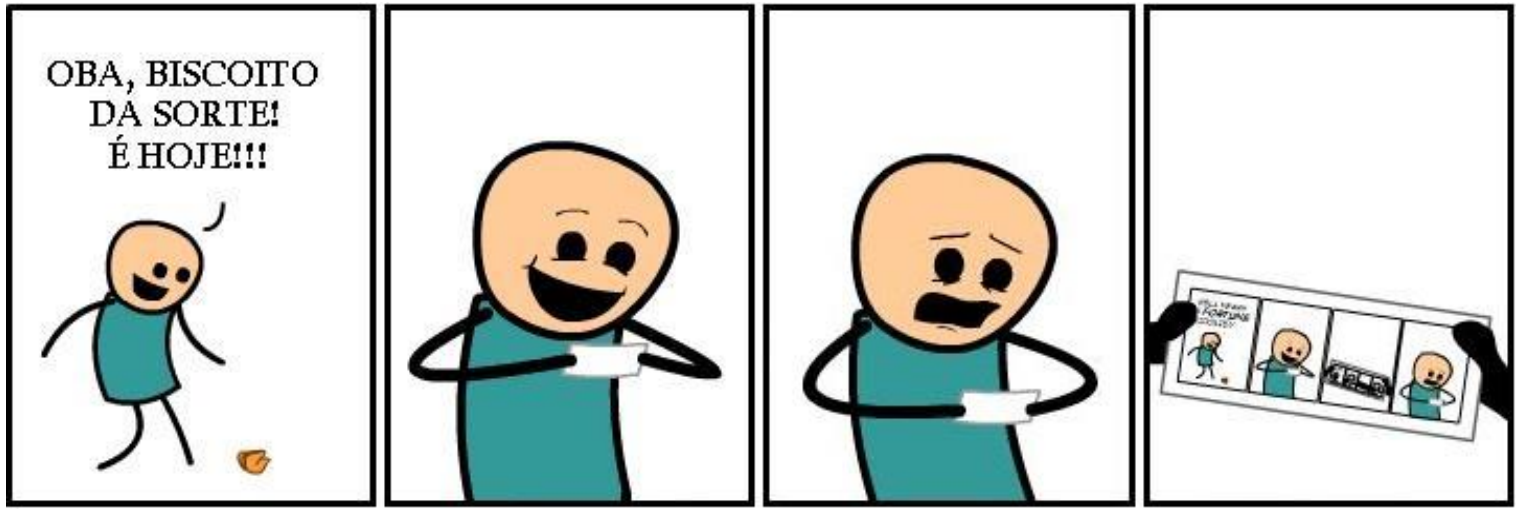
Introdução: Análise de Complexidade de Algoritmos (4)

