Relatório de Análise de Algoritmos de Ordenação

Este relatório apresenta um comparativo de desempenho entre os algoritmos Bubble Sort, Insertion Sort e Quick Sort, os quais foram desenvolvidos durante o semestre letivo da disciplina de Resolução de Problemas Estruturados em Computação.

Os algoritmos foram testados sob três conjuntos de dados distintos, sendo eles números ordenados de forma crescente, aleatória e decrescente. Além disso, cada conjunto possui três tamanhos distintos, 100 elementos, 1000 elementos e 10000 elementos.

Nas tabelas abaixo, para cada um dos tamanhos de conjunto de dados, os tempos de execução para cada algoritmo são medidos em milissegundos (ms).

Conjunto de dados de 100 elementos:

Tipo de conjunto de dados	Bubble Sort	Insertion Sort	Quick Sort
Crescente	0.2166	0.0101	0.1274
Aleatório	0.3727	0.1714	0.0605
Decrescente	0.4922	0.3044	0.317

Conjunto de dados de 1000 elementos:

Tipo de conjunto de dados	Bubble Sort	Insertion Sort	Quick Sort
Crescente	5.0563	0.1215	3.4014
Aleatório	11.3054	7.1248	1.0154
Decrescente	15.8065	9.2806	9.8125

Conjunto de dados de 10000 elementos:

Tipo de conjunto de dados	Bubble Sort	Insertion Sort	Quick Sort
Crescente	162.0863	0.8827	180.9415
Aleatório	320.0825	127.874	3.9856
Decrescente	244.9269	249.6896	248.5297

Para que a análise dos dados seja facilitada, está disposto abaixo para cada uma das tabelas, um gráfico comparando a eficiência dos algoritmos para os conjuntos de dados, onde o eixo y representa o tempo em milissegundos, enquanto o eixo x o conjunto de dados utilizado.

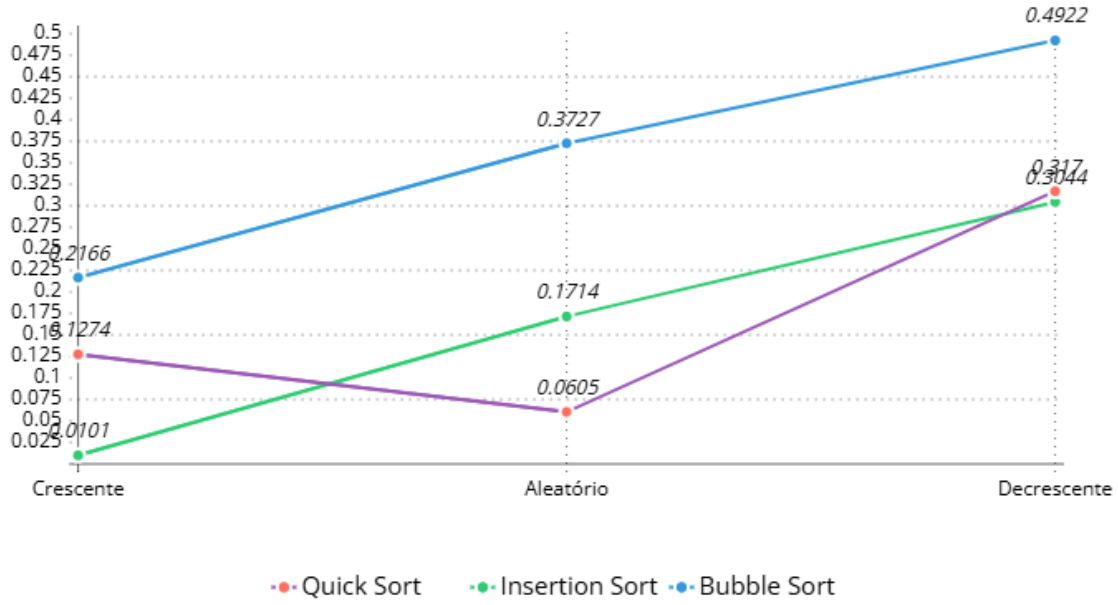

Gráfico comparativo de tempo de execução para os algoritmos em um conjunto de dados de 100 elementos.

