Lecture 09 — Greedy Algorithms (Part 1)

COMPSCI 308 — Design and Analysis of Algorithms

Xing Shi Cai https://newptcai.gitlab.io

Duke Kunshan University, Kunshan, China

Jan-Mar 2026

Summary

Assignments[1]

Practice makes perfect!

Introduction to Algorithms (ITA) 📖
Required Readings:

- Section 16.1.
- Section 16.2.

✏️ Required Exercises:

- Exercises 16.1 — 1–5.
- Exercises 16.2 — 1–5.

[1] 💣 Assignments will not be collected; however, quiz problems will be selected from them. (This includes both Readings and Exercises.)

ITA 16 Greedy Algorithms

Greedy Algorithms

A greedy algorithm picks the best local choice at each step.

It builds a solution by repeating this choice.

It is fast, but it can miss the global optimum.

Figure 1: Greed can be healthy!

ITA 16.1 An Activity-Selection Problem

ITA 16.1 An Activity-Selection Problem

The Problem

A Film Festival Problem

The screening schedule of six 🎞 at a film festival is shown below.

What is the maximum number of films that you can watch in their entirety?

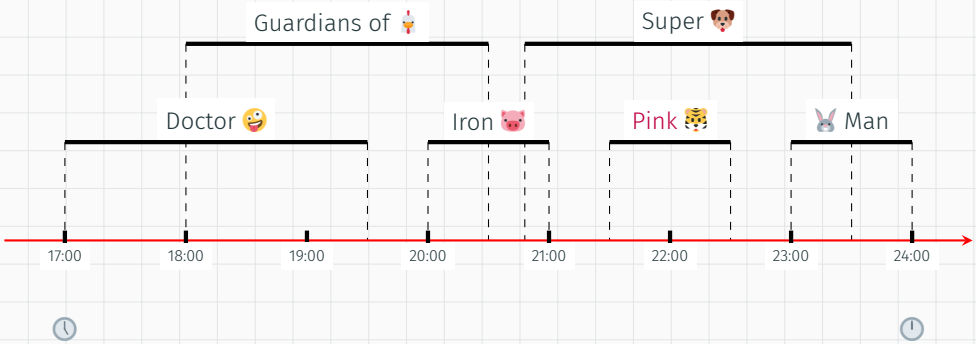


Figure 2: The screening schedule

Activity-Selection Problem

Consider a sequence of n intervals

$$[s_1, f_1), [s_2, f_2), \ldots, [s_n, f_n),$$

where $0 < s_i < f_i < \infty$ for all $i = 1, \ldots, n$.

Activity-Selection Problem

Consider a sequence of n intervals

$$[s_1, f_1), [s_2, f_2), \ldots, [s_n, f_n),$$

where $0 < s_i < f_i < \infty$ for all $i = 1, \ldots, n$.

Assume also that —

$$f_1 \leq f_2 \leq \ldots \leq f_{n-1} \leq f_n.$$

🍰 What should we do if this condition is not met?

Activity-Selection Problem

Consider a sequence of n intervals

$$[s_1, f_1), [s_2, f_2), \ldots, [s_n, f_n),$$

where $0 < s_i < f_i < \infty$ for all $i = 1, \ldots, n$.

Assume also that —

$$f_1 \leq f_2 \leq \ldots \leq f_{n-1} \leq f_n.$$

🍰 What should we do if this condition is not met?

🎯 The objective is to find the maximum set of non-overlapping intervals in the sequence.

ITA 16.1 An Activity-Selection Problem

A Dynamic Programming Solution

Some Notation

Let

$$S_{i,j} = \{[s_k, f_k) : f_i \leq s_k \text{ and } f_k \leq s_j\}.$$

We treat i and j as sentinel boundaries, so $S_{i,j}$ only includes intervals strictly between them.

Thus, we typically add two sentinel endpoints with $f_0 = 0$ and $s_{n+1} = \infty$.

