Design and Analysis of Algorithms

04-03
Dynamic Programming

Knapsack

Imran Ihsan
Assistant Professor, Department of Computer Science
Air University, Islamabad, Pakistan
www.imranihsan.com

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)

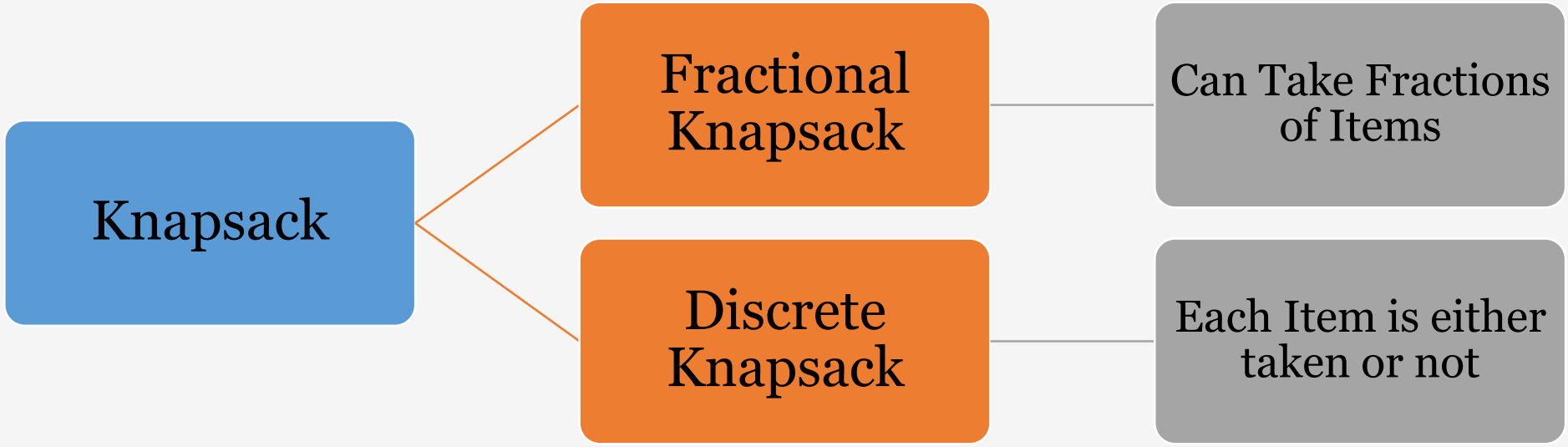
Goal

Maximize value ($)

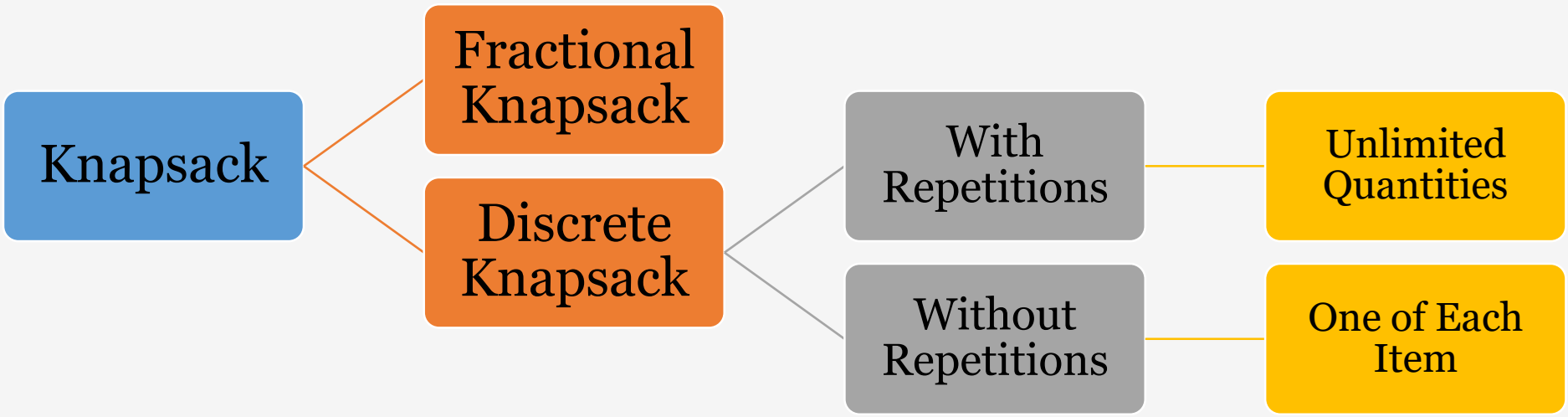
while limiting

Total weight (kg)

