

Unidade I:

Introdução - Algoritmo de Ordenação por Seleção



Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

Agenda

- Introdução sobre Ordenação Interna
- Funcionamento básico
- Algoritmo em C *like*
- Análise dos número de movimentações e comparações
- Estrutura dos nossos códigos em Java e C
- Conclusão

Agenda

- **Introdução sobre Ordenação Interna** 
- Funcionamento básico
- Algoritmo em C *like*
- Análise dos número de movimentações e comparações
- Estrutura dos nossos códigos em Java e C
- Conclusão

Introdução sobre Ordenação Interna

- Muitas aplicações requerem dados de forma ordenada
- Entrada: *array* com n elementos
- A ordenação é dita Interna quando a lista de elementos cabe na memória principal, caso contrário, é dita Externa
- Chave de Pesquisa: Atributo utilizado para ordenar os registros

Análise dos Algoritmos de Ordenação Interna

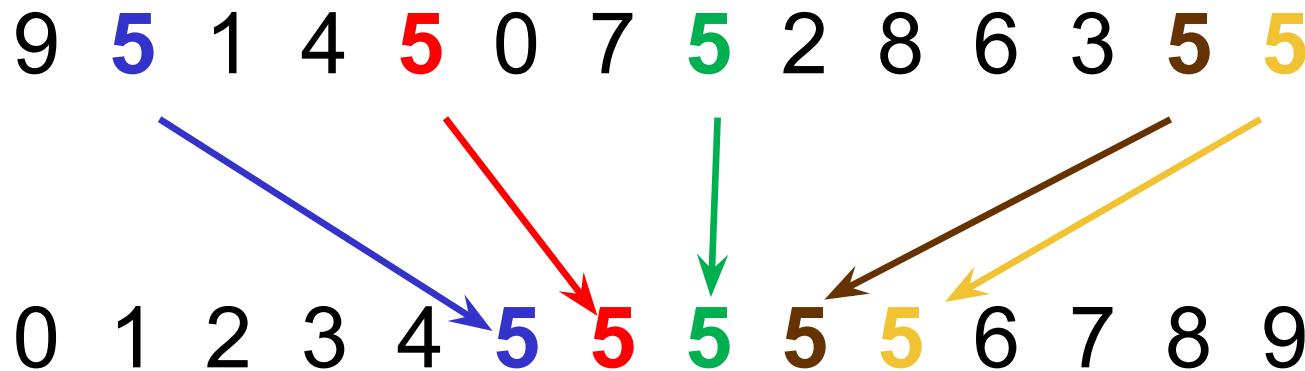
- Operações fundamentais: comparação e movimentação entre elementos do *array*
- O limite inferior em termos do número de comparações (entre elementos do *array*) para a ordenação interna é $\Theta(n \times \lg(n))$
- Logo, a complexidade ótima para a ordenação interna em número de comparações do pior e do caso médio é $\Theta(n \times \lg(n))$
- Vários algoritmos de ordenação interna alcançam esse limite

Algoritmos Estáveis vs. Não Estáveis

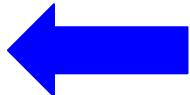
- Um algoritmo é dito estável se depois da execução, os elementos com a mesma chave mantiverem a ordem relativa entre as chaves repetidas

- No exemplo abaixo, a ordem dos elementos azul, vermelho, verde e marrom e amarelo é a mesma

- Antes:



Agenda

- Introdução sobre Ordenação Interna
- **Funcionamento básico** 
- Algoritmo em C like
- Análise dos número de movimentações e comparações
- Estrutura dos nossos códigos em Java e C
- Conclusão

Funcionamento Básico

- Procure o menor elemento do array
- Troque a posição do menor elemento com o primeiro
- Volte ao primeiro passo e considere o array a partir da próxima posição

Exemplo

Legenda: - menor elemento em vermelho
- parte ordenada está de azul

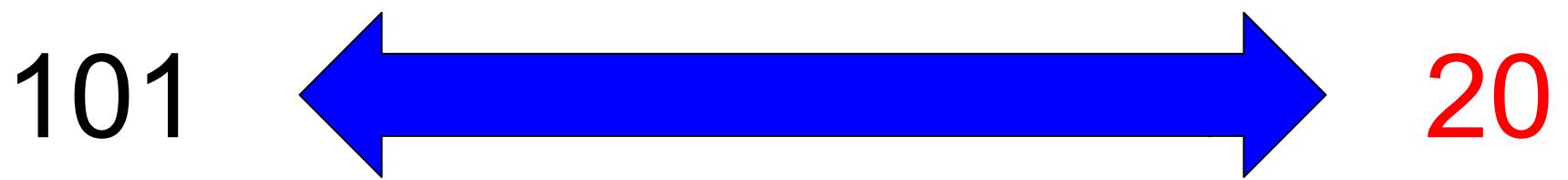
101 115 30 63 47 20

Exemplo

101 115 30 63 47 20

Menor
elemento

Exemplo



Trocando a posição do menor elemento com o primeiro

Exemplo

Parte
ordenada

20

115

30

63

47

101

Parte a ser ordenada

Exemplo

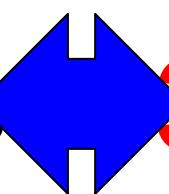
20 115 30 63 47 101

Exemplo

20 115 30 63 47 101

Menor
elemento

Exemplo

20 115  30 63 47 101

Exemplo

20 30

115	63	47	101
-----	----	----	-----

Trocando a posição do menor elemento com o primeiro

Exemplo

20 30 115 63 47 101

Menor
elemento

Exemplo

20 30 115  47 101

Exemplo

20 30 47 63 115 101

Trocando a posição do menor elemento com o primeiro

Exemplo

20 30 47 63 115 101

Menor
elemento

Exemplo

20 30 47 63

115 101

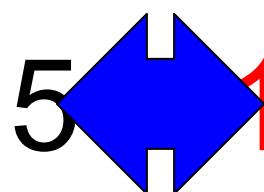
Trocando a posição do menor elemento com o primeiro

Exemplo

20 30 47 63 115 101

Menor
elemento

Exemplo

20 30 47 63 115  101

Exemplo

20 30 47 63 101 115

Trocando a posição do menor elemento com o primeiro

Exemplo

20 30 47 63 101 115

O algoritmo terminou? Por que?

Agenda

- Introdução sobre Ordenação Interna
- Funcionamento básico
- **Algoritmo em C *like*** 
- Análise dos número de movimentações e comparações
- Estrutura dos nossos códigos em Java e C
- Conclusão

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

1

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j])  
            menor = j;  
    }  
    swap(menor, i);  
}
```

(Obs.1): O maior valor de i é $(n-2)$, pois repetimos enquanto i for menor que $(n-1)$

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

2

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

(Obs.2): No final, o elemento da posição (n-1) será o maior, pois os (n-1) menores elementos já foram separados

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

3

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

(Obs.3): i endereça a posição do elemento a ser inserido no conjunto ordenado

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    4 → for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

(Obs.4): O laço interno procura a posição do menor elemento no conjunto a ser ordenado

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor; 5  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

(Obs.5): j começa na primeira posição a ser comparada com a posição menor

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```



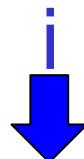
(Obs.6): O *swap* troca o conteúdo das posições menor e i

```
public void swap(int a, int b) {  
    int temp = array[a];  
    array[a] = array[b];  
    array[b] = temp;  
}
```

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

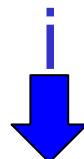


101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $0 < 5$



101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor
i

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor
↓
i ↓ j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $1 < 6$

menor
i j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $101 > 115$

menor
↓
i ↓ j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor
↓
i

j
↓

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $2 < 6$

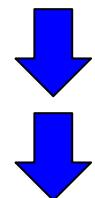
menor
i
j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

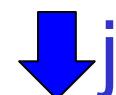
true: $101 > 30$



menor



i



j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor
↓
i
↓
j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor

↓ i

↓ j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $3 < 6$

menor

↓ i

↓ j

↓ j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $30 > 63$

menor

↓ i

↓ j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor

↓ i

↓

↓ j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $4 < 6$

menor

↓ i

↓

↓ j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $30 > 47$

menor

↓ i

↓

↓ j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor

↓ i

↓

↓ j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $5 < 6$

menor

↓ i

↓

↓ j

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $30 > 20$

menor

↓ i

↓

↓ j

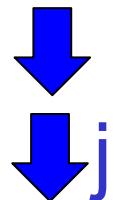
101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $30 > 20$

menor



i

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor
↓
j
↓
i

101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++) {  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $6 < 6$

menor

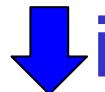
j



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

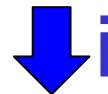
menor

 i

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

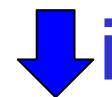
```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++) {  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```



20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $1 < 5$ 

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor
↓ i

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor
↓
i ↓
j

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $2 < 6$

menor
i j

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $115 > 30$

menor
↓
i ↓
j

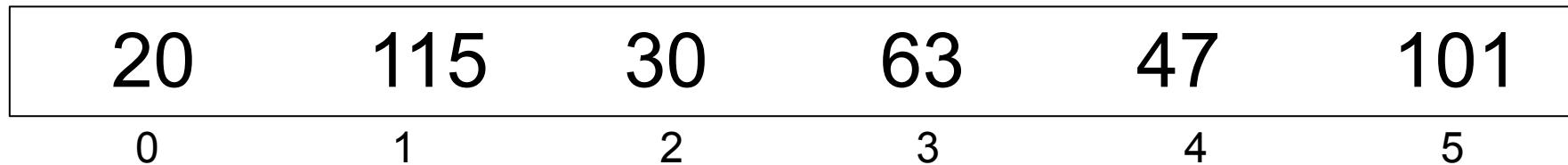
20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $115 > 30$

menor
↓
i j



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor

↓ i

↓ j

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $3 < 6$  menor i j

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $30 > 63$

↓ menor

↓ i

↓ j

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor

↓ i

↓ j

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $4 < 6$

↓ menor

↓ i

↓ j

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $30 > 47$

↓menor

↓i

↓j

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor

↓ i

↓ j

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $5 < 6$

↓ menor

↓ i

↓ j

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: 30 > 101

↓ menor

↓ i

↓ j

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor

↓ i

j
↓

20	115	30	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $6 < 6$

↓ menor

↓ i

j
↓

20	115	30	63	47	101
0	1	2	3	4	5

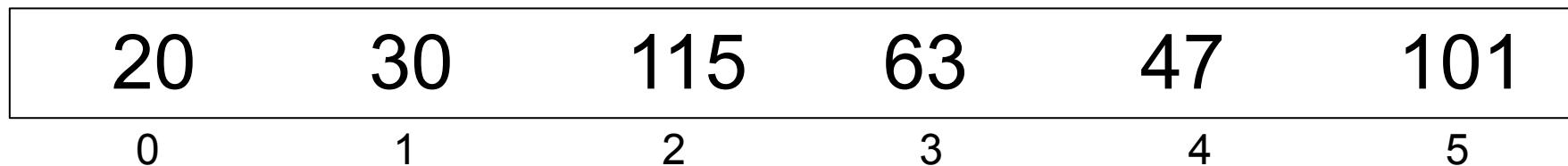
Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor

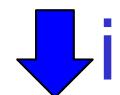
↓ i

j
↓



Algoritmo em C *like*

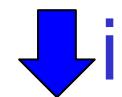
```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++) {  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```



20	30	115	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

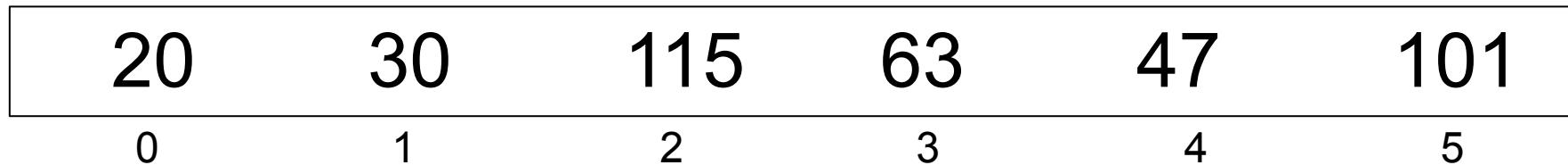
true: $2 < 5$ 

20	30	115	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor
i



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor
↓
i ↓
j

20	30	115	63	47	101
0	1	2	3	4	5

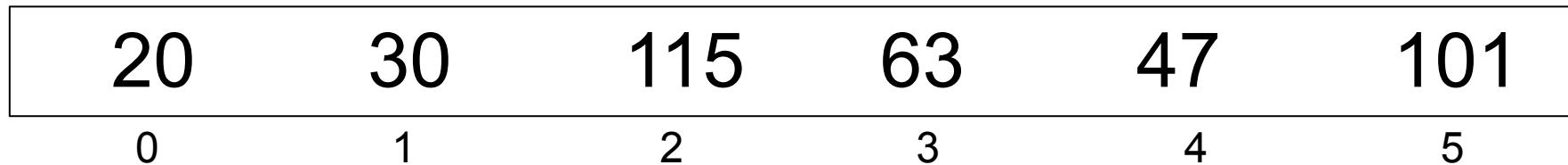
Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $3 < 6$ 

menor

i j

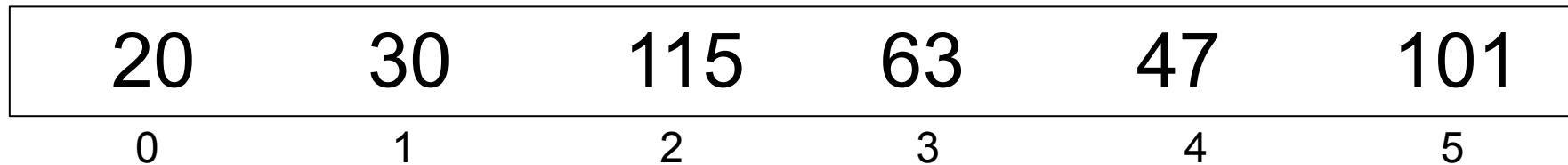


Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $115 > 63$

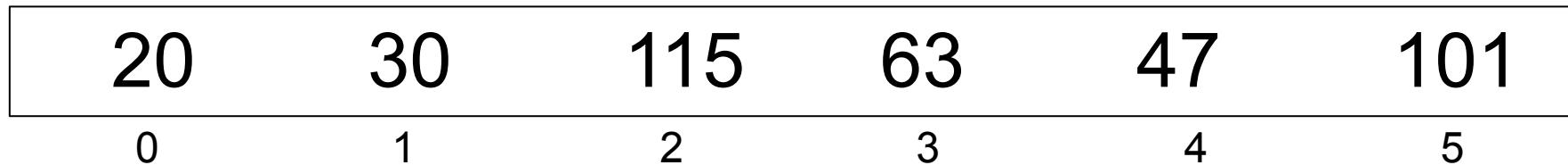
menor
i j



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor
↓
i
↓
j



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor

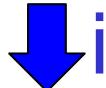
↓ i

↓ j

20	30	115	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

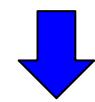
```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $4 < 6$  menor i j

20	30	115	63	47	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $63 > 47$  menor i j

20	30	115	63	47	101
0	1	2	3	4	5

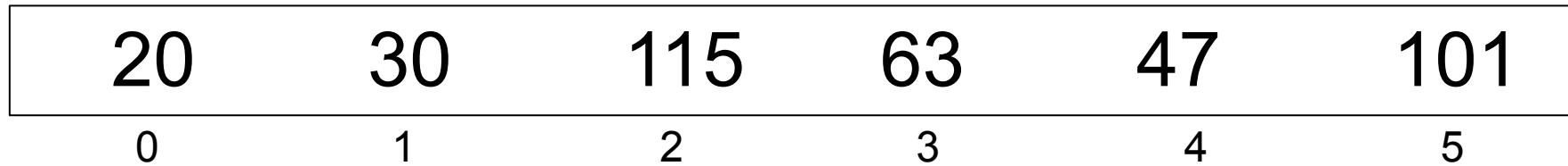
Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $63 > 47$

menor
j

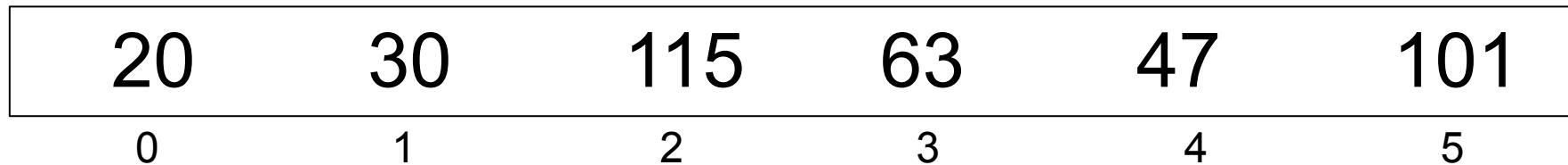
i



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor
↓
j
↓

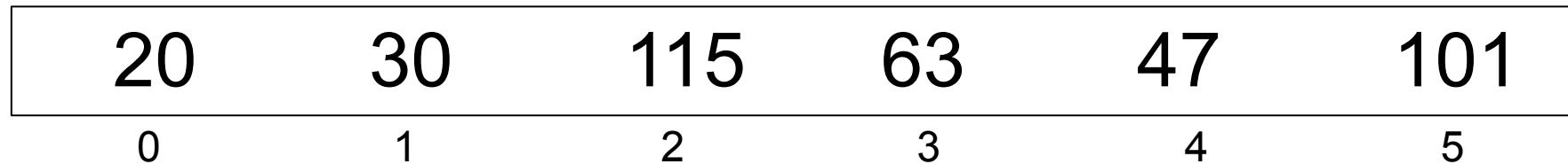


Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $5 < 6$

↓ menor
↓ i
↓ j

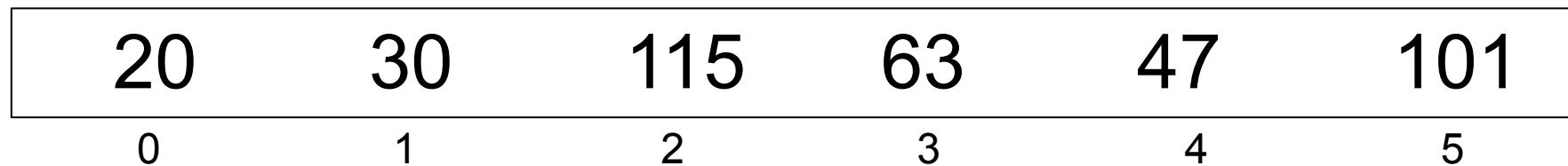


Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

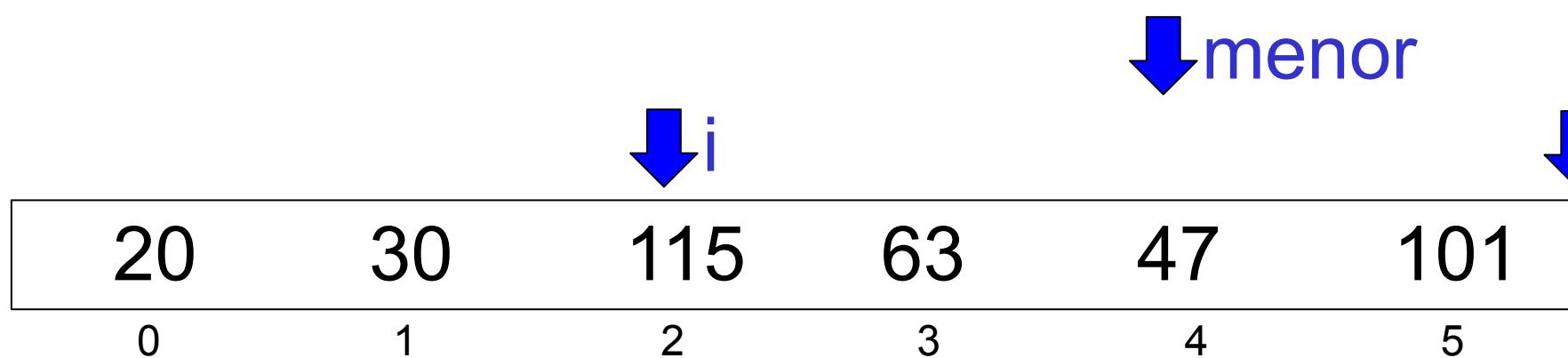
false: 47 > 101

menor
↓
j
↓



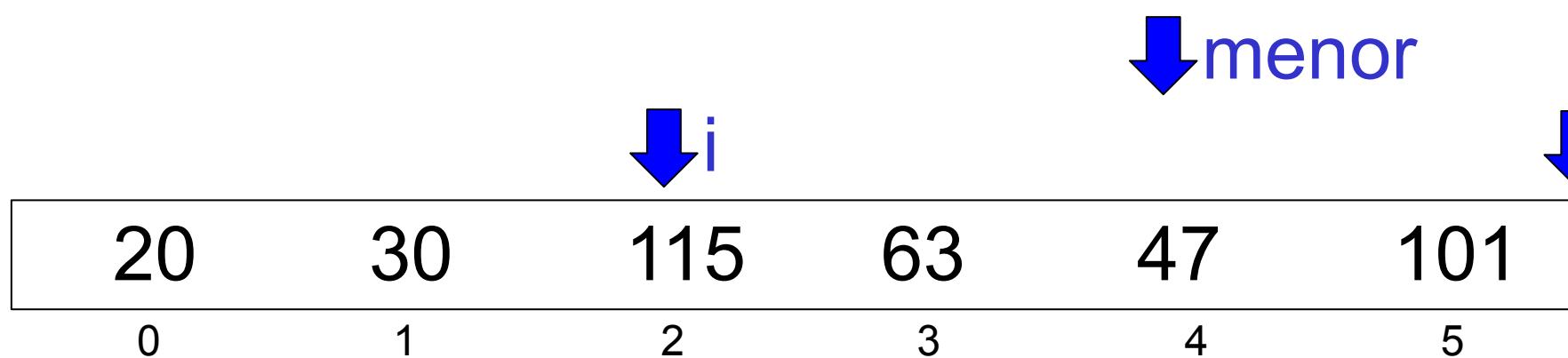
Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $6 < 6$ 

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

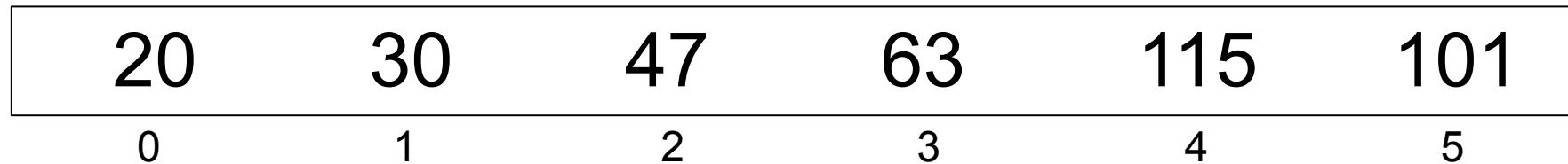
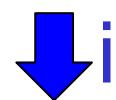
↓ menor

↓ i

20	30	47	63	115	101
0	1	2	3	4	5

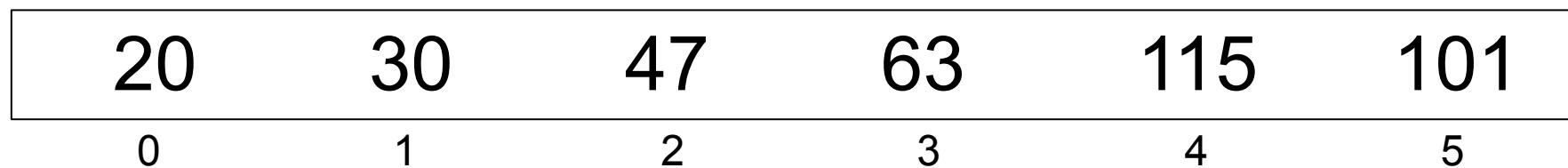
Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++) {  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $3 < 5$ 

