*Análise de Desempenho dos Algoritmos de Ordenação Selection Sort e Quick Sort*

Bruno Natanael Gomes Dias

Instituto Federal de Educação, Ciência e Tecnologia Rio Grande do Norte (IFRN)

**I. RESUMO**

Este trabalho mostra uma simples análise de desempenho entre os famosos algoritmos de ordenação Selection Sort e Quick Sort, utilizando a linguagem C++ para a implementação. Tendo como principais parâmetros de comparação a relação entre o tamanho dos arrays gerados de forma aleatória e o tempo de execução, foi feita uma bateria de testes empíricos utilizando os tamanhos de array: [10.000, 50.000, 100.000, 500.000, 1.000.000], em que os resultados foram salvos em um arquivo para que, utilizando Python, fossem gerados gráficos para comparação. O Quick Sort se mostrou infinitamente mais eficiente, principalmente com vetores grandes; quanto aos vetores menores, a diferença é mais sutil. Sendo assim, o Quick Sort é melhor em todas as situações, e o Selection Sort deve ser usado principalmente para fins didáticos.

**II. INTRODUÇÃO**

O principal intuito do trabalho é poder analisar o impacto de desempenho que, muitas vezes, uma simples linha de código pode gerar em seu programa final. Pois, na maioria dos casos, quanto mais rápido for resolvido o problema em questão, melhor seu código é. Por fim, foi solicitada a escolha de dois algoritmos de ordenação, onde seriam feitos testes para análise de desempenho. Na ocasião, optei pelo Selection Sort e pelo Quick Sort, sem nenhum critério específico, com o intuito de ver o quão contrastantes dois algoritmos que têm o mesmo propósito podem ser.

**III. METODOLOGIA**

A A análise de desempenho dos algoritmos de ordenação Selection Sort e Quick Sort foi realizada de forma empírica, em meu computador pessoal. O objetivo foi comparar o tempo de execução dos algoritmos para cinco tamanhos de vetores, com valores inteiros gerados aleatoriamente.

Os algoritmos implementados foram desenvolvidos em C++. Os vetores utilizados nos testes foram preenchidos com números inteiros aleatórios no intervalo de 0 a 99.999.

Foram utilizados os seguintes tamanhos de vetor:

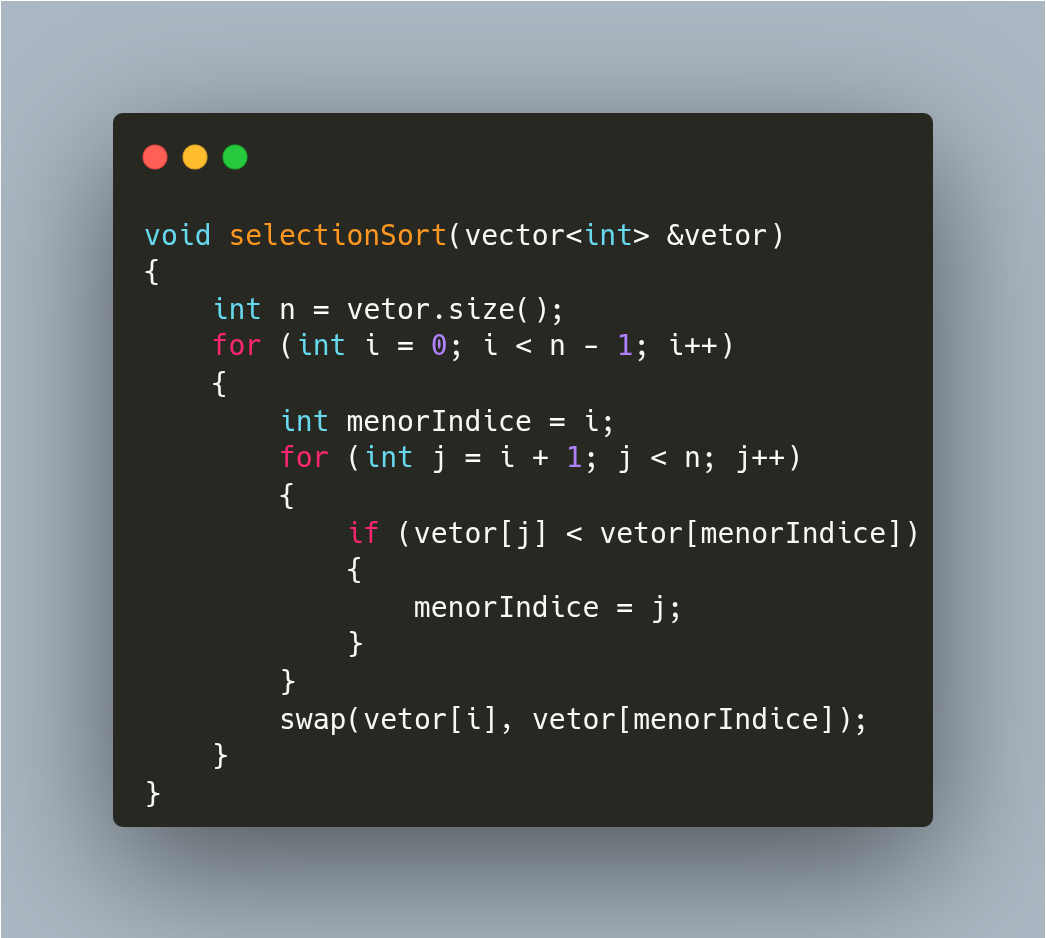
* 10.000 elementos
* 50.000 elementos
* 100.000 elementos
* 500.000 elementos
* 1.000.000 elementos

