

# Unidade IV:


## Ordenação Interna - Algoritmo de Inserção



**PUC Minas**

Instituto de Ciências Exatas e Informática  
Departamento de Ciência da Computação

- Funcionamento básico
- Algoritmo em C *like*
- Análise dos número de comparações e movimentações
- Conclusão

- **Funcionamento básico** 
- Algoritmo em C *like*
- Análise dos número de comparações e movimentações
- Conclusão

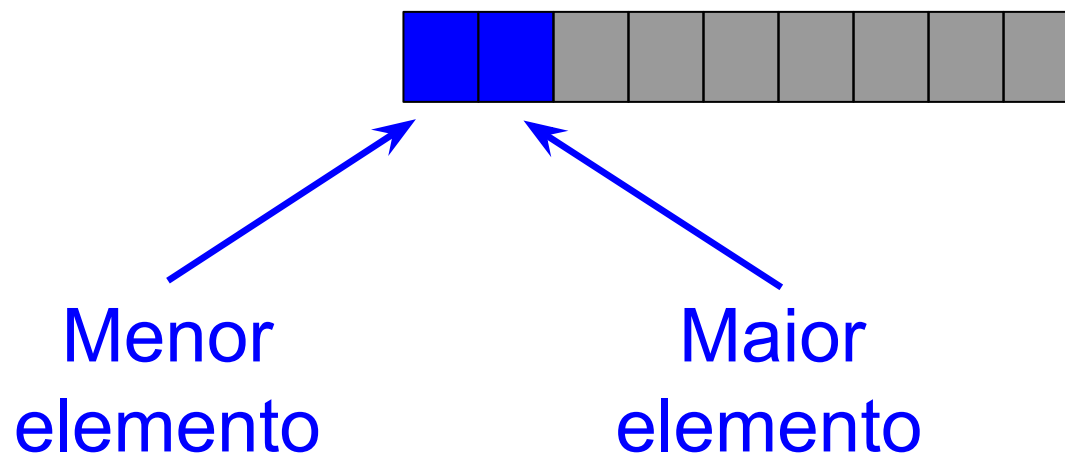
# Funcionamento Básico

- Temos duas sequências (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



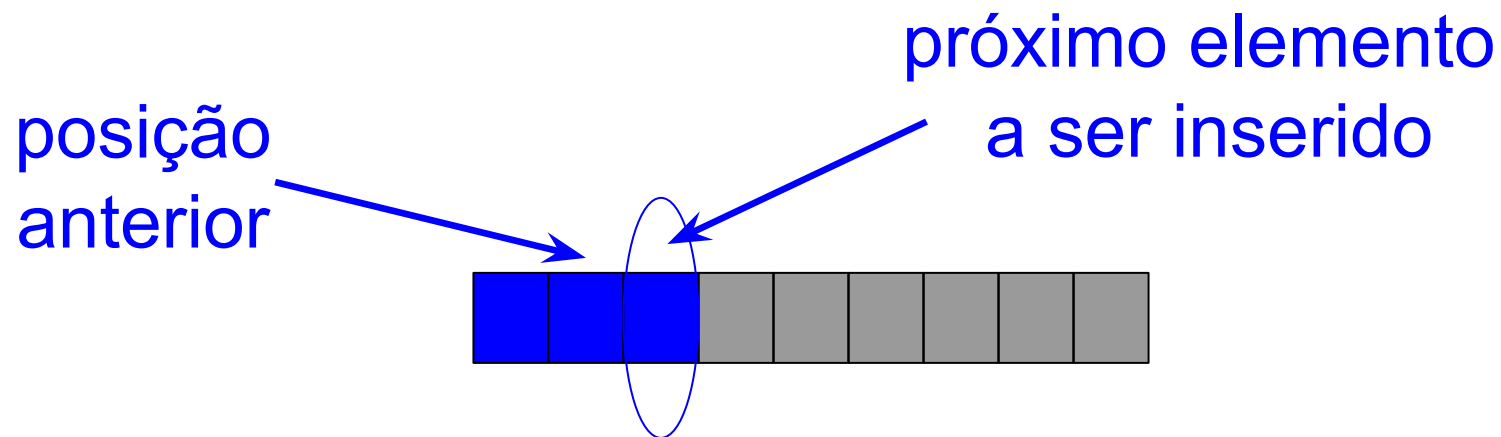
# Funcionamento Básico

- Temos duas sequências (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



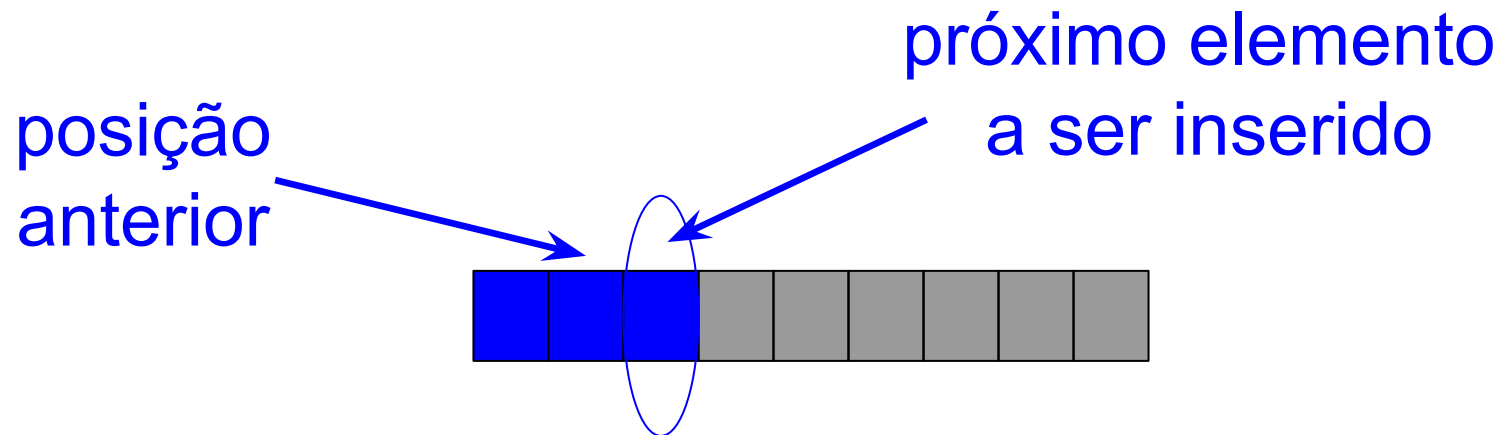
# Funcionamento Básico

- Temos duas sequências (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



# Funcionamento Básico

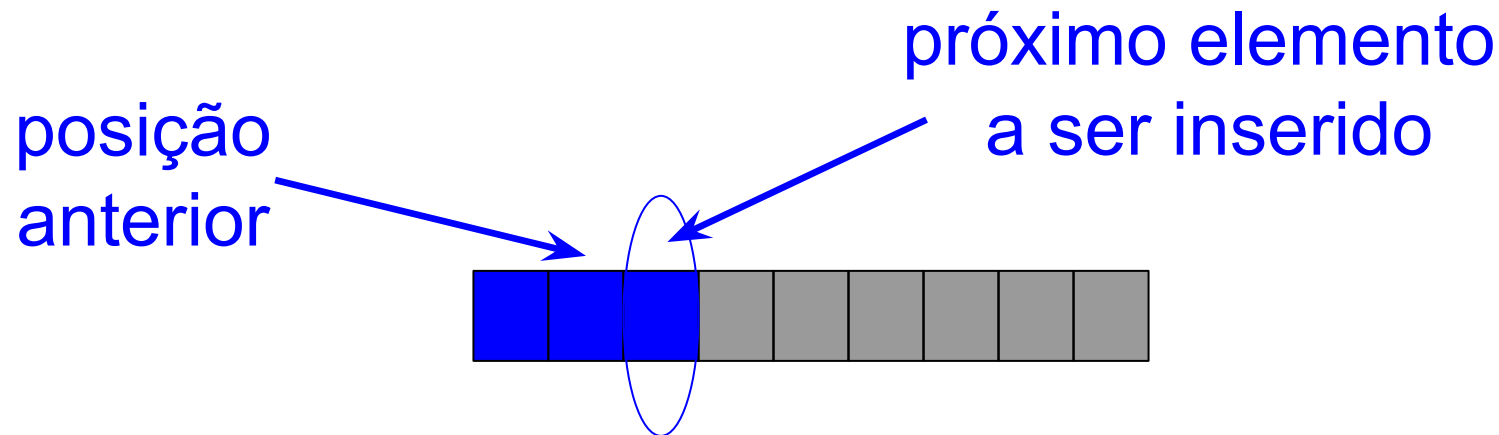
- Temos duas sequências (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



Se ele for maior ou igual ao da posição anterior, os três estão nas posições atuais corretas

# Funcionamento Básico

- Temos duas sequências (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



Senão, copiamos o novo elemento para uma área temporária e subimos em uma posição todos os demais que forem maiores que o novo



# Funcionamento Básico

- Temos duas sequências (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



# Funcionamento Básico

- Temos duas sequências (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



# Funcionamento Básico

- Temos duas sequências (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



# Funcionamento Básico

- Temos duas sequências (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



# Funcionamento Básico

- Temos duas sequências (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



# Funcionamento Básico

- Temos duas sequências (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



101      115      30      63      47      20

101 115 30 63 47 20

Inicialmente, temos um elemento,  
logo, ele está na posição correta

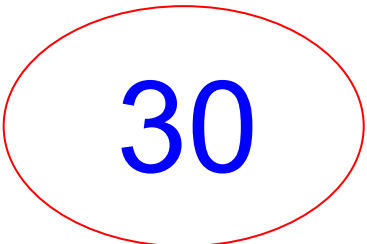


101   115   30   63   47   20



Comparamos o 101 e 115 e,  
como o novo elemento é o  
maior, os dois estão em ordem

101    115    30    63    47    20



Comparamos o 115 e 30 e, como o novo elemento é o menor, vamos procurar sua posição na lista ordenada

30 variável  
temporária

101    115    30    63    47    20

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 30

30 variável  
temporária

101      115      63      47      20

---

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 30



30

variável  
temporária

101

115

63

47

20

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 30

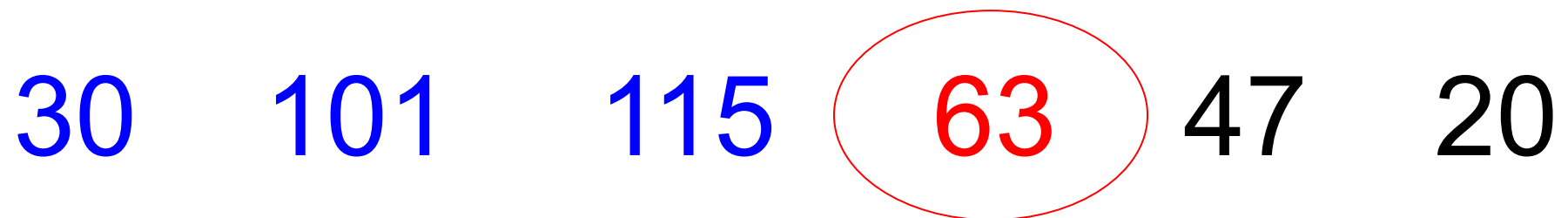


Encontramos a  
posição do 30

30    101    115    63    47    20

Encontramos a  
posição do 30

30    101    115    63    47    20



Comparamos o 115 e 63 e, como o novo elemento é o menor, vamos procurar sua posição na lista ordenada



63variável  
temporária

30    101    115    \_\_\_\_\_    47    20

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 63

63variável  
temporária

30    101    \_\_\_\_\_    115    47    20

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 63

**63**variável  
temporária

30

---

101

115

47

20

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 63

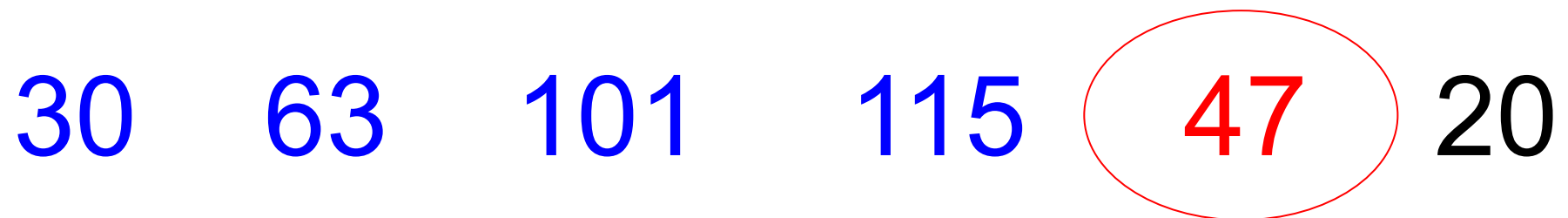


Encontramos a  
posição do 63

30    63    101    115    47    20

Encontramos a  
posição do 63

30    63    101    115    47    20



Comparamos o 115 e 47 e, como o novo elemento é o menor, vamos procurar sua posição na lista ordenada

