

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
- Conclusão

Agenda

- **Introdução sobre Ordenação Interna** 
- Funcionamento básico
- Algoritmo em C *like*
- Análise dos número de movimentações e comparações
- 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

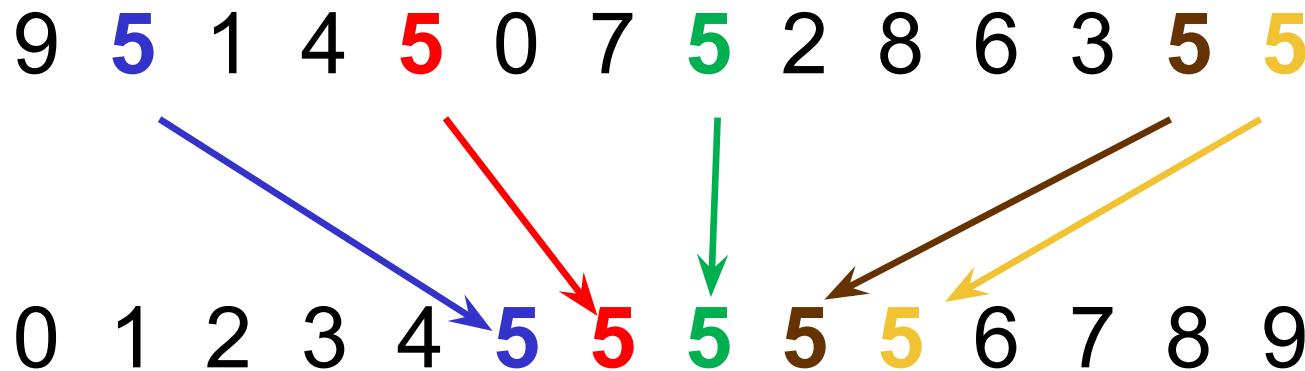
- Operações fundamentais: comparação e movimentação entre elementos do *array*
- O limite inferior em termos do número de comparações (entre elementos do *array*) para a ordenação interna é $\Theta(n \times \lg(n))$
- Logo, a complexidade ótima para a ordenação interna em número de comparações do pior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

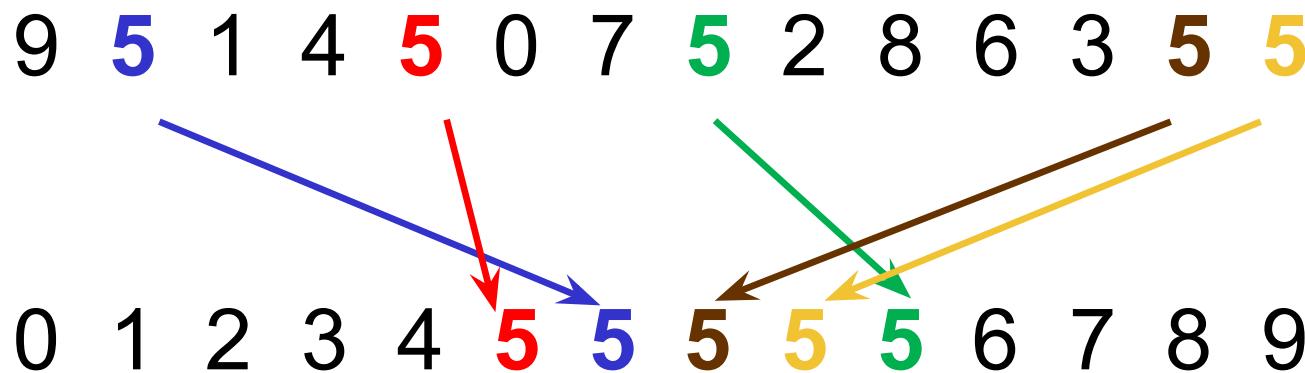
- Antes:



Algoritmos Estáveis vs. Não Estáveis

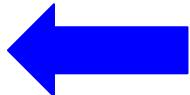
- Um algoritmo é dito estável se depois da execução, os elementos com a mesma chave mantiverem a ordem relativa entre as chaves repetidas
- No exemplo abaixo, a ordem dos elementos azul, vermelho, verde e marrom e amarelo não foi mantida (algoritmo não estável)

- Antes:



- Depois:

Agenda

- Introdução sobre Ordenação Interna
- **Funcionamento básico** 
- Algoritmo em C like
- Análise dos número de movimentações e comparações
- 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

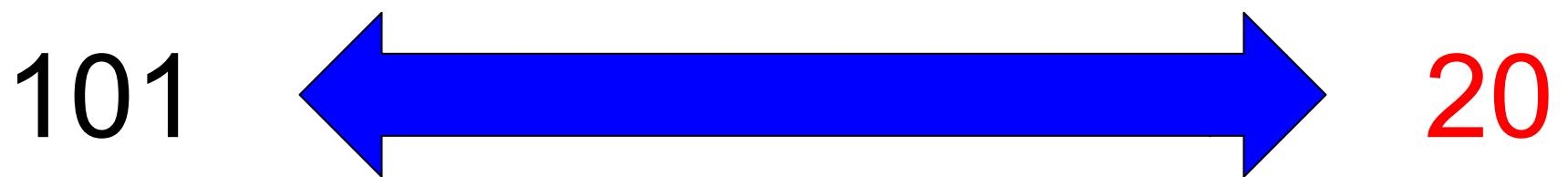
101 115 30 63 47 20

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

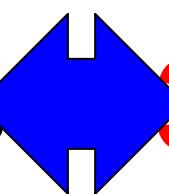
20 115 30 63 47 101

Exemplo

20 115 30 63 47 101

Menor
elemento

Exemplo

20 115  30 63 47 101

Exemplo

20 30

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

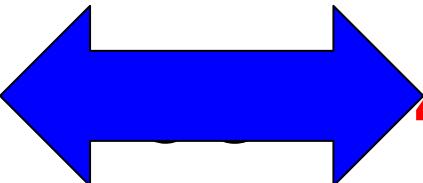
Trocando a posição do menor elemento com o primeiro

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

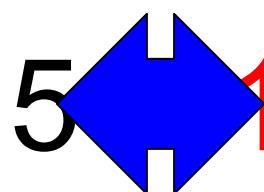
Trocando a posição do menor elemento com o primeiro

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
- Conclusão

Algoritmo em C *like*

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

(Obs.1): No Seleção, os valores de i serão: 0, 1, 2, 3, ... e (n-2)

O laço externo não é executado quando i igual a (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): As variáveis ***n*** e ***array*** são globais (em C/C++). Em Java/C#, elas são atributos da classe

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

Quem deseja apostar um chocolate que, em Somatórios, alguém perguntará o motivo do somatório de comparações começar em zero e terminar em $(n-2)$?

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

Algoritmo em C *like*

4

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

Olha o chocolate novamente...

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

Algoritmo em C *like*

5

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

6

(Obs.6): 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; 7  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

(Obs.7): 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);  
}
```



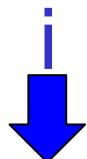
(Obs.8): O *swap* troca o conteúdo das posições menor e i

```
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);  
}
```

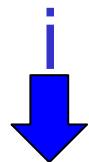


101	115	30	63	47	20
0	1	2	3	4	5

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: $2 < 6$

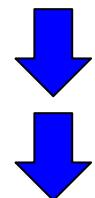
menor
i
j

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);  
}
```

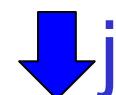
true: $101 > 30$



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

↓ 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
↓
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
↓
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

