Greedy Algorithms

- Knapsack
- Coin Change
- Huffman Code
- Scheduling

Optimization Problems

- Optimization problem: a problem of finding the best solution from all feasible solutions.

- Two common techniques:
  - Greedy Algorithms
  - Dynamic Programming (global)

Elements of Greedy Strategy

- ***Greedy-choice property***: A global optimal solution can be arrived at by making locally optimal (greedy) choices

- ***Optimal substructure***: an optimal solution to the problem contains within it optimal solutions to sub-problems

Greedy Algorithms

A greedy algorithm works in phases. At each phase:

- You take the best you can get right now, without regard for future consequences
- You hope that by choosing a *local* optimum at each step, you will end up at a *global* optimum

Greedy algorithms typically consist of

- A set of ***candidate solutions***
- ***Function*** that checks if the candidates are ***feasible***
- ***Selection function*** indicating at a given time which is the most **promising candidate** not yet used
- ***Objective function*** giving the value of a solution; this is the function we are trying to optimize

Analysis

- The selection function is usually based on the objective function; they may be identical. But, often there are several plausible ones.
- At every step, the procedure chooses the best **candidate**, without worrying about the future. It never changes its mind: **once a candidate is included in the solution, it is there for good; once a candidate is excluded, it's never considered again.**
- Greedy algorithms do NOT always yield optimal solutions, but for many problems they do.

Greedy vs DP

- Greedy and Dynamic Programming are methods for solving optimization problems.
- Greedy algorithms are usually more efficient than DP solutions.
- However, often you need to use dynamic programming since the optimal solution cannot be guaranteed by a greedy algorithm.
- DP provides efficient solutions for some problems for which a brute force approach would be very slow.
- To use Dynamic Programming we need only show that the principle of optimality applies to the problem.

Examples of Greedy Algorithms

- Knapsack
- Coin Change
- Data compression
  - Huffman coding
- Scheduling
  - Activity Selection
  - Task Scheduling
  - Minimizing time in system
  - Deadline scheduling
- Graph Algorithms
  - Breath First Search (shortest path 4 un-weighted graph)
  - Dijkstra’s (shortest path) Algorithm
  - Minimum Spanning Trees

The 0/1 Knapsack problem

- Given a knapsack with weight $W > 0$.
- A set S of n items with weights $w_i > 0$ and benefits $b_i > 0$ for $i = 1,\ldots,n$.
- $S = \{ (item_1, w_1, b_1), (item_2, w_2, b_2), \ldots, (item_n, w_n, b_n) \}$
- Find a subset of the items which does not exceed the weight W of the knapsack and maximizes the benefit.

