Relatório de Análise de Algoritmos de Ordenação

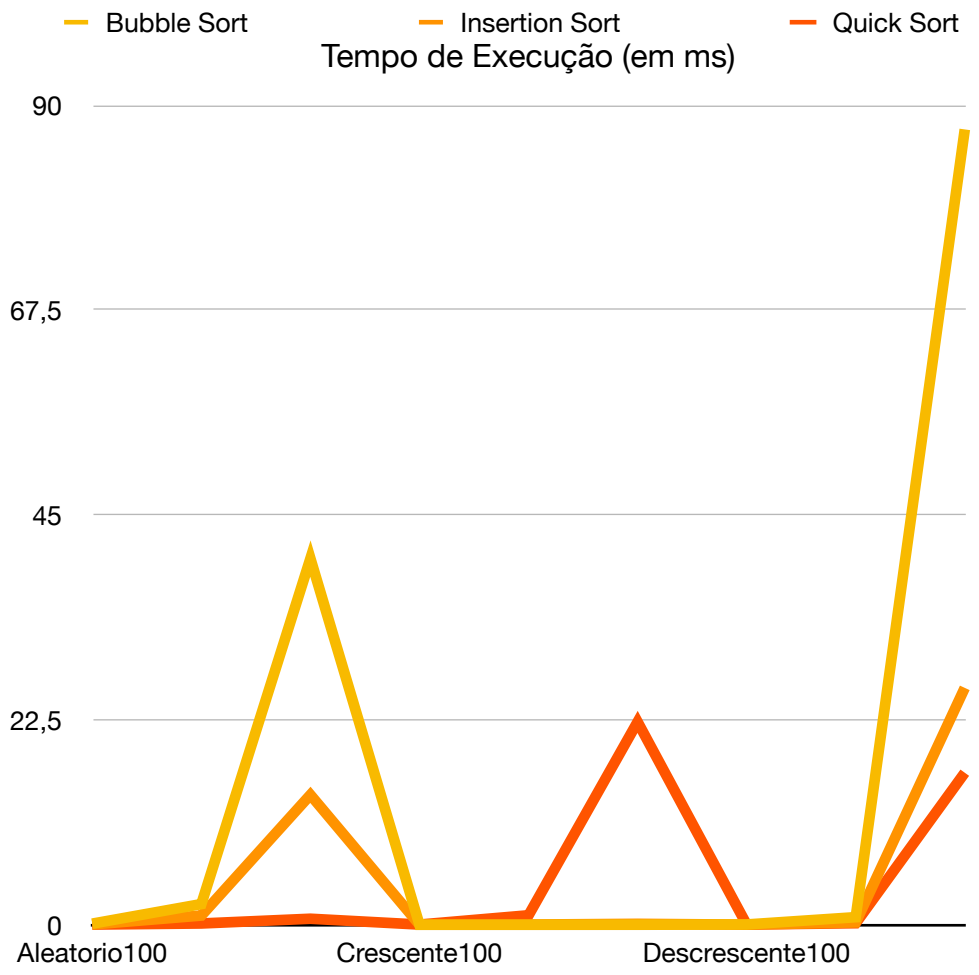
Laura Guillarducci e João Davi

1. Introdução

Este relatório traz uma visão geral de como os algoritmos **Bubble Sort**, **Insertion Sort** e **Quick Sort** se comportam ao ordenar diferentes tipos de listas: uma totalmente aleatória, uma já em ordem crescente e outra em ordem decrescente.
Foram feitos testes com listas pequenas, médias e grandes (100, 1.000 e 10.000 números) para ver como o desempenho muda conforme a quantidade de dados aumenta.

1.1 Resultados de Tempo de Execução (em ns)

Conjunto de dados	Bubble Sort	Insertion Sort	Quick Sort
Aleatório 100	120.042	29.250	15.625
Aleatório 1000	2.270.875	1.008.958	144.333
Aleatório 10000	40.218.792	14.271.709	663.834
Crescente 100	791	9.291	16.625
Crescente 1000	417	16.583	1.043.667
Crescente 10000	4.708	95.833	22.284.542
Decrescente 100	29.291	11.709	21.375
Decrescente 1000	831.208	260.500	171.292
Decrescente 1000	87.358.708	26.019.917	16.703.959



2. Análise dos Resultados

2.1 Dados Aleatórios

Algoritmo mais eficiente: **Quick Sort**
Análise:

Para listas aleatórias, o **Quick Sort** apresentou o melhor desempenho em todos os tamanhos, chegando a ser **60 vezes mais rápido** que o Bubble Sort no conjunto de 10.000 elementos.

