Greedy Algorithms: Maximizing Loot

Michael Levin

Department of Computer Science and Engineering
University of California, San Diego

Outline

Maximizing Loot

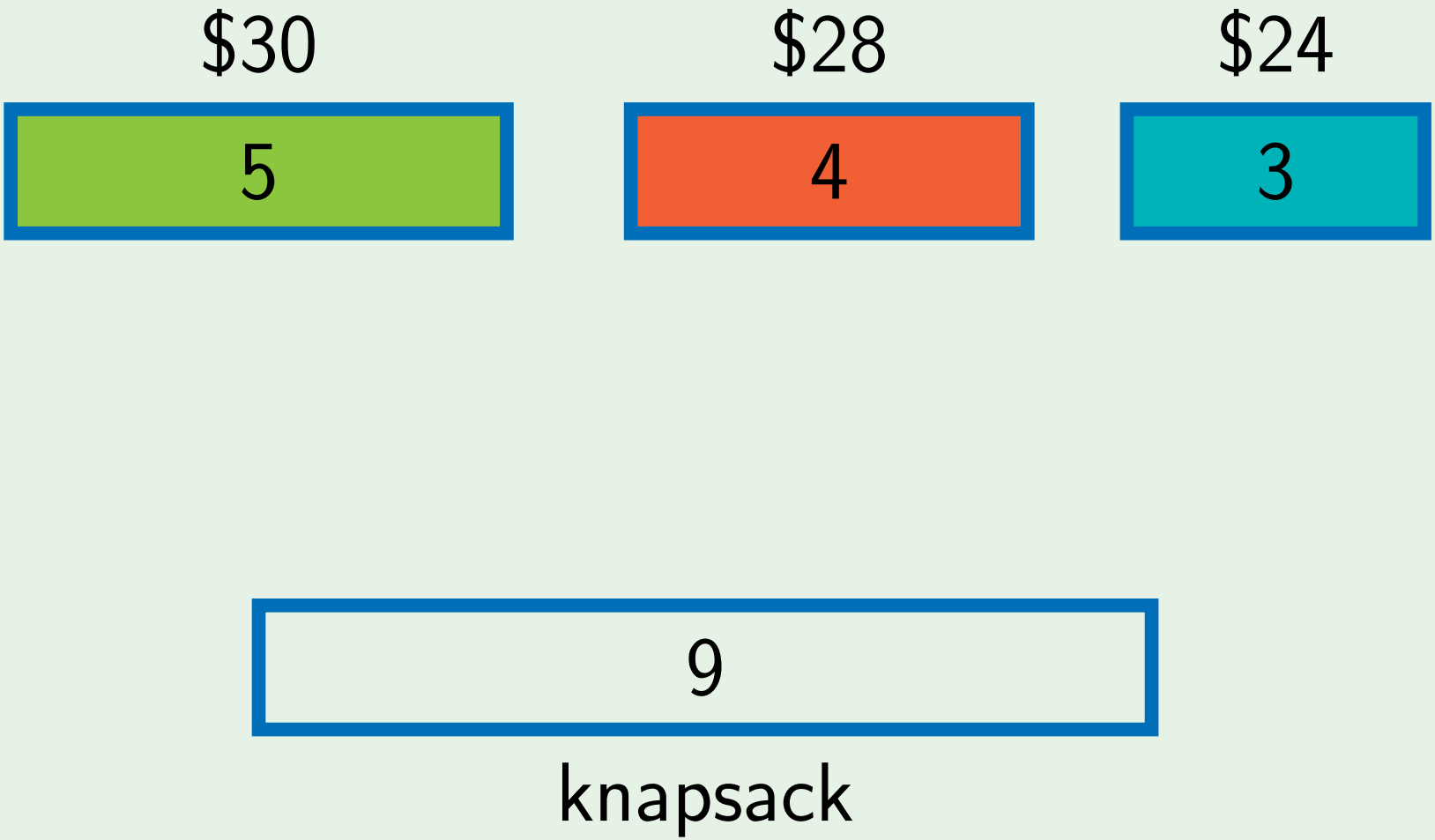
Maximizing Loot

Fractional knapsack

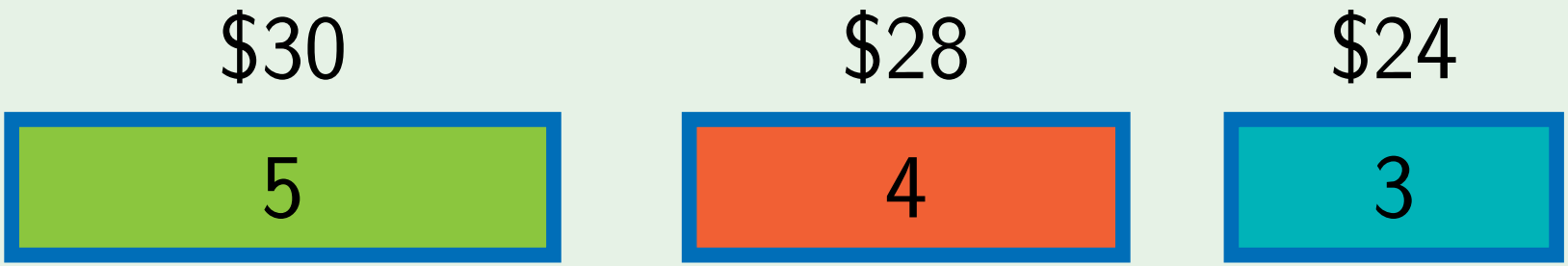
Input: Weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$ of n items; capacity W.

