



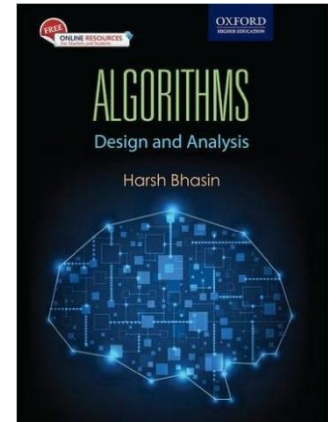
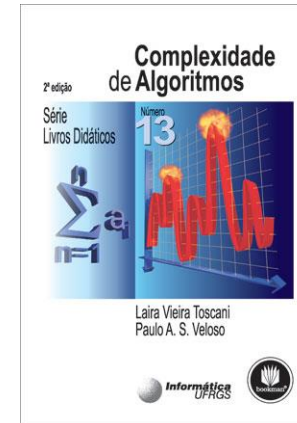
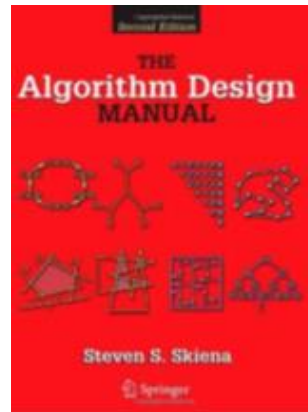
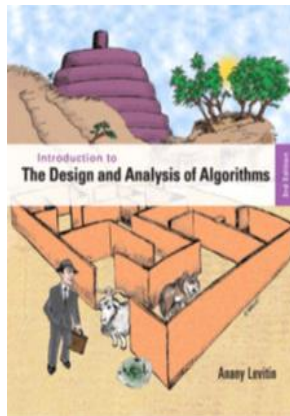
# Unidade 10 – Análise de Algoritmos com Estruturas de Dados Lineares

## Parte 1



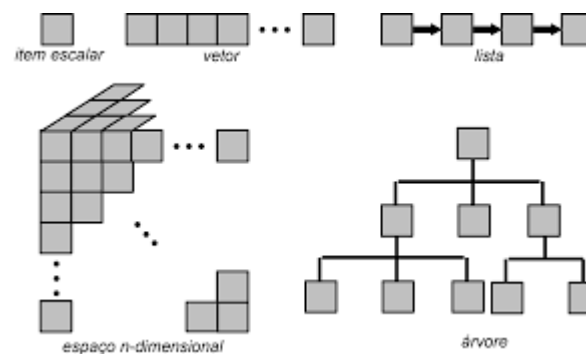
# Bibliografia

- 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



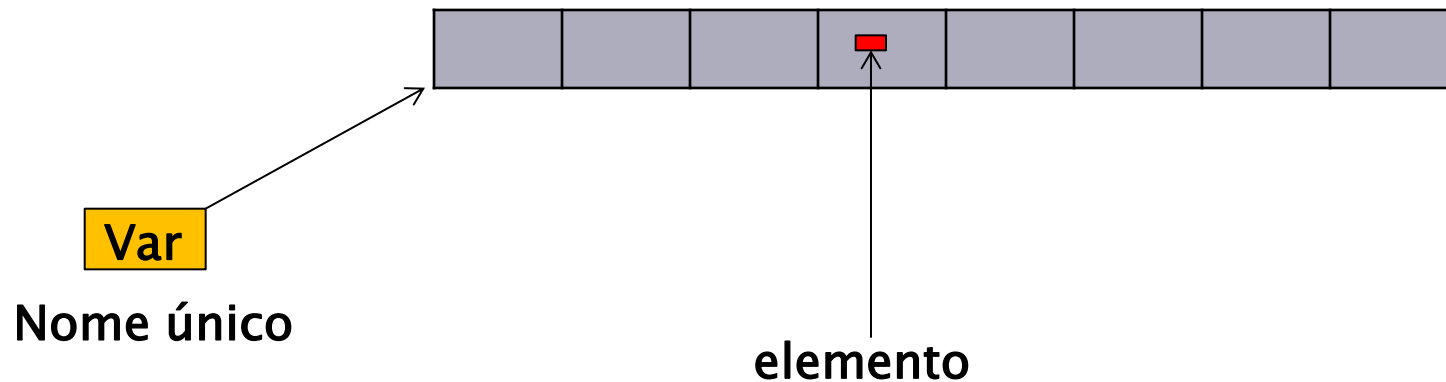
# Introdução

- É essencial ser capaz de desenvolver algoritmos, analisá-los e implementá-los da forma mais eficiente possível;
- Esta eficiência pode ser obtida com o emprego de estruturas de dados convenientes para o problema a ser resolvido pelo algoritmo;
- Cada estrutura de dados é composta por um conjunto de operações tais como: create, delete, update e travessia (tipo abstrato de dados).



# Arrays

- ◆ Um **array** é um conjunto de variáveis do mesmo tipo a qual atribuímos um nome único.
- ◆ Cada informação (dado) no **array** é chamada de **elemento** do **array**.





# Arrays

- ◆ Para fazermos referência à um elemento de um array devemos usar o nome do array em conjunto com um número inteiro chamado **índice**.
- ◆ O primeiro elemento do array tem índice **0**, o segundo **1**, e assim por diante.

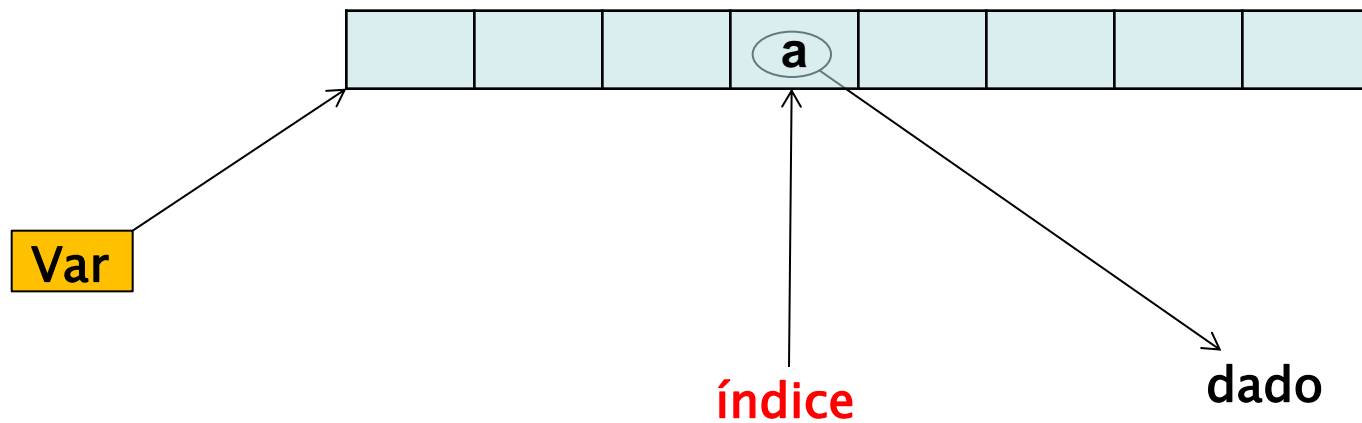


Índice é como o rótulo de uma caixa ...



# Índice de um array

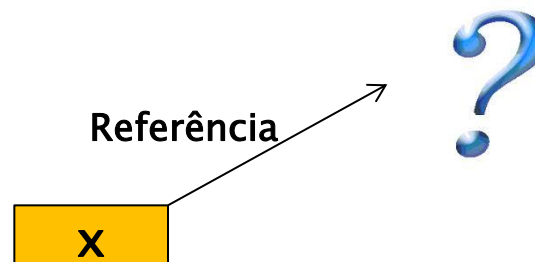
- ◆ Pode ser representado pela avaliação de uma expressão que deve resultar em um valor inteiro maior ou igual a zero.



## Variáveis array

```
int [ ] x;
```

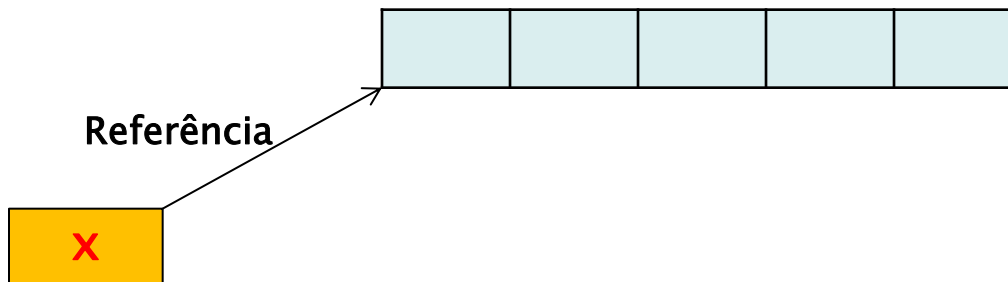
- ❖ A variável **x** corresponde a uma **referência** a um array de inteiros que ainda não foi criado.
- ❖ Portanto, neste ponto ainda não foi alocada memória para o array.



