	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.

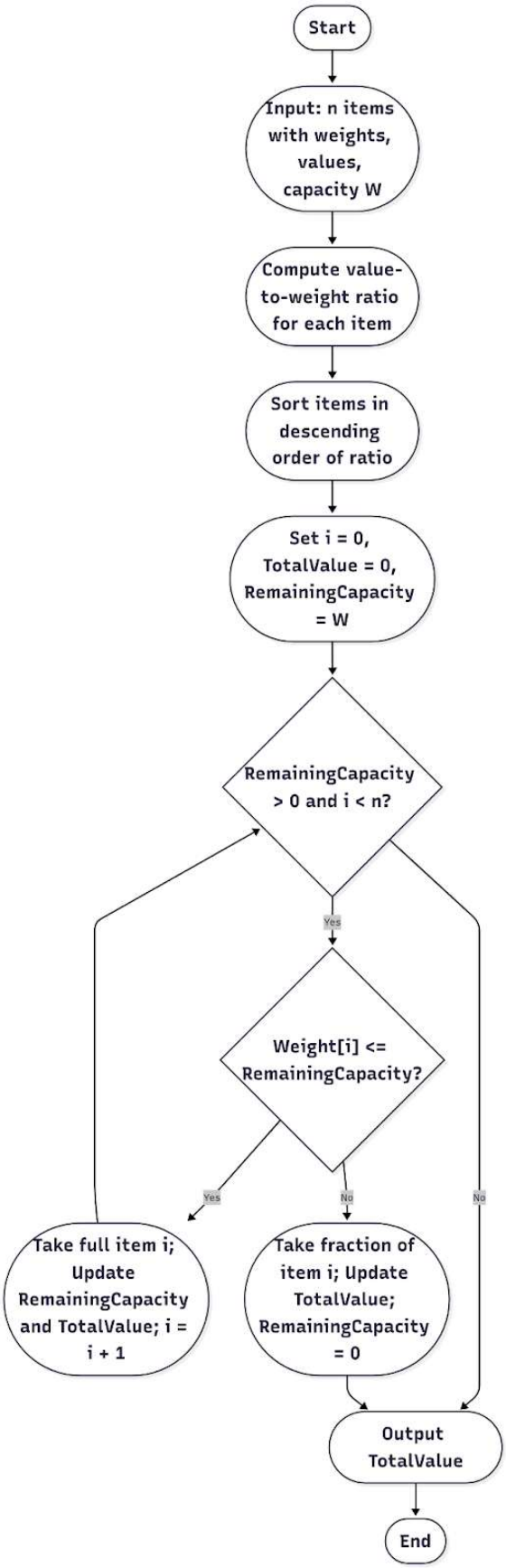
Unlike the 0/1 Knapsack, the fractional version allows partial selection, making it solvable optimally in polynomial time.

Algorithm (Greedy Strategy):

1. Compute the value-to-weight ratio $\mathbf{v_i}$ / $\mathbf{w_i}$ 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
No
Weight[i] <= RemainingCapacity?
Yes
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:

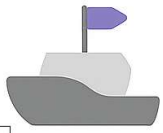
Sample Example:

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

Item	Weight (kg)	Utility	Taken	Value
Medicine	5	50	Full	50
Food	10	30	Full	
Water	20	20	Partial	up to 20



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:

```
import java.util.*;

class ReliefItem {
    String name;
    double weight, utilityValue;
    boolean divisible;
    int priority; // 1 = High, 2 = Medium, 3 = Low

    ReliefItem(String name, double weight, double utilityValue, boolean divisible, int priority) {
        this.name = name;
        this.weight = weight;
        this.utilityValue = utilityValue;
```

```
        this.divisible = divisible;
        this.priority = priority;
    }

    double valuePerWeight() {
        return utilityValue / weight;
    }

    @Override
    public String toString() {
        return String.format("%-20s %-10.2f %-10.2f %-12d %-15.2f %-15s",
                name, weight, utilityValue, priority, valuePerWeight(), (divisible ? "Divisible" :
"Indivisible"));
    }
}

public class FractionalKnapsackPriority {

    // Comparator for sorting by priority first, then value/weight
    static Comparator<ReliefItem> comparator = (a, b) -> {
        if (a.priority == b.priority) {
            return Double.compare(b.valuePerWeight(), a.valuePerWeight());
        }
        return Integer.compare(a.priority, b.priority);
    };

    static double distributeSupplies(List<ReliefItem> items, double capacity) {
        items.sort(comparator);

        System.out.println("\n=== Sorted Relief Items (by Priority, then Value/Weight) ===");
        System.out.printf("%-20s %-10s %-10s %-12s %-15s %-15s\n",
                "Item", "Weight", "Value", "Priority", "Value/Weight", "Type");
        for (ReliefItem item : items) {
            System.out.println(item);
        }

        double totalUtility = 0.0;
        double totalWeightCarried = 0.0;

        System.out.println("\n=== Items Selected for Transport ===");

        for (ReliefItem item : items) {
            if (capacity <= 0) break;
```

