DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

# *Analysis and Synthesis of Algorithms*
# *Design of Algorithms*

## Brute-Force Algorithmic Approach

# Overview

- What is the Brute-Force Approach
- Why Study it?
- Examples
- Limitations

# What is Brute Force?

- A straightforward Approach to Problem Solving
- Based on Problem Statement and definitions of the concepts involved
- Examples:
  – Numeric Calculations
  – Search by Enumeration of all possible domain points
- Why the name "Brute Force" ?
  – No cleverness on the implementation
  – Straightforward Implementation (very simple algorithm)
  – Uses computing power not cleverness…

# Why Brute Force?

- Measure (or yard stick) for Algorithm Performance
  - Space and Time
- Correctness and Optimality
  - Algorithm is simple – it's correctness is trivially established
  - Optimality is usually ensured – all domain is explored…

# Brute Force Example

- We want to compute $a^n = \underbrace{a \times a \times \ldots \times a}_{\textbf{n times}}$

- In RSA (Rivest, Shamir, and Adleman) encryption algorithm we need to compute $\mathbf{a^n \bmod m}$ for a > 1 and large n.
- Basic Approach: Multiply **1** by **a** **n** times
  - So, **n-1** multiplications by **a**…
- Can we do Better?

# Brute Force Example

- Given n orderable items (*e.g.*, numbers) how can you rearrange them in non-decreasing order?
- **SelectionSort**:
  - Several passes (i goes from 0 to n-2)
  - Searches for the smallest item among the last (n-i) elements and swaps it with $A_i$

$$A_0 \leq A_1 \leq \ldots \leq A_{i-1} \mid A_i, \ldots\ldots, A_{min}, \ldots\ldots, A_{n-1}$$

**already sorted** **the last n-i elements**

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

# Brute Force: Selection Sort

**SelectionSort**(A[0,..n-1])
**for** i ⇐ 0 **to** n-2 **do**
min ⇐ i
**for** j ⇐ i+1 **to** n-1 **do**
**if** A[j] < A[min]
min ⇐ j
swap A[i] and A[min]

| 89 45 68 90 29 34 **17**
17 | 45 68 90 **29** 34 89
17 29 | 68 90 45 **34** 89
17 29 34 45 | 90 **68** 89
17 29 34 45 68 | 90 **89**
17 29 34 45 68 89 | 90

**Input size: n (number of integer values)**
**Key op: "<", does not depend on type**

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$
$$= \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1]$$
$$= \frac{(n-1)n}{2}$$

$C(n) = \Theta(n^2)$
# of key swaps = $\Theta(n)$

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

# Searching

- Search for a key, K in an array A[0 … n-1]?

# Sequential Search

**SequentialSearch**(A[0..n-1], K)
//Output: index of the first element in A, whose
//value is equal to K or -1 if no such element is found

i = 0
**while** i < n **and** A[i] ≠ K **do**
  i = i+1
 **if** i < n
  **return** i
 **else**
  **return** -1

Input size: n
Basic op: <, ≠

$C_{worst}(n) = 2n+2$

**Can you improve on it?**

# Searching

- Search for a key, K in an array A[0 … n-1]?
- Basic Brute-Force is $\Theta(n)$
- What if the Elements are Sorted (non-decreasing order)?

# Exhaustive Search

- Traveling Salesman Problem (TSP)
  - Find the shortest tour through a given set of n cities that visits each city exactly once before returning to the city where it started
  - Can be conveniently modeled by a weighted graph; vertices are cities and edge weights are distances
  - Same as finding "Hamiltonian Circuit": find a cycle that passes through all vertices exactly once

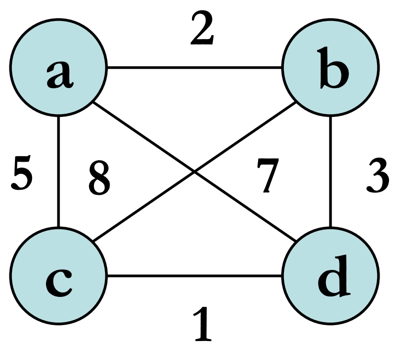
# Exhaustive Search: TSP (cont.)

