Greedy Algorithms

Lecture 12

Review: The Knapsack Problem

- *0-1 knapsack problem*:
  - The thief must choose among *n* items, where the *i*th item worth v_i dollars and weighs w_i pounds
  - Carrying at most *W* pounds, maximize value
    - Note: assume v_i, w_i, and *W* are all integers
    - "0-1" b/c each item must be taken or left in entirety
- A variation, the *fractional knapsack problem*:
  - Thief can take fractions of items
  - Think of items in 0-1 problem as gold ingots, in fractional problem as buckets of gold dust

Review: The Knapsack Problem And Optimal Substructure

- Both variations exhibit optimal substructure
- To show this for the 0-1 problem, consider the most valuable load weighing at most *W* pounds
    - If we remove item *j* from the load, what do we know about the remaining load?
    - A: remainder must be the most valuable load weighing at most $W - w_j$ that thief could take from museum, excluding item *j*

Solving The Knapsack Problem

- The optimal solution to the 0-1 problem cannot be found with the same greedy strategy
  - Greedy strategy: take in order of dollars/pound
  - Example: 3 items weighing 10, 20, and 30 pounds, knapsack can hold 50 pounds
    - Suppose 3 items are worth \$60, \$100, and \$120.
    - Will greedy strategy work?

Fractional knapsack problem

- The setup is same, but the thief can take fractions of items, meaning that the items can be broken into smaller pieces so that thief may decide to carry only a fraction of x_i of item i, where $0 \leq x_i \leq 1$.Exhibit greedy choice property.
  - Greedy algorithm exists.
- Exhibit optimal substructure property.
  - ?????

Greedy Solution - Fractional Knapsack Problem

- There are *n* items in a store. **For i =1,2, . . . , n**, item **i** has weight $w_i > 0$ and worth $v_i > 0$. Thief can carry a maximum weight of *W* pounds in a knapsack.
- In this version of a problem the items can be broken into smaller piece, so the thief may decide to carry only a fraction x_i of object *i*, where $0 \le x_i \le 1$. Item ***i*** contributes $x_i w_i$ to the total weight in the knapsack, and $x_i v_i$ to the value of the load.
- In Symbol, the fraction knapsack problem can be stated as follows. maximize $^{n}S_{i=1}\ x_i v_i$ subject to constraint $^{n}S_{i=1}\ x_i w_i \le W$
- It is clear that an optimal solution must fill the knapsack exactly, for otherwise we could add a fraction of one of the remaining objects and increase the value of the load. Thus in an optimal solution $^{n}S_{i=1}\ x_i w_i = W$.

The knapsack problem

- **n** objects, each with a weight $w_i > 0$

  a profit $V_i > 0$

  capacity of knapsack: W

Maximize
$$\sum_{1 \le i \le n} v_i x_i$$

Subject to
$$\sum_{1 \le i \le n} w_i x_i \le W$$

$0 \le x_i \le 1$, $1 \le i \le n$

The knapsack Pseudo Code

- The greedy algorithm:

  Step 1: Sort p_i/w_i into nonincreasing order.

  Step 2: Put the objects into the knapsack according

  to the sorted sequence as possible as we can.

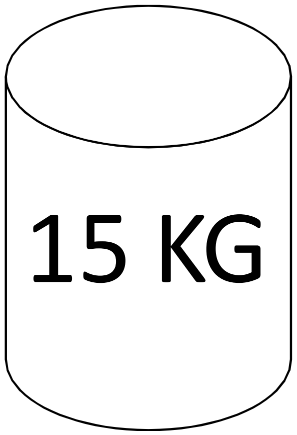
Algorithm

Greedy-fractional-knapsack ***(w, v, W)***

```
FOR $i$ =1 to  do
   $x[i]$ =0
weight = 0
while weight < $W$ do
   $i$ = best remaining item
       IF weight + $w[i]$ $\leq$ $W$ then
       $x[i]$ = 1
           weight = weight + $w[i]$
       else
           $x[i]$ = ($w$ - weight) / $w[i]$
           weight = $W$
return $x$
```

Knapsack Problem

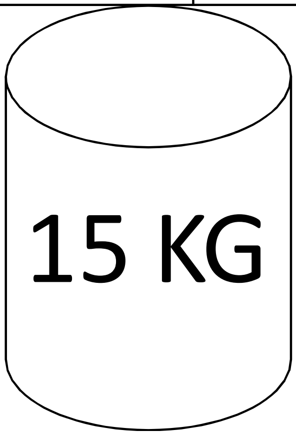
Objects**(O)**	1	2	3	4	5	6	7
Profits (P)	10	5	15	7	6	18	3
Weight (W)	2	3	5	7	1	4	1



