# CSE3318: Greedy Algorithms
## Fractional Knapsack
## by
## Dr. Bhanu Jain



All slides are based on: ***Introduction to Algorithms***, by Thomas H. Cormen, Charles E. Leiserson, Ronald E. Rivest, Clifford Stein,3rd edition (CLRS)

# Greedy Algorithms

- **For many optimization problems**, **using dynamic programming** to determine the best choices is an overkill; simpler, more efficient algorithms suffice
- **A greedy algorithm** always makes the choice that looks best at the moment
  - Make a locally optimal choice in hope of getting a globally optimal solution
  - Example: Play cards, Invest on stocks, etc.
  - Do not always yield optimal solutions.
  - They do for some problems with optimal substructure (like Dynamic Programming)
  - Easier to code than DP
  - E.g., problem of fractional knapsack, scheduling unit-time tasks with deadlines and penalties, minimum-spanning-tree algorithms, Dijkstra's algorithm for shortest paths from a single source
- In **Dynamic Programming**, we choose at each step, but the choice may depend on the solution to sub-problems.
- In a **Greedy Algorithm**, we make whatever choice seems best at the moment and then solve the sub-problems arising after the choice is made.

# Greedy Algorithms: Fractional Knapsack

- The Fractional Knapsack problem is a greedy algorithm problem where:
  - We have **n items**, each with a **weight** $w_i$ and a **value** $v_i$.
  - A knapsack has a maximum weight capacity W.
  - **Unlike the 0/1 Knapsack problem, where we must either take an item fully or leave it, here we can take fractions of items.**
- **Objective:**
- Maximize the total value placed in the knapsack while ensuring the total weight does not exceed W.

# Greedy Algorithms: Fractional Knapsack

- **Time Complexity**
- Sorting the items: O(nlogn)
- Processing items (greedy selection): O(n)
- Overall complexity: O(nlogn)

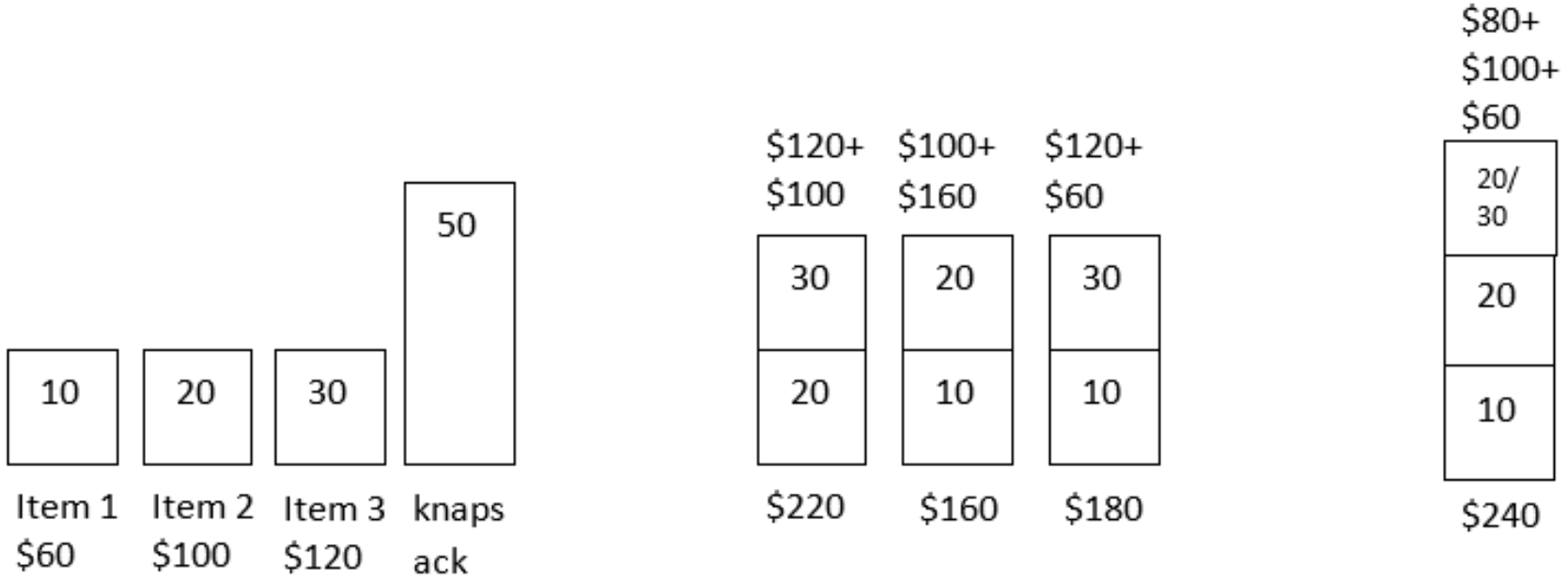
- This is much more efficient than the 0/1 Knapsack problem because the Fractional Knapsack problem has an optimal greedy solution.
- .

