Análise de Complexidade de Tempo do Método Merge Sort

Eduardo Costa de Paiva
eduardocspv@gmail.com
Frederico Franco Calhau
fredericoffc@gmail.com
Gabriel Augusto Marson
gabrielmarson@live.com

Faculdade de Computação
Universidade Federal de Uberlândia

16 de dezembro de 2015

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Sumário

Capítulo 1

Introdução

Este documento foi feito com o intuito de exibir uma análise do algoritmo Merge Sort com relação a tempo. Além disso, será feita uma comparação da curva de tempo do que se espera do algoritmo, ou seja, $O(n^2)$ com o caso prático.

1.1 Diretório

Dada a seguinte organização das pastas, utilizamos o arquivo testdriver.py, executando, uma função conveniente por vez. Para mais informações vá até ao apêndice.

OBS.: É necessário instalar o programa tree pelo terminal. Isso pode ser feito da seguinte maneira.

```
> sudo apt-get install tree
```

A seguir é mostrada a organização das pastas sendo que os diretórios significativas para o projeto são Codigos e Relatorio além do raíz:

```
tree --charset=ASCII -d
.
|-- Codigos
|   |-- Bubble
|   |   `-- __pycache__
|   |-- Bucket
|   |   `-- __pycache__
|   |-- Counting
|   |   `-- __pycache__
|   |-- Heap
|   |   `-- __pycache__
|   |-- Insertion
|   |   `-- __pycache__
|   |-- Merge
|   |   `-- __pycache__
|   |-- Quick
|   |   `-- __pycache__
|   `-- Selection
|       `-- __pycache__
|-- Other
```

```
|-- Plot
|-- __pycache__
`-- relatorio
    |-- imagens
    |   |-- Bubble
    |   |-- Bucket
    |   |-- Counting
    |   |-- Heap
    |   |-- Insertion
    |   |-- Merge
    |   |-- Quick
    |   |-- Radix
    |   `-- Selection
    |-- Relatorio_Bubble
    |-- Relatorio_Bucket
    |-- Relatorio_Counting
    |-- Relatorio_Heap
    |-- Relatorio_Insertion
    |-- Relatorio_Merge
    |-- Relatorio_Selection
    `-- Resultados
        |-- Bubble
        |-- Bucket
        |-- Counting
        |-- Heap
        |-- Insertion
        |-- Merge
        |-- Quick
        `-- Selection

47 directories
```

1.2 Códigos de programas

Seguem os códigos utilizados na análise de tempo do algoritmo Merge Sort.

1. MergeSort.py: Disponível na Listagem 1.1.

Listagem 1.1: MergeSort.py

```
1  import math
2
3  @profile
4  def intercala(A,p,q,r):
5    B = [0] *len(A)
6    for i in range(p,(q+1)):
7      B[i] = A[i]
8    for j in range(q+1,(r+1)):
9      B[r+q+1-j] = A[j]
10
11   i = p
12   j = r
13
14   for k in range (p,(r+1)):
15     if(B[i] <= B[j]):
```

```
16        A[k] = B[i]
17        i = i+1
18      else:
19        A[k] = B[j]
20        j = j-1
21
22  @profile
23  def merge(A):
24    mergeSort(A,0,len(A)-1)
25    return A
26
27
28  @profile
29  def mergeSort(A,esquerda,direita):
30    if(esquerda<direita):
31      meio = math.floor((esquerda+direita)/2)
32      mergeSort(A,esquerda,meio)
33      mergeSort(A,meio+1,direita)
34      intercala(A,esquerda,meio,direita)
35
36
37  #A = [3,20,52,2,54,23,17,18,1,4]
38
39
40  #Para criar uma lista preenchida com 0's e que possua tamanho de A
      basta
41  #print(mergeSort(A))
```

2. testeGeneric.py Disponível na Listagem 1.2

Listagem 1.2: testeGeneric.py

```
1  ##adicionei - Serve para importar arquivos em outro diretório
2  ###  A CADA NOVO MÉTODO MUDAR O IMPORT,  A CHAMADA DA FUNÇÃO E O SYS.
       PATH
3
4  import sys
5  sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final
       /Codigos/Merge')
6  sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final
       ')
7
8  from monitor import *
9  from memoria import *
10
11 from MergeSort import *
12 import argparse
13
14 parser = argparse.ArgumentParser()
15 parser.add_argument("n", type=int, help="número de elementos no vetor
        de teste")
16 args = parser.parse_args()
17
18 v = criavet(args.n)
19 merge(v)
20
21
22
23 ## A EXECUÇÃO DESSE ARQUIVO EH ASSIM
```

```
## NA LINHA DE COMANDO VC MANDA O NOME DO ARQUIVO E O TAMANHO DO ELEMNTO DO vetor
##EXEMPLO testeBubble.py 10
##ele gera um vetor aleatório (criavet) e manda pro bubble_sort
```

3. monitor.py Disponível na Listagem 1.3

Listagem 1.3: monitor.py

```
# Para instalar o Python 3 no Ubuntu 14 ou 15
#
# sudo apt-get install python3 python3-numpy python3-matplotlib ipython3 python3-psutil
#

from math import *
import gc
import random
import numpy as np


from tempo import *

# Vetores de teste
def troca(m,v,n): ## seleciona o nível de embaralhamento do vetor
    m = trunc(m)
    mi = (n-m)//2
    mf = (n+m)//2
    for num in range(mi,mf):
        i = np.random.randint(mi,mf)
        j = np.random.randint(mi,mf)
        #print("i= ", i, " j= ", j)
        t = v[i]
        v[i] = v[j]
        v[j] = t
    return v


def criavet(n, grau=-0.5, inf=-1000, sup=1000):
    passo = (sup - inf)/n
    if grau < 0.0:
        v = np.arange(sup, inf, -passo)
        if grau <= -1.0:
            return v
        else:
            return troca(-grau*n, v, n)
    elif grau > 0.0:
        v = np.arange(inf, sup, passo)
        if grau >= 1.0:
            return v
        else:
            return troca(grau*n, v, n)
    else:
        return np.random.randint(inf, sup, size=n)
        #return [random.random() for i in range(n)] # for bucket sort


#print(criavet(20))
```

```
50
51 #Tipo                                    grau
52 #aleatorio                               0
53 #ordenado crescente                      1
54 #ordenado decresente                     -1
55 #parcialmente ordenado crescente         0.5
56 #parcialmente ordenado descrescente      -0.5
57
58
59 def executa(fn, v):
60     gc.disable()
61     with Tempo(True) as tempo:
62         fn(v)
63     gc.enable()
```

4. testdriver.py Referenciado no apêndice A.