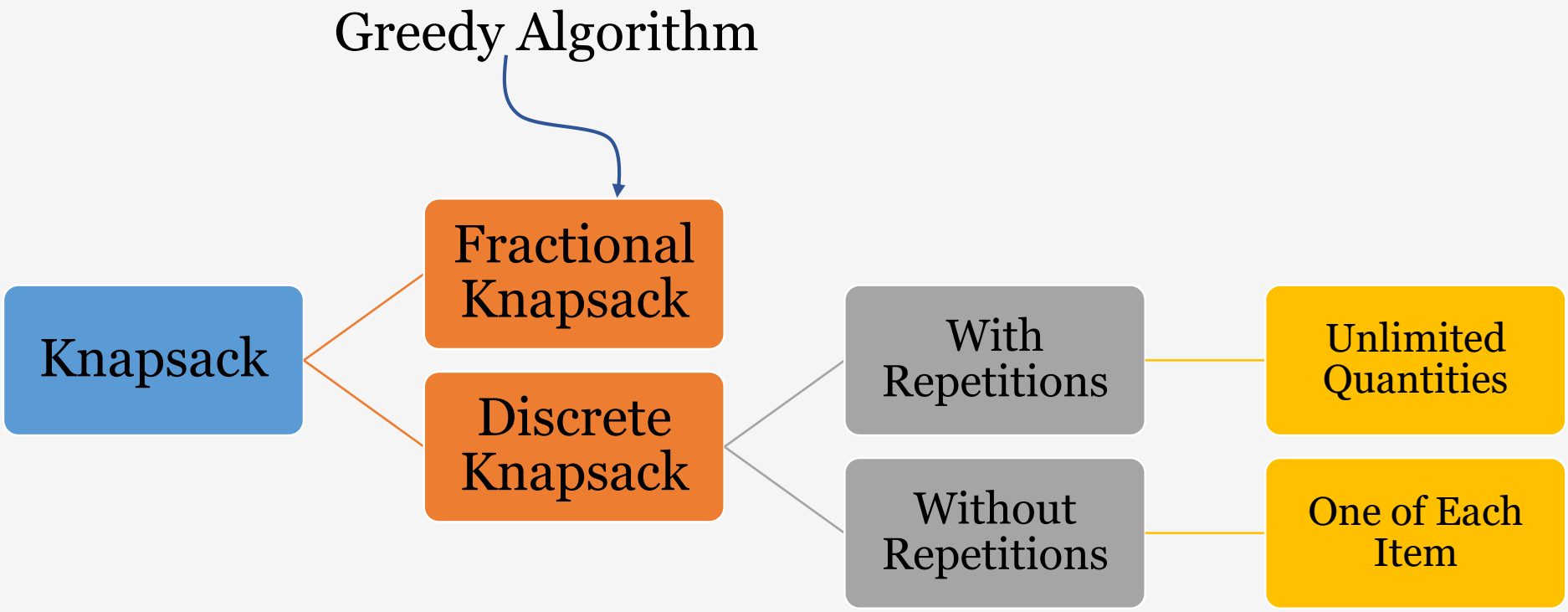
Problem Variations



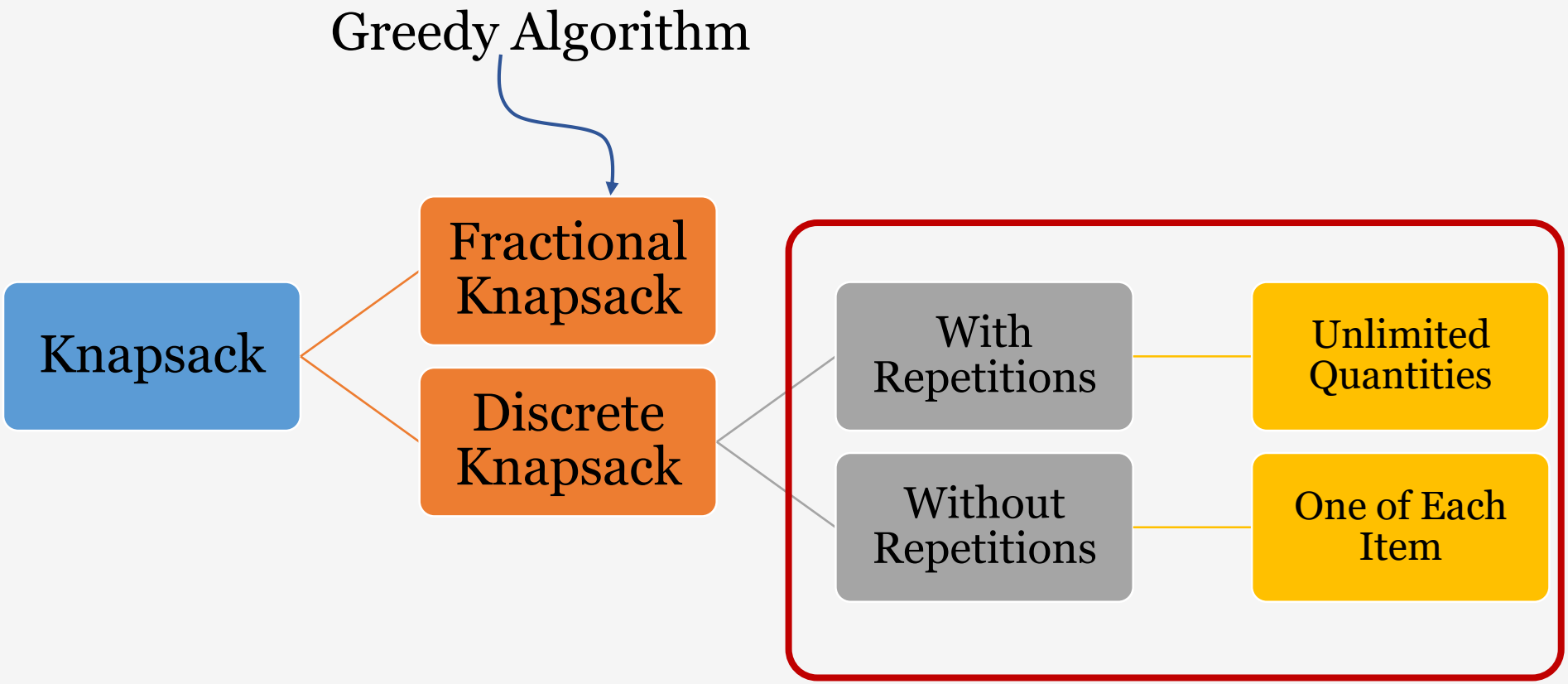
Problem Variations



Problem Variations

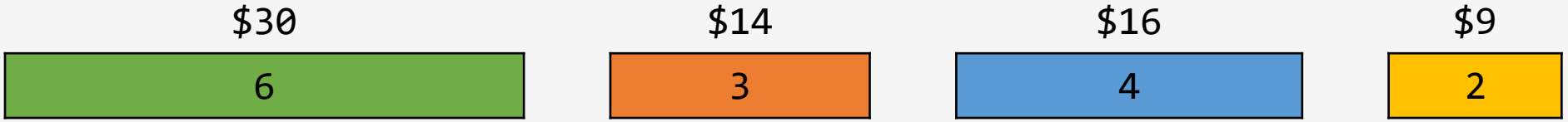


Problem Variations



Greedy Algorithm does not work for Discrete Knapsack.
Need to design a Dynamic Programming Solution

Example



w/o Repeats

