



Estruturas de Dados

Algoritmos de Ordenação

Prof. Thiago Caproni Tavares ¹ Paulo Muniz de Ávila ²

¹thiago.tavares@ifsuldeminas.edu.br

²paulo.avila@ifsuldeminas.edu.br

- 1 Introdução**
- 2 Algoritmos de Ordenação Bubble Sort**
- 3 Algoritmo de Ordenação Insert Sort**
- 4 Algoritmo de Ordenação Selection Sort**
- 5 Algoritmo de Ordenação Merge Sort**
- 6 Algoritmo de Ordenação Quick Sort**
- 7 Ordenando outras estruturas de dados**

- 1** Introdução
- 2** Algoritmos de Ordenação Bubble Sort
- 3** Algoritmo de Ordenação Insert Sort
- 4** Algoritmo de Ordenação Selection Sort
- 5** Algoritmo de Ordenação Merge Sort
- 6** Algoritmo de Ordenação Quick Sort
- 7** Ordenando outras estruturas de dados

*"No mundo da computação, talvez as tarefas mais fundamentais e extensivamente analisadas sejam **ordenação** e **pesquisa**."*

- Ordenação é o processo de arranjar um conjunto de informações semelhantes em uma ordem crescente ou descrente.
- Existem duas categorias gerais de algoritmos de ordenação:
algoritmos que ordenam matrizes (tanto em memória como em arquivos de acesso aleatório em disco) e *algoritmos que ordenam arquivos sequenciais em disco*.

Tipos de Algoritmos de Ordenação

- Existem três métodos gerais para ordenar matrizes:
 - por troca;
 - por seleção;
 - por inserção.
- Para entender esses três métodos, imagine as cartas de um baralho. Para ordenar as cartas, utilizando *troca*, espalhe as cartas e troque as cartas fora de ordem até que todo o baralho esteja ordenado.
- Utilizando *seleção* espalhe as cartas na mesa, selecione a carta de menor valor, retire-a do baralho e segure-a na sua mão. Esse processo continua até que todas as cartas estejam ordenadas na sua mão.
- Utilizando *inserção*, segure todas as cartas em sua mão. Ponha uma carta por vez na mesa, sempre inserindo-a na posição correta.

- Existem muitos algoritmos diferentes para cada método de ordenação. Cada um deles tem seus méritos, mas os critérios gerais para avaliação de um algoritmo são:
 - Em que velocidade ele pode ordenar as informações no caso médio ?
 - Qual a velocidade do seu melhor e pior casos ?
 - Esse algoritmo apresenta um comportamento natural ou não-natural?
 - Ele rearranja elementos de chaves iguais ?

Vamos analisar os critérios:

- É evidente que a velocidade em que um algoritmo particular ordena é de grande importância. A velocidade em que uma matriz pode ser classificada está diretamente relacionada com o número de comparações e o número de trocas que ocorrem, com as trocas exigindo mais tempo.
- Espera-se que um bom algoritmo tenha um bom desempenho considerando o caso médio, mas terrível no pior caso.

Vamos analisar os critérios:

- Um algoritmo de ordenação tem *comportamento natural* quando uma matriz (lista) trabalha o mínimo quando já está ordenada, trabalha mais quanto mais desorganizada estiver a lista e o maior tempo quando a lista está em ordem inversa.
- O último item diz respeito a implementação do algoritmo de ordenação. As operações de troca são as que demandam mais tempo. O algoritmo considera isso, ou seja, ele troca os elementos que possuem chaves iguais ? Vamos ver que existem algoritmos que fazem isso e outros que não.

- 1** Introdução
- 2** Algoritmos de Ordenação Bubble Sort
- 3** Algoritmo de Ordenação Insert Sort
- 4** Algoritmo de Ordenação Selection Sort
- 5** Algoritmo de Ordenação Merge Sort
- 6** Algoritmo de Ordenação Quick Sort
- 7** Ordenando outras estruturas de dados

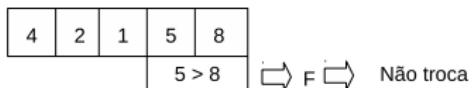
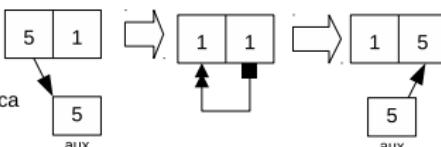
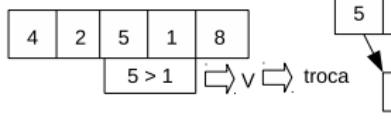
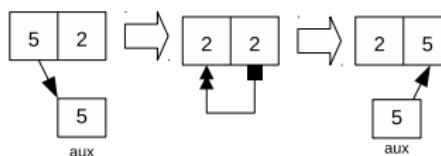
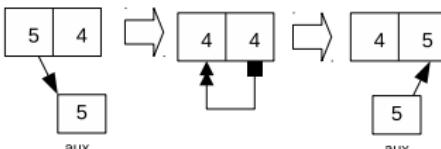
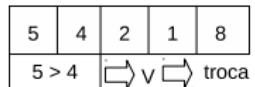
- O algoritmo **Bubble Sort** ou **Ordenação Bolha** é um algoritmo que utiliza o método por *troca*.
- Ele envolve repetidas comparações e, se necessário, a troca de dois elementos adjacentes. Os elementos são como bolhas em um tanque de água - cada uma procura o seu próprio nível. Uma implementação possível é apresentada no próximo slide.

Algoritmo Bubble Sort

```
void bubbleV1(int *item, int numElemento) {  
  
    int n, i, aux;  
  
    for(n=1; n<=numElemento; n++) {  
        for(i=0; i<numElemento-1; i++)  
        {  
            if(item[i] > item[i+1])  
            {  
                aux = item[i];  
                item[i] = item[i+1];  
                item[i+1] = aux;  
            }  
        }  
    }  
}
```

Algoritmo Bubble Sort - Simulação

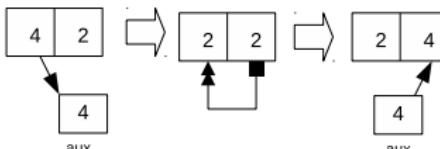
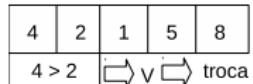
Primeira Execução do laço



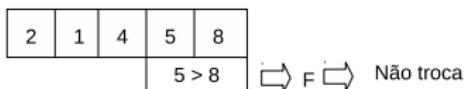
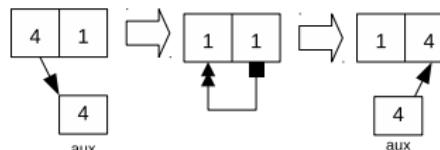
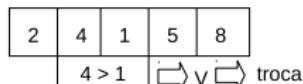
```
for(n=1; n<=numElemento;n++)  
{  
    for(i=0; i<numElemento-1;i++)  
    {  
        if(item[i] > item[i+1])  
        {  
            aux = item[i];  
            item[i] = item[i+1];  
            item[i+1] = aux;  
        }  
    }  
}
```

Algoritmo Bubble Sort - Simulação

Segunda Execução do laço

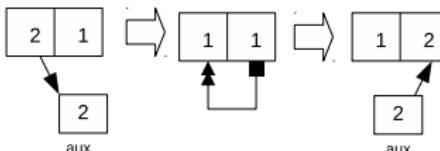


```
for(n=1; n<=numElemento; n++){  
    for(i=0; i<numElemento-1; i++)  
    {  
        if(item[i] > item[i+1])  
        {  
            aux = item[i];  
            item[i] = item[i+1];  
            item[i+1] = aux;  
        }  
    }  
}
```



Terceira Execução do laço

2	1	4	5	8
2 > 1	V	troca		



1	2	4	5	8
2 > 4	F	Não troca		

```
for(n=1; n<=numElemento; n++){  
    for(i=0; i<numElemento-1; i++){  
        if(item[i] > item[i+1])  
        {  
            aux = item[i];  
            item[i] = item[i+1];  
            item[i+1] = aux;  
        }  
    }  
}
```

1	2	4	5	8
4 > 5	F	Não troca		

1	2	4	5	8
5 > 8	F	Não troca		

Quarta Execução do laço

1	2	3	5	8
1 > 2	F	Não troca		

1	2	4	5	8
2 > 4	F	Não troca		

1	2	4	5	8
4 > 5	F	Não troca		

1	2	4	5	8
5 > 8	F	Não troca		

```
for(n=1; n<=numElemento;n++){  
    for(i=0; i<numElemento-1;i++){  
        {  
            if(item[i] > item[i+1])  
            {  
                aux = item[i];  
                item[i] = item[i+1];  
                item[i+1] = aux;  
            }  
        }  
    }  
}
```

Quinta Execução do laço

1	2	3	5	8
1 > 2	F	Não troca		

1	2	4	5	8
2 > 4	F	Não troca		

1	2	4	5	8
4 > 5	F	Não troca		

1	2	4	5	8
5 > 8	F	Não troca		

```
for(n=1; n<=numElemento;n++){  
    for(i=0; i<numElemento-1;i++){  
        {  
            if(item[i] > item[i+1])  
            {  
                aux = item[i];  
                item[i] = item[i+1];  
                item[i+1] = aux;  
            }  
        }  
    }  
}
```

- Em um algoritmo de ordenação o fator relevante que determina seu tempo de execução é o número de comparações realizadas. Considerando que o algoritmo foi implementado para um vetor de 5 posições. Verifica-se que o número de interações no primeiro laço é 5. O segundo laço possui 4 interações, mas como está interno a primeiro será executado $5 \times 4 = 20$ vezes.
- Aplicando a mesma ideia sobre o algoritmo com um vetor de tamanho n , ele realizará $n(n - 1) = n^2 - n$ comparações.
- Dessa forma, pode-se dizer que o tempo de execução do algoritmo **bubble sort** é $O(n^2)$, pois:

$$n^2 - n \leq cn^2, c = 1, n \geq 1.$$

- Nesse algoritmo de ordenação serão efetuadas comparações entre os dados armazenados em um vetor de tamanho n . Cada elemento de posição i será comparado com o de posição $i+1$ e, quando a ordenação que se busca (crescente ou decrescente) é encontrada, uma troca de posição entre os dados é feita.
- Nesse próximo exemplo iremos executar uma ordenação **decrescente**. Observe o sinal de maior e menor no *if* de comparação do algoritmo no próximo slide.
- Os sinais $>$ e $<$ na estrutura *if* controla se a ordenação será por ordem **crescente** ou **decrescente**.

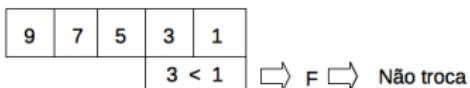
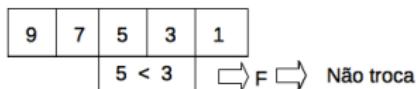
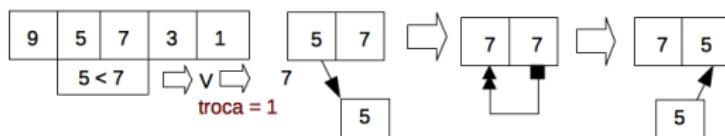
Algoritmo Bubble Sort Melhorado

```
void bubbleMelhorado(char *item, int count){  
    int n=1,i,troca=1,aux;  
  
    while(n<=count && troca == 1){  
        troca = 0;  
        for(i=0;i<count-1;i++){  
            if(item[i] < item[i+1]){  
                troca = 1;  
                aux = item[i];  
                item[i] = item[i+1];  
                item[i+1] = aux;  
            }  
        }  
        n=n+1;  
    }  
}
```

Algoritmo Bubble Sort Melhorado - Simulação

Primeira Execução do laço

Inicio → troca = 0

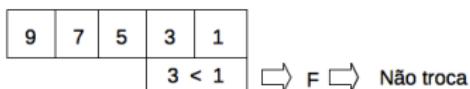
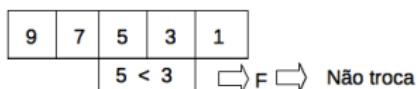
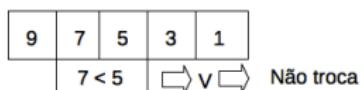


```
void bubbleMelhorado(char *item,int count){  
    int n=1,i;  
    int troca=1;  
    int aux;  
  
    while(n<=count && troca == 1){  
        troca = 0;  
        for(i=0;i<count-1;i++){  
            if(item[i] < item[i+1]){  
                troca = 1;  
                aux = item[i];  
                item[i] = item[i+1];  
                item[i+1] = aux;  
            }  
        }  
        n=n+1;  
    }  
}
```

Fim → troca = 1 → DEVE CONTINUAR

Segunda Execução do Laço

Inicio → troca = 0



```
void bubbleMelhorado(char *item,int count){  
  
    int n=1,i;  
    int troca=1;  
    int aux;  
  
    while(n<=count && troca == 1){  
        troca = 0;  
        for(i=0;i<count-1;i++){  
            if(item[i] < item[i+1]){  
                troca = 1;  
                aux = item[i];  
                item[i] = item[i+1];  
                item[i+1] = aux;  
            }  
        }  
        n=n+1;  
    }  
}
```

Fim → troca = 0 → VETOR ORDENADO, SAI DO LAÇO

- Nessa versão melhorada é possível observamos a presença de melhor e pior caso.
- O **melhor** caso ocorre quando o vetor submetido a ordenação está ordenado. Nessa caso nenhuma troca será realizada e a estrutura *while* será executada apenas **uma** vez. Considerando que o tamanho da entrada como **n**, o número de comparações realizadas será $n - 1$ (laço *for*). Portanto, no melhor caso, esse algoritmo gasta $O(n)$.
- O **pior** caso ocorre quando o vetor está na ordem inversa. Nesse caso, a estrutura *while* será executa n vezes. A estrutura *for* será executada $n - 1$ vezes. Logo, no pior caso, esse algoritmo gasta $n(n - 1) = n^2 - n$, ou seja, $O(n^2)$.

- 1 Introdução
- 2 Algoritmos de Ordenação Bubble Sort
- 3 Algoritmo de Ordenação Insert Sort
- 4 Algoritmo de Ordenação Selection Sort
- 5 Algoritmo de Ordenação Merge Sort
- 6 Algoritmo de Ordenação Quick Sort
- 7 Ordenando outras estruturas de dados

- Nesse algoritmo de ordenação será inicialmente eleito o segundo elemento do vetor para iniciar as comparações.
- A cada laço do algoritmo o elemento eleito é deslocado para direita.
- O objetivo é manter os elementos à esquerda do número eleito sempre ordenados.
- Esse algoritmo possui **pior** e **melhor** caso, dependendo da maneira como os dados do vetor de entrada estão organizados.
- Vamos analisar o algoritmo em detalhes nos próximos slides.

Primeira Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória					
$*\text{item} =$			5	8	2
$\text{count} =$			1	8	5
$i =$?	?	?
$j =$?	?	?
$\text{eleito} =$?	?	?

Primeira Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória					
$*\text{item} =$			5	8	2
$\text{count} =$			5		
$i =$			1	$j =$	
$\text{eleito} =$?	$?$	

Primeira Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória					
<code>*item =</code>			<code>count =</code>		
<input type="text"/>	<input type="text"/>				
<code>i =</code>	<input type="text"/>	<code>j =</code>	<input type="text"/>	<code>eleito =</code>	<input type="text"/>
	<input type="text"/>		<input type="text"/>		<input type="text"/>

Primeira Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table><tr><td>5</td><td>8</td><td>2</td><td>1</td><td>8</td></tr></table>	5	8	2	1	8
5	8	2	1	8		
count =	<input type="text"/>					
i =	<input type="text"/> 1					
j =	<input type="text"/> 0					
eleito =	<input type="text"/> 8					

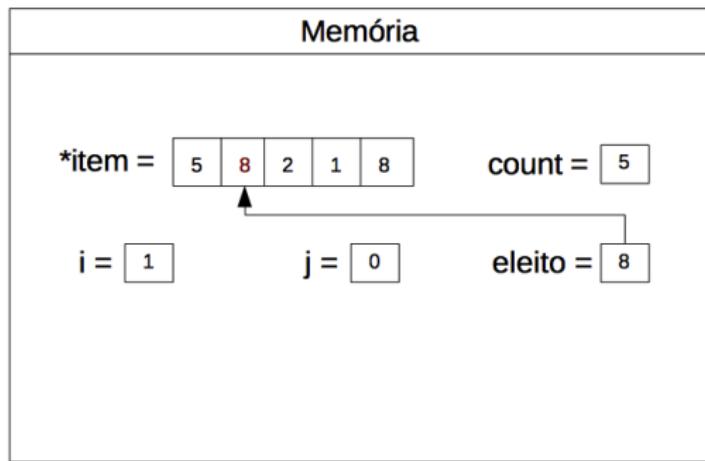
Primeira Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
        0 >= 0 e 5>8 → F  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>5</td><td>8</td><td>2</td><td>1</td><td>8</td></tr></table>	5	8	2	1	8
5	8	2	1	8		
count =	<input type="text"/>					
i =	<input type="text"/> 1					
j =	<input type="text"/> 0					
eleito =	<input type="text"/> 8					

Primeira Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while( j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```



Segunda Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória					
$*\text{item} =$			5	8	2
$\text{count} =$			5		
$\text{i} =$			2	0	
$\text{j} =$					8
$\text{eleito} =$					

Segunda Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória					
<code>*item =</code>			<code>count =</code>		
<input type="text"/>	<input type="text"/>				
<code>i =</code>	<input type="text"/>	<code>j =</code>	<input type="text"/>	<code>eleito =</code>	<input type="text"/>
	<input type="text"/>		<input type="text"/>		<input type="text"/>

Segunda Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória					
<code>*item =</code>			<code>count =</code>		
<code>5 8 2 1 8</code>			<code>5</code>		
<code>i =</code>		<code>2</code>	<code>j =</code>		<code>1</code>
<code>eleito =</code>		<code>2</code>			

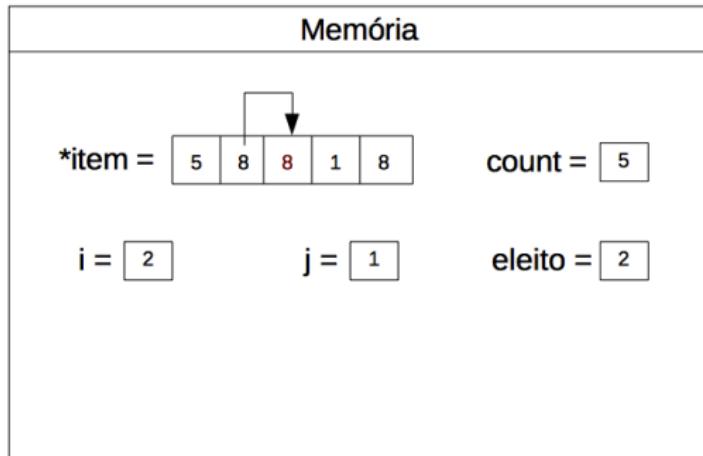
Segunda Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
        1 >= 0 e 8>2 → V  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória					
<code>*item =</code>			<code>count =</code> <input type="text" value="5"/>		
<code>i =</code> <input type="text" value="2"/>			<code>j =</code> <input type="text" value="1"/>		
<code>eleito =</code> <input type="text" value="2"/>					

Segunda Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```



Segunda Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória					
<code>*item =</code>			<code>count =</code>		
<code>5 8 8 1 8</code>			<code>5</code>		
<code>i =</code>		<code>2</code>	<code>j =</code>		<code>0</code>
<code>eleito =</code>		<code>2</code>			

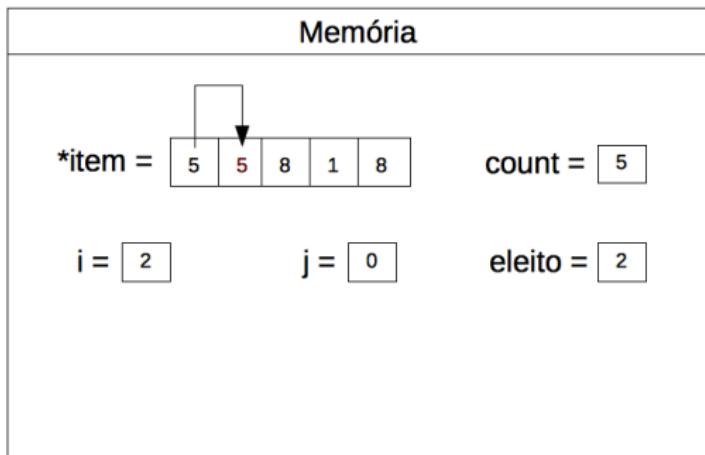
Segunda Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
        0 >= 0 e 5>2 → V  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória					
<code>*item =</code>			<code>count =</code>		
<input type="text"/>	<input type="text"/>				
<code>i =</code>	<input type="text"/>	<code>j =</code>	<input type="text"/>	<code>eleito =</code>	<input type="text"/>
	<input type="text"/>		<input type="text"/>		<input type="text"/>

Segunda Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```



Segunda Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória								
*item =	<table border="1"><tr><td>5</td><td>5</td><td>8</td><td>1</td><td>8</td></tr></table>	5	5	8	1	8	count =	<input type="text"/>
5	5	8	1	8				
i =	<input type="text"/> 2	j =	<input type="text"/> -1					
eleito =	<input type="text"/> 2							

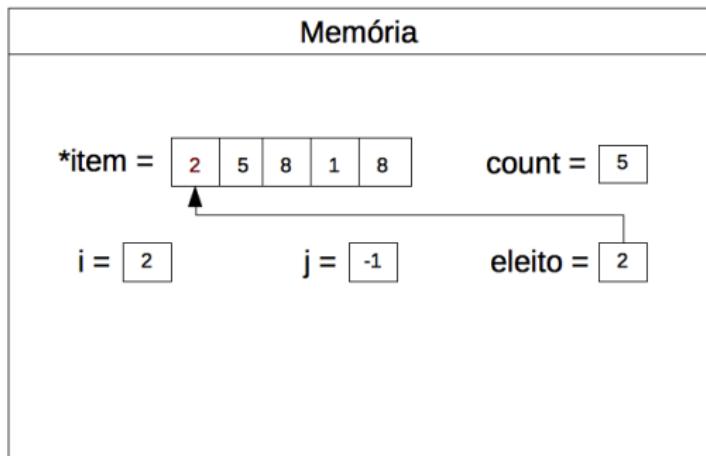
Segunda Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
        -1 >= 0 → F  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>5</td><td>5</td><td>8</td><td>1</td><td>8</td></tr></table>	5	5	8	1	8
5	5	8	1	8		
count =	<input type="text"/>					
i =	<input type="text"/> 2					
j =	<input type="text"/> -1					
eleito =	<input type="text"/> 2					