Capítulo 2

Gráficos

Seguem os Gráficos utilizadas no processo de análise do método Merge Sort:

1. Para um vetor aleatório
   (a) Complexidade de custo do método Merge Sort disponível na lista de imagens 2.1.

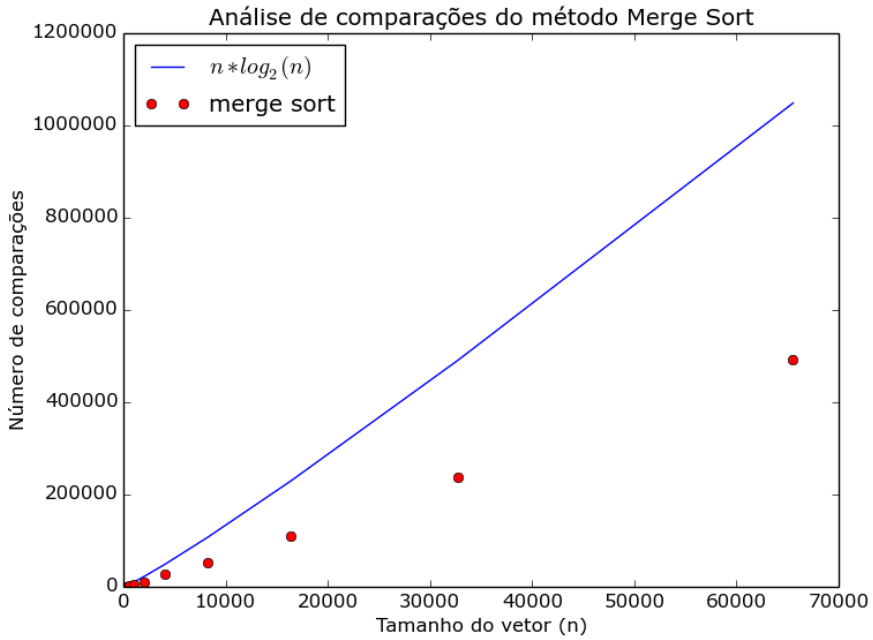

Figura 2.1: *Complexidade de custo do método Merge Sort (Vetor Aleatório)*

   (b) Complexidade de tempo do método Merge Sort disponível na lista de imagens 2.2.
   (c) Complexidade de tempo do método Merge Sort com mínimos quadrados disponível na lista de imagens 2.3.

2. Para um vetor ordenado crescente

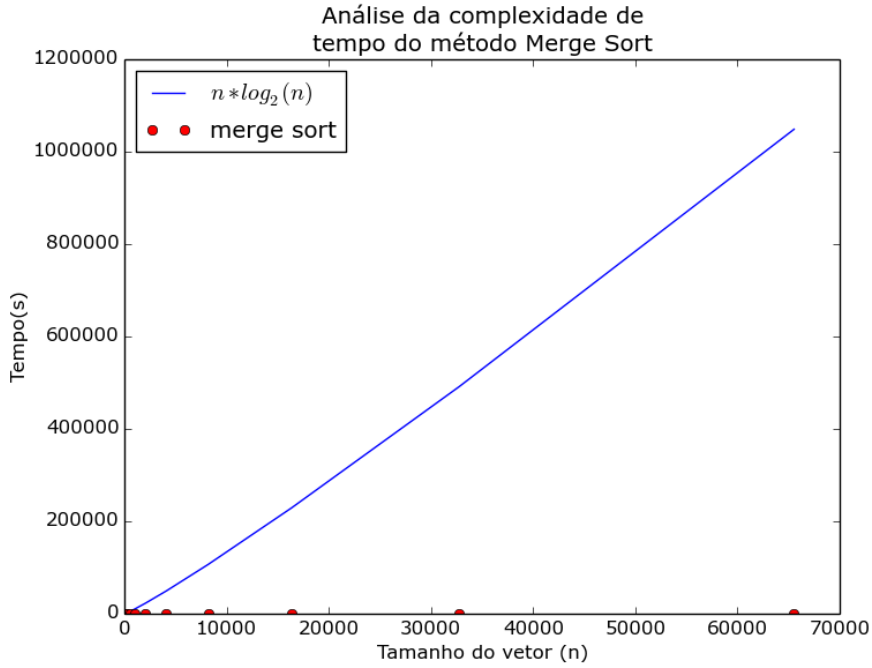

Figura 2.2: *Complexidade de tempo do método Merge Sort (Vetor Aleatório)*

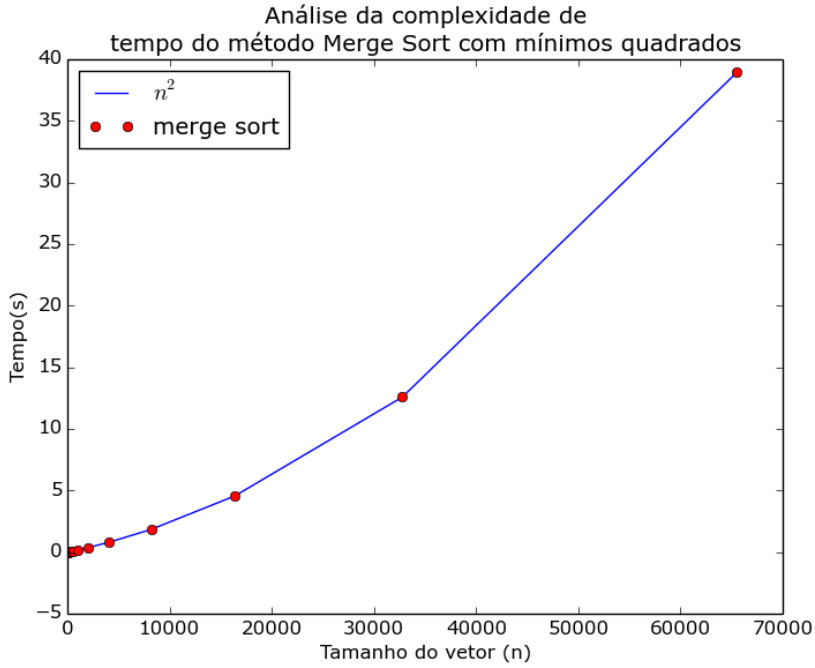

Figura 2.3: *Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Aleatório)*

(a) Complexidade de custo do método Merge Sort disponível na lista de imagens 2.4.

