Recursion

D.E ZEGOUR

École Supérieure d'Informatique

ESI

Recursion

Introductory Examples

Factorial

Multiplication of natural numbers

Fibonacci sequence

Binary search

Recursion

Introductory Examples : Factorial

n! = 1 if n=0
n! = n (n-1) (n-2) ....1 if n > 0

```
Prod := N
FOR X = N-1 ,2 -1 :
   Prod := Prod * X
ENDFOR
```

n! = 1 if n =0
n! = n (n-1)! if n > 0

Fact(0) = 1
Fact(n) = n * Fact(n-1) if n > 0

Recursive definition

4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1!
1! = 1 * 0!
0! = 1

```
Fact(N):
IF N = 0
   Fact := 1
ELSE
  Fact := N * Fact(N-1)
ENDIF
```

Recursion

Introductory Examples : Multiplication

a * b = a if b = 1
a * b = a * (b-1) + a if b > 1

Mult(a, b) = a if b = 1
Mult(a, b) = Mult(a, b-1) + a if b > 1

```
Mult(A, B):
IF B = 1
    Mult := A
ELSE
    Mult := Mult(a, b-1) + a
ENDIF
```

Recursion

Introductory Examples : Synthesis

In recursive definitions:
- A base case is explicitly defined,
  0! = 1
  a * 1 = a
- A general case,
  n! is defined in terms of (n-1)!;
  a * b is defined in terms of a * (b-1).

Invalid definitions:
n! = (n+1)! / (n+1)
a * b = a * (b+1) - a

Ensure that from the general case, you reach the base case. For factorial, n is successively decreased by 1 until it reaches 0. For multiplication, b gradually decreases by one until it reaches 1.

Recursion

Introductory Examples : Fibonacci sequence

Fib(n) = n if n =0 ou n=1

Fib(n) = Fib(n-1) + Fib(n-2) if n >= 2

The Fib function references itself twice.

Recursion

Introductory Examples : Binary search in an ordered array

Used for both mathematical and computer science applications.

```
Search(X, Bi, Bs) :

IF Bi > Bs
      Search:= 0
   ELSE
      Mid := ( Bi + Bs ) DIV 2
      IF X = A(Mid)
           Search:= Mid
       ELSE
           IF X < A(Mid)
                Search (X, Bi, Mid - 1)
            ELSE
                Search (X , Mid + 1, Bs)
           ENDIF
     ENDIF
ENDIF
```

Recursion

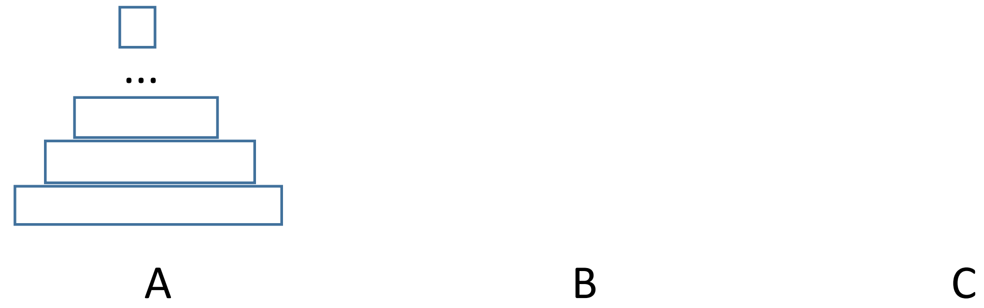
**Introductory Examples : ** Properties

A recursive algorithm should not generate an infinite sequence of calls to itself.

There must always be a way to exit recursive calls, a "way out."

(1) Factorial 0! = 1
(2) Multiplication a * 1 = a
(3) Fibonnacci fib(0) = 0 ; fib(1) = 1;
(4) Binary search
IF Bi > Bs : Search:= 0
IF A(Mid)= X: Search := Mid

Recursion



Designing Recursive Algorithms

For some problems, it is much easier to search for a recursive solution rather than an iterative one.

Example : Towers of Hanoi

We have 3 towers A, B, and C.

Initially, n disks of different diameters are placed on A.

