

# Análise de Complexidade de Tempo do Método Radix 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

17 de dezembro de 2015

# Lista de Figuras

2.1	Complexidade de tempo do método Radix Sort (Vetor Aleatório) . . . . .	11
2.2	Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Aleatório) . . . . .	12
2.3	Complexidade de tempo do método Radix Sort (Vetor Ordenado Crescente)	12
2.4	Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Ordenado Crescente) . . . . .	13
2.5	Complexidade de tempo do método Radix Sort (Vetor Ordenado Decrescente)	13
2.6	Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Ordenado Decrescente) . . . . .	14
2.7	Complexidade de tempo do método Radix Sort (Vetor Parcialmente Ordenado Crescente) . . . . .	14
2.8	Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Parcialmente Ordenado Crescente) . . . . .	15
2.9	Complexidade de tempo do método Radix Sort (Vetor Parcialmente Ordenado Decrescente) . . . . .	15
2.10	Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Parcialmente Ordenado Decrescente) . . . . .	16

# Lista de Tabelas

3.1	Vetor Aleatório . . . . .	17
3.2	Vetor Ordenado Crescente . . . . .	17
3.3	Vetor Ordenado Decrescente . . . . .	18
3.4	Vetor Parcialmente Ordenado Decrescente . . . . .	18
3.5	Vetor Parcialmente Ordenado Crescente . . . . .	18

# Lista de Listagens

1.1	RadixSort.py	7
1.2	testeGeneric.py	8
1.3	monitor.py	9
A.1	testdriver.py	21

# 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>17</b>
<b>4 Análise</b>	<b>19</b>
<b>5 Citações e referências bibliográficas</b>	<b>20</b>
<b>Apêndice</b>	<b>21</b>
<b>A Códigos extensos</b>	<b>21</b>
A.1 testdriver.py . . . . .	21

# Capítulo 1

## Introdução

Este documento foi feito com o intuito de exibir uma análise do algoritmo Radix 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__
|   |-- Radix
|   |   `-- __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_Radix
|-- Relatorio_Selection
|-- Resultados
|   |-- Bubble
|   |-- Bucket
|   |-- Counting
|   |-- Heap
|   |-- Insertion
|   |-- Merge
|   |-- Quick
|   |-- Radix
|   |-- Selection

```

51 directories

## 1.2 Códigos de programas

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

### 1. RadixSort.py: Disponível na Listagem 1.1.

Listagem 1.1: RadixSort.py

```

1
2 @profile
3 def radix(A):
4     A = [ int(x) for x in A ]
5     n = len(A)
6     maior = max(A)
7     d = _contaDigitos(maior)
8     radixSort(A, n, d)
9
10 @profile

```

```

11 def radixSort(A, n, d):
12     exp = 1
13     maior = max(A)
14     while exp < maior:
15         _countingSort(A, exp)
16         exp *= 10
17
18
19 @profile
20 # função countingsort adaptada
21 def _countingSort(A, k):
22     contador = [0] * 10 # Contador é o histograma
23     B = [0] * len(A)
24     n = len(A)
25     for i in range(0, n):
26         contador[(A[i] // k) % 10] += 1
27
28     for i in range(1, len(contador)):
29         contador[i] += contador[i - 1]
30
31     for j in range((n - 1), -1, -1):
32         B[contador[(A[j] // k) % 10] - 1] = A[j]
33         contador[(A[j] // k) % 10] -= 1
34
35     for i in range(0, n):
36         A[i] = B[i]
37
38 @profile
39 def _contaDigitos(valor):
40     digitos = 0
41     while (valor != 0):
42         digitos += 1
43         valor //= 10
44     return digitos
45
46
47
48
49
50
51
52
53 #lista = [170, 45, 75, 90, 802, 24, 2, 66]
54 #radix(lista)
55 #print(lista)

```

---

## 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/Radix')
6 sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final
   ')

