

## **Algoritmos e Estruturas de Dados**

### **Disciplina 301477**

Programa de Pós-graduação em  
Computação Aplicada

**Prof. Alexandre Zaghetto**  
<http://alexandre.zaghetto.com>  
[zaghetto@unb.br](mailto:zaghetto@unb.br)



<http://www.nickgentry.com/>

Universidade de Brasília  
Instituto de Ciências Exatas  
Departamento de Ciência da Computação

# **Módulo 13**

## **Classificação e Pesquisa**

---

## 1. Classificação e Pesquisa

- **Classificação:**

- ✓ A classificação consiste em ordenar elementos, seguindo algum critério para isso. Por exemplo, a classificação alfabética para dados literais, ou crescente e decrescente para dados numéricos.
- ✓ Para isso temos muitos métodos conhecidos, dentre eles: *InsertionSort*, *BubbleSort*, *QuickSort*, *HeapSort*, *ShellSort*.
- ✓ Neste aula trataremos especificamente dos métodos *BubbleSort* e *Quicksort*.

## 1. Classificação e Pesquisa

- **Pesquisa:**

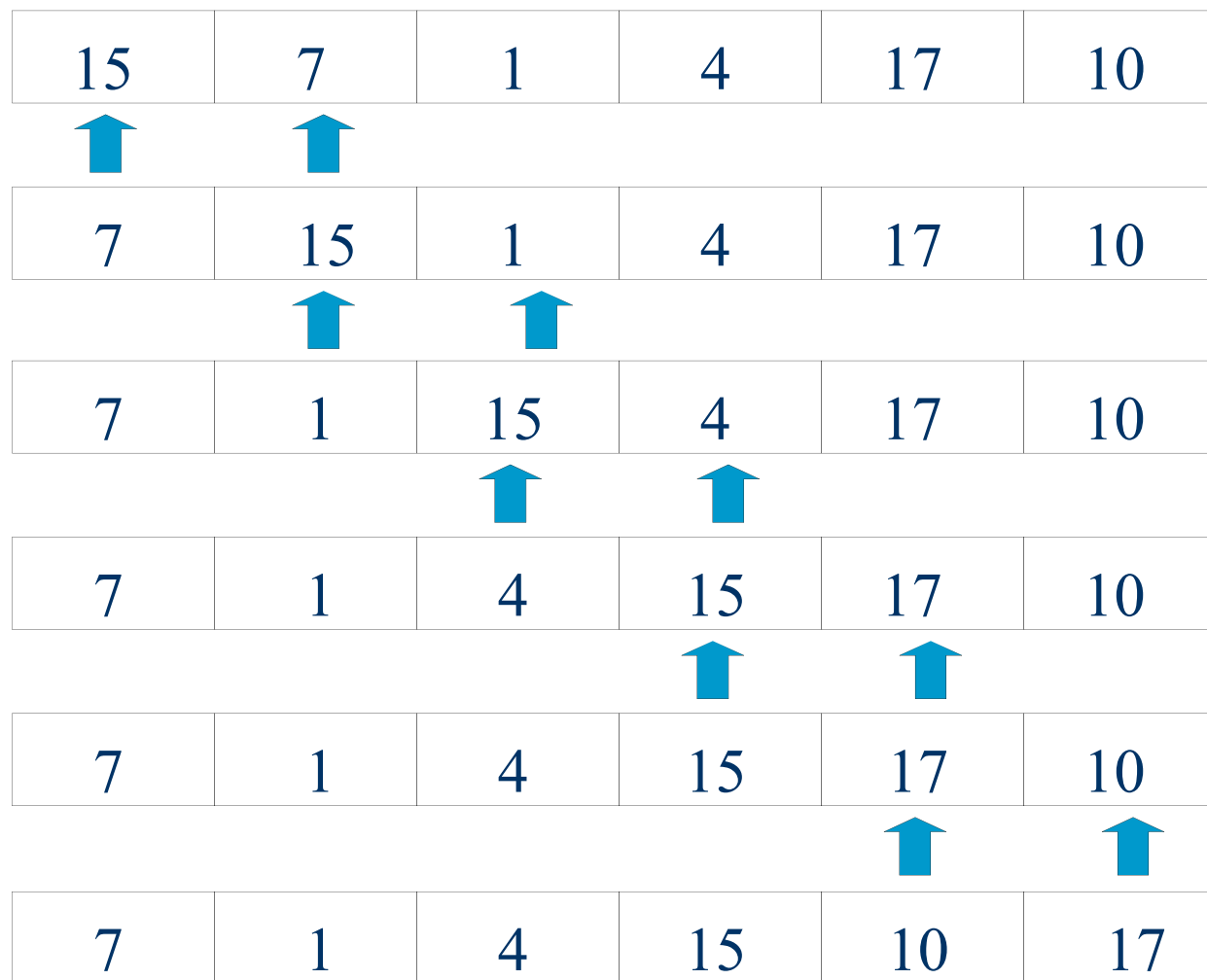
- ✓ Já a pesquisa, consiste na verificação da existência de um valor dentro de um conjunto de dados.
- ✓ Trataremos neste capítulo da *pesquisa sequencial* e da *pesquisa binária*.