- Hamiltonian circuit: A sequence of n+1 adjacent vertices $v_{i_0}, v_{i_1}, \ldots, v_{i_{n-1}}, v_{i_0}$

***How can we solve TSP?***

- **Solution:** Derive all possible tours by generating all permutations of (n-1) intermediate cities, and compute the tour length. Find the shortest among them

# Traveling Salesman (TSP)




Consider only when b precedes c

$\frac{1}{2}$ **(n-1)! permutations**

| | |
|---|---|
| a → b → c → d → a | 2+8+1+7 = 18 |
| a → b → d → c → a | 2+3+1+5 = 11 ⟵ **optimal** |
| a → c → b → d → a | 5+8+3+7 = 23 |
| a → c → d → b → a | 5+1+3+2 = 11 ⟵ **optimal** |
| a → d → b → c → a | 7+3+8+5 = 23 |
| a → d → c → b → a | 7+1+8+2 = 18 |

# Exhaustive Search: Knapsack Problem

- Given n items of weights $w_1$, $w_2$, …, $w_n$ and values $v_1$, $v_2$, …, $v_n$ and a knapsack of capacity W, find the most valuable subset of the items that fit into the knapsack
  - A transport plane has to deliver the most valuable set of items to a remote location without exceeding its capacity

***How can we solve it ?***

# Knapsack Problem (contd.)

- **Solution:** Generate all possible subsets of the n items, compute total weight of each subset to identify feasible subsets, and find the subset of the largest value

***What is the time efficiency ?***

$$\Omega(2^n)$$



U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

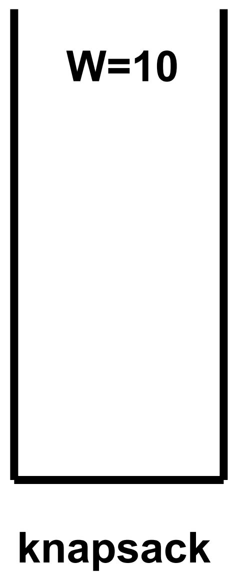
# Knapsack Problem (contd.)

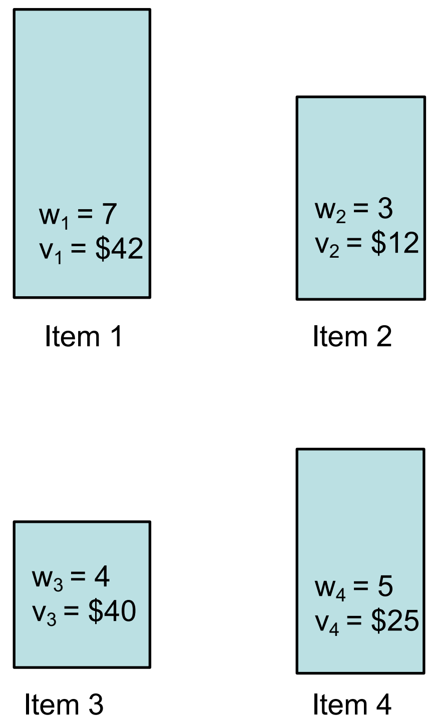
- Alternative Algorithmic Strategy:
  - How about taking items in decreasing order of value/weight?
  - Subject to fitting in the remaining knapsack availability…
  - Sort all items, them pick them linearly while they fit in knapsack.

