

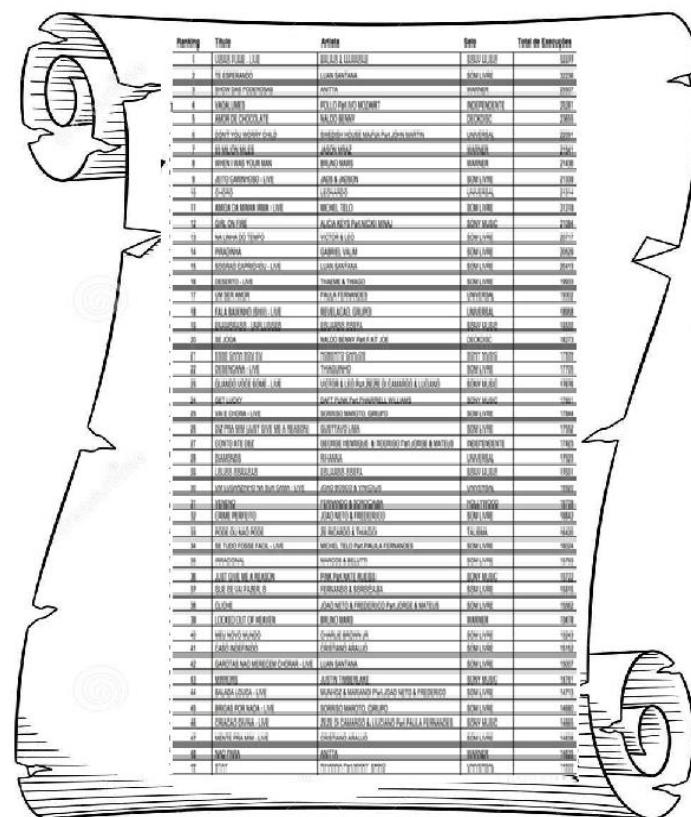
# Métodos de Ordenação de Arquivos: BubbleSort - QuickSort - InsertionSort

**Códigos de Alta Performance**

PROFa. PATRÍCIA MAGNA - [profpaticia.magna@fiap.com.br](mailto:profpaticia.magna@fiap.com.br)

# Ordenação ou Classificação de Arquivos

- Fazemos as operações de busca ou recuperação de informação nas tarefas mais comuns nos programas:
  - Localizar um cliente nos cadastros da empresa buscando por CPF;
  - Consultamos o preço de um produto pelo código de barras; etc
- Tarefa de importância no dia a dia
  - Localizar um nome na agenda
  - Achar um livro na estante
- Se a busca for em um conjunto grande de informações esta deve ser feita de forma mais rápida possível.



Ranking	Titulo	Autore	Data	Total em Circulação
1	O Senhor dos Anéis	J.R.R. Tolkien	1954	1000
2	O Hobbit	J.R.R. Tolkien	1937	1000
3	O Anel	J.R.R. Tolkien	1954	1000
4	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
5	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
6	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
7	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
8	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
9	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
10	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
11	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
12	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
13	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
14	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
15	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
16	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
17	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
18	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
19	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
20	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
21	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
22	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
23	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
24	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
25	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
26	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
27	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
28	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
29	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
30	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
31	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
32	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
33	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
34	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
35	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
36	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
37	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
38	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
39	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
40	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
41	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
42	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
43	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
44	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
45	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
46	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
47	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
48	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
49	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000
50	O Senhor dos Anéis: O Anel	J.R.R. Tolkien	1954	1000

Eficiência aumenta se este conjunto tiver algum tipo de ordenação.

Registro: cada um dos itens

RM	Nome	Prova1	Prova2	Media
4376	Adriano Santos	8,0	7,0	7,5
2915	Alessandro Teodoro	10,0	9,5	9,8
5832	Arthur De Souza	7,0	7,0	7,0
8603	Bruno Prado	10,0	8,5	9,3
3284	Carlos Antônio De Souza	10,0	8,5	9,3
5133	David Bento	10,0	8,5	9,3
1656	Diego Lopes	8,0	6,0	7,0
5894	Fábio Moreira	9,5	8,0	8,8
7859	Fernando Lima	8,0	8,0	8,0
1769	Guilherme Maia	8,0	7,0	7,5
6514	Henrique Pereira Da Costa	10,0	9,5	9,8
4776	Luiz Fernando Silva	9,5	8,0	8,8
7365	Marcelo Pereira	8,0	8,0	8,0
9151	Marco Antônio Dos Santos	8,0	7,0	7,5
8390	Mauro Marques	9,5	8,0	8,8
5487	Rodrigo Peres	10,0	9,5	9,8

