Análise de Complexidade de Tempo do Método Bubble 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 Bubble 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__
|   |-- Radix
|   `-- 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

48 directories
```

1.2 Códigos de programas

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

1. BubbleSort.py: Disponível na Listagem 1.1.

Listagem 1.1: BubbleSort.py

```
1  import numpy as np
2
3  @profile
4  def bubble_sort(a):
5      """ Implementação do método da bolha """
6      for i in range(len(a)):
7          for j in range(len(a)-1-i):
8              if a[j] > a[j+1]:
9                  t = a[j]
10                 a[j] = a[j+1]
11                 a[j+1] = t
12
13 #    print(a)  O PRINT BUGA O TESTDRIVER
```

2. testeGeneric.py Disponível na Listagem 1.2

Listagem 1.2: testeGeneric.py

```
##adicionei - Serve para importar arquivos em outro diretório
###  A CADA NOVO MÉTODO MUDAR O IMPORT,  A CHAMADA DA FUNÇÃO E O SYS.PATH

import sys
sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/Codigos/Merge')
sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final')

from monitor import *
from memoria import *

from MergeSort import *
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("n", type=int, help="número de elementos no vetor de teste")
args = parser.parse_args()

v = criavet(args.n)
merge(v)



## 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):
```

```
20          i = np.random.randint(mi,mf)
21          j = np.random.randint(mi,mf)
22          #print("i= ", i, " j= ", j)
23          t = v[i]
24          v[i] = v[j]
25          v[j] = t
26      return v
27
28
29  def criavet(n, grau=-0.5, inf=-1000, sup=1000):
30      passo = (sup - inf)/n
31      if grau < 0.0:
32          v = np.arange(sup, inf, -passo)
33          if grau <= -1.0:
34              return v
35          else:
36              return troca(-grau*n, v, n)
37      elif grau > 0.0:
38          v = np.arange(inf, sup, passo)
39          if grau >= 1.0:
40              return v
41          else:
42              return troca(grau*n, v, n)
43      else:
44          return np.random.randint(inf, sup, size=n)
45          #return [random.random() for i in range(n)] # for bucket sort
46
47
48
49  #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 Bubble Sort:

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

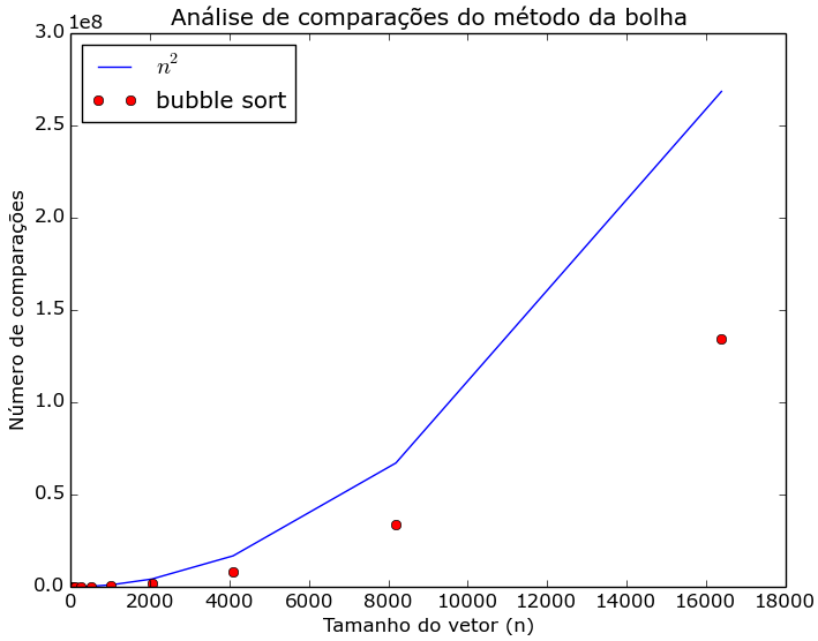

Figura 2.1: *Complexidade de custo do método da bolha (Vetor Aleatório)*

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

2. Para um vetor ordenado crescente
   (a) Complexidade de custo do método da bolha disponível na lista de imagens 2.4.
   (b) Complexidade de tempo do método da bolha disponível na lista de imagens 2.5.

