Dynamic Programming: Knapsack

Alexander S. Kulikov

Steklov Institute of Mathematics at St. Petersburg
Russian Academy of Sciences

Algorithmic Toolbox
Data Structures and Algorithms

Outline

1. Problem Overview
2. Knapsack with Repetitions
3. Knapsack without Repetitions
4. Final Remarks

TV commercial placement

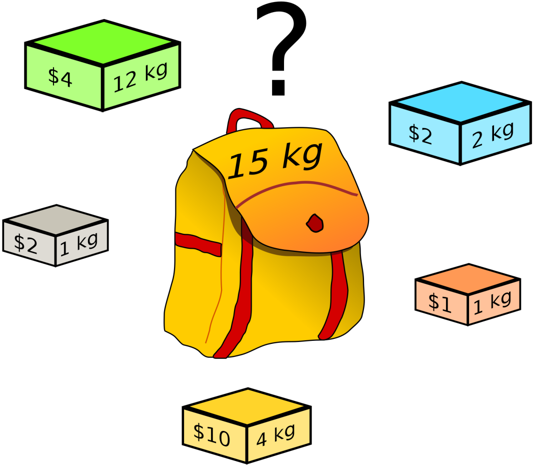
Select a set of TV commercials (each commercial has duration and cost) so that the total revenue is maximal while the total length does not exceed the length of the available time slot.

Optimizing data center performance

Purchase computers for a data center to achieve the maximal performance under limited budget.

Knapsack Problem

(knapsack is another word for backpack)



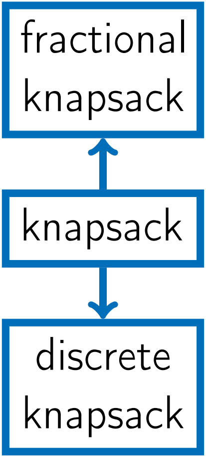
Goal

Maximize value ($) while limiting total weight (kg)

