

12
.6

Fractional Knapsack

MEDIUM

What You Will Learn

- ✓ Why fractional knapsack is greedy-solvable (but 0/1 knapsack is not)
- ✓ Sort by value-per-weight ratio to maximize value
- ✓ How to handle partial items
- ✓ The fundamental difference between fractional and 0/1 knapsack

Intuition First

You have a bag (knapsack) with a weight limit. There are items, each with a weight and value. You want to fill the bag to maximize total value. The twist: you can take fractions of items (like pouring liquid gold into the bag).

The greedy insight: always grab the item with the highest value-per-kg first. Think of it like a spice market — if saffron gives you 100 value per gram and pepper gives 2 per gram, fill up on saffron first. If saffron runs out and bag isn't full, take pepper next.

The Algorithm

- Compute ratio = value / weight for each item
- Sort items by ratio DESCENDING (highest value density first)
- Greedily take items in order
- If full item fits: take it all (capacity -= weight)
- If item doesn't fully fit: take the fraction that fills remaining capacity

Visual — Knapsack Filling

Capacity = 50 kg

Item	Weight	Value	Value/Weight	Take how much?
Item1	10	60	6.0 ★	Take all 10 kg → value=60
Item2	20	100	5.0 ★	Take all 20 kg → value=100

Item3	30	120	4.0	Only 20 kg left → take $20/30 = 0.67$ fraction → value=80
Item4	40	40	1.0	Bag full → SKIP

Total value = 60 + 100 + 80 = 240 ✓ (max possible for 50kg bag)

■ Dry Run — capacity=50

After sorting by ratio: Item1(6.0), Item2(5.0), Item3(4.0), Item4(1.0)

Step 1	Item1: $w=10 \leq 50$. Take all. value+=60. remaining=40 Bag: [Item1:10kg] Remaining: 40kg
Step 2	Item2: $w=20 \leq 40$. Take all. value+=100. remaining=20 Bag: [Item1:10kg][Item2:20kg] Remaining: 20kg
Step 3	Item3: $w=30 > 20$. Take fraction= $20/30$. value+= $120 \cdot (20/30)=80$. remaining=0 Bag: [Item1:10kg][Item2:20kg][Item3:20kg] FULL
Step 4	Remaining=0 → STOP. Total value = 240 ✓

■ Java Code

```
public static double fractionalKnapsack(int capacity, int[] weight, int[] value) {
    int n = weight.length;
    // Create item array [weight, value]
    double[][] items = new double[n][2];
    for (int i = 0; i < n; i++) {
        items[i][0] = weight[i];
        items[i][1] = value[i];
    }
    // Sort by value/weight ratio (descending)
    Arrays.sort(items, (a, b) -> Double.compare(b[1]/b[0], a[1]/a[0]));
    double totalValue = 0.0;
    double remaining = capacity;
    for (double[] item : items) {
        if (remaining <= 0) break;
        if (item[0] <= remaining) {
            // Take entire item
            totalValue += item[1];
            remaining -= item[0];
        } else {
            // Take fractional part
            totalValue += item[1] * (remaining / item[0]);
            remaining = 0;
        }
    }
}
```

```
return totalValue;
}
```

Time & Space Complexity

Case	Time	Space	What it really means
All Cases	$O(n \log n)$	$O(n)$	Sort dominates; $O(n)$ for items array

Fractional vs 0/1 Knapsack — CRITICAL DIFFERENCE

- Fractional: can take parts → greedy works → $O(n \log n)$
- 0/1 Knapsack: must take whole or nothing → greedy FAILS → need DP → $O(n \cdot W)$
- Counter-example for 0/1: capacity=3, items=(w=2,v=3),(w=3,v=4),(w=1,v=2)
- Greedy picks ($v/w = 1.5$) = item1 + item3 = 5. Optimal: item2+item3 = 6 (greedy wrong!)

✓ Fractional Knapsack

- Can take fraction of item
- Greedy by v/w ratio
- $O(n \log n)$
- Always optimal

✗ 0/1 Knapsack

- All or nothing
- Greedy FAILS
- DP needed: $O(n \cdot W)$
- Greedy gives wrong answer

Common Mistakes

- Sorting by value alone (not ratio) — a heavy high-value item might not be worth it
- Using integer division for ratio — always cast to double/float
- Forgetting to handle the fractional case when item weight > remaining capacity

Pro Tips

- The ratio v/w is also called 'value density' — a key concept in optimization
- If all items have equal weight, just sort by value (ratio reduces to value)
- Real-world: portfolio optimization, resource allocation — same greedy logic

■ Visual Thinking

Picture a bag with a zip on top showing remaining space.

Items are colored containers — some dense (small but high value), some fluffy (big but low value).

You pour items in by density order — densest first.

Last item: pour until bag is full, stop mid-pour for the fraction.

Animation: bag fills with colored liquid. Each item pours in order of density.

Last item pours partially — a gauge shows exactly what fraction fills the remaining space.

■ Practice Problems

[Med] capacity=60, weights=[10,40,20,30], values=[60,40,100,120] — max value?

[Med] Same input but 0/1 knapsack — why does greedy give wrong answer?

[Hard] Fractional knapsack with k bags — distribute items across k bags to maximize min value

[Hard] What if items can be refilled (repeated knapsack)? How does greedy change?

■ Quick Revision Box

- Compute ratio = value/weight for each item
- Sort by ratio DESCENDING
- Take full items while capacity remains
- Take fraction of last fitting item
- $O(n \log n)$ time — greedy is optimal for fractional
- NOT valid for 0/1 knapsack — use DP there