0/1 Knapsack problem

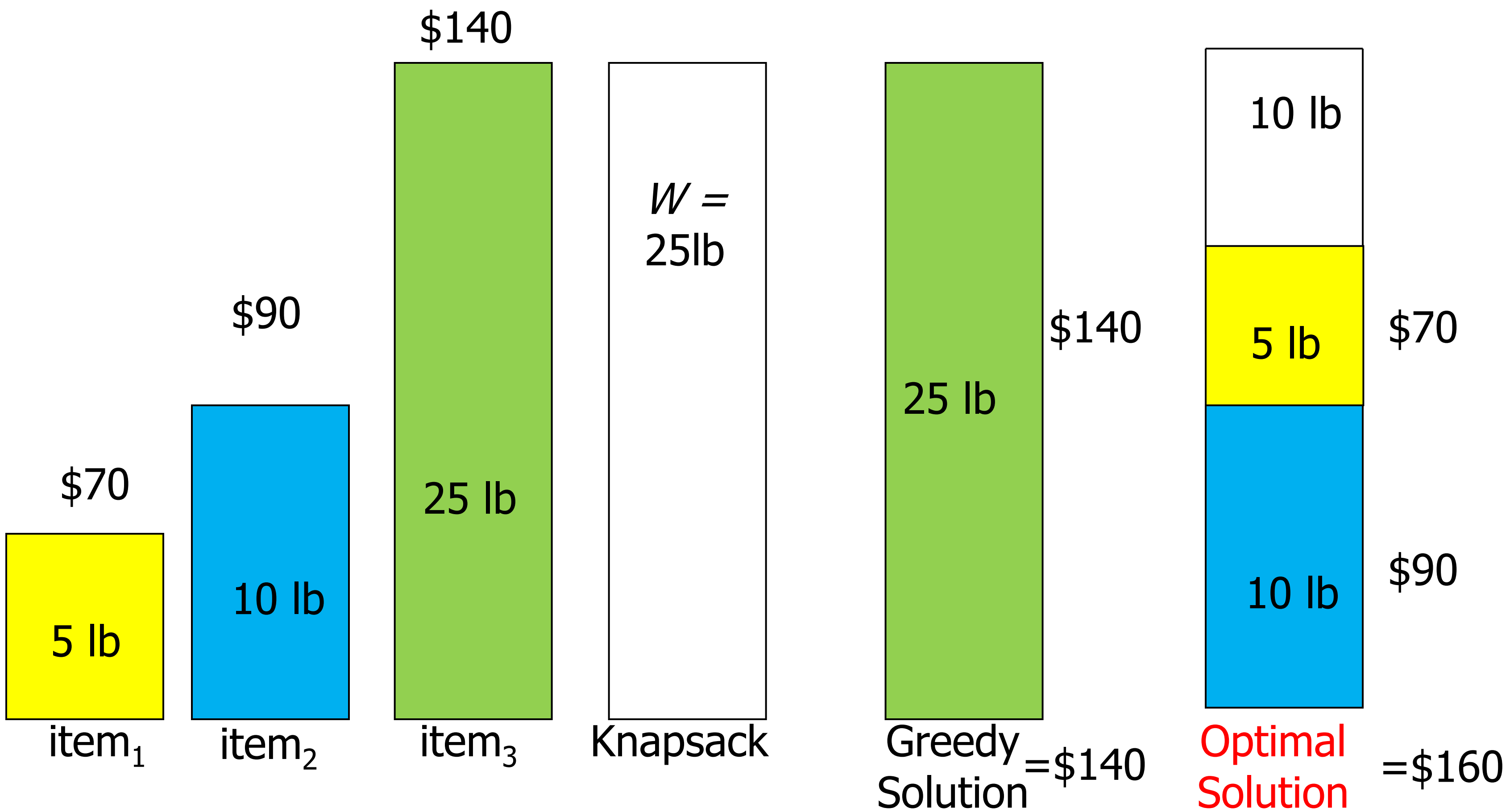
Determine a subset *T* of { 1, 2, …, *n* } that satisfies the following:

$$\max \sum_{i \in T} b_i \text{ where } \sum_{i \in T} w_i \le W$$

In 0/1 knapsack a specific item is either selected or not

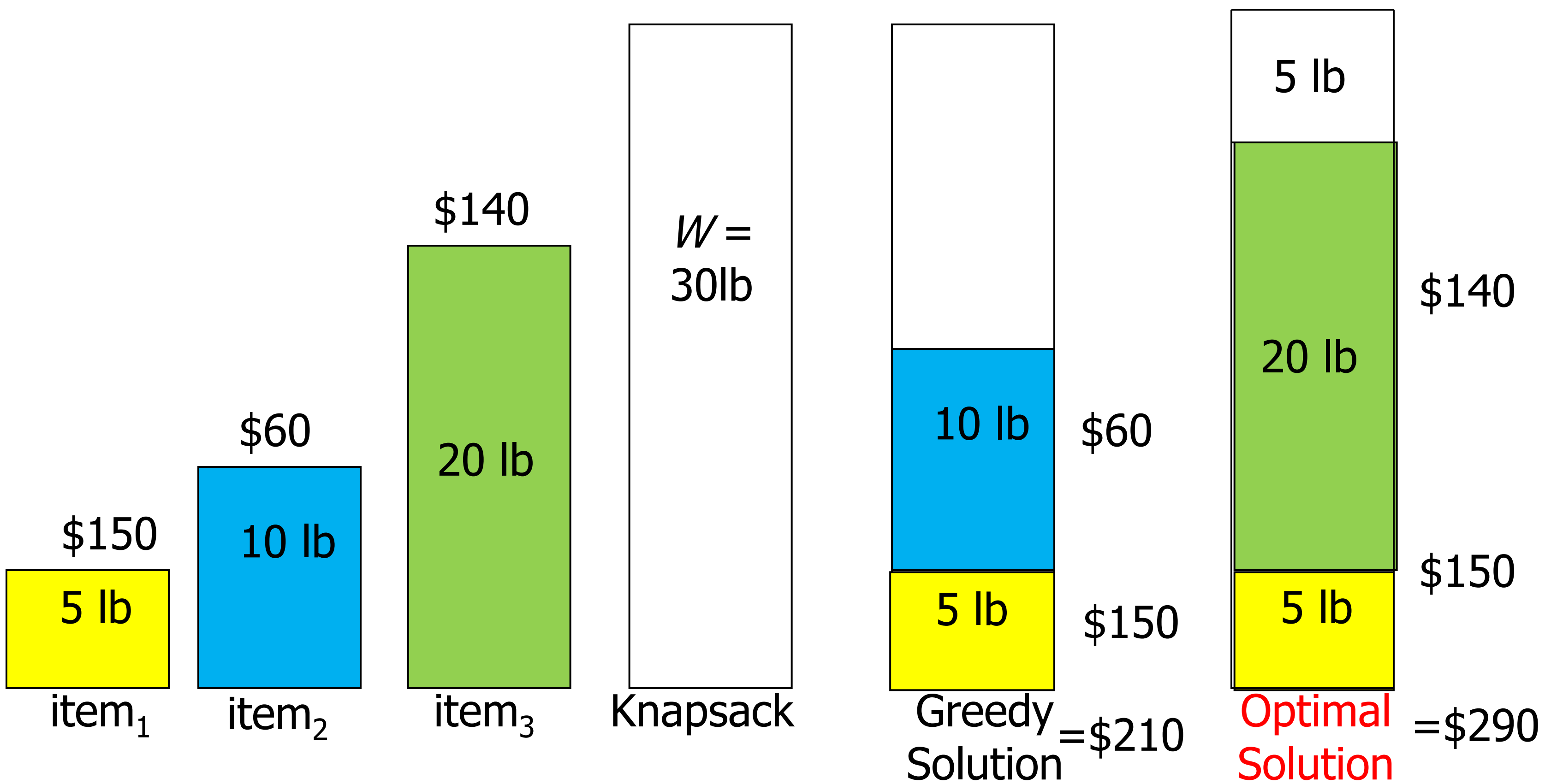
Greedy 1: Selection criteria: *Maximum beneficial* item.
Counter Example:

$S = \{ (item_1 , 5, \$70), (item_2 ,10, \$90), (item_3, 25, \$140) \}$



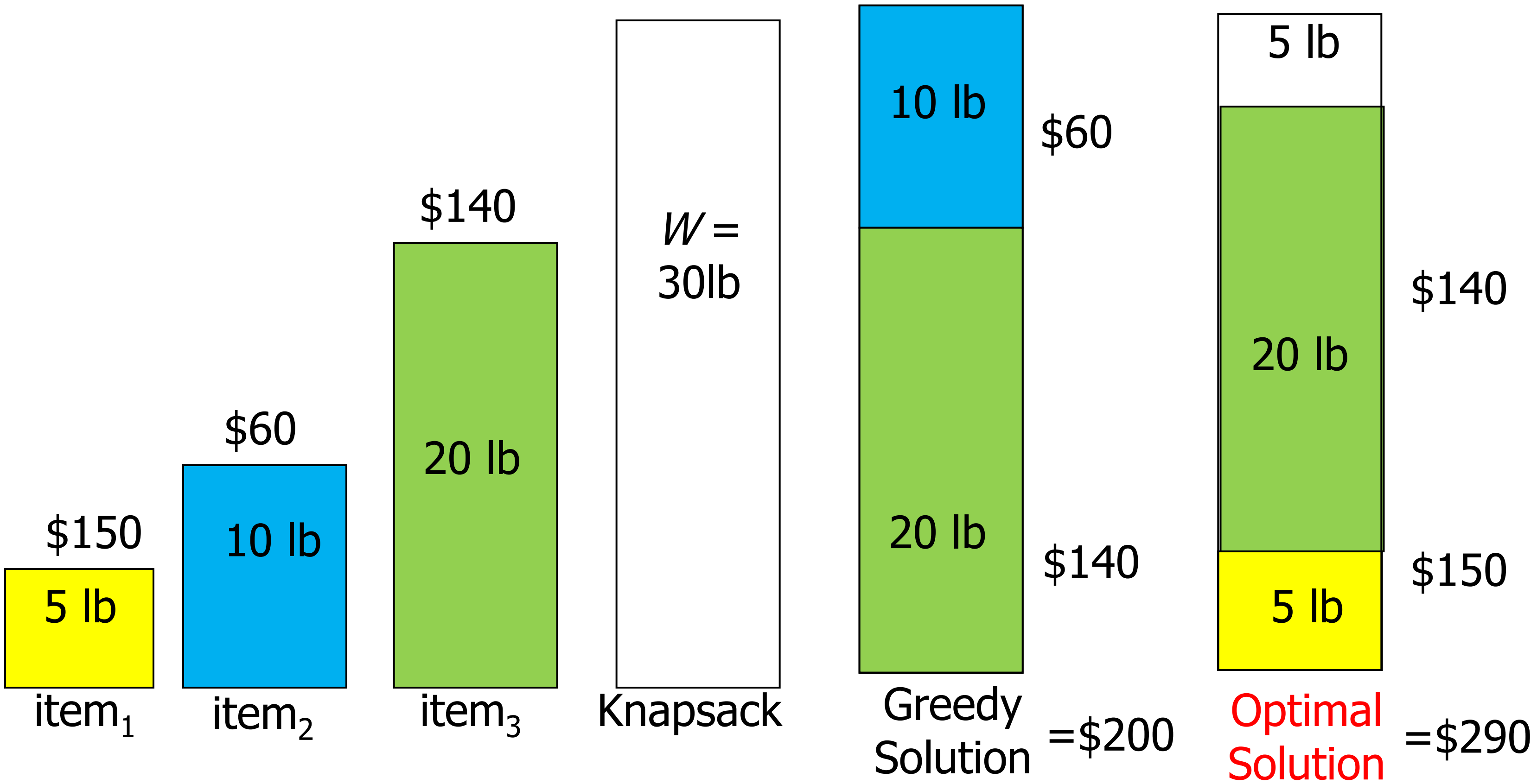
Greedy 2: Selection criteria: *Minimum weight* item
Counter Example:

