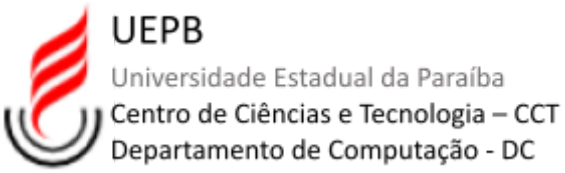


DISCIPLINA: Laboratório de estrutura de dados

Luiz José Mendonça Duarte
Raquel Melo de Queiroz

Análise de comportamento de algoritmos

Utilizando Bubble, Selection e Insertion Sort

Campina Grande - PB
2024

1. Introdução

Este relatório foi desenvolvido com o objetivo da coleta de dados a partir da análise dos algoritmos de ordenação: Bubble, Selection e Insertion Sort, a fim de exibir seus desempenhos.

Visão geral do projeto

- **Introdução:** Apresentação do projeto;
- **Descrição geral sobre o método utilizado:** Descrição dos métodos, passos e especificações de arquivos tais como tabelas e gráficos.
- **Resultados e análise:** Resultados obtidos e comentados a partir da coleta dos dados.

2. Descrição geral sobre o método utilizado

Inicialmente, usamos os algoritmos de ordenação Bubble, Selection e Insertion Sort para analisarmos seu comportamento mediante a diferentes aplicações no algoritmo, tais como:

- Tempo de execução;
- Número de trocas.

As diferentes aplicações se referem à:

- 5 (cinco) tentativas de execução para calcular a média de trocas realizadas e a média do tempo usado;

Utilizamos os números de entrada: 100, 10.000 e 20.000, aplicando cada um deles respectivamente a cada algoritmo de ordenação, testando cada entrada 5 (cinco) vezes.

Ao final, fizemos a média do número de trocas, e a média do tempo.

Descrição da implementação da ferramenta (IDE) utilizada:

- Uso do Eclipse IDE 2023-09.

3. Resultados e análise

A partir da elaboração das tabelas e dos gráficos, podemos ter uma análise firme e concisa do comportamento desses algoritmos.

Bubble Sort

100 números

	Trocas	Tempo
1ª tentativa	**2547**	**0.4038ms**
2ª tentativa	**2326**	**0.3491ms**
3ª tentativa	**2360**	**0.3903ms**
4ª tentativa	**2700**	**0.2211ms**
5ª tentativa	**2133**	**0.3727ms**
Média	**2533.2**	**0.3472ms**

Note que o número de trocas foi relativamente baixo, uma vez que estamos tratando da entrada de 100 números. E o tempo foi extremamente baixo. Provando que com o valor 100 números de entrada, não tivemos grandes dificuldades em executar o código.

10.000 números

	Trocas	Tempo
1ª tentativa	**25102654**	**261.3392ms**
2ª tentativa	**25055558**	**318.507ms**
3ª tentativa	**25000140**	**272.1164ms**
4ª tentativa	**25071396**	**264.2897ms**

5ª tentativa	**25424021**	**237.5527ms**
Média	**25090753.8**	**270.3618ms**

20.000 números

	Trocas	Tempo
1ª tentativa	**100535317**	**1015.7593ms**
2ª tentativa	**100404842**	**1258.4245ms**
3ª tentativa	**99460901**	**1692.8701ms**
4ª tentativa	**100898321**	**1018.2173ms**
5ª tentativa	**100712096**	**1091.2997ms**
Média	**100828895.4**	**1215.91498ms**

A partir do uso da entrada de 10.000 e, posteriormente, 20.000 números, podemos perceber que o tempo aumentou em grande quantidade em comparação à primeira entrada. Posteriormente fica claro que de 10.000 números para 20.000 números de entrada, tivemos um valor dobrado dos números de entrada. Entretanto, analisando a média do tempo utilizado, podemos perceber que o tempo não foi respectivamente o dobro, mas sim quase 5x maior do que o anterior (10.000).

