Unidade IV:

Ordenação Interna - Algoritmo de Inserção

Prof. Max do Val Machado

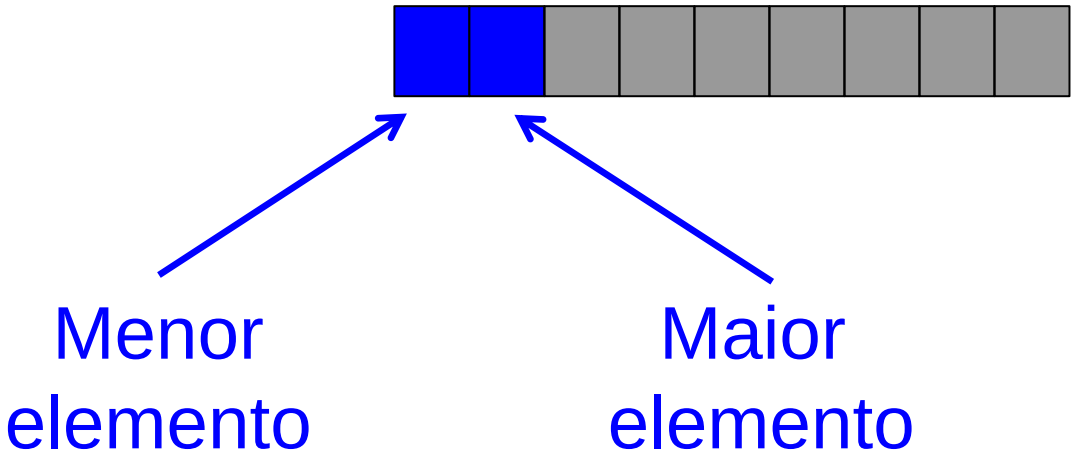
Instituto de Ciências Exatas e Informática
Curso de Ciência da Computação

Funcionamento Básico

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

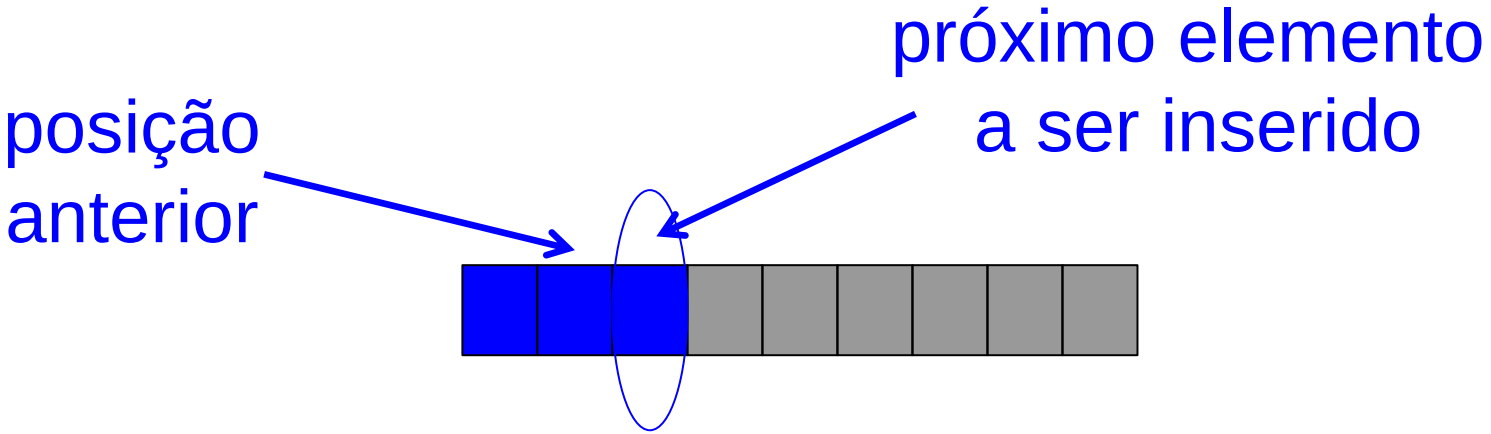
Funcionamento Básico

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



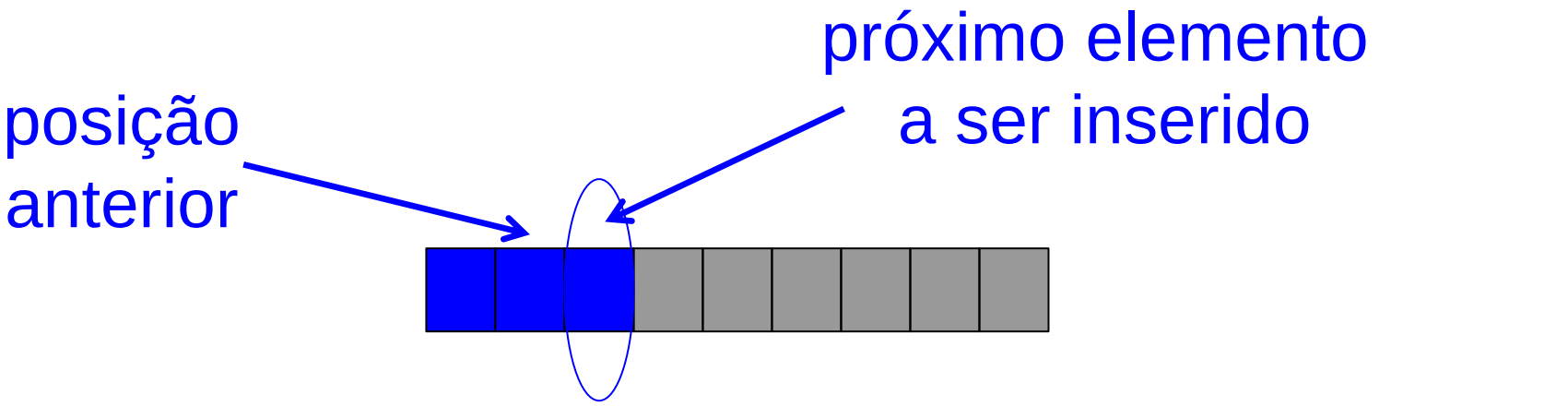
Funcionamento Básico

- Temos duas sequência (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

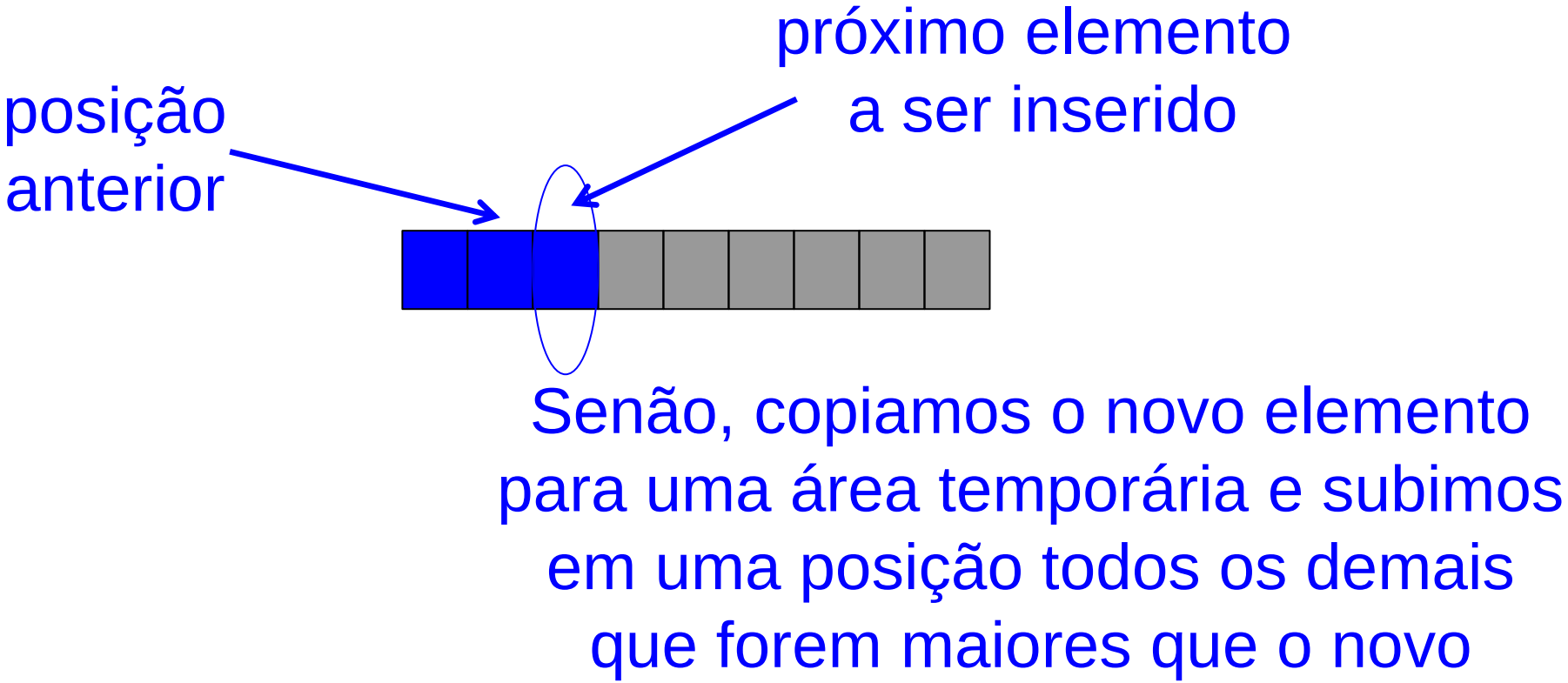
- Temos duas sequência (a ordenada e a ordenar) e, em cada passo, aumentamos a ordenada com um elemento que deve ser inserido em sua posição correta (ordenada)



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

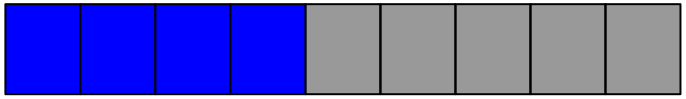
Funcionamento Básico

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



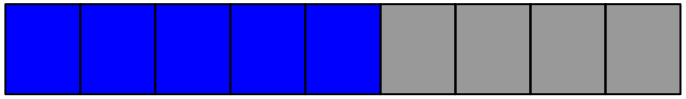
Funcionamento Básico

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



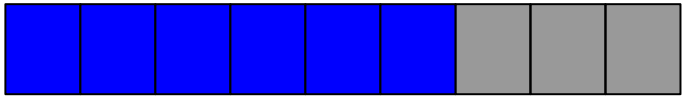
Funcionamento Básico

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



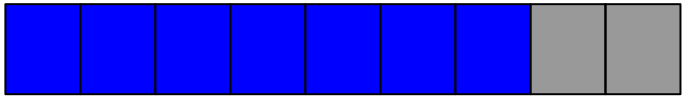
Funcionamento Básico

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



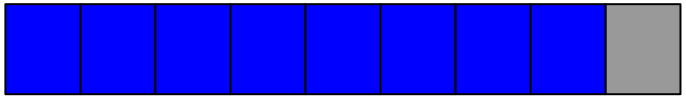
Funcionamento Básico

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



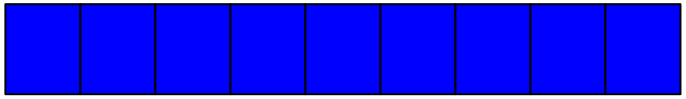
Funcionamento Básico

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



Exemplo

101 115 30 63 47 20

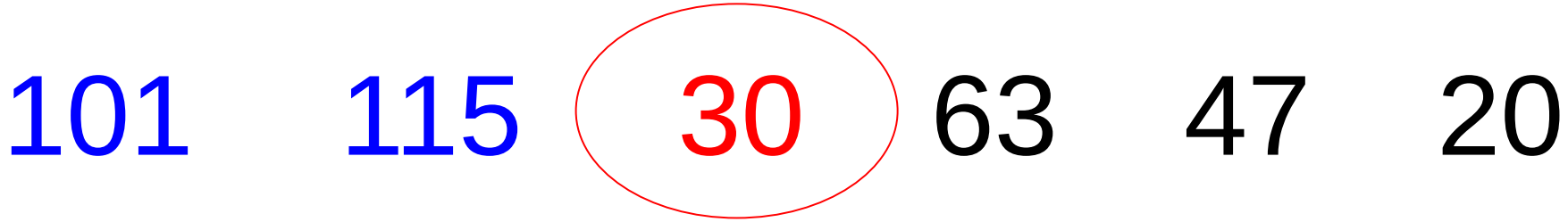
Exemplo

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

Exemplo

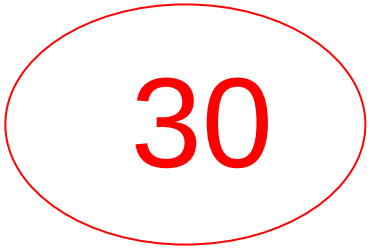
Comparamos o 101 e 115 e, como o novo elemento é o maior, os dois estão em ordem

