



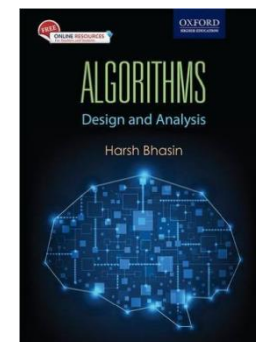
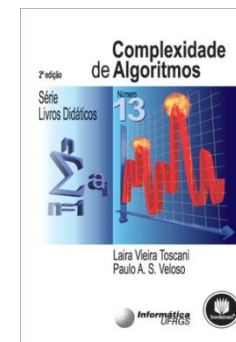
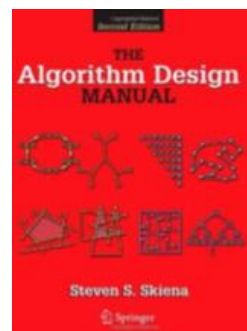
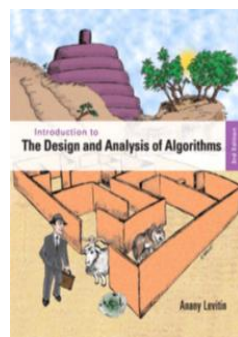
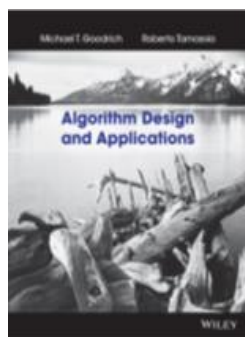
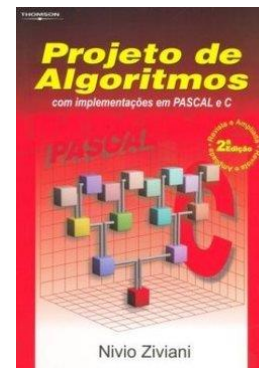
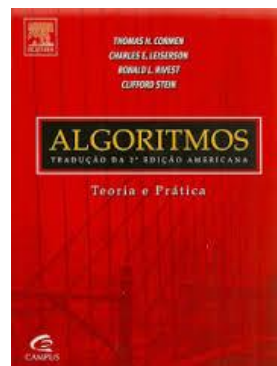
Unidade 6 – Tópicos de Funções Recursivas

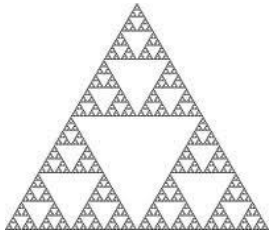


Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

Bibliografia

- **Algoritmos – Teoria e Prática – Cormen – Segunda Edição – Editora Campus, 2002**
- **Projeto de Algoritmos – Nivio Ziviani – Pioneira Informática - 1993**
- **Algorithm Design and Applications – Michael T. Goodrich, Roberto Tamassia, Wiley, 2015**
- **Introduction to the Design and Analysis of Algorithms – Anany Levitin, Pearson, 2012**
- **The Algorithm Design Manual – Steven S. Skiena, Springer, 2008**
- **Complexidade de Algoritmos – Série Livros Didáticos – UFRGS**
- **Algorithms – Design and Analysis – Harsh Bhasin – Oxford University Press – 2015**
- **Notas de Aulas – Prof. Dr. Marcelo Henriques de Carvalho – UFMS – FACOM**





Introdução

- Repetição de instruções pode ser obtida por meio iterações;
- Outra forma de se implementar repetições é por meio de Recursão;
- Recursão ocorre quando uma função faz chamada de si própria;
- Entretanto, a fim de gerar uma resposta, uma condição de término deve ocorrer;
- Algoritmos recursivos são representados por Recorrências;
- Uma Recorrência é uma expressão que fornece o valor de uma função em termos dos valores “anteriores” da mesma função.





Exemplo – Fatorial

▣ O fatorial de um inteiro positivo n , denotado por $n!$, é definido por:

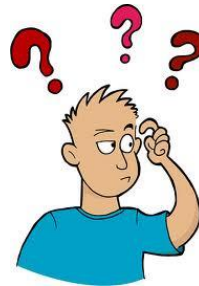
$$n! = \begin{cases} 1 & \text{se } n=0 \\ n.(n-1).(n-2)\dots 3.2.1 & \text{se } n \geq 1 \end{cases}$$



Função Fatorial

■ Exemplo: $5! = 5.4.3.2.1 = 120$

Será que a função fatorial pode ser definida de forma recursiva ?