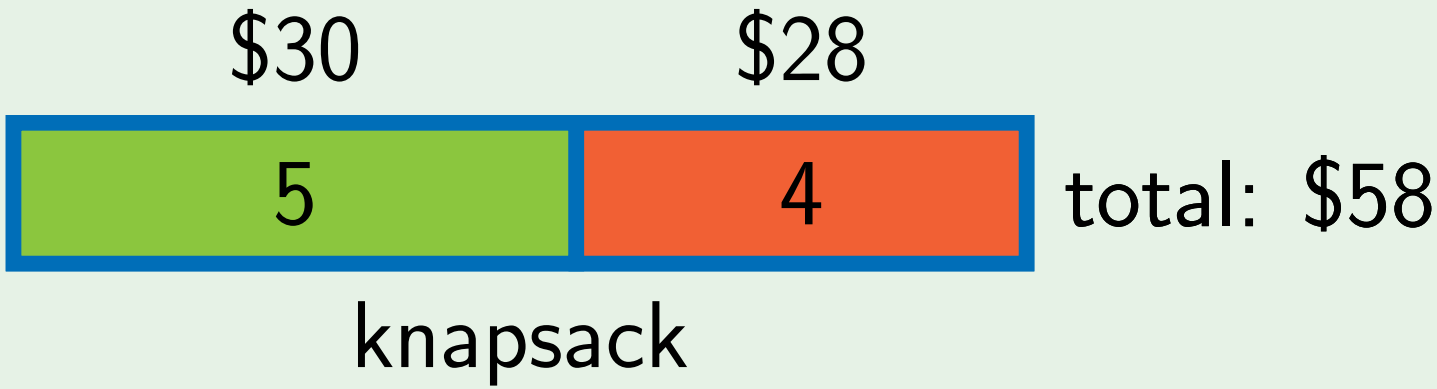
Output: The maximum total value of fractions of items that fit into a knapsack of capacity W.