Segunda Execução do Laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```



Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>2</td><td>5</td><td>8</td><td>1</td><td>8</td></tr></table>	2	5	8	1	8
2	5	8	1	8		
count =	<input type="text" value="5"/>					
i =	<input type="text" value="3"/>					
j =	<input type="text" value="-1"/>					
eleito =	<input type="text" value="2"/>					

Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>2</td><td>5</td><td>8</td><td>1</td><td>8</td></tr></table>	2	5	8	1	8
2	5	8	1	8		
count =	<input type="text" value="5"/>					
i =	<input type="text" value="3"/>					
j =	<input type="text" value="-1"/>					
eleito =	<input type="text" value="1"/>					

Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória					
$*\text{item} =$			2	5	8
$\text{count} =$			5		
$\text{i} =$			3	$\text{j} =$	
$\text{eleito} =$			1		

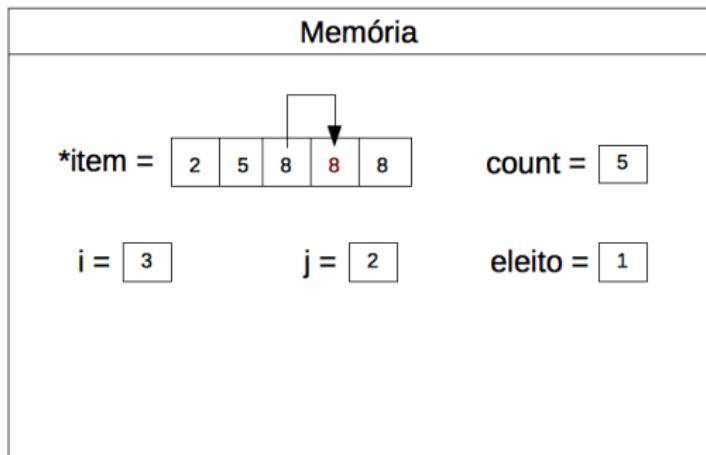
Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
        2 >= 0 e 8>1 → V  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>2</td><td>5</td><td>8</td><td>1</td><td>8</td></tr></table>	2	5	8	1	8
2	5	8	1	8		
count =	<input type="text"/>					
i =	<input type="text"/> 3					
j =	<input type="text"/> 2					
eleito =	<input type="text"/> 1					

Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```



Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>2</td><td>5</td><td>8</td><td>8</td><td>8</td></tr></table>	2	5	8	8	8
2	5	8	8	8		
count =	<input type="text"/>					
i =	<input type="text"/> 3					
j =	<input type="text"/> 1					
eleito =	<input type="text"/> 1					

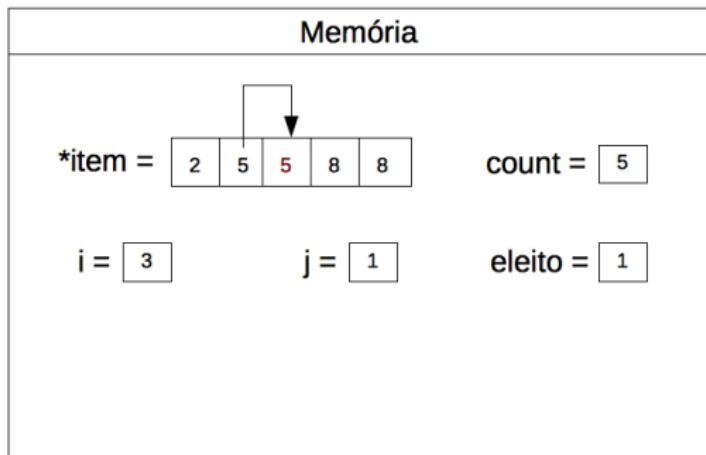
Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
        1 >= 0 e 5>1 → V  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>2</td><td>5</td><td>8</td><td>8</td><td>8</td></tr></table>	2	5	8	8	8
2	5	8	8	8		
count =	<input type="text"/>					
i =	<input type="text"/> 3					
j =	<input type="text"/> 1					
eleito =	<input type="text"/> 1					

Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```



Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória							
*item =	<table border="1"><tr><td>2</td><td>5</td><td>5</td><td>8</td><td>8</td></tr></table>	2	5	5	8	8	count = <input type="text" value="5"/>
2	5	5	8	8			
i =	<input type="text" value="3"/>	j = <input type="text" value="0"/> eleito = <input type="text" value="1"/>					

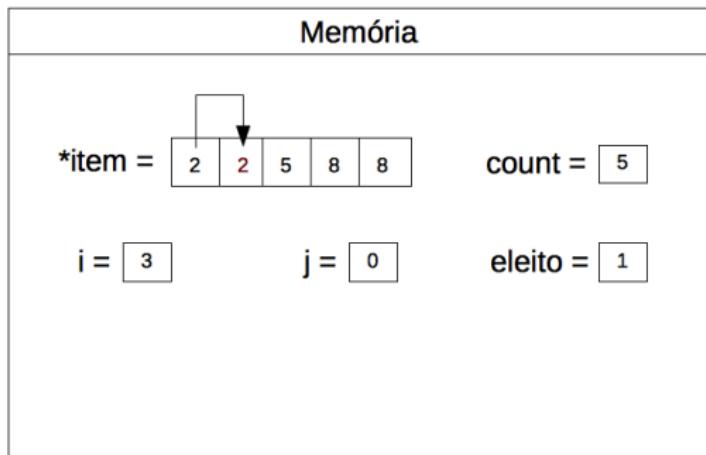
Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
        0 >= 0 e 2>1 → V  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>2</td><td>5</td><td>5</td><td>8</td><td>8</td></tr></table>	2	5	5	8	8
2	5	5	8	8		
count =	<input type="text"/>					
i =	<input type="text"/> 3					
j =	<input type="text"/> 0					
eleito =	<input type="text"/> 1					

Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```



Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória							
*item =	<table border="1"><tr><td>2</td><td>2</td><td>5</td><td>8</td><td>8</td></tr></table>	2	2	5	8	8	count = <input type="text"/>
2	2	5	8	8			
i =	<input type="text"/> 3	j = <input type="text"/> -1					
eleito =	<input type="text"/> 1						

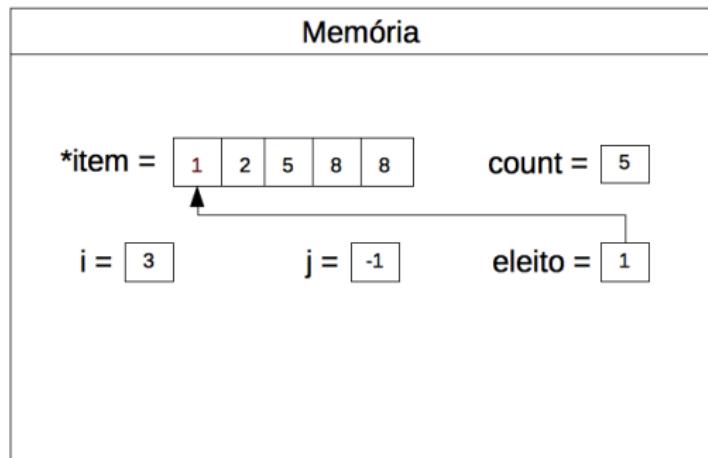
Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
        -1 >= 0 → F  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>2</td><td>2</td><td>5</td><td>8</td><td>8</td></tr></table>	2	2	5	8	8
2	2	5	8	8		
count =	<input type="text"/>					
i =	<input type="text"/> 3					
j =	<input type="text"/> -1					
eleito =	<input type="text"/> 1					

Terceira Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```



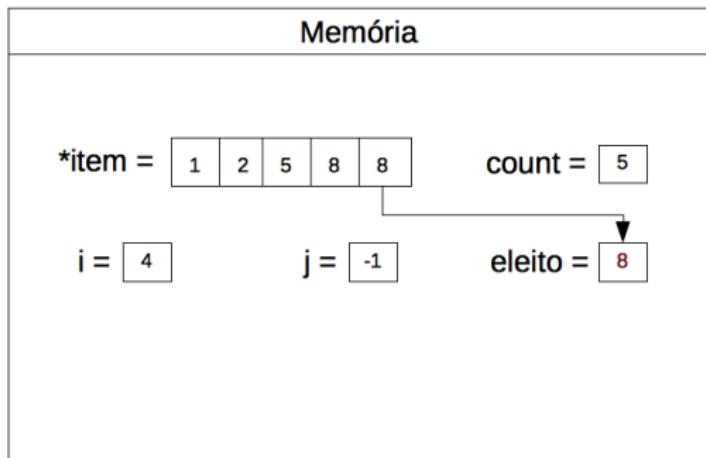
Quarta Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>2</td><td>5</td><td>8</td><td>8</td></tr></table>	1	2	5	8	8
1	2	5	8	8		
count =	<input type="text"/>					
i =	<input type="text" value="4"/>					
j =	<input type="text" value="-1"/>					
eleito =	<input type="text" value="1"/>					

Quarta Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```



Quarta Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>2</td><td>5</td><td>8</td><td>8</td></tr></table>	1	2	5	8	8
1	2	5	8	8		
count =	<input type="text"/>					
i =	<input type="text"/> 4					
j =	<input type="text"/> 3					
eleito =	<input type="text"/> 8					

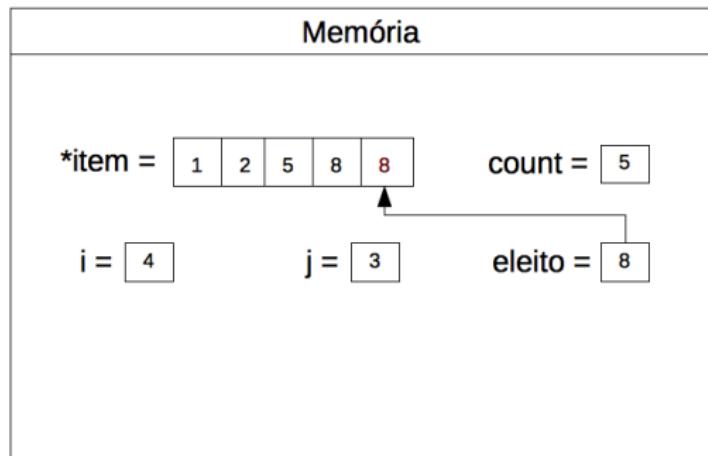
Quarta Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
        3 >= 0 e 8 > 8 → F  
        while( j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
        item[j+1] = eleito;  
    }  
}
```

Memória								
*item =	<table border="1"><tr><td>1</td><td>2</td><td>5</td><td>8</td><td>8</td></tr></table>	1	2	5	8	8	count =	<input type="text"/>
1	2	5	8	8				
i =	<input type="text"/> 4	j =	<input type="text"/> 3					
eleito =	<input type="text"/> 8							

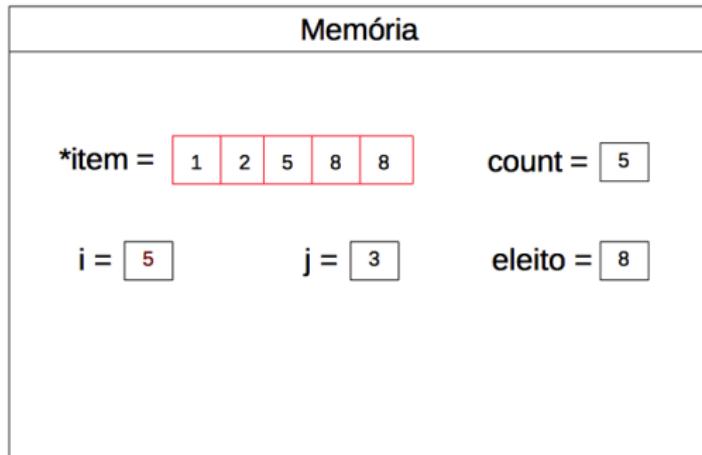
Quarta Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```



Quarta Execução do laço

```
void insertionSort(int *item, int count){  
    int i,j,eleito;  
    5 <= 4 → F  
    for(i=1; i<=count-1;i++){  
        eleito = item[i];  
        j = i - 1;  
  
        while(j >= 0 && item[j] > eleito){  
            item[j+1] = item[j];  
            j = j - 1;  
        }  
  
        item[j+1] = eleito;  
    }  
}
```



- No algoritmo anterior existem duas estruturas de repetição. A estrutura interna *for* executa $n - 1$ vezes, onde n é o tamanho do vetor.
- A estrutura de repetição *while* será executada a princípio também $n - 1$ vezes, porém, dependendo do vetor de entrada, essa estrutura pode ser executada menos vezes.
- O pior caso do algoritmo ocorre quando o vetor de entrada possui os elementos na *ordem inversa*, ou seja, o vetor está em ordem decrescente e busca-se a ordenação crescente.
- Nesse caso, cada elemento do vetor $X[j]$ é comparado com cada elemento no subvetor $X[0..j-1]$, e então executa-se a linha de comparação **while(j >= 0 && X[j] > eleito)**, $j + 2$ vezes para cada valor de $j = 0, 1, \dots, n - 2$.

- O número de comparações é dado pela fórmula a seguir:

$$T(n) = 2 + 3 + 4 + \dots + n$$

$$T(n) = \left(\sum_{i=1}^n i \right) - 1$$

$$T(n) = \frac{(1+n)n}{2} - 1$$

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

$$T(n) = O(n^2)$$

- O melhor caso do algoritmo ocorre quando o vetor de entrada possui os elementos já ordenados.
- Nesse caso, para cada $j = 0, 1, 2, \dots, n - 2$, tem-se que a condição $X[j] > \text{eleito}$, no laço *while* é **falsa**.
- O custo de execução do melhor caso é $T(n) = O(n - 1)$, já que se tem $n - 1$ valores para j .

- 1 Introdução
- 2 Algoritmos de Ordenação Bubble Sort
- 3 Algoritmo de Ordenação Insert Sort
- 4 Algoritmo de Ordenação Selection Sort
- 5 Algoritmo de Ordenação Merge Sort
- 6 Algoritmo de Ordenação Quick Sort
- 7 Ordenando outras estruturas de dados

- Nesse método de ordenação cada elemento do vetor, a partir do primeiro, será comparado com o menor ou maior, dependendo da ordenação desejada, número dentre aqueles que estão à direita do elemento eleito.
- O elemento eleito é sempre deslocado para direita e todos os elementos à esquerda do eleito ficam sempre ordenados.
- O elemento eleito está na posição i . Os elementos à direita do eleito estão nas posições $i + 1$ à $n - 1$, sendo n o número de elementos do vetor. Todos os elementos à esquerda, ou seja, $i - 1$ à 0 estarão ordenados.

Algoritmo Selection Sort

```
void selectionSort( int *item , int count){  
    int i , eleito , j , menor , pos ;  
    for( i=0; i<count-1; i++ ){  
        eleito = item[ i ];  
        menor = item[ i+1 ];  
        pos = i + 1;  
  