Selection Sort

100 números

	Trocas	Tempo
1ª tentativa	**2592**	**0.0048ms**
2ª tentativa	**2603**	**0.0043ms**
3ª tentativa	**2544**	**0.004ms**
4ª tentativa	**2399**	**0.0063ms**
5ª tentativa	**2362**	**0.0065ms**
Média	**2500**	**0.00518ms**

10.000 números

	Trocas	Tempo
1ª tentativa	**24765166**	**0.0478ms**
2ª tentativa	**25107682**	**0.0712ms**
3ª tentativa	**25138108**	**0.0769ms**
4ª tentativa	**24982479**	**0.0882ms**
5ª tentativa	**24671100**	**0.0837ms**
Média	**24.932.907**	**0.07356ms**

20.000 números

	Trocas	Tempo
1ª tentativa	**99602583**	**0.0833ms**
2ª tentativa	**100227290**	**0.0802ms**
3ª tentativa	**99595404**	**0.097ms**
4ª tentativa	**98599172**	**0.0871ms**
5ª tentativa	**100172993**	**0.1153ms**
Média	**99.639.488,4**	**0.09258ms**

Insertion Sort

100 trocas

	Trocas	Tempo
1ª tentativa	**2608**	**0.2245ms**
2ª tentativa	**2296**	**0.1188ms**
3ª tentativa	**2406**	**0.1358ms**
4ª tentativa	**2367**	**0.1246ms**
5ª tentativa	**2204**	**0.1174ms**

Média	**2376.2**	**0.14422ms**

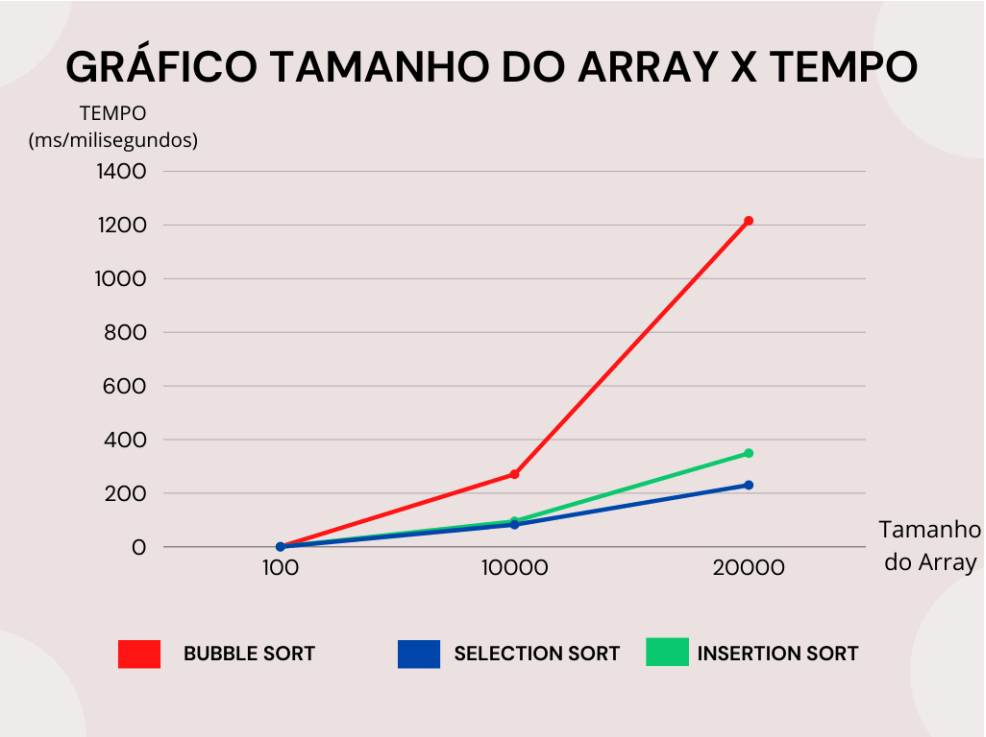
10.000 trocas

	Trocas	**Tempo**
1ª tentativa	**24808178**	**117.0085ms**
2ª tentativa	**25132235**	**77.9845ms**
3ª tentativa	**24772792**	**116.7189ms**
4ª tentativa	**25180075**	**85.1521ms**
5ª tentativa	**25313860**	**82.6878ms**
Média	**25041428**	**95.31036ms**