Example

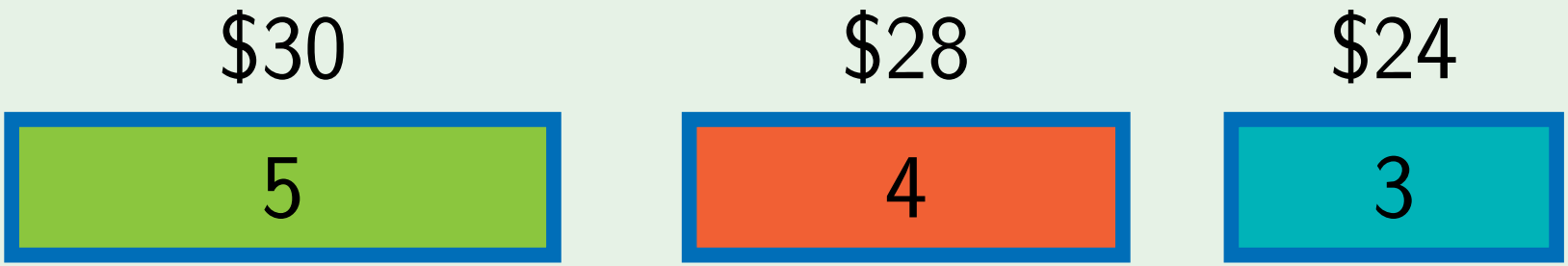


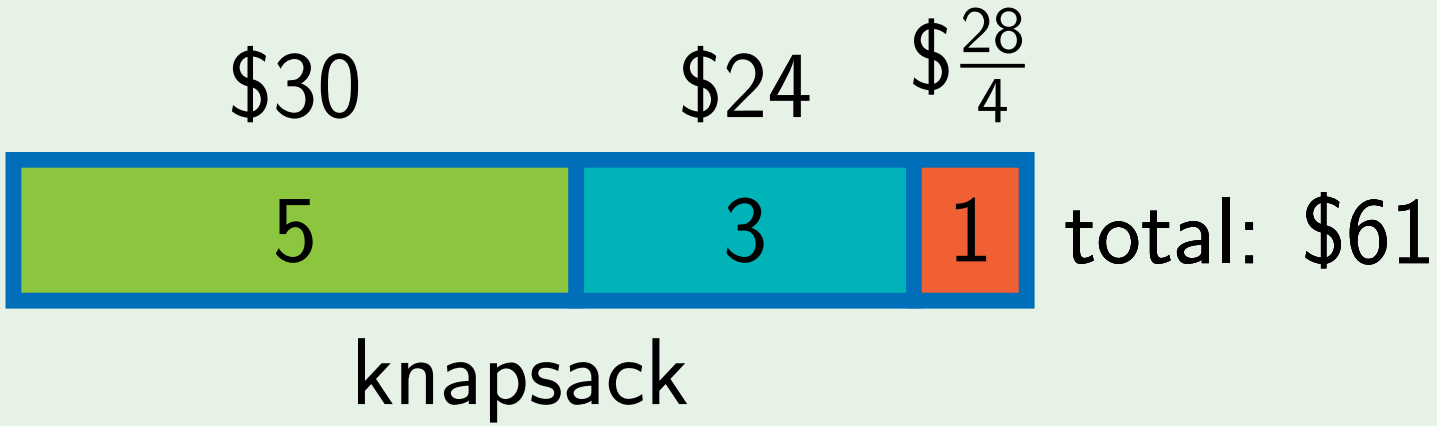
Example



