	Pimpri Chinchwad Education Trust's **Pimpri Chinchwad college of Engineering**
Assignment No: 3	

Problem Statement

A devastating flood has hit multiple villages in a remote area. The government and NGOs are organizing an emergency relief operation. A rescue team has a limited-capacity boat that can carry a maximum weight of WWW kilograms.
The boat must transport critical supplies—including food, medicine, and drinking water—from a relief center to the affected villages.

Each relief item has:

- Weight (wiw_iwi) in kilograms
- Utility value (viv_ivi) indicating importance (e.g., medicine has higher value than food)
- Divisibility property: some items can be divided (food, water), others must be taken whole (medical kits)

Goals

1. Implement the Fractional Knapsack Algorithm to maximize total utility value.
2. Prioritize high-value items while considering weight constraints.
3. Allow partial selection of divisible items.
4. Ensure the boat carries the most critical supplies within weight limit WWW.

Course Objectives

1. Understand computational complexity of algorithms.
2. Select appropriate algorithmic strategies for real-world problems.

Course Outcomes

1. Analyze the asymptotic performance of algorithms.
2. Solve computational problems using Greedy or Divide and Conquer paradigms.

Theory

The Fractional Knapsack Problem is a classic optimization problem:
Given nnn items, each with a weight wiw_iwi and value viv_ivi, and a maximum capacity WWW,
the goal is to maximize total value by selecting items (or fractions of them) up to capacity.

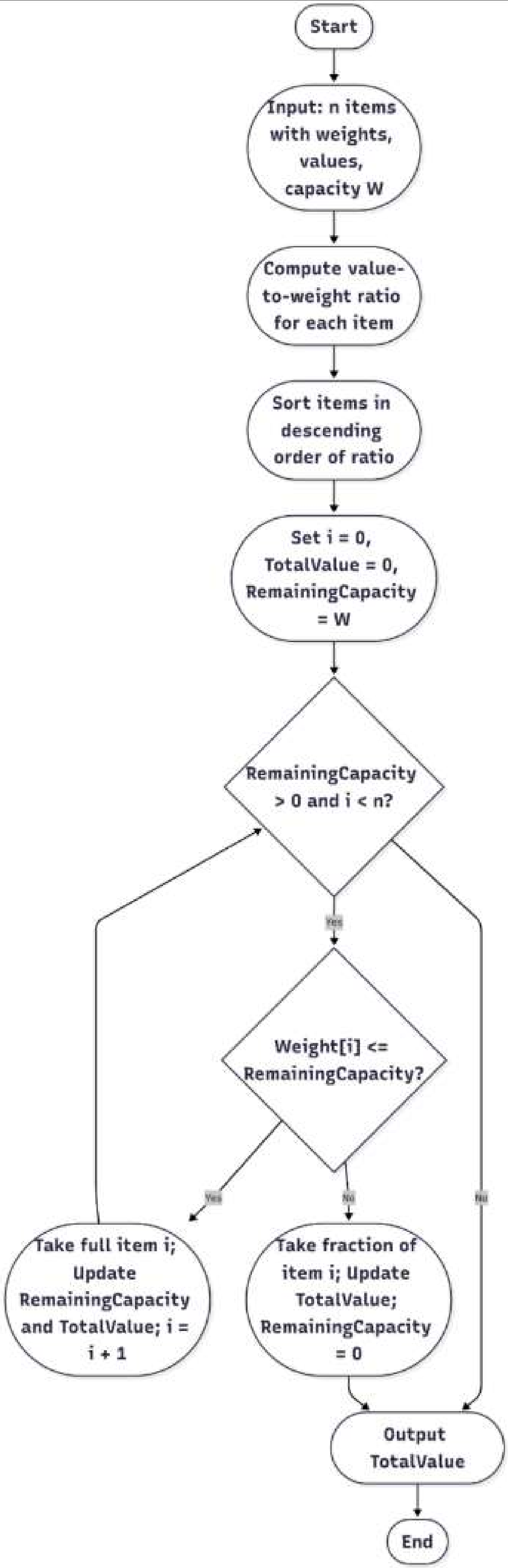
Unlike the 0/1 Knapsack, the Fractional Knapsack allows partial selection, making it solvable optimally in polynomial time.

Algorithm (Greedy Strategy)

1. Compute the **value-to-weight ratio** ri=vi/wi= frac{v_i}{w_i}ri=wivi for each item.
2. Sort items in descending order of rir_iri.
3. Fill the knapsack:
   - Take full items while capacity allows.
   - Take fractional parts only for divisible items when capacity is insufficient.
   - Skip indivisible items that cannot fit.

Working of the Greedy Method

- **Greedy-choice property:** Choosing the highest ratio item at each step leads to the optimal solution.
- **Optimal substructure:** Once part of the knapsack is filled, the remaining capacity forms a smaller subproblem of the same type.


