Laboratório de Estrutura de Dados

Primeira versão do projeto da disciplina

Comparação entre os algoritmos de ordenação elementar

Caio Sérgio Ramalho Lima

1. Introdução

Este relatório apresenta os resultados obtidos no projeto desenvolvido na disciplina de Laboratório de Estrutura de Dados e Algoritmos (LEDA), cujo objetivo foi avaliar o desempenho de diferentes algoritmos de ordenação aplicados a conjuntos de dados reais.

Durante o projeto, implementamos uma ferramenta capaz de aplicar diversos algoritmos de ordenação sobre dados estruturados contendo atributos como user, text e mentionedPersonCount. Os testes foram realizados considerando três cenários: melhor caso, caso médio e pior caso, para cada algoritmo.

Os resultados obtidos foram organizados em tabelas e gráficos, permitindo uma análise comparativa clara entre os algoritmos quanto ao tempo de execução. Este relatório está dividido em: uma explicação do método utilizado, o ambiente de testes, os resultados com comparativos e a análise final.

2. Descrição geral sobre o método utilizado

Os testes foram realizados com três categorias de dados textuais e numéricos, utilizando listas de objetos a serem ordenados conforme cada atributo. Os algoritmos avaliados incluem:

- **Insertion Sort**: algoritmo simples, eficaz para listas pequenas, baseia-se na construção incremental de uma lista ordenada.
- **Selection Sort**: seleciona repetidamente o menor elemento restante, de complexidade quadrática.
- **Merge Sort**: algoritmo estável e eficiente baseado em divisão e conquista.
- **Quick Sort**: algoritmo eficiente que particiona os dados ao redor de um pivô.
- **Quick Sort com Mediana de 3**: variação do Quick Sort com escolha aprimorada de pivô.

- **Heap Sort**: baseado em heap binário, garante complexidade O(n log n) em todos os casos.
- **Counting Sort**: adequado para dados inteiros com intervalo conhecido e limitado.

Cada algoritmo foi executado três vezes por atributo, simulando os cenários de melhor, médio e pior caso, conforme padrões já conhecidos de complexidade. A implementação da ferramenta foi feita em Java, com uma interface simples que permitia a escolha do algoritmo e do atributo a ser ordenado. O tempo de execução foi medido usando System.nanoTime().

Descrição geral do ambiente de testes

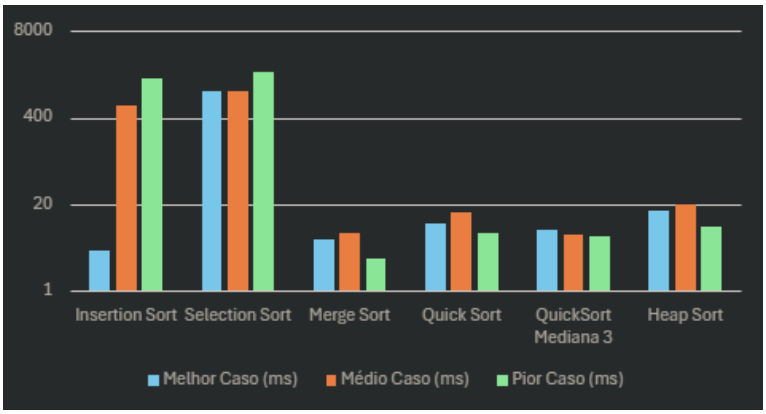
- **Processador:** AMD Ryzen 5 5600G
- **Memória RAM:** 16 GB DDR4
- **Sistema Operacional:** Windows 10
- **Java:** JDK 17

3. Resultados e Análise

Antes de partir para os comparativos focados, é importante observar os dados completos obtidos para cada atributo analisado. As tabelas a seguir mostram os tempos médios, mínimos e máximos (melhor, médio e pior caso) de execução para os algoritmos utilizados.