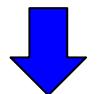
1



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

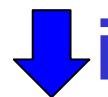
4

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);  
}
```

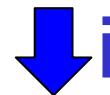


20	115	30	63	47	101
0	1	2	3	4	5

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

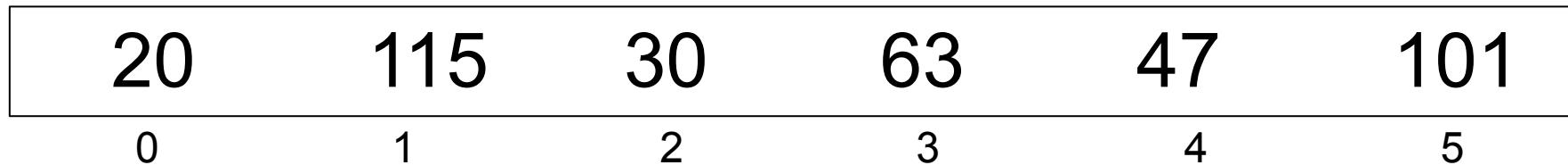
20	115	30	63	47	101
0	1	2	3	4	5

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

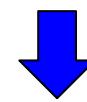
↓ i

↓ 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

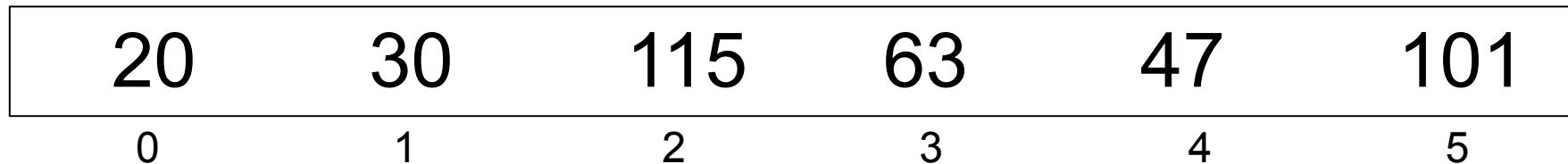
Algoritmo em C *like*

```
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

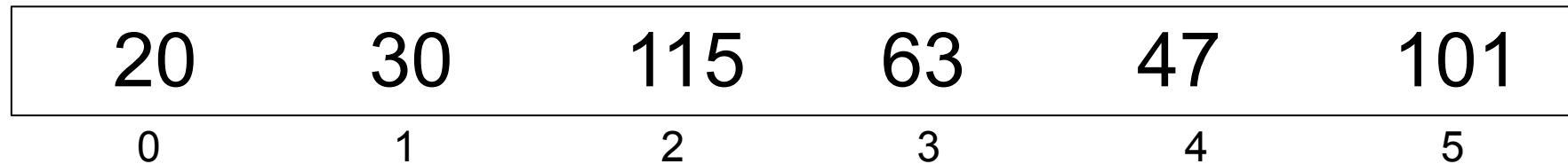
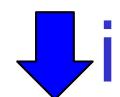
↓ i

j
↓



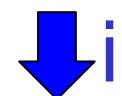
Algoritmo em C *like*

```
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);  
}
```



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);  
}
```

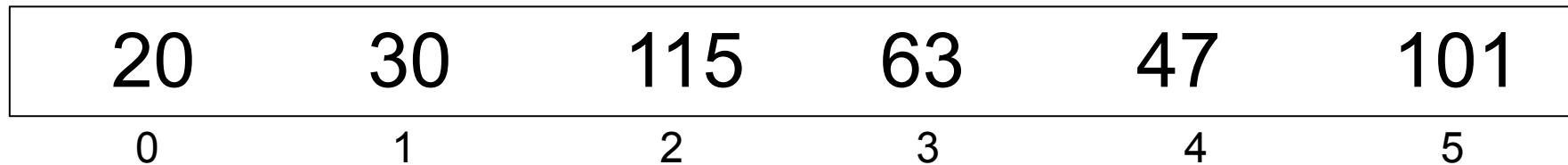
true: $2 < 5$ 

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

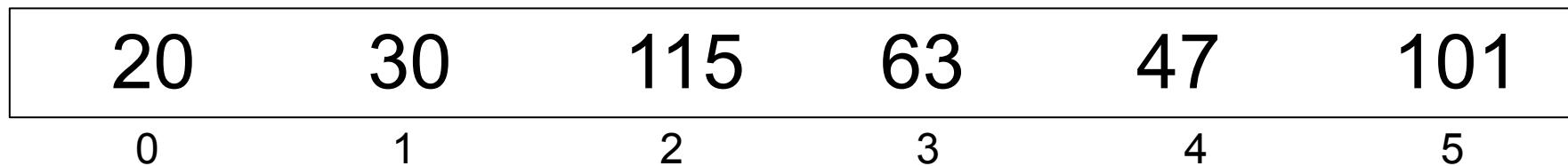
Algoritmo em C *like*

