Relatório de Análise de Algoritmos de Ordenação

1. Introdução

Este relatório tem como objetivo apresentar uma análise simulada do desempenho de três algoritmos de ordenação: Bubble Sort, Insertion Sort e Quick Sort. Esses algoritmos foram aplicados a conjuntos de dados de tamanhos variados, organizados em três ordens distintas: aleatória, ordenada em ordem crescente e ordenada em ordem decrescente. Os tempos de execução foram medidos e registrados em milissegundos (ms) para destacar as diferenças de desempenho entre os algoritmos em cada tipo de conjunto de dados.

2. Metodologia

Neste experimento, os algoritmos de ordenação foram executados em três conjuntos de dados pré-definidos:

- **Aleatório:** Conjunto de dados com elementos organizados em ordem aleatória.
- **Ordenado Crescente:** Conjunto com elementos organizados em ordem crescente.
- **Ordenado Decrescente:** Conjunto com elementos organizados em ordem decrescente.

Cada algoritmo foi implementado para ordenar os elementos de cada conjunto de dados, e o tempo de execução foi medido em milissegundos (ms) para cada execução.

3. Resultados

Resultados do tempo de execução de cada tipo de arquivo

Aleatório

x	*aleatorio_100.csv*	*aleatorio_1000.csv*	*aleatorio_10000.csv*
Bubble Sort	0.6879 ms	3.5257 ms	101.842 ms
Insertion Sort	0.6461 ms	2.022 ms	27.5385 ms
Quick Sort	0.5201 ms	0.721 ms	2.3492 ms

- Cada execução de cada arquivo gerou os tempos da tabela de cima.

As médias dos tempos de execução para cada algoritmo são:

- 🥇 **Quick Sort:** 1.1968 ms
- 🥈 **Insertion Sort:** 10.0689 ms
- 🥉 **Bubble Sort:** 35.3519 ms

Crescente

x	*crescente_100.csv*	*crescente_1000.csv*	*crescente_10000.csv*
Bubble Sort	0.5188 ms	2.7916 ms	16.0385 ms
Insertion Sort	0.5152 ms	0.6108 ms	0.736 ms
Quick Sort	0.6089 ms	3.2297 ms	21.7374 ms

- Cada execução de cada arquivo gerou os tempos da tabela de cima.

As médias dos tempos de execução para cada algoritmo são:

- 🥇 **Insertion Sort:** 0.6207 ms
- 🥈 **Bubble Sort:** 6.4496 ms
- 🥉 **Quick Sort:** 8.5253 ms

Decrescente

x	*decrescente_100.csv*	*decrescente_1000.csv*	*decrescente_10000.csv*
Bubble Sort	0.8049 ms	3.4761 ms	32.5785 ms
Insertion Sort	0.7183 ms	2.6649 ms	43.3098 ms
Quick Sort	0.7463 ms	2.9748 ms	37.7004 ms

- Cada execução de cada arquivo gerou os tempos da tabela de cima.

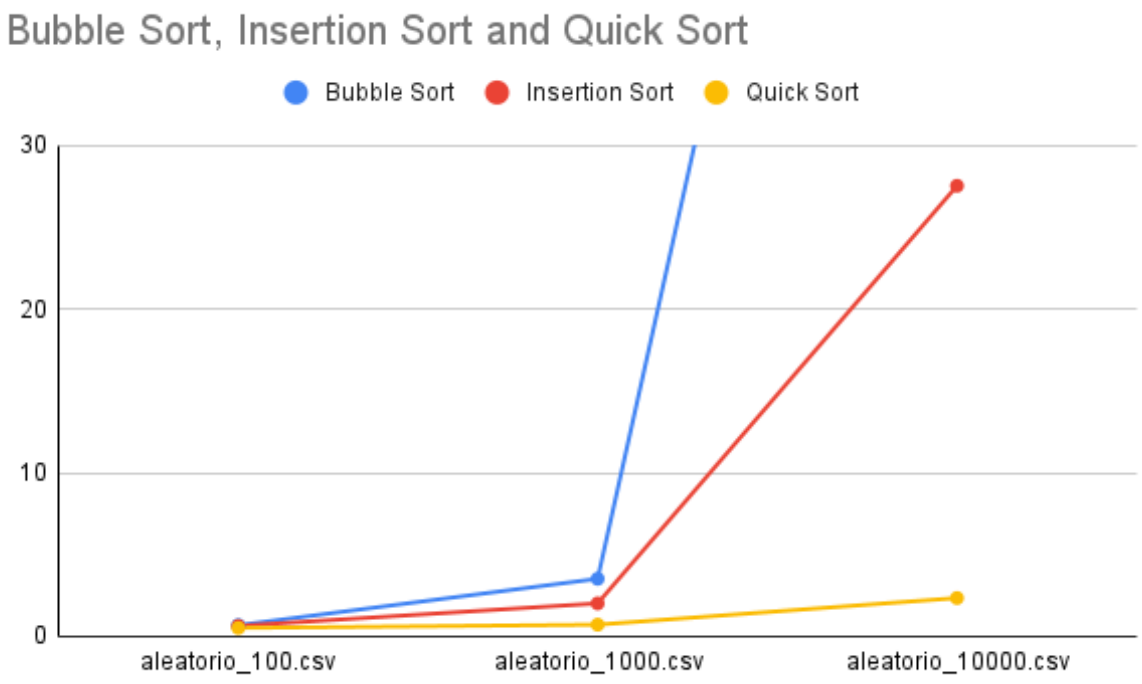
As médias dos tempos de execução para cada algoritmo são:

- 🥇 **Bubble Sort:** 12.29 ms
- 🥈 **Quick Sort:** 13.81 ms
- 🥉 **Insertion Sort:** 15.56 ms

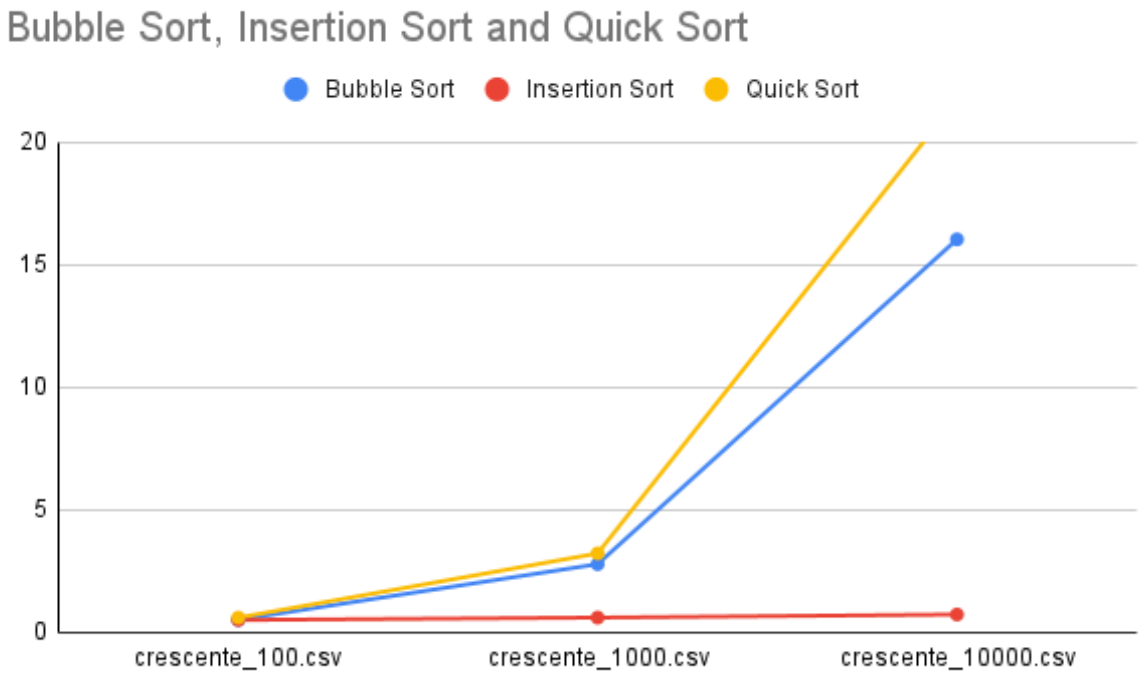
4. Gráficos

Segue gráficos gerados para melhor visualização dos tempos de algoritmos

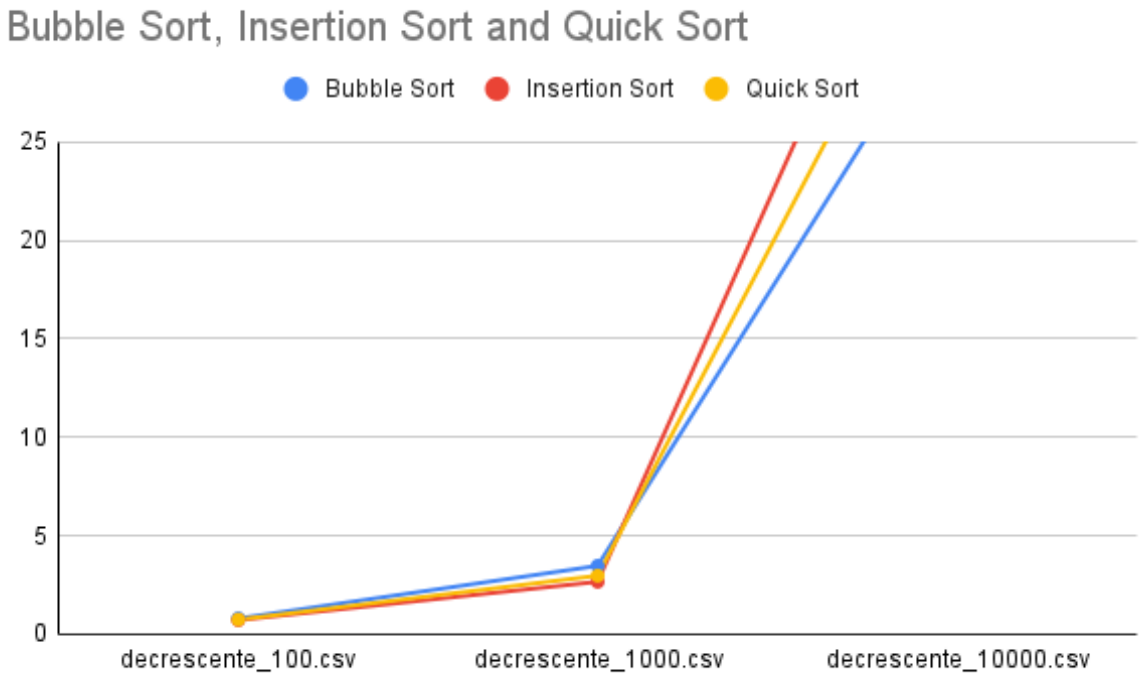
Aleatório



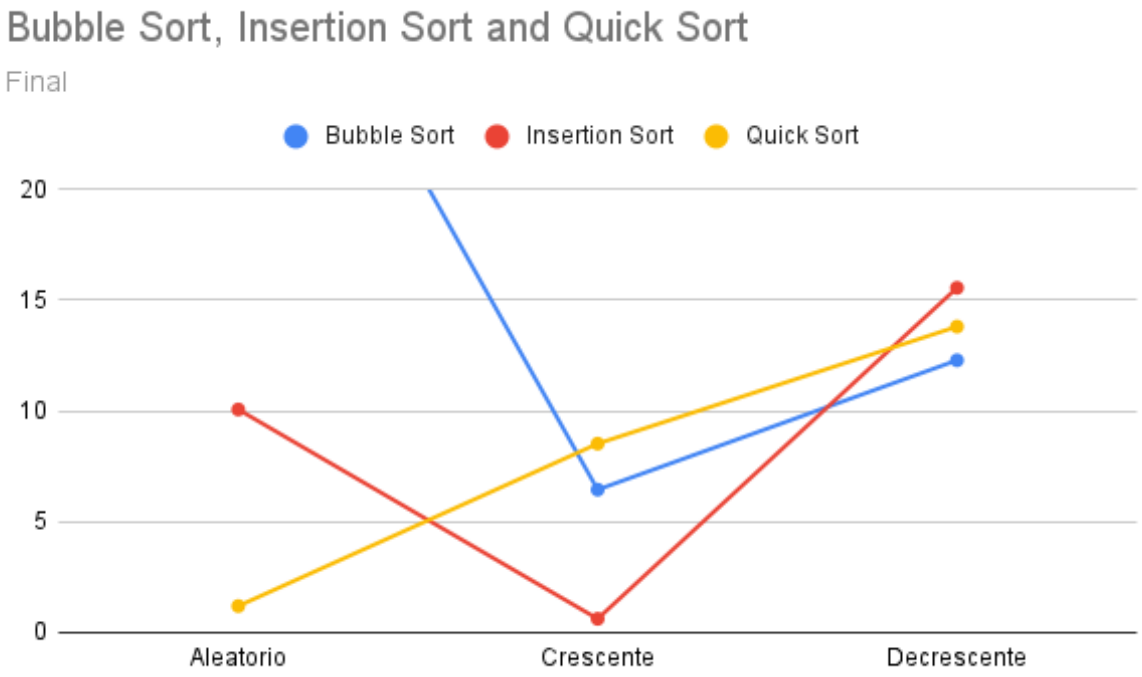
Crescente



Decrescente



Final



5. Análise de Resultados

1. Aleatório

Quando os dados são aleatórios, observamos que o **Quick Sort** foi o algoritmo mais eficiente, com uma média de tempo de execução de **1.1968 ms**. Esse desempenho superior pode ser atribuído à sua abordagem eficiente de divisão e conquista, que tende a ser muito mais rápida em cenários aleatórios.

