

Comparação de Algoritmos de Ordenação

Em termos de tempo
de execução

Cauã dos Santos
Mateus Ryu Hashimoto
Miguel de Paulo Leite de Oliveira
Pedro Lauria Nassif

Universidade de Taubaté, UNITAU
Engenharia de Computação
Análise de Algoritmos
Professor Eduardo Hidenori Enari

Taubaté, 28 Maio 2025

Conteúdo

Lista de Figuras	2
1 Introdução	3
1.1 Objetivo	3
2 Metodologia	4
2.1 Preparação	4
2.2 Execução	4
2.3 Reconsiderações	4
3 Resultados e Analises	5
3.1 Resultados para DataSet Ordenado	5
3.1.1 Analise para Dados Ordenados	6
3.2 Resultados para DataSet Inversamente Ordenado	7
3.2.1 Analise para Dados Inversamente Ordenados	8
3.3 Resultados para DataSet Aleatoriamente Ordenado	8
3.3.1 Analise para Dados Aleatoriamente Ordenados	9
4 Conclusões	11
Appendices	12
A Fonte	13

Lista de Figuras

3.1	Tabela Dados Ordeanados	5
3.2	Gráfico Performance Dados Ordenados em escala Logaritmica	5
3.3	Performance Bubble, Insertion e Selection em Dados Ordenados	6
3.4	Performance Restante em Dados Ordenados	6
3.5	Tabela Dados Inversamente Ordenados	7
3.6	Gráfico Performance Dados Inversamente Ordenados em escala Logaritmica	7
3.7	Performance Bubble, Insertion e Selection em Dados Inversamente Ordenados	7
3.8	Performance Restante em Dados Inversamente Ordenados	8
3.9	Tabela Dados Aleatoriamente Ordenados	8
3.10	Gráfico Performance Dados Aleatoriamente Ordenados em escala Logaritmica	9
3.11	Performance Bubble, Insertion e Selection em Dados Aleatoriamente Ordenados	9
3.12	Performance Restante em Dados Aleatoriamente Ordenados	9

Capítulo 1

Introdução

Todo engenheiro de computação deve ter conhecimento sobre algoritmos de ordenação, visto que estes estão presentes em diversas aplicações. Este projeto desafia seus participantes a se familiarizarem não só com a implementação de tais algoritmos, mas também com seu funcionamento interno, através da análise feita por este relatório.

1.1 Objetivo

O objetivo deste projeto é comparar o desempenho de diferentes algoritmos de ordenação em termos de tempo, utilizando uma variedade de conjuntos de dados.

Capítulo 2

Metodologia

2.1 Preparação

Para a medição do tempo de execução, primeiro os algoritmos foram implementados em python e depois executados nos seguintes grupos de dados:

- 10000: ordenados, Inversamente ordenados e aleatórios
- 20000: ordenados, Inversamente ordenados e aleatórios
- 40000: ordenados, Inversamente ordenados e aleatórios
- 80000: ordenados, Inversamente ordenados e aleatórios
- 100000: ordenados, Inversamente ordenados e aleatórios

2.2 Execução

Os algoritmos foram executados um a um através do script *make.sh* com cada um dos dados apresentados na seção *apresentação* e os resultados de tempo gasto, comparações e trocas salvos no arquivo *metrics.txt* que então foram analisados através do notebook jupyter para gerar as figuras e tabelas presentes no relatório.

2.3 Reconsiderações

Durante a execução do projeto, foram feitas as seguintes reconsiderações:

- Os algoritmos foram executados apenas uma vez, por conta da demora de execução dos mesmos.
- As quantidades de comparações e trocas foram medidas para alguns algoritmos, visto que outros (como o count sort) não realizam trocas e nem comparações, por conta disso e da restrição de tempo para entrega, a análise dessas variáveis não foi incluída no relatório final
- A complexidade de espaço não foi medida, visto que a implementação do cálculo dessa métrica acarretou no aumento grandioso do já longo tempo de execução dos algoritmos, por isso e pela restrição de tempo, essa métrica não foi nem sequer computada.