Knapsack Problem

Objects**(O)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Profits (P)	**10**	**5**	**15**	**7**	**6**	**18**	**3**
Weight (W)	**2**	**3**	**5**	**7**	**1**	**4**	**1**

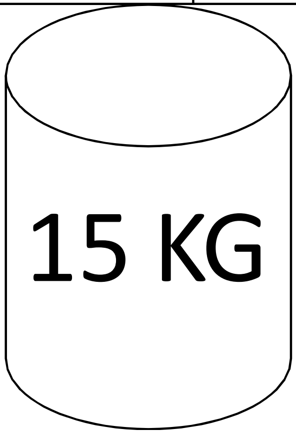
X	X_1	X_2	X_3	X_4	X_5	X_6	X_7



Knapsack Problem

Objects**(O)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Profits (P)	**10**	**5**	**15**	**7**	**6**	**18**	**3**
Weight (W)	**2**	**3**	**5**	**7**	**1**	**4**	**1**
P/W	**5**	**1.3**	**3**	**1**	**6**	**4.5**	**3**

X	X_1	X_2	X_3	X_4	X_5	X_6	X_7

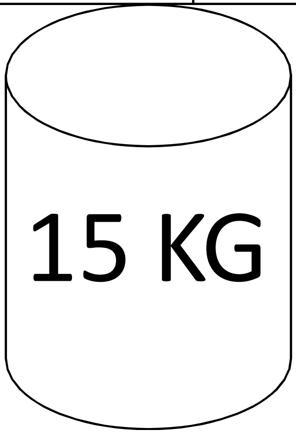


Knapsack Problem

Objects**(O)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Profits (P)	**10**	**5**	**15**	**7**	**6**	**18**	**3**
Weight (W)	**2**	**3**	**5**	**7**	**1**	**4**	**1**
P/W	**5**	**1.3**	**3**	**1**	**6**	**4.5**	**3**

X	X_1	X_2	X_3	X_4	X_5	X_6	X_7

15 – 1 = 14



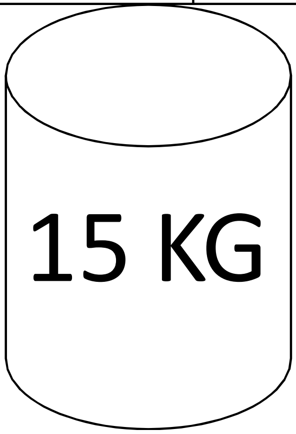
Knapsack Problem

Objects(**O**)	1	2	3	4	5	6	7
Profits (P)	10	5	15	7	6	18	3
Weight (W)	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3

X	X_1	X_2	X_3	X_4	X_5	X_6	X_7

15 – 1 = 14

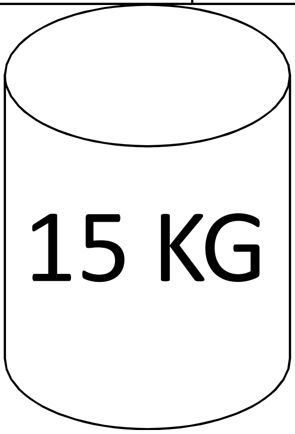
14 – 2 = 12



Knapsack Problem

Objects**(O)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Profits (P)	**10**	**5**	**15**	**7**	**6**	**18**	**3**
Weight (W)	**2**	**3**	**5**	**7**	**1**	**4**	**1**
P/W	**5**	**1.3**	**3**	**1**	**6**	**4.5**	**3**

X	**X_1**	**X_2**	**X_3**	**X_4**	**X_5**	**X_6**	**X_7**



15 – 1 = 14

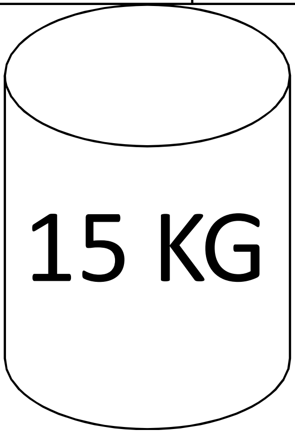
14 – 2 = 12

12 – 4 = 8

Knapsack Problem

Objects**(O)**	1	2	3	4	5	6	7
Profits (P)	**10**	**5**	**15**	**7**	**6**	**18**	**3**
Weight (W)	**2**	**3**	**5**	**7**	**1**	**4**	**1**
P/W	**5**	**1.3**	**3**	**1**	**6**	**4.5**	**3**

