Lecture 22: Thinking Recursively

Comp 102

Fakhir Shaheen

Forman Christian University

The Towers of Brahma

1883: A Puzzle in a Box

- A French maths teacher, **Édouard Lucas**, sells a toy puzzle.



1883: A Puzzle in a Box

- A French maths teacher, **Édouard Lucas**, sells a toy puzzle.
- On the box is a story.



1883: A Puzzle in a Box

- A French maths teacher, **Édouard Lucas**, sells a toy puzzle.
- On the box is a story.



In a temple there are three diamond needles and **64 golden disks**. Priests move the disks, one at a time, day and night. When the last disk is moved, **the world ends**.

The Rules

1. Move **one** disk at a time.



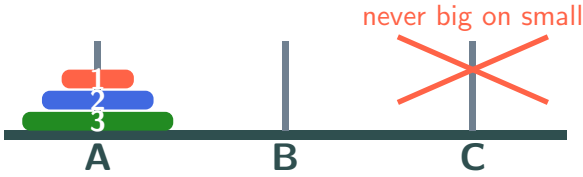
The Rules

1. Move **one** disk at a time.
2. Take only the **top** disk.



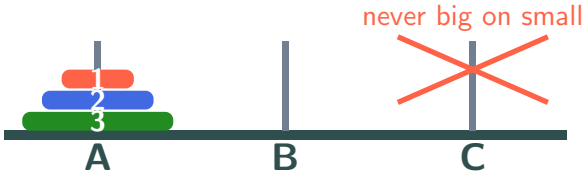
The Rules

1. Move **one** disk at a time.
2. Take only the **top** disk.
3. **Never** put a big disk on a smaller one.



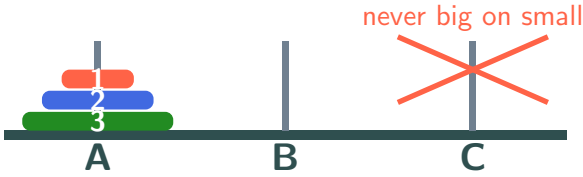
The Rules

1. Move **one** disk at a time.
2. Take only the **top** disk.
3. **Never** put a big disk on a smaller one.
4. Goal: the whole tower from **A** to **C**.



The Rules

1. Move **one** disk at a time.
2. Take only the **top** disk.
3. **Never** put a big disk on a smaller one.
4. Goal: the whole tower from **A** to **C**.



The priests make **one move a second**. How long until the world ends?

Write a guess: a day? 100 years? longer? (30 s)

Today

1. Lecture 21: **how** recursion runs. Today: how to **design** one.

Today

1. Lecture 21: **how** recursion runs. Today: how to **design** one.
2. A **recipe** with four questions, and practice on numbers, strings and lists.

Today

1. Lecture 21: **how** recursion runs. Today: how to **design** one.
2. A **recipe** with four questions, and practice on numbers, strings and lists.
3. Back to the temple: the priests' answer.

A Recipe

WISE: Four Questions

W — **W**hat is the easiest case? base case

WISE: Four Questions

W — **W**hat is the easiest case? base case

I — **I**f not easy, how do I make it smaller?

WISE: Four Questions

W — **W**hat is the easiest case? base case

I — **I**f not easy, how do I make it smaller?

S — **S**olve the smaller job (trust it!)

WISE: Four Questions

W — **W**hat is the easiest case? base case

I — **I**f not easy, how do I make it smaller?

S — **S**olve the smaller job (trust it!)

E — **E**xtend: use it to answer the big job

Design One Together: Add the Digits

`digit_sum(472)` $\rightarrow 4 + 7 + 2 = \mathbf{13}$

472

Design One Together: Add the Digits

`digit_sum(472)` $\rightarrow 4 + 7 + 2 = \mathbf{13}$

W One digit, like 7? The answer is 7.

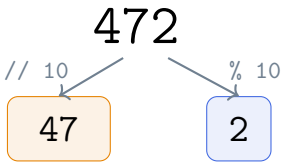
472

Design One Together: Add the Digits

`digit_sum(472)` → 4 + 7 + 2 = **13**

W One digit, like 7? The answer is 7.

I Cut off the last digit.
`472 // 10` is 47, `472 % 10` is 2.



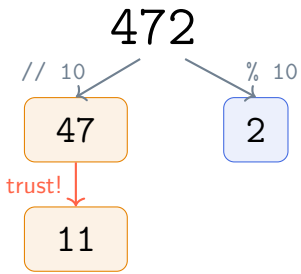
Design One Together: Add the Digits

`digit_sum(472)` → 4 + 7 + 2 = **13**

W One digit, like 7? The answer is 7.

I Cut off the last digit.
`472 // 10` is 47, `472 % 10` is 2.

S Trust it: `digit_sum(47)` is 11.



Design One Together: Add the Digits

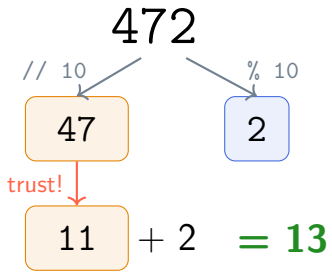
`digit_sum(472)` → 4 + 7 + 2 = **13**

W One digit, like 7? The answer is 7.

I Cut off the last digit.
`472 // 10` is 47, `472 % 10` is 2.

S Trust it: `digit_sum(47)` is 11.

E Add the cut digit: 11 + 2 = 13.



Add the Digits, in Python

```
1  def digit_sum(n):
2      if n < 10:
3          return n
4      rest = n // 10
5      last = n % 10
6      return digit_sum(rest) + last
7
8  print(digit_sum(472))
```

Add the Digits, in Python

```
1  def digit_sum(n):
2      if n < 10:
3          return n
4      rest = n // 10
5      last = n % 10
6      return digit_sum(rest) + last
7
8  print(digit_sum(472))
```

W: one digit: give it back.

I: `rest = n // 10`

S: `digit_sum(rest)`

E: add `last = n % 10`.

Add the Digits, in Python

```
1  def digit_sum(n):
2      if n < 10:
3          return n
4      rest = n // 10
5      last = n % 10
6      return digit_sum(rest) + last
7
8  print(digit_sum(472))
```

digit_sum(472)

W: one digit: give it back.

I: `rest = n // 10`

S: `digit_sum(rest)`

E: add `last = n % 10`.

Add the Digits, in Python

```
1  def digit_sum(n):
2      if n < 10:
3          return n
4      rest = n // 10
5      last = n % 10
6      return digit_sum(rest) + last
7
8  print(digit_sum(472))
```

```
digit_sum(472)
  = digit_sum(47) + 2
```

W: one digit: give it back.