Exemplo



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

Exemplo

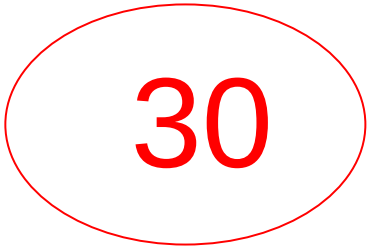


variável
temporária

101 115 ____ 63 47 20

Enquanto procuramos,
deslocamos os elementos
maiores que o 30

Exemplo



variável temporária

101 ____ 115 63 47 20

Enquanto procuramos, deslocamos os elementos maiores que o 30

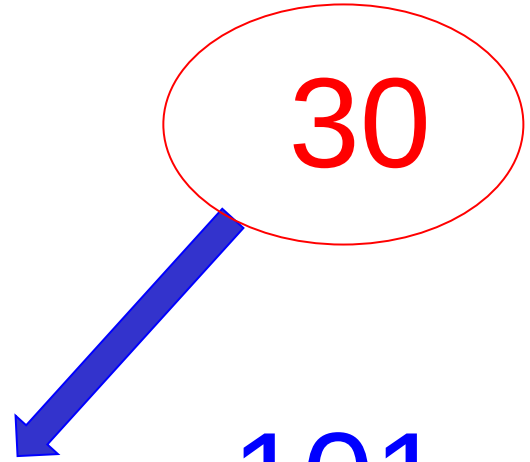
Exemplo

30 variável temporária

_____ 101 115 63 47 20

Enquanto procuramos, deslocamos os elementos maiores que o 30

Exemplo



101 115 63 47 20

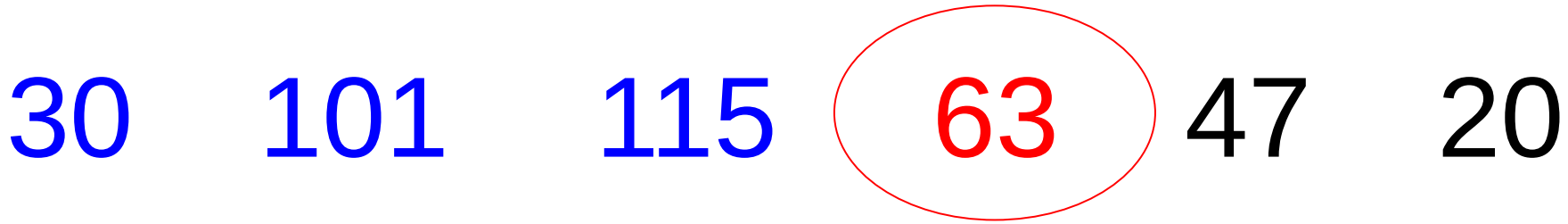
Encontramos a
posição do 30

Exemplo

30 101 115 63 47 20

Encontramos a
posição do 30

Exemplo



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

Exemplo

63 variável temporária

30 101 115 ____ 47 20

Enquanto procuramos, deslocamos os elementos maiores que o 63

Exemplo

63 variável temporária

30 101 ____ 115 47 20

Enquanto procuramos, deslocamos os elementos maiores que o 63

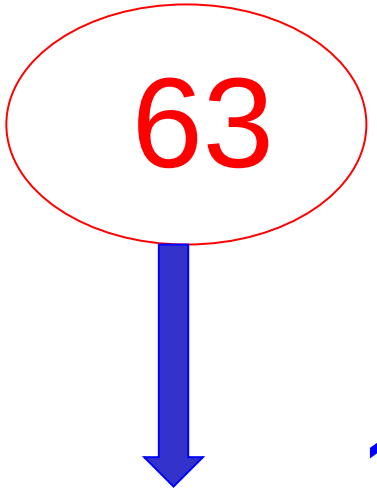
Exemplo

63 variável temporária

30 ____ 101 115 47 20

Enquanto procuramos, deslocamos os elementos maiores que o 63

Exemplo



30 _____ 101 115 47 20

Encontramos a
posição do 63

Exemplo

30 63 101 115 47 20

Encontramos a
posição do 63

Exemplo

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

Exemplo

variável
temporária

30 63 101 115 ____ 20

Enquanto procuramos,
deslocamos os elementos
maiores que o 47

Exemplo

47 variável temporária

30 63 101 ____ 115 20

Enquanto procuramos, deslocamos os elementos maiores que o 47

Exemplo

variável
temporária

30 63 ____ 101 115 20

Enquanto procuramos,
deslocamos os elementos
maiores que o 47

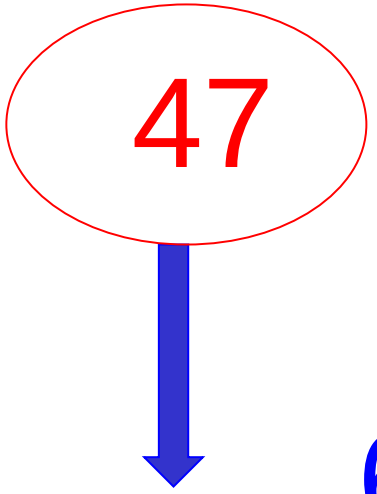
Exemplo

variável
temporária

30 ____ 63 101 115 20

Enquanto procuramos,
deslocamos os elementos
maiores que o 47

Exemplo



30 _____ 63 101 115 20

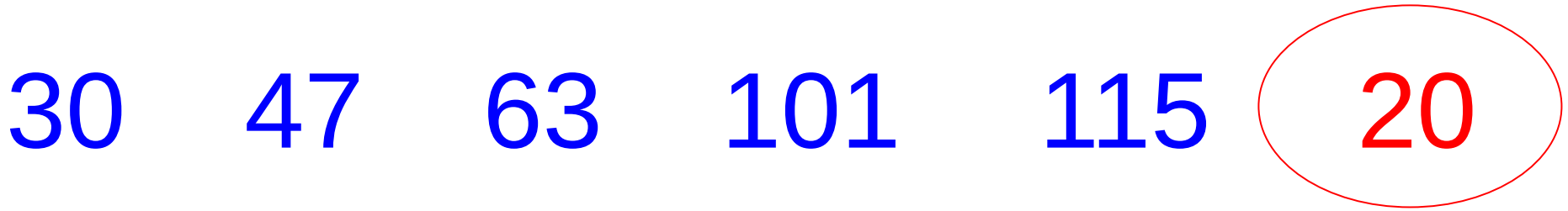
Encontramos a
posição do 47

Exemplo

30 47 63 101 115 20

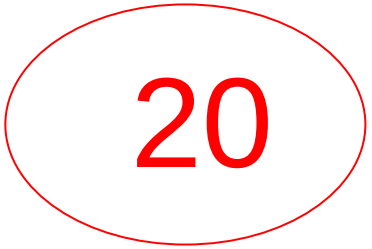
Encontramos a posição do 47

Exemplo



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

Exemplo

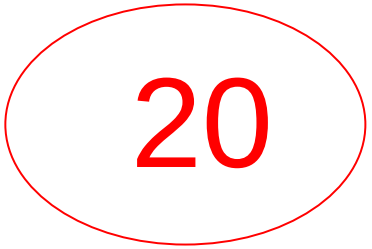


variável
temporária

30 47 63 101 115 ____

Enquanto procuramos,
deslocamos os elementos
maiores que o 20

Exemplo

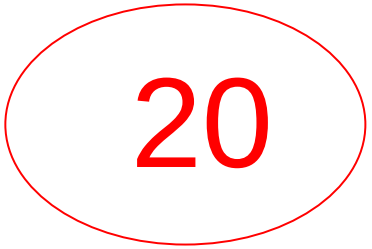


variável
temporária

30 47 63 101 ____ 115

Enquanto procuramos,
deslocamos os elementos
maiores que o 20

Exemplo

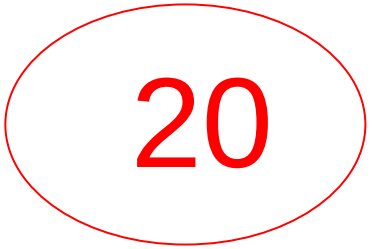


variável
temporária

30 47 63 ____ 101 115

Enquanto procuramos,
deslocamos os elementos
maiores que o 20

Exemplo

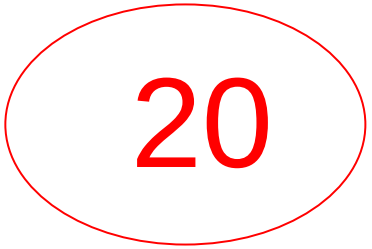


variável
temporária

30 47 ____ 63 101 115

Enquanto procuramos,
deslocamos os elementos
maiores que o 20

Exemplo

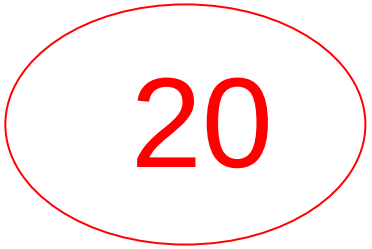


variável
temporária

30 ____ 47 63 101 115

Enquanto procuramos,
deslocamos os elementos
maiores que o 20

Exemplo



variável
temporária

_____ 30 47 63 101 115

Enquanto procuramos,
deslocamos os elementos
maiores que o 20

Exemplo

Encontramos a
posição do 20

Exemplo

20 30 47 63 101 115

Encontramos a posição do 20

Exercício

- 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

- 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

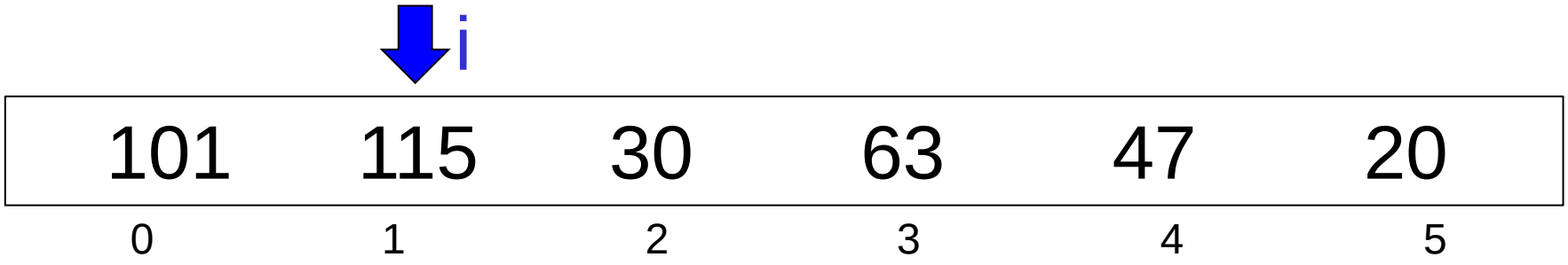
Algoritmo em C *like*

```
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];
                j--;
        }
        array[j + 1] = tmp;
}
```

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

Algoritmo em C *like*

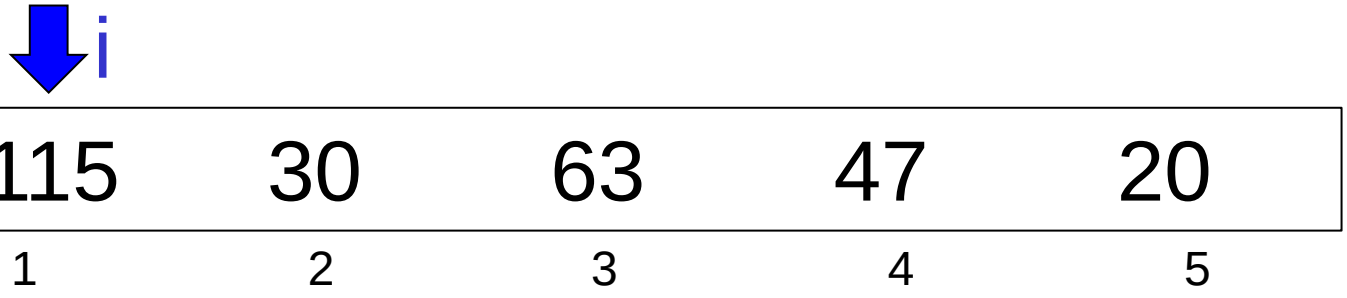
```
for (int i = 1; i < n; i++) {
    int tmp = array[i];
    int j = i - 1;
    while ( (j >= 0) && (array[j] > tmp) ){
        array[j + 1] = array[j];
        j--;
    }
    array[j + 1] = tmp;
}
```



Algoritmo em C *like*

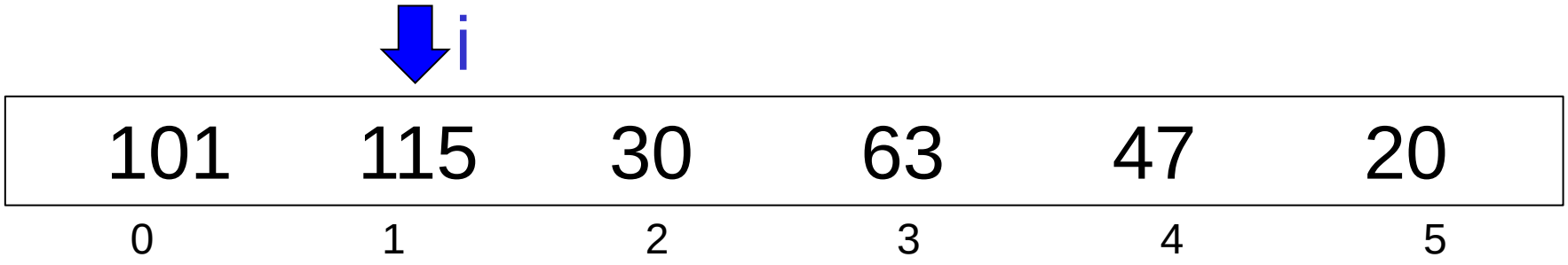
```
for (int i = 1; i < n; i++) {
        int tmp = array[i];
        int j = i - 1;
        while ( (j >= 0) && (array[j] > tmp) ){
                array[j + 1] = array[j];
                j--;
        }
        array[j + 1] = tmp;
}
```

true: 1 < 6



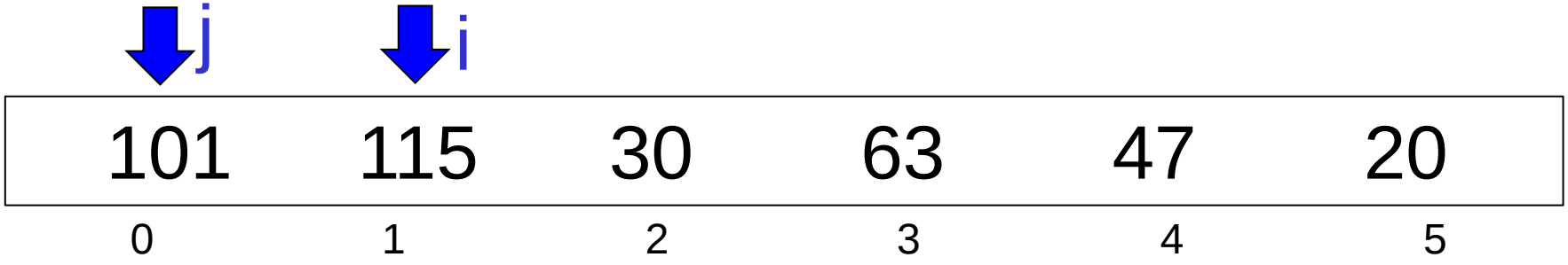
Algoritmo em C *like*

```
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;
}
```

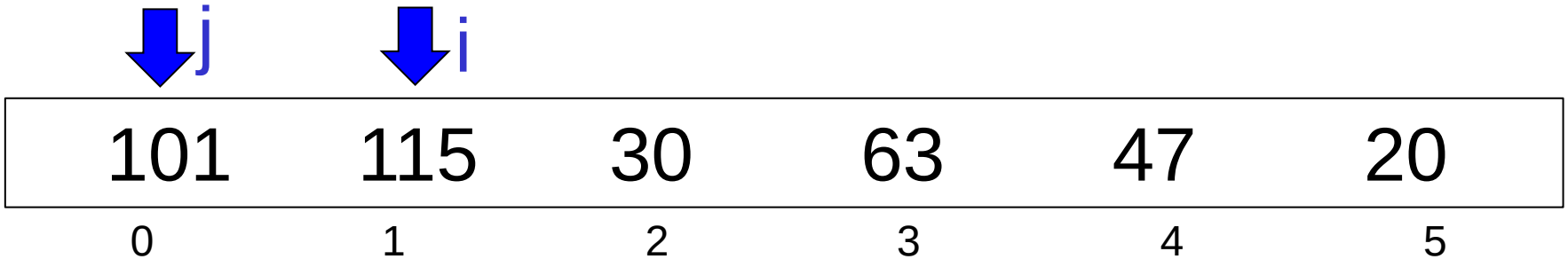


Algoritmo em C *like*

```
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;
}
```

