# CH08-320201
# Algorithms and Data Structures

## **Lecture 18 — 17 Apr 2018**

Prof. Dr. Michael Sedlmair

Jacobs University
Spring 2018

# 4. Design Concepts

# Recap

- We have been looking into different algorithms and, in particular, emphasized one design concept, namely, the **divide-&-conquer** strategy, which was based on recursions and whose analysis was given by recurrences.
- Now, we are going to look into further design concepts.

# 4.1 Greedy Algorithms

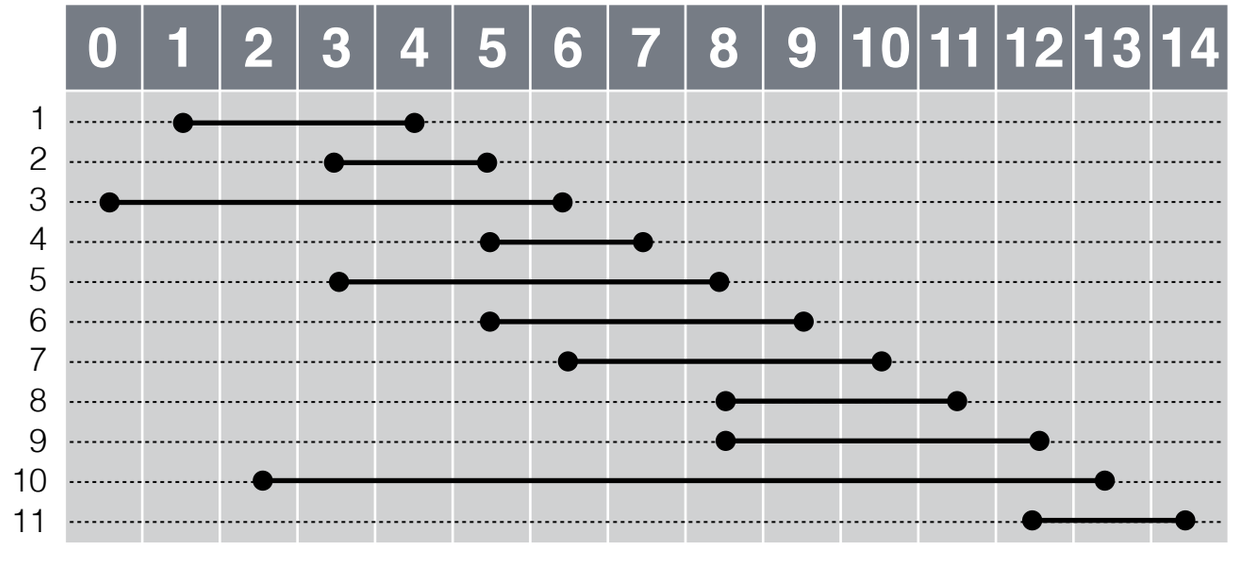
# Activity-selection problem

- Suppose we have a set $S=\{a_1,a_2,\ldots,a_n\}$ of n activities.
- The activities wish to use a resource, which can only be used by one activity at a time.
- Each activity $a_i$ has a start time $s_i$ and a finish time $f_i$, where $0 \le s_i < f_i < \infty$.
- Two activities $a_i$ and $a_j$ are **compatible**, if $[s_i,f_i)$ and $[s_j,f_j)$ are disjoint.
- The activity-selection problem is to select a **maximum-size subset** of mutually compatible activities.

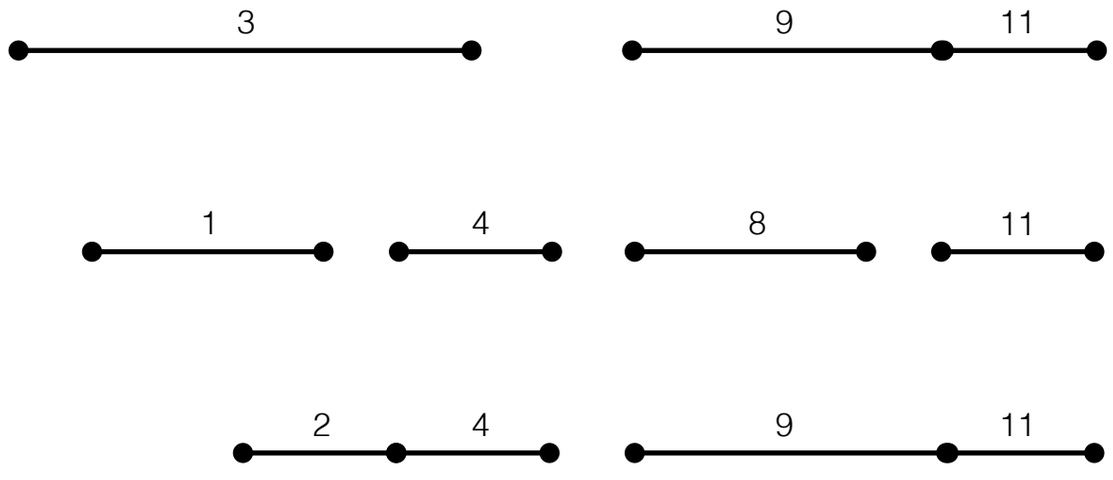
# Activity-selection problem

- Example:

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 1 | 3 | 0 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| $f_i$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |




- $\{a_3, a_9, a_{11}\}$ is a subset of mutually compatible activities.
- $\{a_1, a_4, a_8, a_{11}\}$ is a largest subset of mutually compatible activities.
- $\{a_2, a_4, a_9, a_{11}\}$ is another largest subset of mutually compatible activities.