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

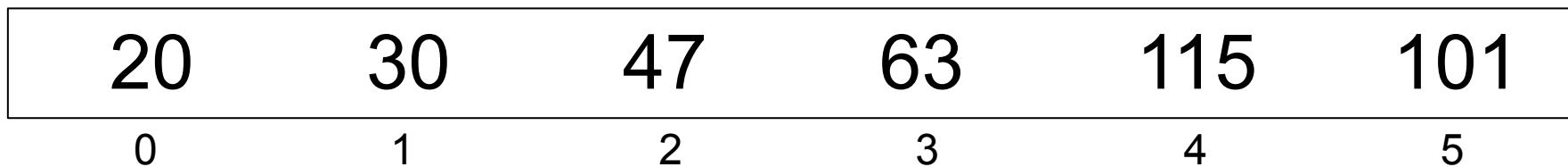
↓ menor
↓ i

20	30	47	63	115	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor
↓ i ↓ j

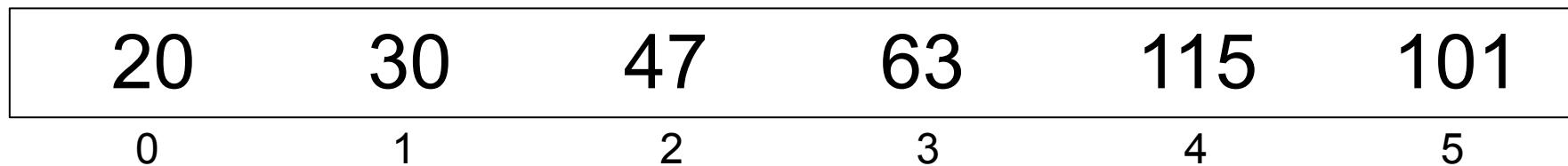


Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $4 < 6$ 

menor
i j



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: 63 > 115

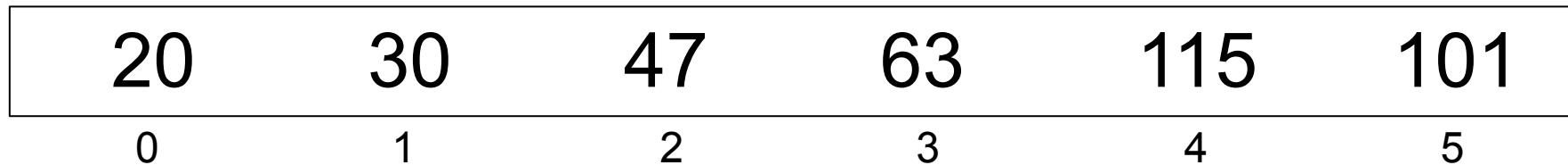
↓ menor
↓ i ↓ j

20	30	47	63	115	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor
↓ i
↓ j

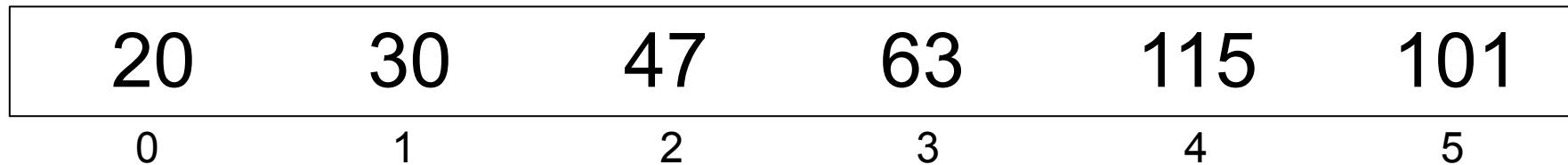


Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $5 < 6$

↓ menor
↓ i
↓ j



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: 63 > 101

↓ menor
↓ i
↓ j

20	30	47	63	115	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```



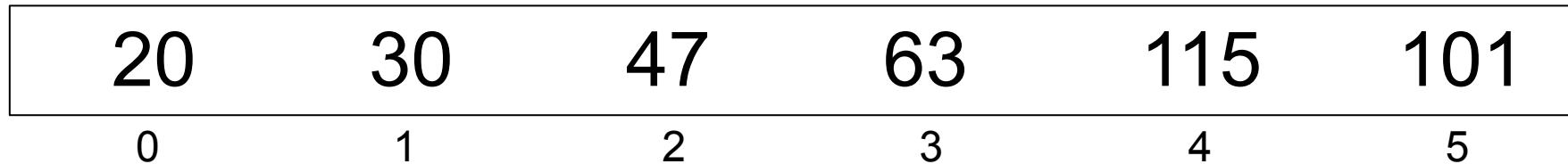
Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $6 < 6$

↓ menor
↓ i

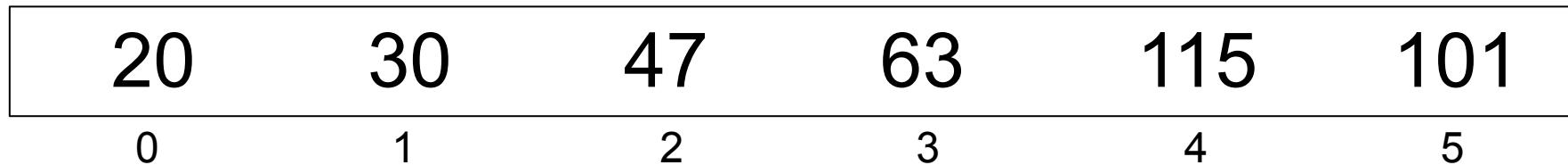
↓ j



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor
↓ i



Algoritmo em C *like*

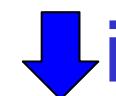
```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++) {  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ i

20	30	47	63	115	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

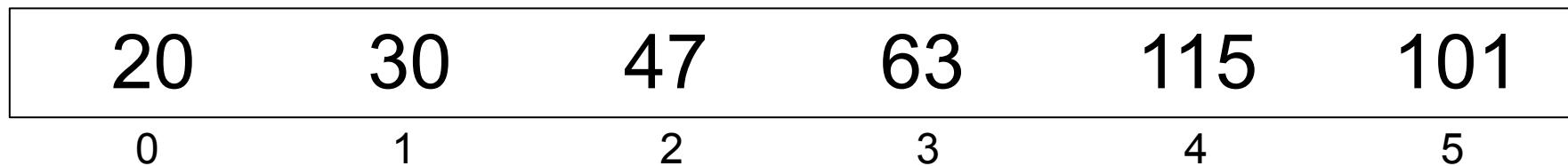
true: $4 < 5$  i

20	30	47	63	115	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor
↓ i



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ menor
↓ i ↓ j

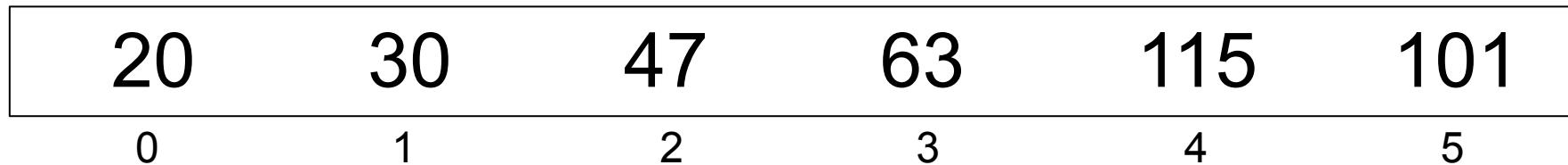
20	30	47	63	115	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $5 < 6$

↓ menor
↓ i ↓ j

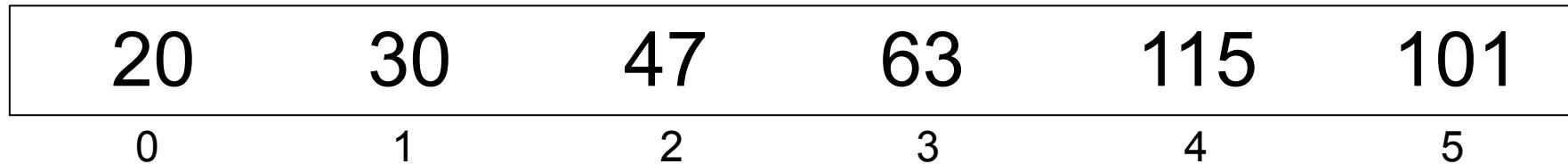


Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

true: $115 > 101$

↓ menor
↓ i ↓ j



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor

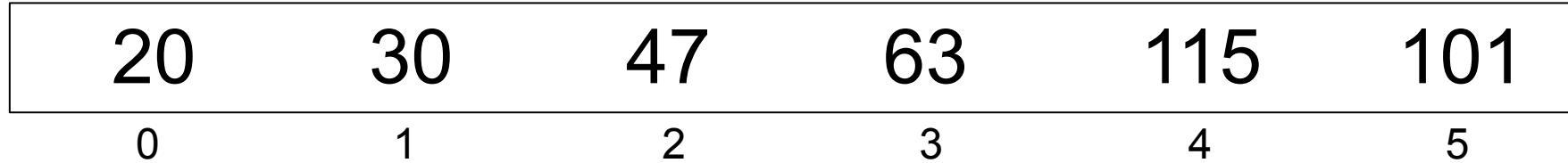
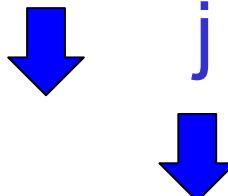


20	30	47	63	115	101
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor

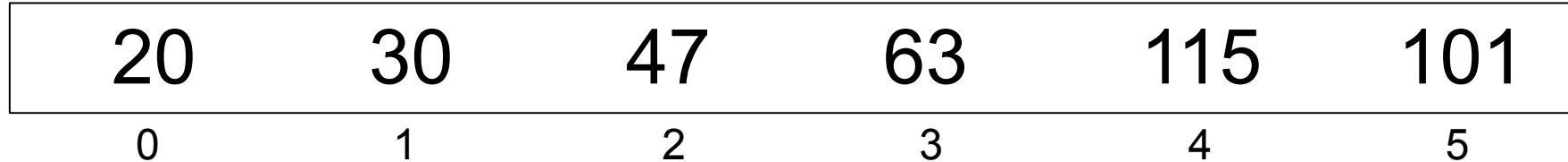
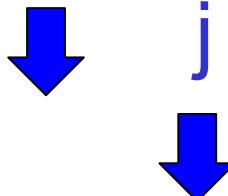


Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $6 < 6$

menor



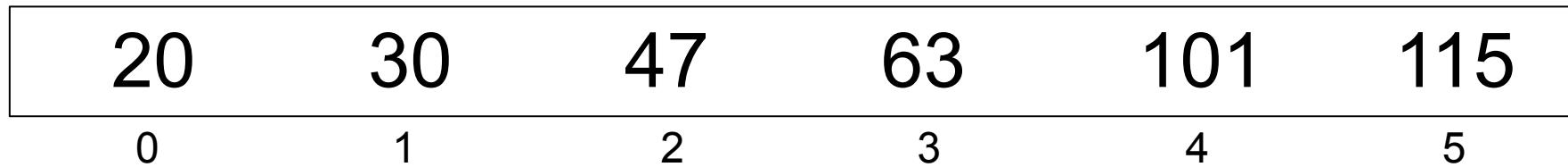
Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

menor



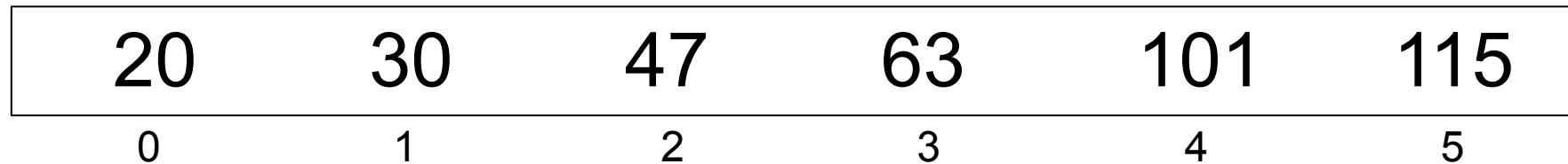
↓ i



Algoritmo em C *like*

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++) {  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ i

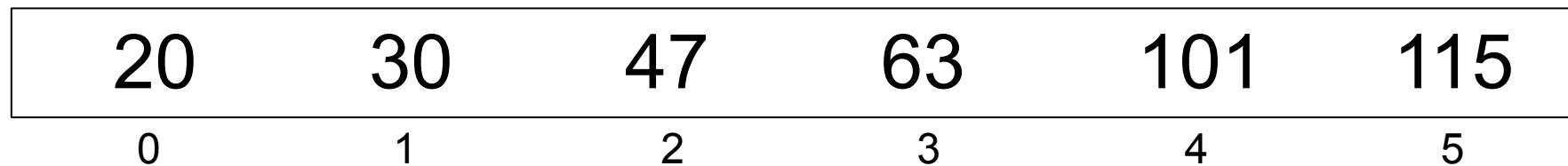


Algoritmo em C *like*

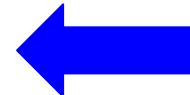
```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $5 < 5$

↓ i



Agenda

- Introdução sobre Ordenação Interna
- Funcionamento básico
- Algoritmo em C *like*
- **Análise dos número de movimentações e comparações** 
- Estrutura dos nossos códigos em Java e C
- Conclusão

Análise do Número de Movimentações

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Quantas
movimentações
(entre elementos
do *array*) são
realizadas?

