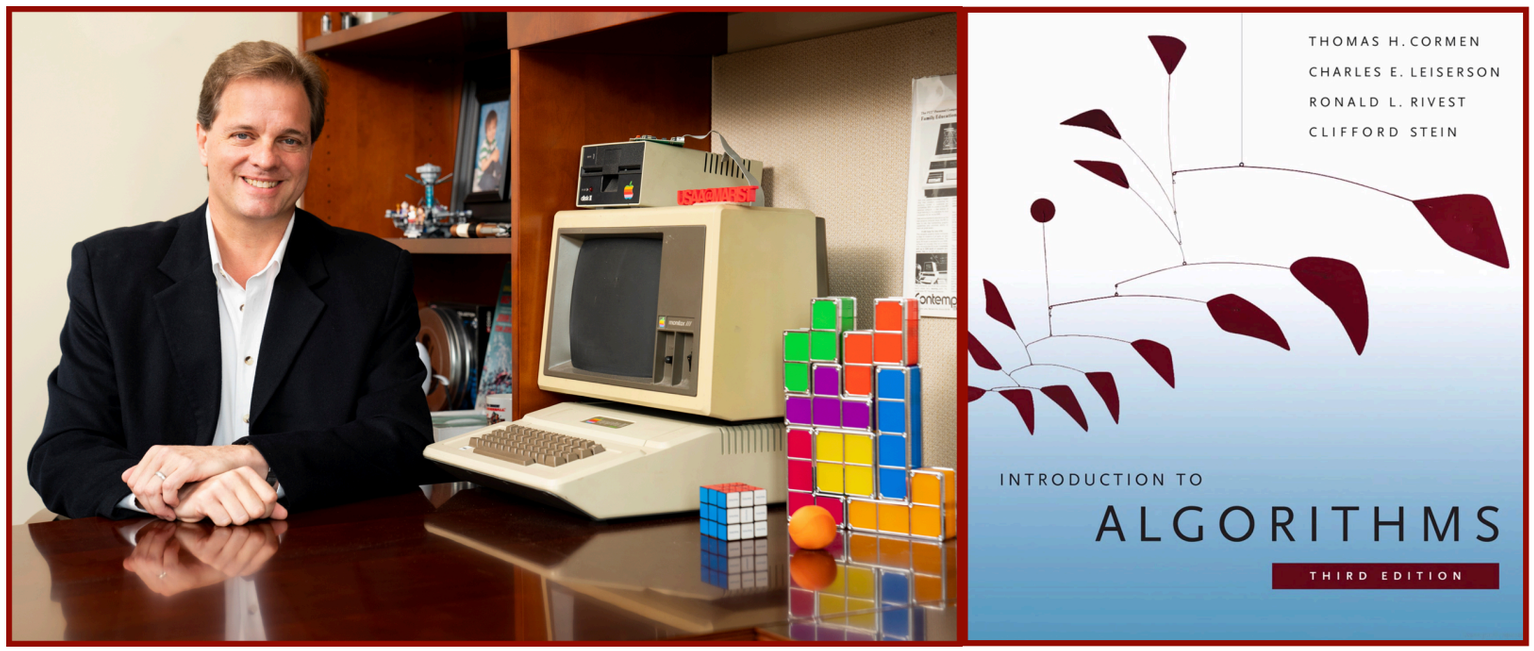
Greedy Algorithms



Alan G. Labouseur, Ph.D.
Alan.Labouseur@Marist.edu

Optimization Problems

Consider optimization problems:

- shortest paths on a map
- network routing
- activity scheduling with constraints
- minimal spanning trees

In this world of optimization problems, how can we insure that we get… the most?

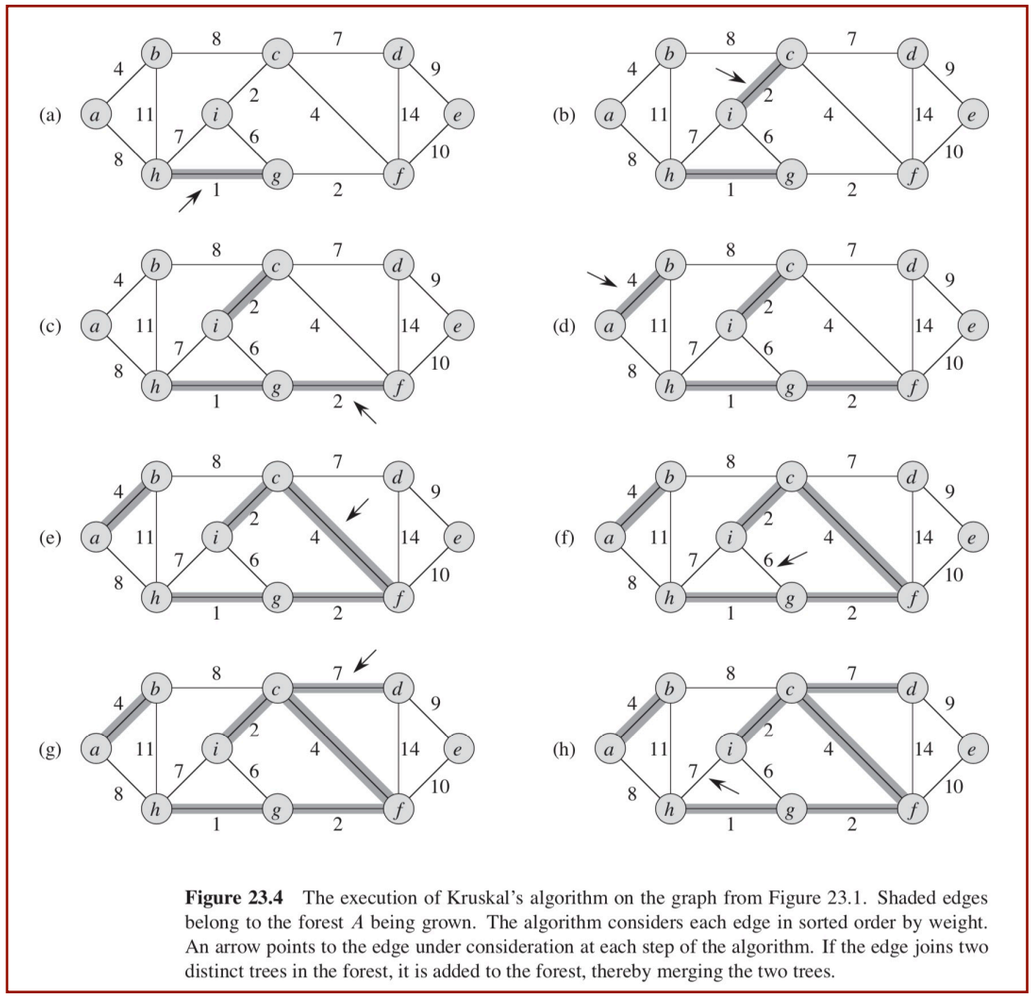


Figure 23.4 The execution of Kruskal's algorithm on the graph from Figure 23.1. Shaded edges belong to the forest A being grown. The algorithm considers each edge in sorted order by weight. An arrow points to the edge under consideration at each step of the algorithm. If the edge joins two distinct trees in the forest, it is added to the forest, thereby merging the two trees.

Maximizing our take

How can we insure that we get… the most?

- Make choices one at a time.
- Never look back.
- Hope for the best.

Sometime this actually works, but not always,
and we have to make smart choices.

This is the nature of “greedy” algorithms.

Greedy Algorithms

How can we insure that we get… the most?

- Make choices one at a time.
- Never look back.
- Hope for the best.

Sometime this actually works, but not always,
and we have to make smart choices.

This is the nature of “greedy” algorithms.

A greedy algorithm always makes the choice that looks best in the moment. In other words…

- make **locally optimal** choices
- and hope they lead to the **globally optimal** solution.

This does not always work. But for some kinds of problems, it does.