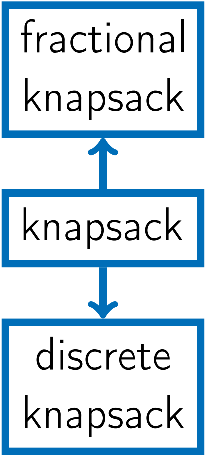
Problem Variations

knapsack

Problem Variations



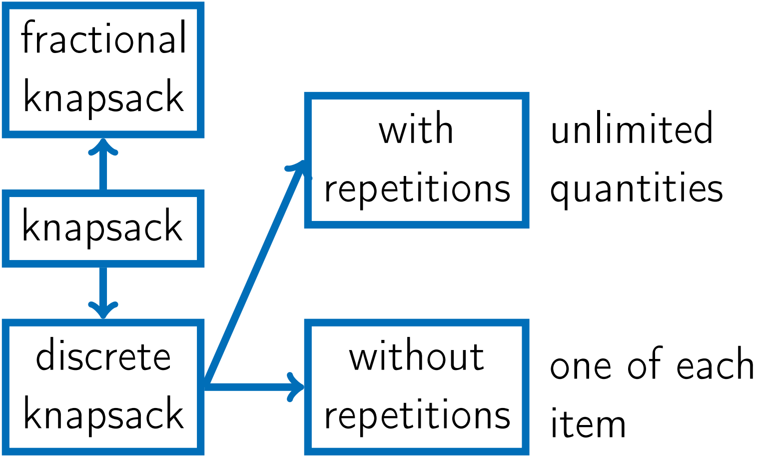
Problem Variations



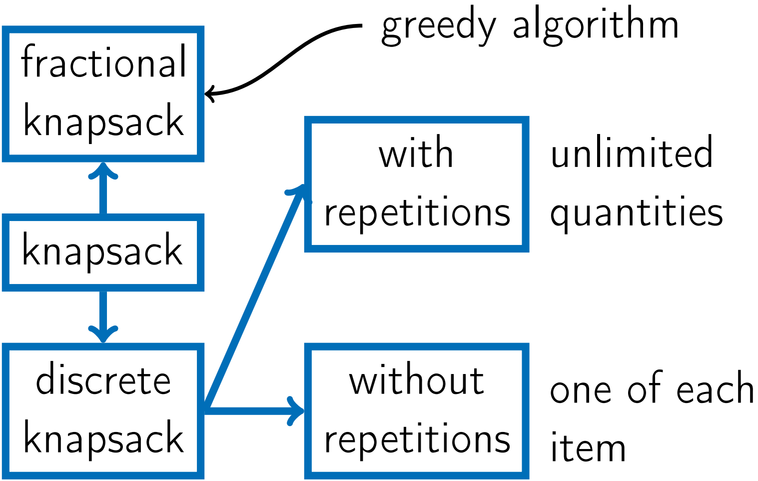
can take fractions of items

each item is either taken or not

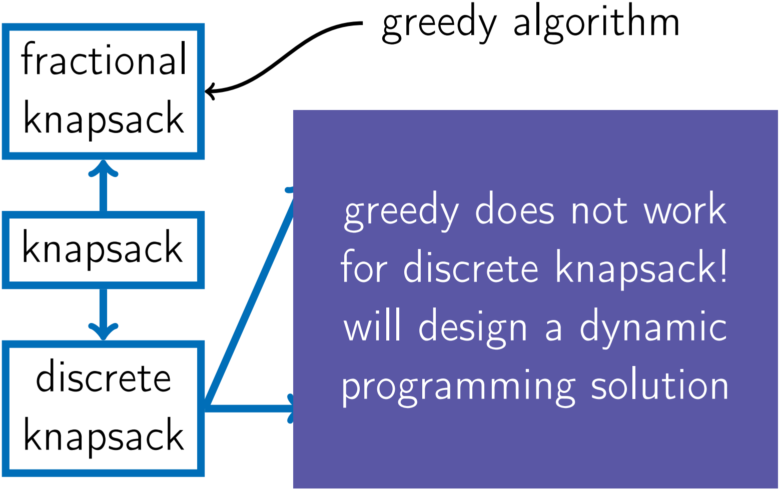
Problem Variations



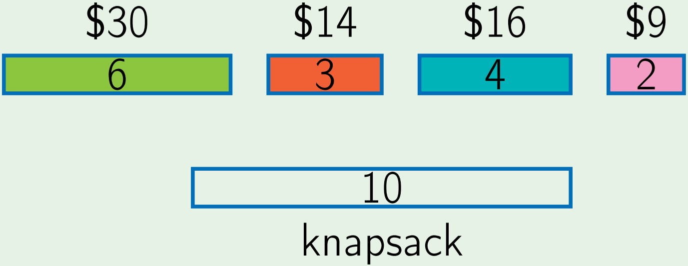
Problem Variations



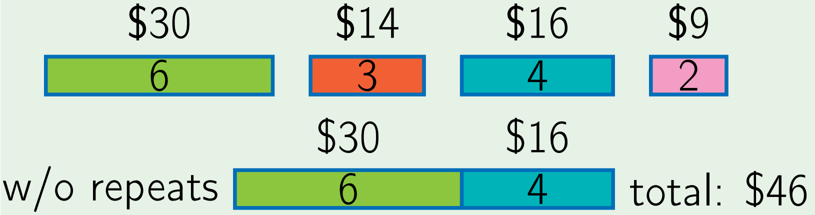
Problem Variations



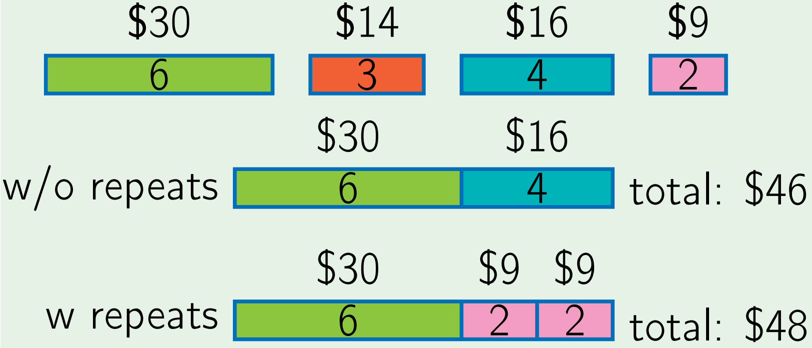