Total: $46

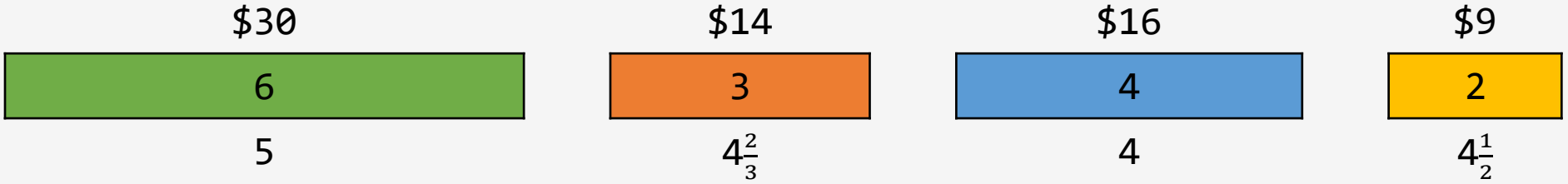
with Repeats

Total: $48

Fractional

Total: $48.5

Example



w/o Repeats

Total: $46

with Repeats

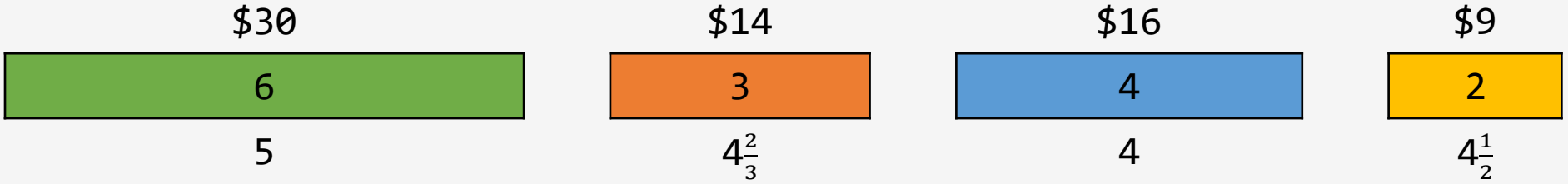
Total: $48

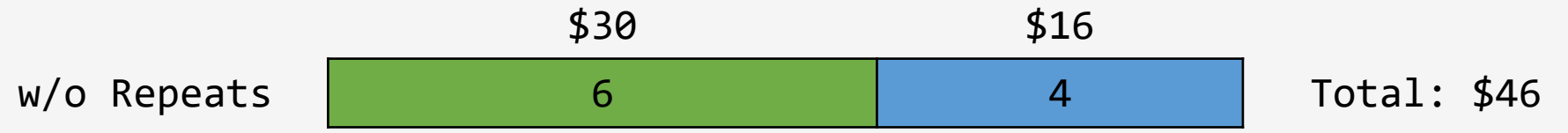
Fractional

Total: $48.5

Why does greedy fail for the discrete knapsack?