***What is the time efficiency ?***

$$\Omega(n \log n)$$

# Knapsack Problem (contd.)

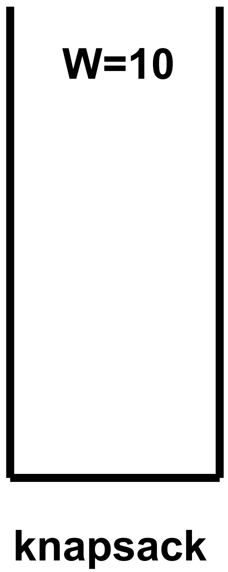


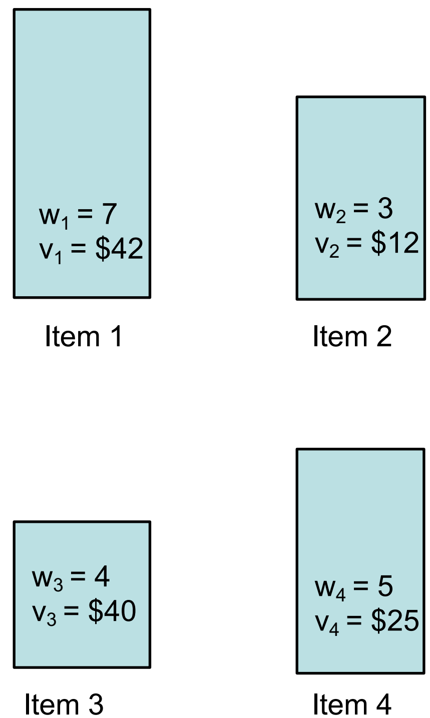




**Item 1: \$6/unit**
**Item 2: \$4/unit**
**Item 3: \$10/unit**
**Item 4: \$5/unit**

| subset | weight | value |
|---|---|---|
| Ø | 0 | \$0 |
| {1} | 7 | \$42 |
| {2} | 3 | \$12 |
| {3} | 4 | \$40 |
| {4} | 5 | \$25 |
| {1,2} | 10 | \$54 |
| {1,3} | 11 | !feasible |
| {1,4} | 12 | !feasible |
| {2,3} | 7 | \$52 |
| {2,4} | 8 | \$37 |
| **{3,4}** | **9** | **\$65** |
| {1,2,3} | 14 | !feasible |
| {1,2,4} | 15 | !feasible |
| {1,3,4} | 16 | !feasible |
| {2,3,4} | 12 | !feasible |
| {1,2,3,4} | 19 | !feasible |

