Comparação de Algoritmos de Ordenação

Ian Patrick da Costa Soares
CES-11: Algoritmos e Estruturas de Dados

22 de setembro de 2024

Resumo

Este relatório apresenta uma análise comparativa entre diferentes algoritmos de ordenação: BubbleSort, QuickSort, MergeSort com vetor temporário local e MergeSort com vetor temporário local e estático. Os algoritmos foram testados com entradas geradas aleatoriamente para determinar o tamanho máximo que cada um consegue ordenar em até 2 segundos. Além disso, foram realizadas análises detalhadas com múltiplas entradas para comparar o número de comparações e o tempo de execução.

Sumário

1 Introdução

A ordenação de dados é uma operação fundamental em Ciência da Computação, utilizada em diversas aplicações que vão desde a organização de informações em bancos de dados até a otimização de algoritmos de busca. Este relatório tem como objetivo comparar o desempenho de diferentes algoritmos de ordenação em termos de tempo de execução e número de comparações realizadas.

2 Metodologia

Para realizar a comparação, foram implementados e testados os seguintes algoritmos de ordenação:

- **BubbleSort**
- **QuickSort**
- **MergeSort com vetor temporário local**
- **MergeSort com vetor temporário local e estático**

Os testes foram conduzidos em um ambiente controlado, onde diversas entradas geradas aleatoriamente foram utilizadas para determinar o tamanho máximo que cada algoritmo consegue ordenar em até 2 segundos. Posteriormente, foram realizadas análises com múltiplas entradas para comparar o desempenho dos algoritmos.

2.1 Ambiente de Teste

- **Processador**: 11th Gen Intel(R) Core(TM) i9-11900H @ 2.50GHz 2.50 GHz
- **Memória RAM**: 16.0 GB (utilizável: 15.7 GB)
- **Sistema Operacional**: Windows 11 Home Single Language
- **Compilador**: Visual Studio Code 1.93

2.2 Procedimento de Teste

1. **Determinação do Tamanho Máximo em 2 Segundos**: Cada algoritmo foi executado com entradas de tamanhos crescentes até que o tempo de execução ultrapassasse 2 segundos. O maior tamanho que ainda respeitou o limite de tempo foi registrado.
2. **Testes Detalhados**: Após determinar os tamanhos máximos, foram realizadas execuções com múltiplas entradas para cada algoritmo, coletando dados de tamanho da entrada, número de comparações e tempo de execução.

3 Resultados

3.1 Tamanho Máximo em 2 Segundos

A tabela a seguir apresenta o tamanho máximo de entrada que cada algoritmo conseguiu ordenar em até 2 segundos.

Tabela 1: Tamanho Máximo de Entrada em 2 Segundos

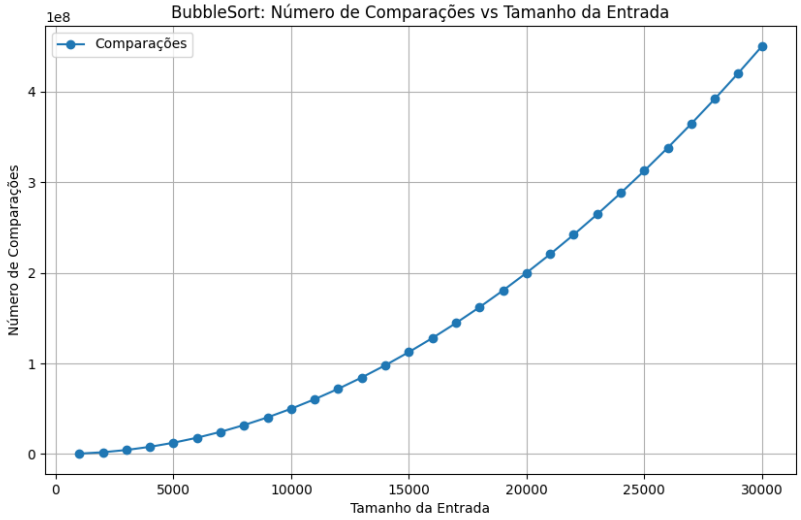
Algoritmo	Tamanho Máximo	Número de Comparações	Tempo de Execução (s)
Bubble	1.50e+04	1.12e+08	1.889
Quick	3.50e+06	9.51e+07	1.994
Merge	3.20e+06	6.52e+07	1.942
Merge (static)	3.10e+06	6.30e+07	1.995