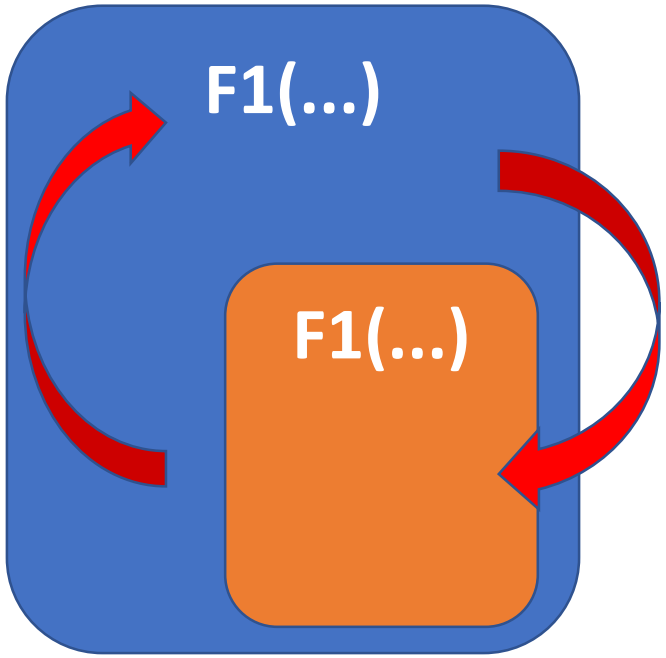
Prof. Renê Rodrigues Veloso

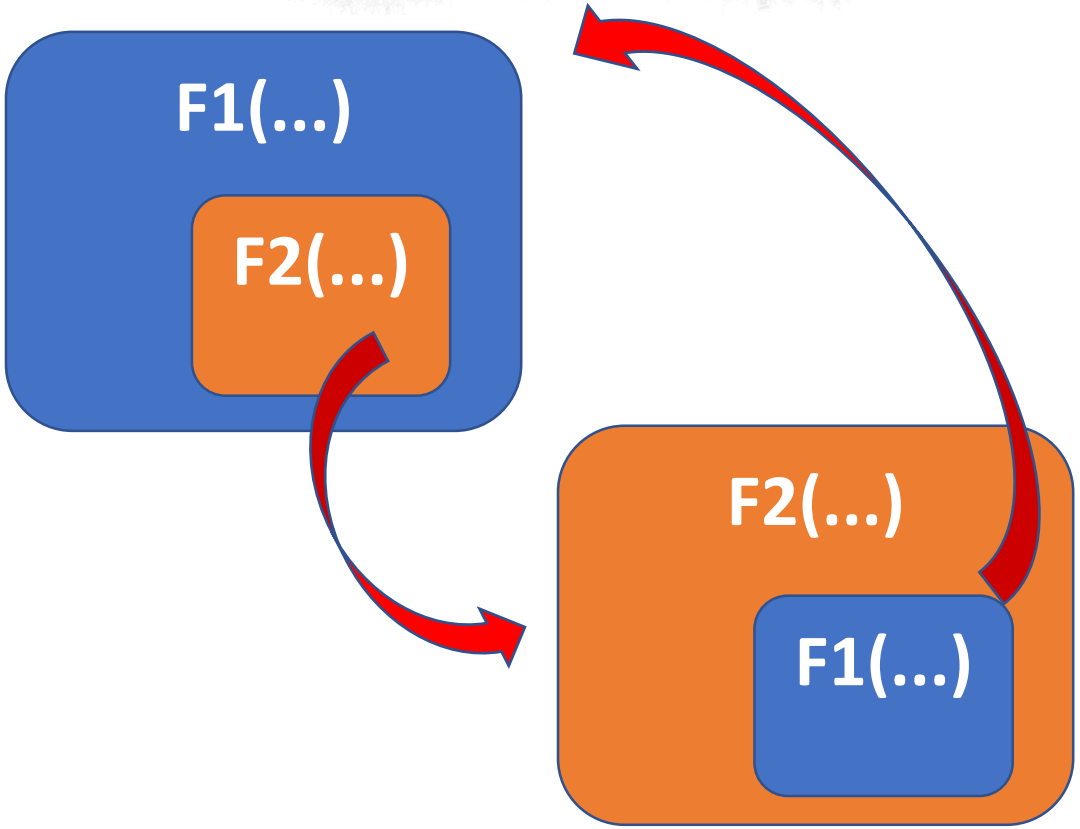
Recursividade



- Uma função recursiva é uma função definida em termos dela mesma.
  - ...ou que chama a si mesma, diretamente ou não.

Recursividade: direta e indireta





A função fatorial

$$n! = \begin{cases} 1 & se\ n = 0 \\ n.(n-1)! & se\ n \geq 1 \end{cases}$$

```
int fatorial(int n) {
    if ( n == 0)
        return 1;
    return n*fatorial(n-1);
}
```