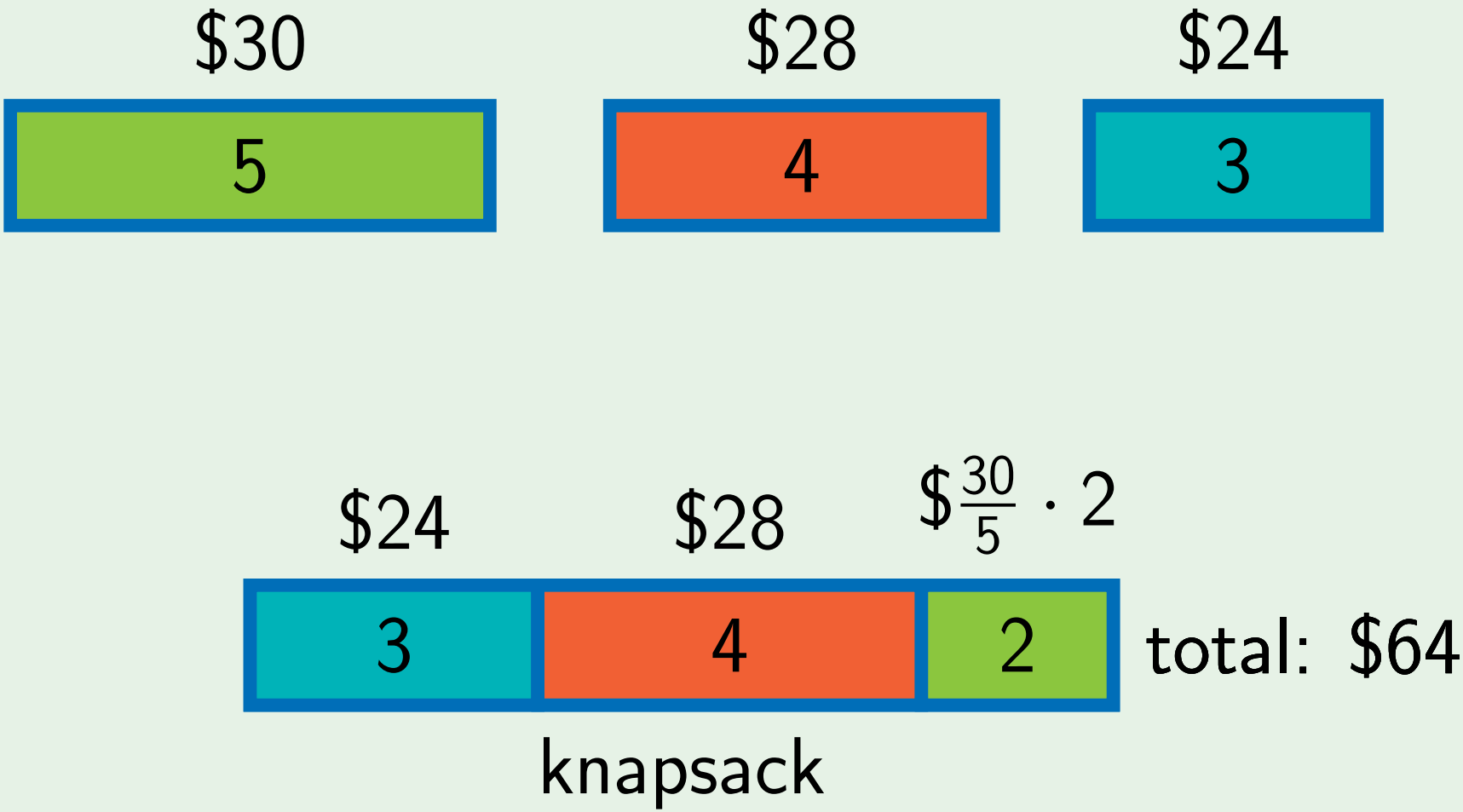


Example

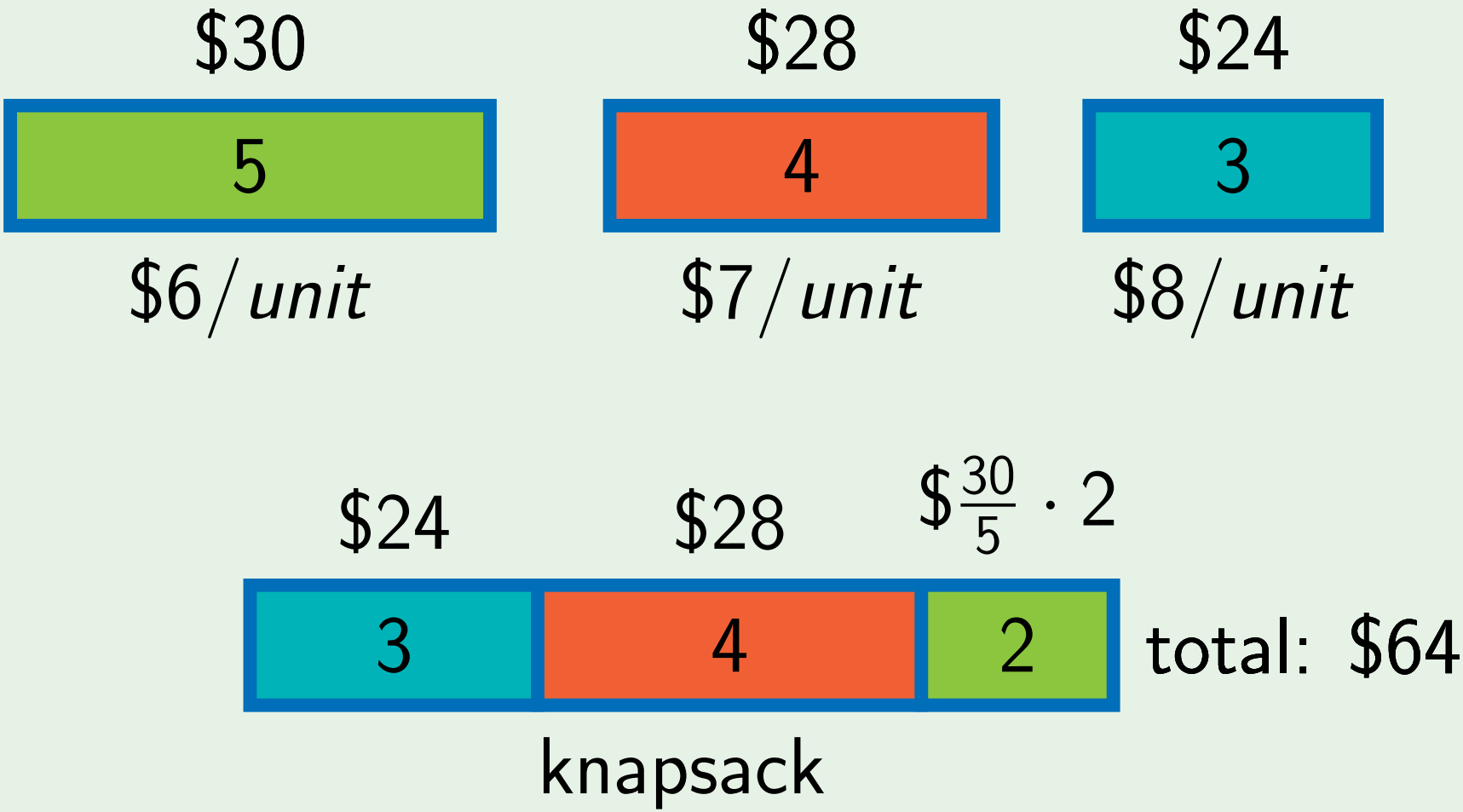




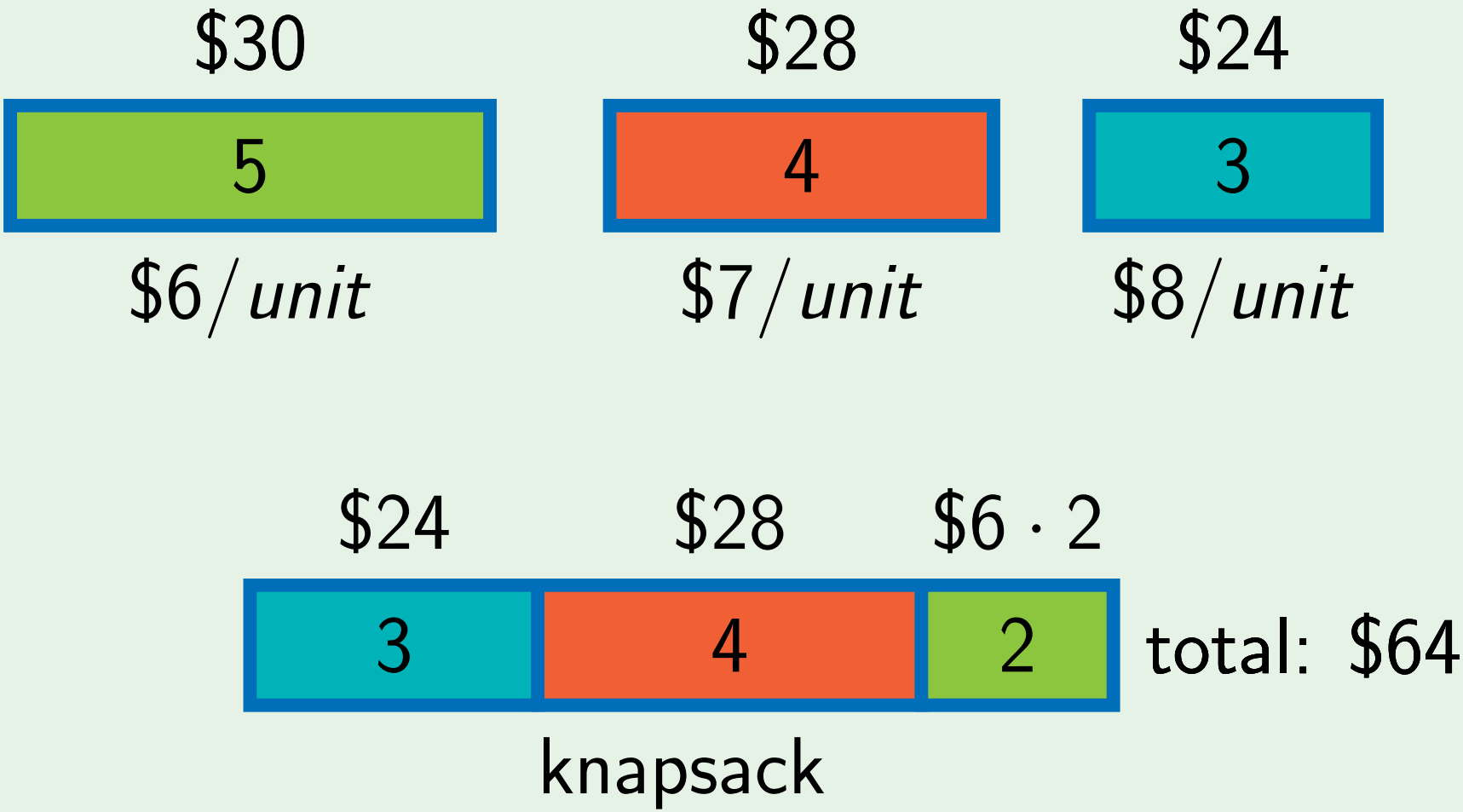
Example



Example


$30
5
$6/unit
$28
4
$7/unit
$24
3
$8/unit
$24
$28
$\frac{30}{5} \cdot 2$
3
4
2
total: $64
knapsack