Análise do Número de Movimentações

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++) {  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Quantas movimentações (entre elementos do *array*) são realizadas?

Análise do Número de Movimentações

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++) {  
        if (array[menor] > array[j])  
            menor = j;  
    }  
    swap(menor, i);  
}
```

O laço externo realiza $(n - 1)$ trocas, ou seja, $3(n - 1)$ movimentações

Quantas movimentações (entre elementos do array) são realizadas?

$$M(n) = 3(n - 1)$$

Exercício Resolvido (1)

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Faça com que
nossa código
conte o número de
movimentações?

Exercício Resolvido (1)

```
int mov = 0;
for (int i = 0; i < (n - 1); i++) {
    int menor = i;
    for (int j = (i + 1); j < n; j++){
        if (array[menor] > array[j]){
            menor = j;
        }
    }
    swap(menor, i);
    mov += 3;
}
System.out.println(mov);
```

Faça com que
nossa código
conte o número de
movimentações?

Análise do Número de Comparações

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Quantas
comparações
(entre elementos
do *array*) são
realizadas?

Análise do Número de Comparações

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Temos somente um comando de comparação

Quantas comparações (entre elementos do *array*) são realizadas?

Análise do Número de Comparações

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Executamos o laço interno
 $(n - (i + 1))$ vezes

Ou seja, $(n - i - 1)$ vezes

Análise do Número de Comparações

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Executamos o laço interno
 $(n - (i + 1))$ vezes

Ou seja, $(n - i - 1)$ vezes

Exemplo: $n = 5$

Para $i = 0$, os valores de j serão 1, 2, 3 e 4 $(5 - 0 - 1) = 4$ vezes

Para $i = 1$, os valores de j serão 2, 3 e 4 $(5 - 1 - 1) = 3$ vezes

Para $i = 2$, os valores de j serão 3 e 4 $(5 - 2 - 1) = 2$ vezes

Para $i = 3$, o valor de j será 4 $(5 - 3 - 1) = 1$ vez

Análise do Número de Comparações

```

for (int i = 0; i < (n - 1); i++) {
    int menor = i;
    for (int j = (i + 1); j < n; j++){
        if (array[menor] > array[j]){
            menor = j;
        }
    }
    swap(menor, i);
}

```

Executamos o laço interno
($n - (i + 1)$) vezes

Ou seja, $(n - i - 1)$ vezes

i	0	1	2	3	...	n-2
$c(i) = (n - (i+1))$	$n-1$	$n-2$	$n-3$	$n-4$...	1

$$\sum_{i=0}^{n-2} (n - i - 1)$$

Análise do Número de Comparações

- Como o laço interno é executado $(n - i - 1)$ vezes e o externo $(n - 1)$ vezes, logo:

$$C(n) = \frac{n^2}{2} - \frac{n}{2} = \Theta(n^2)$$

Análise do Número de Comparações

- Como o laço interno é executado $(n - i - 1)$ vezes e o externo $(n - 1)$ vezes, logo:

$$C(n) =$$

Resolução de somatórios



Unidade II

Análise do Número de Comparações

- Sendo,

$$C(n) = (n - 1) + (n - 2) + (n - 3) + \dots + 1$$

- Podemos colocar da forma abaixo?

$$C(n) = \sum_{i=0}^{i < n-1} (n - i - 1)$$

- E assim?

$$C(n) = \sum_{i=0}^{n-2} (n - i - 1)$$

Análise do Número de Comparações

- Na Unidade de Somatórios, vamos aprender que:

$$C(n) = \sum_{i=0}^{n-2} (n - i - 1) = \sum_{i=0}^{n-2} (n) - \sum_{i=0}^{n-2} (i) - \sum_{i=0}^{n-2} (1)$$

Análise do Número de Comparações

- Agora, podemos fazer as duas substituições abaixo, certo?

$$C(n) = \sum_{i=0}^{n-2} (n - i - 1) = \sum_{i=0}^{n-2} (n) - \sum_{i=0}^{n-2} (i) - \sum_{i=0}^{n-2} (1)$$

$n * (n-1)$ $- (n-1)$

Análise do Número de Comparações

- Agora, podemos fazer as duas substituições abaixo, certo?

$$C(n) = \sum_{i=0}^{n-2} (n - i - 1) = \sum_{i=0}^{n-2} (n) - \sum_{i=0}^{n-2} (i) - \sum_{i=0}^{n-2} (1)$$

$n * (n-1)$ - $(n-1)$

- Logo:

$$C(n) = (n-1)(n) - (n-1) - \sum_{i=0}^{n-2} (i)$$

Análise do Número de Comparações

- Deturpando o somatório, podemos fazer:

$$C(n) = (n - 1)(n) - (n - 1) - \sum_{i=0}^{n-2} (i)$$

$$C(n) = (n - 1)(n) - (n - 1) - \sum_{i=1}^{n-1} (i - 1)$$

Análise do Número de Comparações

- Separando o “i” e “-1” em dois somatórios, temos:

$$C(n) = (n - 1)(n) - (n - 1) - \sum_{i=1}^{n-1} (i - 1)$$

$$C(n) = (n - 1)(n) - (n - 1) - \sum_{i=1}^{n-1} (i) + \sum_{i=1}^{n-1} (1)$$

Análise do Número de Comparações

- Resolvendo o segundo somatório, temos:

$$C(n) = (n - 1)(n) - (n - 1) - \sum_{i=1}^{n-1} (i) + \sum_{i=1}^{n-1} (1)$$

$$C(n) = (n - 1)(n) - (n - 1) - \sum_{i=1}^{n-1} (i) + (n - 1)$$

Análise do Número de Comparações

- Simplificando $-(n-1) + (n-1)$, temos:

$$C(n) = (n - 1)(n) - (n - 1) - \sum_{i=1}^{n-1} (i) + (n - 1)$$

$$C(n) = (n - 1)(n) - \sum_{i=1}^{n-1} (i)$$

Análise do Número de Comparações

- Na unidade sobre Somatórios, vamos aprender:

$$\sum_{i=1}^{n-1} (i) = 1 + 2 + \dots + (n-1) = \frac{(n-1)(n)}{2}$$

- Assim:

$$C(n) = (n-1)(n) - \sum_{i=1}^{n-1} (i) = (n-1)(n) - \frac{(n-1)(n)}{2}$$

Análise do Número de Comparações

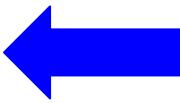
- Simplificando, temos:

$$C(n) = (n-1)n - \frac{(n-1)n}{2} = \frac{(n-1)n}{2} = \frac{n^2}{2} - \frac{n}{2}$$

- Finalmente:

$$C(n) = \frac{n^2}{2} - \frac{n}{2} = \Theta(n^2)$$

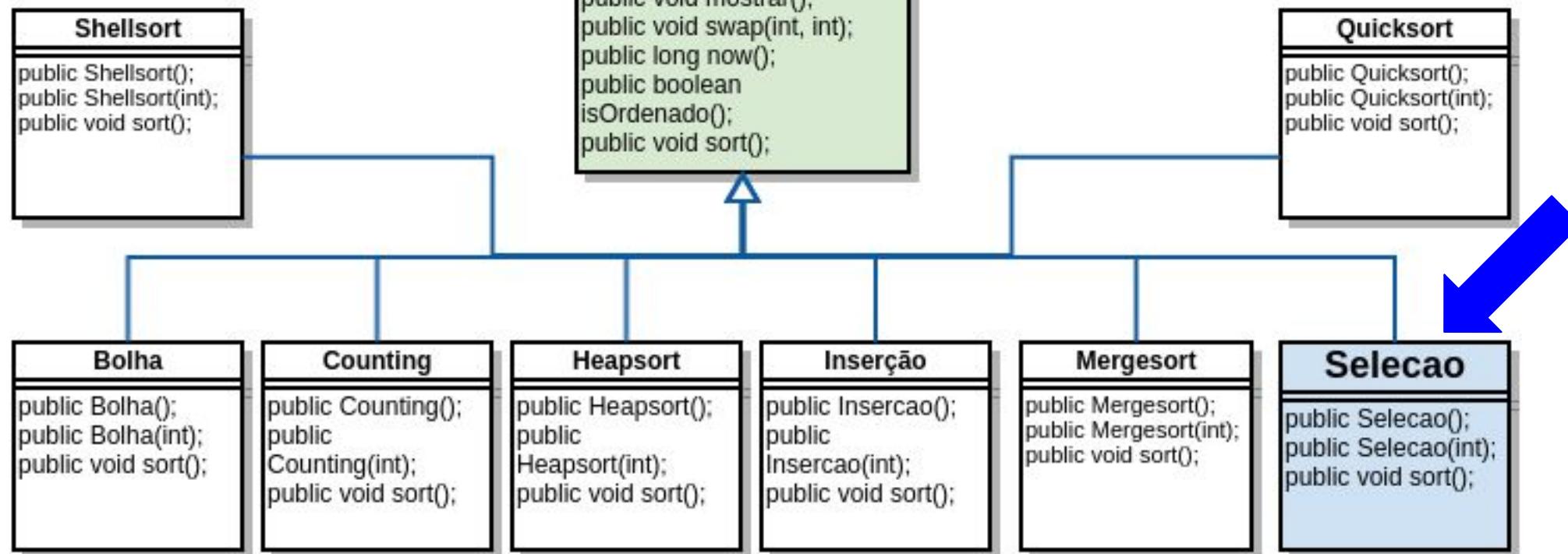
Agenda

- Introdução sobre Ordenação Interna
- Funcionamento básico
- Algoritmo em C *like*
 - Estrutura do Código em Java
 - Estrutura do Código em C
 - makefile para C/C++
- Análise dos número de movimentações e comparações
- **Estrutura dos nossos códigos em Java e C** 
- Conclusão

Estrutura do Código em Java

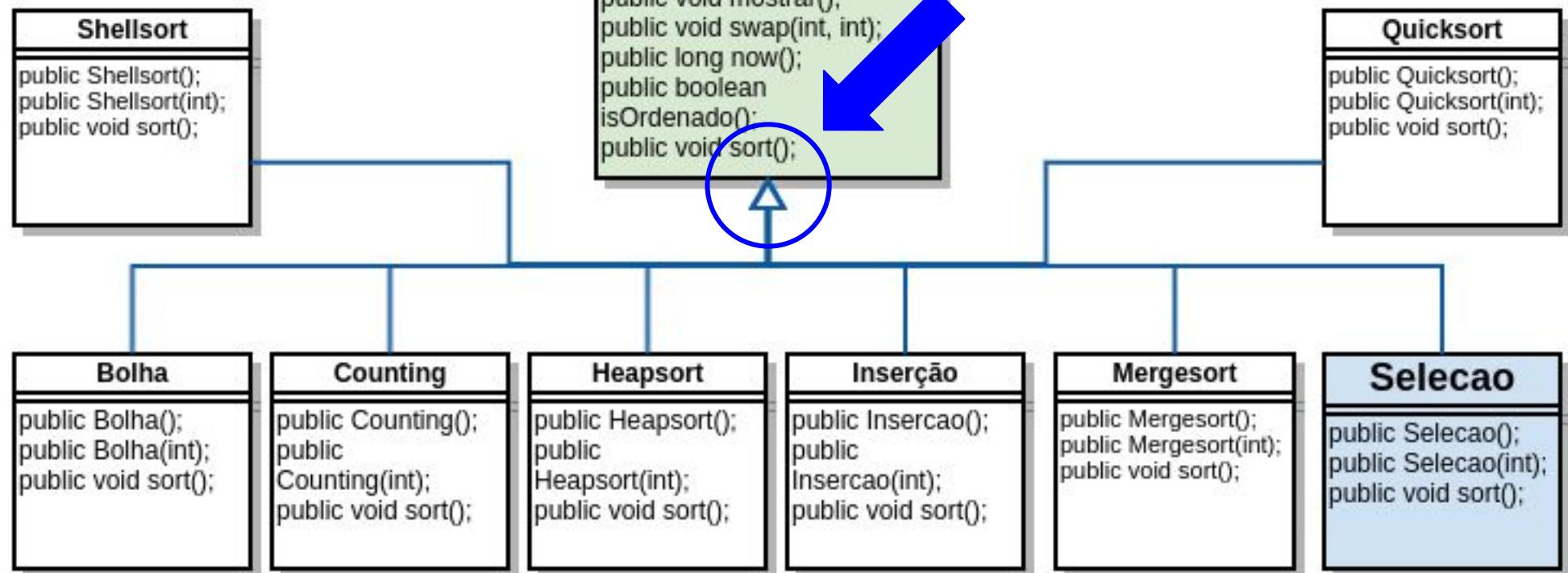


Classes importantes para esta Unidade



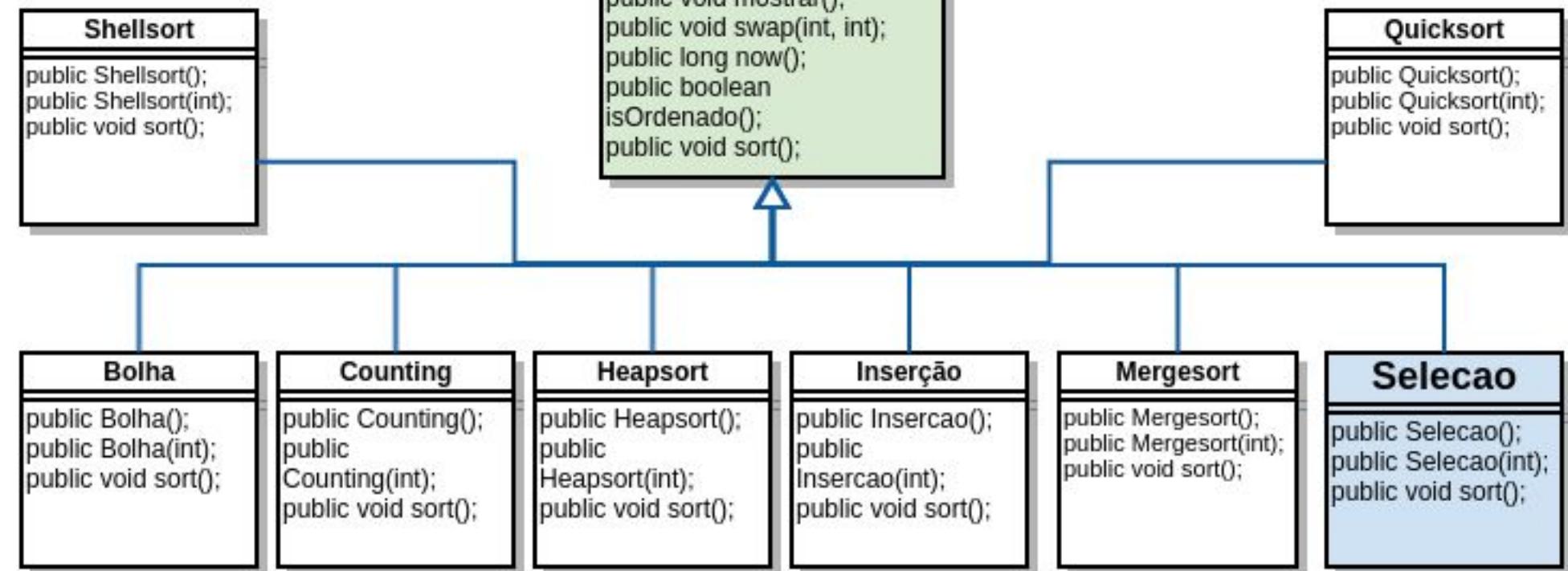
Estrutura do Código em Java

Símbolo de herança



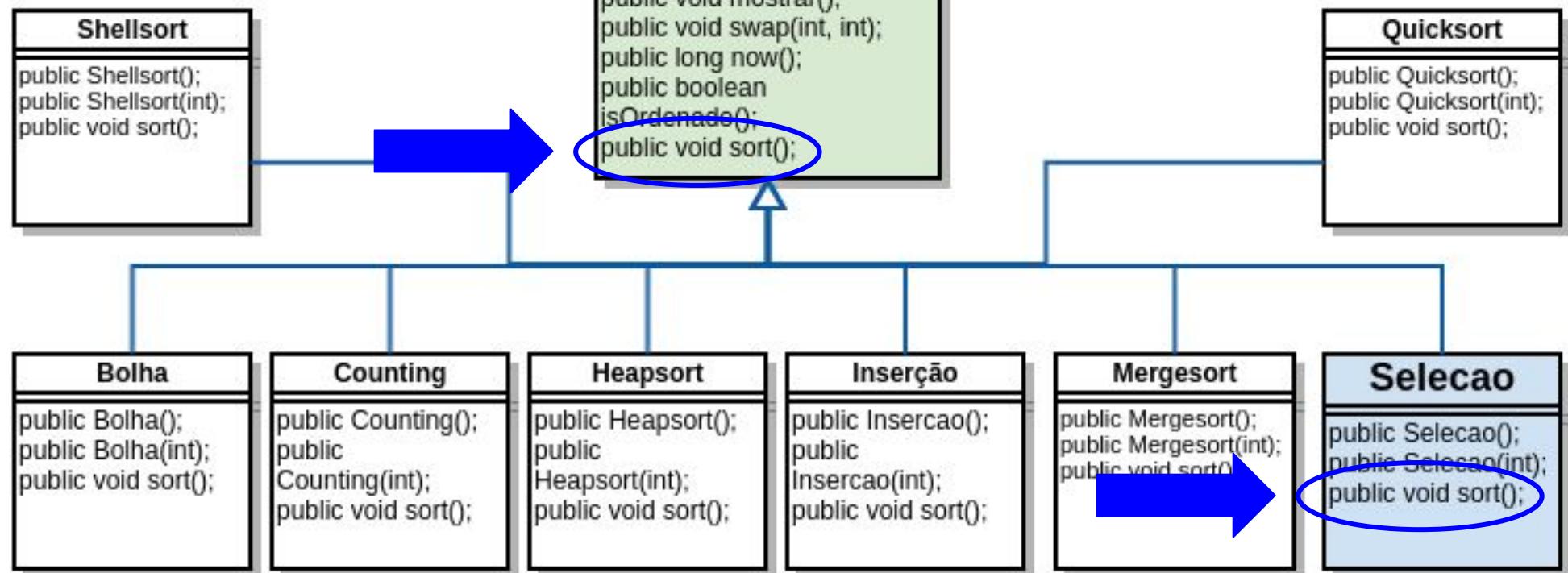
Estrutura do Código em Java

Atributos com visibilidade protegida



Estrutura do Código em Java

Método @Override nas subclasses



Classe Geração

Geração
protected int[] array; protected int n; public Geracao(); public Geracao(int); public void crescente(); public void decrescente(); public void aleatorio(); public void entradaPadrao(); public void entrada(int[]); public void mostrar(int); public void mostrar(); public void swap(int, int); public long now(); public boolean isOrdenado(); public void sort();



Classe Geração