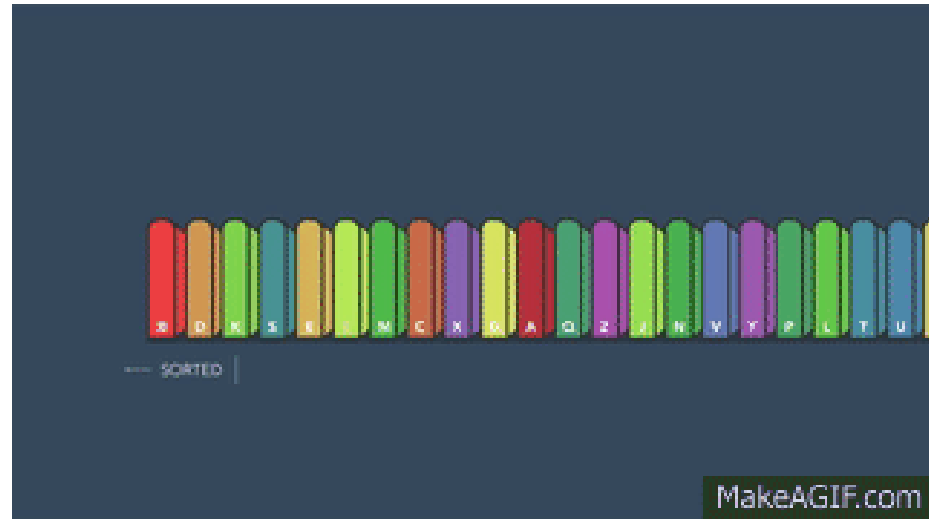
Arquivo de tamanho n

Arquivo ordenado pela chave K (Nome)

- Para  $i > j$ ,  $k[i] > k[j]$
- Chave não precisa ser pedaço do registro

- Três considerações básicas
  - Tempo de codificação
  - Tempo de computação, ou seja, de processamento
  - Espaço utilizado
- Discussões adicionais durante algoritmos

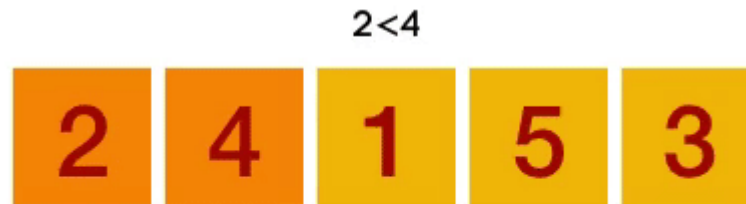
- Bubble Sort
- Quick Sort
- Insertion Sort
- Shell Sort
- Selection Sort
- Heap Sort
- Merge Sort
- Entre outros...



## Método da Bolha (*BubbleSort*)

Fazer os elementos mais “leves” borbulharem para cima e se deslocarem para sua posição final no vetor ordenado

- Um dos mais simples e fáceis de programar
- Um dos menos eficientes
- Ideia básica
  - Várias passagens pelo arquivo, trocando registros fora de lugar



25	57	48	37	12	92	86	33	Arquivo original
----	----	----	----	----	----	----	----	------------------

Comparar elemento  $i$  com o elemento  $i+1$ , trocar se necessário

## Trocas realizadas na 1ª iteração

25	57	48	37	12	92	86	33	Arquivo original
25	48	<del>57</del> 37	<del>57</del> 12	57	86	<del>92</del> 33	92	1ª iteração



# Bubble Sort


25	57	48	37	12	92	86	33	Arquivo original
----	----	----	----	----	----	----	----	------------------

25	48	37	12	57	86	33	92	1ª iteração
25	37	12	48	57	33	86	92	2ª iteração
25	12	37	48	33	57	86	92	3ª iteração
12	25	37	33	48	57	86	92	4ª iteração
12	25	33	37	48	57	86	92	5ª iteração
12	25	33	37	48	57	86	92	6ª iteração
12	25	33	37	48	57	86	92	7ª iteração
12	25	33	37	48	57	86	92	8ª iteração