Tabelas completas de resultados por atributo

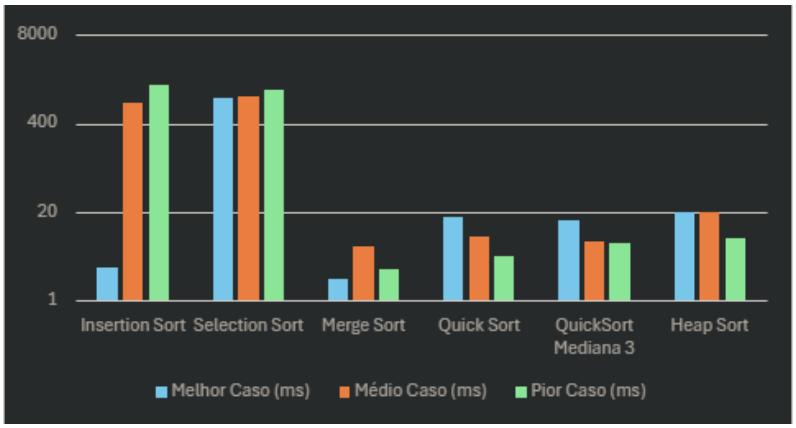
Atributo: User



Para o atributo user, que corresponde a strings (nomes de usuário), observa-se uma grande diferença entre os algoritmos de ordenação simples e os mais avançados. Insertion Sort e Selection Sort apresentaram tempos significativamente mais altos nos cenários médio e pior caso. O Selection Sort, em especial, teve desempenho ruim até no melhor caso, com quase 1 segundo de execução.

Por outro lado, algoritmos como Merge Sort e Quick Sort (principalmente com mediana de 3) foram bastante eficientes, com tempos baixos mesmo no pior caso. O Merge Sort teve um comportamento curioso: seu pior caso foi mais rápido que o melhor, possivelmente devido à distribuição dos dados de entrada.

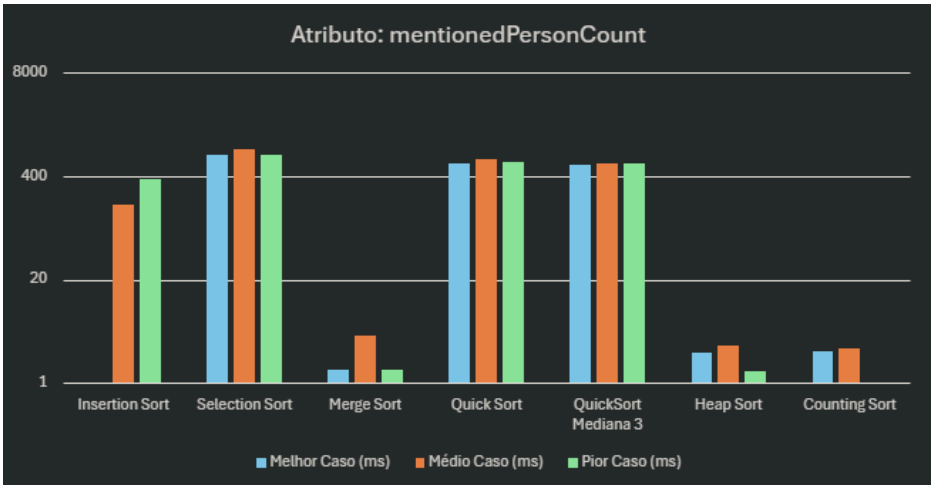
Atributo: date



A ordenação por date (datas dos tweets) lida com strings de data que seguem um padrão fixo (dd/MM/yyyy). Isso facilita a comparação, mas ainda depende de uma ordenação lexicográfica precisa.

O Merge Sort teve desempenho estável e excelente em todos os casos. Já o Insertion Sort, embora eficiente nos melhores e médios casos, apresentou forte queda de desempenho no pior caso (808 ms). Selection Sort e as duas versões do Quick Sort foram menos eficientes, com tempos altos mesmo nos melhores casos. O Heap Sort teve desempenho variável, mas relativamente bom, especialmente no pior caso.