```



```

7
8 from monitor import *
9 from memoria import *
10
11 from RadixSort 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 radix(v)
20
21
22
23 ## A EXECUÇÃO DESSE ARQUIVO EH ASSIM
24 ## NA LINHA DE COMANDO VC MANDA O NOME DO ARQUIVO E O TAMANHO DO
    ELEMNTTO 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=-1, 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](#).

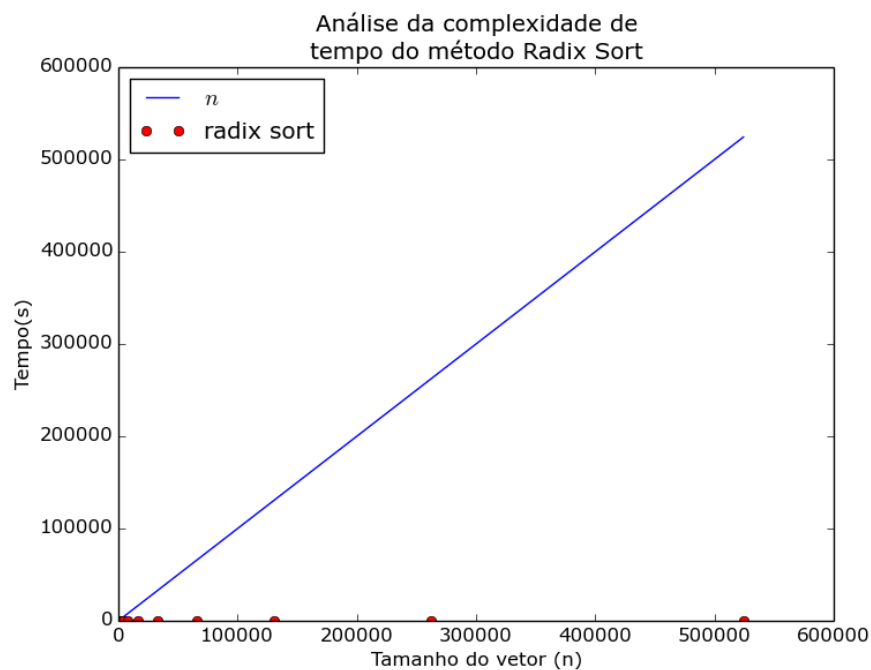
# Capítulo 2

## Gráficos

Seguem os Gráficos utilizadas no processo de análise do método Radix Sort: OBS.: Como o método Radix Sort não realiza comparações, não foi possível listar o gráfico de comparações.

### 1. Para um vetor aleatório

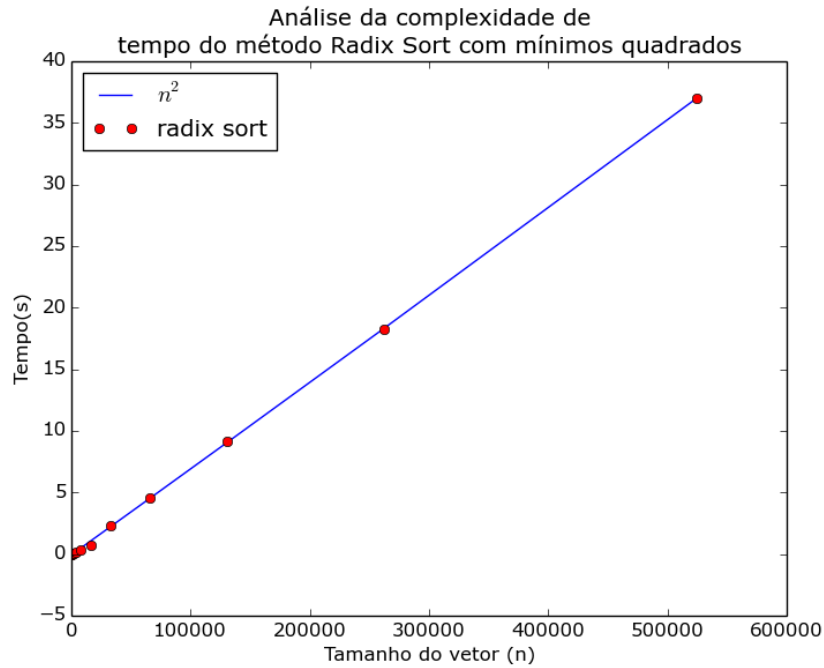