        for( j=i+1; j<count ; j++ ){  
            if( item[ j ] < menor ){  
                menor = item[ j ];  
                pos = j ;  
            }  
        }  
        if( menor < eleito ){  
            item[ i ] = item[ pos ];  
            item[ pos ] = eleito ;  
        }  
    }  
}
```

Primeira Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória								
*item =	<table border="1"><tr><td>5</td><td>8</td><td>2</td><td>1</td><td>8</td></tr></table>	5	8	2	1	8	count = <table border="1"><tr><td>5</td></tr></table>	5
5	8	2	1	8				
5								
i =	<table border="1"><tr><td>?</td></tr></table>	?	j = <table border="1"><tr><td>?</td></tr></table> eleito = <table border="1"><tr><td>?</td></tr></table>	?	?			
?								
?								
?								
menor = <table border="1"><tr><td>?</td></tr></table>	?	pos = <table border="1"><tr><td>?</td></tr></table>	?					
?								
?								

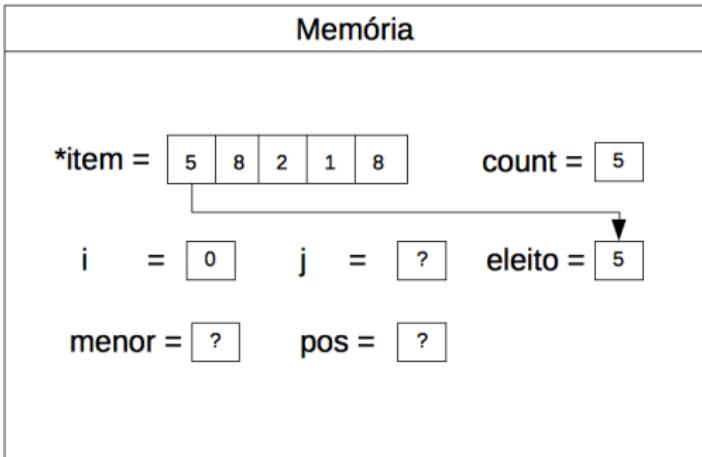
Primeira Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>5</td><td>8</td><td>2</td><td>1</td><td>8</td></tr></table>	5	8	2	1	8
5	8	2	1	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>0</td></tr></table>	0				
0						
j =	<table border="1"><tr><td>?</td></tr></table>	?				
?						
eleito =	<table border="1"><tr><td>?</td></tr></table>	?				
?						
menor =	<table border="1"><tr><td>?</td></tr></table>	?				
?						
pos =	<table border="1"><tr><td>?</td></tr></table>	?				
?						

Primeira Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



Primeira Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória

*item =

5	8	2	1	8
---	---	---	---	---

 count =

5

i =

0

 j =

?

 eleito =

5

menor =

8

 pos =

?

Primeira Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>5</td><td>8</td><td>2</td><td>1</td><td>8</td></tr></table>	5	8	2	1	8
5	8	2	1	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>0</td></tr></table>	0				
0						
j =	<table border="1"><tr><td>?</td></tr></table>	?				
?						
eleito =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
menor =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
pos =	<table border="1"><tr><td>1</td></tr></table>	1				
1						

Primeira Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória

*item =

5	8	2	1	8
---	---	---	---	---

 count =

5

i =

0

 j =

1

 eleito =

5

menor =

8

 pos =

1

Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){ 8 < 8 → F  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória								
*item =	<table border="1"><tr><td>5</td><td>8</td><td>2</td><td>1</td><td>8</td></tr></table>	5	8	2	1	8	count = <table border="1"><tr><td>5</td></tr></table>	5
5	8	2	1	8				
5								
i =	<table border="1"><tr><td>0</td></tr></table>	0	j = <table border="1"><tr><td>1</td></tr></table> eleito = <table border="1"><tr><td>5</td></tr></table>	1	5			
0								
1								
5								
menor =	<table border="1"><tr><td>8</td></tr></table>	8	pos = <table border="1"><tr><td>1</td></tr></table>	1				
8								
1								

Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>5</td><td>8</td><td>2</td><td>1</td><td>8</td></tr></table>	5	8	2	1	8
5	8	2	1	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>0</td></tr></table>	0				
0						
j =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
eleito =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
menor =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
pos =	<table border="1"><tr><td>1</td></tr></table>	1				
1						

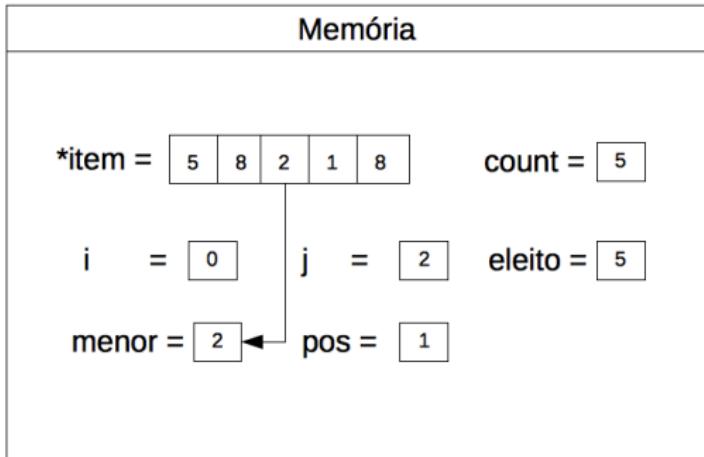
Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){ 2 < 8 → V  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>5</td><td>8</td><td>2</td><td>1</td><td>8</td></tr></table>	5	8	2	1	8
5	8	2	1	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>0</td></tr></table>	0				
0						
j =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
eleito =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
menor =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
pos =	<table border="1"><tr><td>1</td></tr></table>	1				
1						

Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória			
*item =	5	8	2
1	8		count = 5
i =	0	j =	2
eleito = 5			
menor = 2		pos = 2	

Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória			
*item =	5	8	2
1	8		count = 5
i =	0	j =	3
eleito = 5			
menor =	2	pos =	2

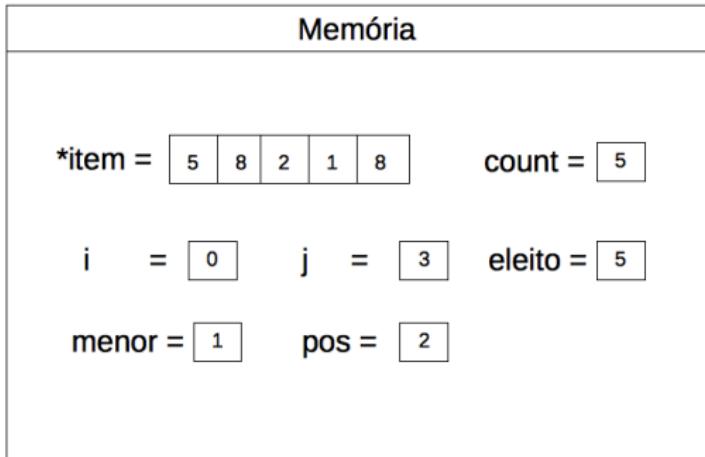
Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){ 1 < 2 → V  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória			
*item =	5	8	2
1	2	3	4
5	6	7	8
count =	5		
i =	0	j =	3
1	2	3	4
5	6	7	8
eleito =	5		
menor =	2	pos =	2

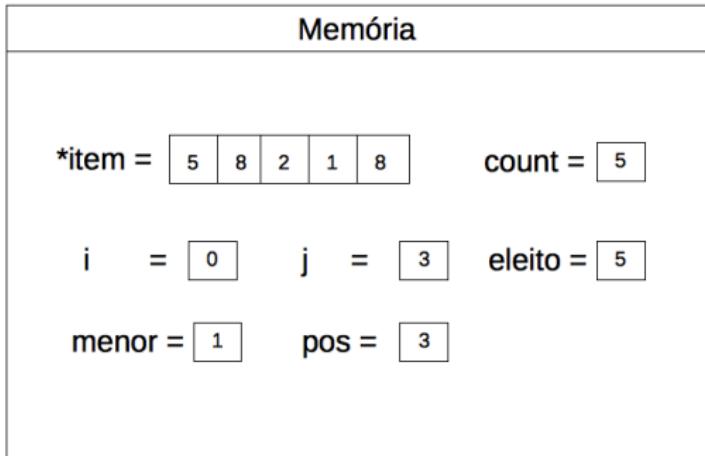
Primeira Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória					
$*item$ =			5	8	2
1	2	3	4	5	count = 5
i =	0	j =	4	eleito =	5
menor =	1	pos =	3		

Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>5</td><td>8</td><td>2</td><td>1</td><td>8</td></tr></table>	5	8	2	1	8
5	8	2	1	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>0</td></tr></table>	0				
0						
j =	<table border="1"><tr><td>4</td></tr></table>	4				
4						
eleito =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
menor =	<table border="1"><tr><td>1</td></tr></table>	1				
1						
pos =	<table border="1"><tr><td>3</td></tr></table>	3				
3						

Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
        5 < 5 → F  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>5</td><td>8</td><td>2</td><td>1</td><td>8</td></tr></table>	5	8	2	1	8
5	8	2	1	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>0</td></tr></table>	0				
0						
j =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
eleito =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
menor =	<table border="1"><tr><td>1</td></tr></table>	1				
1						
pos =	<table border="1"><tr><td>3</td></tr></table>	3				
3						

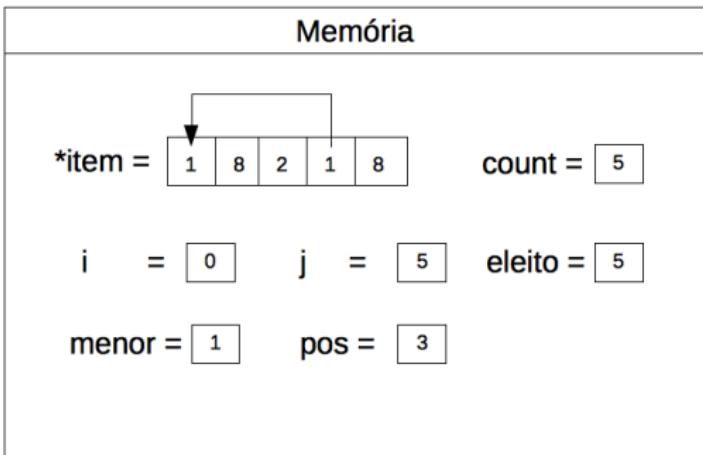
Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
        1 < 5 → V  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>5</td><td>8</td><td>2</td><td>1</td><td>8</td></tr></table>	5	8	2	1	8
5	8	2	1	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>0</td></tr></table>	0				
0						
j =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
eleito =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
menor =	<table border="1"><tr><td>1</td></tr></table>	1				
1						
pos =	<table border="1"><tr><td>3</td></tr></table>	3				
3						

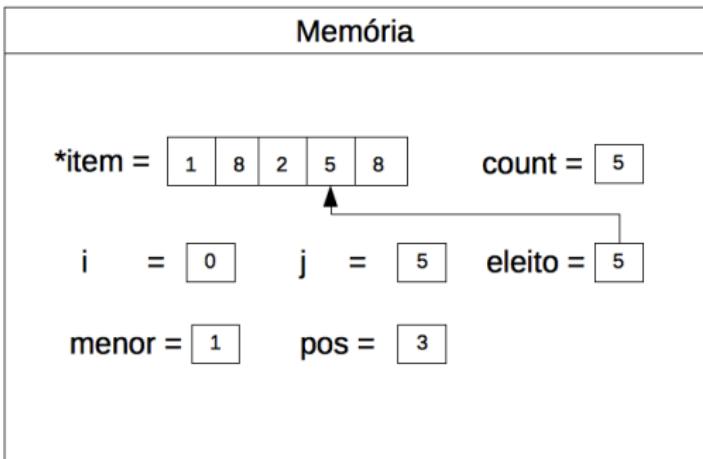
Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



Primeira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



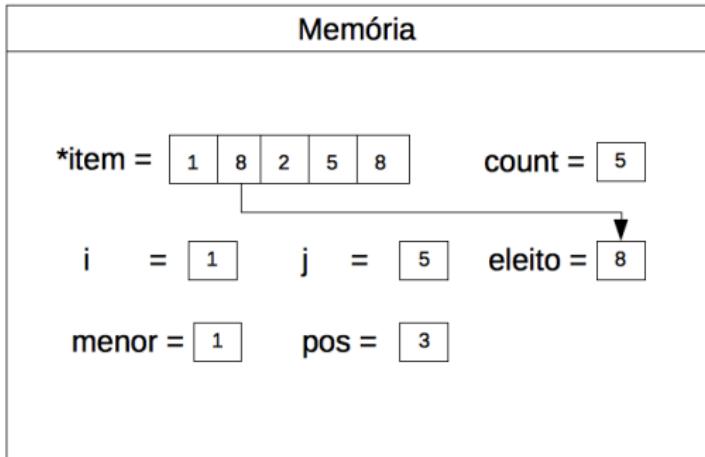
Segunda Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória								
*item =	<table border="1"><tr><td>1</td><td>8</td><td>2</td><td>5</td><td>8</td></tr></table>	1	8	2	5	8	count = <table border="1"><tr><td>5</td></tr></table>	5
1	8	2	5	8				
5								
i =	<table border="1"><tr><td>1</td></tr></table>	1	j = <table border="1"><tr><td>5</td></tr></table> eleito = <table border="1"><tr><td>5</td></tr></table>	5	5			
1								
5								
5								
menor =	<table border="1"><tr><td>1</td></tr></table>	1	pos = <table border="1"><tr><td>3</td></tr></table>	3				
1								
3								

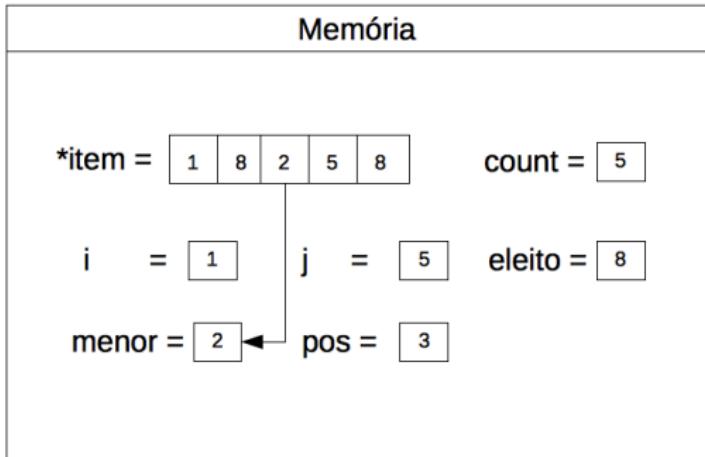
Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória									
*item =	<table border="1"><tr><td>1</td><td>8</td><td>2</td><td>5</td><td>8</td></tr></table>	1	8	2	5	8	count =	<table border="1"><tr><td>5</td></tr></table>	5
1	8	2	5	8					
5									
i =	<table border="1"><tr><td>1</td></tr></table>	1	j =	<table border="1"><tr><td>5</td></tr></table>	5				
1									
5									
eleito =	<table border="1"><tr><td>8</td></tr></table>	8	menor =	<table border="1"><tr><td>2</td></tr></table>	2				
8									
2									
pos =	<table border="1"><tr><td>2</td></tr></table>	2							
2									

Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>8</td><td>2</td><td>5</td><td>8</td></tr></table>	1	8	2	5	8
1	8	2	5	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>1</td></tr></table>	1				
1						
j =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
eleito =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
menor =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
pos =	<table border="1"><tr><td>2</td></tr></table>	2				
2						

Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){ 2 < 2 → F  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória								
*item =	<table border="1"><tr><td>1</td><td>8</td><td>2</td><td>5</td><td>8</td></tr></table>	1	8	2	5	8	count = <table border="1"><tr><td>5</td></tr></table>	5
1	8	2	5	8				
5								
i =	<table border="1"><tr><td>1</td></tr></table>	1	j = <table border="1"><tr><td>2</td></tr></table> eleito = <table border="1"><tr><td>8</td></tr></table>	2	8			
1								
2								
8								
menor =	<table border="1"><tr><td>2</td></tr></table>	2	pos = <table border="1"><tr><td>2</td></tr></table>	2				
2								
2								

Segunda Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória									
*item =	<table border="1"><tr><td>1</td><td>8</td><td>2</td><td>5</td><td>8</td></tr></table>	1	8	2	5	8	count =	<table border="1"><tr><td>5</td></tr></table>	5
1	8	2	5	8					
5									
i =	<table border="1"><tr><td>1</td></tr></table>	1	j =	<table border="1"><tr><td>3</td></tr></table>	3				
1									
3									
eleito =	<table border="1"><tr><td>8</td></tr></table>	8	menor =	<table border="1"><tr><td>2</td></tr></table>	2				
8									
2									
pos =	<table border="1"><tr><td>2</td></tr></table>	2							
2									

Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){ 5 < 2 → F  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória								
*item =	<table border="1"><tr><td>1</td><td>8</td><td>2</td><td>5</td><td>8</td></tr></table>	1	8	2	5	8	count = <table border="1"><tr><td>5</td></tr></table>	5
1	8	2	5	8				
5								
i =	<table border="1"><tr><td>1</td></tr></table>	1	j = <table border="1"><tr><td>3</td></tr></table> eleito = <table border="1"><tr><td>8</td></tr></table>	3	8			
1								
3								
8								
menor =	<table border="1"><tr><td>2</td></tr></table>	2	pos = <table border="1"><tr><td>2</td></tr></table>	2				
2								
2								

Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>8</td><td>2</td><td>5</td><td>8</td></tr></table>	1	8	2	5	8
1	8	2	5	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>1</td></tr></table>	1				
1						
j =	<table border="1"><tr><td>4</td></tr></table>	4				
4						
eleito =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
menor =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
pos =	<table border="1"><tr><td>2</td></tr></table>	2				
2						

Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){ 8 < 2 → F  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória			
*item =	1	8	2 5 8
count =	5		
i =	1	j =	4 eleito = 8
menor =	2	pos =	2

Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
        5 < 5 → F  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>8</td><td>2</td><td>5</td><td>8</td></tr></table>	1	8	2	5	8
1	8	2	5	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>1</td></tr></table>	1				
1						
j =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
eleito =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
menor =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
pos =	<table border="1"><tr><td>2</td></tr></table>	2				
2						

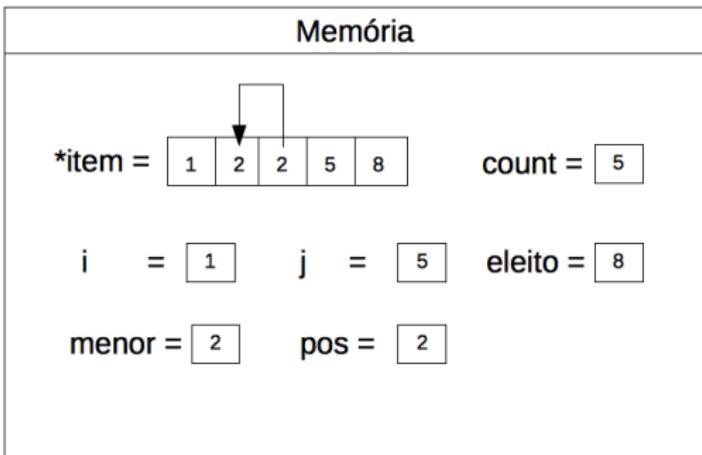
Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
        2 < 8 → V  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória								
*item =	<table border="1"><tr><td>1</td><td>8</td><td>2</td><td>5</td><td>8</td></tr></table>	1	8	2	5	8	count = <table border="1"><tr><td>5</td></tr></table>	5
1	8	2	5	8				
5								
i =	<table border="1"><tr><td>1</td></tr></table>	1	j = <table border="1"><tr><td>5</td></tr></table> eleito = <table border="1"><tr><td>8</td></tr></table>	5	8			
1								
5								
8								
menor =	<table border="1"><tr><td>2</td></tr></table>	2	pos = <table border="1"><tr><td>2</td></tr></table>	2				
2								
2								

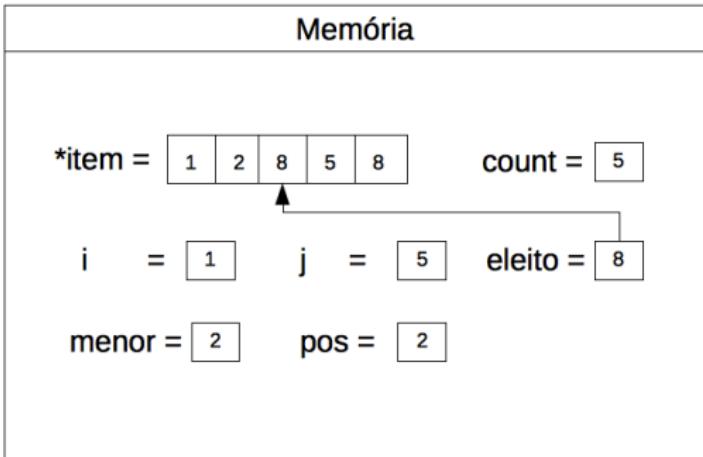
Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



Segunda Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



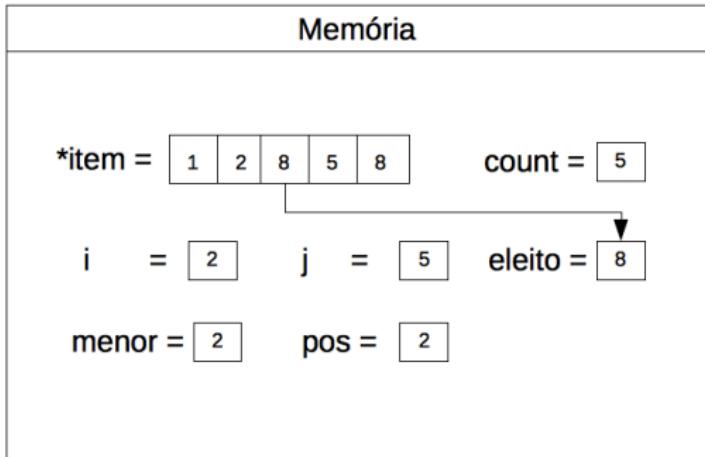
Terceira Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>2</td><td>8</td><td>5</td><td>8</td></tr></table>	1	2	8	5	8
1	2	8	5	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
j =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
eleito =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
menor =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
pos =	<table border="1"><tr><td>2</td></tr></table>	2				
2						

Terceira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



Terceira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória									
*item =	<table border="1"><tr><td>1</td><td>2</td><td>8</td><td>5</td><td>8</td></tr></table>	1	2	8	5	8	count =	<table border="1"><tr><td>5</td></tr></table>	5
1	2	8	5	8					
5									
i =	<table border="1"><tr><td>2</td></tr></table>	2	j =	<table border="1"><tr><td>5</td></tr></table>	5				
2									
5									
eleito =	<table border="1"><tr><td>8</td></tr></table>	8	menor =	<table border="1"><tr><td>5</td></tr></table>	5				
8									
5									
pos =	<table border="1"><tr><td>2</td></tr></table>	2							
2									

Terceira Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória			
*item =	1	2	8
5			
count =	5		
i =	2	j =	5
			eleito = 8
menor =	5	pos =	3

Terceira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória					
$*\text{item} =$			1	2	8
$\text{count} =$			5		
$i =$			2	$j =$	3
$\text{eleito} =$			8		
$\text{menor} =$			5	$\text{pos} =$	3

Terceira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){ 5 < 5 → F  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória								
*item =	<table border="1"><tr><td>1</td><td>2</td><td>8</td><td>5</td><td>8</td></tr></table>	1	2	8	5	8	count = <table border="1"><tr><td>5</td></tr></table>	5
1	2	8	5	8				
5								
i =	<table border="1"><tr><td>2</td></tr></table>	2	j = <table border="1"><tr><td>3</td></tr></table> eleito = <table border="1"><tr><td>8</td></tr></table>	3	8			
2								
3								
8								
menor =	<table border="1"><tr><td>5</td></tr></table>	5	pos = <table border="1"><tr><td>3</td></tr></table>	3				
5								
3								

Terceira Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>2</td><td>8</td><td>5</td><td>8</td></tr></table>	1	2	8	5	8
1	2	8	5	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
j =	<table border="1"><tr><td>4</td></tr></table>	4				
4						
eleito =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
menor =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
pos =	<table border="1"><tr><td>3</td></tr></table>	3				
3						

Terceira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){ 8 < 5 → F  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>2</td><td>8</td><td>5</td><td>8</td></tr></table>	1	2	8	5	8
1	2	8	5	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
j =	<table border="1"><tr><td>4</td></tr></table>	4				
4						
eleito =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
menor =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
pos =	<table border="1"><tr><td>3</td></tr></table>	3				
3						

Terceira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
        5 < 5 → F  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>2</td><td>8</td><td>5</td><td>8</td></tr></table>	1	2	8	5	8
1	2	8	5	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
j =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
eleito =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
menor =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
pos =	<table border="1"><tr><td>3</td></tr></table>	3				
3						

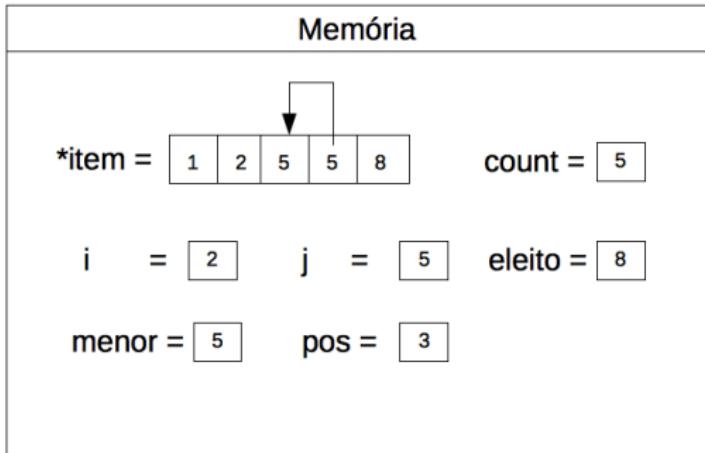
Terceira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
        5 < 8 → V  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>2</td><td>8</td><td>5</td><td>8</td></tr></table>	1	2	8	5	8
1	2	8	5	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>2</td></tr></table>	2				
2						
j =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
eleito =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
menor =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
pos =	<table border="1"><tr><td>3</td></tr></table>	3				
3						

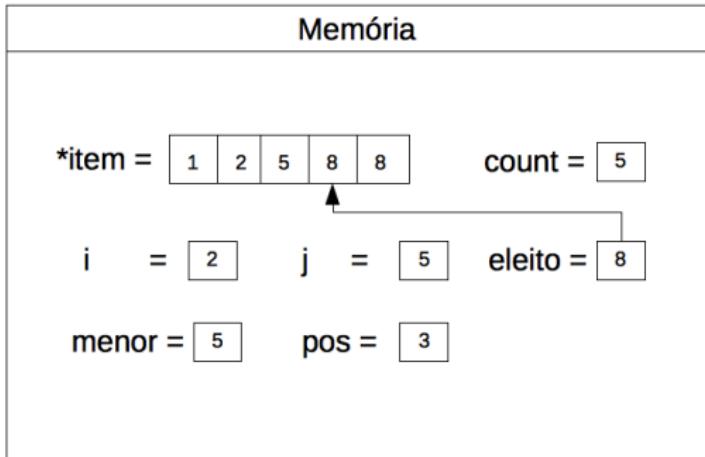
Terceira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



Terceira Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



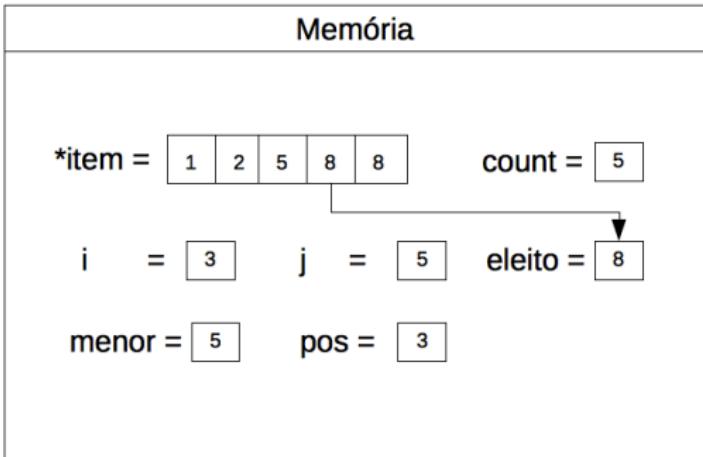
Quarta Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória								
*item =	<table border="1"><tr><td>1</td><td>2</td><td>5</td><td>8</td><td>8</td></tr></table>	1	2	5	8	8	count = <table border="1"><tr><td>5</td></tr></table>	5
1	2	5	8	8				
5								
i =	<table border="1"><tr><td>3</td></tr></table>	3	j = <table border="1"><tr><td>5</td></tr></table> eleito = <table border="1"><tr><td>8</td></tr></table>	5	8			
3								
5								
8								
menor =	<table border="1"><tr><td>5</td></tr></table>	5	pos = <table border="1"><tr><td>3</td></tr></table>	3				
5								
3								

Quarta Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```



Quarta Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória			
*item =	1	2	5 8 8
count =	5		
i =	3	j =	5 eleito = 8
menor =	8	pos =	3

Quarta Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória			
*item =	1	2	5 8 8
count =	5		
i =	3	j =	5 eleito = 8
menor =	8	pos =	4

Quarta Execução do laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>2</td><td>5</td><td>8</td><td>8</td></tr></table>	1	2	5	8	8
1	2	5	8	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>3</td></tr></table>	3				
3						
j =	<table border="1"><tr><td>4</td></tr></table>	4				
4						
eleito =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
menor =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
pos =	<table border="1"><tr><td>4</td></tr></table>	4				
4						

Quarta Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){ 8 < 8 → F  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória			
*item =	1	2	5 8 8
count =	5		
i =	3	j =	4 eleito = 8
menor =	8	pos =	4

Quarta Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
        5 < 5 → F  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>2</td><td>5</td><td>8</td><td>8</td></tr></table>	1	2	5	8	8
1	2	5	8	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>3</td></tr></table>	3				
3						
j =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
eleito =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
menor =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
pos =	<table border="1"><tr><td>4</td></tr></table>	4				
4						

Quarta Execução do laço

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
        8 < 8 → F  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória						
*item =	<table border="1"><tr><td>1</td><td>2</td><td>5</td><td>8</td><td>8</td></tr></table>	1	2	5	8	8
1	2	5	8	8		
count =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
i =	<table border="1"><tr><td>3</td></tr></table>	3				
3						
j =	<table border="1"><tr><td>5</td></tr></table>	5				
5						
eleito =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
menor =	<table border="1"><tr><td>8</td></tr></table>	8				
8						
pos =	<table border="1"><tr><td>4</td></tr></table>	4				
4						

Testa o laço

