

**Aula 05:
- Recursão (parte 1)**

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

3Q-20107

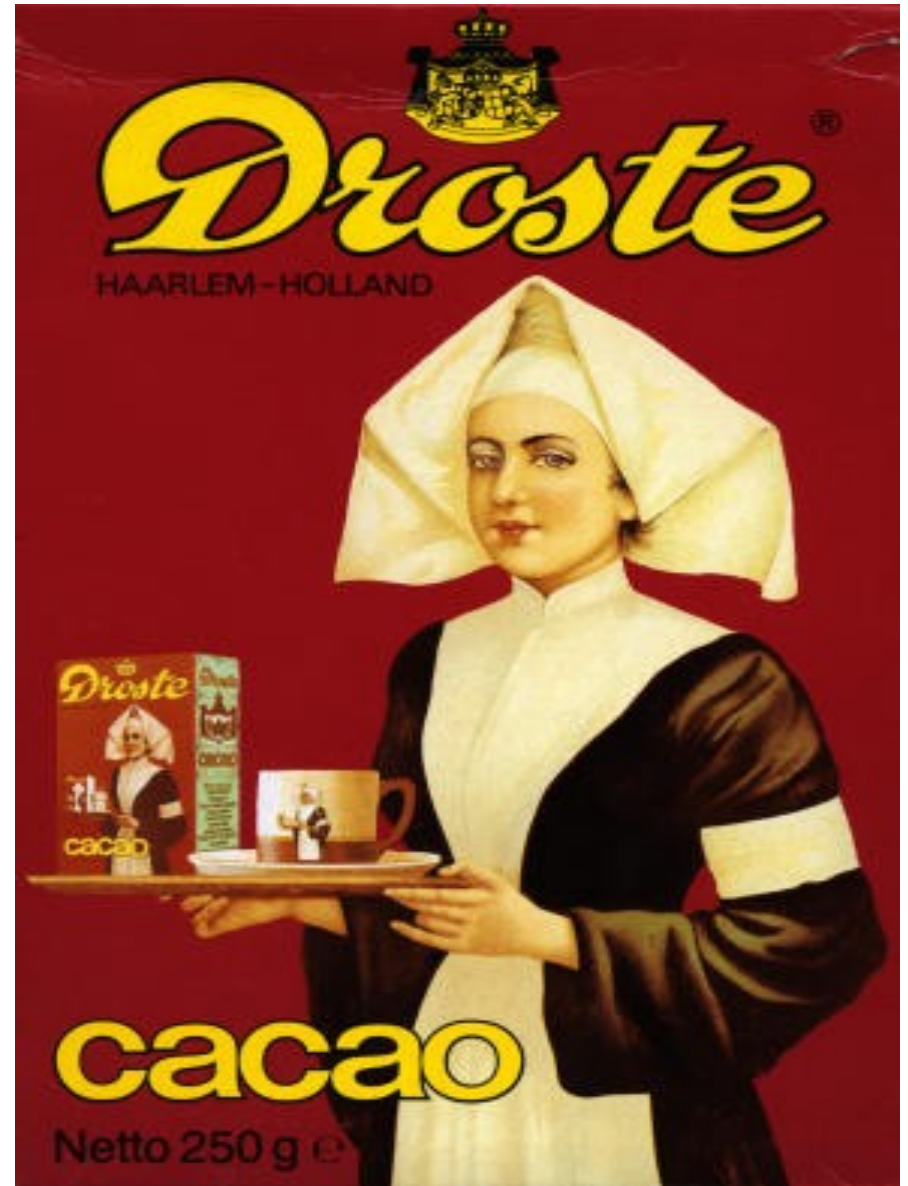
Recursão:

Se você ainda não entendeu;

Ver: "*Recursão*".



Efeito Droste



Anuncio de cacão com uma imagem recursiva.

Ciclo da Água



U.S. Department of the Interior
U.S. Geological Survey
<http://ga.water.usgs.gov/edu/watercycle.html>

Recursão

- O conceito de recursão é de fundamental importância em computação!
- Muitos problemas computacionais têm a seguinte propriedade:
 - Cada instância do problema contém uma instância menor do mesmo problema.**
 - Dizemos que esses problemas têm ***estrutura recursiva***.

Recursão

Para resolver um tal problema é natural aplicar o seguinte método:

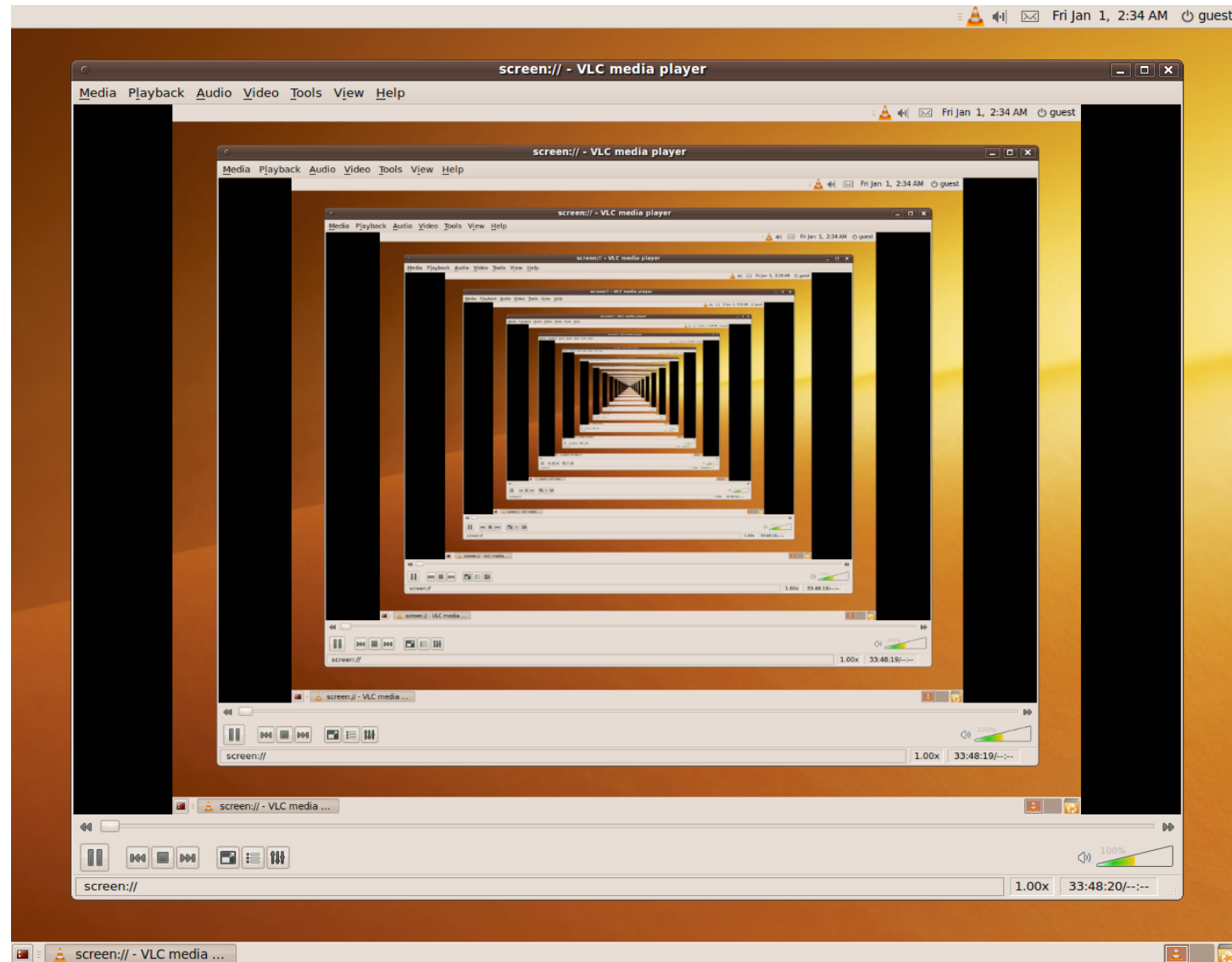
- **Se a instância em questão é pequena:**
 - Resolva-a diretamente
(use força bruta se necessário)
- **Senão**
 - Reduza-a a uma instância menor do mesmo problema
 - Aplique o método à instância menor e volte à instância original.

