Relatório Comparativo de Algoritmos de Ordenação

Lucas Teixeira Reis

[1]Pontifícia Universidade Católica de Minas Gerais (PUC-MG)

Resumo. *O relatório em questão compara, utilizando registros feitos previamente, o comportamento dos algoritmos de Seleção, Inserção, Bolha e Quicksort. Além disso, também é feita uma análise sobre os pontos fortes e fracos de cada um.*

1. Introdução

Saber quando utilizar um determinado algoritmo pode ser a virada de chave para um programa mais rápido, eficiente e seguro. Em *Algoritmos e Estruturas de Dados I*, nós aprendemos o básico sobre ordenação de arrays ou vetores, porém os algoritmos estudados não são eficientes para todos os casos. Dessa forma, em *Algoritmos e Estruturas de Dados II*, nós nos empenhamos para aprender novos algoritmos de ordenação, assim como também aprendemos a analisar a complexidade desses algoritmos. Isso é importante para que possamos utilizá-los em seus melhores momentos e, por consequência, consigamos ter um programa cada vez mais eficiente.

2. Metodologia

Para realizarmos este relatório, construímos quatro algoritmos contendo cada um dos métodos de ordenação previamente citados. Em cada um desses algoritmos, construímos quatro arrays de tamanhos 100, 1000, 10000 e 100000, que foram preenchidos aleatoriamente e, após isso, ordenados. Enquanto cada array é ordenado, foi contabilizado o tempo de execução em milissegundos, a quantidade de comparações e a quantidade de movimentações. Isso foi feito para cada um dos arrays e para cada um dos algoritmos. Cada algoritmo foi feito na linguagem *Java*.

Com os resultados em mãos, foram plotados vários gráficos utilizando a biblioteca **matplotlib**, da linguagem de programação *Python*. Esses gráficos serviram de argumento para analisarmos cada um dos algoritmos, atributo por atributo capturado pelos nossos algoritmos, ou seja, analisaremos o tempo, as comparações e as movimentações de cada algoritmo. Para finalizar, teremos uma conclusão abordando as vantagens de cada um.

3. Tempo de Execução

O tempo de execução de cada um dos algoritmos foi calculado em milissegundos. A seguir, podemos visualizar o tempo de execução de cada algoritmo, considerando o tamanho de cada array.