47 variável  
temporária

30    63    101    115    \_\_\_\_\_    20

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 47

47 variável temporária

30    63    101    \_\_\_\_\_    115    20

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 47



47variável  
temporária

30    63    \_\_\_\_\_    101    115    20

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 47

47variável  
temporária

30

---

63

101

115

20

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 47



Encontramos a  
posição do 47

30    47    63    101    115    20

Encontramos a  
posição do 47

30    47    63    101    115    20

Comparamos o 115 e 20 e, como o novo elemento é o menor, vamos procurar sua posição na lista ordenada

20 variável  
temporária

30    47    63    101    115    \_\_\_\_\_

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 20

20 variável  
temporária

30    47    63    101    \_\_\_\_\_    115

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 20

20 variável  
temporária

30    47    63    \_\_\_\_\_    101    115

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 20



20 variável  
temporária

30    47                63    101    115

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 20

20 variável  
temporária

30      47      63      101      115

---

Enquanto procuramos,  
deslocamos os elementos  
maiores que o 20



20

variável  
temporária

30

47

63

101

115


Enquanto procuramos,  
deslocamos os elementos  
maiores que o 20



Encontramos a  
posição do 20

20      30      47                  63      101    115

Encontramos a  
posição do 20

- Funcionamento básico
- **Algoritmo em C *like*** 
- Análise dos número de comparações e movimentações
- Conclusão

# Exercício Resolvido (1)

- Uma dúvida básica sobre o operador AND pode prejudicar a compreensão do nosso algoritmo. Assim, o que será escrito na tela pelo programa abaixo?

```
class ExercicioDuvidaAND {  
    public static boolean m1(){  
        System.out.println("m1");  
        return false;  
    }  
    public static boolean m2(){  
        System.out.println("m2");  
        return true;  
    }  
    public static void main (String[] args) {  
        System.out.println("inicio");  
        boolean and = m1() && m2();  
        System.out.println("fim: " + and);  
    }  
}
```

## Exercício Resolvido (1)

- Uma dúvida básica sobre o operador AND pode prejudicar a compreensão do nosso algoritmo. Assim, o que será escrito na tela pelo programa abaixo?

```
class ExercicioDuvidaAND {  
    public static boolean m1(){  
        System.out.println("m1");  
        return false;  
    }  
    public static boolean m2(){  
        System.out.println("m2");  
        return true;  
    }  
    public static void main (String[] args) {  
        System.out.println("inicio");  
        boolean and = m1() && m2();  
        System.out.println("fim: " + and);  
    }  
}
```



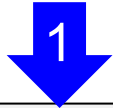
TELA
inicio m1 fim: false



Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j]; // Deslocamento  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

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

Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

(Obs.1):  $i$  começa com 1, pois quando temos um elemento, nosso conjunto está ordenado

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

# Algoritmo em C *like*

2

```
for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
```

(Obs.2): i aponta para o 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 = 1; i < n; i++) {  
    int tmp = array[i];  
    3 → int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

(Obs.3): j começa na maior posição do conjunto já ordenado e, no laço interno, ele é decrementado

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

Algoritmo em C *like*


```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    4 → while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

(Obs.4): O laço interno faz duas tarefas: i) procura a posição de inserção do novo elemento; e ii) desloca os elementos maiores que o novo elemento

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

Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

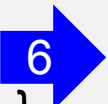


(Obs.5): A primeira cláusula do laço interno serve apenas para não acessarmos posições negativas na segunda cláusula

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

Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```



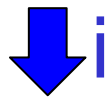
(Obs.6): Neste ponto, inserimos o novo elemento na posição correta

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

# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```



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

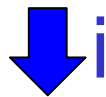


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

true:  $1 < 6$

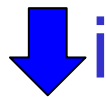


101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

115 tmp

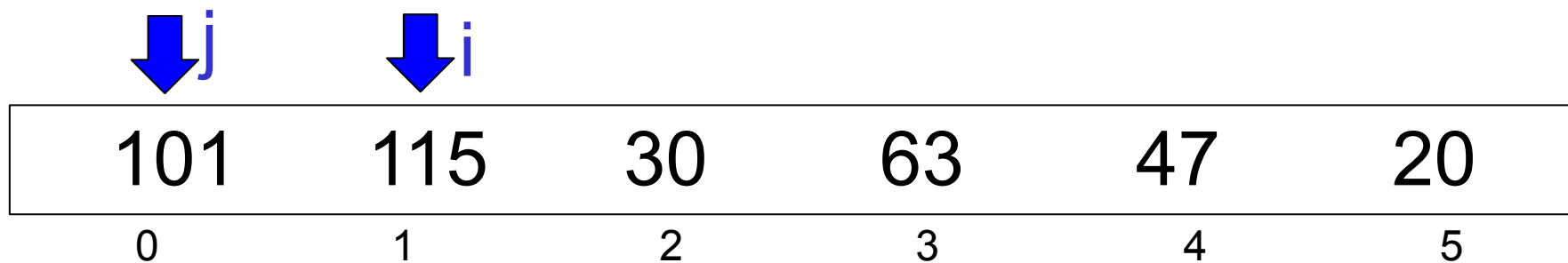


101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

115 tmp



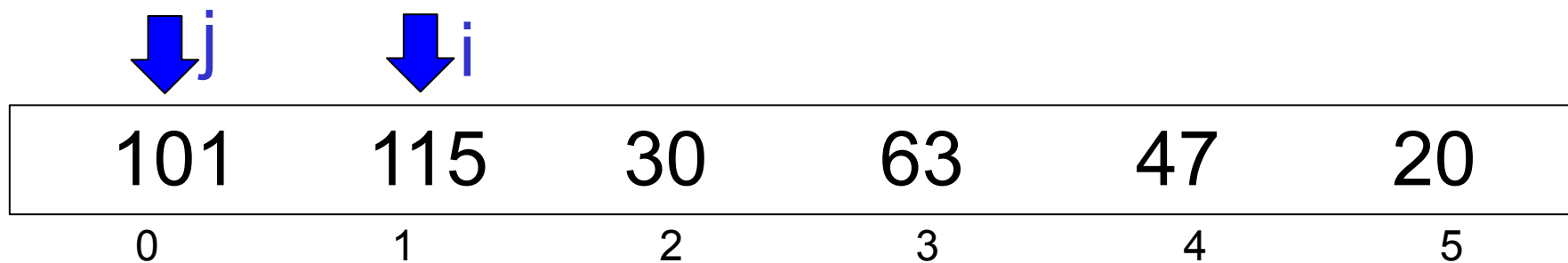
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

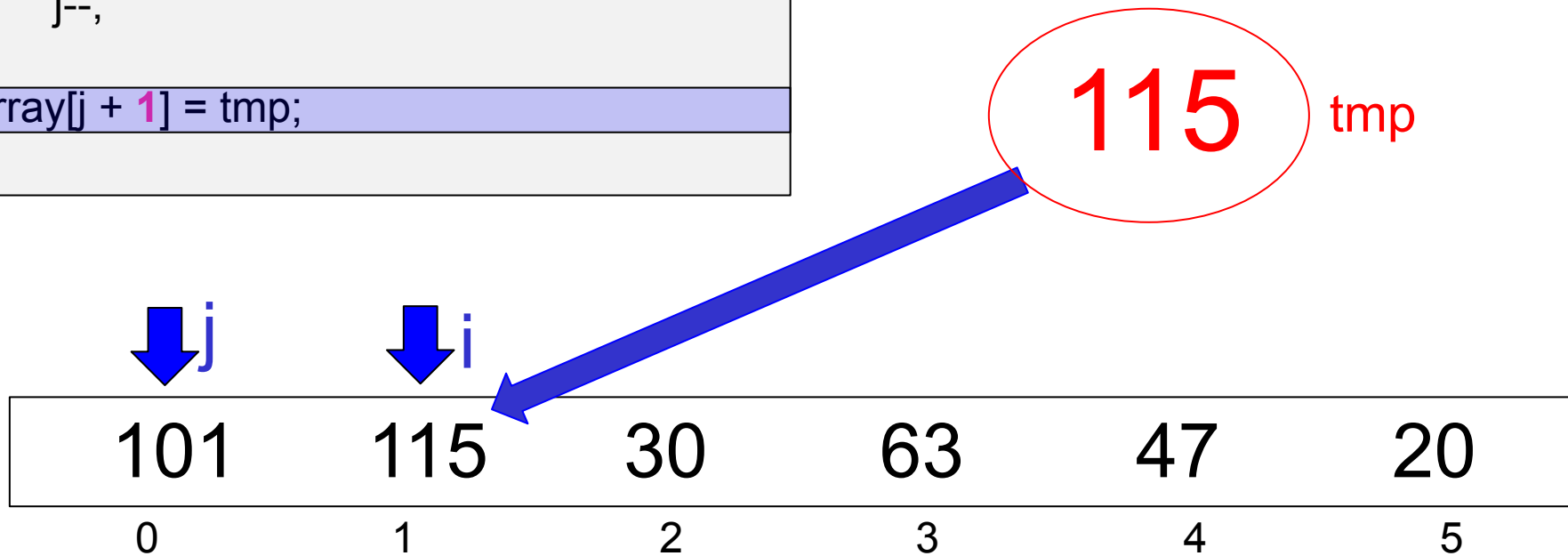
false:  $0 \geq 0 \ \&\& \ 101 > 115$

**115** tmp



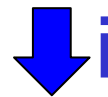
Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```



# Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
```