A aplicação do método produz um algoritmo recursivo.

A recursão pode ser infinita.

Não esqueça de definir **o caso base** (condição de parada)

<https://www.youtube.com/watch?v=ANf0vsDHI4c>



Fonte: <http://en.wikipedia.org/wiki/Recursion>



| | N | Factorial |
|----|----|---------------------|
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 6 |
| 4 | 4 | 24 |
| 5 | 5 | 120 |
| 6 | 6 | 720 |
| 7 | 7 | 5040 |
| 8 | 8 | 40320 |
| 9 | 9 | 362880 |
| 10 | 10 | 3628800 |
| 11 | 11 | 39916800 |
| 12 | 12 | 479001600 |
| 13 | 13 | 6227020800 |
| 14 | 14 | 87178291200 |
| 15 | 15 | 1307674368000 |
| 16 | 16 | 20922789888000 |
| 17 | 17 | 355687428096000 |
| 18 | 18 | 6402373705728000 |
| 19 | 19 | 121645100408832000 |
| 20 | 20 | 2432902008176640000 |

This site is supported by donations to [The OEIS Foundation](#).

THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES®

founded in 1964 by N. J. A. Sloane

The On-Line Encyclopedia of Integer Sequences® (OEIS®)

Enter a sequence, word, or sequence number:

1,2,3,6,11,23,47,106,235

Search

[Hints](#) [Welcome](#) [Video](#)

For more information about the Encyclopedia, see the [Welcome](#) page.

Languages: [English](#) [Shqip](#) [العربية](#) [Bangla](#) [Български](#) [Català](#) [中文 \(正體字, 简化字 \(1\), 简化字 \(2\)\)](#)
[Hrvatski](#) [Čeština](#) [Dansk](#) [Nederlands](#) [Esperanto](#) [Eesti](#) [فارسی](#) [Suomi](#) [Français](#) [Deutsch](#) [Ελληνικά](#) [עברית](#)
[हिंदी](#) [Magyar](#) [Igbo](#) [Bahasa Indonesia](#) [Italiano](#) [日本語](#) [ಕನ್ನಡ](#) [한국어](#) [Lietuvių](#) [मराठी](#) [Bokmål](#) [Nynorsk](#) [Polski](#) [Português](#)
[Română](#) [Русский](#) [Српски](#) [Slovenščina](#) [Español](#) [Svenska](#) [Tagalog](#) [ภาษาไทย](#) [Türkçe](#) [Українська](#) [اردو](#) [Tiếng Việt](#) [Cymraeg](#)

[Lookup](#) | [Welcome](#) | [Wiki](#) | [Register](#) | [Music](#) | [Plot 2](#) | [Demos](#) | [Index](#) | [Browse](#) | [More](#) | [WebCam](#)
[Contribute new seq. or comment](#) | [Format](#) | [Transforms](#) | [Superseeker](#) | [Recent](#) | [More pages](#)
[The OEIS Community](#) | Maintained by [The OEIS Foundation Inc.](#)

[License Agreements](#), [Terms of Use](#), [Privacy Policy](#).

Last modified October 6 05:33 EDT 2016. Contains 276787 sequences.

THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES®

founded in 1964 by N. J. A. Sloane

| | N | Factorial |
|----|----|---------------------|
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 6 |
| 4 | 4 | 24 |
| 5 | 5 | 120 |
| 6 | 6 | 720 |
| 7 | 7 | 5040 |
| 8 | 8 | 40320 |
| 9 | 9 | 362880 |
| 10 | 10 | 3628800 |
| 11 | 11 | 39916800 |
| 12 | 12 | 479001600 |
| 13 | 13 | 6227020800 |
| 14 | 14 | 87178291200 |
| 15 | 15 | 1307674368000 |
| 16 | 16 | 20922789888000 |
| 17 | 17 | 355687428096000 |
| 18 | 18 | 6402373705728000 |
| 19 | 19 | 121645100408832000 |
| 20 | 20 | 2432902008176640000 |

[Hints](#)

(Greetings from [The On-Line Encyclopedia of Integer Sequences!](#))

Search: **seq:1,2,6,24,120**

Displaying 1-10 of 348 results found.

page 1 [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) ... [35](#)

Sort: relevance | [references](#) | [number](#) | [modified](#) | [created](#) Format: long | [short](#) | [data](#)

A000142 Factorial numbers: $n! = 1*2*3*4*...*n$ (order of symmetric group S_n , number of permutations of n letters). +20
1721

(Formerly M1675 N0659)

1, **1**, **2**, **6**, **24**, **120**, 720, 5040, 40320, 362880, 3628800, 39916800, 479001600, 6227020800, 87178291200, 1307674368000, 20922789888000, 355687428096000, 6402373705728000, 121645100408832000, 2432902008176640000, 51090942171709440000, 1124000727777607680000 ([list](#); [graph](#); [refs](#); [listen](#); [history](#); [text](#); [internal format](#))

OFFSET 0,3

COMMENTS The earliest publication that discusses this sequence appears to be the Sopher Yezirah [Book of Creation], circa AD 300. (See Knuth, also the Zeilberger link) - [N. J. A. Sloane](#), Apr 07 2014

For $n \geq 1$, $a(n)$ is the number of $n \times n$ (0,1) matrices with each row and column containing exactly one entry equal to 1.

