	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		

ASSIGNMENT NO.3

Code:

```cpp
#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 well suited for flood relief supply distribution because relief goods such as food, water, or medicine can often be divided into smaller portions. For example, even if the knapsack (or truck/boat capacity) cannot carry an entire box of supplies, it can still take a fraction of it. This flexibility ensures that the available capacity is used as efficiently as possible to maximize the total utility (value) of delivered supplies.

In contrast, the 0/1 Knapsack approach does not allow items to be split: each item must be taken entirely or left out. This is suitable for indivisible goods (like a generator or a tent), but in flood relief situations, indivisibility leads to unused capacity and less effective distribution.

Thus, the key difference is:

- **Fractional Knapsack** → Items can be broken into fractions; best for divisible resources.
- **0/1 Knapsack** → Items are indivisible; best for discrete, non-splittable resources.

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) represents the "value per unit weight" of each item. In the greedy strategy of the Fractional Knapsack algorithm, items are first sorted in descending order of this ratio. The algorithm then selects items starting from the one with the highest ratio, because it provides the maximum value for the least weight.

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.

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.