20.000 trocas

	Trocas	**Tempo**
1ª tentativa	**99778748**	**425.4543ms**
2ª tentativa	**100092744**	**349.1732ms**
3ª tentativa	**100380274**	**260.392ms**
4ª tentativa	**100012380**	**280.6455ms**
5ª tentativa	**100963840**	**425.286ms**
Média	**100044997.2**	**348.790ms**

Gráficos

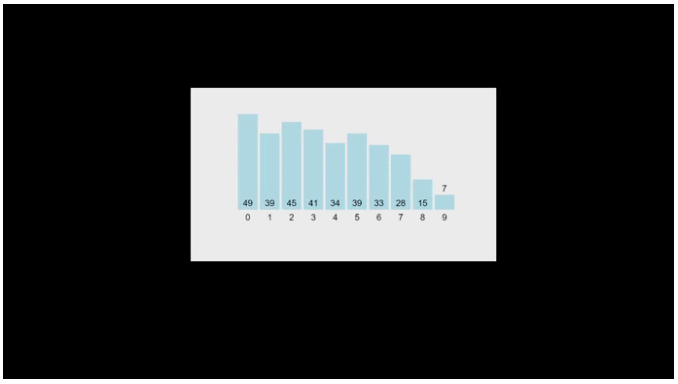


OBS: Note que os valores do bubble e do insertion sort estão muito próximos.

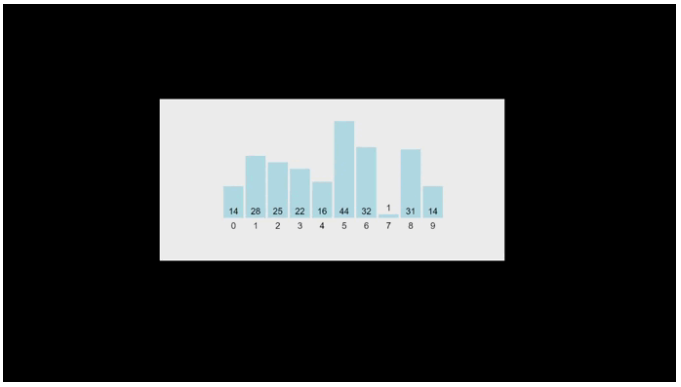
4. Análise dos dados

Dessa forma, percebemos que o algoritmo mais rápido e que efetua menos trocas é o Insertion, pois ele é o único desses O(n), enquanto o Bubble e Selection são O(n^2), sendo assim, em um melhor caso que os elementos do array estão ordenados o Insertion Sort possui um desempenho bem melhor que o seus concorrentes.

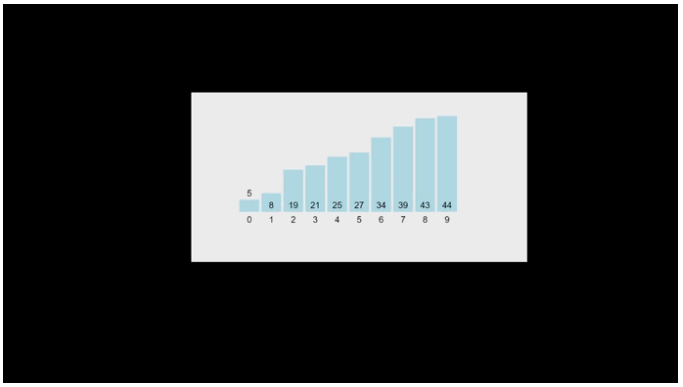
Com os dados de troca e a forma de funcionamento desses algoritmos também conseguimos perceber o “porquê” do desempenho, o Bubble sort por exemplo, é um algoritmo que tende a demorar mais devido ao fato de que ele percorre elemento por elemento comparando para ver se a necessidade de troca, dessa forma, se a lista estiver em pior caso. Ele tende a demorar mais que o normal.



Temos ao lado exemplos do Bubble sort, em pior, médio e melhor caso respectivamente.
Percebe-se o quanto ele demora no pior caso(primeiro gif), pois ele confere todos os elementos e compara todos.



No médio caso, ocorre em tempo razoável devido ao fato que tem alguns elementos ordenados e outros não, então ele não faz tantas trocas.



Já no melhor caso, ele passa muito rápido devido ao fato que ele só compara todos os elementos.