Example



Example

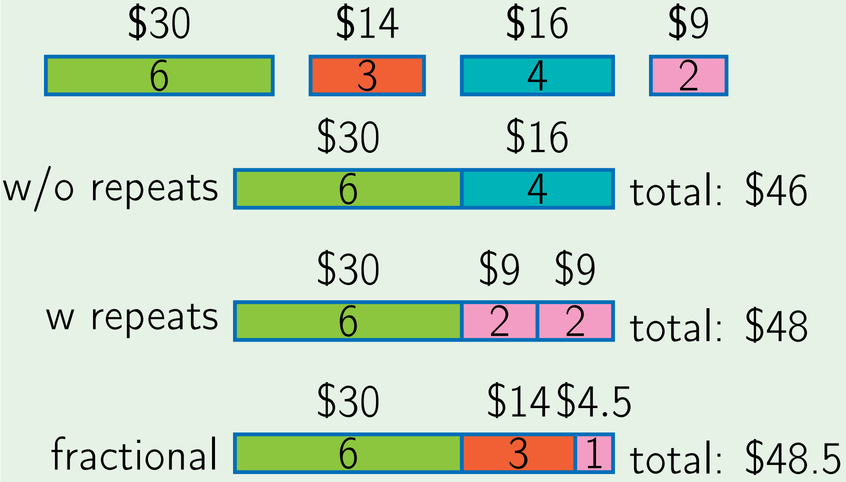


Example



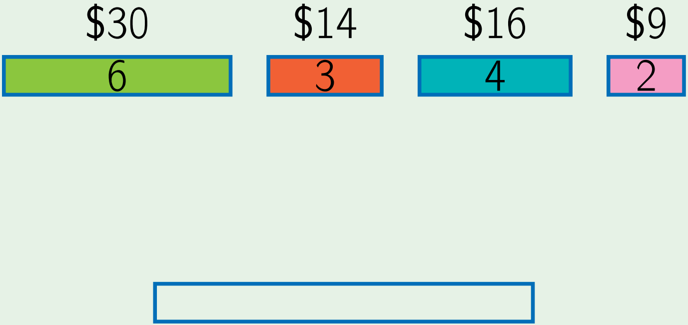
$30
$14
$16
$9
6
3
4
2
$30
$16
w/o repeats
6
4
total: $46
$30
$9
$9
w repeats
6
2
2
total: $48

Example


$30
$14
$16
$9
6
3
4
2
$30
$16
w/o repeats
6
4
total: $46
$30
$9
$9
w repeats
6
2
2
total: $48
$30
$14
$4.5
fractional
6
3
1
total: $48.5

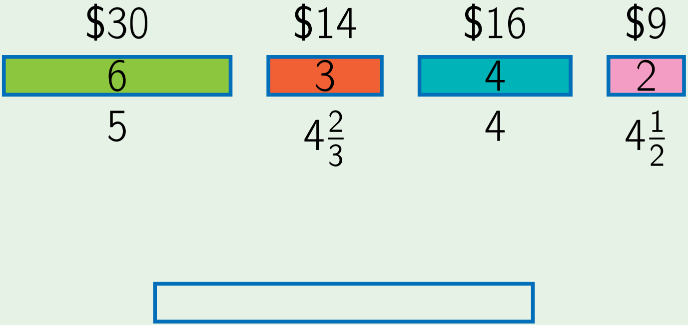
Why does greedy fail for the discrete knapsack?

