INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ

CAMPUS TERESINA-CENTRAL

DIRETORIA DE ENSINO

Estrutura de Dados
Aula 7 – Ordenação

Professora: Elanne Cristina O. dos Santos

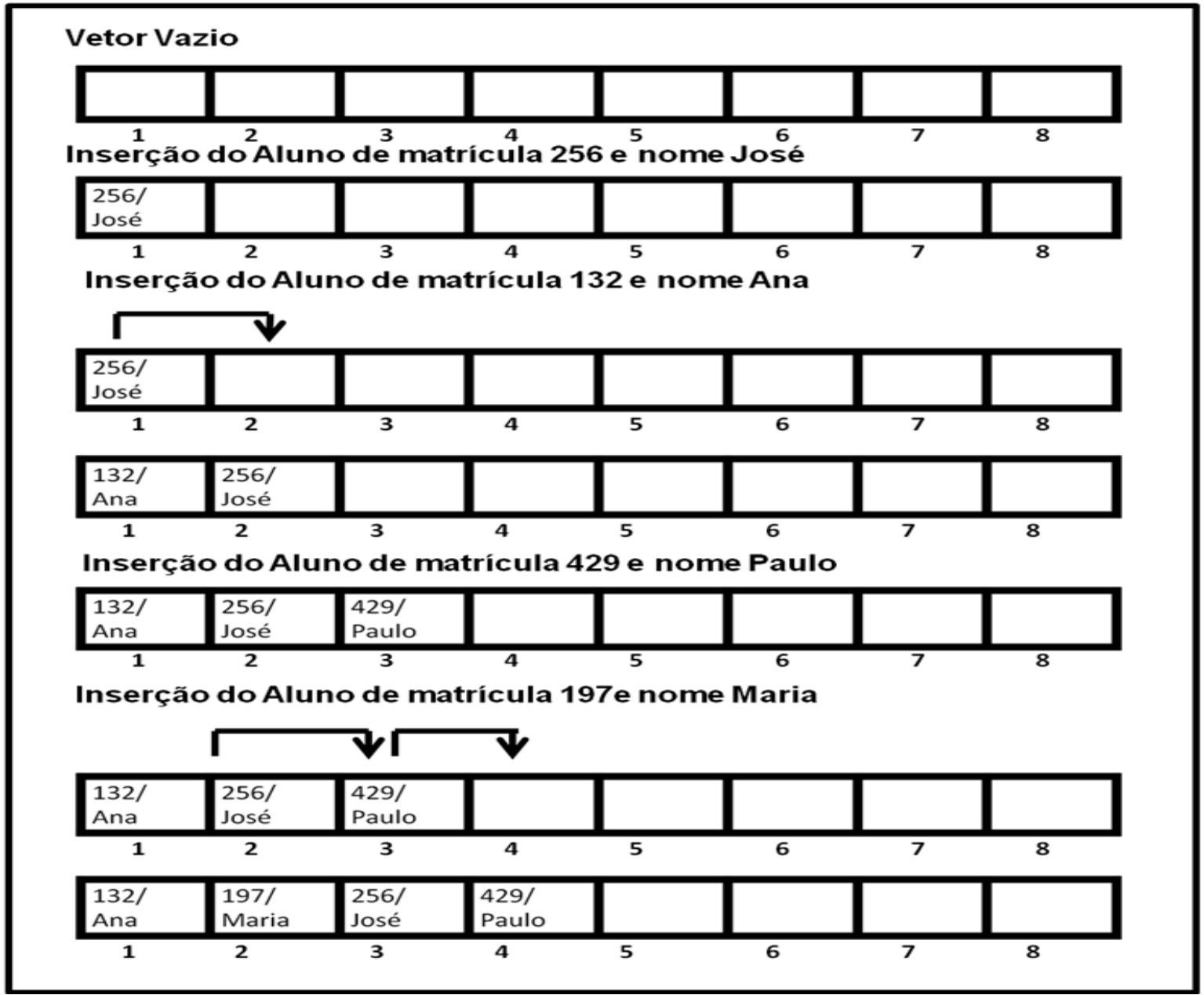
elannecristina.santos@gmail.com
elannecristina.santos@ifpi.edu.br

LISTA ESTÁTICA ORDENADA

- Para inserir um elemento em uma lista ordenada podem ocorrer cinco possibilidades:

1. a lista está cheia:nesse caso a inserção é cancelada;
2. a lista está vazia: o elemento é colocado na primeira posição do vetor;
3. o elemento a ser inserido é menor do que o primeiro da lista;
4. o elemento a ser inserido é maior do que o ultimo da lista;
5. o elemento novo será inserido entre elementos da lista.

LISTA ESTÁTICA ORDENADA



EXEMPLO..

```
int numeros[30];
int quant=0;
void inserirOrdenado(int valor){
    int pos=-1;
    //busca a posicao aonde o novo deve ser inserido
    for(int i=0;i<quant;i++){
        if (valor<numeros[i]){
            pos=i;
            break;
        }
    }
    if (pos==-1){
        numeros[quant]=valor;
        quant++;
    }
    else{//quando insere no meio do vetor
        for(int i=quant;i>pos;i--)
            numeros[i]=numeros[i-1];
        numeros[pos]=valor;
        quant++;
    }
}
```

LISTA DINÂMICA ORDENADA

- Para inserir um elemento em uma lista ordenada podem ocorrer cinco possibilidades:

1. a lista está vazia: o elemento é colocado na primeira posição do vetor;
2. o elemento a ser inserido é menor do que o primeiro da lista;
3. o elemento a ser inserido é maior do que o ultimo da lista;
4. o elemento novo será inserido entre elementos da lista.