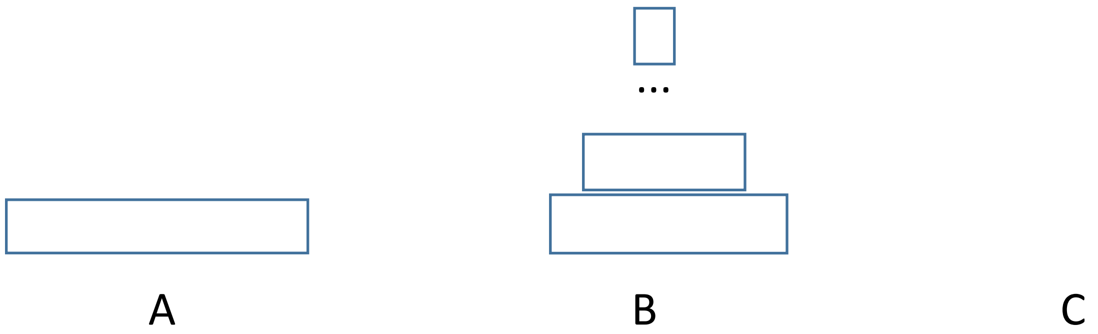
A larger disk should never be placed on a smaller disk.

Problem: Move the n disks from A to C, using B as an intermediary, while adhering to the following two rules:

Rule 1: At any given moment, only the disk at the top of a tower can be moved to another tower.

Rule 2: A larger disk should not be placed on a smaller disk.

Recursion



Designing Recursive Algorithms

Hard to find an iterative solution (try...)

Simple and elegant recursive solution.

Let's assume that we have a solution for n-1 disks.

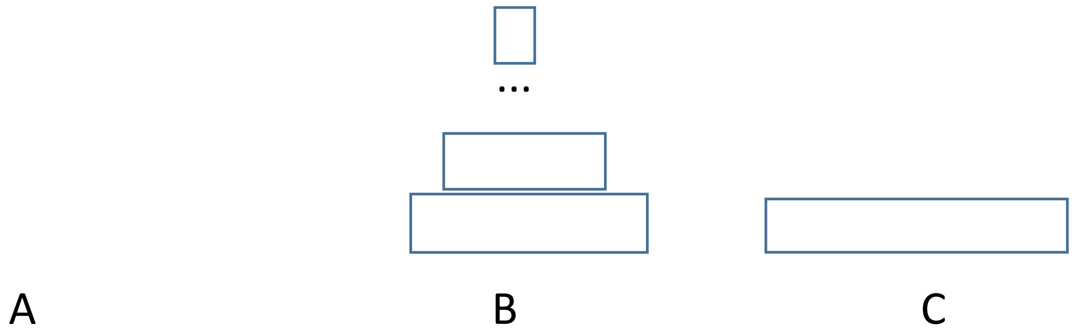
A solution for n disks using the solution for n-1 disks.

n=1 constitutes the trivial case: move the single disk from A to C.

To move n disks from A to C, using B as an auxiliary, proceed as follows:

1. Move the top n-1 disks from A to B with C as an auxiliary.

Récursivité



Designing Recursive Algorithms

Hard to find an iterative solution (try...)

Simple and elegant recursive solution.

Let's assume that we have a solution for n-1 disks.

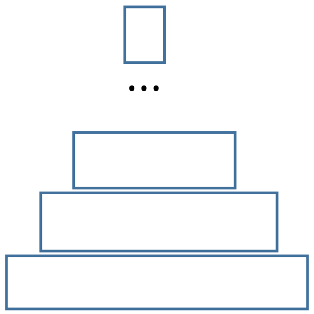
A solution for n disks using the solution for n-1 disks.

n=1 constitutes the trivial case: move the single disk from A to C.

To move n disks from A to C, using B as an auxiliary, proceed as follows:

1. Move the top n-1 disks from A to B with C as an auxiliary.
2. Move the remaining disk from A to C.

Recursion



Designing Recursive Algorithms

A B C

Hard to find an iterative solution (try...)

Simple and elegant recursive solution.

Let's assume that we have a solution for n-1 disks.

A solution for n disks using the solution for n-1 disks.

n=1 constitutes the trivial case: move the single disk from A to C.

To move n disks from A to C, using B as an auxiliary, proceed as follows:

1. Move the top n-1 disks from A to B with C as an auxiliary.
2. Move the remaining disk from A to C.
3. Move the n-1 disks from B to C with A as an auxiliary.

Recursion

Designing Recursive Algorithms

The algorithm is as follows:

```
HanoiTower(N, A, C, B):

If N = 1 {base case}
      Write("move disk 1 from", A, "to", C)
Else
      {Move the N-1 disks from A to B with C as auxiliary}
      HanoiTower(N-1, A, B, C)
      {Move the remaining disk from A to C}
      Write("move disk", N,"from", A,"to", C )
      {Move the N-1 disks from B to C, using A as auxiliary}
      HanoiTower(N-1, B, C, A)
End If
```

Number of moves : $2^n - 1$

n=3 → 7
n=6 → 63
n= 10 → 1,023
n= 15 → 32,767
n=20 → 1,048,575