Greedy Algorithms

Problem: 0-1 Knapsack

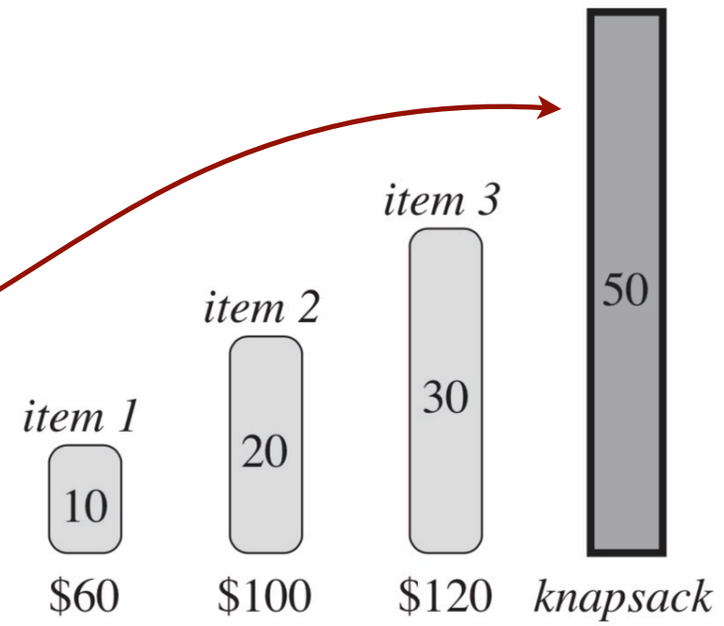
The ***0-1 knapsack problem*** is the following. A thief robbing a store finds n items. The ith item is worth v_i dollars and weighs w_i pounds, where v_i and w_i are integers. The thief wants to take as valuable a load as possible, but he can carry at most W pounds in his knapsack, for some integer W. Which items should he take? (We call this the 0-1 knapsack problem because for each item, the thief must either take it or leave it behind; he cannot take a fractional amount of an item or take an item more than once.)

Imagine trying to steal a bunch of golden idols. Each could be a different weight. You cannot divide the idols; each one is everything or nothing (i.e., no “partial credit”).

Greedy Algorithms

Problem: 0-1 Knapsack

More abstractly (but less fun) ponder this instance of the 0-1 Knapsack problem:



Your knapsack holds 50 lbs.

~~item~~ idol #1 weighs 10 lbs and is worth US$60.

Idol #2 weighs 20 lbs and is worth US$100.

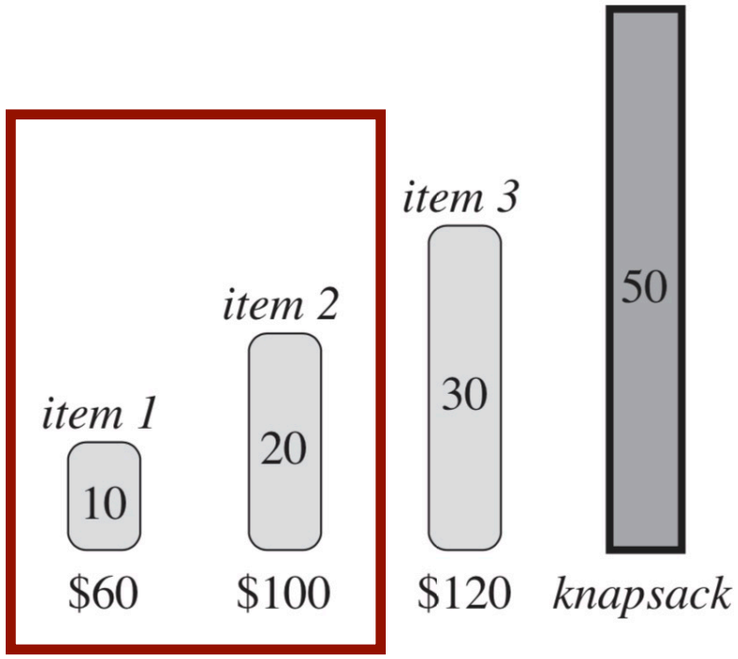
Idol #3 weighs 30 lbs and is worth US$120.

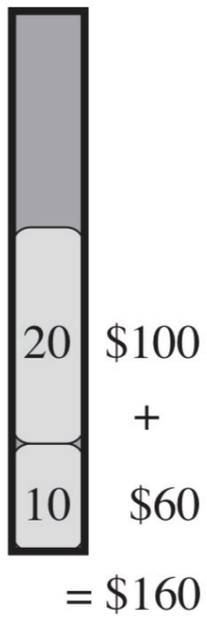
How do you fill your 50 lb. capacity knapsack to achieve optimal (maximum in this case) value?

Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?

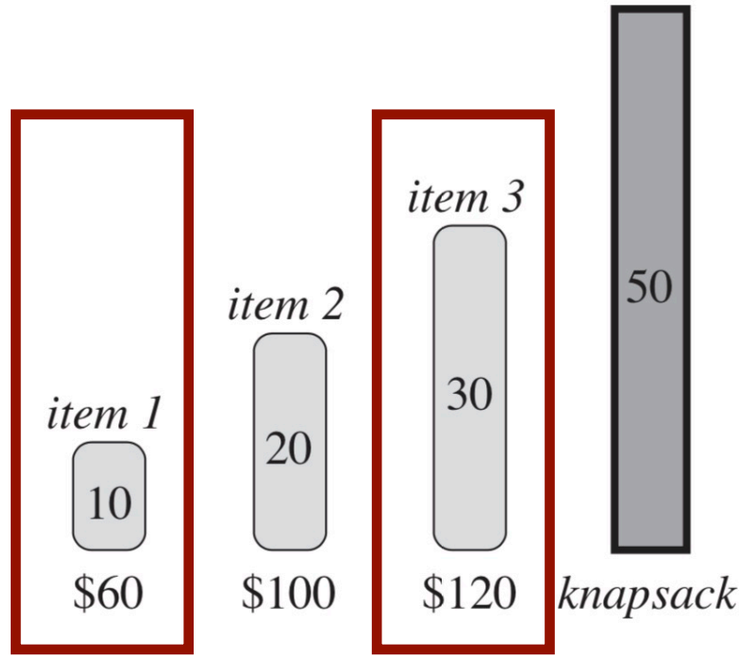


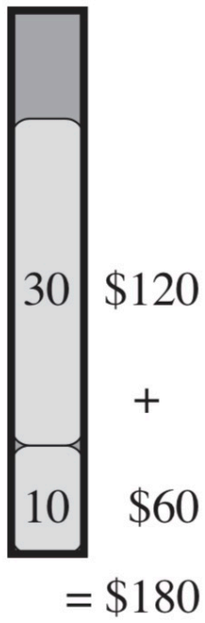


Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?