Example

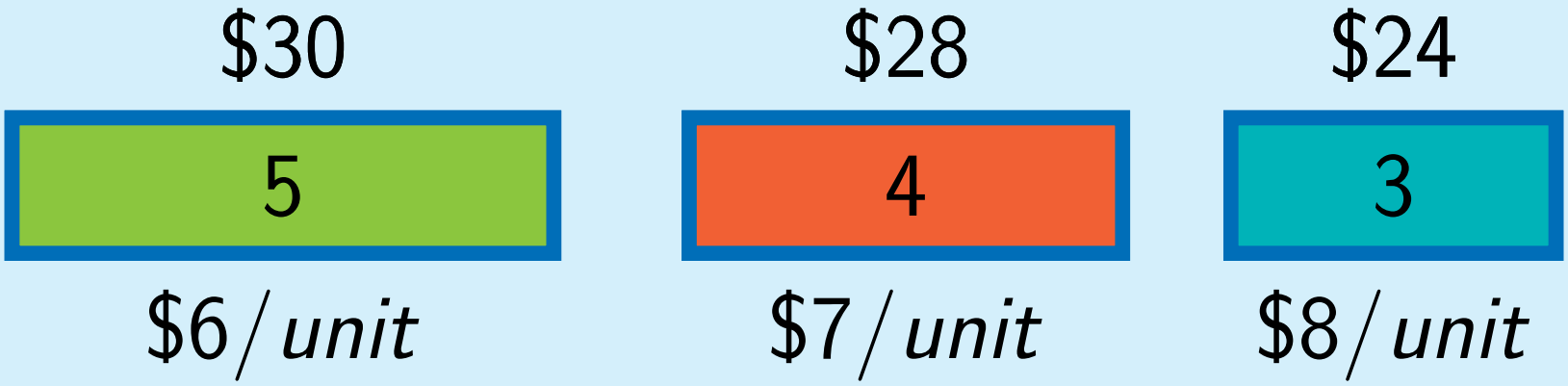


Safe choice

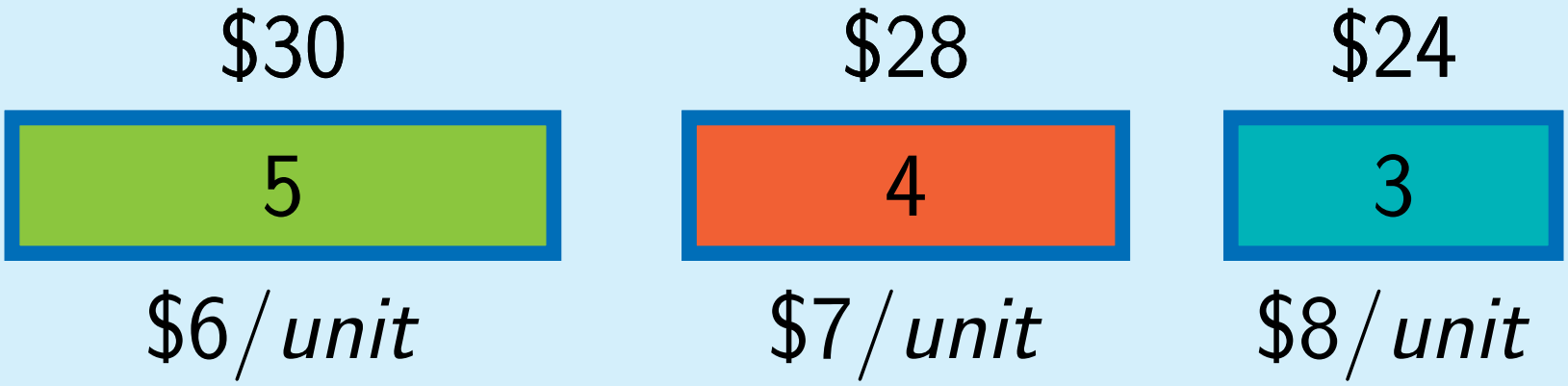
Lemma

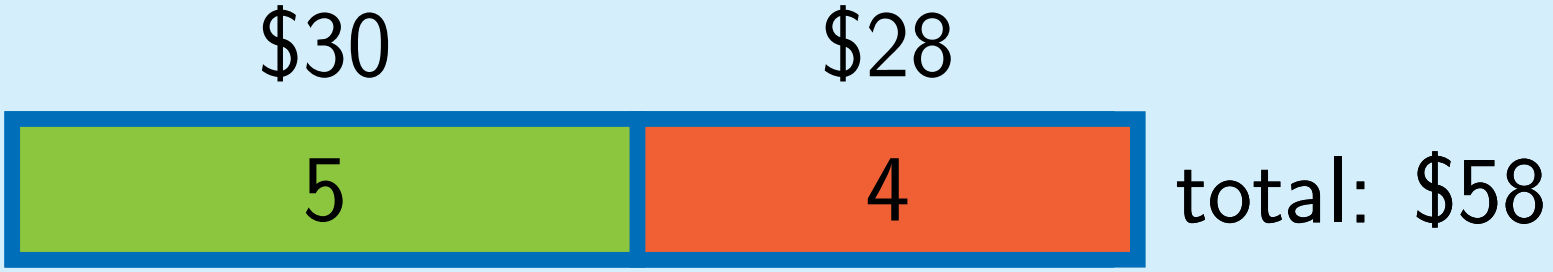
There exists an optimal solution that uses as much as possible of an item with the maximal value per unit of weight.