# Sorting

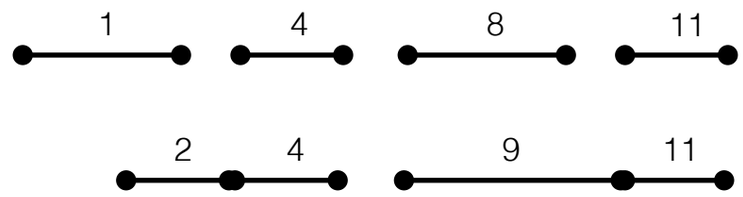
- We can apply a sorting algorithm to the finish times, which operates in O(n lg n) time.
- Then, we can assume that the activities are sorted, i.e., $f_1 \leq f_2 \leq \ldots \leq f_n$

# Greedy algorithm

- A greedy algorithm always makes the choice that looks best at the moment.
- I.e., it makes a locally optimal choice in the hope that it will lead to a globally optimal solution.

# Greedy approach

- After sorting, $a_1$ has the earliest finish time $f_1$.
- A greedy approach starts with taking $a_1$ as a locally optimal choice.
- **Lemma**:
  The greedy choice of picking $a_1$ as first choice is optimal.
- **Proof**:
  - Suppose A is a globally optimal solution for set S.
  - Let $a_k \in A$ be the activity with earliest finish time $f_k$ in A.
  - If k=1, then $a_1 \in A$ and we are done.
  - If k>1, then we can replace A by $(A \setminus \{a_k\}) \cup \{a_1\}$.
  - Since $f_1 \leq f_k$, this is still an optimal solution.
  - Hence, we can always start with $a_1$.



# Greedy approach

- After the first step, we consider the subproblem $S' = \{a_i \in S: s_i \geq f_1\}$.
- We apply the same greedy strategy.
- **Lemma**:
  $A \setminus \{a_1\}$ is the optimal solution for S'.
- **Proof**:
  - Let B be a solution for S' that is larger than $A \setminus \{a_1\}$.
  - Then, $B \cup \{a_1\}$ would be solution for S that is larger than A.
  - Contradiction!

# Greedy approach

- From the two lemmata, it follows by induction that the greedy approach delivers the globally optimal solution.

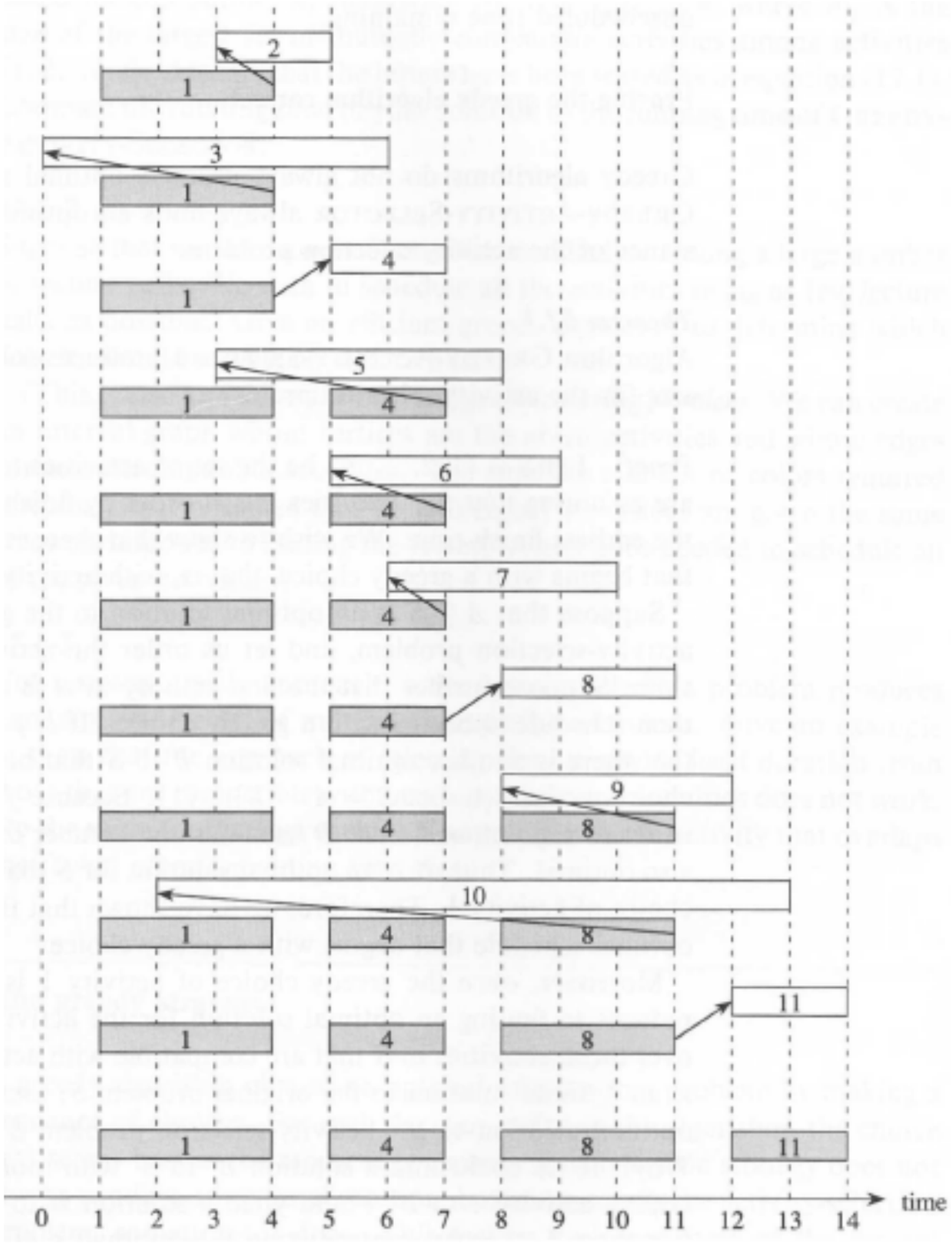
# Greedy algorithm

```
Greedy-Selector (S)
  // Assume S = {a1,…,an} with activities sorted by fi.
  A := {a1}
  j := 1
  for i := 2 to n
    if si ≥ fj
    then A := A U {ai}
      j := i
return A
```