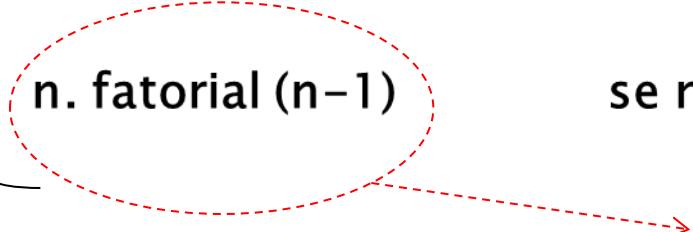


Fatorial – Definição Recursiva

Observe que $\text{fatorial}(5) = 5.(4.3.2.1) = 5.\text{fatorial}(4)$

```
fatorial(5)
5 * fatorial(5 - 1)
  4 * fatorial(4 - 1)
    3 * fatorial(3 - 1)
      2 * fatorial(2 - 1)
        1 * fatorial(1 - 1)
          1
```

$$\text{fatorial}(n) = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot \text{fatorial}(n-1) & \text{se } n \geq 1 \end{cases}$$

 **Recorrência**



Funções Recursivas

- Possuem um ou mais **casos básicos**, os quais são definidos de forma não-recursiva em termos de quantidades fixas. No caso da função fatorial, o caso básico é $n = 0$.
- Também possuem ou ou mais **casos recursivos**, os quais são definidos por meio da aplicação da definição da função.



Função Fatorial – Pseudocódigo

```
fatorial (n)

    if (n = 0)
        fatorial = 1

    else
        fatorial = n * fatorial(n-1)
```




Função Fatorial

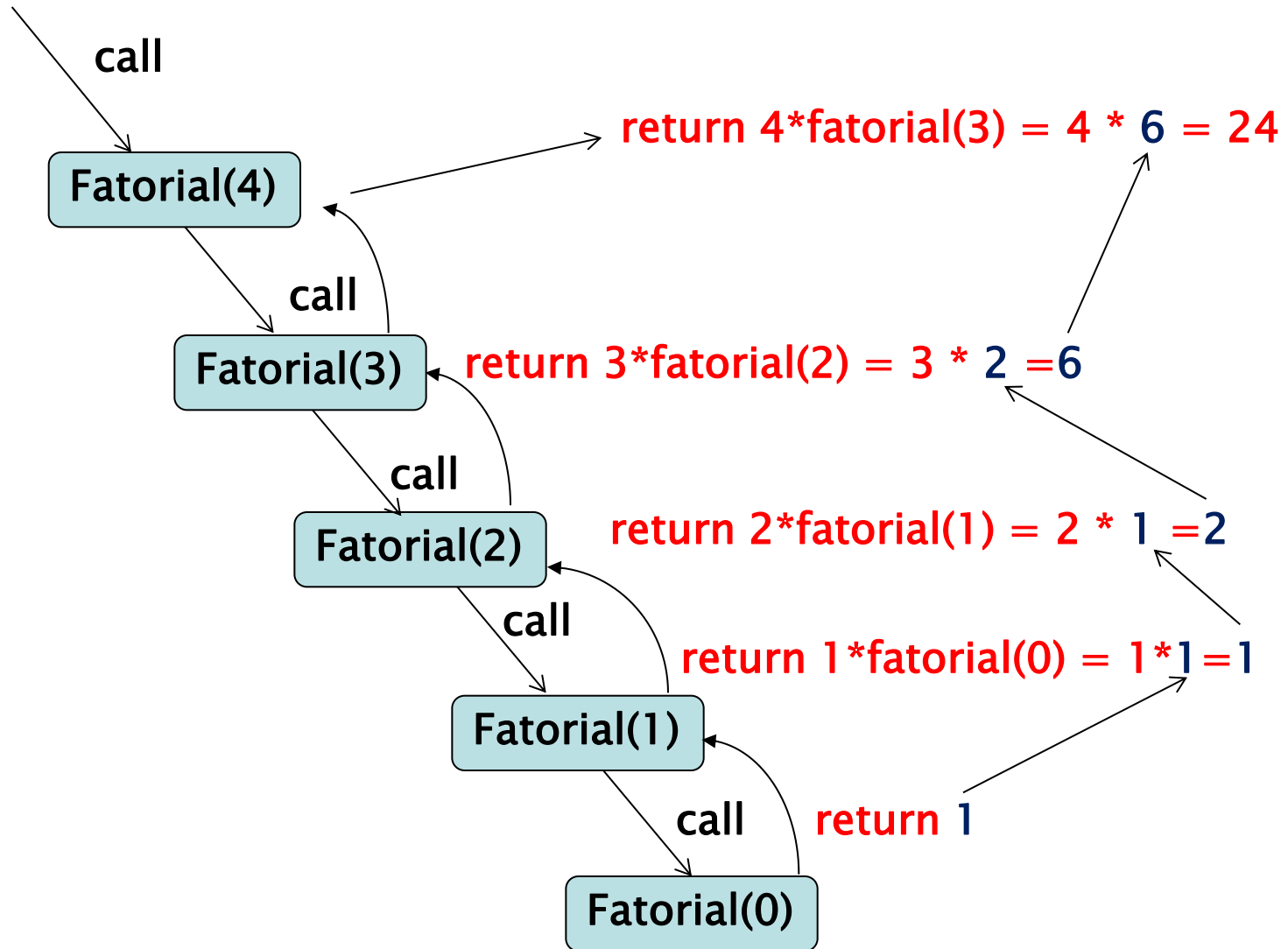
```
package maua;

public class fat {

    public static void main(String[] args) {
        int n=4;
        System.out.println("Fatorial(" + n + ") = " +
            fatorial(n) );
    }

    public static int fatorial(int n) {
        if (n==0)
            return 1; //caso básico
        else
            return(n*fatorial(n-1)); //caso recursivo
    }
}
```