## 1. Classificação e Pesquisa

- *BubbleSort*:

- ✓ Embora o método *BubbleSort* não seja o método mais eficiente para classificar uma grande quantidade de dados, ele é muito simples, e por motivos didáticos utilizaremos o mesmo para apresentar uma forma de classificação.
- ✓ Este método consiste em ler todo o vetor, comparando os elementos vizinhos entre si. Caso estejam fora da ordem (determinada pela classificação em questão), os mesmos trocam de posição entre si.
- ✓ Procede-se assim até o final do vetor. O algoritmo repete as instruções acima até que o vetor seja percorrido e não haja mais nenhuma troca de posição entre os elementos consecutivos do vetor.

## 1. Classificação e Pesquisa

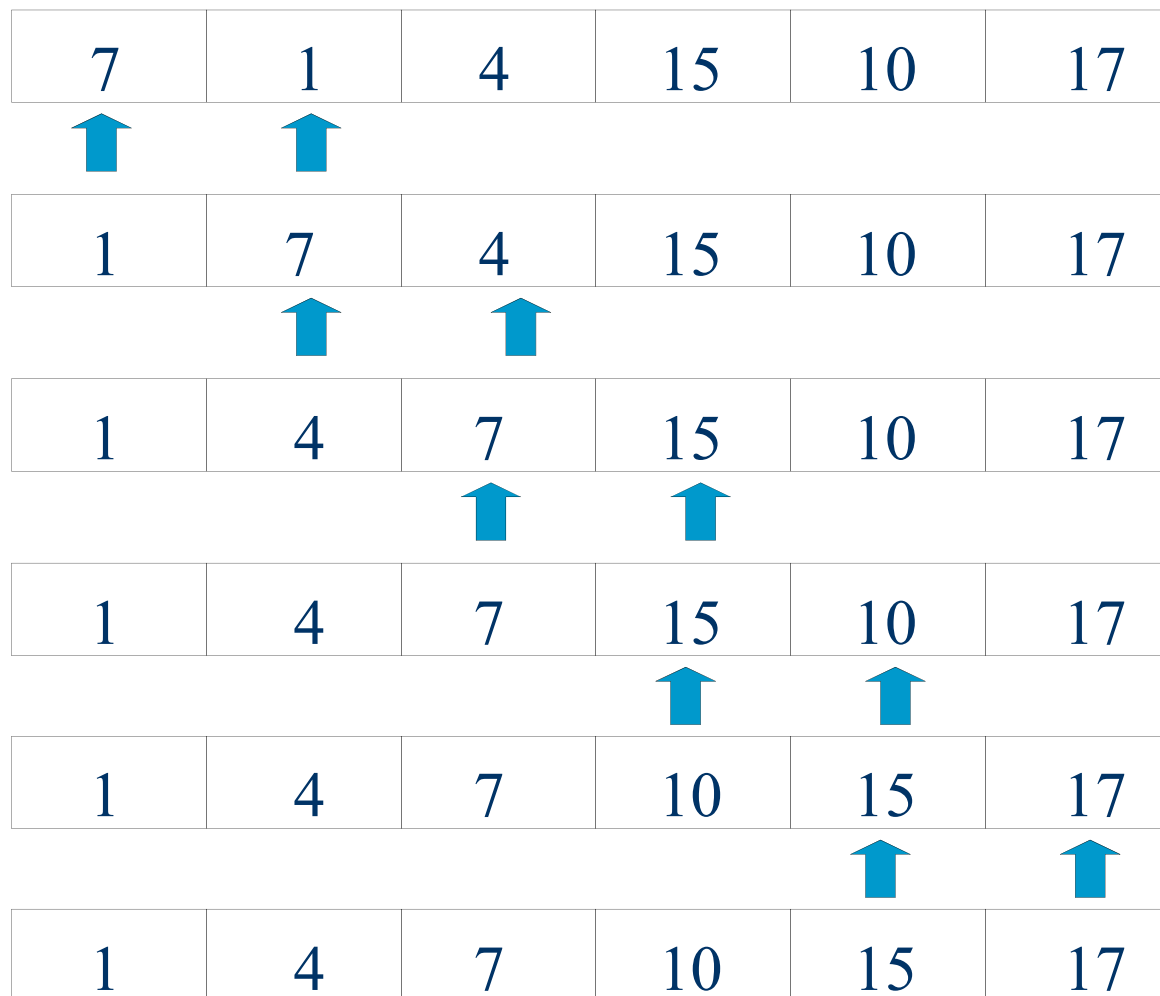


## 1. Classificação e Pesquisa

- *BubbleSort:*

- ✓ O vetor está agora mais próximo de uma ordenação, mas ainda não da ordenação desejada.
- ✓ Isso indica que devemos repetir o processo mais vezes até que o vetor esteja ordenado.
- ✓ Executando mais uma vez o trecho de algoritmo...

## 1. Classificação e Pesquisa





## 1. Classificação e Pesquisa

- *BubbleSort*:

- ✓ O número máximo de execuções do trecho do algoritmo para que o vetor fique ordenado é  $N-1$  vezes, onde  $N$  é o número de elementos do vetor.
- ✓ É sempre necessário repetir  $N-1$  vezes?
  - No exemplo apresentado em apenas duas execuções do algoritmo o vetor já estava ordenado!
- ✓ Como controlar o número de vezes?
  - Se o vetor já estiver ordenado, não precisa repetir o passo mais uma vez.
  - Se não houve trocas entre os elementos do vetor ao executar o trecho do algoritmo, então ele está ordenado.