Capítulo 3

Resultados e Analises

3.1 Resultados para DataSet Ordenado

Time Performance (seconds) on Ordered Data					
Size	10000	20000	40000	80000	100000
Algorithm					
Bubble	2.446205	10.407193	41.720630	180.158629	282.438182
Count	0.000991	0.002641	0.004656	0.007915	0.011064
Heap	0.029200	0.062837	0.141775	0.295202	0.371031
Insertion	0.000571	0.001152	0.002322	0.004835	0.005824
Merge	0.012367	0.026255	0.054467	0.115752	0.150197
Radix	0.013012	0.027121	0.053647	0.104612	0.160977
Selection	1.991915	9.388031	32.990227	141.962518	221.004655

Figura 3.1: Tabela Dados Ordeanados

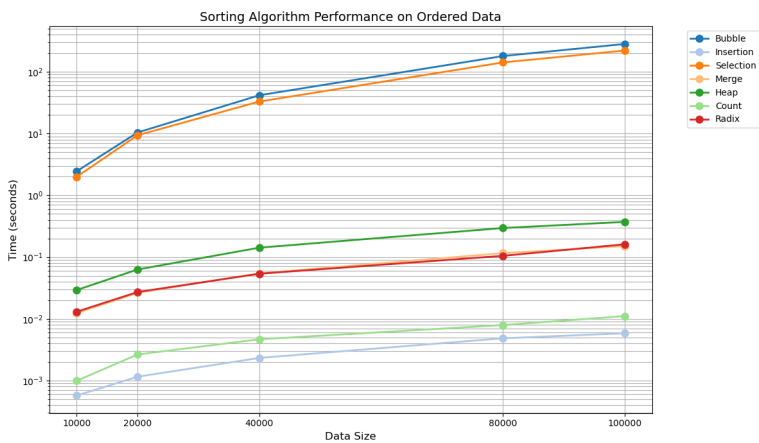


Figura 3.2: Gráfico Performance Dados Ordenados em escala Logaritmica

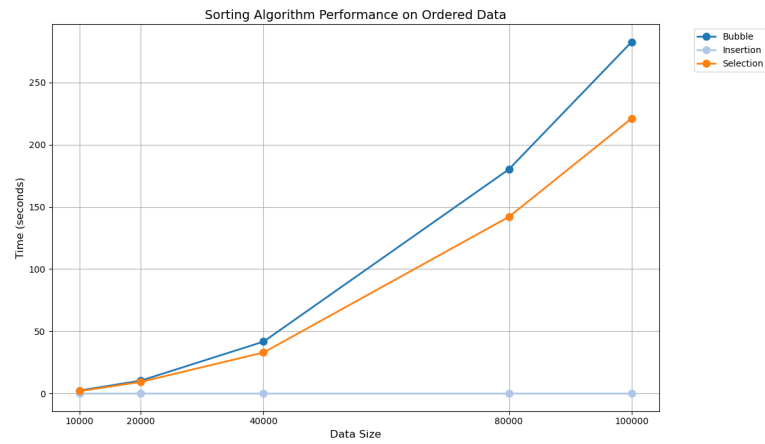


Figura 3.3: Performance Bubble, Insertion e Selection em Dados Ordenados

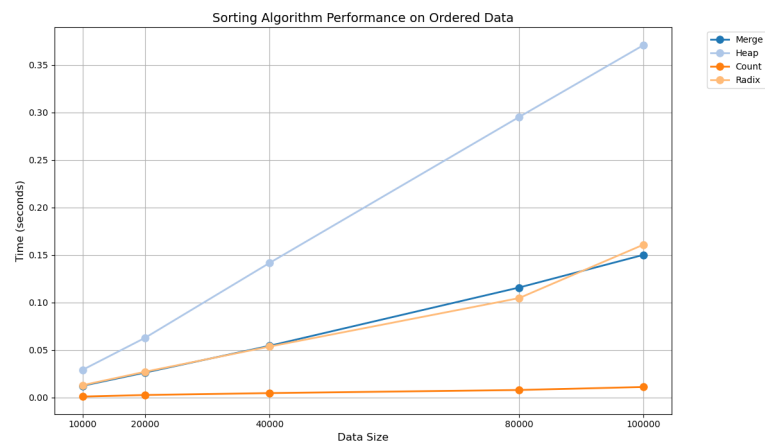


Figura 3.4: Performance Restante em Dados Ordenados

3.1.1 Análise para Dados Ordenados

Para os dados Ordenados:

- Pior Algoritmo: Bubble Sort
- Melhor Algoritmo: Insertion Sort

Pela análise dos gráficos, observa-se os seguintes comportamentos:

- Parabólico: Selection e Bubble. Comportamento esperado
- Logaritmico: Radix e Merge. Esperado para Merge, inesperado para Radix
- Linear: Heap e Insertion. Comportamento Inesperado

3.2 Resultados para DataSet Inversamente Ordenado

Time Performance (seconds) on Reverse Ordered Data

Size	10000	20000	40000	80000	100000
Algorithm					
Bubble	6.243569	25.477908	103.943412	424.523627	672.475248
Count	0.001281	0.002074	0.004293	0.008087	0.009846
Heap	0.031675	0.056419	0.121598	0.259416	0.328837
Insertion	5.290499	22.049225	85.254229	349.443866	546.964891
Merge	0.012248	0.025945	0.055972	0.114078	0.144794
Radix	0.013042	0.026472	0.051483	0.107809	0.162742
Selection	2.074581	8.428733	33.204356	143.188906	227.884410