# Example

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 1 | 3 | 0 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| $f_i$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

**Alternative**:
Recursive function



# Greedy approach

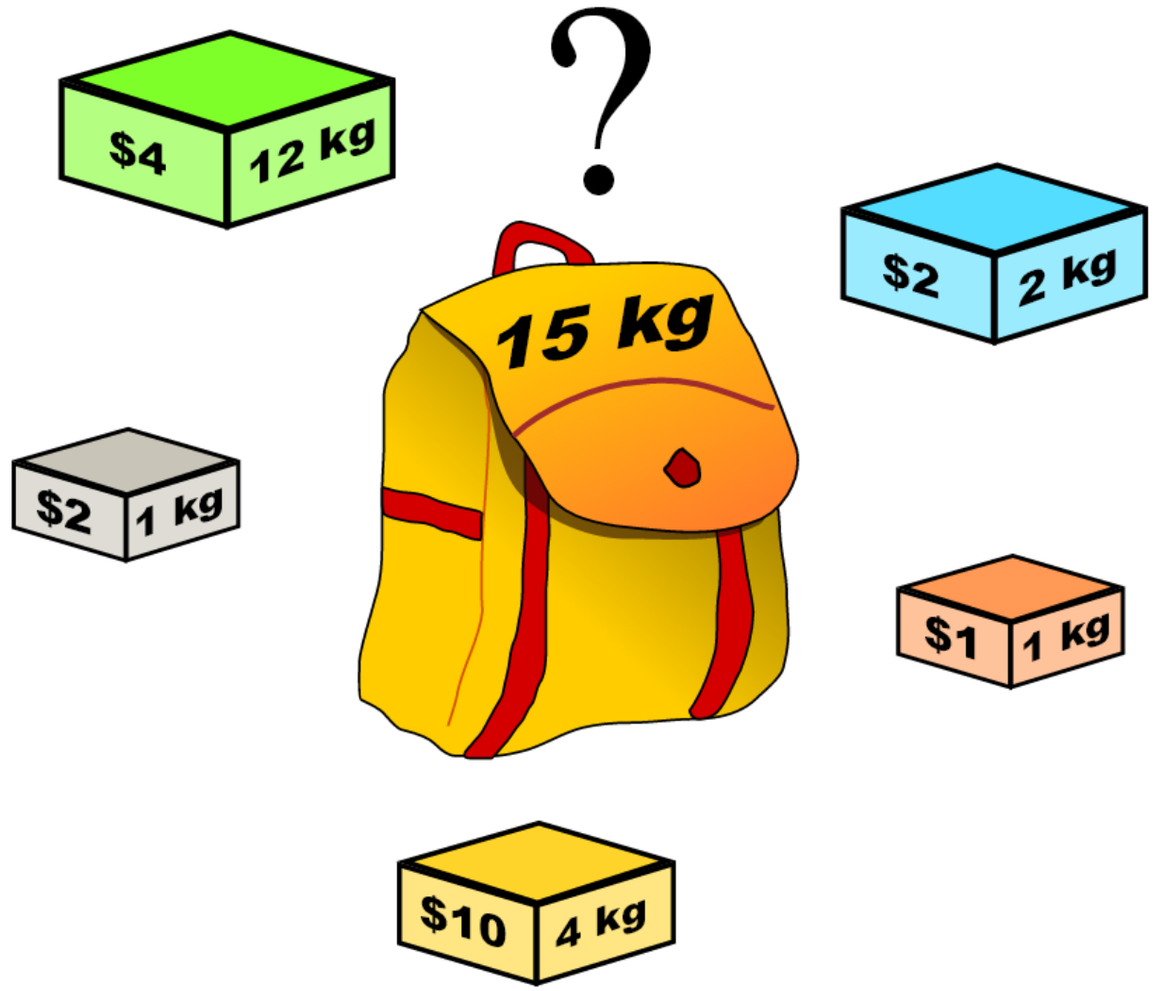
- Time complexity:
  - O(n) — if already sorted
  - O(n lg n) — if not sorted
- Comparison to brute-force approach:
  - Brute-force approach would try all combinations, reject all the combinations that have incompatibilities, and pick among the remaining ones one with maximum number of activities.
  - Time complexity: $O(2^n)$

# Greedy algorithm

- Greedy algorithms build upon solving subproblems.
- Greedy approaches make a locally optimal choice.
- There is no guarantee that this will lead to a globally optimal solution.
- Having found a global optimum requires a proof.