- Algoritmo básico

```
Repita i, de 0 até tamanho do arquivo  
  Repita j, de 0 até (n-i-1)  
    Se  $x[j] > x[j+1]$  troca elementos de posição
```

- Mas pode ser melhorado parando quando não há mais trocas
- A melhoria exige uma variável sinalizador (*flag*)

5 4 3 1 2 | 

# Bubble Sort

25	48	37	12	57	86	33	92	1ª iteração
25	37	12	48	57	33	86	92	2ª iteração
25	12	37	48	33	57	86	92	3ª iteração
12	25	37	33	48	57	86	92	4ª iteração
12	25	33	37	48	57	86	92	5ª iteração
12	25	33	37	48	57	86	92	6ª iteração
12	25	33	37	48	57	86	92	7ª iteração
12	25	33	37	48	57	86	92	8ª iteração

Já estava ordenado e não era necessário executar essas iterações

- Algoritmo modificado

```
Repita i, de 0 até tamanho do arquivo ou não haja troca
  Repita j, de 0 até n-i-1
    Se  $x[j] > x[j+1]$  troca elementos de posição
```

- Vamos à implementação do método equivalente, tal que o protótipo seja
  - `public void bubbleSort (int vetor[], int n)`

vetor a ser  
ordenado

Quantidade de  
elementos no vetor  
a ser ordenado

- Algoritmo muito simples para ser implementado
- Para o caso do arquivo estar já ordenado ou quase ordenado a eficiência desse algoritmo é  **$O(n)$**  no **melhor caso**
- Baixo desempenho do algoritmo sem otimização:
  - Há  $n-1$  passos, e  $n-1$  comparações por passo
  - Total de comparações
    - $(n-1)*(n-1) = n^2 - 2n + 1$
  - Complexidade de ordem quadrática
    - Dobrar o tamanho  $\rightarrow$  quadruplicar tempo
- Esse método é  **$O(n^2)$**  no pior caso e no caso médio.
- Pouca necessidade de espaço adicional
  - Uma variável temporária

# Método QuickSort

Técnica de dividir para conquistar.

## Método de Ordenação Quicksort

- Esse método baseia-se na técnica de dividir para conquistar.
- Deve-se escolher um elemento, chamado de pivô, em uma posição específica dentro do vetor, por exemplo, o último elemento.
- O pivô é colocado em sua posição correta no vetor ordenado (posição  $j$ ) e o vetor é dividido em 2 subvetores:
  - os elementos das posições 0 à  $j-1$  são menores do que pivô.
  - os elementos das posições  $j+1$  a  $n-1$  sejam maiores do que o pivô.
- Repete-se recursivamente o método até cada subvetor ter apenas 1 elemento.

Unsorted Array



# Passo a Passo do Quicksort

`quickSort(vetor, 0, 5)`

0	1	2	3	4	5
28	36	12	72	20	54

pivô

posição final do pivô = 4

0	1	2	3
28	36	12	20

4
54

5
72



# Passo a Passo do Quicksort

quickSort(vetor, 0, 3)