Example





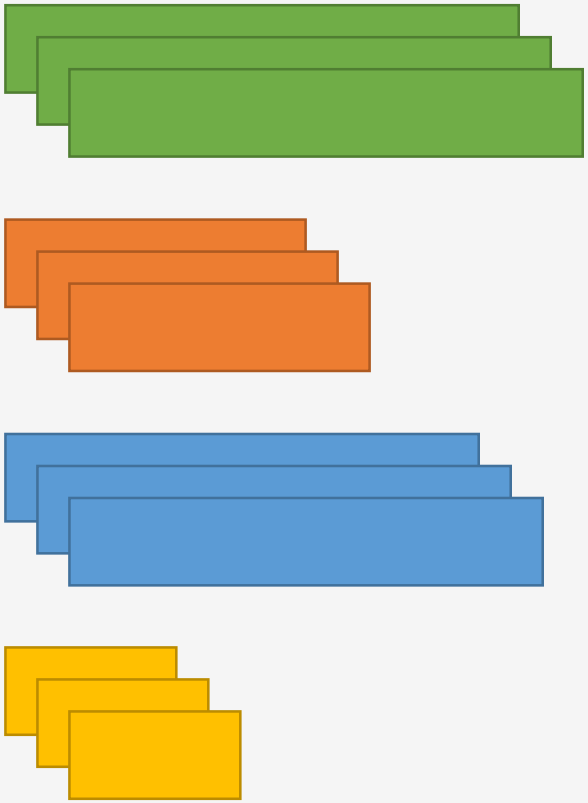
Why does greedy fail for the discrete knapsack?

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

Discrete Knapsack

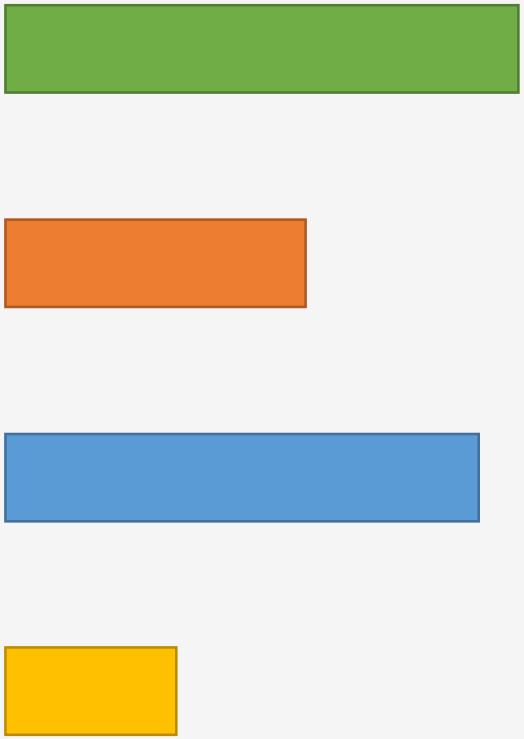
With repetitions:

unlimited quantities



Without repetitions:

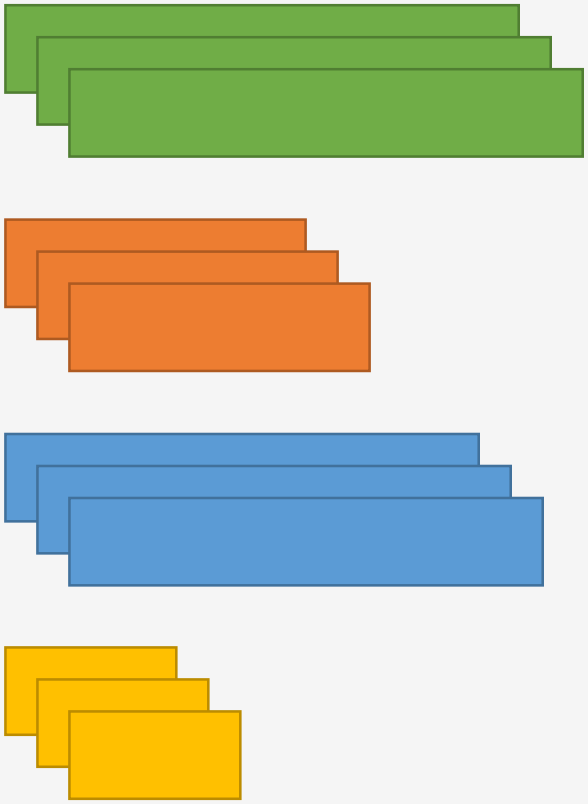
one of each item



Discrete Knapsack

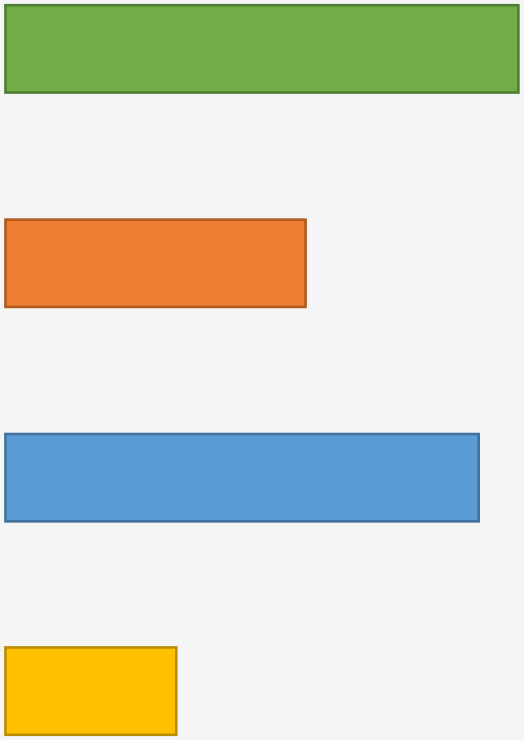
With repetitions:

unlimited quantities



Without repetitions:

one of each item



Knapsack with 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 any number of times.

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

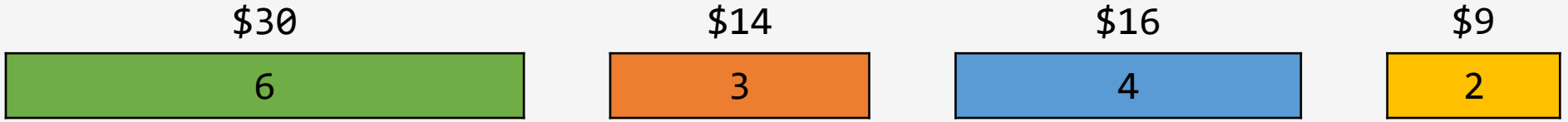
Let value(w) be the maximum value of knapsack of weight w.

$$value(w) = \max_{i:\, w_i \leq w} \{value(w - w_i) + v_i\}$$

Knapsack with Repetitions Problem

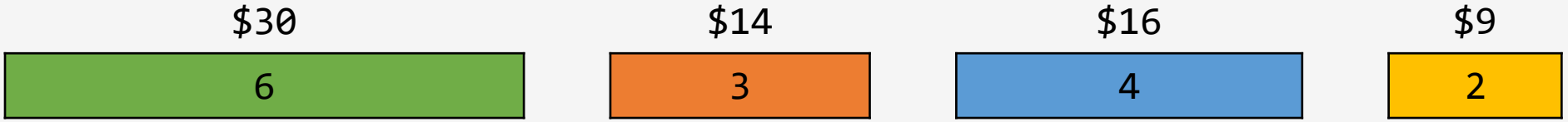
```
Knapsack(W)
    value(0) ← 0
    for w from 1 to W:
        value(w) ← 0
         for i from 1 to n:
             if w_i ≤ w:
               val ← value(w – w_i ) + v_i
               if val > value(w):
                  value(w) ← val
    return value(W)
```

Example: W = 10



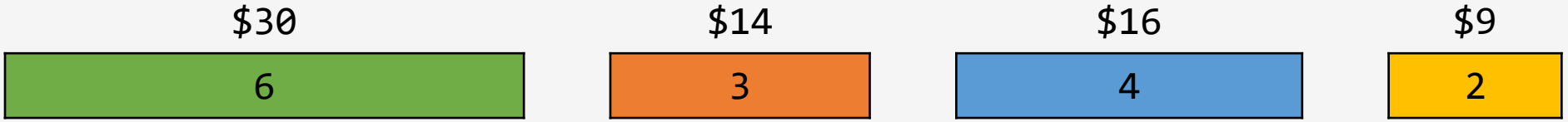
0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0

Example: W = 10



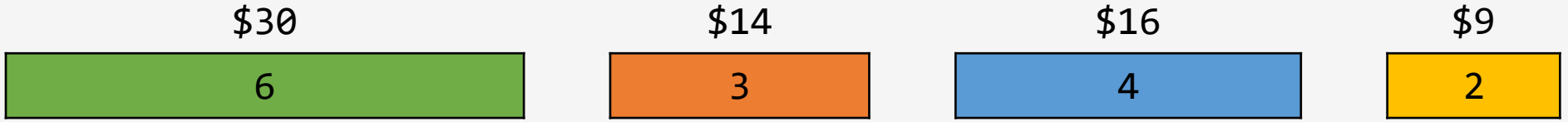
0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0

Example: W = 10



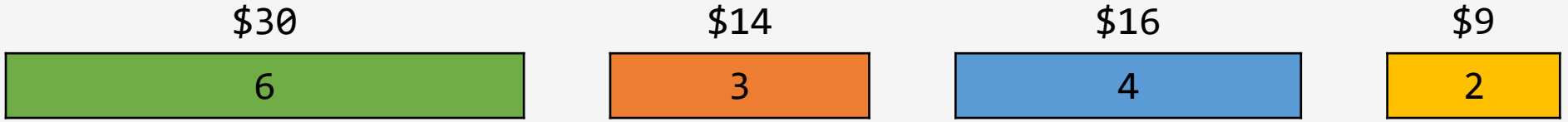
0	1	2	3	4	5	6	7	8	9	10
0	0	9	0	0	0	0	0	0	0	0