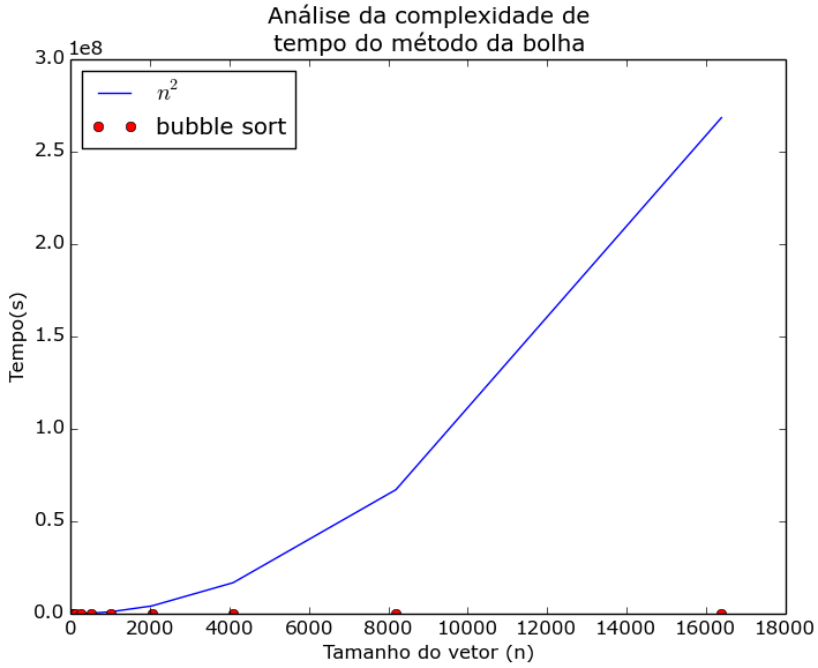

Figura 2.2: *Complexidade de tempo do método da bolha (Vetor Aleatório)*

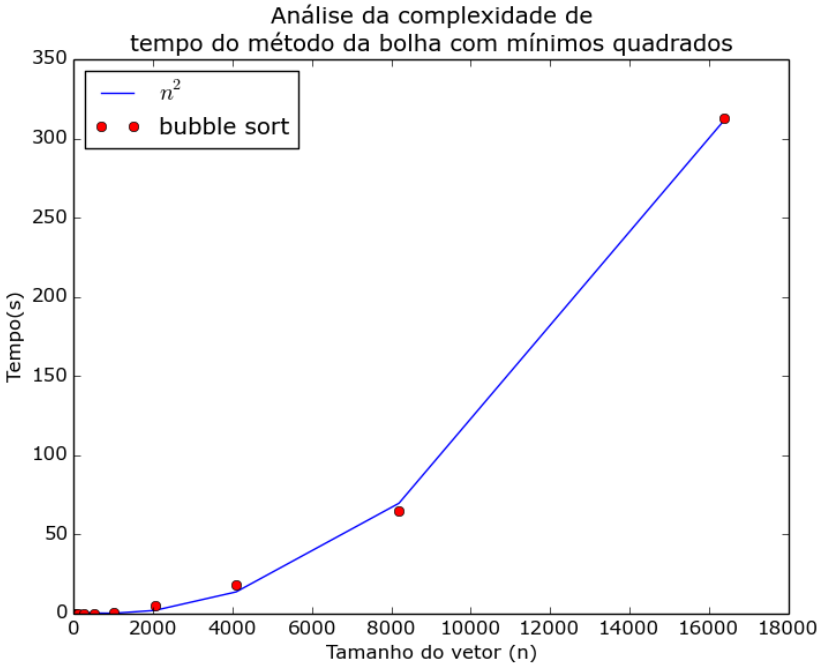

Figura 2.3: *Complexidade de tempo do método da bolha com mínimos quadrados (Vetor Aleatório)*

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

3. Para um vetor ordenado decrescente

(a) Complexidade de custo do método da bolha disponível na lista de imagens 2.7.

(b) Complexidade de tempo do método da bolha disponível na lista de imagens 2.8.

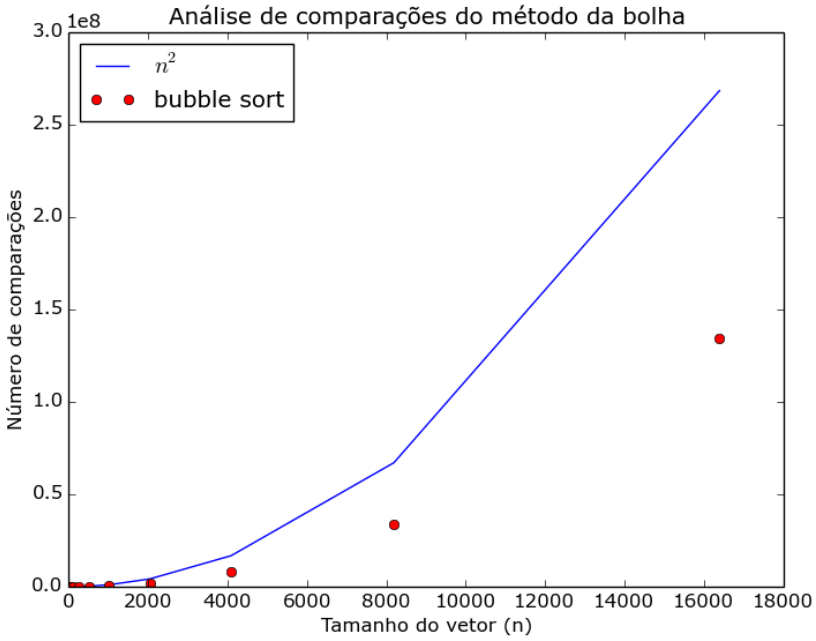

Figura 2.4: *Complexidade de custo do método da bolha (Vetor Ordenado Crescente)*