```
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

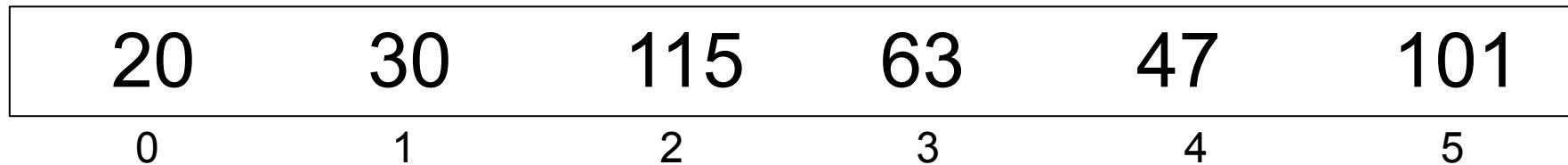


Algoritmo em C *like*

```
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: $115 > 63$

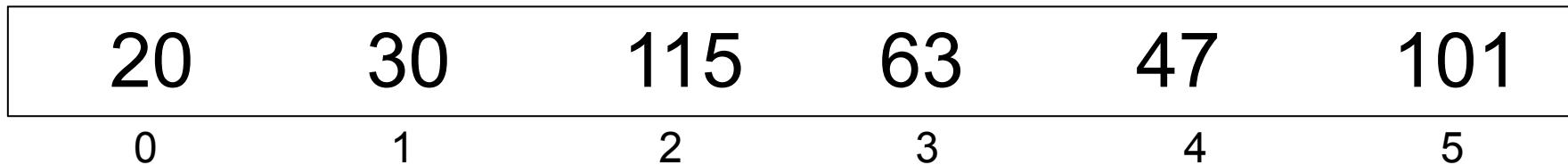
menor
i j



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

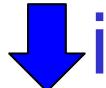
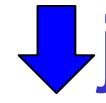
↓ i

↓ j

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

Algoritmo em C *like*

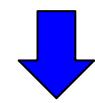
```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

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

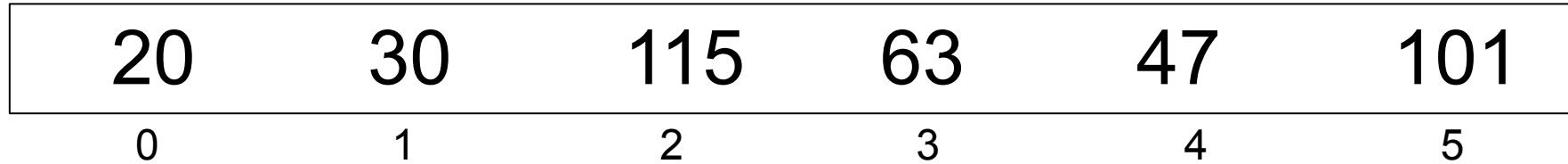
Algoritmo em C *like*

```
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

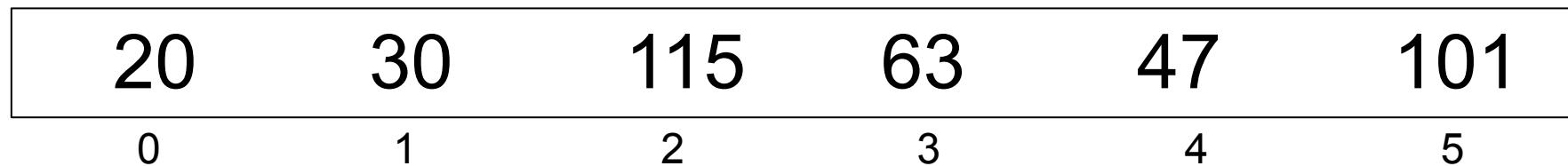
i



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

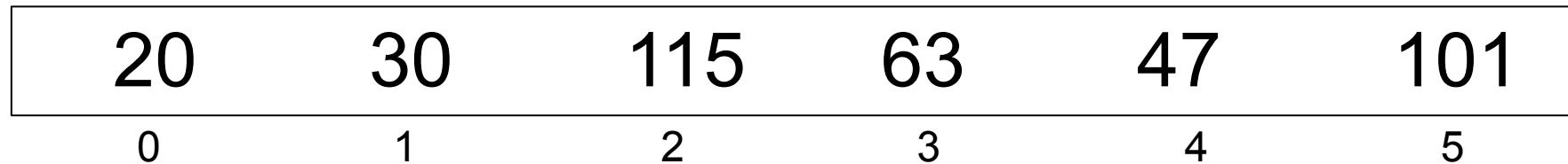
20	30	115	63	47	101
0	1	2	3	4	5

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);  
}
```