Example: W = 10



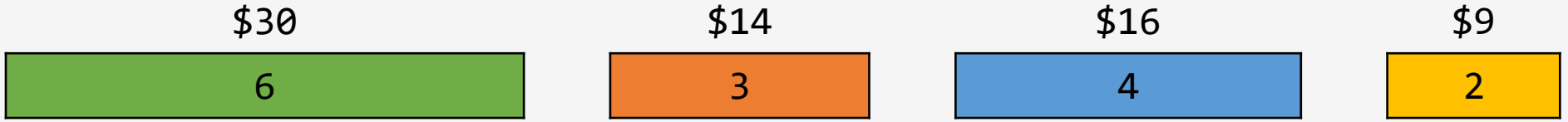
0	1	2	3	4	5	6	7	8	9	10
0	0	9	0	0	0	0	0	0	0	0

Example: W = 10



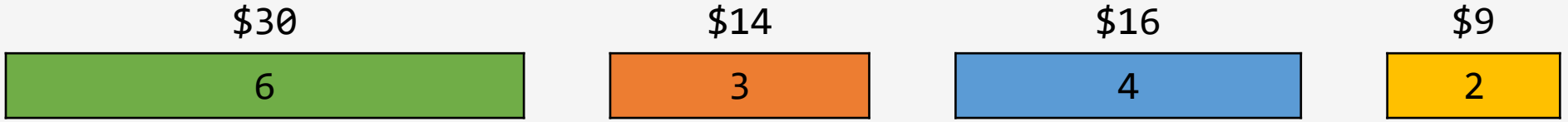
0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	0	0	0	0	0	0	0

Example: W = 10



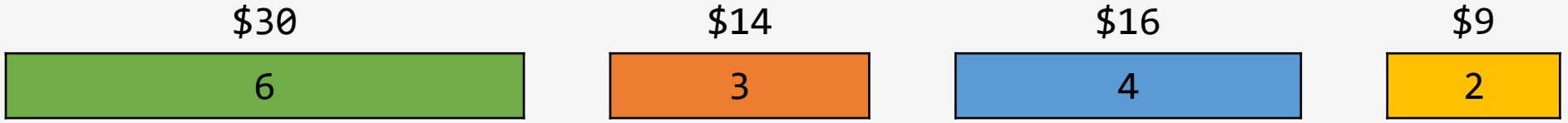
0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	0	0	0	0	0	0	0

Example: W = 10



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

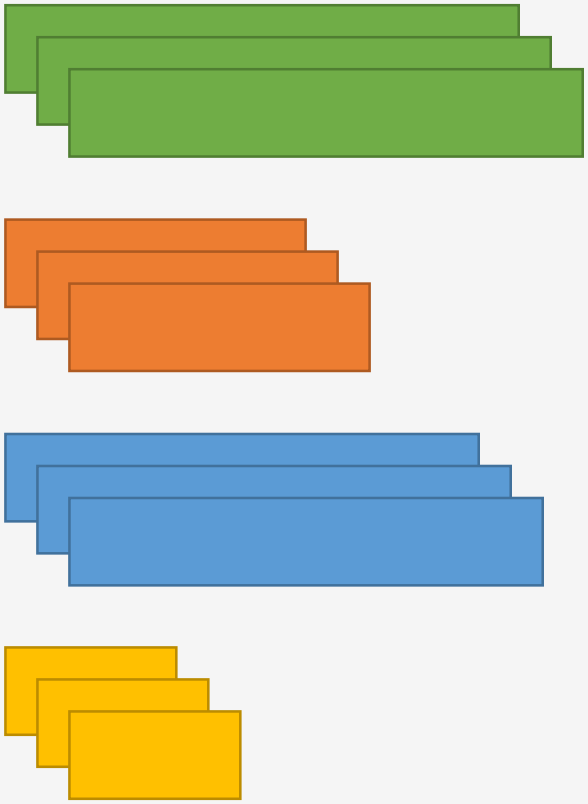


0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	18	23	30	32	39	44	48

Discrete Knapsack

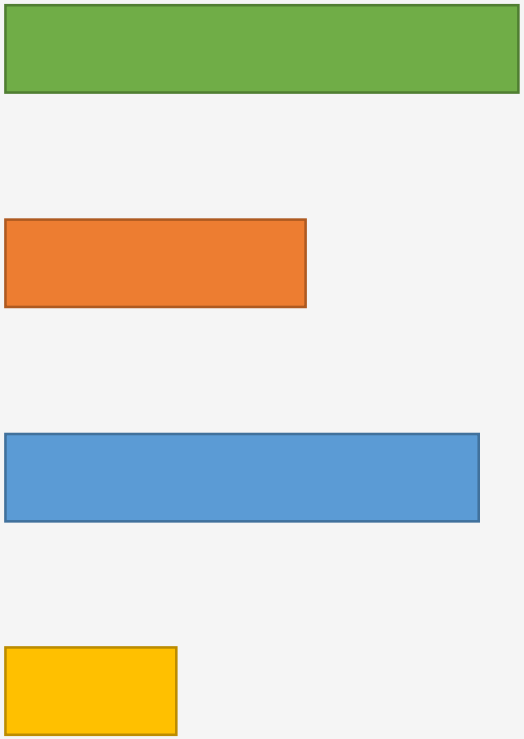
With repetitions:

unlimited quantities



Without repetitions:

one of each item



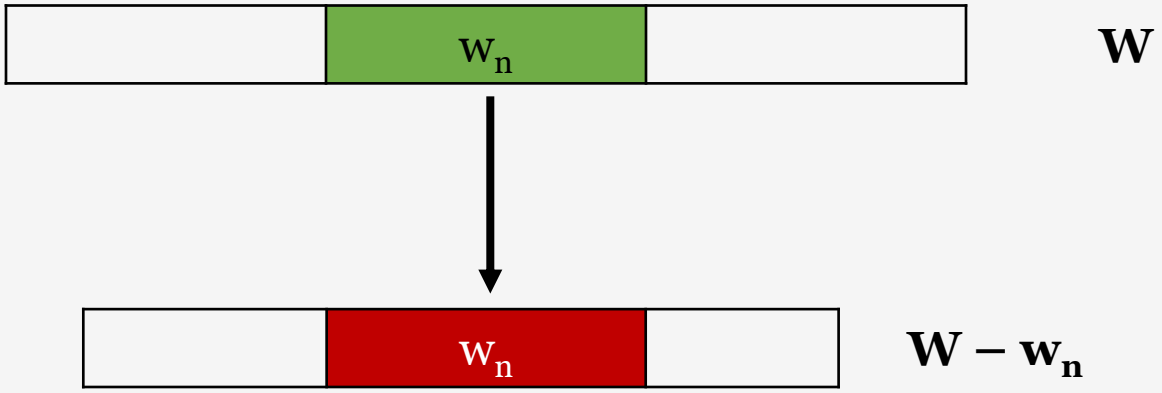
Knapsack without Repetitions Problem

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?



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 ≤ w ≤ W and 0 ≤ i ≤ n, value(w, i) is the maximum value achievable using a knapsack of weight w and items 1, . . . , i .

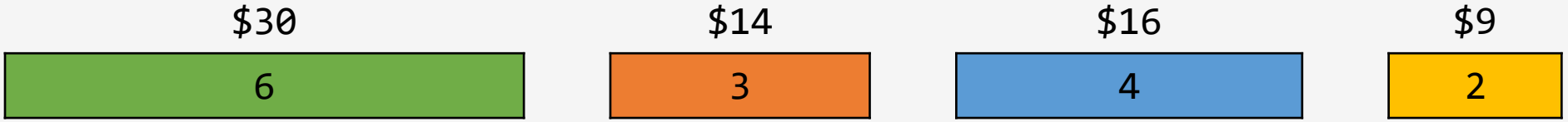
The i -th item is either used or not: value(w, i) is equal to.