# Definindo um array

```
x = new int[5];
```

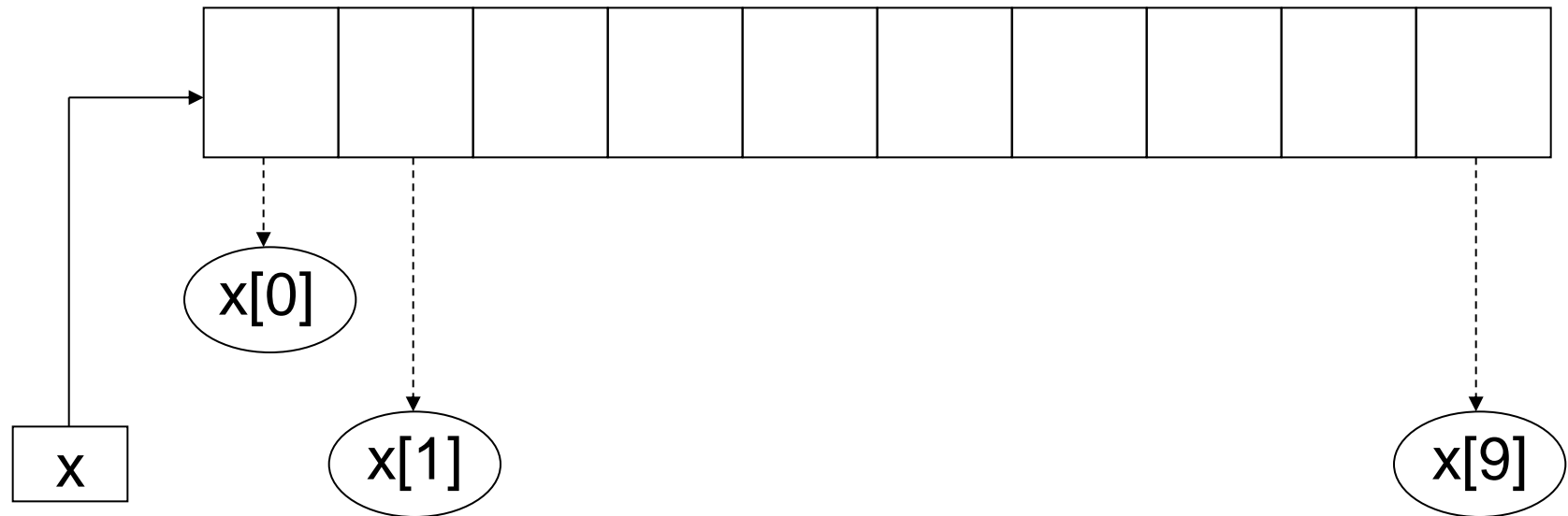
- ❖ O statement acima cria um array que irá armazenar 5 valores inteiros e grava uma referência ao array na variável x.
- ❖ A referência é simplesmente o endereço onde o array está na memória.





# Acesso aos elementos do array

```
int[ ] x = new int[10];
```



# Iniciando arrays

- ◆ Podemos inicializar um array explicitando os valores em tempo de declaração.
- ◆ Com este procedimento o tamanho do array e, consequente alocação de memória, é definido.

```
int [ ] x = {2,3,5,7,11,13,17} ;
```

ou `int[ ] x = new int [ ] { 2,3,5,7,11,13,17 } ;`

O array acima tem 7 elementos inteiros.



# Atribuição de arrays

```
int [ ] x = new int[100];
```

```
x[0] = 2;
```

```
x[1] = 3;
```

❖ Obs. Os demais itens do array são inicializados em **zero** (valor default)



## Imprimindo os elementos do array

```
package maua;
public class ArrayPrint {

    public static void main(String[] args) {
        Integer[] x = new Integer[50];

        for (int i=0 ; i < x.length ; i++)
            x[i]=i;

        imprimeArray(x);
    }

    public static void imprimeArray (Integer[] array ) {

        Integer n = array.length;

        for (int i = 0 ; i < n ; i++)
            System.out.println(array[i]);

    }

}
```



```
public static void imprimeArray (Integer[] array ) {  
  
    Integer n = array.length;  
  
    for (int i = 0 ; i < n ; i++)  
        System.out.println(array[i]);  
  
}
```

Qual o ordem de Complexidade da  
Função `imprimeArray`?



## Qual o ordem de Complexidade da Função `imprimeArray`

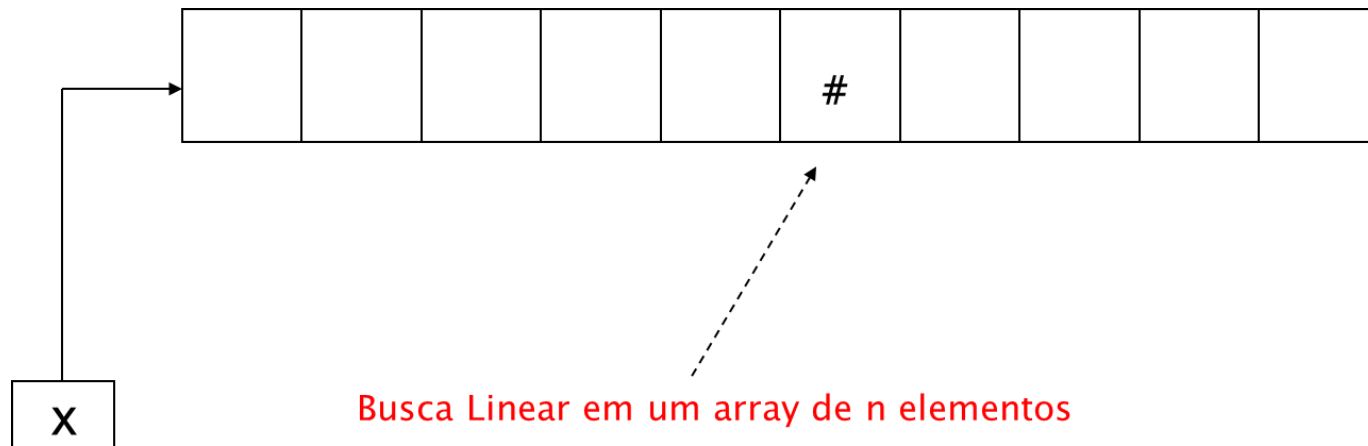
```
public static void imprimeArray (Integer[] array ) {  
  
    Integer n = array.length;  
  
    for (int i = 0 ; i < n ; i++)  
        System.out.println(array[i]);  
  
}
```

Ordem de Complexidade:  $O(n)$



## Pesquisa Linear

- ◆ Considerando um array com  $n$  elementos, a pesquisa linear corresponde a atravessar linearmente o array para encontrar um determinado elemento;



Busca Linear em um array de  $n$  elementos



## Busca Linear

```
package maua;

public class BuscaLinear {

    public static void main(String[] args) {
        Integer[] x = new Integer[50];

        for (int i=0 ; i < x.length ; i++)
            x[i]=i+1;

        Integer argumento = 50;

        Integer n = (BuscaLinearArray(x, argumento) ) ;

        if ( n == -1)
            System.out.println("Valor não existente no
                               array...");
        else
            System.out.println("Valor " + argumento + "
                               encontrado na posição: " + n );
    }
}
```





## Busca Linear

```
public static Integer BuscaLinearArray (Integer[] array , Integer argumento ) {  
  
    Integer n = array.length;  
  
    for (int i = 0 ; i < n ; i++)  
        if (array[i] == argumento)  
            return i;  
    return -1;  
  
}
```



## Busca Linear

```
public static Integer BuscaLinearArray (Integer[] array , Integer argumento ) {  
  
    Integer n = array.length;  
  
    for (int i = 0 ; i < n ; i++)  
        if (array[i] == argumento)  
            return i;  
    return -1;  
  
}  
}
```

Qual o ordem de Complexidade da  
Função BuscaLinearArray?



# Ordem de Complexidade da Função BuscaLinearArray

```
public static Integer BuscaLinearArray (Integer[] array , Integer argumento ) {  
  
    Integer n = array.length;  
  
    for (int i = 0 ; i < n ; i++)  
        if (array[i] == argumento)  
            return i;  
    return -1;  
  
}  
  
}
```

- ◆ Se o elemento estiver presente logo na primeira posição do array, então a ordem de complexidade seria  **$O(1)$** ;
- ◆ No caso extremo, quando o elemento não estiver presente no array, todos os  $n$  elementos do array deverão ser visitados, o que corresponde a complexidade  **$O(n)$** .



## Revertendo a ordem dos elementos de um array

```
package maua;

import java.util.Arrays;

public class RevertArray {

    public static void main(String[] args) {
        Integer[] x = new Integer[9];

        for (int i=0 ; i < x.length ; i++)
            x[i]=i+1;

        System.out.println(Arrays.toString(x));

        System.out.println("\n");

        System.out.println(Arrays.toString(Reorder(x)));

    }
```



## Revertendo a ordem dos elementos de um array

```
public static Integer[] Reorder (Integer[] array ) {  
  
    Integer n = array.length;  
  
    Integer j = (n/2), temp;  
  
    for (int i = 0; i < j ; i++) {  
        temp = array[i];  
        array[i] = array[n-1-i];  
        array[n-1-i] = temp;  
    }  
    return array;  
}
```



# Revertendo a ordem dos elementos de um array

```
public static Integer[] Reorder (Integer[] array ) {

    Integer n = array.length;

    Integer j = (n/2), temp;

    for (int i = 0; i < j ; i++) {
        temp = array[i];
        array[i] = array[n-1-i];
        array[n-1-i] = temp;
    }
    return array;
}
```

Qual o ordem de Complexidade da  
Função Reorder?



# Ordem de Complexidade da Função Reorder

```
public static Integer[] Reorder (Integer[] array ) {  
  
    Integer n = array.length;  
  
    Integer j = (n/2), temp;  
  
    for (int i = 0; i < j ; i++) {  
        temp = array[i];  
        array[i] = array[n-1-i];  
        array[n-1-i] = temp;  
    }  
    return array;  
}
```

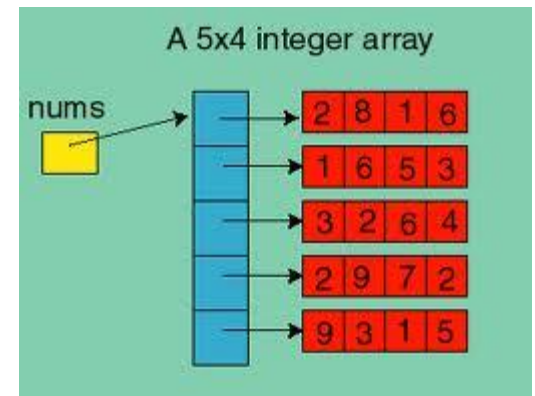
- ◆ Todas as operações internas loop são executadas  $n/2$  vezes;
- ◆ Assim, a ordem de complexidade da função Reorder é  **$O(n)$** .



