Fractional Knapsack Problem

Given the weights and values of **N** items, put these items in a knapsack of capacity **W** to get the maximum total value in the knapsack. In **Fractional Knapsack**, we can break items for maximizing the total value of the knapsack

Note: In the 0-1 Knapsack problem, we are not allowed to break items. We either take the whole item or don't take it.

Input:
Items as (value, weight) pairs
arr[] = {{60, 10}, {100, 20}, {120, 30}}
Knapsack Capacity, W = 50

Output: *Maximum possible value = 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:
Items as (value, weight) pairs
arr[] = {{500, 30}}
Knapsack Capacity, W = 10
Output: *166.667*

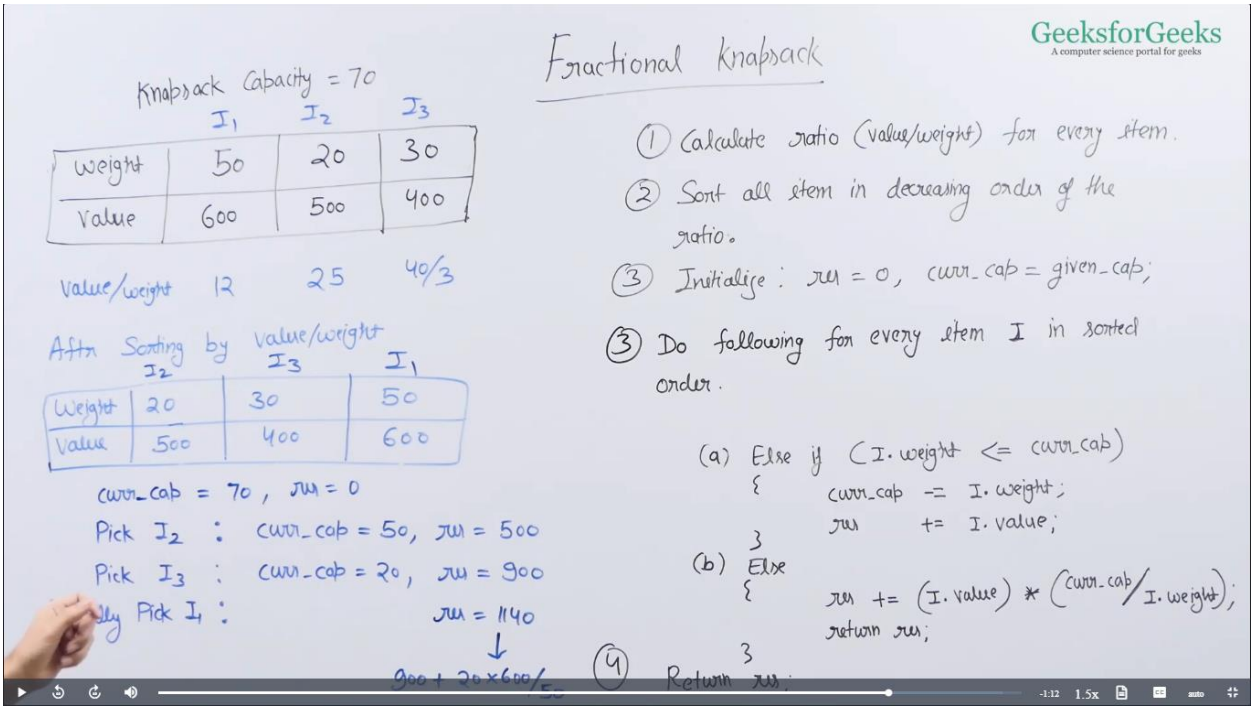
Naive Approach: Try all possible subsets with all different fractions but that will be very inefficient.

