Análise Experimental de Algoritmos de Ordenação: Bubble Sort, Quick Sort e Merge Sort

Arthur Henrique Damann, Lucas Gabriel Devigili

Departamento de Sistemas e Computação

Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brazil

adamann@furb.br, ldevigili@furb.br

1. Introdução

Algoritmos de ordenação são componentes fundamentais na Computação, aplicados em áreas como bancos de dados, inteligência artificial, estruturas de dados e sistemas operacionais. A eficiência de um algoritmo de ordenação impacta diretamente o desempenho geral de aplicações que manipulam grandes volumes de dados.

Neste trabalho, foram analisados três algoritmos clássicos: Bubble Sort, Quick Sort e Merge Sort. O objetivo é comparar seu desempenho tanto de forma assintótica — considerando o comportamento teórico de complexidade — quanto experimentalmente, por meio de medições práticas de tempo de execução em um ambiente controlado.

A combinação dessas duas abordagens permite observar se o comportamento real dos algoritmos confirma suas expectativas teóricas e compreender possíveis divergências entre teoria e prática.

2. Metodologia

A implementação dos algoritmos foi desenvolvida na **linguagem Java**, seguindo as versões clássicas encontradas na literatura. A medição de desempenho foi feita considerando **apenas o tempo de execução do processo de ordenação**, excluindo operações de entrada e saída de dados.

Configuração do ambiente de testes:

- **Sistema Operacional:** Windows 11 Home
- **Processador:** AMD Ryzen 5 5600X (6 núcleos, 12 threads)
- **Memória RAM:** 16 GB DDR4
- **Linguagem:** Java 17
- **IDE utilizada:** Visual Studio Code

Entradas de teste:

- Vetores com **100.000 elementos inteiros** gerados de forma aleatória.
- Cada algoritmo foi executado **10 vezes** com diferentes conjuntos de dados.
- A métrica utilizada foi o **tempo médio de execução (em segundos)**.

Os experimentos foram conduzidos com o objetivo de comparar a eficiência prática dos algoritmos. Todas as execuções foram realizadas no mesmo computador, sem outros processos pesados em segundo plano, buscando reduzir interferências externas.

O código-fonte e os dados dos experimentos estão disponíveis publicamente em:
https://github.com/Tutu-69/Analise-de-Algoritmos-2025/tree/main/Trabalho2-p1

3. Resultados

A Tabela 1 apresenta os tempos simulados (em segundos) para os dez testes realizados com cada algoritmo.

Tabela 1. Tempos de execução (em segundos)

Teste	Bubble Sort	Quick Sort	Merge Sort
1	7.843	0.013	0.015
2	7.923	0.017	0.016
3	7.842	0.011	0.015
4	7.701	0.013	0.014
5	7.806	0.012	0.015
6	7.812	0.012	0.015
7	7.974	0.012	0.014
8	7.789	0.013	0.014
9	7.793	0.012	0.016
10	7.826	0.014	0.014

A média dos tempos foi:

- BubbleSort: 7.821s
- Quicksort: 0.0129s
- MergeSort: 0.0148s

4. **Análise Assintótica**

Nesta seção são apresentados os comportamentos teóricos de complexidade de cada algoritmo. A análise considera que cada instrução básica consome uma unidade de tempo.

3.1. Bubble Sort

O **Bubble Sort** é um algoritmo simples que percorre repetidamente o vetor, comparando elementos adjacentes e trocando-os de posição se estiverem fora de ordem.

Passos:

1. Comparar elementos adjacentes.
2. Trocar se estiverem na ordem incorreta.
3. Repetir o processo até que não ocorram mais trocas.

Melhor caso:

- O vetor já está ordenado.
- Em uma passagem, nenhuma troca ocorre.
- Complexidade: **O(n)** (ainda há comparações, mas apenas uma varredura é necessária).

Pior caso:

- O vetor está totalmente invertido.
- Cada elemento precisa ser comparado e trocado em todas as passagens.
- Complexidade: **$O(n^2)$**

Casos típicos:

- Comportamento quadrático em praticamente todas as entradas médias.
- Pouco eficiente para grandes conjuntos de dados.

3.2. Quick Sort

O **Quick Sort** é um algoritmo recursivo baseado na estratégia **dividir para conquistar**. Ele seleciona um **pivô**, particiona o vetor em dois subvetores (menores e maiores que o pivô) e aplica a ordenação recursivamente.

Melhor caso:

- O pivô divide o vetor em duas partes de tamanhos semelhantes a cada iteração.
- Complexidade: **O(n log n)**

Pior caso:

- O pivô sempre é o menor ou maior elemento (ex: vetor já ordenado).
- Ocorre degeneração e o algoritmo se comporta como o Bubble Sort.
- Complexidade: **$O(n^2)$**

Caso médio:

- Pivôs aleatórios geralmente produzem divisões equilibradas.
- Complexidade esperada: **O(n log n)**

3.3. Merge Sort

O **Merge Sort** também segue o paradigma **dividir para conquistar**, mas é estável e não depende da escolha de pivô. Ele divide o vetor em metades, ordena cada metade e depois **intercala (merge)** os resultados.

Melhor, médio e pior caso:

- O número de divisões e intercalamentos é sempre o mesmo.

- O comportamento é previsível e estável.
- Complexidade em todos os casos: **O(n log n)**

Observação:

- Embora eficiente, requer **espaço adicional O(n)** para o vetor auxiliar usado na intercalação.

5. Discussão

Os resultados experimentais confirmam o comportamento teórico esperado.

O Bubble Sort, devido à sua complexidade quadrática ($O(n^2)$), apresentou tempos de execução muito superiores aos demais, tornando-se impraticável para grandes volumes de dados. Essa diferença reforça sua utilização apenas como ferramenta didática para introdução de algoritmos de ordenação.

Tanto o Quick Sort quanto o Merge Sort obtiveram tempos próximos e compatíveis com a complexidade O(n log n). No entanto, observou-se uma pequena vantagem do Quick Sort, justificada por seu melhor aproveitamento da memória cache, já que trabalha com partições locais do vetor.

Por outro lado, o Merge Sort mostrou maior estabilidade nos tempos de execução, pois seu desempenho não depende da natureza dos dados de entrada, sendo ideal em contextos que exigem previsibilidade.

A diferença prática entre os dois algoritmos é, portanto, resultado do equilíbrio entre custos de memória (Merge Sort) e otimização de acesso local (Quick Sort).

6. Conclusão

O estudo permitiu observar, na prática, a correspondência entre o comportamento teórico e o desempenho real dos algoritmos de ordenação.

O Bubble Sort mostrou-se inviável para grandes volumes de dados, confirmando seu caráter educacional.

Já o Quick Sort e o Merge Sort apresentaram excelente desempenho, com tempos na ordem de milissegundos, demonstrando a eficiência dos algoritmos com complexidade O(n log n).

O Quick Sort apresentou desempenho ligeiramente superior nas medições, tornando-se a melhor opção na maioria dos cenários testados.

Em contrapartida, o Merge Sort é preferível quando a estabilidade da ordenação é um requisito importante. Assim, os resultados confirmam que o desempenho prático dos algoritmos acompanha suas análises assintóticas, com variações explicáveis pela natureza das implementações e pela arquitetura de hardware.