# Array de arrays

- ◆ Usam mais de um índice para acessar os elementos do array.
- ◆ São usados para tabelas e outros arranjos mais complexos.
- ◆ São também chamados de arrays bidimensionais, uma vez que têm duas dimensões.
- ◆ No exemplo, o primeiro índice se refere à quantidade de linhas e o segundo índice corresponde à quantidade de colunas da tabela.

```
int [ ] [ ]  nums = new int [5][4];
```





# Carga de Array de arrays

```
package maua;

public class ArrayArray {

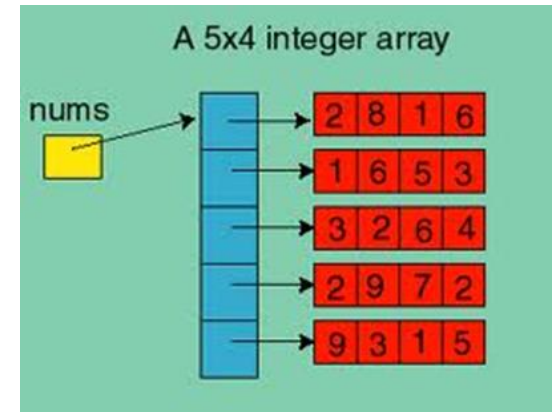
    public static void main(String[] args) {

        int[][] nums = new int[5][4];

        for (int r=0; r < nums.length; r++) {

            for (int c=0; c < nums[r].length; c++) {
                nums [r][c] = (int) (Math.random() * 10);
                System.out.print(" " + nums[r][c]);
            }
            System.out.println("");
        }

    }
}
```



# Imprimindo array de arrays

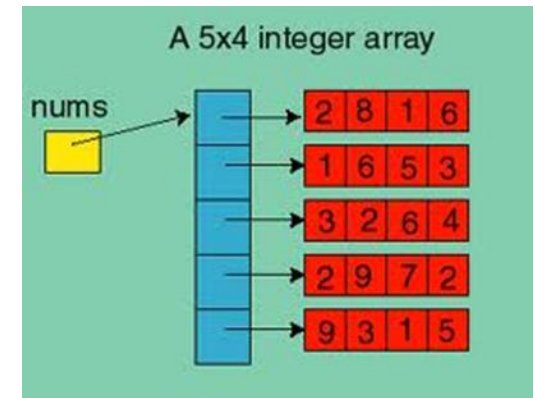
```
package maua;

public class PrintArray {

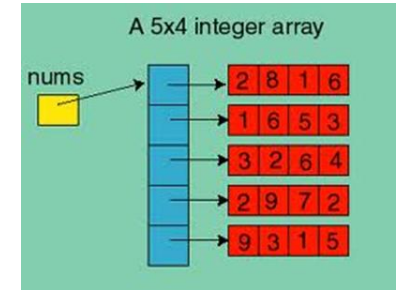
    public static void main(String[] args) {
        int[][] nums = new int[5][4];

        for (int r=0; r < nums.length; r++)
            for (int c=0; c < nums[r].length; c++)
                nums[r][c] = (int) (Math.random() * 10);

        printArray2D(nums);
    }
}
```



# Imprimindo array de arrays



```
public static void printArray2D (int[][][] array) {
    for (int r=0; r < array.length; r++) {
        for (int c=0; c < array[r].length; c++)
            System.out.print ( array[r][c] + " " );
        System.out.print("\n");
    }
}
```



## Imprimindo Array de Array

```
public static void printArray2D (int[][] array) {  
    for (int r=0; r < array.length; r++) {  
        for (int c=0; c < array[r].length; c++)  
            System.out.print ( array[r][c] + " " );  
        System.out.print("\n");  
    }  
}
```

Qual o ordem de Complexidade da  
Função **printArray2D**?



# Ordem de Complexidade da Função `printArray2D`

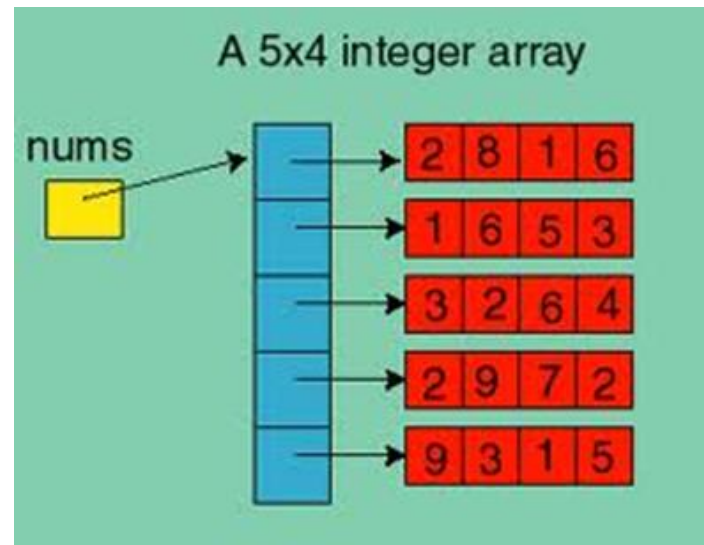
```
public static void printArray2D (int[][] array) {
    for (int r=0; r < array.length; r++) {
        for (int c=0; c < array[r].length; c++)
            System.out.print ( array[r][c] + " " );
        System.out.print("\n");
    }
}
```

- ◆ Admitindo-se uma matriz de  $n$  linhas e  $n$  colunas, as operações executadas internamente ao loop serão:  $n \times n$  (loop dentro de loop);
- ◆ Assim, a ordem de complexidade do algoritmo será  $O(n^2)$ .



# Atenção

- ◆ Na verdade, Java não tem arrays multidimensionais.
- ◆ Todos os arrays em Java são uni-dimensionais.
- ◆ A expressão `nums[i]` refere-se ao *i*th sub-array da tabela.

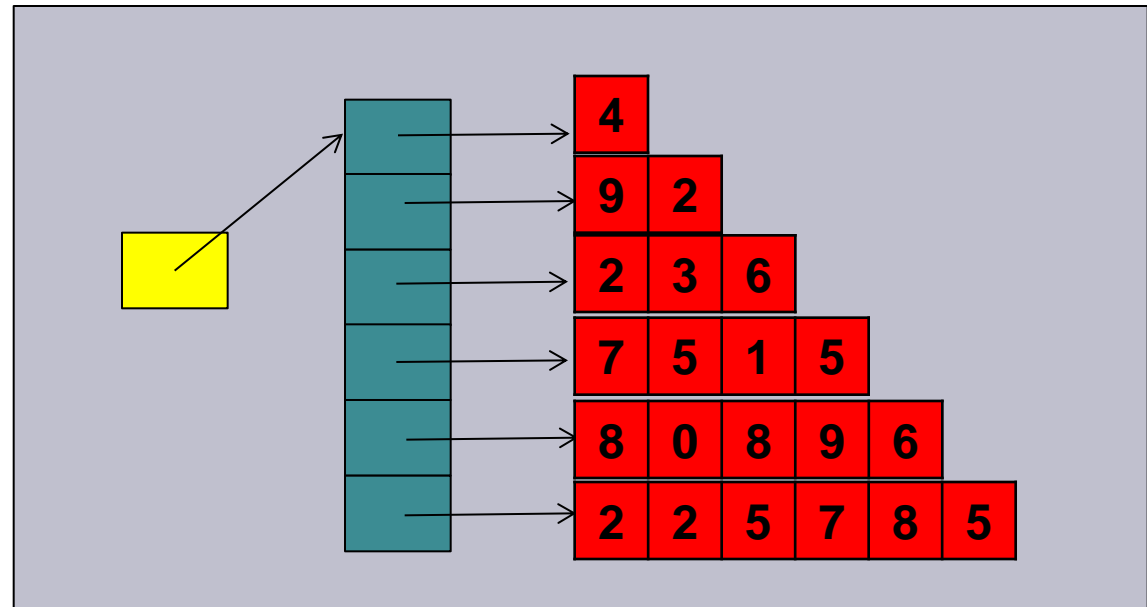




# Arrays irregulares

- Em Java, é possível criar-se arrays no qual diferentes linhas têm diferentes colunas.

4
9 2
2 3 6
7 5 1 5
8 0 8 9 6
2 2 5 7 8 5



# Arrays irregulares

```
package maua;

public class ArrayRagged {
    public static void main(String[] args) {

        int NMAX = 5;
        int [][] matriz_triangular = new int[NMAX + 1] [];

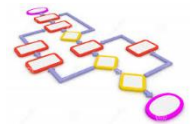
        for (int n=0; n <= NMAX ; n++ )
            matriz_triangular[n] = new int[n+1];

        for (int n=0; n<matriz_triangular.length; n++) {
            for (int k=0; k < matriz_triangular[n].length; k++) {
                matriz_triangular[n][k] =
                    (int) (10* Math.random());
                System.out.print(" " + matriz_triangular[n][k]);
            }
            System.out.println("");
        }
    }
}
```

◆ Ordem de complexidade:  $O(n^2)$ .







## Estruturas do Tipo Lista

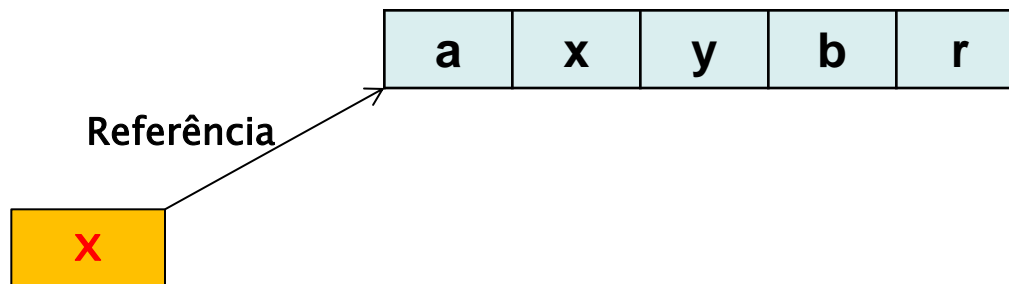
- Uma lista ou sequência é uma estrutura de dados abstrata que implementa uma coleção ordenada de valores, onde o mesmo valor pode ocorrer mais de uma vez.
- Uma lista é um tipo abstrato de dados ( especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados ).



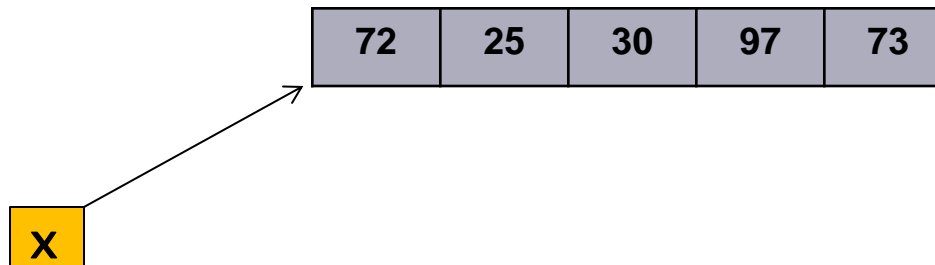
Listas podem ser implementadas por meio de arrays?



