

# Análise de Complexidade de Tempo do Método Merge Sort

Eduardo Costa de Paiva

[eduardocspv@gmail.com](mailto:eduardocspv@gmail.com)

Frederico Franco Calhau

[fredericoffc@gmail.com](mailto:fredericoffc@gmail.com)

Gabriel Augusto Marson

[gabrielmarson@live.com](mailto:gabrielmarson@live.com)

Faculdade de Computação  
Universidade Federal de Uberlândia

16 de dezembro de 2015

# Lista de Figuras

|      |  |    |
|------|--|----|
| 2.1  | Complexidade de custo do método Merge Sort (Vetor Aleatório) . . . . .   | 11 |
| 2.2  | Complexidade de tempo do método Merge Sort (Vetor Aleatório) . . . . .   | 12 |
| 2.3  | Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Aleatório) . . . . .                         | 12 |
| 2.4  | Complexidade de custo do método Merge Sort (Vetor Ordenado Crescente) .  | 13 |
| 2.5  | Complexidade de tempo do método Merge Sort (Vetor Ordenado Crescente)  | 13 |
| 2.6  | Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Ordenado Crescente) . . . . .                | 14 |
| 2.7  | Complexidade de custo do método Merge Sort (Vetor Ordenado Decrescente)  | 14 |
| 2.8  | Complexidade de tempo do método Merge Sort (Vetor Ordenado Decrescente)  | 15 |
| 2.9  | Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Ordenado Decrescente) . . . . .              | 15 |
| 2.10 | Complexidade de custo do método Merge Sort (Vetor Parcialmente Ordenado Crescente) . . . . .                         | 16 |
| 2.11 | Complexidade de tempo do método Merge Sort (Vetor Parcialmente Ordenado Crescente) . . . . .                         | 16 |
| 2.12 | Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Parcialmente Ordenado Crescente) . . . . .   | 17 |
| 2.13 | Complexidade de custo do método Merge Sort (Vetor Parcialmente Ordenado Decrescente) . . . . .                       | 17 |
| 2.14 | Complexidade de tempo do método Merge Sort (Vetor Parcialmente Ordenado Decrescente) . . . . .                       | 18 |
| 2.15 | Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Parcialmente Ordenado Decrescente) . . . . . | 18 |

# Lista de Tabelas

|     |   |    |
|-----|---|----|
| 3.1 | Vetor Aleatorio . . . . .                         | 19 |
| 3.2 | Vetor Ordenado Crescente . . . . .                | 20 |
| 3.3 | Vetor Ordenado Decrescente . . . . .              | 20 |
| 3.4 | Vetor Parcialmente Ordenado Crescente . . . . .   | 21 |
| 3.5 | Vetor Parcialmente Ordenado Decrescente . . . . . | 21 |

# Lista de Listagens

|     |                 |    |
|-----|-----------------|----|
| 1.1 | MergeSort.py    | 7  |
| 1.2 | testeGeneric.py | 8  |
| 1.3 | monitor.py      | 9  |
| A.1 | testdriver.py   | 24 |

# Sumário

|  |           |
|--|-----------|
| <b>Lista de Figuras</b>                        | <b>2</b>  |
| <b>Lista de Tabelas</b>                        | <b>3</b>  |
| <b>1 Introdução</b>                            | <b>6</b>  |
| 1.1 Diretório . . . . .                        | 6         |
| 1.2 Códigos de programas . . . . .             | 7         |
| <b>2 Gráficos</b>                              | <b>11</b> |
| <b>3 Tabelas</b>                               | <b>19</b> |
| <b>4 Análise</b>                               | <b>22</b> |
| <b>5 Citações e referências bibliográficas</b> | <b>23</b> |
| <b>Apêndice</b>                                | <b>24</b> |
| <b>A Códigos extensos</b>                      | <b>24</b> |
| A.1 testdriver.py . . . . .                    | 24        |

# 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 raiz:

```
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
41 #basta
42 #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.
3      PATH
4
5  import sys
6  sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final
7      /Codigos/Merge')
8  sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final
9      ')
10
11  from monitor import *
12  from memoria import *
13
14  from MergeSort import *
15  import argparse
16
17  parser = argparse.ArgumentParser()
18  parser.add_argument("n", type=int, help="número de elementos no vetor
19      de teste")
20  args = parser.parse_args()
21
22  v = criavet(args.n)
23  merge(v)
24
25  ## A EXECUÇÃO DESSE ARQUIVO EH ASSIM

```



```

24 ## NA LINHA DE COMANDO VC MANDA O NOME DO ARQUIVO E O TAMANHO DO
    ELEMNT0 DO vetor
25 ##EXEMPLO testeBubble.py 10
26 ##ele gera um vetor aleatório (criavet) e manda pro bubble_sort