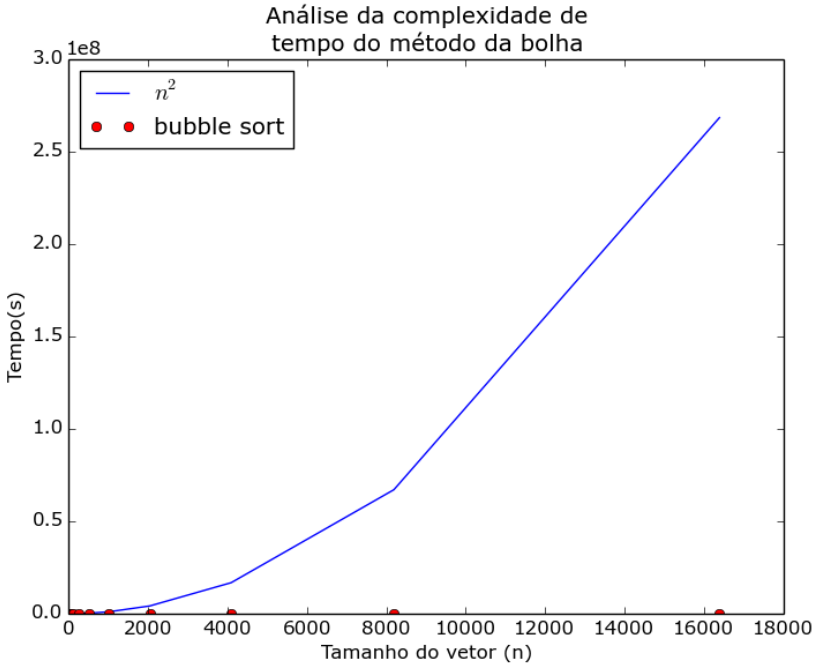

Figura 2.5: *Complexidade de tempo do método da bolha (Vetor Ordenado Crescente)*

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

4. Para um vetor parcialmente ordenado crescente

(a) Complexidade de custo do método da bolha disponível na lista de imagens 2.10.

(b) Complexidade de tempo do método da bolha disponível na lista de imagens 2.11.

2.0

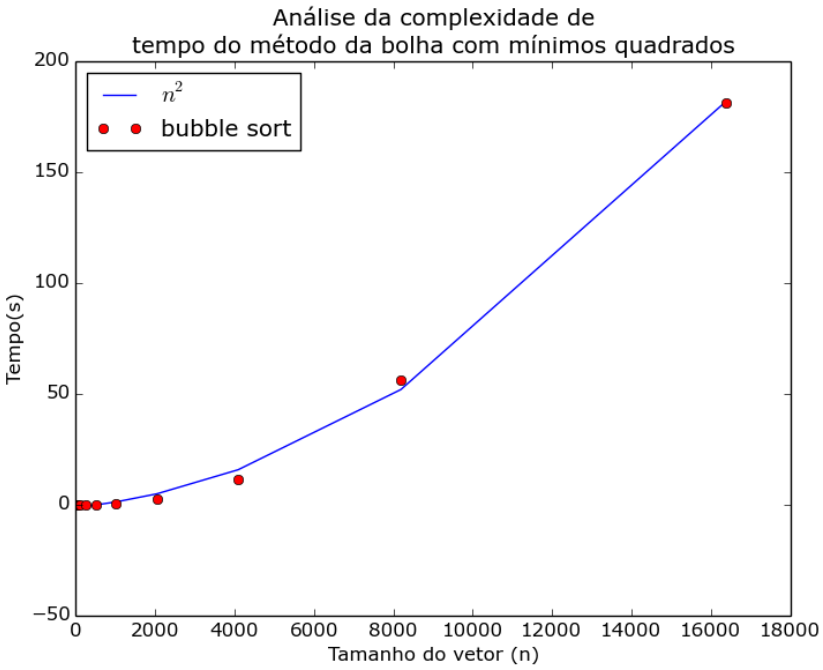

Figura 2.6: *Complexidade de tempo do método da bolha com mínimos quadrados (Vetor Ordenado Crescente)*