## Implementando listas com arrays...



Exemplo: lista de 5 valores aleatórios entre 0 e 100



# Lista de 5 valores aleatórios entre 0 e 100

```
package maua;

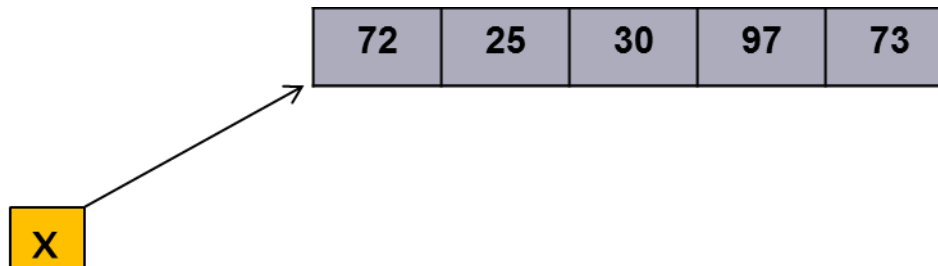
public class Lista_Aleatoria {

    public static void main(String[] args) {

        int[] x = new int[5];
        for (int i=0; i<x.length ; i++) {
            x[i] = (int)(100.0 * Math.random());
            System.out.print(" " + x[i]);
        }

    }

}
```

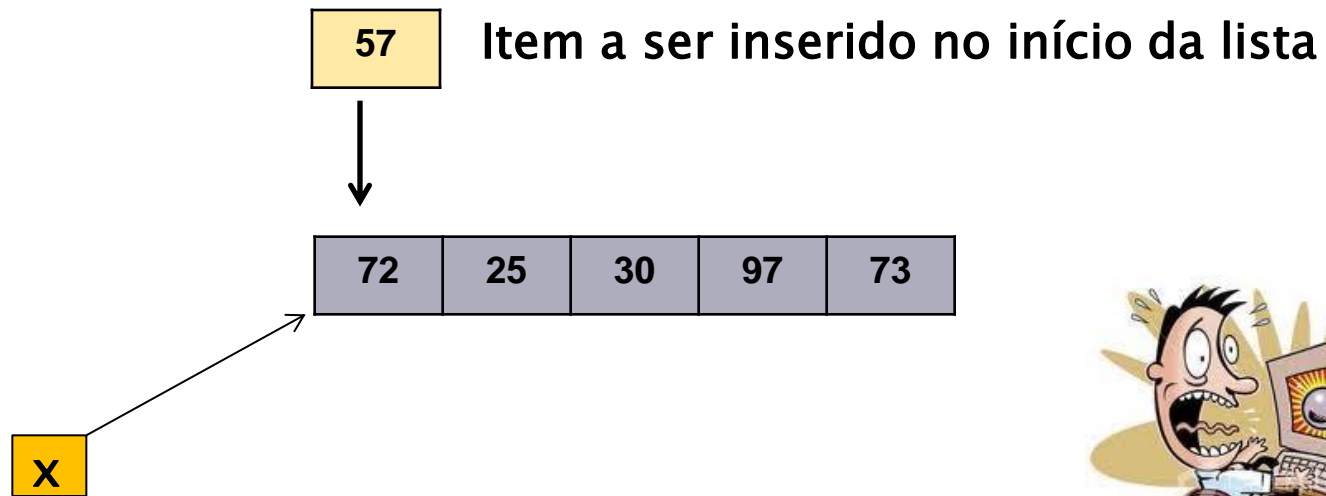




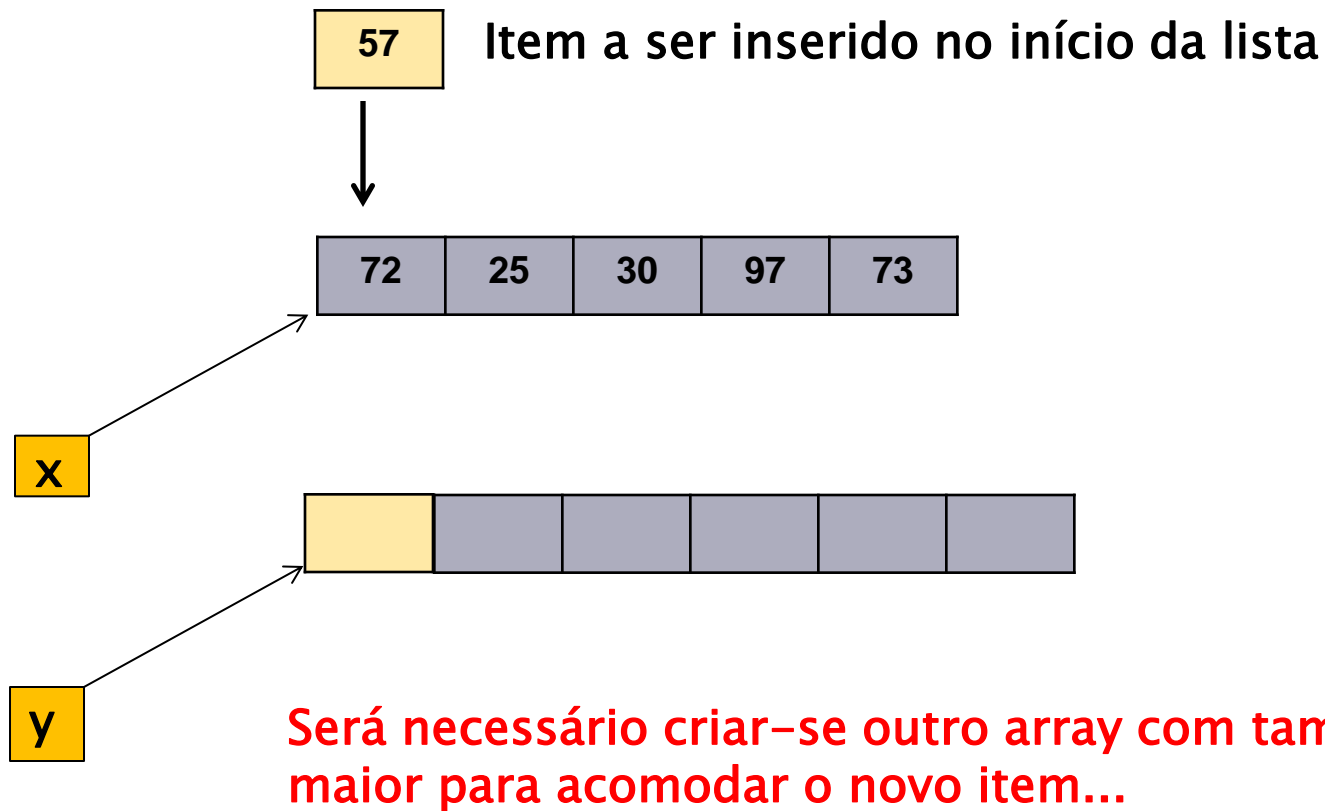
Mas, há alguns inconvenientes em se implementar listas com arrays...



Considere a necessidade de se inserir um item no início ou metade da lista ...

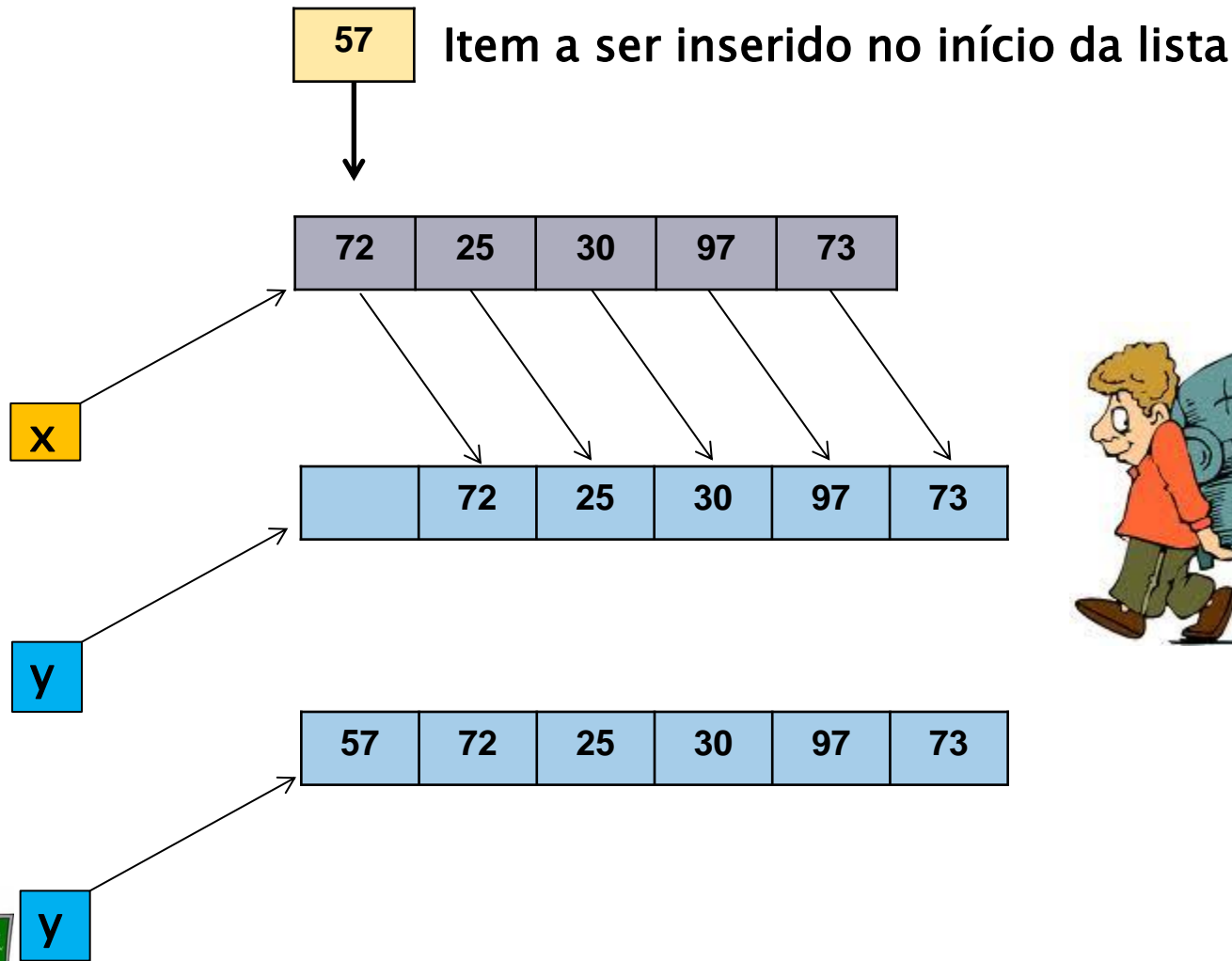


Primeiro, arrays têm tamanho fixo...