```

---

### 3. monitor.py Disponível na Listagem 1.3

Listagem 1.3: monitor.py

```

1 # Para instalar o Python 3 no Ubuntu 14 ou 15
2 #
3 # sudo apt-get install python3 python3-numpy python3-matplotlib
    ipython3 python3-psutil
4 #
5
6 from math import *
7 import gc
8 import random
9 import numpy as np
10
11
12 from tempo import *
13
14 # Vetores de teste
15 def troca(m,v,n): ## seleciona o nível de embaralhamento do vetor
16     m = trunc(m)
17     mi = (n-m)//2
18     mf = (n+m)//2
19     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 decrescente                -1
55 #parcialmente ordenado crescente     0.5
56 #parcialmente ordenado decrescente  -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](#).

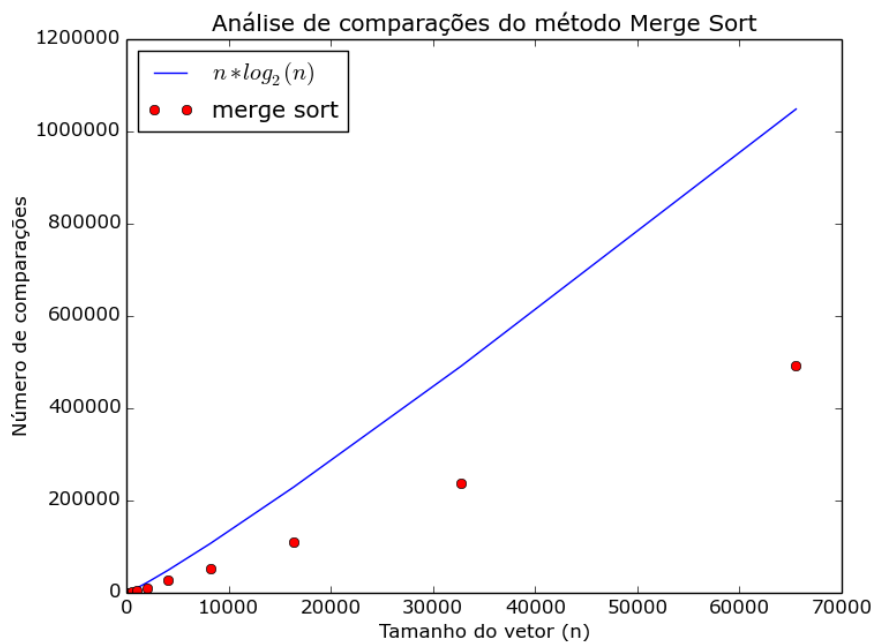
# Capítulo 2

## 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](#).

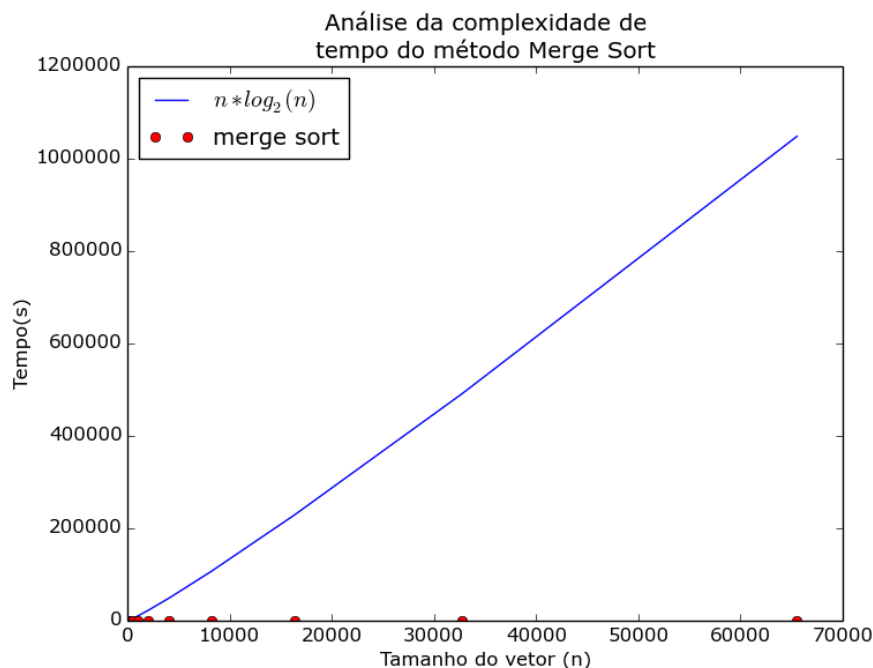


**Figura 2.1:** *Complexidade de custo do método Merge Sort (Vetor Aleatório)*

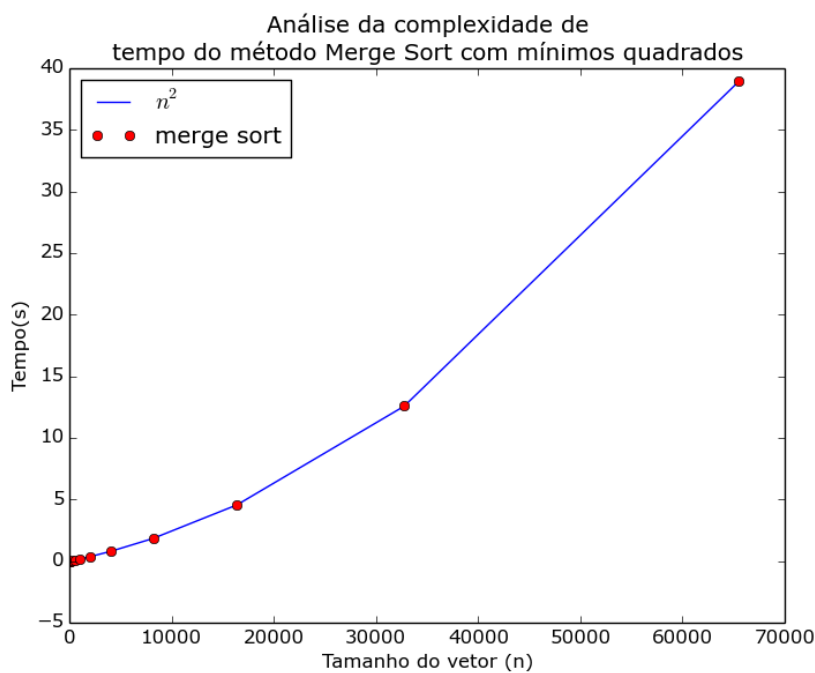
(b) Complexidade de tempo do método Merge Sort disponível na lista de imagens [2.2](#).

(c) Complexidade de tempo do método Merge Sort com mínimos quadrados disponível na lista de imagens [2.3](#).

2. Para um vetor ordenado crescente