## 1. Classificação e Pesquisa

- *BubbleSort*:
  - ✓ A implementação desse algoritmo fica como exercício para o aluno.

## 1. Classificação e Pesquisa

- *QuickSort*: este método parte do princípio de que é mais rápido ordenar dois vetores com  $n/2$  elementos cada um, do que um com  $n$  elementos (conceito **dividir para conquistar**).
- O primeiro passo é dividir o vetor original.
- Esse procedimento é denominado **particionamento**.
  - ✓ Deve-se escolher uma das posições do vetor a qual é denominada **pivô**:

$V[i]$ .

## 1. Classificação e Pesquisa

- ✓ Uma vez escolhido o pivô, os elementos do vetor são movimentados de forma que:
  - O subvetor à esquerda do pivô contenha somente os elementos cujos valores são menores que o pivô; e
  - O subvetor da direita contenha valores maiores ou iguais ao valor do pivô.

$V[0], \dots, V[i-1], V[i], V[i+1], \dots, V[n-1]$

- ✓ O procedimento é repetido em cada subvetor até que o vetor todo esteja ordenado.
- ✓ Existem várias formas de se escolher o pivô. Vamos escolher o valor do meio do (sub)vetor.

## 1. Classificação e Pesquisa

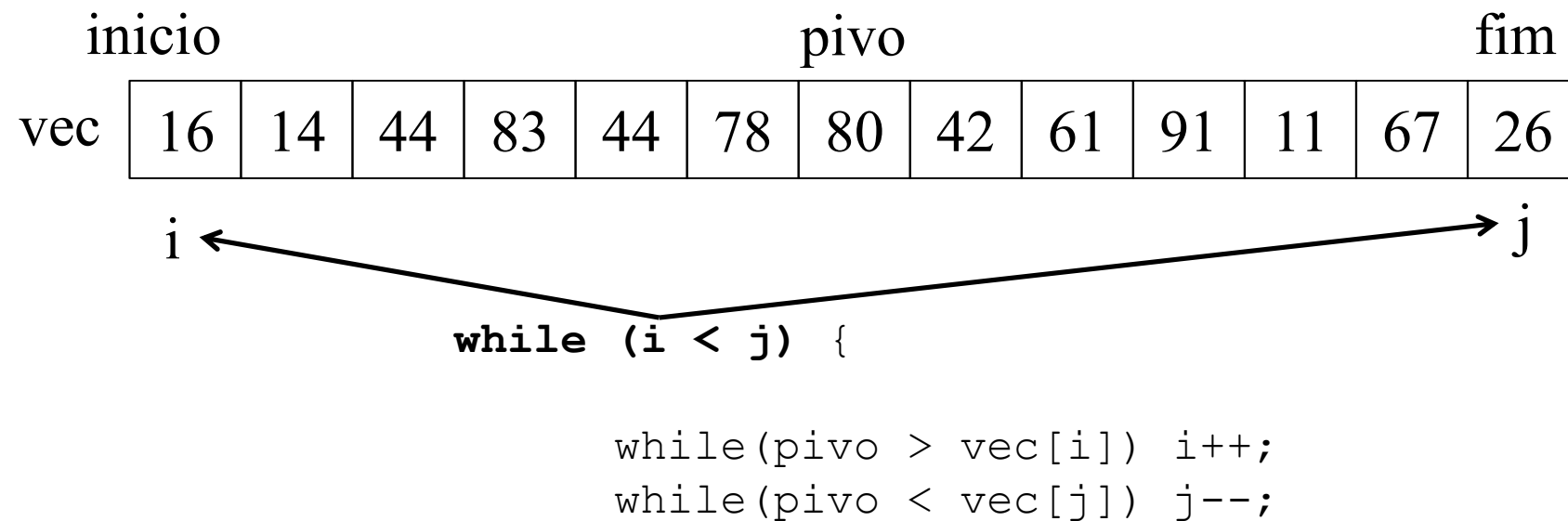
- Algoritmo:
  - a) O pivô é escolhido no meio do vetor. O elemento é colocado numa variável auxiliar **pivo**;
  - b) São iniciadas duas variáveis auxiliares  $i = \text{inicio}$  e  $j = \text{fim}$ ;
  - c) O vetor é percorrido do início até que se encontre um  $V[i] \geq \mathbf{pivo}$  ( $i$  é incrementado no processo).
  - d) O vetor é percorrido a partir do fim até que se encontre um  $V[j] \leq \mathbf{pivo}$  ( $j$  é decrementado no processo).
  - e)  $V[i]$  e  $V[j]$  são trocados;  $i$  é incrementado de 1 e  $j$  é decrementado de 1.

## 1. Classificação e Pesquisa

- Algoritmo:
  - f) O processo é repetido até que  $i$  e  $j$  se cruzem em algum ponto do vetor (ou seja,  $i > j$ ).
  - g) Quando são obtidos os dois segmentos do vetor por meio do processo de partição, realiza-se a ordenação de cada um deles de forma recursiva.