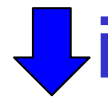


101	115	30	63	47	20
0	1	2	3	4	5

# Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
```

true:  $2 < 6$

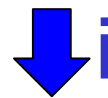


101	115	30	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

30 tmp



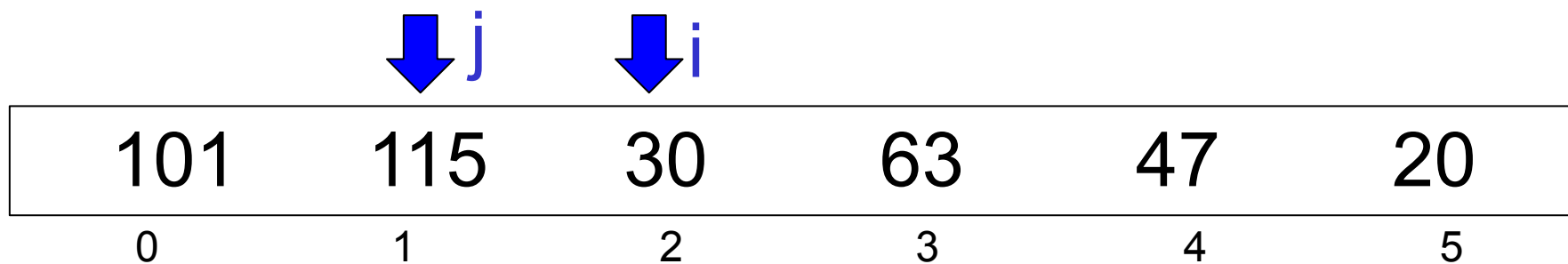
101	115	30	63	47	20
0	1	2	3	4	5



Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

30 tmp



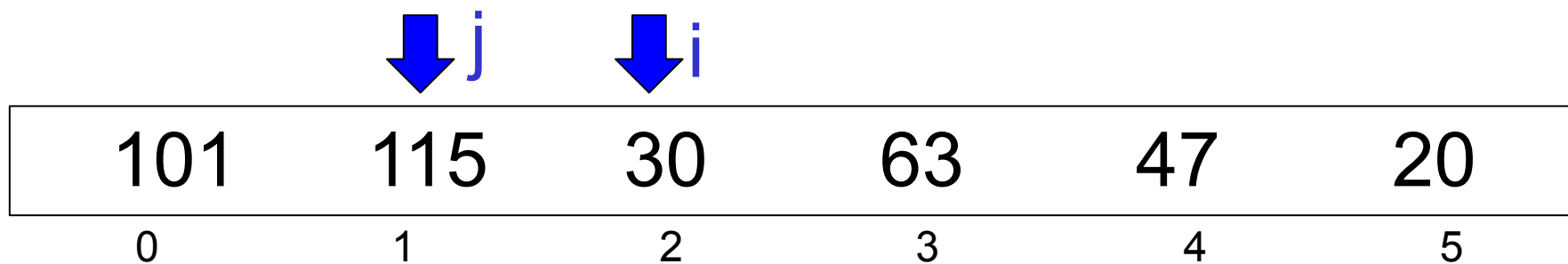
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

true:  $1 \geq 0 \ \&\& \ 115 > 30$

30 tmp

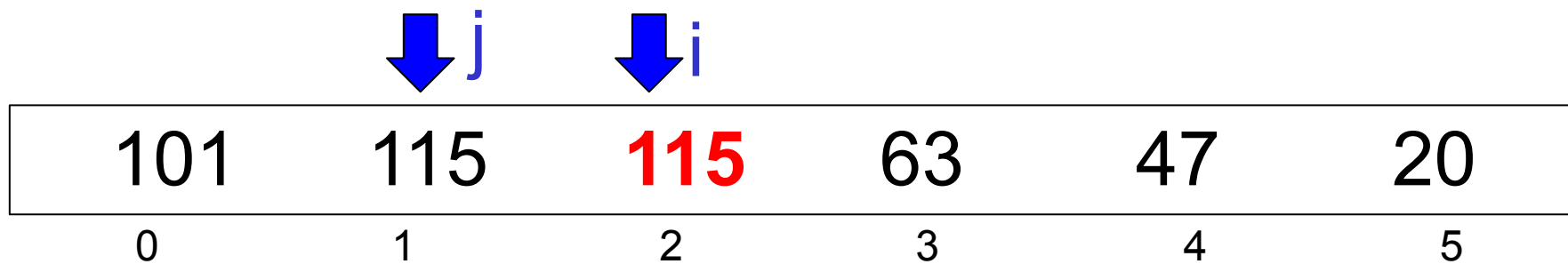


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

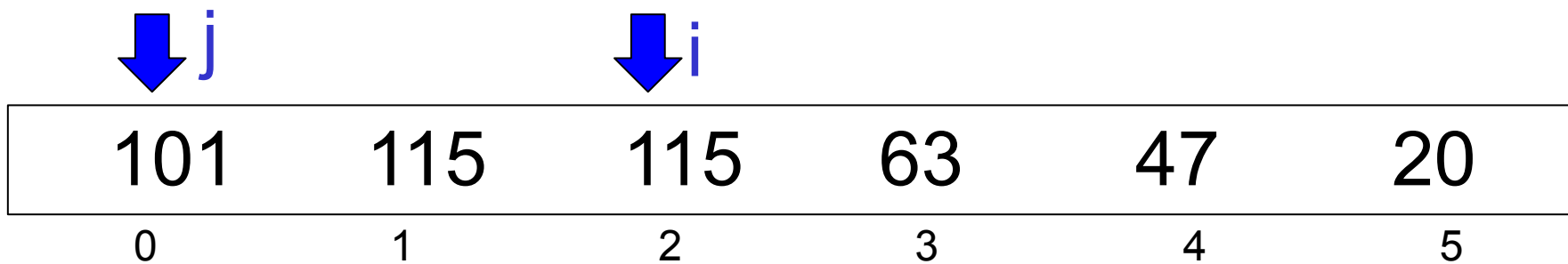
30 tmp



Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

30 tmp



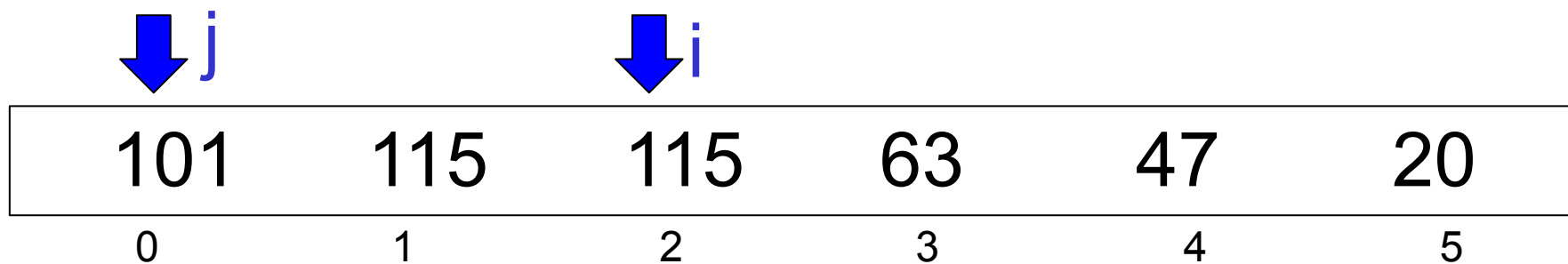
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

true:  $0 \geq 0 \ \&\& \ 101 > 30$

30 tmp

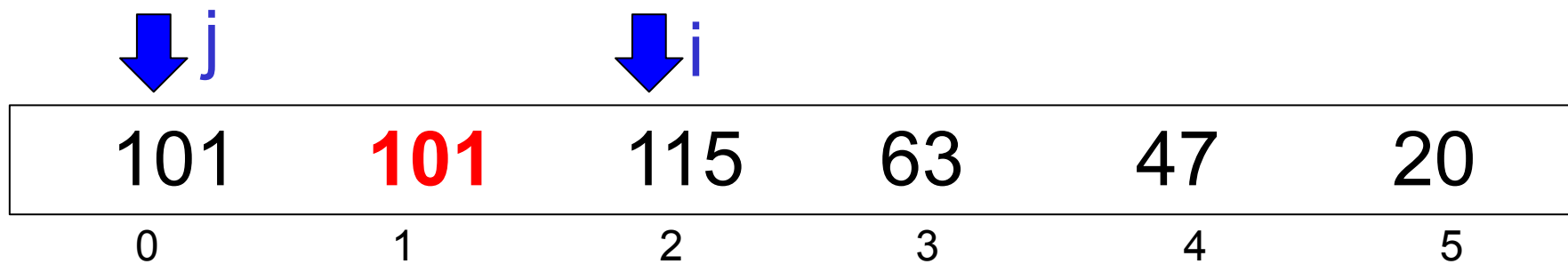


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

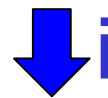
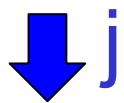
30 tmp



Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

30 tmp







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

# Algoritmo em C *like*

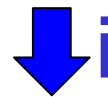
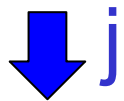
```
for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
```

Quando a 1ª cláusula é false,  
o AND não executa a segunda

false:  $-1 \geq 0$  &&   



30 tmp



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

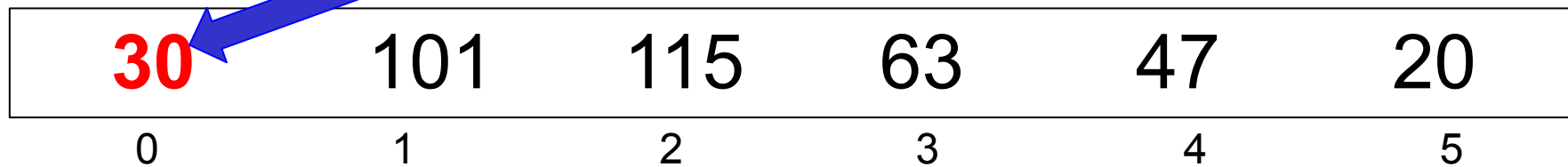


Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```


↓ j

↓ i



Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```




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

Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

true:  $3 < 6$

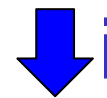


30	101	115	63	47	20
0	1	2	3	4	5

Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

63 tmp



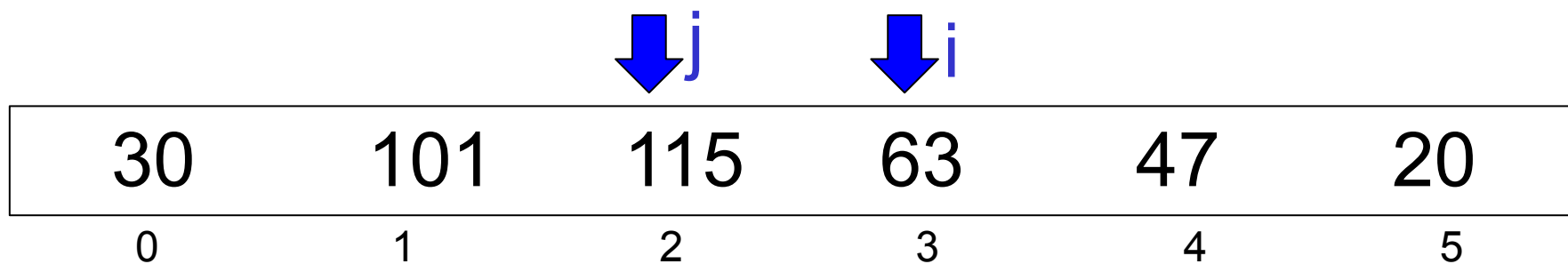
30	101	115	63	47	20
0	1	2	3	4	5

# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

63 tmp



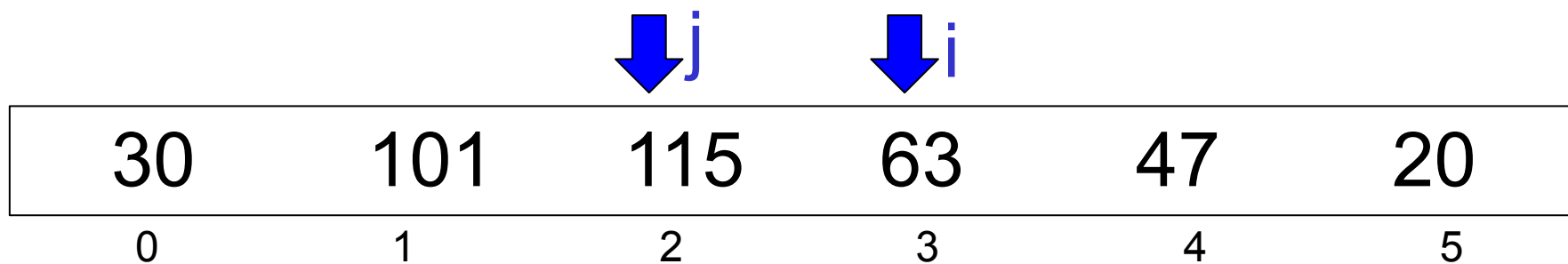
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

true:  $2 \geq 0 \ \&\& \ 115 > 63$

**63** tmp

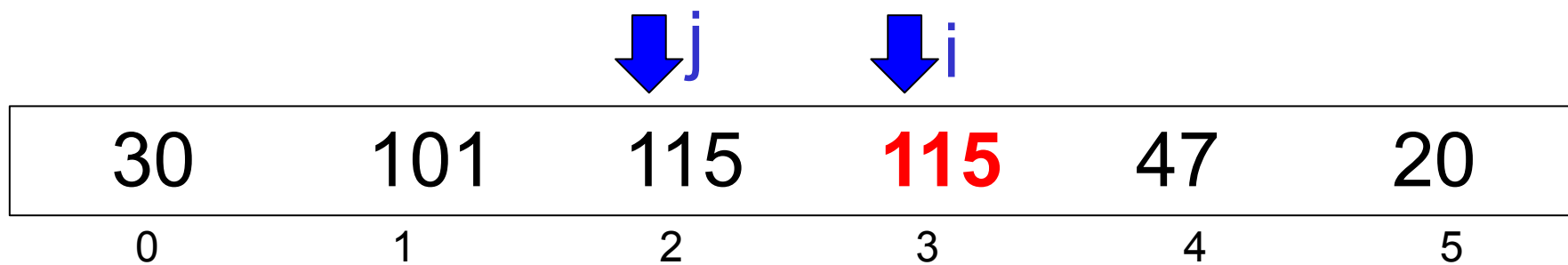


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

63 tmp

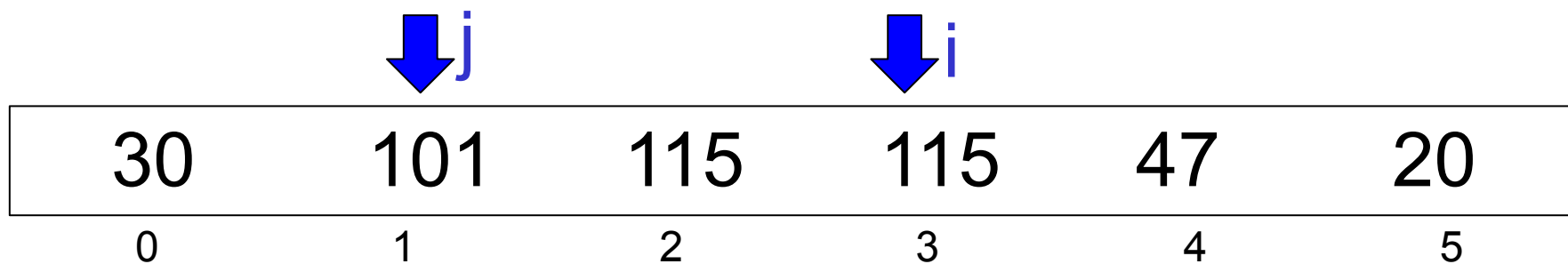


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