# Knapsack problem

A thief breaks into a museum. Fabulous paintings, sculptures, and jewels are everywhere. The thief has a good eye for the value of these objects, and knows that each will fetch hundreds or thousands of dollars on the clandestine art collector's market. But, the thief has only brought a single knapsack to the scene of the robbery, and can take away only what he can carry. What items should the thief take to maximize the haul?



# Knapsack problem

**Problem:**

- Given some items, pack the knapsack to get the maximum total value.
- Each item has some weight and some value.
- Total weight that we can carry is no more than some fixed number W.
- We must consider weights of items as well as their values.

| Item # | Weight | Value |
|---|---|---|
| 1 | 1 | 8 |
| 2 | 3 | 6 |
| 3 | 5 | 5 |

# Knapsack problem

- Given a knapsack with maximum capacity *W*, and a set *S* consisting of *n* items
- Each item *i* has some weight $w_i$ and value $v_i$ (assuming that all $w_i$ and *W* are integer values)
- How to pack the knapsack to achieve maximum total value of packed items?
- Mathematically:

$$\max \sum_{i \in T} v_i \text{ subject to } \sum_{i \in T} w_i \leq W$$

# Brute-force approach

**Algorithm:**

- Generate all $2^n$ subsets
- Discard all subsets whose sum of the weights exceed $W$ (not feasible)
- Select the maximum total benefit of the remaining (feasible) subsets

**Time complexity:**

$O(2^n)$

# Brute-force approach

S = {(*item*$_1$, 5, \$70), (*item*$_2$, 10, \$90), (*item*$_3$, 25, \$140)} , W = 25

Subsets:

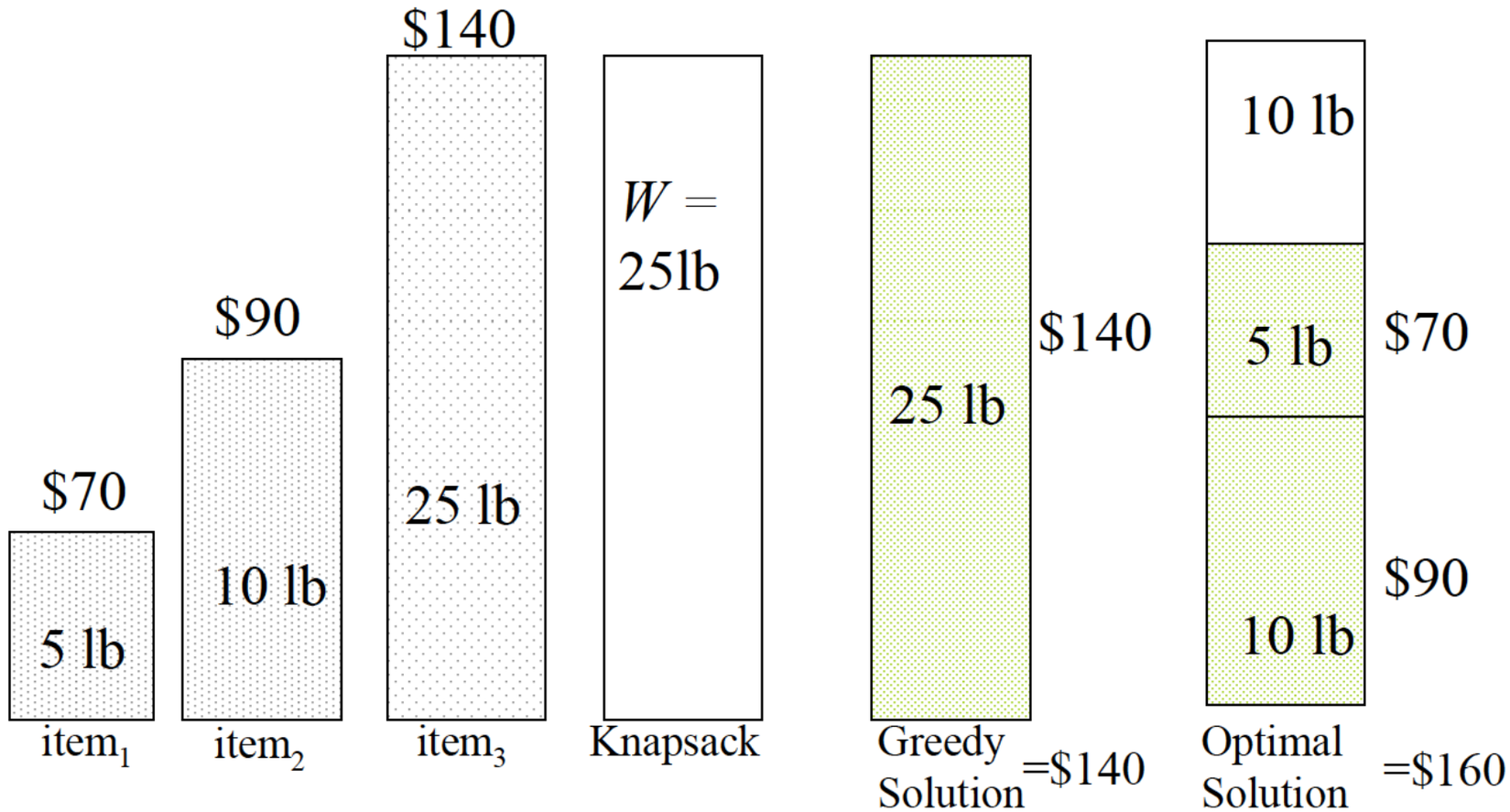
1. {}
2. {(*item*$_1$, 5, \$70)} Profit=\$70
3. {(*item*$_2$, 10, \$90)} Profit=\$90
4. {(*item*$_3$, 25, \$140)} Profit=\$140
5. {(*item*$_1$, 5, \$70), (*item*$_2$, 10, \$90)}. Profit=\$160
6. {(*item*$_2$, 10, \$90), (*item*$_3$, 25, \$140)} exceeds W
7. {(*item*$_1$, 5, \$70), (*item*$_3$, 25, \$140)} exceeds W
8. {(*item*$_1$, 5, \$70), (*item*$_2$, 10, \$90),(*item*$_3$, 25, \$140)} exceeds W