Atributo: mentionedPersonCount

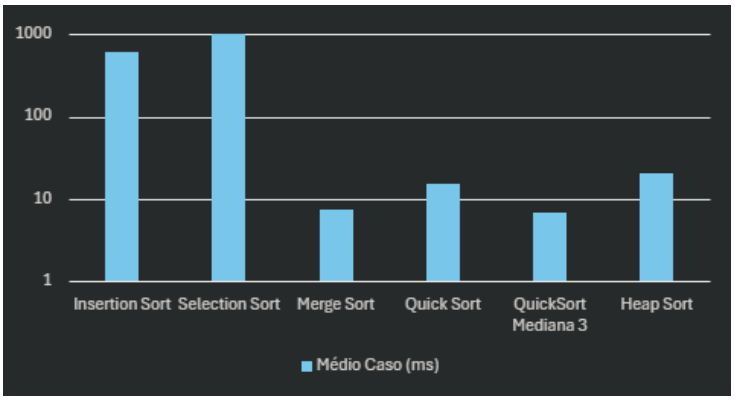


Este atributo é numérico e com valores limitados, favorecendo algoritmos especializados como o Counting Sort. Isso é evidenciado pelos tempos extremamente baixos obtidos, especialmente no pior caso, com apenas 0,407 ms.

Heap Sort e Merge Sort também apresentaram bons resultados, com tempos baixos e estáveis. Em contrapartida, algoritmos baseados em comparação, como Quick Sort e Selection Sort, tiveram desempenho inferior, com tempos na casa das centenas de milissegundos.

Comparação entre os algoritmos de ordenação

Atributo: user (Caso Médio)



Neste cenário, observamos uma grande diferença de desempenho entre os algoritmos simples (como Insertion e Selection Sort) e os mais avançados. O QuickSort com Mediana de 3 foi o mais rápido no caso médio, superando inclusive o Merge Sort. A melhoria em relação ao Quick Sort tradicional deve-se à escolha mais equilibrada de pivô, que evita partições muito desbalanceadas.

Algoritmos quadráticos como Insertion e Selection Sort foram os mais lentos, com tempos médios acima de 600 ms e 1000 ms, respectivamente. Isso evidencia sua limitação quando aplicados a listas maiores ou desordenadas, mesmo com strings curtas (como os nomes de usuários).

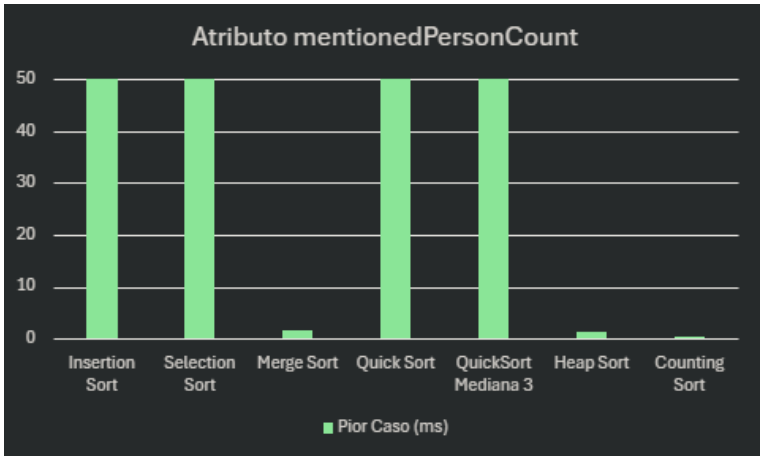
Atributo: date (Pior Caso)

No pior caso da ordenação por date, o Merge Sort e o Heap Sort se destacaram com desempenhos excelentes, registrando menos de 5 ms. Isso confirma a estabilidade do Merge Sort e a consistência do Heap Sort em cenários altamente desordenados.

Datas, mesmo representadas como strings, seguem um formato padronizado (dd/MM/yyyy), o que ajuda algoritmos baseados em comparação a funcionarem bem, desde que a estratégia de ordenação seja eficiente.

Já os algoritmos quadráticos, como Insertion e Selection Sort, apresentaram tempos muito altos, com o Selection passando de 2 segundos. Os dois tipos de QuickSort tiveram tempos acima de 1 segundo, o que mostra que, embora sejam rápidos em média, podem sofrer bastante em piores cenários, especialmente se o pivô for mal escolhido.