```

class Geracao {
    protected int[] array;
    protected int n;

    public Geracao(){
        array = new int[100];
        n = array.length;
    }

    public Geracao(int tamanho){
        array = new int[tamanho];
        n = array.length;
    }

    public void crescente() {
        for (int i = 0; i < n; i++) array[i] = i;
    }

    public void decrescente() {
        for (int i = 0; i < n; i++) array[i] = n - 1 - i;
    }
}

```

```

public void aleatorio() {
    Random rand = new Random();
    crescente();
    for (int i = 0; i < n; i++)
        swap(i, Math.abs(rand.nextInt()) % n);
}

public void entradaPadrao() {
    n = MyIO.readInt();
    array = new int[n];
    for (int i = 0; i < n; i++)
        array[i] = MyIO.readInt();
}

public void entrada(int[] vet){
    n = vet.length;
    array = new int[n];
    for (int i = 0; i < n; i++) array[i] = vet[i];
}

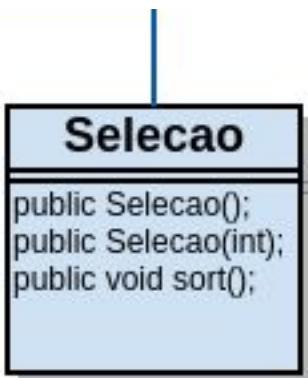
```

Classe Geração

```
public void mostrar() {  
    System.out.print("[");  
    for (int i = 0; i < n; i++)  
        System.out.print(" (" + i + ")" + array[i]);  
    System.out.println("]");  
}  
  
public void swap(int i, int j) {  
    int temp = array[i];  
    array[i] = array[j];  
    array[j] = temp;  
}  
  
public long now(){  
    return new Date().getTime();  
}
```

```
public boolean isOrdenado(){  
    boolean resp = true;  
    for (int i = 1; i < n; i++) {  
        if (array[i] < array[i-1]){  
            i = n;  
            resp = false;  
        }  
    }  
    return resp;  
}  
  
public void sort(){  
    System.out.println("Método a ser  
        implementado nas subclasses.");  
}
```

Classe Seleção



Classe Seleção

```
class Selecao extends Geracao {  
    public Selecao(){  
        super();  
    }  
    public Selecao(int tamanho){  
        super(tamanho);  
    }  
    @Override  
    public void sort() {  
        for (int i = 0; i < (n - 1); i++) {  
            int menor = i;  
            for (int j = (i + 1); j < n; j++){  
                if (array[menor] > array[j]){  
                    menor = j;  
                }  
            }  
            swap(menor, i);  
        }  
    }  
}
```

super() é o construtor da superclasse (pai)

@Override é facultativo, contudo, boa prática

Classe Principal

Principal

```
public static void main (String[])
```

Classe Principal

```
class Principal {  
    public static void main(String[] args){  
        Geracao algoritmo;  
        int n = (args.length < 1) ? 1000 :  
            Integer.parseInt(args[0]);  
        double inicio, fim;  
  
        //Inicialização do algoritmo de ordenação  
        //algoritmo = new Bolha(n);  
        //algoritmo = new Countingsort(n);  
        //algoritmo = new Heapsort(n);  
        //algoritmo = new Insercao(n);  
        //algoritmo = new Mergesort(n);  
        //algoritmo = new Quicksort(n);  
        algoritmo = new Selecao(n);  
        //algoritmo = new Shellsort(n);  
  
        //Geração do conjunto a ser ordenado  
        algoritmo.aleatorio();  
        //algoritmo.crescente();
```

```
        //algoritmo.decrescente();  
  
        //Mostrar o conjunto a ser ordenado  
        //algoritmo.mostrar();  
  
        //Execução do algoritmo de ordenação  
        inicio = algoritmo.now();  
        algoritmo.sort();  
        fim = algoritmo.now();  
  
        //Mostrar o conjunto ordenado, tempo de  
        //execução e status da ordenação  
        //algoritmo.mostrar();  
        System.out.println("Tempo para ordenar:  
" + (fim-inicio)/1000.0 + " s.");  
        System.out.println("isOrdenado: " +  
        algoritmo.isOrdenado());  
    }  
}
```

Classe Principal

```

class Principal {
    public static void main(String[] args){
        Geracao algoritmo;
        1   int n = (args.length < 1) ? 1000 :
            Integer.parseInt(args[0]);
        double inicio, fim;

        //Inicialização do algoritmo de ordenação
        //algoritmo = new Bolha(n);
        //algoritmo = new Countingsort(n);
        2   //algoritmo = new Heapsort(n);
        //algoritmo = new Insercao(n);
        //algoritmo = new Mergesort(n);
        //algoritmo = new Quicksort(n);
        algoritmo = new Selecao(n);
        //algoritmo = new Shellsort(n);

        //Geração do conjunto a ser ordenado
        algoritmo.aleatorio();
        3   //algoritmo.crescente();
    }
}

```

```

//algoritmo.decrescente();

//Mostrar o conjunto a ser ordenado
4   //algoritmo.mostrar();

//Execução do algoritmo de ordenação
inicio = algoritmo.now();
algoritmo.sort();
fim = algoritmo.now();
5

//Mostrar o conjunto ordenado, tempo de
execução e status da ordenação
//algoritmo.mostrar();
System.out.println("Tempo para ordenar:
" + (fim-inicio)/1000.0 + " s.");
System.out.println("isOrdenado: " +
algoritmo.isOrdenado());
}
}

```

Classe Principal

```

class Principal {
    public static void main(String[] args){
        Geracao algoritmo;
        1 int n = (args.length < 1) ? 1000 :
            Integer.parseInt(args[0]);
        ...
    }
}

Operador ternário e uso
do array de argumentos

//algoritmo = new Boina(n);
//algoritmo = new Countingsort(n);
//algoritmo = new Heapsort(n);
//algoritmo = new Insercao(n);
//algoritmo = new Mergesort(n);
//algoritmo = new Quicksort(n);
algoritmo = new Selecao(n);
//algoritmo = new Shellsort(n);

//Geração do conjunto a ser ordenado
algoritmo.aleatorio();
//algoritmo.crescente();

```

```

//algoritmo.decrescente();