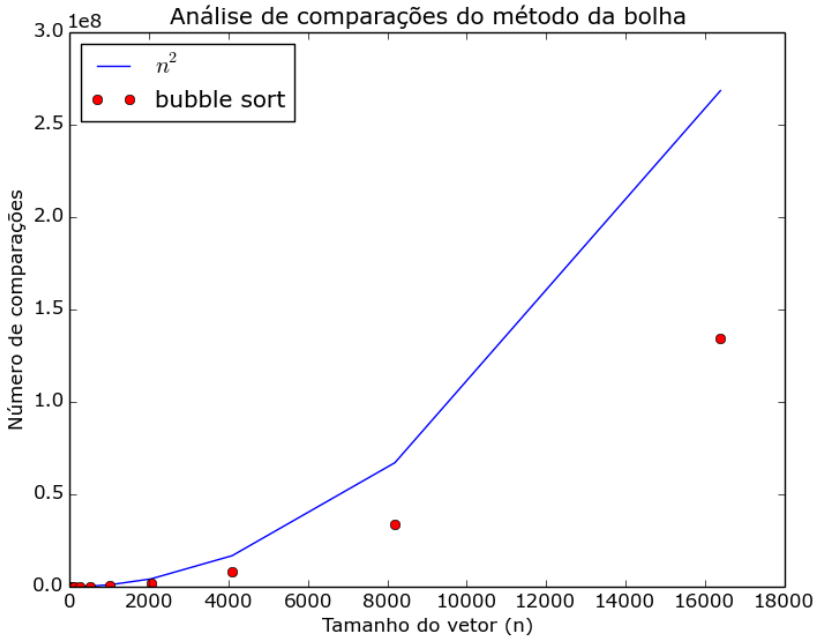

Figura 2.7: *Complexidade de custo do método da bolha (Vetor Ordenado Decrescente)*

(c) Complexidade de tempo do método da bolha com mínimos quadrados disponível na lista de imagens 2.12.

5. Para um vetor parcialmente ordenado decrescente

(a) Complexidade de custo do método da bolha disponível na lista de imagens 2.13.

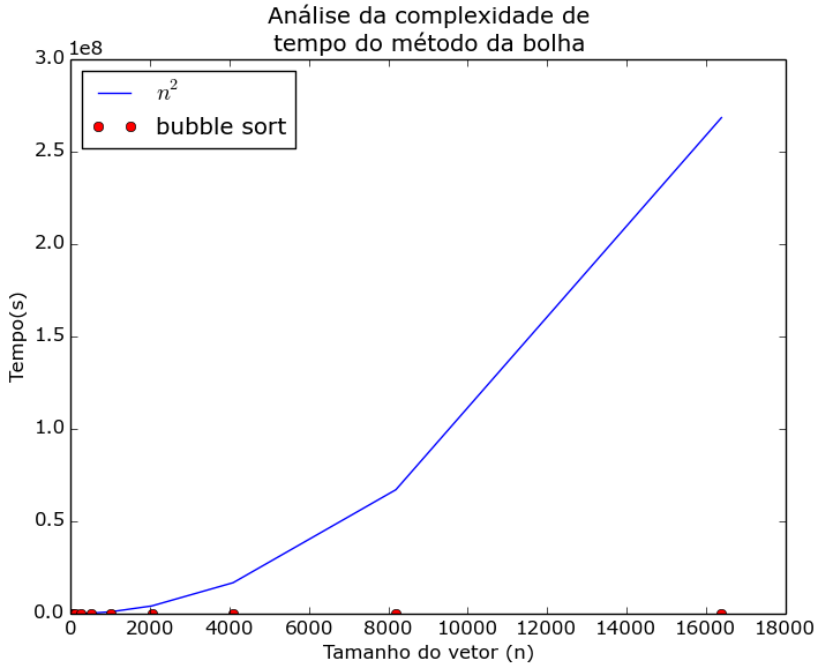

Figura 2.8: *Complexidade de tempo do método da bolha (Vetor Ordenado Decrescente)*

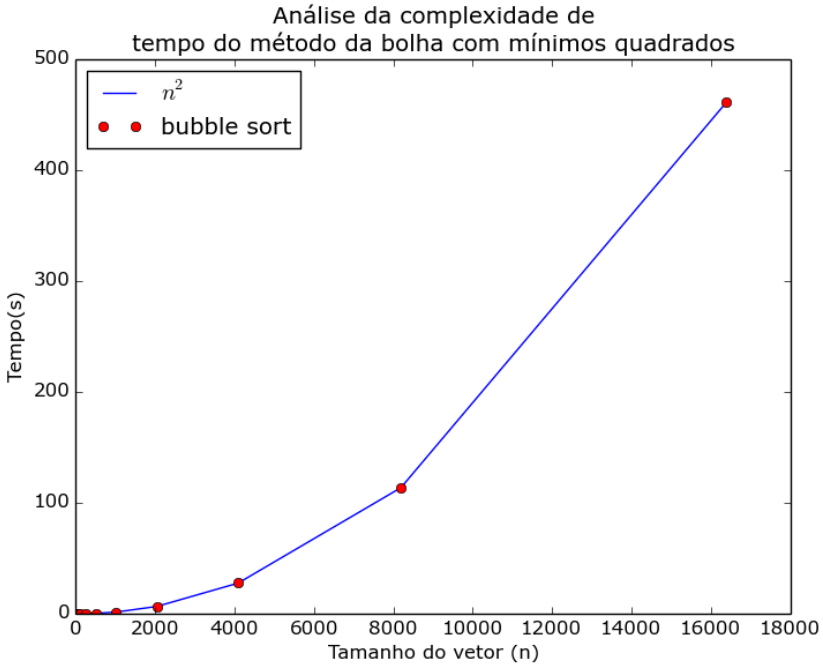

Figura 2.9: *Complexidade de tempo do método da bolha com mínimos quadrados (Vetor Ordenado Decrescente)*

(b) Complexidade de tempo do método da bolha disponível na lista de imagens 2.14.

