INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Departamento de Teoria da Computação (IEC-T)

Lab 3 Ordenação de Strings

Comparação entre Bubble, Merge e Quick Sort em relação à eficiência de tempo

T28.4 Lucas Guedes Tamborelli

Professor: Armando Ramos Gouveia

São José dos Campos - São Paulo
2024

1 Introdução

Primeiramente, objetiva-se ordenar uma grande quantidade de *strings* desordenadas, em ordem alfabética, utilizando diferentes métodos de ordenação. Há muitos métodos possíveis, e para esta análise, foram tomados: o *Bubble Sort*, o *Quick Sort* e o *Merge Sort*. Além disso, para o *Merge Sort*, foram exploradas duas abordagens: uma utilizando um *array* auxiliar de *strings Static*, que permanece alocado na memória até o término da execução do programa, e outra sem o uso de *Static*. Dessa forma, realiza-se essas análises dos diferentes métodos para comparar seus desempenhos de ordenação em diferentes situações.

2 Resultados

2.1 Análise dos dois segundos

O teste em questão, trata-se de descobrir o número máximo de entradas que o programa de cada um dos métodos de ordenação consegue rodar em até dois segundos recebendo como entradas *strings* de 50 caracteres, geradas a partir do gerador fornecido no *Classroom* de CES-11.

- Bubble Sort: Obteve-se o tamanho máximo de aproximadamente 11300 ($\approx 10^4$) entradas
- Merge Sort: Obteve-se o tamanho máximo de aproximadamente 210000 ($\approx 2,1 \cdot 10^5$) entradas.
- Merge Sort com Static: Obteve-se o tamanho máximo de aproximadamente 2600000 ($\approx 2,6 \cdot 10^6$) entradas.
- Quick Sort: Obteve-se o tamanho máximo de aproximadamente 3500000 ($\approx 3,5 \cdot 10^6$) entradas, no entanto, esse valor pode variar um pouco considerando-se a aleatoriedade intrínseca ao QuickSort, devido a considerável diferença entre o pior caso e o caso médio, isto é, $O(n^2)$ e $O(n \cdot log(n))$ respectivamente.

2.2 Bubble Sort

Os gráficos 1 e 2 foram obtidos com os dados da tabela 1, gerada com o número de entradas múltiplos de mil e iniciando em mil.

Comparações	**Tempo (s)**	**Entradas**
998001	0,012	1000
3996001	0,054	2000
8994001	0,134	3000
15992001	0,230	4000
24990001	0,375	5000
35988001	0,548	6000
48986001	0,752	7000
63984001	0,983	8000
80982001	1,240	9000
99980001	1,570	10000
120978001	1,951	11000
143976001	2,331	12000
168974001	2,795	13000
195972001	3,655	14000
224970001	4,534	15000