3.2 Comparação Detalhada dos Algoritmos

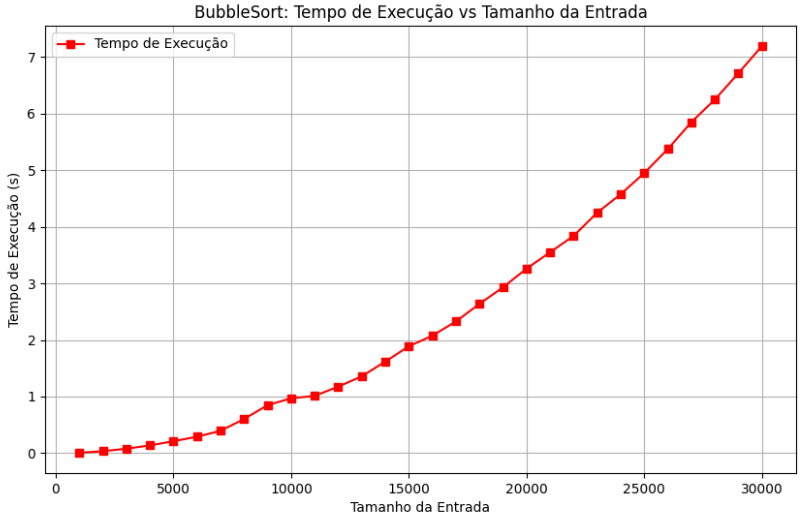
3.2.1 BubbleSort

Tabela 2: Desempenho do BubbleSort

Tamanho da Entrada	Comparações	Tempo (s)
1.00e+03	4.99e+05	0.008
1.00e+04	4.99e+07	0.969
2.00e+04	1.99e+08	3.260
3.00e+04	4.50e+08	7.191



(a) Número de Comparações



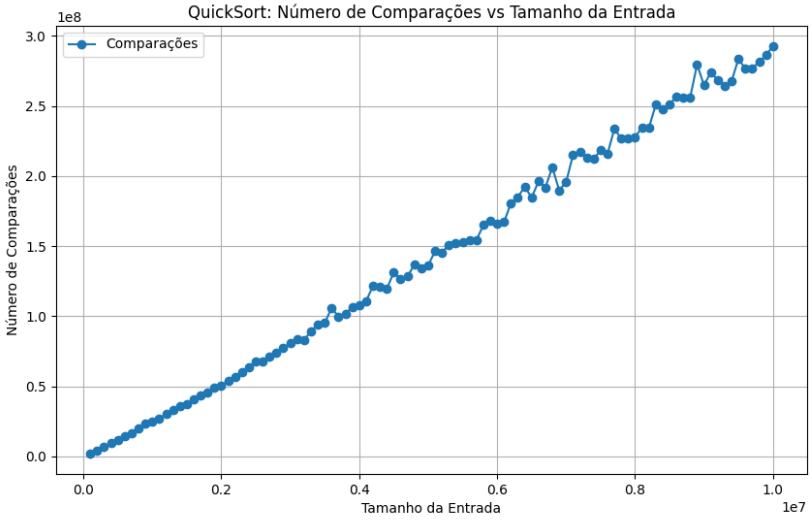
(b) Tempo de Execução

Figura 1: Desempenho do BubbleSort

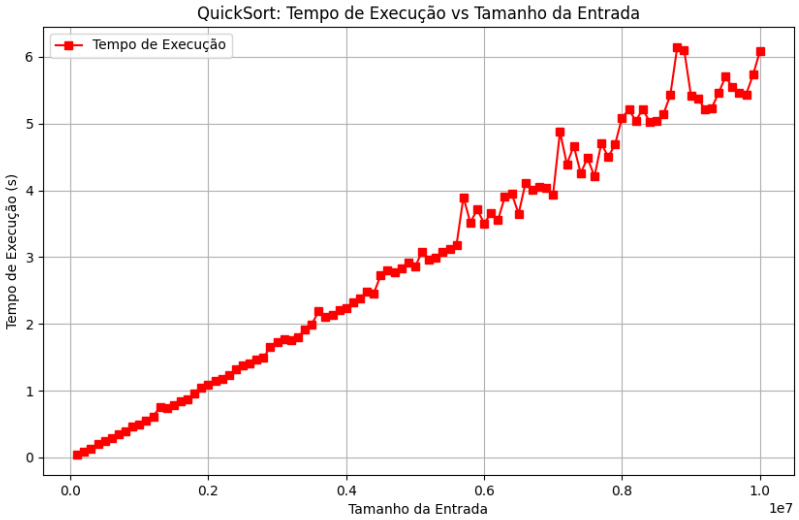
3.2.2 QuickSort

Tabela 3: Desempenho do QuickSort

Tamanho da Entrada	Comparações	Tempo (s)
1.00e+05	1.97e+06	0.041
1.00e+06	2.49e+07	0.495
2.50e+06	6.76e+07	1.372
5.00e+06	1.36e+08	2.855
1.00e+07	2.93e+08	6.086