This sequence is the BinomialMean transform of [A000354](#). (See [A075271](#) for definition.) - [John W. Layman](#), Sep 12 2002. This is easily verified from the Paul Barry formula for [A000354](#), by interchanging summations and using the formula: $\sum_k (-1)^k C(n-i, k) = \text{KroneckerDelta}(i, n)$. - [David Callan](#), Aug 31 2003

Number of distinct subsets of $T(n-1)$ elements with 1 element A, 2 elements B, ..., $n-1$ elements X (e.g., at $n=5$, we consider the distinct subsets of ABBCCDDDD and there are $5! = 120$). - [Jon Perry](#), Jun 12 2003

$n!$ is the smallest number with that prime signature. E.g., $720 = 2^4 * 3^2 * 5$. - [Amarnath Murthy](#), Jul 01 2003

$a(n)$ is the permanent of the $n \times n$ matrix M with $M(i, j) = 1$. - [Philippe Deléham](#), Dec 15 2003

Given n objects of distinct sizes (e.g., areas, volumes) such that each object is sufficiently large to simultaneously contain all previous objects, then $n!$ is the total number of essentially different arrangements using all n objects. Arbitrary levels of nesting of objects are permitted within arrangements. (This application of the sequence was inspired by considering leftover moving boxes.) If the restriction exists that each object is able or permitted to contain at most one smaller (but possibly nested) object at a time, the resulting sequence begins 1,2,5,15,52 (Bell Numbers?). Sets of nested wooden boxes or traditional nested Russian dolls come to mind here. - [Rick L. Shepherd](#), Jan 14 2004

From [Michael Somos](#), Mar 04 2004; edited by [M. F. Hasler](#), Jan 02 2015:

(Start)

Stirling transform of [2, 2, 6, 24, 120, ...] is [A052856](#) = [2, 2, 4, 14, 76,



Fatorial de um número

Fatorial de um número

Considere a função fatorial (representado por **n!**) para um número inteiro, **n**, não-negativo arbitrário

$$\text{fatorial}(n) = \begin{cases} 1 & , \text{ se } n=0 \\ n \times \text{fatorial}(n - 1) & , \text{ caso contrario} \end{cases}$$

$$\text{fatorial}(n) = \begin{cases} 1 & , \text{ se } n=0 \\ 1 & , \text{ se } n=1 \\ n \times \text{fatorial}(n - 1) & , \text{ caso contrario} \end{cases}$$

Fatorial de um número

```
1  #include <stdio.h>
2
3  int fatorial(int n) {
4      if (n==0)
5          return 1;
6      else
7          return fatorial(n-1)*n;
8  }
9
10 int main()
11 {
12     int num;
13
14     scanf("%d", &num);
15     printf("%d\n", fatorial(num) );
16
17     return 0;
18 }
```

| | N | Factorial |
|----|----|---------------------|
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 6 |
| 4 | 4 | 24 |
| 5 | 5 | 120 |
| 6 | 6 | 720 |
| 7 | 7 | 5040 |
| 8 | 8 | 40320 |
| 9 | 9 | 362880 |
| 10 | 10 | 3628800 |
| 11 | 11 | 39916800 |
| 12 | 12 | 479001600 |
| 13 | 13 | 6227020800 |
| 14 | 14 | 87178291200 |
| 15 | 15 | 1307674368000 |
| 16 | 16 | 20922789888000 |
| 17 | 17 | 355687428096000 |
| 18 | 18 | 6402373705728000 |
| 19 | 19 | 121645100408832000 |
| 20 | 20 | 2432902008176640000 |

Número de vezes em que a função **Fatorial** é chamada?

Fatorial de um número

```
1  #include <stdio.h>
2
3  int fatorial(int n) {
4      if (n==0)
5          return 1;
6      else
7          return fatorial(n-1)*n;
8  }
9
10 int main()
11 {
12     int num;
13
14     scanf("%d", &num);
15     printf("%d\n", fatorial(num) );
16
17     return 0;
18 }
```

| | N | Factorial |
|----|----|---------------------|
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 6 |
| 4 | 4 | 24 |
| 5 | 5 | 120 |
| 6 | 6 | 720 |
| 7 | 7 | 5040 |
| 8 | 8 | 40320 |
| 9 | 9 | 362880 |
| 10 | 10 | 3628800 |
| 11 | 11 | 39916800 |
| 12 | 12 | 479001600 |
| 13 | 13 | 6227020800 |
| 14 | 14 | 87178291200 |
| 15 | 15 | 1307674368000 |
| 16 | 16 | 20922789888000 |
| 17 | 17 | 355687428096000 |
| 18 | 18 | 6402373705728000 |
| 19 | 19 | 121645100408832000 |
| 20 | 20 | 2432902008176640000 |

Número de vezes em que a função **Fatorial** é chamada? **n+1**

Fatorial → Primorial

O primorial de um número inteiro positivo n é o produto de todos os primos menores ou iguais a n .

- É denotado por $n\#$

$$1\# = 1$$

$$2\# = 2$$

$$3\# = 2 \cdot 3 = 6$$

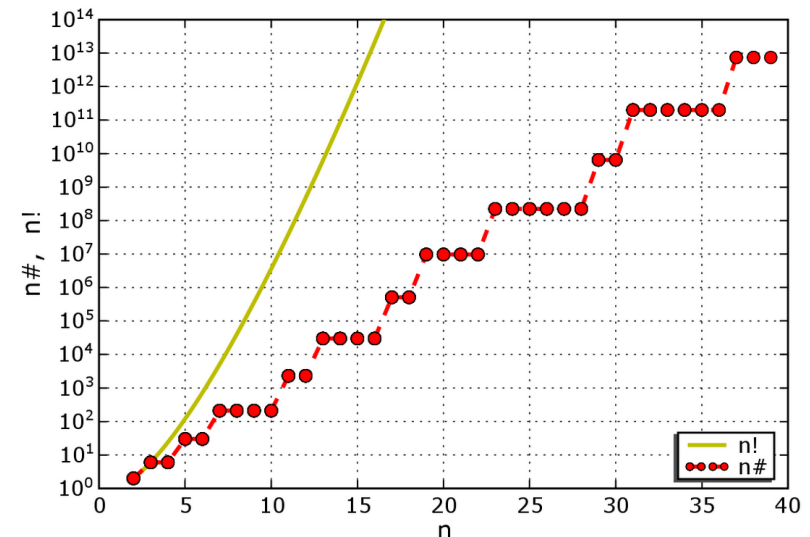
$$4\# = 2 \cdot 3 = 6$$

$$5\# = 2 \cdot 3 \cdot 5 = 30$$

$$6\# = 2 \cdot 3 \cdot 5 = 30$$

$$7\# = 2 \cdot 3 \cdot 5 \cdot 7 = 210$$

<https://en.wikipedia.org/wiki/Primorial>



- [LAB]** Crie uma função recursiva que, dado um número inteiro positivo, devolva:
 - o seu Primorial.
 - o número de multiplicações necessárias para calcular seu primorial.