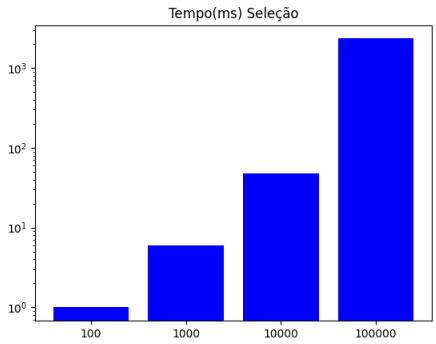

Figure 1. Tempo - Seleção

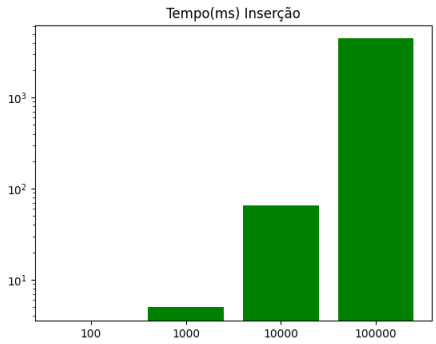

Figure 2. Tempo - Inserção

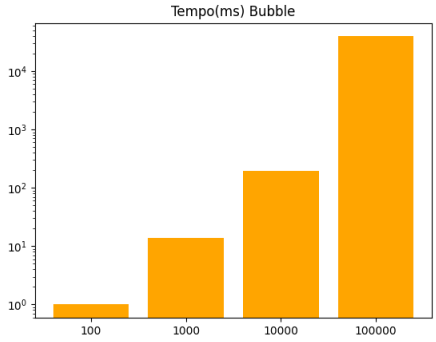

Figure 3. Tempo - Bolha

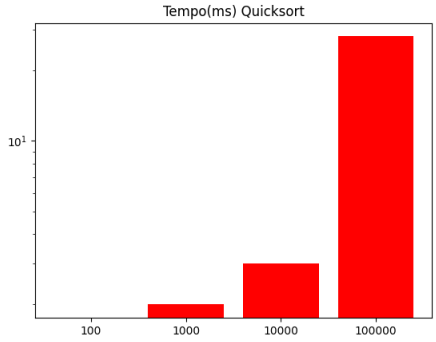

Figure 4. Tempo - Quicksort

Como podemos ver, todos os algoritmos ficam mais lentos conforme o tamanho dos arrays cresce, mas, dentre eles, podemos observar que os algoritmos de Inserção e Quicksort são quase instantâneos quando temos arrays de tamanho 100. Além disso, também podemos notar, pela escala do eixo Y, que eles são bem mais rápidos no caso médio, já que os arrays foram criadas de forma completamente aleatória. Isso fica ainda mais evidente nos gráficos a seguir, que mostram um comparativo geral de tempo de cada algoritmo lado a lado, desta vez separando cada gráfico pelo tamanho do array.

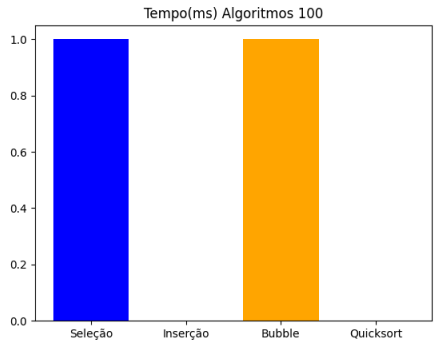

Figure 5. Geral - 100

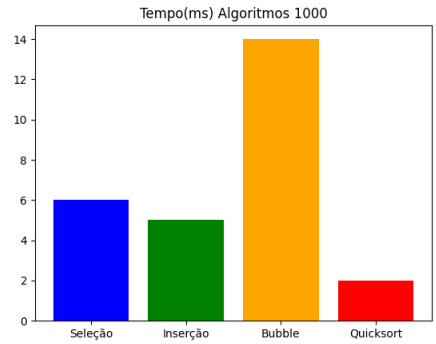

Figure 6. Geral - 1000

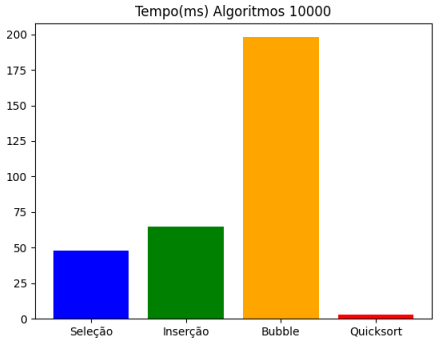

Figure 7. Geral - 10000

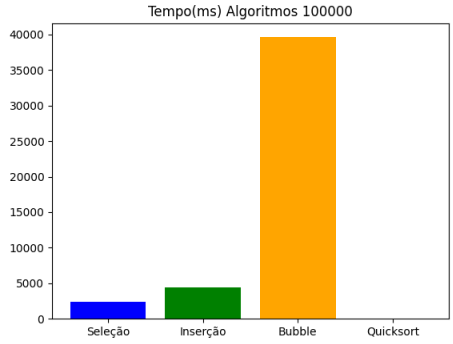

Figure 8. Geral - 100000

Vendo os gráficos lado a lado, fica fácil afirmar que o algoritmo mais rápido e o mais lento nessas condições são o Quicksort e o Bolha, respectivamente.

4. Quantidade de Comparações

A quantidade de comparações de cada um dos algoritmos foi calculada de forma que, cada vez que o algoritmo passasse por uma condicional para verificar se um número era maior que o outro, era incrementada uma variável do tipo inteiro. A seguir, poderemos visualizar a quantidade de comparações de cada um dos algoritmos, considerando o tamanho de cada array.

