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

Problem Statement: Emergency Relief Supply Distribution

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

Each type of relief item has:

- A weight (wi) in kilograms.
- Utility value (vi) indicating its importance (e.g., medicine has higher value than food).
- Some items can be divided into smaller portions (e.g., food and water), while others must be taken as a whole (e.g., medical kits).

Goals:

1. Implement the Fractional Knapsack algorithm to maximize the total utility value of the supplies transported.
2. Prioritize high-value items while considering weight constraints.
3. Allow partial selection of divisible items (e.g., carrying a fraction of food packets).
4. Ensure that the boat carries the most critical supplies given its weight limit W.

Course Objectives:

1. To know the basics of computational complexity of various algorithms.
2. To select appropriate algorithm design strategies to solve real-world problems.

Course Outcomes: After learning the course, students will be able to:

1. Analyze the asymptotic performance of algorithms.
2. Solve computational problems by applying suitable paradigms such as Divide and Conquer or Greedy methodologies.

Theory:

The Fractional Knapsack Problem is a classic optimization problem: given a set of items—each with weight w_i and value v_i—and a maximum capacity W, the objective is to maximize total value by selecting items (or fractions of them) up to capacity.

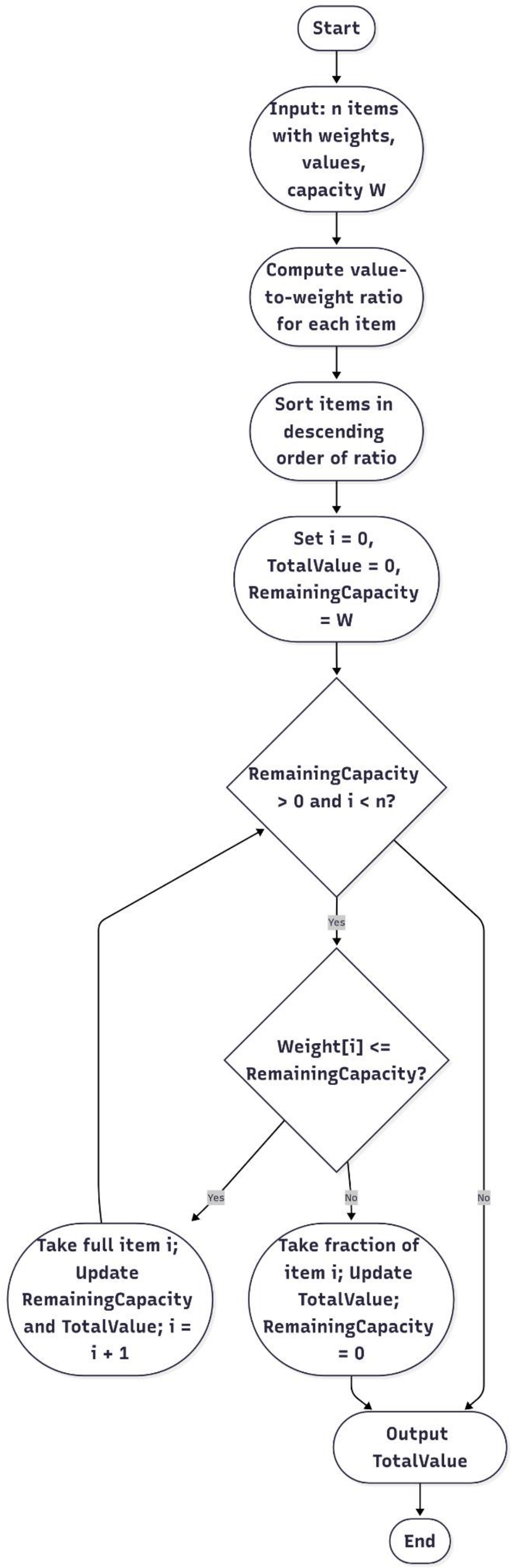
Unlike the 0/1 Knapsack, the fractional version allows partial selection, making it solvable optimally in polynomial time.

Algorithm (Greedy Strategy):

1. Compute the value-to-weight ratio $\mathbf{v_i / w_i}$ for each item.
2. Sort items in **descending order** of this ratio.
3. Fill the knapsack:
    - Take items fully if they fit.
    - If capacity runs out, take the exact fraction needed of the current item (only if divisible).
    - Skip indivisible items that do not fit.

Working of Greedy:

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


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(n log n)**
- Selection: **O(n)**
- **Total: O(n log n)**