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

            System.out.printf(" - %s: %.2f kg, Utility = %.2f, Priority = %d, Type = Divisible\n",
                    item.name, takenWeight, takenValue, item.priority);

        } else if (item.weight <= capacity) {
            totalUtility += item.utilityValue;
            capacity -= item.weight;
            totalWeightCarried += item.weight;

            System.out.printf(" - %s: %.2f kg, Utility = %.2f, Priority = %d, Type = Indivisible\n",
                    item.name, item.weight, item.utilityValue, item.priority);
        }
    }

    System.out.println("\n===== Final Report =====");
    System.out.printf("Total weight carried: %.2f kg\n", totalWeightCarried);
    System.out.printf("Total utility value carried: %.2f units\n", totalUtility);

    return totalUtility;
}

public static void main(String[] args) {
    // Automatically generate 10 realistic relief items
    List<ReliefItem> items = Arrays.asList(
            new ReliefItem("Medicine Kits", 10, 90, false, 1),
            new ReliefItem("Food Packets", 25, 80, true, 2),
            new ReliefItem("Drinking Water", 20, 75, true, 1),
            new ReliefItem("Blankets", 15, 50, false, 3),
            new ReliefItem("Tents", 35, 120, false, 2),
            new ReliefItem("Sanitary Kits", 8, 65, false, 1),
            new ReliefItem("Clothes", 18, 70, true, 2),
            new ReliefItem("Fuel Canisters", 22, 95, false, 1),
            new ReliefItem("Cooking Utensils", 12, 60, false, 2),
            new ReliefItem("Baby Food", 6, 55, true, 1)
    );

    double boatCapacity = 100; // Maximum capacity (kg)
    System.out.println("=== Emergency Relief Supply Distribution ===");
```

```
        System.out.println("Boat capacity: " + boatCapacity + " kg");

        distributeSupplies(items, boatCapacity);
    }
}
```

Output:

```
=== Emergency Relief Supply Distribution ===
Boat capacity: 100.0 kg

=== Sorted Relief Items (by Priority, then Value/Weight) ===
Item                Weight     Value      Priority     Value/Weight    Type
Baby Food           6.00       55.00      1            9.17            Divisible
Medicine Kits       10.00      90.00      1            9.00            Indivisible
Sanitary Kits       8.00       65.00      1            8.13            Indivisible
Fuel Canisters      22.00      95.00      1            4.32            Indivisible
Drinking Water      20.00      75.00      1            3.75            Divisible
Cooking Utensils    12.00      60.00      2            5.00            Indivisible
Clothes             18.00      70.00      2            3.89            Divisible
Tents               35.00      120.00     2            3.43            Indivisible
Food Packets        25.00      80.00      2            3.20            Divisible
Blankets            15.00      50.00      3            3.33            Indivisible

=== Items Selected for Transport ===
 - Baby Food: 6.00 kg, Utility = 55.00, Priority = 1, Type = Divisible
 - Medicine Kits: 10.00 kg, Utility = 90.00, Priority = 1, Type = Indivisible
 - Sanitary Kits: 8.00 kg, Utility = 65.00, Priority = 1, Type = Indivisible
 - Fuel Canisters: 22.00 kg, Utility = 95.00, Priority = 1, Type = Indivisible
 - Drinking Water: 20.00 kg, Utility = 75.00, Priority = 1, Type = Divisible
 - Cooking Utensils: 12.00 kg, Utility = 60.00, Priority = 2, Type = Indivisible
 - Clothes: 18.00 kg, Utility = 70.00, Priority = 2, Type = Divisible
 - Food Packets: 4.00 kg, Utility = 12.80, Priority = 2, Type = Divisible

===== Final Report =====
Total weight carried: 100.00 kg
Total utility value carried: 522.80 units
PS D:\DSA>




 - Clothes: 18.00 kg, Utility = 70.00, Priority = 2, Type = Divisible
 - Food Packets: 4.00 kg, Utility = 12.80, Priority = 2, Type = Divisible

===== Final Report =====
Total weight carried: 100.00 kg
Total utility value carried: 522.80 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.