Segundo, o esforço computacional terá tempo proporcional ao tamanho do array...



# Tipo Abstrato de Dados: List\_Array

```
int[] lista;  
int ultimo_item;
```

← dados

```
List_Array()  
List_Array(n)  
imprime_Lista()  
imprime_Primeiro()  
imprime_Ultimo()  
Imprime_Item(localizacao)  
insereItem_Inicio(novoitem)  
insereItem_Fim(novoitem)  
insereItem(novo_item, localizacao)  
insereItem_MetadeLista(novoitem)  
altera_Ultimo(item)  
altera_Primeiro(item)  
alteraItem(item, localizacao)  
Deleta_Ultimo()  
Deleta_Primeiro()  
Deleta_Item(localizacao)
```

← operações



# Implementação

```
package maua;

public class List_Array {
    int[] lista;
    int ultimo_item;

    public List_Array() {
        lista = new int[0];
        ultimo_item = -1;
    }

    public List_Array(int n) {
        lista = new int[n];
        ultimo_item = -1;
    }
}
```



# Implementação

```
package maua;
```

```
public class List_Array {
```

```
    int[] lista;
```

```
    int ultimo_item;
```

```
    public List_Array() {  
        lista = new int[0];  
        ultimo_item = -1;  
    }
```

```
    public List_Array(int n) {  
        lista = new int[n];  
        ultimo_item = -1;  
    }
```

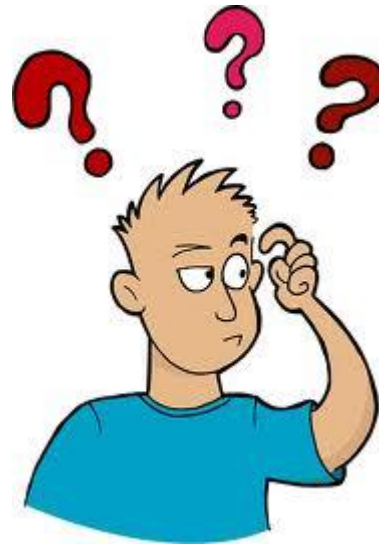


7





# Como imprimir os elementos da lista ?



# Imprimindo os elementos da lista

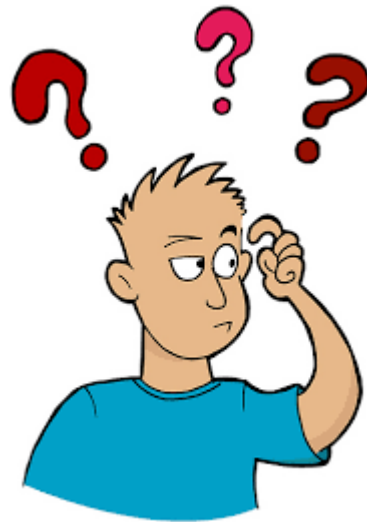
```
public void Imprime_Lista() {  
    Integer n = lista.length;  
    if (n == 0)  
        System.out.println("A lista está vazia...");  
    else System.out.print("\nLista:");  
  
    for (int i=0; i<n; i++){  
        System.out.print("  " + lista[i]);  
    }  
    System.out.println("");  
}
```

◆ Ordem de complexidade:  $O(n)$ .

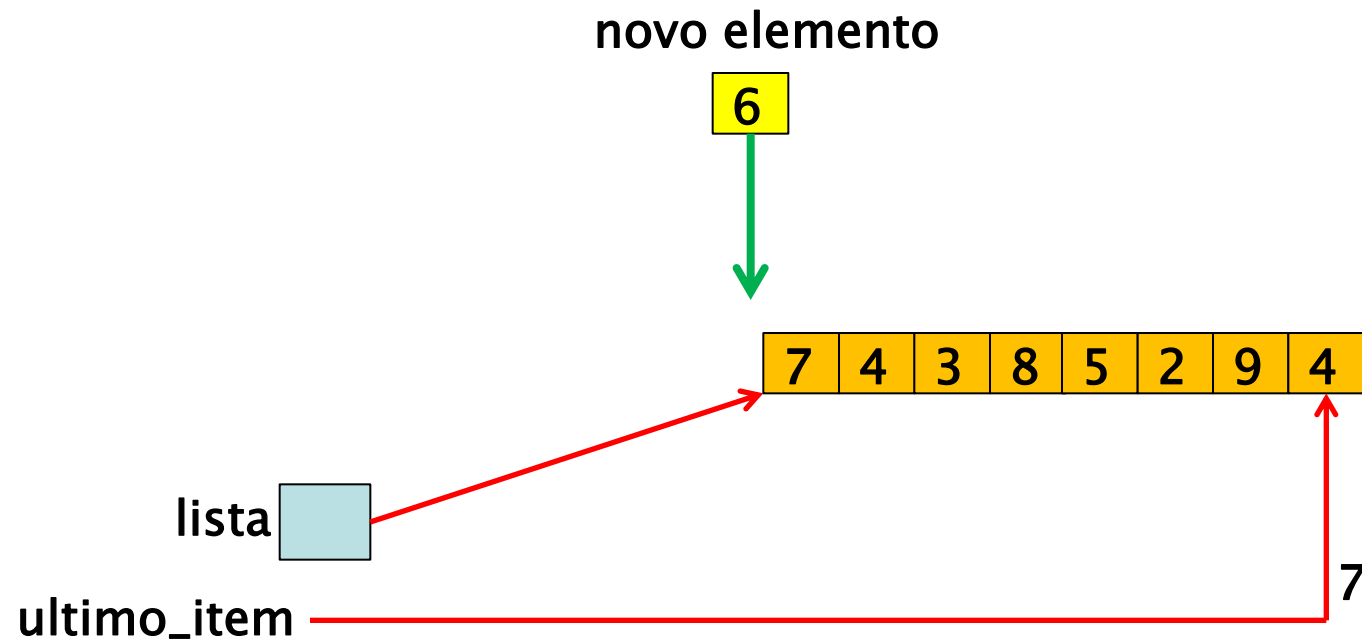




Como inserir um elemento no início da lista ?



# Inserção no início da lista





# Inserção no início da lista

novo elemento

6

7 4 3 8 5 2 9 4

lista

ultimo\_item

7

0 0 0 0 0 0 0 0 0

**Primeiro:** cria-se um novo array com um elemento a mais...



# Inserção no início da lista

novo elemento

6



7 4 3 8 5 2 9 4

lista

ultimo\_item

7

nova\_lista



0 7 4 3 8 5 2 9 4

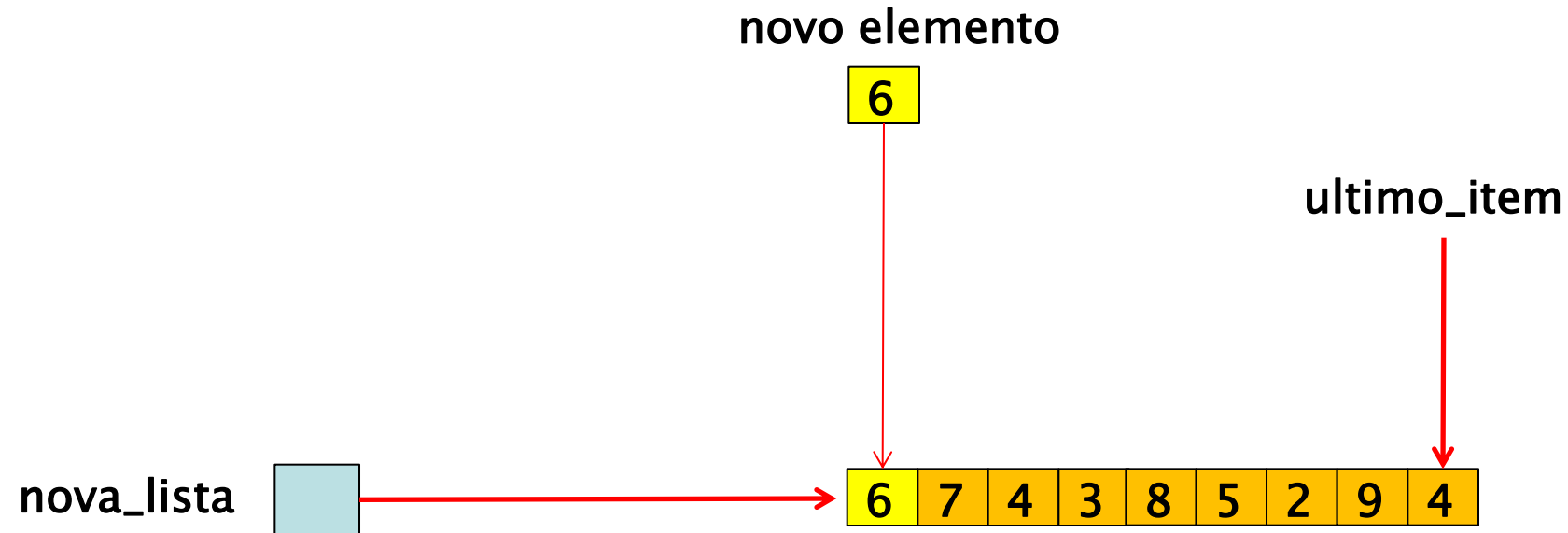
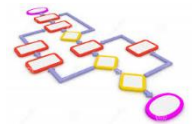
ultimo\_item



**Segundo:** copia-se os elementos da lista antiga para a lista nova



# Inserção no início da lista



**Terceiro:** novo elemento é inserido na primeira posição da nova lista...



# Implementação

```
public void insereItem_Inicio(int novo_item) {  
  
    int[] nova_lista = new int[lista.length+1];  
  
    for (int i=0; i<lista.length; i++)  
        nova_lista[i+1]=lista[i];  
  
    nova_lista[0] = novo_item;  
    lista = nova_lista;  
    ultimo_item = lista.length-1;  
  
}
```