$S = \{ (item_1 , 5, \$150), (item_2 ,10, \$60), (item_3, 20, \$140) \}$



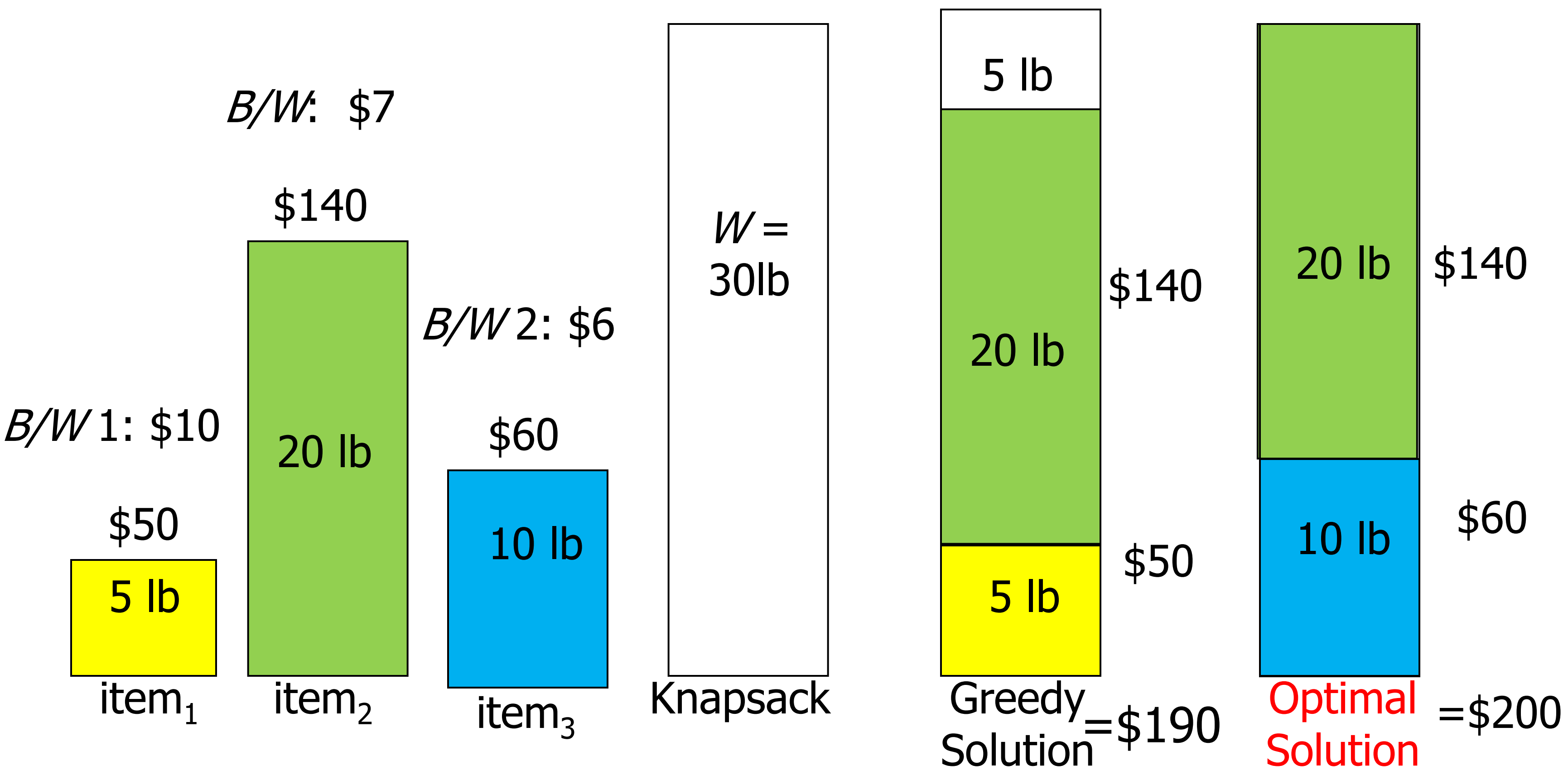
Greedy 3: Selection criteria: *Maximum weight* item
Counter Example:

$S = \{ (\ item_1 , 5, \$150), (item_2 ,10, \$60), (\ item_3, 20, \$140) \}$



Greedy 4: Selection criteria: *Maximum benefit per unit* item
Counter Example

$S = \{ (item_1 , 5, \$50), (item_2, 20, \$140) (item_3 ,10, \$60), \}$



Fractional Knapsack

Let *k* be the index of the last item included in the knapsack. We may be able to include the whole or only a fraction of item *k*

Without item k totweight = $\sum_{i=1}^{k-1} w_i$

FWK = $\sum_{i=1}^{k-1} b_i$ + min{(***W*** - *totweight),*w_k } X (b_k / w_k)

min{(***W*** - *totweight),*w_k }, means that we either take the whole of item k when the knapsack can include the item without violating the constraint, or we fill the knapsack by a fraction of item

A Greedy Algorithm for Fractional Knapsack

In this problem a fraction of any item may be chosen

The greedy algorithm uses the *maximum benefit per unit* selection criteria

1. Calculate $v_i = b_i / w_i$ for $1 \leq i \leq n$ Θ(n)
2. Sort items in decreasing b_i / w_i. Θ(nlgn)
3. *Add items to knapsack (starting at the first) until there are no more items*, or until the capacity W is exceeded.

   If knapsack is not yet full, fill knapsack with a fraction of next unselected item. Θ(n)

Running time: Θ(nlgn)

The Fractional Knapsack Algorithm

- Greedy choice: Keep taking item with highest **value** (benefit to weight ratio)
  - Use a heap-based priority queue to store the items, then the time complexity is O(n log n).