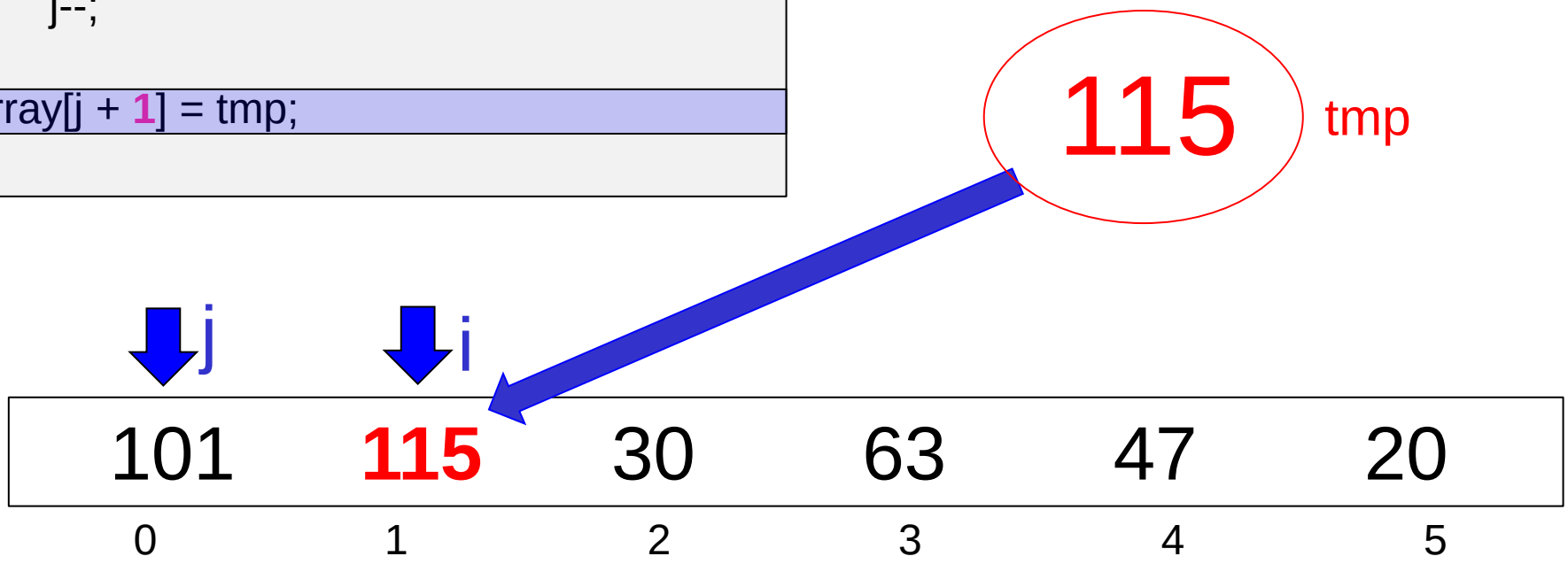
false: 0 >= 0 && 101 > 105

115 tmp



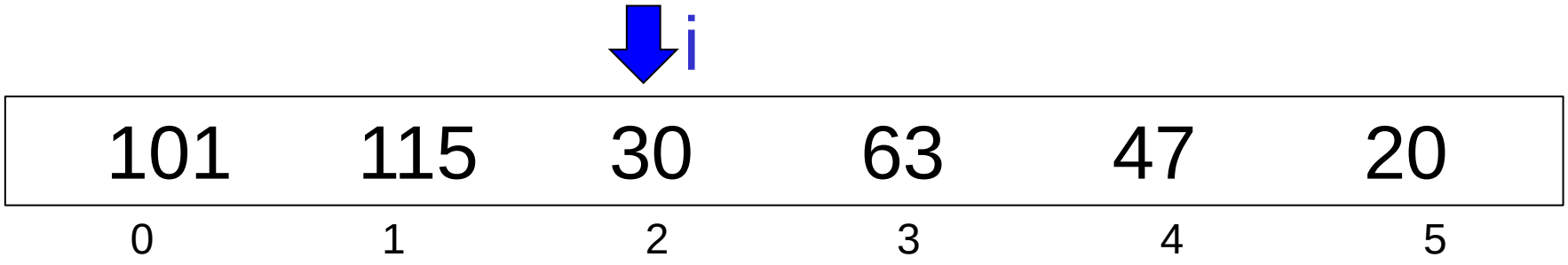
Algoritmo em C *like*

```
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*

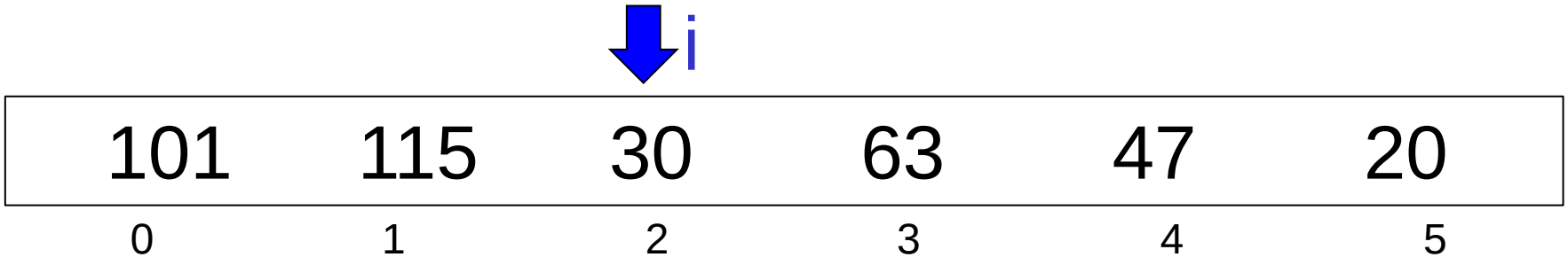
```
for (int i = 1; i < n; i++) {
        int tmp = array[i];
        int j = i - 1;
        while ( (j >= 0) && (array[j] > tmp) ){
                array[j + 1] = array[j];
                j--;
        }
        array[j + 1] = tmp;
}
```



Algoritmo em C *like*

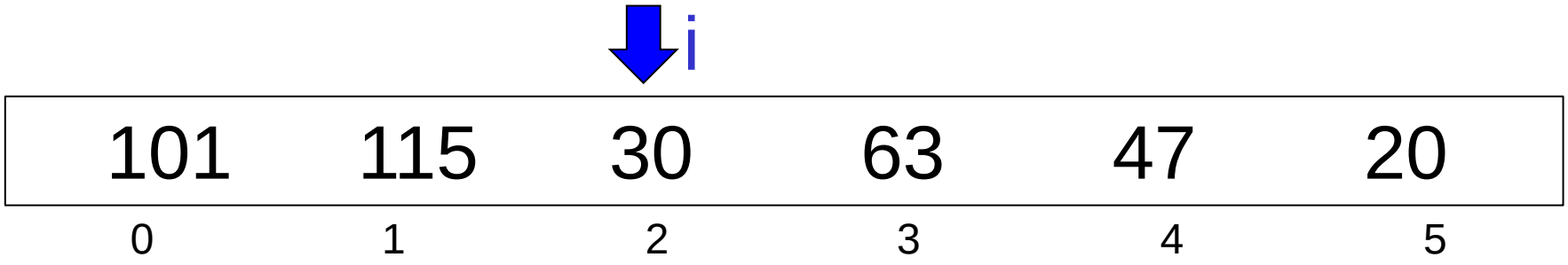
```
for (int i = 1; i < n; i++) {
        int tmp = array[i];
        int j = i - 1;
        while ( (j >= 0) && (array[j] > tmp) ){
                array[j + 1] = array[j];
                j--;
        }
        array[j + 1] = tmp;
}
```

true: 2 < 6



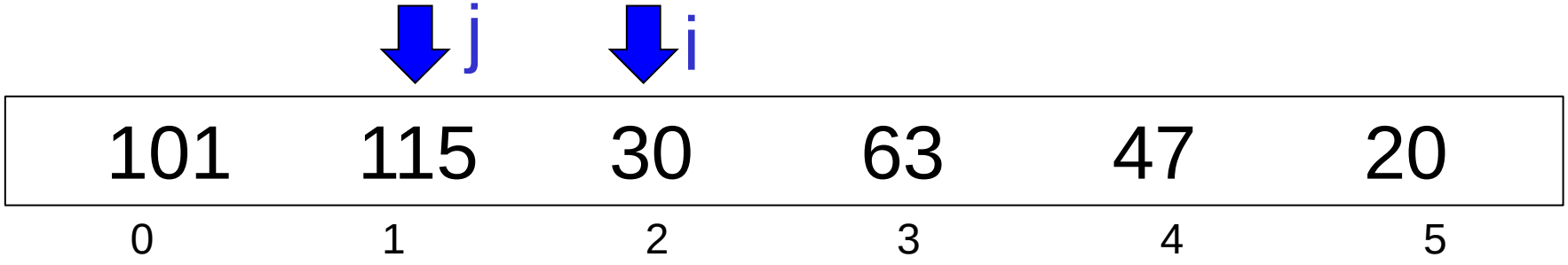
Algoritmo em C *like*

```
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;
}
```

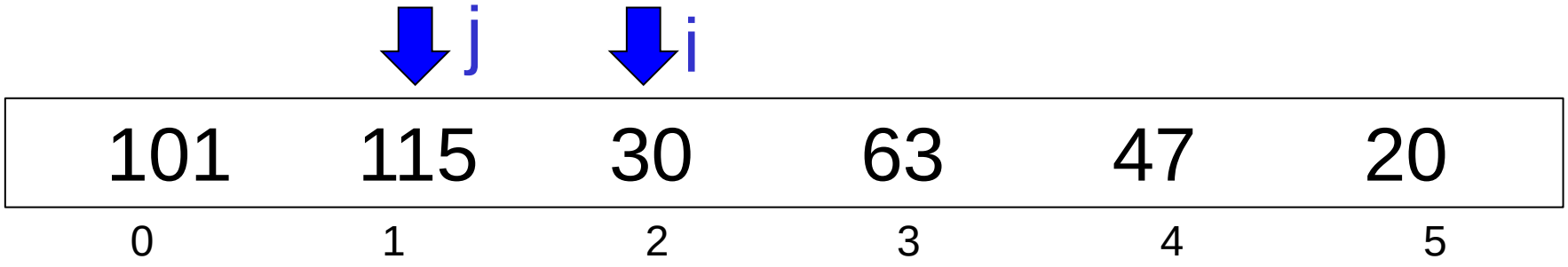


Algoritmo em C *like*

```
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;
}
```

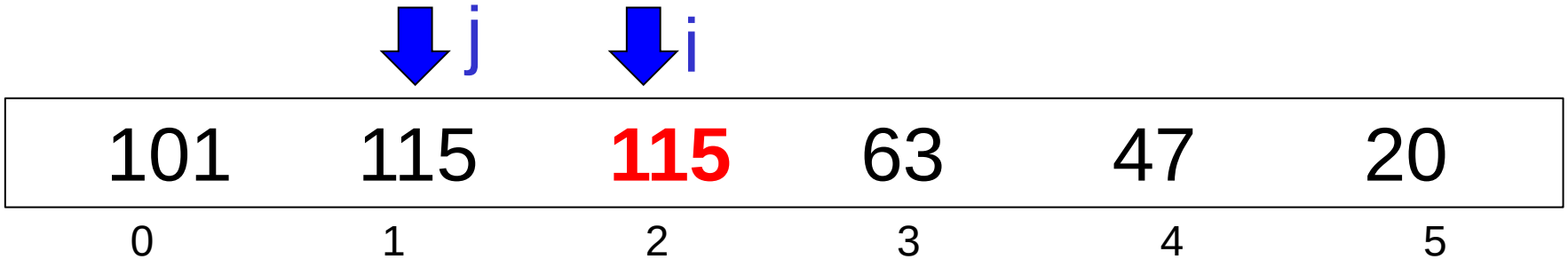
true: 1 >= 0 && 115 > 30

30 tmp



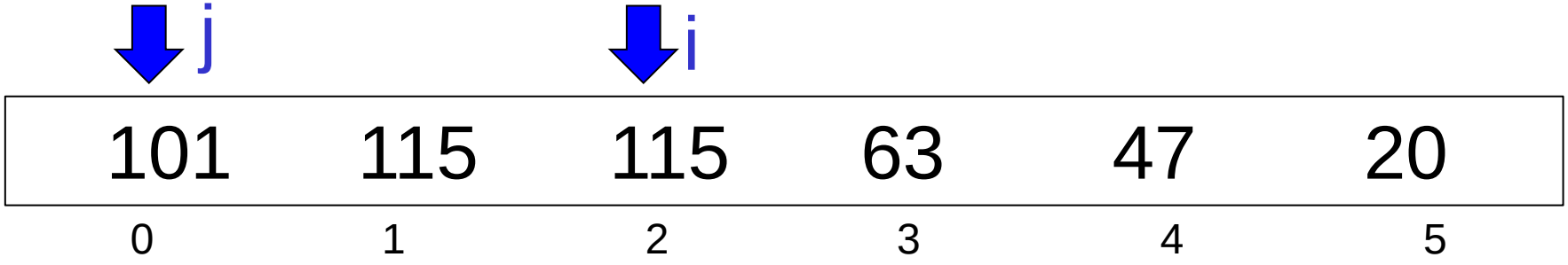
Algoritmo em C *like*

```
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;
}
```



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;
}
```

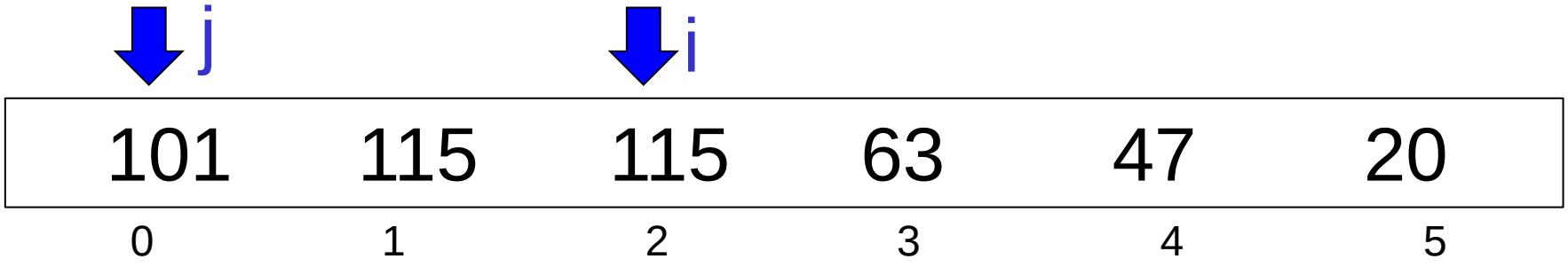


Algoritmo em C *like*

```
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;
}
```

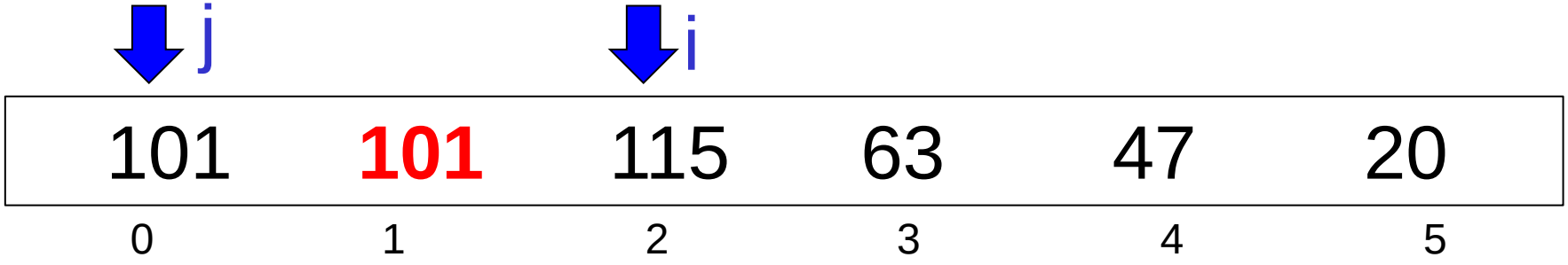
true: 0 >= 0 && 101 > 30

30 tmp



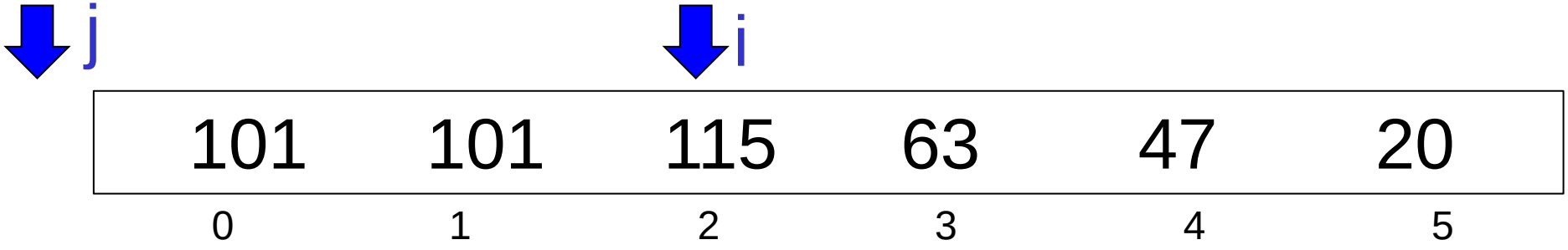
Algoritmo em C *like*

```
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;
}
```