63 tmp





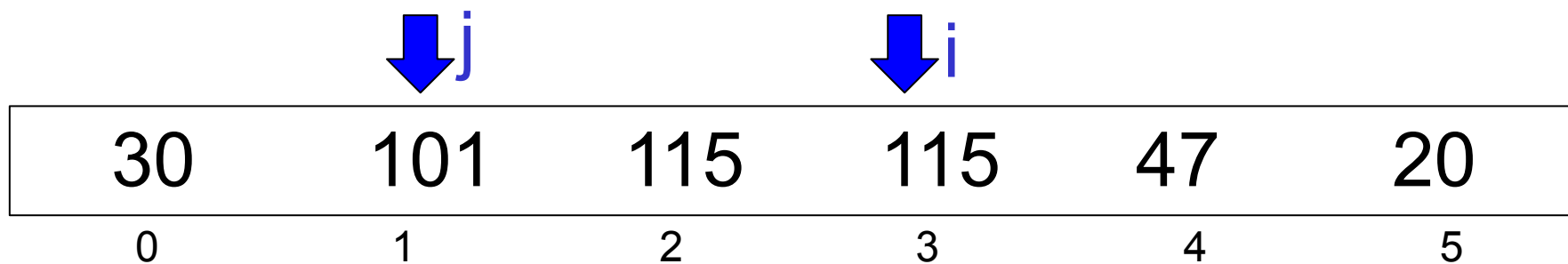
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

true:  $1 \geq 0 \ \&\& \ 101 > 63$

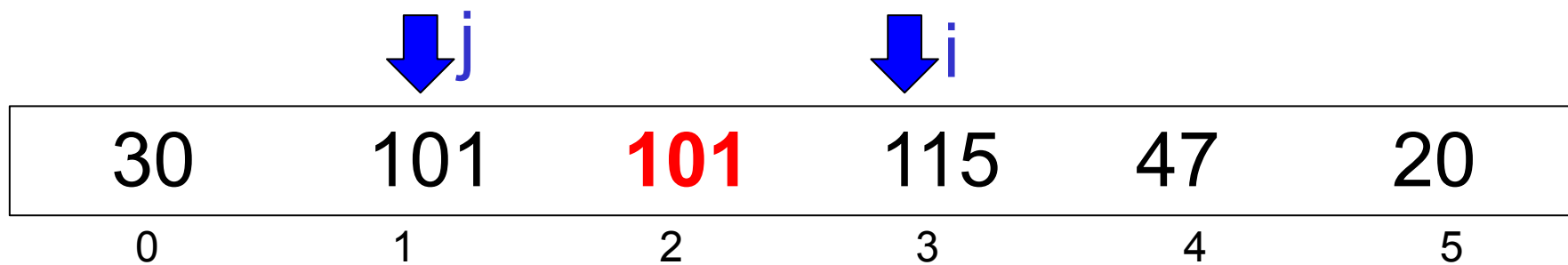
63 tmp



Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

63 tmp

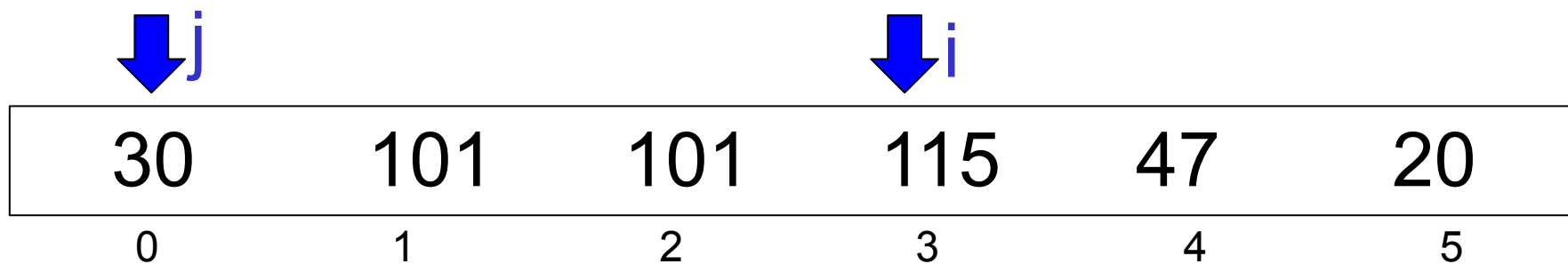


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

63 tmp



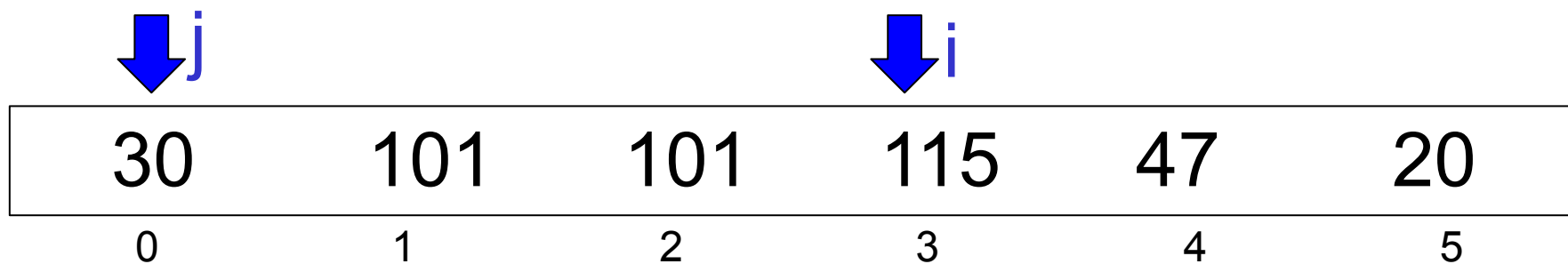
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

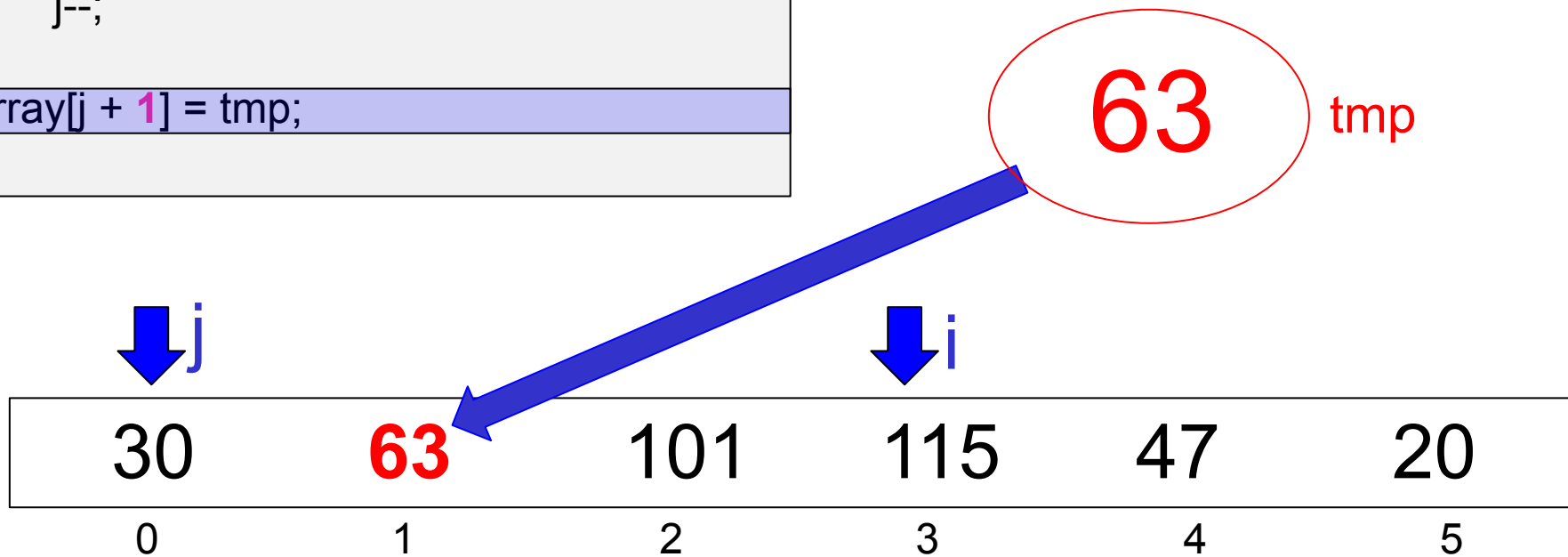
false:  $0 \geq 0 \ \&\& \ 30 > 63$

**63** tmp



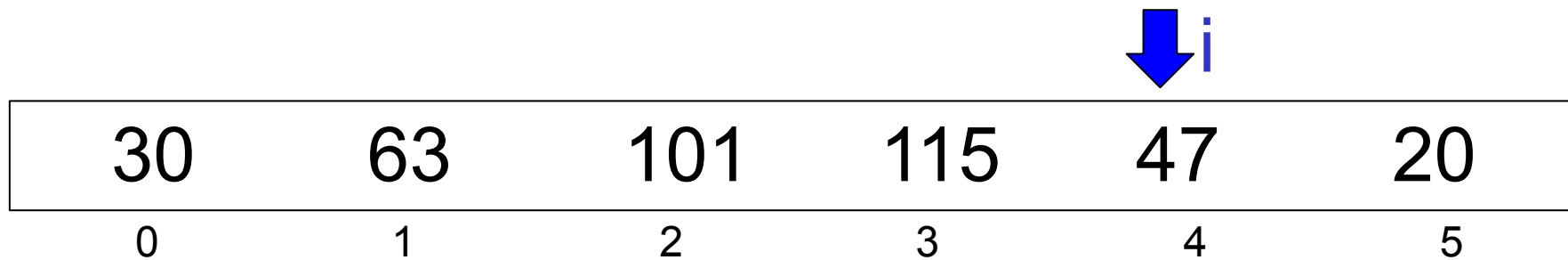
Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```




Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```



Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

true:  $4 < 6$ 

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

Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

47 tmp



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



# Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

47 tmp

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

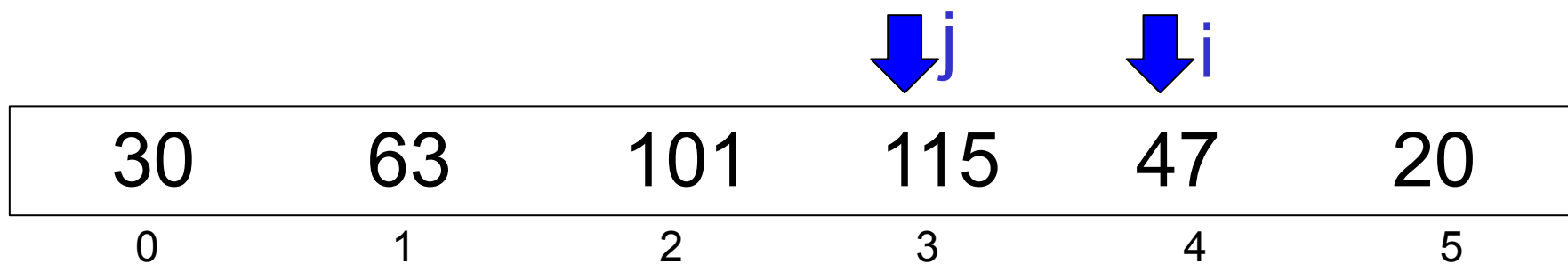
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

true:  $3 \geq 0 \ \&\& \ 115 > 47$

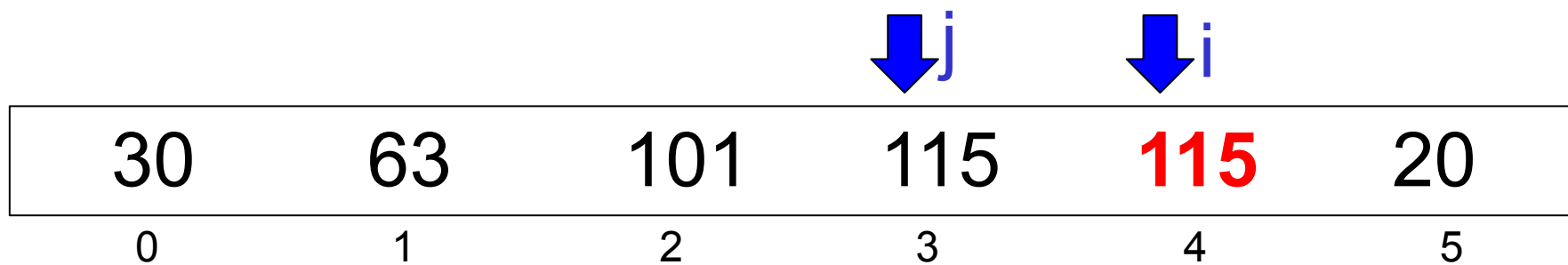
**47** tmp



Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

47 tmp

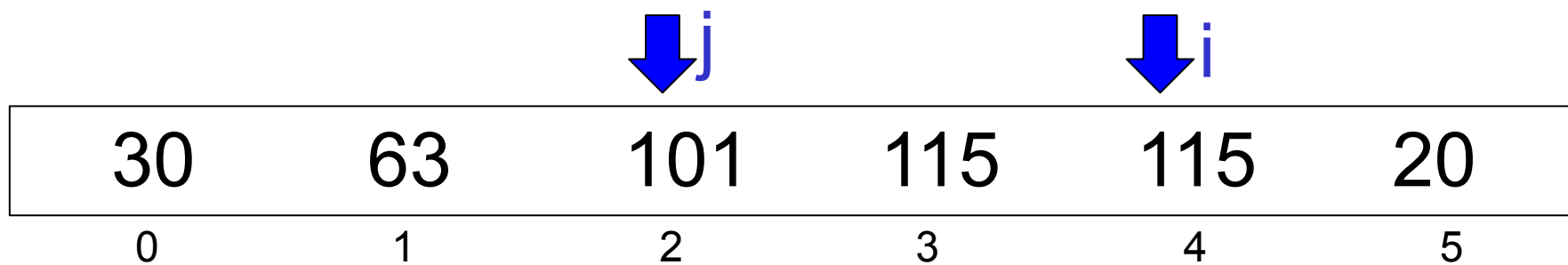


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

47 tmp



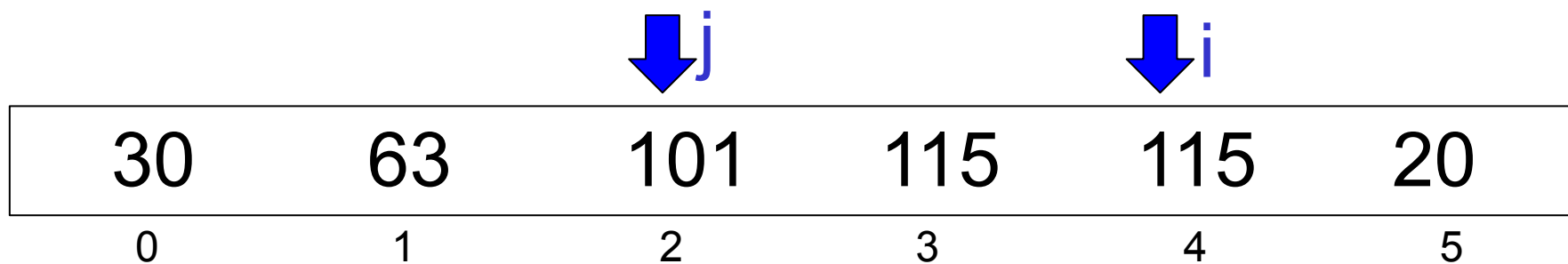
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

true:  $2 \geq 0 \ \&\& \ 101 > 47$

47 tmp

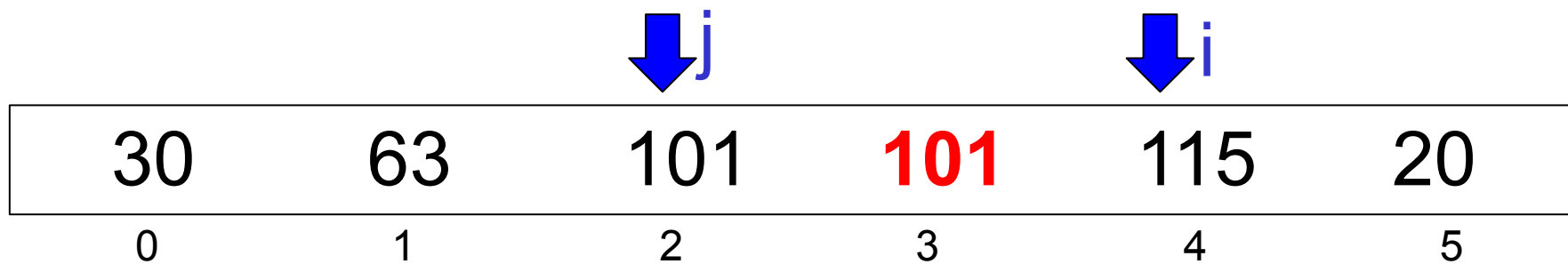


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

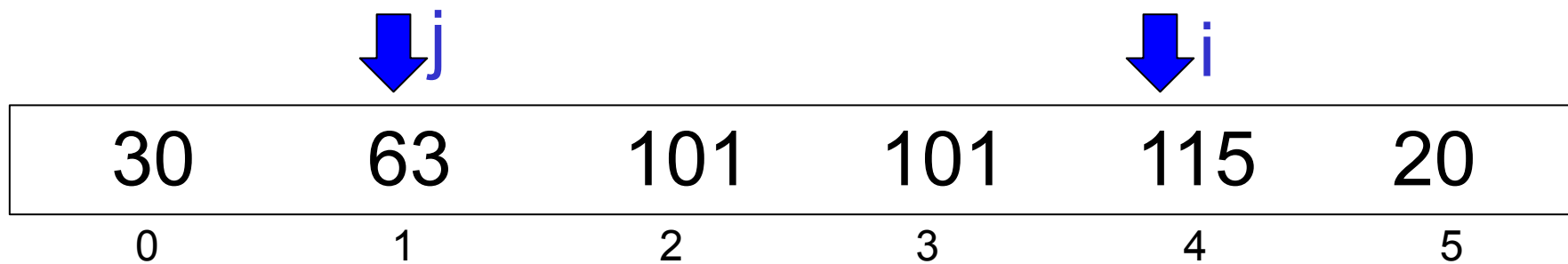
47 tmp



Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

47 tmp



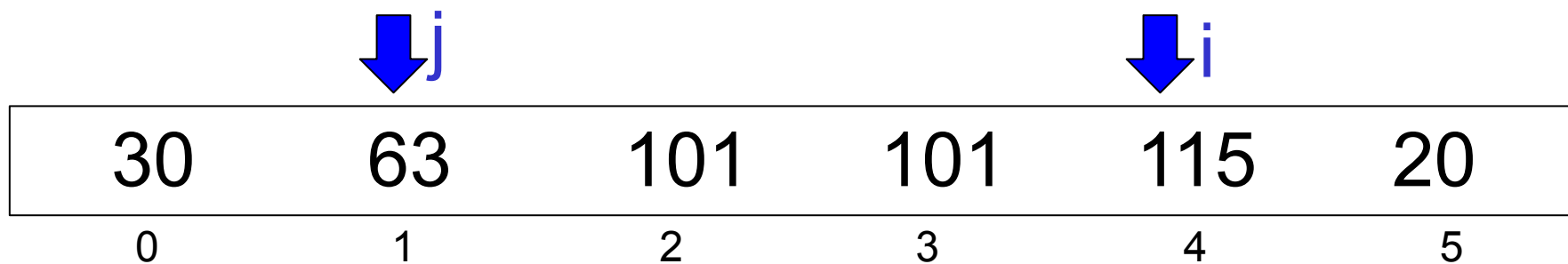
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

true:  $1 \geq 0 \ \&\& \ 63 > 47$

**47** tmp



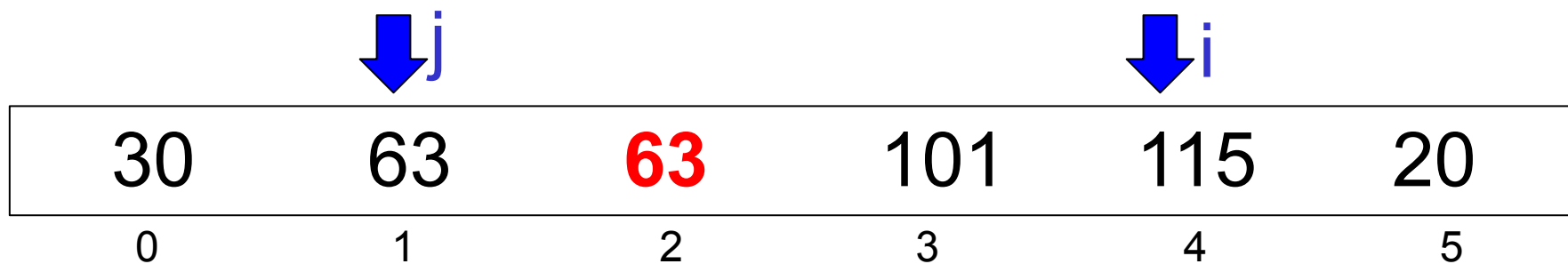


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

47 tmp

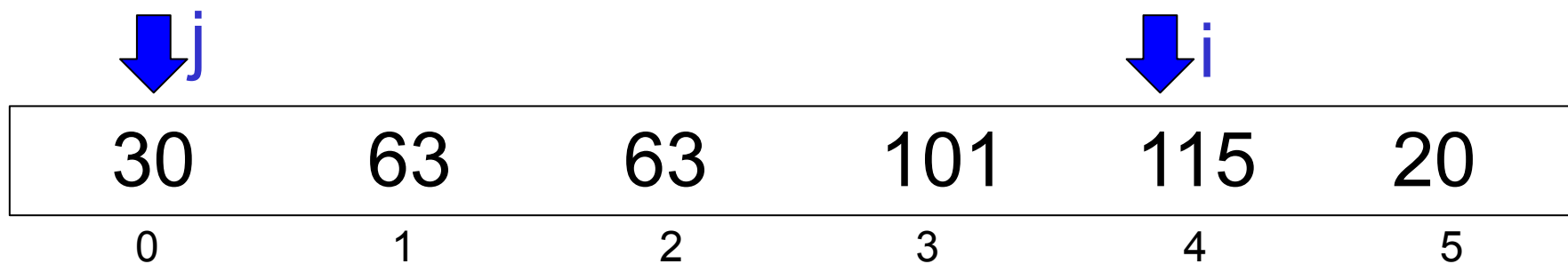


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

47 tmp



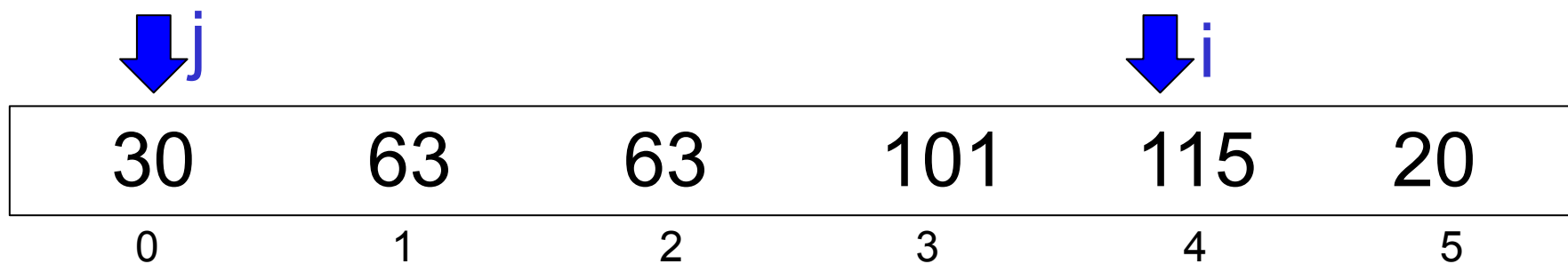
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

false:  $0 \geq 0 \ \&\& \ 30 > 47$

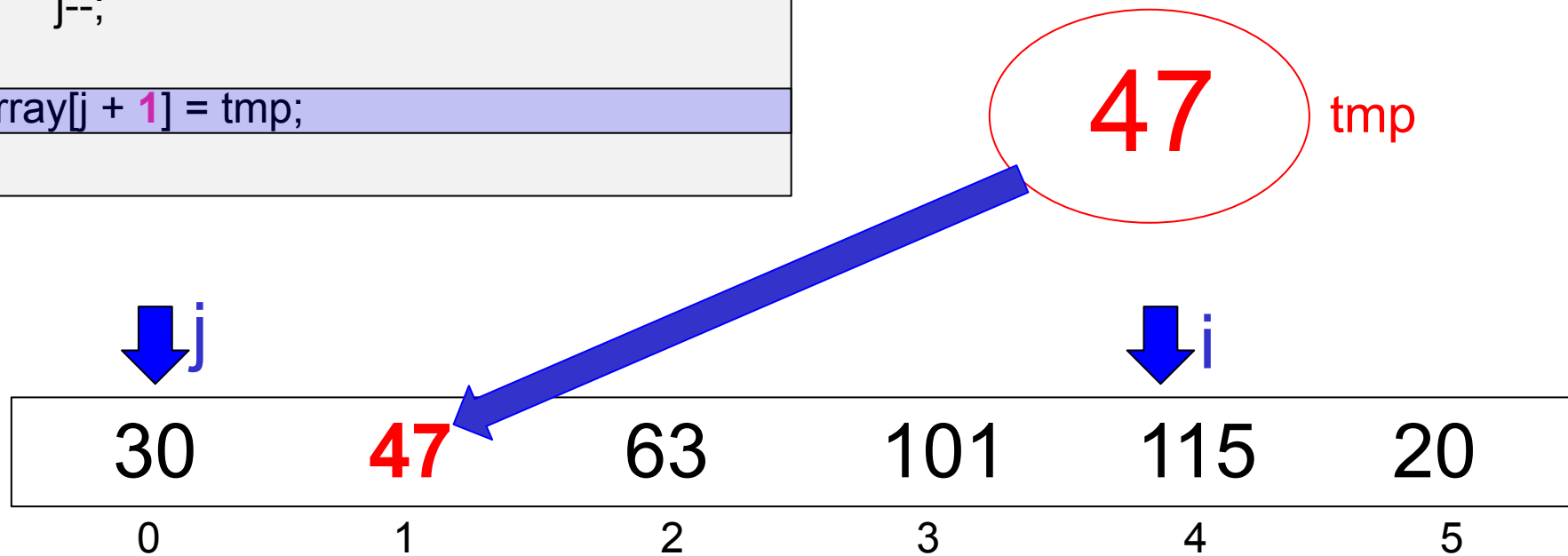
47 tmp



# Algoritmo em C *like*

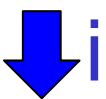
```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```