0	1	2	3
28	36	12	20

pivô

4
54

5
72

li = ls

0	1
12	20

li = ls

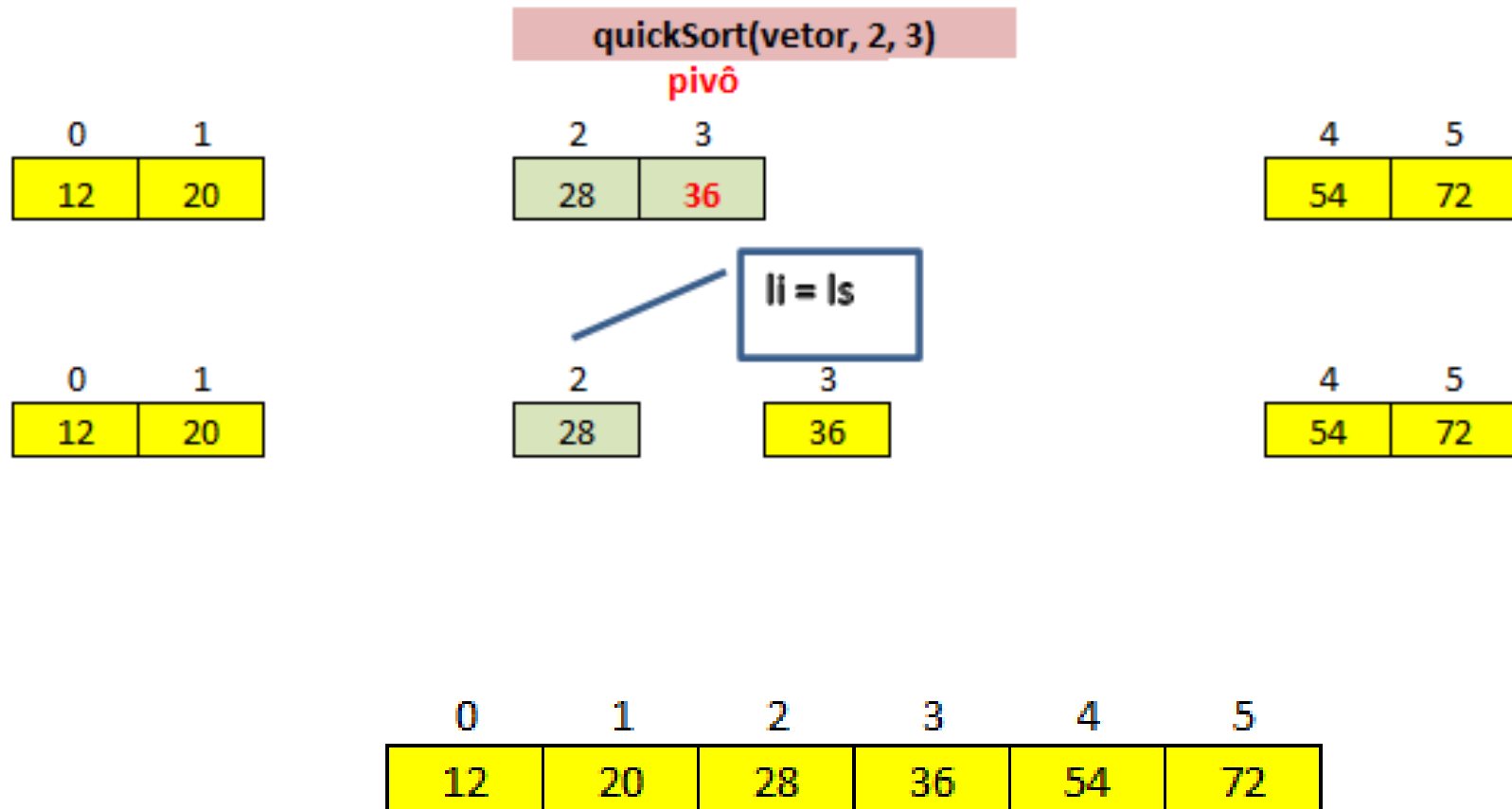
2	3
28	36

4	5
54	72

ls (limite superior)

li (limite inferior)

# Passo a Passo do Quicksort



```
public static void quickSort(int vetor[],int li,int ls)
{
    int j;
    if (li<ls){
        j = particiona(vetor, li, ls);
        quickSort(vetor, li, j-1);
        quickSort(vetor, j+1,ls);
    }
}
```

ls (limite superior)

li (limite inferior)

## Implementação do Método particiona()

```
public static int particiona (int vetor[], int li, int ls){
    int pivo,abaixo,temp,acima;
    pivo=vetor[ls];
    acima=ls;
    abaixo=li;
    while(abaixo<acima) {
        while(vetor[abaixo]<pivo && abaixo<ls) {
            abaixo++;
        }
        while (vetor[acima]>=pivo && acima > abaixo) {
            acima--;
        }
        if (abaixo<acima){
            temp= vetor[abaixo];
            vetor[abaixo]= vetor[acima];
            vetor[acima]=temp;
        }
    }
    vetor[ls]= vetor[acima];
    vetor[acima]=pivo;
    return acima;
}
```