# Greedy approach

- Use a greedy approach to be more efficient.
- What would be the greedy choice?
  - Maximum beneficial item
  - Minimum weight item
  - Maximum weight item
  - Maximum value per unit item
- Asymptotic time complexity: O(n lg n)

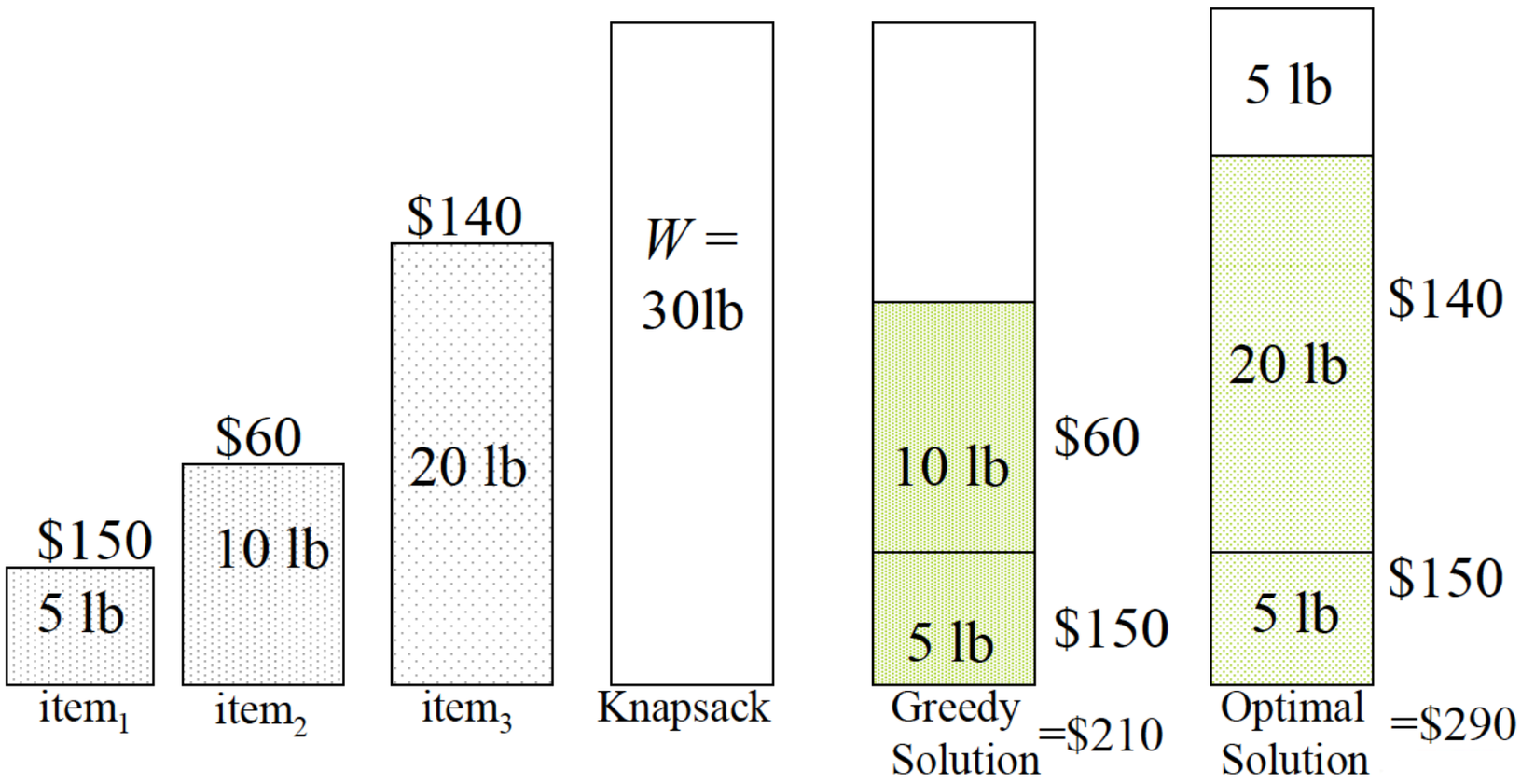
