Functional Languages
7th Lecture

Recursion

$$n! = \begin{cases} 1 & \text{if } n == 0 \\ n * (n-1)! & \text{otherwise} \end{cases}$$

- A recursive function applies itself
- Can be finite or infinite
- Most recursive functions have
  - base case
  - recursive case
- Example: factorial, Fibonacci, power

fact 4 = 4 * fact (4 - 1)

= 4 * fact 3

= 4 * 3 * fact (3 - 1)

= 4 * 3 * fact 2

= 4 * 3 * 2 * fact (2 - 1)

= 4 * 3 * 2 * fact 1

= 4 * 3 * 2 * 1 * fact (1 - 1)

= 4 * 3 * 2 * 1 * fact 0

= 4 * 3 * 2 * 1 * 1

= 24

reached the base case

---

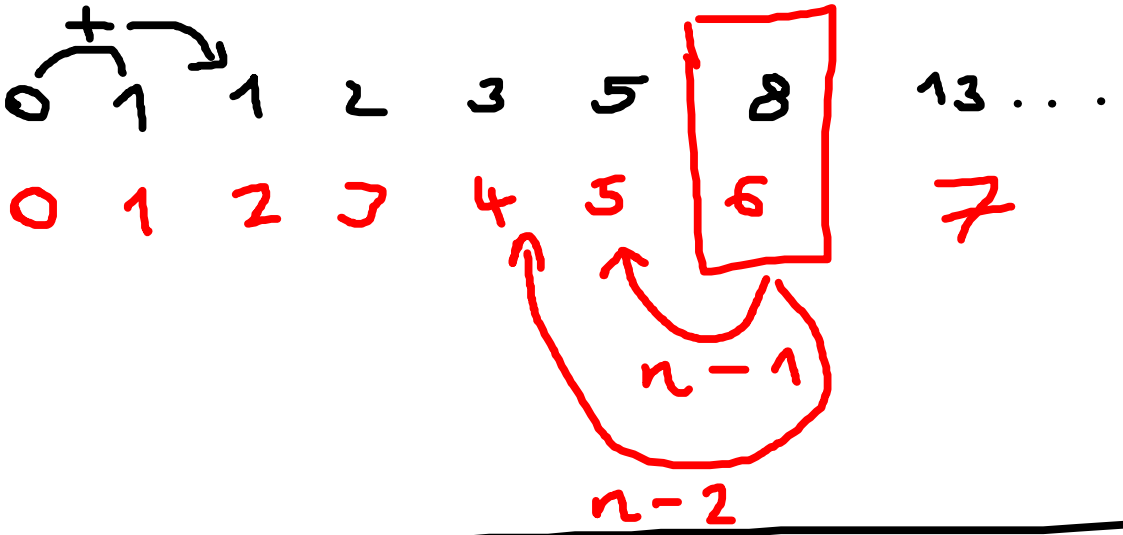
fact 0 = 1 ← base case (non-recursive case)

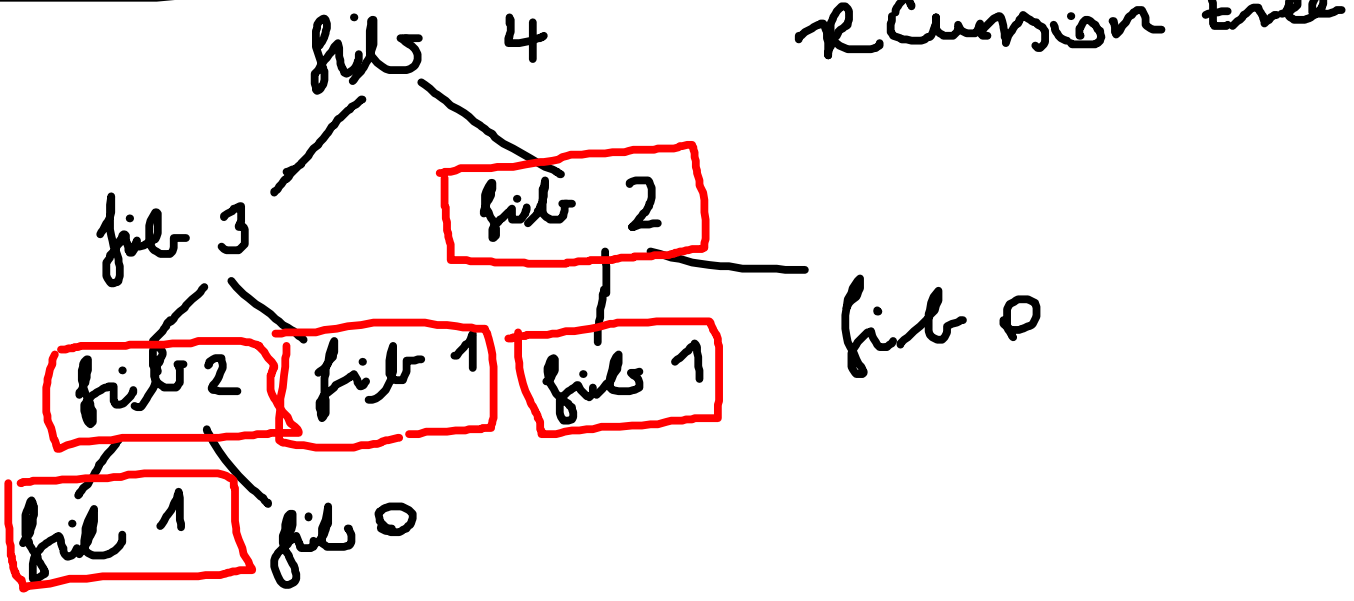
fact n = n * fact (n - 1) ← recursive case

without base case:

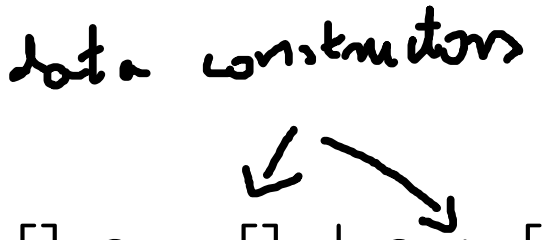
fact 4 = 4 * 3 * 2 * 1 * 0 * (-1) * (-2) ...

Fibonacci numbers





Recursion over lists



- Remember, `data [] a = [] | a : [a]`
- List itself is a recursive data structure
- Recursion is natural when traversing a list
- Principle:
  1. Break down a problem to similar subproblem(s)
  2. Solve the subproblem(s) recursively (because they are similar)
  3. Combine the solution(s) of the subproblem(s) and adjust them to solve the original problem
- Example: `length`, `minimum`, `sum`, `last`, `sort`, `concat`

mylength [5, 6, 7, 10] =

first rest

= 1 + mylength [6, 7, 10]

= 1 + 1 + mylength [7, 10]

= 1 + 1 + 1 + mylength [10]

= 1 + 1 + 1 + 1 + mylength [] ← base case

= 1 + 1 + 1 + 1 + 0

= 4