🍰 Which intervals are in $S_{2,2}$?

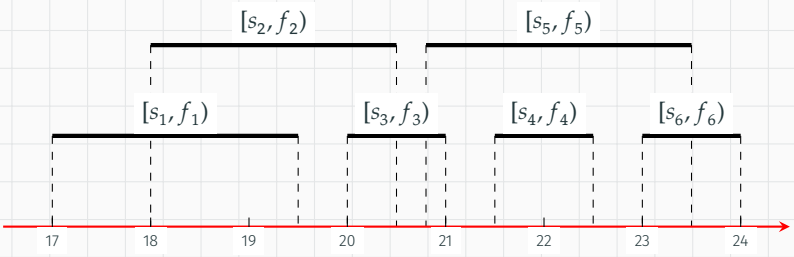


Figure 3: Six intervals

Some Notation

Let

$$S_{i,j} = \{[s_k, f_k) : f_i \leq s_k \text{ and } f_k \leq s_j\}.$$

We treat i and j as sentinel boundaries, so $S_{i,j}$ only includes intervals strictly between them.

Thus, we typically add two sentinel endpoints with $f_0 = 0$ and $s_{n+1} = \infty$.

🍰 Which intervals are in $S_{2,3}$?

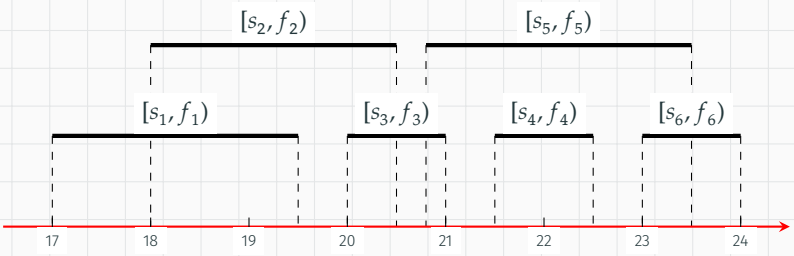


Figure 3: Six intervals

Some Notation

Let

$$S_{i,j} = \{[s_k, f_k) : f_i \leq s_k \text{ and } f_k \leq s_j\}.$$

We treat i and j as sentinel boundaries, so $S_{i,j}$ only includes intervals strictly between them.

Thus, we typically add two sentinel endpoints with $f_0 = 0$ and $s_{n+1} = \infty$.

🍰 Which intervals are in $S_{1,6}$?

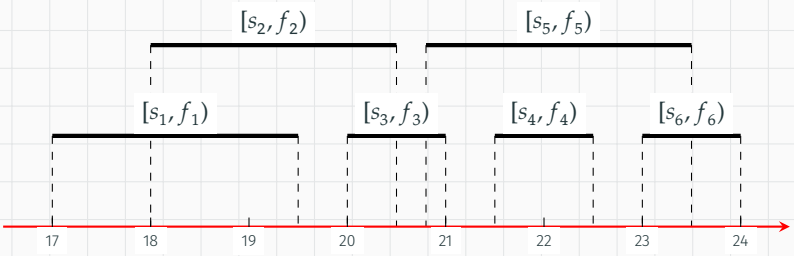


Figure 3: Six intervals

Some Notation

Let

$$S_{i,j} = \{[s_k, f_k) : f_i \leq s_k \text{ and } f_k \leq s_j\}.$$

We treat i and j as sentinel boundaries, so $S_{i,j}$ only includes intervals strictly between them.

Thus, we typically add two sentinel endpoints with $f_0 = 0$ and $s_{n+1} = \infty$.

🍰 Which intervals are in $S_{2,6}$?