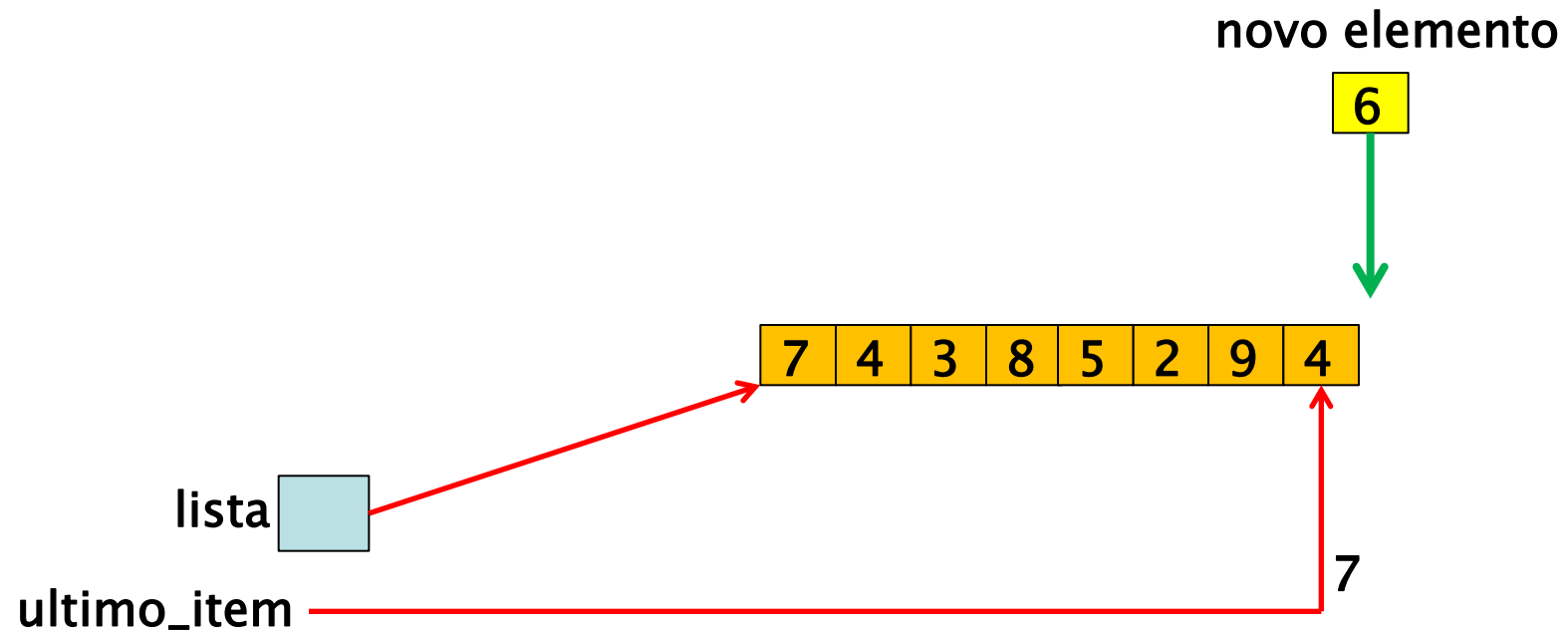
◆ Ordem de complexidade:  $O(n)$ .



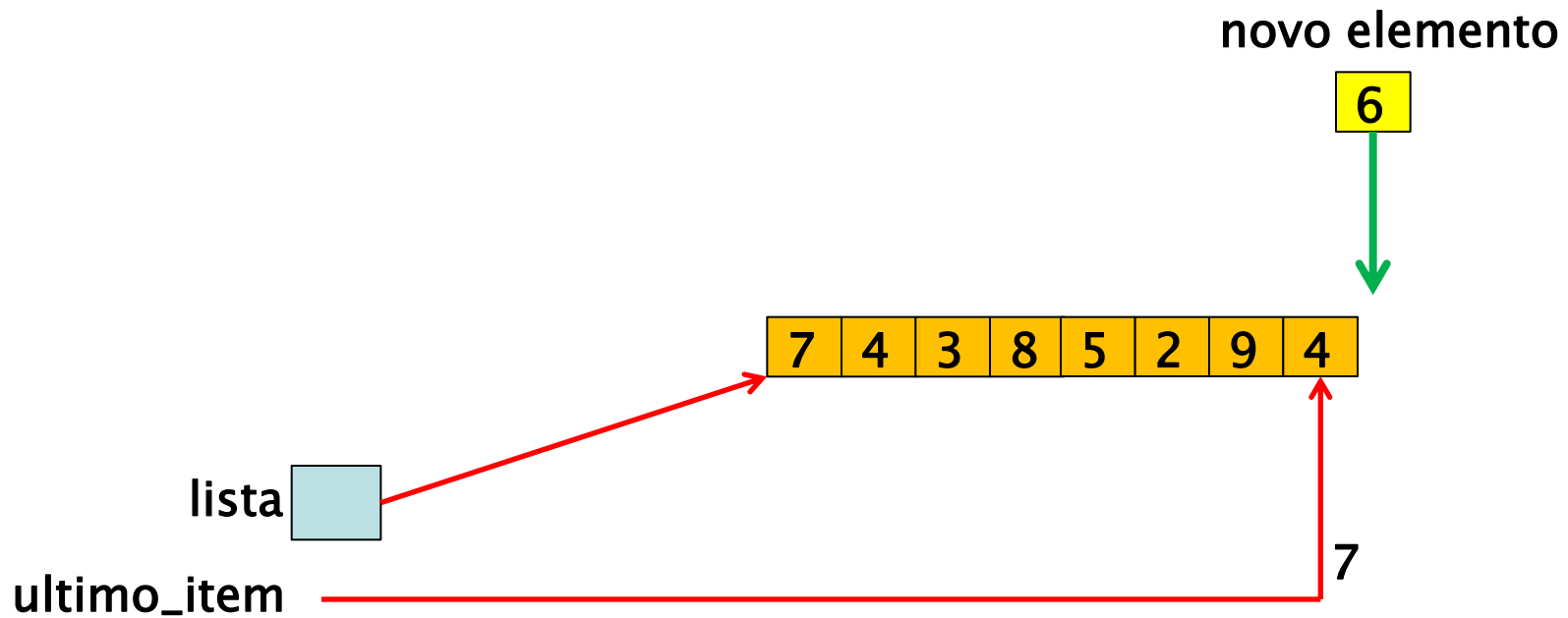
Como inserir um elemento no fim da lista ?