# Implementação do Método particiona()

```
public static int particiona (int vetor[], int li, int ls){
    int pivo, abaixo, temp, acima;
```

```
quickSort(vetor, 0, 5)
```

					pivô
0	1	2	3	4	5
28	36	12	72	20	54

posição final do pivô = 4

0	1	2	3		4		5
28	36	12	20		54		72

```
    if (abaixo < acima){
        temp= vetor[abaixo];
        vetor[abaixo]= vetor[acima];
        vetor[acima]=temp;
    }
    vetor[ls]= vetor[acima];
    vetor[acima]=pivo;
    return acima;
}
```

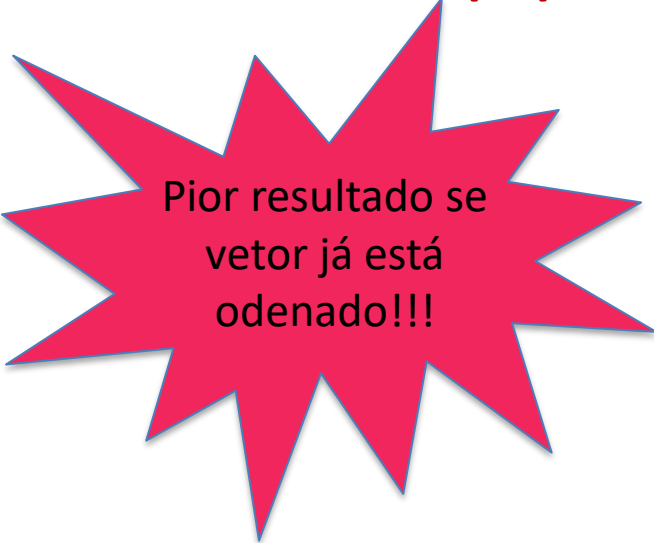
## Considerações sobre Eficiência

- Em um arquivo com  $n$  registros, supondo que o pivô fique sempre no meio, tem-se em relação ao número de comparações:
  - $n + 2*(n/2) + 4*(n/4) + 8*(n/8) + \dots + n*(n/n) =$   
 $= n + n + n + n + n \dots (m \text{ vezes}) \rightarrow m * n$
  - Se  $n = 2^m \Rightarrow m = \log_2 n$
- Assim,  $m * n = \log n * n \Rightarrow O(n \log n)$ 
  - Esta é uma função de eficiência considerada relativamente eficiente.

JAVA: `Arrays.sort()`

The sorting algorithm is a Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm offers  $O(n \log(n))$  performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.

- Quando o vetor já está ordenado e o pivô é vetor[ls].
- Nestas condições verifica-se que o pivô está na posição correta e o sub-vetor a ser ordenado é 1 a n-1 elementos.
- Assim, deverão ser ordenados (n-1) sub-vetores, sendo que o 1º com n, o 2º com n-1 e assim por diante.
- $n + (n-1) + (n-2) + \dots + 2 \Rightarrow n*n + \text{constante} \Rightarrow O(n^2)$



Pior resultado se  
vetor já está  
ordenado!!!

## Método por Inserção (*InsertionSort*)

Ordenação por inserção coloca cada elemento em seu lugar correto diretamente.



25	57	48	37
----	----	----	----

Vamos partir do seguinte vetor não ordenado

25	57	48	37
----	----	----	----

- Inserindo o primeiro elemento

25
----

- Inserindo o segundo elemento ( $25 < 57$ )

25	57
----	----

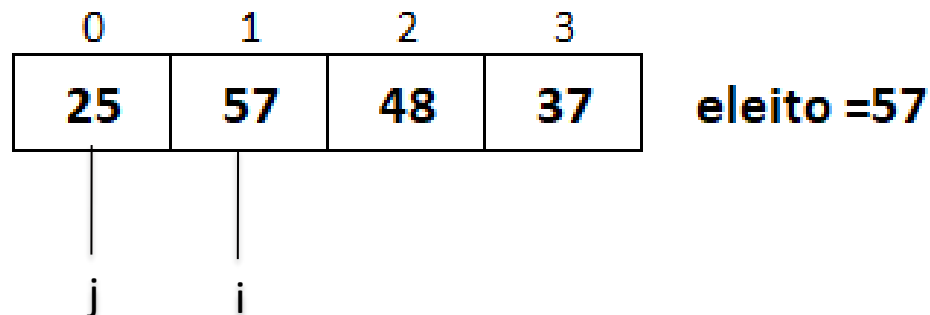
- Inserindo o terceiro elemento ( $48 < 57$  e  $48 > 25$ )