Trace de Recursão





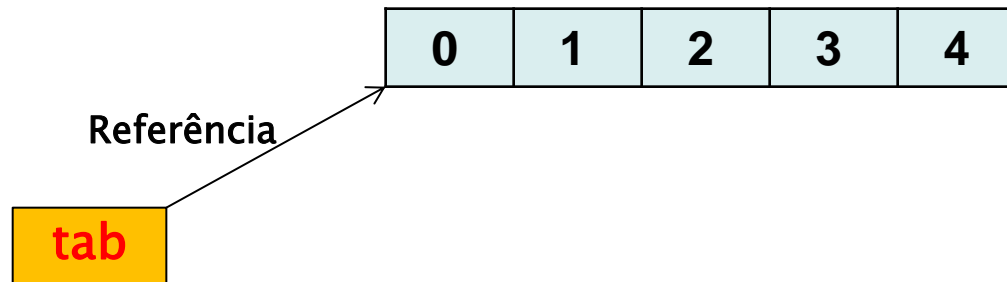
Recursão Linear

- Corresponde à forma mais simples de recursão.
- Neste tipo de recursão uma única chamada recursiva é feita de cada vez.
- Este tipo de recursão é útil quando num algoritmo se deseja obter o **primeiro** ou **último** elemento de uma lista no qual os elementos restantes têm a mesma estrutura da estrutura original.

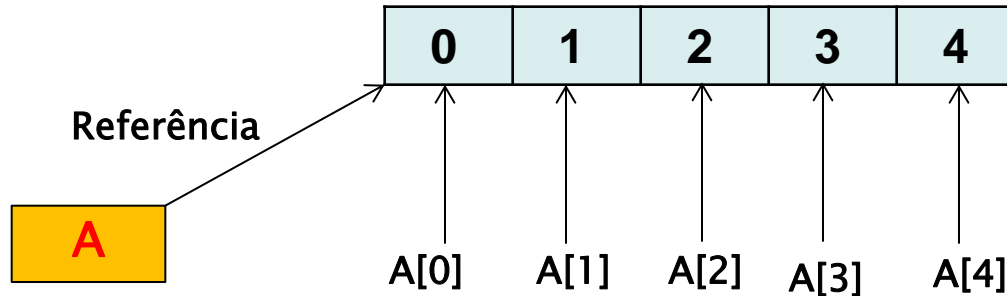


Soma dos elementos de um array

- Seja um array **A**, de n inteiros, no qual deseja-se obter a soma.
- Pode-se resolver este problema com o uso de recursão linear, observando-se que a soma de todos os n inteiros no array **A** é igual a $A[0]$ se $n = 1$, ou a soma dos primeiros $(n-1)$ inteiros em **A** mais o último elemento em **A**.