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

2.0

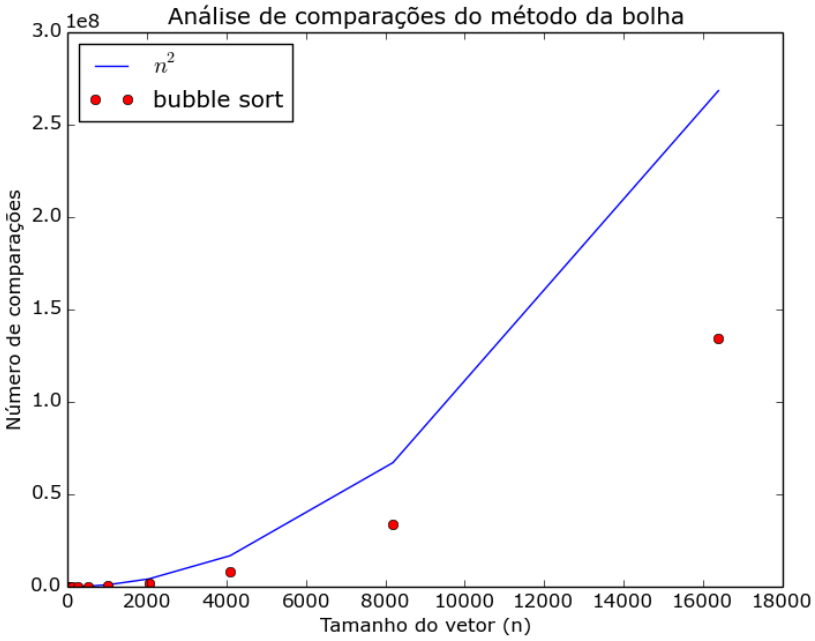

Figura 2.10: *Complexidade de custo do método da bolha (Vetor Parcialmente Ordenado Crescente)*

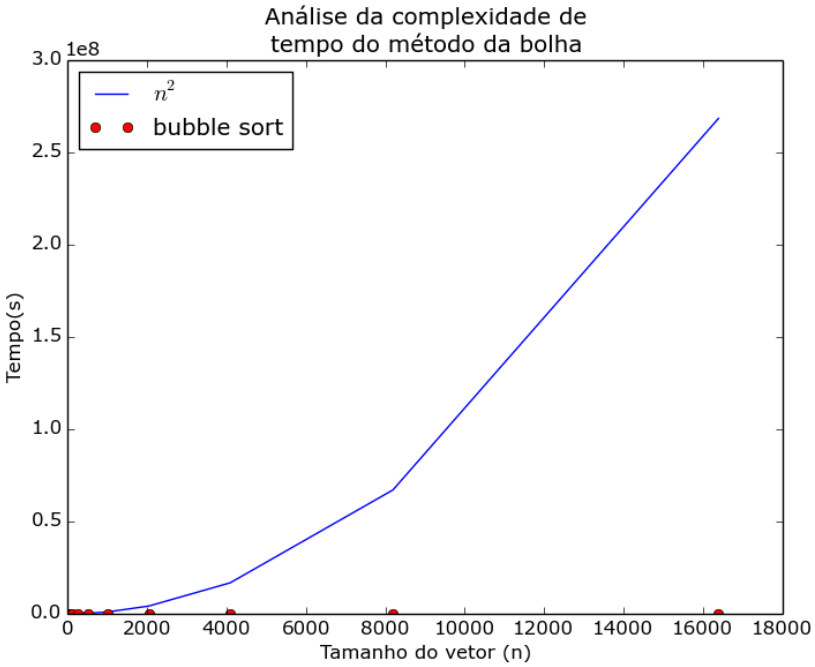

Figura 2.11: *Complexidade de tempo do método da bolha (Vetor Parcialmente Ordenado Crescente)*