# Inserção no fim da lista



# Inserção no fim da lista



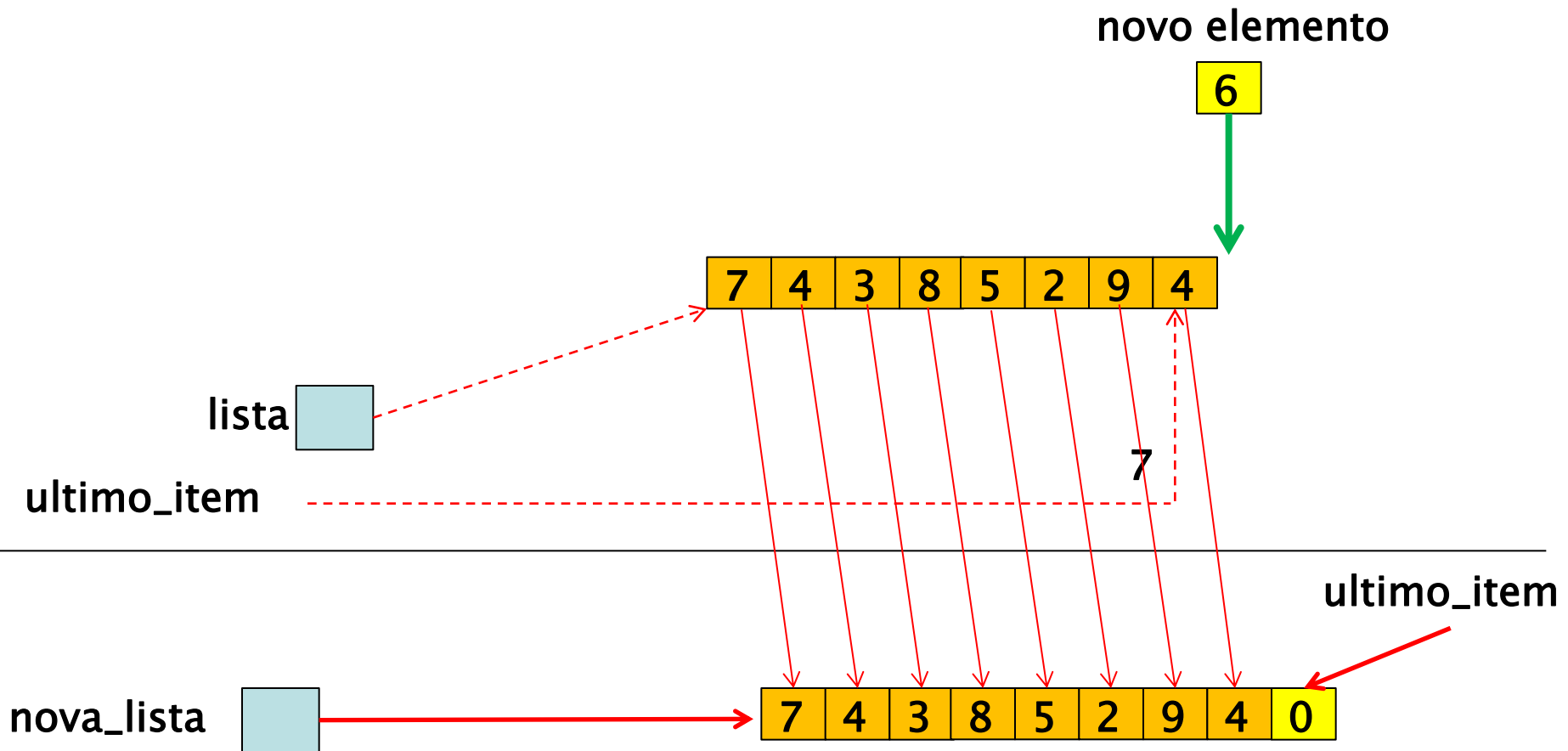
0 0 0 0 0 0 0 0 0



**Primeiro:** cria-se um novo array com um elemento a mais...



# Inserção no fim da lista

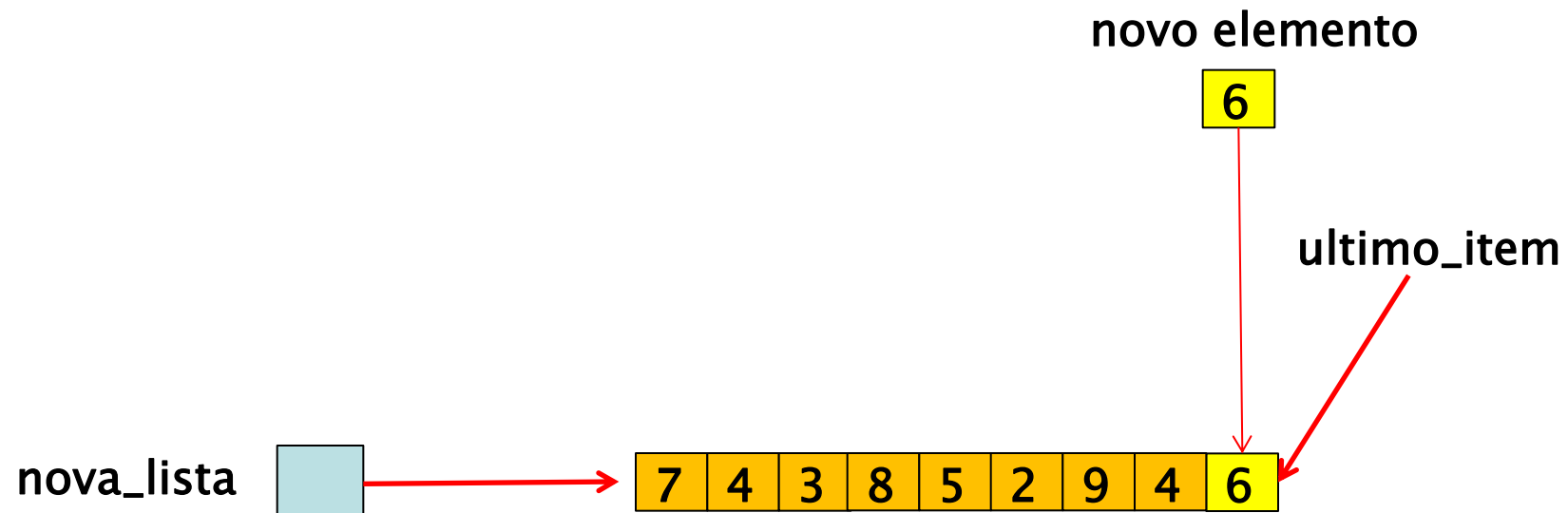


**Segundo:** copia-se os elementos da lista antiga para a lista nova





# Inserção no fim da lista



**Terceiro:** novo elemento é inserido na última posição da nova lista...



# Implementação

```
public void insereItem_Fim(int novo_item) {  
  
    int[] nova_lista = new int[lista.length+1];  
  
    for(int i=0; i<lista.length; i++)  
        nova_lista[i] = lista[i];  
  
    nova_lista[lista.length] = novo_item;  
    lista = nova_lista;  
    ultimo_item = lista.length-1;  
  
}
```

◆ Ordem de complexidade:  $O(n)$ .

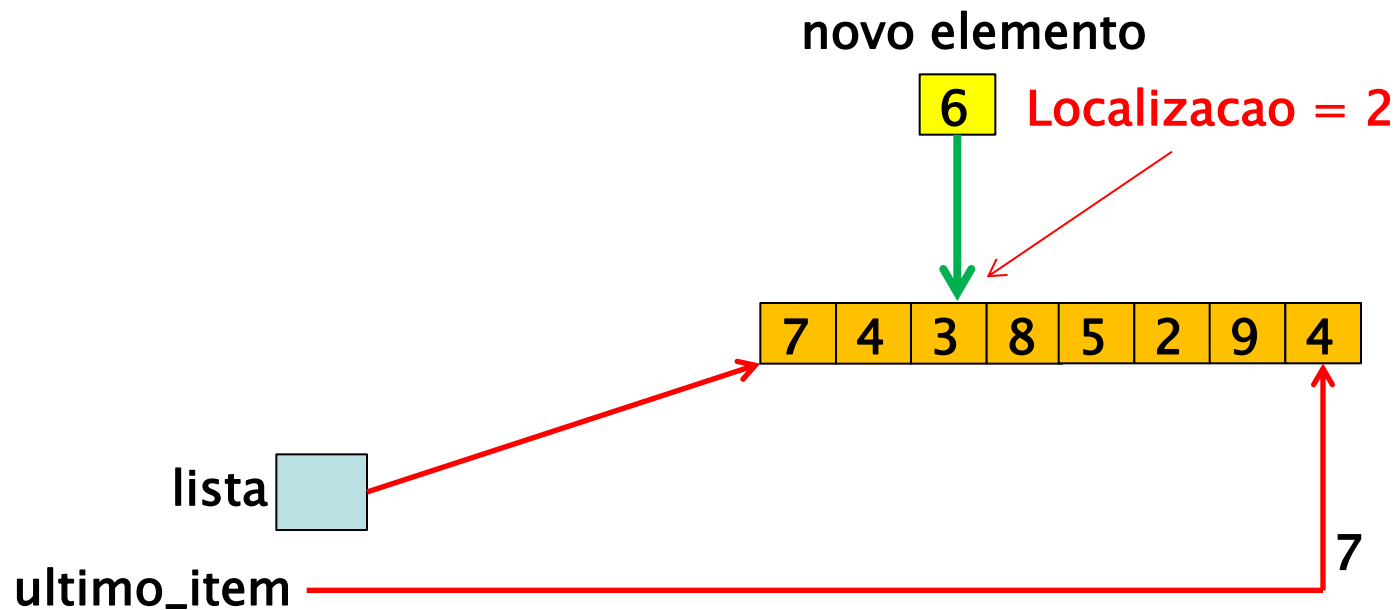




Como inserir um elemento numa posição qualquer da lista ?



# Inserção numa posição qualquer da lista



# Inserção numa posição qualquer da lista

novo elemento

6 Localizacao = 2

7 4 3 8 5 2 9 4

lista

ultimo\_item

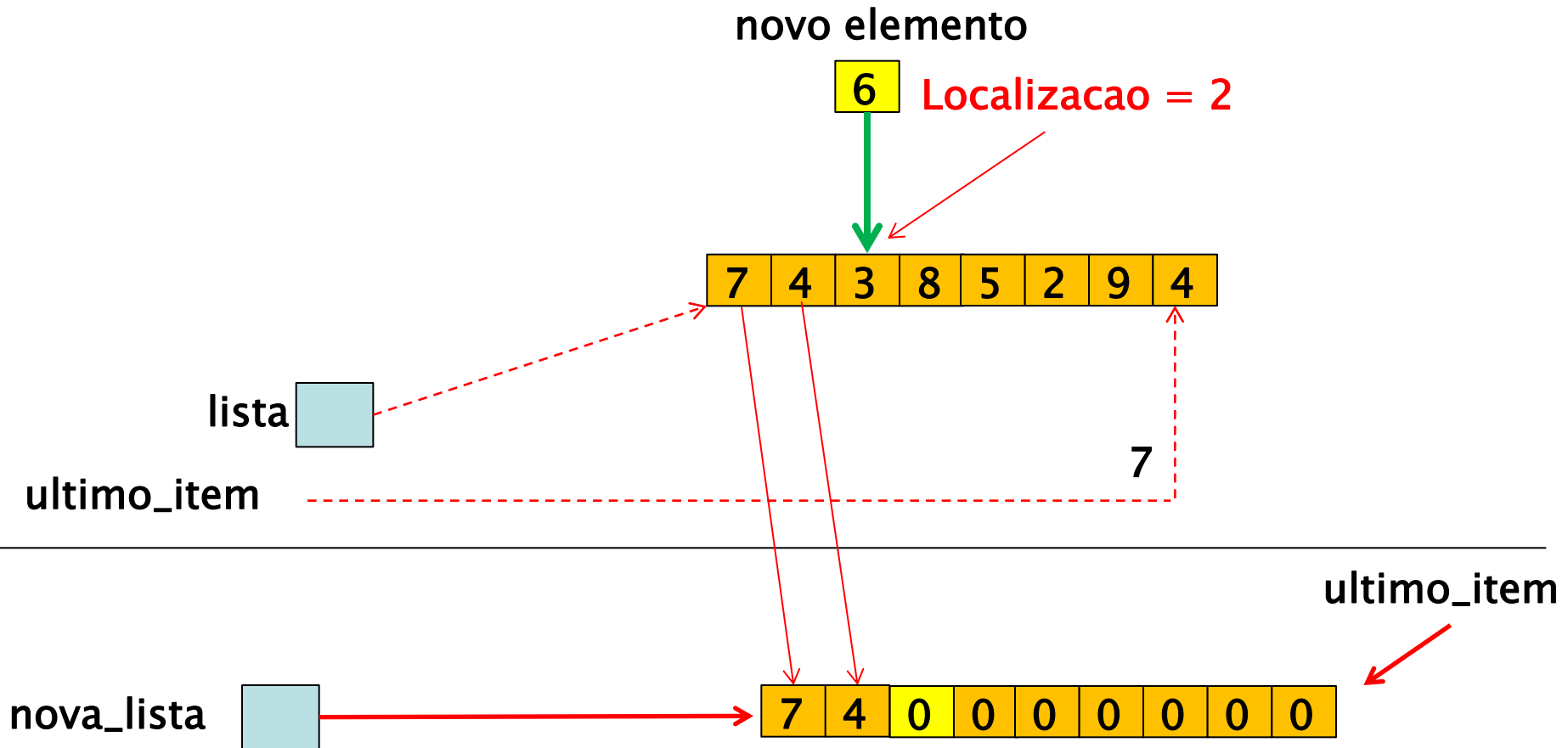
7

0 0 0 0 0 0 0 0 0

Primeiro: cria-se um novo array com um elemento a mais...



