**Arrays**

1. **Two Pointers**:
   * Used for problems involving subarrays, pairs, or reversing arrays.
   * Example: Find a pair with a given sum, and move zeros to the end.
2. **Sliding Window**:
   * Ideal for problems involving fixed-size subarrays or finding a subarray with a certain property.
   * Example: Maximum sum subarray of size k, longest substring without repeating characters.
3. **Prefix Sum**:
   * Helps in calculating the sum of elements in a subarray efficiently.
   * Example: Find subarray with a given sum, range sum query.

**Strings**

1. **Two Pointers**:
   * Useful for palindrome checking, reversing strings, and substring problems.
   * Example: Longest palindromic substring, reverse words in a string.
2. **Hashing**:
   * Efficient for counting characters, anagram checking, and finding the first non-repeating character.
   * Example: Valid anagram, group anagrams, first unique character.
3. **Dynamic Programming**:
   * Solves problems involving subsequences, substrings, and edit distance.
   * Example: Longest common subsequence, edit distance, longest palindromic subsequence.

**Linked Lists**

1. **Two Pointers (Fast and Slow)**:
   * Detect cycles, find the middle of the list, and remove the nth node from the end.
   * Example: Detect cycle in a linked list, and find the middle element.
2. **Reversal**:
   * Useful for problems requiring the reversal of a linked list or sublist.
   * Example: Reverse a linked list, reverse nodes in k-group.
3. **Merge**:
   * Often used in sorting or merging two sorted lists.
   * Example: Merge two sorted linked lists, and merge k sorted lists.

**Stacks**

1. **Monotonic Stack**:
   * Helps solve problems related to next/previous greater or smaller elements.
   * Example: Next greater element, daily temperatures.
2. **Parenthesis Matching**:
   * Used for validating and matching parentheses.
   * Example: Valid parentheses, longest valid parentheses.
3. **Undo Mechanism**:
   * Applications like browser history or text editor undo.
   * Example: Implement a basic calculator, and decode string.

**Queues**

1. **Sliding Window (Monotonic Queue)**:
   * Efficient for finding the maximum or minimum in a sliding window.
   * Example: Sliding window maximum, first negative integer in every window of size k.
2. **Breadth-First Search (BFS)**:
   * Used in graph traversal, shortest path in unweighted graphs, and level order traversal in trees.
   * Example: Shortest path in a grid, word ladder.

**Trees**

1. **Depth-First Search (DFS)**:
   * Preorder, inorder, and postorder traversals.
   * Example: Binary tree traversals, path sum.
2. **Breadth-First Search (BFS)**:
   * Level order traversal and finding the shortest path.
   * Example: Level order traversal, minimum depth of a binary tree.
3. **Dynamic Programming on Trees**:
   * Used for problems involving subtree properties.
   * Example: Maximum path sum, longest path in a tree.

**Graphs**

1. **Depth-First Search (DFS)**:
   * Used for connectivity, cycle detection, and topological sorting.
   * Example: Connected components, topological sort.
2. **Breadth-First Search (BFS)**:
   * Ideal for shortest path in unweighted graphs and level order traversal.
   * Example: Shortest path in an unweighted graph, bipartite graph checking.
3. **Dijkstra’s Algorithm**:
   * Finds the shortest path in weighted graphs.
   * Example: Shortest path in a weighted graph.
4. **Union-Find**:
   * Used for disjoint set operations and connectivity.
   * Example: Detect cycles in an undirected graph, Kruskal's algorithm for MST.

**Dynamic Programming (DP)**

1. **Memoization**:
   * Top-down approach storing results of subproblems.
   * Example: Fibonacci numbers, unique paths.
2. **Tabulation**:
   * Bottom-up approach filling a table iteratively.
   * Example: Knapsack problem, longest increasing subsequence.
3. **State Representation**:
   * Define states and transitions between states.
   * Example: Coin change, edit distance.

**Heaps**

1. **Priority Queue**:
   * Efficiently retrieve the smallest/largest element.
   * Example: Kth largest element in a stream, merge k sorted lists.
2. **Heap Sort**:
   * Sorting using a heap data structure.
   * Example: Sorting an array, scheduling problems.

**Backtracking**

1. **Combinatorial Problems**:
   * Generates all possible solutions and backtracks to find the correct ones.
   * Example: N-queens, permutations, and combinations.
2. **Constraint Satisfaction Problems**:
   * Solves problems by trying to build a solution incrementally.
   * Example: Sudoku solver, crossword puzzle.

**Greedy Algorithms**

1. **Activity Selection**:
   * Choosing the optimal solution at each step.
   * Example: Interval scheduling, Huffman coding.
2. **Fractional Knapsack**:
   * Optimization problems where local optimal choices lead to a global solution.
   * Example: Fractional knapsack problem, minimum cost to connect ropes.

**Recursion**

1. **Divide and Conquer**:
   * Divides a problem into smaller subproblems, solves them independently, and combines the results.
   * Example: Merge sort, quicksort.
2. **Tail Recursion**:
   * Optimized recursion where the recursive call is the last statement.
   * Example: Tail-recursive factorial calculation.