Somatório de números

Um exemplo de somatória

Dados dois número inteiros, **n** e **k**, crie uma função iterativa para calcular a seguinte somatória:

$$1^k + 2^k + 3^k + \dots + n^k$$

```
long int somatoriaIte(int n, int k) {  
    long int i, soma=0;  
  
    for (i=1; i<=n; i++)  
        soma = soma + pow(i, k);  
  
    return soma;  
}
```

Um exemplo de somatória

```
1  #include <stdio.h>
2  #include <math.h>
3
4  long int somatoriaIte(int n, int k) {
5      long int i, soma=0;
6
7      for (i=1; i<=n; i++)
8          soma = soma + pow(i, k);
9
10     return soma;
11 }
12
13 int main()
14 {
15     int num, k;
16
17     scanf("%d %d", &num, &k);
18     printf("%ld\n", somatoriaIte(num, k) );
19 }
```

```
$ gcc somatoria.c -lm -o somatoria.exe
$ ./somatoria.exe
4
20
1102999460754
```

Um exemplo de somatória

Dados dois número inteiros, **n** e **k**, crie uma função iterativa para calcular a seguinte somatória:

$$1^k + 2^k + 3^k + \dots + n^k$$

$$S(n, k) = \begin{cases} 0 & , \text{ se } n = 0 \\ n^k + S(n - 1, k) & , \text{ se } n > 0 \end{cases}$$

Um exemplo de somatória

Dados dois número inteiros, **n** e **k**, crie uma função iterativa para calcular a seguinte somatória:

$$1^k + 2^k + 3^k + \dots + n^k$$

$$S(n, k) = \begin{cases} 0 & , \text{ se } n = 0 \\ n^k + S(n - 1, k) & , \text{ se } n > 0 \end{cases}$$

```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```

```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```

n=4, k=99

```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```

n=4, k=99

$4^{99} +$

n=3, k=99

```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```

n=4, k=99

$4^{99} +$

n=3, k=99

$3^{99} +$

n=2, k=99

```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```

n=4, k=99

$4^{99} +$

n=3, k=99

$3^{99} +$

n=2, k=99

$2^{99} +$

n=1, k=99


```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```

n=4, k=99

$4^{99} +$

n=3, k=99

$3^{99} +$

n=2, k=99

$2^{99} +$

n=1, k=99

$1^{99} +$

n=0, k=99

```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```

n=4, k=99

$4^{99} +$

n=3, k=99

$3^{99} +$

n=2, k=99

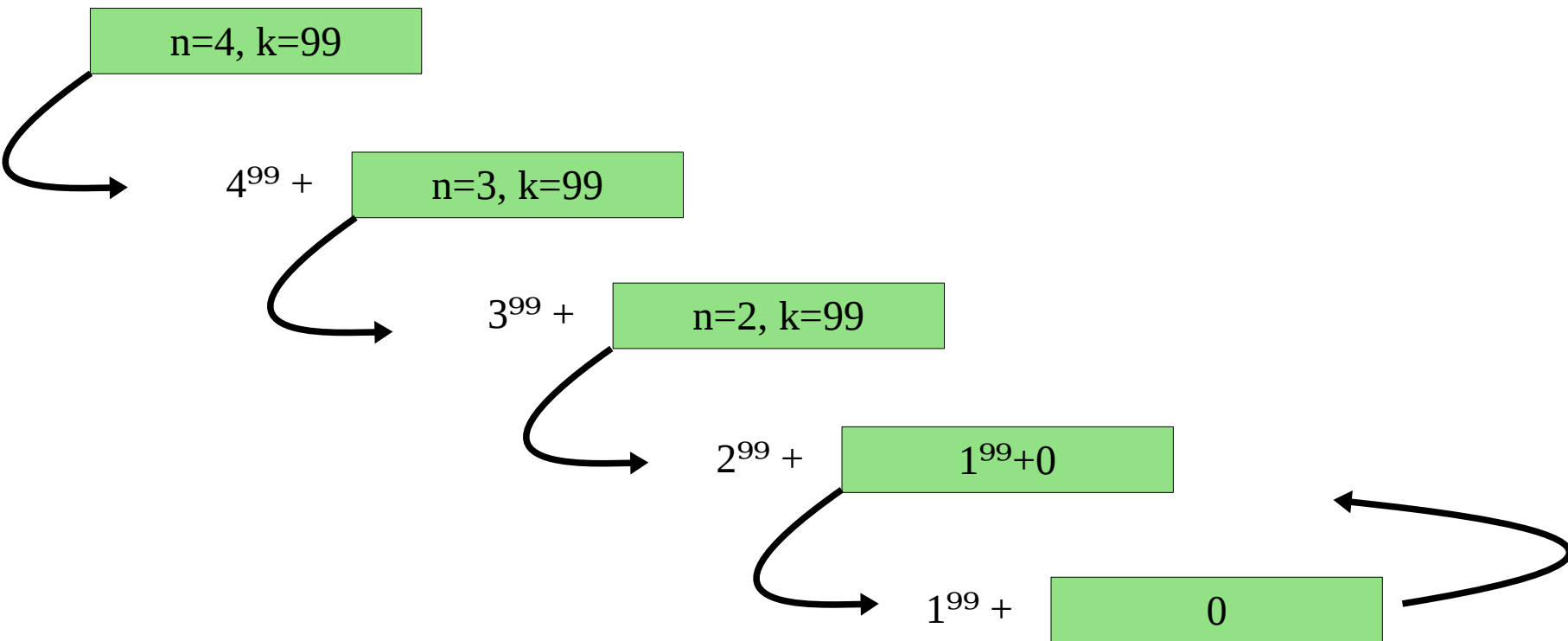
$2^{99} +$

n=1, k=99

$1^{99} +$

0

```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```



```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```

n=4, k=99

$4^{99} +$

n=3, k=99

$3^{99} +$

$2^{99} + 1^{99} + 0$

$2^{99} +$

$1^{99} + 0$

$1^{99} +$

0