Soma dos elementos de um array



▣ Se $n=1$, $S(n) = a[0]$

▣ Se $n > 1$, $S(n) = S(n-1) + A[n-1]$

$$\begin{aligned}
 S_5 &= S_4 + A[4] \\
 &= S_3 + A[3] + A[4] \\
 &= S_2 + A[2] + A[3] + A[4] \\
 &= S_1 + A[1] + A[2] + A[3] + A[4] \\
 &= A[0] + A[1] + A[2] + A[3] + A[4]
 \end{aligned}$$



Soma dos elementos de um array

Pseudocódigo

soma_Rec(A,n)

```
if (n = 1)
    return A[0];
else
    return soma_Rec(A,n-1) + A[n-1];
```



Soma recursiva em array

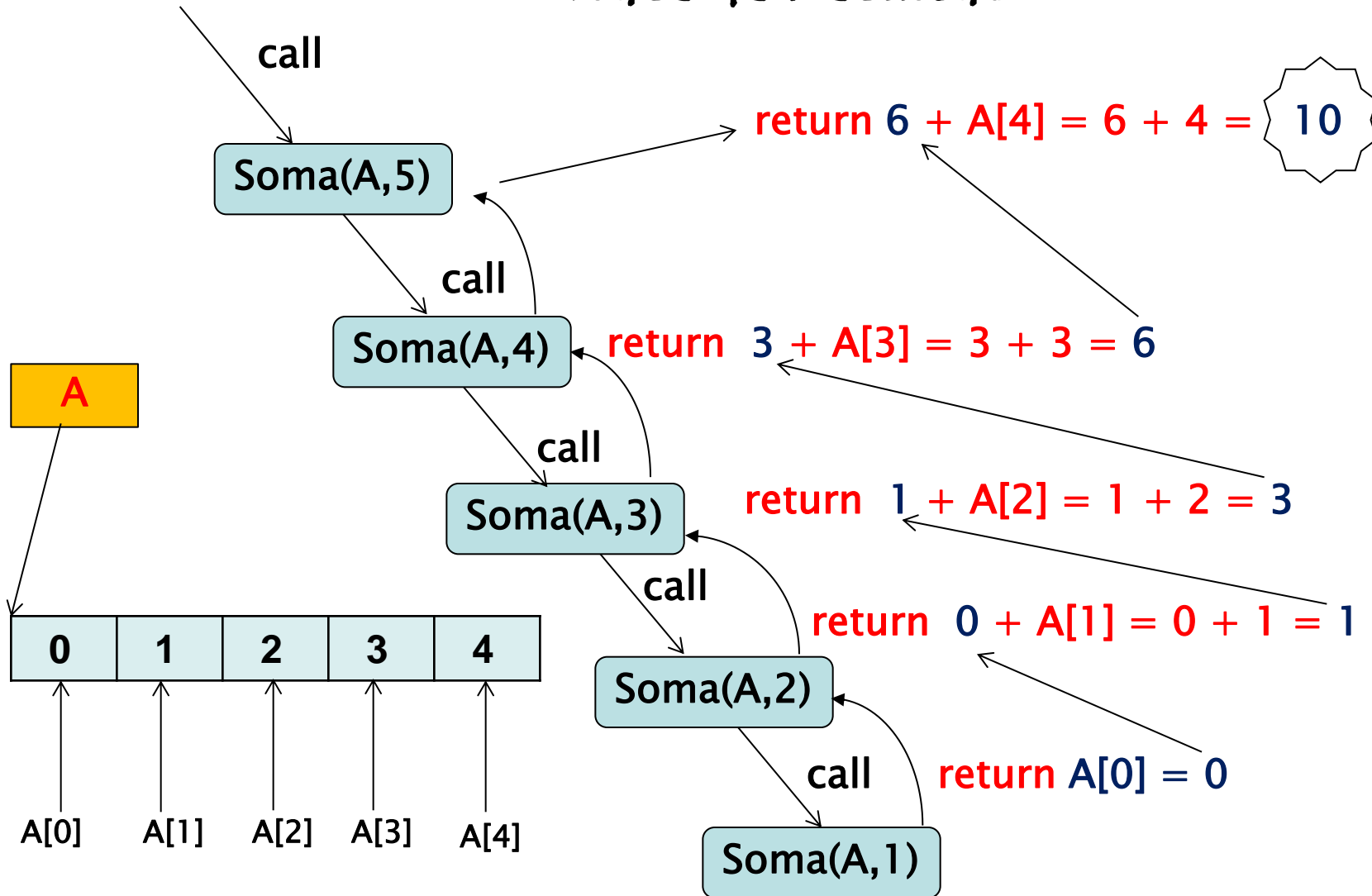
```
package maua;

public class Soma_Array {

    public static void main(String[] args) {
        int[] tab = new int[5];
        for (int i=0;i<tab.length;i++)
            tab[i] = i;
        System.out.println("A soma dos elementos do
            array é: " + soma_Rec(tab,tab.length) );
    }

    public static int soma_Rec(int[] A, int n) {
        if (n == 1)
            return A[0];
        else
            return soma_Rec(A,n-1) + A[n-1];
    }
}
```