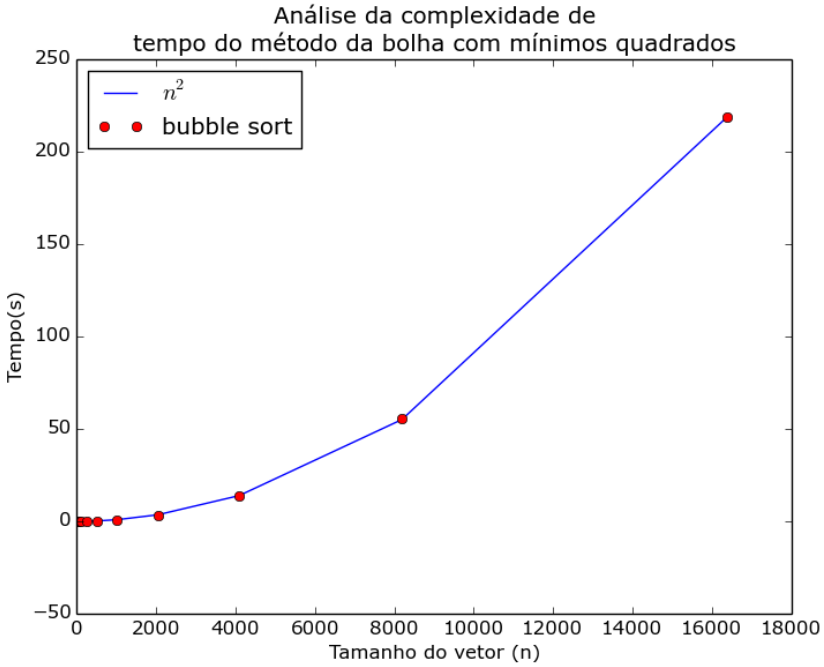

Figura 2.12: *Complexidade de tempo do método da bolha com mínimos quadrados (Vetor Parcialmente Ordenado Crescente)*

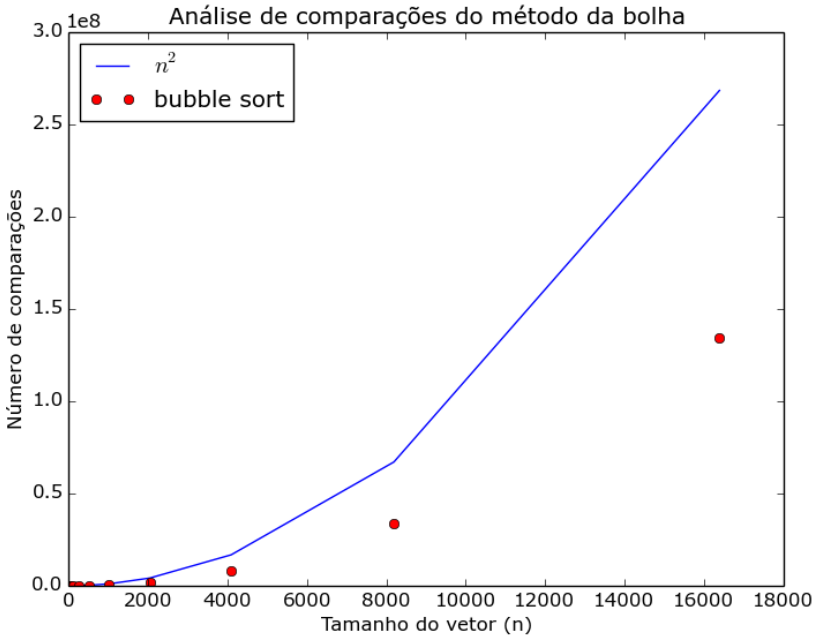

Figura 2.13: *Complexidade de custo do método da bolha (Vetor Parcialmente Ordenado Decrescente)*

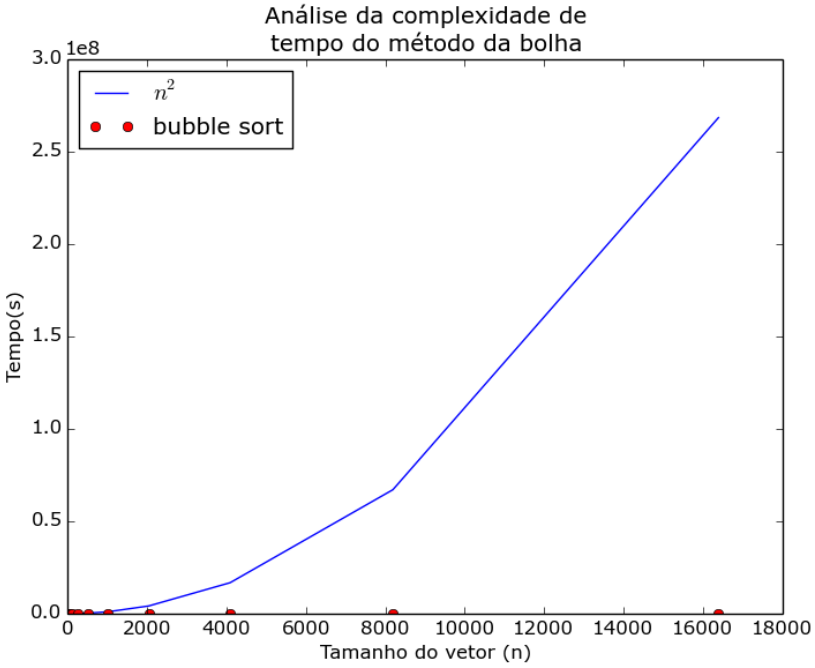

Figura 2.14: *Complexidade de tempo do método da bolha (Vetor Parcialmente Ordenado Decrescente)*