Example



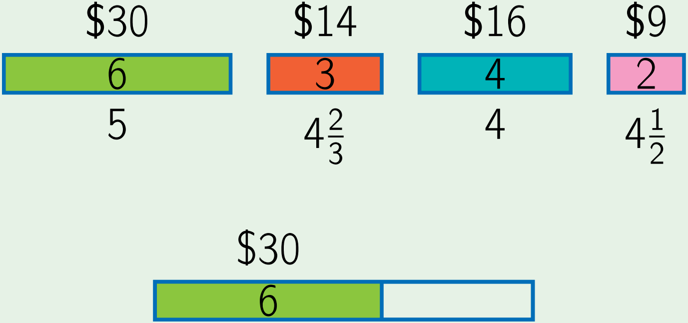
Why does greedy fail for the discrete knapsack?

Example



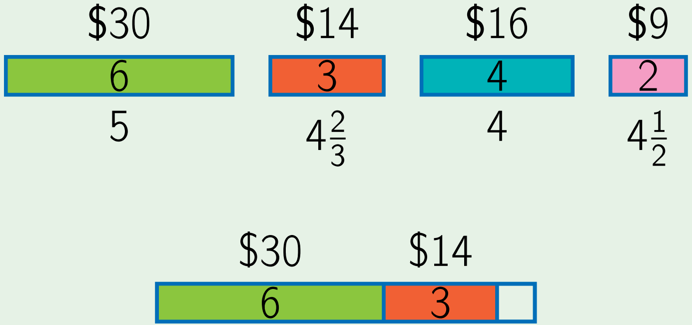
Why does greedy fail for the discrete knapsack?

Example



Why does greedy fail for the discrete knapsack?

Example



Why does greedy fail for the discrete knapsack?

Example

\$30	\$14	\$16	\$9
6	3	4	2
5	$4\frac{2}{3}$	4	$4\frac{1}{2}$

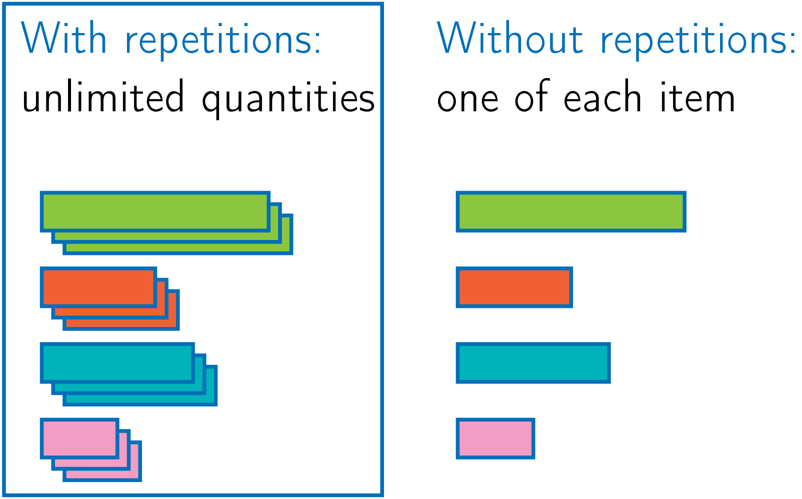
taking an element of maximum value per unit of weight is not safe!

Outline

1. Problem Overview
2. Knapsack with Repetitions
3. Knapsack without Repetitions
4. Final Remarks

With repetitions:
unlimited quantities

Without repetitions:
one of each item


With repetitions:
unlimited quantities
Without repetitions:
one of each item

Knapsack with repetitions problem

Input: Weights $w_1, \dots, w_n$ and values $v_1, \dots, v_n$ of n items; total weight W (v_i's, w_i's, and W are non-negative integers).

Output: The maximum value of items whose weight does not exceed W. Each item can be used any number of times.

Subproblems

- Consider an optimal solution and an item in it:

Subproblems

- Consider an optimal solution and an item in it:

- If we take this item out then we get an optimal solution for a knapsack of total weight $W - w_i$.

Subproblems

Let $value(w)$ be the maximum value of knapsack of weight w.