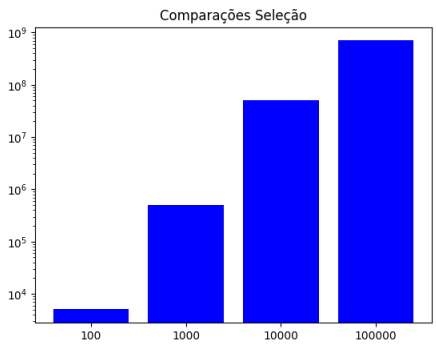

Figure 9. Comparações - Seleção

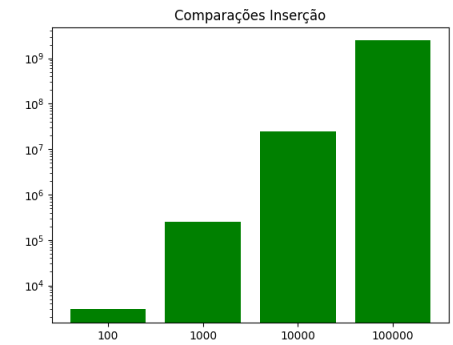

Figure 10. Comparações - Inserção

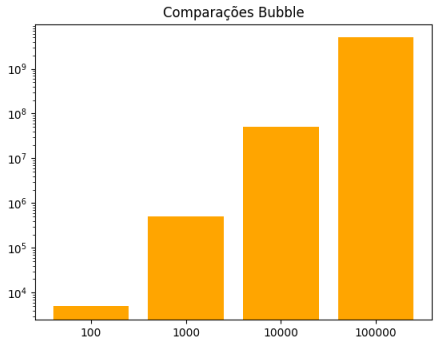

Figure 11. Comparações - Bolha

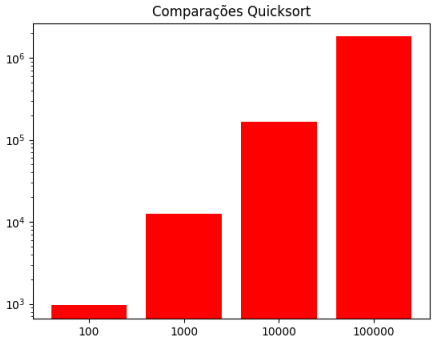

Figure 12. Comparações - Quicksort

Como podemos ver, novamente o tamanho do array é um grande problema para cada um dos algoritmos, mas temos algumas mudanças. Diferentemente do tempo, desta vez conseguimos observar que o algoritmo de Seleção fica bem parelho com o algoritmo de Bolha. Isso ocorre porque cada um deles compara quase todo o array em cada execução. Já o Quicksort e o Inserção vão depender bem mais de como o array foi montado e da escolha do pivô para ter um número maior ou menor de comparações, o que lhes confere mais vantagem nesse quesito.

Vamos agora ver os algoritmos lado a lado:

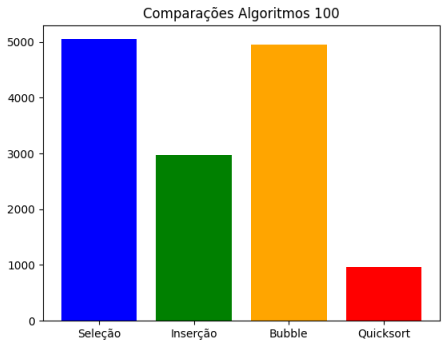

Figure 13. Geral - 100

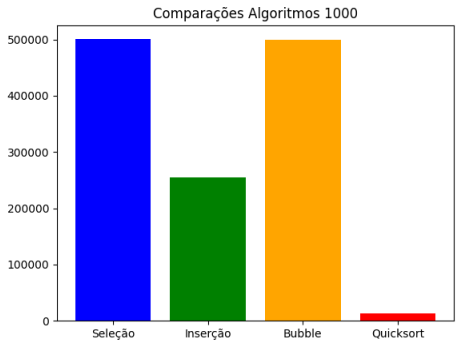

Figure 14. Geral - 1000

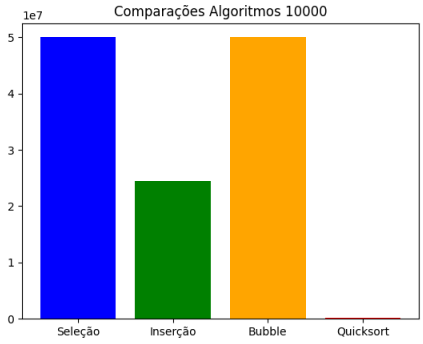

Figure 15. Geral - 10000

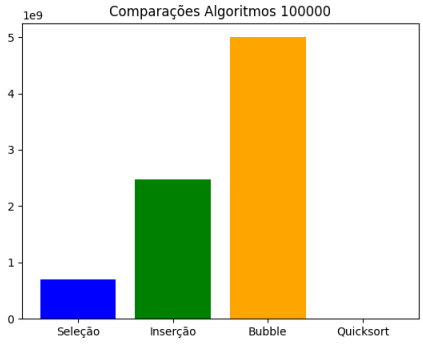

Figure 16. Geral - 100000

Como podemos ver, novamente conseguimos perceber que o Quicksort é o algoritmo mais eficiente. Porém, desta vez, os algoritmos de Seleção e Bolha ficam bem próximos no quesito comparação na ordenação de cada um dos arrays!

5. Quantidade de Movimentações

Assim como na quantidade de comparações, a quantidade de movimentações foi calculada com o auxílio de uma variável do tipo inteiro que era incrementada a cada swap ou a cada vez que um dado era movido para frente no nosso código.

Vamos ver os gráficos de cada um individualmente:

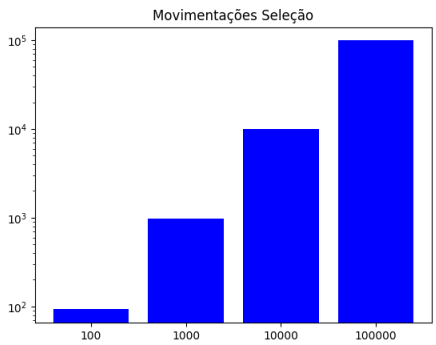

Figure 17. Movimentações - Seleção

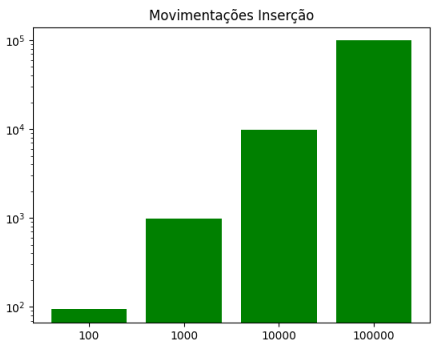

Figure 18. Movimentações - Inserção

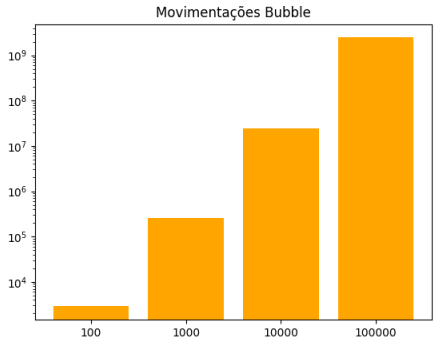

Figure 19. Movimentações - Bolha

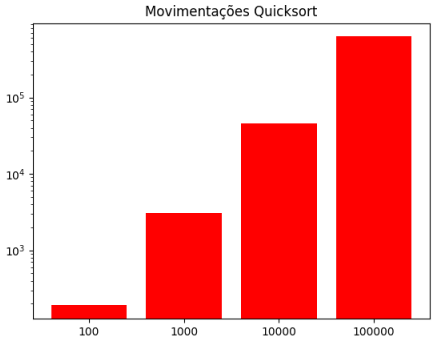

Figure 20. Movimentações - Quicksort

Como mostrado nos gráficos, novamente temos o Quicksort como o mais eficiente dentre os quatro. Mas, desta vez, assim como na comparação de tempo, o algoritmo de Bolha é o menos eficiente no quesito de movimentações. Isso ocorre porque, a cada comparação bem-sucedida do Bolha — que, como vimos na última seção, são muitas —, o Bolha faz uma movimentação. Isso é diferente nos outros algoritmos: o Seleção só movimenta quando encontra o menor número, e no caso do Quicksort e do Inserção, como discutimos anteriormente, vai depender muito da escolha do pivô e de como o array foi construído.

