

Universidade Federal do Rio Grande do Norte
Instituto Metr pole Digital

AULA 08

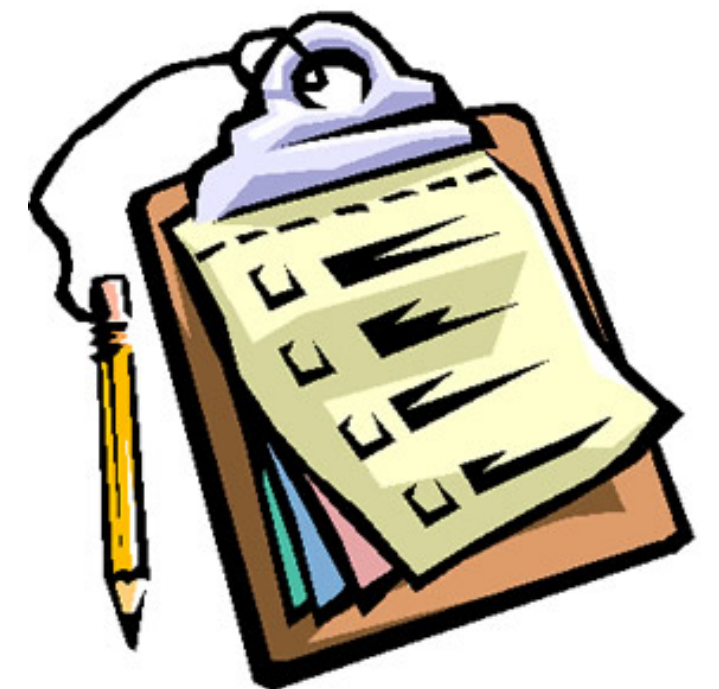
Estrutura de dados b sico I (EDB1)

Prof. Msc. Janiheryson Felipe (Felipe)

Natal, RN
2023

OBJETIVO DA AULA

- Apresentar os algoritmos recursivos;
 - Conhecer a recursão e os processos recursivos
 - Conhecer alguns dos principais algoritmos recursivos