# Knapsack Problem (contd.)



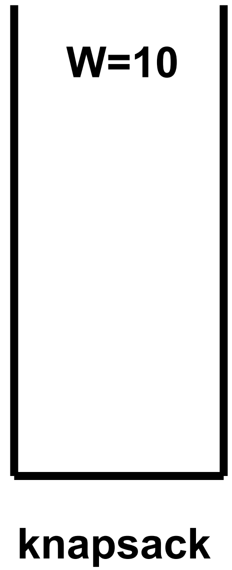



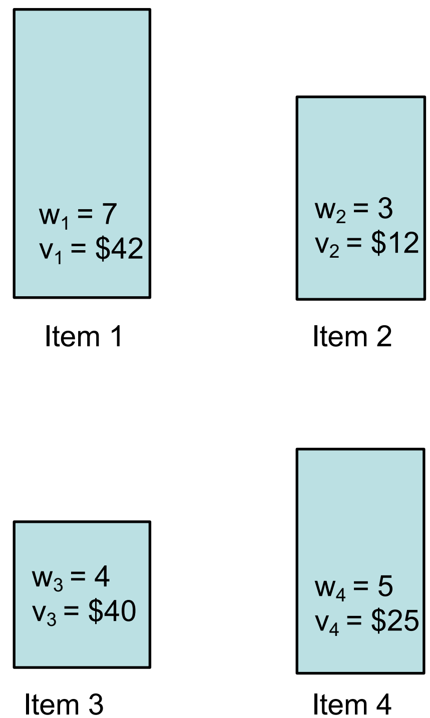

**Item 1: $6/unit**
**Item 2: $4/unit**
**Item 3: $10/unit**
**Item 4: $5/unit**

**Picking Order: {3, 1, 4, 2}** **Weight:**

| subset | weight | value |
|---|---|---|
| Ø | 0 | $0 |
| {1} | 7 | $42 |
| {2} | 3 | $12 |
| {3} | 4 | $40 |
| {4} | 5 | $25 |
| {1,2} | 10 | $54 |
| {1,3} | 11 | !feasible |
| {1,4} | 12 | !feasible |
| {2,3} | 7 | $52 |
| {2,4} | 8 | $37 |
| **{3,4}** | **9** | **$65** |
| {1,2,3} | 14 | !feasible |
| {1,2,4} | 15 | !feasible |
| {1,3,4} | 16 | !feasible |
| {2,3,4} | 12 | !feasible |
| {1,2,3,4} | 19 | !feasible |

# Knapsack Problem (contd.)



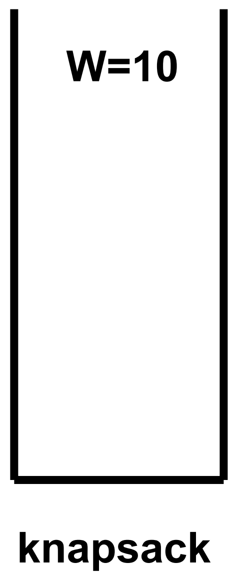



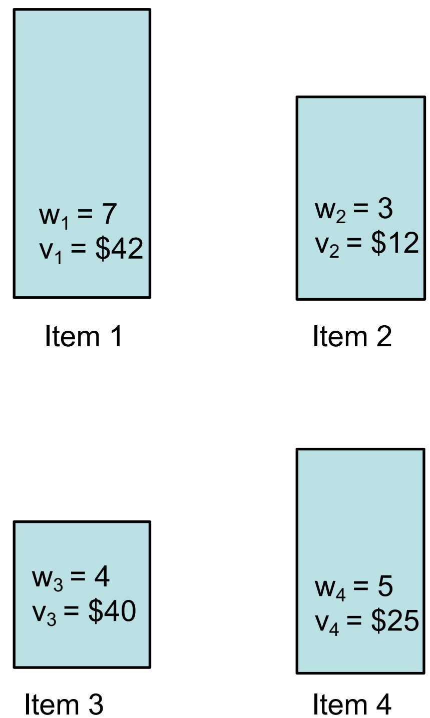

Item 1: $6/unit
Item 2: $4/unit
Item 3: $10/unit
Item 4: $5/unit

**Picking Order: {3, 1, 4, 2}** **Weight: 4**

| subset | weight | value |
|---|---|---|
| Ø | 0 | $0 |
| {1} | 7 | $42 |
| {2} | 3 | $12 |
| {3} | 4 | $40 |
| {4} | 5 | $25 |
| {1,2} | 10 | $54 |
| {1,3} | 11 | !feasible |
| {1,4} | 12 | !feasible |
| {2,3} | 7 | $52 |
| {2,4} | 8 | $37 |
| **{3,4}** | **9** | **$65** |
| {1,2,3} | 14 | !feasible |
| {1,2,4} | 15 | !feasible |
| {1,3,4} | 16 | !feasible |
| {2,3,4} | 12 | !feasible |
| {1,2,3,4} | 19 | !feasible |

