stack
	Traveling Salesman Problem (TSP)	$\Omega (n^2)$	$\theta(n^2 * 2^n)$	$O(n^2 * 2^n)$	$T(n) = (n-1) * T(n-1) + O(n)$	$O(n^2)$ for DP table, $O(n)$ for path storage