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`



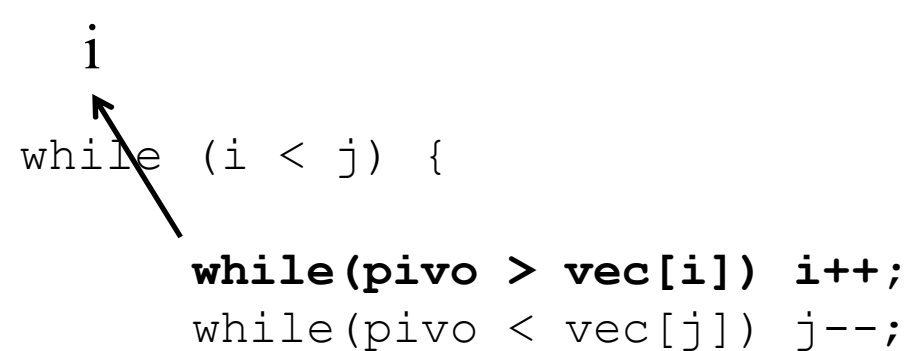
## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio			pivo								fim	
vec	16	14	44	83	44	78	80	42	61	91	11	67	26

$i$   $j$

```
while (i < j) {  
    while(pivo > vec[i]) i++;  
    while(pivo < vec[j]) j--;
```





## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio				pivo				fim				
vec	16	14	44	83	44	78	80	42	61	91	11	67	26