```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```

n=4, k=99

$4^{99} +$

$3^{99} + 2^{99} + 1^{99} + 0$

$3^{99} +$

$2^{99} + 1^{99} + 0$

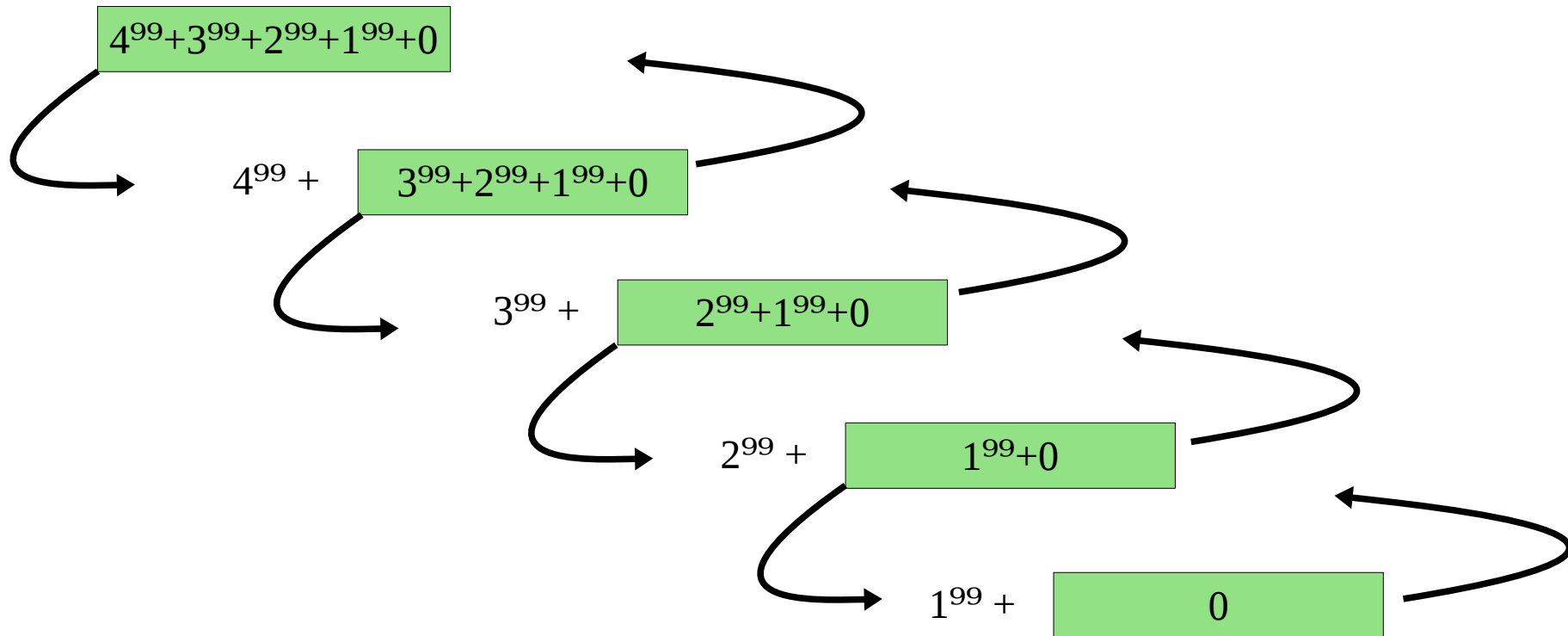
$2^{99} +$

$1^{99} + 0$

$1^{99} +$

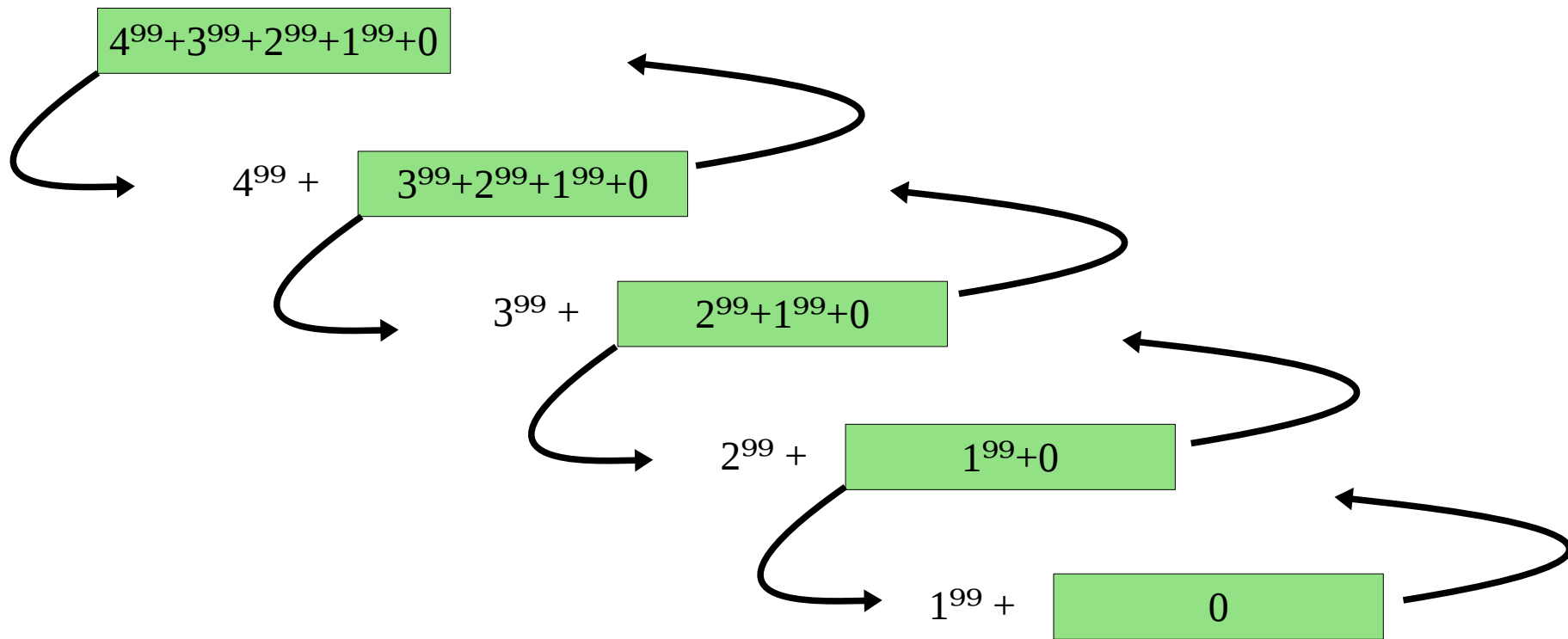
0

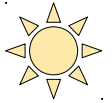
```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```



```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```

$$1^{99} + 2^{99} + 3^{99} + 4^{99}$$





```
long int somatoriaIte(int n, int k) {  
    long int i, soma=0;  
  
    for (i=1; i<=n; i++)  
        soma = soma + pow(i, k);  
  
    return soma;  
}
```

Um laço e um acumulador



```
long int somatoriaRec(int n, int k) {  
    if (n==0)  
        return 0;  
    else  
        return pow(n, k) + somatoriaRec(n-1, k);  
}
```

Sem laço e sem acumulador?





Matrioska



Matrioska

Função recursiva

- Uma função recursiva é **definida em seus próprios termos**.
- Toda função pode ser escrita como função recursiva **sem o uso de interação** (laços).
- Reciprocamente, **qualquer função recursiva pode ser descrita através de interações sucessivas**.
- **Ingredientes:**
 - Definição de casos bases (que não envolvem recursão)
 - Passos recursivos, com decremento na entrada, no sentido do caso base.

Recursão

Para resolver um tal problema é natural aplicar o seguinte método:

- **Se a instância em questão é pequena:**
 - Resolva-a diretamente
(use força bruta se necessário)
- **Senão**
 - Reduza-a a uma instância menor do mesmo problema
 - Aplique o método à instância menor e volte à instância original.

A aplicação do método produz um algoritmo recursivo.