O **Insertion Sort**, com uma média de **10.0689 ms**, também se saiu razoavelmente bem, embora seu desempenho tenha sido significativamente mais lento que o do Quick Sort. Este algoritmo, embora eficiente para conjuntos pequenos e quase ordenados, sofre um grande impacto no desempenho quando aplicado a dados aleatórios.

Por último, o **Bubble Sort** teve o pior desempenho, com uma média de **35.3519 ms**. O Bubble Sort, mostra uma performance lenta, principalmente com dados aleatórios, onde seu comportamento de troca repetitiva de elementos é mais pronunciado.

2. Crescente

Em dados já ordenados de forma crescente, o algoritmo **Insertion Sort** se destacou com o melhor desempenho, com uma média de **0.6207 ms**. Isso ocorre porque o **Insertion Sort** é altamente eficiente quando os dados já estão ordenados ou quase ordenados, já que ele faz apenas comparações mínimas e inserções diretas. Isso é refletido em tempos de execução significativamente menores em comparação aos outros algoritmos.

O **Bubble Sort**, com uma média de **6.4496 ms**, foi o segundo melhor, mas ainda muito mais lento que o Insertion Sort. Embora o Bubble Sort seja relativamente eficiente com dados já ordenados, ele ainda requer várias passagens para verificar a ordenação, o que contribui para seu tempo de execução maior.

Já o **Quick Sort**, com uma média de **8.5253 ms**, foi o algoritmo menos eficiente para conjuntos de dados ordenados crescentemente. Isso pode ser explicado pelo fato de que o Quick Sort, quando mal particionado (como ocorre com dados já ordenados ou quase ordenados), pode sofrer uma degradação de desempenho devido a divisões desbalanceadas.

3. Decrescente

Quando aplicados a conjuntos de dados ordenados de forma decrescente, os tempos de execução mostraram uma inversão nos resultados. O **Bubble Sort** teve o melhor desempenho, com uma média de **12.29 ms**. Isso ocorre porque o Bubble Sort, ao detectar rapidamente que os elementos precisam ser trocados, consegue atingir rapidamente a ordenação de dados invertidos, embora ainda esteja longe de ser tão eficiente quanto o Quick Sort.

O **Quick Sort** obteve uma média de **13.81 ms**, ficando em segundo lugar, o que sugere que, em conjuntos decrescentes, o algoritmo não se beneficia tanto quanto em dados aleatórios. Embora ainda seja eficiente, o Quick Sort não se sai tão bem quanto o Bubble Sort devido à forma como suas divisões podem ser desequilibradas em conjuntos ordenados inversamente.

Por fim, o **Insertion Sort** apresentou o pior desempenho, com uma média de **15.56 ms**, embora o algoritmo fosse o segundo melhor nos conjuntos ordenados crescentemente. Isso se deve ao fato de que o **Insertion Sort** exige muito mais deslocamentos de elementos em listas ordenadas de forma decrescente, levando a um aumento no tempo de execução.

6. Conclusão

A análise dos tempos de execução revela que, em **conjuntos de dados aleatórios**, o **Quick Sort** se destaca, enquanto os algoritmos **Bubble Sort** e **Insertion Sort** mostram um desempenho consideravelmente inferior.

Em **conjuntos de dados ordenados crescentemente**, o **Insertion Sort** é o mais eficiente, aproveitando-se de sua eficiência para listas parcialmente ordenadas. O **Bubble Sort** ainda mostra desempenho razoável, enquanto o **Quick Sort** sofre uma degradação devido a particionamentos ineficientes.

Já em **conjuntos decrescentes**, o **Bubble Sort** se comporta de maneira mais eficiente, embora ainda com desempenho inferior ao de Quick Sort para listas aleatórias. O **Insertion Sort** mostra um desempenho significativamente mais fraco devido ao alto número de deslocamentos necessários para reordenar os dados.

Portanto, **Quick Sort** é o algoritmo ideal para dados aleatórios de grandes dimensões, enquanto **Insertion Sort** é melhor para dados já ordenados ou quase ordenados. O **Bubble Sort**, embora simples, pode ser eficaz em listas pequenas e em dados invertidos, mas, de maneira geral, apresenta um desempenho inferior aos outros algoritmos.