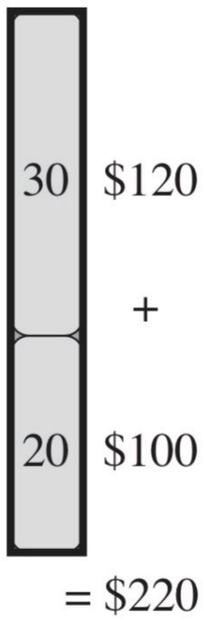


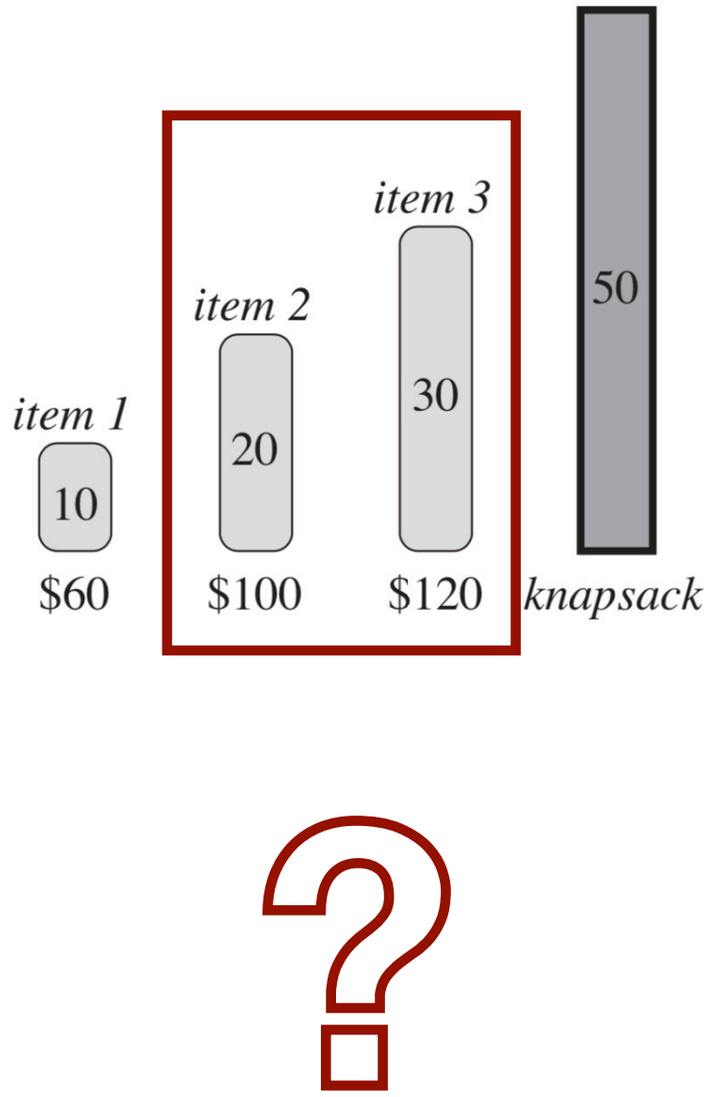


Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?

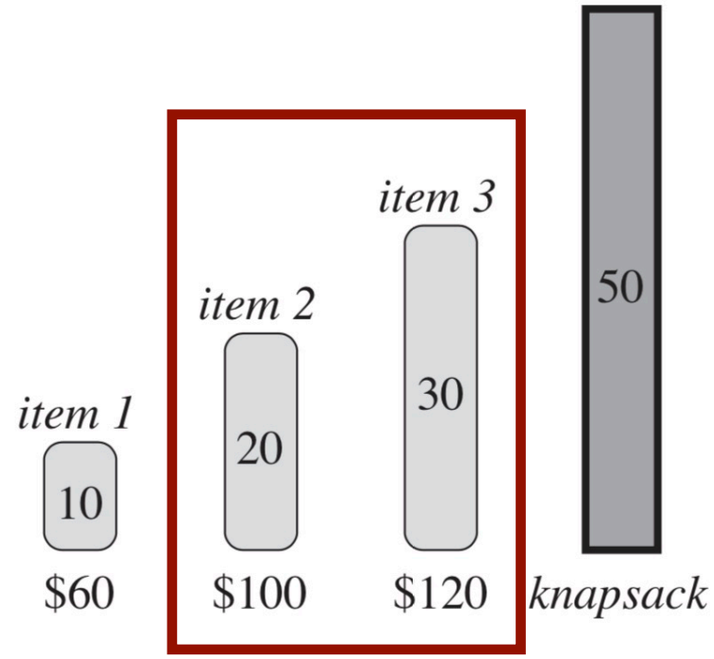


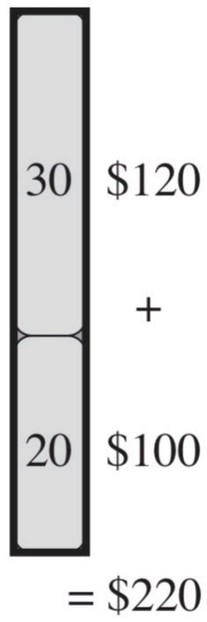


Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?



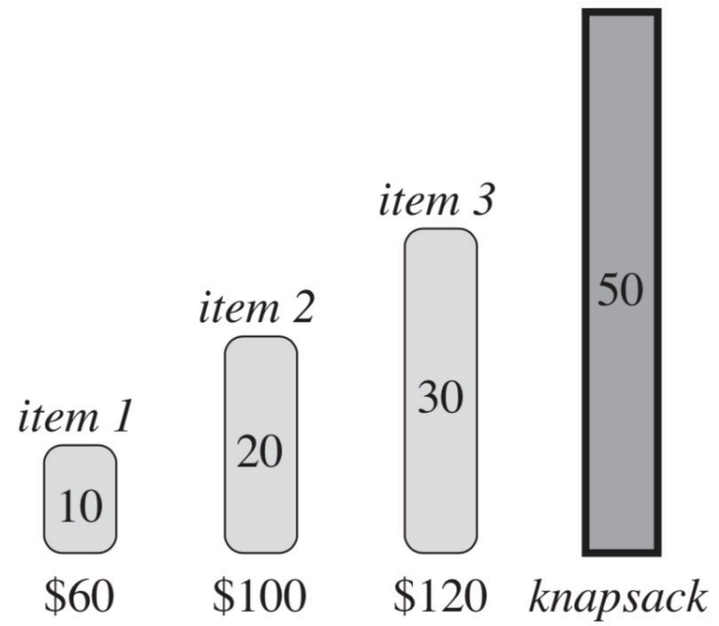


Winner! But how do we program it?

Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?



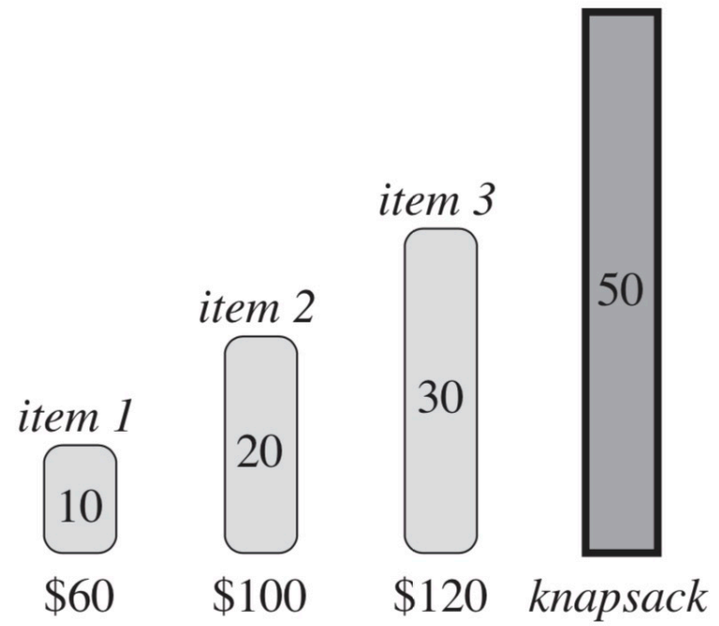
Notice the *unit-value*.

item 1: \$60/10 lbs = \$6/lb
item 2: \$100/20 lbs = \$5/lb
item 3: \$120/30 lbs = \$4/lb

Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?



item 1: \$60/10 lbs = \$6/lb
item 2: \$100/20 lbs = \$5/lb
item 3: \$120/30 lbs = \$4/lb

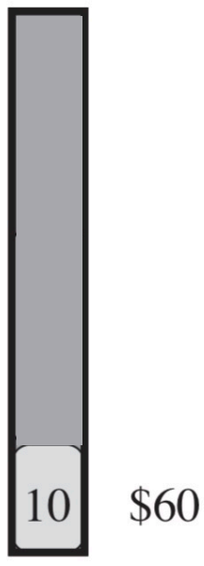
How do we program it?

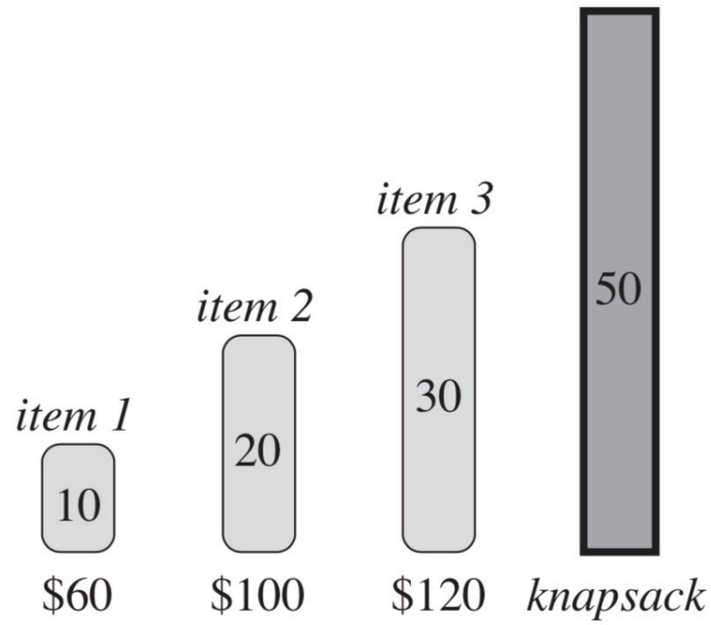
Sort by highest *value per unit* and take them in that order.

Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?





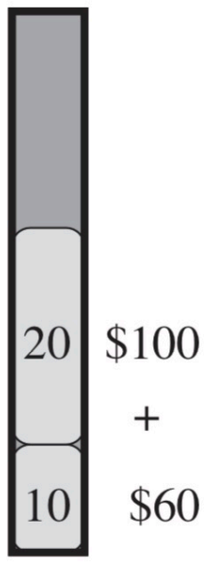
item 1: \$60/10 lbs = \$6/lb
item 2: \$100/20 lbs = \$5/lb
item 3: \$120/30 lbs = \$4/lb

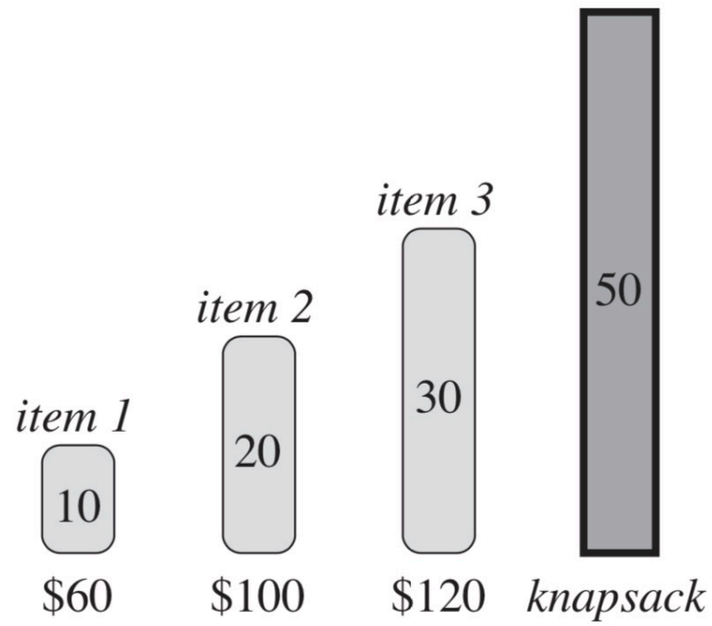
Sort by highest *value per unit* and take them in that order.

Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?





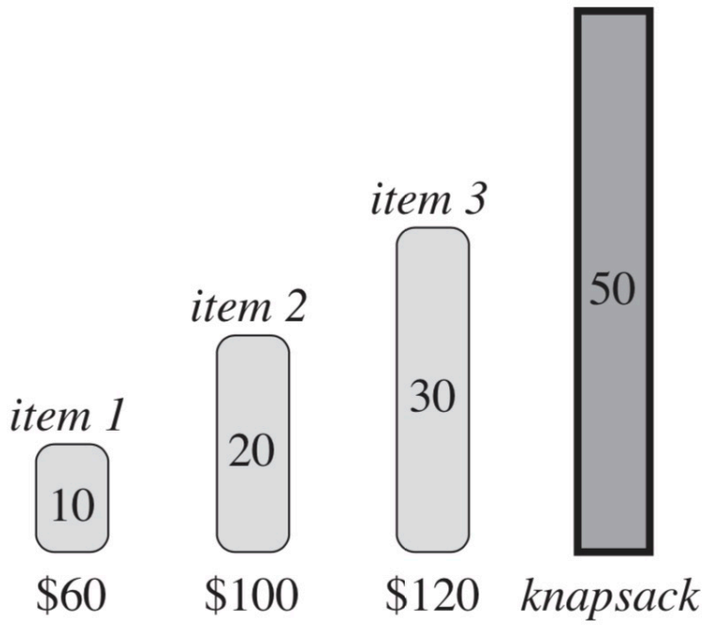
item 1: \$60/10 lbs = \$6/lb
item 2: \$100/20 lbs = \$5/lb
item 3: \$120/30 lbs = \$4/lb

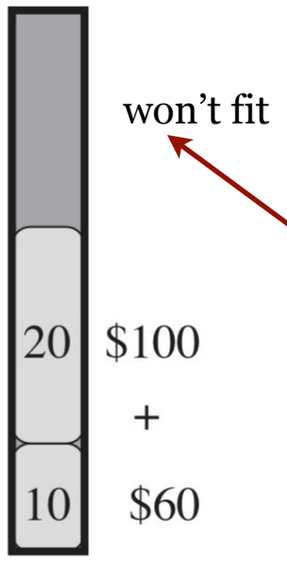
Sort by highest *value per unit* and take them in that order.

Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?





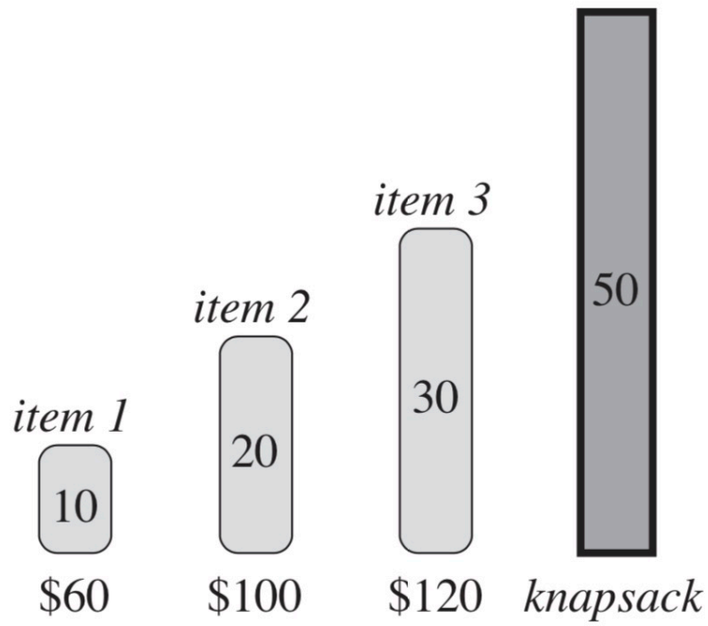
item 1: \$60/10 lbs = \$6/lb
item 2: \$100/20 lbs = \$5/lb
item 3: \$120/30 lbs = \$4/lb

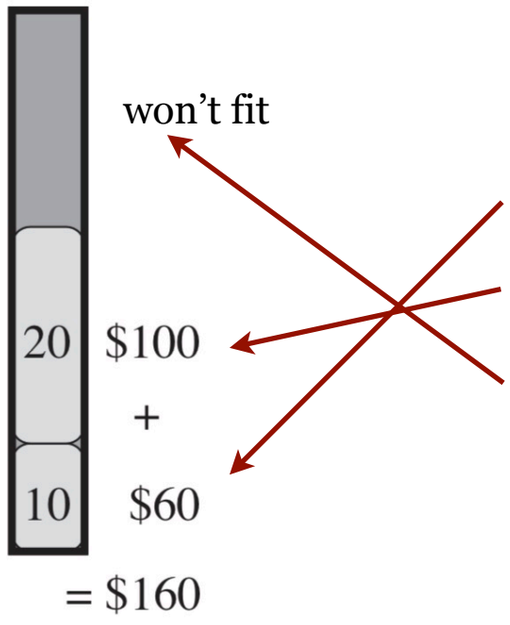
Sort by highest *value per unit* and take them in that order.

Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?