```
void selectionSort(int *item, int count){  
    int i,eleito,j,menor,pos;  
    4 < 4 → F  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória								
*item =	<table border="1"><tr><td>1</td><td>2</td><td>5</td><td>8</td><td>8</td></tr></table>	1	2	5	8	8	count = <table border="1"><tr><td>5</td></tr></table>	5
1	2	5	8	8				
5								
i =	<table border="1"><tr><td>4</td></tr></table>	4	j = <table border="1"><tr><td>5</td></tr></table> eleito = <table border="1"><tr><td>8</td></tr></table>	5	8			
4								
5								
8								
menor =	<table border="1"><tr><td>8</td></tr></table>	8	pos = <table border="1"><tr><td>4</td></tr></table>	4				
8								
4								

Algoritmo Selection Sort - Simulação

Fim

```
void selectionSort(int *item, int count){  
  
    int i,eleito,j,menor,pos;  
  
    for(i=0; i<count-1;i++){  
        eleito = item[i];  
        menor = item[i+1];  
        pos = i + 1;  
  
        for(j=i+1;j<count;j++){  
            if(item[j] < menor){  
                menor = item[j];  
                pos = j;  
            }  
        }  
  
        if(menor < eleito){  
            item[i] = item[pos];  
            item[pos] = eleito;  
        }  
    }  
}
```

Memória									
$*item =$				1	2	5	8	8	count = 5
$i =$				4	$j =$				$eleito = 8$
$menor =$				8	$pos =$				4

- Nesse algoritmo cada elemento i da primeira até a penúltima posição é trocado de lugar com o menor elemento que se encontra entre as posições $(i + 1)$ e $(n - 1)$.
- Logo, para $i = 0$ na estrutura **for** externa, a estrutura **for** interna irá executar $n - 1$ vezes. Para $i = 1$, a estrutura **for** interna executa $n - 2$ vezes, e assim por diante, até que no último valor de i a estrutura **for** interna executa apenas uma única vez.
- Podemos representar o tempo de execução como um somatório de termos:

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

- A fórmula anterior pode ser expressa por uma progressão aritmética de razão 1:

$$T(n) = \frac{(a_1 + a_n).n}{2}$$

$$T(n) = \frac{(1 + n - 1).(n - 1)}{2}$$

$$T(n) = \frac{n.(n - 1)}{2}$$

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

$$T(n) = \frac{n^2}{2} - \frac{n}{2}.$$

- Logo, o tempo de execução do algoritmo **Selection Sort** é $\Theta(n^2)$. Independente do vetor de entrada, o algoritmo se comportará da mesma maneira.
- A notação $\Theta(n^2)$ indica que o tempo de execução do algoritmo é limitado superiormente e inferiormente pela função n^2 .

- 1 Introdução
- 2 Algoritmos de Ordenação Bubble Sort
- 3 Algoritmo de Ordenação Insert Sort
- 4 Algoritmo de Ordenação Selection Sort
- 5 Algoritmo de Ordenação Merge Sort
- 6 Algoritmo de Ordenação Quick Sort
- 7 Ordenando outras estruturas de dados

- Nesse algoritmo de ordenação, o vetor é dividido em subvetores com a metade do tamanho original por meio de um procedimento recursivo. Essa divisão ocorre até que o vetor fique com apenas um elemento e estes sejam ordenados e intercalados.
- Aplica-se nesse algoritmo a técnica da **divisão e conquista**, técnica recursiva que envolve três passos em cada nível de recursão:
 - Dividir o problema em um certo número de subproblemas.
 - Conquistar os subproblemas solucionando-os recursivamente. Se os tamanhos dos subproblemas são suficientemente pequenos, então, solucionar os subproblemas de forma simples.
 - Combinar as soluções dos subproblemas na solução do problema original.

- No algoritmo **Merge Sort** tem-se a técnica da divisão e conquista aplicada da seguinte forma:
 - Dividir: dividir a sequência de n elementos a serem ordenados em duas subsequências de $\frac{n}{2}$ elementos cada;
 - Conquistar: ordenar as duas subsequências recursivamente utilizando o ordenação por intercalação;
 - Combinar: intercalar as duas subsequências ordenadas para produzir a solução.

Algoritmo Merge Sort - Simulação

*vetor =

0	1	2	3	4	5
5	4	3	1	2	6

Divide o vetor em duas metades.

metadeTamanho=(Início + Fim) / 2

metadeTamanho=(0 + 5) / 2

`metadeTamanho = 2` (divisão de inteiros).

Algoritmo Merge Sort - Simulação

*vetor =

Divide o vetor em duas metades.
metadeTamanho=(Início + Fim) / 2
metadeTamanho=(0 + 2) / 2
metadeTamanho = 1 (divisão de inteiros).

Algoritmo Merge Sort - Simulação

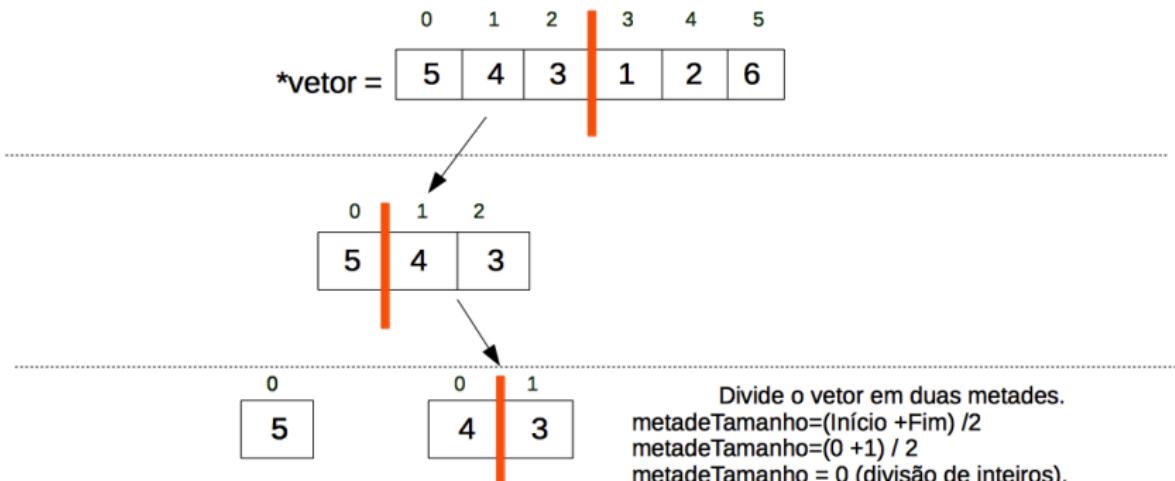


Subvetor com
1 elemento.
Caso base da
recursão.

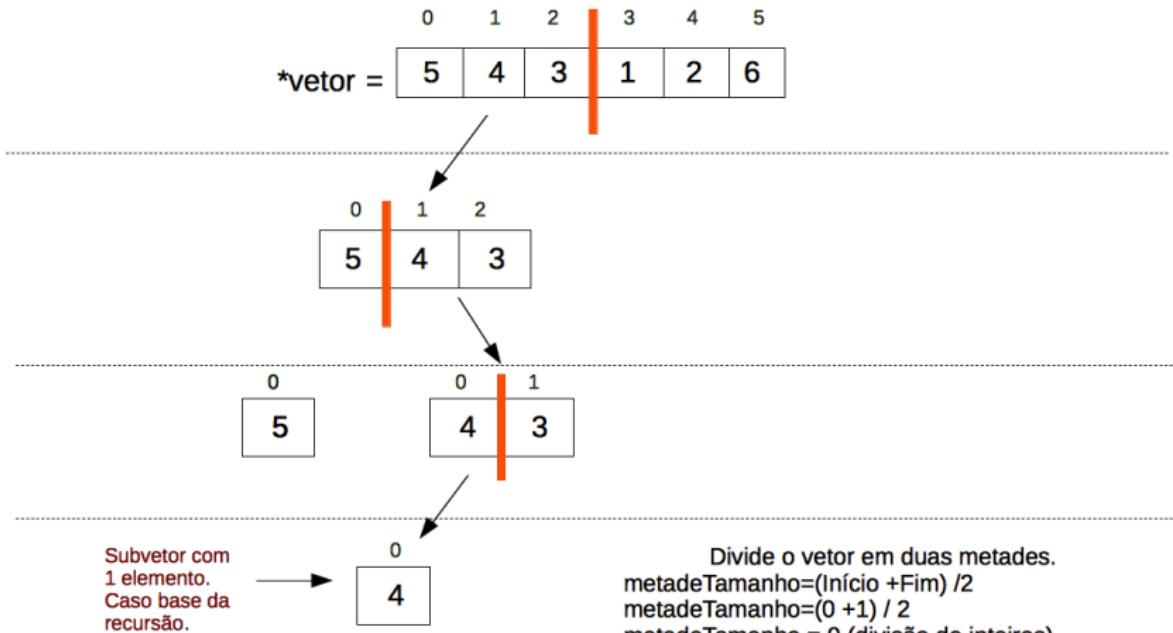


Divide o vetor em duas metades.
 $\text{metadeTamanho} = (\text{Início} + \text{Fim}) / 2$
 $\text{metadeTamanho} = (0 + 1) / 2$
 $\text{metadeTamanho} = 0$ (divisão de inteiros).

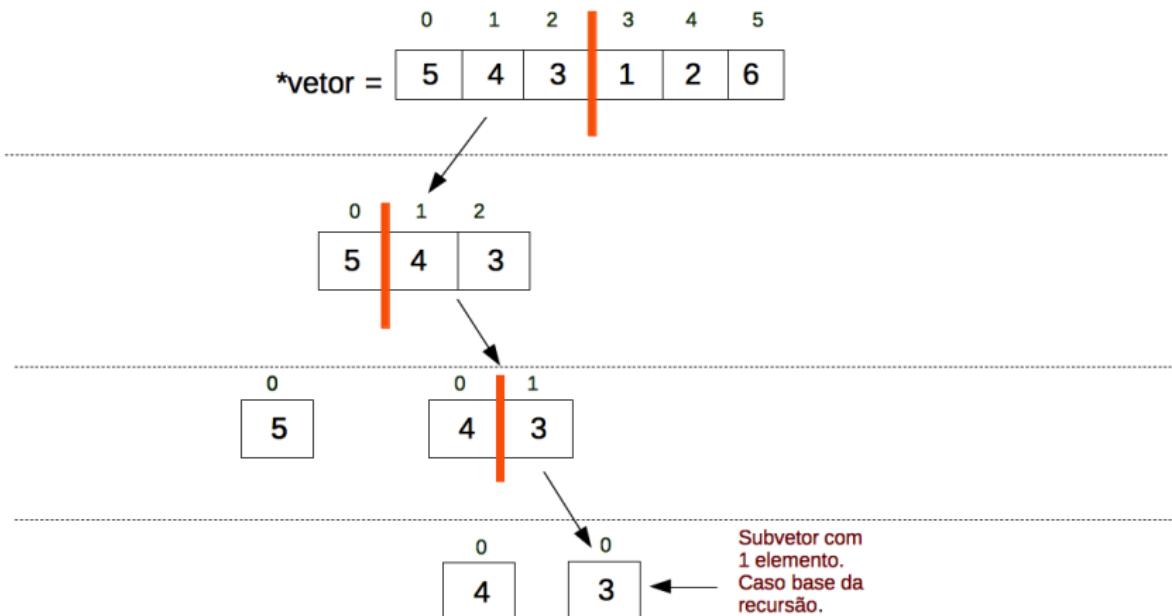
Algoritmo Merge Sort - Simulação



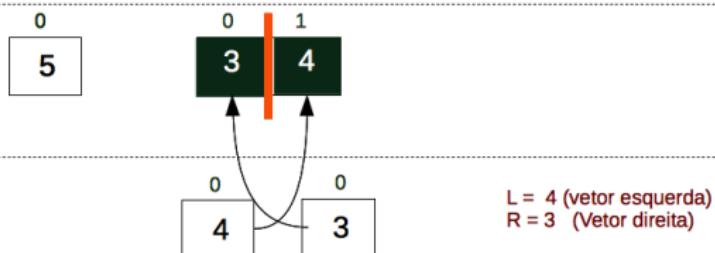
Algoritmo Merge Sort - Simulação



Algoritmo Merge Sort - Simulação

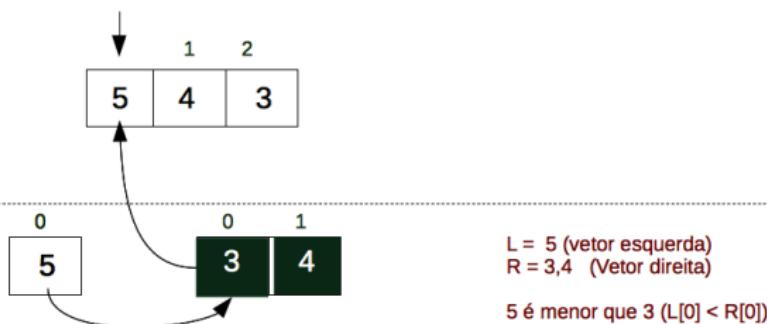
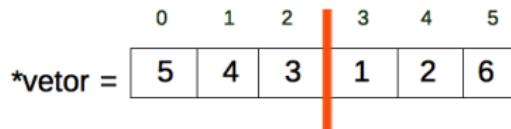


Algoritmo Merge Sort - Simulação



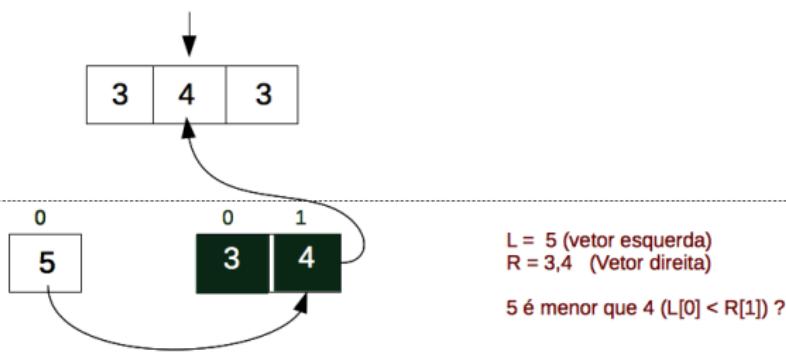
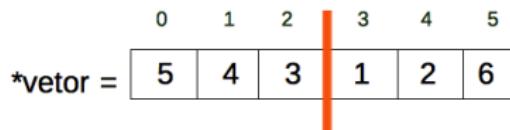
4 é menor que 3 ? Troca

Algoritmo Merge Sort - Simulação



5 < 3?
3 é menor e entra na posição 0.

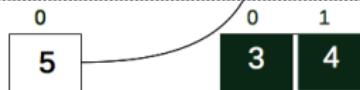
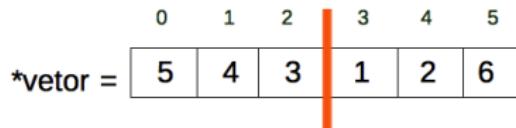
Algoritmo Merge Sort - Simulação



5 < 4?

4 é menor e entra na posição 1.

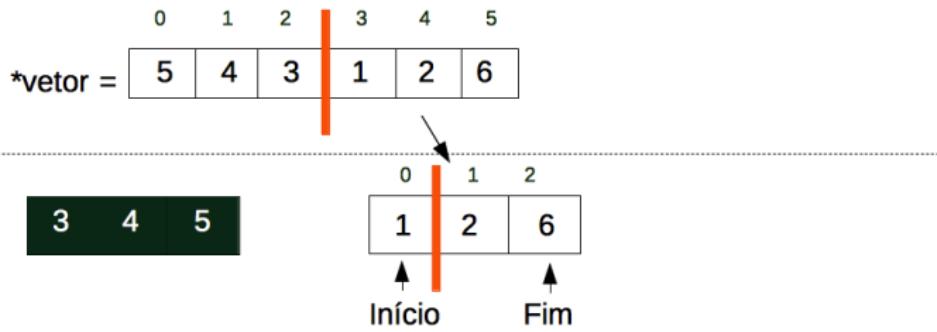
Algoritmo Merge Sort - Simulação



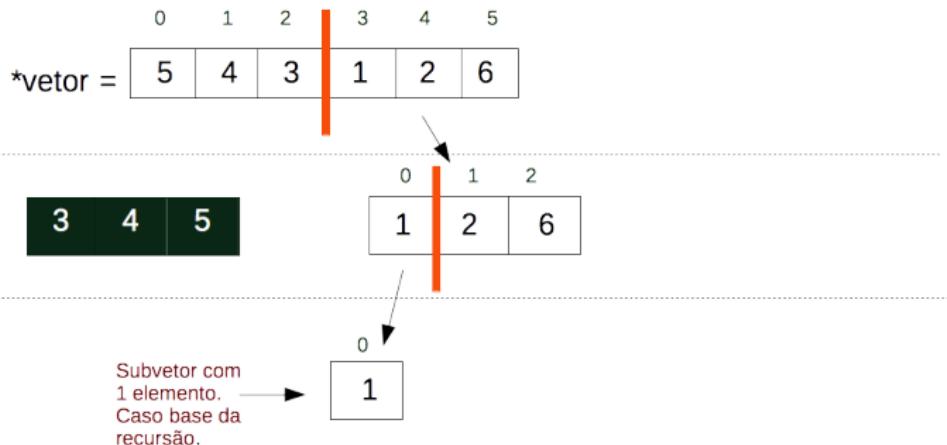
L = 5 (vetor esquerda)
R = 3,4 (Vetor direita)

O vetor à direita não tem mais elementos para comparar.
Copie todos os elementos do vetor a esquerda que já
estão ordenados. 5 vai para posição 2 no subvetor.