(b) Complexidade de tempo do método Merge Sort disponível na lista de imagens 2.5.

(c) Complexidade de tempo do método Merge Sort com mínimos quadrados disponível na lista de imagens 2.6.

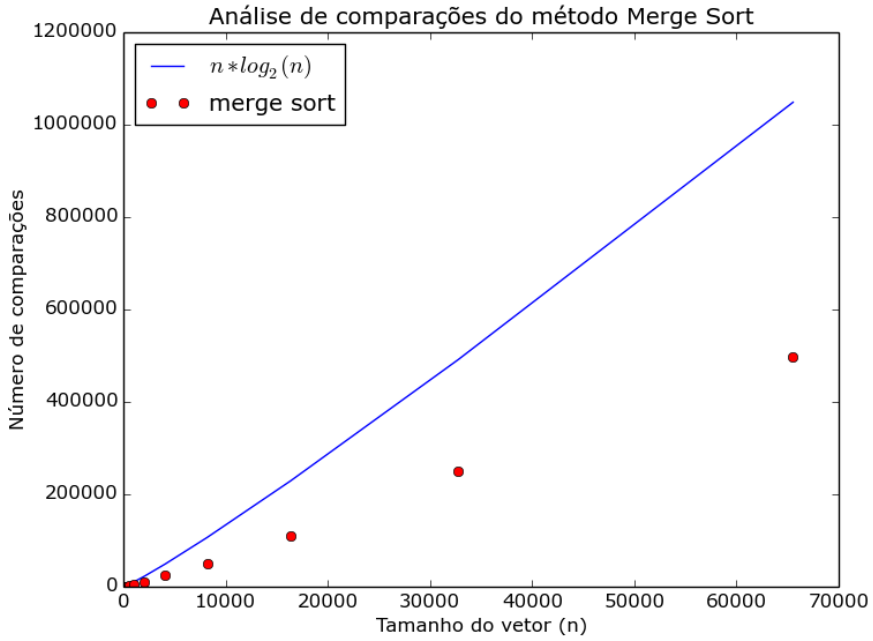

Figura 2.4: *Complexidade de custo do método Merge Sort (Vetor Ordenado Crescente)*

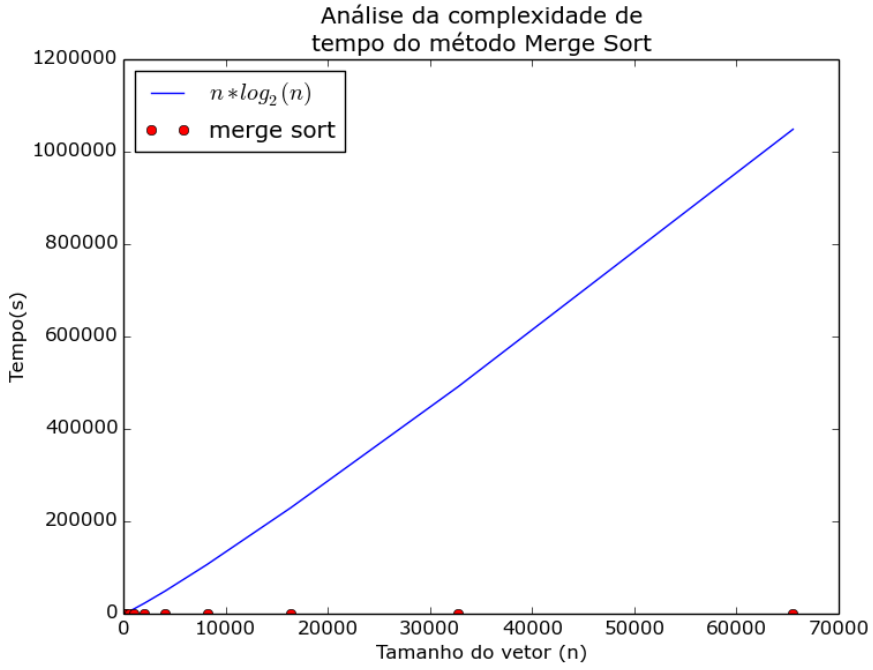

Figura 2.5: *Complexidade de tempo do método Merge Sort (Vetor Ordenado Crescente)*

3. Para um vetor ordenado decrescente

   (a) Complexidade de custo do método Merge Sort disponível na lista de imagens 2.7.

   (b) Complexidade de tempo do método Merge Sort disponível na lista de imagens 2.8.

   (c) Complexidade de tempo do método Merge Sort com mínimos quadrados disponível na lista de imagens 2.9.

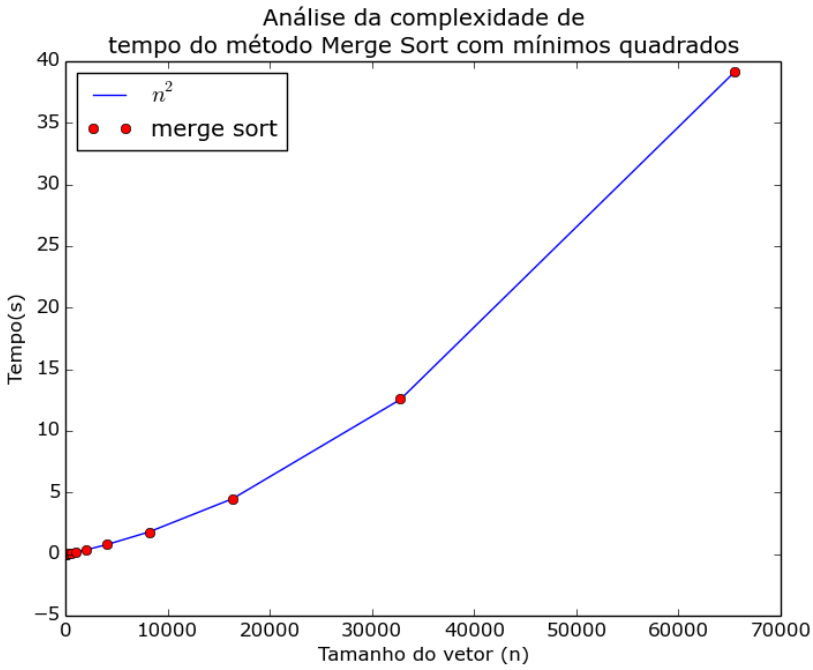

Figura 2.6: *Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Ordenado Crescente)*