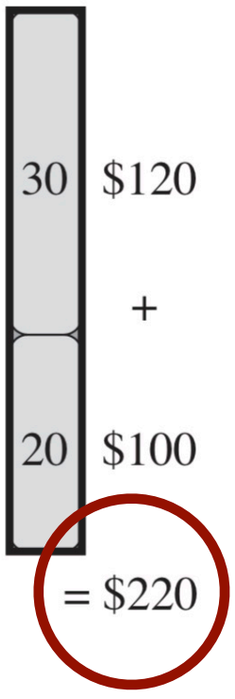
item 1: $60/10 lbs = $6/lb
item 2: $100/20 lbs = $5/lb
item 3: $120/30 lbs = $4/lb

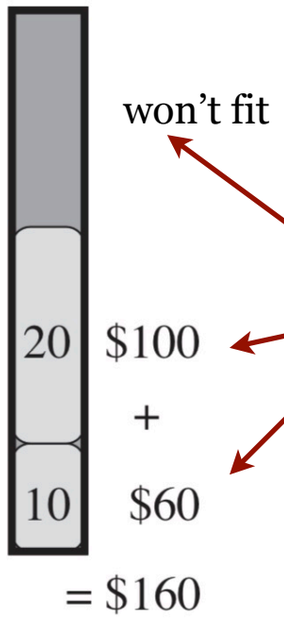
Sort by highest *value per unit* and take them in that order.

Greedy Algorithms

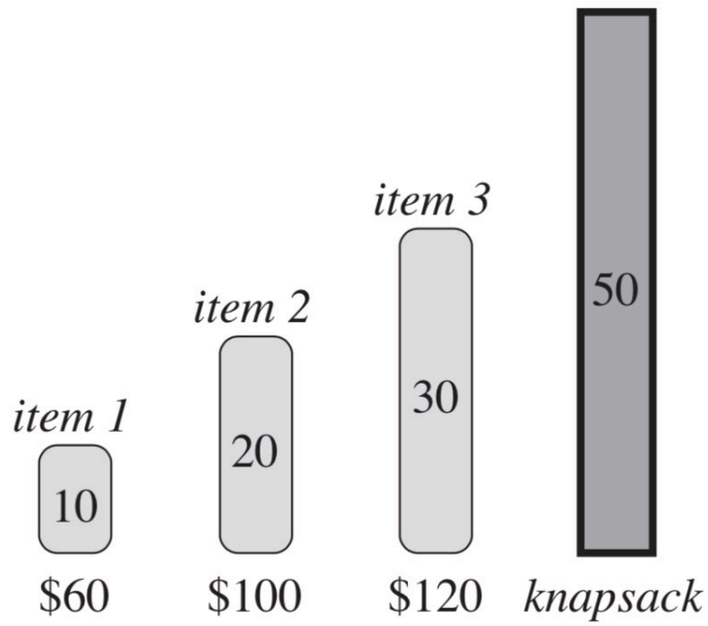
Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?





Oops.



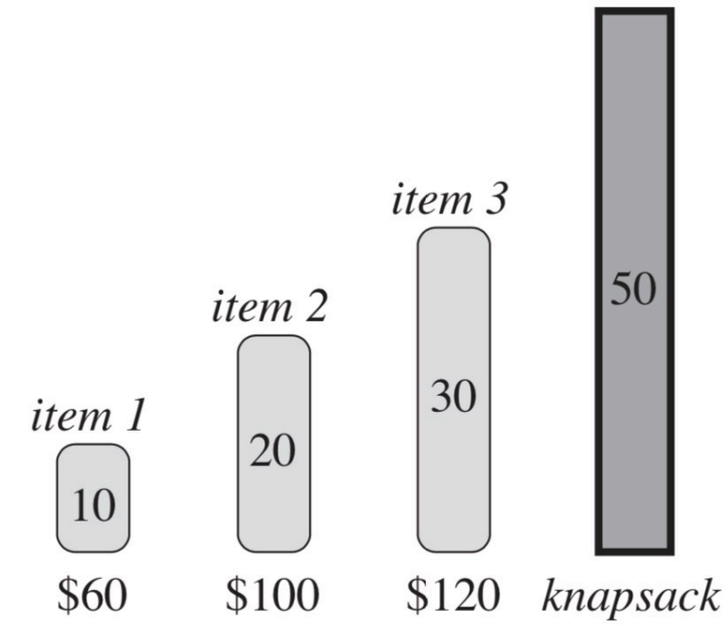
item 1: \$60/10 lbs = \$6/lb
item 2: \$100/20 lbs = \$5/lb
item 3: \$120/30 lbs = \$4/lb

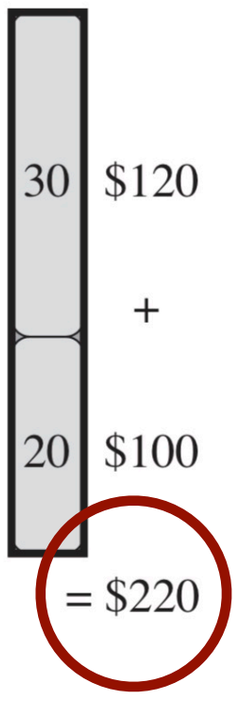
Sort by highest *value per unit* and take them in that order.

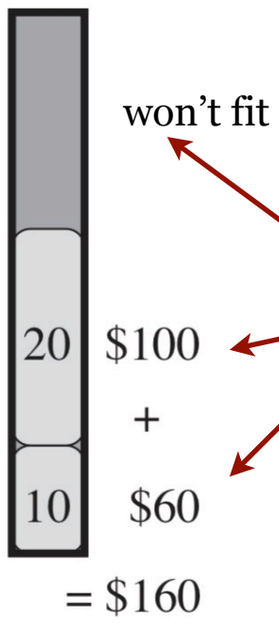
Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?







item 1: \$60/10 lbs = \$6/lb
item 2: \$100/20 lbs = \$5/lb
item 3: \$120/30 lbs = \$4/lb

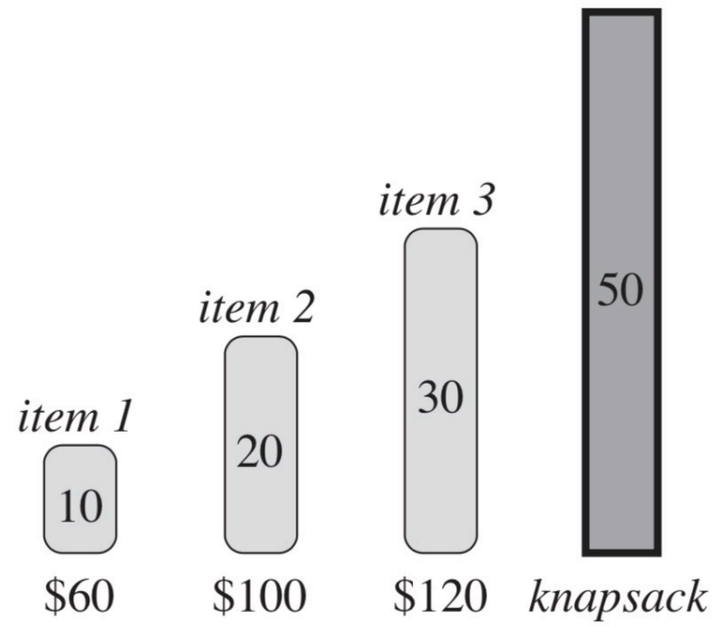
Sort by highest *value per unit* and take them in that order.

A greedy approach **does not work** for 0-1 knapsack.

Greedy Algorithms

Problem: 0-1 Knapsack

How do you fill your 50 lb. capacity knapsack to achieve maximum value?



A greedy approach **does not work** for 0-1 knapsack.

But it seems promising.

Let's change the problem just a little…

item 1: $60/10 lbs = $6/lb
item 2: $100/20 lbs = $5/lb
item 3: $120/30 lbs = $4/lb

Sort by highest *value per unit* and take them in that order.