# Knapsack Problem (contd.)



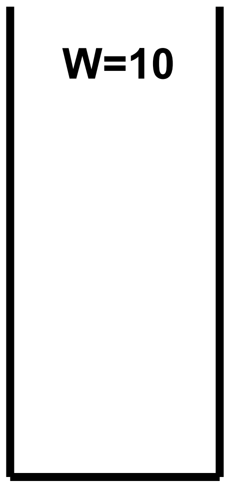




Item 1: $6/unit
Item 2: $4/unit
Item 3: $10/unit
Item 4: $5/unit

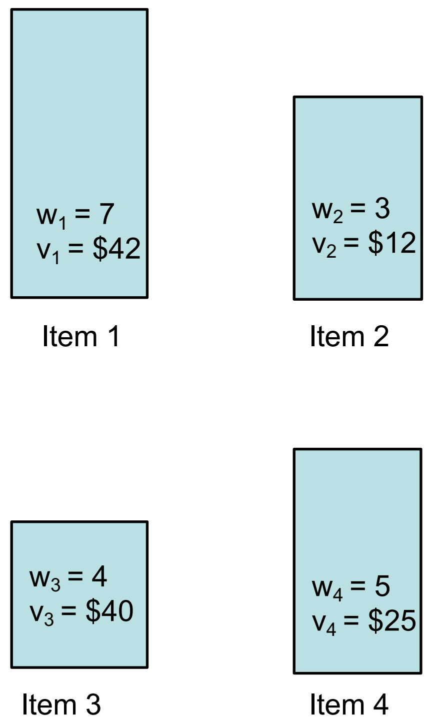
**Picking Order: {3, 1, 4, 2}**  **Weight: 4+7=~~11~~**

| subset | weight | value |
|---|---|---|
| Ø | 0 | $0 |
| {1} | 7 | $42 |
| {2} | 3 | $12 |
| {3} | 4 | $40 |
| {4} | 5 | $25 |
| {1,2} | 10 | $54 |
| {1,3} | 11 | !feasible |
| {1,4} | 12 | !feasible |
| {2,3} | 7 | $52 |
| {2,4} | 8 | $37 |
| **{3,4}** | **9** | **$65** |
| {1,2,3} | 14 | !feasible |
| {1,2,4} | 15 | !feasible |
| {1,3,4} | 16 | !feasible |
| {2,3,4} | 12 | !feasible |
| {1,2,3,4} | 19 | !feasible |

# Knapsack Problem (contd.)




knapsack

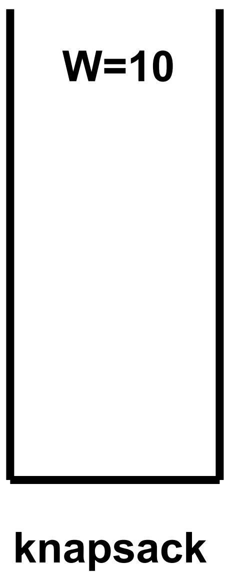


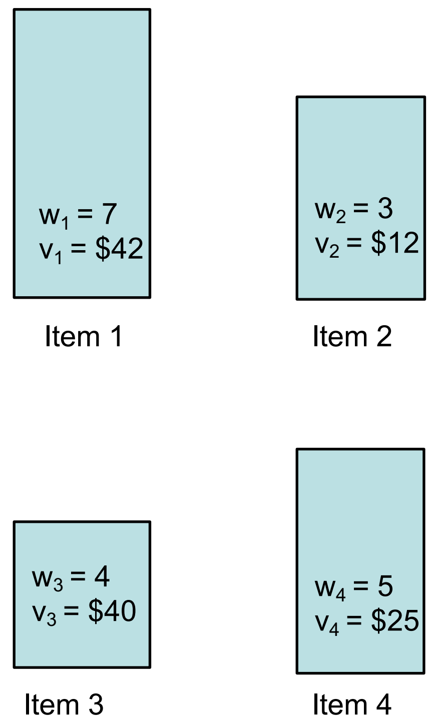

Item 1: $6/unit
Item 2: $4/unit
Item 3: $10/unit
Item 4: $5/unit

Picking Order: {3, 1, 4, 2}   Weight: 4+5=9

| subset | weight | value |
|---|---|---|
| Ø | 0 | $0 |
| {1} | 7 | $42 |
| {2} | 3 | $12 |
| {3} | 4 | $40 |
| {4} | 5 | $25 |
| {1,2} | 10 | $54 |
| {1,3} | 11 | !feasible |
| {1,4} | 12 | !feasible |
| {2,3} | 7 | $52 |
| {2,4} | 8 | $37 |
| **{3,4}** | **9** | **$65** |
| {1,2,3} | 14 | !feasible |
| {1,2,4} | 15 | !feasible |
| {1,3,4} | 16 | !feasible |
| {2,3,4} | 12 | !feasible |
| {1,2,3,4} | 19 | !feasible |