Figura 3.5: Tabela Dados Inversamente Ordenados

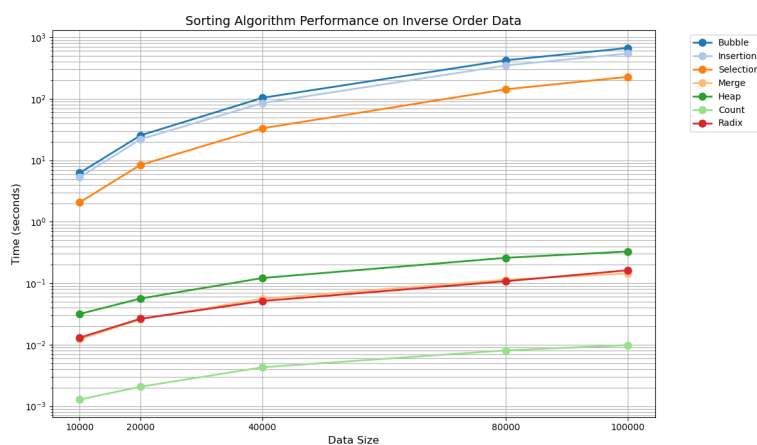


Figura 3.6: Gráfico Performance Dados Inversamente Ordenados em escala Logaritmica

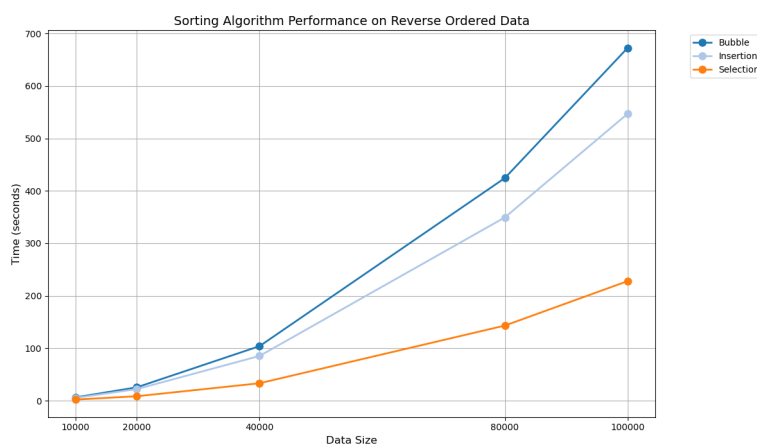


Figura 3.7: Performance Bubble, Insertion e Selection em Dados Inversamente Ordenados

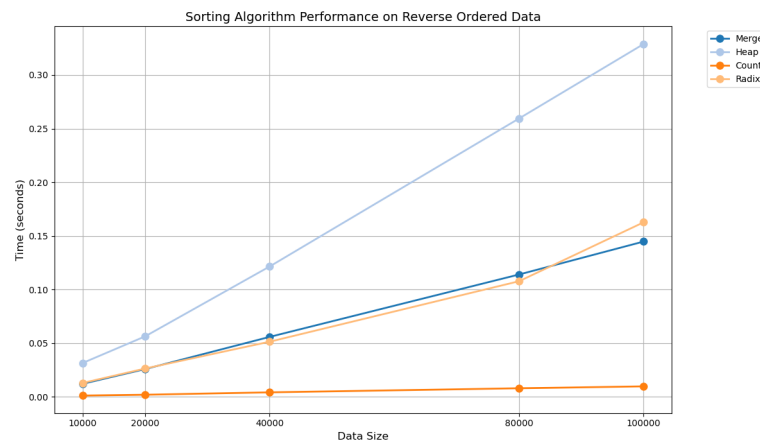


Figura 3.8: Performance Restante em Dados Inversamente Ordenados

3.2.1 Análise para Dados Inversamente Ordenados

Para os dados Inversamente Ordenados:

- Pior Algoritmo: Bubble Sort
- Melhor Algoritmo: Count Sort

Pela análise dos gráficos, observa-se os seguintes comportamentos:

- Parabólico: Selection, Bubble e Insertion. Comportamento esperado
- Logarítmico: Radix, Merge e Heap. Inesperado para Radix
- Linear: Count. Comportamento Inesperado

3.3 Resultados para DataSet Aleatoriamente Ordenado

Time Performance (seconds) on Ramdon Data					
Size	10000	20000	40000	80000	100000
Algorithm					
Bubble	4.594275	18.459088	74.987911	313.320278	497.353979
Count	0.000992	0.002244	0.004226	0.008138	0.010917
Heap	0.028111	0.060526	0.132589	0.283970	0.364093
Insertion	2.688185	10.735552	43.996993	178.892856	277.858983
Merge	0.015869	0.032941	0.070579	0.148076	0.188890
Quick	0.009166	0.020654	0.039777	0.084266	0.124811
Radix	0.014270	0.026870	0.058935	0.109538	0.159961
Selection	2.116239	8.360220	33.848114	148.323356	230.244180