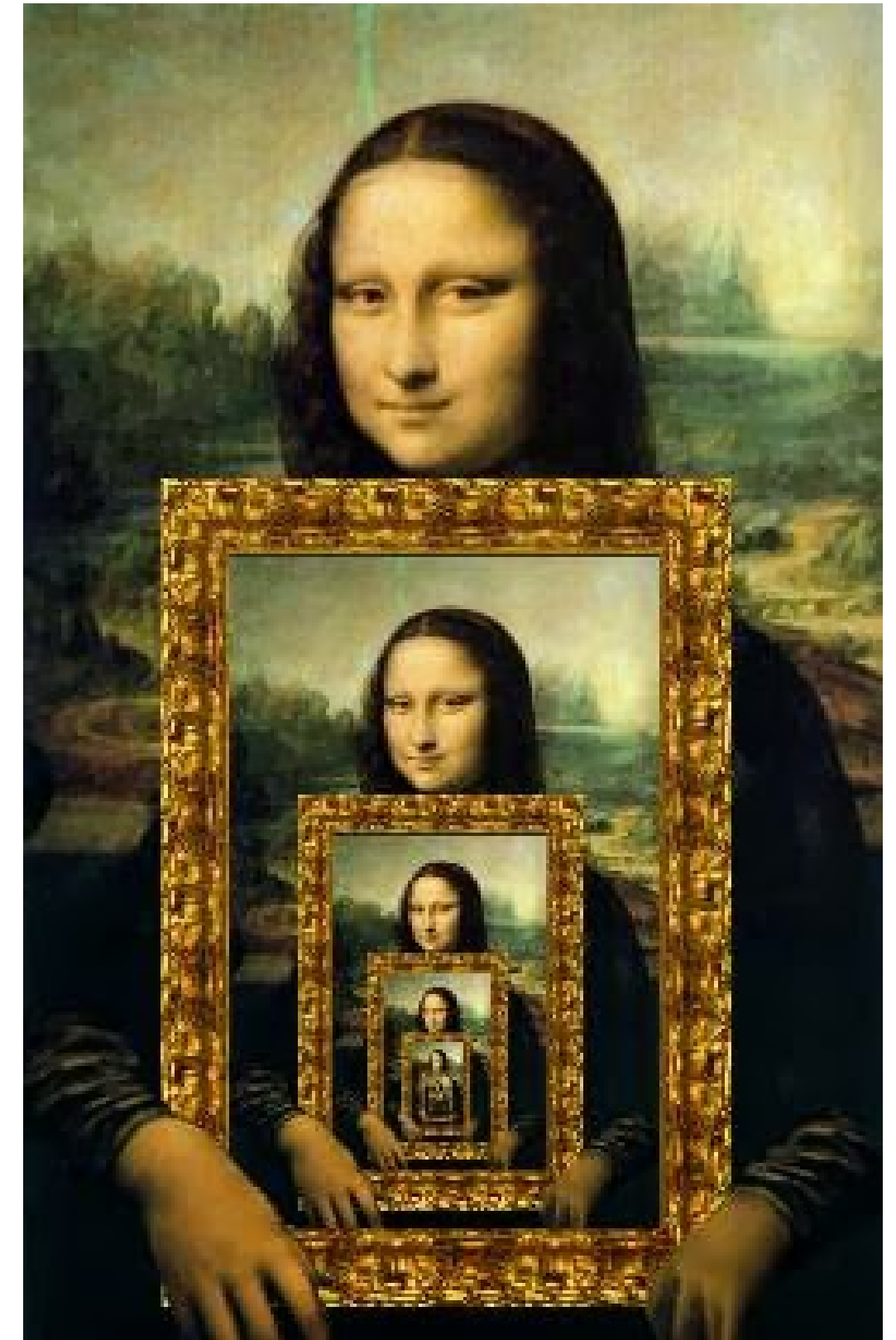




RECURSÃO

QUESTÕES INICIAIS

- O que é Recursão?
- Quais são os tipos de recursão?
- Quando a recursão é útil?
- A recursão é sempre melhor que a iteração?
- Quais funções iterativas podem ser implementadas de forma recursiva?



O QUE É RECURSÃO

Recursão é um conceito em programação de computadores e matemática em que uma função chama a si mesma para resolver um problema (diretamente) ou há uma chamada mútua entre duas ou mais funções (indiretamente)..



O QUE É RECURSÃO

A recursão é uma técnica em que um problema é dividido em subproblemas menores e a solução é encontrada através da repetição da mesma função com os subproblemas menores até que o problema original seja resolvido.



RECURSÃO MATEMÁTICA

$$f(x) = \begin{cases} 1 & \text{se } x = 0 \\ x \cdot f(x-1) & \text{se } x > 0 \end{cases}$$

O que a formula acima representa?

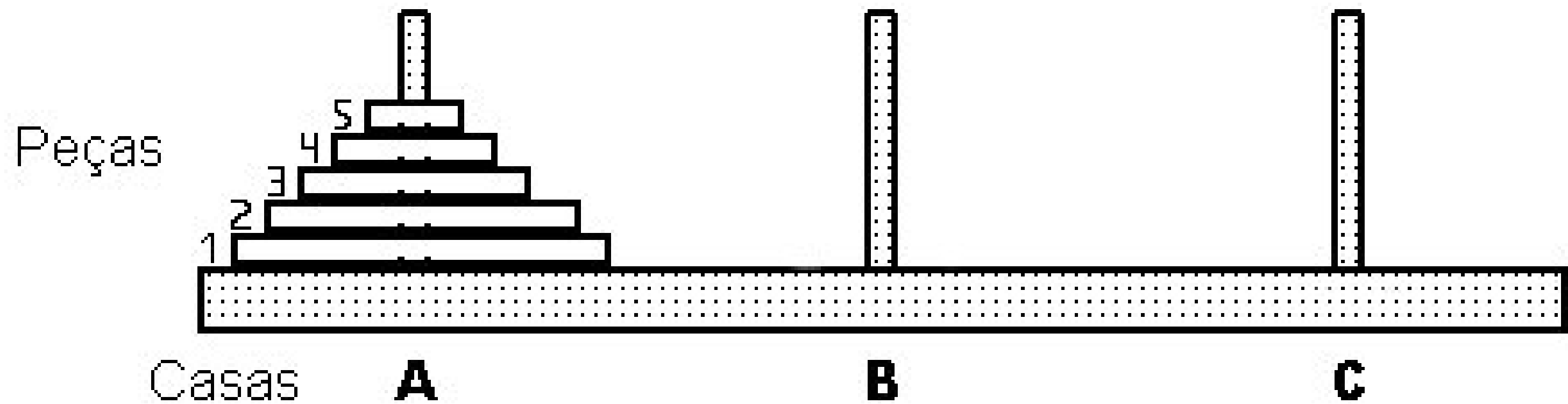
RECURSÃO MATEMÁTICA

$$f(x) = \begin{cases} 1 & \text{se } x = 0 \\ x \cdot f(x-1) & \text{se } x > 0 \end{cases}$$

Esta função fornece o fatorial do número natural x .