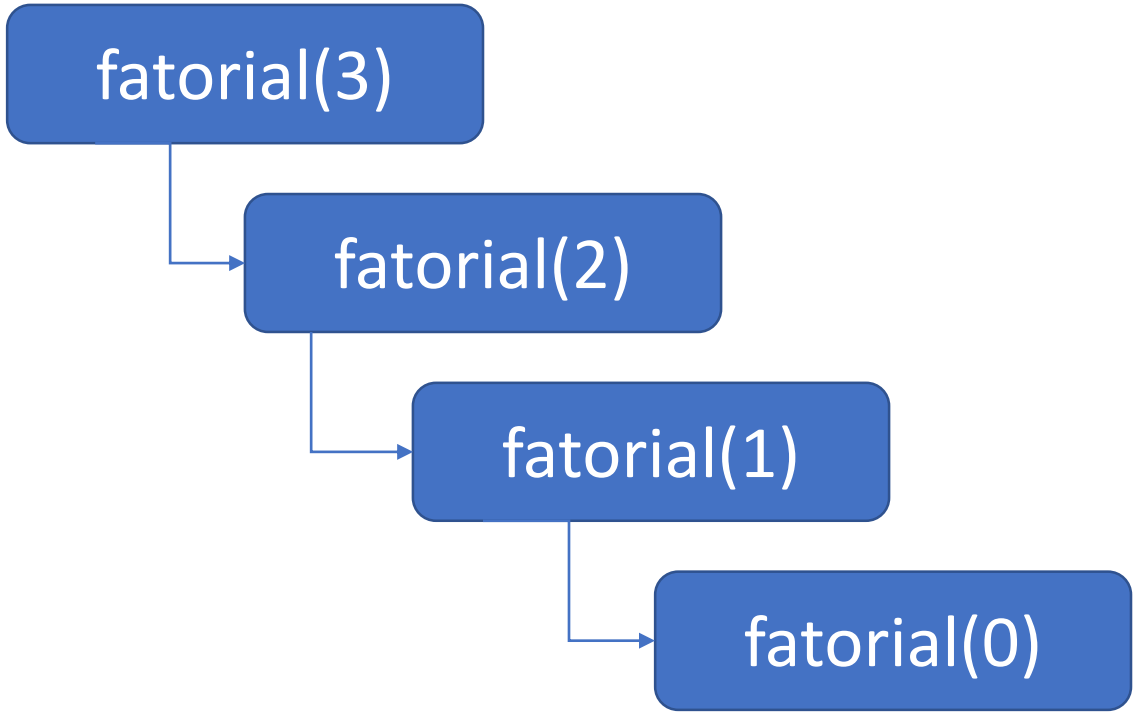
Caso base ou ponto-de-parada

Caso recursivo ou
Passo de recursão

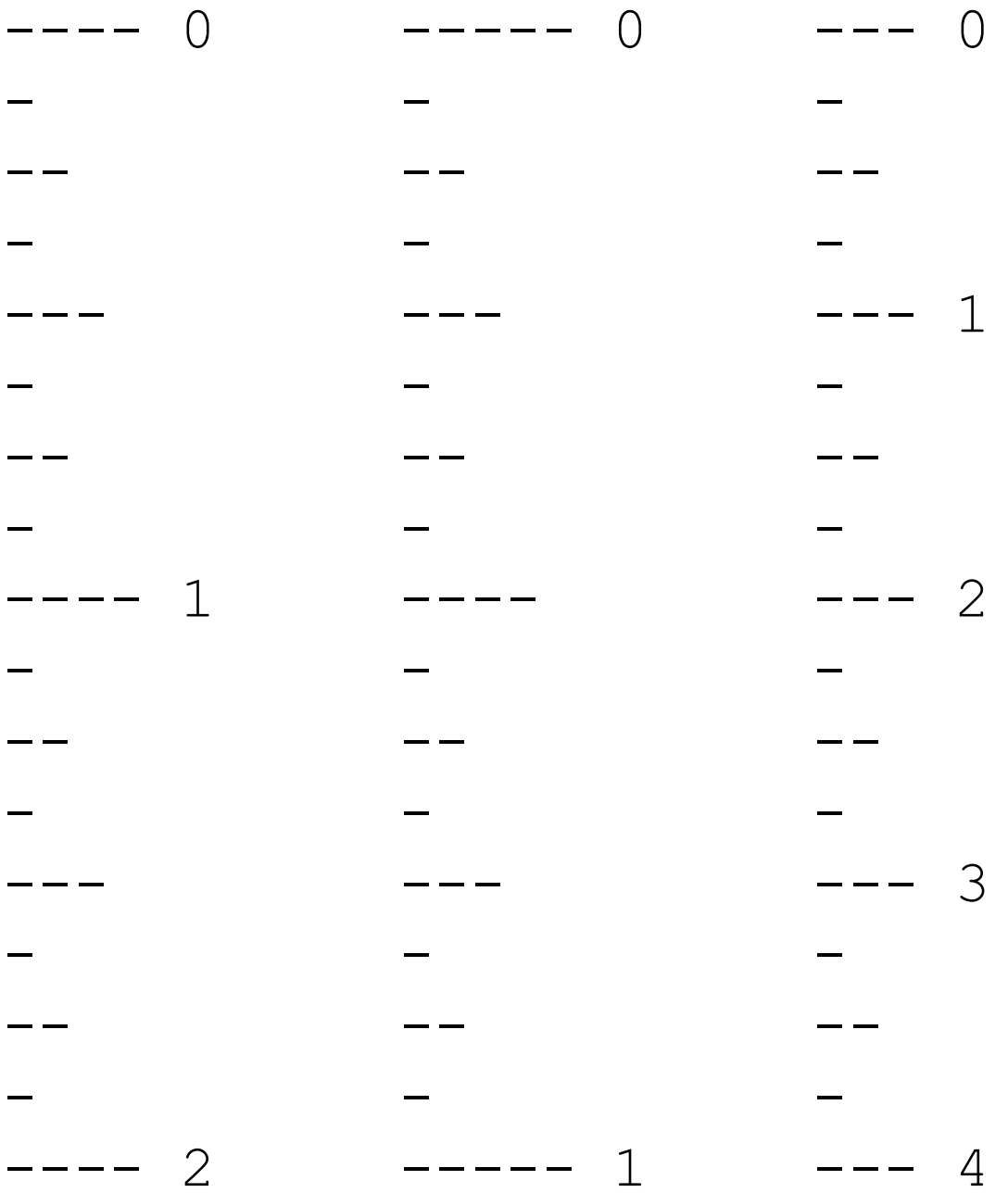
A função fatorial

$$n! = \begin{cases} 1 & se\ n = 0 \\ n.(n-1)! & se\ n \geq 1 \end{cases}$$

```
int fatorial(int n) {
    if ( n == 0)
        return 1;
    return n*fatorial(n-1);
}
```