Algorithm ***FKnapsack***(***S, W***)

Input: set ***S*** of items w/ benefit b_i and weight w_i; max. weight W

Output: amount x_i of each item i to maximize benefit with weight at most W

for *each item i in S*

$\quad \boldsymbol{x_i} \leftarrow 0$

$\quad \boldsymbol{v_i} \leftarrow \boldsymbol{b_i} / \boldsymbol{w_i}$ {value}

$\boldsymbol{w} \leftarrow 0$ {current total weight}

while $\boldsymbol{w} < \boldsymbol{W}$

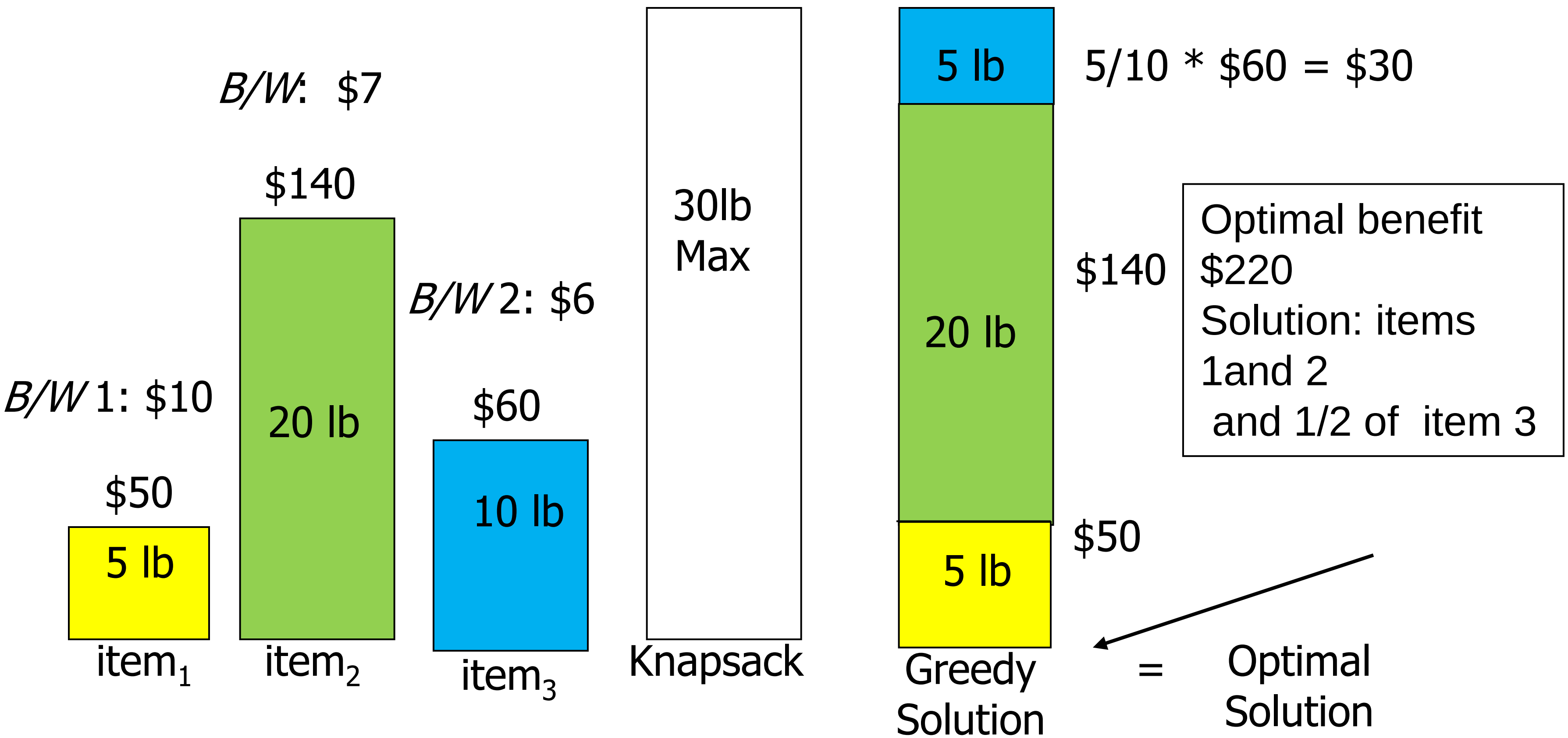
$\quad$ remove item $\boldsymbol{i}$ with highest $\boldsymbol{v_i}$