Algoritmo em C *like*


```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```



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

Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

true:  $5 < 6$ 

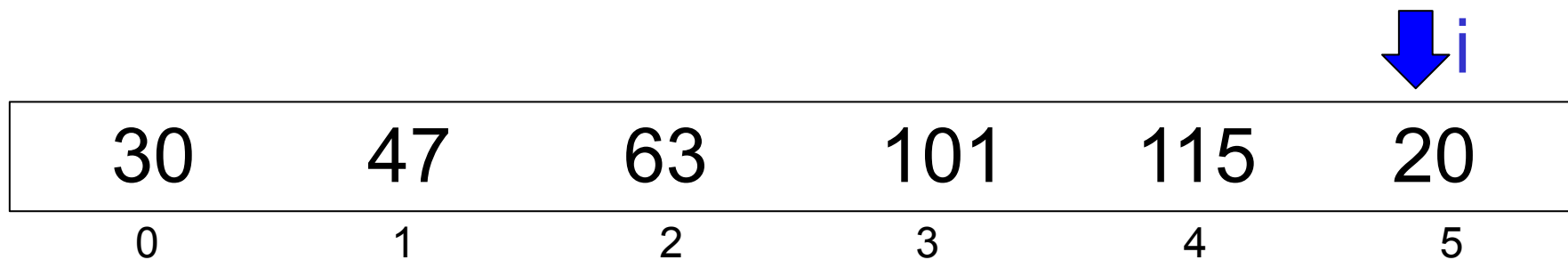
30	47	63	101	115	20
0	1	2	3	4	5

# Algoritmo em C *like*