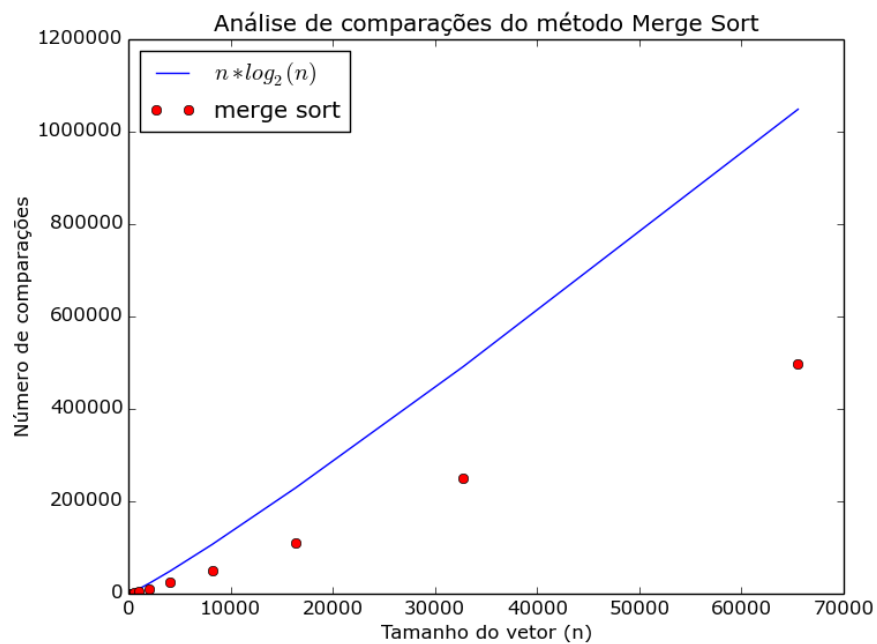


**Figura 2.2:** Complexidade de tempo do método Merge Sort (Vetor Aleatório)

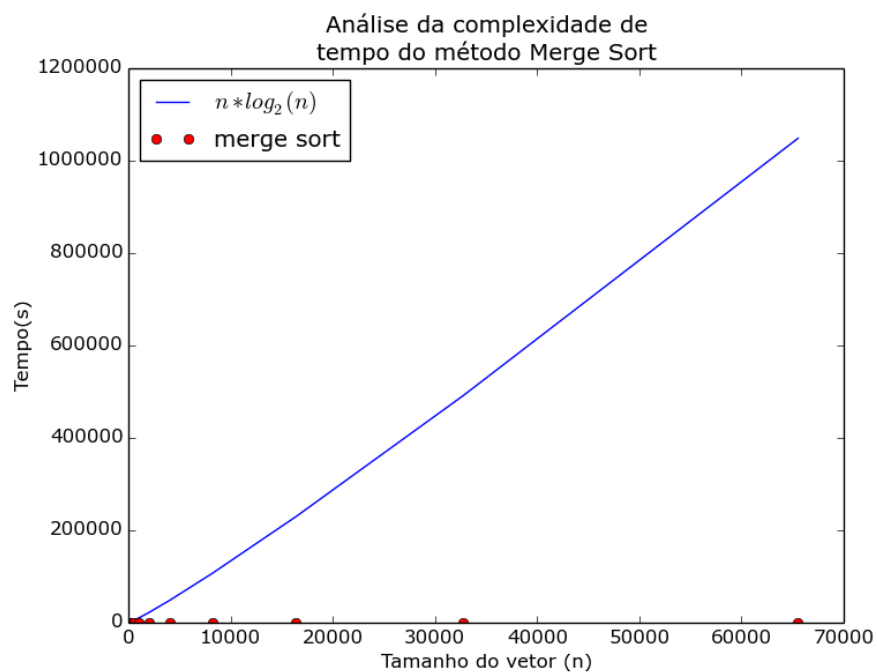


**Figura 2.3:** Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Aleatório)

- (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.



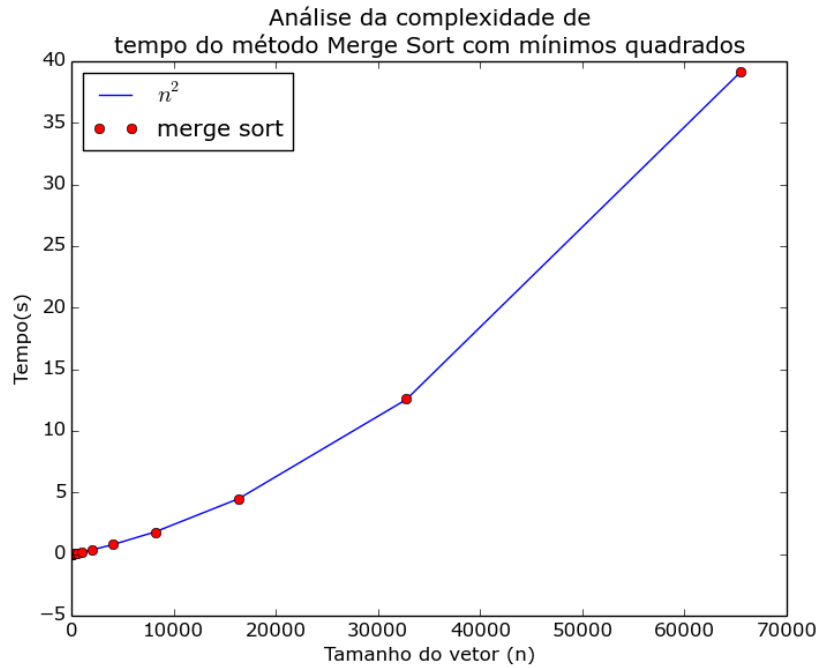
**Figura 2.4:** Complexidade de custo do método Merge Sort (Vetor Ordenado Crescente)



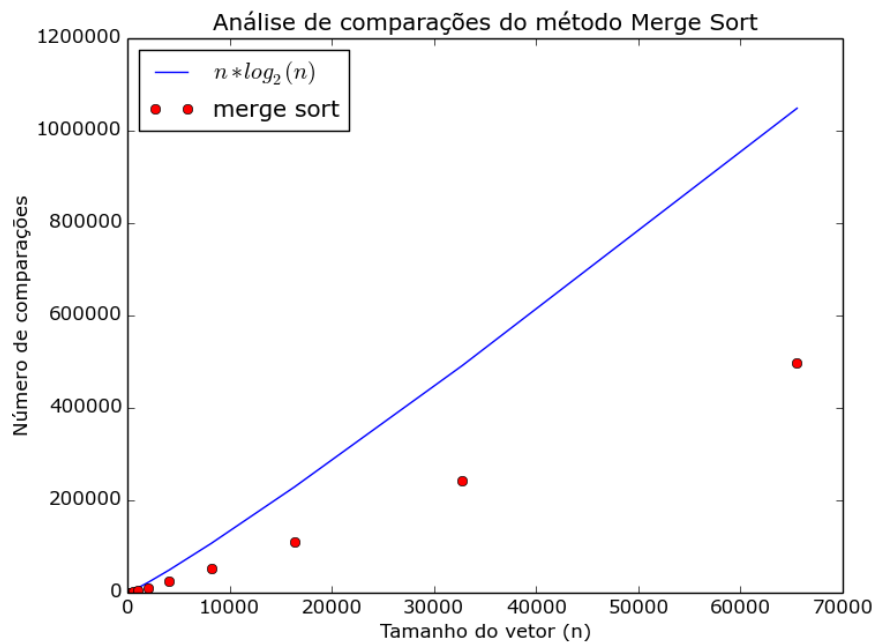
**Figura 2.5:** Complexidade de tempo do método Merge Sort (Vetor Ordenado Crescente)

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](#).



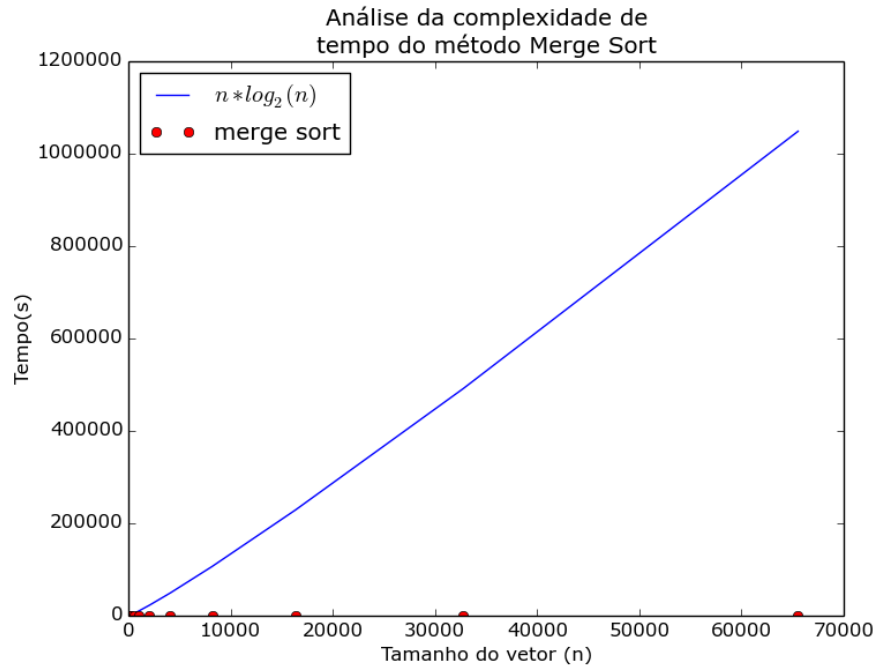
**Figura 2.6:** Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Ordenado Crescente)



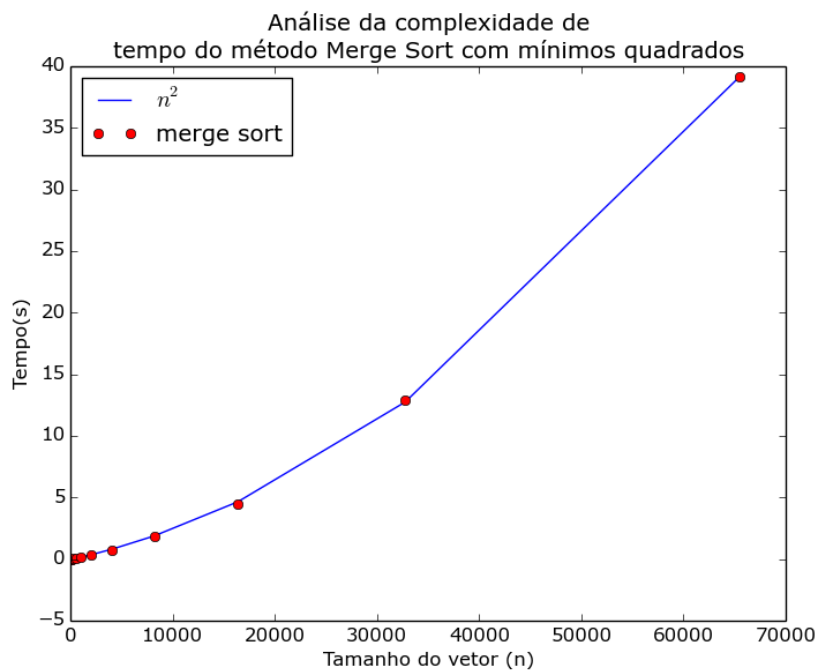
**Figura 2.7:** Complexidade de custo do método Merge Sort (Vetor Ordenado Decrescente)

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)

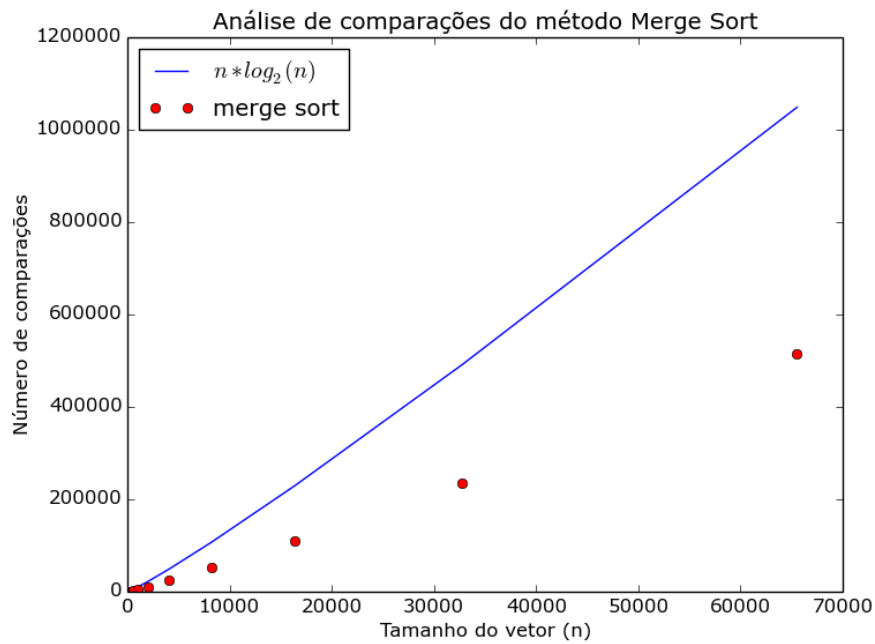


**Figura 2.9:** Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Ordenado Decrescente)

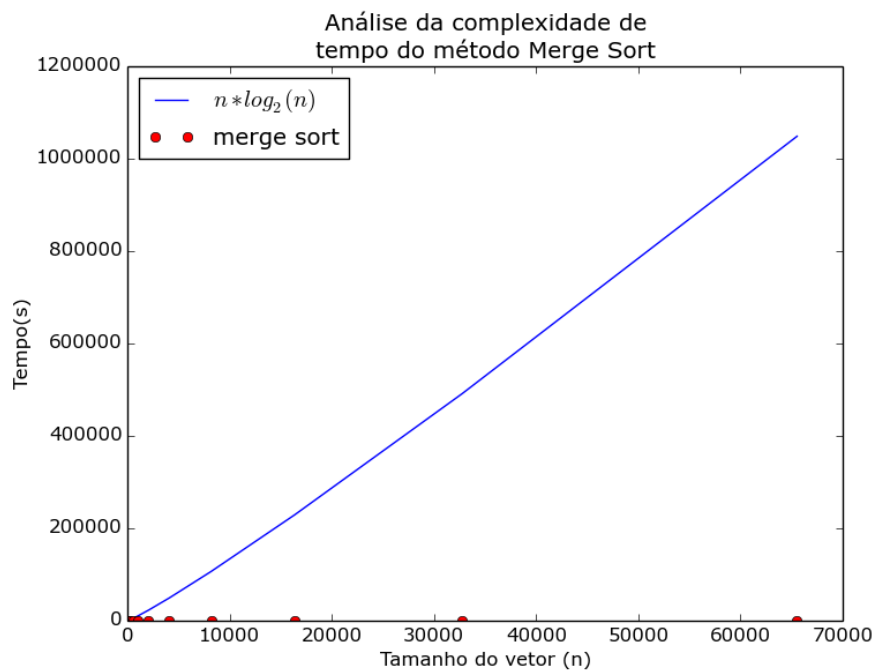
nível na lista de imagens [2.12](#).

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)

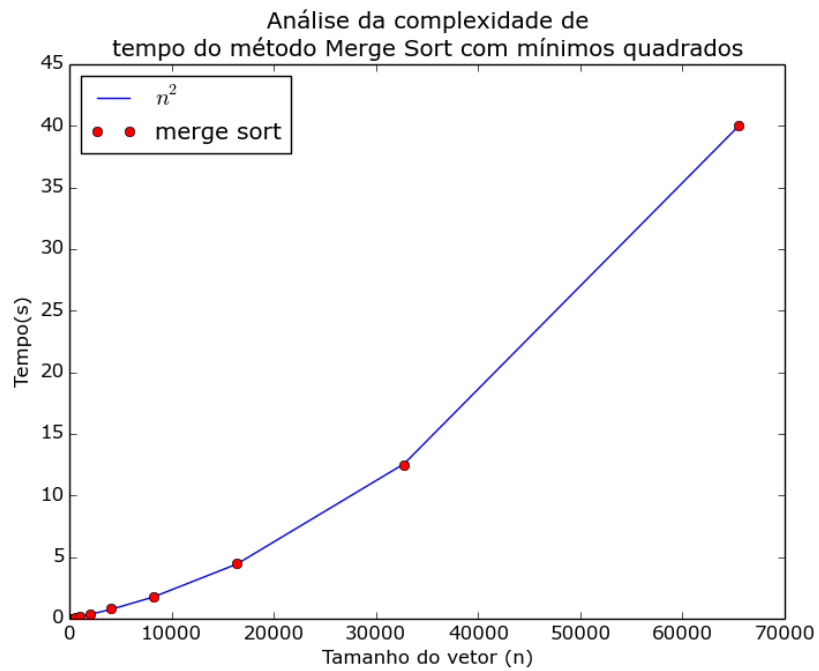