Então $f(x) = x!$

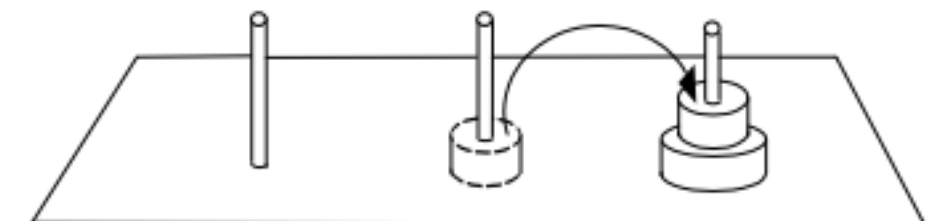
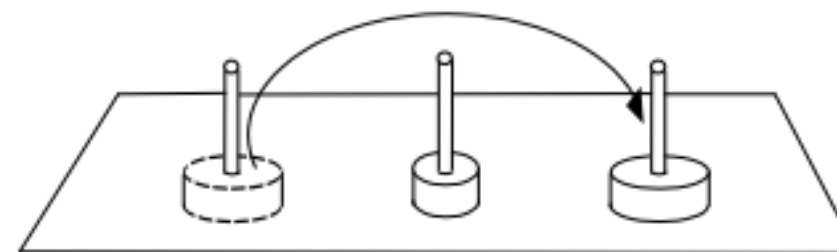
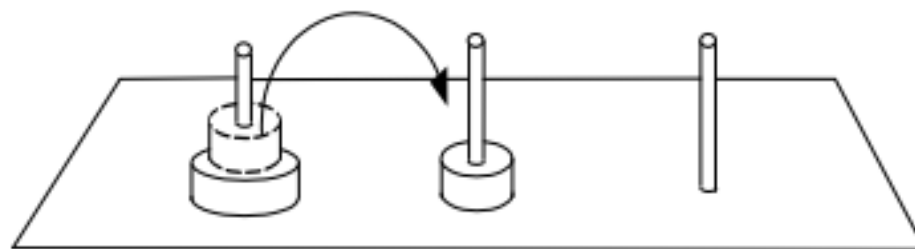
RECURSÃO MATEMÁTICA



A torre de Hanoi pode ser
implementada de forma recursiva?

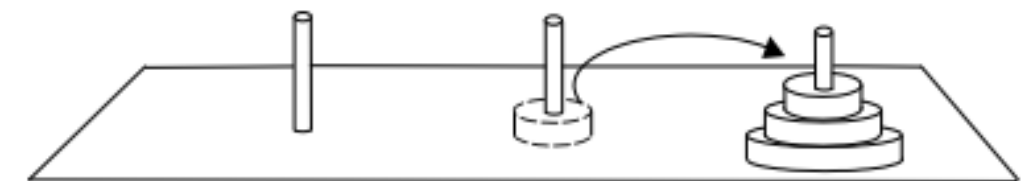
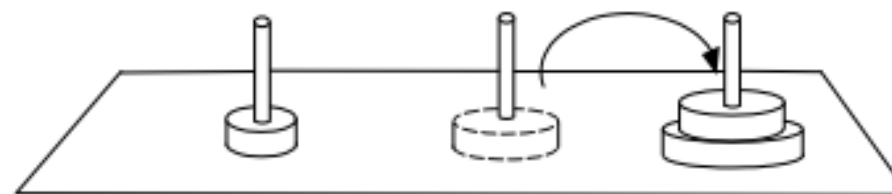
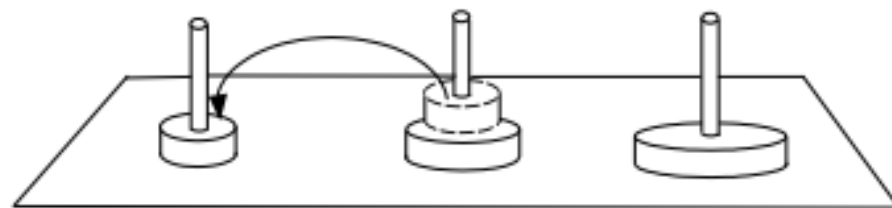
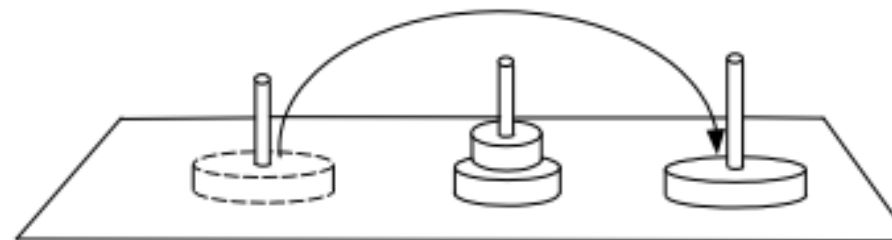
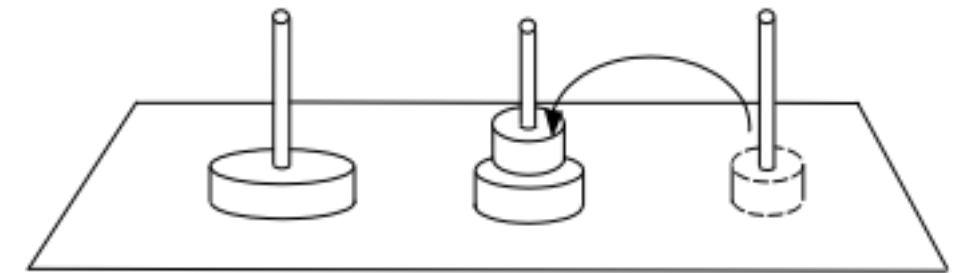
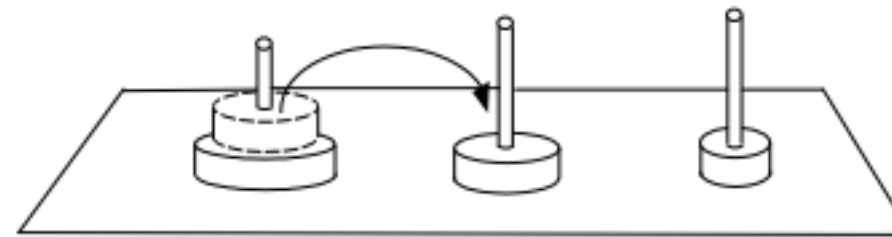
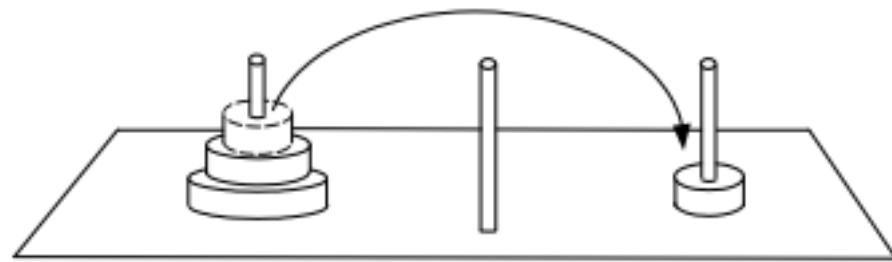
RECURSÃO MATEMÁTICA