```

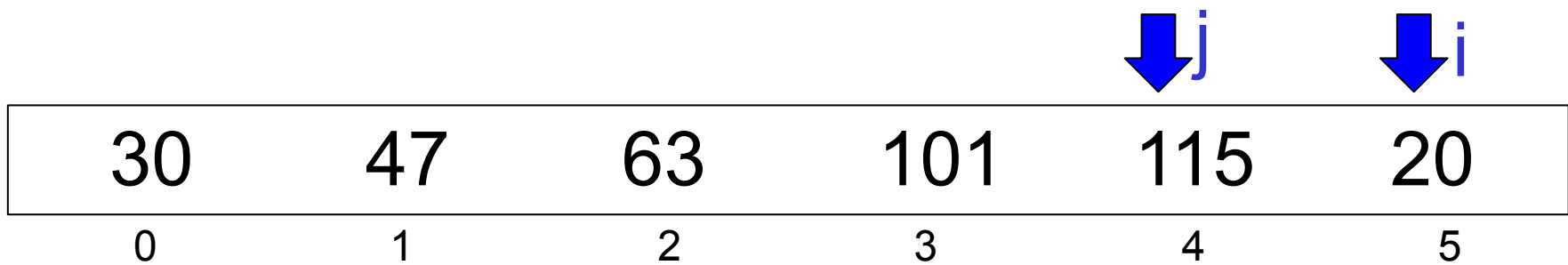
for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

20 tmp



# Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```





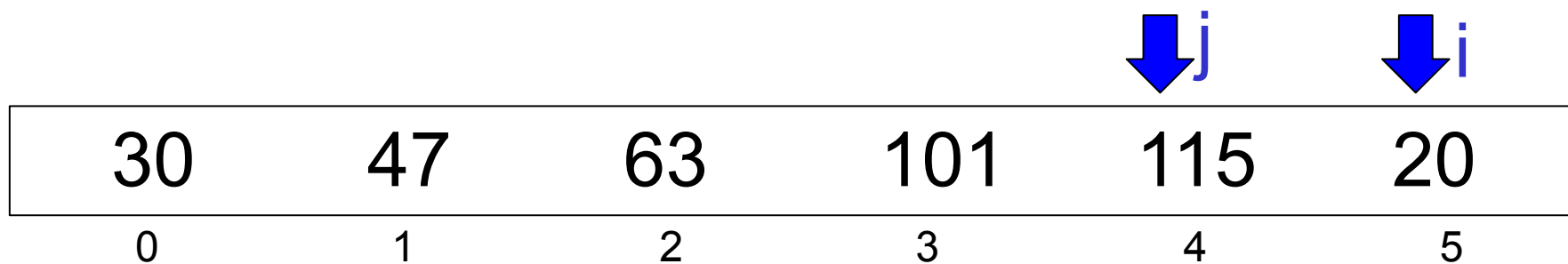
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

true:  $4 \geq 0 \ \&\& \ 115 > 20$

20 tmp

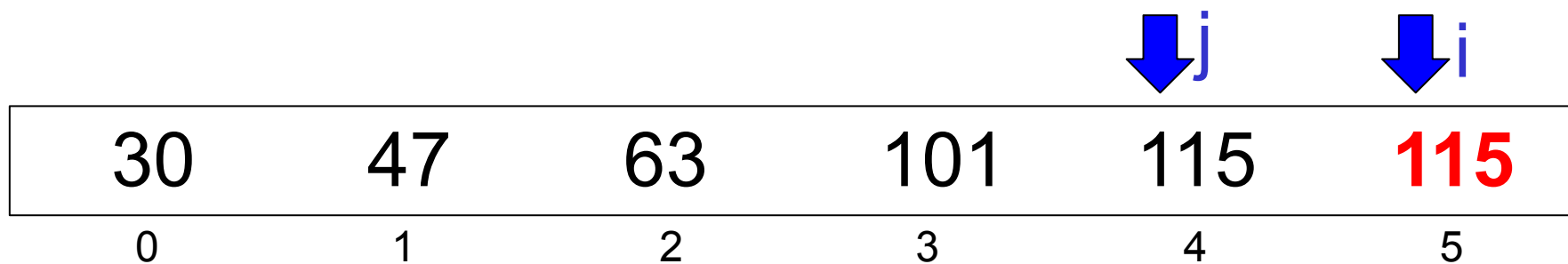


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

20 tmp

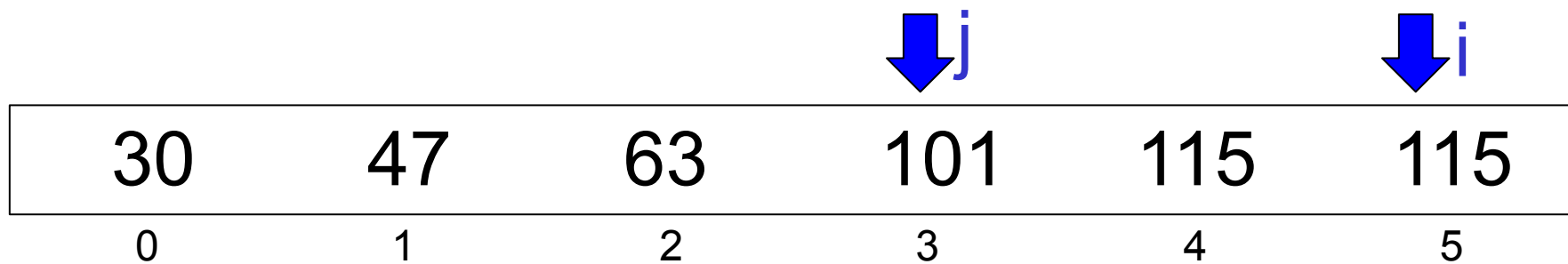


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

20 tmp

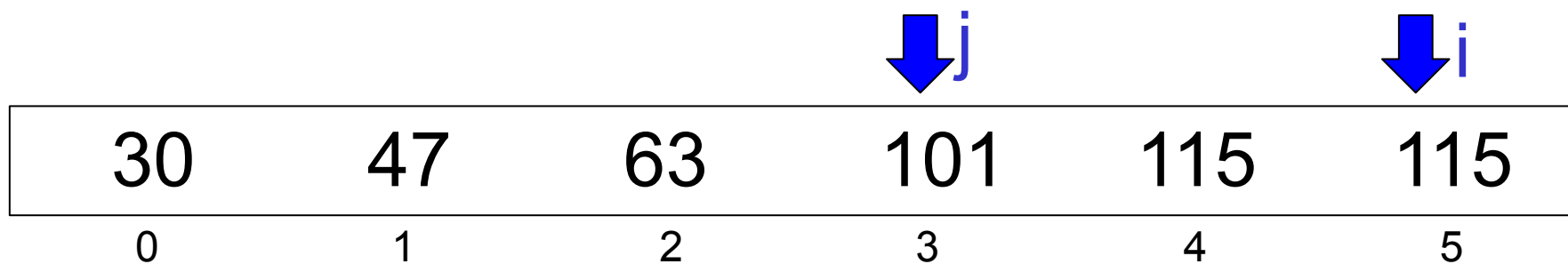


Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

true:  $3 \geq 0 \ \&\& \ 101 > 20$

20 tmp

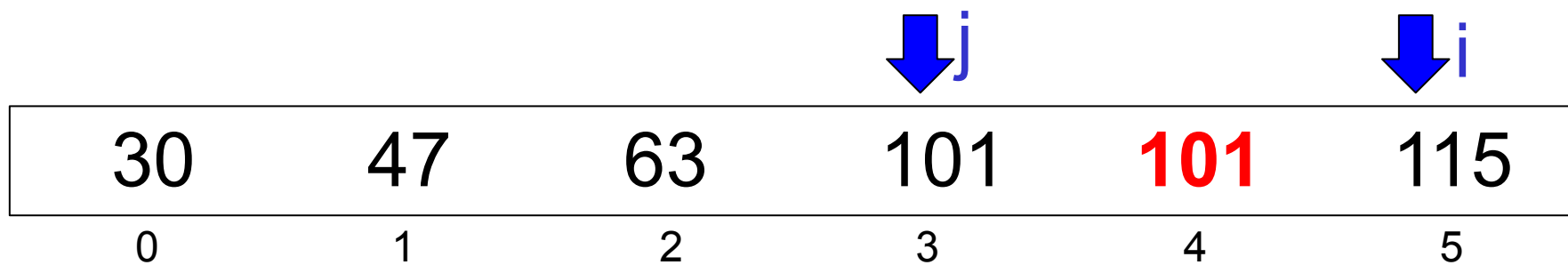


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

20 tmp

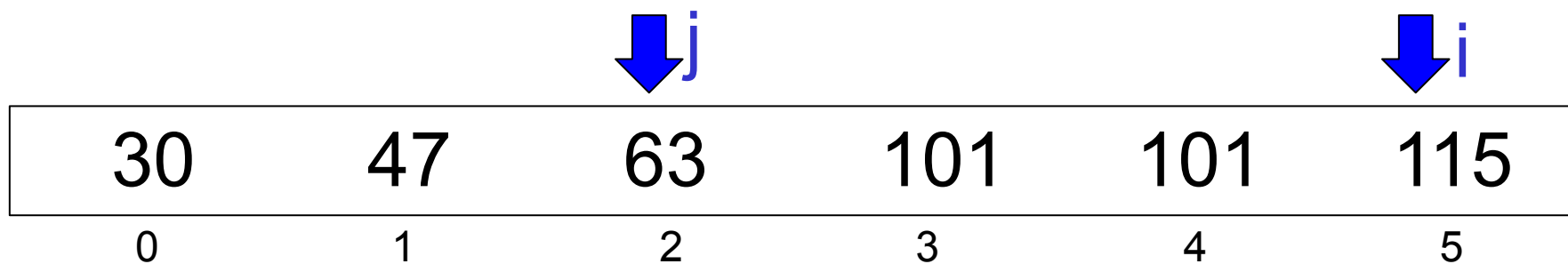


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

20 tmp



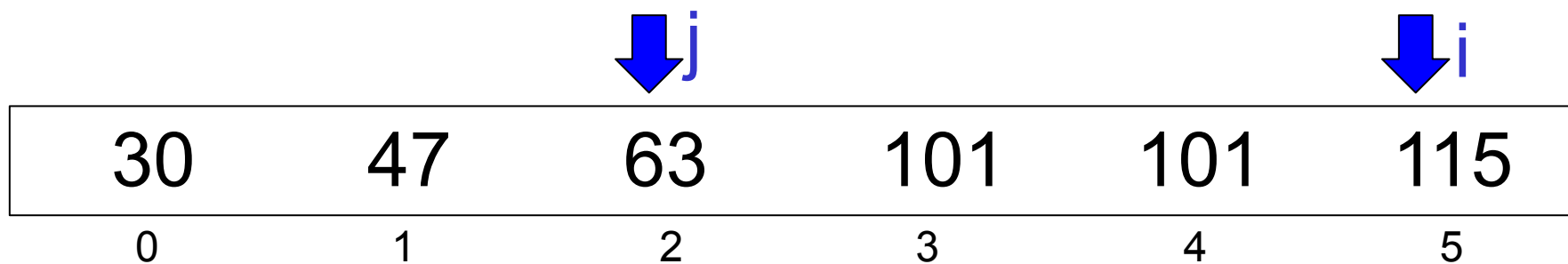
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