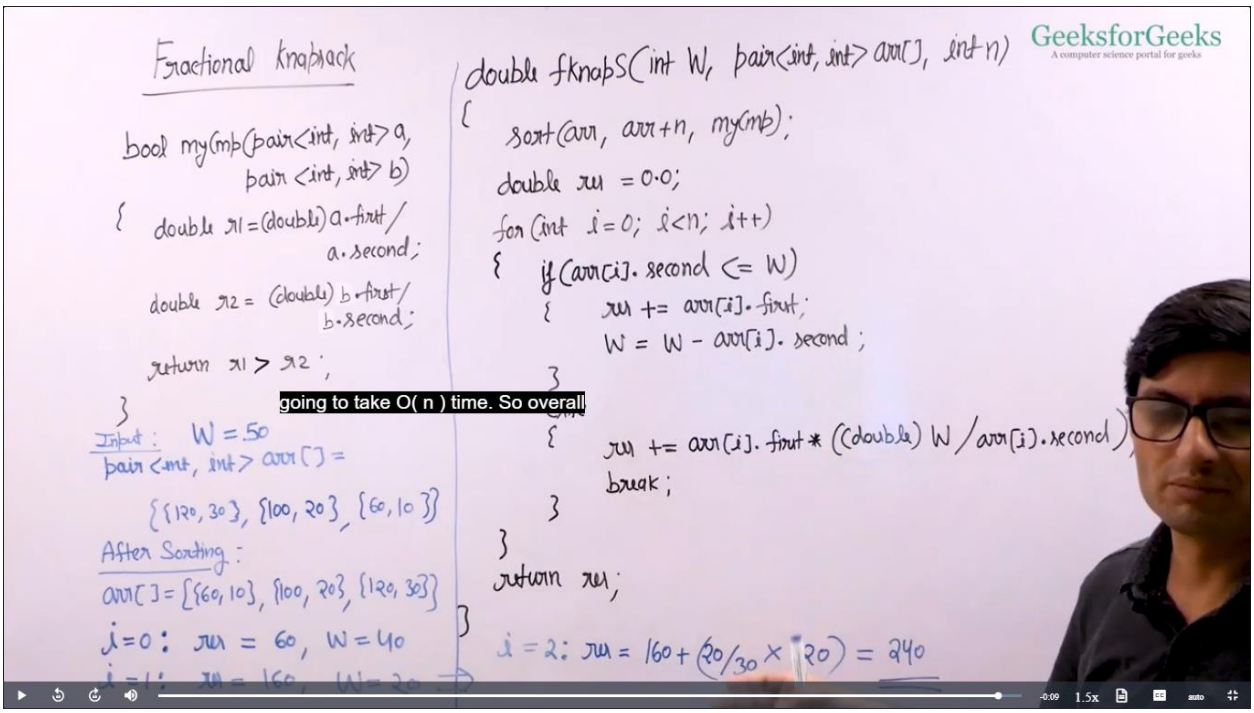
Greedy approach for fractional knapsack problem:

*An **efficient solution** is to use the Greedy approach. The basic idea of the greedy approach is to calculate the ratio value/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 until we can't add the next item as a whole and at the end add the next item as much as we can. Which will always be the optimal solution to this problem.*

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

- Calculate the ratio(value/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





Time Complexity: O(N log N)
Auxiliary Space: O(N)

```
// class implemented
/*
struct Item{
   int value;
   int weight;
};
*/


class Solution {
  public:
```

```
// Comparison function to sort Item according to profit/weight ratio
static bool cmp(struct Item a, struct Item b)
{
    double r1 = (double)a.value / (double)a.weight;
    double r2 = (double)b.value / (double)b.weight;
    return r1 > r2;
}

// Function to get the maximum total value in the knapsack.
double fractionalKnapsack(int W, Item arr[], int N)
{
// Sorting Item on basis of ratio
sort(arr, arr + N, cmp);

double finalvalue = 0.0;

// Looping through all items
for (int i = 0; i < N; i++) {

    // If adding Item won't overflow,add it completely
    if (arr[i].weight <= W) {
        W -= arr[i].weight;
        finalvalue += arr[i].value;
    }

    // If we can't add current Item add fractional part of it
    else {
        finalvalue+= arr[i].value* ((double)W / (double)arr[i].weight);
        break;
    }
}
return finalvalue;
}};
```