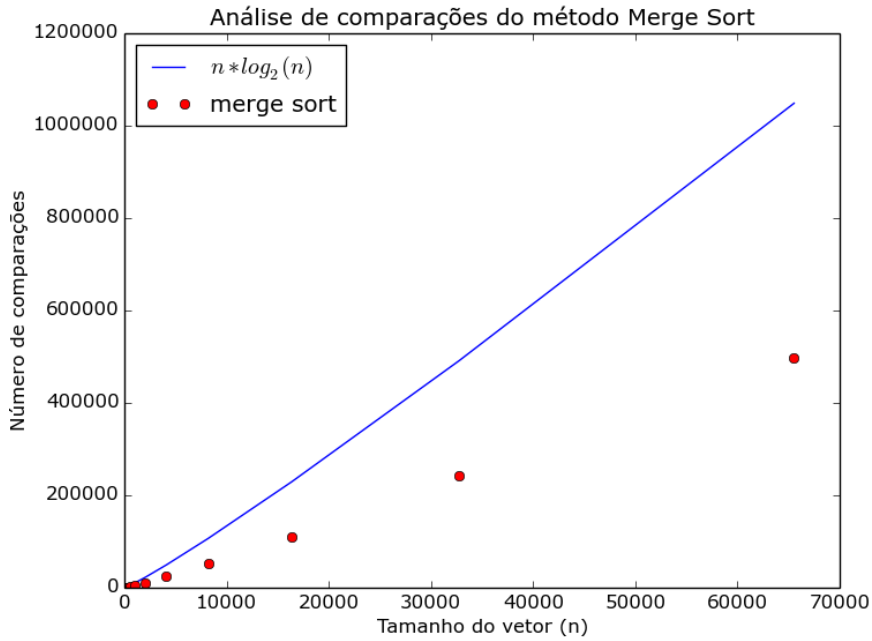

Figura 2.7: *Complexidade de custo do método Merge Sort (Vetor Ordenado Decrescente)*

4. Para um vetor parcialmente ordenado crescente

   (a) Complexidade de custo do método Merge Sort disponível na lista de imagens 2.10.

   (b) Complexidade de tempo do método Merge Sort disponível na lista de imagens 2.11.

   (c) Complexidade de tempo do método Merge Sort com mínimos quadrados dispo-

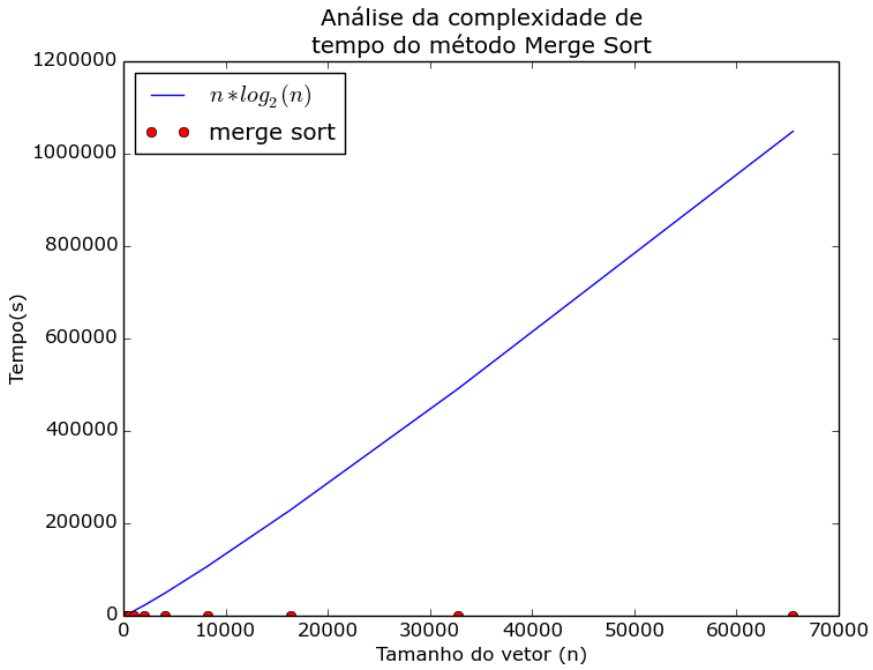

Figura 2.8: *Complexidade de tempo do método Merge Sort (Vetor Ordenado Decrescente)*

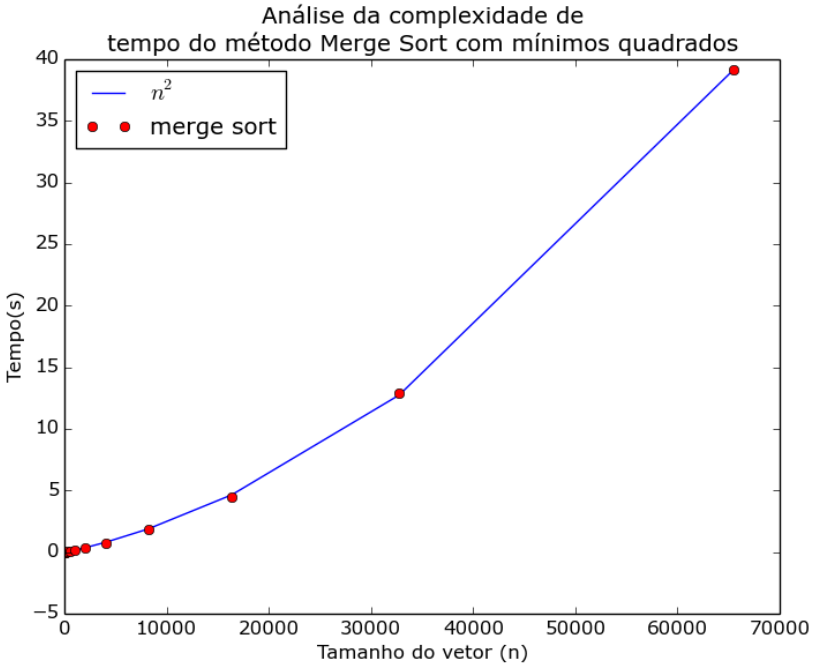

Figura 2.9: *Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Ordenado Decrescente)*

nível na lista de imagens 2.12.