true:  $2 \geq 0 \ \&\& \ 63 > 20$

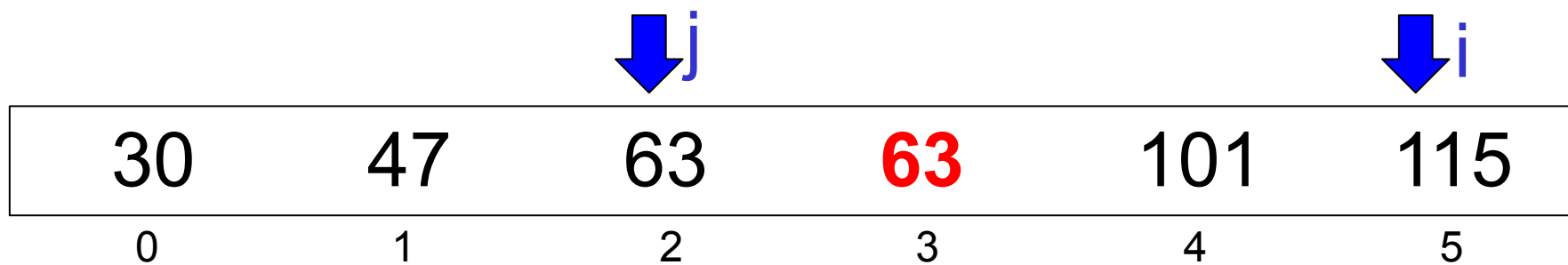
20 tmp



Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

20 tmp



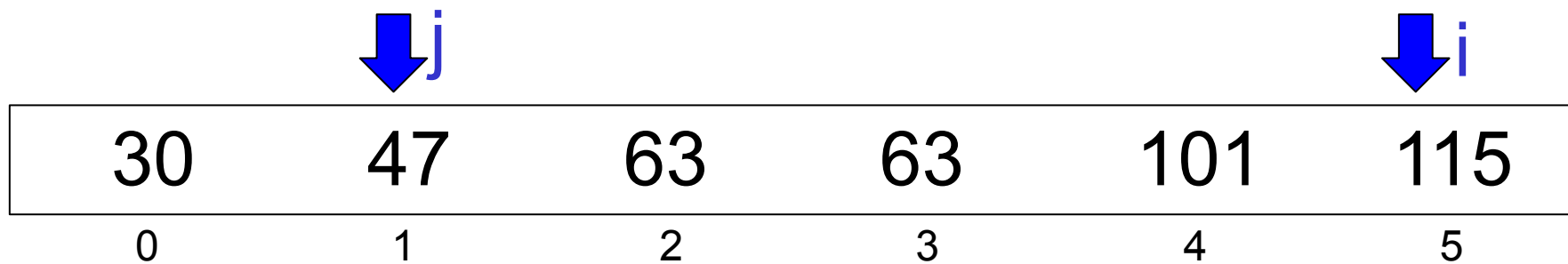


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

20 tmp

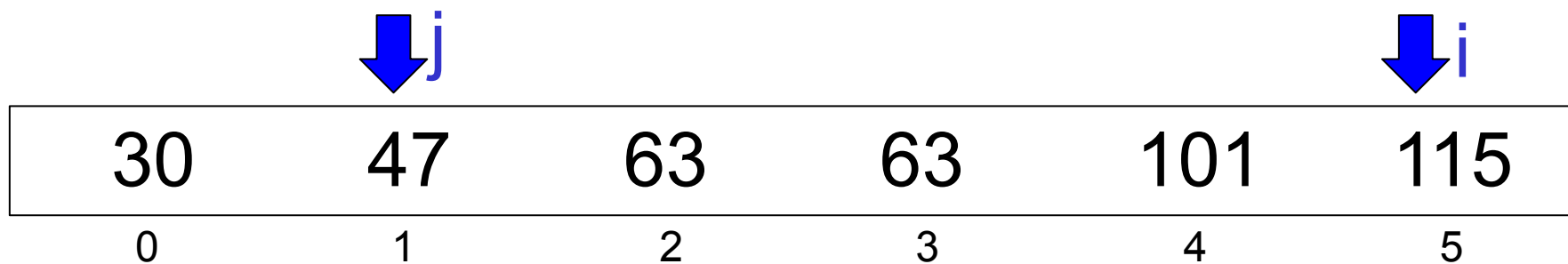


Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```

true:  $1 \geq 0 \ \&\& \ 47 > 20$

20 tmp

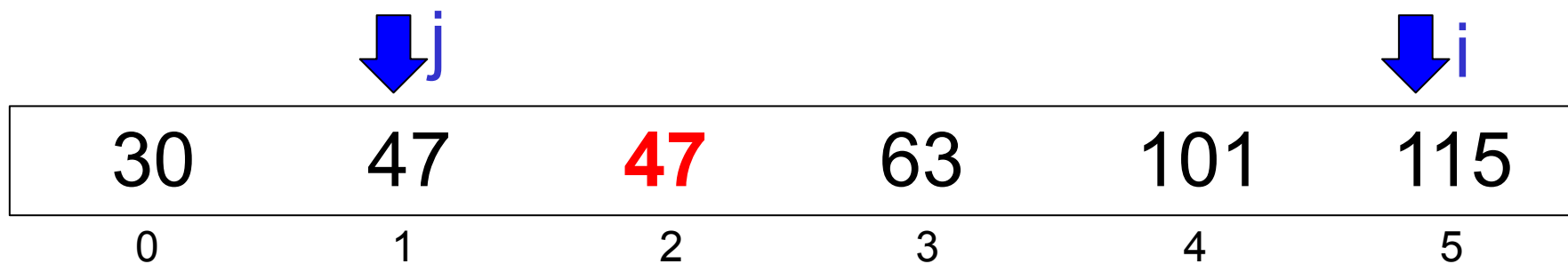


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

20 tmp

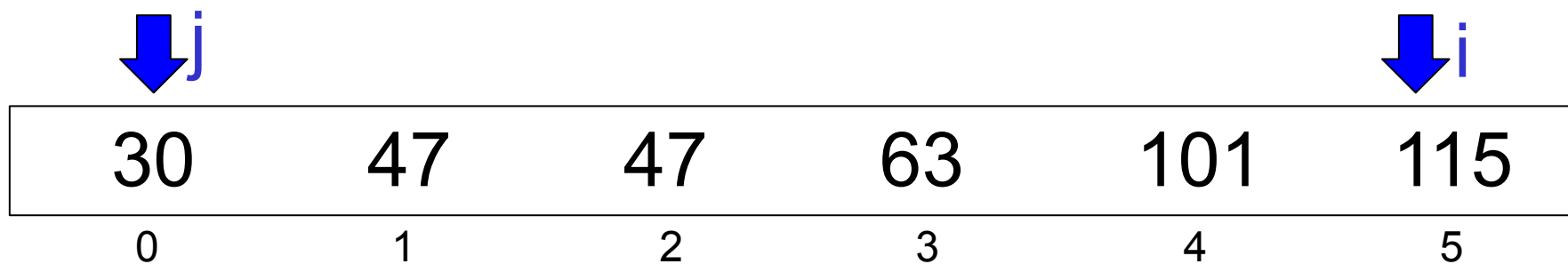


# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

20 tmp



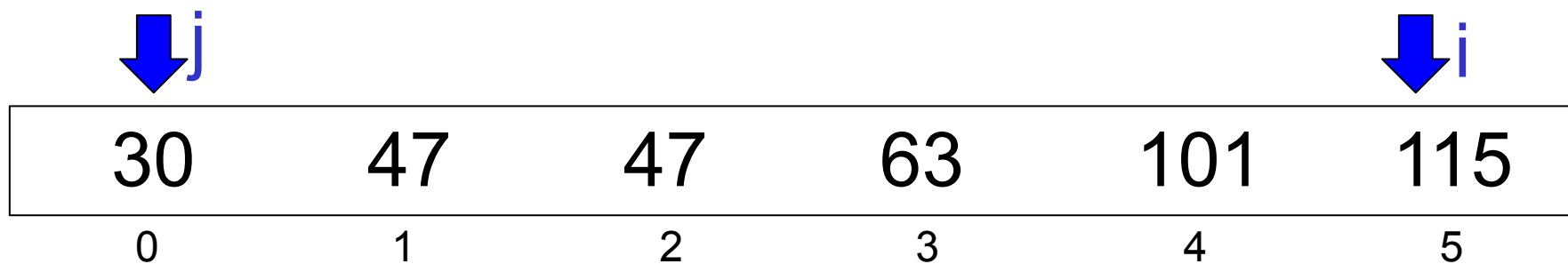
# Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

true:  $0 \geq 0 \ \&\& \ 30 > 20$

20 tmp

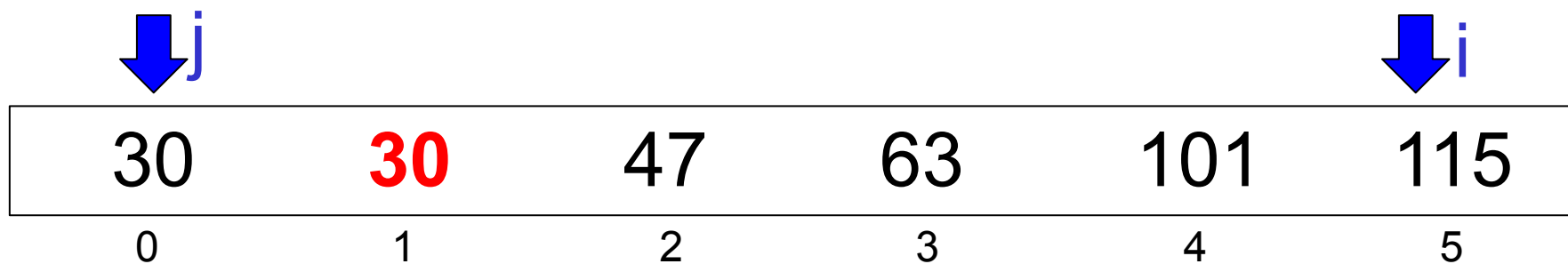


## Algoritmo em C *like*

```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

20 tmp

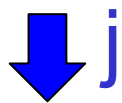


# Algoritmo em C *like*

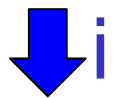
```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

20 tmp






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




# Algoritmo em C *like*

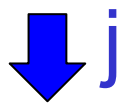
```

for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
    
```

false:  $-1 \geq 0$  &&   



20 tmp



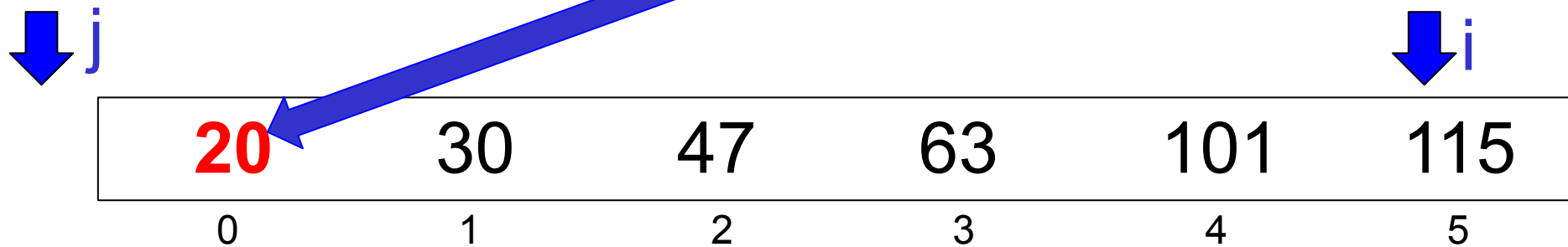
30	30	47	63	101	115
0	1	2	3	4	5





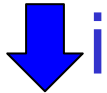
Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```



Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {  
    int tmp = array[i];  
    int j = i - 1;  
    while ( (j >= 0) && (array[j] > tmp) ){  
        array[j + 1] = array[j];  
        j--;  
    }  
    array[j + 1] = tmp;  
}
```



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


# Algoritmo em C *like*