Tabela 1: Tabela de Comparações, Tempo e Entradas do algoritmo Bubble Sort

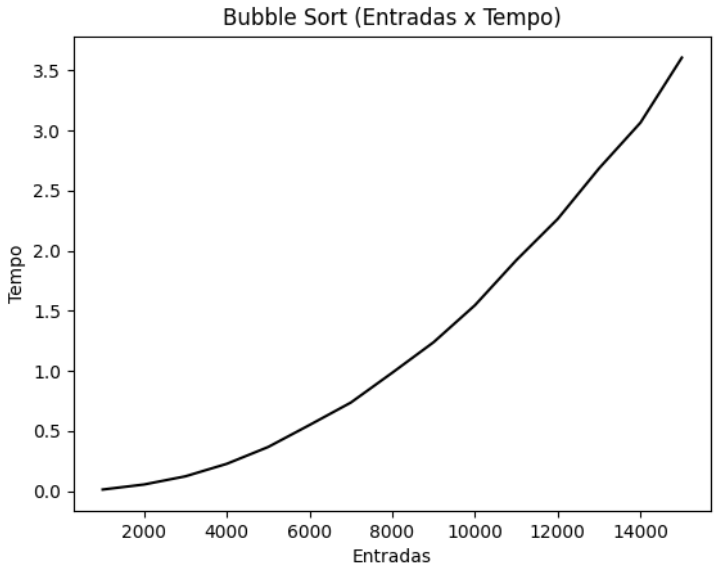

Figura 1: Gráfico do número de entradas no *Bubble Sort* por tempo, em segundos

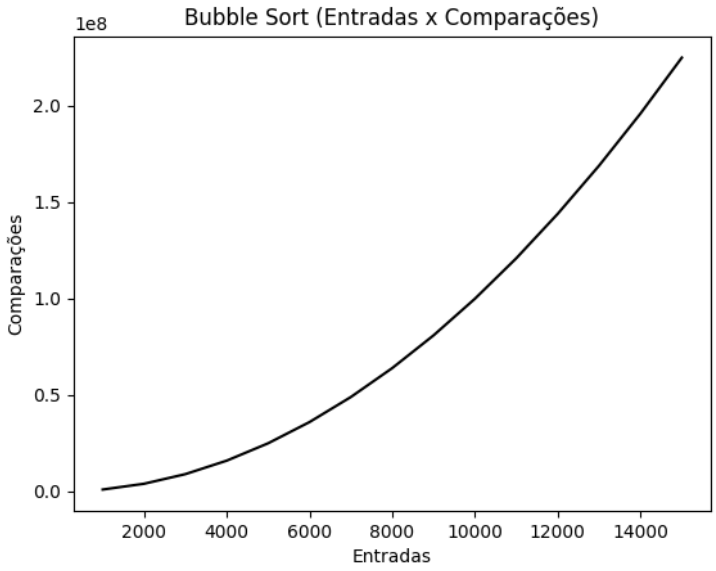

Figura 2: Gráfico do número de entradas no *Bubble Sort* por comparações realizadas

2.3 Merge Sort

A partir da comparações entre as duas possibilidades de Merge Sort, realizadas na secção 2.1, percebeu-se que utilizar o vetor auxiliar *static* era superior em relação ao tempo gasto pelo programa, uma vez que, não há mais a declaração, a alocação de memória e a cópia do *array* a cada chamada da função. Portanto, toma-se o *merge static* para as futuras análises gráficas de tempo e de comparações.

Os gráficos 3 e 4 foram obtidos com os dados da tabela 2, gerada com o número de entradas múltiplos de $5 \cdot 10^5$ e iniciando em $5 \cdot 10^5$.

Comparações	Tempo(s)	Entradas
8836086	0.2760	500000
18675407	0.5750	1000000
28908269	0.9090	1500000
39347008	1.5560	2000000
50025539	2.1050	2500000
60819233	2.3720	3000000
71721034	3.0500	3500000
82697469	3.3940	4000000
93820784	3.5370	4500000
105050347	4.2910	5000000
116319778	4.4800	5500000
127640462	5.2810	6000000
139006463	5.4380	6500000
150441891	6.2090	7000000
161910535	6.8840	7500000