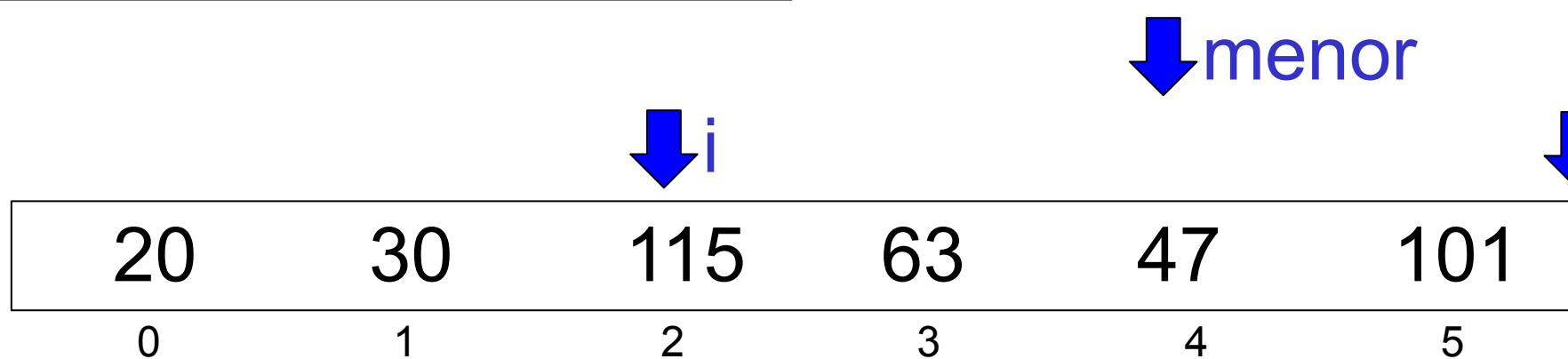
false: 47 > 101

menor
↓
j
↓



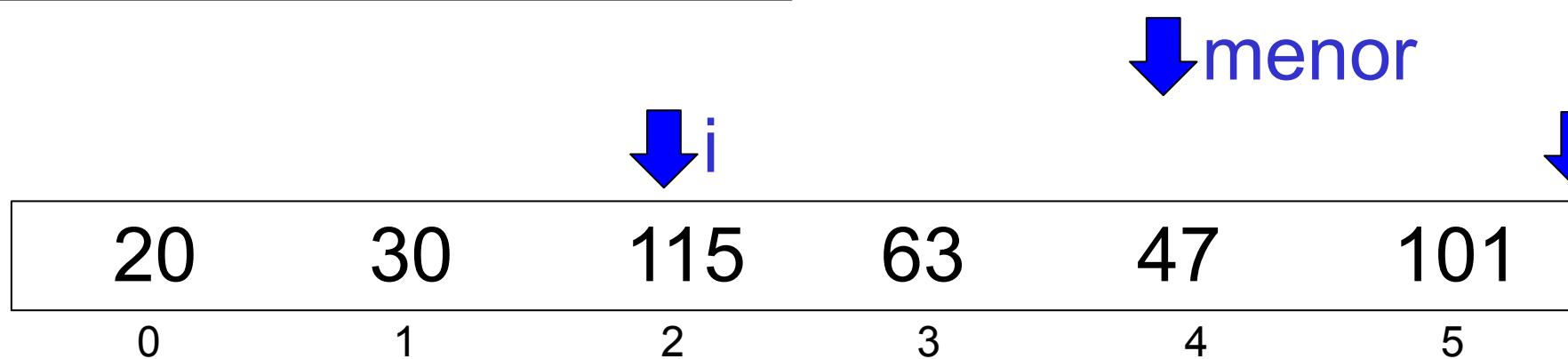
Algoritmo em C *like*

```
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);  
}
```

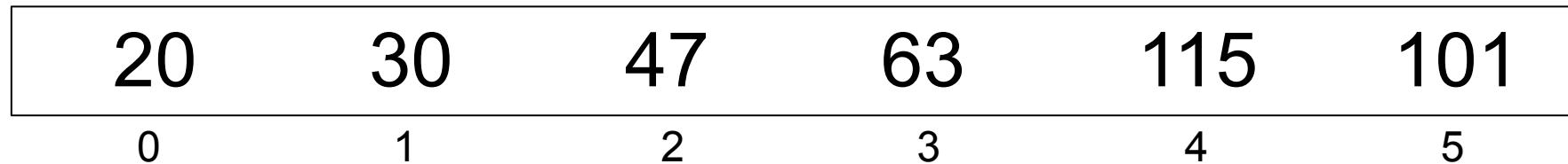
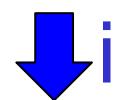
↓ menor