Vamos ver agora as movimentações gerais:

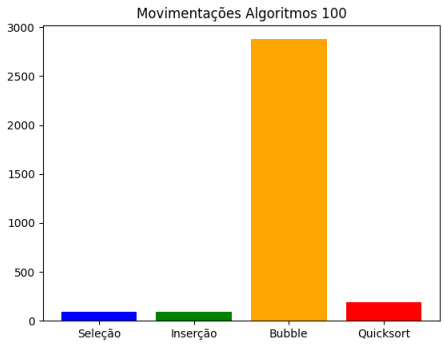

Figure 21. Geral - 100

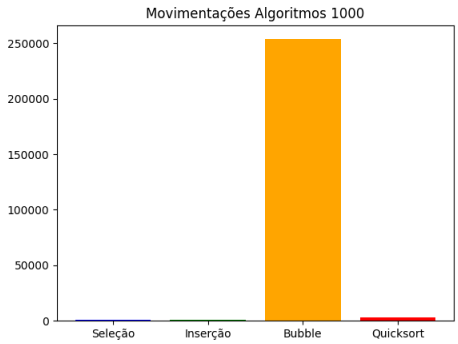

Figure 22. Geral - 1000

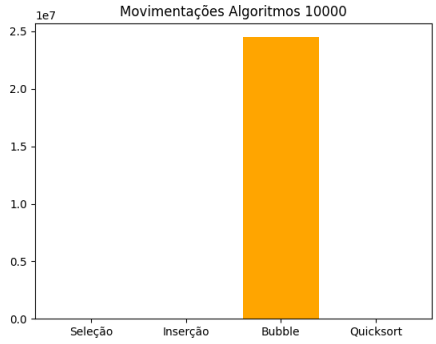

Figure 23. Geral - 10000

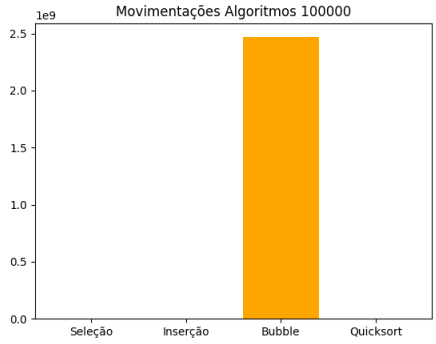

Figure 24. Geral - 100000

Agora fica bem fácil perceber que o Bolha se destaca negativamente na quantidade de movimentações, deixando claro que, se na hora de construirmos o nosso algoritmo quisermos evitar o algoritmo com maior número de movimentações, devemos evitar o Bolha!

6. Conclusão

Com isso, podemos concluir que o Quicksort é o melhor algoritmo nos casos médios. Como pudemos perceber, ele foi o algoritmo mais eficiente em quase todos os gráficos. Isso ocorre por conta de como ele é construído, buscando otimizar quase todos os requisitos de uma ordenação: escolher o pivô para evitar comparações e movimentações e ir analisando cada parte do array individualmente.

Mas isso se refere somente ao quesito desempenho. Os outros algoritmos se destacam por outras qualidades não mencionadas ainda neste relatório. O Bolha, por exemplo, apesar de ser o pior no quesito performance, tem uma implementação simples e é estável. Já o Inserção também é estável e, no caso médio, é bem mais eficiente que o Bolha, embora, no pior caso, possa ter um desempenho equivalente.

O algoritmo de Seleção seria um intermediário entre todos eles. Não é estável nem o mais eficiente, mas, ao mesmo tempo, tem uma implementação simples e fácil de manter, assim como o Bolha.

Dessa forma, conseguimos perceber que, considerando os algoritmos presentes neste relatório, Inserção e Bolha levam vantagem quando o assunto é estabilidade; Quicksort e Inserção levam vantagem quando o assunto é desempenho; e Seleção e Bolha levam vantagem quando o assunto é simplicidade. Portanto, é necessário pensar bem em qual algoritmo usar em cada situação, pois isso pode definir o desempenho do nosso programa.