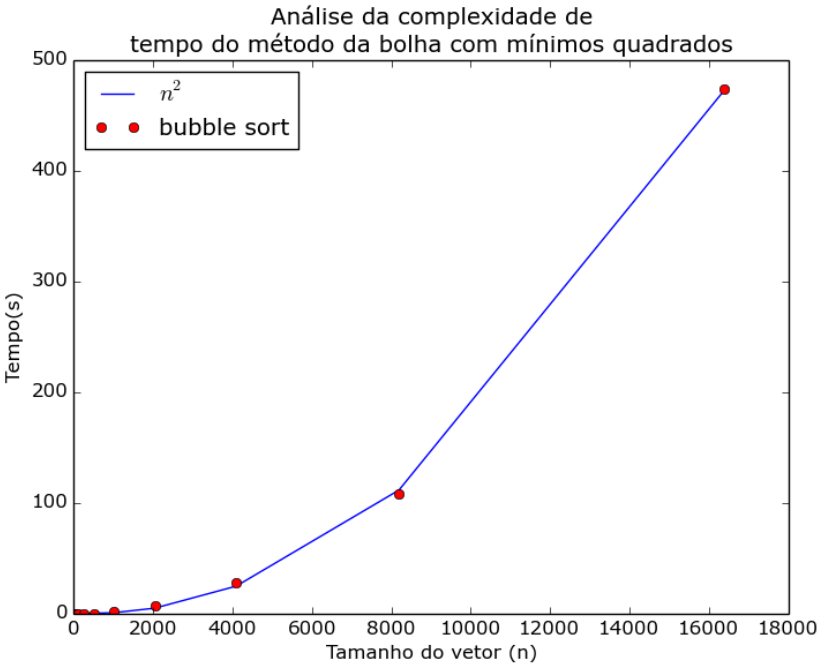

Figura 2.15: *Complexidade de tempo do método da bolha com mínimos quadrados (Vetor Parcialmente Ordenado Decrescente)*

Capítulo 3

Tabelas

Seguem as tabelas utilizadas para a análise do método Bubble 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	496	0.000867
64	2016	0.003210
128	8128	0.013136
256	32640	0.053045
512	130816	0.208341
1024	523776	0.825390
2048	2096128	3.348710
4096	8386560	13.570900
8192	33550336	55.394400
16384	134209536	218.900000

Tabela 3.5: *Vetor Parcialmente Ordenado Decrescente*

Tamanho do Vetor	Comparações	Tempo(s)
32	496	0.001050
64	2016	0.004241
128	8128	0.017106
256	32640	0.067391
512	130816	0.267413
1024	523776	1.584400
2048	2096128	6.835360
4096	8386560	28.157500
8192	33550336	108.491000
16384	134209536	473.634000

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 Bubble 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 Bubble Sort é $O(n^2)$.

