Algorithm Pattern Identification Guide

This guide helps you identify which algorithm pattern to use based on problem characteristics. For each pattern, we provide:

1. **How to Identify:** Key signs that suggest using this pattern
2. **Example Problem Types:** Typical problems that use this pattern
3. **Time & Space Complexity:** General complexity characteristics

Array Patterns

Sliding Window

How to Identify:

- Problems involving contiguous subarrays or substrings
- Finding max/min/sum over a contiguous sequence of fixed or variable size
- Problems asking for longest/shortest subarray with a given property

Example Problem Types:

- Maximum sum subarray of size K
- Longest substring with K distinct characters
- Minimum size subarray with a given sum
- String permutations or anagrams

Time & Space Complexity:

- Time: O(n) where n is array length (each element processed at most twice)
- Space: O(1) for fixed window, O(k) where k is window size for variable window

Two Pointers

How to Identify:

- Problems involving sorted arrays or linked lists
- Need to find pairs or triplets satisfying certain conditions
- Problems asking for in-place array modification
- Finding intersections or palindromes

Example Problem Types:

- Two Sum, Three Sum
- Container with Most Water
- Remove duplicates from sorted array
- Palindrome verification

Time & Space Complexity:

- Time: O(n) for most implementations or O(n^2) for nested two pointers
- Space: O(1) as typically implemented in-place

Kadane's Algorithm

How to Identify:

- Finding maximum/minimum sum subarray
- Requiring contiguous elements with optimal value
- When greedy approach with local/global optimal values works

Example Problem Types:

- Maximum sum subarray
- Maximum product subarray
- Circular array maximum sum

Time & Space Complexity:

- Time: O(n)
- Space: O(1)

Prefix Sums

How to Identify:

- Range sum queries
- Cumulative operations on arrays
- Checking for specific sum conditions over subarrays

Example Problem Types:

- Range sum queries
- Subarray sum equals K
- Count number of subarrays with specific properties

Time & Space Complexity:

- Time: O(n) for preprocessing, O(1) for queries
- Space: O(n) for storing prefix sums

Linked List Patterns

Fast & Slow Pointers

How to Identify:

- Cycle detection problems
- Finding middle element
- Finding nth element from the end
- Identifying if linked list has a cycle

Example Problem Types:

- Detect cycle in linked list
- Find cycle start point
- Find middle of linked list
- Palindrome linked list

Time & Space Complexity:

- Time: O(n)
- Space: O(1)

Linked List Reversal

How to Identify:

- Problems requiring reversal of all or part of a linked list
- Problems involving K-groups or alternative reverse operations

Example Problem Types:

- Reverse linked list
- Reverse nodes in K-group
- Reverse alternating K elements

Time & Space Complexity:

- Time: O(n)
- Space: O(1) for iterative solutions, O(n) for recursive solutions

Tree Patterns

Tree Traversal

How to Identify:

- Problems requiring visiting all nodes in a tree
- Node relationship problems
- Searching or collecting data from all nodes

Example Problem Types:

- Preorder, inorder, postorder traversal
- Level order traversal
- Path sum problems
- Tree serialization/deserialization

Time & Space Complexity:

- Time: O(n) where n is number of nodes
- Space: O(h) where h is tree height for recursion, O(n) worst case

Binary Search Tree

How to Identify:

- Ordered operations on trees
- Search, insertion, deletion with ordering requirements

- Validation of BST properties

Example Problem Types:

- Validate BST
- Insert/delete in BST
- Kth smallest element in BST
- Floor/ceiling values in BST

Time & Space Complexity:

- Time: O(h) where h is tree height
- Space: O(h) for recursion stack

Trie (Prefix Tree)

How to Identify:

- Dictionary operations on strings
- Prefix matching problems
- Problems involving character-by-character processing
- Autocomplete or spell-checker functionalities

Example Problem Types:

- Implement prefix tree
- Word search in a dictionary
- Autocomplete feature
- Longest common prefix

Time & Space Complexity:

- Time: O(m) for insertions and searches, where m is key length
- Space: O(n * m) where n is number of keys, m is average key length

Graph Patterns

Breadth-First Search (BFS)

How to Identify:

- Shortest path in unweighted graphs
- Level-by-level exploration
- Finding minimum steps to reach a target
- Connected components

Example Problem Types:

- Shortest path between two nodes
- Web crawler
- Connected components
- Level order traversal of tree

Time & Space Complexity:

- Time: O(V + E) where V is vertices and E is edges
- Space: O(V) for queue

Depth-First Search (DFS)

How to Identify:

- Exploring all possible paths
- Backtracking problems
- Cycle detection
- Topological sorting

Example Problem Types:

- Maze solving
- Generating all paths between two vertices
- Detecting cycles
- Islands in a grid (connected components)

Time & Space Complexity:

- Time: O(V + E) where V is vertices and E is edges
- Space: O(V) for recursion stack

Dijkstra's Algorithm

How to Identify:

- Finding shortest path in weighted graph with non-negative weights
- Optimizing distance between nodes
- Pathfinding with cost considerations

Example Problem Types:

- Network routing
- GPS navigation
- Flight scheduling

Time & Space Complexity:

- Time: $O((V + E) \log V)$ with binary heap implementation
- Space: $O(V)$

Union-Find (Disjoint Set)

How to Identify:

- Problems involving connected components
- Need to check if elements are in same set
- Dynamic connectivity problems
- Cycle detection in undirected graphs

Example Problem Types:

- Kruskal's algorithm for MST
- Find connected components
- Redundant connection detection

Time & Space Complexity:

- Time: $O(\alpha(n))$ amortized per operation where α is inverse Ackermann function
- Space: $O(n)$

Dynamic Programming Patterns

0/1 Knapsack Pattern

How to Identify:

- Discrete items with values/weights
- Binary decisions (include/exclude)
- Maximizing/minimizing value with constraints

Example Problem Types:

- 0/1 Knapsack
- Subset Sum
- Equal Subset Sum Partition
- Minimum Subset Sum Difference

Time & Space Complexity:

- Time: O(n * C) where n is items, C is capacity
- Space: O(n * C), can be optimized to O(C)

Unbounded Knapsack Pattern

How to Identify:

- Items can be used multiple times
- Selecting repeated elements with constraints
- Max/min value problems with unlimited supply

Example Problem Types:

- Coin Change (min coins)
- Coin Change II (number of ways)
- Rod Cutting
- Maximum ribbon cut

Time & Space Complexity:

- Time: O(n * C) where n is item types, C is capacity
- Space: O(C)

Longest Common Subsequence (LCS) Pattern

How to Identify:

- Problems comparing sequences
- Finding common elements or differences between strings
- Edit distance variations

Example Problem Types:

- Longest Common Subsequence
- Shortest Common Supersequence
- Edit Distance
- Longest Palindromic Subsequence

Time & Space Complexity:

- Time: O(m * n) where m, n are string lengths
- Space: O(m * n)

Fibonacci Sequence Pattern

How to Identify:

- Problems with recursive relation f(n) = f(n-1) + f(n-2)
- Current state depends on 1-2 previous states
- Counting distinct ways to reach a target

Example Problem Types:

- Fibonacci Numbers
- Staircase
- House Thief (similar to non-adjacent elements)
- Jump Game variations

Time & Space Complexity:

- Time: O(n)
- Space: O(n) can be optimized to O(1)

Backtracking Patterns

Subsets Pattern

How to Identify:

- Generating all possible subsets/combinations/permutations
- Need to explore multiple choices at each step
- Building combinations with specific constraints

Example Problem Types:

- Generate Subsets/Powerset
- Permutations
- Combinations
- Letter Combinations of Phone Number

Time & Space Complexity:

- Time: O(2^n) for subsets, O(n!) for permutations
- Space: O(n) for recursion stack

Constraint Satisfaction Pattern

How to Identify:

- Problems with complex constraints
- Search space can be pruned early
- Need to find all valid solutions or one valid solution

Example Problem Types:

- N-Queens
- Sudoku Solver
- Word Search

- Palindrome Partitioning

Time & Space Complexity:

- Time: Exponential, but pruning reduces actual runtime
- Space: O(n) for recursion stack

Heap Patterns

Top K Elements Pattern

How to Identify:

- Finding top/smallest K elements
- Stream processing with limited memory
- Maintaining a running set of maximum/minimum elements

Example Problem Types:

- Kth Largest Element
- K Closest Points to Origin
- Top K Frequent Elements
- Sort K-sorted Array

Time & Space Complexity:

- Time: O(n log k) for processing n elements with heap of size k
- Space: O(k) for the heap

Two Heaps Pattern

How to Identify:

- Median calculation problems
- Balancing elements on either side of a midpoint
- Processing stream data with statistics

Example Problem Types:

- Find Median from Data Stream
- Sliding Window Median
- IPO (maximize capital)

Time & Space Complexity:

- Time: O(log n) per element insertion
- Space: O(n) for storing all elements

Additional Patterns

Binary Search Variations

How to Identify:

- Sorted arrays or matrix
- Problems where search space can be halved each time
- Finding exact match or closest element
- Monotonically increasing/decreasing properties

Example Problem Types:

- Standard binary search
- Search in rotated sorted array
- Search for a range
- Find minimum in rotated sorted array

Time & Space Complexity:

- Time: O(log n)
- Space: O(1) iterative, O(log n) recursive

Greedy Algorithms

How to Identify:

- Local optimal choice leads to global optimum
- Problems where you can make choices without reconsidering

- Optimization problems with "obvious" next steps

Example Problem Types:

- Activity selection
- Huffman coding
- Fractional knapsack
- Interval scheduling

Time & Space Complexity:

- Time: Usually O(n log n) due to sorting
- Space: Usually O(1) or O(n)

Bit Manipulation

How to Identify:

- Problems involving binary representation
- XOR, AND, OR operations
- Problems requiring space optimization
- Numeric problems that can exploit bit properties

Example Problem Types:

- Counting bits
- Finding single number among duplicates
- Power set generation via bits
- Bit manipulation tricks

Time & Space Complexity:

- Time: O(1) to O(n) depending on problem
- Space: Usually O(1)

Problem-to-Pattern Matching Table

If you see this...	Consider this pattern...
Contiguous subarrays/substrings	Sliding Window

If you see this...	Consider this pattern...
Paired elements in sorted array	Two Pointers
Max/min subarray sum	Kadane's Algorithm
Range queries	Prefix Sums
Cycle detection	Fast & Slow Pointers
Dictionary operations	Trie
Shortest path (unweighted)	BFS
All possible paths	DFS
Shortest path (weighted)	Dijkstra's
Connect components	Union-Find
Choice of items under constraints	0/1 Knapsack
String comparison	Longest Common Subsequence
State depends on previous states	Fibonacci Pattern
Generating all combinations	Subsets/Backtracking
Top/smallest K elements	Heap
Median of stream	Two Heaps
Sorted array search	Binary Search
Local -> global optimization	Greedy
Binary operations	Bit Manipulation