$\quad \boldsymbol{x_i} \leftarrow \min\{\boldsymbol{w_i}, \boldsymbol{W} - \boldsymbol{w}\}$

$\quad \boldsymbol{w} \leftarrow \boldsymbol{w} + \min\{\boldsymbol{w_i}, \boldsymbol{W} - \boldsymbol{w}\}$

Example of applying the optimal greedy algorithm for Fractional Knapsack Problem

$S = \{ (\ item_1, 5, \$50\ ), (\ item_2, 20, \$140\ )\ (item_3, 10, \$60\ ), \}$



Example of applying the optimal greedy algorithm for Fractional Knapsack Problem

W=30
S = { (item1 , 5, $50), (item2 ,20, $140), (item3, 10, $60) }
Note: items are already sorted by benefit/weight

Applying the algorithm:
Current weight in knapsack=0, Current benefit=0.
Can item 1 fit? 0+5<30 so select it. Current benefit=0+50
Can item 2 fit? 5+20<30, so select. Current benefit =50+140=190
Can item 3 fit? 25+10>30. No.
We can add 5 to knapsack (30-25).

So select 5/10=0.5 of item 3.
Current benefit=190+30=220

Fractional Knapsack has greedy choice property

That is, if b_i/w_i is the maximum ratio, then there exists an optimal solution that contains item x_i up to the extent of min{w_i, W}.

Proof (by contradiction): Assume that there does not exist an optimal solution that contains x_i. Let O = {x_j , .., x_k } be an optimal solution that does not contain x_i. Let x_t be the item with maximum weight w_t in O.

1) If $w_t \geq w_i$, then replacing w_i amount of x_t by w_i amount of x_i. This will either increase the value of the solution if $b_i/w_i > b_t/w_t$ or be an alternative maximum solution if $b_i/w_i = b_t/w_t$

2) If $w_t < w_i$, then

   a) Let S be a subset of items in O whose is total weight is greater than w_i. Replacing w_i of this total weight by w_i of x_i will improve the value of the solution.

Fractional Knapsack has greedy choice property

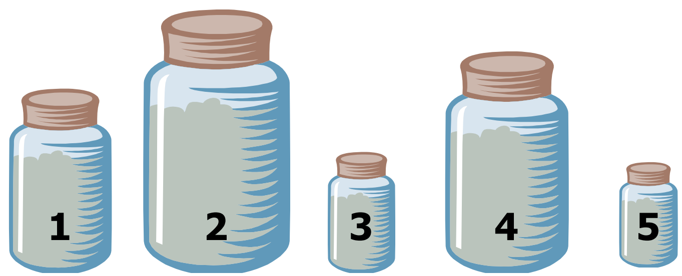
b) If no such set S exists then the sum of the weights of all items in O = W ≤ w_i. Replace all the items in O by W units of x_i and the solution will improve (or leading to an alternative solution containing x_i).

Therefore we have shown that adding item x_i to O will improve the solution or lead to an alternative maximum solution.

Another Example

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

Items:



Items:	1	2	3	4	5
Weight:	4 ml	8 ml	2 ml	6 ml	1 ml
Benefit:	\$12	\$32	\$40	\$30	\$50
Value: ($ per ml)	3	4	20	5	50

10 ml

"knapsack"

Solution:

- 1 ml of 5
- 2 ml of 3
- 6 ml of 4
- 1 ml of 2

Coin Change

- Coin changing problem (informal):
  - Given certain amount of change: A
  - The denominations of coins are: 25, 10, 5, 1
  - How to use the fewest coins to make this change?
- A = 25q + 10d + 5n + p, what are the q, d, n, and p, minimizing (q+d+n+p)
- Can you design an algorithm to solve this problem?