Proof



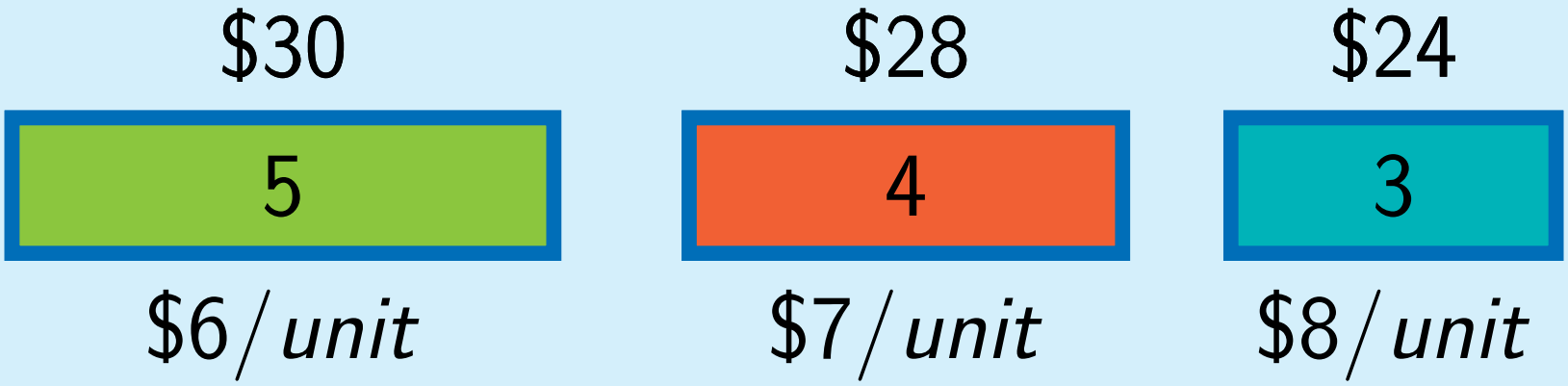
Proof

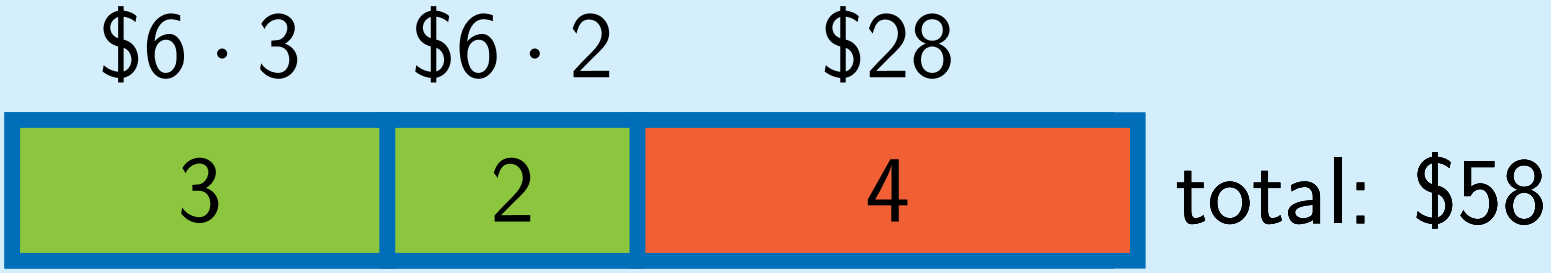




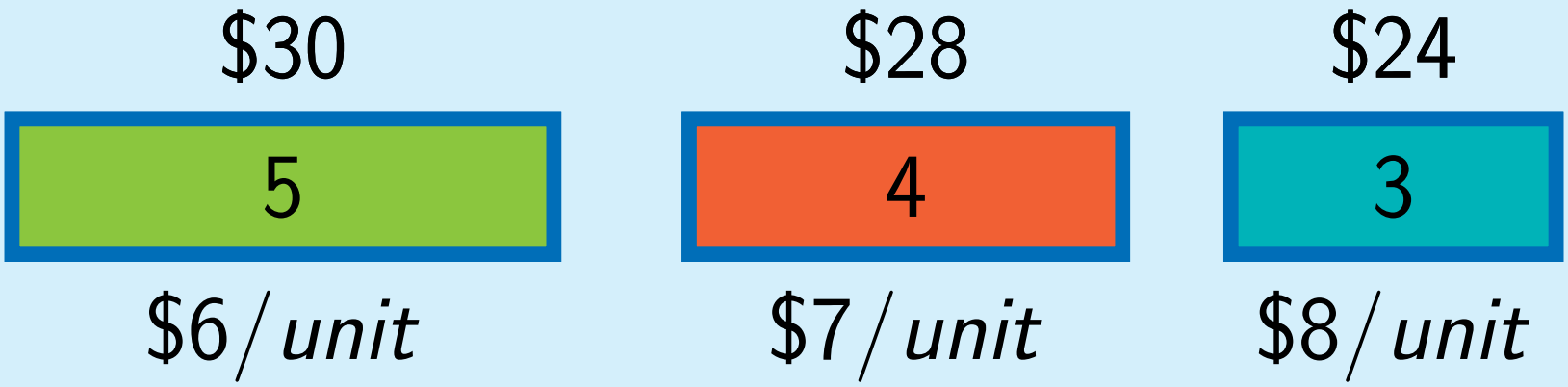
total: $58

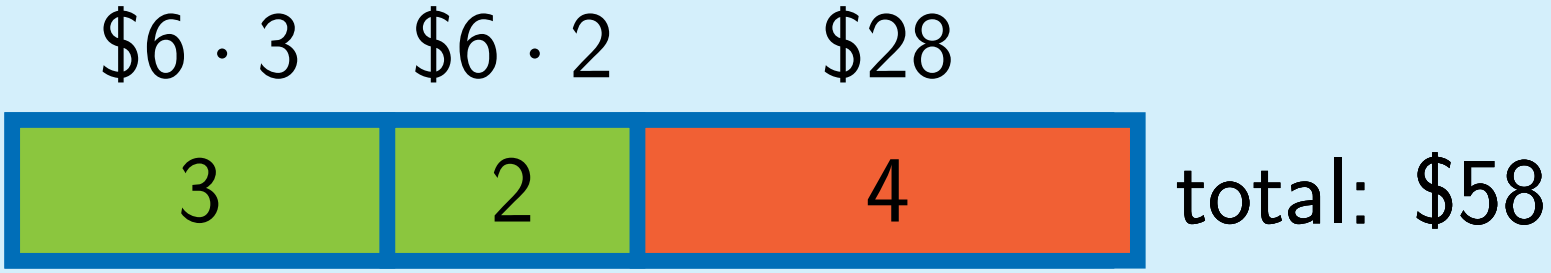
Proof

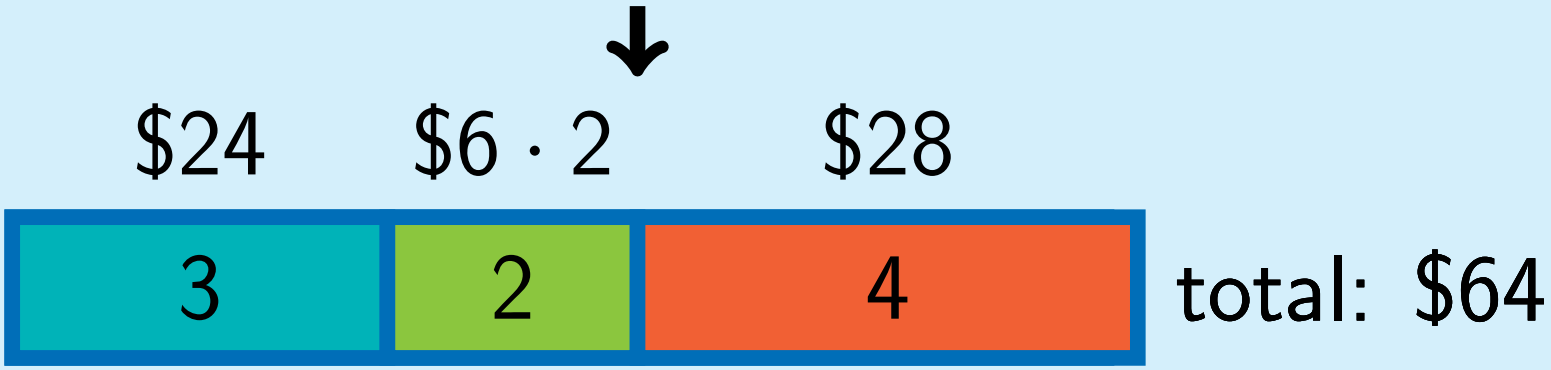




Proof


$30
$28
$24
5
4
3
$6/unit
$7/unit
$8/unit


$6 · 3
$6 · 2
$28
3
2
4
total: $58


$24
$6 · 2
$28
3
2
4
total: $64

Greedy Algorithm

- While knapsack is not full

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$
- If item fits into knapsack, take all of it

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$
- If item fits into knapsack, take all of it
- Otherwise take so much as to fill the knapsack

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$
- If item fits into knapsack, take all of it
- Otherwise take so much as to fill the knapsack
- Return total value and amounts taken

Outline

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$
- If item fits into knapsack, take all of it
- Otherwise take so much as to fill the knapsack
- Return total value and amounts taken

BestItem($w_1, v_1, \ldots, w_n, v_n$)