I: `rest = n // 10`

S: `digit_sum(rest)`

E: add `last = n % 10`.

Add the Digits, in Python

```
1  def digit_sum(n):
2      if n < 10:
3          return n
4      rest = n // 10
5      last = n % 10
6      return digit_sum(rest) + last
7
8  print(digit_sum(472))
```

```
digit_sum(472)
  = digit_sum(47) + 2
    = digit_sum(4) + 7 + 2
```

W: one digit: give it back.

I: `rest = n // 10`

S: `digit_sum(rest)`

E: add `last = n % 10`.

Add the Digits, in Python

```
1  def digit_sum(n):
2      if n < 10:
3          return n
4      rest = n // 10
5      last = n % 10
6      return digit_sum(rest) + last
7
8  print(digit_sum(472))
```

```
digit_sum(472)
  = digit_sum(47) + 2
    = digit_sum(4) + 7 + 2
      = 4 + 7 + 2 = 13
```

W: one digit: give it back.

I: `rest = n // 10`

S: `digit_sum(rest)`

E: add `last = n % 10`.

Add the Digits, in Python

```
1  def digit_sum(n):
2      if n < 10:
3          return n
4      rest = n // 10
5      last = n % 10
6      return digit_sum(rest) + last
7
8  print(digit_sum(472))
```

```
digit_sum(472)
  = digit_sum(47) + 2
    = digit_sum(4) + 7 + 2
      = 4 + 7 + 2 = 13
```

W: one digit: give it back.

I: `rest = n // 10`

S: `digit_sum(rest)`

E: add `last = n % 10`.

Lecture 21 used `n - 1`. Here: `n // 10`.

You choose how the job gets smaller.

You Try! *(answer the four questions on paper, 3 min)*

`reverse("hello")` → `"olleh"`

`reverse("cat")` → `"tac"`

W Which strings are easy to reverse?

I How do you make a string smaller?

S Pretend `reverse` of the smaller string already works.

E How do you get the whole answer from it?

Think: Write Small Examples

Stuck? Write the answers for small inputs, each one letter longer.

```
reverse("o")     →
reverse("lo")    →
reverse("llo")   →
reverse("ello")  →
reverse("hello") →
```

Think: Write Small Examples

Stuck? Write the answers for small inputs, each one letter longer.

```
reverse("o")     →  "o"
reverse("lo")    →  "ol"
reverse("llo")   →  "oll"
reverse("ello")  →  "olle"
reverse("hello") →  "olleh"
```

Think: Write Small Examples

Stuck? Write the answers for small inputs, each one letter longer.

```
reverse("o")     →  "o"
reverse("lo")    →  "ol"
reverse("llo")   →  "oll"
reverse("ello")  →  "olle"
reverse("hello") →  "olleh"
```

Orange: the answer **one row up**. **Blue**: the first letter, now at the end.

Think: Write Small Examples

Stuck? Write the answers for small inputs, each one letter longer.

```
reverse("o")     →  "o"
reverse("lo")    →  "ol"
reverse("llo")   →  "oll"
reverse("ello")  →  "olle"
reverse("hello") →  "olleh"
```

Orange: the answer **one row up**. **Blue**: the first letter, now at the end.

```
reverse(s) = reverse(s[1:]) + s[0]
```

Reverse, in Python

```
1 def reverse(s):
2     if len(s) <= 1:
3         return s
4     return reverse(s[1:]) + s[0]
5
6 print(reverse("cat"))
```

Reverse, in Python

```
1 def reverse(s):
2     if len(s) <= 1:
3         return s
4     return reverse(s[1:]) + s[0]
5
6 print(reverse("cat"))
```

W: "" or one letter: give it back.

I: drop the first letter.

S: `reverse(s[1:])`

E: put `s[0]` at the **end**.

Reverse, in Python

```
1 def reverse(s):
2     if len(s) <= 1:
3         return s
4     return reverse(s[1:]) + s[0]
5
6 print(reverse("cat"))
```

```
reverse("cat")
```

W: "" or one letter: give it back.

I: drop the first letter.

S: `reverse(s[1:])`

E: put `s[0]` at the **end**.

Reverse, in Python

```
1 def reverse(s):
2     if len(s) <= 1:
3         return s
4     return reverse(s[1:]) + s[0]
5
6 print(reverse("cat"))
```

```
reverse("cat")
  = reverse("at") + "c"
```

W: "" or one letter: give it back.

I: drop the first letter.

S: `reverse(s[1:])`

E: put `s[0]` at the **end**.

Reverse, in Python

```
1  def reverse(s):
2      if len(s) <= 1:
3          return s
4      return reverse(s[1:]) + s[0]
5
6  print(reverse("cat"))
```

```
reverse("cat")
  = reverse("at") + "c"
    = reverse("t") + "a" + "c"
```

W: "" or one letter: give it back.

I: drop the first letter.

S: `reverse(s[1:])`

E: put `s[0]` at the **end**.

Reverse, in Python

```
1  def reverse(s):
2      if len(s) <= 1:
3          return s
4      return reverse(s[1:]) + s[0]
5
6  print(reverse("cat"))
```

```
reverse("cat")
  = reverse("at") + "c"
    = reverse("t") + "a" + "c"
      = "t" + "a" + "c" = "tac"
```

W: "" or one letter: give it back.

I: drop the first letter.

S: `reverse(s[1:])`

E: put `s[0]` at the **end**.

Strings and Lists

You Try! *(WISE on paper, 3 min)*

Lecture 9 checked a palindrome with a trick:

```
return s == s[::-1]
```

Now do it **without** `[::-1]`. Write `is_palindrome(s)`.

`"racecar"` → `True`

`"hello"` → `False`

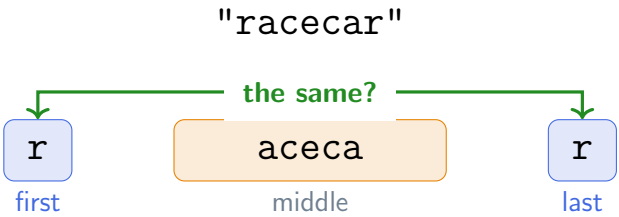
`"abba"` → `True`

Hint: this time the smaller job does not just drop the **first** letter...