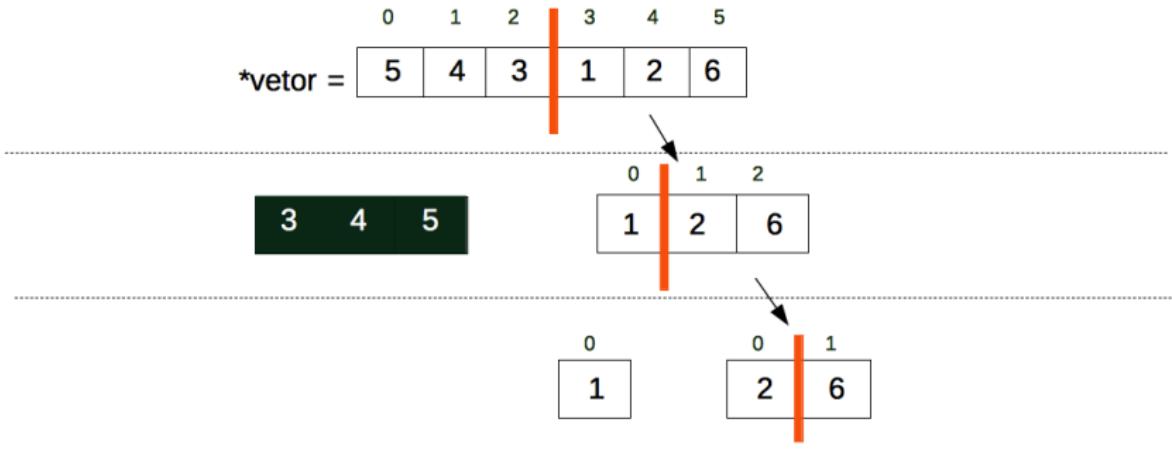
Algoritmo Merge Sort - Simulação



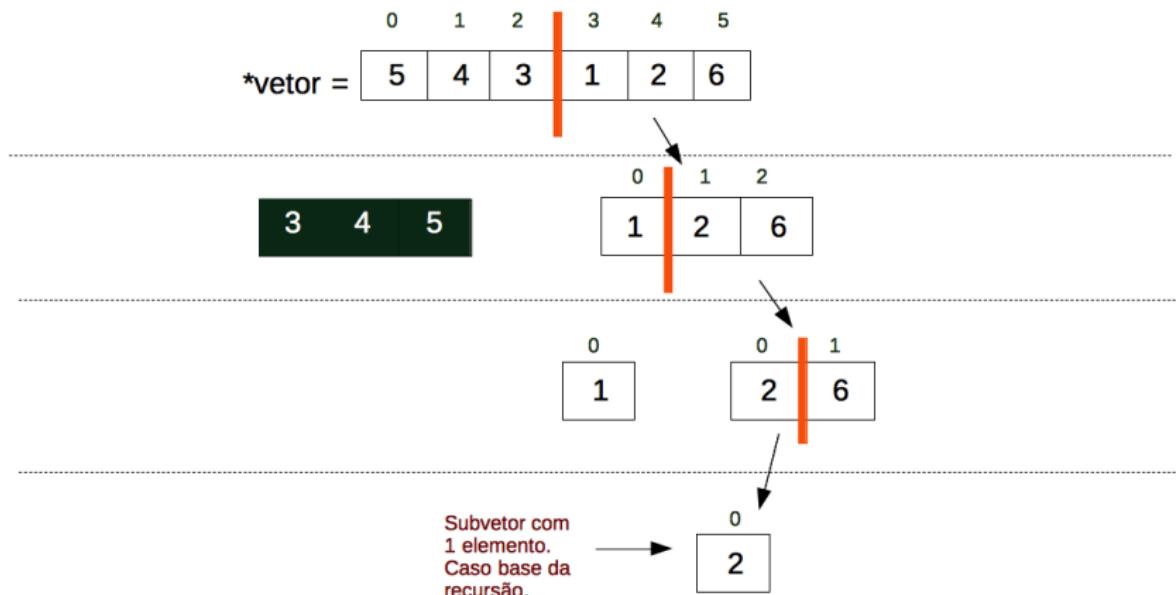
Algoritmo Merge Sort - Simulação



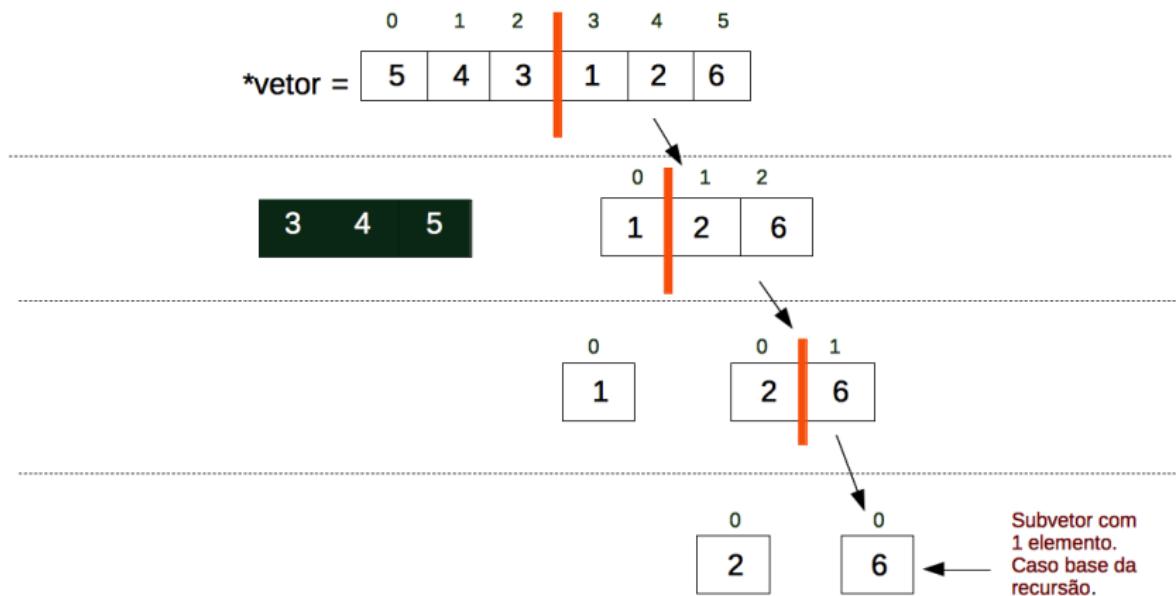
Algoritmo Merge Sort - Simulação



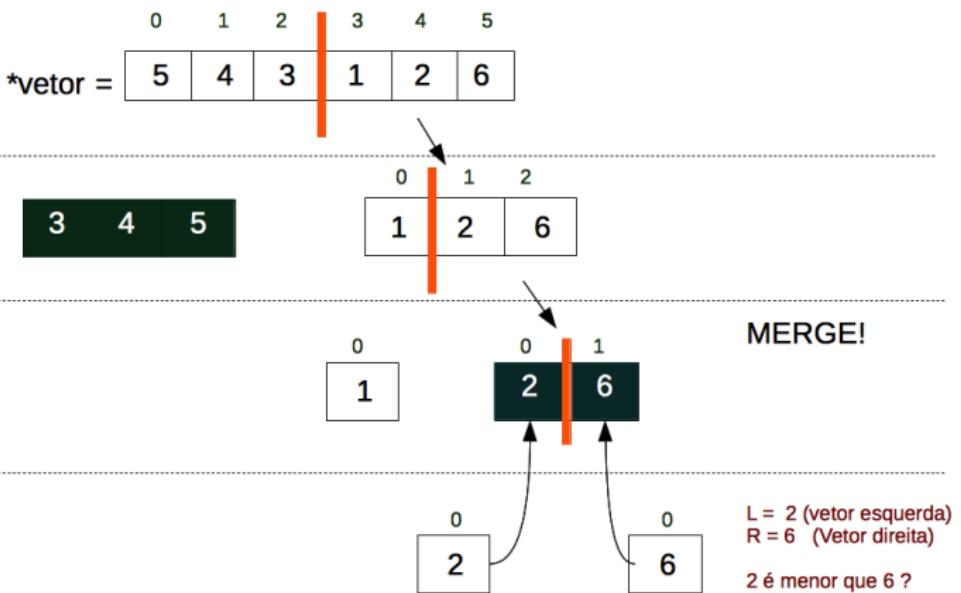
Algoritmo Merge Sort - Simulação



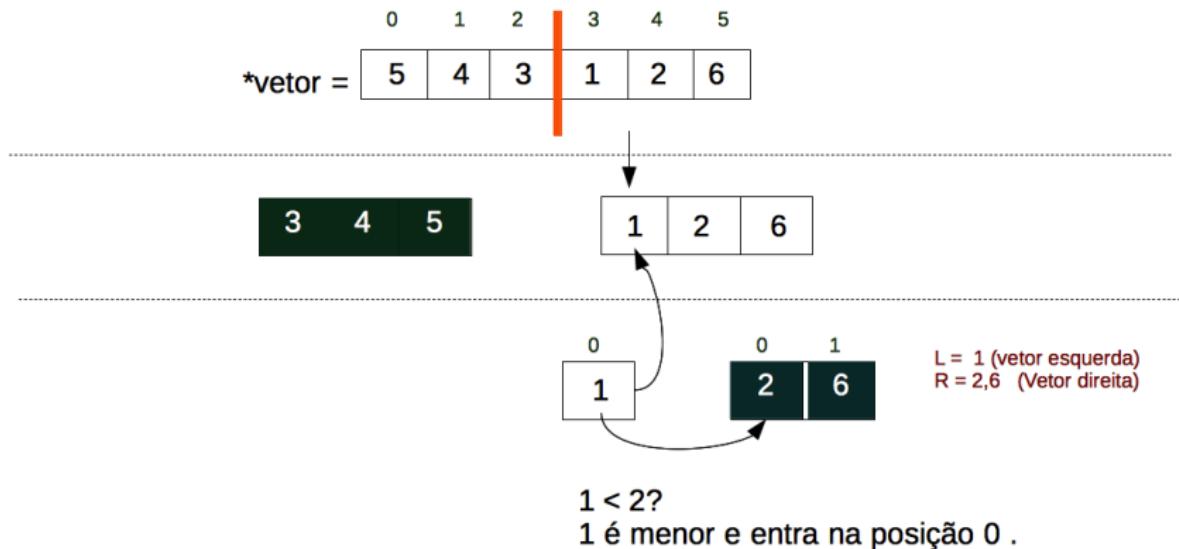
Algoritmo Merge Sort - Simulação



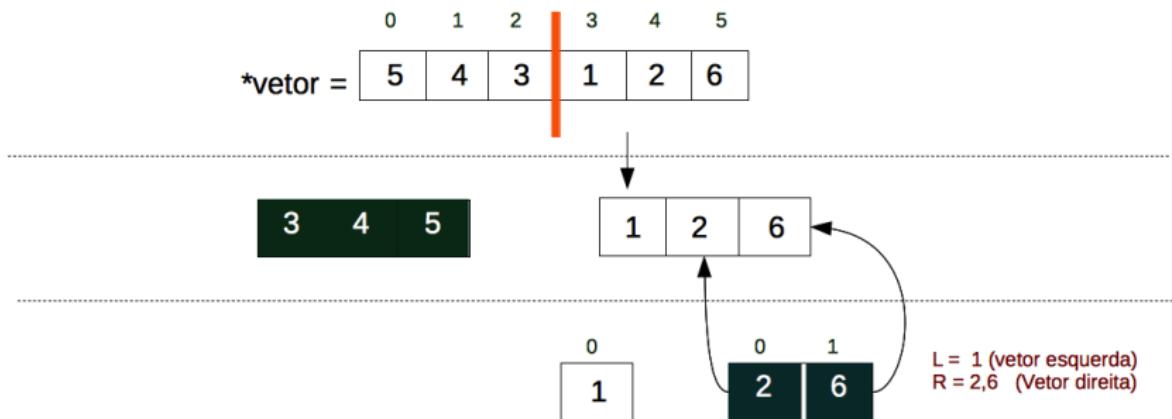
Algoritmo Merge Sort - Simulação



Algoritmo Merge Sort - Simulação

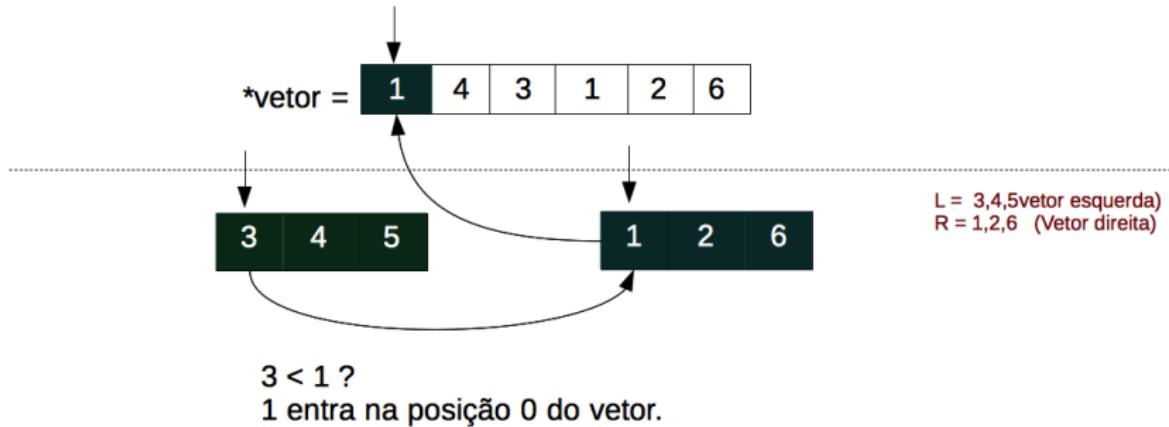


Algoritmo Merge Sort - Simulação

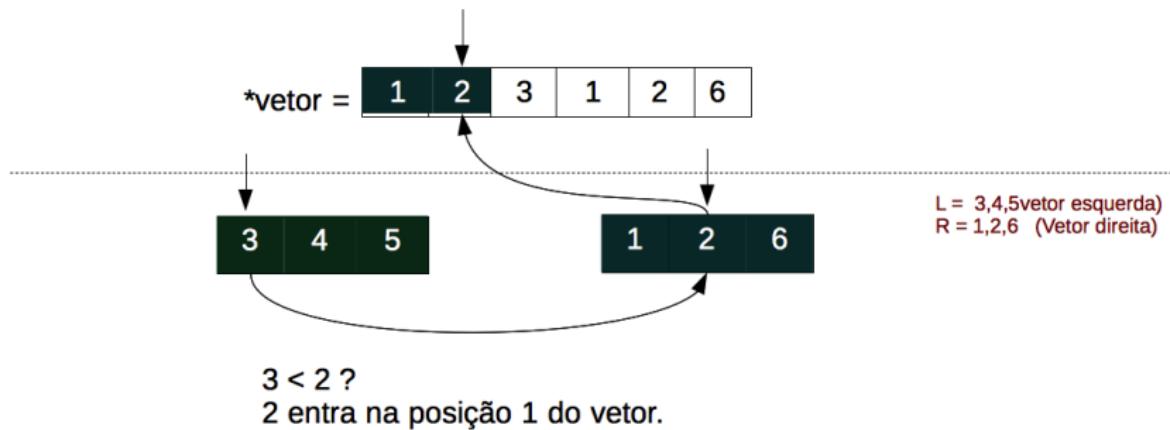


O vetor à esquerda não tem mais elementos para comparar. O vetor à direita está ordenado. Copiar !

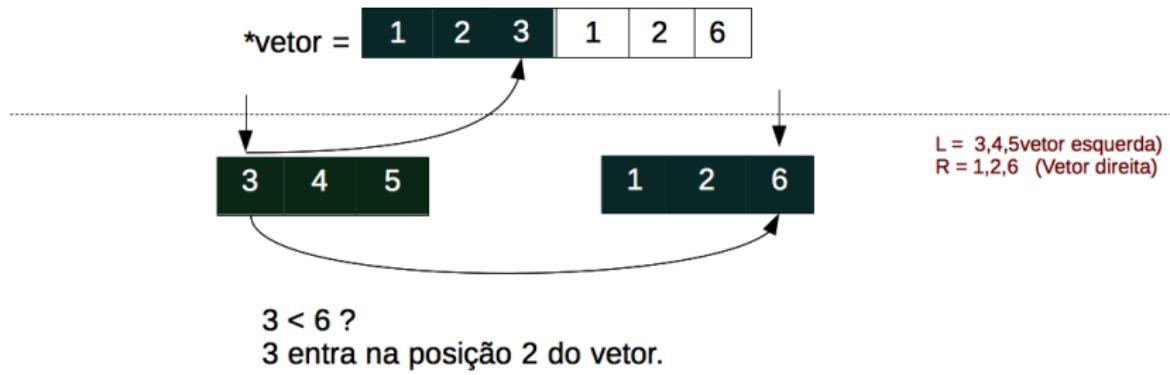
Algoritmo Merge Sort - Simulação



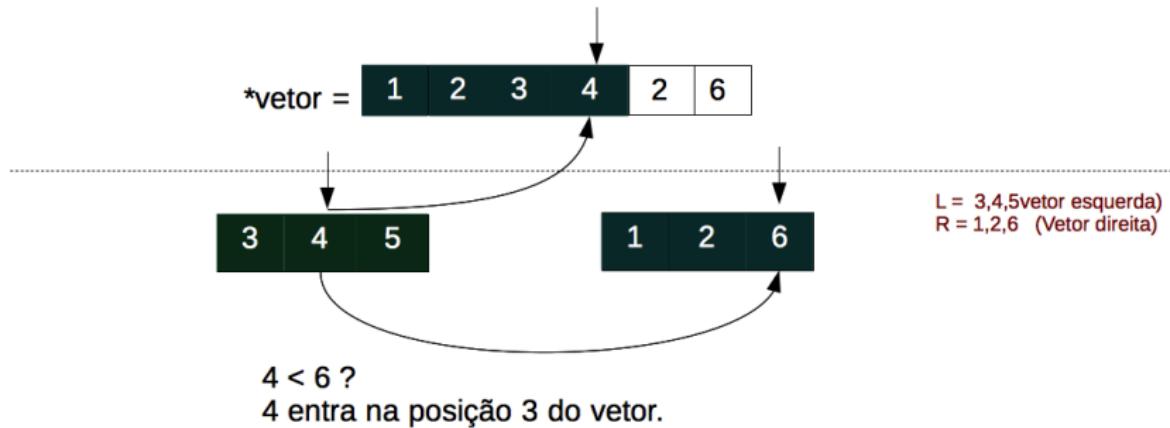
Algoritmo Merge Sort - Simulação



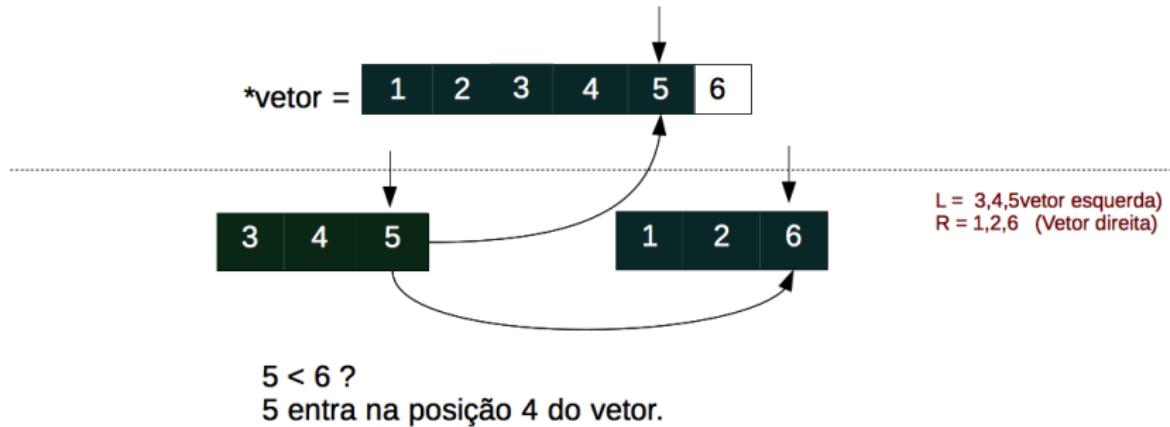
Algoritmo Merge Sort - Simulação



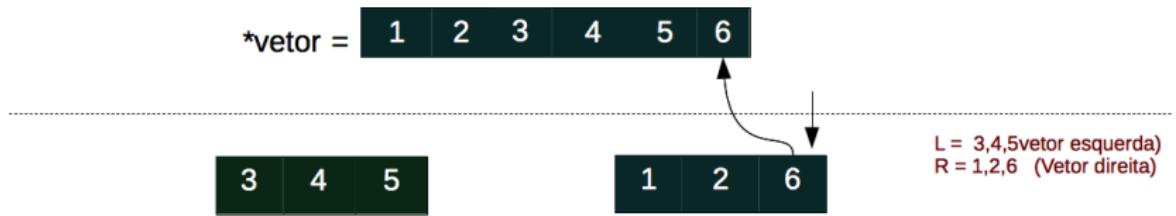
Algoritmo Merge Sort - Simulação



Algoritmo Merge Sort - Simulação



Algoritmo Merge Sort - Simulação



Vetor à esquerda não tem mais elementos para comparar. Copiar todos os elementos restantes do vetor à direita.

Algoritmo Merge Sort - Simulação

*vetor = [1 | 2 | 3 | 4 | 5 | 6]

Vetor Ordenado !!!

- Analisando o **merge** é possível verificar que o algoritmo faz duas chamadas recursivas (metade esquerda do vetor e metade direita do vetor). Após a fase de divisão, o algoritmo executa a intercalação das duas metades.
- Para calcularmos a complexidade do algoritmo **merge** é necessário obtermos a expressão de recorrência desse algoritmo *recursivo*.

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

- Sendo que, $2T\left(\frac{n}{2}\right)$ correspondem às duas chamadas recursivas e n o tempo gasto para intercalação das duas metades do vetor.

- A recursão obtida anteriormente será solucionada pelo **método master**.
- O método master apresenta um teorema para resolver quase todas as recorrências que possuem a forma:

$$a \cdot T\left(\frac{n}{b}\right) + f(n)$$

- O método possui três casos:
 - ① Se $f(n) = O(n^{\log_b a - k})$, para qualquer $k > 0$, então $T(n) = \theta(n^{\log_b a})$.
 - ② Se $f(n) = \theta(n^{\log_b a})$, então $T(n) = \theta(n^{\log_b a} \cdot \log n)$.
 - ③ Se $f(n) = \Omega(n^{\log_b a + k})$, para algum $k > 0$
e se $a \cdot f(n/b) \leq c \cdot f(n)$ para alguma constante $c < 1$
e para todo n suficientemente grande, então $T(n) = \Theta(f(n))$.

- Considerando a expressão de recursão do **Merge Sort** podemos solucionar a complexidade do algoritmo aplicando o caso (2) do *método master*.

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad (1)$$

- Dessa forma, temos que os valores são: $a=2$, $b=2$, $f(n) = n$.
- Precisamos verificar se $f(n) = \theta(n^{\log_b a})$. Segue a resolução:

$$f(n) = \theta(n^{\log_b a}), f(n)=n.$$

$$n = \theta(n^{\log_2 2}), a=2 \text{ e } b=2.$$

$$n = \theta(n^1), \text{ então pelo método master}$$

$$T(n) = \Theta(n \cdot \log n).$$

- No algoritmo Merge Sort, qualquer que seja o vetor de entrada, o algoritmo trabalhará da mesma maneira.

- 1 Introdução**
- 2 Algoritmos de Ordenação Bubble Sort**
- 3 Algoritmo de Ordenação Insert Sort**
- 4 Algoritmo de Ordenação Selection Sort**
- 5 Algoritmo de Ordenação Merge Sort**
- 6 Algoritmo de Ordenação Quick Sort**
- 7 Ordenando outras estruturas de dados**

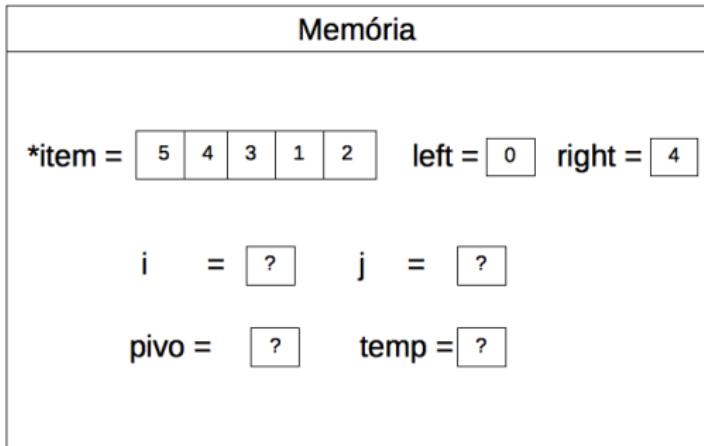
- O *Quick Sort* é baseado na ideia de **partições**. O procedimento geral é selecionar um valor, chamado de *pivô*, e , então, fazer a partição do vetor em dois subvetores, com todos os elementos maiores ou igual ao *pivô* de um lado e os menores do outro. O procedimento ocorre até que o vetor fique com apenas um elemento.

Introdução ao algoritmo Quick Sort

- Esse algoritmo é baseado na técnica de **divisão e conquista** da seguinte forma:
 - ➊ **Dividir:** o vetor $X[p..r]$ é partitionado em dois subvetores **não** vazios $X[p..q]$ e $X[q+1..r]$, tais que cada elemento de $X[p..q]$ é menor ou igual a cada elemento de $X[q+1..r]$. O índice q é calculado como parte do processo. Para calcular q , escolhe-se o elemento que encontra-se na metade do vetor original, chamado de **pivô** e rearranjam-se os elementos do vetor de forma que os elementos que ficarem à esquerda de q são menores (ou iguais) ao pivô e os que ficarem à direita de q são maiores (ou iguais) ao pivô.
 - ➋ **Conquistar:** Os dois subvetores são ordenados $X[p..q]$ e $X[q+1..r]$ por chamadas recursivas ao **Quick Sort**.
 - ➌ **Combinar:** durante o processo recursivo, os elementos vão sendo ordenados no próprio vetor, não exigindo nenhum processamento nesta etapa.
- Vamos analisar o algoritmo passo a passo nos próximos slides.

