DS & Algo Interview Core Concepts

Part 1: Core Information & Usage

Concept	Key Clues/Trigger Words	Example Problems	When to Use	When NOT to Use
Hashing	"unique", "duplicates", "frequency", "count", "O(1) lookup", "anagram"	Two Sum, Valid Anagram, Group Anagrams	Need fast lookups, counting frequencies, detecting duplicates	When order matters more than speed → use Array/LinkedList; very limited memory → use Two Pointers
Array	"contiguous", "index", "subarray", "in-place", "constant space"	Maximum Subarray, Product of Array Except Self	Need random access, simple data structure	Dynamic sizing needed → use LinkedList/Vector; frequent insertions/deletions → use LinkedList
Two Pointers	"sorted array", "palindrome", "opposite ends", "meeting in middle"	Two Sum II, Valid Palindrome, Container With Most Water	Sorted arrays, palindromes, finding pairs	Unsorted data → use Hashing; need all combinations → use Backtracking
Tree	"hierarchical", "parent-child", "root", "leaf", "binary", "BST"	Binary Tree Inorder, Validate BST, Lowest Common Ancestor	Hierarchical data, searching, sorting	Linear data → use Array/LinkedList; memory constraints → use Array with parent pointers
Heap/Priority Queue	"kth largest/smallest", "top k", "priority", "merge k"	Kth Largest Element, Merge k Sorted Lists	Need extremes, priority processing, streaming data	Need all elements sorted → use Sorting; stable sorting required → use Merge Sort
Tries	"prefix", "dictionary", "autocomplete", "word search"	Implement Trie, Word Search II	String prefix operations, dictionaries	Few strings → use Hashing; memory-critical → use Compressed Trie/Suffix Array

Concept	Key Clues/Trigger Words	Example Problems	When to Use	When NOT to Use
Binary Search	"sorted", "find target", "O(log n)", "divide and conquer"	Search in Rotated Array, Find Peak Element	Searching in sorted space, optimization problems	Unsorted data → use Hashing; need all elements → use Linear Search
Sliding Window	"substring", "subarray", "window", "consecutive", "max/min in range"	Longest Substring Without Repeating, Max Sum Subarray of Size K	Contiguous subarrays/substrings with constraints	Non-contiguous elements → use Hashing/DP; complex patterns → use Backtracking
Linked List	"node", "pointer", "next", "cycle", "reverse"	Reverse Linked List, Detect Cycle, Merge Two Sorted Lists	Dynamic data, frequent insertions/deletions	Need random access → use Array; cache performance critical → use Array
Stack	"LIFO", "matching brackets", "undo", "function calls", "DFS"	Valid Parentheses, Daily Temperatures, DFS	Parsing, DFS, undo operations, matching patterns	Need random access → use Array; FIFO behavior needed → use Queue
Backtracking	"all combinations", "permutations", "generate all", "explore paths"	N-Queens, Sudoku Solver, Generate Parentheses	Generate all solutions, constraint satisfaction	Large search spaces without pruning → use DP/Greedy; time-critical → use Heuristics
Graph	"nodes", "edges", "connected", "path", "cycle", "BFS", "DFS"	Clone Graph, Course Schedule, Number of Islands	Modeling relationships, pathfinding	Simple linear/hierarchical → use Array/Tree; limited relationships → use simpler structures
1-D DP	"optimal substructure", "overlapping subproblems", "fibonacci", "stairs"	Climbing Stairs, House Robber, Coin Change	Optimization with overlapping subproblems	Greedy works → use Greedy; no overlapping subproblems → use simple recursion

