Análise comparativa de algoritimos de ordenação[[1]](#footnote-2)\*

Guilherme W. Back1, Guilherme Pedro Milani2, João Henrique Stolf Galeazzi3

1. Introdução

Este trabalho se propõe a realizar uma **análise experimental e comparativa** de três algoritmos de ordenação amplamente utilizados: **Bubble Sort**, **Quick Sort** e **Merge Sort**. A pesquisa é motivada pela necessidade de ir além da análise teórica de complexidade (Big O), investigando o **desempenho real** dessas implementações em um ambiente de teste controlado. Para isso, os três algoritmos foram implementados e submetidos a testes utilizando a mesma entrada de dados e executados múltiplas vezes para garantir a minimização de variações e a obtenção de médias de tempo confiáveis. O relatório subsequente detalhará a metodologia de avaliação, apresentará os resultados experimentais obtidos.

1. Metodologia

A análise experimental dos algoritmos Bubble Sort, Quick Sort e Merge Sort foi conduzida seguindo um procedimento rigoroso para garantir a validade e a comparabilidade dos resultados de desempenho.

### Implementação e Ambiente de Teste

A implementação dos três algoritmos de ordenação foi realizada na linguagem **Java**, utilizando o **Java Development Kit (JDK).**

O código-fonte para a avaliação experimental está contido na classe principal OrdenacaoExperimental, onde as implementações dos algoritmos (BubbleSort, MergeSort, e QuickSort) são chamadas. O tempo de execução de cada algoritmo foi medido usando a função de alta resolução System.nanoTime(), que fornece o tempo atual do sistema em nanossegundos, ideal para medir durações de código curtas e críticas

### Estratégia de Avaliação Experimental

A avaliação de desempenho baseou-se nos seguintes parâmetros e procedimentos:

1. **Métrica de Desempenho:** O critério principal de avaliação foi o **tempo de execução** necessário para que cada algoritmo completasse a ordenação do *array* de entrada. Este tempo foi acumulado e, posteriormente, convertido em **milissegundos (ms)** para facilitar a leitura.
2. **Entrada de Dados Única:** Para garantir uma comparação justa, todos os algoritmos foram testados com o **mesmo conjunto de dados aleatórios** em cada iteração do experimento.
3. **Tamanho da Entrada (TAMANHO\_ENTRADA):** O tamanho do *array* de entrada foi fixado em **10.000 elementos**.
4. **Geração de Dados:** O método gerarArrayAleatorio foi responsável por criar um *array* de inteiros (tipo Integer[]) de tamanho fixo, preenchido com valores aleatórios não negativos.
5. **Execuções Repetidas (NUM\_TESTES):** Cada algoritmo foi executado **10 vezes** com uma nova cópia do mesmo *array* aleatório gerado para aquela iteração.
6. **Cálculo da Média:** O tempo total acumulado em nanossegundos ao longo das 10 execuções foi dividido pelo número total de testes para obter o **tempo médio de execução** (em milissegundos).
7. Resultados

Os experimentos de avaliação de desempenho foram realizados com sucesso, submetendo os três algoritmos de ordenação a testes repetidos utilizando a mesma entrada de dados aleatórios. A tabela a seguir consolida o tempo médio de execução obtido em duas rodadas de testes, demonstrando a estabilidade dos resultados em milissegundos (ms).

| Algoritmo | Média dos Testes (Rodada 1) | Média dos Testes (Rodada 2) | Média dos Testes (Rodada 3) |
| --- | --- | --- | --- |
| **Bubble Sort** | **150,432 ms** | **145,108 ms** | **145,273 ms** |
| **Merge Sort** | **4,922 ms** | **4,011 ms** | **4,693 ms** |
| **Quick Sort** | **0,881 ms** | **0,888 ms** | **0,854 ms** |

### **Detalhes dos Experimentos**

| Parâmetro | Valor |
| --- | --- |
| **Tamanho da Entrada** | **10.000 elementos** |
| **Número de Testes (por algoritmo)** | **10 repetições** |
| **Métrica de Medição** | Tempo médio de execução (nanossegundos convertido para milissegundos) |

1. Discussão

A análise experimental demonstrou uma **enorme disparidade** no desempenho entre os três algoritmos de ordenação, confirmando as previsões da teoria de complexidade de algoritmos. Os algoritmos **não tiveram o mesmo desempenho**; na verdade, a diferença de tempo de execução entre o algoritmo mais rápido e o mais lento foi de mais de 170 vezes.

### Comparação do Desempenho Algorítmico

Os resultados podem ser categorizados em duas classes de eficiência, baseadas na complexidade de tempo:

| Algoritmo | Tempo Médio Aproximado | Fator de Desempenho (vs. Quick Sort) |
| --- | --- | --- |
| **Bubble Sort** | 147,77 ms | ≈167 vezes mais lento |
| **Merge Sort** | 4,47 ms | ≈5 vezes mais lento |
| **Quick Sort** | 0,88 ms | Base (mais rápido) |

### Conclusão da Discussão: O Melhor Algoritmo

Com base nos resultados experimentais obtidos com *arrays* de 10.000 elementos preenchidos com dados aleatórios, o **Quick Sort é o melhor algoritmo** em termos de velocidade de execução.

Seu desempenho consistentemente rápido (em média 0,88 ms) o estabelece como a escolha mais eficiente para a ordenação de dados nesta escala e tipo de distribuição, superando tanto o algoritmo Bubble Sort quanto o outro algoritmo Merge Sort.

1. Conclusão

Em suma, a análise experimental dos algoritmos de ordenação Bubble Sort, Quick Sort e Merge Sort, utilizando 10.000 elementos de entrada, comprovou a intrínseca relação entre a complexidade teórica e o desempenho prático. Foi notória a ineficiência do **Bubble Sort**, que se mostrou impraticável em comparação com os outros métodos. O **Quick Sort** emergiu como o algoritmo mais rápido e eficiente na prática (cerca de 0,88 ms), superando o **Merge Sort** (aproximadamente 4,47 ms) devido à sua menor sobrecarga e melhor aproveitamento da memória.

Conclui-se que, para a ordenação de um volume moderado de dados aleatórios, a implementação do Quick Sort oferece o melhor compromisso entre velocidade e eficiência, sendo a escolha superior para otimizar o tempo de processamento.

1. \*\*\* [↑](#footnote-ref-2)