Esses valores foram escolhidos para permitir uma boa amostragem.

Após isso, criei um script em Python que coletava os resultados do teste que o C++ executava e os passava para um arquivo .CSV com as informações do algoritmo executado, tempo de execução e tamanho do array de entrada. A partir desse CSV, foi gerado um gráfico utilizando a biblioteca Matplotlib para fins de comparação.

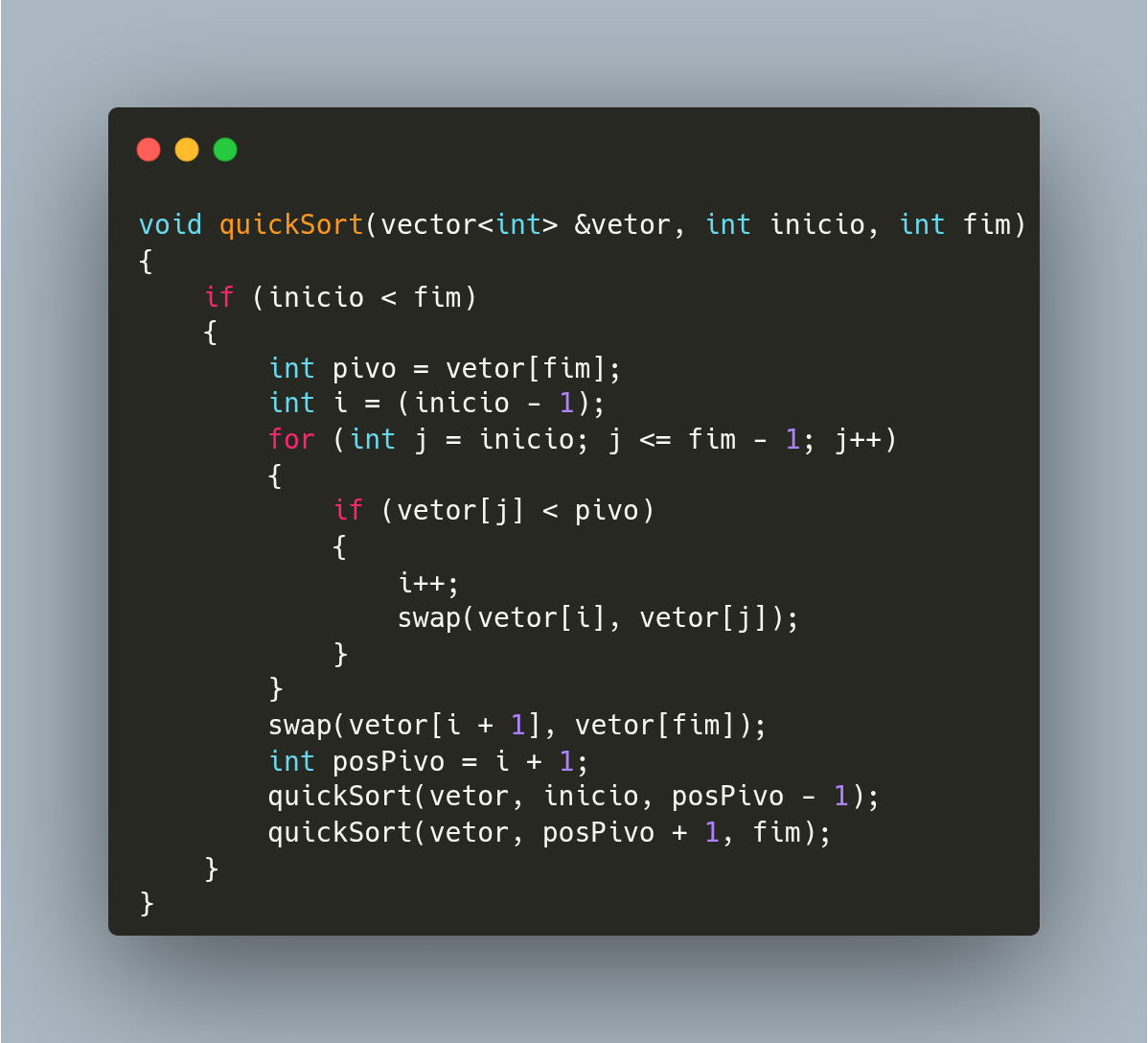
**IV. IMPLEMENTAÇÃO**

**Selection Sort:**



A implementação do Selection Sort se dá a partir de dois laços, “i” e “j”, em que “i" é o menor índice, enquanto “j” percorre todo o array para comparar com “i”. Caso seja menor, ocorre a troca de lugar; caso contrário, passa para o próximo índice do array, repetindo todo o processo.

**Quick Sort:**

A implementação do Quick Sort se dá a partir de uma função recursiva que escolhe um elemento como pivô, e então dois índices percorrem o array para comparar os elementos com o pivô. Os valores menores que o pivô é colocado à sua esquerda, e os maiores, à direita. Após essa separação, a função é chamada novamente para as partes esquerda e direita do array, repetindo todo o processo até que o vetor esteja completamente ordenado.

**V. RESULTADOS**

**Pior caso:**

O Selection Sort tem sua complexidade computacional de O(n²), pois, como podemos verificar em sua implementação, são apresentados dois laços aninhados, fazendo com que a lista seja percorrida n² vezes, independentemente da entrada.

O Quick Sort, assim como o algoritmo citado acima, tem sua complexidade computacional O(n²). Porém, para que isso ocorra, é necessário uma situação bem específica, na qual o pivô seja o último elemento e a lista já esteja ordenada.

