**Greedy Algorithm :**

A **greedy algorithm** builds up a solution piece by piece, choosing the locally optimal choice at each step, with the hope of finding the global optimum.

**Example:**

1. **Problem**: Given a set of activities with start and end times, select the maximum number of activities that don't overlap.
2. **Greedy Choice**: Always pick the activity that ends the earliest (this leaves more room for subsequent activities).
3. **Steps**:
4. Sort activities by finish time.
5. Pick the first activity, then keep picking the next activity that starts after the current activity finishes.
6. **Time Complexity**: (O(n \log n)) (due to sorting).
7. **Space Complexity**: (O(1)) (constant space apart from input).

|  |
| --- |
| **Algorithm** **Time Complexity** **Space Complexity** **Data Structure** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Kruskals Algorithm** | (O(E \log E)) | (O(V)) | Union-Find |
| **Prims Algorithm** | (O(E \log V)) | (O(V)) | Min Heap |
| **0/1 Knapsack (DP)** | (O(nW)) | (O(nW)) | 2D DP Table |
| **Optimal Merge Pattern** | (O(n \log n)) | (O(n)) | Min Heap |
| **Huffman Tree** | (O(n \log n)) | (O(n)) | Min Heap, Binary Tree |
| **Dijkstras Algorithm** | (O(E \log V)) | (O(V)) | Min Heap |
| **Bellman-Ford Algorithm** | (O(VE)) | (O(V)) | Array |

**Dynamic Programming (DP):**

**Dynamic Programming** is a method for solving problems by breaking them down into overlapping subproblems, solving each just once, and storing their solutions for reuse.

**Types of DP**:

1. **Memoization (Top-Down)**: Recursive approach where you store results in a table to avoid recomputing.
2. **Tabulation (Bottom-Up)**: Iterative approach where you build up the solution from the smallest subproblems.

**Common DP Algorithms**:

1. **0/1 Knapsack Problem**: Solves the knapsack problem by storing solutions to subproblems where different items and capacities are considered.
2. **Longest Common Subsequence (LCS)**: Finds the longest subsequence common to two strings.
3. **Fibonacci Series**: Uses DP to compute Fibonacci numbers without redundant calculations

|  |
| --- |
| **Algorithm** **Time Complexity** **Space Complexity** **Data Structure** |

|  |  |  |  |
| --- | --- | --- | --- |
| **0/1 Knapsack** | (O(nW)) | (O(nW)) | DP Table (Array) |
| **Longest Common Subsequence (LCS)** | (O(m.n)) | (O(m. n)) | DP Table (2D Array) |
| **Matrix Chain Multiplication** | (O(n^3)) | (O(n^2)) | DP Table (2D Array) |
| **Fibonacci (DP)** | (O(n)) | (O(n)) | Array |