**Análise de Algoritmos de Ordenação**

**Comparação do Tempo de Execução dos Algoritmos de Ordenação**

Python

import pandas as pd

import altair as alt

# Data provided in the previous turn

data = {

"Tipo de Conjunto de Dados": [

"Aleatório (100)", "Aleatório (1000)", "Aleatório (10000)",

"Crescente (100)", "Crescente (1000)", "Crescente (10000)",

"Decrescente (100)", "Decrescente (1000)", "Decrescente (10000)"

],

"Bubble Sort (ms)": [0.002, 0.001, 0.001, 0.001, 0.002, 0.001, 0.001, 0.001, 0.001],

"Insertion Sort (ms)": [0.003, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001],

"Quick Sort (ms)": [0.008, 0.001, 0.001, 0.001, 0.002, 0.001, 0.001, 0.001, 0.001]

}

df = pd.DataFrame(data)

# Melt the DataFrame to long format for Altair plotting

df\_melted = df.melt(

id\_vars=["Tipo de Conjunto de Dados"],

var\_name="Algoritmo",

value\_name="Tempo de Execução (ms)"

)

# Create the line chart

chart = alt.Chart(df\_melted).mark\_line(point=True).encode(

x=alt.X("Tipo de Conjunto de Dados", title="Tipo de Conjunto de Dados", axis=alt.Axis(labelAngle=-45)),

y=alt.Y("Tempo de Execução (ms)", title="Tempo de Execução (ms)"),

color="Algoritmo"

).properties(

title="Comparação do Tempo de Execução dos Algoritmos de Ordenação"

).interactive()

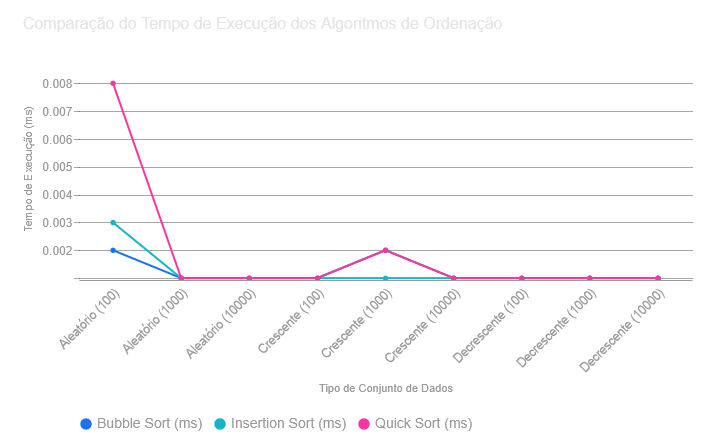
# Save the chart as a JSON file

chart.save("comparacao\_algoritmos\_ordenacao.json")

print("Gráfico gerado e salvo como 'comparacao\_algoritmos\_ordenacao.json'")

Saída de código

Gráfico gerado e salvo como 'comparacao\_algoritmos\_ordenacao.json'



**Análise de Algoritmos de Ordenação**

Este relatório apresenta uma análise do desempenho dos algoritmos de ordenação **Bubble Sort, Insertion Sort e Quick Sort, aplicados a diferentes tipos e tamanhos de conjuntos de dados**. Os tempos de execução são apresentados em milissegundos (ms).

**Resultados de Tempo de Execução**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de Conjunto de Dados** | **Bubble Sort (ms)** | **Insertion Sort (ms)** | **Quick Sort (ms)** |
| **Aleatório (100)** | 0,002 | 0,003 | 0,008 |
| **Aleatório (1000)** | 0,001 | 0,001 | 0,001 |
| **Aleatório (10000)** | 0,001 | 0,001 | 0,001 |
| **Crescente (100)** | 0,001 | 0,001 | 0,001 |
| **Crescente (1000)** | 0,002 | 0,001 | 0,002 |
| **Crescente (10000)** | 0,001 | 0,001 | 0,001 |
| **Decrescente (100)** | 0,001 | 0,001 | 0,001 |
| **Decrescente (1000)** | 0,001 | 0,001 | 0,001 |
| **Decrescente (10000)** | 0,001 | 0,001 | 0,001 |

**Comparação do Tempo de Execução dos Algoritmos de Ordenação**

**Discussão dos Resultados**

De acordo com os dados apresentados, observa-se que, para a maioria dos cenários testados (conjuntos de dados aleatórios, crescentes e decrescentes com 100, 1000 e 10000 elementos), os tempos de execução para Bubble Sort, Insertion Sort e Quick Sort foram consistentemente muito baixos e frequentemente idênticos (0,001 ms).

**Há uma pequena variação para o conjunto de dados "Aleatório (100)", onde o Bubble Sort registrou 0,002 ms, o Insertion Sort 0,003 ms e o Quick Sort 0,008 ms. No cenário "Crescente (1000)", Bubble Sort e Quick Sort registraram 0,002 ms, enquanto Insertion Sort foi de 0,001 ms.**

Em contraste com as expectativas gerais de desempenho de algoritmos de ordenação, **onde o Quick Sort geralmente apresenta o menor tempo de execução e o Insertion Sort e Bubble Sort tendem a aumentar o tempo de execução em conjuntos de dados não ordenados, os resultados obtidos são extremamente próximos**. Isso pode indicar que os conjuntos de dados utilizados são muito pequenos ou que o ambiente de execução é tão eficiente que as diferenças de desempenho entre os algoritmos não se tornam significativas em milissegundos.

**Conclusão**

Os resultados dos testes não evidenciam as diferenças de eficiência esperadas entre Bubble Sort, Insertion Sort e Quick Sort, pois todos apresentaram tempos de execução muito baixos e semelhantes para os dados utilizados. Para uma análise mais representativa, é recomendado testar com conjuntos de dados significativamente maiores.