25	48	57
----	----	----

- Inserindo o quarto elemento ( $37 < 57$  e  $37 < 48$  e  $37 > 25$ )

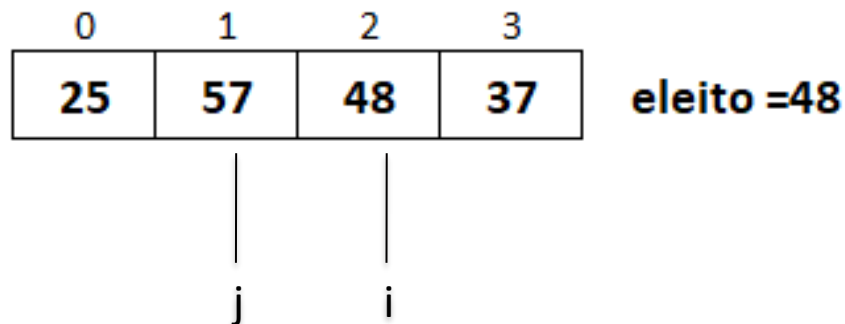
25	37	48	57
----	----	----	----

```
int eleito, j, i;
for(i = 1; i < 4; i++) {
    eleito = vetor[i]; //eleito vetor[1] = 57
    j = i-1; //j=0
    while (j >= 0 && vetor[j] > eleito) {
        // j =0 (true)    25 > 57 (false)
        vetor[j+1] = vetor[j];
        j = j-1;
    }
    vetor[j+1] = eleito; //vetor[1] = 57
}
```

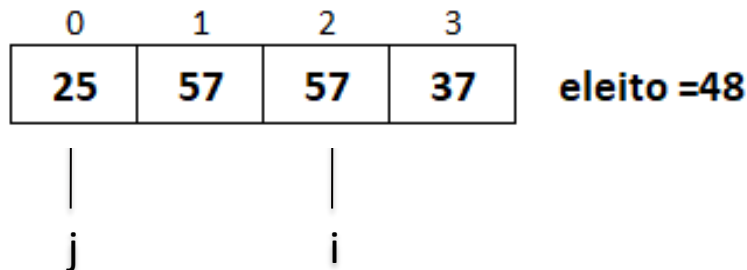


## Ordenação Insertionsort:

```
int eleito, j, i;
for(i = 1; i < 4; i++) { //i=2
    eleito = vetor[i]; //eleito vetor[2] = 48
    j = i-1; //j=1
    while (j >= 0 && vetor[j] > eleito) {
        // j=1 (true)      57 > 48 (true)
        vetor[j+1] = vetor[j]; //vetor[2] = 57
        j = j-1; //j=0
    }
    vetor[j+1] = eleito;
}
```

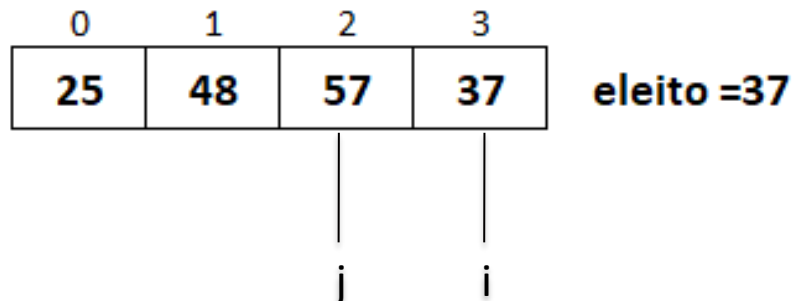


```
int eleito, j, i;
for(i = 1; i <4; i++) { //i=2
    eleito = vetor[i]; //eleito vetor[2] = 48
    j = i-1; //j=1
    while (j >= 0 && vetor[j] > eleito) {
        // j=0 (true)      25 > 48 (false)
        vetor[j+1] = vetor[j];
        j = j-1;
    }
    vetor[j+1] = eleito; //vetor[1] = 48
}
```