Greedy Algorithms

Problem: Fractional Knapsack

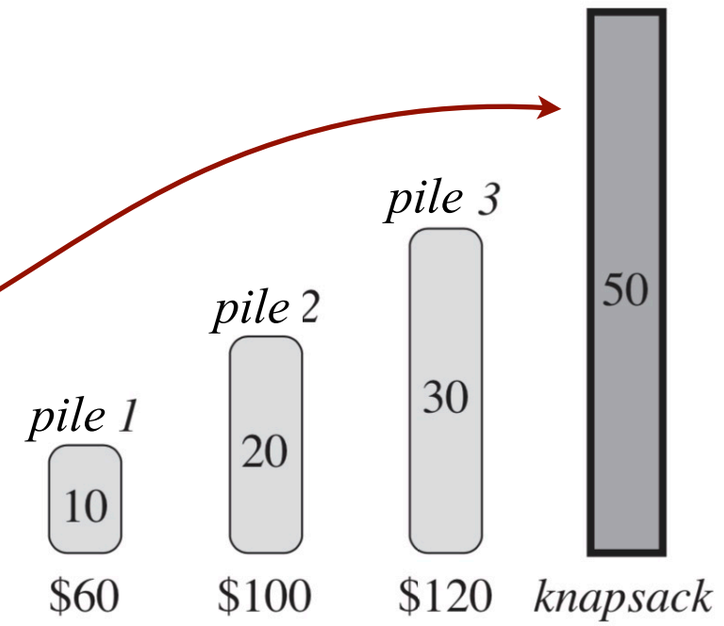
In the ***fractional knapsack problem***, the setup is the same, but the thief can take fractions of items, rather than having to make a binary (0-1) choice for each item. You can think of an item in the 0-1 knapsack problem as being like a gold idol and an item in the fractional knapsack problem as more like gold dust.

Imagine trying to steal from piles of gold dust varying in purity. The gold in each pile could be a different weight. But since it's dust, you can take any fraction of the pile you like; it's **not** everything or nothing (i.e., there **is** "partial credit" in this case).

Greedy Algorithms

Problem: Fractional Knapsack

Ponder this instance of the Fractional Knapsack problem:



Your knapsack holds 50 lbs.

pile #1 weighs 10 lbs and is worth US$60 total.

pile #2 weighs 20 lbs and is worth US$100 total.

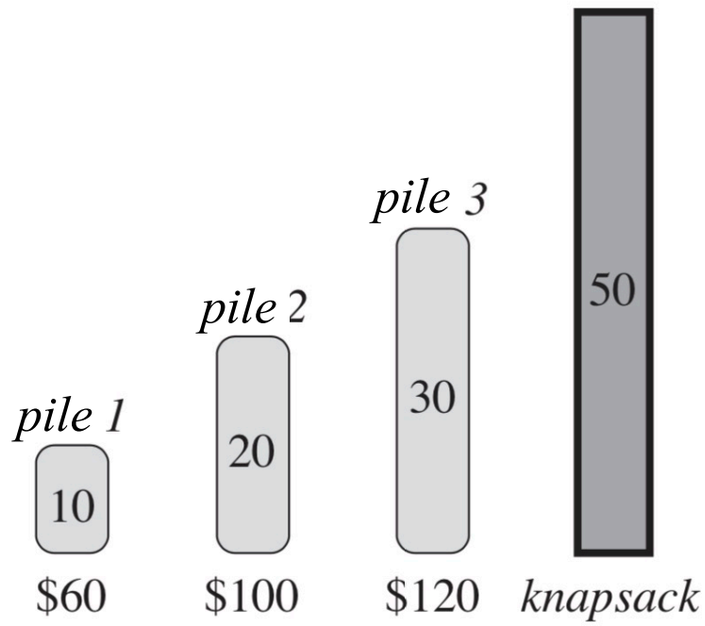
pile #3 weighs 30 lbs and is worth US$120 total.

How do you fill your 50 lb. capacity knapsack to achieve optimal (maximum in this case) value?

Greedy Algorithms

Problem: Fractional Knapsack

Let's try the same algorithm.



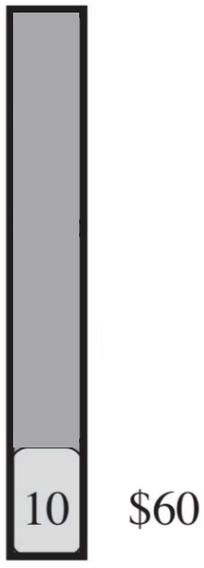
pile 1: \$60/10 lbs = \$6/lb
pile 2: \$100/20 lbs = \$5/lb
pile 3: \$120/30 lbs = \$4/lb

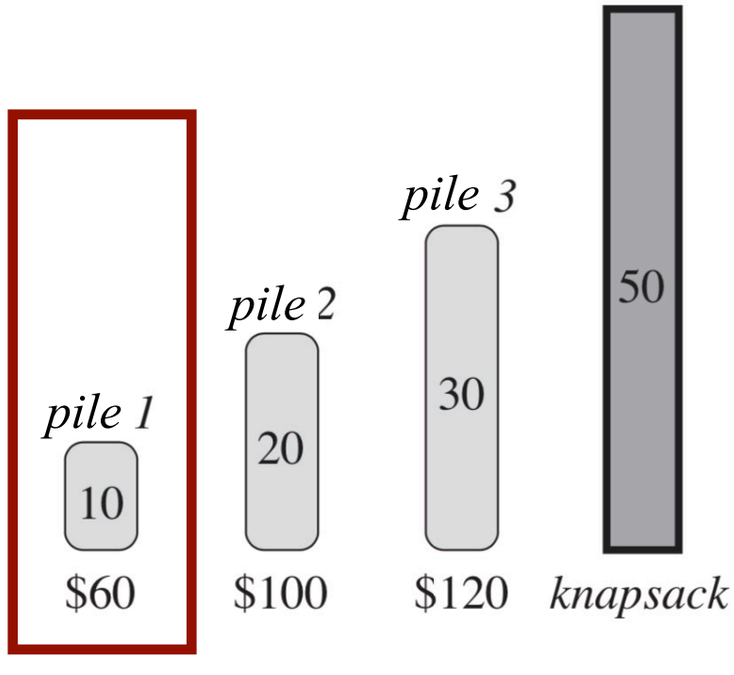
Sort by highest *value per unit* and take them in that order.

Greedy Algorithms

Problem: Fractional Knapsack

Let's try the same algorithm.
Take all of the most valuable pile.





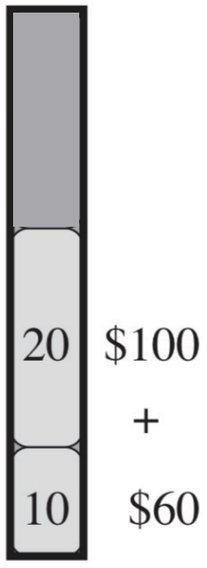
pile 1: \$60/10 lbs = \$6/lb
pile 2: \$100/20 lbs = \$5/lb
pile 3: \$120/30 lbs = \$4/lb

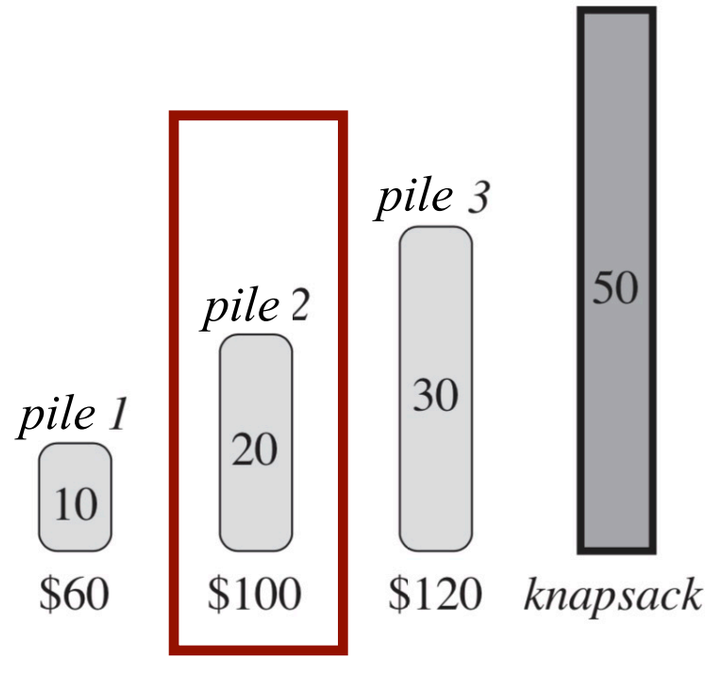
Sort by highest *value per unit* and take them in that order.