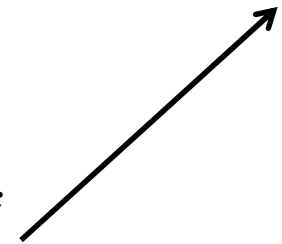
i

j

```
while (i < j) {
```

```
    while(pivo > vec[i]) i++;
```

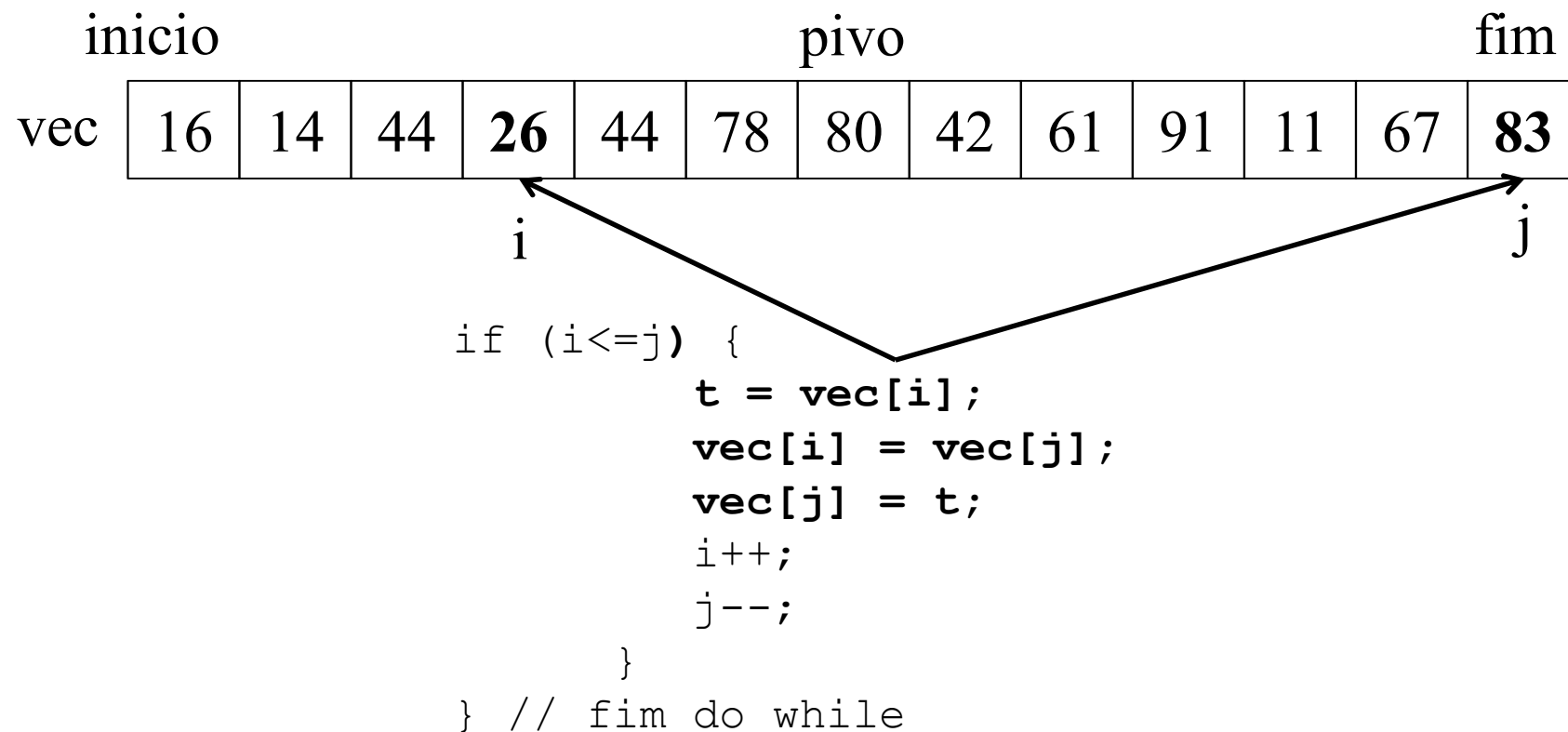
```
    while(pivo < vec[j]) j--;
```





## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

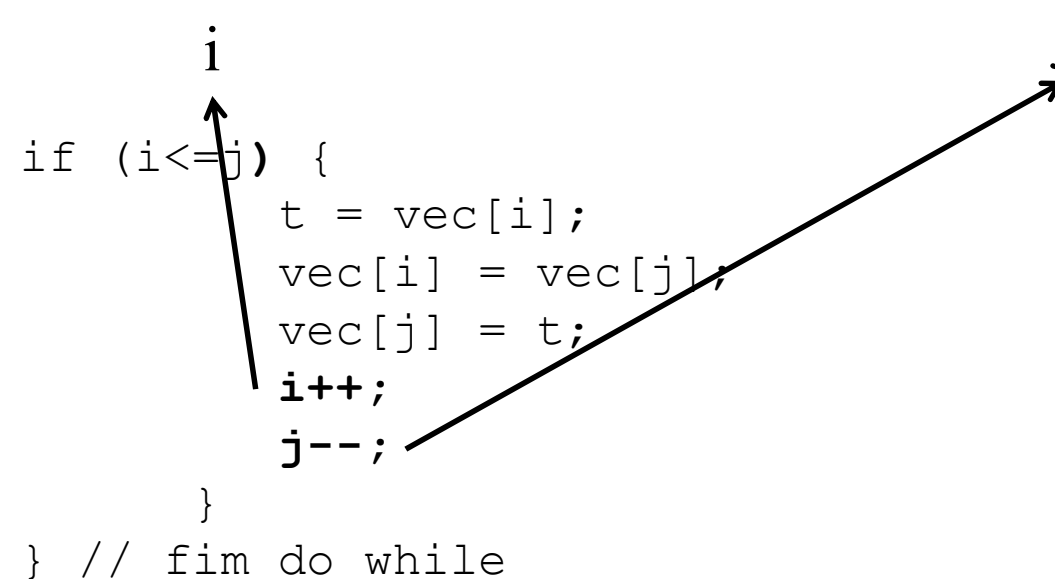


## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio					pivo					fim			
vec	16	14	44	26	44	78	80	42	61	91	11	67	83	


```
if (i <= j) {  
    t = vec[i];  
    vec[i] = vec[j];  
    vec[j] = t;  
    i++;  
    j--;  
}  
// fim do while
```



## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio					pivo					fim			
vec	16	14	44	26	44	78	80	42	61	91	11	67	83	

*i*  *j*

```
while (i < j) {  
  
    while(pivo > vec[i]) i++;  
    while(pivo < vec[j]) j--;
```

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio				pivo				fim				
vec	16	14	44	26	44	78	80	42	61	91	11	67	83

i

j

```
while (i < j) {
```

```
    while(pivo > vec[i]) i++;
```

```
    while(pivo < vec[j]) j--;
```

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio				pivo				fim				
vec	16	14	44	26	44	78	80	42	61	91	11	67	83

i

j

```
while (i < j) {
```


```
    while(pivo > vec[i]) i++;
```

```
    while(pivo < vec[j]) j--;
```

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio					pivo					fim			
vec	16	14	44	26	44	78	80	42	61	91	11	67	83	

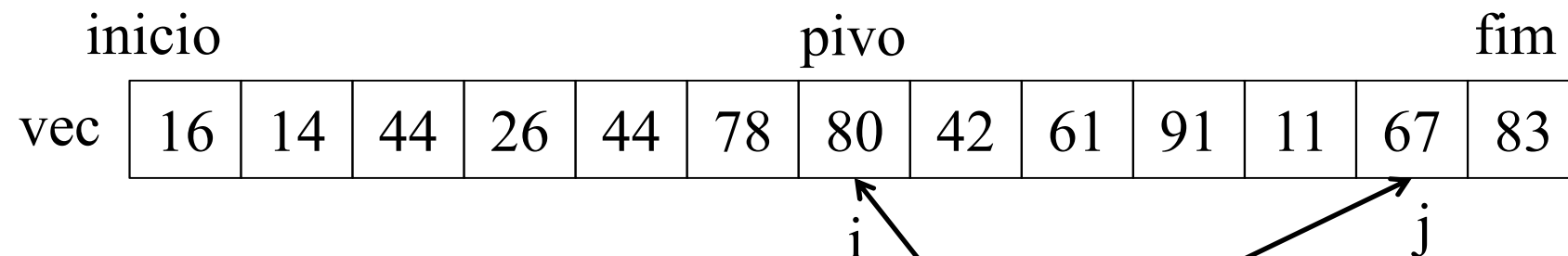


```
if (i<=j) {  
    t = vec[i];  
    vec[i] = vec[j];  
    vec[j] = t;  
    i++;  
    j--;  
}  
} // fim do while
```