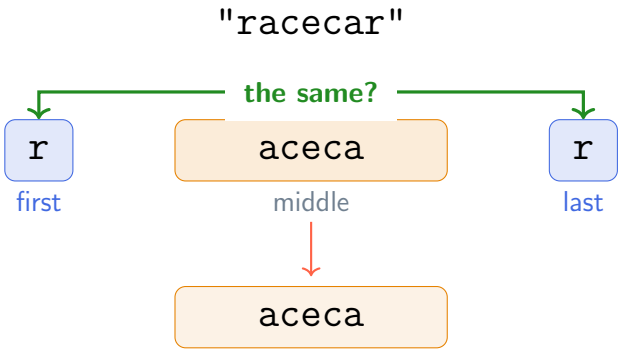
Think: Shrink From Both Ends

"racecar"

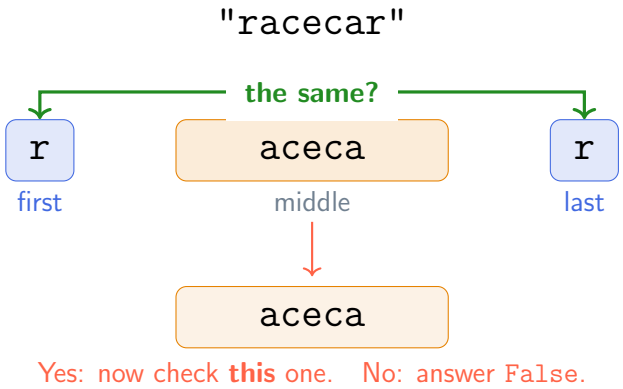
Think: Shrink From Both Ends



Think: Shrink From Both Ends



Think: Shrink From Both Ends



The middle is `s[1:-1]`: drop the first and the last letter.

Palindrome, in Python

```
1 def is_palindrome(s):
2     if len(s) <= 1:
3         return True
4     if s[0] != s[-1]:
5         return False
6     return is_palindrome(s[1:-1])
```

Palindrome, in Python

```
1 def is_palindrome(s):
2     if len(s) <= 1:
3         return True
4     if s[0] != s[-1]:
5         return False
6     return is_palindrome(s[1:-1])
```

W: 0 or 1 letters: `True`.

I: drop both ends.

S: check the middle, `s[1:-1]`.

E: only if the ends **match**. If not: `False` at once.

Palindrome, in Python

```
1 def is_palindrome(s):
2     if len(s) <= 1:
3         return True
4     if s[0] != s[-1]:
5         return False
6     return is_palindrome(s[1:-1])
```

```
is_palindrome("racecar")
```

W: 0 or 1 letters: `True`.

I: drop both ends.

S: check the middle, `s[1:-1]`.

E: only if the ends **match**. If not: `False` at once.

Palindrome, in Python

```
1 def is_palindrome(s):
2     if len(s) <= 1:
3         return True
4     if s[0] != s[-1]:
5         return False
6     return is_palindrome(s[1:-1])
```

```
is_palindrome("racecar")
  r = r → is_palindrome("aceca")
```

W: 0 or 1 letters: `True`.

I: drop both ends.

S: check the middle, `s[1:-1]`.

E: only if the ends **match**. If not: `False` at once.

Palindrome, in Python

```
1  def is_palindrome(s):
2      if len(s) <= 1:
3          return True
4      if s[0] != s[-1]:
5          return False
6      return is_palindrome(s[1:-1])
```

```
is_palindrome("racecar")
  r = r → is_palindrome("aceca")
    a = a → is_palindrome("cec")
```

W: 0 or 1 letters: `True`.

I: drop both ends.

S: check the middle, `s[1:-1]`.

E: only if the ends **match**. If not: `False` at once.

Palindrome, in Python

```
1  def is_palindrome(s):
2      if len(s) <= 1:
3          return True
4      if s[0] != s[-1]:
5          return False
6      return is_palindrome(s[1:-1])
```

```
is_palindrome("racecar")
  r = r → is_palindrome("aceca")
    a = a → is_palindrome("cec")
      c = c → is_palindrome("e")
```

W: 0 or 1 letters: `True`.

I: drop both ends.

S: check the middle, `s[1:-1]`.

E: only if the ends **match**. If not: `False` at once.

Palindrome, in Python

```
1  def is_palindrome(s):
2      if len(s) <= 1:
3          return True
4      if s[0] != s[-1]:
5          return False
6      return is_palindrome(s[1:-1])
```

```
is_palindrome("racecar")
  r = r → is_palindrome("aceca")
    a = a → is_palindrome("cec")
      c = c → is_palindrome("e")
        one letter → True
```

W: 0 or 1 letters: `True`.

I: drop both ends.

S: check the middle, `s[1:-1]`.

E: only if the ends **match**. If not: `False` at once.

Read & Explain! *(one sentence, 1 min)*

What does `mystery` do?

```
1 def mystery(s):
2     if s == "":
3         return 0
4     rest = mystery(s[1:])
5     if s[0] in "aeiou":
6         return 1 + rest
7     return rest
```

Read & Explain! *(one sentence, 1 min)*

What does `mystery` do?

```
1 def mystery(s):
2     if s == "":
3         return 0
4     rest = mystery(s[1:])
5     if s[0] in "aeiou":
6         return 1 + rest
7     return rest
```

It **counts the vowels** in `s`.

`mystery("banana")` is 3.

Read & Explain! *(one sentence, 1 min)*

What does `mystery` do?

```
1 def mystery(s):
2     if s == "":
3         return 0
4     rest = mystery(s[1:])
5     if s[0] in "aeiou":
6         return 1 + rest
7     return rest
```

It **counts the vowels** in `s`.

`mystery("banana")` is 3.

W: `""` has 0 vowels.
E: add 1 if the first letter is a vowel.

You Try! *(WISE on paper, 3 min)*

Find the biggest number in a list, **without** `max`.

`find_max([3, 7, 2, 9, 1])` → `9`

The same recipe works on a **list**:

W Which list has an easy answer?

I How do you make a list smaller?

S Pretend `find_max` of the smaller list already works.

E You have the biggest of the rest. What now?

Find Max, in Python

```
1 def find_max(L):
2     if len(L) == 1:
3         return L[0]
4     best_of_rest = find_max(L[1:])
5     if L[0] > best_of_rest:
6         return L[0]
7     return best_of_rest
```

Find Max, in Python

```
1 def find_max(L):
2     if len(L) == 1:
3         return L[0]
4     best_of_rest = find_max(L[1:])
5     if L[0] > best_of_rest:
6         return L[0]
7     return best_of_rest
```

W: one number: it is the biggest.

I: drop the first number.