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;
}
```



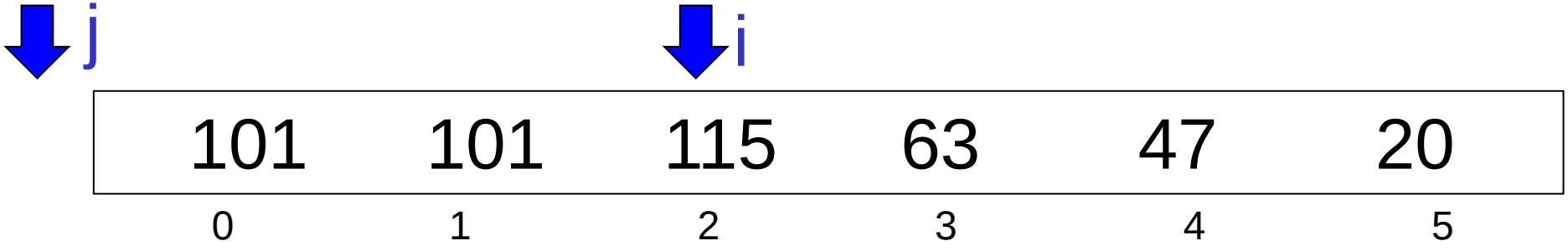
Algoritmo em C *like*

```
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;
}
```

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

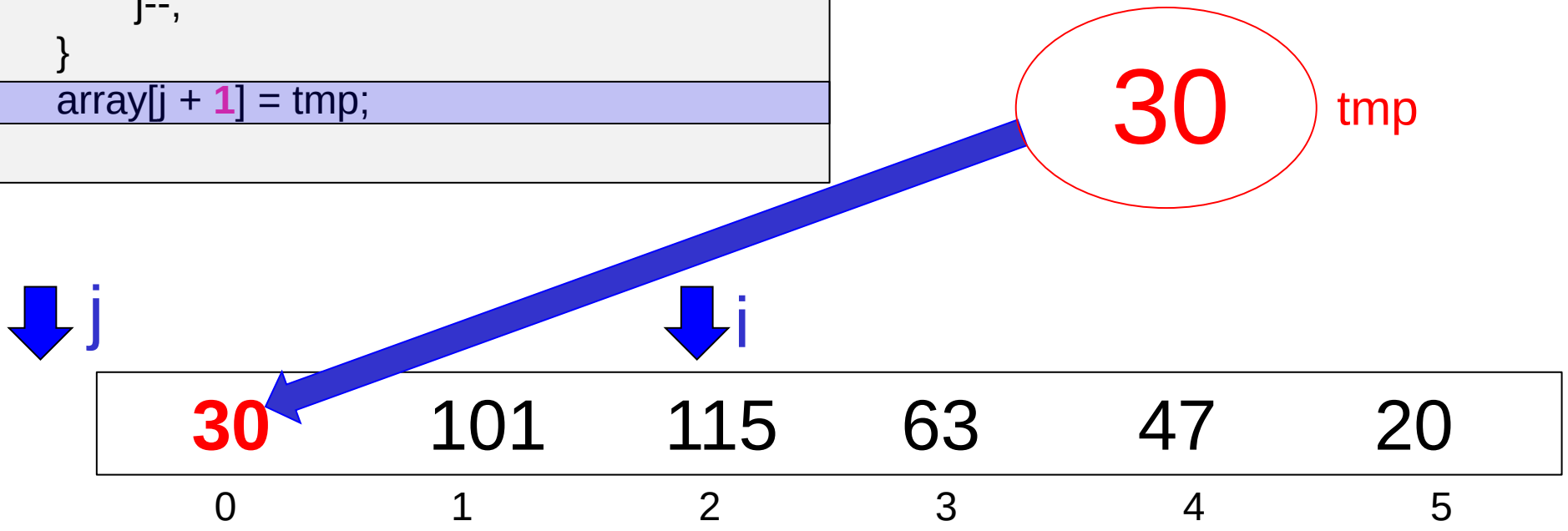
false: -1 >= 0 && ☹ ☹ ☹ ☹

30 tmp



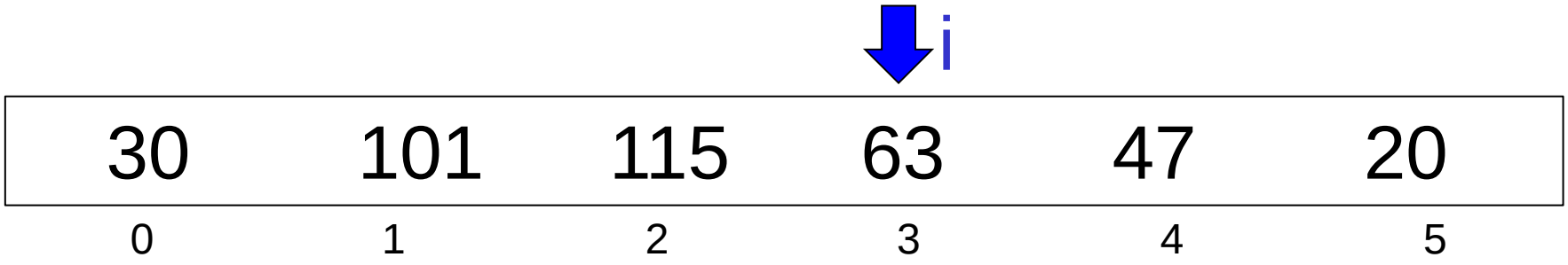
Algoritmo em C *like*

```
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;
}
```



Algoritmo em C *like*

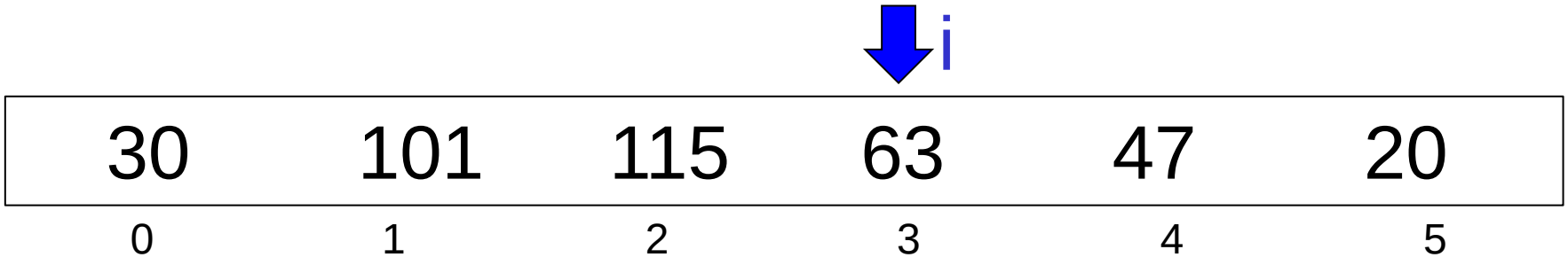
```
for (int i = 1; i < n; i++) {
        int tmp = array[i];
        int j = i - 1;
        while ( (j >= 0) && (array[j] > tmp) ){
                array[j + 1] = array[j];
                j--;
        }
        array[j + 1] = tmp;
}
```



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;
}
```

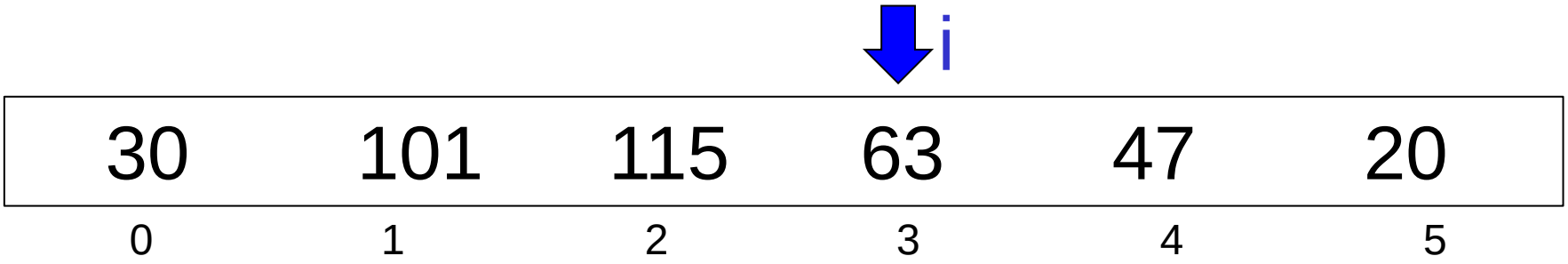
true: 3 < 6



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;
}
```

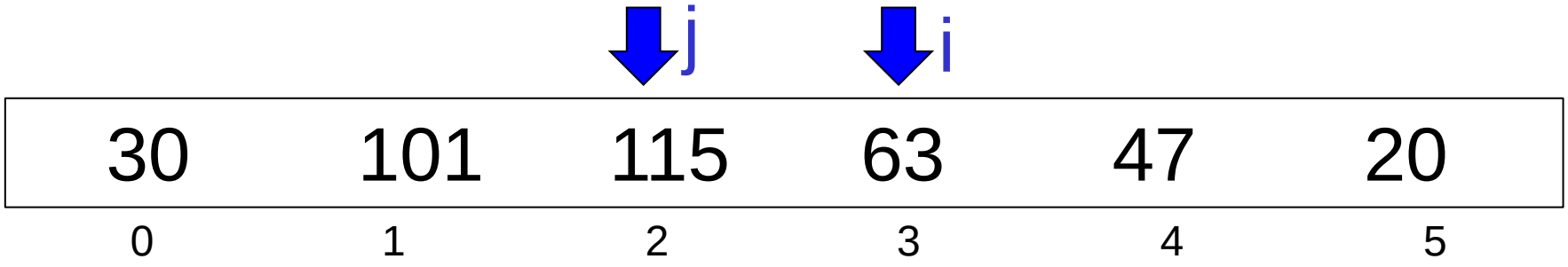
63 tmp



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

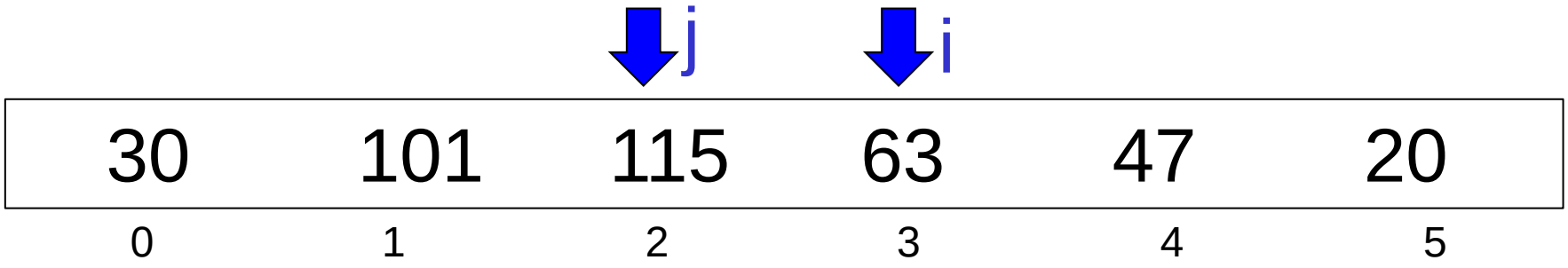


Algoritmo em C *like*

```
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: 2 >= 0 && 115 > 63

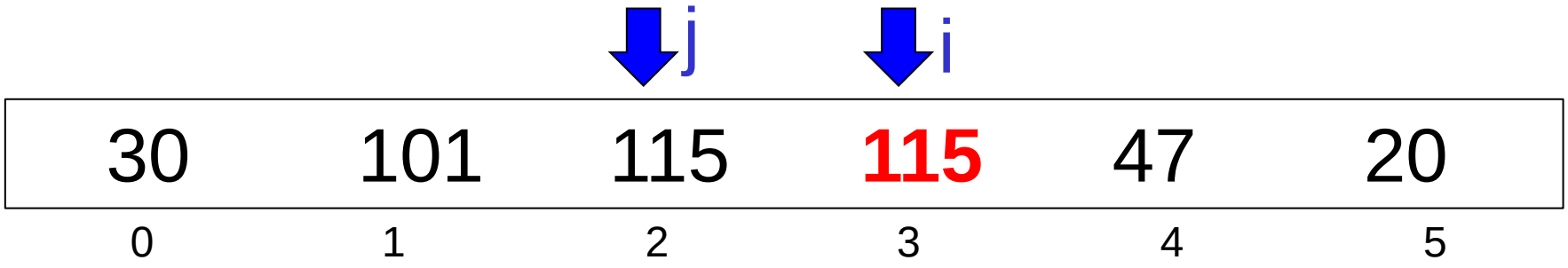
63 tmp



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;
}
```

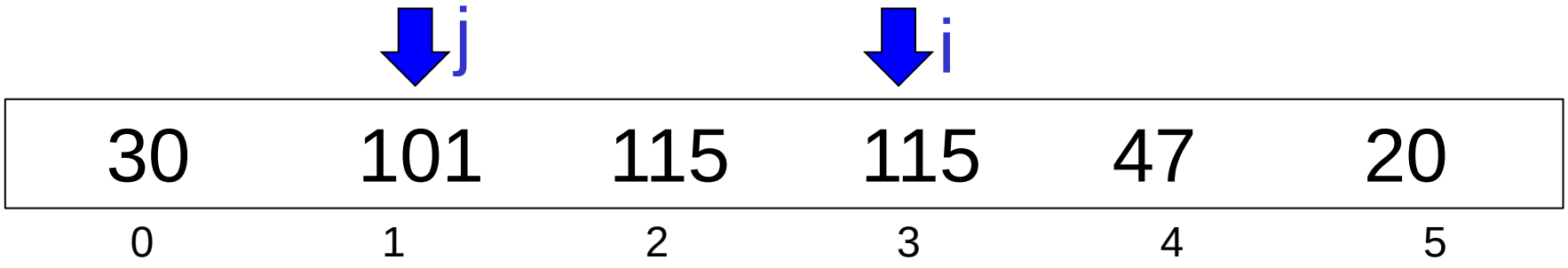
63 tmp



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

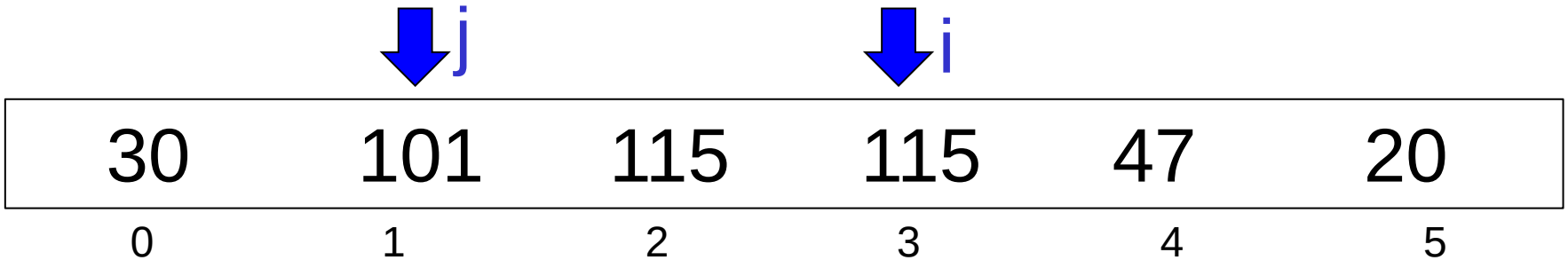


Algoritmo em C *like*