O **Insertion Sort** obteve desempenho intermediário, mantendo certa eficiência, mas ainda bem abaixo do Quick Sort em listas maiores.

O **Bubble Sort** teve o pior desempenho, como esperado para esse tipo de entrada.

Motivos:

Em listas aleatórias, o **Quick Sort** aproveita bem sua estratégia de divisão do problema, reduzindo o tamanho das sublistas a cada etapa.

O impacto do tamanho dos dados é muito significativo: quanto maior o conjunto, maior a vantagem do **Quick Sort**.

Bubble Sort e **Insertion Sort** realizam muitas comparações e trocas desnecessárias nesse cenário, o que prejudica sua eficiência.

2.2 Dados em Ordem Crescente

Algoritmo mais eficiente: **Bubble Sort**
Análise:

O **Bubble Sort** foi o algoritmo mais eficiente nesse caso, sendo **4.700 vezes mais rápido** que o Quick Sort no teste com 10.000 elementos.

O **Quick Sort** apresentou o pior desempenho, levando 22,28 ms para ordenar uma lista que já estava ordenada.

O **Insertion Sort** ficou em posição intermediária.

Motivos:

Quando a lista já está ordenada, o **Bubble Sort** percorre todos os elementos uma única vez e identifica rapidamente que não há necessidade de trocas.

Isso faz com que o **Bubble Sort** termine sua execução praticamente de imediato.

Utilizando o último elemento como pivô, uma lista ordenada leva o **Quick Sort** ao pior caso.

2.3 Dados em Ordem Decrescente

Algoritmo mais eficiente: **Quick Sort** (listas maiores) / **Insertion Sort** (lista pequena)
Análise:

Com **100 elementos**, o Insertion Sort foi o mais rápido.

Nos conjuntos de **1.000 e 10.000 elementos**, o Quick Sort obteve o melhor desempenho.

O **Bubble Sort** novamente teve o pior tempo, chegando a 87,36 ms para ordenar 10.000

elementos.

Motivos:

No cenário inverso ao ordenado, o **Bubble Sort** encontra seu pior caso.
São necessárias muitas trocas para mover os elementos até suas posições corretas, o que aumenta consideravelmente o tempo total.

O **Insertion Sort** tipo de entrada, pois cada elemento precisa ser reposicionado passando por vários outros. Apesar disso, em listas pequenas o custo ainda é baixo, justificando seu bom desempenho no conjunto de 100 elementos.

Mesmo com a lista inicialmente invertida, a divisão das sublistas ainda ocorre de forma razoável para o **Quick Sort**.
Mesmo q não seja a divisão ideal, é suficiente para o algoritmo ser eficiente em volumes maiores.
Por isso, o **Quick Sort** continua se destacando nesse cenário.

3. Conclusão

Impacto do tamanho dos dados

- **Listas pequenas (100 elementos):** as diferenças entre os algoritmos são menores, o **Insertion Sort** é boa opção.
- **Listas médias (1.000 elementos):** o **Quick Sort** já demonstra vantagem, especialmente em dados aleatórios e decrescentes.
- **Listas grandes (10.000 elementos):** o **Quick Sort** é o mais eficiente na maioria dos cenários, exceto quando os dados já estão ordenados, caso em que o **Bubble Sort** se torna o mais rápido.

Impacto da ordem inicial

- **Listas aleatórias: Quick Sort** é, de forma consistente, a melhor opção.
- **Listas já ordenadas: Bubble Sort** se torna surpreendentemente o mais eficiente.
- **Listas em ordem inversa: Quick Sort** se mantém superior em conjuntos grandes, enquanto o **Insertion Sort** se destaca em listas menores.

"ChatGPT 5.1 foi utilizado apenas para auxiliar em conversões numéricas e no cálculo de quantas vezes um algoritmo foi mais rápido que outro (Por exemplo na frase:"…chegando a ser **60 vezes mais rápido** que o Bubble Sort no conjunto de 10.000 elementos. …") . Todo o material foi posteriormente revisado pelos autores, que se responsabilizam totalmente pelo conteúdo apresentado."