//Mostrar o conjunto a ser ordenado
//algoritmo.mostrar();

//Execução do algoritmo de ordenação
inicio = algoritmo.now();
algoritmo.sort();
fim = algoritmo.now();

//Mostrar o conjunto ordenado, tempo de
execução e status da ordenação
//algoritmo.mostrar();
System.out.println("Tempo para ordenar:
" + (fim-inicio)/1000.0 + " s.");
System.out.println("isOrdenado: " +
algoritmo.isOrdenado());
}
}

```

Classe Principal

```

class Principal {
    public static void main(String[] args){
        Geracao algoritmo;
        int n = (args.length < 1) ? 1000 :
            Integer.parseInt(args[0]);
        double inicio, fim;

        //Inicialização do algoritmo de ordenação
        //algoritmo = new Bolha(n);
        //algoritmo = new Countingsort(n);
        2 //algoritmo = new Heapsort(n);
        //algoritmo = new Insercao(n);
        //algoritmo = new Mergesort(n);
        //algoritmo = new Quicksort(n);
        algoritmo = new Selecao(n);
        //algoritmo = new Shellsort(n);
    }
}

```

Mudando comentários,
trocamos de algoritmo

```

//algoritmo.decrecente();
//Mostrar o conjunto a ser ordenado
//algoritmo.mostrar();

//Execução do algoritmo de ordenação
inicio = algoritmo.now();
algoritmo.sort();
fim = algoritmo.now();

//Mostrar o conjunto ordenado, tempo de
execução e status da ordenação
//algoritmo.mostrar();
System.out.println("Tempo para ordenar:
" + (fim-inicio)/1000.0 + " s.");
System.out.println("isOrdenado: " +
algoritmo.isOrdenado());
}
}

```

Classe Principal

```

class Principal {
    public static void main(String[] args){
        Geracao algoritmo;
        int n = (args.length < 1) ? 1000 :
            Integer.parseInt(args[0]);
        double inicio, fim;

        //Inicialização do algoritmo de ordenação
        //algoritmo = new Bolha(n);
        //algoritmo = new Countingsort(n);
        //algoritmo = new Heapsort(n);
        //algoritmo = new Insercao(n);
        //algoritmo = new Mergesort(n);
        //algoritmo = new Quicksort(n);
        algoritmo = new Selecao(n);

        Mudando comentários,
        trocamos a ordem inicial
        nado
    }

    3   algoritmo.aleatorio();
        //algoritmo.crescente();
    }
}

```

```

        //algoritmo.decrescente();

        //Mostrar o conjunto a ser ordenado
        //algoritmo.mostrar();

        //Execução do algoritmo de ordenação
        inicio = algoritmo.now();
        algoritmo.sort();
        fim = algoritmo.now();

        //Mostrar o conjunto ordenado, tempo de
        //execução e status da ordenação
        //algoritmo.mostrar();
        System.out.println("Tempo para ordenar:
" + (fim-inicio)/1000.0 + " s.");
        System.out.println("isOrdenado: " +
algoritmo.isOrdenado());
    }
}

```

Classe Principal

```

class Principal {
    public static void main(String[] args){
        Geracao algoritmo;
        int n = (args.length < 1) ? 1000 :
            Integer.parseInt(args[0]);
        double inicio, fim;

        //Inicialização do algoritmo de ordenação
        //algoritmo = new Bolha(n);
        //algoritmo = new Countingsort(n);
        //algoritmo = new Heapsort(n);
        //algoritmo = new Insercao(n);
        //algoritmo = new Mergesort(n);
        //algoritmo = new Quicksort(n);
        algoritmo = new Selecao(n);
        //algoritmo = new Shellsort(n);

        //Geração do conjunto a ser ordenado
        algoritmo.aleatorio();
        //algoritmo.crescente();
    }
}

```

4 //algoritmo.decrecente();

//Mostrar o conjunto a ser ordenado
//algoritmo.mostrar();

**Mudando comentários,
mostramos o array**

fim = algoritmo.now();

//Mostrar o conjunto ordenado, tempo de
execução e status da ordenação
//algoritmo.mostrar();
System.out.println("Tempo para ordenar:
" + (fim-inicio)/1000.0 + " s.");
System.out.println("isOrdenado: " +
algoritmo.isOrdenado());
}

}

Classe Principal

```

class Principal {
    public static void main(String[] args){
        Geracao algoritmo;
        int n = (args.length < 1) ? 1000 :
            Integer.parseInt(args[0]);
        double inicio, fim;
        //Inicialização
        //algoritmo = new Boina(n);
        //algoritmo = new Countingsort(n);
        //algoritmo = new Heapsort(n);
        //algoritmo = new Insercao(n);
        //algoritmo = new Mergesort(n);
        //algoritmo = new Quicksort(n);
        algoritmo = new Selecao(n);
        //algoritmo = new Shellsort(n);

        //Geração do conjunto a ser ordenado
        algoritmo.aleatorio();
        //algoritmo.crescente();
    }
}

```

Mostrar tempo de execução e se ordenado

```

//algoritmo.decrescente();

//Mostrar o conjunto a ser ordenado
//algoritmo.mostrar();

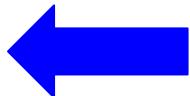
//Execução do algoritmo de ordenação
inicio = algoritmo.now();
algoritmo.sort();
fim = algoritmo.now();

//Mostrar o conjunto ordenado, tempo de execução e status da ordenação
//algoritmo.mostrar();
System.out.println("Tempo para ordenar:
" + (fim-inicio)/1000.0 + " s.");
System.out.println("isOrdenado: " +
algoritmo.isOrdenado());
}
}

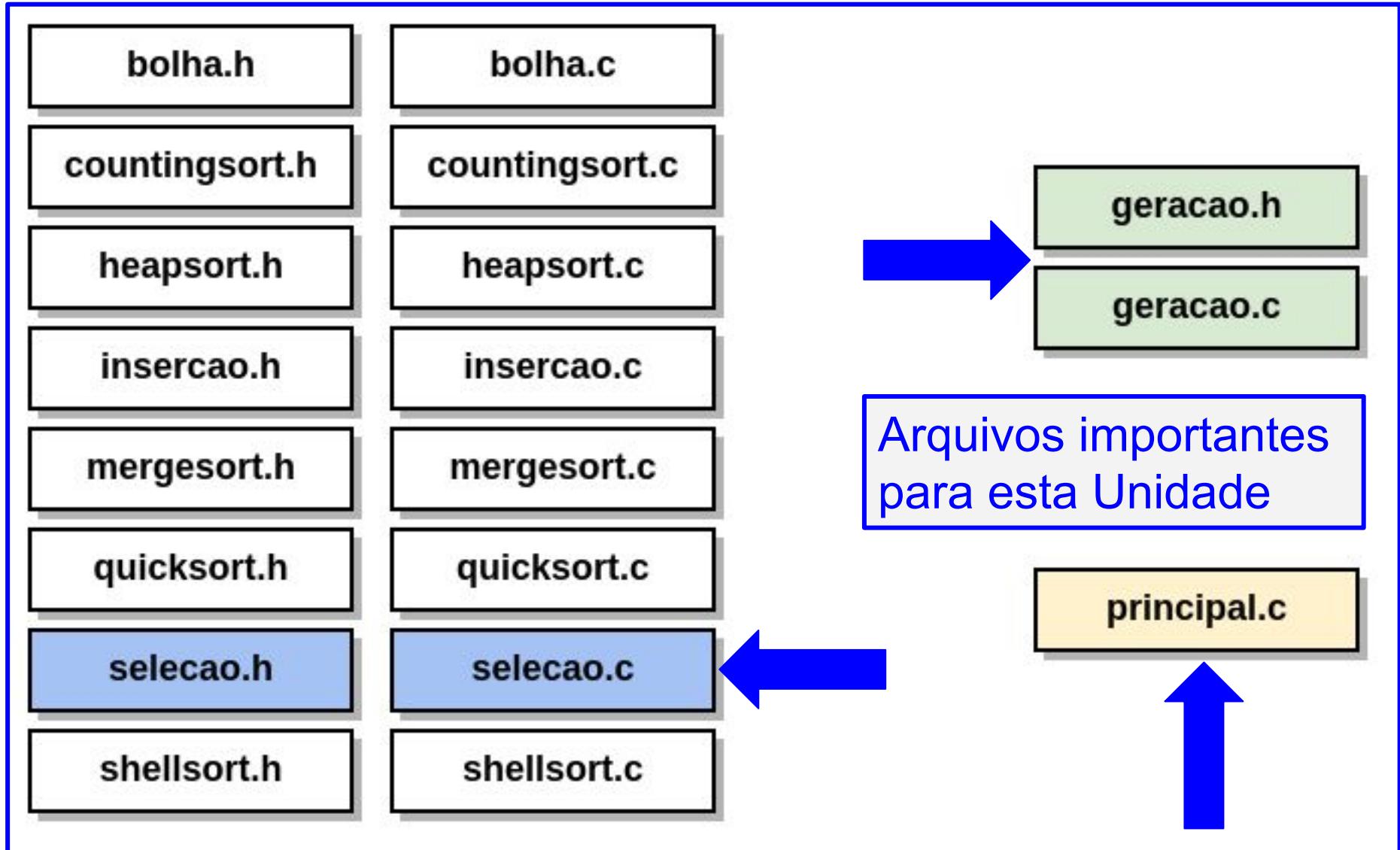
```

5

Agenda

- Introdução sobre Ordenação Interna
- Funcionamento básico
- Algoritmo em C *like*
 - Estrutura do Código em Java
 - **Estrutura do Código em C**
 - makefile para C/C++
- Análise dos número de movimentações e comparações
- **Estrutura dos nossos códigos em Java e C** 
- Conclusão

Estrutura do Código em C



Estrutura do Código em C

selecao.h

selecao.c

Arquivos selecao.h e selecao.c

```
//selecao.h
```

```
#ifndef SELECAO_H
#define SELECAO_H
//=====
#include "geracao.h"
//=====
void selecao(int *array, int n);
//=====
#endif
```

```
//selecao.c
```

```
#include "selecao.h"
//=====
void selecao(int *array, int n){
    for (int i = 0; i < (n - 1); i++) {
        int menor = i;
        for (int j = (i + 1); j < n; j++){
            if (array[menor] > array[j]){
                menor = j;
            }
        }
        swap(&array[menor], &array[i]);
    }
//=====
```

Estrutura do Código em C

geracao.h

geracao.c

Arquivos geracao.h e geracao.c

/geracao.h

```
#ifndef GERACAO_H
#define GERACAO_H
//=====
#include <stdbool.h>
//=====
void swap(int *i, int *j);
//=====
void crescente(int *array, int n);
//=====
void decrescente(int *array, int n);
//=====
void aleatorio(int *array, int n);
//=====
void mostrar(int *array, int n);
//=====
bool isOrdenado(int *array, int n);
//=====
#endif
```

//geracao.c



Estrutura do Código em C

principal.c

Arquivo principal.c

```
#include "bolha.h"
#include "countingsort.h"
#include "heapsort.h"
#include "insercao.h"
#include "mergesort.h"
#include "quicksort.h"
#include "selecao.h"
#include "shellsort.h"

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

=====
int main(int argc, char **argv) {
    //Declaração de variáveis
    int n = (argc < 2) ? 1000 : atoi(argv[1]);
    int *array = (int*) malloc(n*sizeof(int));
    clock_t inicio, fim;
    double total;
```

```
//Geração do conjunto a ser ordenado
aleatorio(array, n);
//crescente(array, n);
//decrescente(array, n);

//Mostrar o conjunto a ser ordenado
//mostrar(array, n);

//Execução do algoritmo de ordenação
inicio = clock();
//bolha(array, n);
//countingsort(array, n);
//heapsort(array, n);
//insercao(array, n);
//mergesort(array, n);
//quicksort(array, n);
selecao(array, n);
//shellsort(array, n);
fim = clock();
```

Arquivo principal.c

```
total = ((fim - inicio) / (double)CLOCKS_PER_SEC);

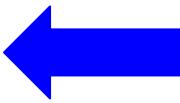
//Mostrar o conjunto ordenado, tempo de execução e status da ordenação
//algoritmo.mostrar(array, n);
printf("Tempo para ordenar: %f s.\n", total);
printf("isOrdenado: %s\n", isOrdenado(array, n) ? "true" : "false");