Tabela 2: Tabela de Comparações, Tempo e Entradas do algoritmo Merge Sort

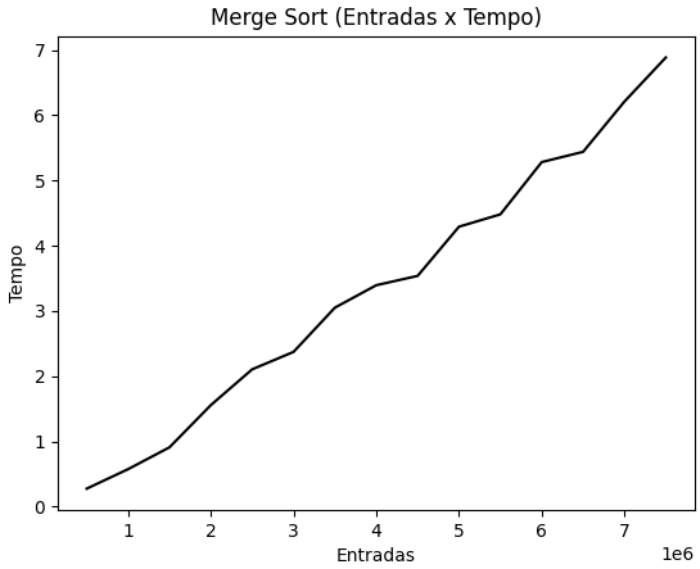

Figura 3: Gráfico do número de entradas no *Merge Sort* por tempo, em segundos

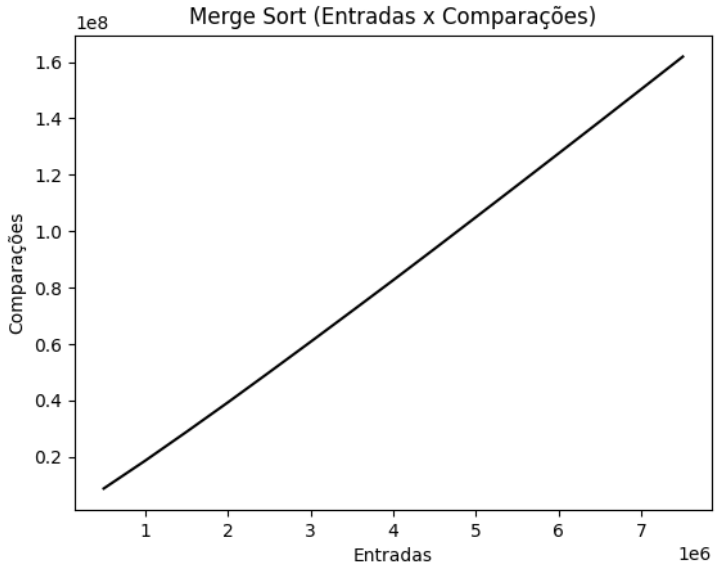

Figura 4: Gráfico do número de entradas no *Merge Sort* por comparações realizadas

2.4 Quick Sort

Os gráficos 5 e 6 foram obtidos com os dados da tabela 2, gerada com o número de entradas múltiplos de $5 \cdot 10^5$

Comparações	Tempo (s)	Entradas
15536862	0.252	500000
33774420	0.510	1000000
54853169	0.837	1500000
76555900	1.165	2000000
101610909	1.534	2500000
126100946	1.986	3000000
154732580	2.348	3500000
180662960	2.476	4000000
215534606	2.797	4500000
245495513	3.288	5000000
290003883	3.576	5500000
323923544	3.997	6000000
367065367	4.418	6500000
408035982	4.858	7000000
448930680	5.278	7500000

Tabela 3: Tabela de Comparações, Tempo e Entradas do algoritmo Quick Sort

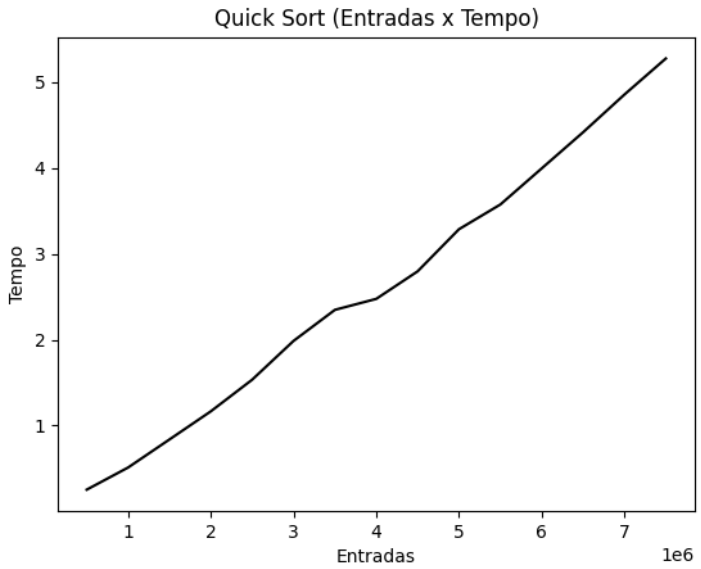

Figura 5: Gráfico do número de entradas no *Quick Sort* por tempo, em segundos

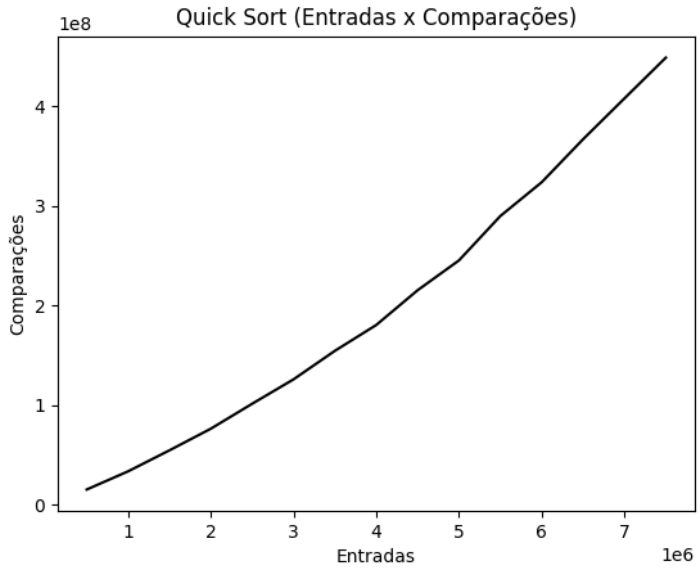

Figura 6: Gráfico do número de entradas no *Quick Sort* por comparações realizadas

3 Análise dos Dados

Todas as strings utilizadas foram geradas utilizando-se do gerador fornecido no *Classsroom* de CES-11.