# Greedy Algorithms: Fractional Knapsack

1. Compute the value-to-weight ratio for each item: $r_i=v_i/w_i$
2. Sort items in descending order of $r_i$
3. Iterate through the sorted items:
    - If the full item fits in the remaining knapsack capacity, take it entirely.
    - If not, take the maximum possible fraction that fits.
4. Stop when the knapsack is full.

.

# Knapsack Algorithms: 0-1 versus fractional



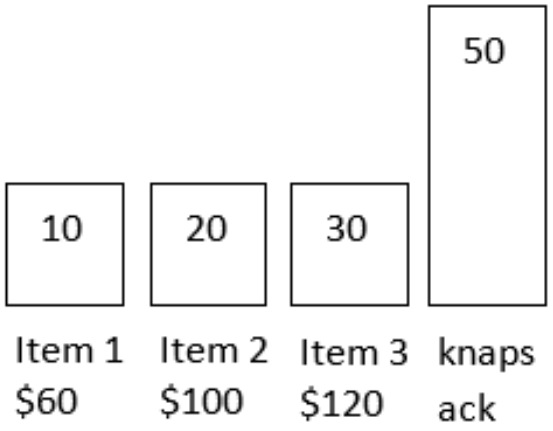

**0/1 knapsack problem:**

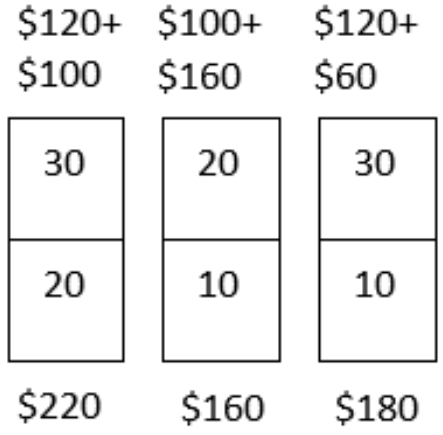
**Given:** A set S of n items, with each item i having
$w_i$ - a positive weight
$b_i$ - a positive benefit
**Goal**: Choose items with maximum total benefit but with at most weight W.

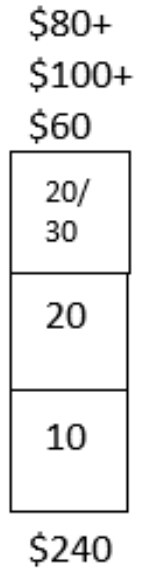
The items are either placed in the knapsack or not. **Fractional parts of items are not selected**

**Fractional knapsack problem:** The setup is the same as the 0-1 knapsack problem, but **fractions of items can be selected for the knapsack**, rather than making a binary (0-1) choice for each item

# Knapsack Algorithms: 0-1 versus fractional




**Greedy strategy for Fractional knapsack problem**. **Dynamic programming for the 0-1 knapsack problem**

(a) Select a **subset of the three items** such that their weight is under 50 pounds

(b) The optimal subset includes items 2 and 3. Any solution with item 1 is not optimal, even though item 1 has the highest value per pound.

(c) **Fractional knapsack problem**: selecting the items **in order of greatest value per pound** offers us an **optimal solution**

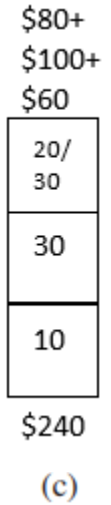
# Knapsack Algorithms: 0-1 versus fractional