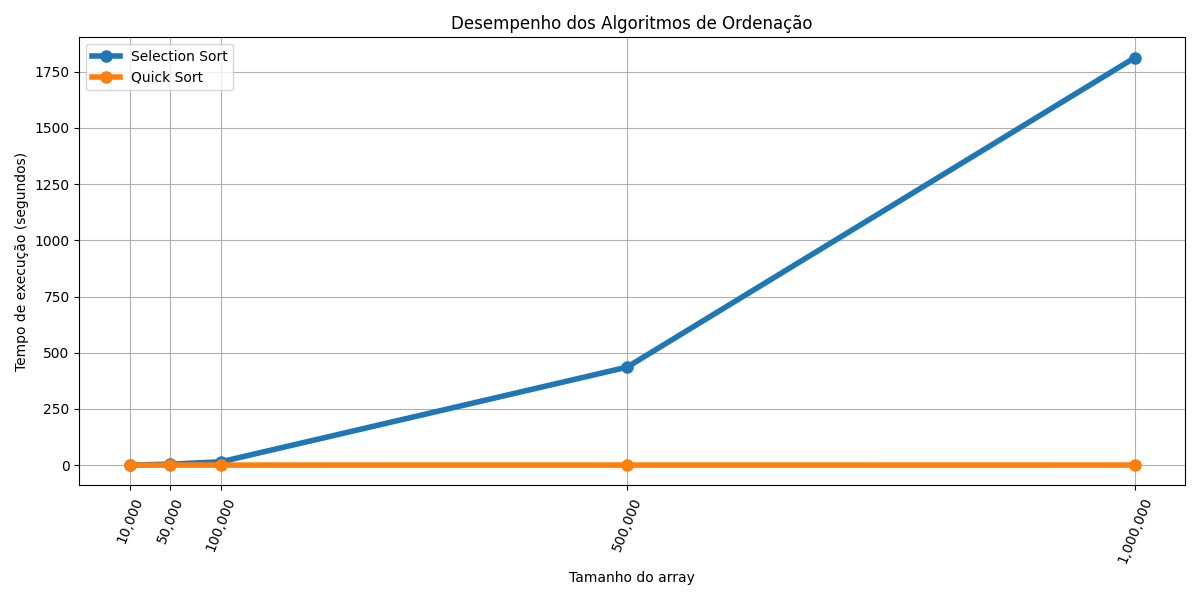
**Melhor caso:**

O Selection Sort, como mencionado acima, independentemente de sua entrada, tem sua complexidade computacional O(n²), percorrendo totalmente a lista n vezes.

Já o Quick Sort, em seu melhor caso, pode chegar a O(n log n), pois separa a lista em duas a cada passo, algo semelhante ao que acontece com a busca binária, utilizando a estratégia de dividir para conquistar.

C**omplexidade Empírica:**

A seguir, será possível observar o resultado da bateria de testes práticos, a fim de medir o desempenho em função do tempo, utilizando a biblioteca Matplotlib do Python.



A linha azul representa o Selection Sort, e a linha laranja representa o Quick Sort.

|  |  |  |
| --- | --- | --- |
| **Tamanho** | **Algoritmo** | **Tempo(s)** |
| 10.000 | Selection Sort | 0.13660 |
| 10.000 | Quick Sort | 0.00135 |
| 50.000 | Selection Sort | 4.04204 |
| 50.000 | Quick Sort | 0.00815 |
| 100.000 | Selection Sort | 14.7455 |
| 100.000 | Quick Sort | 0.01686 |
| 500.000 | Selection Sort | 435.853 |
| 500.000 | Quick Sort | 0.09424 |
| 1.000.000 | Selection Sort | 1812.66 |
| 1.000.000 | Quick Sort | 0.19834 |

Aqui está a tabela com os resultados obtidos para melhor visualização.

**Comparativo:**

Comparativamente, conseguimos ver que, quando a entrada é pequena, a diferença não é tão notável. Porém, ao escalonar o tamanho do array, o contraste no desempenho se torna gritante, e o Quick Sort se mostra superior em todos os aspectos.

**VI. CONCLUSÃO**

Com base nos testes realizados, foi possível observar de forma clara a diferença de desempenho entre os algoritmos Selection Sort e Quick Sort. Conforme o tamanho do vetor aumentava, o Quick Sort se mostrava cada vez mais eficiente, enquanto o Selection Sort apresentava um crescimento exponencial no tempo de execução, o que já era esperado pela sua complexidade O(n²).

Mesmo sendo dois algoritmos com o mesmo objetivo, a forma como eles resolvem o problema faz total diferença no tempo final. O Selection Sort é um algoritmo mais simples, mas pouco eficiente em situações reais, enquanto o Quick Sort se mostrou extremamente rápido, principalmente com vetores maiores.

Portanto, é possível concluir que o Quick Sort é a melhor escolha em praticamente todos os cenários de uso prático, e o Selection Sort deve ser utilizado apenas com fins didáticos, para compreensão inicial dos conceitos de ordenação.

**VII. REFERENCIAS**

PYTHON SOFTWARE FOUNDATION. *Matplotlib Documentation*. Disponível em: <https://matplotlib.org/stable/contents.html>. Acesso em: 09 jul. 2025. Lafore (2004) apresenta o **Selection Sort.** Sedgewick e Wayne (2011) exploram o **Quick Sort.**