↓ i

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

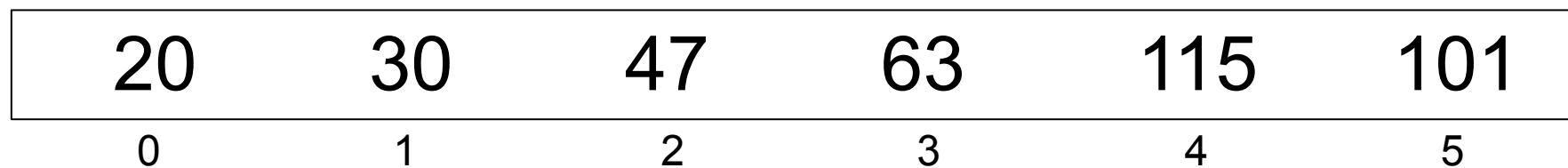
Algoritmo em C *like*

```
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);  
}
```

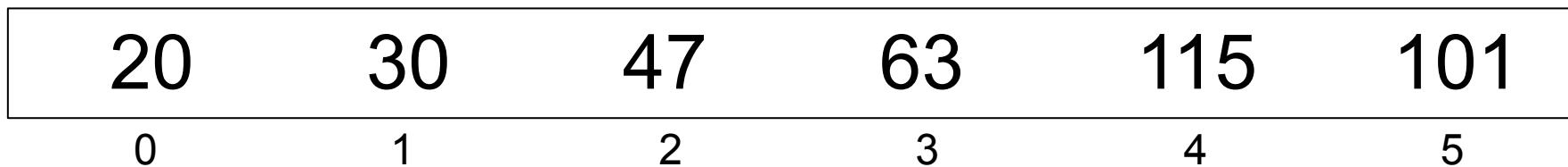
↓ menor
↓ i

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

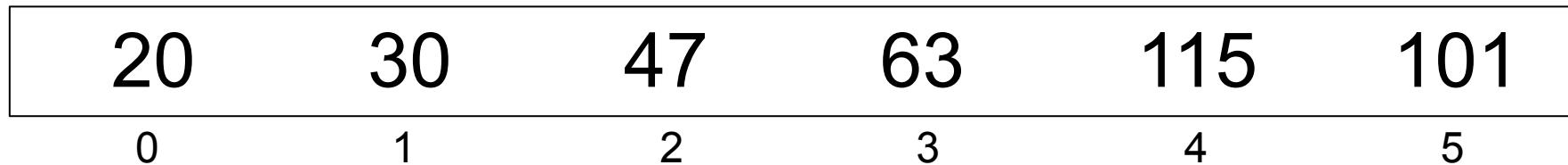


Algoritmo em C *like*

```
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

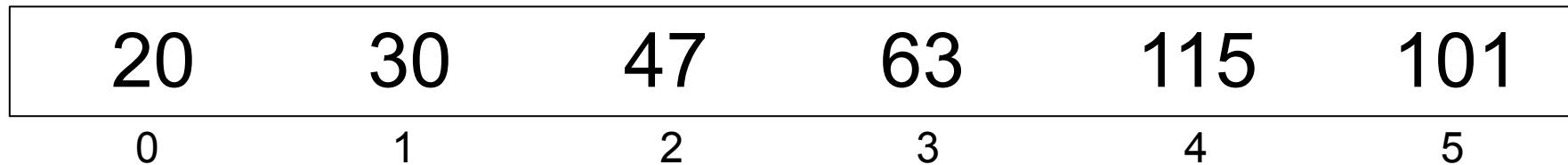


Algoritmo em C *like*