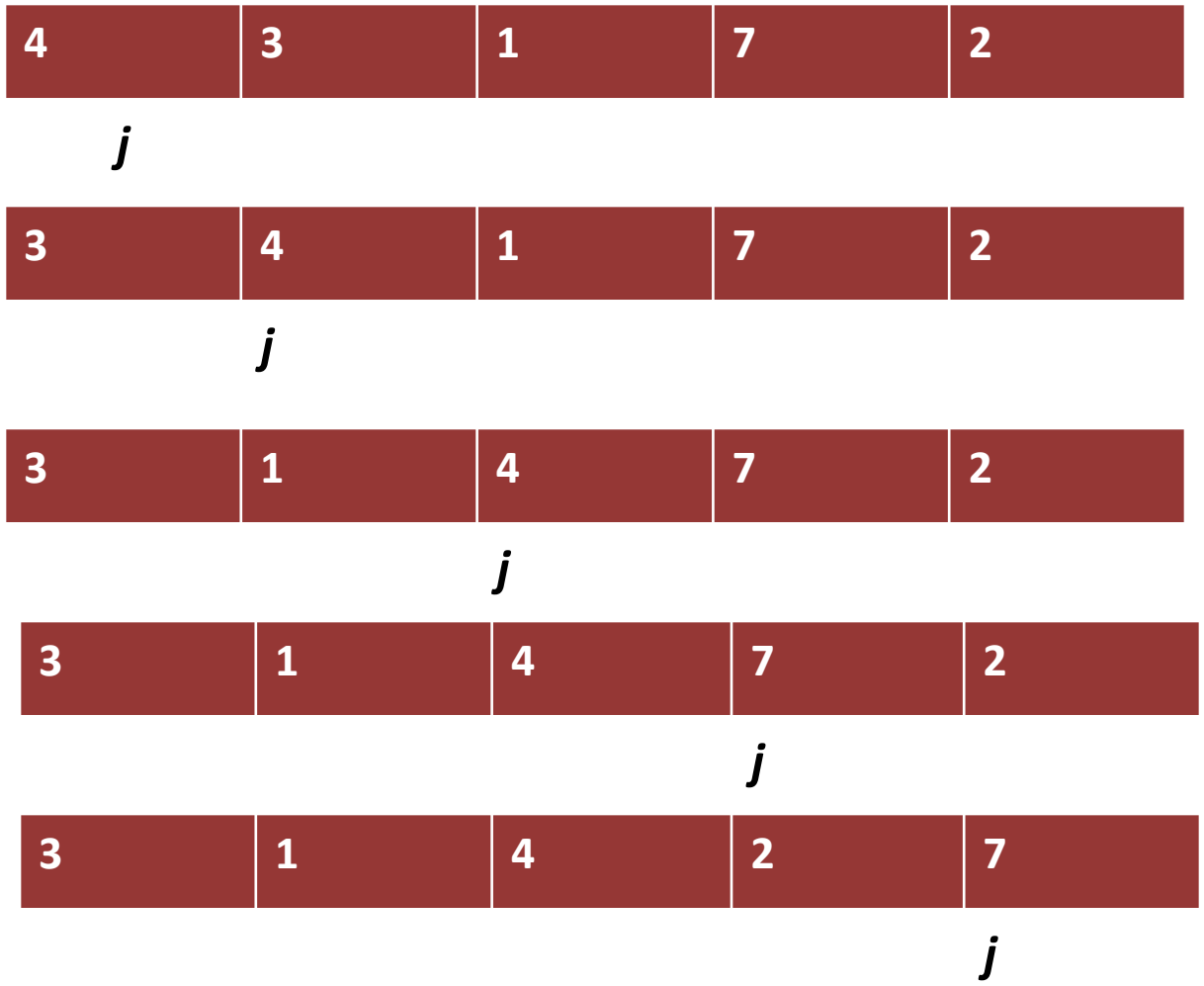
DESSA VEZ A MANIPULAÇÃO SE DÁ ATRAVÉS DOS PONTEIROS!!

Ordenação pelo Método da Bolha Bubblesort

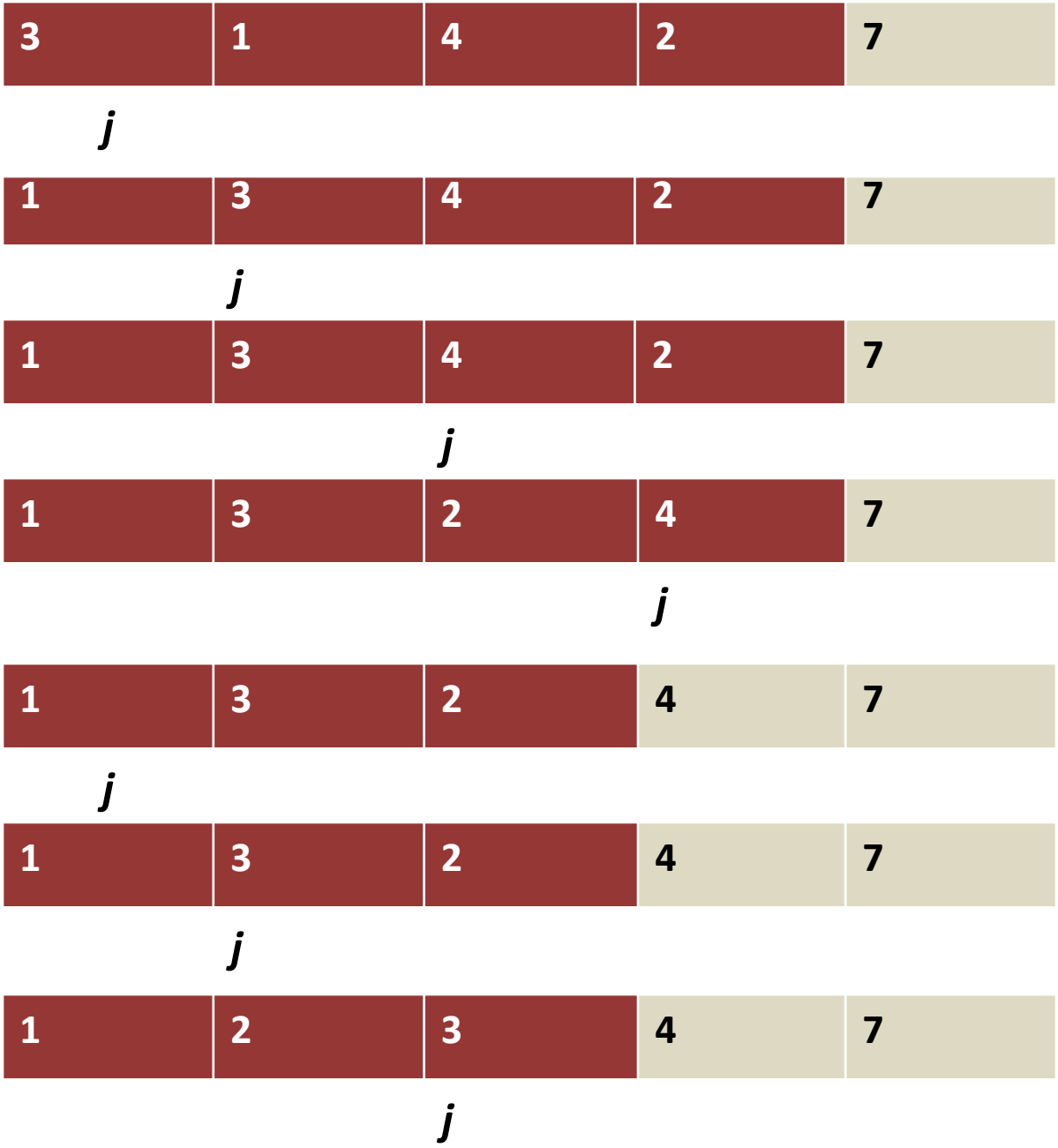
- Algoritmo:
  - Percorra o vetor inteiro comparando elementos adjacentes (dois a dois)
  - Troque as posições dos elementos se eles estiverem fora de ordem
  - Repita os dois passos acima com os primeiros n-1 itens, depois com os primeiros n-2 itens, até que reste apenas o um item