```
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 >= 0 && 101 > 63

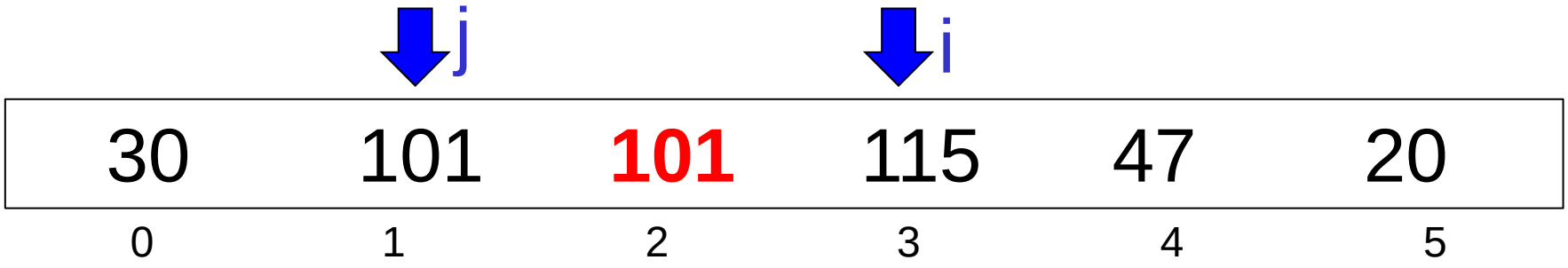
63 tmp



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;
}
```

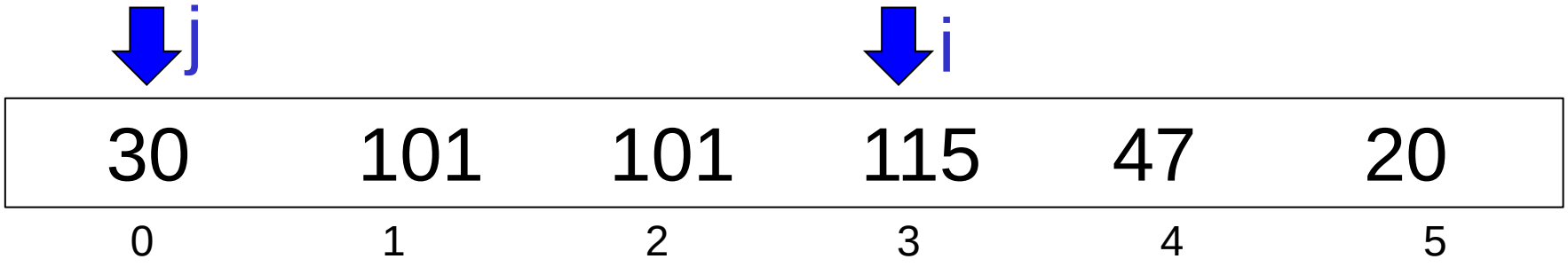
63 tmp



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

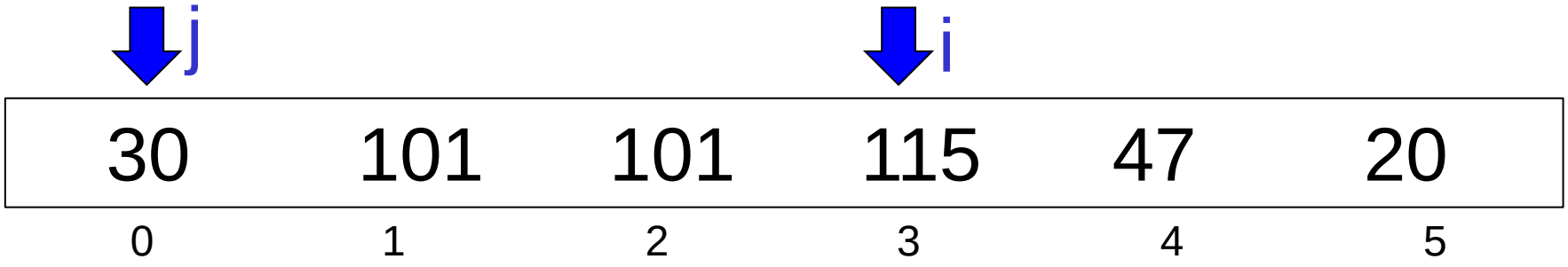


Algoritmo em C *like*

```
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;
}
```

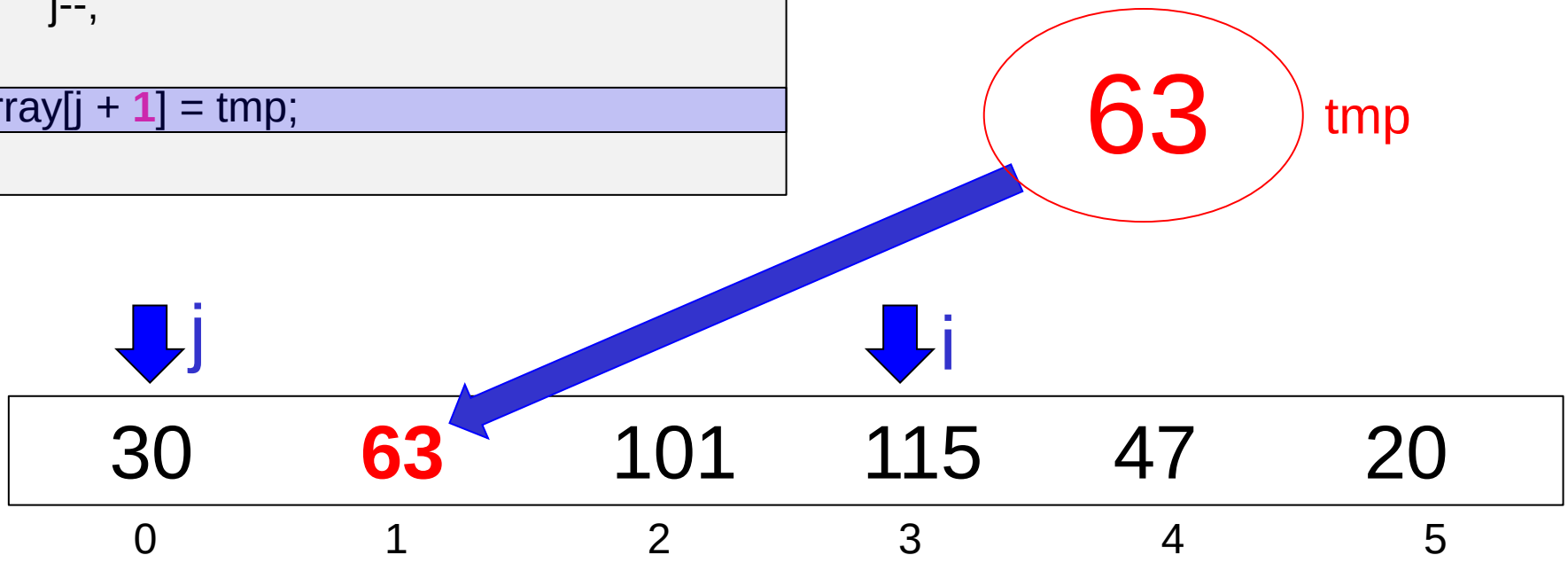
false: 0 >= 0 && 30 > 63

63 tmp



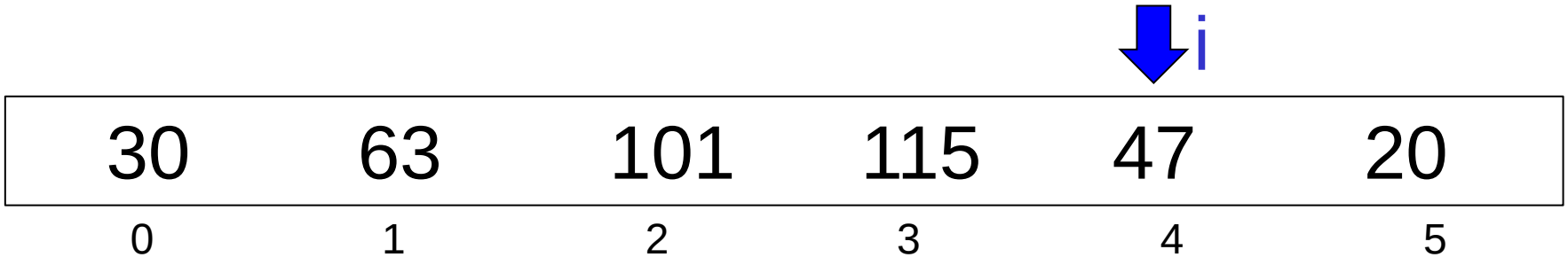
Algoritmo em C *like*

```
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*

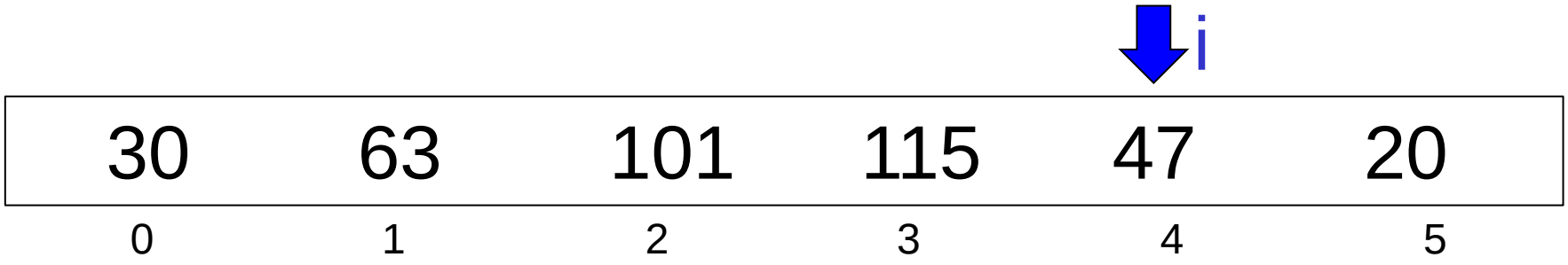
```
for (int i = 1; i < n; i++) {
        int tmp = array[i];
        int j = i - 1;
        while ( (j >= 0) && (array[j] > tmp) ){
                array[j + 1] = array[j];
                j--;
        }
        array[j + 1] = tmp;
}
```



Algoritmo em C *like*

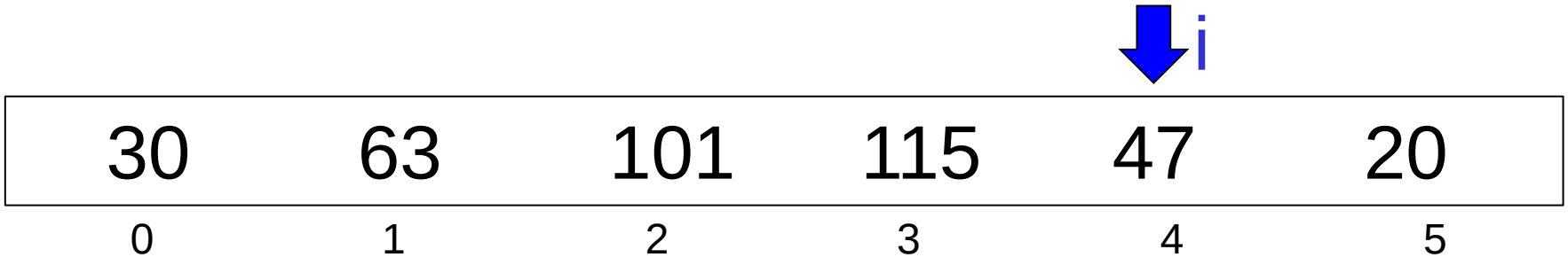
```
for (int i = 1; i < n; i++) {
        int tmp = array[i];
        int j = i - 1;
        while ( (j >= 0) && (array[j] > tmp) ){
                array[j + 1] = array[j];
                j--;
        }
        array[j + 1] = tmp;
}
```

true: 4 < 6



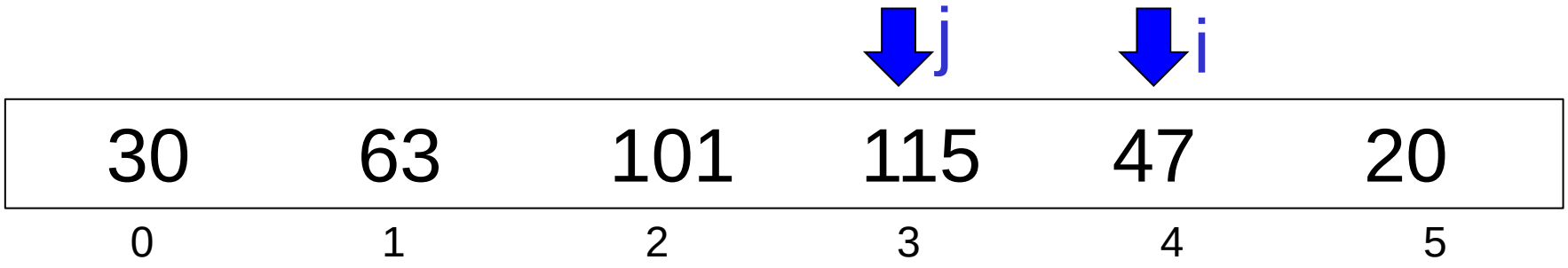
Algoritmo em C *like*

```
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;
}
```

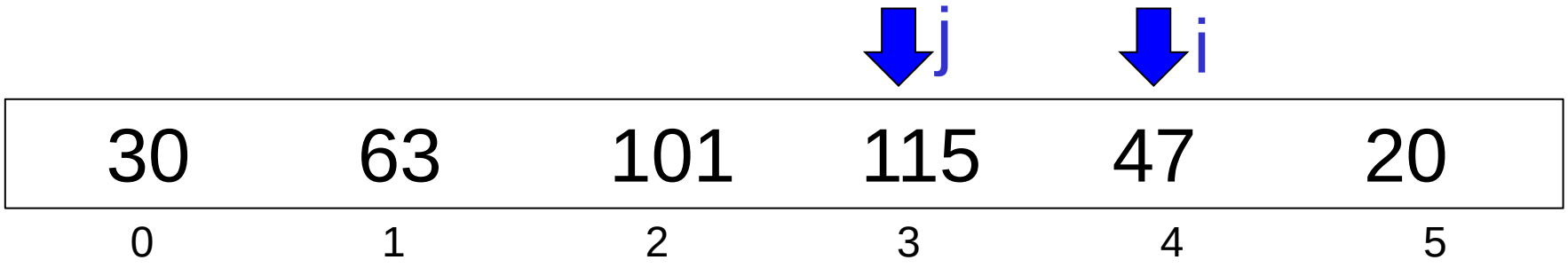


Algoritmo em C *like*

```
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;
}
```

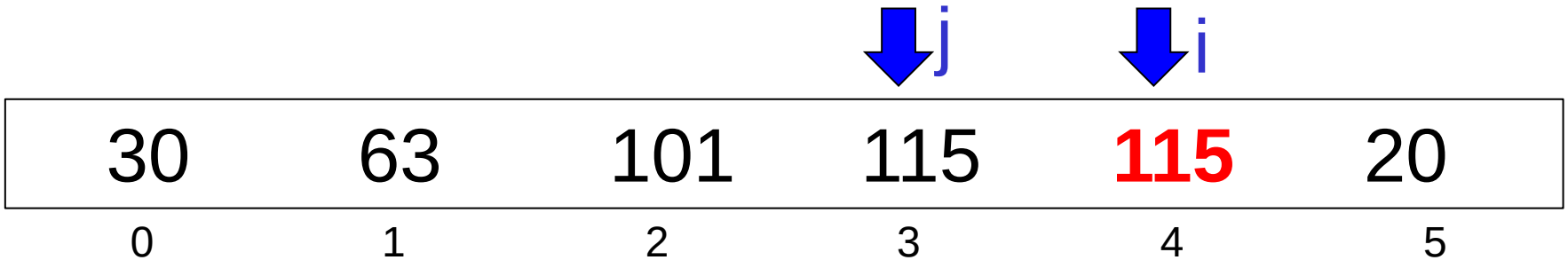
true: 3 >= 0 && 115 > 47

47 tmp



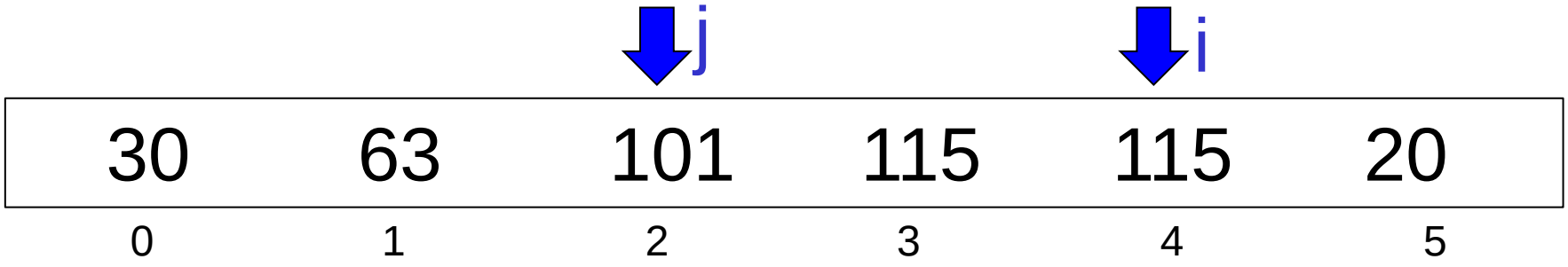
Algoritmo em C *like*

```
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;
}
```