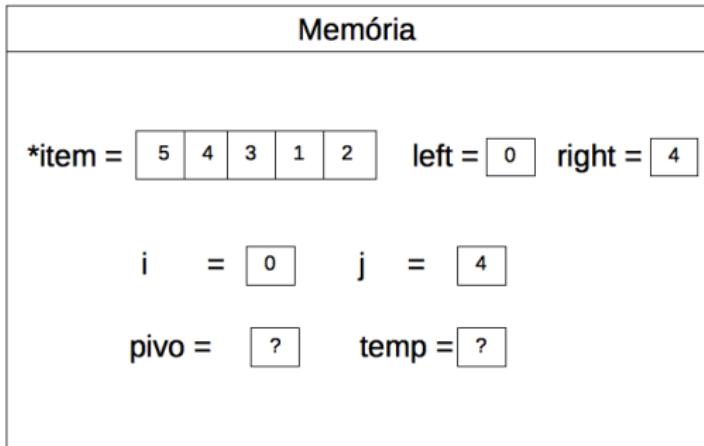
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){
```

```
    int i,j;
```

```
    int pivo,temp;
```

```
    i = left; j=right;
```

Item[$(0+4)/2$]

```
    pivo = item[(left+right)/2];
```

```
    do {
```

```
        while(item[i] < pivo && i<right) i++;
        while(pivo<item[j] && j>left) j--;
```

```
        if(i<=j){
```

```
            temp = item[i];
            item[i] = item[j];
            item[j] = temp;
            i++; j--;
        }
```

```
} while(i<=j);
```

```
if(left<j)
```

```
    qs1(item,left,j);
```

```
if(i<right)
```

```
    qs1(item, i, right);
```

```
}
```

Memória

i

j

*item =

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

 left =

0

 right =

4

i

=

0

j

=

4

pivo =

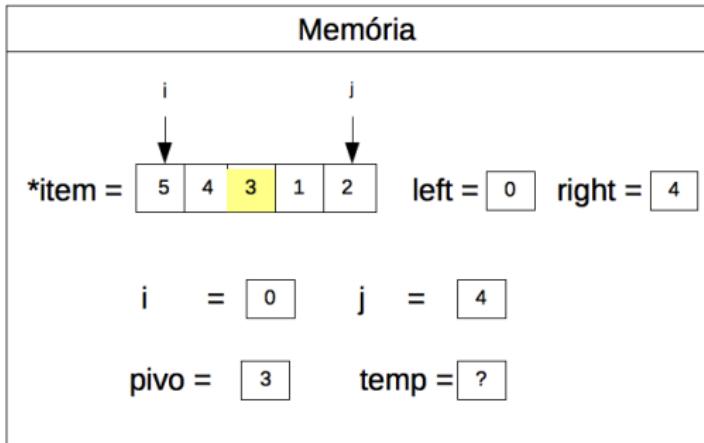
3

temp =

?

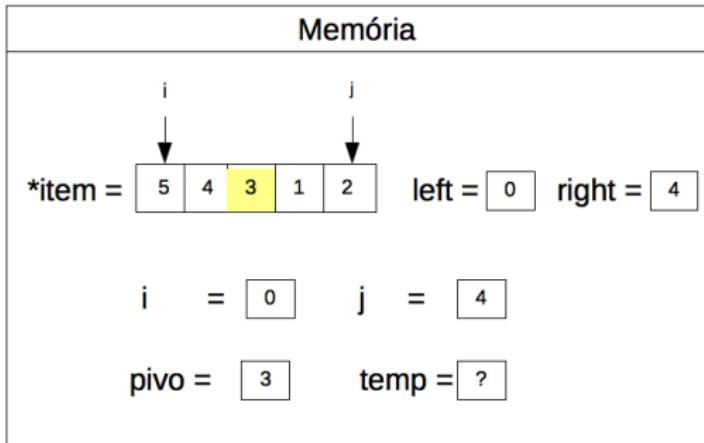
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {      5 < 3 (F)  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



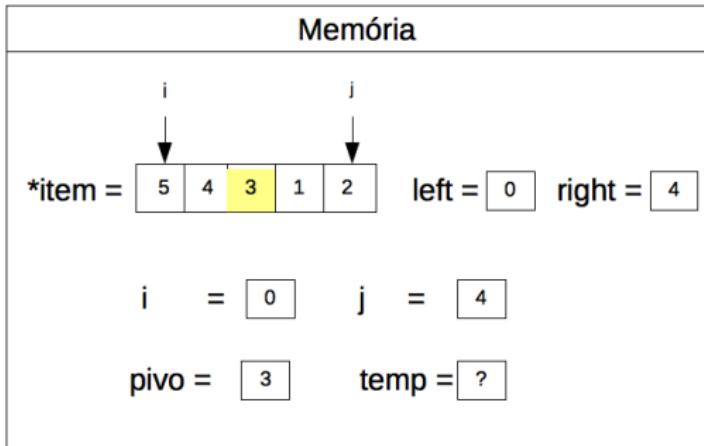
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
        if(i<=j){  
            3 < 2 (F)  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



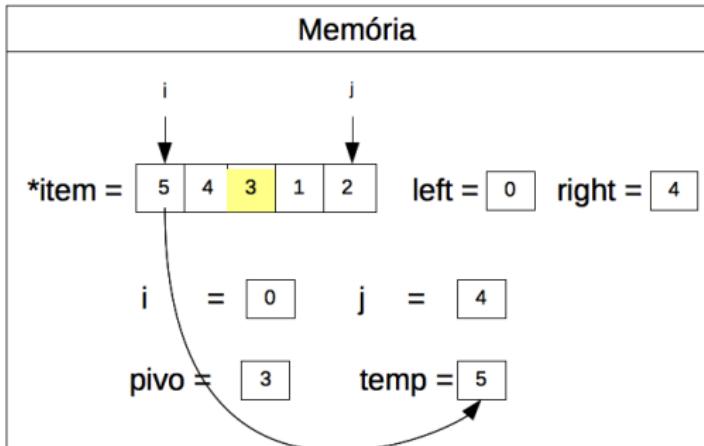
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){          0 < 4 (V)  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



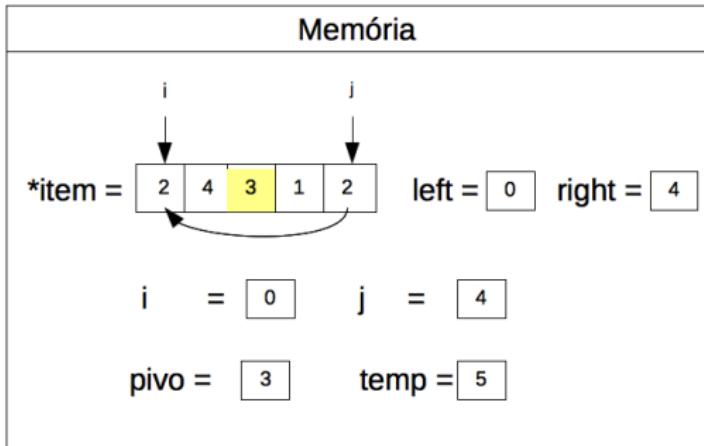
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



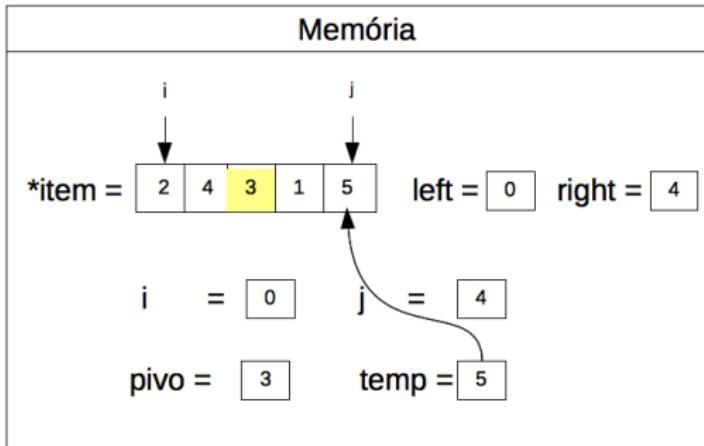
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



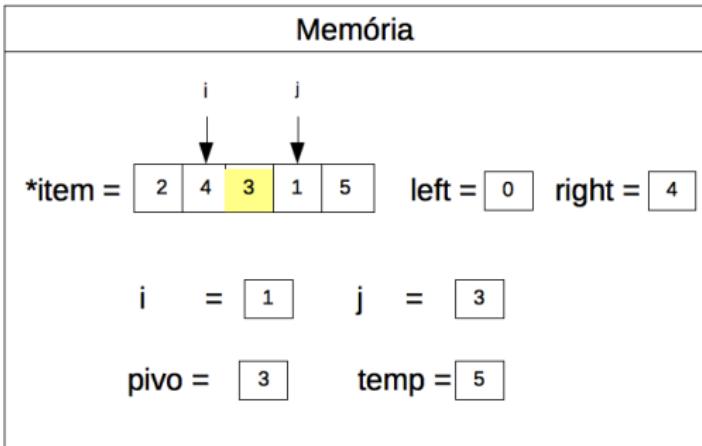
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



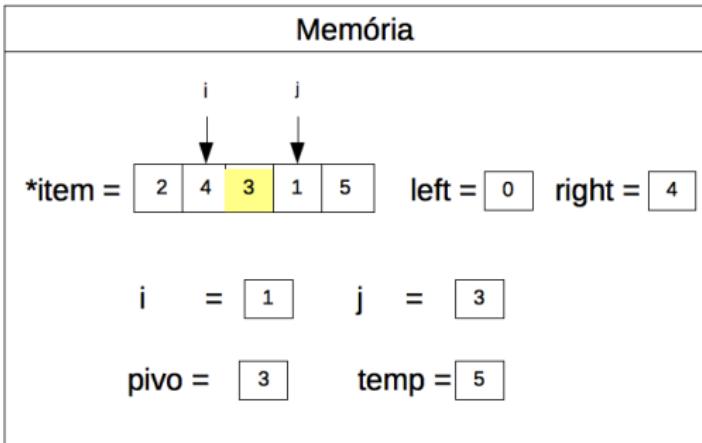
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



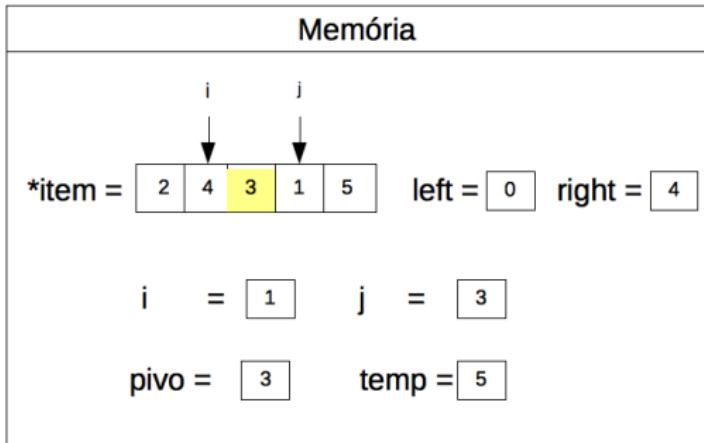
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
        1 <= 3 (V)  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item, left, j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



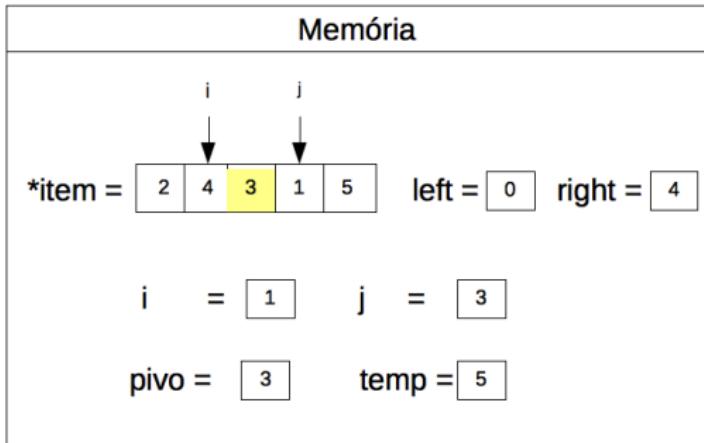
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        if(i<=j) {  
            while(item[i] < pivo && i<right) i++;  
            while(pivo<item[j] && j>left) j--;  
  
            if(i<=j){  
                temp = item[i];  
                item[i] = item[j];  
                item[j] = temp;  
                i++; j--;  
            }  
        } while(i<=j);  
  
        if(left<j)  
            qs1(item,left,j);  
  
        if(i<right)  
            qs1(item, i, right);  
    }  
}
```



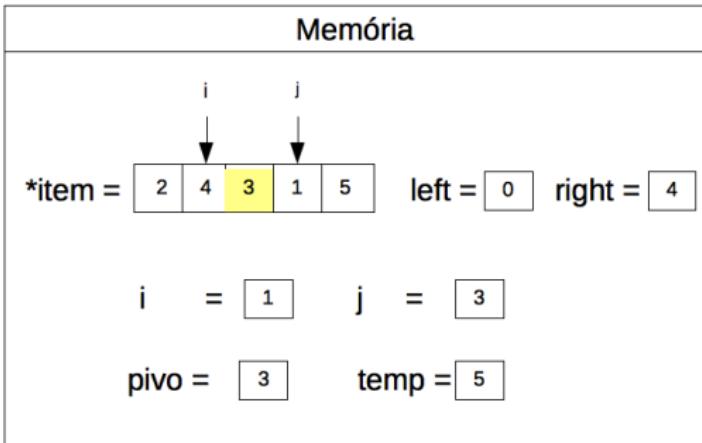
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
            3 < 1 (F)  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



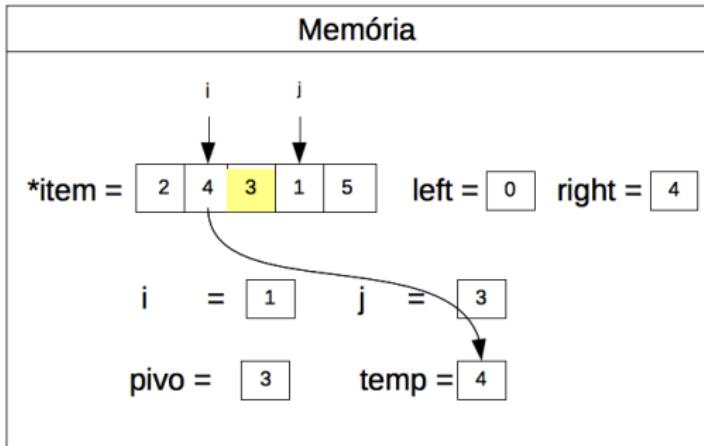
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            1 < 3 (V)  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



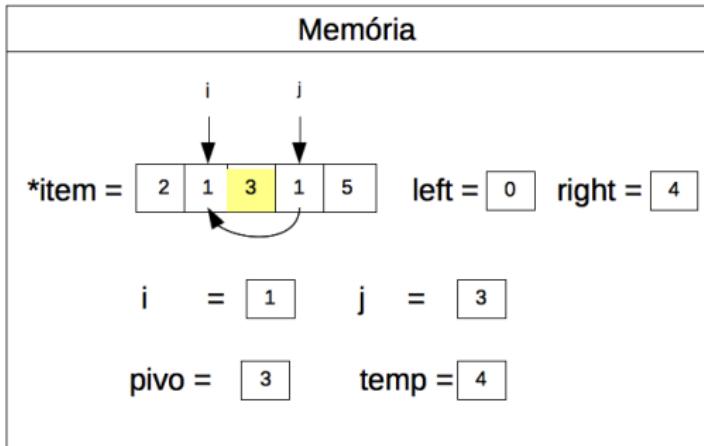
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



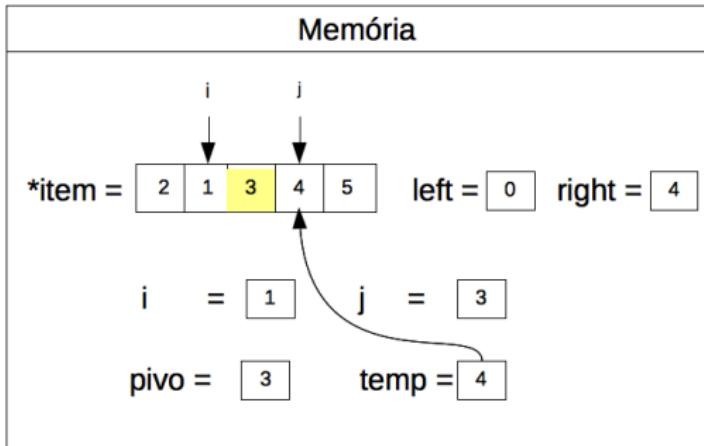
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



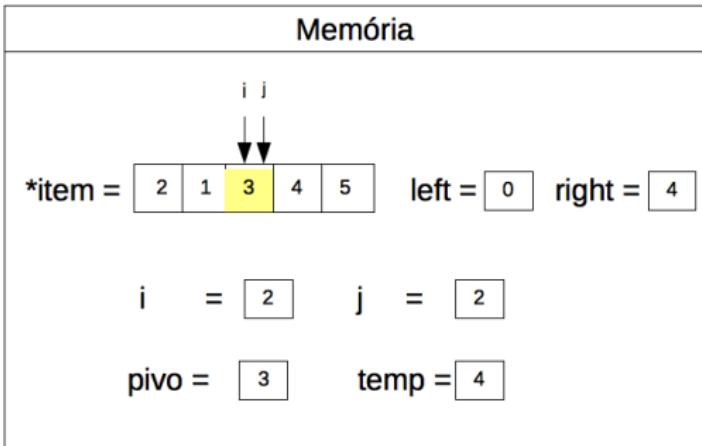
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



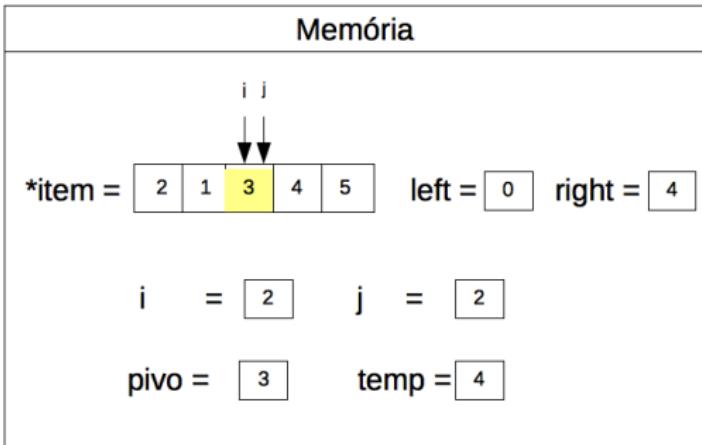
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



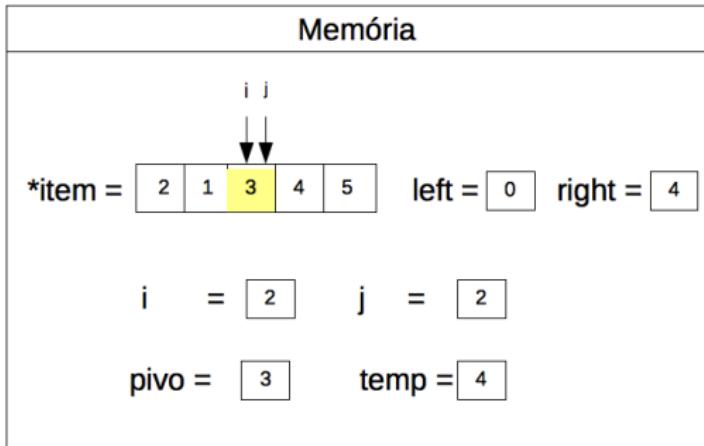
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
        2 <= 2 (V)  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



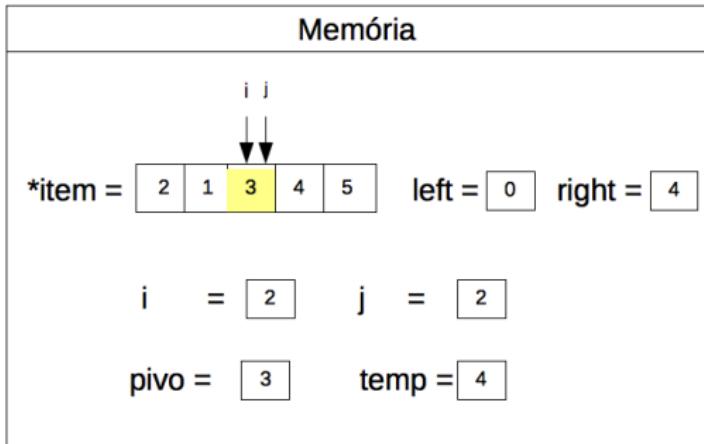
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {      3 < 3 (F)  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



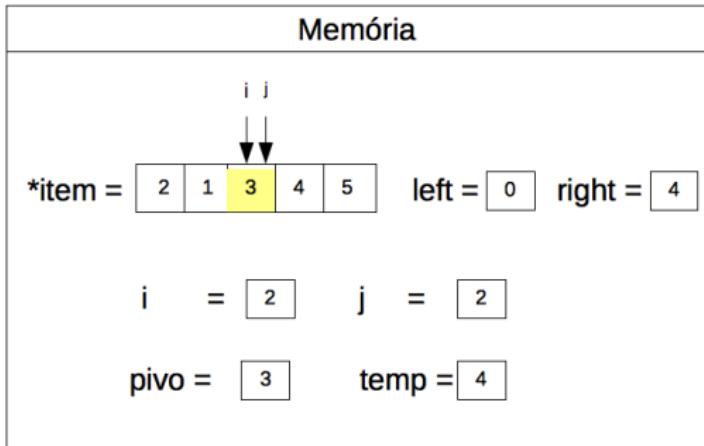
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
        if(i<=j){  
            3 < 3 (F)  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



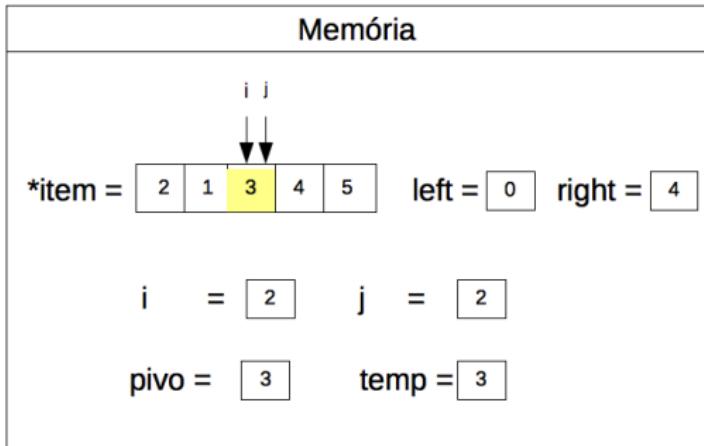
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            2 <= 2 (V)  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