# Knapsack Problem (contd.)



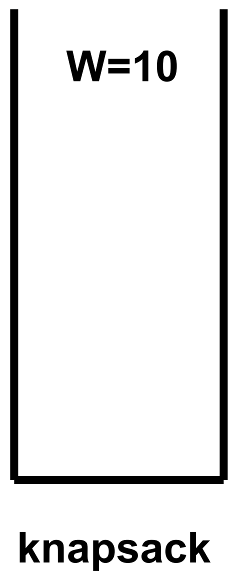



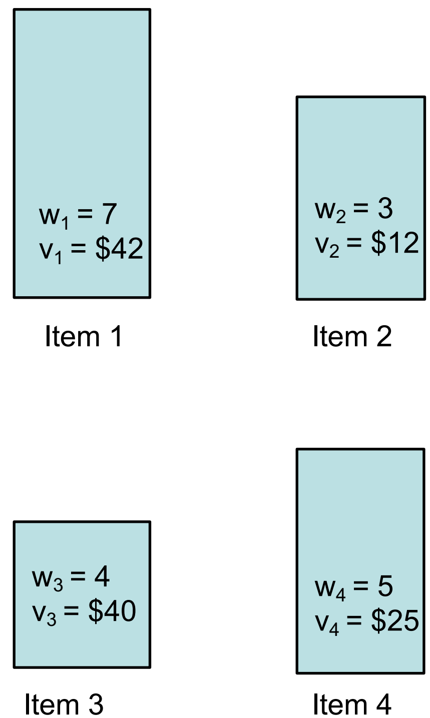

**Item 1: $6/unit**
**Item 2: $4/unit**
**Item 3: $10/unit**
**Item 4: $5/unit**

**Picking Order: {3, 1, 4, 2}**  **Weight: $9+3=\cancel{12}$**

| subset | weight | value |
|---|---|---|
| ∅ | 0 | $0 |
| {1} | 7 | $42 |
| {2} | 3 | $12 |
| {3} | 4 | $40 |
| {4} | 5 | $25 |
| {1,2} | 10 | $54 |
| {1,3} | 11 | !feasible |
| {1,4} | 12 | !feasible |
| {2,3} | 7 | $52 |
| {2,4} | 8 | $37 |
| **{3,4}** | **9** | **$65** |
| {1,2,3} | 14 | !feasible |
| {1,2,4} | 15 | !feasible |
| {1,3,4} | 16 | !feasible |
| {2,3,4} | 12 | !feasible |
| {1,2,3,4} | 19 | !feasible |

