	Pimpri Chinchwad Education Trust's **Pimpri Chinchwad College of Engineering (PCCoE)** (An Autonomous Institute) Affiliated to Savitribai Phule Pune University (SPPU) ISO 21001:2018 Certified by TUV	
Department: Information Technology **Academic Year**: 2025-26 **Semester:** V		
DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY		
Name : Avishkar Jadhav **PRN :** 123B1F032		

ASSIGNMENT NO.3

Aim:- The aim of this study is to design and implement the Fractional Knapsack algorithm to maximize the total utility value of critical relief supplies transported during emergency flood relief operations. The solution ensures optimal use of limited boat capacity while prioritizing high-utility items such as medicine, food, and drinking water.

Objective:

- To model the flood relief supply transport as an optimization problem.
- To implement the **Fractional Knapsack Algorithm** for maximizing utility value.
- To prioritize **high-utility items** under given weight capacity constraints.
- To allow **fractional selection of divisible items** (e.g., food, water).
- To evaluate efficiency and effectiveness of the greedy approach.

Problem Statement:

Scenario: Emergency Relief Supply Distribution

A devastating flood has hit multiple villages in a remote area. The government, along with NGOs, is conducting an emergency relief operation. A rescue team must transport essential supplies from a relief center to affected villages using a boat with limited capacity W.

Each relief item has:

- Weight (wi) in kilograms.
- Utility value (vi) indicating its importance (e.g., medicine > food).
- Some items are divisible (food, water) and can be taken in fractions, while others are indivisible (medical kits, equipment).

As a logistics manager, your tasks are:

1. Implement the Fractional Knapsack algorithm.

2. Prioritize high-utility items while considering the boat's weight limit.

3. Allow partial selection of divisible items.

4. Ensure the boat carries maximum possible critical supplies within W.

Explanation:

The **Fractional Knapsack** problem is a classic **Greedy Algorithm** approach where:

- Items are selected based on **utility-to-weight ratio (vi/wi)**.
- Highest ratio items are taken first.
- If an item cannot fit entirely, only a fraction of it is taken (if divisible).
- Process continues until the capacity W is filled.

This ensures maximum utility value is achieved under limited capacity.

Algorithm:

Steps:

1. Input the number of items n and boat capacity W.
2. For each item, read weight wi and utility vi.
3. Compute the ratio vi/wi for each item.
4. Sort items in descending order of ratio.
5. Initialize total value = 0.
6. For each item:
   ○ If the item can fit entirely → add full weight and value. ○ Else → take fraction of item that fits into remaining capacity. 7. Stop when weight limit W is reached.
8. Output maximum utility value and list of selected items.

Pseudocode:

```
struct Item {
 int weight;
 int value;
};
double fractionalKnapsack(int W, Item arr[], int n) {
 // Compute ratio and sort
 sort(arr, arr+n, by value/weight ratio descending);
 int currentWeight = 0;
 double finalValue = 0.0;
 for (int i = 0; i < n; i++) {
 if (currentWeight + arr[i].weight <= W) {
 // Take whole item
 currentWeight += arr[i].weight;
 finalValue += arr[i].value;
 }
 else {
 // Take fraction of item
 int remain = W - currentWeight;
 finalValue += arr[i].value * ((double) remain / arr[i].weight);  break;
 }
 }
 return finalValue;
}
```

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>

using namespace std;

struct Item {
    string name;
    double weight;
    double value;
    bool divisible;
    int priority;

    Item(string n, double w, double v, bool d, int p)
        : name(n), weight(w), value(v), divisible(d), priority(p) {}

    double valuePerWeight() const {
        return value / weight; }
};

// Sort by priority first, then value per weight
bool compare(const Item& a, const Item& b) {
    if (a.priority == b.priority)
        return a.valuePerWeight() > b.valuePerWeight();
    return a.priority < b.priority;
}

double fractionalKnapsack(vector<Item>& items, double capacity, double& totalWeightCarried) {
    sort(items.begin(), items.end(), compare);

    cout << "\nSorted Items (by Priority, then Value/Weight):\n";
    cout << left << setw(20) << "Item"
         << setw(10) << "Weight"
         << setw(10) << "Value"
         << setw(12) << "Priority"
         << setw(15) << "Value/Weight"
         << setw(15) << "Type" << "\n";

    for (const auto& item : items) {
        cout << left << setw(20) << item.name
             << setw(10) << item.weight
             << setw(10) << item.value
             << setw(12) << item.priority
             << setw(15) << fixed << setprecision(2) << item.valuePerWeight()
             << setw(15) << (item.divisible ? "Divisible" : "Indivisible")
             << "\n";
    }

    double totalValue = 0.0;
    totalWeightCarried = 0.0;
    cout << "\nItems selected for transport:\n";

    for (const auto& item : items) {
        if (capacity <= 0) break;

        if (item.divisible) {
            double takenWeight = min(item.weight, capacity);
            double takenValue = item.valuePerWeight() * takenWeight;
            totalValue += takenValue;
            capacity -= takenWeight;
            totalWeightCarried += takenWeight;