Trça de Recursão





Propriedade importante da Recursão

- ❏ O método recursivo deve sempre assegurar que o procedimento [pára](#).
- ❏ Isso é assegurado por meio da escrita do caso para **n = 1**;
- ❏ A chamada recursiva sempre é feita com um valor de parâmetro inferior a **n**, no caso **n-1**.

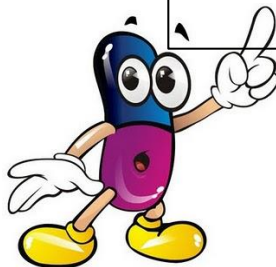
```
public static int soma_Rec(int[] A, int n) {  
    if (n == 1)  
        return A[0];  
    else  
        return soma_Rec(A, n-1) + A[n-1];  
}
```



Série de Fibonacci

- A sucessão de **Fibonacci** ou sequência de **Fibonacci** é uma sequência de números naturais, na qual os primeiros dois termos são **0** e **1**, e cada termo subsequente corresponde à soma dos dois precedentes.
- Os números de **Fibonacci** são, portanto, compostos pela seguinte sequência de números inteiros:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...





Série de Fibonacci

- ▣ Em termos matemáticos, a sequência é definida recursivamente pela fórmula abaixo, sendo os dois primeiros termos $F_0 = 0$ e $F_1 = 1$.

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



Série de Fibonacci – Pseudocódigo

Fibonacci(n)

if (n <= 1)

return n ;

else

return *Fibonacci*(n-1) + *Fibonacci*(n-2);



Série de Fibonacci

```
package maua;

public class Fibo {

    public static void main(String[] args) {
        int n=10;
        System.out.println (Fibonacci(n));
    }

    public static int  Fibonacci(int n) {
        if (n <= 1) return n ;
        else
            return (Fibonacci(n-1) + Fibonacci(n-2));
    }
}
```



Atividade

- Defina um algoritmo recursivo para encontrar o **máximo** elemento de um array **A** de **n** elementos.





Máximo elemento de um array

```
package maua;

public class Max_Recursivo {

    public static void main(String[] args) {

        int[] tab = { 4,6,8,1,4,9,10,4 } ;

        int n = tab.length;

        imprime(tab);

        System.out.println ("Maximo: " + max_Recursive(tab,n) );

    }
```



Máximo elemento de um array

```
public static void imprime(int[] v) {  
    System.out.print("Array: ");  
    for (int i=0; i<v.length; i++) {  
        System.out.print ( v[i] + "  " );  
    }  
    System.out.println("");  
}
```


Máximo elemento de um array

```
public static int max_Recursive(int[] A, int n) {  
    if (n == 1) return A[0];  
    else {  
        int x = max_Recursive(A, n-1);  
        if (x < A[n-1])  
            return A[n-1];  
        else  
            return x;  
    }  
}
```



Atividade

- Considere a função abaixo:



```
Func (int a)
    if (a < 2 )
        return 1
    else
        return (a-1) * Func(a-1)
```

O que faz esta função ?



Atividade

```
package maua;

public class Exer_Recur1 {

    public static void main(String[] args) {

        int n = 5;
        System.out.println("\n" + Func(n) );

    }

    public static int Func (int a) {
        if (a < 2 )
            return 1;
        else
            return (a-1) * Func(a-1);
    }
}
```





Atividade



- Escreva uma implementação na Linguagem **C** ou em **Java** que recebe do usuário um **String** e chama uma função recursiva que retorna um valor booleano representando **true** se o **String** corresponder a uma **Palíndromo** ou **false** caso contrário.

- Exemplo de Palíndromo:

A B C C B A

X Y Z A Z Y X

TATTARRATTAT

RACECAR

ROTATOR

PULLUP

REDDER ...