# Greedy 1: Maximum beneficial item

S = {(*item*$_1$, 5, \$70), (*item*$_2$, 10, \$90), (*item*$_3$, 25, \$140)},
W = 25



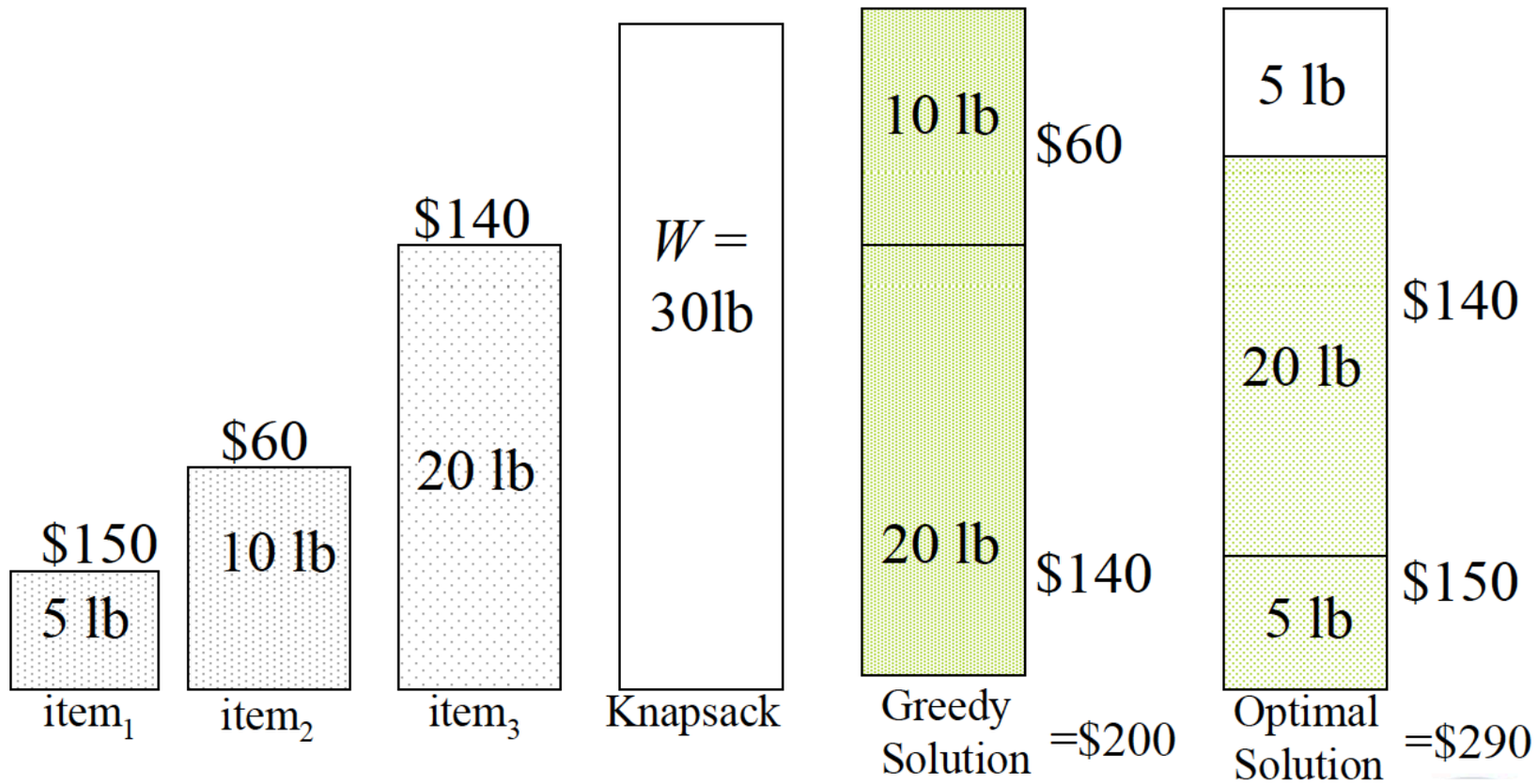
# Greedy 2: Minimum weight item

S = {(*item*$_1$, 5, \$150), (*item*$_2$, 10, \$60), (*item*$_3$, 20, \$140)}
W = 30