## 1. Classificação e Pesquisa

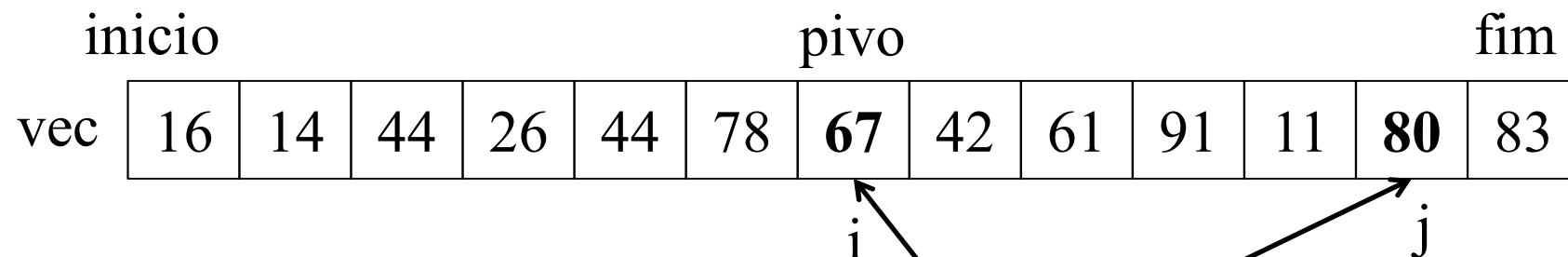
- Exemplo: `quicksort(vec, inicio, fim);`



```
if (i <= j) {  
    t = vec[i];  
    vec[i] = vec[j];  
    vec[j] = t;  
    i++;  
    j--;  
}  
} // fim do while
```

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`



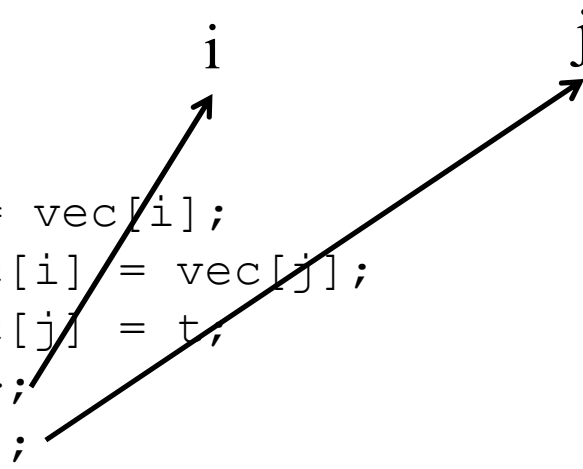
```
if (i <= j) {  
    t = vec[i];  
    vec[i] = vec[j];  
    vec[j] = t;  
    i++;  
    j--;  
}  
} // fim do while
```

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio				pivo				fim				
vec	16	14	44	26	44	78	67	42	61	91	11	80	83

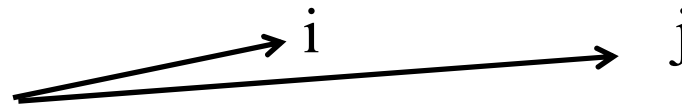
```
if (i<=j) {  
    t = vec[i];  
    vec[i] = vec[j];  
    vec[j] = t;  
    i++;  
    j--;  
}  
} // fim do while
```



## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio					pivo					fim			
vec	16	14	44	26	44	78	67	42	61	91	11	80	83	



```
while (i < j) {  
  
    while(pivo > vec[i]) i++;  
    while(pivo < vec[j]) j--;
```

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio				pivo				fim				
vec	16	14	44	26	44	78	67	42	61	91	11	80	83

```
while (i < j) {
```

```
    while(pivo > vec[i]) i++;
```

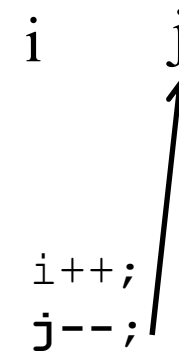
```
    while(pivo < vec[j]) j--;
```

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio				pivo				fim				
vec	16	14	44	26	44	78	67	42	61	91	11	80	83


```
while (i < j) {  
    while(pivo > vec[i]) i++;  
    while(pivo < vec[j]) j--;
```



## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

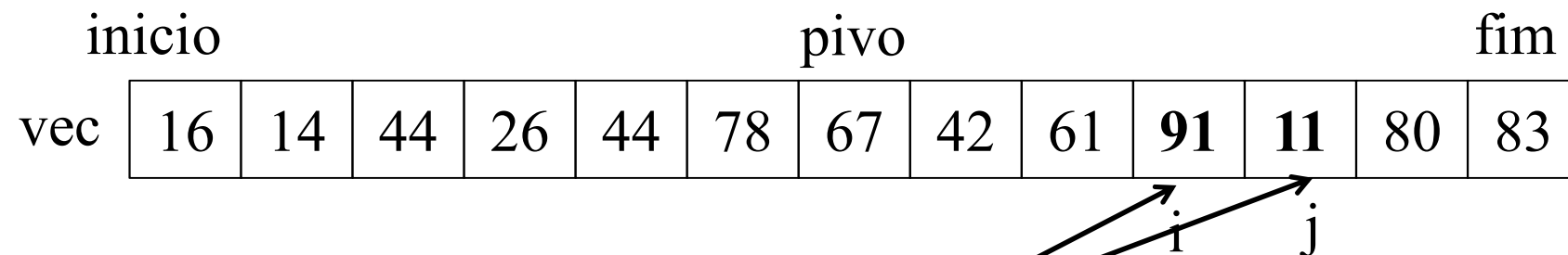
	inicio				pivo				fim				
vec	16	14	44	26	44	78	67	42	61	91	11	80	83



```
if (i<=j) {  
    t = vec[i];  
    vec[i] = vec[j];  
    vec[j] = t;  
    i++;  
    j--;  
}  
} // fim do while
```