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;
}
```

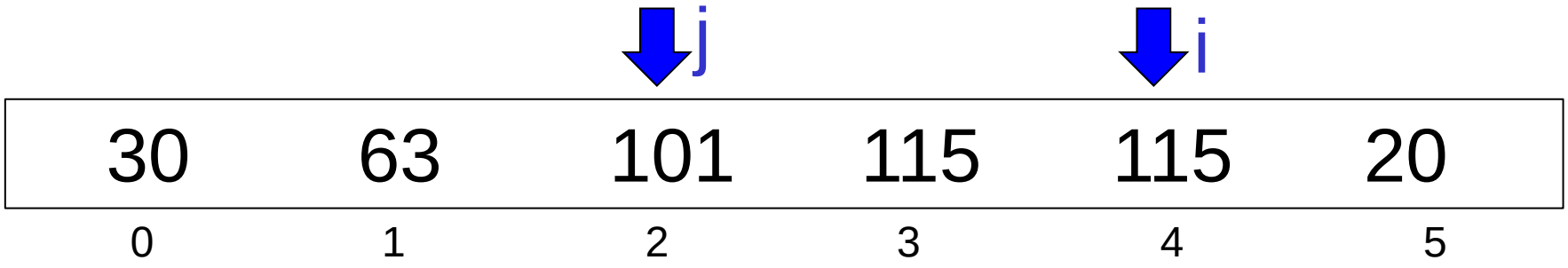


Algoritmo em C *like*

```
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;
}
```

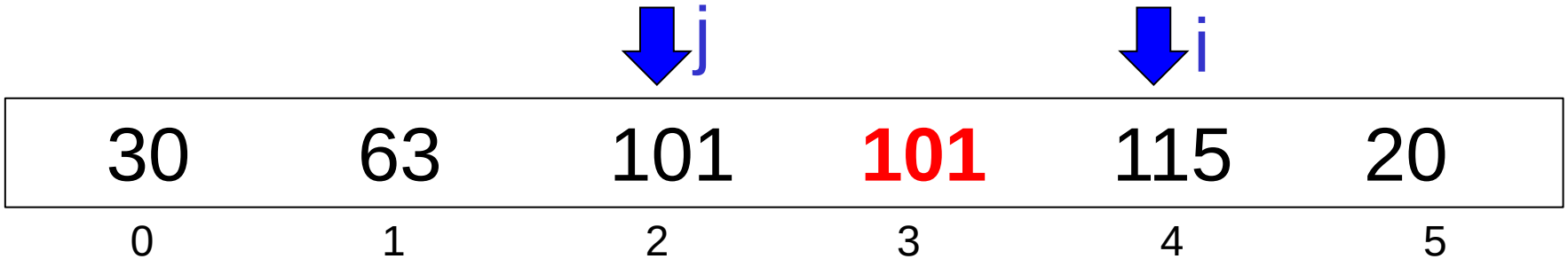
true: 2 >= 0 && 101 > 47

47 tmp



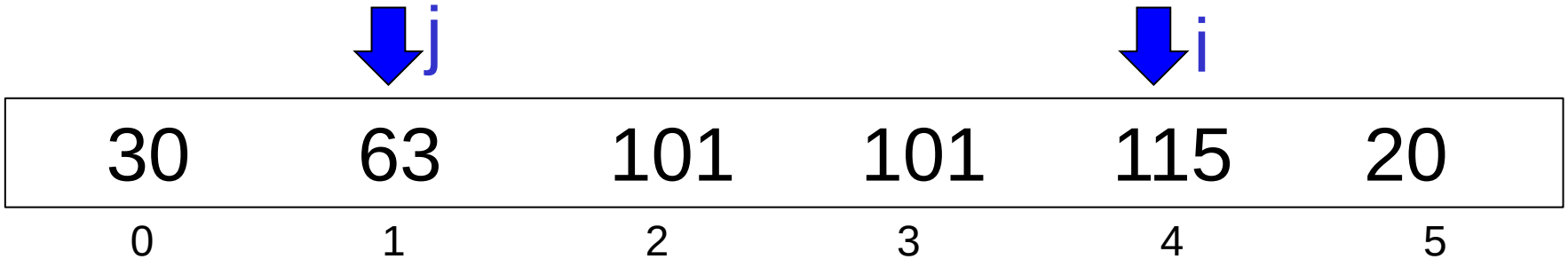
Algoritmo em C *like*

```
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;
}
```



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;
}
```

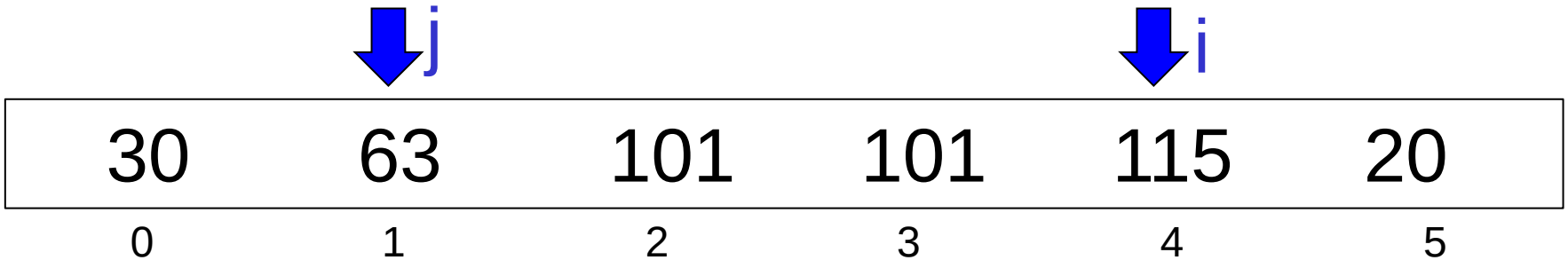


Algoritmo em C *like*

```
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;
}
```

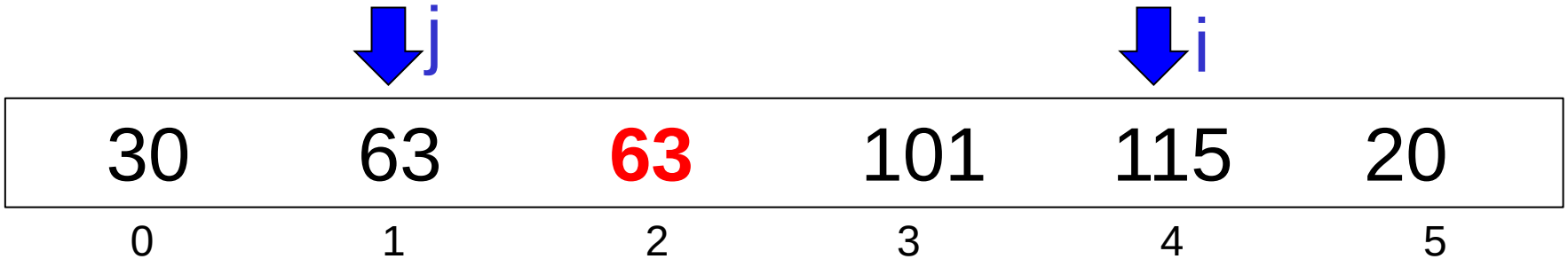
true: 1 >= 0 && 63 > 47

47 tmp



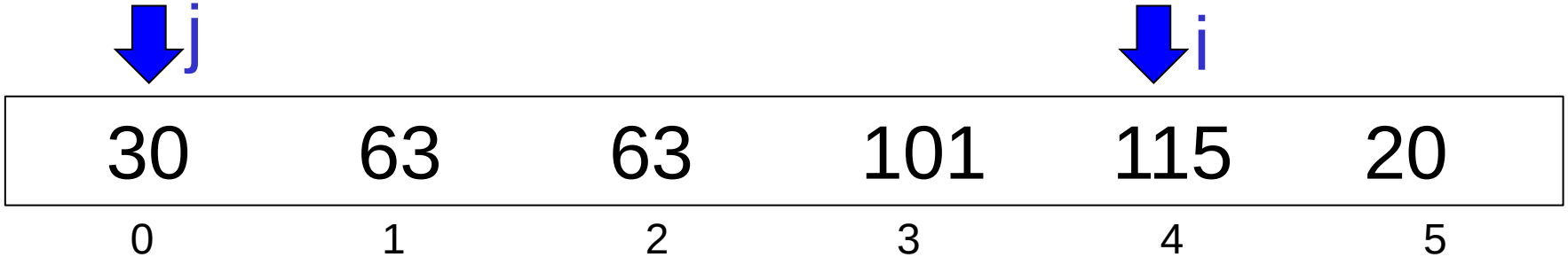
Algoritmo em C *like*

```
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;
}
```



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;
}
```

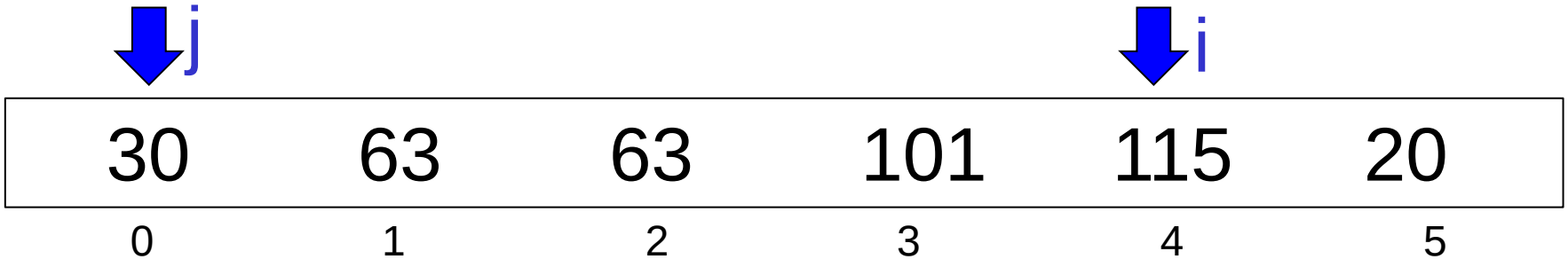


Algoritmo em C *like*

```
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;
}
```

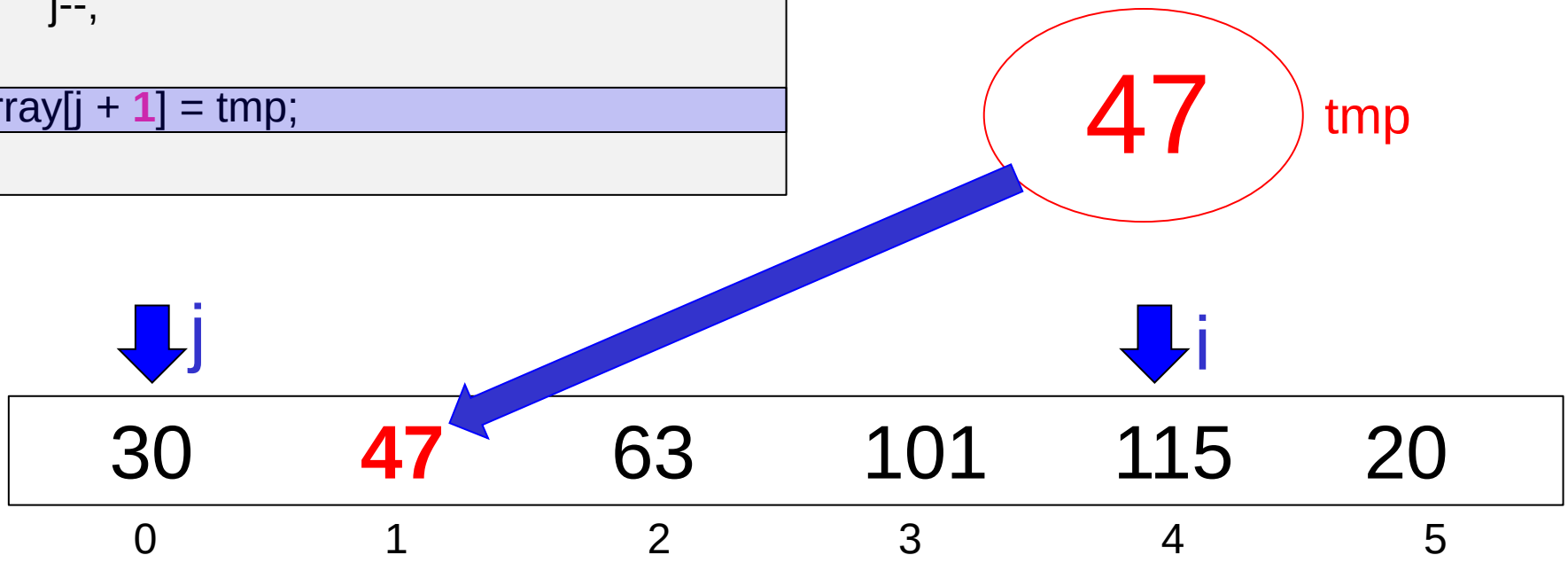
false: 0 >= 0 && 30 > 47

47 tmp



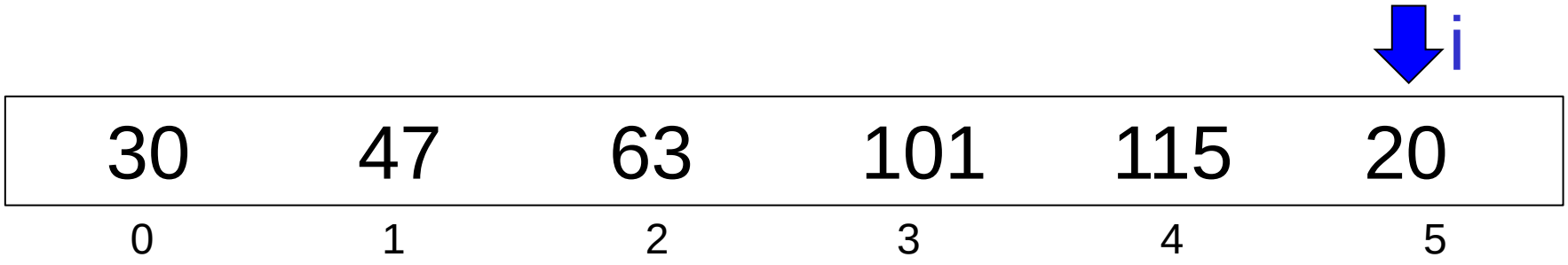
Algoritmo em C *like*