# Knapsack Problem (contd.)



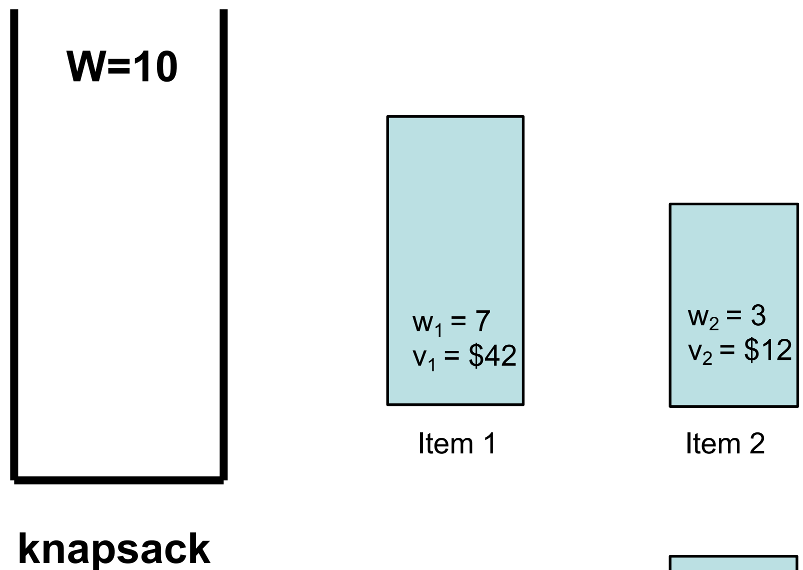

**Item 1: \$6/unit**
**Item 2: \$4/unit**
**Item 3: \$10/unit**
**Item 4: \$5/unit**

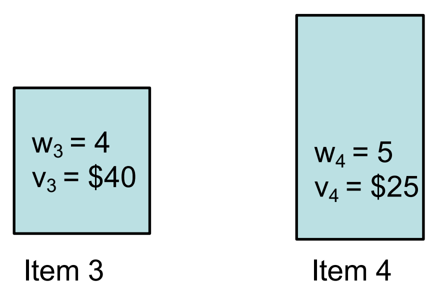
**Picking Order: {3, 1, 4, 2}**

**Weight: 9**
**Value: 65**

| subset | weight | value |
|---|---|---|
| Ø | 0 | \$0 |
| {1} | 7 | \$42 |
| {2} | 3 | \$12 |
| {3} | 4 | \$40 |
| {4} | 5 | \$25 |
| {1,2} | 10 | \$54 |
| {1,3} | 11 | !feasible |
| {1,4} | 12 | !feasible |
| {2,3} | 7 | \$52 |
| {2,4} | 8 | \$37 |
| **{3,4}** | **9** | **\$65** |
| {1,2,3} | 14 | !feasible |
| {1,2,4} | 15 | !feasible |
| {1,3,4} | 16 | !feasible |
| {2,3,4} | 12 | !feasible |
| {1,2,3,4} | 19 | !feasible |