Atributo: mentionedPersonCount (Pior Caso)



Neste atributo numérico, foi utilizada uma escala de valores inteiros entre 0 e 50 no pior caso, justamente para testar a eficiência do Counting Sort nesse cenário. O resultado foi extremamente expressivo: o algoritmo registrou apenas 0.407 ms, sendo o mais rápido disparado entre todos.

Essa performance se deve ao fato de o Counting Sort não depender de comparações, mas de contagens diretas em vetores indexados pelos próprios valores. Com um intervalo tão pequeno, o algoritmo consegue atingir desempenho quase instantâneo.

Heap Sort e Merge Sort também apresentaram tempos muito baixos (abaixo de 1.5 ms), confirmando sua robustez em cenários gerais. São alternativas adequadas quando não é possível usar Counting Sort, como no caso de dados com intervalo grande ou valores negativos.

Os algoritmos baseados em comparação (como Quick Sort e sua variação com mediana) foram significativamente mais lentos (acima de 500 ms), enquanto os quadráticos (Insertion e Selection Sort) continuaram apresentando baixo desempenho, com o Selection Sort ultrapassando 740 ms.

Resumindo, o Counting Sort se destacou com 0.407 ms, por não depender de comparação e se beneficiar de um intervalo pequeno (0 a 50). Merge e Heap Sort também

foram eficientes, com tempos abaixo de 1.5 ms. Os demais algoritmos foram significativamente mais lentos.

Resumo Comparativo

- **Melhor algoritmo por atributo**

user (caso médio):

- QuickSort com Mediana de 3: teve o melhor desempenho nesse cenário, inclusive superando o Merge Sort. Sua escolha equilibrada de pivô garantiu partições mais estáveis e execução mais rápida em listas com strings curtas e moderadamente desordenadas.

date (pior caso):

Merge Sort: apresentou o menor tempo no pior caso com datas, demonstrando excelente desempenho mesmo em cenários extremos. Sua estabilidade e divisão eficiente o tornam ideal para esse tipo de dado ordenado por comparação.

mentionedPersonCount (pior caso):

Counting Sort – foi o mais eficiente, com tempo extremamente baixo (0.407 ms). O dado foi simulado em uma escala de 0 a 50, o que permitiu ao algoritmo operar em sua máxima eficiência. É a escolha ideal para valores numéricos com intervalo limitado.

Atributo	Melhor Algoritmo	Tempo Padrão (ms)	Observação
User (médio caso)	Quick Sort com Mediana de 3	142	Superou MergeSort com o pivô mais estável
Date (pior caso)	Merge Sort	3,8	Estável mesmo em cenário desordenado
mentionedPersonCount (pior caso)	Counting Sort	0,407	Ideal para inteiros com intervalo pequeno

- **Algoritmo mais estável**

O Merge Sort desempenhou-se de forma consistente e eficiente em todos os atributos e casos (melhor, médio e pior). Apresentou baixa variação nos tempos de execução, sendo uma opção confiável para cenários em que previsibilidade e estabilidade são essenciais.

- **Algoritmo com pior desempenho geral**

O Selection Sort foi o algoritmo mais lento em praticamente todos os cenários, inclusive no melhor caso. Sua complexidade quadrática ($O(n^2)$) o tornou inviável para listas médias ou grandes. Não apresentou vantagem competitiva em nenhum tipo de dado analisado.

Considerações Finais

A análise realizada mostrou que a escolha do algoritmo de ordenação deve levar em conta não apenas a complexidade, mas principalmente o tipo de dado e o contexto de uso. Em cenários reais, como o ordenamento de tweets por nome de usuário, data ou número de pessoas mencionadas, a performance pode variar significativamente.

- Merge Sort e Quick Sort com Mediana de 3 destacaram-se em dados textuais por eficiência e estabilidade;
- Counting Sort foi imbatível em atributos numéricos com intervalo limitado;
- Algoritmos quadráticos, como Insertion e especialmente Selection Sort, foram ineficazes em listas maiores.

Essa comparação evidencia que não existe um algoritmo "universalmente melhor", mas sim o mais adequado a cada tipo de dado e situação. O entendimento do comportamento dos algoritmos diante de diferentes cenários é essencial para a tomada de decisão em aplicações reais.