```
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: $63 > 115$

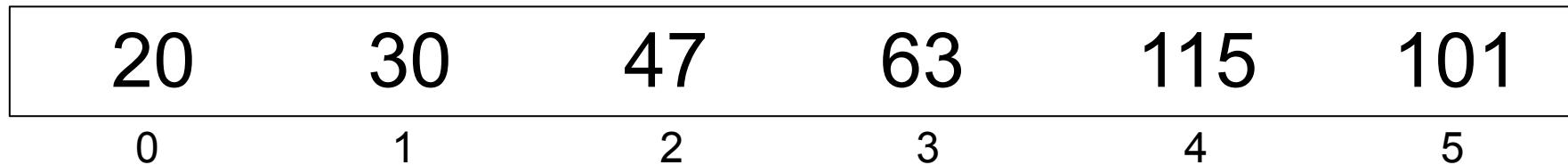
↓ menor
↓ i ↓ j



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

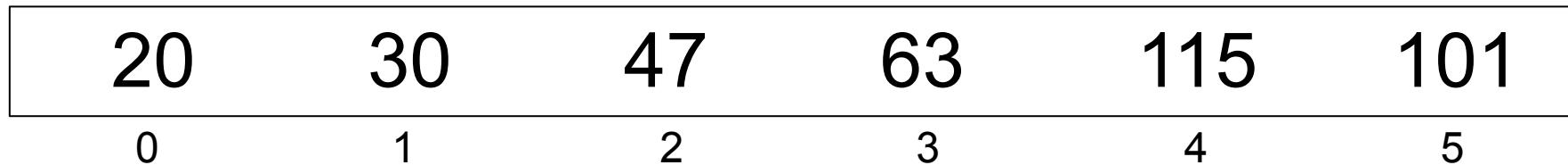


Algoritmo em C *like*

```
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);  
}
```

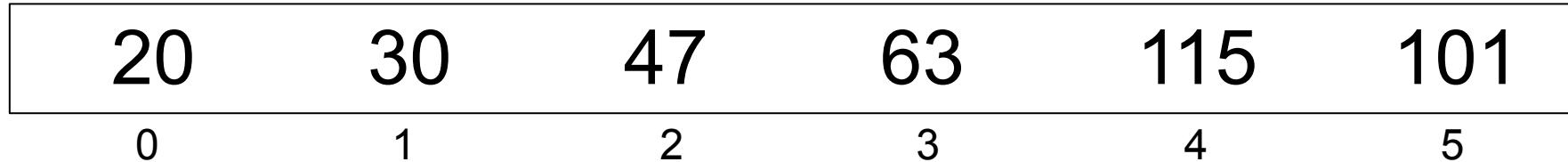
false: $6 < 6$



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*

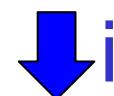
```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++) {  
        if (array[menor] > array[j]) {  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

↓ 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

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

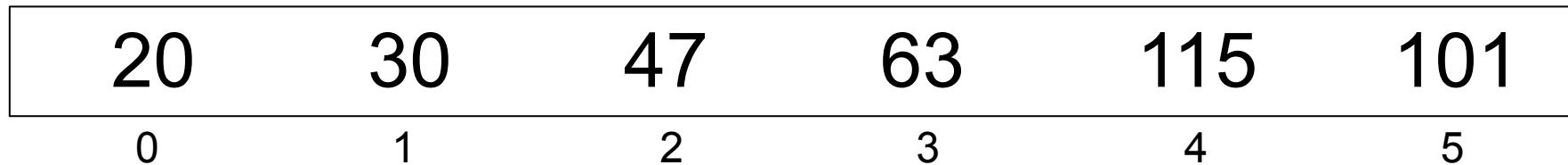
20	30	47	63	115	101
0	1	2	3	4	5

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

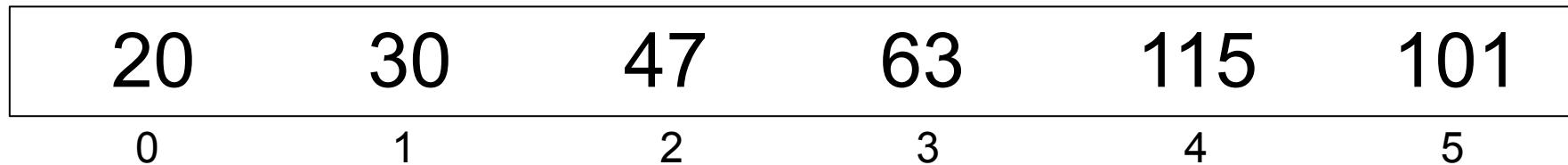


Algoritmo em C *like*

