Problem Statement:

Write a program to solve the fractional knapsack problem using a greedy method.

Fractional Knapsack Problem Overview:

In the **fractional knapsack problem**, we are given:

- A knapsack with a maximum weight capacity `W`.
- A set of `n` items, where each item `i` has:
  - Weight `w[i]`
  - Value `v[i]`

The goal is to maximize the total value of the items placed in the knapsack. Unlike the **0/1 knapsack problem**, where you must either take the whole item or leave it, the **fractional knapsack problem** allows taking a fraction of an item.

Greedy Strategy for the Fractional Knapsack Problem:

To solve this problem using a greedy approach, we follow these steps:

1. **Compute Value-to-Weight Ratio**:
   - For each item, compute the ratio of value to weight:

$$\text{Value-to-weight ratio} = \frac{v[i]}{w[i]}$$

2. **Sort Items by Value-to-Weight Ratio**:
   - Sort the items in descending order of their value-to-weight ratio. This ensures that we pick items with the highest value per unit weight first.
3. **Select Items for the Knapsack**:
   - Initialize the total value as `0` and the remaining capacity as `W`.
   - Traverse the sorted list of items, and for each item:
     - If the item can fully fit in the remaining capacity, add it to the knapsack.
     - If the item cannot fully fit, take a fraction of the item such that the knapsack is filled to capacity.
4. **Return the Maximum Total Value**:
   - Continue this process until the knapsack is full or all items have been considered.
   - The total value accumulated is the maximum value that can be obtained with the given knapsack capacity.

Procedure (Fractional Knapsack Problem):

1. Sort Items by Value-to-Weight Ratio:

- Calculate the value-to-weight ratio for each item.
- Sort the items in descending order based on this ratio.

2. Select Items to Maximize Value:

- Start by taking as much as possible from the item with the highest value-to-weight ratio. • If you can't take the entire item due to capacity constraints, take a fraction of the item and update the remaining capacity.

3. Stop When the Knapsack is Full:

- The algorithm terminates when the knapsack's capacity is full or there are no more items to process.

Explanation of the Code:

- **Class `Item`**:
  - Represents an item with two attributes: `value` and `weight`.
- **Function `fractional_knapsack(items, capacity)`**:
  - Sorts the items based on their value-to-weight ratio.
  - Iterates through the sorted items and adds as much of each item as possible to the knapsack until it is full.
  - Returns the maximum value that can be obtained.

Example Walkthrough:

Consider the following items with their values and weights:

- Item 1: Value = 60, Weight = 10
- Item 2: Value = 100, Weight = 20
- Item 3: Value = 120, Weight = 30

Knapsack capacity = 50.

1. **Step 1 - Compute Value-to-Weight Ratios**:
   - Item 1: 60 / 10 = 6
   - Item 2: 100 / 20 = 5
   - Item 3: 120 / 30 = 4
2. **Step 2 - Sort Items**:
   - Items are sorted by value-to-weight ratio in descending order: Item 1, Item 2, Item 3.
3. **Step 3 - Select Items**:

- Take all of Item 1 (weight 10, value 60). Remaining capacity: 50 - 10 = 40.
- Take all of Item 2 (weight 20, value 100). Remaining capacity: 40 - 20 = 20.
- Take 20/30 fraction of Item 3, which gives a value of $120 \times 20/30 = 80$

4. **Step 4 - Compute Maximum Value**:
   - Total value = 60 (Item 1) + 100 (Item 2) + 80 (fraction of Item 3) = 240.

Time and Space Complexity Analysis:

Time Complexity:

- Sorting the items by value-to-weight ratio: **O(n log n)**, where `n` is the number of items.
- Iterating through the sorted items to select them: **O(n)**.
- **Overall Time Complexity**: **O(n log n)**.

Space Complexity:

- Storing the list of items: **O(n)**.
- **Overall Space Complexity**: **O(n)**.

Sample Output

```
Maximum value we can obtain = 240.0
```

Conclusion:

The **fractional knapsack problem** is solved efficiently using a greedy method. The algorithm prioritizes items with the highest value-to-weight ratio and takes as much of each item as possible until the knapsack is full. This approach ensures that the total value is maximized within the given weight capacity.