Implementation (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
        // else if indivisible and can't fit, skip
    return totalValue
```

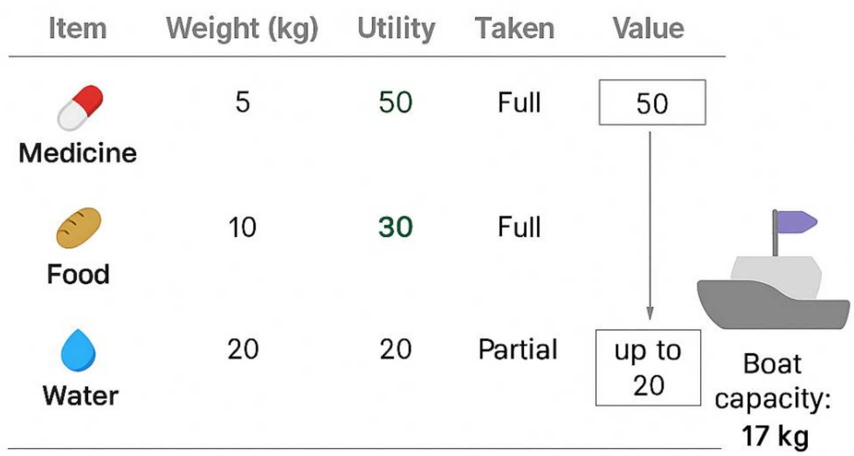
Output:

Sample 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 (if needed)	up to 20

Maximum Utility = 100 (if all fit in given W).

Boat capacity: 17 kg



Step 1: Calculate utility per kg

- Medicine: 50 / 5 = 10
- Food: 30 / 10 = 3
- Water: 20 / 20 = 1

Preference order: *Medicine → Food → Water*

Step 2: Fill the boat

1. **Take Medicine fully**:
    - Weight used = 5 kg
    - Utility = 50
    - Remaining capacity = 17 - 5 = 12 kg

2. **Take Food fully** (weight 10 kg, divisible):
    - Fits completely
    - Utility = 30
    - Remaining capacity = 12 - 10 = 2 kg
3. **Take Water partially** (weight 20 kg, divisible):
    - Only 2 kg fits
    - Utility = 20 × (2/20) = 2

Step 3: Total Utility

- Medicine: 50
- Food: 30
- Water (partial): 2

Total Utility = 82

#CODE

```
#include <iostream>

#include <vector>

#include <iomanip>

#include <algorithm>

using namespace std;


struct Item {

   string name;
```

```
    double weight;
    double value;
    int priority;
    bool isDivisible;
    double ratio; // value/weight
};

// Comparator: sort by priority first (ascending), then value/weight descending
bool cmp(Item a, Item b) {
    if (a.priority != b.priority)
        return a.priority < b.priority;
    return a.ratio > b.ratio;
}

int main() {
    double W;
    cout << "Enter maximum boat capacity (kg): ";
    cin >> W;

    vector<Item> items = {
        {"First Aid Box", 2.0, 70, 1, false},
        {"Medicine Kit", 6.0, 120, 1, false},
        {"Food Pack", 4.0, 80, 2, true},
```

```
    {"Water Bottles", 5.0, 50, 2, true},
    {"Blankets", 3.0, 40, 3, false}
};

// Compute value/weight ratio
for (auto &item : items)
    item.ratio = item.value / item.weight;

// Sort items
sort(items.begin(), items.end(), cmp);

// Display sorted items
cout << fixed << setprecision(2);
cout << "\nSorted Items (by priority, then value/weight):\n";
cout << left << setw(20) << "Item"
     << setw(8) << "Weight"
     << setw(8) << "Value"
     << setw(8) << "Priority"
     << setw(12) << "Value/Weight"
     << setw(10) << "Type" << endl;
cout << "----------------------------------------------------------------------\n";

for (auto &item : items)
```

```
    cout << setw(20) << item.name
         << setw(8) << item.weight
         << setw(8) << item.value
         << setw(8) << item.priority
         << setw(12) << item.ratio
         << setw(10) << (item.isDivisible ? "Divisible" : "Indivisible") << endl;

// Select items
double remainingCapacity = W;
double totalUtility = 0;
cout << "\nItems selected for transport:\n";
cout << "--------------------------------------------------\n";

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

    double takenWeight = 0;
    double utility = 0;

    if (item.weight <= remainingCapacity) {
        takenWeight = item.weight;
        utility = item.value;
```

```
        } else if (item.isDivisible) {
            takenWeight = remainingCapacity;
            utility = item.ratio * takenWeight;
        } else {
            continue; // skip indivisible item if it doesn't fit
        }

        remainingCapacity -= takenWeight;
        totalUtility += utility;

        cout << "- " << item.name << ": " << takenWeight << " kg, Utility = " << utility
             << ", Type = " << (item.isDivisible ? "Divisible" : "Indivisible") << endl;
    }

    cout << "===== Final Report =====\n";
    cout << "Total weight carried: " << W - remainingCapacity << " kg\n";
    cout << "Total utility value carried: " << totalUtility << " units\n";

    return 0;
}
```

#OUTPUT

Enter maximum boat capacity (kg): 10

Sorted Items (by priority, then value/weight):

Item	Weight	Value	Priority	Value/Weight	Type
First Aid Box	2.00	70.00	1	35.00	Indivisible
Medicine Kit	6.00	120.00	1	20.00	Indivisible
Food Pack	4.00	80.00	2	20.00	Divisible
Water Bottles	5.00	50.00	2	10.00	Divisible
Blankets	3.00	40.00	3	13.33	Indivisible

Items selected for transport:

--

- First Aid Box: 2.00 kg, Utility = 70.00, Type = Indivisible

- Medicine Kit: 6.00 kg, Utility = 120.00, Type = Indivisible

- Food Pack: 2.00 kg, Utility = 40.00, Type = Divisible

===== Final Report =====

Total weight carried: 10.00 kg

Total utility value carried: 230.00 units

Conclusion:

The **Fractional Knapsack Algorithm** maximizes utility by prioritizing items with the **highest value-to-weight ratio**. In emergency relief logistics, this ensures that life-saving items like **medicine kits** are transported first, followed by food and water as space allows. The greedy algorithm is both **efficient (O(n log n))** and **optimal** for fractional cases, making it highly effective in disaster management where quick, resource-optimized decisions are critical.