            cout << " - " << item.name << ": " << takenWeight << " kg, Utility = " << takenValue
                 << ", Priority = " << item.priority << ", Type = Divisible\n";
        } else {
            if (item.weight <= capacity) {
                totalValue += item.value;
                capacity -= item.weight;
                totalWeightCarried += item.weight;

                cout << " - " << item.name << ": " << item.weight << " kg, Utility = " << item.value
                     << ", Priority = " << item.priority << ", Type = Indivisible\n";  }
        }
    }
    return totalValue;
}

int main() {
    vector<Item> items = {
        Item("Medical Kits",     10, 100, false, 1),
        Item("Food Packets",     20, 60,  true,  3),
        Item("Drinking Water",   30, 90,  true,  2),
        Item("Blankets",         15, 45, false, 3),
        Item("Infant Formula",    5, 50, false, 1)
    };

    double capacity;
    cout << "Enter maximum weight capacity of the boat (in kg): ";
    cin >> capacity;

    double totalWeightCarried;
    double maxValue = fractionalKnapsack(items, capacity, totalWeightCarried);

    cout << "\n===== Final Report =====\n";
    cout << "Total weight carried: " << fixed << setprecision(2) << totalWeightCarried << " kg\n";
    cout << "Total utility value carried: " << fixed << setprecision(2) << maxValue << " units\n";

    return 0;
}
```

Output:

```
Enter maximum weight capacity of the boat (in kg): 100

Sorted Items (by Priority, then Value/Weight):
Item             Weight    Value     Priority    Value/Weight   Type
Medical Kits       10        100        1           10.00         Indivisible
Infant Formula     5.00      50.00      1           10.00         Indivisible
Drinking Water     30.00     90.00      2            3.00          Divisible
Food Packets       20.00     60.00      3            3.00          Divisible
Blankets           15.00     45.00      3            3.00         Indivisible

Items selected for transport:
 - Medical Kits: 10.00 kg, Utility = 100.00, Priority = 1, Type = Indivisible
 - Infant Formula: 5.00 kg, Utility = 50.00, Priority = 1, Type = Indivisible
 - Drinking Water: 30.00 kg, Utility = 90.00, Priority = 2, Type = Divisible
 - Food Packets: 20.00 kg, Utility = 60.00, Priority = 3, Type = Divisible
 - Blankets: 15.00 kg, Utility = 45.00, Priority = 3, Type = Indivisible

===== Final Report =====
Total weight carried: 80.00 kg
Total utility value carried: 345.00 units
```

Time Complexity:

- Sorting step: **O(n log n)**
- Iteration over items: **O(n)**
- **Total:** O(n log n)

Space Complexity:

- O(1) (in-place, except for sorting overhead).

Questions:

1. **Why is the Fractional Knapsack algorithm suitable for solving the flood relief supply distribution problem, and how does it differ from the 0/1 Knapsack approach?**

Ans:

The **Fractional Knapsack algorithm** is ideal for flood relief supply distribution because many relief items, like food, water, or medicine, can be divided into smaller portions. For example, if a truck can carry only part of a food box, it can still take that portion and use the available space effectively to help more people.

On the other hand, the **0/1 Knapsack algorithm** requires that each item be taken whole or not at all. This is suitable for indivisible items like generators or tents but can result in wasted space during emergencies, reducing the total supplies delivered.

Key difference:

- **Fractional Knapsack** → Items can be split; best for divisible goods like food or medicine.
- **0/1 Knapsack** → Items cannot be split; best for whole items like equipment.

Example:
A truck can carry 50 kg. There are two items:

1. 40 kg of food with a value of 100
2. 30 kg of medicine with a value of 90

- **Fractional Knapsack:** It takes all of food (40 kg) and 10 kg of medicine → total value = 100 + (90 × 10/30) = 100 + 30 = **130**.
- **0/1 Knapsack:** It can only take one full item → either 40 kg food (value 100) or 30 kg medicine (value 90).

2. Explain how the utility-to-weight ratio (vi/wi) guides the greedy selection process in the Fractional Knapsack algorithm, and why this ensures an optimal solution.

Ans:
The **utility-to-weight ratio (vi/wi)** shows how much value you get for each unit of weight. The algorithm sorts by this ratio in descending order and chooses the best ones first.
This ensures that:

1. The most beneficial items (per unit of weight) are always chosen first.
2. If the knapsack cannot fit an entire item, taking a fraction of it still provides the highest possible incremental value.
3. By construction, no other arrangement of items can yield a higher total value, since every choice is locally optimal and contributes to a globally optimal solution.

Example:

You have 50 kg capacity and three items:

20 kg of water, value 100 → ratio = 5

50 kg of food, value 120 → ratio = 2.4

30 kg of medicine, value 90 → ratio = 3

The algorithm will pick:

All 20 kg of water → value 100

All 30 kg of medicine → value 90
Now only 0 kg is left → total value = **190**, which is the best possible outcome.

Conclusion: The Fractional Knapsack algorithm provides an efficient greedy solution for the emergency relief supply distribution problem. By prioritizing items with the highest utility-to-weight ratio and allowing fractional selection for divisible goods, the algorithm ensures maximum utility within the boat's limited capacity. This makes it highly effective for disaster management scenarios where both time and resources are critical.