5. Para um vetor parcialmente ordenado decrescente

   (a) Complexidade de custo do método Merge Sort disponível na lista de imagens 2.13.

   (b) Complexidade de tempo do método Merge Sort disponível na lista de imagens

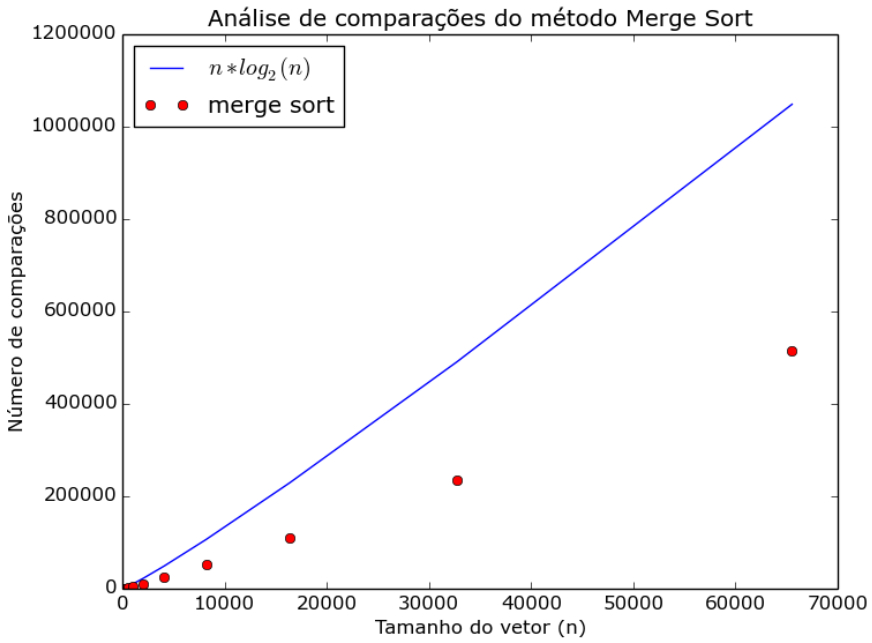

Figura 2.10: *Complexidade de custo do método Merge Sort (Vetor Parcialmente Ordenado Crescente)*

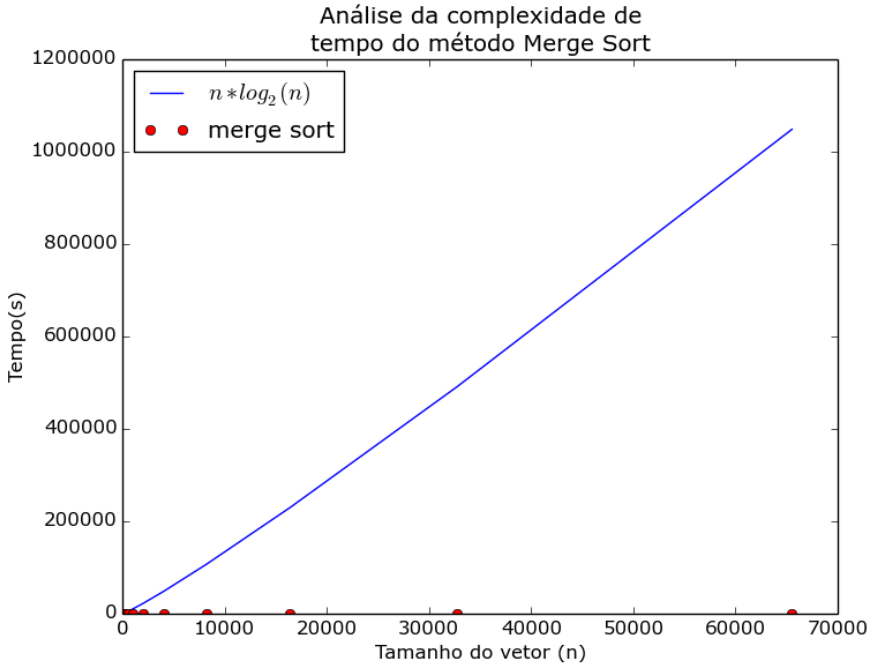

Figura 2.11: *Complexidade de tempo do método Merge Sort (Vetor Parcialmente Ordenado Crescente)*

2.14.

(c) Complexidade de tempo do método Merge Sort com mínimos quadrados disponível na lista de imagens 2.15.

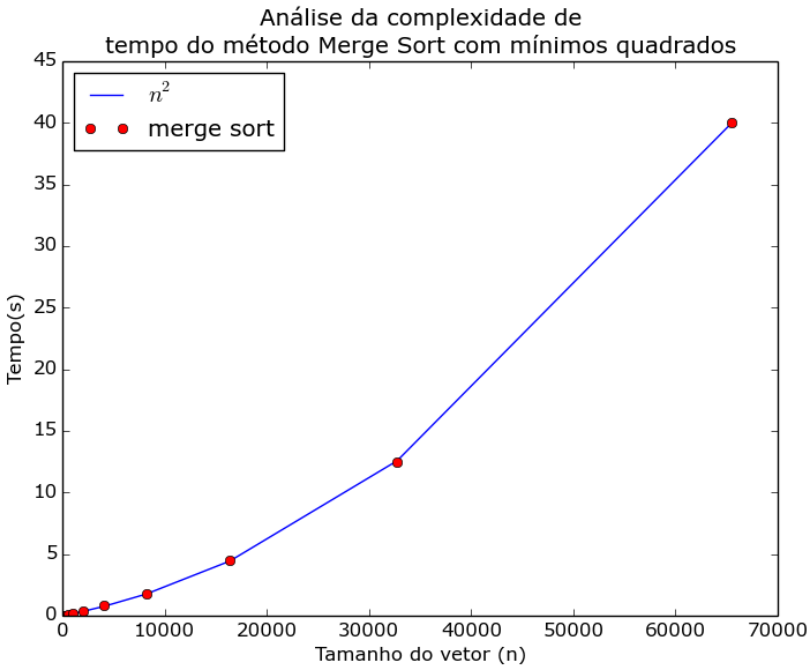

Figura 2.12: *Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Parcialmente Ordenado Crescente)*