- Se por acaso a torre tivesse apenas 1 disco é obvio que apenas um movimento seria suficiente então temos $f(1) = 1$
- Se a torre tivesse 2 discos teremos que fazer 3 movimentos $f(2) = 3$



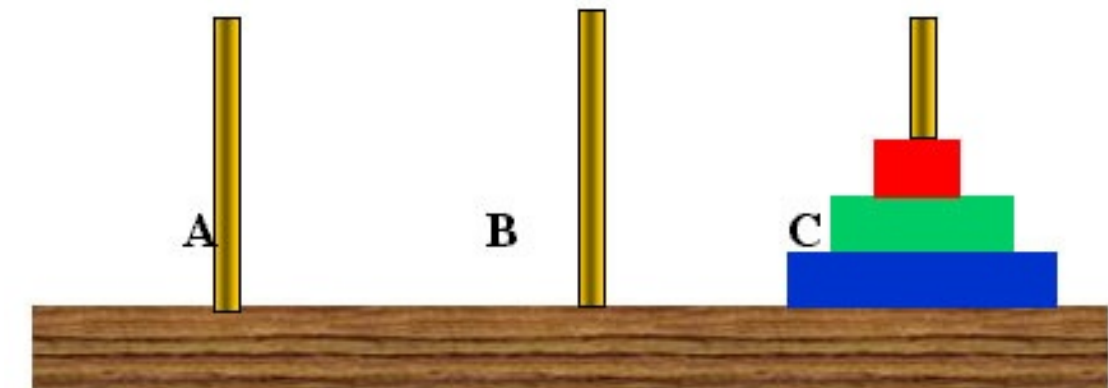
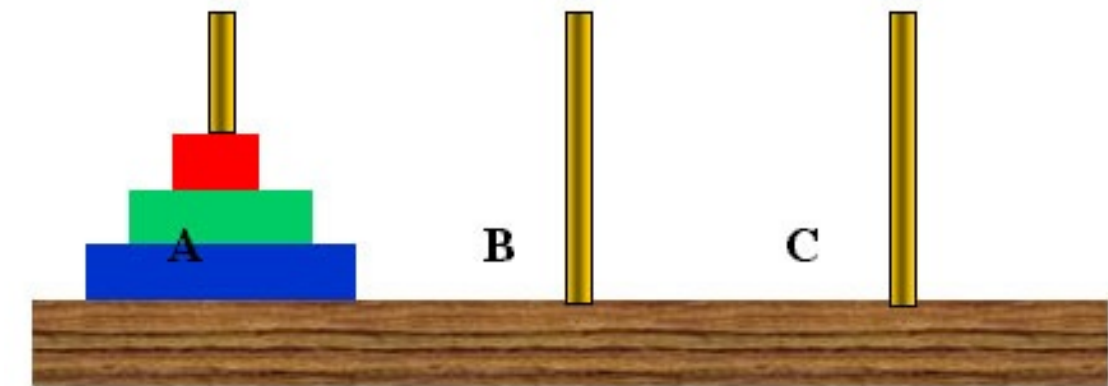
RECURSÃO MATEMÁTICA