- (a) Complexidade de tempo do método Radix Sort disponível na lista de imagens [2.1](#).



**Figura 2.1:** *Complexidade de tempo do método Radix Sort (Vetor Aleatório)*

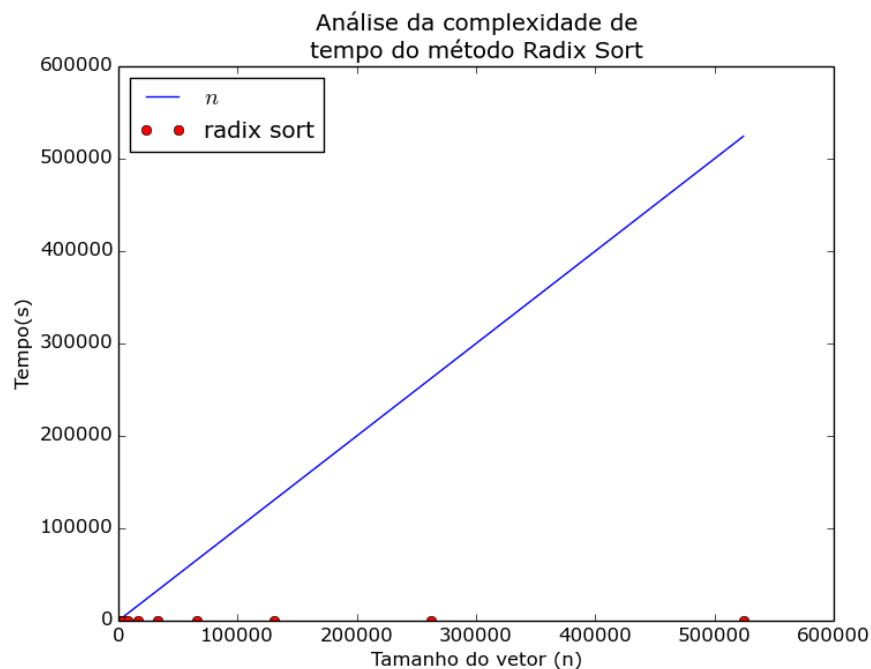
- (b) Complexidade de tempo do método Radix Sort com mínimos quadrados disponível na lista de imagens [2.2](#).

### 2. Para um vetor ordenado crescente



**Figura 2.2:** Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Aleatório)

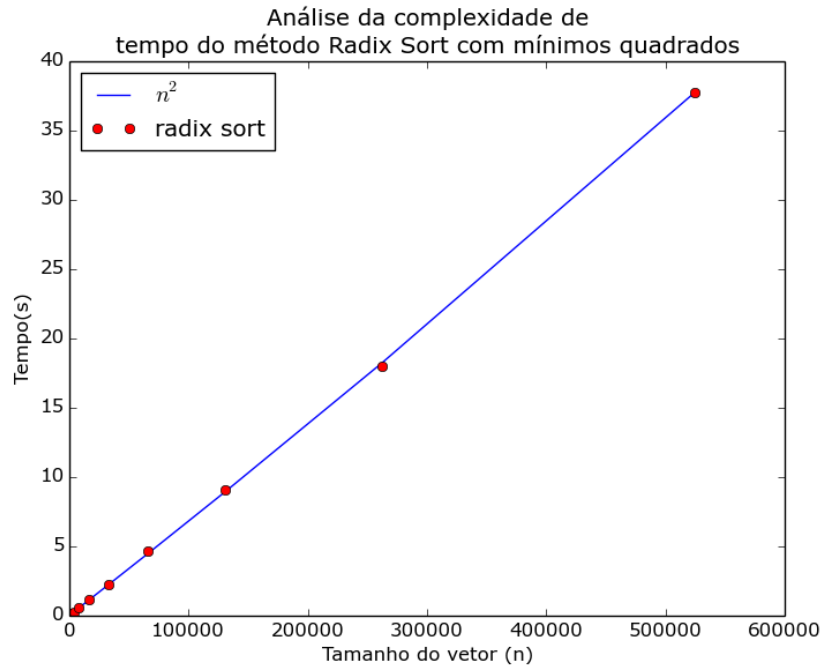
- (a) Complexidade de tempo do método Radix Sort disponível na lista de imagens 2.3.



**Figura 2.3:** Complexidade de tempo do método Radix Sort (Vetor Ordenado Crescente)

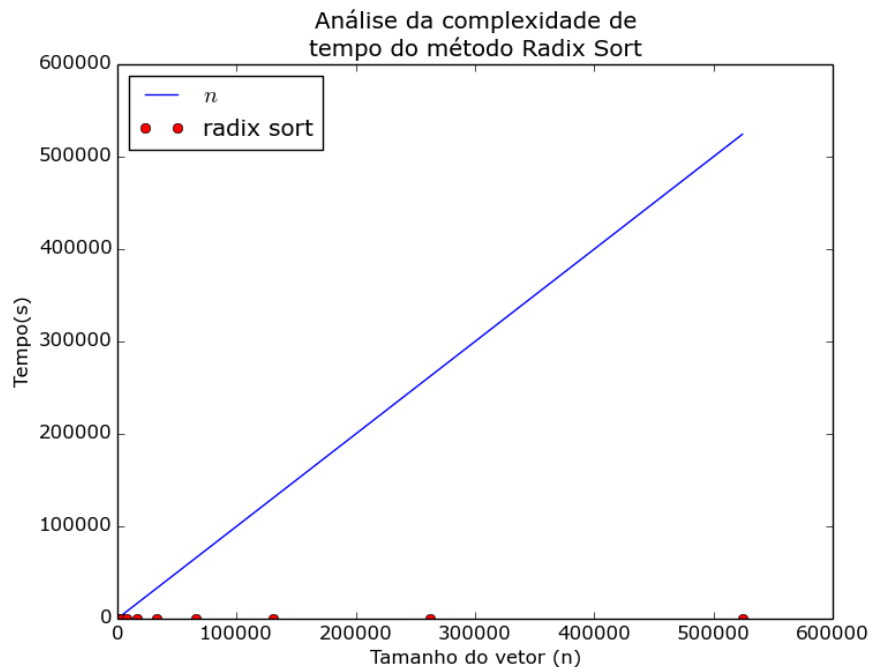