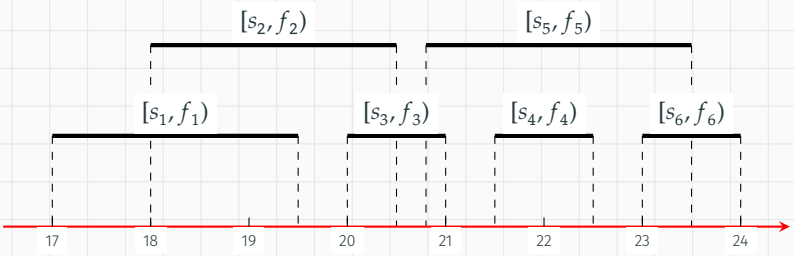


Figure 3: Six intervals

A Dynamic Programming Solution

Let $A_{i,j}$ be the maximum subset of non-overlapping intervals in $S_{i,j}$.

💡 Key observation — If $A_{i,j}$ includes $[s_k, f_k)$, then

$$A_{i,j} = A_{i,k} \cup \{[s_k, f_k)\} \cup A_{k,j},$$

where the union is over three disjoint sets.

🍰 Does $[s_i, f_i)$ or $[s_j, f_j)$ belong to $A_{i,j}$?

A Dynamic Programming Solution

Let $A_{i,j}$ be the maximum subset of non-overlapping intervals in $S_{i,j}$.

💡 Key observation — If $A_{i,j}$ includes $[s_k, f_k)$, then

$$A_{i,j} = A_{i,k} \cup \{[s_k, f_k)\} \cup A_{k,j},$$

where the union is over three disjoint sets.

Thus, letting $c[i,j] = \left|A_{i,j}\right|$, we have the recursion —

$$c[i,j] = \begin{cases} 0 & \text{if } i \geq j, \\ \max_{[s_k,f_k) \in A_{i,j}} c[i,k] + 1 + c[k,j] & \text{otherwise.} \end{cases}$$

This naturally leads to a *dynamic programming* algorithm.

ITA 16.1 An Activity-Selection Problem

A Greedy Algorithm

A Key Observation

In the maximum subset of non-overlapping intervals, there exists one interval that finishes first.

Note that we can replace it by $[s_1, f_1)$ while still having an optimal solution.

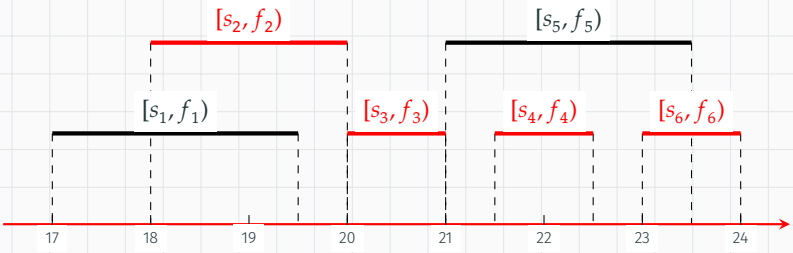


Figure 4: Swap the first interval in the optimal solution

A Key Observation

In the maximum subset of non-overlapping intervals, there exists one interval that finishes first.

Note that we can replace it by $[s_1, f_1)$ while still having an optimal solution.

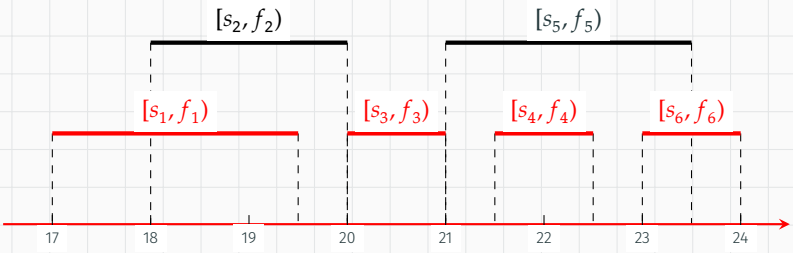


Figure 4: Swap the first interval in the optimal solution

Theorem 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \qquad \text{for all } k = 0, \dots, n.$$