**Figura 2.11:** Complexidade de tempo do método Merge Sort (Vetor Parcialmente Ordenado Crescente)

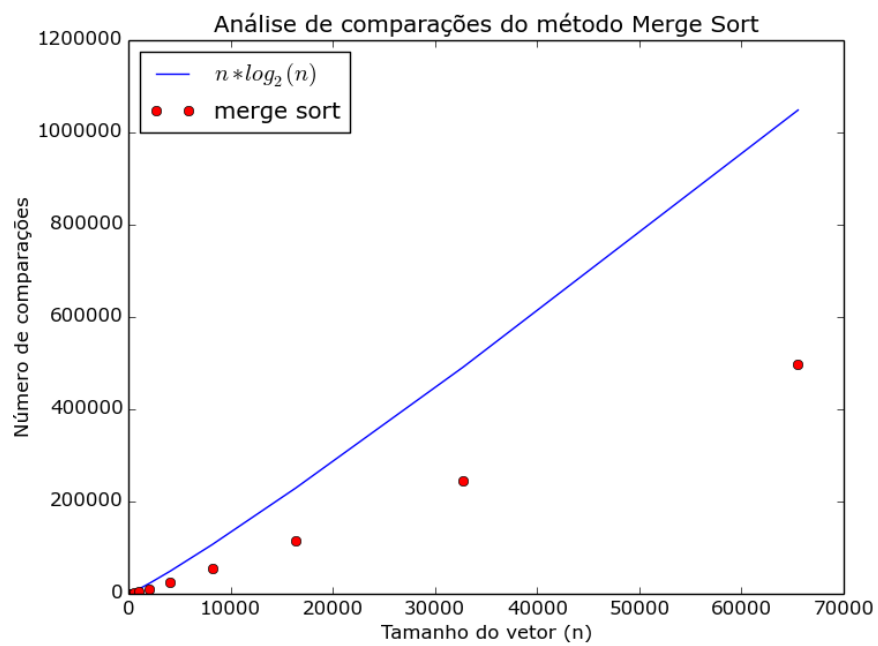
2.14.

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

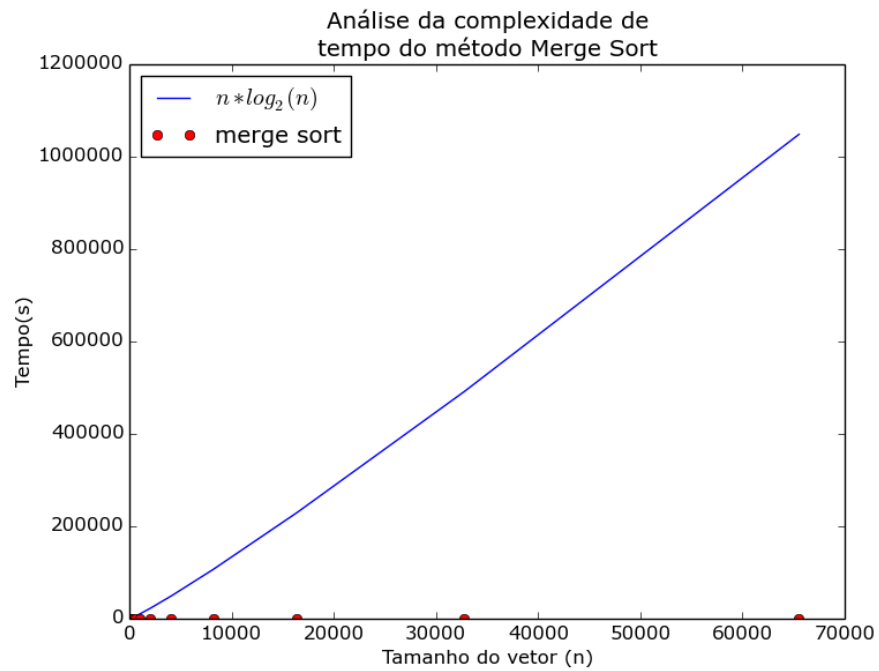




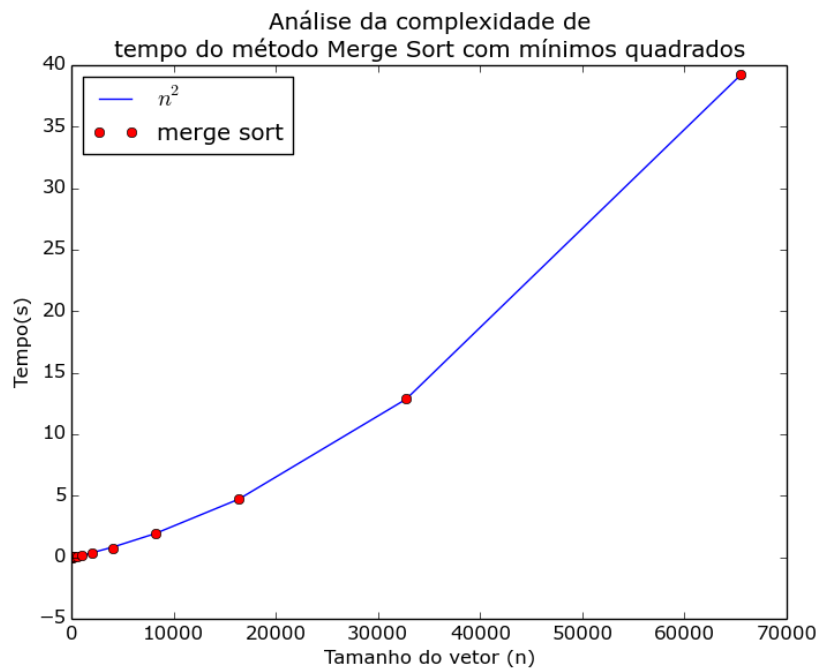
**Figura 2.12:** Complexidade de tempo do método Merge Sort com mínimos quadrados (Vetor Parcialmente Ordenado Crescente)



**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               | 95          | 0.002739  |
| 64               | 219         | 0.006053  |
| 128              | 475         | 0.013715  |
| 256              | 1203        | 0.033465  |
| 512              | 2334        | 0.067288  |
| 1024             | 5539        | 0.162378  |
| 2048             | 11118       | 0.331024  |
| 4096             | 26759       | 0.837335  |
| 8192             | 51853       | 1.817348  |
| 16384            | 110392      | 4.521457  |
| 32768            | 237663      | 12.608580 |
| 65536            | 491980      | 39.949010 |

**Tabela 3.2:** *Vetor Ordenado Crescente*

| <b>Tamanho do Vetor</b> | <b>Comparações</b> | <b>Tempo(s)</b> |
|-------------------------|--------------------|-----------------|
| 32                      | 94                 | 0.002563        |