- Se a torre tivesse 3 discos teremos que fazer 7 movimentos
 $f(3) = 7$



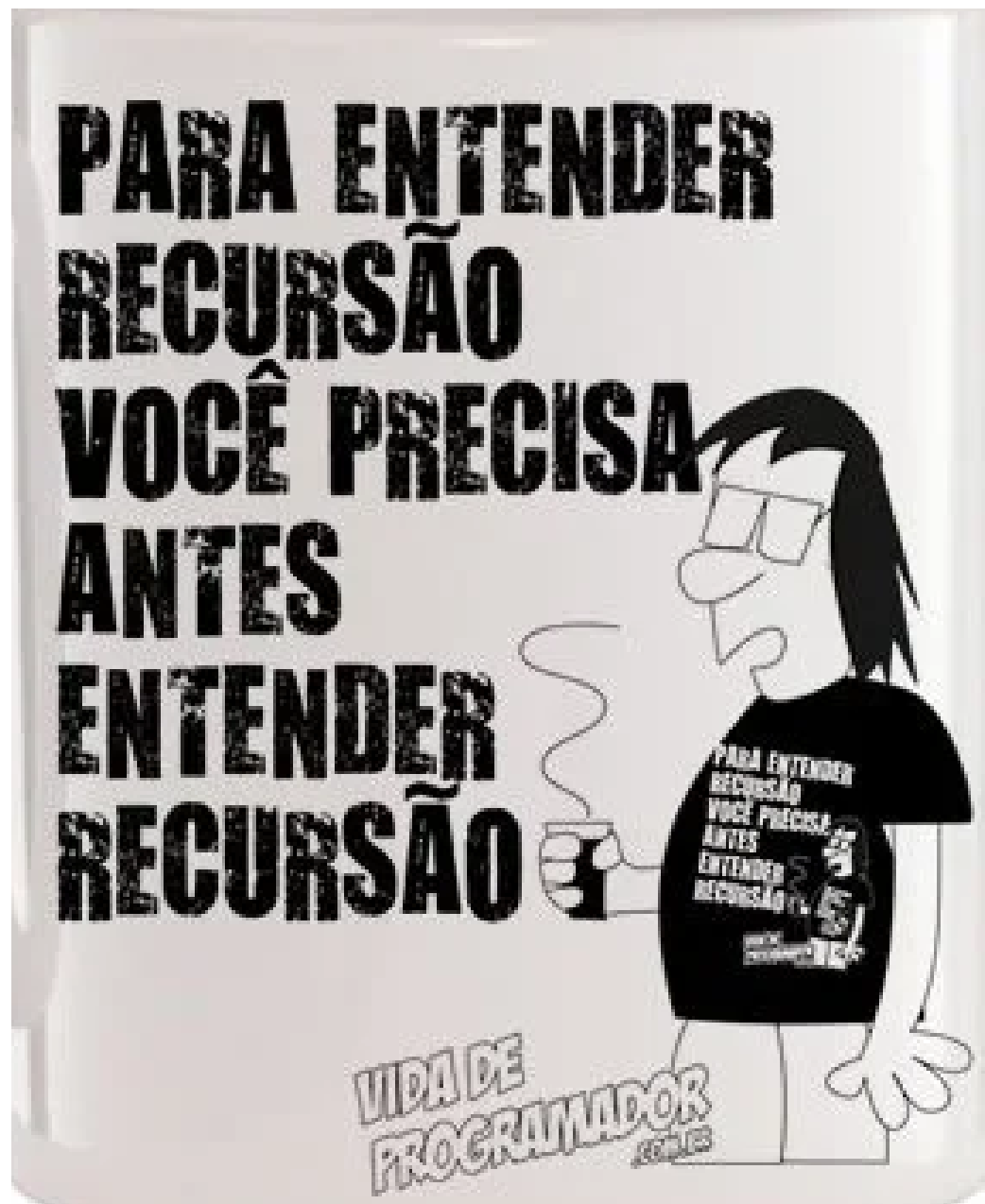
RECURSÃO MATEMÁTICA

- $f(1) = 1$
- $f(2) = 2.f(1) + 1 = 2.\textcolor{red}{1} + 1 = 3$
- $f(3) = 2.f(2) + 1 = 2.\textcolor{red}{3} + 1 = 7$
- $f(4) = 2.f(3) + 1 = 2.\textcolor{red}{7} + 1 = 15$
- $f(5) = 2.f(4) + 1 = 2.\textcolor{red}{15} + 1 = 31$



$$f(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2.f(n-1) + 1 & \text{se } n > 1 \end{cases}$$

ELEMENTOS DA RECURSÃO