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

Knapsack without Repetitions Problem

```
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

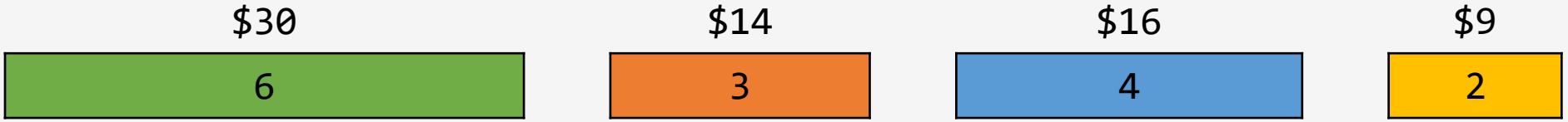


	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal Solution

1	2	3	4

Example: Reconstructing a Solution

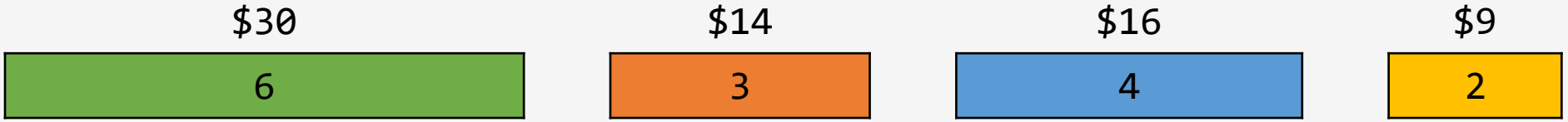


	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal Solution

1	2	3	4

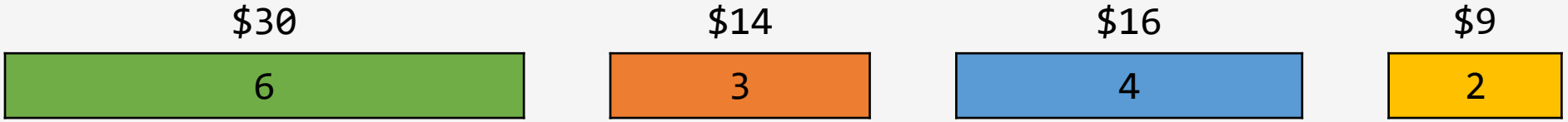
Example: Reconstructing a Solution



	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

	1	2	3	4
Optimal Solution				0

Example: Reconstructing a Solution

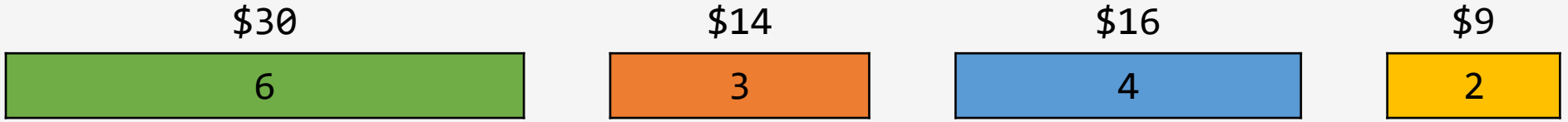


	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	30	30	30	30	30
2	0	0	0	14	14	14	30	30	30	44	44
3	0	0	0	14	16	16	30	30	30	44	46
4	0	0	9	14	16	23	30	30	39	44	46

Optimal Solution

1	2	3	4
			0

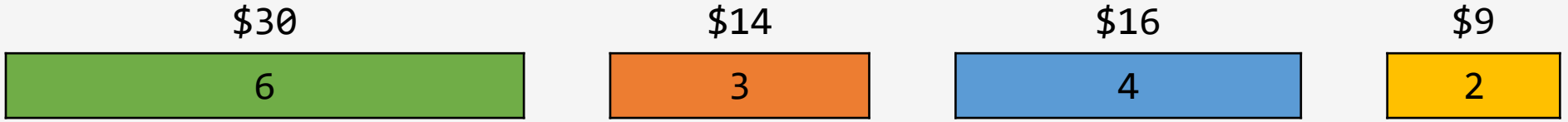
Example: Reconstructing a Solution



	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

	1	2	3	4
Optimal Solution			1	0

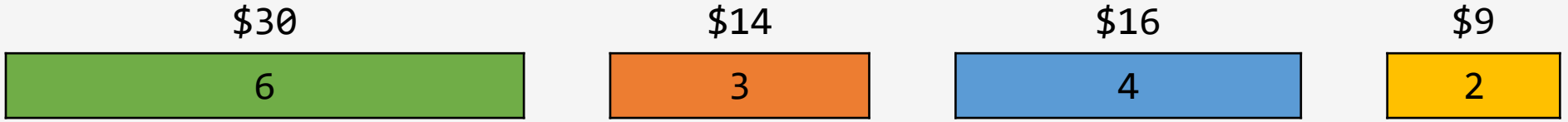
Example: Reconstructing a Solution



	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

	1	2	3	4
Optimal Solution			1	0

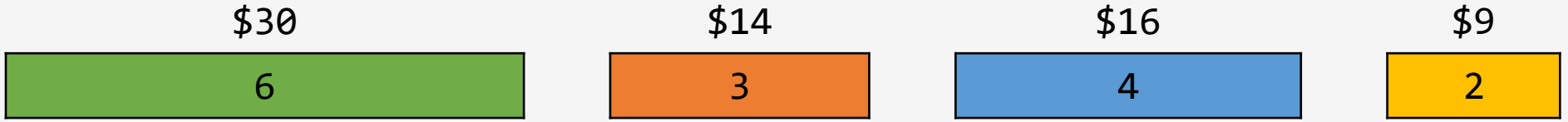
Example: Reconstructing a Solution



	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

	1	2	3	4
Optimal Solution		0	1	0

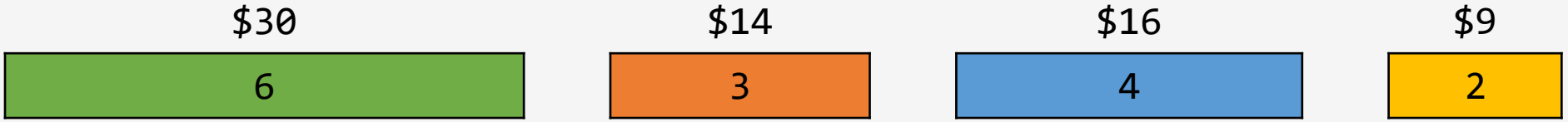
Example: Reconstructing a Solution



	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

	1	2	3	4
Optimal Solution		0	1	0

Example: Reconstructing a Solution



	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

	1	2	3	4
Optimal Solution	1	0	1	0