Um exemplo complicado: régua inglesa



Um exemplo complicado: régua inglesa

- Um intervalo com uma marca central de tamanho L >= 1 é composto da seguinte forma:

  - Um intervalo com uma marca central de tamanho L -1;
  - Uma única marca de tamanho L;
  - Um intervalo com uma marca central de tamanho L-1;

```
desenhaUmaMarca(tamanho, rotulo) {
     para i =0 até i < tamanho:
          imprima('-')
      se rotulo >=0:
          imprima(rotulo)
}

desenhaMarcas(tamanho) {
     se tamanho > 0:
          desenhaMarcas(tamanho-1)
          desenhaUmaMarca(tamanho,-1)
          desenhaMarcas(tamanho-1)
}


desenhaRegua(comprimento, tamanhomarca) {
       desenhaUmaMarca(tamanhomarca, 0)
       para i = 1 até i<=comprimento:
            desenhaMarcas(tamanhomarca-1)
            desenhaUmaMarca(tamanhomarca, i)
}
```

Formas de recursão: linear e n-ária

- Recursão linear
  - Forma mais simples de recursão
  - Nó máximo, uma chamada recursiva em cada entrada de função
  - Usada em problemas no formato: processamento do primeiro ou último elemento mais um conjunto restante que tem a mesma estrutura do conjunto original
- Recursão n-ária
  - Quando são feitas duas ou mais chamadas recursivas em cada entrada
  - Usada em problemas no formato: dividir-e-conquistar; duas ou mais partes do mesmo problema, onde cada parte possui a mesma estrutura do original

Recursão linear

- Exemplo a)
  - Somando elementos de um arranjo de maneira recursiva

A =

4	2	3	6	5	1	0	6	4	2

int somaLinear(int A[], int n){...}

Recursão linear

- Exemplo b)
  - Invertendo os elementos de um arranjo de maneira recursiva

A =

1	2	3	4	5	6	7	8	9	10

int inverteVetor(int A[], int i, int j){...}

Recursão linear

- Exemplo c)
  - Imprimindo os elementos de um arranjo de maneira recursiva

A =

1	2	3	4	5	6	7	8	9	10

int imprimeVetor(int A[], int n){...}

Recursão Binária

- Exemplo a)
  - Somando elementos de um arranjo de maneira recursiva

A =

4	2	3	6	5	1	0	6	4	2

Recursão Binária

- Exemplo b)
  - Encontrando o maior elemento de maneira recursiva

A =

4	2	3	6	5	1	0	6	4	2

Exercícios

1. Apresente o resultado da seguinte função:

   Para as chamadas f1(0), f1(1) e f1(5)

```
int f1(int n)
{
    if (n == 0)
       return (1);
    else
       return(n * f1(n-1));
}
```

Exercícios

2. Apresente o resultado da seguinte função:

   Para as chamadas f2(0), f2(1) e f2(5)

```
int f2(int n)
{
    if (n == 0)
       return (1);
    if (n == 1)
       return (1);
    else
       return(f2(n-1)+ 2 * f2(n-2));
}
```

Exercícios

3. Verifique o que as funções a seguir imprimem e retornam:

(a)
```
func (int n)
{
     if (n == 0)
          printf("fim");
     else
     {
          printf("%d ",n);
          func(n-1);
     }
}
```

(b)
```
func (int n)
{
     if (n == 0)
          printf("fim");
     else
     {
          func(n-1);
          printf("%d ",n);
     }
}
```

Exercícios

3. Verifique o que as funções a seguir imprimem e retornam:

(c)
```
func (int n)
{
     if (n == 0)
          printf("fim");
     else
     {
          printf("%d ",n);
          func(n-1);
          printf("%d ",n);
     }
}
```

(d)
```
func (int n)
{
     if (n == 0)
          printf("fim");
     else
     {
          func(n-1);
          printf("%d ",n);
          func(n-1);
     }
}
```

Exercícios

4. Escreva algoritmos recursivos para os seguintes problemas:
   a) Multiplicação de dois números inteiros positivos através de somas sucessivas. Ex.: 6*4 = 4+4+4+4+4+4
   b) Soma de dois números inteiros positivos através incrementos. Ex.: 3+2 = ++(++3)
   c) Dada uma string de tamanho n, imprimir todas as combinações de seus caracteres. Ex.: “ABC” = ABC, ACB, BAC, BCA, CAB, CBA