| 64                      | 239                | 0.006553        |
| 128                     | 476                | 0.013587        |
| 256                     | 1039               | 0.030059        |
| 512                     | 2293               | 0.066084        |
| 1024                    | 5084               | 0.147069        |
| 2048                    | 11228              | 0.331113        |
| 4096                    | 24201              | 0.766142        |
| 8192                    | 51017              | 1.773804        |
| 16384                   | 108998             | 4.481584        |
| 32768                   | 251194             | 12.579250       |
| 65536                   | 497903             | 39.168110       |

**Tabela 3.3:** *Vetor Ordenado Decrescente*

| <b>Tamanho do Vetor</b> | <b>Comparações</b> | <b>Tempo(s)</b> |
|-------------------------|--------------------|-----------------|
| 32                      | 95                 | 0.002664        |
| 64                      | 231                | 0.006120        |
| 128                     | 500                | 0.013601        |
| 256                     | 1112               | 0.031692        |
| 512                     | 2386               | 0.069012        |
| 1024                    | 5132               | 0.149191        |
| 2048                    | 10950              | 0.327610        |
| 4096                    | 23895              | 0.753161        |
| 8192                    | 52772              | 1.847495        |
| 16384                   | 109071             | 4.501062        |
| 32768                   | 242107             | 12.890980       |
| 65536                   | 497543             | 39.163110       |

**Tabela 3.4:** *Vetor Parcialmente Ordenado Crescente*

| Tamanho do Vetor | Comparações | Tempo(s)  |
|------------------|-------------|-----------|
| 32               | 92          | 0.002681  |
| 64               | 188         | 0.006145  |
| 128              | 485         | 0.013650  |
| 256              | 1086        | 0.031333  |
| 512              | 2314        | 0.067024  |
| 1024             | 5056        | 0.150097  |
| 2048             | 11063       | 0.329353  |
| 4096             | 24663       | 0.797015  |
| 8192             | 52275       | 1.806541  |
| 16384            | 110549      | 4.538953  |
| 32768            | 234740      | 12.476440 |
| 65536            | 514762      | 40.043390 |

**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')
```