# Ordenação Insertionsort:

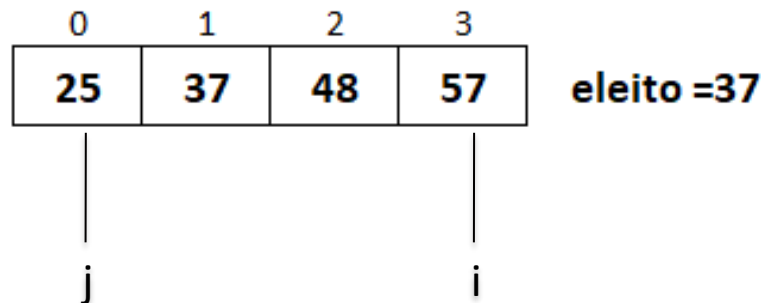
```
int eleito, j, i;
for(i = 1; i <4; i++) { //i=3
    eleito = vetor[i]; //eleito vetor[3] = 37
    j = i-1; //j=2
    while (j >= 0 && vetor[j] > eleito) {
        //j=2 (true)          57 > 37 (true)
        vetor[j+1] = vetor[j]; //vetor[3] = 57
        j = j-1; //j=1
    }
    vetor[j+1] = eleito;
}
```



```
int eleito, j, i;
for(i = 1; i <4; i++) { //i=3
    eleito = vetor[i]; //eleito vetor[2] = 37
    j = i-1; //j=2
    while (j >= 0 && vetor[j] > eleito) {
        // j=1 (true)      48 > 37 (true)
        vetor[j+1] = vetor[j]; //vetor[2] = 48
        j = j-1; // j=0
    }
    vetor[j+1] = eleito;
}
```

0	1	2	3	
25	48	57	57	eleito =37
	j		i	

```
int eleito, j, i;
for(i = 1; i <4; i++) { //i=3
    eleito = vetor[i]; //eleito vetor[2] = 48
    j = i-1; //j=2
    while (j >= 0 && vetor[j] > eleito) {
        // j=0 (true)      25 > 37 (false)
        vetor[j+1] = vetor[j];
        j = j-1;
    }
    vetor[j+1] = eleito; // vetor[1] = 37
}
```





## Projeto OrdenaInsertionSort

- Algoritmo simples para ser implementado
- O número de comparações em um arquivo com  $n$  elementos deve ser:
  - $1+2+3+ \dots +(n-1) = (n-1) * n/2$
  - Portanto, sua função de eficiência é  **$O(n^2)$**
- O número de trocas segue o mesmo número de comparações.
- Apesar de ser  $O(n^2)$  no caso de inversamente ordenado esse método tem, como ponto positivo, a fato de exigir 1 espaço adicional de memória adicional (variável **eleito**)

<https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao/>

<https://www.youtube.com/watch?v=vmT3XUBoxiQ>

bubblesort	=	(0:19)	242.920	comparações
quicksort	=	(2:49)	20.220	comparações
insertion	=	(0:19)	465.540	comparações

<https://www.youtube.com/watch?v=ZZuD6iUe3Pc>

1. Crie um projeto capaz de executar os 3 métodos estudados. Devem ser criados 3 vetores com 2048 elementos inteiros:
  - ordenado
  - inversamente ordenado
  - Números aleatórios (deve ser o mesmo para ser aplicado nos 3 métodos)

Execute os 3 métodos de ordenação com 3 vetores e preencha a tabela:

Método	Tipo do Vetor	Comparações	Trocas
Bubblesort	ordenado		
	inversamente		
	aleatório		
Quicksort	ordenado		
	inversamente		
	aleatório		
Insertionsort	ordenado		
	inversamente		
	aleatório		

2. Qual método você escolhe como sendo o mais eficiente?

# REFERÊNCIAS



- ASCÊNCIO, A.F.G; ARAUJO, G.S. **Estruturas de Dados: Algoritmos, Análise de Complexidade e Implementações em JAVA e C/C++**. São Paulo, Ed.Pearson Prentice Hall, 2010.
- PEREIRA, S.L.; **Estruturas de Dados Fundamentais: Conceitos e Aplicações**. São Paulo, Ed. Érica, 1996.
- TENEMBAUM, A.M et al.; **Estruturas de Dados usando C**. Makron Books Ltda, 1995.
- ZIVIANI, Nivio. Projeto de algoritmos com implementações em Pascal e C. São Paulo: Pioneira, 2000.

Copyright © 2021  
Profa. Patrícia Magna

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, dos professores.