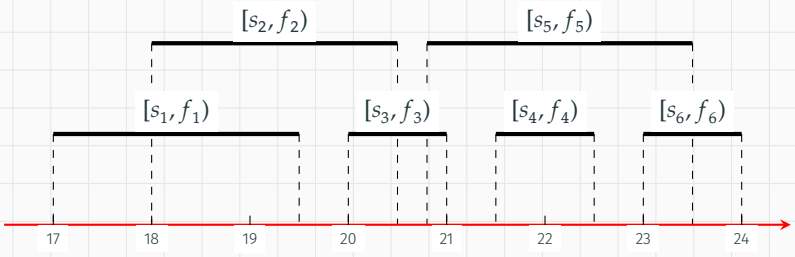


Figure 5: What is S_0?

Theorem 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \qquad \text{for all } k = 0, \dots, n.$$

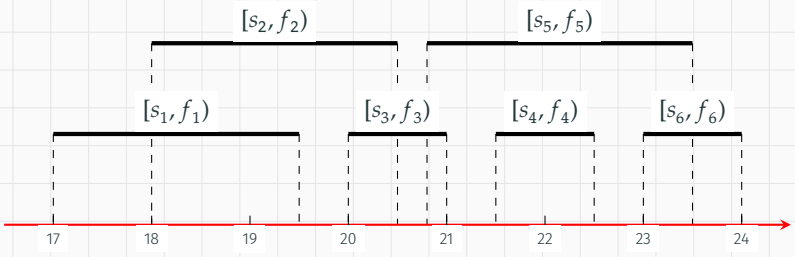


Figure 5: What is S_1?

Theorem 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \qquad \text{for all } k = 0, \dots, n.$$

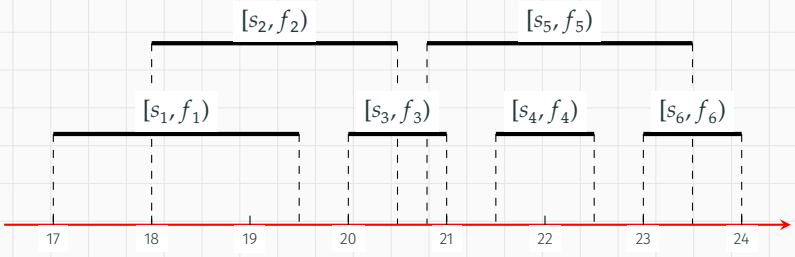


Figure 5: What is S_2?

Theorem 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \qquad \text{for all } k = 0, \ldots, n.$$

Let $[s_m, f_m)$ be the interval that finishes first in S_k.

🍰 Is it true that we always have $[s_m, f_m) = [s_{k+1}, f_{k+1})$?

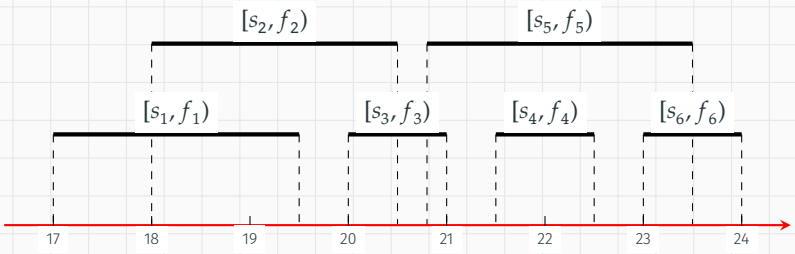


Figure 5: 🍰 What is $[s_m, f_m)$ for S_3?

Theorem 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \qquad \text{for all } k = 0, \dots, n.$$

Let $[s_m, f_m)$ be the interval that finishes first in S_k.

