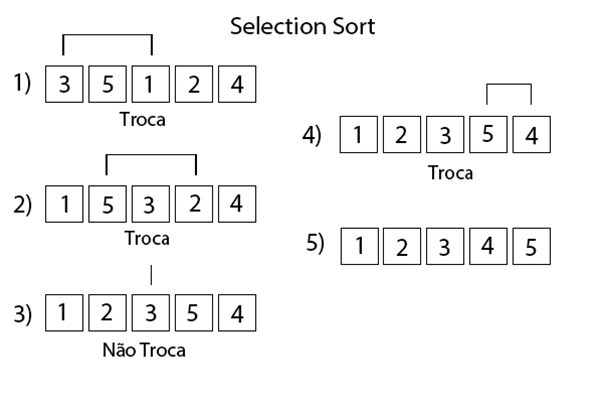
**Selection Sort**  
  
Neste algoritmo de ordenação é escolhido um número a partir do primeiro, este é comparado com os números a partir da sua direita, quando encontrado um número menor, o número escolhido ocupa a posição do menor número encontrado. Este número encontrado será o próximo número escolhido, caso não for encontrado nenhum número menor que este escolhido, ele é colocado na primeira posição (Atual posição do número escolhido), e o próximo número à sua direita vai ser o escolhido para fazer as comparações. É repetido esse processo até que a lista esteja ordenada.



**Passo a passo:**

* Neste passo o primeiro número escolhido foi o 3, ele foi comparado com todos os números à sua direita e o menor número encontrado foi o 1, então os dois trocam de lugar.
* O mesmo processo do passo 1 acontece, o número escolhido foi o 5 e o menor número encontrado foi o 2.
* Não foi encontrado nenhum número menor que 3, então ele fica na mesma posição.
* O número 5 foi escolhido novamente e o único número menor que ele à sua direita é o 4, então eles trocam.
* Vetor já ordenado

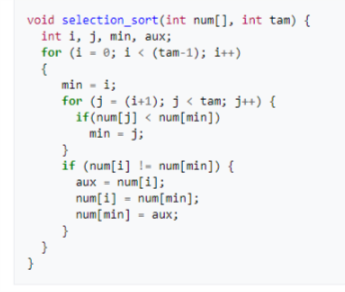
## **Vantagens**

* Ele é um algoritmo simples de ser implementado em comparação aos demais.
* Não necessita de um vetor auxiliar (in-place).
* Por não usar um vetor auxiliar para realizar a ordenação, ele ocupa menos memória.
* Ele é uns dos mais velozes na ordenação de vetores de tamanhos pequenos.

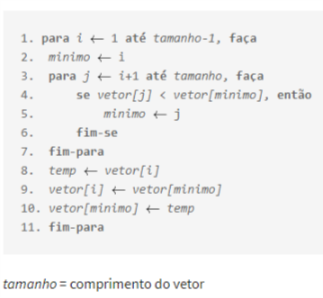
## **Desvantagens**

* Ele é um dos mais lentos para vetores de tamanhos grandes.
* Ele não é estável.
* Ele faz sempre **n^2** *comparações, independente do vetor está ordenado ou não.*

**Implementação em C:**



**Implementação em Portugol:**



<https://pt.wikipedia.org/wiki/Selection_sort>

<https://tiagomadeira.com/2006/01/ordenacao-por-selecao/>

<http://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261>

\*nota: Não sei se é realmente necessário colocar o conteúdo abaixo de desempenho, mas achei interessante xD.