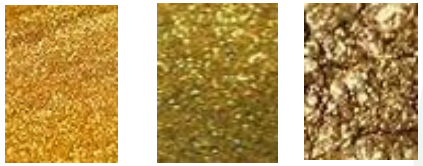
ValuePerUnitWeight = $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$

$$\text{TotalValue} = \sum_{j=1}^{i} x_j\, v_j = TotalValue + x_j v_j$$

$$\text{TotalWeight} = \sum_{j=1}^{i} x_j\, w_j = TotalWeight + x_j w_j$$

$$\text{M} - \text{TotalWeight} = M - \sum_{j=1}^{i} x_j\, w_j$$



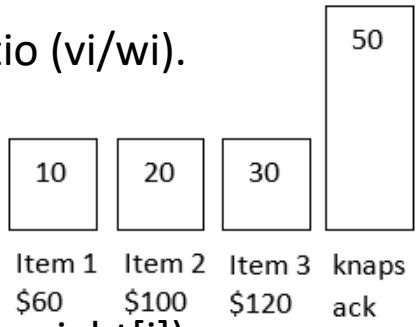
**Greedy strategy for Fractional knapsack problem**

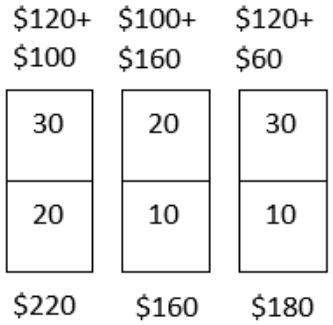
- **Sort items** in the order of non-increasing of the value of unit cost
- Accept items in the descending order of value/unit weight ratio
- Selecting an item for the knapsack reduces the capacity of the knapsack

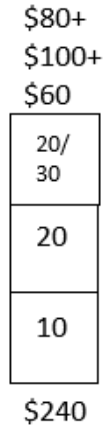
# Knapsack Algorithms: 0-1 versus fractional

**GREEDY-FRACTIONAL-KNAPSACK(W, n)**

1. Sort the items in decreasing order of value-to-weight ratio (vi/wi).
2. t**otalValue** ← 0;  **remainingCapacit**y ← W. //Initialize
3. for i ← 1 to n do
4.  if remainingCapacity = 0 then
5.  return totalValue
6.  weightAdded ← minimum of (remainingCapacity and weight[i]).
7.  totalValue ← totalValue + (weightAdded) * (vi/wi).
8.  remainingCapacity ← remainingCapacity - (taken weight).
9. return totalValue




**Greedy strategy for Fractional knapsack problem**

- **Sort items** in the order of non-increasing value of the unit cost
- Accept items in the descending order of value/unit weight ratio
- Selecting an item for the knapsack reduces the capacity of the knapsack

**Greedy strategy for Fractional knapsack problem**

Max bag weight before item selection(M): 50

| **itemWeight(w),** | **itemValue(v),** | **itemValuePerUnitWeight(v/w),** | **totalValue**, | **weightAdded,** | **remainingCapacity M - totalWeight** |
|---|---|---|---|---|---|
| **10**, | 60, | **6.0,** | 60, | **10,** | 40 |
| **20,** | 100, | **5.0,** | 160, | **20,** | 20 |
| **30,** | 120, | **4.0,** | 240, | **20,** | 0 |

Max bag value after item selection: **240**

# Knapsack Algorithms: 0-1 versus fractional

**Given:**

| Item | Weight (wi) | Value (vi) | Value/Weight Ratio (ri) |
|---|---|---|---|
| 1 | 10 | 60 | 6.0 |
| 2 | 20 | 100 | 5.0 |
| 3 | 30 | 120 | 4.0 |

Knapsack capacity: **W = 50**

**Solution Process:**

1. Sort by value-to-weight ratio: Item 1 → Item 2 → Item 3.
2. Take Item 1 (10 units, full) → Value = 60
3. Take Item 2 (20 units, full) → Value = 160
4. Take 2/3rd of Item 3 (20 of 30 units) → Value = **240 (60+160+80)**

Final Maximum Value: 240

$$\text{ValuePerUnitWeight} = \frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$$

$$\text{TotalValue} = \sum_{j=1}^{i} x_j\, v_j = TotalValue + x_j v_j$$

$$\text{TotalWeight} = \sum_{} x_j\, w_j = TotalWeight + x_j w_j$$

$$\text{M} - \text{TotalWeight} = M - \sum_{j=1}^{i} x_j\, w_j$$

# Analysis of Algorithms Cost Analysis

**Key Differences from 0/1 Knapsack**

| Feature | 0/1 Knapsack | Fractional Knapsack |
|---|---|---|
| Selection | Entire item or none | Can take fractions of items |
| Solution Approach | Dynamic Programming (O(W × n)) | Greedy Algorithm (O(n log n)) |

# The END!