Exemplo – Bolha Bubblesort

I = 4



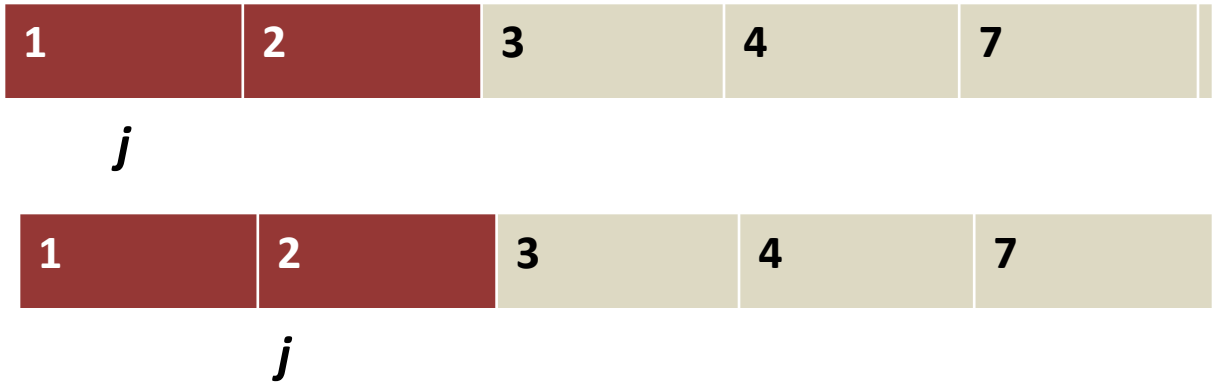
I = 3



I = 2

- .....

I = 1



```
void bolha(int quant, int* v){
    int i, j;
    for(i=quant-1; i>=1; i--){
        for (j=0; j<i; j++){
                if (v[j] > v[j+1]){
                        int temp = v[j];
                        v[j] = v[j + 1];
                        v[j+1] = temp;
                }
        } } }
```

?

- Na situação:

I = 4

- Quantas comparações vão acontecer?
- Todas são necessárias neste caso?

?

- Número de operações não se altera se vetor já está (parcialmente) ordenado

  – Como melhorar?

```
void bolha(int quant, int* v){
    int i, j;
    int troca;
    for(i=quant-1; i>=1; i--){
        troca = 0;
        for (j=0; j<i; j++){
                if (v[j] > v[j+1]){
                        int temp = v[j];
                        v[j] = v[j + 1];
                        v[j+1] = temp;
                        troca = 1;
                } }
        if (troca==0){
                break;
        } } }
```

Método da Bolha Melhorado: Termina execução quando nenhuma troca é realizada após uma passada pelo vetor

❑RESUMINDO BUBBLE SORT...

- Melhor caso: vetor ordenado
- Pior caso: vetor invertido
- Método muito simples, mas custo alto:
  - Adequado apenas se arquivo pequeno
  - Ruim se registros muito grandes

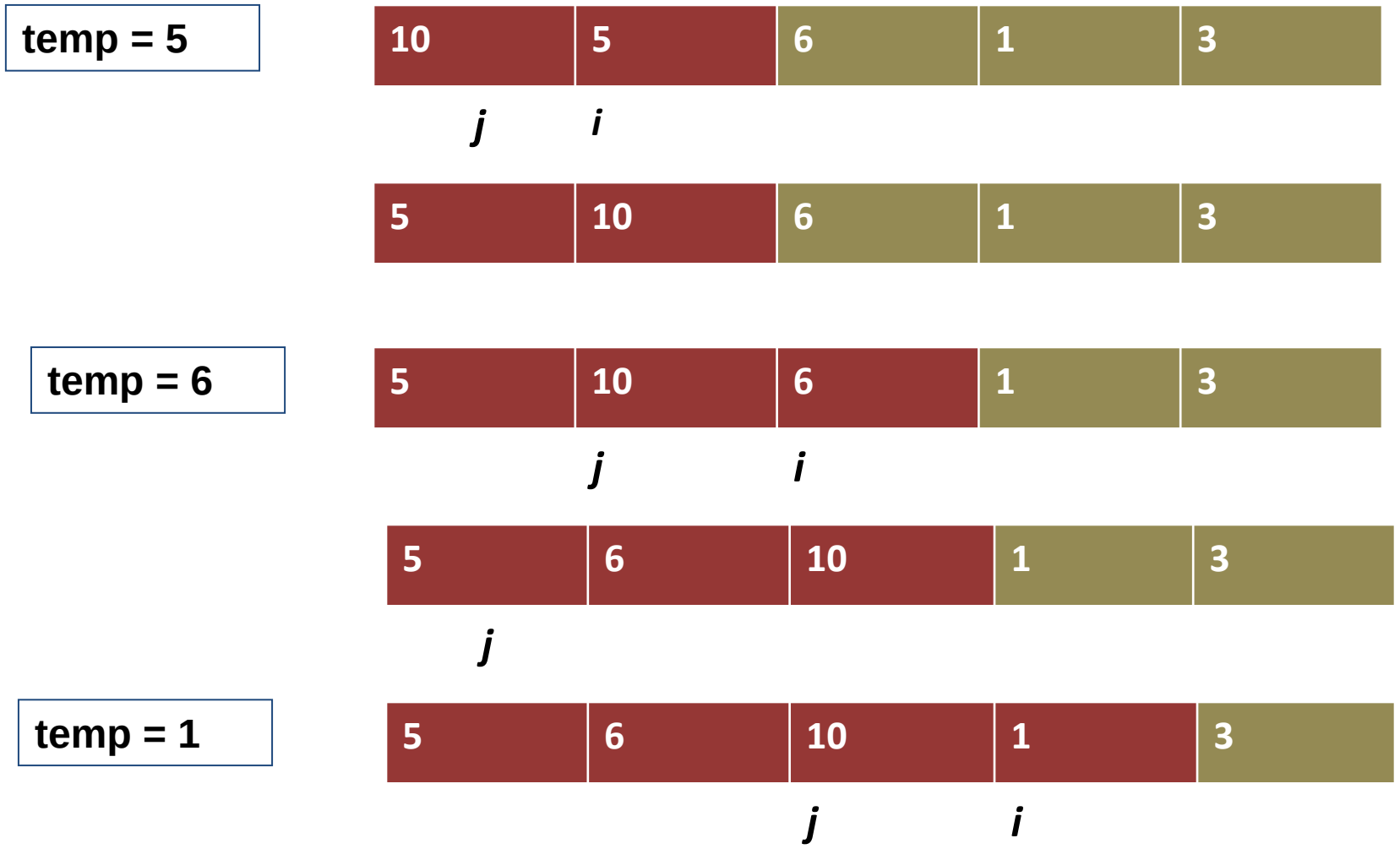
ORDENAÇÃO POR INSERÇÃO

- Ordena um vetor da esquerda para a direita, por ordem crescente ou decrescente.
- À medida que o vetor vai sendo percorrido ele deixa seus elementos à esquerda ordenados.