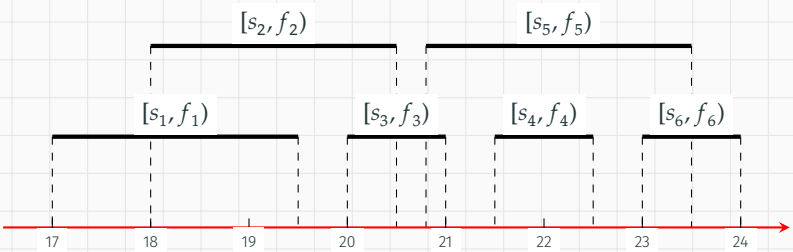


Figure 5: What is $[s_m, f_m)$ for S_4?

Theorem 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \qquad \text{for all } k = 0, \ldots, n.$$

Let $[s_m, f_m)$ be the interval that finishes first in S_k.

Then *one of* the maximum subsets of S_k containing non-overlapping intervals includes $[s_m, f_m)$.

Proof Sketch — Let A_k be the maximum subset of S_k containing non-overlapping intervals.

Replace the earliest-finish interval in A_k by $[s_m, f_m)$.

Then we get another maximum subset of non-overlapping intervals.

A Greedy Algorithm

Recall that the activities are sorted by finish time, with $f_0 = 0$ as a sentinel.

Theorem 16.1 leads to a greedy algorithm —

```
1: Recursive-Selector(s, f, k, n)
2: m ← k + 1
3: while m ≤ n  &  s[m] < f[k] do
4:     m ← m + 1
5: if m ≤ n then
6:     return {[s_m, f_m)} ∪ Recursive-Selector(s, f, m, n)
7: else
8:     return ∅
```

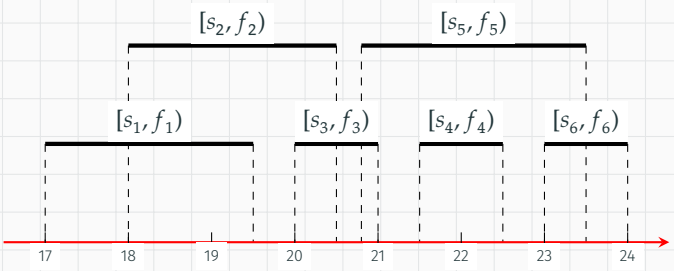
🍰 What is the ⏱️ complexity of Recursive-Selector$(s, f, 0, n)$?

An Iterative Implementation

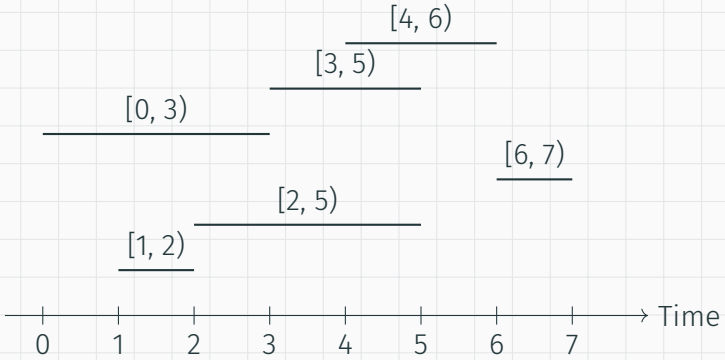
We can implement the same rule iteratively —

Greedy-Selector(s, f)
$n \leftarrow \text{length}(s)$
$A \leftarrow \{[s_1, f_1)\}$
$k \leftarrow 1$
for $m \leftarrow 2$ **to** n **do**
  if $s[m] \geq f[k]$ **then**
    $A \leftarrow A \cup \{[s_m, f_m)\}$
    $k \leftarrow m$
return A

Let us try it out —



What is the maximum number of non-overlapping intervals among the following 6 intervals?



Time for Toilet

Suppose that we also require that there is at least 15 minutes between two screenings that we select.

🍰 How should we modify our greedy algorithm?