S: `find_max(L[1:])`

E: me, or the best of the rest?

Find Max, in Python

```
1 def find_max(L):
2     if len(L) == 1:
3         return L[0]
4     best_of_rest = find_max(L[1:])
5     if L[0] > best_of_rest:
6         return L[0]
7     return best_of_rest
```

```
find_max([3, 7, 2])
```

W: one number: it is the biggest.

I: drop the first number.

S: `find_max(L[1:])`

E: me, or the best of the rest?

Find Max, in Python

```
1 def find_max(L):
2     if len(L) == 1:
3         return L[0]
4     best_of_rest = find_max(L[1:])
5     if L[0] > best_of_rest:
6         return L[0]
7     return best_of_rest
```

```
find_max([3, 7, 2])
  find_max([7, 2])
```

W: one number: it is the biggest.

I: drop the first number.

S: `find_max(L[1:])`

E: me, or the best of the rest?

Find Max, in Python

```
1 def find_max(L):
2     if len(L) == 1:
3         return L[0]
4     best_of_rest = find_max(L[1:])
5     if L[0] > best_of_rest:
6         return L[0]
7     return best_of_rest
```

```
find_max([3, 7, 2])
  find_max([7, 2])
    find_max([2])
```

W: one number: it is the biggest.

I: drop the first number.

S: `find_max(L[1:])`

E: me, or the best of the rest?

Find Max, in Python

```
1  def find_max(L):
2      if len(L) == 1:
3          return L[0]
4      best_of_rest = find_max(L[1:])
5      if L[0] > best_of_rest:
6          return L[0]
7      return best_of_rest
```

```
find_max([3, 7, 2])
  find_max([7, 2])
    find_max([2])
      one number:  2
```

W: one number: it is the biggest.

I: drop the first number.

S: `find_max(L[1:])`

E: me, or the best of the rest?

Find Max, in Python

```
1  def find_max(L):
2      if len(L) == 1:
3          return L[0]
4      best_of_rest = find_max(L[1:])
5      if L[0] > best_of_rest:
6          return L[0]
7      return best_of_rest
```

```
find_max([3, 7, 2])
  find_max([7, 2])
    find_max([2])
      one number:  2

  7 vs 2 → 7
```



W: one number: it is the biggest.

I: drop the first number.

S: `find_max(L[1:])`

E: me, or the best of the rest?

Find Max, in Python

```
1  def find_max(L):
2      if len(L) == 1:
3          return L[0]
4      best_of_rest = find_max(L[1:])
5      if L[0] > best_of_rest:
6          return L[0]
7      return best_of_rest
```

```
find_max([3, 7, 2])
  find_max([7, 2])
    find_max([2])
      one number:  2

  7 vs 2 → 7
3 vs 7 → 7
```

up

W: one number: it is the biggest.

I: drop the first number.

S: `find_max(L[1:])`

E: me, or the best of the rest?

Find Max, in Python

```
1  def find_max(L):
2      if len(L) == 1:
3          return L[0]
4      best_of_rest = find_max(L[1:])
5      if L[0] > best_of_rest:
6          return L[0]
7      return best_of_rest
```

W: one number: it is the biggest.

I: drop the first number.

S: `find_max(L[1:])`

E: me, or the best of the rest?

```
find_max([3, 7, 2])
  find_max([7, 2])
    find_max([2])
      one number:  2

  7 vs 2 → 7
3 vs 7 → 7

= 7
```



Many Ways to Make a Job Smaller

the job	the smaller copy	how
`sum_to(5)`	`sum_to(4)`	`n - 1` (Lecture 21)

Many Ways to Make a Job Smaller

the job	the smaller copy	how
`sum_to(5)`	`sum_to(4)`	`n - 1` (Lecture 21)
`digit_sum(472)`	`digit_sum(47)`	`n // 10`

Many Ways to Make a Job Smaller

the job	the smaller copy	how
`sum_to(5)`	`sum_to(4)`	`n - 1` (Lecture 21)
`digit_sum(472)`	`digit_sum(47)`	`n // 10`
`reverse("cat")`	`reverse("at")`	`s[1:]`

Many Ways to Make a Job Smaller

the job	the smaller copy	how
`sum_to(5)`	`sum_to(4)`	`n - 1` (Lecture 21)
`digit_sum(472)`	`digit_sum(47)`	`n // 10`
`reverse("cat")`	`reverse("at")`	`s[1:]`
`is_palindrome("abba")`	`is_palindrome("bb")`	`s[1:-1]`

Many Ways to Make a Job Smaller

the job	the smaller copy	how
`sum_to(5)`	`sum_to(4)`	`n - 1` (Lecture 21)
`digit_sum(472)`	`digit_sum(47)`	`n // 10`
`reverse("cat")`	`reverse("at")`	`s[1:]`
`is_palindrome("abba")`	`is_palindrome("bb")`	`s[1:-1]`
`find_max([3, 7, 2])`	`find_max([7, 2])`	`L[1:]`

Many Ways to Make a Job Smaller

the job	the smaller copy	how
`sum_to(5)`	`sum_to(4)`	`n - 1` (Lecture 21)
`digit_sum(472)`	`digit_sum(47)`	`n // 10`
`reverse("cat")`	`reverse("at")`	`s[1:]`
`is_palindrome("abba")`	`is_palindrome("bb")`	`s[1:-1]`
`find_max([3, 7, 2])`	`find_max([7, 2])`	`L[1:]`

The art: spot the **smaller copy** inside the job.

The base case (**W**) is where the shrinking **stops**.

Back to the Temple

Three Disks, Seven Moves

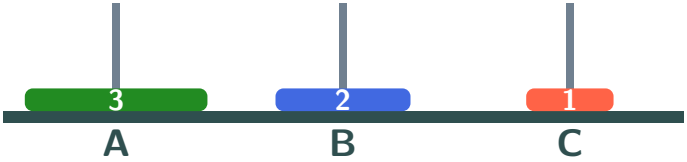


Three Disks, Seven Moves



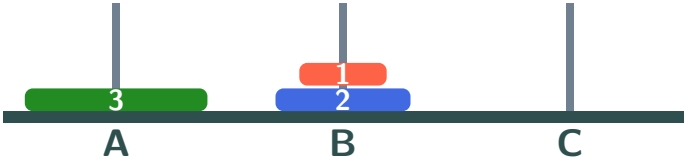
1. disk 1 from A to C

Three Disks, Seven Moves



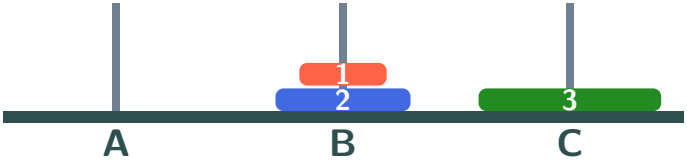
1. disk 1 from A to C
2. disk 2 from A to B