Capítulo 5

Citações e referências bibliográficas

Apêndice A

Códigos extensos

A.1 testdriver.py

Listagem A.1: testdriver.py

```
import subprocess
import numpy as np
import matplotlib.pyplot as plt
import sys , shutil


##PRA CADA NOVO METODO TEM QUE MUDAR
#Sys.path()

## PARA CADA VETOR NOVO OU NOVO MÉTODO TEM QUE MUDAR
#Para o executa_teste a chamada das funções e o shutil.move()
#para os plots        a chamada das funções e o savefig

sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
    Codigos/Selection')  ## adicionei o código de ordenação
sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
    relatorio/Resultados/Selection') ## adicionei o resultado do
    executa_teste


def executa_teste(arqteste, arqsaida, nlin, intervalo):
    """Executa uma sequência de testes contidos em arqteste, com:
       arqsaida: nome do arquivo de saída, ex: tBolha.dat
       nlin: número da linha no arquivo gerado pelo line_profiler contendo
             os dados de interesse. Ex: 14
       intervalo: tamanhos dos vetores: Ex: 2 ** np.arange(5,10)
    """
    f = open(arqsaida,mode='w', encoding='utf-8')
    f.write('#      n   comparações      tempo(s)\n')

    for n in intervalo:
        cmd = ' '.join(["kernprof -l -v", "testeGeneric.py", str(n)])
        str_saida = subprocess.check_output(cmd, shell=True).decode('utf-8
            ')
        linhas = str_saida.split('\n')
        unidade_tempo = float(linhas[1].split()[2])
```

```
33          #print("CMD:", cmd, "\nSTR_SAIDA: ",str_saida,"\nLINHAS: ",linhas
                ,"\nUNIDADE_TEMPO: ",unidade_tempo)
34          #print("Linhas4:",linhas[4]," -----> Linhas 4 float: ",linhas[4].
                split()[2])
35          tempo_total = float(linhas[3].split()[2])
36          lcomp = linhas[nlin].split()
37          num_comps = int(lcomp[1])
38          str_res = '{:>8} {:>13} {:13.6f}'.format(n, num_comps, tempo_total
                )
39          print(str_res)
40          f.write(str_res + '\n')
41      f.close()
42      shutil.move("tSelection_vetor_aleatorio.dat", "/home/gmarson/Git/
            AnaliseDeAlgoritmos/Trabalho_Final/relatorio/Resultados/Selection/
            tSelection_vetor_aleatorio.dat")
43
44  #executa_teste("testeGeneric.py", "tSelection_vetor_aleatorio.dat", 14, 2
        ** np.arange(5,15))
45
46  def plota_teste1(arqsaida):
47      n, c, t = np.loadtxt(arqsaida, unpack=True)
48      #print("n: ",n,"\nc: ",c,"\nt: ",t)
49      #n eh o tamanho da entrada , c eh o tanto de comparações e t eh o
            tempo gasto
50      plt.plot(n, n ** 2, label='$n^2$')  ## custo esperado bubble Sort
51      plt.plot(n, c, 'ro', label='selection sort')
52
53      # Posiciona a legenda
54      plt.legend(loc='upper left')
55
56      # Posiciona o título
57      plt.title('Análise de comparações do método da seleção')
58
59      # Rotula os eixos
60      plt.xlabel('Tamanho do vetor (n)')
61      plt.ylabel('Número de comparações')
62
63      plt.savefig('relatorio/imagens/Selection/selection_plot_1_aleatorio.
            png')
64      plt.show()
65
66
67
68  def plota_teste2(arqsaida):
69      n, c, t = np.loadtxt(arqsaida, unpack=True)
70      plt.plot(n, n ** 2, label='$n^2$')
71      plt.plot(n, t, 'ro', label='selection sort')
72
73      # Posiciona a legenda
74      plt.legend(loc='upper left')
75
76      # Posiciona o título
77      plt.title('Análise da complexidade de \ntempo do método da seleção')
78
79      # Rotula os eixos
80      plt.xlabel('Tamanho do vetor (n)')
81      plt.ylabel('Tempo(s)')
82
```

```
83      plt.savefig('relatorio/imagens/Selection/selection_plot_2_aleatorio.
            png')
84      plt.show()
85
86
87
88
89  def plota_teste3(arqsaida):
90      n, c, t = np.loadtxt(arqsaida, unpack=True)
91
92      # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
93      # o método dos mínimos quadrados
94      coefs = np.polyfit(n, t, 2)
95      p = np.poly1d(coefs)
96
97      plt.plot(n, p(n), label='$n^2$')
98      plt.plot(n, t, 'ro', label='selection sort')
99
100     # Posiciona a legenda
101     plt.legend(loc='upper left')
102
103     # Posiciona o título
104     plt.title('Análise da complexidade de \ntempo do método da seleção com
            mínimos quadrados')
105
106     # Rotula os eixos
107     plt.xlabel('Tamanho do vetor (n)')
108     plt.ylabel('Tempo(s)')
109
110     plt.savefig('relatorio/imagens/Selection/selection_plot_3_aleatorio.
            png')
111     plt.show()
112
113 plota_teste1("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Selection/tSelection_vetor_aleatorio.dat")
114 plota_teste2("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Selection/tSelection_vetor_aleatorio.dat")
115 plota_teste3("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Selection/tSelection_vetor_aleatorio.dat")
116
117
118 def plota_teste4(arqsaida):
119     n, c, t = np.loadtxt(arqsaida, unpack=True)
120
121     # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
122     # o método dos mínimos quadrados
123     coefs = np.polyfit(n, c, 2)
124     p = np.poly1d(coefs)
125
126     plt.plot(n, p(n), label='$n^2$')
127     plt.plot(n, c, 'ro', label='bubble sort')
128
129     # Posiciona a legenda
130     plt.legend(loc='upper left')
131
132     # Posiciona o título
133     plt.title('Análise da complexidade de \ntempo do método da bolha')
134
135     # Rotula os eixos
```

```
136     plt.xlabel('Tamanho do vetor (n)')
137     plt.ylabel('Número de comparações')
138 
139     plt.savefig('bubble4.png')
140     plt.show()
141 
142 def plota_teste5(arqsaida):
143     n, c, t = np.loadtxt(arqsaida, unpack=True)
144 
145     # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
146     # o método dos mínimos quadrados
147     coefs = np.polyfit(n, c, 2)
148     p = np.poly1d(coefs)
149 
150     # set_yscale('log')
151     # set_yscale('log')
152     plt.semilogy(n, p(n), label='$n^2$')
153     plt.semilogy(n, c, 'ro', label='bubble sort')
154 
155     # Posiciona a legenda
156     plt.legend(loc='upper left')
157 
158     # Posiciona o título
159     plt.title('Análise da complexidade de \ntempo do método da bolha')
160 
161     # Rotula os eixos
162     plt.xlabel('Tamanho do vetor (n)')
163     plt.ylabel('Número de comparações')
164 
165     plt.savefig('bubble5.png')
166     plt.show()
```