Fractional Knapsack in C++

```
#include <iostream>
#include <algorithm>
using namespace std;

class Item {
public:
    int wt, val;

    Item(int w, int v) {
        wt = w;
        val = v;
    }

    bool operator<(const Item& i) const {
        return (double)val / wt >
(double)i.val / i.wt;
    }
};

double fracKnapsack(Item arr[], int n,
int W) {
    sort(arr, arr + n);
    double res = 0.0;

    for (int i = 0; i < n; i++) {
        if (arr[i].wt <= W) {
            res += arr[i].val;
            W -= arr[i].wt;
        } else {
            res += (arr[i].val * (double)W) /
arr[i].wt;
            break;
        }
    }
    return res;
}

int main() {
    Item arr[] = {Item(10, 60), Item(40,
40), Item(20, 100), Item(30, 120)};
    int n = sizeof(arr) / sizeof(arr[0]);
    int W = 50;
    cout << fracKnapsack(arr, n, W) <<
endl;
    return 0;
}
```

Problem Summary:

You are given:

- Items with weight wt and value val
- A maximum capacity W of the knapsack
- You can **take fractions of items**

Goal: Maximize the total value in the knapsack.

🎒 Input
Item arr[] = {Item(10, 60), Item(40, 40), Item(20, 100), Item(30, 120)};
int W = 50;

➤ Step 1: Calculate Value/Weight Ratio and Sort Descending

Item	Weight	Value	Value/Weight
0	10	60	6.00
1	40	40	1.00
2	20	100	5.00
3	30	120	4.00

🔀 After Sorting by Value/Weight (Descending):

Index	Weight	Value	Value/Weight
0	10	60	6.00
2	20	100	5.00
3	30	120	4.00
1	40	40	1.00

🧮 Step 2: Fill the Knapsack

Initial:
W = 50, res = 0.0

➤ Iteration Table

Iteration	Item	Weight	Value	Can Take Fully?	Action	New W	res
0	0	10	60	✅ Yes	Take full item: res += 60, W -= 10	40	60.0
1	2	20	100	✅ Yes	Take full	20	160.0

Iteration	Item	Weight	Value	Can Take Fully?	Action	New W	res
					item: res += 100, W -= 20		
2	3	30	120	✗ No	Take fraction: res += 120 * 20/30 = 80.0	0	240.0
3	1	-	-	-	Not processed (knapsack full)	0	240.0

✓ Final Output
240

240