Subproblems

Let $value(w)$ be the maximum value of knapsack of weight w.

$$value(w) = \max_{i\colon w_i \le w} \{value(w - w_i) + v_i\}$$

Knapsack(W)

```
value(0) ← 0
for w from 1 to W:
    value(w) ← 0
    for i from 1 to n:
        if w_i ≤ w:
            val ← value(w − w_i) + v_i
            if val > value(w):
                value(w) ← val
return value(W)
```

Example: $W = 10$

\$30	\$14	\$16	\$9
6	3	4	2

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0

Example: $W = 10$

\$30	\$14	\$16	\$9
6	3	4	2

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0

Example: $W = 10$

$30	$14	$16	$9
6	3	4	2

0	1	2	3	4	5	6	7	8	9	10
0	0	9	0	0	0	0	0	0	0	0

Example: $W = 10$

\$30	\$14	\$16	\$9
6	3	4	2

0	1	2	3	4	5	6	7	8	9	10
0	0	9	0	0	0	0	0	0	0	0

Example: $W = 10$

\$30	\$14	\$16	\$9
6	3	4	2

0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	0	0	0	0	0	0	0

Example: $W = 10$

\$30	\$14	\$16	\$9
6	3	4	2

0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	0	0	0	0	0	0	0

Example: $W = 10$

\$30	\$14	\$16	\$9
6	3	4	2

0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	18	0	0	0	0	0	0

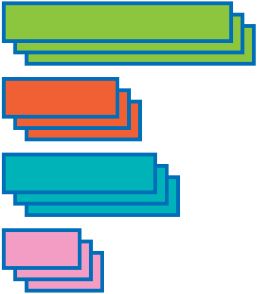
Example: $W = 10$

\$30	\$14	\$16	\$9
6	3	4	2

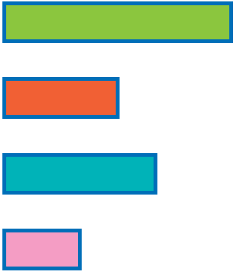
0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	18	23	30	32	39	44	48

Outline

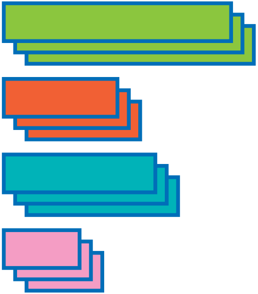
With repetitions:
unlimited quantities



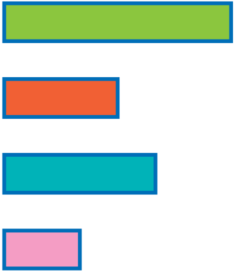
Without repetitions:
one of each item



With repetitions:
unlimited quantities



Without repetitions:
one of each item



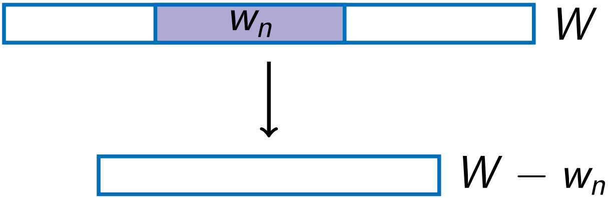
Knapsack without repetitions problem

Input: Weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$ of n items; total weight W (v_i's, w_i's, and W are non-negative integers).

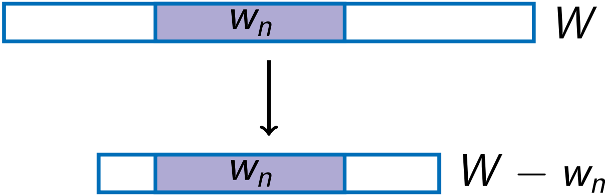
Output: The maximum value of items whose weight does not exceed W. Each item can be used at most once.

Same Subproblems?

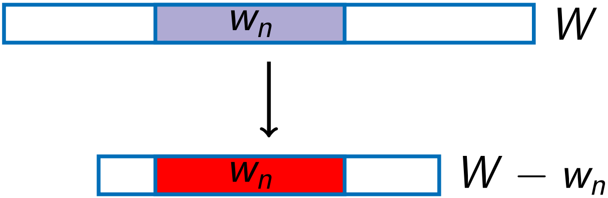
Same Subproblems?



Same Subproblems?