Coin changing problem

- Greedy choice
  - Choose as many of the largest coins available.
- Optimal substructure
  - After the greedy choice, assuming the greedy choice is correct, can we get the optimal solution from a subproblem.
  - Given A = 63 cents
    - Assuming we have chosen 2*25 = 50
    - Is two quarters + optimal **coin**(63-50) the optimal solution of 63 cents?

Coin Change

- Step 1: A = 63

Coin Change

- Step 1: $A = 63$, $q = 2$ 

Coin Change

- Step 1: A = 63, q = 2 

- Step 2: (63-50) = 13

Coin Change

- Step 1: A = 63, q = 2 

- Step 2: (63-50) = 13, d = 1 

Coin Change

- Step 1: A = 63, q = 2 

- Step 2: (63-50) = 13, d = 1 

- Step 3: (13-10) = 3

Coin Change

- Step 1: A = 63, q = 2 

- Step 2: (63-50) = 13, d = 1 

- Step 3: (13-10) = 3

Coin Change

- Step 1: A = 63, q = 2 
- Step 2: (63-50) = 13, d = 1 
- Step 3: (13-10) = 3, p = 3 

Coin Change

- Step 1: A = 63, q = 2 
- Step 2: (63-50) = 13, d = 1 
- Step 3: (13-10) = 3, p = 3 
- Number of coins = 6

Coin Change

- Step 1: A = 63, q = 2

- Step 2: (63-50) = 13, d = 1

- Step 3: (13-10) = 3, p = 3

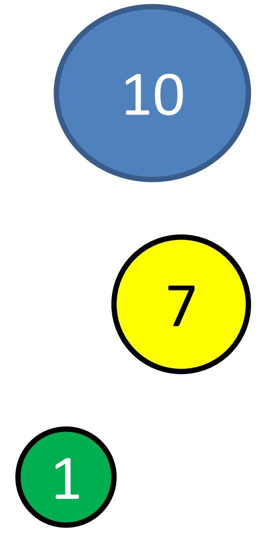
- Number of coins = 6
- For coin denominations of 25, 10, 5, 1
  - The greedy choice property is not violated

A failure of the Greedy Algorithm

- Suppose in a fictional monetary system, we have 1 cent, 7 cent, and 10 cent coins
- The greedy algorithm results in a solution, but not in an optimal solution

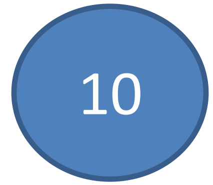
Coin Change Fail

- Step 1: A = 15

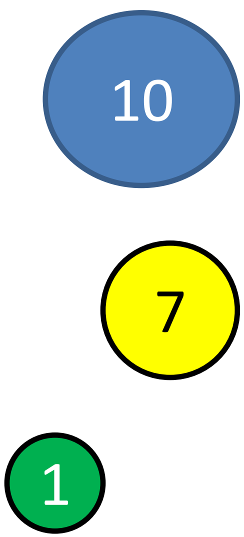


Coin Change Fail

- Step 1: A = 15

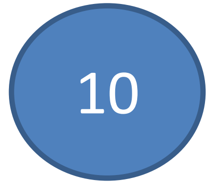


- Step2: (15-10) = 5



Coin Change Fail

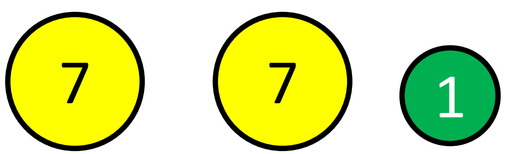
- Step 1: A = 15

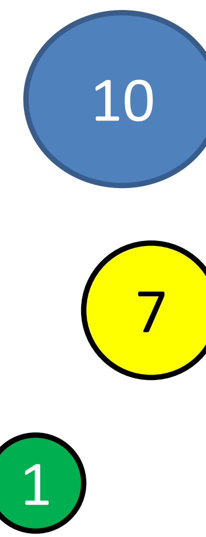


- Step2: (15-10) = 5

This is six coins

The optimal solution is three coins





Greedy and Coin Change

Only optimal under certain conditions.

- The problem has the optimal substructure property
- The algorithm satisfies the greedy-choice property
- You will explore this more in Project 2