```
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*

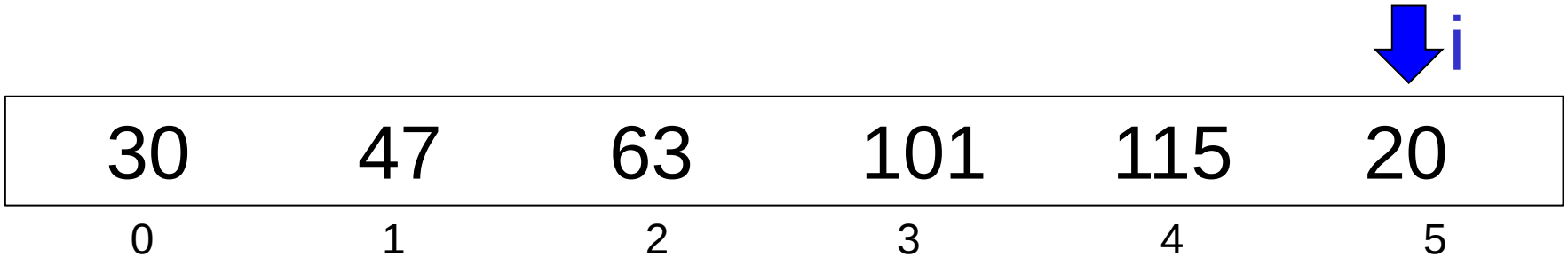
```
for (int i = 1; i < n; i++) {
        int tmp = array[i];
        int j = i - 1;
        while ( (j >= 0) && (array[j] > tmp) ){
                array[j + 1] = array[j];
                j--;
        }
        array[j + 1] = tmp;
}
```



Algoritmo em C *like*

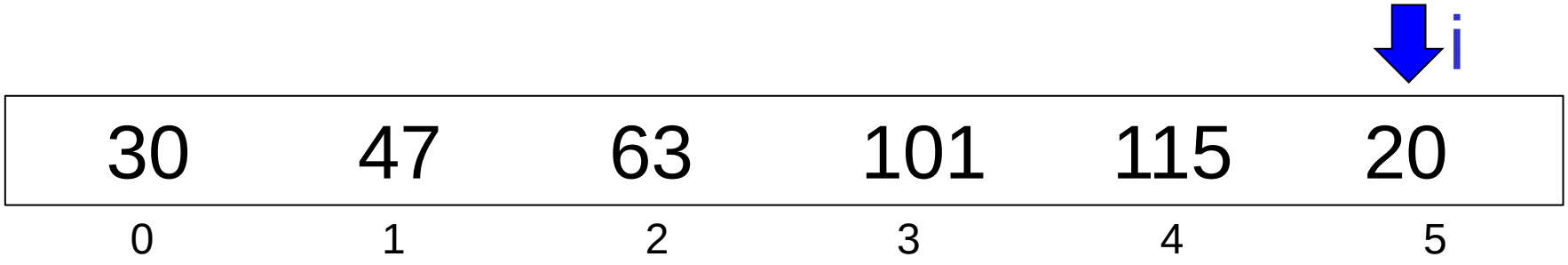
```
for (int i = 1; i < n; i++) {
        int tmp = array[i];
        int j = i - 1;
        while ( (j >= 0) && (array[j] > tmp) ){
                array[j + 1] = array[j];
                j--;
        }
        array[j + 1] = tmp;
}
```

true: 5 < 6



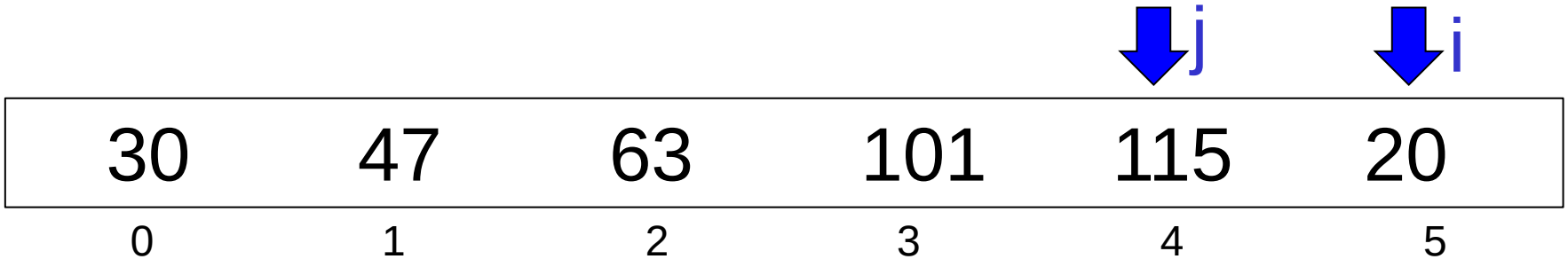
Algoritmo em C *like*

```
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;
}
```

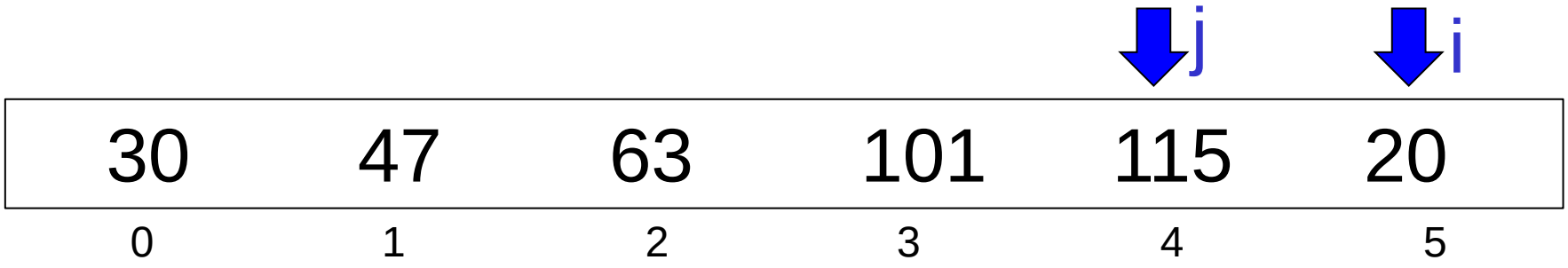


Algoritmo em C *like*

```
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;
}
```

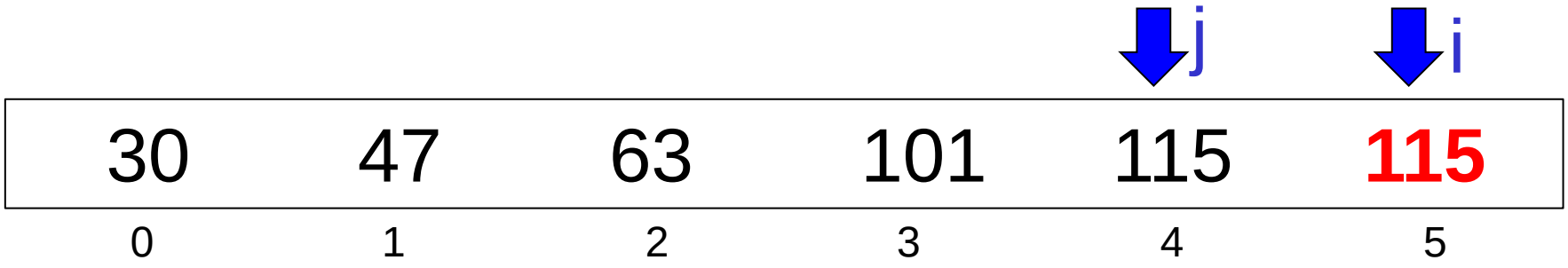
true: 4 >= 0 && 115 > 20

20 tmp



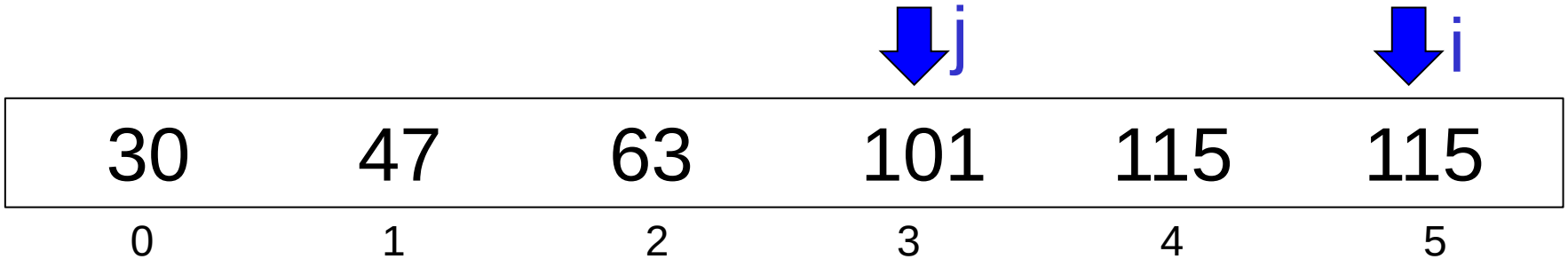
Algoritmo em C *like*

```
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;
}
```



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;
}
```

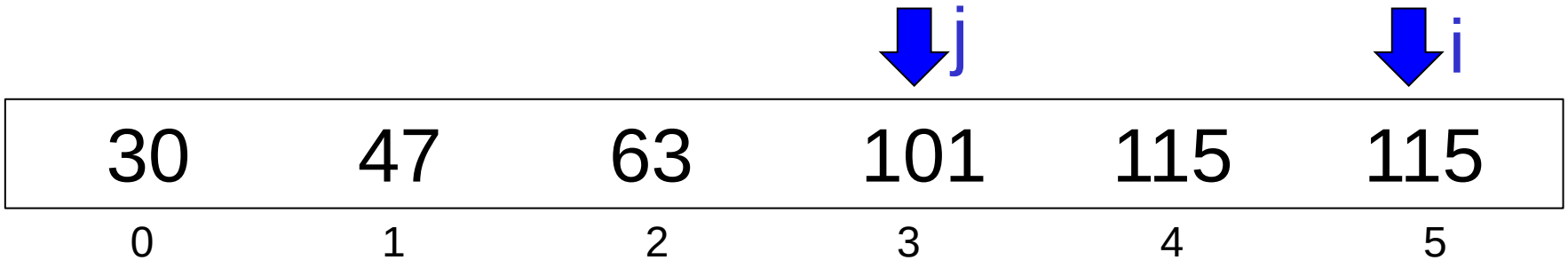


Algoritmo em C *like*

```
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;
}
```

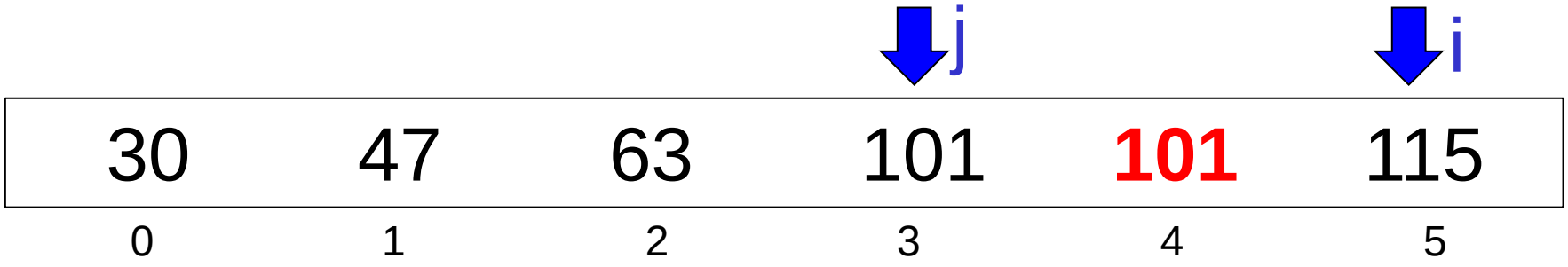
true: 3 >= 0 && 101 > 20

20 tmp



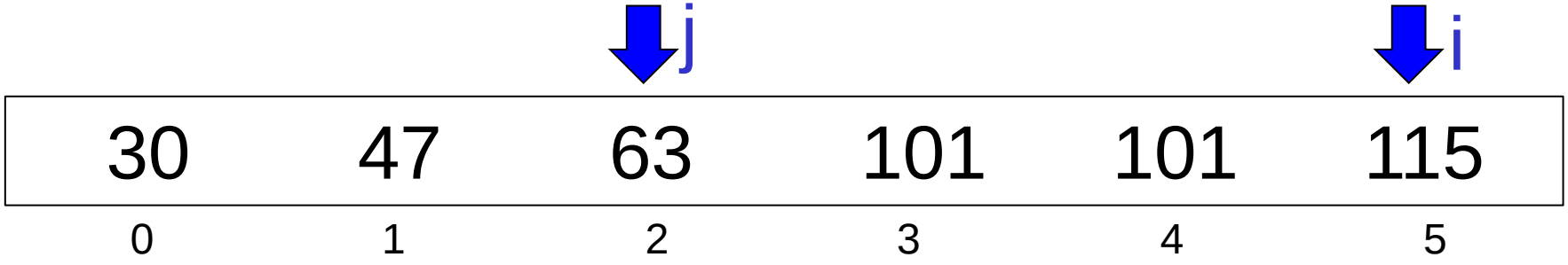
Algoritmo em C *like*

```
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;
}
```



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;
}
```

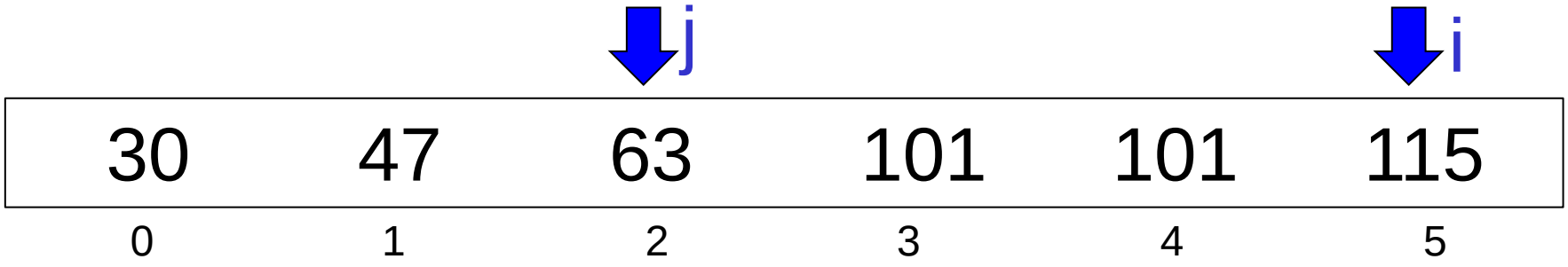


Algoritmo em C *like*

```
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;
}
```

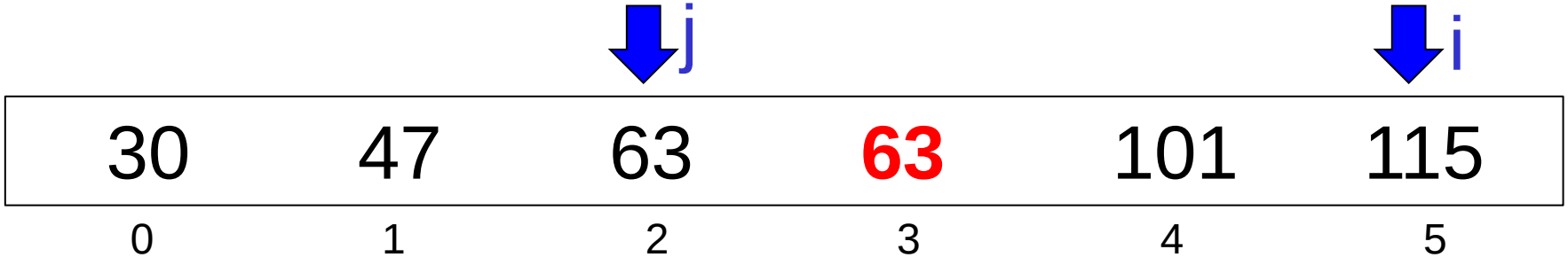
true: 2 >= 0 && 63 > 20

20 tmp



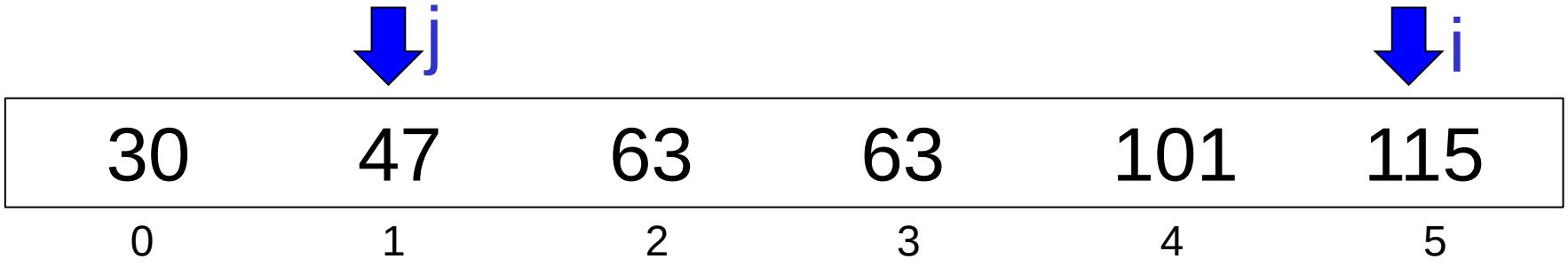
Algoritmo em C *like*

```
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;
}
```