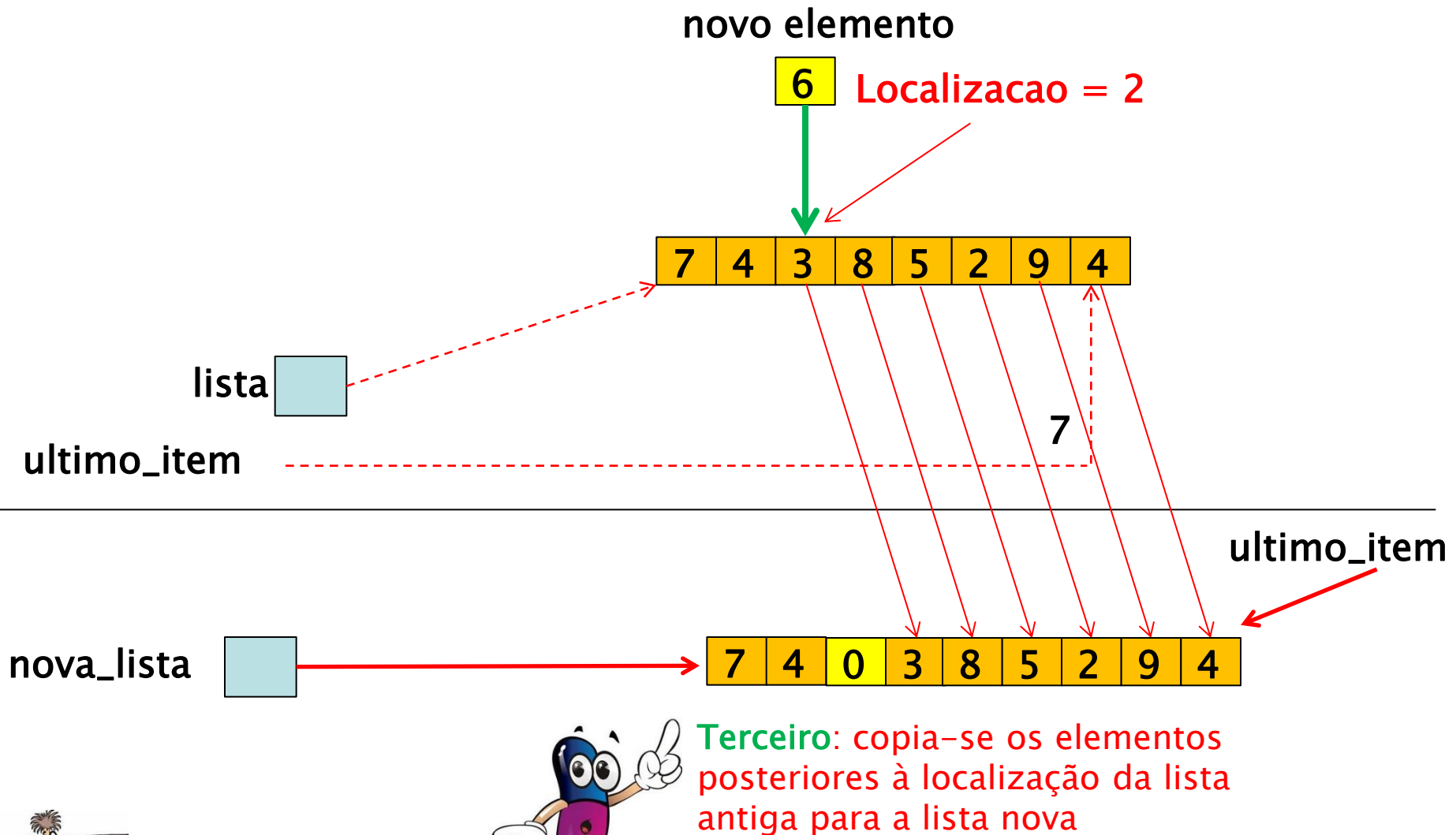
# Inserção numa posição qualquer da lista



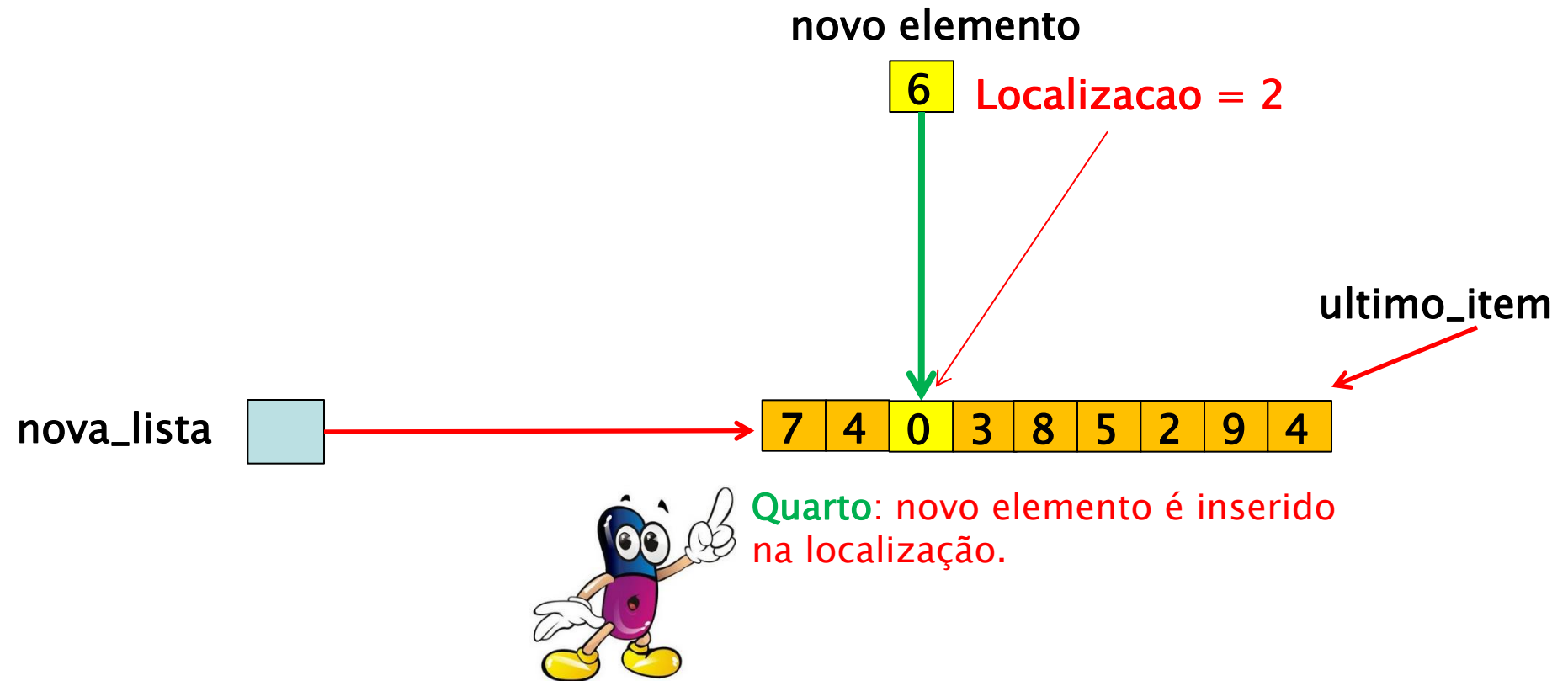
**Segundo:** copia-se os elementos anteriores à localização da lista antiga para a lista nova



# Inserção numa posição qualquer da lista



# Inserção numa posição qualquer da lista





# Implementação

```
public void insereItem(int novo_item, int localizacao) {  
  
    if (localizacao < 0 || localizacao > this.ultimo_item)  
        System.out.println("*** ERRO: Localização inválida...");  
    else {  
        int[] nova_lista = new int[lista.length+1];  
  
        for (int i =0; i < localizacao ; i++ )  
            nova_lista[i] = lista[i];  
  
        for(int i=localizacao; i<lista.length; i++)  
            nova_lista[i+1] = lista[i];  
  
        nova_lista[localizacao] = novo_item;  
        lista = nova_lista;  
        ultimo_item = lista.length-1;  
    }  
}
```

◆ Ordem de complexidade:  $O(n)$ .



# Implementação

```
public void altera_Ultimo(int novo_item){  
    if (ultimo_item == -1)  
        System.out.println("A lista está vazia... não há  
                             como alterar o último elemento");  
  
    else lista[ultimo_item] = novo_item ;  
}  
  
public void altera_Primeiro(int novo_item){  
    if (ultimo_item == -1)  
        System.out.println("A lista está vazia...  
                             não há como alterar o primeiro elemento");  
    else lista[0] = novo_item ;  
}
```

◆ Ordem de complexidade:  $O(1)$ .



# Implementação

```

public void alteraItem(int novo_item, int localizacao) {
    if (localizacao < 0 || localizacao > this.ultimo_item)
        System.out.println("*** ERRO: Localização inválida...");
    else lista[localizacao] = novo_item;
}

public void imprime_Primeiro() {
    if (ultimo_item == -1)
        System.out.println("A lista está vazia... não há primeiro
                             elemento");
    else System.out.println ("Primeiro elemento da lista: " +
                             lista[0] );
}

public void imprime_Ultimo() {
    if (ultimo_item == -1)
        System.out.println("A lista está vazia... não há último
                             elemento");
    else System.out.println ("Último elemento da lista: " +
                             lista[ultimo_item] );
}

```

◆ Ordem de complexidade:  $O(1)$ .





```
package maua;
```

```
public class Test_List_Array {  
  
    public static void main(String[] args) {  
  
        List_Array x = new List_Array();  
        x.Imprime_Lista();  
        x.imprime_Primeiro();  
        x.imprime_Ultimo();  
  
        x.insereItem_Inicio(7);  
        x.Imprime_Lista();  
        x.imprime_Primeiro();  
        x.imprime_Ultimo();  
  
        x.insereItem_Fim(99);  
        x.Imprime_Lista();  
        x.imprime_Primeiro();  
        x.imprime_Ultimo();  
    }  
}
```



```
package maua;

public class Test_List_Array {

    public static void main(String[] args) {

        x = new List_Array(5);
        x.insereItem_Fim(88);
        x.Imprime_Lista();
        x.imprime_Primeiro();
        x.imprime_Ultimo();

        x.insereItem_Inicio(11);
        x.Imprime_Lista();
        x.imprime_Primeiro();
        x.imprime_Ultimo();

        x.insereItem(44, 2);
        x.Imprime_Lista();

    }
}
```





```
package maua;
```

```
public class Test_List_Array {  
  
    public static void main(String[] args) {  
  
        x = new List_Array(5);  
  
        x.altera_Primeiro(999999);  
        x.Imprime_Lista();  
  
        x.altera_Ultimo(777777);  
        x.Imprime_Lista();  
  
        x.alteraItem(1111, 0);  
        x.Imprime_Lista();  
  
        x.alteraItem(2222, 1);  
        x.Imprime_Lista();  
  
        x.alteraItem(3333, 2);  
        x.Imprime_Lista();  
  
    }  
}
```