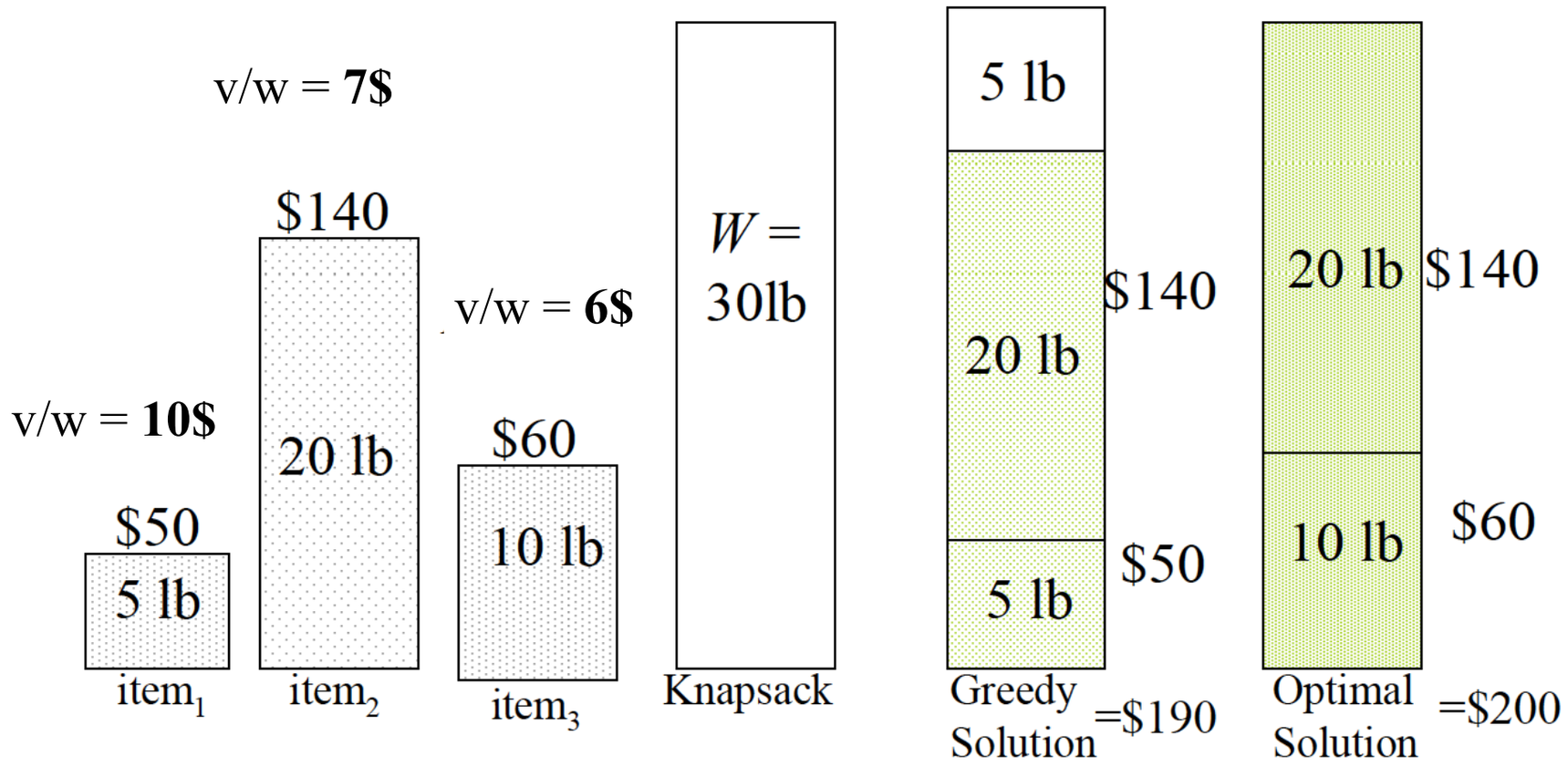
# Greedy 3: Maximum weight item

S = {(*item*$_1$, 5, \$150), (*item*$_2$, 10, \$60), (*item*$_3$, 20, \$140)}
W = 30



# Greedy 4: Maximum value per weight

S = {(*item*$_1$, 5, \$50), (*item*$_2$, 20, \$140), (*item*$_3$, 10, \$60)}
W = 30