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;
}
```

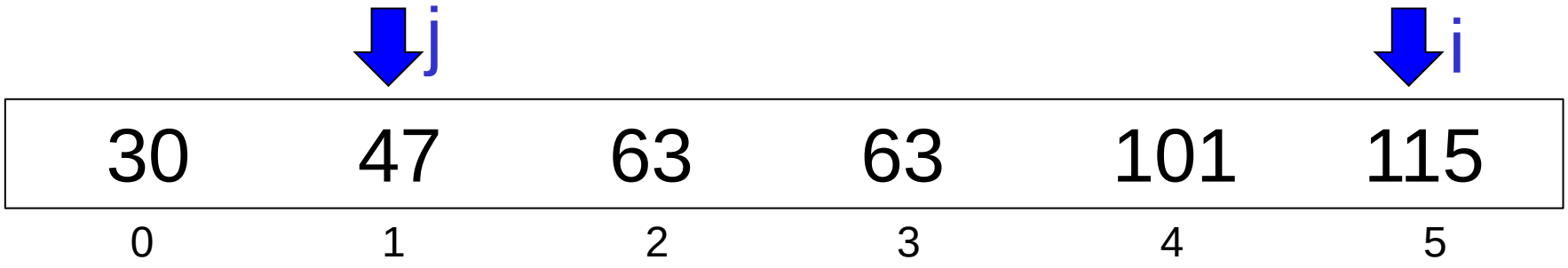


Algoritmo em C *like*

```
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;
}
```

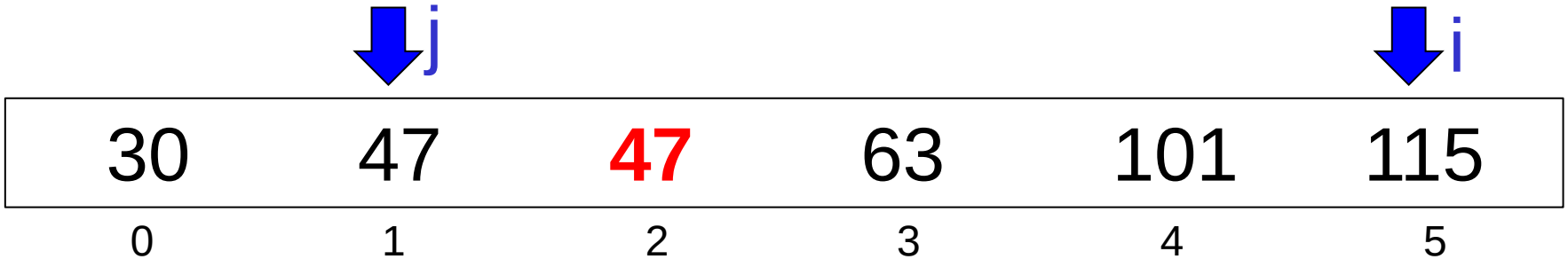
true: 1 >= 0 && 47 > 20

20 tmp



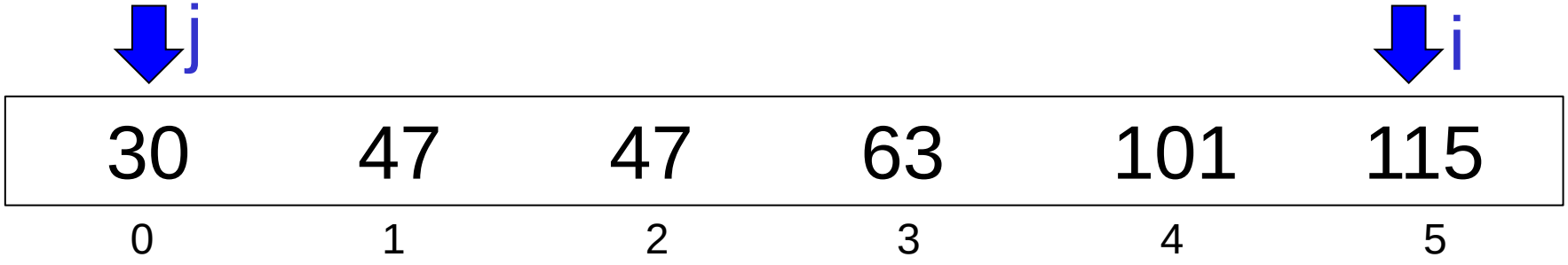
Algoritmo em C *like*

```
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;
}
```



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;
}
```

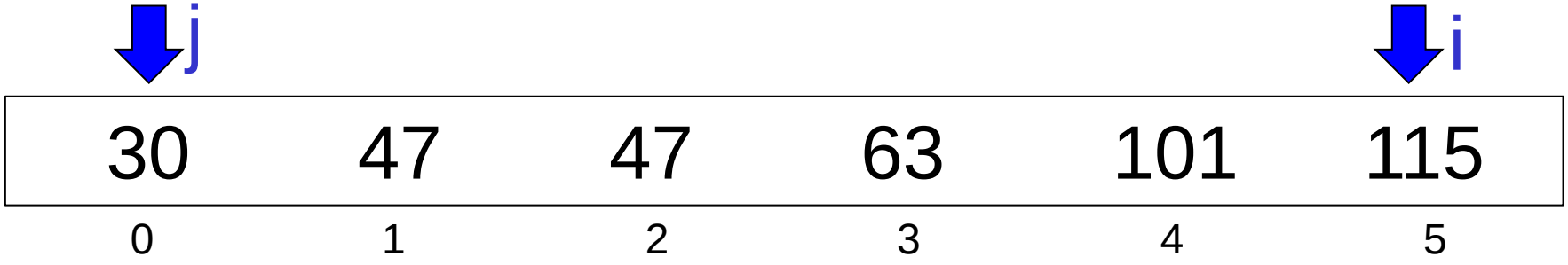


Algoritmo em C *like*

```
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;
}
```

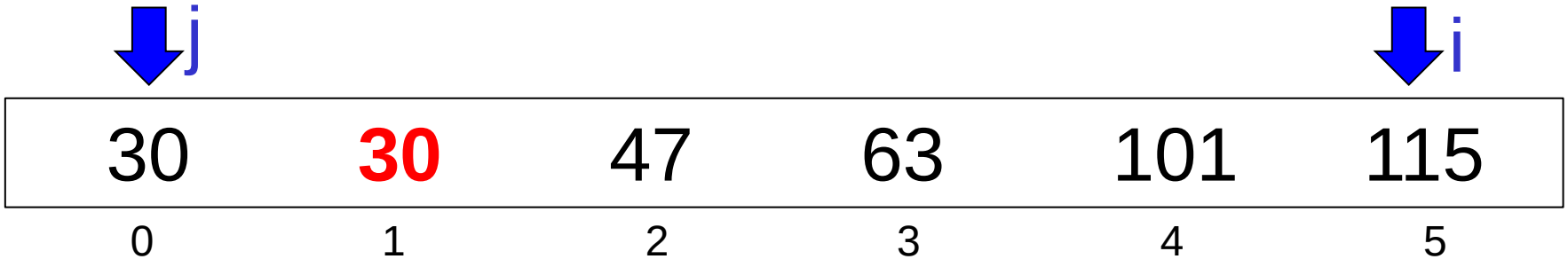
true: 0 >= 0 && 30 > 20

20 tmp



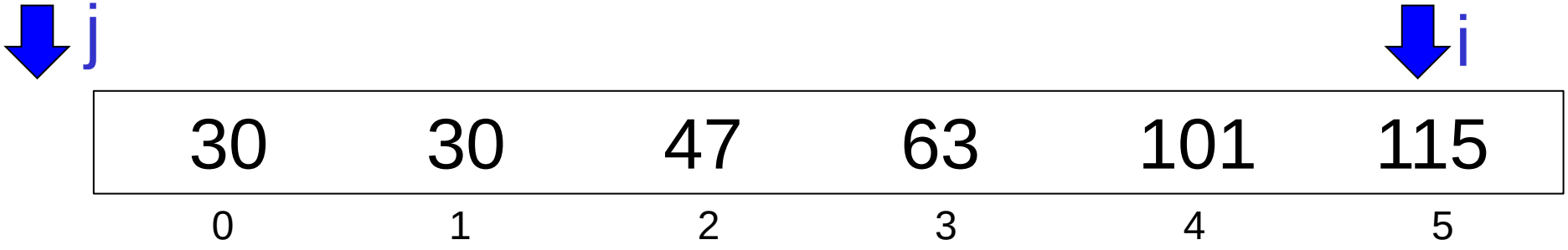
Algoritmo em C *like*

```
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;
}
```



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;
}
```

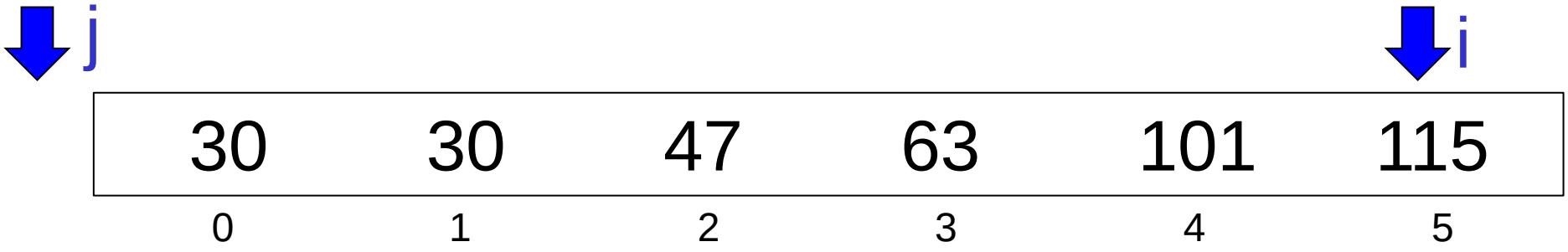


Algoritmo em C *like*

```
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;
}
```

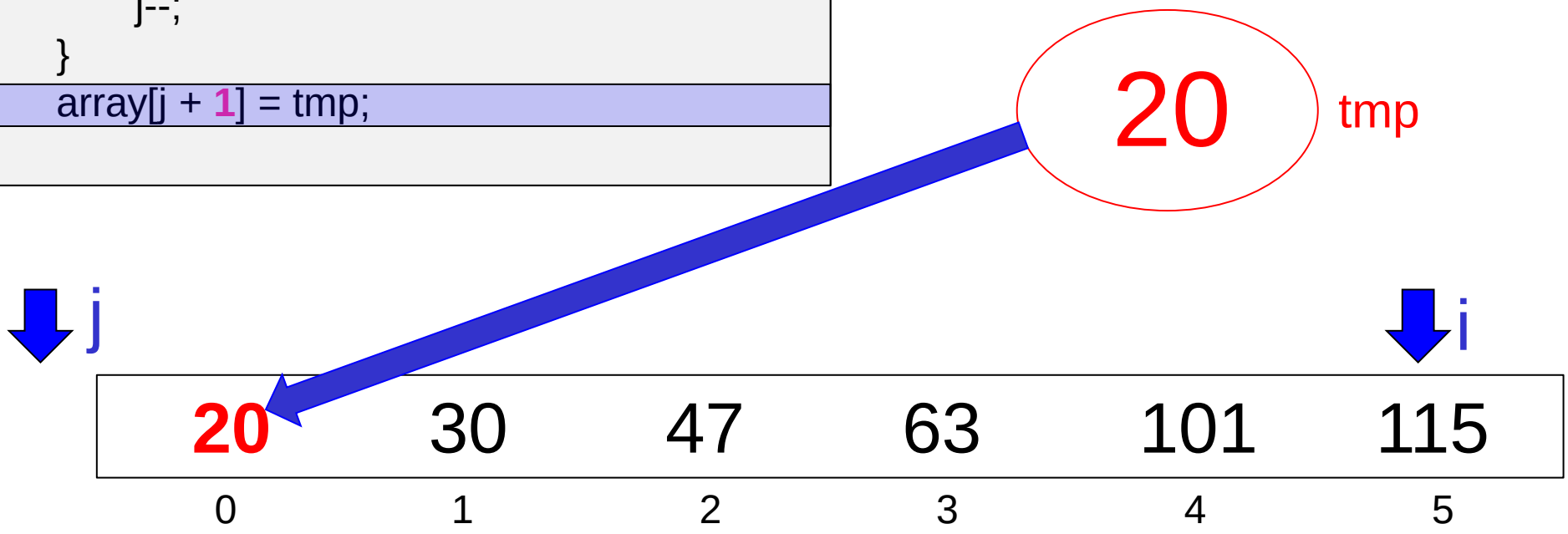
false: -1 >= 0 && ☹ ☹ ☹ ☹

20 tmp



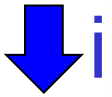
Algoritmo em C *like*

```
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;
}
```



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

i

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

Exercício

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

1	2	3	4	5	6

6	5	4	3	2	1

Exercício

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

1	2	3	4	5	6

1 ↔ 2

2 ↔ 3

3 ↔ 4

4 ↔ 5

5 ↔ 6

Exercício

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

6	5	4	3	2	1

$6 \leftrightarrow 5$

5	6	4	3	2	1

$6 \leftrightarrow 4$

$5 \leftrightarrow 4$

4	5	6	3	2	1

. . .

Análise do Número de Comparações

- Na i-ésima interação do anel interno, tem-se que:
  - melhor caso: $\boxed{C_i = 1}$
  - pior caso: $\boxed{C_i = i - 1}$
  - caso médio: $\boxed{C_i = \frac{(i-1)+(1)}{2} = \frac{i}{2}}$ (considera-se que todas as permutações de n são igualmente prováveis)

Análise do Número de Comparações

- Como o anel interno é realizado *(n – 1)* vezes, tem-se que:
  - melhor caso: $$C(n) = (1 + 1 + ... + 1) = n - 1$$
  - pior caso: $$C(n) = (1 + 2 + ... + (n - 1)) = \frac{(n-1)n}{2}$$
  - caso médio: $$C(n) = \frac{1}{2}(2 + 3 + ... + n) = \frac{n^2}{4} + \frac{n}{4} - \frac{1}{2}$$

Análise do Número de Movimentações

- O número de movimentações na i-ésima interação é igual a:

$$M_i(n) = C_i(n) - 1 + 2 = C_i(n) + 1$$

- Logo, o número de movimentos é igual a:
  - melhor caso: $M(n) = (2 + 2 + ... + 2) = 2(n - 1)$
  - pior caso: $M(n) = (3 + 4 + ... + (n + 1)) = \frac{n^2}{2} + \frac{3n}{2} - 2$
  - caso médio: $M(n) = \frac{1}{2}(4 + 5 + ... + (n + 2)) = \frac{n^2}{4} + \frac{5n}{4} - \frac{3}{2}$

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

- 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

- 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 O (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.