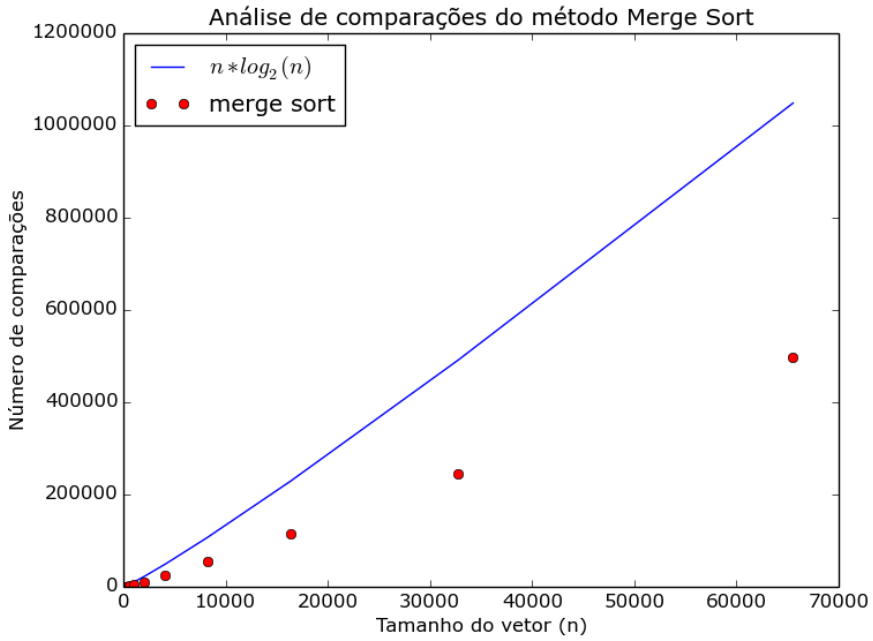

Figura 2.13: *Complexidade de custo do método Merge Sort (Vetor Parcialmente Ordenado Decrescente)*

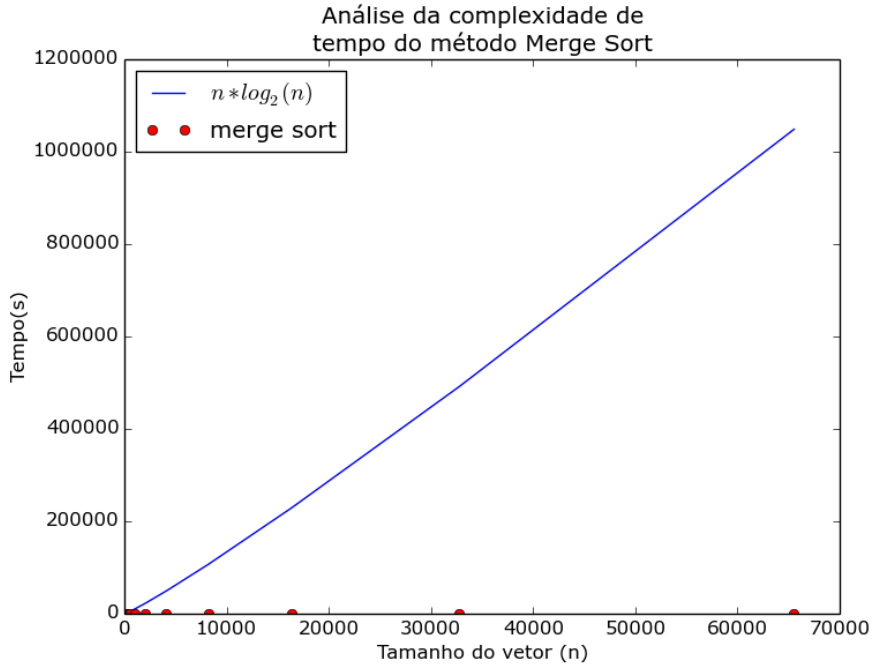

Figura 2.14: *Complexidade de tempo do método Merge Sort (Vetor Parcialmente Ordenado Decrescente)*

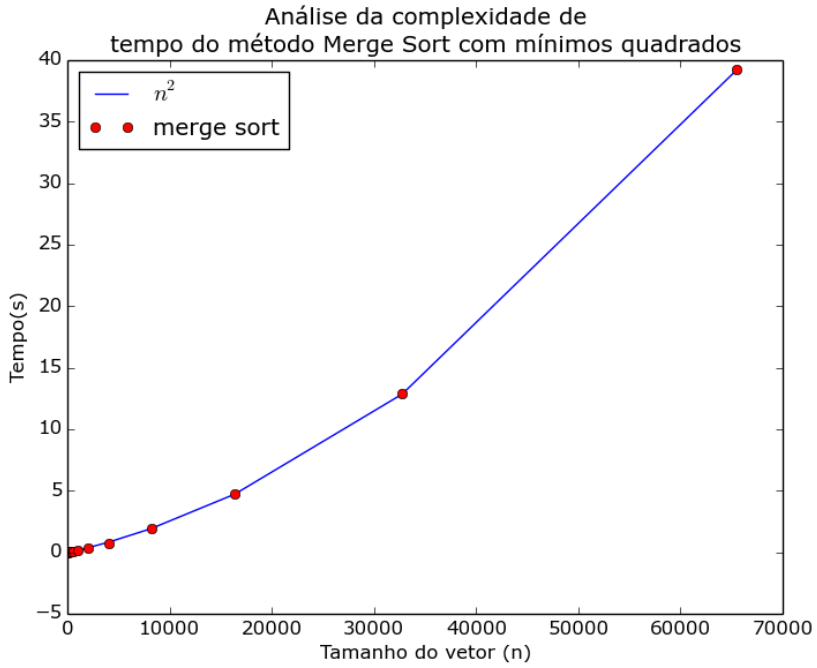

Figura 2.15: *Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Parcialmente Ordenado Decrescente)*

Capítulo 3

Tabelas

Seguem as tabelas utilizadas para a análise do método Merge Sort.