Plotando-se o gráfico da complexidade esperada para cada algoritmo, pode-se verificar a coêrencia dos resultados obtidos:

- **Bubble Sort:** O algoritmo possui complexidade teórica de $O(n^2)$. A supeposição do gráfico 1 com a curva ajustada: $y = a \cdot x^2 + b \cdot x$ evidencia o crescimento quadrático como esperado.

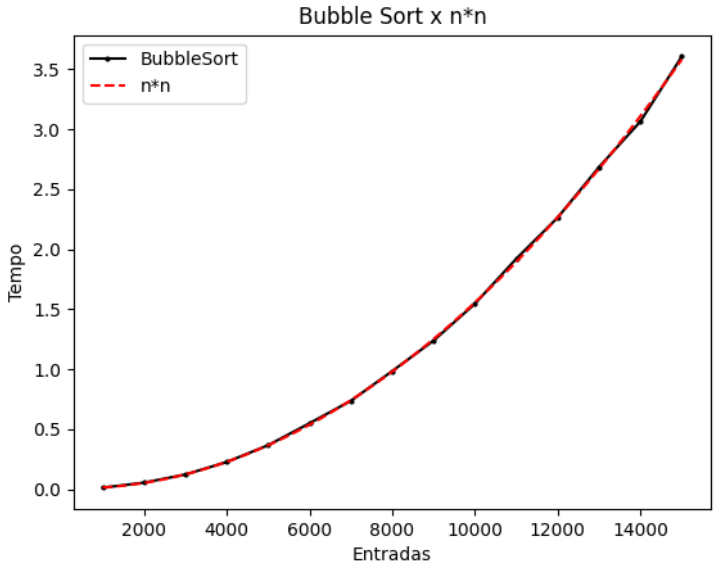

Figura 7: Gráfico do número de entradas por tempo no *Bubble Sort* com a curva $O(n^2)$

- **Quick Sort:** O algoritmo possui complexidade teórica de $O(n \log n)$ para o caso médio. A superposição do gráfico 5 com a curva ajustada: $y = a \cdot x \cdot log(x) + b \cdot x$, confirma a semelhança com o comportamento previsto, com uma ressalva para as pequenas variações, uma vez que, o algoritmo possui certa variação a depender do caso aleatório.

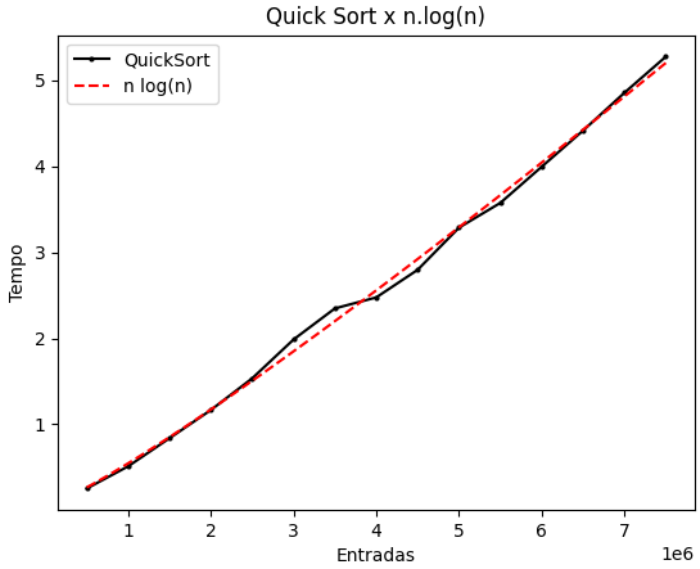

Figura 8: Gráfico do número de entradas por tempo no *Quick Sort* com a curva $O(n \cdot log(n))$

- **Merge Sort:** O algoritmo também com complexidade $O(n \log n)$. A superposição do gráfico 3 com a curva ajustada: $y = a \cdot x \cdot log(x) + b \cdot x$ evidencia novamente o crescimento teórico.