Neste gráfico, pode-se notar algumas diferenças entre o desempenho dos algoritmos. O Insertion Sort, devido a sua natureza, consegue em sua primeira iteração concluir que o conjunto já está ordenado, encerrando imediatamente a sua execução.

O Bubble Sort, por sua vez, faz uso de comparações entre dois elementos para posicionar o maior na última posição, ainda que o conjunto já esteja ordenado, isto é, o algoritmo precisa pelo menos uma vez, percorrer o conjunto inteiro com comparações um a um.

O Quick Sort, por outro lado, busca subdividir o conjunto em dois a cada iteração, buscando ordenar subconjuntos menores de forma recursiva. No entanto, a escolha do pivô para esse algoritmo foi feita a partir de seu último elemento, o que acaba sendo ineficaz para casos em que esse elemento é o maior ou o menor do conjunto, pois o subconjunto criado a cada iteração é o maior possível, maximizando o número de iterações.

Note que os comportamentos para máximo e mínimo serão semelhantes nos próximos gráficos. O Insertion sort se destacará em comparação aos outros quando em um conjunto já ordenado, enquanto todos possuem um mal desempenho quando o conjunto é ordenado de forma descendente, pois não só o número de iterações, mas também o de trocas de posição são maximizadas.

À medida que o número de elementos a serem ordenados aumenta, novos fatos podem ser observados. O Bubble Sort, por exemplo, assume uma postura linear em seu comportamento, isto é, em média, o desempenho cai proporcionalmente em relação ao número de elementos no conjunto, e, embora não transpareça no gráfico, o mesmo acontece para o Insertion Sort.

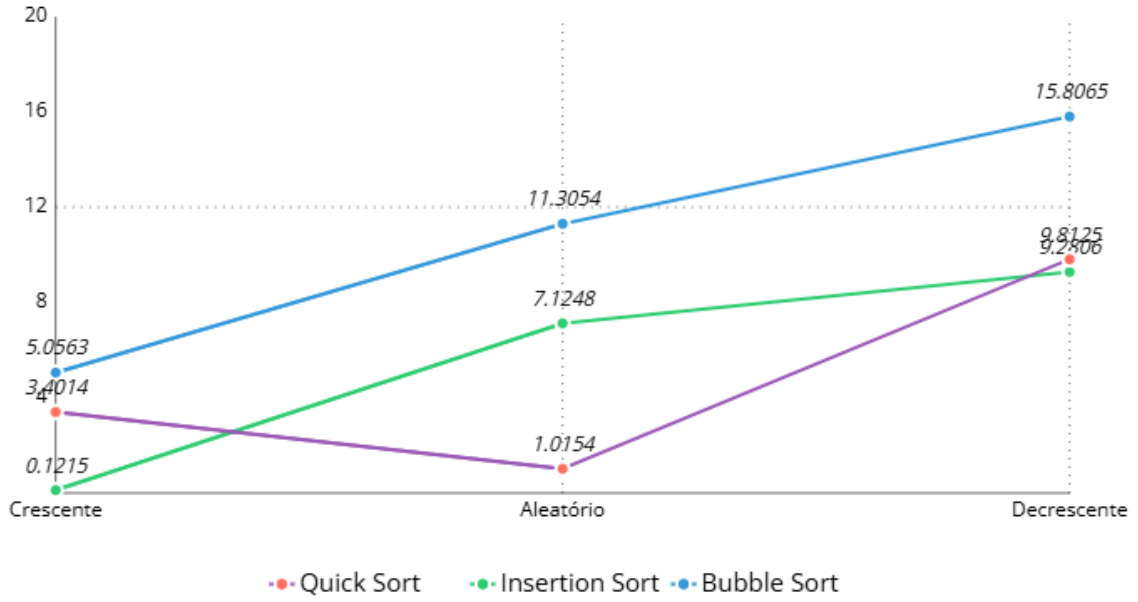

Gráfico comparativo de tempo de execução para os algoritmos em um conjunto de dados de 1000 elementos.