(a) Número de Comparações



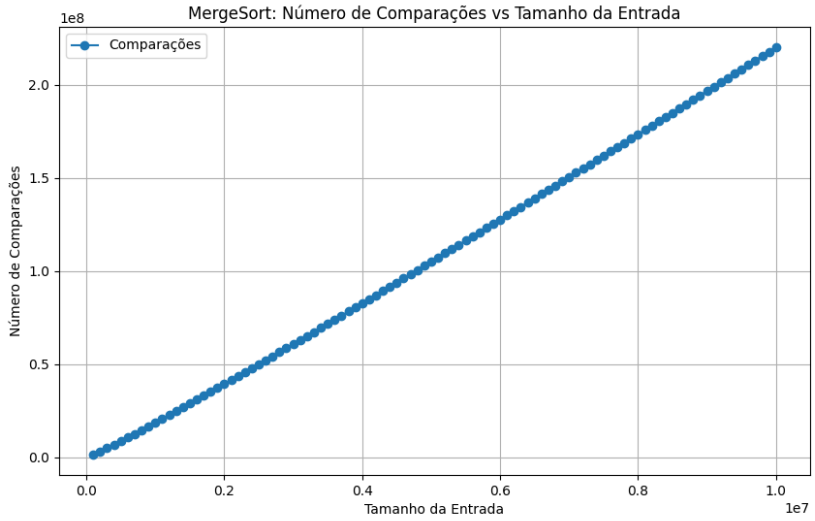
(b) Tempo de Execução

Figura 2: Desempenho do QuickSort

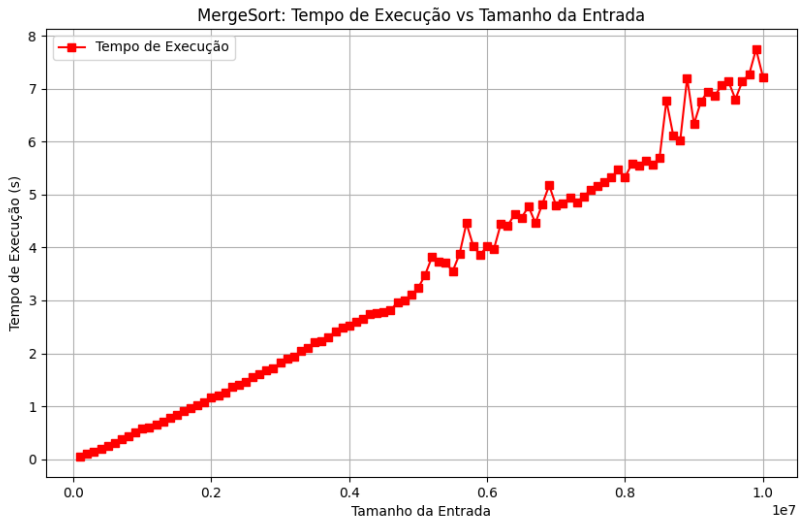
3.2.3 MergeSort com Vetor Temporário Local

Tabela 4: Desempenho do MergeSort (Vetor Temporário Local)

Tamanho da Entrada	Comparações	Tempo (s)
1.00e+05	1.54e+06	0.049
1.00e+06	1.87e+07	0.576
2.50e+06	5.00e+07	1.467
5.00e+06	1.05e+08	3.233
1.00e+07	2.20e+08	7.220