Tabela 3.1: *Vetor Aleatorio*

Tamanho do Vetor	Comparações	Tempo(s)
32	496	0.000756
64	2016	0.003263
128	8128	0.012834
256	32640	0.050149
512	130816	0.200732
1024	523776	0.882822
2048	2096128	4.874270
4096	8386560	18.039000
8192	33550336	65.256400
16384	134209536	312.534000

Tabela 3.2: *Vetor Ordenado Crescente*

Tamanho do Vetor	Comparações	Tempo(s)
32	496	0.000457
64	2016	0.001817
128	8128	0.007373
256	32640	0.028836
512	130816	0.114824
1024	523776	0.513465
2048	2096128	2.845550
4096	8386560	11.410900
8192	33550336	56.293100
16384	134209536	181.343000

Tabela 3.3: *Vetor Ordenado Decrescente*

Tamanho do Vetor	Comparações	Tempo(s)
32	496	0.001207
64	2016	0.004553
128	8128	0.018195
256	32640	0.073946
512	130816	0.299655
1024	523776	1.427660
2048	2096128	6.381420
4096	8386560	28.260700
8192	33550336	113.214000
16384	134209536	461.349000

Tabela 3.4: *Vetor Parcialmente Ordenado Crescente*

Tamanho do Vetor	Comparações	Tempo(s)
32	87	0.002646
64	221	0.006141
128	597	0.016766
256	1058	0.030065
512	2316	0.066540
1024	5330	0.157471
2048	10864	0.329800
4096	23856	0.752486
8192	54738	1.895006
16384	114088	4.755551
32768	245564	12.874460
65536	496850	39.231150

Tabela 3.5: *Vetor Parcialmente Ordenado Decrescente*

Tamanho do Vetor	Comparações	Tempo(s)
32	87	0.002646
64	221	0.006141
128	597	0.016766
256	1058	0.030065
512	2316	0.066540
1024	5330	0.157471
2048	10864	0.329800
4096	23856	0.752486
8192	54738	1.895006
16384	114088	4.755551
32768	245564	12.874460
65536	496850	39.231150

Capítulo 4

Análise

Podemos observar que todas as curvas de todos os gráficos, exceto os de complexidade de tempo sem a interpolação dos mínimos quadrados(Gráficos 2.2,2.5,2.8,2.11,2.14), apresentaram uma correspondência forte com a curva da função $F(x) = x^2$, o que nos permite concluir que, dada a complexidade de tempo do algoritmo Merge Sort por $G(x)$ então $F(x) = c*G(x)$ sendo que c é uma constante maior que zero e $x > x_0$. Portanto, o Merge Sort é $O(n^2)$.