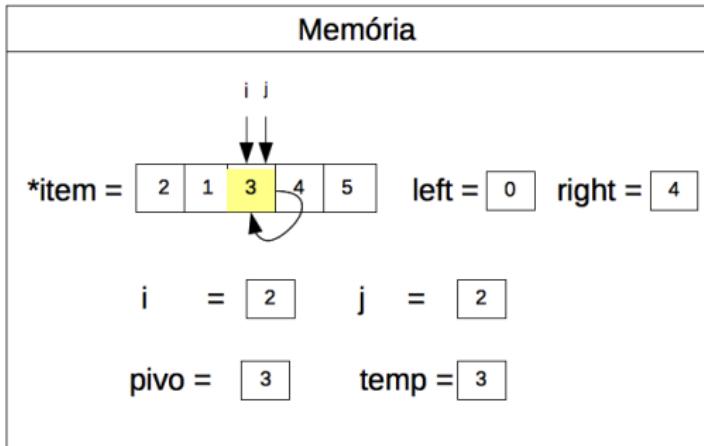
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



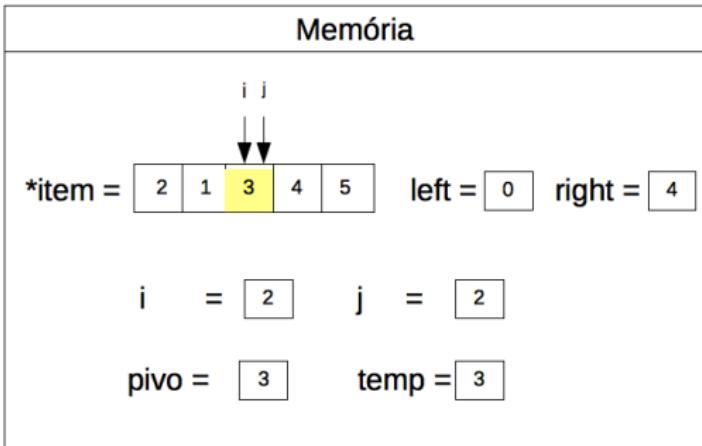
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



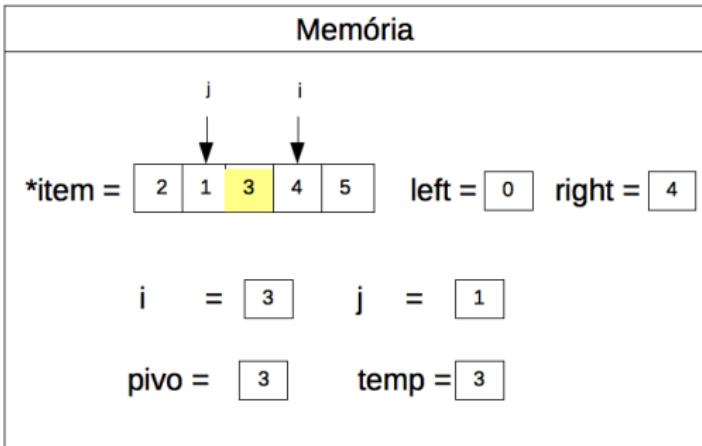
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



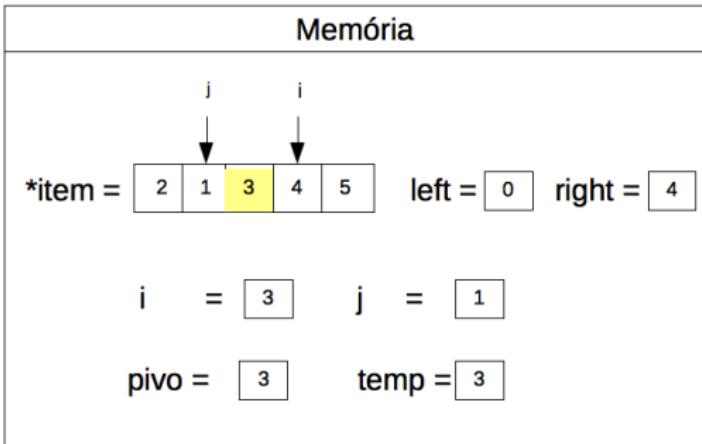
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



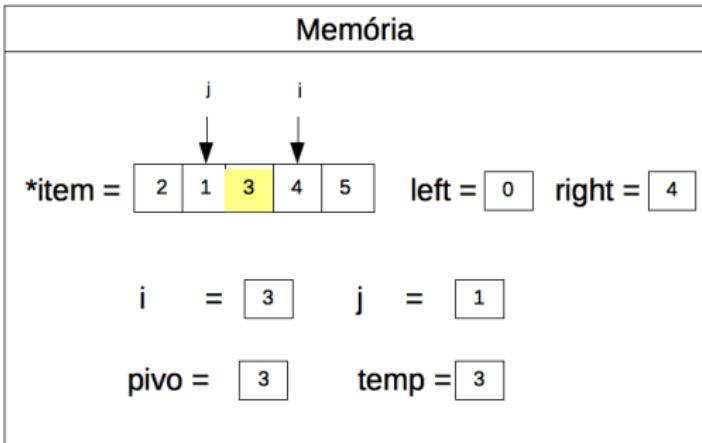
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);      3 <= 1 (F)  
  
    if(left<j)  
        qs1(item, left, j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)          0 < 1 (V)  
        qs1(item, left, j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){
```

```
    int i,j;  
    int pivo,temp;
```

```
    i = left; j=right;  
    pivo = item[(left+right)/2];
```

```
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;
```

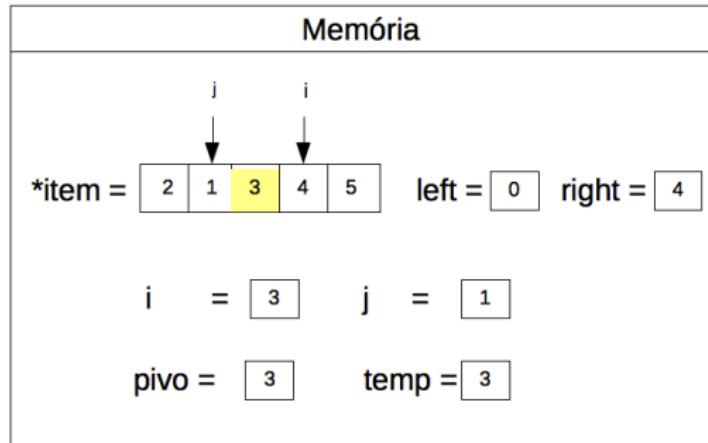
```
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);
```

```
    if(left<j)          Recursão !!!
```

```
        qs1(item,left,j);
```

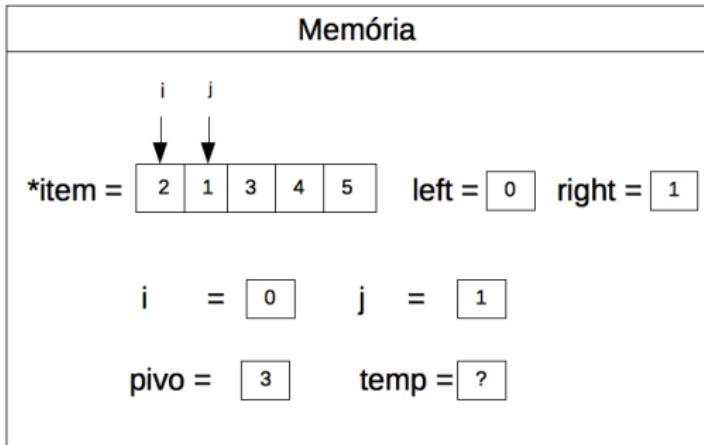
```
    if(i<right)  
        qs1(item, i, right);
```

```
}
```



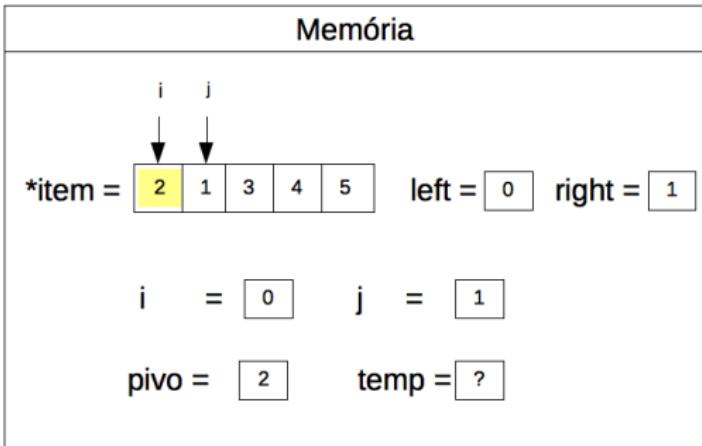
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



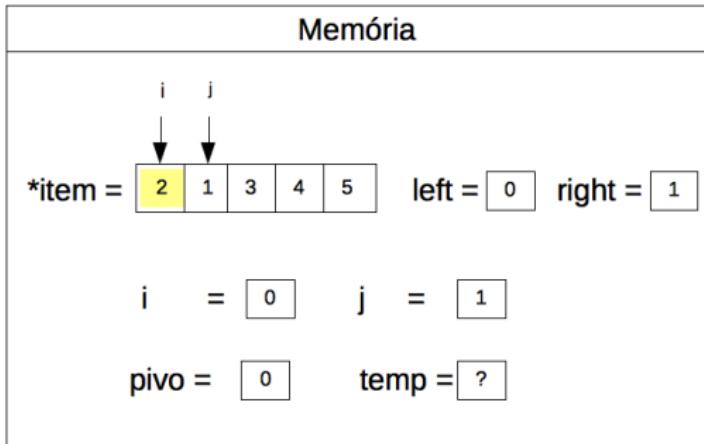
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
    Item[(0+1)/2]  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



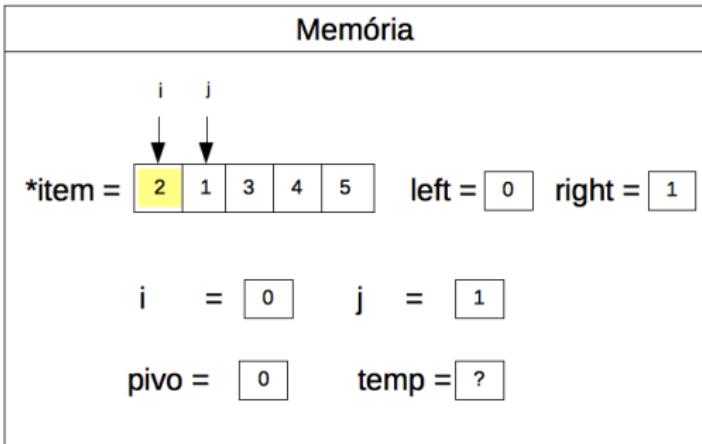
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {      2 < 2 (F)  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



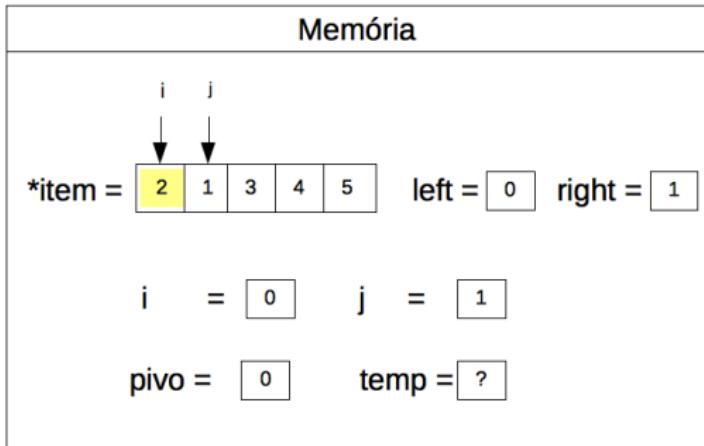
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
        if(i<=j){  
            2 < 1 (F)  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



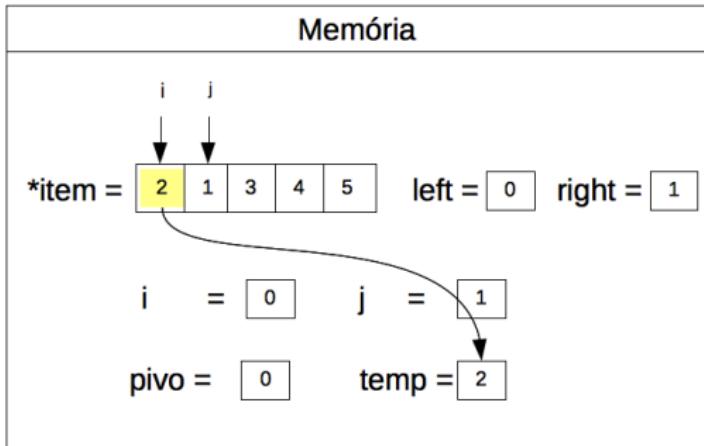
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){      0 < 1 (V)  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



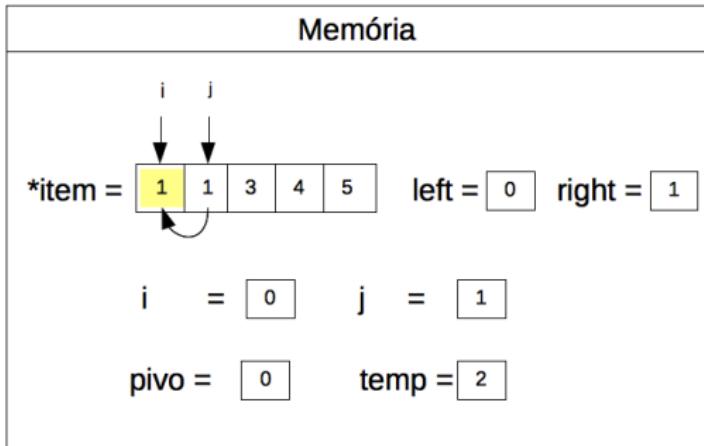
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



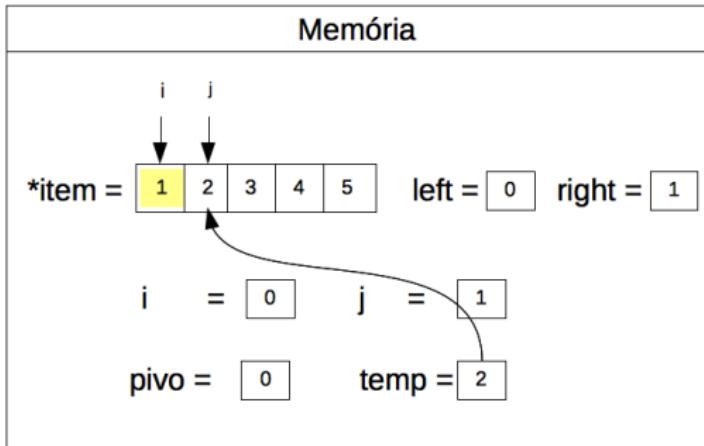
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



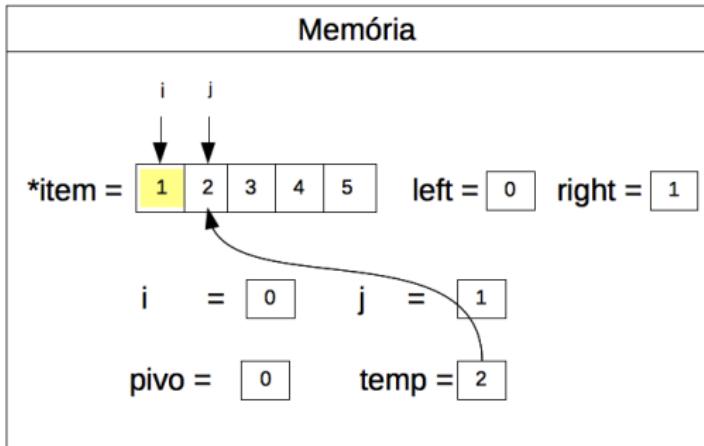
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



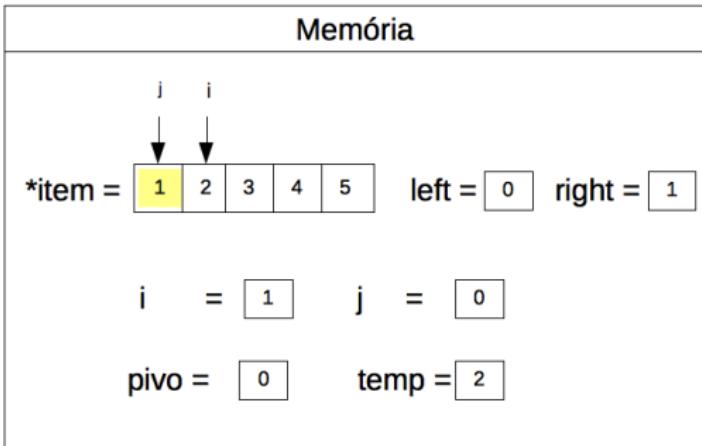
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



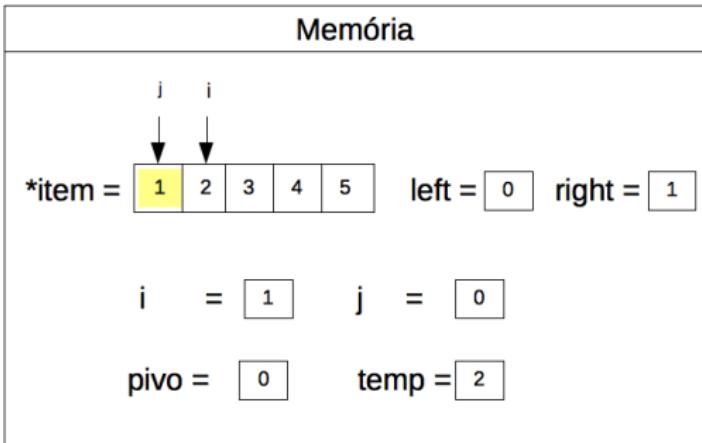
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



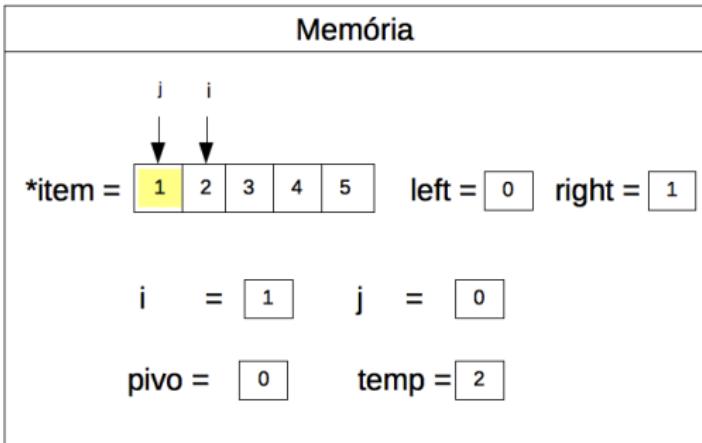
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
        1 <= 0 (F)  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



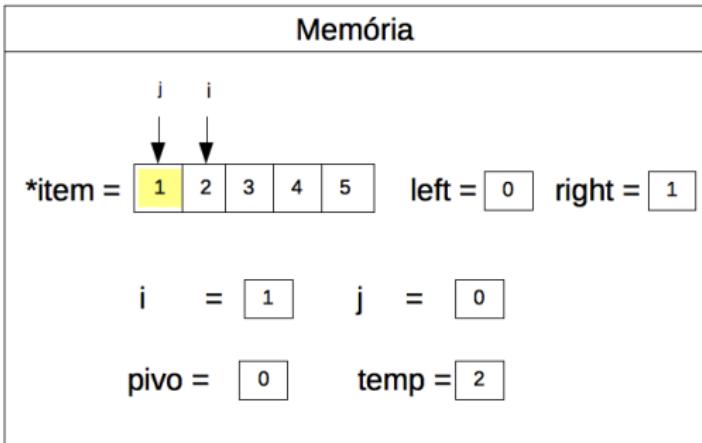
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
    0 < 0 (F)  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



Algoritmo Quick Sort - Simulação

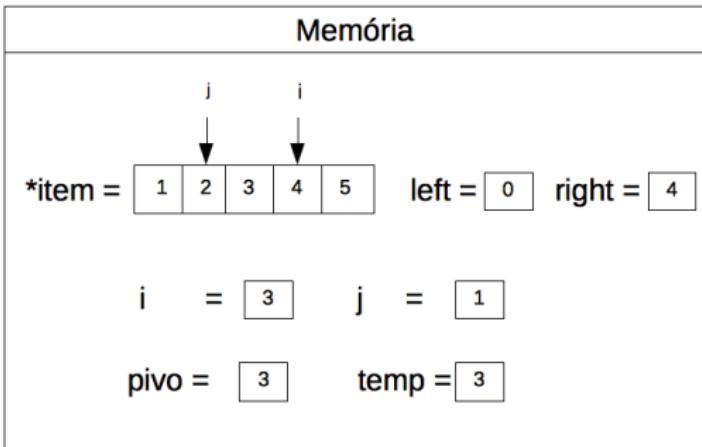
```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
        1 < 1 (F)  
    if(i<right)  
        qs1(item, i, right);  
}
```



Recursão Acabou !
Retorna !

Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
    3 < 4 (V)  
    if(i<right)  
        qs1(item, i, right);  
}
```



Observe que todos os valores menores que o pivô estão a esquerda e todos os valores maiores que o pivô estão a direita !