Figura 3.9: Tabela Dados Aleatoriamente Ordenados

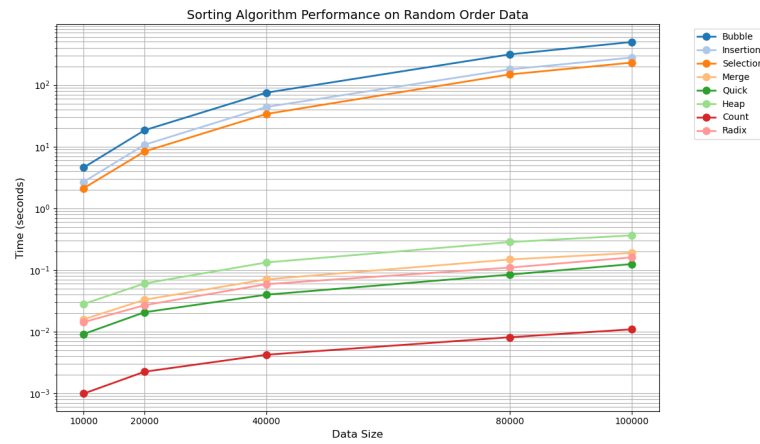


Figura 3.10: Gráfico Performance Dados Aleatoriamente Ordenados em escala Logaritmica

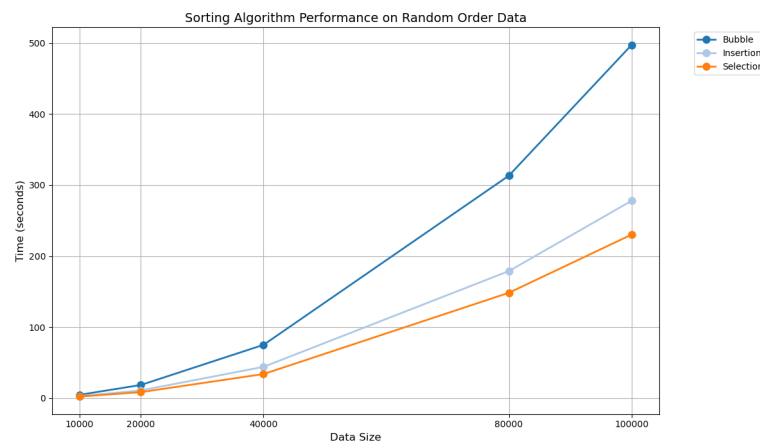


Figura 3.11: Performance Bubble, Insertion e Selection em Dados Aleatoriamente Ordenados



Figura 3.12: Performance Restante em Dados Aleatoriamente Ordenados

3.3.1 Análise para Dados Aleatoriamente Ordenados

Para os dados Inversamente Aleatoriamente Ordenados:

- Pior Algoritmo: Bubble Sort
- Melhor Algoritmo: Count Sort

Pela análise dos gráficos, observa-se os seguintes comportamentos:

- Parabólico: Selection, Bubble e Insertion. Comportamento esperado
- Logaritmico: Radix, Merge e Quick. Esperado apenas para Merge
- Linear: Count e Heap. Comportamento Inesperado para Heap

Capítulo 4

Conclusões

Um dos comportamentos inesperados para o Radix muito provavelmente vem da implementação do mesmo, que precisa encontrar o máximo dentre os elementos, consequentemente varrendo os mesmos até o final pelo menos uma vez. O comportamento dos algoritmos também permite concluir que o melhor deles em termos de tempo de execução foi o Counting sort, exceto para o caso dos já ordenados, que foi o Insertion Sort, comportamento esperado do mesmo visto que ele apenas compara uma vez cada elemento, que por conta de estarem na ordem correta não precisam ser comparados com os anteriores, o que acarreta no comportamento linear.

Em conclusão, o projeto permitiu um aprofundamento no conhecimento dos diversos algoritmos de ordenação bem como seu funcionamento interno e requisitos de sistema. Uma futura pesquisa poderia pegar a base do projeto porém além de realizar as comparações de tempo dentro da mesma linguagem, realizar a implementação dos mesmos algoritmos em outras linguagens e então comparar o tempo entre elas.

Appendices

Apêndice A

Fonte

Todo o código fonte e os materiais utilizados para produção do relatório se encontram no repositório abaixo:

https://github.com/sheeptraveler/sorting_algorithms_comparison_python