(a) Número de Comparações



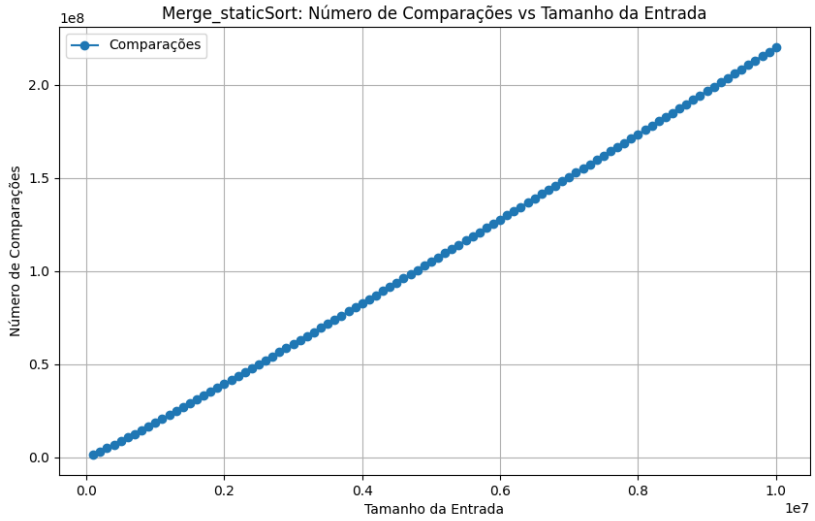
(b) Tempo de Execução

Figura 3: Desempenho do MergeSort com Vetor Temporário Local

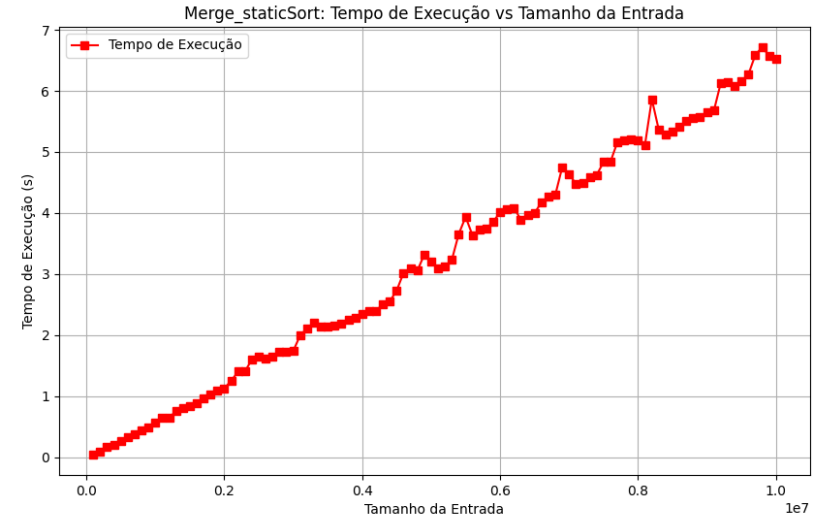
3.2.4 MergeSort com Vetor Temporário Local e Estático

Tabela 5: Desempenho do MergeSort (Vetor Temporário Local e Estático)

Tamanho da Entrada	Comparações	Tempo (s)
1.00e+05	1.54e+06	0.042
1.00e+06	1.87e+07	0.563
2.50e+06	5.00e+07	1.649
5.00e+06	1.05e+08	3.205
1.00e+07	2.20e+08	6.531



(a) Número de Comparações



(b) Tempo de Execução

Figura 4: Desempenho do MergeSort com Vetor Temporário Local e Estático

4 Análise dos Dados

Os resultados obtidos estão em conformidade com as expectativas teóricas dos algoritmos analisados. O **BubbleSort**, com complexidade temporal de $O(n^2)$, apresentou um crescimento quadrático no número de comparações e no tempo de execução conforme o tamanho da entrada aumentava. Já os algoritmos **QuickSort** e **MergeSort**, ambos com complexidade média de $O(n \cdot \log(n))$, demonstraram um crescimento logarítmico mais eficiente, possibilitando o processamento de entradas maiores em tempos significativamente menores.

4.1 Relação entre Tempo e Número de Comparações

A relação entre o tempo gasto e o número de comparações realizadas foi diretamente proporcional para todos os algoritmos. No caso do **BubbleSort**, o aumento exponencial nas comparações resultou em um aumento acentuado no tempo de execução. Por outro lado, **QuickSort** e **MergeSort** mantiveram uma relação mais balanceada, refletindo sua eficiência superior em ordenar grandes conjuntos de dados.

4.2 MergeSort com Vetor Temporário Local vs. Estático

A comparação entre as duas variantes do **MergeSort** revelou que a utilização de um vetor temporário estático não impactou significativamente o número de comparações ou o tempo de execução. Contudo, a versão estática permitiu a ordenação de entradas maiores sem risco de estouro de pilha, oferecendo maior flexibilidade em contextos onde grandes conjuntos de dados são comuns.

4.3 Outras Observações

Além das análises mencionadas, foi observado que o desempenho dos algoritmos pode variar dependendo da distribuição dos dados de entrada. Por exemplo, **QuickSort** pode apresentar desempenho pior em casos de entradas já ordenadas ou quase ordenadas, a menos que seja implementado com estratégias de pivô adequadas.

Para gerar as entradas usadas, foi utilizado o gerador de sequências de DNA disponibilizado no google classroom da disiciplina.

5 Conclusão

Este estudo comparativo demonstrou claramente as diferenças de desempenho entre os algoritmos de ordenação analisados. Enquanto **BubbleSort** é adequado apenas para pequenos conjuntos de dados devido à sua complexidade $O(n^2)$, **QuickSort** e **MergeSort** se destacaram pela eficiência em lidar com grandes volumes de dados graças à sua complexidade $O(n \cdot \log(n))$. A variação do **MergeSort** com vetor temporário estático mostrou-se uma alternativa viável para evitar estouro de pilha em aplicações que requerem a ordenação de conjuntos de dados extensos.

6 Referências

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.
- Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley.
- Wikipedia contributors. (2024). *Merge sort*. Recuperado de `https://pt.wikipedia.org/wiki/Merge_sort`