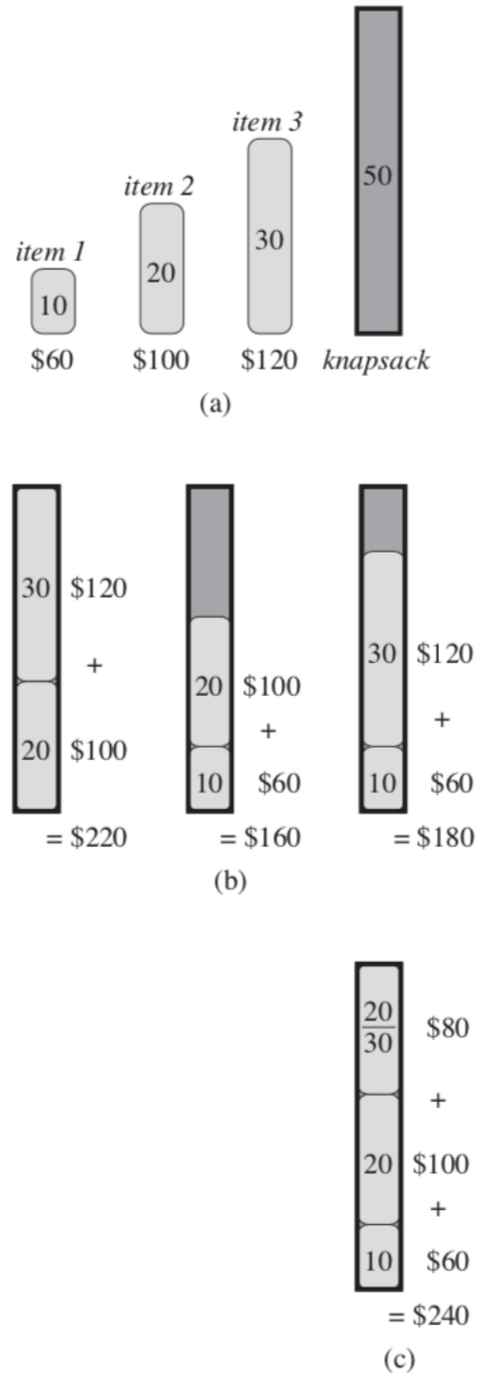


# Greedy approach

- As we had said, the locally optimal choice of a greedy approach does not necessary lead to a globally optimal one.
- For the knapsack problem, the greedy approach actually fails to produce a globally optimal solution.
- However, it produces an approximation, which sometimes is good enough.

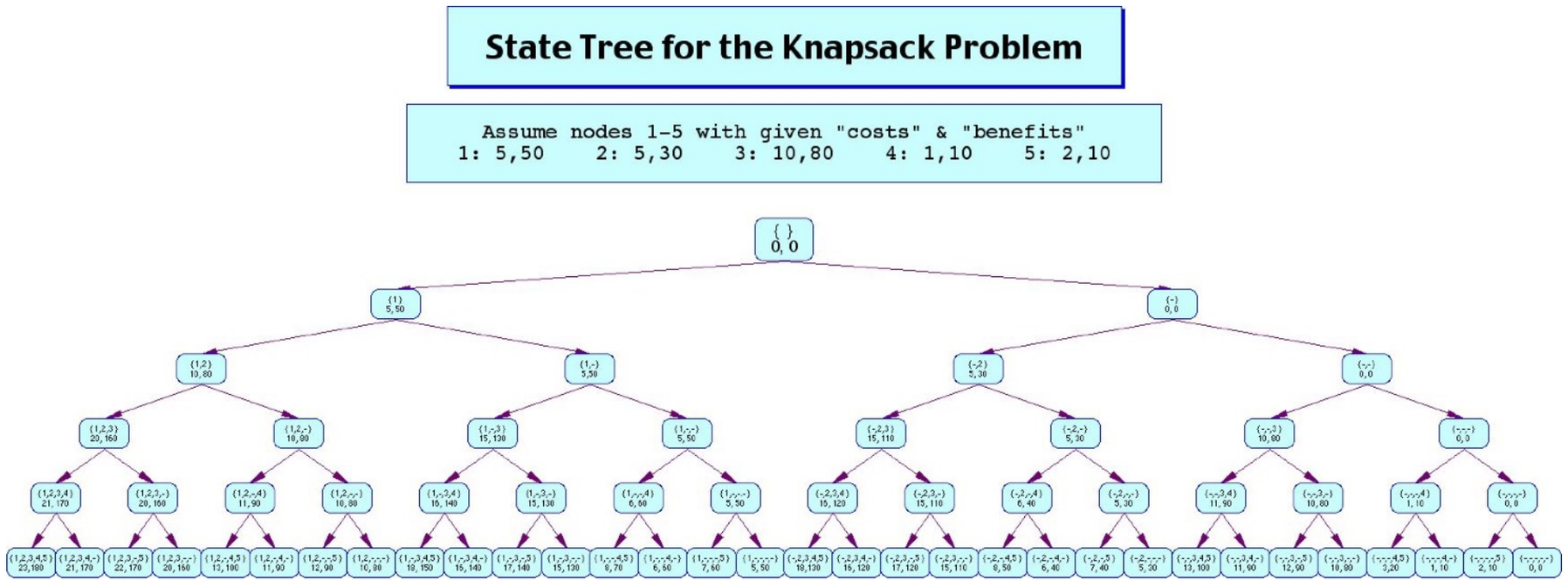
# 0-1 VS. fractional knapsack problem

- 0-1 fractional knapsack problem
  - Either take (1) or leave an object (0)
  - Greedy fails to produce global optimum
- fractional knapsack problem
  - you can take fractions of an object
  - Greedy strategy: value per weight $v/w$
    —> begin taking as much as possible of item with greatest $v/w$, then with next greater $v/w$,…
  - Leads to global optimum (proof by contradiction)
- What is the difference?



# Alternatives 0-1 knapsack

- Brute-Force
  - benefit: it finds the optimum
  - drawback: it takes very long — $O(2^n)$
  - because recomputing the results of the same subproblems over and over again



# Dynamic programming

- Optimal substructure:
  - optimal solution to problem consists of optimal solutions to subproblems
- Overlapping subproblems:
  - few subproblems in total, many recurring instances of each
- Main idea:
  - use a table to store solved subproblems