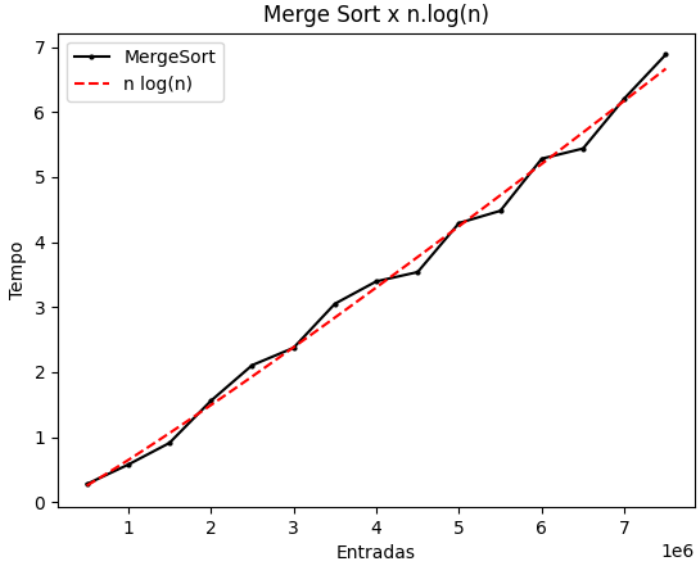

Figura 9: Gráfico do número de entradas por tempo no *Merge Sort* com a curva $O(n \cdot log(n))$

Os gráficos sobrepostos permitem visualisar o quão o algoritmo se aproximou do previsto para uma quantidade de 15 entradas com diferentes quantidades de strings de 50 caracteres, listadas nas tabelas 1, 2 e 3.

Ainda, o número de comparações e tempo gasto no programa são, aproximadamente, proporcionais entre si, nos três algoritmos de ordenação. Isto é, o tempo total de comparação entre strings é consideravelmente superior ao tempo total das outras operações, de modo que a maior parte do tempo é advindo da quantidade de comparações realizadas entre as *strings*.

Portanto, os resultados se comportam de acordo com as previsões teóricas esperadas. O *Bubble* apresentou-se impraticável para grandes entradas devido a sua complexidade em $O(n^2)$. O *Merge* e o *Quick* apresentaram-se com complexidade $O(n \cdot log(n))$, no entanto, o *Quick* demonstrou maior eficiência, devido à sua menor constante multiplicativa, para o caso médio, a qual não é evidenciada na notação "O".

Código em Python para gerar os gráficos de cada algoritmo

```
import math
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.optimize import curve_fit

f = open("DADOS.txt", "r")
y= [] #tempo
x= [] #entradas
z= [] #comparações

#Os dados devem estar na ordem tempo, entradas, comparações.
for i, valor in enumerate(f.readlines()):
    v = float(valor.strip())
    if (i % 3 == 0):
        y.append(v)

    elif (i % 3 == 1):
        x.append(v)

    else:
        z.append(v)

#Plotagem do gráfico de tempo por entradas
plt.ylabel("Tempo")
plt.xlabel("Entradas")
plt.plot(x, y, color = "black")
plt.title("Sort (Entradas x Tempo)")
plt.show()

#Plotagem do gráfico de comparações por entradas
```

```
plt.ylabel("Comparações")
plt.xlabel("Entradas")
plt.plot(x, z, color = "black")
plt.title("Sort (Entradas x Comparações)")
plt.show()

#Funcao modelo para complexidade
def modelo_f(x, a,b, c):
    return  a*x*(np.log(x))+x*b + c

#popt: parâmetros ótimos
#pcov: estimativa do erro
popt, pcov = curve_fit(modelo_f, x, y, p0 =[1,1,1])

a_opt, b_opt, c_opt = popt
x_modelo = np.linspace(min(x), max(x), 100)
y_modelo = modelo_f(x_modelo, a_opt, b_opt, c_opt)

#Plotagem da funcao modelo com o primeiro gráfico
plt.ylabel("Tempo")
plt.xlabel("Entradas")
plt.plot(x, y, '-o',markersize=2, color = "black", label = "Sort")
plt.legend(loc = "upper left")
plt.plot(x_modelo, y_modelo, color = "red", linestyle = "--", label = "n log(n)")
plt.legend(loc = "upper left")
plt.title("Sort x n.log(n)")
plt.show()
```