Three Disks, Seven Moves



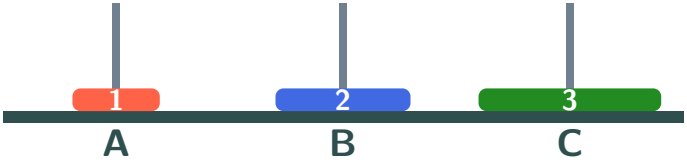
1. disk 1 from A to C
2. disk 2 from A to B
3. disk 1 from C to B

Three Disks, Seven Moves



1. disk 1 from A to C
2. disk 2 from A to B
3. disk 1 from C to B
4. **disk 3 from A to C**

Three Disks, Seven Moves



1. disk 1 from A to C
2. disk 2 from A to B
3. disk 1 from C to B
4. **disk 3 from A to C**
5. disk 1 from B to A

Three Disks, Seven Moves



1. disk 1 from A to C
2. disk 2 from A to B
3. disk 1 from C to B
4. **disk 3 from A to C**
5. disk 1 from B to A
6. disk 2 from B to C

Three Disks, Seven Moves



1. disk 1 from A to C
2. disk 2 from A to B
3. disk 1 from C to B
4. **disk 3 from A to C**
5. disk 1 from B to A
6. disk 2 from B to C
7. disk 1 from A to C

Three Disks, Seven Moves



1. disk 1 from A to C
2. disk 2 from A to B
3. disk 1 from C to B
4. **disk 3 from A to C**
5. disk 1 from B to A
6. disk 2 from B to C
7. disk 1 from A to C

Moves 1–3 and 5–7 each move a tower of **2**.

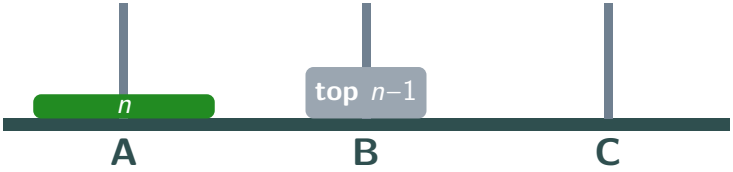
The Smaller Copy Inside

To move n disks from **A** to **C**:



The Smaller Copy Inside

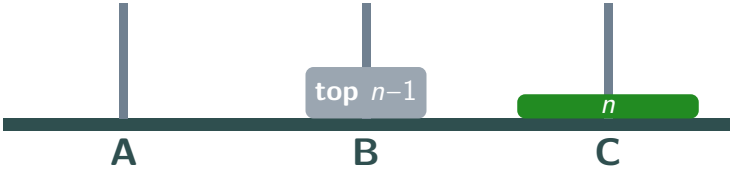
To move n disks from **A** to **C**:



1. Move the top $n-1$ from A to **B**. **a smaller tower: recursion!**

The Smaller Copy Inside

To move n disks from **A** to **C**:



1. Move the top $n-1$ from A to **B**. **a smaller tower: recursion!**
2. Move disk n, the biggest, from A to C. **one move**

The Smaller Copy Inside

To move n disks from **A** to **C**:



1. Move the top $n-1$ from A to **B**. **a smaller tower: recursion!**
2. Move disk n, the biggest, from A to C. **one move**
3. Move the top $n-1$ from B to **C**. **recursion again!**

The Smaller Copy Inside

To move n disks from **A** to **C**:



1. Move the top $n-1$ from A to **B**. **a smaller tower: recursion!**
2. Move disk n, the biggest, from A to C. **one move**
3. Move the top $n-1$ from B to **C**. **recursion again!**

W: **0** disks: nothing to do.

Hanoi, in Python

```
1 def hanoi(n, start, spare, goal):
2     if n == 0:
3         return
4     hanoi(n - 1, start, goal, spare)
5     print("Move disk", n, "from", start, "to", goal)
6     hanoi(n - 1, spare, start, goal)
7
8 hanoi(3, "A", "B", "C")
```

Hanoi, in Python

```
1 def hanoi(n, start, spare, goal):
2     if n == 0:
3         return
4     hanoi(n - 1, start, goal, spare)
5     print("Move disk", n, "from", start, "to", goal)
6     hanoi(n - 1, spare, start, goal)
7
8 hanoi(3, "A", "B", "C")
```

- Line 4: the top $n-1$ go to the **spare** needle. So the goal is the spare, for now.

Hanoi, in Python

```
1 def hanoi(n, start, spare, goal):
2     if n == 0:
3         return
4     hanoi(n - 1, start, goal, spare)
5     print("Move disk", n, "from", start, "to", goal)
6     hanoi(n - 1, spare, start, goal)
7
8 hanoi(3, "A", "B", "C")
```

- Line 4: the top $n-1$ go to the **spare** needle. So the goal is the spare, for now.
- Line 6: then from the spare to the **goal**.

Hanoi, in Python

```
1 def hanoi(n, start, spare, goal):
2     if n == 0:
3         return
4     hanoi(n - 1, start, goal, spare)
5     print("Move disk", n, "from", start, "to", goal)
6     hanoi(n - 1, spare, start, goal)
7
8 hanoi(3, "A", "B", "C")
```

- Line 4: the top $n-1$ go to the **spare** needle. So the goal is the spare, for now.
- Line 6: then from the spare to the **goal**.
- The function calls itself **twice**.

You Try! *(WISE on paper, 3 min)*

Write `moves(n)`: how many moves for a tower of n disks?

`moves(1)` → 1 `moves(2)` → 3 `moves(3)` → 7

W How many moves for **0** disks?

I What is the smaller tower?

S Pretend `moves(n - 1)` already works.

E The picture: a tower of $n-1$, then **one** disk, then a tower of $n-1$ again.

Moves, in Python

```
1 def moves(n):
2     if n == 0:
3         return 0
4     return 2 * moves(n - 1) + 1
```

disks n	1	2	3	4	5	6	10	20
moves	1	3	7					

Moves, in Python

```
1 def moves(n):
2     if n == 0:
3         return 0
4     return 2 * moves(n - 1) + 1
```

W: 0 disks, 0 moves.

I: a tower of $n-1$.