# Knapsack Problem (contd.)



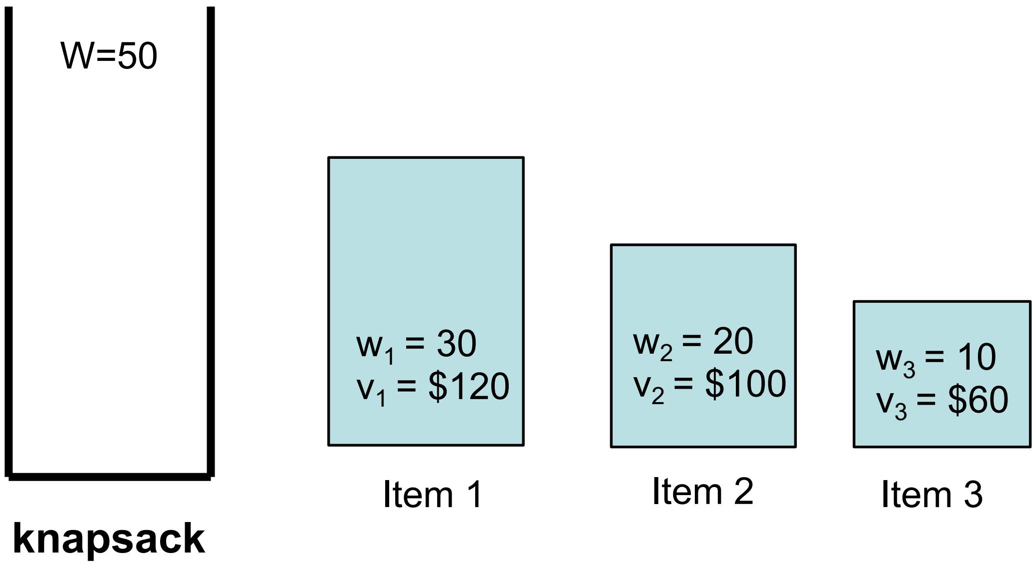




Item 1: $6/unit
Item 2: $4/unit
Item 3: $10/unit
Item 4: $5/unit

| subset | weight | value |
|---|---|---|
| Ø | 0 | $0 |
| {1} | 7 | $42 |
| {2} | 3 | $12 |
| {3} | 4 | $40 |
| {4} | 5 | $25 |
| {1,2} | 10 | $54 |
| {1,3} | 11 | !feasible |
| {1,4} | 12 | !feasible |
| {2,3} | 7 | $52 |
| {2,4} | 8 | $37 |
| **{3,4}** | **9** | **$65** |
| {1,2,3} | 14 | !feasible |
| {1,2,4} | 15 | !feasible |
| {1,3,4} | 16 | !feasible |
| {2,3,4} | 12 | !feasible |
| {1,2,3,4} | 19 | !feasible |

***Works for this example!***

# Knapsack Problem (contd.)




Item 1: \$4/unit
Item 2: \$5/unit
Item 3: \$6/unit

**{Item3, Item2} = \$60+\$100 = \$160**

{Item3, Item1} = \$60+\$120 = \$180

{Item2, Item1} = \$100+\$120 = \$220

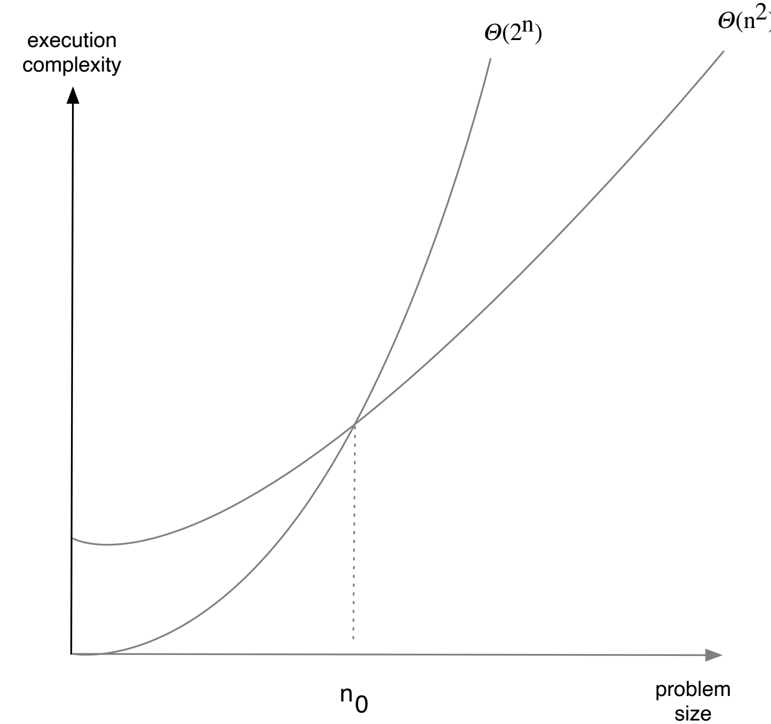
**Doesn't work for this example**

# Exhaustive Search at Large

- Traveling Salesman and Knapsack problems:
  - Exhaustive search leads to Exponential Time Complexity.
- These are NP-hard problems
  - No known Polynomial-time Algorithms
- Most famous unsolved problem in Computer Science:
  - P vs. NP problem
  - If you solve it, no need to Graduate…
  - More details in later chapters

# Brute-Force: A Note on Complexity

- Likely Exponential Search/Optimization
- Only Feasible for Small Problem Instances
- Still…
  - Simple Approach
  - Provides Correctness




- Think "Hybrid" Algorithms
  - For small problem sizes: Brute-Force
  - For large problem sizes: Polynomial (or better)

# Summary: Exhaustive Search

- A Brute Force Approach to Combinatorial Problems (which require generation of permutations, or subsets)
- Generate every element of Problem Domain
- Select Feasible ones (ones that satisfy constraints)
- Find the desired one (the one that optimizes some objective function)
- Works, but Only Feasible for Small Problem Sizes.