ORDENAÇÃO POR INSERÇÃO

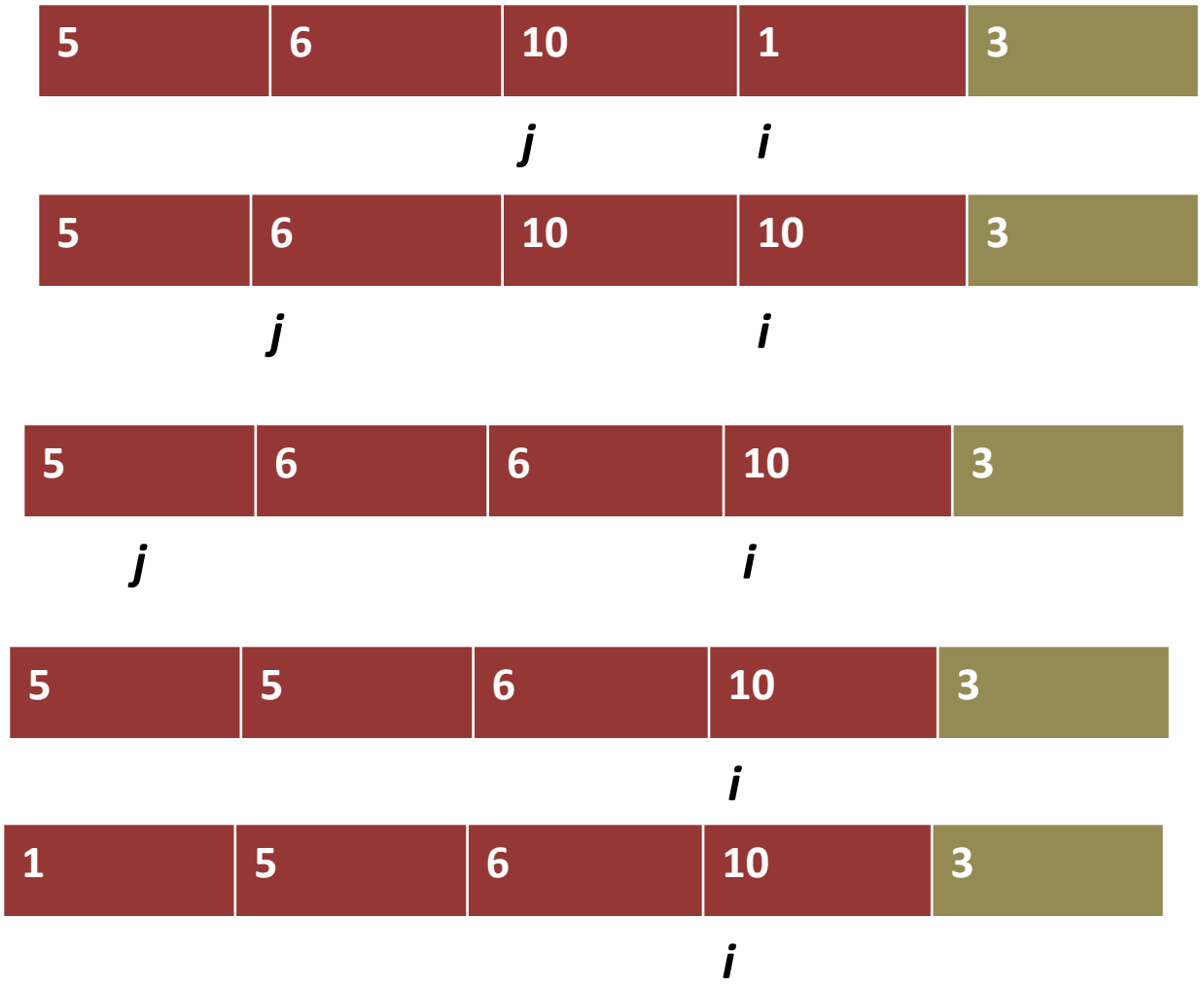
- Percorre um vetor, sempre a partir do primeiro índice desordenado (inicialmente o 2º elemento), e em seguida procura inseri-lo na posição correta, comparando-o com o seu(s) anterior(s) e trocando-os de lugar enquanto ele for menor que seu(s) precedente(s).