S: `moves(n - 1)`

E: two of those, plus one.

disks n	1	2	3	4	5	6	10	20
moves	1	3	7					

Moves, in Python

```
1 def moves(n):
2     if n == 0:
3         return 0
4     return 2 * moves(n - 1) + 1
```

W: 0 disks, 0 moves.

I: a tower of $n-1$.

S: `moves(n - 1)`

E: two of those, plus one.

disks n	1	2	3	4	5	6	10	20
moves	1	3	7	15	31	63		

Moves, in Python

```
1 def moves(n):
2     if n == 0:
3         return 0
4     return 2 * moves(n - 1) + 1
```

W: 0 disks, 0 moves.

I: a tower of $n-1$.

S: `moves(n - 1)`

E: two of those, plus one.

disks n	1	2	3	4	5	6	10	20
moves	1	3	7	15	31	63	1,023	1,048,575

Moves, in Python

```
1  def moves(n):
2      if n == 0:
3          return 0
4      return 2 * moves(n - 1) + 1
```

W: 0 disks, 0 moves.

I: a tower of $n - 1$.

S: `moves(n - 1)`

E: two of those, plus one.

disks n	1	2	3	4	5	6	10	20
moves	1	3	7	15	31	63	1,023	1,048,575

- Moves for n disks: $2^n - 1$. Hint from Lecture 18: $2^{10} = 1{,}024$.

Moves, in Python

```
1  def moves(n):
2      if n == 0:
3          return 0
4      return 2 * moves(n - 1) + 1
```

W: 0 disks, 0 moves.

I: a tower of $n - 1$.

S: `moves(n - 1)`

E: two of those, plus one.

disks n	1	2	3	4	5	6	10	20
moves	1	3	7	15	31	63	1,023	1,048,575

- Moves for n disks: $2^n - 1$. Hint from Lecture 18: $2^{10} = 1{,}024$.
- One more disk → twice the moves. That is $O(2^n)$, the exponential family.

When Does the World End?

- `moves(64)` is $2^{64} - 1$.
  $= 18{,}446{,}744{,}073{,}709{,}551{,}615$

When Does the World End?

- `moves(64)` is $2^{64} - 1$.
  $= 18{,}446{,}744{,}073{,}709{,}551{,}615$
- One move a second: about **585 billion years**.

When Does the World End?

- `moves(64)` is $2^{64} - 1$.
  $= 18{,}446{,}744{,}073{,}709{,}551{,}615$
- One move a second: about **585 billion years**.
- The universe is about **13.8 billion** years old.

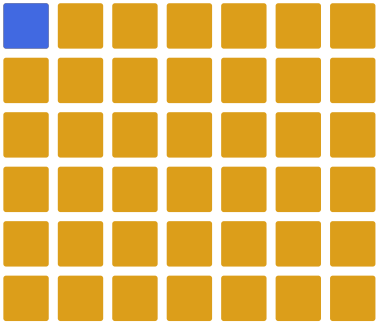
one box = the age of the universe



When Does the World End?

- `moves(64)` is $2^{64} - 1$.
  $= 18{,}446{,}744{,}073{,}709{,}551{,}615$
- One move a second: about **585 billion years**.
- The universe is about **13.8 billion** years old.
- So: about **42** times the age of the universe.

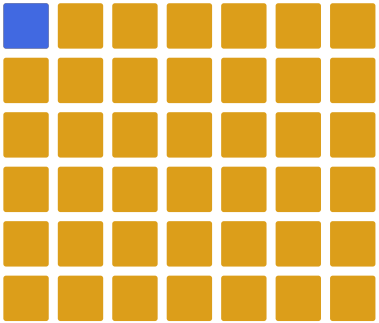
one box = the age of the universe



When Does the World End?

- `moves(64)` is $2^{64} - 1$.
  $= 18{,}446{,}744{,}073{,}709{,}551{,}615$
- One move a second: about **585 billion years**.
- The universe is about **13.8 billion** years old.
- So: about **42** times the age of the universe.

one box = the age of the universe



A 6-line program. A run time the world will never see.

Lists Inside Lists

Folders Inside Folders, Again

A list can hold numbers **and other lists**, as deep as you like. Add up every number:

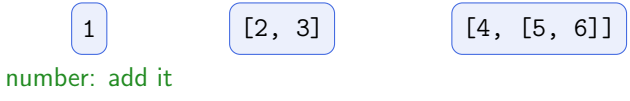
```
nested_sum([1, [2, 3], [4, [5, 6]]])  →  21
```



Folders Inside Folders, Again

A list can hold numbers **and other lists**, as deep as you like. Add up every number:

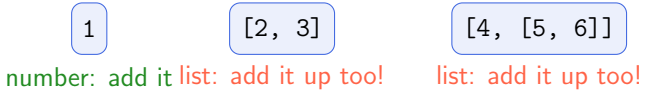
```
nested_sum([1, [2, 3], [4, [5, 6]]])  →  21
```



Folders Inside Folders, Again

A list can hold numbers **and other lists**, as deep as you like. Add up every number:

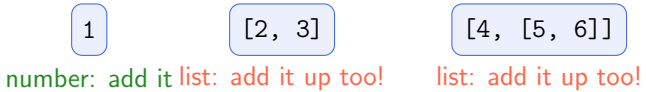
```
nested_sum([1, [2, 3], [4, [5, 6]]])  →  21
```



Folders Inside Folders, Again

A list can hold numbers **and other lists**, as deep as you like. Add up every number:

```
nested_sum([1, [2, 3], [4, [5, 6]]])  →  21
```



How many loops inside loops would you need? You **do not know**. Recursion does not care.

Nested Sum, in Python

```
1  def nested_sum(L):
2      if L == []:
3          return 0
4      first = L[0]
5      rest = nested_sum(L[1:])
6      if type(first) == list:
7          return nested_sum(first) + rest
8      return first + rest
```

Nested Sum, in Python

```
1  def nested_sum(L):
2      if L == []:
3          return 0
4      first = L[0]
5      rest = nested_sum(L[1:])
6      if type(first) == list:
7          return nested_sum(first) + rest
8      return first + rest
```

W: [] adds up to 0.

I: first item + the rest.

S: add up the rest. If the first item is a list, add it up too.

E: add them with +.

Nested Sum, in Python

```
1 def nested_sum(L):
2     if L == []:
3         return 0
4     first = L[0]
5     rest = nested_sum(L[1:])
6     if type(first) == list:
7         return nested_sum(first) + rest
8     return first + rest
```

W: [] adds up to 0.

I: first item + the rest.

S: add up the rest. If the first item is a list, add it up too.