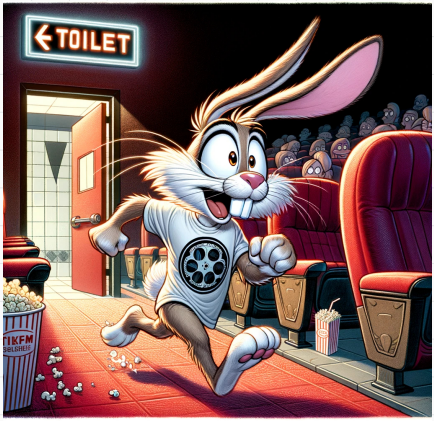


Figure 5: Rushing to 🚽 between films

ITA 16.2 Elements of the Greedy Strategy

General recipe for a greedy algorithm —

1. Formulate the problem so one greedy choice leaves a single subproblem.
2. Prove optimal substructure: greedy choice + optimal subproblem solution is optimal.

General recipe for a greedy algorithm —

1. Formulate the problem so one greedy choice leaves a single subproblem.
2. Prove optimal substructure: greedy choice + optimal subproblem solution is optimal.

Such problems have —

- Greedy-choice Property — A locally optimal (greedy) choice leads towards a globally optimal solution.
- Optimal Substructure — An optimal solution contains optimal solutions to subproblems.

ITA 16.2 Elements of the Greedy Strategy

Greedy Versus Dynamic Programming

A 🐰 who is robbing a store finds n items.

The i-th item is worth v_i 💵 and weighs w_i kg, where v_i and w_i are integers.

The 🐰 can carry at most W kg in his 🎒.

How can the 🐰 maximize his revenue?

Figure 6: A 🐰 robbing a store

In the fractional knapsack problem, we allow the 🐰 to take fractions of the items.

💡 Think of the 0-1 version as robbing a 💎 store, and the fractional version as robbing a 🧀 store.

Figure 7: A 🐰 robbing a store

Greedy Algorithm or Dynamic Programming?

As the example below shows, the algorithm does not always work for the 0-1 knapsack problem.

But it does work for the fractional knapsack problem.

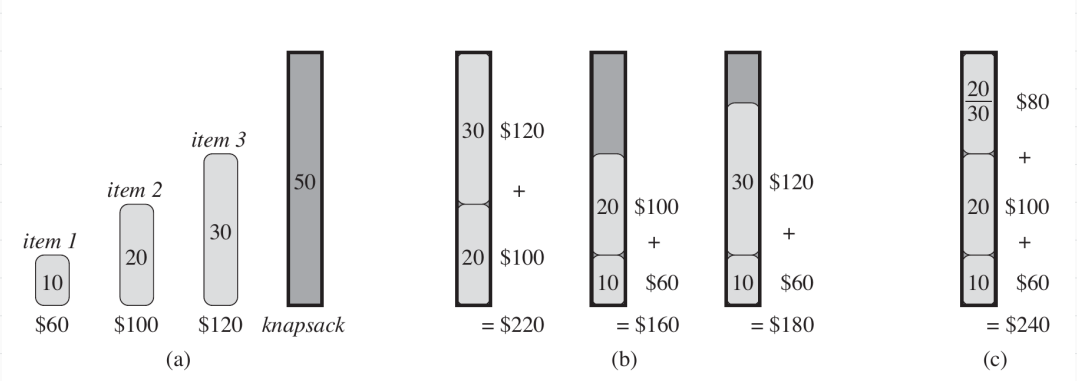


Figure 8: The 0-1 and fractional knapsack problem

Solution for the 0-1 Knapsack Problem

Let us use dynamic programming to solve the 0-1 knapsack problem —

Knapsack(v, w, n, W)
Initialize $K[0 \ldots n][0 \ldots W]$
for $i \leftarrow 0$ to n do
    for $j \leftarrow 0$ to W do
        if $i = 0$ or $j = 0$ then
            $K[i][j] \leftarrow$ _____
        else if $w[i] \leq j$ then
            $K[i][j] \leftarrow$ _____
        else
            $K[i][j] \leftarrow$ _____
return $K[n][W]$

