Greedy Paradigm

By: Aaryan Vinod Kumar

In algorithm design, a **paradigm** refers to a *general strategy or approach* to solve problems.
The **Greedy Paradigm** is one such strategy that builds a solution **step-by-step**, always choosing the **locally optimal choice** at each step ; with the *hope* that these local choices will lead to a **globally optimal solution**.

In simple terms:

> *"At every stage, pick the best-looking option right now."*

Greedy algorithms don't always guarantee the best overall result ; but for certain types of problems, they're the fastest and most efficient way to reach the correct solution.

 Uploaded October 2025

Core Idea

The greedy approach is based on **two main principles**:

1. **Greedy Choice Property:**
   A global optimum can be reached by choosing a local optimum at each step.

2. **Optimal Substructure:**
   A problem has optimal substructure if its optimal solution can be constructed from optimal solutions of its subproblems.

If a problem satisfies both properties, the greedy method often provides the **optimal solution**.

Greedy Paradigm in Simple Terms

Let's take an analogy ; imagine you're collecting coins to make ₹10 with the fewest coins possible.
You'll naturally start by picking the largest denomination available (₹5, then ₹2, then ₹1).

That's the greedy strategy ; pick what looks *best right now*.

 Uploaded October 2025

General Steps of a Greedy Algorithm

Greedy algorithms are effective when:

- The problem exhibits **optimal substructure**.
- The **local optimal choices lead to a global optimum**.
- A **dynamic programming** approach would work but is unnecessary due to simplicity or constraints.

Common characteristics:

- Involves **sorting**, **selection**, or **repeated decisions**.
- Often easier and faster (O(n log n) or O(n)).

Uploaded October 2025

Popular Greedy Problems

Problem	Description
Fractional Knapsack	Pick items with maximum value per weight until the bag is full.
Activity Selection	Select maximum number of non-overlapping activities.
Huffman Encoding	Generate optimal prefix codes for data compression.
Job Sequencing with Deadlines	Schedule jobs to maximize profit.
Minimum Spanning Tree	Use Kruskal’s or Prim’s Algorithm.
Dijkstra’s Algorithm	Find shortest path (works for non-negative weights).

Uploaded October 2025

Example 1: Fractional Knapsack Problem

Problem:

Given `n` items with values and weights, choose items to maximize total value in a knapsack of capacity `W`.
You can take *fractions* of an item.

Approach:

1. Calculate **value/weight ratio** for each item.

2. Sort items in **descending order** of ratio.

3. Pick items until capacity allows; take fraction if needed.

 Uploaded October 2025

C++ Implementation:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Item {
    int value, weight;
};

bool cmp(Item a, Item b) {
    double r1 = (double)a.value / a.weight;
    double r2 = (double)b.value / b.weight;
    return r1 > r2;
}

double fractionalKnapsack(int W, vector<Item> arr) {
    sort(arr.begin(), arr.end(), cmp);
    double totalValue = 0.0;
```



```
    for (auto &item : arr) {
        if (W >= item.weight) {
            totalValue += item.value;
            W -= item.weight;
        } else {
            totalValue += item.value * ((double)W /
item.weight);
            break;
        }
    }
    return totalValue;
}

int main() {
    int W = 50;
    vector<Item> arr = {{60, 10}, {100, 20}, {120, 30}};
    cout << "Maximum value in Knapsack: " <<
fractionalKnapsack(W, arr);
}
```

 Uploaded October 2025

Complexity Analysis

Operation	Complexity
Sorting items	O(n log n)
Selecting items	O(n)
Total	O(n log n)

Why Greedy Works Here

Because fractional items are allowed, the locally optimal choice (highest value/weight ratio) leads to a globally optimal total.

Uploaded October 2025

Advantages of the Greedy Paradigm

- **Simple & Fast:** Usually easier to implement than DP.
- **Efficient:** Often achieves optimal or near-optimal results.
- **Useful in Real-Time Systems:** Decisions made instantly without backtracking.

Limitations

- Doesn't always give the global optimum.
- Works only when **Greedy Choice Property** and **Optimal Substructure** are satisfied.
- Must be **proven** correct for each problem ; you can't blindly apply it.

Examples Where Greedy Fails

1. **0/1 Knapsack Problem:**
   Can't take fractions, so local decisions might block better global solutions.
   → Requires **Dynamic Programming**.

2. **Graph Problems with Negative Weights:**
   Dijkstra's algorithm fails if edges have negative weights.
   → Use **Bellman-Ford Algorithm** instead.

Uploaded October 2025