E: add them with +.

`type(x) == list`: Lecture 13.

Nested Sum, in Python

```
1  def nested_sum(L):
2      if L == []:
3          return 0
4      first = L[0]
5      rest = nested_sum(L[1:])
6      if type(first) == list:
7          return nested_sum(first) + rest
8      return first + rest
```

`nested_sum([1, [2, [3]]])` → 6

W: [] adds up to 0.

I: first item + the rest.

S: add up the rest. If the first item is a list, add it up too.

E: add them with +.

`type(x) == list`: Lecture 13.

Recursion or a Loop?

Recursion

- Shapes inside shapes: folders, lists in lists.
- Puzzles like the towers.

A loop

- Simple counting and adding.
- Anything more than about **1000** calls deep.

Recursion or a Loop?

Recursion

- Shapes inside shapes: folders, lists in lists.
- Puzzles like the towers.

A loop

- Simple counting and adding.
- Anything more than about **1000** calls deep.

`sum_to(5000)` from Lecture 21: `RecursionError`. A loop adds 5000 numbers easily.

More Practice

You Try! *(WISE on paper, 3 min)*

How many digits does a number have? Write `count_digits(n)`.

`count_digits(4096)` → 4 `count_digits(7)` → 1

W Which numbers are easy?

I How do you make a number smaller? (Look back at `digit_sum`.)

S Pretend `count_digits` of the smaller number already works.

E How many digits did you cut off?

Count the Digits, in Python

```
1 def count_digits(n):
2     if n < 10:
3         return 1
4     rest = n // 10
5     return count_digits(rest) + 1
6
7 print(count_digits(4096))
```

Count the Digits, in Python

```
1 def count_digits(n):
2     if n < 10:
3         return 1
4     rest = n // 10
5     return count_digits(rest) + 1
6
7 print(count_digits(4096))
```

W: one digit: 1.

I: `rest = n // 10`

S: `count_digits(rest)`

E: add 1 for the cut digit.

Count the Digits, in Python

```
1 def count_digits(n):
2     if n < 10:
3         return 1
4     rest = n // 10
5     return count_digits(rest) + 1
6
7 print(count_digits(4096))
```

count_digits(4096)

W: one digit: 1.

I: `rest = n // 10`

S: `count_digits(rest)`

E: add 1 for the cut digit.

Count the Digits, in Python

```
1  def count_digits(n):
2      if n < 10:
3          return 1
4      rest = n // 10
5      return count_digits(rest) + 1
6
7  print(count_digits(4096))
```

```
count_digits(4096)
  = count_digits(409) + 1
```

W: one digit: 1.

I: `rest = n // 10`

S: `count_digits(rest)`

E: add 1 for the cut digit.

Count the Digits, in Python

```
1 def count_digits(n):
2     if n < 10:
3         return 1
4     rest = n // 10
5     return count_digits(rest) + 1
6
7 print(count_digits(4096))
```

```
count_digits(4096)
  = count_digits(409) + 1
    = count_digits(40) + 1 + 1
```

W: one digit: 1.

I: `rest = n // 10`

S: `count_digits(rest)`

E: add 1 for the cut digit.

Count the Digits, in Python

```
1  def count_digits(n):
2      if n < 10:
3          return 1
4      rest = n // 10
5      return count_digits(rest) + 1
6
7  print(count_digits(4096))
```

```
count_digits(4096)
  = count_digits(409) + 1
    = count_digits(40) + 1 + 1
      = count_digits(4) + 1 + 1 + 1
```

W: one digit: 1.

I: `rest = n // 10`

S: `count_digits(rest)`

E: add 1 for the cut digit.

Count the Digits, in Python

```
1  def count_digits(n):
2      if n < 10:
3          return 1
4      rest = n // 10
5      return count_digits(rest) + 1
6
7  print(count_digits(4096))
```

```
count_digits(4096)
  = count_digits(409) + 1
    = count_digits(40) + 1 + 1
      = count_digits(4) + 1 + 1 + 1
        = 1 + 1 + 1 + 1 = 4
```

W: one digit: 1.

I: `rest = n // 10`

S: `count_digits(rest)`

E: add 1 for the cut digit.

Count the Digits, in Python

```
1  def count_digits(n):
2      if n < 10:
3          return 1
4      rest = n // 10
5      return count_digits(rest) + 1
6
7  print(count_digits(4096))
```

```
count_digits(4096)
  = count_digits(409) + 1
    = count_digits(40) + 1 + 1
      = count_digits(4) + 1 + 1 + 1
        = 1 + 1 + 1 + 1 = 4
```

W: one digit: `1`.

I: `rest = n // 10`

S: `count_digits(rest)`

E: add `1` for the cut digit.

The same shape as `digit_sum`. Only **E** changed: `+ 1`, not `+ last`.

You Try! *(WISE on paper, 3 min)*

Write `stutter(s)`: every letter, **twice**.

`stutter("cat")` → `"ccaatt"`

Stuck? Write small examples first: `stutter("t")`, `stutter("at")`, `stutter("cat")`.

W Which string is easy?

I How do you make a string smaller?

S Pretend `stutter` of the smaller string already works.

E What do you put in **front** of it?

Stutter, in Python

```
1 def stutter(s):
2     if s == "":
3         return ""
4     double = s[0] + s[0]
5     return double + stutter(s[1:])
6
7 print(stutter("cat"))
```

Stutter, in Python

```
1 def stutter(s):
2     if s == "":
3         return ""
4     double = s[0] + s[0]
5     return double + stutter(s[1:])
6
7 print(stutter("cat"))
```

W: "" gives "".

I: drop the first letter.

S: `stutter(s[1:])`

E: the first letter, twice, in **front**.

Stutter, in Python

```
1 def stutter(s):
2     if s == "":
3         return ""
4     double = s[0] + s[0]
5     return double + stutter(s[1:])
6
7 print(stutter("cat"))
```

```
stutter("cat")
```

W: "" gives "".

I: drop the first letter.

S: `stutter(s[1:])`

E: the first letter, twice, in **front**.

Stutter, in Python

```
1  def stutter(s):
2      if s == "":
3          return ""
4      double = s[0] + s[0]
5      return double + stutter(s[1:])
6
7  print(stutter("cat"))
```

```
stutter("cat")
  = "cc" + stutter("at")
```

W: "" gives "".

I: drop the first letter.

S: `stutter(s[1:])`

E: the first letter, twice, in **front**.