X	X_1	X_2	X_3	X_4	X_5	X_6	X_7



15 – 1 = 14

14 – 2 = 12

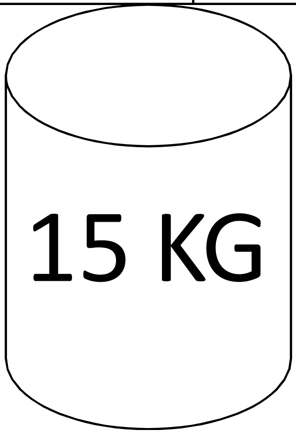
12 – 4 = 8

8 – 5 = 3

Knapsack Problem

Objects**(O)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Profits (P)	**10**	**5**	**15**	**7**	**6**	**18**	**3**
Weight (W)	**2**	**3**	**5**	**7**	**1**	**4**	**1**
P/W	**5**	**1.3**	**3**	**1**	**6**	**4.5**	**3**

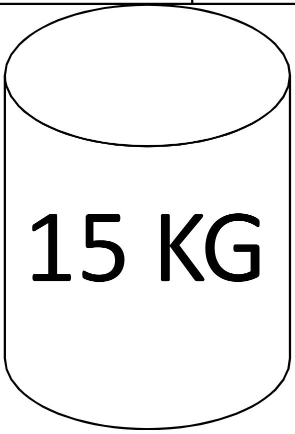
X	**X_1**	**X_2**	**X_3**	**X_4**	**X_5**	**X_6**	**X_7**



15 – 1 = 14
14 – 2 = 12
12 – 4 = 8
8 – 5 = 3
3 – 1 = 2

Knapsack Problem

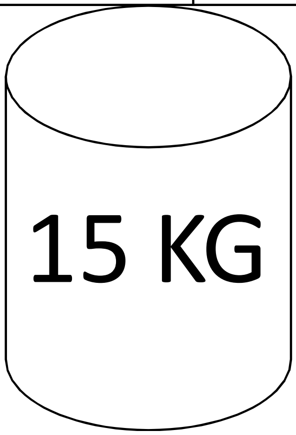
Objects**(O)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Profits (P)	**10**	**5**	**15**	**7**	**6**	**18**	**3**
Weight (W)	**2**	**3**	**5**	**7**	**1**	**4**	**1**
P/W	**5**	**1.3**	**3**	**1**	**6**	**4.5**	**3**
	↓	2 /3	↓		↓	↓	↓
X	**X_1**	**X_2**	**X_3**	**X_4**	**X_5**	**X_6**	**X_7**



15 – 1 = 14
14 – 2 = 12
12 – 4 = 8
8 – 5 = 3
3 – 1 = 2
2 – 2 = 0

Knapsack Problem

Objects**(O)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Profits (P)	**10**	**5**	**15**	**7**	**6**	**18**	**3**
Weight (W)	**2**	**3**	**5**	**7**	**1**	**4**	**1**
P/W	**5**	**1.3**	**3**	**1**	**6**	**4.5**	**3**
	↓	2 /3	↓	0	↓	↓	↓
X	**X_1**	**X_2**	**X_3**	**X_4**	**X_5**	**X_6**	**X_7**



15 – 1 = 14
14 – 2 = 12
12 – 4 = 8
8 – 5 = 3
3 – 1 = 2
2 – 2 = 0

Knapsack Problem

Objects**(O)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Profits (P)	**10**	**5**	**15**	**7**	**6**	**18**	**3**
Weight (W)	**2**	**3**	**5**	**7**	**1**	**4**	**1**
P/W	**5**	**1.3**	**3**	**1**	**6**	**4.5**	**3**
	↓	2 /3	↓	0	↓	↓	↓
X	**X_1**	**X_2**	**X_3**	**X_4**	**X_5**	**X_6**	**X_7**

Calculate Weight

$\sigma \; x_{i\,w_i=} \; 1\,x\,2 + 2/3\,x\,3 + 1\,x\,5 + 0\,x\,7 + 1\,x\,1 + 1\,x\,4 + 1\,x\,1$

$2 + 2 + 5 + 0 + 1 + 4 + 1 = 15$

Calculate Profit

⍰ $x_{i\,p_i=} \; 1\,X\,10 + \frac{2}{3x5} + 1x15 + 1x6 + 1x18 + 1x3$

$10 + 2x1.3 + 15 + 6 + 18 + 3 = 54.6$

Conclusion

- *Constraints*

σ x_i w_i $\leq$m where m = 15

- *Objective*

MAX � x_i $p_i =$

Analysis

- If the items are already sorted into decreasing order of v_i / $w_{i,}$ then the while-loop takes a time in ***O(n)***;
  Therefore, the total time including the sort is in ***O(n log n)*****.**
- If we keep the items in heap with largest v_i/w_i at the root. Then
  - creating the heap takes ***O(n)*** time
  - while-loop now takes ***O(log n)*** time (since heap property must be restored after the removal of root)

- Although this data structure does not alter the worst-case, it may be faster if only a small number of items are need to fill the knapsack.