Solution for the Fractional Knapsack Problem

But for the fractional knapsack problem, we can use a 🤑 algorithm —

```
Knapsack(v, w, n, W)
V ← 0                                   ▷ Total value initialized to 0
SortByRatio(v, w)                       ▷ Sort by value-to-weight ratio in decreasing order
for i ← 1 to n do
    if w[i] ≤ W then
        V ← V + v[i]                    ▷ Take the whole item
        W ← W − w[i]                    ▷ Decrease available weight
    else
        V ← V + _____________________   ▷ Take fraction of the item
        break
return V                                ▷ Return the total value
```

Exercise

How should we modify Knapsack (fractional version), if item i has only $u_i \leq w_i$ weight available?

💡 Treat item i as having weight u_i and value $v_i \cdot (u_i/w_i)$, then run greedy as usual.

```
Knapsack(v, w, n, W)
V ← 0
SortByRatio(v, w)
for i ← 1 to n do
    if w[i] ≤ W then
        V ← V + v[i]
        W ← W − w[i]
    else
        V ← V + v[i] · W/w[i]        ▷ Take fraction W/w[i]
        break
return V
```

```
Knapsack-Cap(v, w, u, n, W)
for i ← 1 to n do
    v[i] ← v[i] · _____              ▷ Scale value
    w[i] ← _____                     ▷ Cap weight
return Knapsack(v, w, n, W)
```

Exercise

How should we modify Knapsack (fractional version), if we may take from at most m items (fractions allowed)?

💡 Sort by v_i/w_i, but stop after taking from m items.

Knapsack(v, w, n, W)
$V \leftarrow 0$
SortByRatio(v, w)
for $i \leftarrow 1$ **to** n **do**
  if $w[i] \leq W$ **then**
    $V \leftarrow V + v[i]$
    $W \leftarrow W - w[i]$
  else
    $V \leftarrow V + v[i] \cdot W/w[i]$ ▷ Take fraction $W/w[i]$
    break
return V

Knapsack-At-Most-M(v, w, n, W, m)
$V \leftarrow 0$
$used \leftarrow 0$
SortByRatio(v, w)
for $i \leftarrow 1$ **to** n **do**
  if _____ **then**
    return V ▷ Stop
  $x \leftarrow$ _____
  $V \leftarrow$ _____
  $W \leftarrow W - x$
  $used \leftarrow used + 1$
return V

💬 Greedy Algorithm for the Knapsack Problem?

We have seen that being 🤑 in the fractional version of the knapsack problem is 🆗.

💬 Can you think of some other variations of the knapsack problem for which being 🤑 is *also* 🆗?

Figure 9: It is 🆗 to be 🤑 when the 🐉 is 😴

🪙 The Change-Making Problem

We have infinitely many 🪙 with denominations in $D = \{d_1, \ldots, d_n\}$ where

$$d_1 < d_2 \cdots < d_n.$$

We want to make a specified amount A with these coins while minimizing the number of 🪙.

🍰 Example — $A = 89$ and

$$D = \{2, 5, 11, 23, 50\},$$

can a greedy algorithm work?

The Change-Making Problem

We have infinitely many 🪙 with denominations in $D = \{d_1, \ldots, d_n\}$ where

$$d_1 < d_2 \cdots < d_n.$$

We want to make a specified amount A with these coins while minimizing the number of 🪙.

🍰 If we take the largest coin first at each step,

$$89 = \_\_\_\_\_$$

This uses _____ coins (and is optimal). (🍰 Why?)

The Change-Making Problem

We have infinitely many 🪙 with denominations in $D = \{d_1, \ldots, d_n\}$ where

$$d_1 < d_2 \cdots < d_n.$$

We want to make a specified amount A with these coins while minimizing the number of 🪙.

🍰 What if $A = 51$ with the same denominations?

$$51 = \_\_\_\_\_$$

Does greedy still work?

What are your main takeaways today? Any questions?