Stutter, in Python

```
1  def stutter(s):
2      if s == "":
3          return ""
4      double = s[0] + s[0]
5      return double + stutter(s[1:])
6
7  print(stutter("cat"))
```

```
stutter("cat")
  = "cc" + stutter("at")
    = "cc" + "aa" + stutter("t")
```

W: "" gives "".

I: drop the first letter.

S: `stutter(s[1:])`

E: the first letter, twice, in **front**.

Stutter, in Python

```
1 def stutter(s):
2     if s == "":
3         return ""
4     double = s[0] + s[0]
5     return double + stutter(s[1:])
6
7 print(stutter("cat"))
```

```
stutter("cat")
 = "cc" + stutter("at")
  = "cc" + "aa" + stutter("t")
   = "cc" + "aa" + "tt" + stutter("")
```

W: "" gives "".

I: drop the first letter.

S: `stutter(s[1:])`

E: the first letter, twice, in **front**.

Stutter, in Python

```
1  def stutter(s):
2      if s == "":
3          return ""
4      double = s[0] + s[0]
5      return double + stutter(s[1:])
6
7  print(stutter("cat"))
```

```
stutter("cat")
  = "cc" + stutter("at")
    = "cc" + "aa" + stutter("t")
      = "cc" + "aa" + "tt" + stutter("")
        = "ccaatt"
```

W: "" gives "".

I: drop the first letter.

S: `stutter(s[1:])`

E: the first letter, twice, in **front**.

Stutter, in Python

```
1  def stutter(s):
2      if s == "":
3          return ""
4      double = s[0] + s[0]
5      return double + stutter(s[1:])
6
7  print(stutter("cat"))
```

```
stutter("cat")
  = "cc" + stutter("at")
    = "cc" + "aa" + stutter("t")
      = "cc" + "aa" + "tt" + stutter("")
        = "ccaatt"
```

W: "" gives "".

I: drop the first letter.

S: `stutter(s[1:])`

E: the first letter, twice, in **front**.

Like `reverse`, but the letter goes in front, and twice.

You Try! *(WISE on paper, a bit harder, 4 min)*

Is a list in order, smallest first? Write `is_sorted(L)`.

`is_sorted([2, 5, 5, 9])` → `True`

`is_sorted([2, 7, 4, 9])` → `False`

W Which lists are always in order?

I How do you make a list smaller?

S Pretend `is_sorted` of the smaller list already works.

E Look at the first **two** items. Wrong order? Then what?

Is It Sorted, in Python

```
1  def is_sorted(L):
2      if len(L) <= 1:
3          return True
4      if L[0] > L[1]:
5          return False
6      return is_sorted(L[1:])
7
8  print(is_sorted([2, 7, 4, 9]))
```

Is It Sorted, in Python

```
1 def is_sorted(L):
2     if len(L) <= 1:
3         return True
4     if L[0] > L[1]:
5         return False
6     return is_sorted(L[1:])
7
8 print(is_sorted([2, 7, 4, 9]))
```

W: 0 or 1 items: `True`.

I: drop the first item.

S: `is_sorted(L[1:])`

E: only if the first two are in order. If not: `False` at once.

Is It Sorted, in Python

```
1  def is_sorted(L):
2      if len(L) <= 1:
3          return True
4      if L[0] > L[1]:
5          return False
6      return is_sorted(L[1:])
7
8  print(is_sorted([2, 7, 4, 9]))
```

```
is_sorted([2, 7, 4, 9])
```

W: 0 or 1 items: `True`.

I: drop the first item.

S: `is_sorted(L[1:])`

E: only if the first two are in order. If not: `False` at once.

Is It Sorted, in Python

```
1  def is_sorted(L):
2      if len(L) <= 1:
3          return True
4      if L[0] > L[1]:
5          return False
6      return is_sorted(L[1:])
7
8  print(is_sorted([2, 7, 4, 9]))
```

```
is_sorted([2, 7, 4, 9])
  2 <= 7 → is_sorted([7, 4, 9])
```

W: 0 or 1 items: `True`.

I: drop the first item.

S: `is_sorted(L[1:])`

E: only if the first two are in order. If not: `False` at once.

Is It Sorted, in Python

```
1  def is_sorted(L):
2      if len(L) <= 1:
3          return True
4      if L[0] > L[1]:
5          return False
6      return is_sorted(L[1:])
7
8  print(is_sorted([2, 7, 4, 9]))
```

```
is_sorted([2, 7, 4, 9])
  2 <= 7 → is_sorted([7, 4, 9])
    7 > 4 → False
```

W: 0 or 1 items: `True`.

I: drop the first item.

S: `is_sorted(L[1:])`

E: only if the first two are in order. If not: `False` at once.

Is It Sorted, in Python

```
1  def is_sorted(L):
2      if len(L) <= 1:
3          return True
4      if L[0] > L[1]:
5          return False
6      return is_sorted(L[1:])
7
8  print(is_sorted([2, 7, 4, 9]))
```

```
is_sorted([2, 7, 4, 9])
  2 <= 7 → is_sorted([7, 4, 9])
    7 > 4 → False
```

W: 0 or 1 items: `True`.

I: drop the first item.

S: `is_sorted(L[1:])`

E: only if the first two are in order. If not: `False` at once.

Like `is_palindrome`: stop early with `False`.

Summary

Summary

- **WISE:** easiest case, make it smaller, trust the smaller job, extend.
- **Stuck?** Write small examples and look for the row above inside each answer.
- **Smaller:** `n - 1`, `n // 10`, `s[1:]`, `s[1:-1]`, `L[1:]`.
- **Towers:** `moves(n)` $= 2\times$ `moves(n - 1)` $+1 = 2^n - 1$.

Big Idea

To move a tower of 64, move a tower of 63... twice.

Recursion makes the program **short**.

It does not make the work **small**:
$2^{64} - 1$ moves, **585 billion years**.

Next Time: Randomness

- 1946. A man is sick in bed, playing cards.

Next Time: Randomness

- 1946. A man is sick in bed, playing cards.
- What is his chance to win? The maths is too hard.

Next Time: Randomness

- 1946. A man is sick in bed, playing cards.
- What is his chance to win? The maths is too hard.
- His answer: let the computer play, **again and again**.

Questions?

Exit Ticket

Today's question *(2 minutes)*

(a) How many moves for **5** disks?

(b) Fill the blank. `total([4, 5, 6])` is 15.

```
def total(L):
    if L == []:
        return 0
    return ________
```

COMP 102 — Exit Ticket #22

Name: _____

Roll #: _____ Date: _____

Answer:

Hand it to me on your way out.
No name, no attendance.