**Comparação de Desempenho:**

### Ordem 1

**Tamanho do vetor: 100**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo** | **Tempo(ms)** | **Comparações** | **Movimentações** |
| Bubble sort | 0,0988 | 5050 | 0 |
| Selection Sort | 0,0602 | 4950 | 297 |
| Insertion sort | 0,0038 | 99 | 198 |
| Quick sort | 0,0141 | 606 | 189 |

**Tamanho do vetor: 1000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo** | **Tempo(ms)** | **Comparações** | **Movimentações** |
| Bubble sort | 9,5415 | 500500 | 0 |
| Selection Sort | 5,4587 | 499500 | 2997 |
| Insertion sort | 0,0359 | 999 | 1998 |
| Quick sort | 0,1602 | 9009 | 1533 |

**Tamanho do vetor: 10000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo** | **Tempo(ms)** | **Comparações** | **Movimentações** |
| Bubble sort | 934,5364 | 50005000 | 0 |
| Selection Sort | 508,5891 | 49995000 | 29997 |
| Insertion sort | 0,3558 | 9999 | 19998 |
| Quick sort | 2,0824 | 125439 | 17712 |

### Ordem 2

**Tamanho do vetor: 100**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo** | **Tempo(ms)** | **Comparações** | **Movimentações** |
| Bubble sort | 0,2045 | 5050 | 14850 |
| Selection Sort | 0,0750 | 4950 | 297 |
| Insertion sort | 0,1173 | 99 | 5148 |
| Quick sort | 0,0147 | 610 | 336 |

**Tamanho do vetor: 1000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo** | **Tempo(ms)** | **Comparações** | **Movimentações** |
| Bubble sort | 20,3377 | 500500 | 1498500 |
| Selection Sort | 6,9038 | 499500 | 2997 |
| Insertion sort | 11,4277 | 999 | 501498 |
| Quick sort | 0,1622 | 9016 | 3030 |

**Tamanho do vetor: 10000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo** | **Tempo(ms)** | **Comparações** | **Movimentações** |
| Bubble sort | 1838,0272 | 50005000 | 149985000 |
| Selection Sort | 665,2050 | 49995000 | 29997 |
| Insertion sort | 1074,1171 | 9999 | 50014998 |
| Quick sort | 2,1279 | 125452 | 32712 |

### Ordem 3

**Tamanho do vetor: 100**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo** | **Tempo(ms)** | **Comparações** | **Movimentações** |
| Bubble sort | 0,1596 | 5050 | 6777 |
| Selection Sort | 0,0698 | 4950 | 297 |
| Insertion sort | 0,0570 | 99 | 2457 |
| Quick sort | 0,0314 | 897 | 576 |

**Tamanho do vetor: 1000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo** | **Tempo(ms)** | **Comparações** | **Movimentações** |
| Bubble sort | 16,6730 | 500500 | 756840 |
| Selection Sort | 5,6664 | 499500 | 2997 |
| Insertion sort | 5,7523 | 999 | 254278 |
| Quick sort | 0,3725 | 13138 | 7983 |

**Tamanho do vetor: 10000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo** | **Tempo(ms)** | **Comparações** | **Movimentações** |
| Bubble sort | 1455,9734 | 50005000 | 74237889 |
| Selection Sort | 545,1068 | 49995000 | 29997 |
| Insertion sort | 539,6891 | 9999 | 24765961 |
| Quick sort | 4,5072 | 176065 | 103635 |

**Conclusão:**

**Bubble sort**

Para listas já ordenadas em ordem crescente é o único algoritmo que não realiza movimentações, mas em compensação é o que tem o maior tempo e o maior número de comparações. Não só em listas já ordenadas, mas em todos os casos o bubble sort se mostrou um algoritmo ineficiente.

**Selection sort**

Nas listas de ordem 1 e ordem 3, o selection sort foi o segundo pior algoritmo, mas se mostrou mais eficiente do que o Insertion sort em relação ao tempo e a quantidade de movimentações na lista de ordem 2.

**Insertion Sort**

Na lista de ordem 1, o Insertion sort se mostrou mais eficiente que todos os outros algoritmos em relação ao tempo e comparações. Na lista de ordem 2 foi menos eficiente do que o selection sort e na lista de ordem 3 a diferença de tempo entre o insertion e o selection foi pequena.

**Quick Sort**

O quick sort certamente é o algoritmo mais eficiente em listas totalmente desordenadas, ele se torna muito eficiente em relação aos outros no quesito de tempo. Na lista de ordem 3 e na de ordem 2 a diferença de tempo do quick sort em comparação aos outros foi absurdamente grande.