Números de Fibonacci

Números de Fibonacci

| F_0 | F_1 | F_2 | F_3 | F_4 | F_5 | F_6 | F_7 | F_8 | F_9 | F_{10} | F_{11} | F_{12} | F_{13} | F_{14} | F_{15} | F_{16} | F_{17} | F_{18} | F_{19} | F_{20} |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 | 233 | 377 | 610 | 987 | 1597 | 2584 | 4181 | 6765 |



Números de Fibonacci

| F_0 | F_1 | F_2 | F_3 | F_4 | F_5 | F_6 | F_7 | F_8 | F_9 | F_{10} | F_{11} | F_{12} | F_{13} | F_{14} | F_{15} | F_{16} | F_{17} | F_{18} | F_{19} | F_{20} |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 | 233 | 377 | 610 | 987 | 1597 | 2584 | 4181 | 6765 |

$$Fib(n) = \begin{cases} 0 & , \text{ se } n = 0 \\ 1 & , \text{ se } n = 1 \\ Fib(n - 1) + Fib(n - 2) & , \text{ se } n > 1 \end{cases}$$

Números de Fibonacci

| F_0 | F_1 | F_2 | F_3 | F_4 | F_5 | F_6 | F_7 | F_8 | F_9 | F_{10} | F_{11} | F_{12} | F_{13} | F_{14} | F_{15} | F_{16} | F_{17} | F_{18} | F_{19} | F_{20} |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 | 233 | 377 | 610 | 987 | 1597 | 2584 | 4181 | 6765 |

$$Fib(n) = \begin{cases} 0 & , \text{ se } n = 0 \\ 1 & , \text{ se } n = 1 \\ Fib(n - 1) + Fib(n - 2) & , \text{ se } n > 1 \end{cases}$$

Função é duplamente recursiva
(efetua mais de uma chamada a Fib)

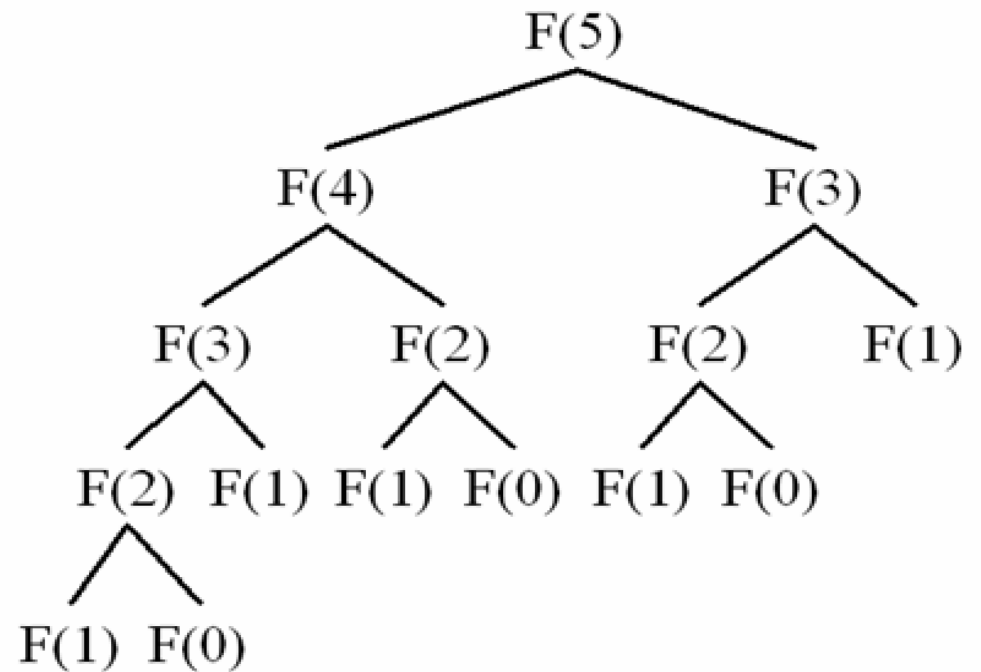
Números de Fibonacci

$$Fib(n) = \begin{cases} 0 & , \text{ se } n = 0 \\ 1 & , \text{ se } n = 1 \\ Fib(n-1) + Fib(n-2) & , \text{ se } n > 1 \end{cases}$$

```
1  #include <stdio.h>
2
3  long int Fib(int n) {
4      if (n==0)
5          return 0;
6      if (n==1)
7          return 1;
8      else
9          return Fib(n-1) + Fib(n-2);
10 }
11
12 int main() {
13     int num;
14     scanf("%d", &num);
15     printf("%ld\n", Fib(num));
16 }
```

Números de Fibonacci

Fib (5)
Fib (4)
Fib (3)
Fib (2)
Fib (1)
Fib (0)
Fib (1)
Fib (2)
Fib (1)
Fib (0)
Fib (3)
Fib (2)
Fib (1)
Fib (0)
Fib (1)



Pensou na eficiência da abordagem recursiva?