Greedy Algorithms

Problem: Fractional Knapsack

Let's try the same algorithm.
Take all of the most valuable pile.
Take all of the 2nd most valuable pile.





pile 1: \$60/10 lbs = \$6/lb
pile 2: \$100/20 lbs = \$5/lb
pile 3: \$120/30 lbs = \$4/lb

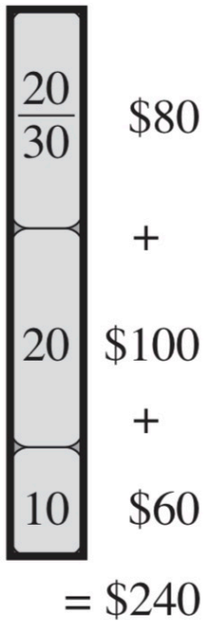
Sort by highest *value per unit* and take them in that order.

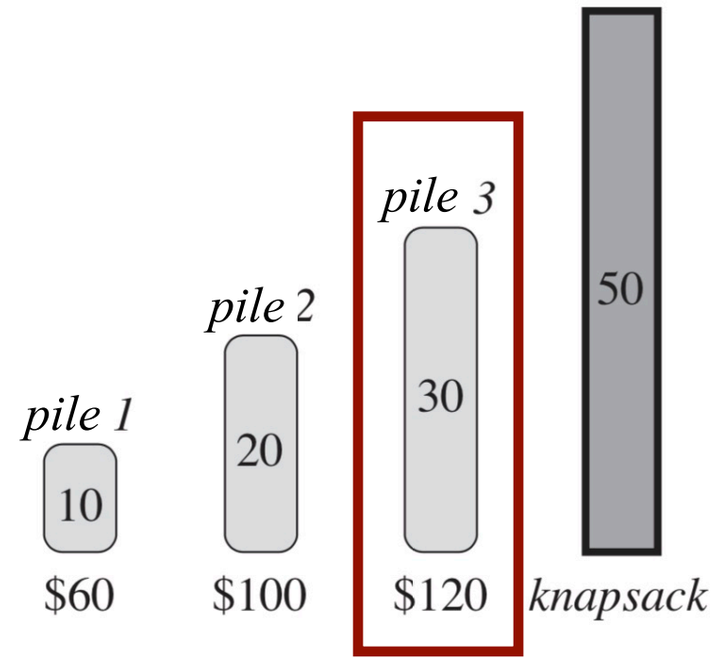
Greedy Algorithms

Problem: Fractional Knapsack

Let's try the same algorithm.

Take all of the most valuable pile.
Take all of the 2nd most valuable pile.
Take as much of the 3rd most valuable pile as will fit.



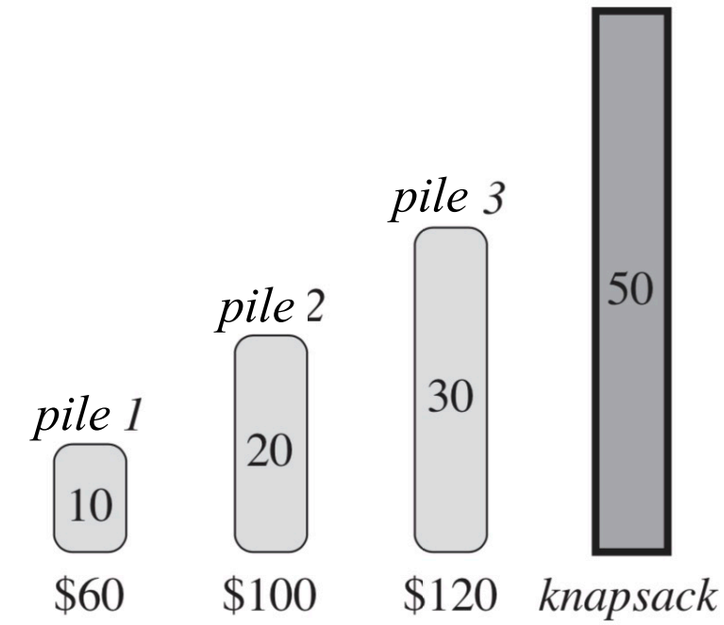


pile 1: \$60/10 lbs = \$6/lb
pile 2: \$100/20 lbs = \$5/lb
pile 3: \$120/30 lbs = \$4/lb

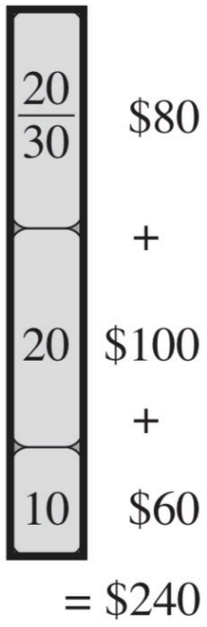
Sort by highest *value per unit* and take them in that order.

Greedy Algorithms

Problem: Fractional Knapsack



Let's try the same algorithm.
Take all of the most valuable pile.
Take all of the 2nd most valuable pile.
Take as much of the 3rd most valuable pile as will fit.



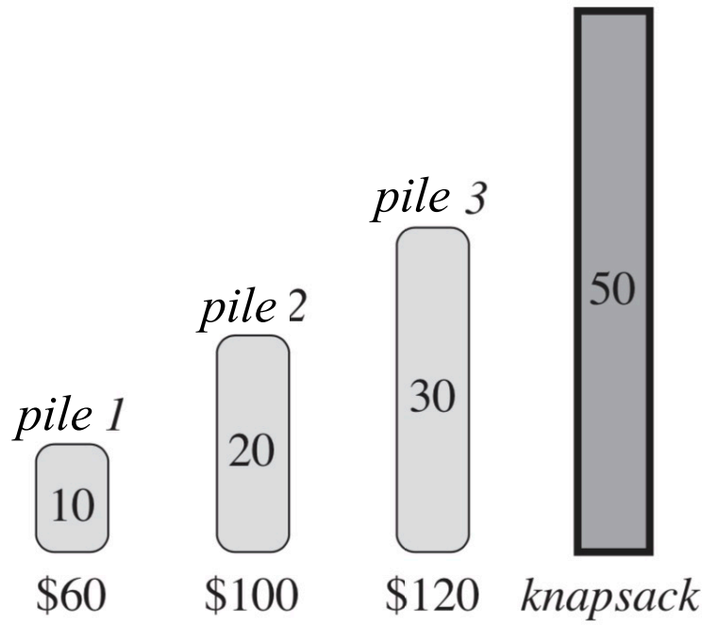
pile 1: $60/10 lbs = $6/lb
pile 2: $100/20 lbs = $5/lb
pile 3: $120/30 lbs = $4/lb

Is this optimal?

Sort by highest *value per unit* and take them in that order.

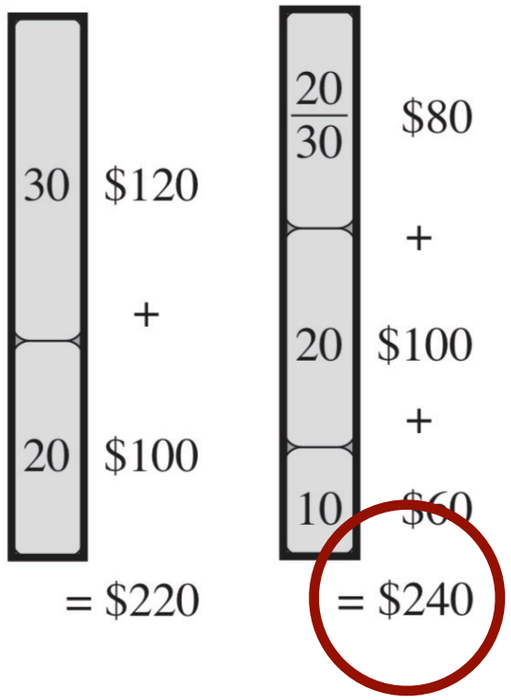
Greedy Algorithms

Problem: Fractional Knapsack



Let's try the same algorithm.

Take all of the most valuable pile.
Take all of the 2nd most valuable pile.
Take as much of the 3rd most valuable pile as will fit.