```
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

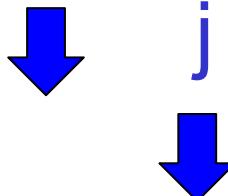


20	30	47	63	115	101
0	1	2	3	4	5

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



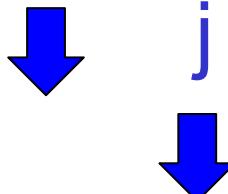
20	30	47	63	115	101
0	1	2	3	4	5

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);  
}
```

false: $6 < 6$

menor



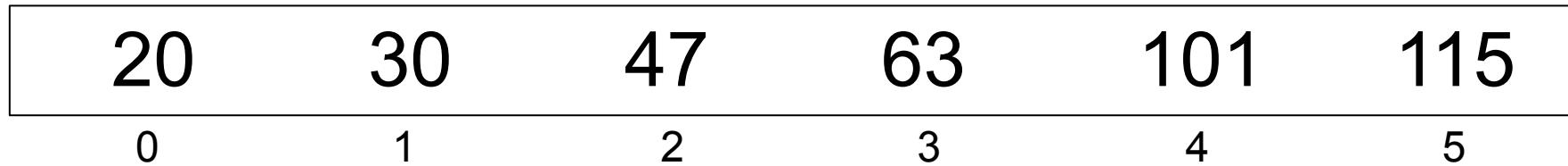
Algoritmo em C *like*

```
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



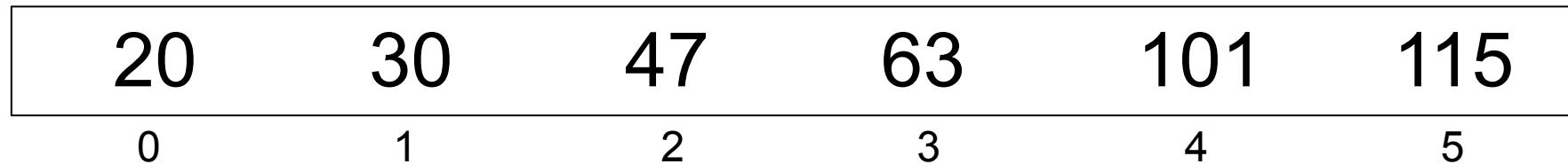
↓ i



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

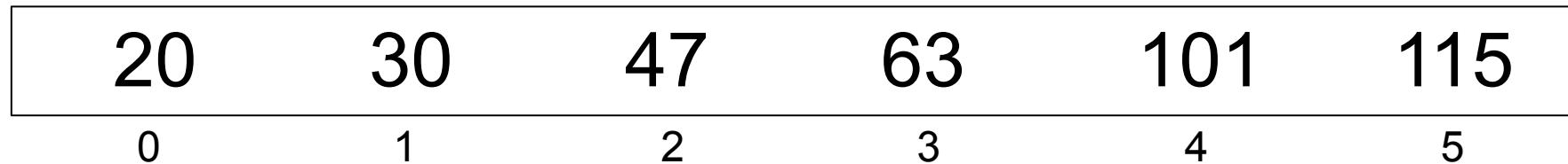


Algoritmo em C *like*

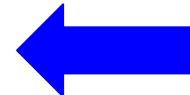
```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

false: $5 < 5$

↓ i



Agenda

- Introdução sobre Ordenação Interna
- Funcionamento básico
- Algoritmo em C *like*
- **Análise dos número de movimentações e comparações** 
- 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("Prática:" + mov);
System.out.println("Teoria:" + (3*n-3));
```

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[j] < array[menor])
            menor = j;
    }
    swap(menor, array[i]);
}

```

i	#cmp	
0	$n - (0+1)$	$(n-1)$
1	$n - (1+1)$	$(n-2)$
2	$n - (2+1)$	$(n-3)$
3	$n - (3+1)$	$(n-4)$
4	$n - (4+1)$	$(n-5)$
...		
$(n-4)$	$n - ((n-4)+1)$	$n - n + 4 - 1 = 3$
$(n-3)$	$n - ((n-3)+1)$	$n - n + 3 - 1 = 2$
$(n-2)$	$n - (n-2)+1)$	$n - n + 2 - 1 = 1$

Temos soma de comparações

Quantas comparações entre elementos (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



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

- Perturbando 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

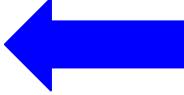
- Simplificando, temos:

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

- Finalmente:

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

Agenda

- Introdução sobre Ordenação Interna
- Funcionamento básico
- Algoritmo em C *like*
- Análise dos número de movimentações e comparações
- **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.