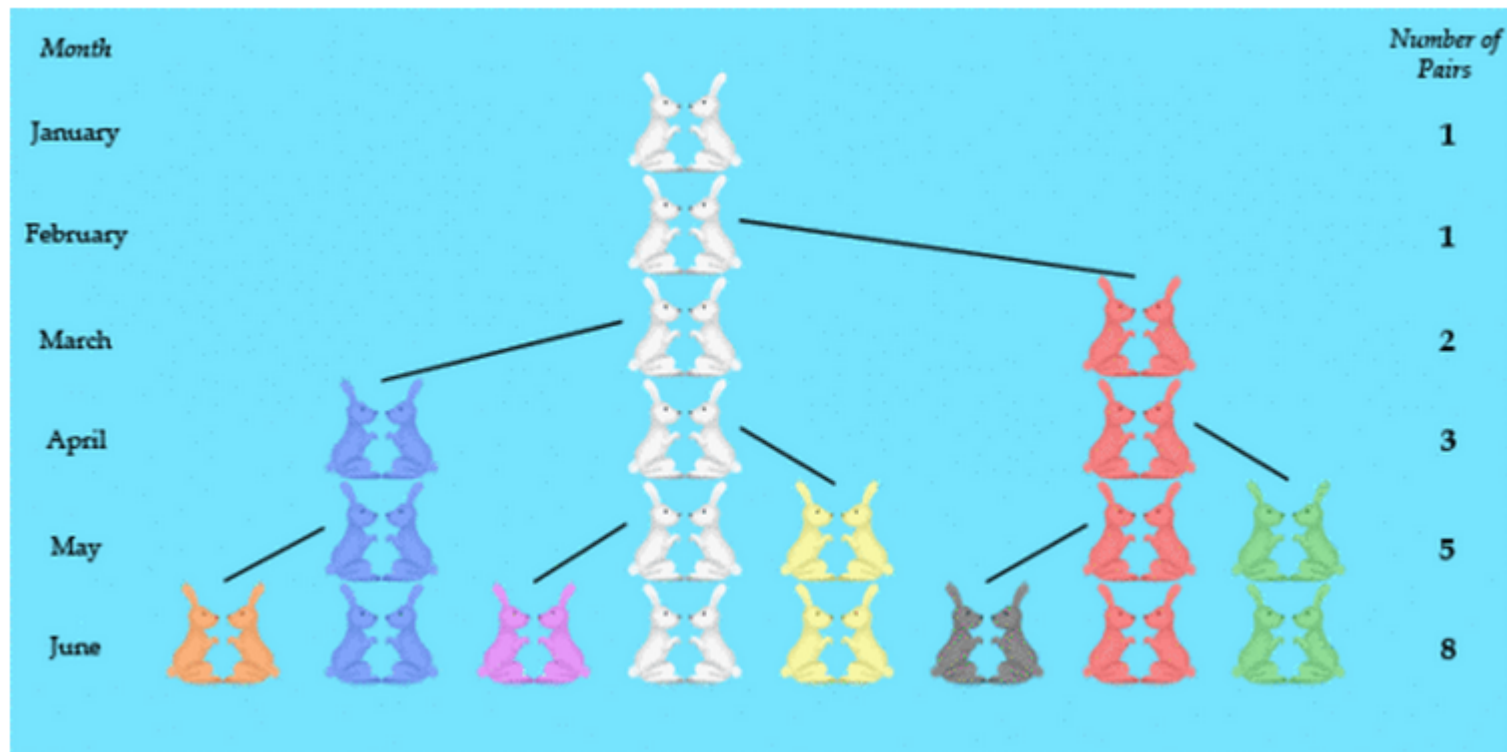
Números de Fibonacci

| n | 10 | 20 | 30 | 50 | 100 |
|------------------|-------|-------|-------|---------|----------------------|
| Recursivo | 8 ms | 1 seg | 2 min | 21 dias | 10 ⁹ anos |
| Iterativo | 1/6ms | 1/3ms | ½ ms | ¾ ms | 1,5 ms |

Fonte: Brassard and Bradley (1996) apud ZIVIANI

Números de Fibonacci

Os números de Fibonacci foram propostos por Leonardo di Pisa (Fibonacci), em 1202, como uma solução para o problema de determinar o tamanho da população de coelhos



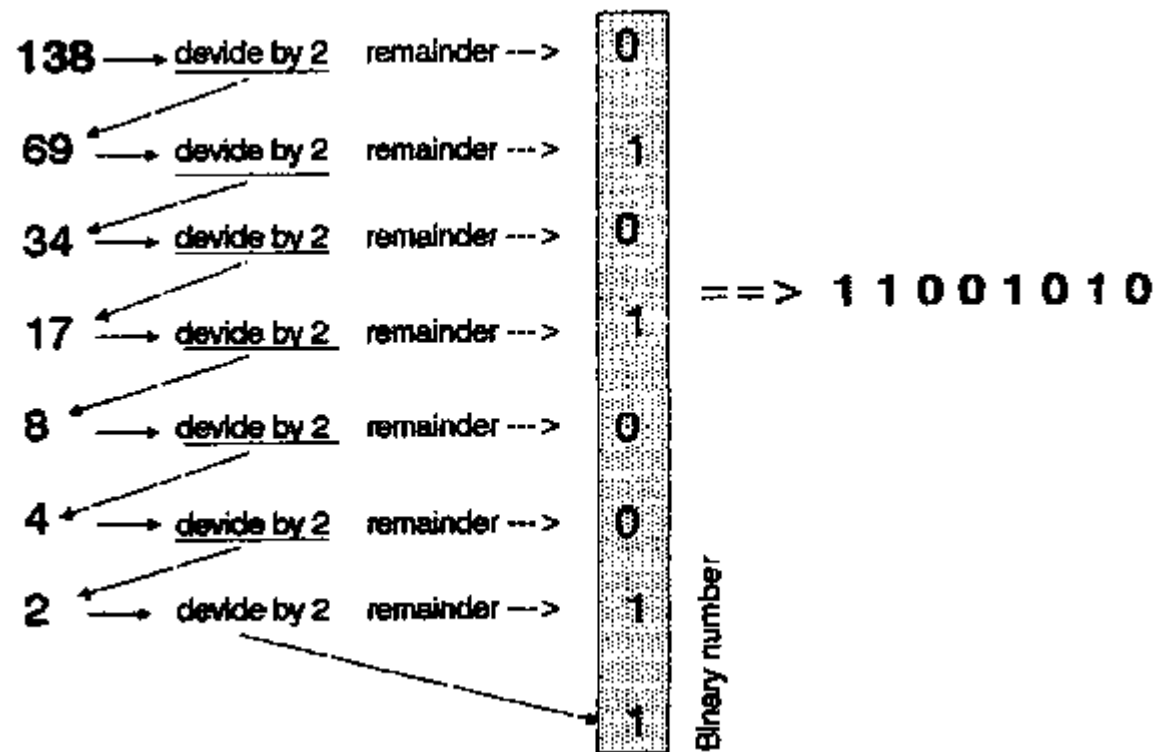
(*) fonte <http://www.oxfordmathcenter.com/drupal7/node/487>



Número de dígitos binários

Número de dígitos binários

Crie uma função que calcula o **número** mínimo de dígitos binários para representar um número inteiro decimal positivo **n**.



138 é representado com, no mínimo, 8 dígitos binários

Número de dígitos binários

$$Num(n) = \begin{cases} 1 & , \text{ se } n = 0 \text{ ou } n = 1 \\ 1 + Num\left(\frac{n}{2}\right) & , \text{ caso contrario} \end{cases}$$

Número de dígitos binários

$$Num(n) = \begin{cases} 1 & , \text{ se } n = 0 \text{ ou } n = 1 \\ 1 + Num\left(\frac{n}{2}\right) & , \text{ caso contrario} \end{cases}$$

```
1  #include <stdio.h>
2
3  int bin(int n) {
4      if (n==1 || n==0)
5          return 1;
6      else
7          return bin(n/2) + 1;
8  }
9
10 int main() {
11     int num;
12     scanf("%d", &num);
13     printf("%d\n", bin(num));
14 }
```

Número de vezes em que a função **bin** é chamada?