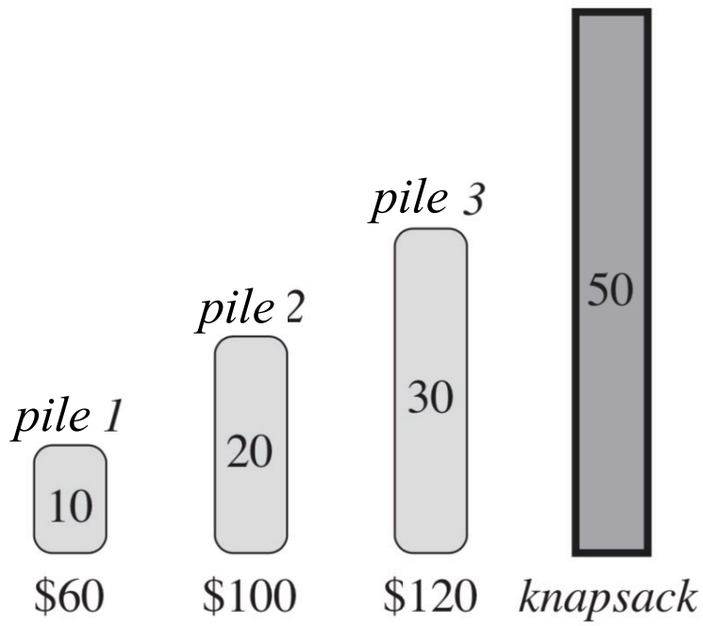
Yes! We have a new winner!

pile 1: \$60/10 lbs = \$6/lb
pile 2: \$100/20 lbs = \$5/lb
pile 3: \$120/30 lbs = \$4/lb

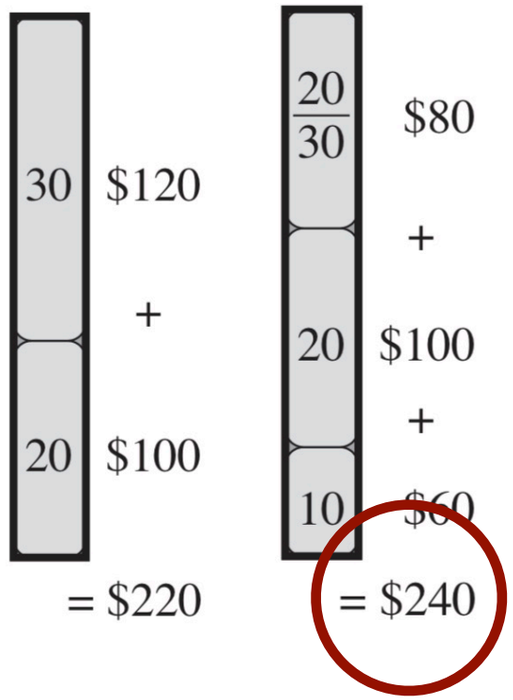
Sort by highest *value per unit* and take them in that order.

Greedy Algorithms

Problem: Fractional Knapsack



Let's try the same algorithm.
Take all of the most valuable pile.
Take all of the 2nd most valuable pile.
Take as much of the 3rd most valuable pile as will fit.



pile 1: $60/10 lbs = $6/lb
pile 2: $100/20 lbs = $5/lb
pile 3: $120/30 lbs = $4/lb

Sort by highest *value per unit* and take them in that order.

A greedy approach **does work** for Fractional Knapsack.

Greedy Algorithms

Example: Spice Heist on Arrakis

Input file

```
-- She who controls the spice controls the universe.

-- Available spice to take in scoops:
spice name = red;    total_price =  4.0;  qty = 4;
spice name = green;  total_price = 12.0;  qty = 6;
spice name = blue;   total_price = 40.0;  qty = 8;
spice name = orange; total_price = 18.0;  qty = 2;

-- Available knapsacks in which to keep spice:
knapsack capacity =  1;
knapsack capacity =  6;
knapsack capacity = 10;
knapsack capacity = 20;
knapsack capacity = 21;
```

Output

Knapsack of capacity 1 is worth 9 quatloos and contains 1 scoop of orange.

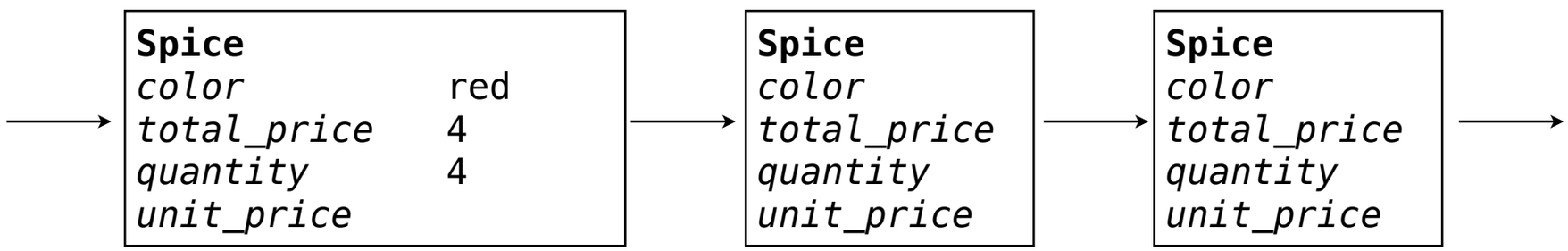
Knapsack of capacity 6 is worth 38 quatloos and contains 2 scoops of orange, 4 scoops of blue.

.
.
.

Greedy Algorithms

Example: Spice Heist on Arrakis — Implementation

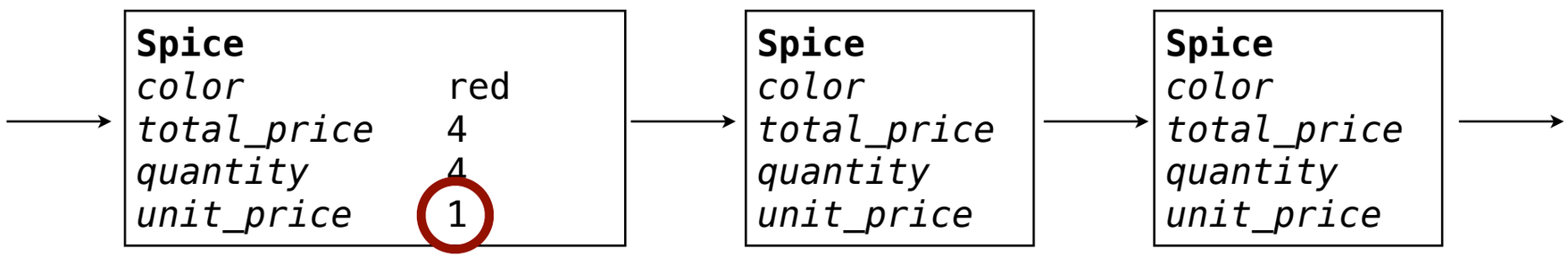
Read and parse file.
Create a list of *Spice* objects.



Greedy Algorithms

Example: Spice Heist on Arrakis — Implementation

Read and parse file.
Create a list of *Spice* objects.



Compute the unit price for each.
Sort the list by `unit_price` from high to low.
Fill each knapsack.
Write the output.

Note: Each “turn” is independent.

Greedy Algorithms

Example: Spice Heist on Arrakis

Input file

```
-- She who controls the spice controls the universe.

-- Available spice to take in scoops:
spice name = red;    total_price =  4.0;  qty = 4;
spice name = green;  total_price = 12.0;  qty = 6;
spice name = blue;   total_price = 40.0;  qty = 8;
spice name = orange; total_price = 18.0;  qty = 2;

-- Available knapsacks in which to keep spice:
knapsack capacity =  1;
knapsack capacity =  6;
knapsack capacity = 10;
knapsack capacity = 20;
knapsack capacity = 21;
```

Output

```
Knapsack of capacity 1 is worth 9 quatloos and
contains 1 scoop of orange.

Knapsack of capacity 6 is worth 38 quatloos and
contains 2 scoops of orange, 4 scoops of blue.
  .
  .
  .
```

- Read and parse file.
- Create a list of *Spice* objects.
- Compute the unit price for each.
- Sort the list by `unit_price` from high to low.
- Fill each knapsack.
- Write the output.