Concept	Key Clues/Trigger Words	Example Problems	When to Use	When NOT to Use
2-D DP	"grid", "matrix", "paths", "edit distance", "LCS"	Unique Paths, Edit Distance, Longest Common Subsequence	2D optimization problems, grid-based problems	1D DP sufficient → use 1D DP; memory constraints → use space-optimized DP
Interval	"merge", "overlap", "schedule", "meeting rooms", "calendar"	Merge Intervals, Meeting Rooms, Insert Interval	Scheduling, time-based problems	Discrete events → use Hashing; non-interval data → use appropriate data structure
Greedy	"optimal choice", "local optimum", "activity selection", "minimum"	Activity Selection, Jump Game, Gas Station	Making locally optimal choices leads to global optimum	Need all possibilities → use DP/Backtracking; optimal substructure not greedy → use DP
Advanced Graphs	"shortest path", "MST", "topological sort", "strongly connected"	Dijkstra's Algorithm, Kruskal's Algorithm, Tarjan's Algorithm	Complex graph problems, optimization	Simple traversal sufficient → use BFS/DFS; performance not critical → use simpler algorithms

Part 2: Technical Details & Risk Management

Concept	Benefits	Risks	Prevention	Common Errors
Hashing	O(1) average lookup/insert, efficient counting	Hash collisions, extra space	Use good hash function, handle collisions	Forgetting to handle duplicates, not considering hash collisions
Array	Direct indexing O(1), cache-friendly, simple	Fixed size, insertion/deletion expensive	Validate bounds, consider overflow	Index out of bounds, off-by-one errors
Two Pointers	O(1) space, often O(n) time, elegant	Only works on sorted/special structures	Ensure correct pointer movement	Infinite loops, wrong convergence condition

Concept	Benefits	Risks	Prevention	Common Errors
Tree	Hierarchical representation, efficient search in BST	Can be unbalanced, recursive overhead	Balance trees, iterative alternatives	Stack overflow, null pointer access, wrong base cases
Heap/Priority Queue	O(log n) insert/delete, maintains order	Extra space, not stable	Use correct heap type (min/max)	Wrong heap type, not handling empty heap
Tries	Efficient prefix operations, space-sharing	High memory overhead for sparse data	Compress paths, consider alternatives	Memory leaks, incorrect termination flags
Binary Search	O(log n) time, works on large datasets	Requires sorted data	Handle edge cases, correct mid calculation	Infinite loops, wrong boundary updates
Sliding Window	O(n) time for subarray problems	Complex boundary management	Validate window constraints	Off-by-one errors, wrong window expansion/contraction
Linked List	Dynamic size, O(1) insertion/deletion	No random access, pointer management	Careful pointer manipulation	Null pointer dereference, losing references, memory leaks
Stack	Simple LIFO operations, function call simulation	Limited access pattern	Check empty before pop	Stack overflow, not handling empty stack
Backtracking	Explores all possibilities, memory efficient	Exponential time complexity	Prune early, optimize constraints	Not pruning invalid paths, incorrect state restoration
Graph	Models complex relationships	Can be complex, cycle detection needed	Use appropriate representation	Infinite loops in cycles, not marking visited
1-D DP	Avoids recomputation, optimal solutions	Space complexity, state design	Identify recurrence relation	Wrong state transitions, not handling base cases
2-D DP	Handles complex state relationships	High space complexity	Consider space optimization	Wrong dimensions, incorrect state transitions
Interval	Efficient interval operations	Sorting overhead, edge cases	Sort by start time, handle overlaps	Not sorting intervals, wrong overlap detection

Concept	Benefits	Risks	Prevention	Common Errors
Greedy	Simple and efficient	Doesn't always work, hard to prove correctness	Prove greedy choice property	Using when optimal substructure doesn't hold
Advanced Graphs	Solves complex graph problems	Complex implementation, high complexity	Understand algorithm requirements	Wrong algorithm choice, implementation bugs

Quick Reference Tips:

- **Time Complexity Hints**: O(1) → Hash, O(log n) → Binary Search/Heap, O(n) → Linear scan/Two pointers, O(n log n) → Sort-based, O(n²) → Nested loops/2D DP
- **Space Optimization**: Consider iterative vs recursive, rolling arrays for DP, in-place algorithms
- **Common Patterns**: Sort first, use auxiliary data structures, transform the problem
- **Interview Strategy**: Start with brute force, optimize step by step, discuss trade-offs