Um algoritmo recursivo é definido em função de um caso base e passos recursivos:

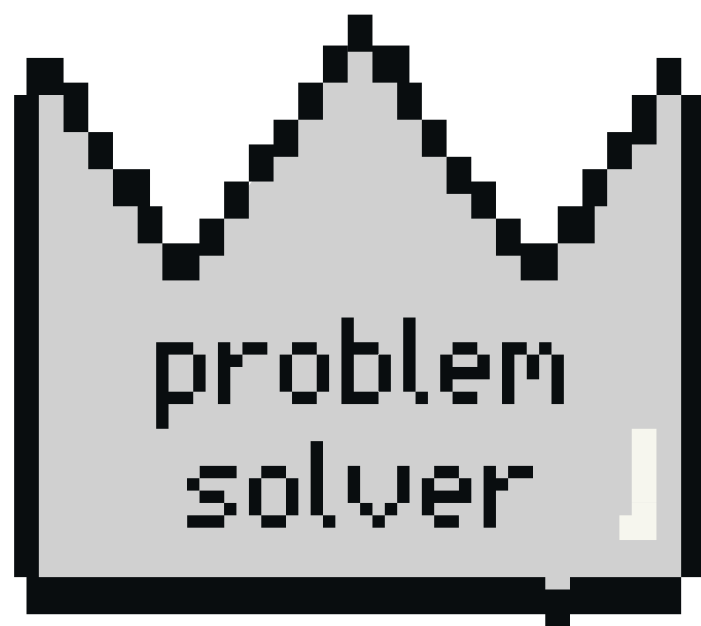
- **Caso Base (condição de parada):** Instancia mais simples do problema.
- **Passo recursivo:** Redução do problema em problemas menores.

CONSIDERE O SEGUINTE PROBLEMA...

Crie um algoritmo que, dado um inteiro n , retorne a soma de 1 até n .

$$F(n) = 1 + 2 + 3 + 4 + \dots + n-2 + n-1 + n$$

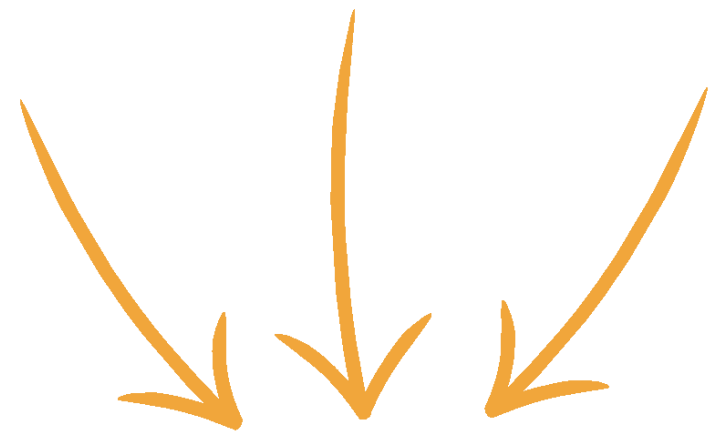
$$F(n) = F(n-1) + n$$



CONSIDERE O SEGUINTE PROBLEMA...