```
for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
```

false:  $6 < 6$

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



- Funcionamento básico
- Algoritmo em C *like*
- **Análise dos número de comparações e movimentações** 
- Conclusão

## Exercício Resolvido (1)

- Mostre todas as comparações entre elementos do *array* para os *arrays* abaixo

a)

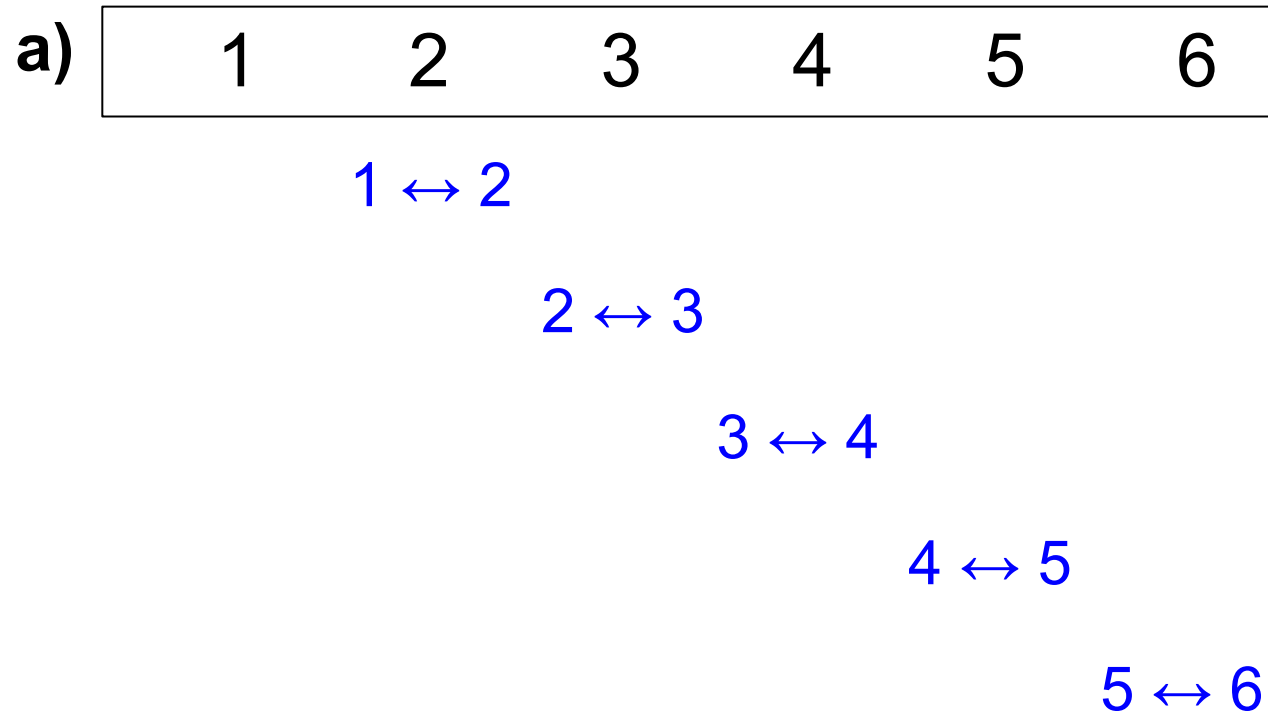
1	2	3	4	5	6
---	---	---	---	---	---

b)

6	5	4	3	2	1
---	---	---	---	---	---

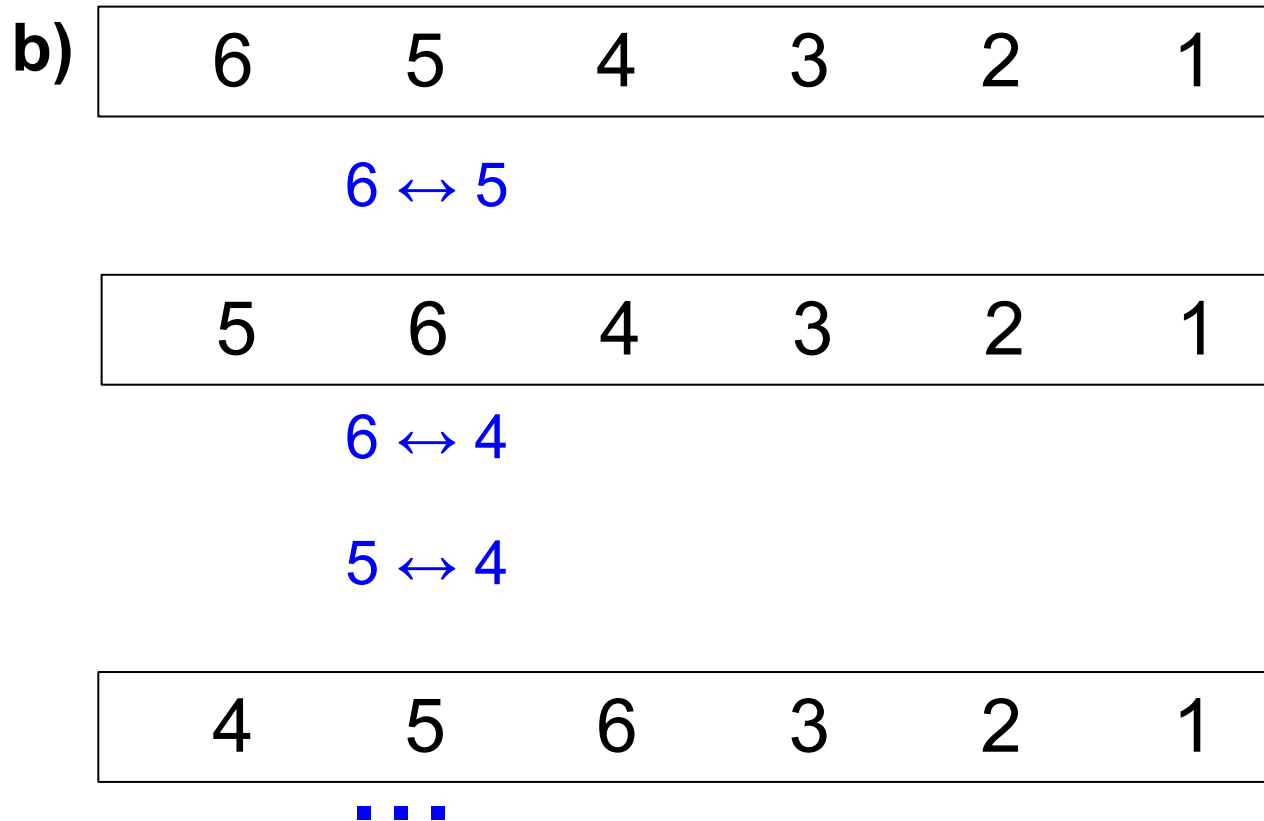
## Exercício Resolvido (1)

- Mostre todas as comparações entre elementos do *array* para os *arrays* abaixo



## Exercício Resolvido (1)

- Mostre todas as comparações entre elementos do *array* para os *arrays* abaixo



# Análise do Número de Comparações

- **Melhor caso:**

- Efetuamos uma comparação em cada iteração do laço externo
- Repetimos o laço externo  $(n - 1)$  vezes
- $C(n) = (n - 1) = \Theta(n)$

```
1:  for (int i = 1; i < n; i++) {  
2:      int tmp = array[i];  
3:      int j = i - 1;  
4:      while ( (j >= 0) && (array[j] > tmp) ){  
5:          array[j + 1] = array[j];  
6:          j--;  
7:      }  
8:      array[j + 1] = tmp;  
9:  }
```



# Análise do Número de Comparações

- **Pior caso:**

- Efetuamos  $i$  comparações em cada iteração do laço externo
- Repetimos o laço externo  $(n - 1)$  vezes
- $$C(n) = 1 + 2 + 3 + \dots + (n-1) = \sum_{0 \leq i \leq (n-1)} i = \frac{(n-1)*n}{2} = \Theta(n^2)$$

```
1:  for (int i = 1; i < n; i++) {  
2:      int tmp = array[i];  
3:      int j = i - 1;  
4:      while ( (j >= 0) && (array[j] > tmp) ){  
5:          array[j + 1] = array[j];  
6:          j--;  
7:      }  
8:      array[j + 1] = tmp;  
9:  }
```

# Análise do Número de Movimentações

- O Inserção movimenta elementos nas linhas 2, 5 e 8
- Número de movimentações no laço interno é o de comparações menos um
- Cada iteração do laço externo tem as movimentações do interno mais duas
- $M_i(n) = (C_i(n) - 1) + 2 \Rightarrow$

$$M_i(n) = C_i(n) + 1$$

```
1:  for (int i = 1; i < n; i++) {  
2:      int tmp = array[i];  
3:      int j = i - 1;  
4:      while ( (j >= 0) && (array[j] > tmp) ){  
5:          array[j + 1] = array[j];  
6:          j--;  
7:      }  
8:      array[j + 1] = tmp;  
9:  }
```

# Análise do Número de Movimentações

- Sendo  $M_i(n) = C_i(n) + 1$ , no **melhor caso**, temos:
  - $C(n) = 1 + 1 + 1 + \dots + 1$ ,  $n-1$  vezes  $= (n-1)$
  - $M(n) = 2 + 2 + 2 + \dots + 2$ ,  $n-1$  vezes  $= 2(n-1) = \Theta(n)$

# Análise do Número de Movimentações

• Sendo  $M_i(n) = C_i(n) + 1$ , no **pior caso**, temos:

- $C(n) = 1 + 2 + 3 + \dots + (n-1) = \sum_{0 \leq i \leq (n-1)} i = \frac{(n-1)*n}{2}$
- $M(n) = 2 + 3 + 4 + \dots + ((n-1)+1)$

# Análise do Número de Movimentações

• Sendo  $M_i(n) = C_i(n) + 1$ , no **pior caso**, temos:

$$\circ \quad C(n) = 1 + 2 + 3 + \dots + (n-1) = \sum_{0 \leq i \leq (n-1)} i = \frac{(n-1)*n}{2}$$

$$\circ \quad M(n) = 2 + 3 + 4 + \dots + (n) \Rightarrow$$

$$M(n) = \frac{(1+1)}{2} + \frac{(2+1)}{3} + \frac{(3+1)}{4} + \dots + \frac{((n-1)+1)}{(n)}$$

# Análise do Número de Movimentações

• Sendo  $M_i(n) = C_i(n) + 1$ , no **pior caso**, temos:

$$C(n) = 1 + 2 + 3 + \dots + (n-1) = \sum_{0 \leq i \leq (n-1)} i = \frac{(n-1)*n}{2}$$

$$M(n) = 2 + 3 + 4 + \dots + (n-1) \Rightarrow$$

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

$$M(n) = 1 + 2 + 3 + \dots + (n-1) + \underbrace{1 + 1 + 1 + \dots + 1}_{(n-1) \text{ vezes}}$$

# Análise do Número de Movimentações

• Sendo  $M_i(n) = C_i(n) + 1$ , no **pior caso**, temos:

$$\circ \quad C(n) = 1 + 2 + 3 + \dots + (n-1) = \sum_{0 \leq i \leq (n-1)} i = \frac{(n-1)*n}{2}$$

$$\circ \quad M(n) = 2 + 3 + 4 + \dots + (n-1) \Rightarrow$$

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

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

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

# Análise do Número de Movimentações

• Sendo  $M_i(n) = C_i(n) + 1$ , no **pior caso**, temos:

$$\circ \quad C(n) = 1 + 2 + 3 + \dots + (n-1) = \sum_{0 \leq i \leq (n-1)} i = \frac{(n-1)*n}{2}$$

$$\circ \quad M(n) = 2 + 3 + 4 + \dots + (n-1) \Rightarrow$$

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

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

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

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



# Análise do Número de Movimentações

• Sendo  $M_i(n) = C_i(n) + 1$ , no **pior caso**, temos:

$$\circ \quad C(n) = 1 + 2 + 3 + \dots + (n-1) = \sum_{0 \leq i \leq (n-1)} i = \frac{(n-1)*n}{2}$$

$$\circ \quad M(n) = 2 + 3 + 4 + \dots + (n-1) \Rightarrow$$

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

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

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

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

$$M(n) = -1 + \sum_{0 \leq i \leq n} i$$

# Análise do Número de Movimentações

• Sendo  $M_i(n) = C_i(n) + 1$ , no **pior caso**, temos:

$$\circ \quad C(n) = 1 + 2 + 3 + \dots + (n-1) = \sum_{0 \leq i \leq (n-1)} i = \frac{(n-1)*n}{2}$$

$$\circ \quad M(n) = 2 + 3 + 4 + \dots + (n-1) \Rightarrow$$


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

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

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

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

$$M(n) = -1 + \sum_{0 \leq i \leq n} i = \frac{n(n+1) - 2}{2} = \Theta(n^2)$$

- Funcionamento básico
- Algoritmo em C *like*
- Análise dos número de comparações e movimentações
- **Conclusão** 

# Conclusão

- Melhor caso (comparações e movimentações) – *array* ordenado
- Pior caso (comparações e movimentações) – ordem decrescente
- Método a ser utilizado quando o *array* estiver “quase” ordenado
  - Boa opção se desejarmos adicionar alguns itens em um *array* ordenado porque seu custo será linear
- Algoritmo 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)

- Uma forma de melhorar o Algoritmo de Inserção é considerar a pesquisa binária para procurar a posição em que o novo elemento será inserido na lista ordenada. Nesse caso, realizamos  $\Theta(\lg m)$  comparações, onde  $m$  é o tamanho da lista ordenada, para encontrar a posição de inserção. Em seguida, em uma estrutura de repetição, movemos em uma unidade todos os elementos já ordenados e cuja posição é maior ou igual a de inserção. Implemente o Algoritmo de Inserção com Pesquisa Binária.

## Exercício (3)

- Quando os elementos estão ordenados de forma decrescente tanto o Seleção como o Inserção realizam  $C(n) = \frac{(n-1)(n)}{2} = \frac{n^2}{2} - \frac{n}{2}$  comparações. Nesse caso, qual dos dois algoritmos executará mais rápido? Justifique sua resposta