$maxValuePerWeight \leftarrow 0$

$bestItem \leftarrow 0$

for i from 1 to n:

  if $w_i > 0$:

    if $\frac{v_i}{w_i} > maxValuePerWeight$:

      $maxValuePerWeight \leftarrow \frac{v_i}{w_i}$

      $bestItem \leftarrow i$

return $bestItem$

BestItem($w_1, v_1, \ldots, w_n, v_n$)

$maxValuePerWeight \leftarrow 0$

$bestItem \leftarrow 0$

for i from 1 to n:

  if $w_i > 0$:

    if $\frac{v_i}{w_i} > maxValuePerWeight$:

      $maxValuePerWeight \leftarrow \frac{v_i}{w_i}$

      $bestItem \leftarrow i$

return $bestItem$

BestItem($w_1, v_1, \ldots, w_n, v_n$)

$maxValuePerWeight \leftarrow 0$

$bestItem \leftarrow 0$

```
for i from 1 to n:
   if w_i > 0:
      if v_i / w_i > maxValuePerWeight:
         maxValuePerWeight ← v_i / w_i
         bestItem ← i
return bestItem
```

BestItem($w_1, v_1, \ldots, w_n, v_n$)

$maxValuePerWeight \leftarrow 0$

$bestItem \leftarrow 0$

```
for i from 1 to n:
    if w_i > 0:
        if v_i / w_i > maxValuePerWeight:
            maxValuePerWeight ← v_i / w_i
            bestItem ← i
return bestItem
```

BestItem($w_1, v_1, \ldots, w_n, v_n$)

$maxValuePerWeight \leftarrow 0$

$bestItem \leftarrow 0$

for i from 1 to n:

  if $w_i > 0$:

    if $\frac{v_i}{w_i} > maxValuePerWeight$:

      $maxValuePerWeight \leftarrow \frac{v_i}{w_i}$

      $bestItem \leftarrow i$

return $bestItem$

BestItem($w_1, v_1, \ldots, w_n, v_n$)

$maxValuePerWeight \leftarrow 0$

$bestItem \leftarrow 0$

for i from 1 to n:

  if $w_i > 0$:

    if $\frac{v_i}{w_i} > maxValuePerWeight$:

      $maxValuePerWeight \leftarrow \frac{v_i}{w_i}$

      $bestItem \leftarrow i$

return $bestItem$

BestItem($w_1, v_1, \ldots, w_n, v_n$)

$maxValuePerWeight \leftarrow 0$

$bestItem \leftarrow 0$

```
for i from 1 to n:
    if w_i > 0:
        if v_i / w_i > maxValuePerWeight:
            maxValuePerWeight ← v_i / w_i
            bestItem ← i
return bestItem
```

BestItem($w_1, v_1, \ldots, w_n, v_n$)

$maxValuePerWeight \leftarrow 0$

$bestItem \leftarrow 0$

for i from 1 to n:

  if $w_i > 0$:

    if $\frac{v_i}{w_i} > maxValuePerWeight$:

      $maxValuePerWeight \leftarrow \frac{v_i}{w_i}$

      $bestItem \leftarrow i$

return *bestItem*

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow \texttt{BestItem}(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow \texttt{BestItem}(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow \texttt{BestItem}(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow \texttt{BestItem}(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow \texttt{BestItem}(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow$ `BestItem`$(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow$ `BestItem`$(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow \texttt{BestItem}(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow$ `BestItem`$(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow \texttt{BestItem}(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow \texttt{BestItem}(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Knapsack$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`repeat` n `times:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $i \leftarrow \texttt{BestItem}(w_1, v_1, \ldots, w_n, v_n)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Lemma

The running time of `Knapsack` is $O(n^2)$.

Lemma

The running time of `Knapsack` is $O(n^2)$.

Proof

- `BestItem` uses one loop with n iterations, so it is $O(n)$

Lemma

The running time of `Knapsack` is $O(n^2)$.

Proof

- `BestItem` uses one loop with n iterations, so it is $O(n)$
- Main loop is executed n times, and `BestItem` is called once per iteration

Lemma

The running time of `Knapsack` is $O(n^2)$.

Proof

- `BestItem` uses one loop with n iterations, so it is $O(n)$
- Main loop is executed n times, and `BestItem` is called once per iteration
- Overall, $O(n^2)$ □

Optimization

- It is possible to improve asymptotics!

Optimization

- It is possible to improve asymptotics!
- First, sort items by decreasing $\frac{v}{w}$

Assume $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$

KnapsackFast$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`for` i `from 1 to` n`:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Assume $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$

KnapsackFast$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`for` i `from 1 to` n`:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Assume $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$

KnapsackFast$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`for` i `from 1 to` n`:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Assume $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$

KnapsackFast$(W, w_1, v_1, \ldots, w_n, v_n)$

$amounts \leftarrow [0, 0, \ldots, 0]$
$totalValue \leftarrow 0$
`for` i `from 1 to` n`:`
  `if` $W = 0$`:`
    `return` $(totalValue, amounts)$
  $a \leftarrow \min(w_i, W)$
  $totalValue \leftarrow totalValue + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$
  $amounts[i] \leftarrow amounts[i] + a$
  $W \leftarrow W - a$
`return` $(totalValue, amounts)$

Asymptotics

- Now each iteration is $O(1)$
- `Knapsack` after sorting is $O(n)$
- Sort + `Knapsack` is $O(n \log n)$