Amanda Fila de Lima e Carlos Leonardo Garcia Pscheidt

**Análise sobre algoritmos de ordenação**

Link do repositório: https://github.com/Carloslgp/Sorts

Este relatório apresenta os resultados dos testes de desempenho que realizamos em três algoritmos de ordenação: Insertion Sort, Bubble Sort e Quick Sort.

Os testes foram feitos com listas de números aleatórios, crescentes (já ordenadas) e decrescentes, em tamanhos de 100, 1.000 e 10.000 números.

---

## Resultados dos Testes (Tempo em ms)

A tabela abaixo resume os tempos que cada algoritmo levou:

| Algoritmo | Tipo de Dados | 100 Números (ms) | 1.000 Números (ms) | 10.000 Números (ms) |
|---|---|---|---|---|
| **Insertion Sort** | Aleatório | 0.8829 | 14.911 | 96.4393 |
| **Insertion Sort** | Crescente | 1.0996 | 1.2915 | 2.8417 |
| **Insertion Sort** | Decrescente | 2.0285 | 10.1485 | 154.5954 |
| **Bubble Sort** | Aleatório | 0.6693 | 18.5598 | 272.4899 |
| **Bubble Sort** | Crescente | 0.0692 | 0.1262 | 0.933 |
| **Bubble Sort** | Decrescente | 0.671 | 18.5624 | 176.2755 |
| **Quick Sort** | Aleatório | 0.4248 | 2.413 | 6.2286 |

| | | | | |
|---|---|---|---|---|
| **Quick Sort** | Crescente | 1.0606 | 9.0013 | 129.9381 |
| **Quick Sort** | Decrescente | 1.3813 | 10.9193 | 138.0287 |

---

## Observações

Analisando os resultados, notamos alguns pontos importantes:

1. **Quick Sort em Dados Aleatórios:** O Quick Sort foi o mais rápido de longe para ordenar a lista de 10.000 números aleatórios, levando apenas 6.2ms. Os outros levaram 96ms e 272ms.
2. **Bubble Sort em Dados Crescentes:** O Bubble Sort foi o mais rápido de todos na lista crescente (0.933ms para 10.000 números). Isso aconteceu por causa da otimização no código (BubbleSort/Sort.java): ele usa uma variável trocou. Na primeira verificação, ele percebeu que a lista já estava ordenada, não fez nenhuma troca e parou de executar.
3. **Insertion Sort em Dados Crescentes:** O Insertion Sort também foi muito rápido na lista crescente (2.8ms), mostrando que ele é eficiente quando a lista já está quase ou totalmente ordenada.
4. **O Problema do Quick Sort:** O Quick Sort teve um desempenho muito ruim nas listas crescente (129.9ms) e decrescente (138.0ms). Descobrimos que isso acontece por causa da implementação no arquivo InsertionSort/SortingAlgorithm.java, que é chamada pelo MyArrayList.java. O código escolhe o último elemento como pivô. Em uma lista já ordenada, essa é a pior escolha possível e faz o algoritmo demorar muito mais.
5. **Piores Casos:** O Bubble Sort foi o mais lento em dados aleatórios (272.4ms), e o Insertion Sort foi muito lento em dados decrescentes (154.5ms), pois precisou mover cada elemento para o início da lista.

## Conclusão

Os testes mostram que o Quick Sort é a melhor escolha para dados "normais" (aleatórios), mas a forma como ele foi implementado (escolhendo o último pivô) o torna muito lento para listas que já estão ordenadas.

Para listas que podem já estar ordenadas, o Insertion Sort e o Bubble Sort (com a otimização de parada) são, na verdade, muito mais rápidos.