//Desalocar o espaço de memória do array
free(array);

return 0;
}
```

Como alocamos o vetor,
devemos desalocá-lo

Agenda

- Introdução sobre Ordenação Interna
- Funcionamento básico
- Algoritmo em C *like*
 - Estrutura do Código em Java
 - Estrutura do Código em C
 - **makefile para C/C++**
- Análise dos número de movimentações e comparações
- **Estrutura dos nossos códigos em Java e C** 
- Conclusão

makefile

- Arquivo contendo um conjunto de diretivas usadas pela ferramenta de automação de compilação *make* para gerar um alvo / meta
- Nesse caso, os arquivos serão compilados digitando ***make***

all: exec

exec: principal.o geracao.o bolha.o countingsort.o ... shellsort.o

gcc -o exec principal.o geracao.o bolha.o countingsort.o ... shellsort.o

principal.o: principal.c

gcc -o principal.o principal.c -c -W -Wall -pedantic

bolha.o: bolha.c

gcc -o bolha.o bolha.c -c -W -Wall -pedantic

geracao.o: geracao.c

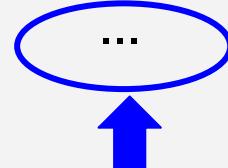
gcc -o geracao.o geracao.c -c -W -Wall -pedantic

clean:

rm -rf *.o *~ exec

limpa:

rm -rf *.o



Criamos um **alvo / comando** para
cada arquivo .c (**pré-requisito**)

all: exec

exec: principal.o geracao.o bolha.o countingsort.o ... shellsort.o

gcc -o exec principal.o geracao.o bolha.o countingsort.o ... shellsort.o

principal.o: principal.c

gcc -o principal.o principal.c -c -W -Wall -pedantic

bolha.o: bolha.c

gcc -o bolha.o bolha.c -c -W -Wall -pedantic

geracao.o: geracao.c

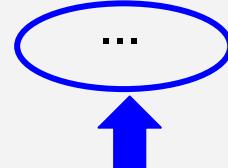
gcc -o geracao.o geracao.c -c -W -Wall -pedantic

clean:

rm -rf *.o *~ exec

limpa:

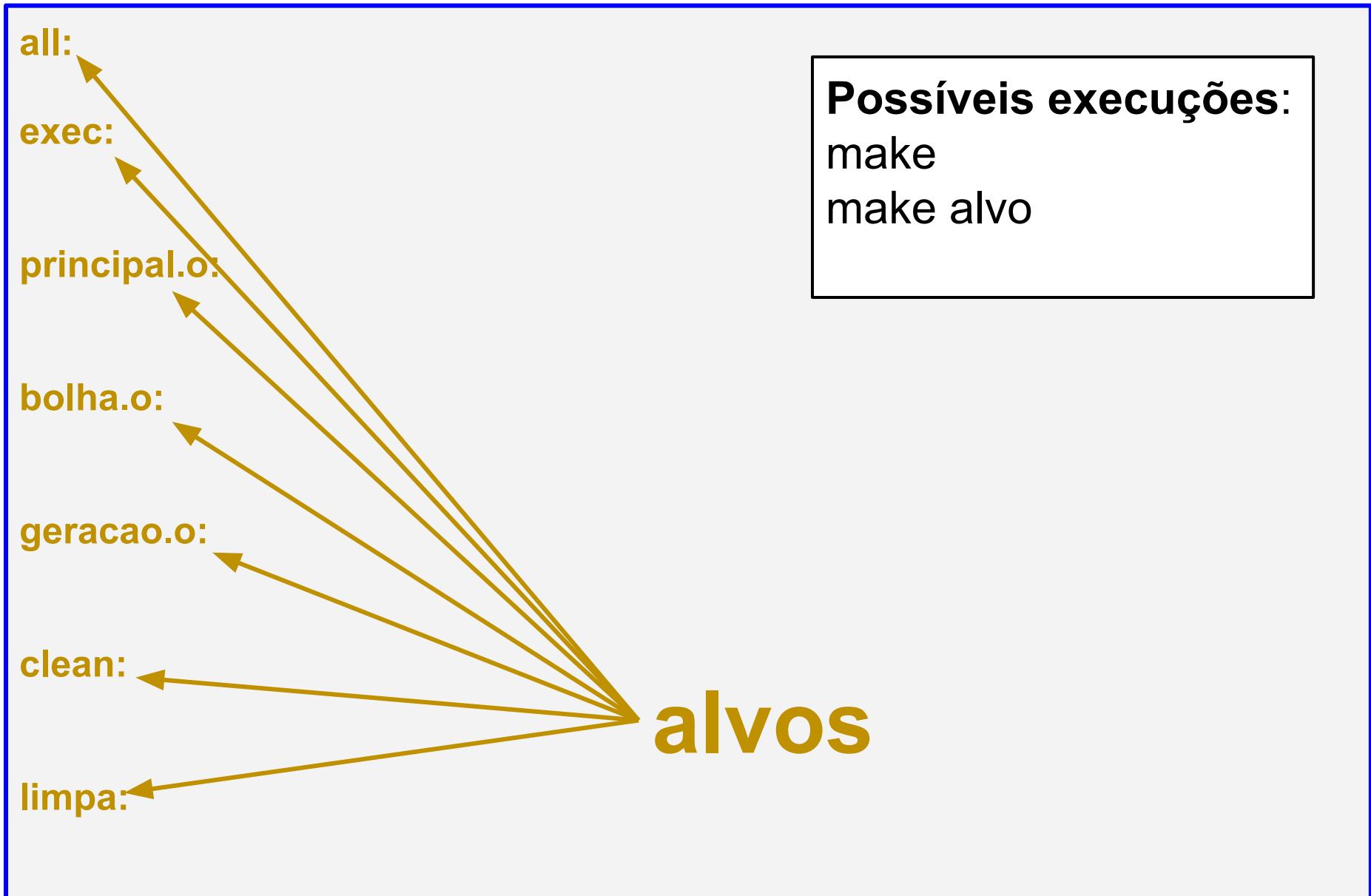
rm -rf *.o



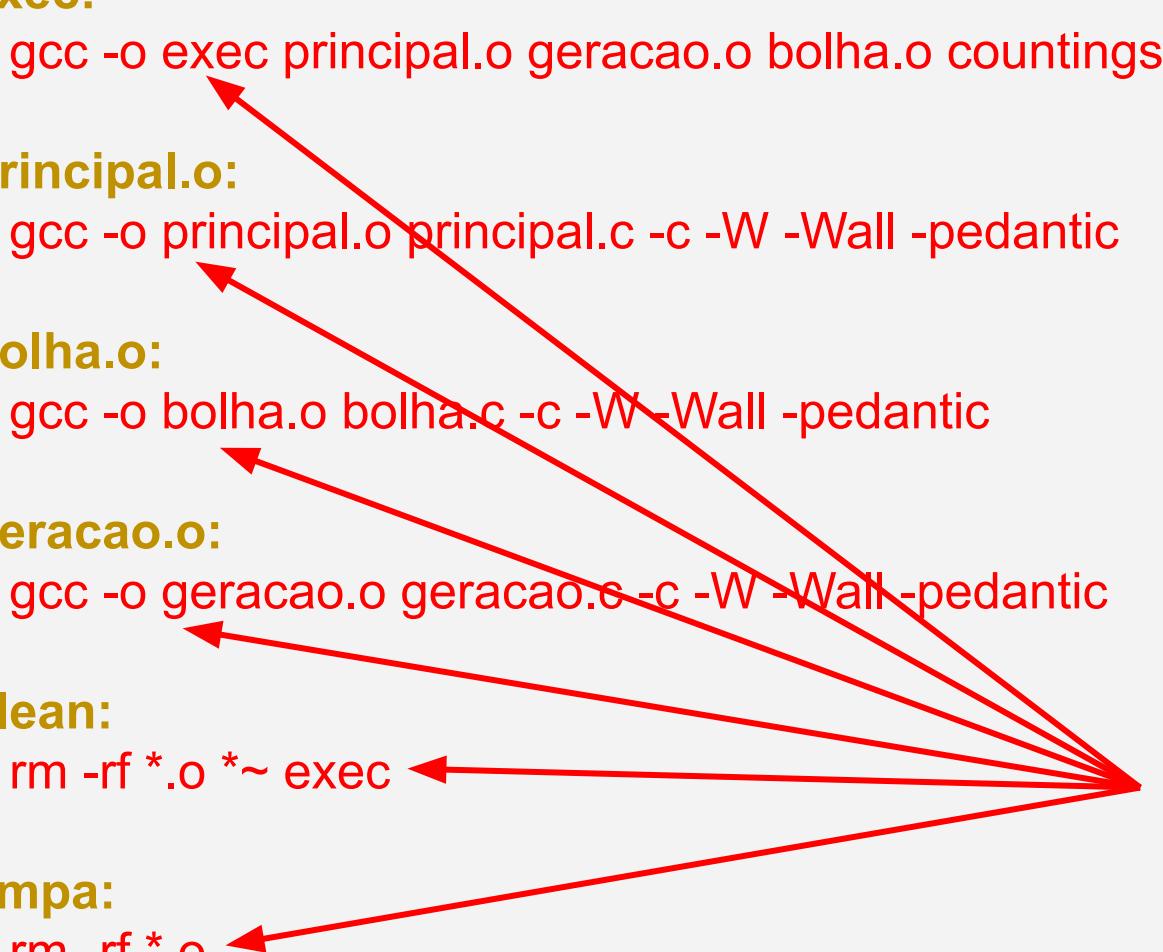
Arquivos

counting.sort.c
heapsort.c
insercao.c
mergesort.c
quicksort.c
selecao.c
shellsort.c

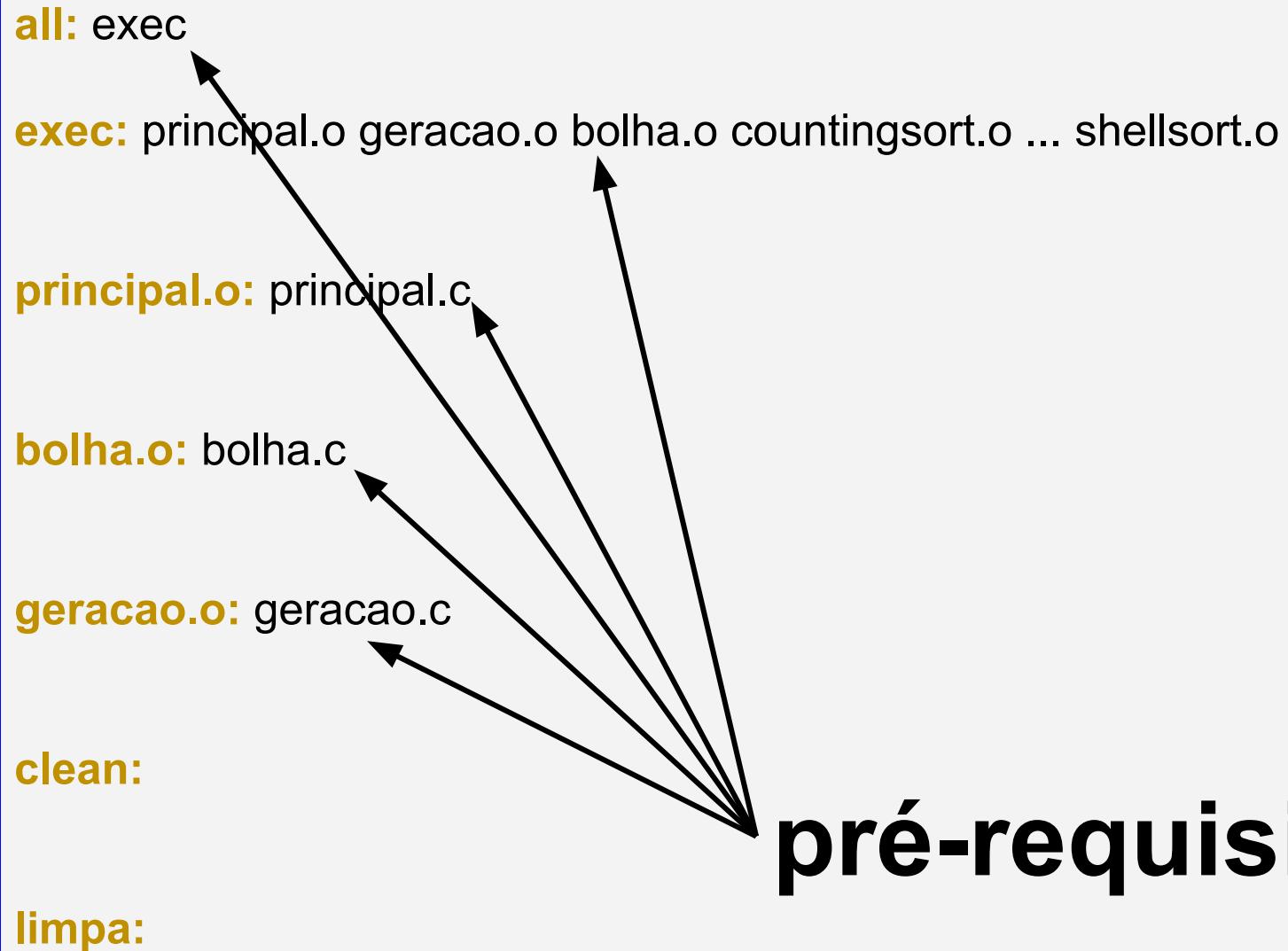
Criamos um **alvo / comando** para
cada arquivo .c (**pré-requisito**)