$i=3; j=1; \text{pivo}=3; \text{temp}=3; \text{left}=0; \text{right}=4.$

Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){
```

```
    int i,j;  
    int pivo,temp;
```

```
    i = left; j=right;  
    pivo = item[(left+right)/2];
```

```
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;
```

```
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);
```

```
    if(left<j)  
        qs1(item, left, j);
```

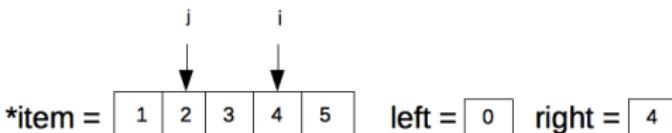
```
    if(i<right)
```

```
        qs1(item, i, right);
```

```
}
```

i=3 e right=4.

Memória



i =

3

 j =

1

pivo =

3

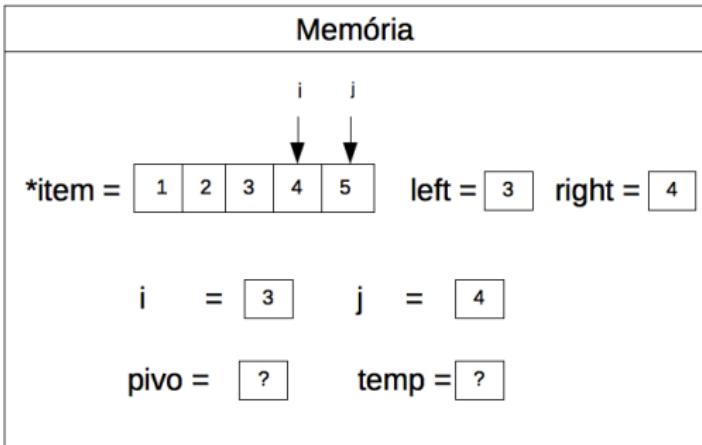
 temp =

3

i = 3; j=1; pivo=3; temp=3; left=0;right=4.

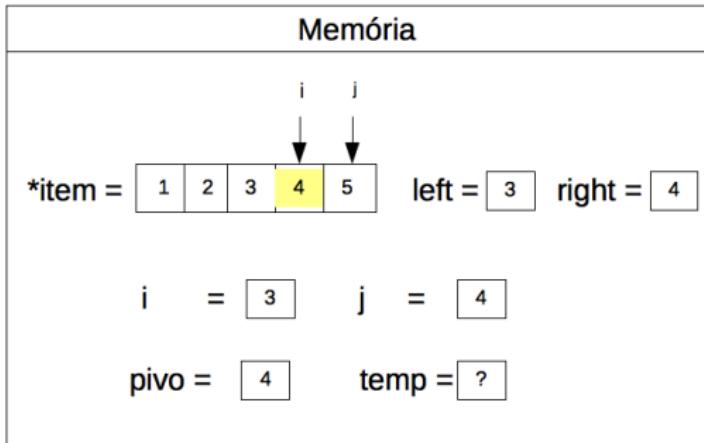
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



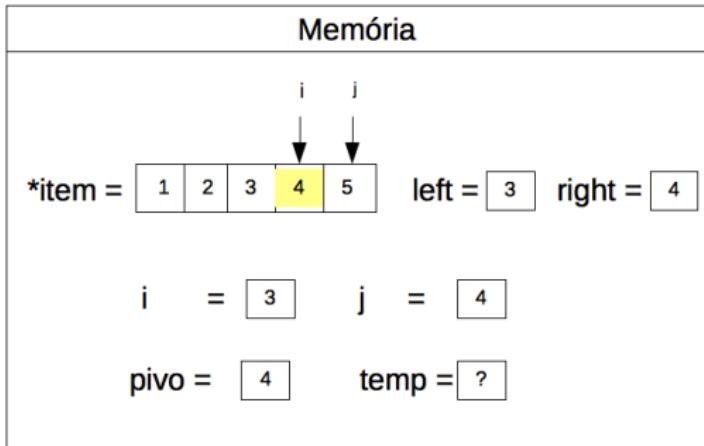
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
    item[(3+4)/2]  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



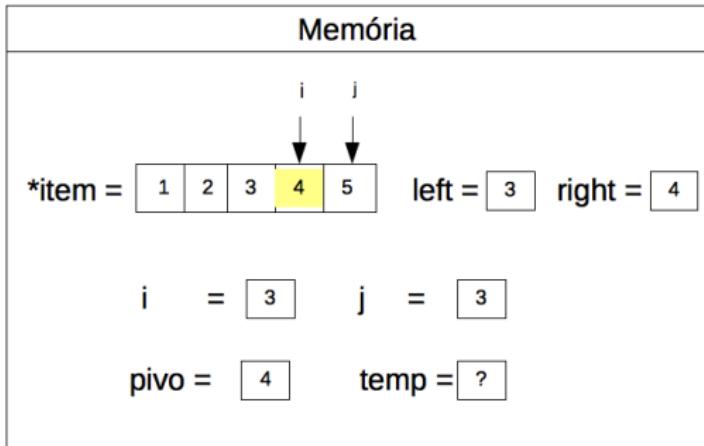
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {          4 < 4 (F)  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



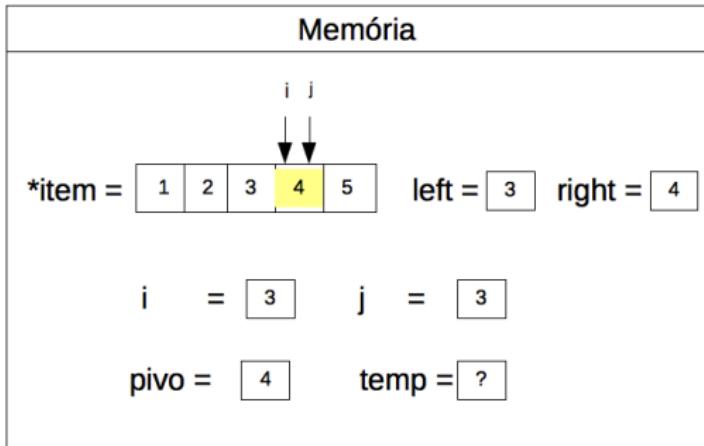
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



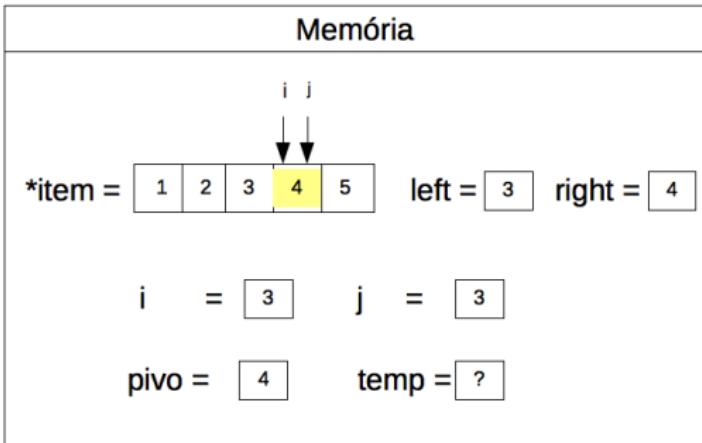
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



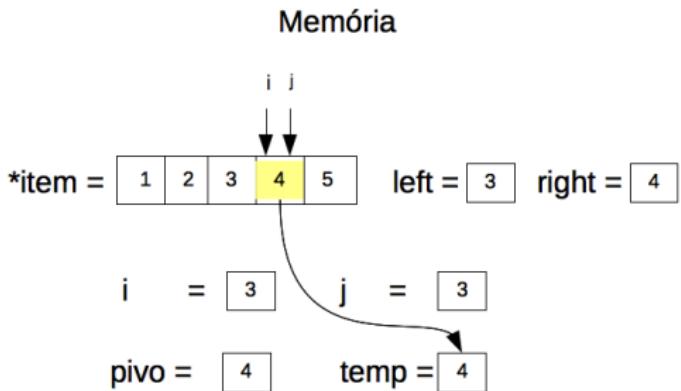
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];      3 <= 3(V)  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



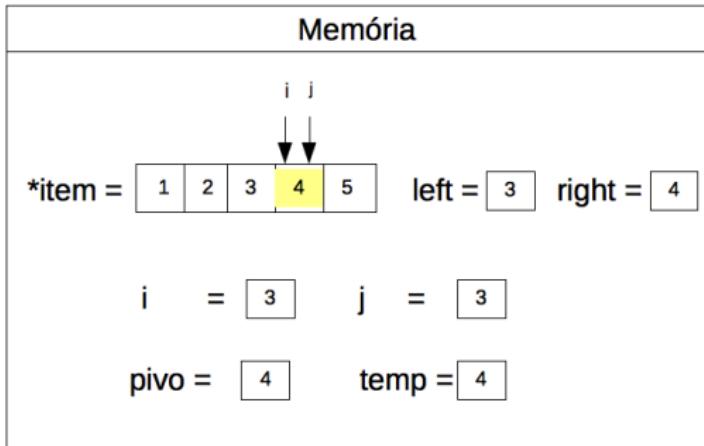
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



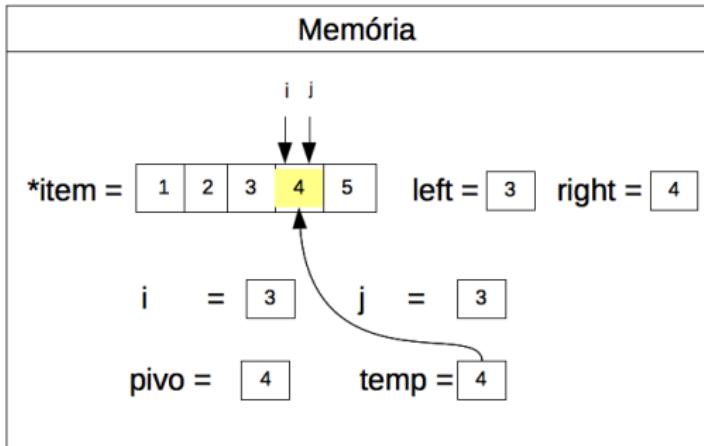
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



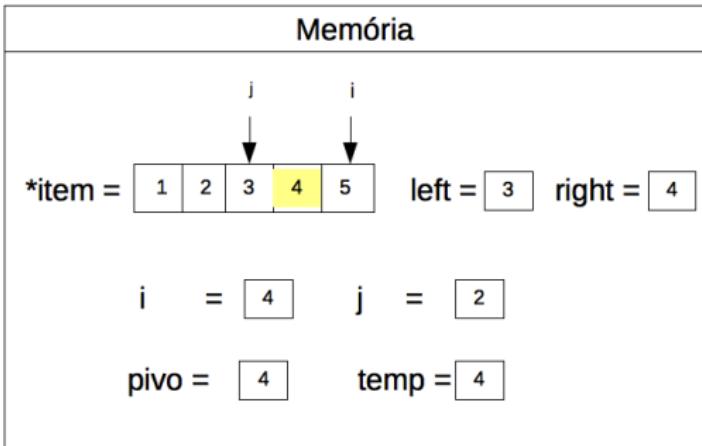
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



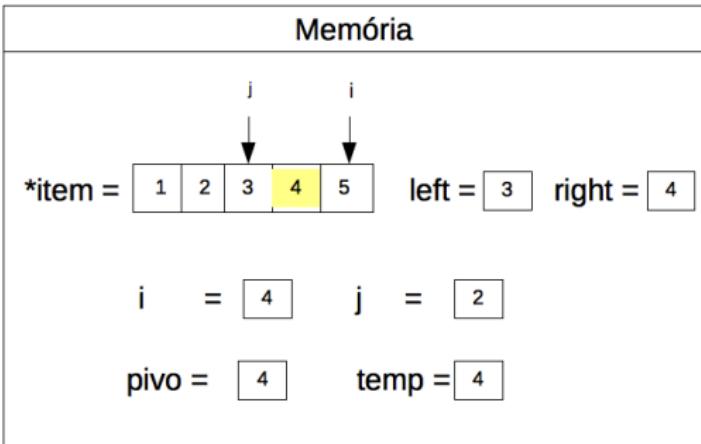
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



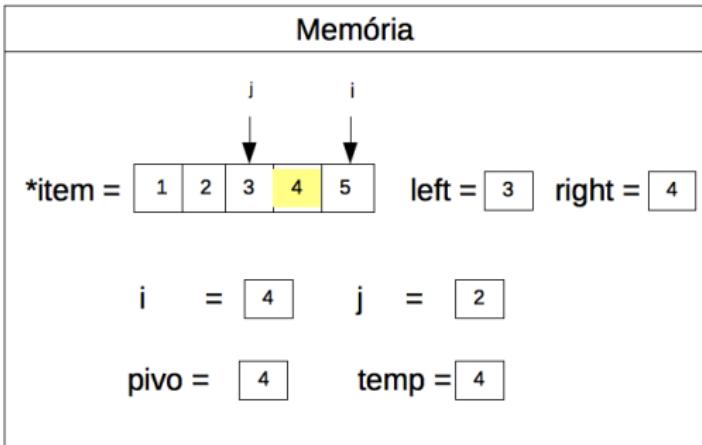
Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)          3 < 2 (F)  
        qs1(item, left, j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){
```

```
    int i,j;  
    int pivo,temp;
```

```
    i = left; j=right;  
    pivo = item[(left+right)/2];
```

```
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;
```

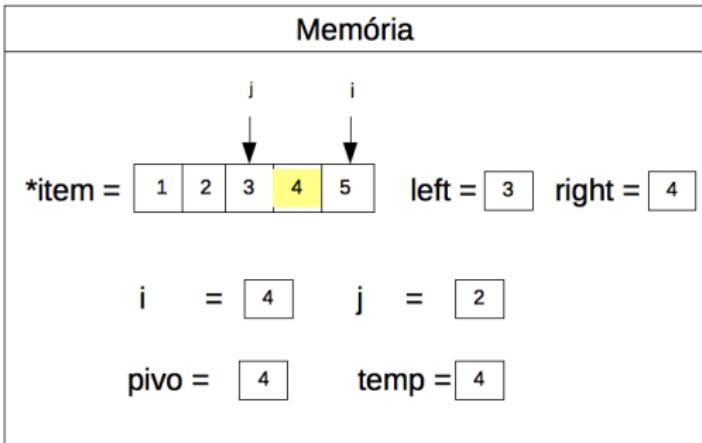
```
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);
```

```
    if(left<j)  
        qs1(item,left,j);
```

```
    if(i<right) 4 < 4 (F)
```

```
        qs1(item, i, right);
```

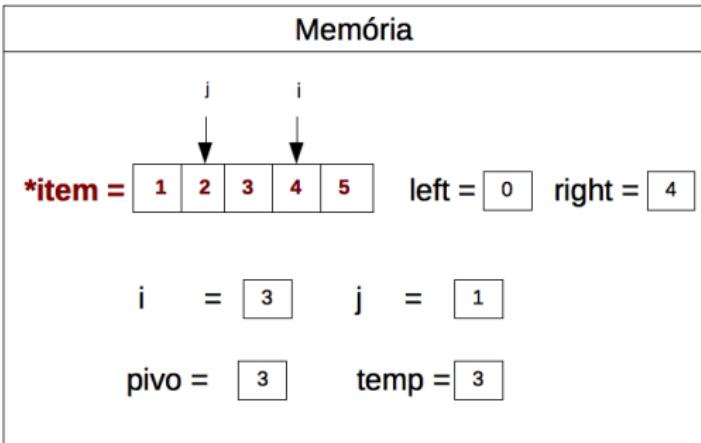
```
}
```



Retorna da chamada recursiva !

Algoritmo Quick Sort - Simulação

```
void qs1(int *item, int left, int right){  
  
    int i,j;  
    int pivo,temp;  
  
    i = left; j=right;  
    pivo = item[(left+right)/2];  
  
    do {  
        while(item[i] < pivo && i<right) i++;  
        while(pivo<item[j] && j>left) j--;  
  
        if(i<=j){  
            temp = item[i];  
            item[i] = item[j];  
            item[j] = temp;  
            i++; j--;  
        }  
    } while(i<=j);  
  
    if(left<j)  
        qs1(item,left,j);  
  
    if(i<right)  
        qs1(item, i, right);  
}
```



I = 3; j=1; pivo=3; temp=3; left=0;right=4.

Ordenado !

- A ideia principal do algoritmo **Quick Sort** é partitionar o vetor em duas partes. Nesse procedimento, o vetor é particionado no índice do pivô, de forma que todos os elementos do lado esquerdo são menores ou iguais que o elemento pivô e todos do lado direito são maiores que o pivô.
- O tempo de execução para arranjar os elementos é limitado pelo tamanho do vetor, no caso **n**. Observe no algoritmo que a condição de parada é quando $i \leq j$, na estrutura **do ... while($i <= j$);**. Considerando que i é o início do vetor e j a última posição, o laço é executado sobre todo o vetor. Logo, o procedimento realizará $O(n)$ comparações.
- No algoritmo **Quick Sort**, existe um aspecto particularmente problemático. Se o valor do pivô, para cada partição, for o maior ou menor elemento valor, então **Quick Sort** se degenerará em um ordenação tão lenta quanto o algoritmo **Insertion Sort**.

- Considerando o **pior caso** (pivô é sempre o maior ou menor elemento do vetor) e sabendo que o custo para particionar (ordenar os subvetores com elementos menores que o pivô à esquerda e os maiores à direita) consome $O(n)$, a recorrência para o tempo do algoritmo **Quick Sort** é:

$$T(n) = n + (n - 1) + (n - 2) + (n - 3) + \dots + 1 = \sum_{i=1}^n i = \frac{n(n + 1)}{2}$$

- Portanto, o tempo para execução do algoritmo **Quick Sort** para o pior caso é $\Theta(n^2)$.
- O pior caso do **Quick Sort** geralmente não ocorre. Veja que a cada recursão seria necessário que o elemento pivô fosse sempre o elemento maior ou menor do vetor.

- O **melhor caso** do algoritmo **Quick Sort** ocorre quando o procedimento de particionamento produz dois subvetores de tamanho $\frac{n}{2}$. A recorrência então é:

$$T(n) = n + T(n/2) + T(n/2) = 2T\left(\frac{n}{2}\right) + n$$

- Já resolvemos essa recorrência no algoritmo **Merge Sort**, veja página 133.
- Sendo assim, no **melhor caso** o algoritmo **Quick Sort** tem tempo de execução:

$$T(n) = \Theta(n \cdot \log n).$$

- 1 Introdução
- 2 Algoritmos de Ordenação Bubble Sort
- 3 Algoritmo de Ordenação Insert Sort
- 4 Algoritmo de Ordenação Selection Sort
- 5 Algoritmo de Ordenação Merge Sort
- 6 Algoritmo de Ordenação Quick Sort
- 7 Ordenando outras estruturas de dados

- Os exemplos apresentados ordenam dados do tipo **inteiros**. Entretanto, qualquer tipo de dado pode ser ordenado simplesmente trocando-se o tipo de dados dos parâmetros das funções.
- É comum que tipo de dados complexos, tais como: *strings* e *estruturas* precisem ser ordenados.
- É importante destacar que o independente do tipo de dados a ser ordenado, a lógica do algoritmo permanece inalterada.
- Os ajustes necessários limitam-se as seções de comparação, troca ou ambas.

Ordenação de Strings

```
void qs_string(char **item, int left, int right){  
    int i, j;  
    char *x;  
    char temp[80];  
  
    i = left ; j = right;  
    x = item[(left+right)/2];  
  
    do {  
        while(strcmp(item[i],x) < 0 && i<right) i++;  
        while(strcmp(item[j],x) > 0 && j>left) j--;  
  
        if(i<=j) {  
            strcpy(temp, item[i]);  
            strcpy(item[i],item[j]);  
            strcpy(item[j],temp);  
            i++;j--;  
        }  
    } while(i<=j);  
  
    if(left < j)  
        qs_string(item, left, j);  
    if(i < right)  
        qs_string(item,i, right);  
}
```

- Observe que o passo de comparação foi alterado para utilizar a função **strcmp()**. A função devolve um número negativo se a primeira string é lexicograficamente menor que a segunda, zero se as strings são iguais e um número positivo se a primeira string é lexicograficamente maior que a segunda.
- Observe também o uso da função **strcpy()** para trocar duas strings.
- Verifique que a lógica de funcionamento do algoritmo não foi alterada.

- Considere a **estrutura** abaixo:

```
struct client {  
    char nome[30];  
    int idade;  
    char cpf[14];  
};
```

- É possível ordenar um *array* de estruturas por qualquer um dos campos: **nome**, **idade** e **cpf**.
- O código no próximo slide apresenta um exemplo que ordena utilizando o algoritmo **Quick Sort** pelo campo **nome**. Analise as alterações no algoritmo.

Ordenação de Estruturas

```
void qs_struct(struct client item[], int left, int right){  
    int i,j;  
    char *x;  
    struct client temp;  
  
    i = left ; j = right;  
    x = item[(left+right)/2].nome;  
  
    do {  
        while(strcmp(item[i].nome,x) < 0 && i<right) i++;  
        while(strcmp(item[j].nome,x) > 0 && j>left) j--;  
  
        if(i<=j) {  
            temp = item[i];  
            item[i] =item[j];  
            item[j] =temp;  
            i++;j--;  
        }  
    } while(i<=j);  
  
    if(left < j)  
        qs_struct(item, left ,j);  
    if(i < right)  
        qs_struct(item ,i ,right );  
}
```

Obrigado pela atenção!!!
thiago.tavares@ifsuldeminas.edu.br



-  ASCENCIO, A.; CAMPOS, E. de. *Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ e Java*. Pearson Prentice Hall, 2008. ISBN 9788576051480. Disponível em: <<https://books.google.com.br/books?id=p-mTPgAACAAJ>>.
-  C: A Reference Manual. Pearson Education, 2007. ISBN 9788131714409. Disponível em: <<https://books.google.com.br/books?id=Wt2NEypdGNIC>>.
-  DAMAS, L. *LINGUAGEM C*. LTC. ISBN 9788521615194. Disponível em: <<https://books.google.com.br/books?id=22-vPgAACAAJ>>.
-  FEOFILOFF, P. *Algoritmos Em Linguagem C*. CAMPUS - RJ, 2009. ISBN 9788535232493. Disponível em: <<http://books.google.com.br/books?id=LfUQai78VQgC>>.
-  KERNIGHAN, B.; RITCHIE, D. *C: a linguagem de programação padrão ANSI*. Campus, 1989. ISBN 9788570015860. Disponível em: <<https://books.google.com.br/books?id=aVWrQwAACAAJ>>.

-  LOPES, A.; GARCIA, G. *Introdução à programação: 500 algoritmos resolvidos*. Campus, 2002. ISBN 9788535210194. Disponível em: <<https://books.google.com.br/books?id=Rd-LPgAACAAJ>>.
-  MIZRAHI, V. *Treinamento em linguagem C*. Pearson Prentice Hall, 2008. ISBN 9788576051916. Disponível em: <<https://books.google.com.br/books?id=7xt7PgAACAAJ>>.
-  SCHILDT, H.; MAYER, R. *C completo e total*. Makron, 1997. ISBN 9788534605953. Disponível em: <<https://books.google.com.br/books?id=PbI0AAAACAAJ>>.