Capítulo 5

Citações e referências bibliográficas

Apêndice A

Códigos extensos

A.1 testdriver.py

Listagem A.1: testdriver.py

```
1  # coding = utf-8
2  import subprocess
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import sys , shutil
6
7
8  ##PRA CADA NOVO METODO TEM QUE MUDAR
9  #Sys.path()
10
11 ## PARA CADA VETOR NOVO OU NOVO METODO TEM QUE MUDAR
12 #Para o executa_teste a chamada das funções e o shutil.move()
13 #para os plots        a chamada das funções e o savefig
14
15 sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
       Codigos/Merge')  ## adicionei o código de ordenação
16 sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
       relatorio/Resultados/Merge') ## adicionei o resultado do executa_teste
17
18
19 def executa_teste(arqteste, arqsaida,  intervalo,nlin=21,nlin2=47,tempo
       =[3,28,39]):
20     """Executa uma sequência de testes contidos em arqteste, com:
21        arqsaida: nome do arquivo de saída, ex: tBolha.dat
22        nlin: número da linha no arquivo gerado pelo line_profiler contendo
23              os dados de interesse. Ex:
24        intervalo: tamanhos dos vetores: Ex: 2 ** np.arange(5,10)
25     """
26     f = open(arqsaida,mode='w', encoding='utf-8')
27     f.write('#      n    comparações     tempo(s)\n')
28
29     for n in intervalo:
30         cmd = ' '.join(["kernprof -l -v", "testeGeneric.py", str(n)])
31         str_saida = subprocess.check_output(cmd, shell=True).decode('utf-8
               ')
32         linhas = str_saida.split('\n')
```

```
33          #for i in linhas:
34          #    print(i)
35          #print (linhas)
36
37          #print(linhas[tempo[0]].split()[2])
38
39
40          tempo_total = float(linhas[tempo[0]].split()[2]) + float(linhas[
                tempo[1]].split()[2]) + float(linhas[tempo[2]].split()[2])
41          unidade_tempo = float(linhas[3].split()[2])
42          lcomp = int(linhas[nlin].split()[2]) + int(linhas[nlin2].split()
                [2])
43
44          #print(linhas[nlin].split()[2])
45
46          #print(linhas[nlin2].split()[2])
47          #print(linhas[nlin3].split()[2])
48
49          #print ("unidade tempo: ",unidade_tempo )
50          #print("lcomp: ",lcomp)
51          #print("tempo total",tempo_total)
52
53          #num_comps = int(lcomp[1])
54          str_res = '{:>8} {:>13}  {:13.6f}'.format(n, lcomp ,tempo_total)
55          print(str_res)
56          f.write(str_res + '\n')
57          lcomp = 0
58      f.close()
59      shutil.move("tMerge_vetor_parcialmente_ordenado_decrescente.dat", "/
            home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/relatorio/
            Resultados/Merge/tMerge_vetor_parcialmente_ordenado_decrescente.dat
            ")
60
61  executa_teste("testeGeneric.py", "
        tMerge_vetor_parcialmente_ordenado_decrescente.dat", 2 ** np.arange
        (5,17))
62
63  def executa_teste_memoria(arqteste, arqsaida, nlin, intervalo):
64      """Executa uma sequência de testes contidos em arqteste, com:
65         arqsaida: nome do arquivo de saída, ex: tBolha.dat
66         nlin: número da linha no arquivo gerado pelo line_profiler contendo
67               os dados de interesse. Ex: 14
68         intervalo: tamanhos dos vetores: Ex: 2 ** np.arange(5,10)
69      """
70      f = open(arqsaida,mode='w', encoding='utf-8')
71      f.write('#      n   comparações      tempo(s)\n')
72
73      for n in intervalo:
74          cmd = ' '.join(["kernprof -l -v ", "testeGeneric.py", str(n)])
75
76          str_saida = subprocess.check_output(cmd, shell=True).decode('utf-8
                ')
77
78          linhas = str_saida.split('\n')
79          for i in linhas:
80              print(i)
81
82          print ("Linhas:",linhas[1])
83
```

```
84          unidade_tempo = float(linhas[1].split()[2])
85
86          str_res = '{:>8} {:>13} {:13.6f}'.format(n, n, n)
87          print(str_res)
88          f.write(str_res + '\n')
89      f.close()
90      #shutil.move("tMerge_memoria.dat", "/home/gmarson/Git/
            AnaliseDeAlgoritmos/Trabalho_Final/relatorio/Resultados/Merge/
            tMerge_memoria.dat")
91
92  #executa_teste_memoria("testeGeneric.py", "tMerge_memoria.dat", 14, 2 **
        np.arange(5,15))
93
94  def plota_teste1(arqsaida):
95      n, c, t = np.loadtxt(arqsaida, unpack=True)
96      #print("n: ",n,"\nc: ",c,"\nt: ",t)
97      #n eh o tamanho da entrada , c eh o tanto de comparações e t eh o
            tempo gasto
98      plt.plot(n, n * np.log2(n), label='$n * log_2(n)$')  ## custo esperado
             bubble Sort
99      plt.plot(n, c, 'ro', label='merge sort')
100     # Posiciona a legenda
101     plt.legend(loc='upper left')
102
103     # Posiciona o título
104     plt.title('Análise de comparações do método Merge Sort')
105
106     # Rotula os eixos
107     plt.xlabel('Tamanho do vetor (n)')
108     plt.ylabel('Número de comparações')
109
110     plt.savefig('relatorio/imagens/Merge/
            merge_plot_1_parcialmente_ordenado_decrescente.png')
111     plt.show()
112
113
114
115 def plota_teste2(arqsaida):
116     n, c, t = np.loadtxt(arqsaida, unpack=True)
117     plt.plot(n, n * np.log2(n), label='$n * log_2(n)$')
118     plt.plot(n, t, 'ro', label='merge sort')
119
120     # Posiciona a legenda
121     plt.legend(loc='upper left')
122
123     # Posiciona o título
124     plt.title('Análise da complexidade de \ntempo do método Merge Sort')
125
126     # Rotula os eixos
127     plt.xlabel('Tamanho do vetor (n)')
128     plt.ylabel('Tempo(s)')
129
130     plt.savefig('relatorio/imagens/Merge/
            merge_plot_2_parcialmente_ordenado_decrescente.png')
131     plt.show()
132
133
134
135
```

```
136 def plota_teste3(arqsaida):
137     n, c, t = np.loadtxt(arqsaida, unpack=True)
138 
139     # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
140     # o método dos mínimos quadrados
141     coefs = np.polyfit(n, t, 2)
142     p = np.poly1d(coefs)
143 
144     plt.plot(n, p(n), label='$n^2$')
145     plt.plot(n, t, 'ro', label='merge sort')
146 
147     # Posiciona a legenda
148     plt.legend(loc='upper left')
149 
150     # Posiciona o título
151     plt.title('Análise da complexidade de \ntempo do método Merge Sort com
            mínimos quadrados')
152 
153     # Rotula os eixos
154     plt.xlabel('Tamanho do vetor (n)')
155     plt.ylabel('Tempo(s)')
156 
157     plt.savefig('relatorio/imagens/Merge/
           merge_plot_3_parcialmente_ordenado_decrescente.png')
158     plt.show()
159 
160 plota_teste1("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
       relatorio/Resultados/Merge/
       tMerge_vetor_parcialmente_ordenado_decrescente.dat")
161 plota_teste2("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
       relatorio/Resultados/Merge/
       tMerge_vetor_parcialmente_ordenado_decrescente.dat")
162 plota_teste3("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
       relatorio/Resultados/Merge/
       tMerge_vetor_parcialmente_ordenado_decrescente.dat")
163 
164 
165 def plota_teste4(arqsaida):
166     n, c, t = np.loadtxt(arqsaida, unpack=True)
167 
168     # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
169     # o método dos mínimos quadrados
170     coefs = np.polyfit(n, c, 2)
171     p = np.poly1d(coefs)
172 
173     plt.plot(n, p(n), label='$n^2$')
174     plt.plot(n, c, 'ro', label='bubble sort')
175 
176     # Posiciona a legenda
177     plt.legend(loc='upper left')
178 
179     # Posiciona o título
180     plt.title('Análise da complexidade de \ntempo do método da bolha')
181 
182     # Rotula os eixos
183     plt.xlabel('Tamanho do vetor (n)')
184     plt.ylabel('Número de comparações')
185 
186     plt.savefig('bubble4.png')
```

```
187     plt.show()
188
189 def plota_teste5(arqsaida):
190     n, c, t = np.loadtxt(arqsaida, unpack=True)
191
192     # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
193     # o método dos mínimos quadrados
194     coefs = np.polyfit(n, c, 2)
195     p = np.poly1d(coefs)
196
197     # set_yscale('log')
198     # set_yscale('log')
199     plt.semilogy(n, p(n), label='$n^2$')
200     plt.semilogy(n, c, 'ro', label='bubble sort')
201
202     # Posiciona a legenda
203     plt.legend(loc='upper left')
204
205     # Posiciona o título
206     plt.title('Análise da complexidade de \ntempo do método da bolha')
207
208     # Rotula os eixos
209     plt.xlabel('Tamanho do vetor (n)')
210     plt.ylabel('Número de comparações')
211
212     plt.savefig('bubble5.png')
213     plt.show()
```