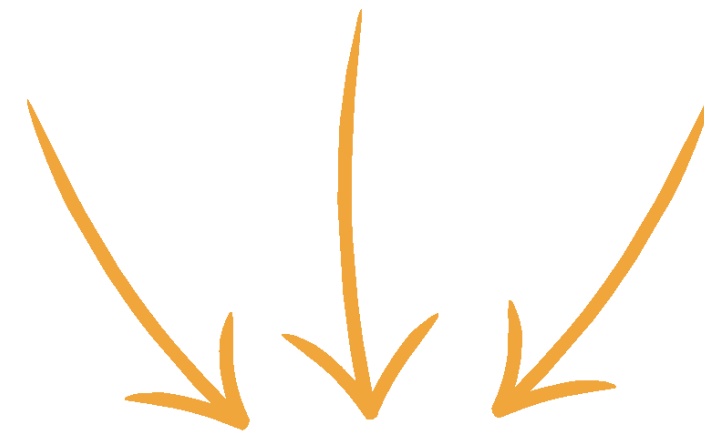
$$F(n) = 1 + 2 + 3 + 4 + \dots + n-2 + n-1 + n$$

Caso base



```
if(n == 0) {  
    return;  
}
```

Passo recursivo



$$F(n) = n + F(n-1)$$

IMPLEMENTAÇÃO DO CÓDIGO

```
#include <stdio.h>
```

```
int soma(n){
```

```
    if(n == 0){
```

```
        return 0;
```

```
    }
```

```
    return n + soma(n-1);
```

```
}
```

```
int main(){
```

```
    printf("Soma é %d", soma(10));
```

```
}
```



Caso base



Passo recursivo

COMPORTAMENTO EM MEMÓRIA

```
#include <stdio.h>

int soma(n){
    if(n == 0){
        return 0;
    }
    return n + soma(n-1);
}

int main(){
    printf("Soma é %d", soma(10));
}
```

Pilha de memória

soma(0)
soma(1)
soma(2)
soma(3)
soma(4)
soma(5)
soma(6)
soma(7)
soma(8)
soma(9)
soma(10)
main()

COMPORTAMENTO EM MEMÓRIA

```
#include <stdio.h>

int soma(n){
    if(n == 0){
        return 0;
    }
    return n + soma(n-1);
}

int main(){
    printf("Soma é %d", soma(10));
}
```

Pilha de memória

0
0+1
0+1+2
0+1+2+3
0+1+2+3+4
0+1+2+3+4+5
0+1+2+3+4+5+6
0+1+2+3+4+5+6+7
0+1+2+3+4+5+6+7+8
0+1+2+3+4+5+6+7+8+9
0+1+2+3+4+5+6+7+8+9+10
main()

COMPORTAMENTO EM MEMÓRIA

```
#include <stdio.h>

int soma(n){
    if(n == 0){
        return 0;
    }
    return n + soma(n-1);
}

int main(){
    printf("Soma é %d", soma(10));
}
```

Pilha de memória

0
1
3
6
10
15
21
28
36
45
55
main()

TIPOS DE RECURSÃO


Existem basicamente cinco tipos de recursão:

A hand-drawn circle with multiple overlapping strokes, giving it a sketchy appearance.

**Recursão
direta**

A hand-drawn circle with multiple overlapping strokes, giving it a sketchy appearance.

**Recursão
de cauda**

A hand-drawn circle with multiple overlapping strokes, giving it a sketchy appearance.

**Recursão
parcial**

A hand-drawn circle with multiple overlapping strokes, giving it a sketchy appearance.

**Recursão
indireta**

A hand-drawn circle with multiple overlapping strokes, giving it a sketchy appearance.

**Recursão
não cauda**

RECURSÃO DIRETA

A recursão direta ocorre quando uma função chama a si mesma diretamente, sem envolver outras funções. Um exemplo simples de recursão direta é o cálculo do fatorial de um número:

```
#include <stdio.h>
```

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

```
int main(){  
    printf("Soma é %d", fatorial(5));  
}
```

RECURSÃO INDIRETA

A recursão indireta ocorre quando uma função chama outra função, que, por sua vez, chama a primeira função novamente, criando um ciclo de chamadas recursivas.

```
void funcao1(int n);

void funcao2(int n) {
    if (n > 0) {
        printf("%d ", n);
        funcao1(n - 1);
    }
}

void funcao1(int n) {
    if (n > 0) {
        printf("%d ", n);
        funcao2(n - 1);
    }
}
```


RECURSÃO DE CAUDA

Neste tipo a chamada recursiva é a última operação a ser executada antes de retornar o resultado. Isso permite que o compilador otimize a chamada recursiva, convertendo-a em um loop de uma única iteração.

```
void exhibeSequencia(int n){  
    if (n == 1) {  
        return 1;  
    }else{  
        printf("%d ", n);  
        exhibeSequencia(n - 1);  
    }  
}
```

RECURSÃO DE NÃO CAUDA

Neste tipo a chamada recursiva não é a última operação a ser executada. Gasta um pouco mais de memória, pois armazena também linhas extras.

```
void exibeSequencia(int n){  
    if (n == 1) {  
        return 1;  
    }else{  
        exibeSequencia(n - 1);  
        printf("%d ", n);  
    }  
}
```

RECURSÃO PARCIAL

Ocorre quando uma função recursiva é definida apenas para alguns casos base e não para todos os casos possíveis. Nesse caso, a função pode entrar em um loop infinito se for chamada com uma entrada que não está definida pela recursão.