## A.1

```
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[
41         tempo[1]].split()[2]) + float(linhas[tempo[2]].split()[2])
42     unidade_tempo = float(linhas[3].split()[2])
43     lcomp = int(linhas[nlin].split()[2]) + int(linhas[nlin2].split(
44         [2])
45
46     #print (linhas[nlin].split()[2])
47
48     #print (linhas[nlin2].split()[2])
49     #print (linhas[nlin3].split()[2])
50
51     #print ("unidade tempo: ",unidade_tempo )
52     #print ("lcomp: ",lcomp)
53     #print ("tempo total",tempo_total)
54
55     #num_comps = int(lcomp[1])
56     str_res = '{:>8} {:>13} {:.13.6f}'.format(n, lcomp ,tempo_total)
57     print(str_res)
58     f.write(str_res + '\n')
59     lcomp = 0
60     f.close()
61     shutil.move("tMerge_vetor_parcialmente_ordenado_decrescente.dat", "
62         home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/relatorio/
63         Resultados/Merge/tMerge_vetor_parcialmente_ordenado_decrescente.dat
64         ")
65
66 executa_teste("testeGeneric.py", "
67     tMerge_vetor_parcialmente_ordenado_decrescente.dat", 2 ** np.arange
68     (5,17))
69
70 def executa_teste_memoria(arqteste, arqsaida, nlin, intervalo):
71     """Executa uma sequência de testes contidos em arqteste, com:
72     arqsaida: nome do arquivo de saída, ex: tBolha.dat
73     nlin: número da linha no arquivo gerado pelo line_profiler contendo
74     os dados de interesse. Ex: 14
75     intervalo: tamanhos dos vetores: Ex: 2 ** np.arange(5,10)
76     """
77     f = open(arqsaida,mode='w', encoding='utf-8')
78     f.write('#          n    comparações          tempo(s)\n')
79
80     for n in intervalo:
81         cmd = ' '.join(["kernprof -l -v ", "testeGeneric.py", str(n)])
82
83         str_saida = subprocess.check_output(cmd, shell=True).decode('utf-8
84             ')
85
86         linhas = str_saida.split('\n')
87         for i in linhas:
88             print(i)
89
90     print ("Linhas:",linhas[1])
```

```

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