Start
Input: n items with weights, values, capacity W
Compute value-to-weight ratio for each item
Sort items in descending order of ratio
Set i = 0, TotalValue = 0, RemainingCapacity = W
RemainingCapacity > 0 and i < n?
Yes
Weight[i] <= RemainingCapacity?
Yes
No
No
Take full item i; Update RemainingCapacity and TotalValue; i = i + 1
Take fraction of item i; Update TotalValue; RemainingCapacity = 0
Output TotalValue
End

Time Complexity

- Ratio computation → O(n)
- Sorting → O(nlogn)
- Selection → O(n)
- **Total:** O(nlogn)

Pseudocode

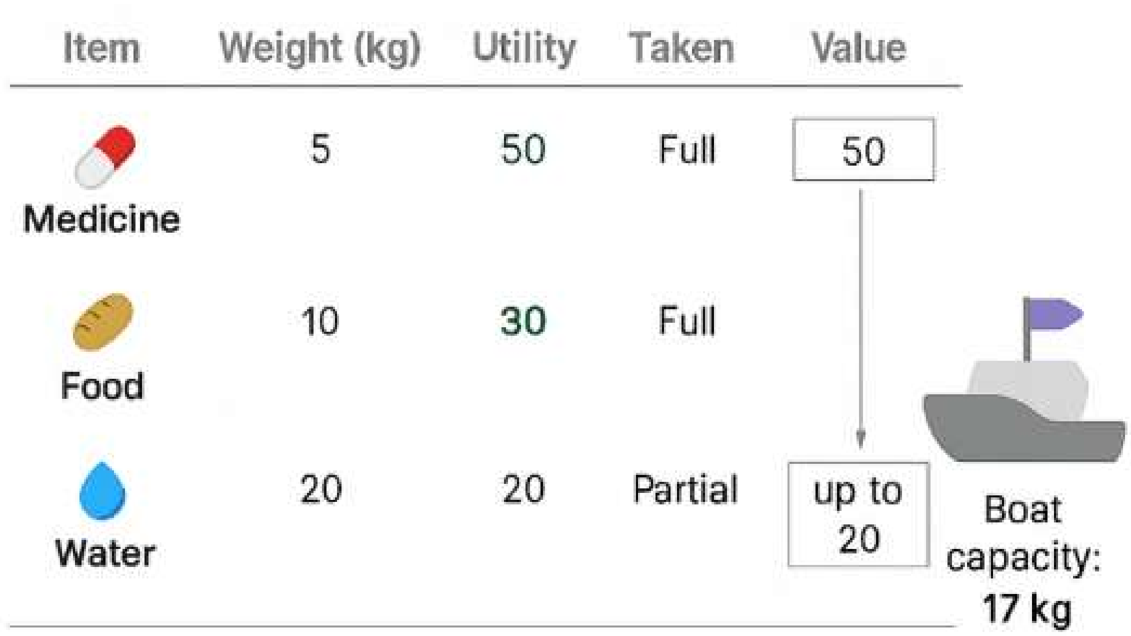
```
function MaxUtilFractional(items, W):

    Input: items = list of (weight w, value v, isDivisible), capacity W

    Output: maximum total utility value

    Compute ratio r[i] = v[i] / w[i] for each item

    Sort items by descending r[i]

    totalValue = 0

    remainingCapacity = W

    for each item in sorted items:

        if remainingCapacity == 0:

            break

        if item.weight <= remainingCapacity:

            totalValue += item.value

            remainingCapacity -= item.weight

        else if item.isDivisible:

            fraction = remainingCapacity / item.weight
```

```
        totalValue += fraction * item.value

        remainingCapacity = 0

    return totalValue
```

Example

Item	Weight (kg)	Utility	Divisible	Taken	Value Obtained
Medicine	5	50	No	Full	50
Food	10	30	Yes	Full	30
Water	20	20	Yes	Partial	2 (fraction)

Boat Capacity: 17 kg
Maximum Utility: 82



Step 1: Compute Utility per kg

- Medicine → 50/5=1050 / 5 = 1050/5=10
- Food → 30/10=330 / 10 = 330/10=3
- Water → 20/20=120 / 20 = 120/20=1

Preference Order: Medicine → Food → Water

Step 2: Fill the Boat

1. Take Medicine (5 kg) → Utility = 50
   Remaining = 12 kg
2. Take Food (10 kg) → Utility = 30
   Remaining = 2 kg
3. Take Water (2/20 fraction) → Utility = 2

Total Utility = 82

Conclusion

The Fractional Knapsack Algorithm maximizes total utility by prioritizing the value-to-weight ratio.
In emergency relief logistics, it ensures life-saving items (like medicines) are transported first, followed by food and water.

The greedy approach provides:

- Optimal results for fractional cases
- Efficient computation in O(nlogn)O(n \log n)O(nlogn) time
- Practical decision-making in real-time disaster management scenarios

C++ Implementation

```
#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, then by value/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;

}
```

Sample 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