```
all:  
  
exec:  
    gcc -o exec principal.o geracao.o bolha.o countingsort.o ... shellsort.o  
  
principal.o:  
    gcc -o principal.o principal.c -c -W -Wall -pedantic  
  
bolha.o:  
    gcc -o bolha.o bolha.c -c -W -Wall -pedantic  
  
geracao.o:  
    gcc -o geracao.o geracao.c -c -W -Wall -pedantic  
  
clean:  
    rm -rf *.o *~ exec  
  
limpa:  
    rm -rf *.o
```



comandos



all: exec

exec: principal.o geracao.o bolha.o countingsort.o ... shellsort.o

gcc -o exec principal.o geracao.o bolha.o countingsort.o ... shellsort.o

principal.o: principal.c

gcc -o principal.o principal.c -c -W -Wall -pedantic

bolha.o: bolha.c

gcc -o bolha.o bolha.c -c -W -Wall -pedantic

geracao.o: geracao.c

gcc -o geracao.o geracao.c -c -W -Wall -pedantic

...

clean:

rm -rf *.o *~ exec

limpa:

rm -rf *.o

Exercício Resolvido (2)

• Na pasta da “Unidade 4/c/”, digite a sequência de comandos abaixo e explique a saída na tela

- 1) make all ; ls
- 2) make clean ; ls
- 3) make ; ls
- 4) make clean ; ls
- 5) make exec ; ls
- 6) make limpa ; ls
- 7) make bolha.o;ls

Exercício Resolvido (2)

- Na pasta da “Unidade 4/c/”, digite a sequência de comandos abaixo e explique a saída na tela

- 1) **make all ; ls**
- 2) **make clean ; ls**
- 3) **make ; ls**
- 4) **make clean ; ls**
- 5) **make exec ; ls**
- 6) **make limpa ; ls**
- 7) **make bolha.o ; ls**

```
:$ make all ; ls
gcc -o principal.o principal.c -c -W -Wall -pedantic
gcc -o geracao.o geracao.c -c -W -Wall -pedantic
gcc -o bolha.o bolha.c -c -W -Wall -pedantic
gcc -o countingsort.o countingsort.c -c -W -Wall -pedantic
gcc -o heapsort.o heapsort.c -c -W -Wall -pedantic
gcc -o insercao.o insercao.c -c -W -Wall -pedantic
gcc -o mergesort.o mergesort.c -c -W -Wall -pedantic
gcc -o quicksort.o quicksort.c -c -W -Wall -pedantic
gcc -o selecao.o selecao.c -c -W -Wall -pedantic
gcc -o shellsort.o shellsort.c -c -W -Wall -pedantic
gcc -o exec principal.o geracao.o bolha.o countingsort.o heapsort.o insercao.o
mergesort.o quicksort.o selecao.o shellsort.o

bolha.c bolha.o countingsort.h exec geracao.h heapsort.c heapsort.o insercao.h
makefile mergesort.h principal.c quicksort.c quicksort.o selecao.h shellsort.c
shellsort.o bolha.h countingsort.c countingsort.o geracao.c geracao.o heapsort.h
insercao.c insercao.o mergesort.c mergesort.o principal.o quicksort.h selecao.c
selecao.o shellsort.h
```

Exercício Resolvido (2)

- Na pasta da “Unidade 4/c/”, digite a sequência de comandos abaixo e explique a saída na tela

- 1) make all ; ls
- 2) **make clean ; ls**
- 3) make ; ls
- 4) make clean ; ls
- 5) make exec ; ls
- 6) make limpa ; ls
- 7) make bolha.o ; ls

```
:$ make clean ; ls  
rm -rf *.o *~ exec  
bolha.c bolha.h countingsort.c countingsort.h geracao.c geracao.h heapsort.c  
heapsort.h insercao.c insercao.h makefile mergesort.c mergesort.h principal.c  
quicksort.c quicksort.h selecao.c selecao.h shellsort.c shellsort.h
```

Exercício Resolvido (2)

- Na pasta da “Unidade 4/c/”, digite a sequência de comandos abaixo e explique a saída na tela

- 1) make all ; ls
- 2) make clean ; ls
- 3) **make ; ls**
- 4) make clean ; ls
- 5) make exec ; ls
- 6) make limpa ; ls
- 7) make bolha.o ; ls

```
:$ make ; ls
gcc -o principal.o principal.c -c -W -Wall -pedantic
gcc -o geracao.o geracao.c -c -W -Wall -pedantic
gcc -o bolha.o bolha.c -c -W -Wall -pedantic
gcc -o countingsort.o countingsort.c -c -W -Wall -pedantic
gcc -o heapsort.o heapsort.c -c -W -Wall -pedantic
gcc -o insercao.o insercao.c -c -W -Wall -pedantic
gcc -o mergesort.o mergesort.c -c -W -Wall -pedantic
gcc -o quicksort.o quicksort.c -c -W -Wall -pedantic
gcc -o selecao.o selecao.c -c -W -Wall -pedantic
gcc -o shellsort.o shellsort.c -c -W -Wall -pedantic
gcc -o exec principal.o geracao.o bolha.o countingsort.o heapsort.o insercao.o
mergesort.o quicksort.o selecao.o shellsort.o

bolha.c bolha.o countingsort.h exec geracao.h heapsort.c heapsort.o insercao.h
makefile mergesort.h principal.c quicksort.c quicksort.o selecao.h shellsort.c
shellsort.o bolha.h countingsort.c countingsort.o geracao.c geracao.o heapsort.h
insercao.c insercao.o mergesort.c mergesort.o principal.o quicksort.h selecao.c
selecao.o shellsort.h
```

Exercício Resolvido (2)

- Na pasta da “Unidade 4/c/”, digite a sequência de comandos abaixo e explique a saída na tela

- 1) make all ; ls
- 2) make clean ; ls
- 3) make ; ls
- 4) **make clean ; ls**
- 5) make exec ; ls
- 6) make limpa ; ls
- 7) make bolha.o ; ls

```
:$ make clean ; ls  
rm -rf *.o *~ exec  
bolha.c bolha.h countingsort.c countingsort.h geracao.c geracao.h heapsort.c  
heapsort.h insercao.c insercao.h makefile mergesort.c mergesort.h principal.c  
quicksort.c quicksort.h selecao.c selecao.h shellsort.c shellsort.h
```

Exercício Resolvido (2)

- Na pasta da “Unidade 4/c/”, digite a sequência de comandos abaixo e explique a saída na tela

- 1) make all ; ls
- 2) make clean ; ls
- 3) make ; ls
- 4) make clean ; ls
- 5) **make exec** ; ls
- 6) make limpa ; ls
- 7) make bolha.o ; ls

```
:$ make ; ls
gcc -o principal.o principal.c -c -W -Wall -pedantic
gcc -o geracao.o geracao.c -c -W -Wall -pedantic
gcc -o bolha.o bolha.c -c -W -Wall -pedantic
gcc -o countingsort.o countingsort.c -c -W -Wall -pedantic
gcc -o heapsort.o heapsort.c -c -W -Wall -pedantic
gcc -o insercao.o insercao.c -c -W -Wall -pedantic
gcc -o mergesort.o mergesort.c -c -W -Wall -pedantic
gcc -o quicksort.o quicksort.c -c -W -Wall -pedantic
gcc -o selecao.o selecao.c -c -W -Wall -pedantic
gcc -o shellsort.o shellsort.c -c -W -Wall -pedantic
gcc -o exec principal.o geracao.o bolha.o countingsort.o heapsort.o insercao.o
mergesort.o quicksort.o selecao.o shellsort.o

bolha.c bolha.o countingsort.h exec geracao.h heapsort.c heapsort.o insercao.h
makefile mergesort.h principal.c quicksort.c quicksort.o selecao.h shellsort.c
shellsort.o bolha.h countingsort.c countingsort.o geracao.c geracao.o heapsort.h
insercao.c insercao.o mergesort.c mergesort.o principal.o quicksort.h selecao.c
selecao.o shellsort.h
```

Exercício Resolvido (2)

- Na pasta da “Unidade 4/c/”, digite a sequência de comandos abaixo e explique a saída na tela

- 1) make all ; ls
- 2) make clean ; ls
- 3) make ; ls
- 4) make clean ; ls
- 5) make exec ; ls
- 6) **make limpa** ; ls
- 7) make bolha.o ; ls

```
:$ make limpa ; ls  
  
rm -rf *.o  
  
bolha.c bolha.h countingsort.c countingsort.h exec geracao.c geracao.h heapsort.c  
heapsort.h insercao.c insercao.h makefile mergesort.c mergesort.h principal.c  
quicksort.c quicksort.h selecao.c selecao.h shellsort.c shellsort.h
```

Exercício Resolvido (2)

- Na pasta da “Unidade 4/c/”, digite a sequência de comandos abaixo e explique a saída na tela

- 1) make all ; ls
- 2) make clean ; ls
- 3) make ; ls
- 4) make clean ; ls
- 5) make exec ; ls
- 6) make limpa ; ls
- 7) **make bolha.o ; ls**

```
:$ make bolha.o ; ls  
gcc -o bolha.o bolha.c -c -W -Wall -pedantic  
bolha.c bolha.h bolha.o countingsort.c countingsort.h exec geracao.c geracao.h  
heapsort.c heapsort.h insercao.c insercao.h makefile mergesort.c mergesort.h  
principal.c quicksort.c quicksort.h selecao.c selecao.h shellsort.c shellsort.h
```

all: exec

exec: principal.o geracao.o bolha.o countingsort.o ... shellsort.o

gcc -o exec principal.o geracao.o bolha.o countingsort.o ... shellsort.o

principal.o: principal.c

gcc -o principal.o principal.c -c -W -Wall -pedantic

bolha.o: bolha.c

gcc -o bolha.o bolha.c -c -W -Wall -pedantic

geracao.o: geracao.c

gcc -o geracao.o geracao.c -c -W -Wall -pedantic

...

clean:

rm -rf *.o *~ exec

limpa:

rm -rf *.o

Podemos otimizar o makefile, contudo,
isso é assunto para outra aula!!!

Agenda

- Introdução sobre Ordenação Interna
- Funcionamento básico
- Algoritmo em C *like*
- Análise dos número de movimentações e comparações
- Estrutura dos nossos códigos em Java e C
- **Conclusão** 

Conclusão

- Vantagem: o número de movimentações é linear e isso é interessante quando os registros são "grandes"
- Desvantagens:
 - $\Theta(n^2)$ comparações
 - Não há melhor caso
 - Algoritmo não Estável

Exercício (1)

- Mostre todas as comparações e movimentações do algoritmo anterior para o array abaixo:

12	4	8	2	14	17	6	18	10	16	15	5	13	9	1	11	7	3
----	---	---	---	----	----	---	----	----	----	----	---	----	---	---	----	---	---

Exercício (2)

- Execute a versão abaixo do Seleção para arrays gerados aleatoriamente.

Em seguida, discuta sobre os números de comparações inseridas e movimentações evitadas pela nova versão do algoritmo

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    if (menor != i){  
        swap(menor, i);  
    }  
}
```

Exercício (3)

- Contabilize os números de comparações e movimentações entre elementos do *array*; calcule os valores teóricos para as duas métricas; e contabilize o tempo de execução. Em seguida, para os códigos em Java e C, gere *arrays* aleatórios (seed 0) com tamanhos 100, 1000 e 10000. Para cada instância (variação de linguagem e tamanho de vetor), faça 33 execuções. Faça um gráfico para os valores médios de cada métrica avaliada (comparações, movimentações e tempo de execução) variando o tamanho do *array*. Nos gráficos de comparações e movimentações, mostre também os resultados teóricos. Cada gráfico terá uma curva para cada linguagem. Interprete os resultados obtidos. Repita o processo para *arrays* gerados de forma crescente e decrescente.