Same Subproblems?



Subproblems

- If the n-th item is taken into an optimal solution:

  then what is left is an optimal solution for a knapsack of total weight $W - w_n$ using items $1, 2, \ldots, n-1$.

Subproblems

- If the n-th item is taken into an optimal solution:

  then what is left is an optimal solution for a knapsack of total weight $W - w_n$ using items $1, 2, \ldots, n-1$.

- If the n-th item is not used, then the whole knapsack must be filled in optimally with items $1, 2, \ldots, n-1$.

Subproblems

For $0 \leq w \leq W$ and $0 \leq i \leq n$, $value(w, i)$ is the maximum value achievable using a knapsack of weight w and items $1, \ldots, i$.

Subproblems

For $0 \leq w \leq W$ and $0 \leq i \leq n$, $value(w, i)$ is the maximum value achievable using a knapsack of weight w and items $1, \ldots, i$.

The i-th item is either used or not: $value(w, i)$ is equal to

$$\max\{value(w-w_i, i-1)+v_i, value(w, i-1)\}$$

$\texttt{Knapsack}(W)$

```
initialize all value(0, j) ← 0
initialize all value(w, 0) ← 0
for i from 1 to n:
  for w from 1 to W:
    value(w, i) ← value(w, i − 1)
    if w_i ≤ w:
      val ← value(w − w_i, i − 1) + v_i
      if value(w, i) < val
        value(w, i) ← val
return value(W, n)
```

Example: reconstructing a solution

\$30	\$14	\$16	\$9
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal solution:

1	2	3	4

Example: reconstructing a solution

\$30	\$14	\$16	\$9
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal solution:

1	2	3	4

Example: reconstructing a solution

$30	$14	$16	$9
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal solution:

1	2	3	4
			0

Example: reconstructing a solution

\$30	\$14	\$16	\$9
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal solution:

1	2	3	4
			0

Example: reconstructing a solution

$30	$14	$16	$9
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal solution:

1	2	3	4
		1	0

Example: reconstructing a solution

\$30	\$14	\$16	\$9
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal solution:

1	2	3	4
		1	0

Example: reconstructing a solution

\$30	\$14	\$16	\$9
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal solution:

1	2	3	4
	0	1	0

Example: reconstructing a solution

$30	$14	$16	$9
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal solution:

1	2	3	4
	0	1	0

Example: reconstructing a solution

\$30	\$14	\$16	\$9
6	3	4	2

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal solution:

1	2	3	4
1	0	1	0

Outline

Memoization

Knapsack(w)

```
if w is in hash table:
  return value(w)
value(w) ← 0
for i from 1 to n:
  if w_i ≤ w:
    val ← Knapsack(w − w_i) + v_i
    if val > value(w):
      value(w) ← val
insert value(w) into hash table with key w
return value(w)
```

What Is Faster?

- If all subproblems must be solved then an iterative algorithm is usually faster since it has no recursion overhead.

What Is Faster?

- If all subproblems must be solved then an iterative algorithm is usually faster since it has no recursion overhead.
- There are cases however when one does not need to solve all subproblems: assume that W and all w_i's are multiples of 100; then $value(w)$ is not needed if w is not divisible by 100.

Running Time

- The running time $O(nW)$ is not polynomial since the input size is proportional to $\log W$, but not W.

Running Time

- The running time $O(nW)$ is not polynomial since the input size is proportional to $\log W$, but not W.
- In other words, the running time is $O(n2^{\log W})$.

Running Time

- The running time $O(nW)$ is not polynomial since the input size is proportional to $\log W$, but not W.
- In other words, the running time is $O(n2^{\log W})$.
- E.g., for

$$W = 71\,345\,970\,345\,617\,824\,751$$

(twenty digits only!) the algorithm needs roughly 10^{20} basic operations.

Running Time

- The running time $O(nW)$ is not polynomial since the input size is p
- I
  O
- E

later, we'll learn why
solving this problem
in polynomial time costs \$1M!

(twenty digits only!) the algorithm needs roughly 10^{20} basic operations.