ORDENAÇÃO POR INSERÇÃO



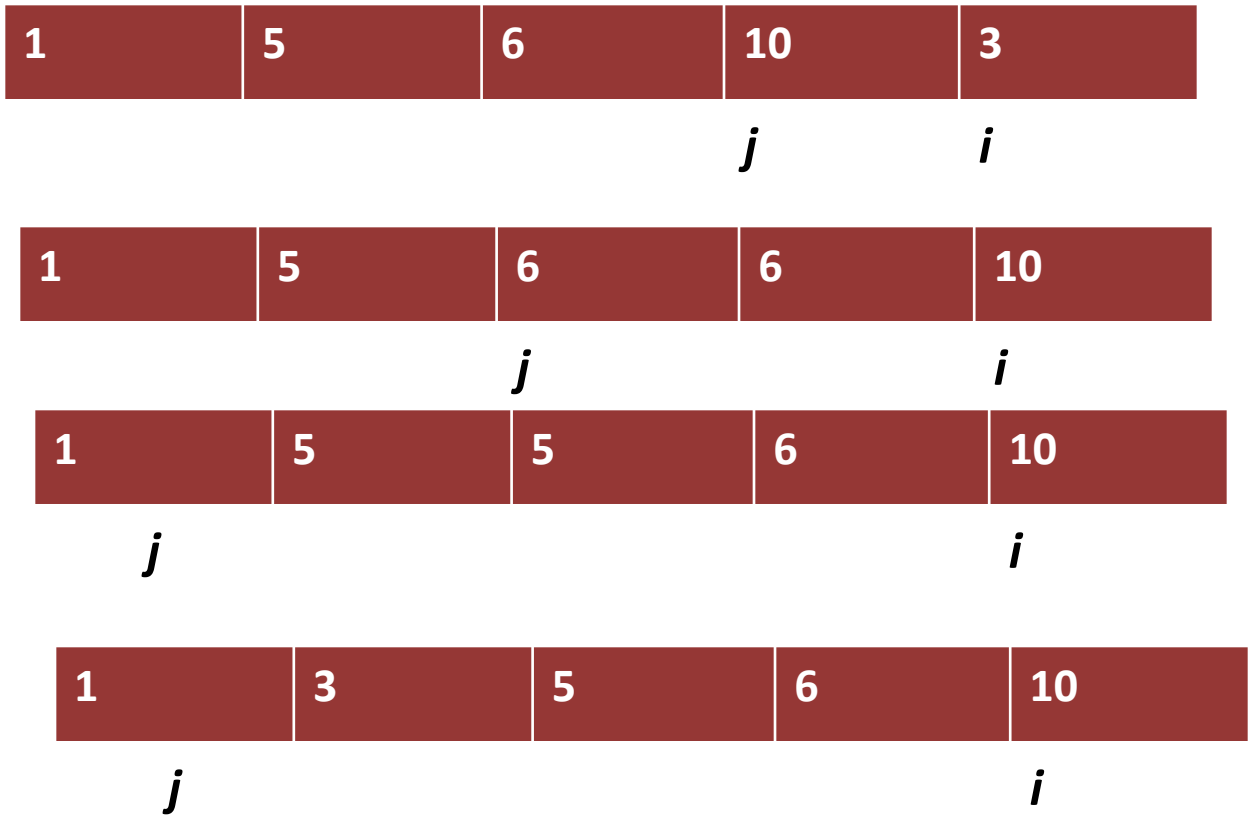
ORDENAÇÃO POR INSERÇÃO

temp = 1



ORDENAÇÃO POR INSERÇÃO

temp=3



ORDENAÇÃO POR INSERÇÃO

```
int myarray[]={10,5,6,1,3};
void ord_insercao(int v[],int tam){
    int j, i, temp;
*   for (i=1;i<tam;i++)
    {    //guarda o elemento que esta verificando

         temp = v[i];
         //verificando os elementos anteriores a posicao i
         j=i-1;
**       while (v[j]>temp && j>=0)
         {
              v[j+1]=v[j];
              j--;
         }
         //insere o elemento na posicao correta (ordenada) ate i.

         v[j+1] = temp;
    }
}
```

QuickSort

- O algoritmo Quick Sort é um método de ordenação muito rápido e eficiente;

- Idéia: semelhante a um dicionário, ordena-se as palavras, tendo como objetivo reduzir o problema original em subproblemas para assim poder ser resolvido mais fácil e rapidamente.

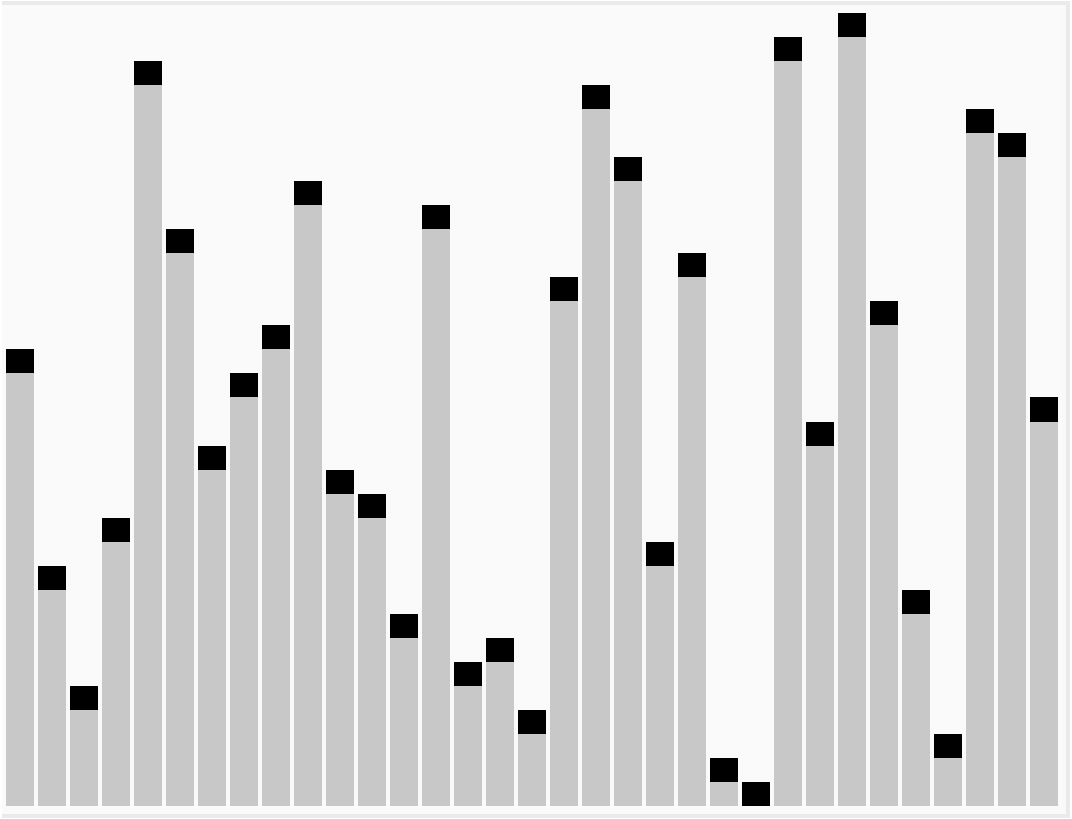
Quicksort

- Este método divide a tabela em duas sub-tabelas, a partir de um elemento chamado **pivô**.

- **Uma das sub-tabelas contém os elementos menores que o pivô** enquanto a **outra contém os maiores**. **O pivô é colocado entre ambas**, ficando na posição correta.

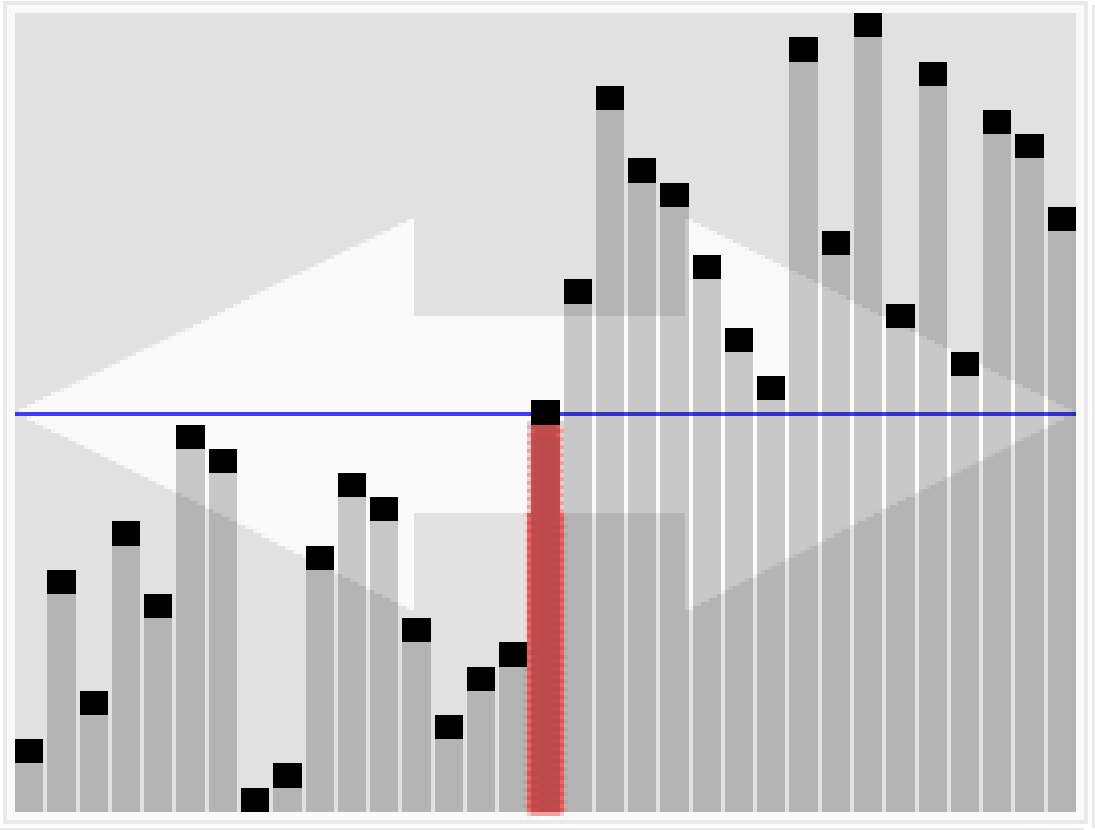
QuickSort

- EXEMPLO: Essas barras de tamanhos diferentes devem ser alinhadas em ordem crescente:



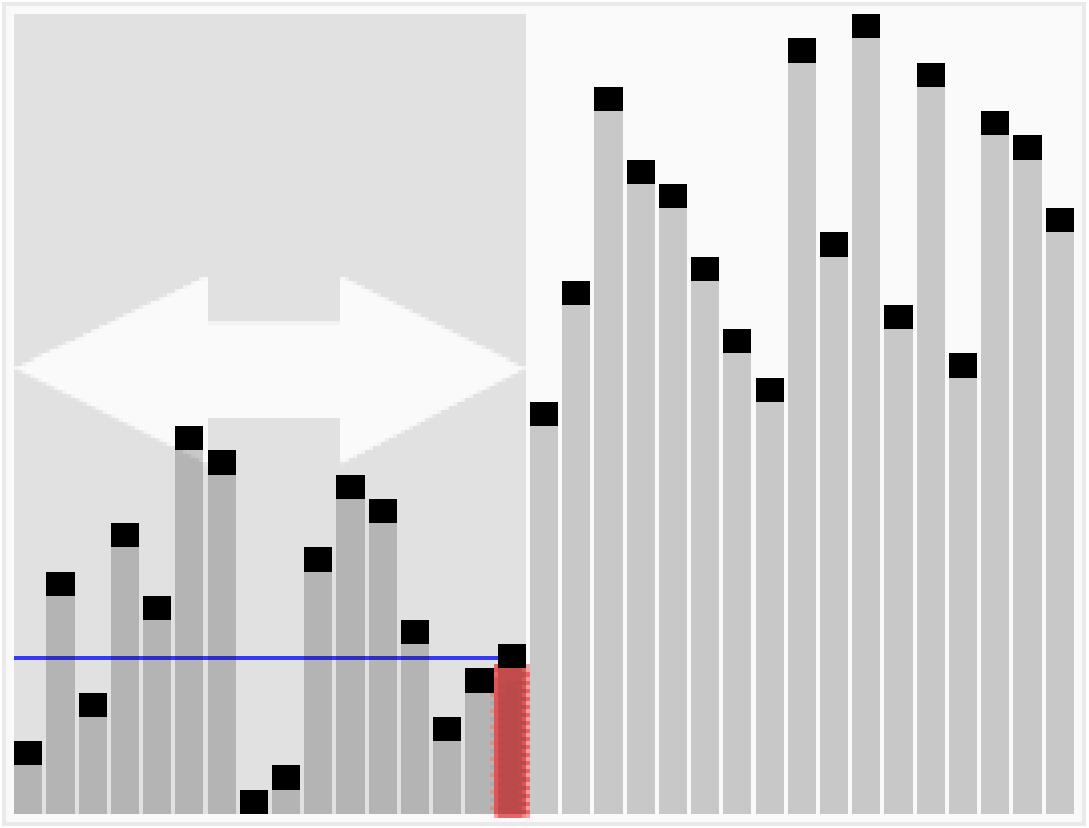
QuickSort

- Os elementos são organizados de maneira que os menores ficam do lado esquerdo do **pivô** e os maiores do lado direito (são as duas sub-tabelas).



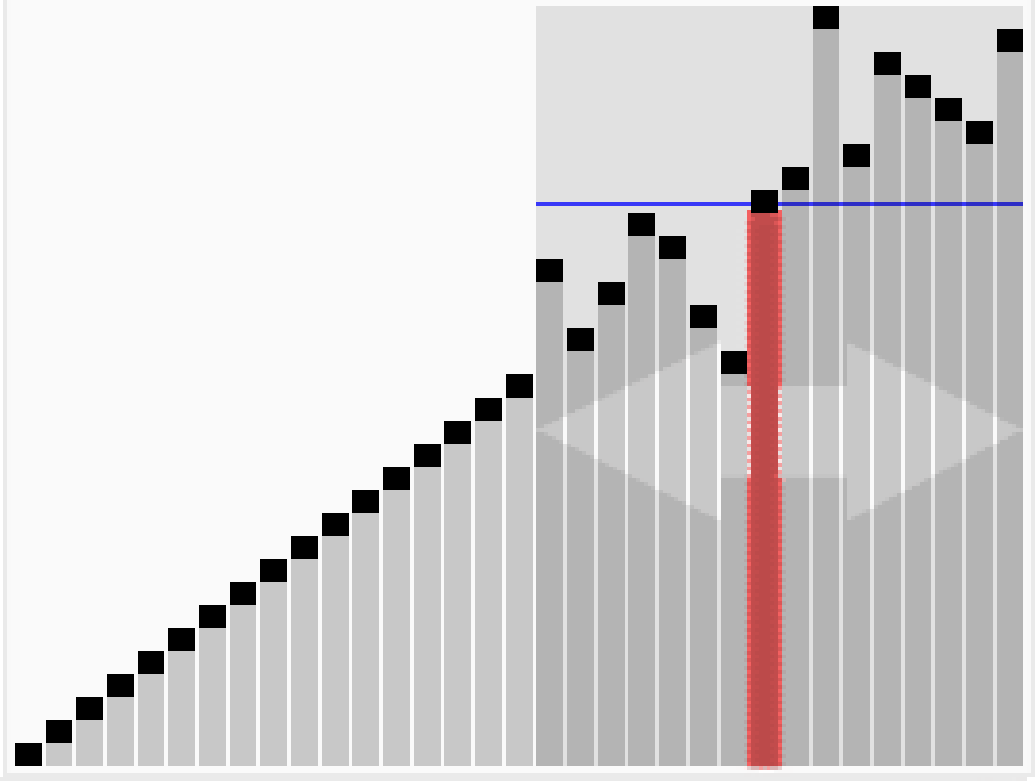
QuickSort

- Depois de encontrar a posição do **pivô** e separar em duas tabelas, ele passa para uma das sub-tabelas e escolhe outro **pivô**.