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

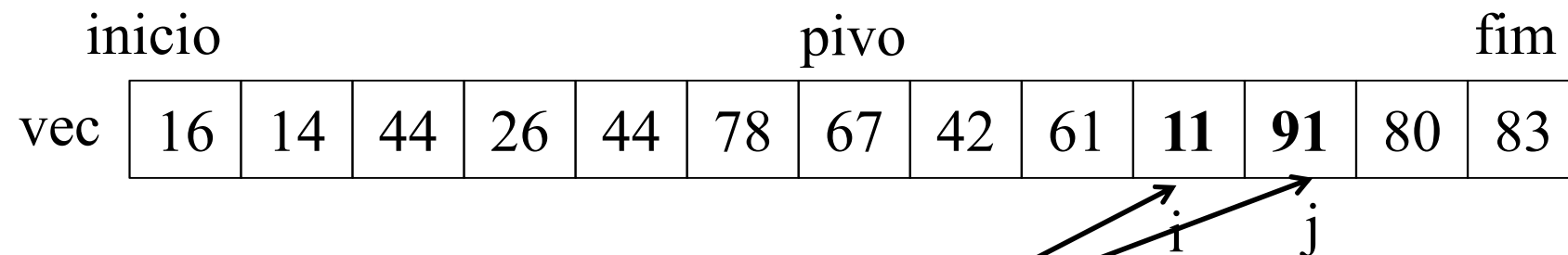


```
if (i <= j) {  
    t = vec[i];  
    vec[i] = vec[j];  
    vec[j] = t;  
    i++;  
    j--;  
}  
} // fim do while
```



## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`



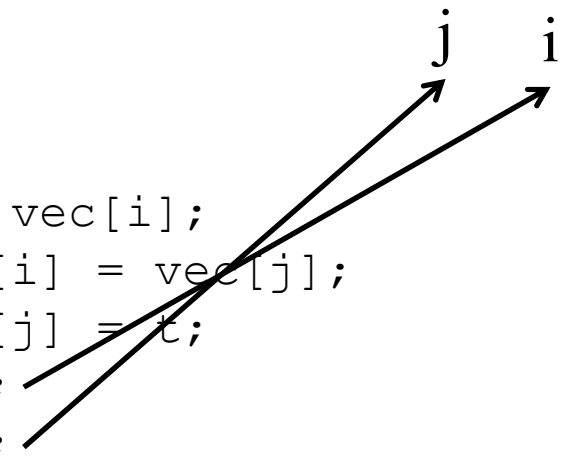
```
if (i <= j) {  
    t = vec[i];  
    vec[i] = vec[j];  
    vec[j] = t;  
    i++;  
    j--;  
}  
} // fim do while
```

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio					pivo					fim			
vec	16	14	44	26	44	78	67	42	61	11	91	80	83	


```
if (i<=j) {  
    t = vec[i];  
    vec[i] = vec[j];  
    vec[j] = t;  
    i++;  
    j--;  
}  
} // fim do while
```



## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio				pivo				fim				
vec	16	14	44	26	44	78	67	42	61	11	91	80	83

FALSO 

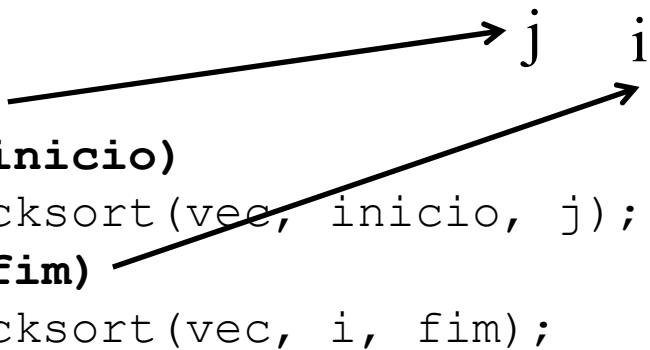
```
while (i < j) {  
  
    while(pivo > vec[i]) i++;  
    while(pivo < vec[j]) j--;
```

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

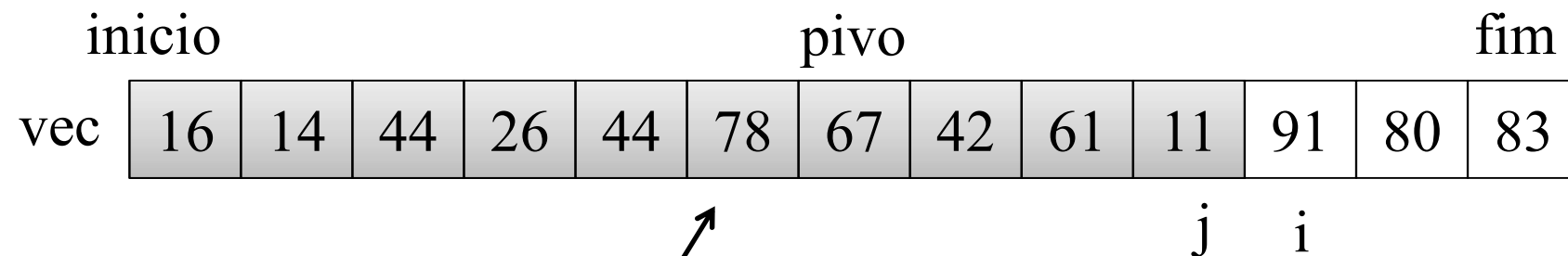
	inicio				pivo				fim				
vec	16	14	44	26	44	78	67	42	61	11	91	80	83

```
if (j > inicio)
    quicksort(vec, inicio, j);
if (i < fim)
    quicksort(vec, i, fim);
```



## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`



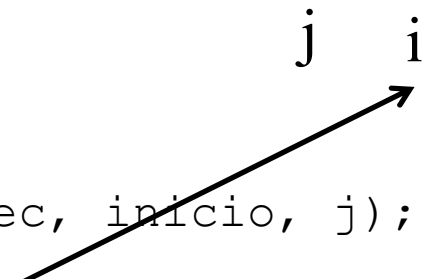
```
if (j > inicio)
    quicksort(vec, inicio, j);
if (i < fim)
    quicksort(vec, i, fim);
```

## 1. Classificação e Pesquisa

- Exemplo: `quicksort(vec, inicio, fim);`

	inicio					pivo					fim		
vec	16	14	44	26	44	78	67	42	61	11	91	80	83

```
if (j > inicio)
    quicksort(vec, inicio, j);
if (i < fim)
    quicksort(vec, i, fim);
```



## 1. Classificação e Pesquisa

- Implementação em C:

```
void quicksort(int vec[], int inicio, int fim) {  
  
    int pivo = vec[ (int)(inicio+fim)/2 ];  
  
    int i = inicio, j = fim, temp;  
  
    while (i < j) {  
  
        while(pivo > vec[i]) i++;  
        while(pivo < vec[j]) j--;
```

## 1. Classificação e Pesquisa

- Implementação em C:

```
    if (i<=j) {
        temp = vec[i];
        vec[i] = vec[j];
        vec[j] = t;
        i++;
        j--;
    }
}

if (j > inicio)
    quicksort(vec, inicio, j);
if (i < fim)
    quicksort(vec, i, fim);
}
```



## 1. Classificação e Pesquisa

- *Pesquisa Seqüencial:*

- ✓ Uma vez que os elementos não estão ordenados por algum critério, o método mais simples de se pesquisar um elemento é a pesquisa seqüencial ou linear.

- ✓ Verificamos seqüencialmente, ou seja, um após o outro, se o elemento desejado se encontra no conjunto.

- Caso isso ocorra, então a pesquisa foi **bem-sucedida**.

- Caso todos os elementos do conjunto sejam verificados e o elemento desejado não esteja dentre eles, dizemos que a pesquisa foi **mal-sucedida**.

## 1. Classificação e Pesquisa

- *Pesquisa Seqüencial:*

- ✓ Podemos perceber facilmente que quanto maior o vetor, menos eficaz a pesquisa sequencial se torna.
- ✓ No pior dos casos teremos que pesquisar todos os elementos, e na média, teremos que pesquisar pelo menos a metade deles para encontrar o elemento desejado.
- ✓ Para reduzir esse número de comparações temos a ***Pesquisa Binária*** que, dado um vetor **ORDENADO**, ela consegue eliminar sempre a metade dos termos do vetor em cada comparação.

## 1. Classificação e Pesquisa

- *Pesquisa Seqüencial:*
  - ✓ A implementação desse algoritmo fica como exercício para o aluno.

## 1. Classificação e Pesquisa

- *Pesquisa Binária:*

- ✓ A pesquisa binária utiliza uma técnica computacional denominada “dividir para conquistar”:
  - O algoritmo inicialmente deve verificar o valor do elemento que está no “meio” do vetor.
  - Se este elemento (do meio) for maior que o valor procurado, deve se restringir a busca à primeira metade do vetor.
  - Caso contrário, deve-se restringir a busca à segunda metade do vetor.
  - Esse procedimento é repetido até que o elemento seja encontrado ou que não haja mais elementos a testar.

## 1. Classificação e Pesquisa

- Exemplos:

Valor de Pesquisa: 25														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
								16	18	20	22	24	26	28
												24	26	28
												24		

## 1. Classificação e Pesquisa

- Exemplos:

Valor de Pesquisa: 25														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
								16	18	20	22	24	26	28
												24	26	28
												24		

## 1. Classificação e Pesquisa

- Exemplos:

Valor de Pesquisa: 25														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
								16	18	20	22	24	26	28
												24	26	28
												24		

## 1. Classificação e Pesquisa

- Exemplos:

Valor de Pesquisa: 25														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
								16	18	20	22	24	26	28
												24	26	28
												24		



## 1. Classificação e Pesquisa

- Exemplos:

Valor de Pesquisa: 25														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
								16	18	20	22	24	26	28
												24	26	28
												24		

## 1. Classificação e Pesquisa

- Exemplos:

Valor de Pesquisa: 8														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
0	2	4	6	8	10	12								
				8	10	12								
				8										

## 1. Classificação e Pesquisa

- Exemplos:

Valor de Pesquisa: 8														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
0	2	4	6	8	10	12								
				8	10	12								
				8										

## 1. Classificação e Pesquisa

- Exemplos:

Valor de Pesquisa: 8														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
0	2	4	6	8	10	12								
				8	10	12								
				8										

## 1. Classificação e Pesquisa

- Exemplos:

Valor de Pesquisa: 8														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
0	2	4	6	8	10	12								
				8	10	12								
				8										

## 1. Classificação e Pesquisa

- Exemplos:

Valor de Pesquisa: 8														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
0	2	4	6	8	10	12								
				8	10	12								
				8										