Given the weights and profits of N items, in the form of {profit, weight} put these items in a knapsack of capacity W to get the maximum total profit in the knapsack. In Fractional Knapsack, we can break items for maximizing the total value of the knapsack.

Input: arr[] = {{60, 10}, {100, 20}, {120, 30}}, W = 50  
Output: 240   
Explanation: By taking items of weight 10 and 20 kg and 2/3 fraction of 30 kg.   
Hence total price will be 60+100+(2/3)(120) = 240

Input:  arr[] = {{500, 30}}, W = 10  
Output: 166.667

Fractional Knapsack Problem using [Greedy algorithm](https://www.geeksforgeeks.org/greedy-algorithms/):

An efficient solution is to use the Greedy approach.

The basic idea of the greedy approach is to calculate the ratio profit/weight for each item and sort the item on the basis of this ratio. Then take the item with the highest ratio and add them as much as we can (can be the whole element or a fraction of it).

This will always give the maximum profit because, in each step it adds an element such that this is the maximum possible profit for that much weight.

Illustration:

Check the below illustration for a better understanding:

Consider the example: arr[] = {{100, 20}, {60, 10}, {120, 30}}, W = 50.

Sorting: Initially sort the array based on the profit/weight ratio. The sorted array will be {{60, 10}, {100, 20}, {120, 30}}.

Iteration:

For i = 0, weight = 10 which is less than W. So add this element in the knapsack. profit = 60 and remaining W = 50 – 10 = 40.

For i = 1, weight = 20 which is less than W. So add this element too. profit = 60 + 100 = 160 and remaining W = 40 – 20 = 20.

For i = 2, weight = 30 is greater than W. So add 20/30 fraction = 2/3 fraction of the element. Therefore profit = 2/3 \* 120 + 160 = 80 + 160 = 240 and remaining W becomes 0.

So the final profit becomes 240 for W = 50.

Follow the given steps to solve the problem using the above approach:

Calculate the ratio (profit/weight) for each item.

Sort all the items in decreasing order of the ratio.

Initialize res = 0, curr\_cap = given\_cap.

Do the following for every item i in the sorted order:

If the weight of the current item is less than or equal to the remaining capacity then add the value of that item into the result

Else add the current item as much as we can and break out of the loop.

Return res.

<https://chat.openai.com/share/4a6ed23d-df54-40f5-a9a6-2f1021af54c3>