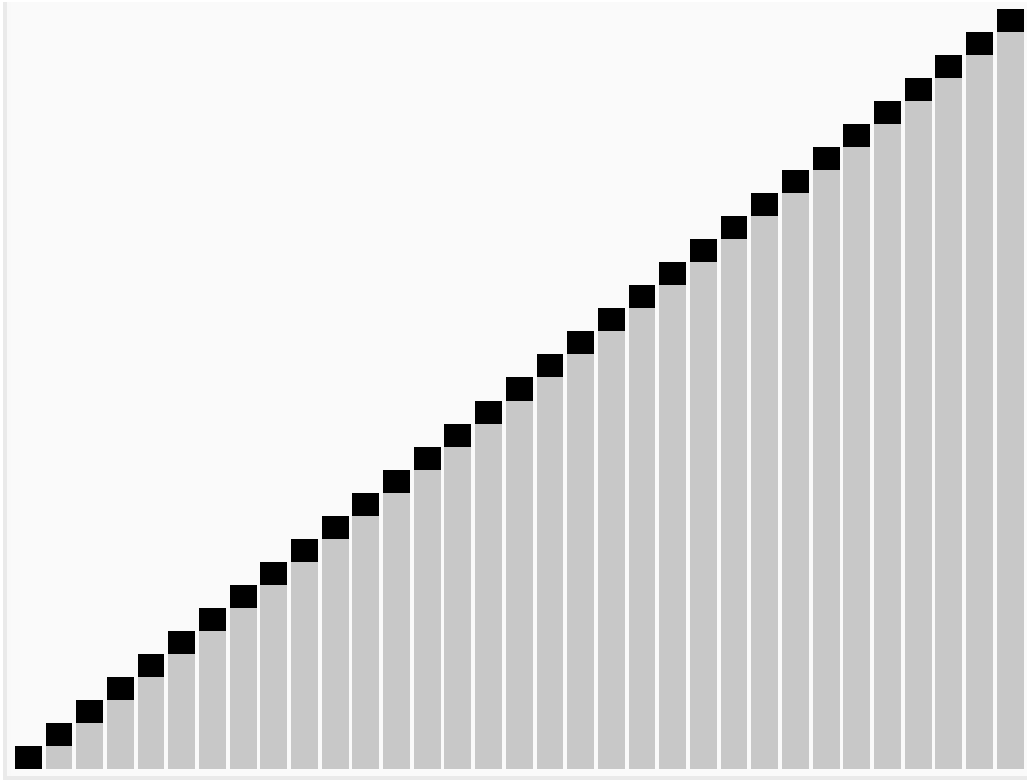
QuickSort

- O pivô do início e todo o lado esquerdo já está ordenado, Agora ele passa para a outra sub-tabela, o lado direito do primeiro **pivô**, escolhe outro **pivô** e ordena.



QuickSort

- E depois de ordenado ele fica assim:



QuickSort

```
#include<stdio.h>
#include<iostream>
using namespace std;
void Quick(int vetor[10], int inicio, int fim);
int main(){
    int vetor[6] = {7, 9, 4, 3,6,1};
    int i;
    Quick(vetor, 0, 5);
    printf("\n2.Vetor ordenado:\n");
    for(i = 0; i <= 5; i++){
        printf("%d ", vetor[i]);
    }
    printf("\n");
}
```

```
void Quick(int vetor[10], int inicio, int fim){
    int pivo, aux, i, j, meio;
    i = inicio;
    j = fim;
    meio = (int) ((i + j) / 2);
    pivo = vetor[meio];
    do{
        while (vetor[i] < pivo) i = i + 1;
        while (vetor[j] > pivo) j = j - 1;
        if(i <= j){
            aux = vetor[i];
            vetor[i] = vetor[j];
            vetor[j] = aux;
            i = i + 1;
            j = j - 1;
        }
    }while(j > i);
    if(inicio < j) {
     Quick(vetor, inicio, j);
    }
    if(i < fim) {
     Quick(vetor, i, fim);
    }
}
```

Quicksort

7	9	4	3	6	1
l					j

1	9	4	3	6	7
	i		j		

1	3	4	9	6	7
		l ,j			

Quicksort

1	3	4	9	6	7
i		j	i		

1	3	4	9	6	7
	l,j		i		

1	3	4	9	6	7
	i	i			j

1	3	4	9	6	7
	i		i		j

1	3	4	9	6	7
	i		i		j

1	3	4	7	6	9
	i			l,j	

Merge Sort

- O princípio básico de funcionamento do Merge Sort é simples e intuitivo: o algoritmo divide a lista a ser ordenada em várias sublistas, sempre em duas partes, até que as listas restantes contenham apenas um elemento.

- Após esta divisão o algoritmo faz o caminho inverso, ou seja, ele começa a "remontar" o vetor, porém ordenando os elementos (que estão em menor número) enquanto os vetores maiores são recriados.

Merge Sort

- Ex. Vetor: 15 99 52 14 50 64 20 77

1. Divisão em duas partes: 15 99 52 14 - 50 64 20 77

2. Divisão da primeira parte em duas outras partes: 15 99 - 52 14

3. Análise do primeiro par: 15- 99
   Como 15 é menor do que 99, a ordem é mantida.

4. Análise do segundo par: 52 -14
   Como 52 é maior do que 14, a ordem é invertida: 14 -52

5. Agora restam dois pares ordenados: 15 99 - 14 52

Merge Sort

- O algoritmo agora compara o menor valor de cada um dos dois vetores: 15 -14
- Como 14 é menor do que 15, logo 14 será o 1º elemento no vetor auxiliar: 14 X X X
- Compara-se 15 com o outro elemento do segundo vetor. Como 15 é menor do que 52: 14 15 X X
- Agora simplesmente comparam-se os dois elementos restantes. 99 é maior do que 52, logo 52 ocupará a terceira posição, e 99 a quarta posição neste vetor de 4 elementos: 14 15 52 99
- O processo é repetido de forma idêntica para a outra primeira metade do vetor inicial: 20 50 64 77
- O próximo e último passo é fazer a fusão destes dois vetores,
- Feito este processo, o resultado final será:

14 15 20 50 52 64 77 99

Merge Sort

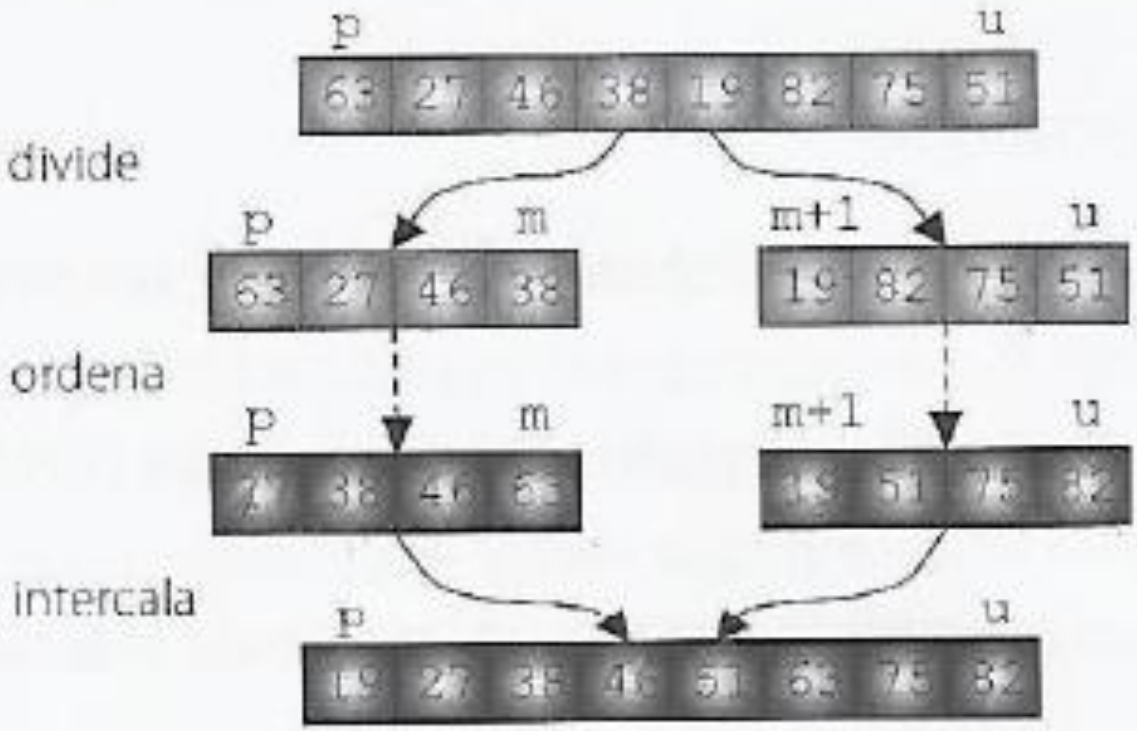


Figura 8.14 | Estratégia recursiva para ordenação por intercalação.

Merge Sort

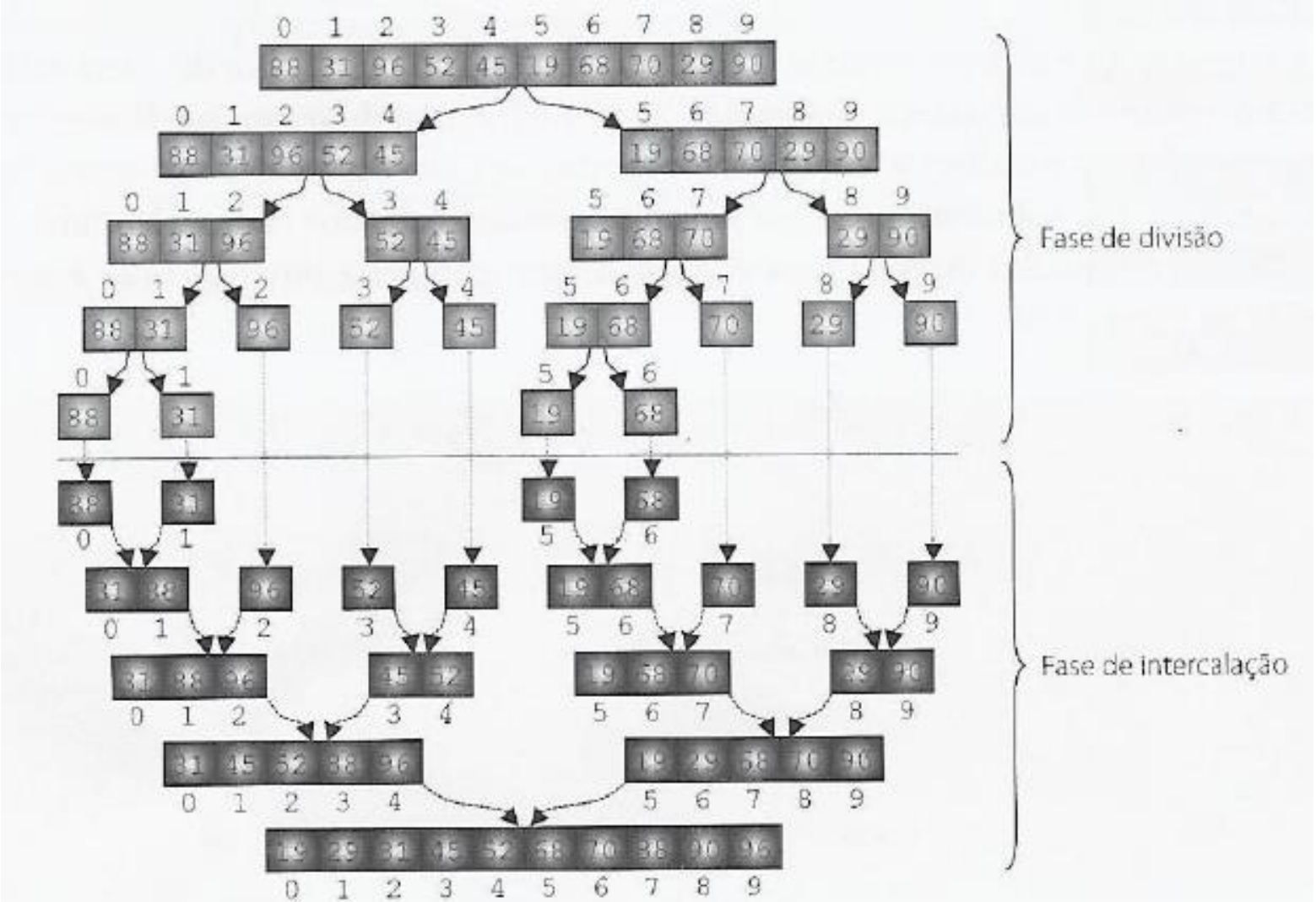


Figura 8.15 | Funcionamento da ordenação por intercalação.

Merge Sort

```
void merge_sort(int v[], int p, int u) {
    if( p == u ) return;
    int m = (p+u)/2;
    merge_sort(v,p,m);
    merge_sort(v,m+1,u);
    intercala(v,p,m,u);
}
```

Merge Sort

```
void intercala(int v[], int p, int m, int u) {
    int *w = malloc((u-p+1)*sizeof(int));
    if( w == NULL ) abort();
    int i = p, j = m+1, k = 0;
    while( i<=m && j<=u )
        if( v[i] < v[j] ) w[k++] = v[i++];
        else w[k++] = v[j++];
    while( i<=m ) w[k++] = v[i++];
    while( j<=u ) w[k++] = v[j++];
    for(k=0; k<=u-p; k++) v[p+k] = w[k];
    free(w);
}
```

Figura 8.13 | Função para intercalação.

Para estudar...

- Faça o método inserir ordenado em uma lista simplesmente encadeada.
- Faça o método inserir ordenado em uma lista duplamente encadeada.
- Usando um vetor inteiros realize as ordenações *“bubble sort”* e *“quicksort”.* Pesquise o tempo de execução dos dois algoritmos. Qual deles é mais eficiente?