```

## A.1

```
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.polyld(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
152             mínimos quadrados')
153
154     # Rotula os eixos
155     plt.xlabel('Tamanho do vetor (n)')
156     plt.ylabel('Tempo(s)')
157
158     plt.savefig('relatorio/imagens/Merge/
159             merge_plot_3_parcialmente_ordenado_decrescente.png')
160     plt.show()
161
162 plota_teste1("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
163     relatorio/Resultados/Merge/
164     tMerge_vetor_parcialmente_ordenado_decrescente.dat")
165 plota_teste2("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
166     relatorio/Resultados/Merge/
167     tMerge_vetor_parcialmente_ordenado_decrescente.dat")
168 plota_teste3("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
169     relatorio/Resultados/Merge/
170     tMerge_vetor_parcialmente_ordenado_decrescente.dat")
171
172
173 def plota_teste4(arqsaida):
174     n, c, t = np.loadtxt(arqsaida, unpack=True)
175
176     # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
177     # o método dos mínimos quadrados
178     coefs = np.polyfit(n, c, 2)
179     p = np.polyld(coefs)
180
181     plt.plot(n, p(n), label='$n^2$')
182     plt.plot(n, c, 'ro', label='bubble sort')
183
184     # Posiciona a legenda
185     plt.legend(loc='upper left')
186
187     # Posiciona o título
188     plt.title('Análise da complexidade de \ntempo do método da bolha')
189
190     # Rotula os eixos
191     plt.xlabel('Tamanho do vetor (n)')
192     plt.ylabel('Número de comparações')
193
194     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()
```

---