O Quick Sort, em contrapartida, se destaca com grande margem quando os elementos são entregues a ele aleatoriamente, margem essa que pode ser mais bem observada quando o conjunto possui 10000 elementos, como mostrado abaixo. A utilização da abordagem de dividir e conquistar, empregada pelo Quick Sort, é mais eficiente, pois cada subconjunto, assim que estabelecido, é ordenado de forma independente, realizando verificações cada vez menores, diferente dos outros métodos, que percorrem todo o conjunto antes da próxima iteração.

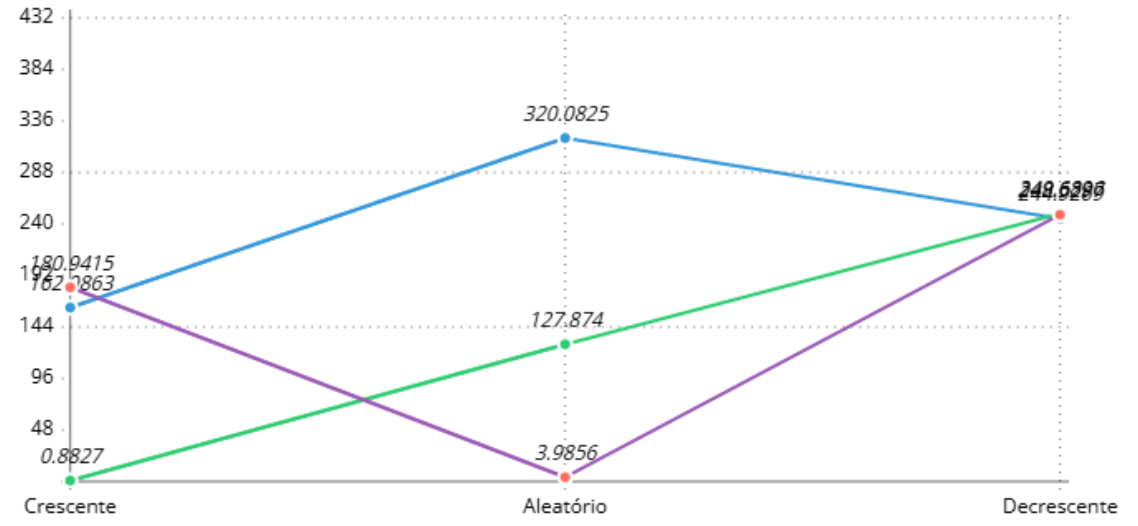

Gráfico comparativo de tempo de execução para os algoritmos em um conjunto de dados de 10000 elementos.

É importante lembrar que os dados dispostos se trata de um único teste para os algoritmos dados os conjuntos de dados. Se, por exemplo, fossem realizados vários testes e tomado sua média para compor as tabelas, as diferenças citadas acima seriam mais acentuadas nos gráficos. Além disso, há de considerar que os

conjuntos não são suficientemente grandes para minimizar as incertezas presentes, como por exemplo, sistema operacional, a capacidade máquina e até mesmo a eficiente do próprio código, que, *certamente*, não é o mais eficaz.

A utilização de um método em detrimento do outro deve ser estudada a fim de identificar qual a necessidade a ser atendida. Uma lista de valores razoavelmente ordenados, por exemplo, o Insertion Sort pode ser uma melhor escolha, entretanto, se levarmos em conta um caso médio, onde os elementos da lista se encontram dispostos de forma aleatória, conclui-se que o Quick Sort é o método a ser escolhido.