Palíndrome Recursiva

```
package maua;
import java.util.*;

public class Palindrome {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Digite um String: ");
        String x = sc.nextLine();

        if ( isPalindrome(x) )
            System.out.println("O string " + x + " é uma PALÍNDROME...");
        else
            System.out.println("O string " + x + " não é uma PALÍNDROME...");
    }
}
```

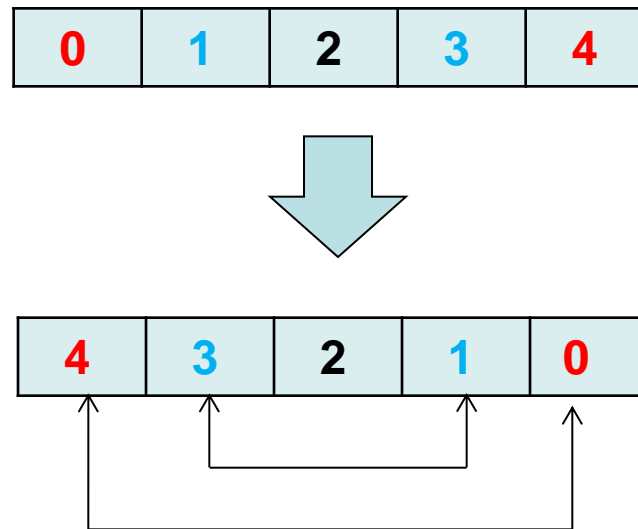
Palíndrome Recursiva

```
public static boolean isPalindrome(String s) {  
    if (s.length() <= 1) return true;  
    else {  
        if (s.charAt(0) != s.charAt((s.length() - 1)) )  
            return false;  
        else  
            return isPalindrome( s.substring(1,s.length()-1) ) ;  
    }  
}
```



Atividade

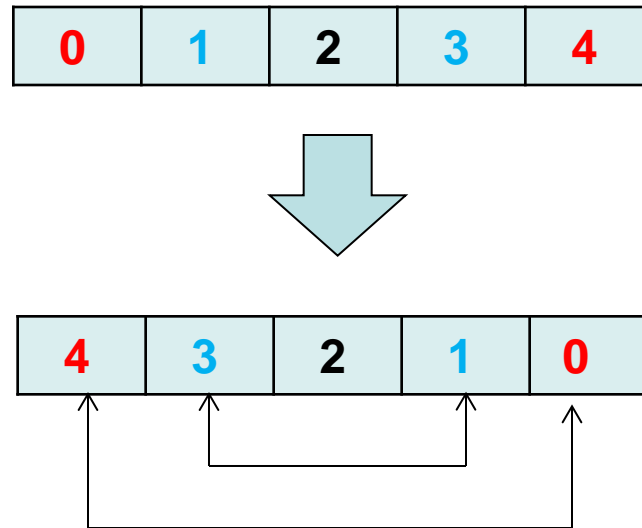
- Escreva uma implementação na Linguagem **C** ou em **Java** que consiste em reverter os n elementos de um array, de modo que o primeiro elemento torna-se o último, o segundo elemento o penúltimo, e assim por diante.





Revertendo os elementos de um array com Recursão Linear

- O problema consiste em reverter os **n** elementos de um array, de modo que o primeiro elemento torna-se o **último**, o segundo elemento o **penúltimo**, e assim por diante.





Revertendo os elementos de um array com Recursão Linear

Pseudocódigo

```
reverte_array(A,i,j) {  
    if (i<j) {  
        swap(v,i,j);  
        reverte_array(v, i+1, j-1);  
    }  
}
```



Revertendo os elementos de um array com Recursão Linear

```
package maua;

public class Recur02 {

    public static void main(String[] args) {

        int[] tab = new int[6];
        for (int i=0; i<tab.length; i++) {
            tab[i] = i;
        }

        imprime(tab);

        reverte_array(tab, 0, tab.length-1);

        imprime(tab);
    }
}
```



Revertendo os elementos de um array com Recursão Linear

```
public static void reverte_array(int[] v, int i, int j) {  
    if (i < j) {  
        swap(v, i, j);  
        reverte_array(v, i+1, j-1);  
    }  
}  
  
public static void swap(int[] v, int i, int j) {  
    int trab = v[i];  
    v[i] = v[j];  
    v[j] = trab;  
}
```



Revertendo os elementos de um array com Recursão Linear

```
public static void imprime(int[] v) {  
  
    System.out.print("Array: ");  
  
    for (int i=0; i<v.length; i++) {  
        System.out.print ( v[i] + " ");  
    }  
    System.out.println("");  
}  
  
}
```