A **Greedy Algorithm** is an approach to solving problems by making the locally optimal choice at each step in the hope that it leads to a globally optimal solution. It does not reconsider choices once made, which makes it fast but not always optimal.

**Key Characteristics of Greedy Algorithms**

1. **Greedy Choice Property** – A global optimal solution can be reached by choosing the best local option at each step.
2. **Optimal Substructure** – A problem exhibits an optimal substructure if an optimal solution to the problem contains optimal solutions to its subproblems.

**When to Use a Greedy Algorithm?**

* The problem must have the **Greedy Choice Property** and **Optimal Substructure**.
* If choosing the locally best solution at each step leads to the overall best solution.

**Examples of Greedy Algorithms**

1. **Activity Selection Problem** – Choosing the maximum number of activities that don’t overlap.
2. **Huffman Encoding** – Building an optimal prefix code tree.
3. **Dijkstra’s Algorithm** – Finding the shortest path in a weighted graph (without negative weights).
4. **Prim’s Algorithm** – Finding the Minimum Spanning Tree (MST).
5. **Kruskal’s Algorithm** – Another way to find MST using edge sorting.
6. **Fractional Knapsack Problem** – Maximizing profit by picking fractions of items.

**Example: Activity Selection Problem**

**Problem Statement**

Given n activities with their start and finish times, select the maximum number of activities that don’t overlap.

**Greedy Approach**

1. **Sort** activities by their **end times**.
2. **Pick the activity** that finishes the earliest and doesn’t overlap.
3. **Repeat** until no more activities can be picked.

**Code**

**Python Implementation:**

def activity\_selection(activities):

activities.sort(key=lambda x: x[1]) # Sort by finish time

selected = []

last\_end = 0

for start, end in activities:

if start >= last\_end:

selected.append((start, end))

last\_end = end

return selected

# Example usage

activities = [(1, 3), (2, 5), (3, 9), (6, 8), (5, 7)]

print(activity\_selection(activities)) # [(1, 3), (5, 7), (6, 8)]

**Greedy vs Dynamic Programming**

| **Feature** | **Greedy** | **Dynamic Programming** |
| --- | --- | --- |
| Decision Making | Locally optimal | Considers all subproblems |
| Efficiency | Faster (O(n log n) or O(n)) | Slower (O(n²) or O(n³)) |
| Reusability | No backtracking | Uses memoization |
| Works for All Cases? | No, only if greedy choice property holds | Yes, for most cases |