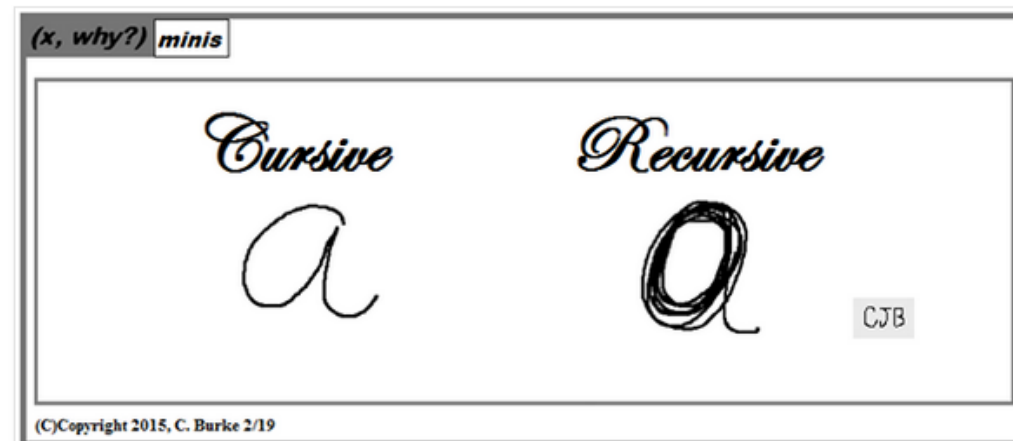
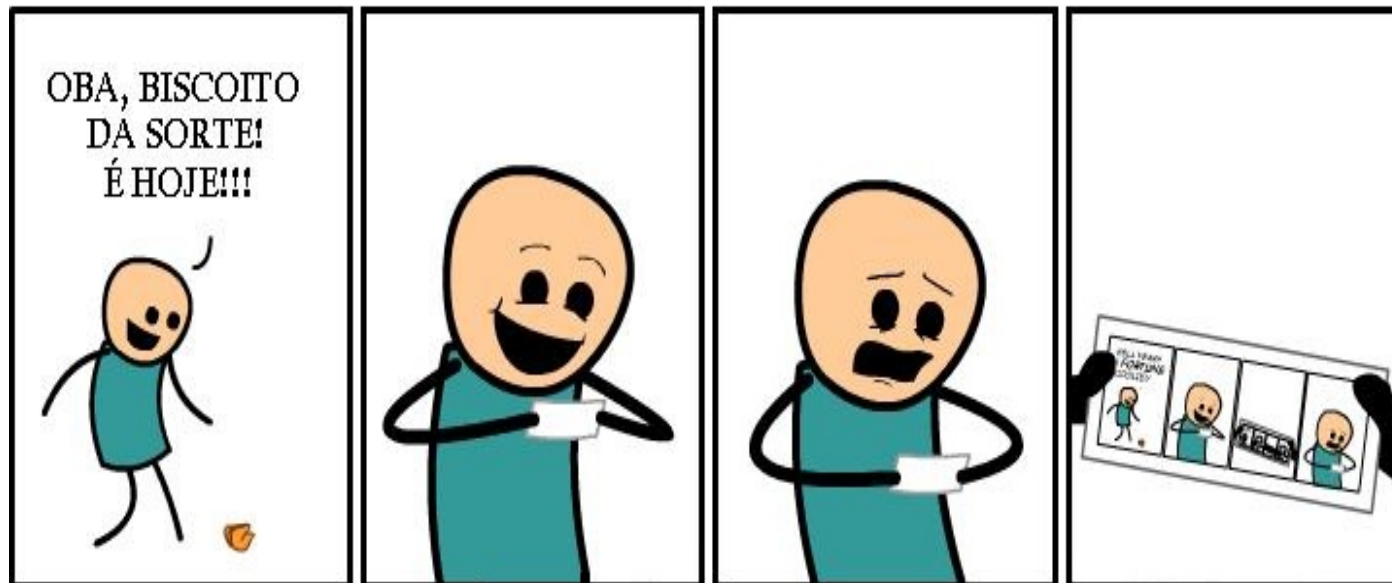
Número de dígitos binários

$$Num(n) = \begin{cases} 1 & , \text{ se } n = 0 \text{ ou } n = 1 \\ 1 + Num\left(\frac{n}{2}\right) & , \text{ caso contrario} \end{cases}$$

```
1  #include <stdio.h>
2
3  int bin(int n) {
4      if (n==1 || n==0)
5          return 1;
6      else
7          return bin(n/2) + 1;
8  }
9
10 int main() {
11     int num;
12     scanf("%d", &num);
13     printf("%d\n", bin(num));
14 }
```

Número de vezes em que a função **bin** é chamada? **floor(log2(n))**

Recursividade é uma das coisas mágicas e interessantes em Lógica de Programação





Teste de avaliação

Questão 1

- Qual é o resultado da execução das seguintes funções para n=5?

```
void conta1(int n) {  
    if (n>0) {  
        printf("\n%d", n);  
        conta1(n-1);  
    }  
}
```

```
void conta2(int n) {  
    if (n>0) {  
        conta2(n-1);  
        printf("\n%d", n);  
    }  
}
```

Resposta de conta1 para n=5

5
4
3
2
1

Resposta de conta2 para n=5

1
2
3
4
5

Questão 2

- Indique o que devolvem as funções F1 e F2 para valores de: a=2, k=5.
- Construa sua representação matemática recursiva.
- Descreva em português o que calcula cada função.

```
int f1(int a, int k) {  
    if (k==1)  
        return a;  
    else  
        return a*f1(a, k-1);  
}
```

```
int f2(int a, int k) {  
    int x;  
  
    if (k==1)  
        return a;  
    else {  
        x = f2(a, k/2);  
        if (k%2==0)  
            return x*x;  
        else  
            return x*x*a;  
    }  
}
```

$$F1(a, k) = \begin{cases} a & , \text{ se } k = 1 \\ a \times F1(a, k-1) & , \text{ caso contrario} \end{cases}$$

$$F2(a, k) = \begin{cases} a & , \text{ se } k = 1 \\ F2(a, \frac{k}{2})^2 & , \text{ se } k \text{ é par} \\ a \times F2(a, \frac{k}{2})^2 & , \text{ se } k \text{ é impar} \end{cases}$$

Ambas as funções calculam a^k .
Para a=2, e b=5 a função devolve 32.

F2 é mais eficiente!

Questão 3

- Descreva em português o que calcula a função M.
- Qual o número total de comparações?

```
1  #include <stdio.h>
2
3  int M(int v[], int n) {
4      if (n==1)
5          return v[0];
6      else {
7          int x;
8          x = M(v, n-1);
9          if (x <= v[n-1])
10             return x;
11          else
12             return v[n-1];
13      }
14 }
15
16 int main() {
17     int v[] = {1,2,3,4,5,6,7,-5,100};
18     printf("%d\n", M(v,9));
19 }
```

A função M, dada um vetor $v[0, \dots, n-1]$ de n elementos devolve o menor valor contido em v .

Número de comparações = n

Questão 4

- Escreva uma versão iterativa da função M.

```
int MIter(int v[], int n) {
    int i;
    int min = v[0];
    for (i=1; i<n; i++){
        if (min>v[i])
            min = v[i];
    }
    return min;
}
```

Versão iterativa

Recursão

- Uma função recursiva é aquela que se chama a si mesma (obrigatoriamente)?

```
1  #include <stdio.h>
2
3  long int Fib(int n) {
4      if (n==0)
5          return 0;
6      if (n==1)
7          return 1;
8      else
9          return Fib(n-1) + Fib(n-2);
10 }
11
12 int main() {
13     int num;
14     scanf("%d", &num);
15     printf("%ld\n", Fib(num));
16 }
```

Recursão

Uma função recursiva não necessariamente é aquela que se chama a si mesma

```
int F1(parametros)
{
    .
    :
    .
    F2(parametros)
    .
    .
}
```

```
int F2(parametros)
{
    .
    :
    .
    F1(parametros)
    .
    .
}
```

$F1(\dots) \rightarrow F2(\dots) \rightarrow F3(\dots) \rightarrow \dots \rightarrow F1(\dots)$

Análise de algoritmos recursivos

- Para fazer a análise é necessário obter a relação de recorrência.
- A fórmula fechada (resolução) da recorrência pode ser obtida por diferentes abordagens, por exemplo:
 - Árvore de recorrência
 - Método mestre
 - Funções geradoras

<http://ustandout.com>



YUSUF - CHEMIST
COBB - EXTRACTOR
ARIADNE - ARCHITECT
SAITO - TOURIST
ARTHUR - POINTMAN
FISCHER - MARK
EAMES - FORGER