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

ALGORITMO DE BUSCA BINÁRIA RECURSIVA

```
int busca_binaria_recursiva(int *vetor, int inicio, int fim, int valor) {  
    if (inicio > fim) {  
        return -1;  
    }  
    int meio = (inicio + fim) / 2;  
    if (vetor[meio] == valor) {  
        return meio;  
    } else if (vetor[meio] < valor) {  
        return busca_binaria_recursiva(vetor, meio + 1, fim, valor);  
    } else {  
        return busca_binaria_recursiva(vetor, inicio, meio - 1, valor);  
    }  
}
```

ALGORITMO DE BUSCA BINÁRIA ITERATIVA

```
int busca_binaria_iterativa(int *vetor, int tamanho, int valor) {  
    int inicio = 0;  
    int fim = tamanho - 1;  
  
    while (inicio <= fim) {  
        int meio = (inicio + fim) / 2;  
  
        if (vetor[meio] == valor) {  
            return meio;  
        } else if (vetor[meio] < valor) {  
            inicio = meio + 1;  
        } else {  
            fim = meio - 1;  
        }  
    }  
    return -1;  
}
```

QUESTÃO 01 - BANCA UFSM

Considere a seguinte função recursiva.

```
def func(arg):  
    if arg == 0:  
        return 0  
    else:  
        return (arg % 2) + 10 * func(arg//2)
```

Qual o valor retornado pela função acima, quando recebe como parâmetro o número 5?

A 101

B 111

C 100

D 010

E 011

QUESTÃO 02 - BANCA CESPE

Julgue o item subsequente, a respeito de algoritmos para ordenação e pesquisa e de programação recursiva.

☐ Certo

Uma função é dita recursiva quando, dentro dela, é feita uma ou mais chamadas a ela mesma.

☐ Errado

QUESTÃO 03 - BANCA FAPEEC

Considere a seguinte função recursiva:

função recursiva(x : inteiro): inteiro início

```
    se x = 1 então
        retorne -x
    senão
        retorne -5 * recursiva(x - 1) + x
    fimse
fimfuncao
```

-- (A) -143.

-- (B) -56.

-- (C) 143.

-- (D) 56.

-- (E) 164.

Qual é o valor retornado pela função se ela for chamada com $x = 4$?

QUESTÃO 04 - BANCA QUADRIX

A situação em que dois subprogramas fazem chamadas recíprocas, como, por exemplo, um subprograma P faz uma chamada a um subprograma J, que, por sua vez, faz uma chamada a P, é caracterizada como uma

- (A) recursividade direta.
- (B) recursividade indireta.
- (C) recursividade simples.
- (D) lista linear simples.
- (E) lista circular.

QUESTÃO 05 - BANCA AOCP

A recursividade é uma importante sub-rotina que pode auxiliar o analista de sistemas a resolver problemas mais complexos. Sabendo disso, assinale a alternativa em que esteja implementado corretamente um algoritmo recursivo.

QUESTÃO 05 - BANCA AOCP

A

```
função fat(x: inteiro): inteiro
  var i, aux: inteiro
  inicio
    aux <- 1
    para i de 1 até x faça
      aux <- aux * i
    fim_para
    fat <- aux
  fim
```

B

```
fat(x: inteiro): inteiro
  var i, fat_aux: inteiro
  inicio
    fat_aux <- 1
    para i de 1 até x faça
      fat_aux <- fat_aux * i
    fim_para
    fat <- fat_aux
  fim
```

QUESTÃO 05 - BANCA AOCP

C

```
função fat(x: inteiro): inteiro
início
    se x = 0 então
        fat <- 1
    senão
        fat <- x * fat(x - 1)
    fim_se
fim
```

E

```
função fat(x: inteiro): inteiro
var fat_aux: inteiro
início
    fat_aux <- 1
senão
    fat_aux <- x * fat_aux
fim_se
fim
```

QUESTÃO 06 - BANCA SUGEP

Considere a função recursiva 'func' definida por

$$\text{func}(1) = 1$$

$$\text{func}(n) = (n - 1) * \text{func}(n - 1)$$

Quais são os valores de $\text{func}(4)$ e $\text{func}(5)$, respectivamente?

--- **A** 24 e 120

--- **B** 12 e 24

--- **C** 6 e 24

--- **D** 1 e 2

--- **E** 2 e 6

PONDERAÇÕES IMPORTANTES

- A recursão é sempre a melhor implementação quando comparado com a interação???
- Quando vale a pena utilizar a recursão e quando não vale???



DÚVIDAS???