- (b) Complexidade de tempo do método Radix Sort com mínimos quadrados disponível na lista de imagens 2.4.

3. Para um vetor ordenado decrescente



**Figura 2.4:** Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Ordenado Crescente)

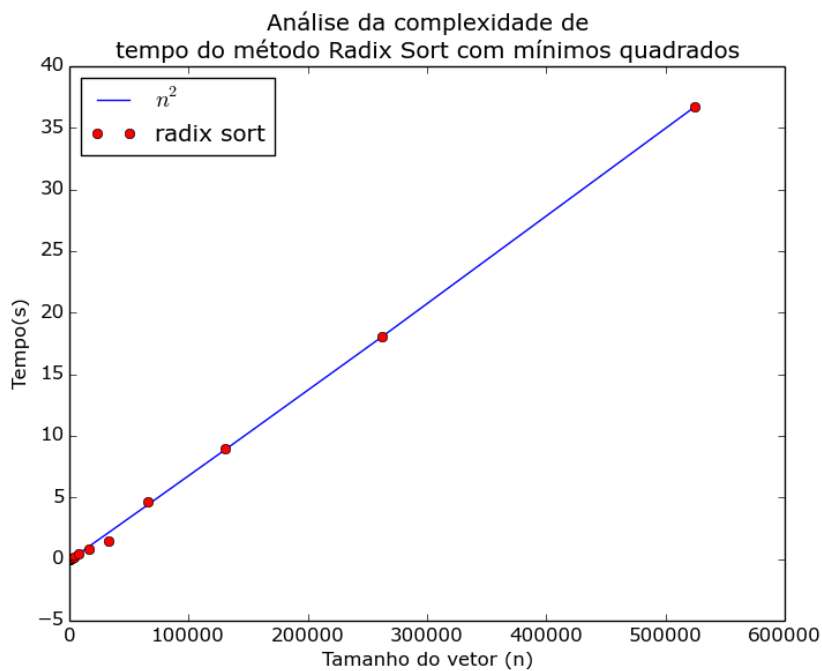
- (a) Complexidade de tempo do método Radix Sort disponível na lista de imagens 2.5.



**Figura 2.5:** Complexidade de tempo do método Radix Sort (Vetor Ordenado Decrescente)

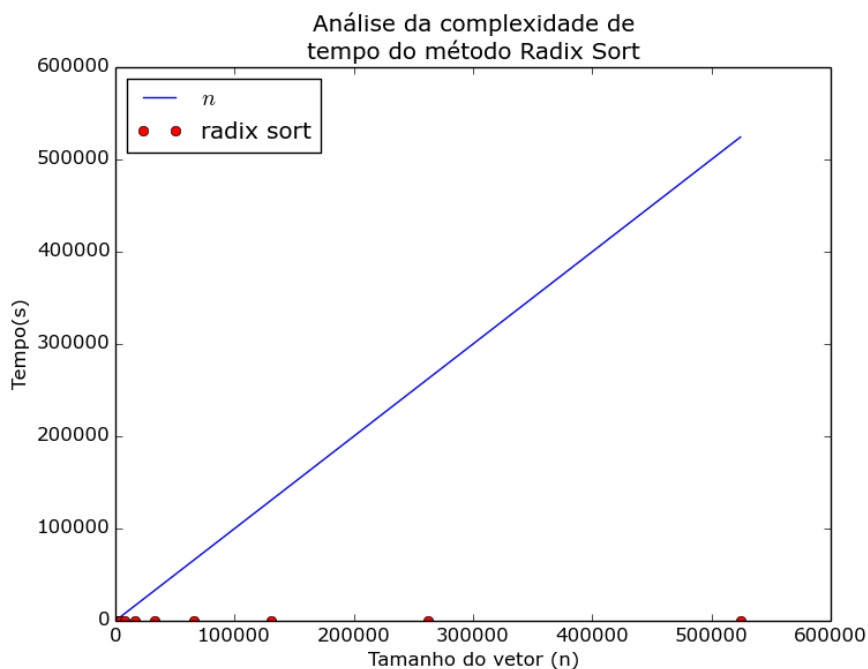
- (b) Complexidade de tempo do método Radix Sort com mínimos quadrados disponível na lista de imagens 2.6.

4. Para um vetor parcialmente ordenado crescente



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

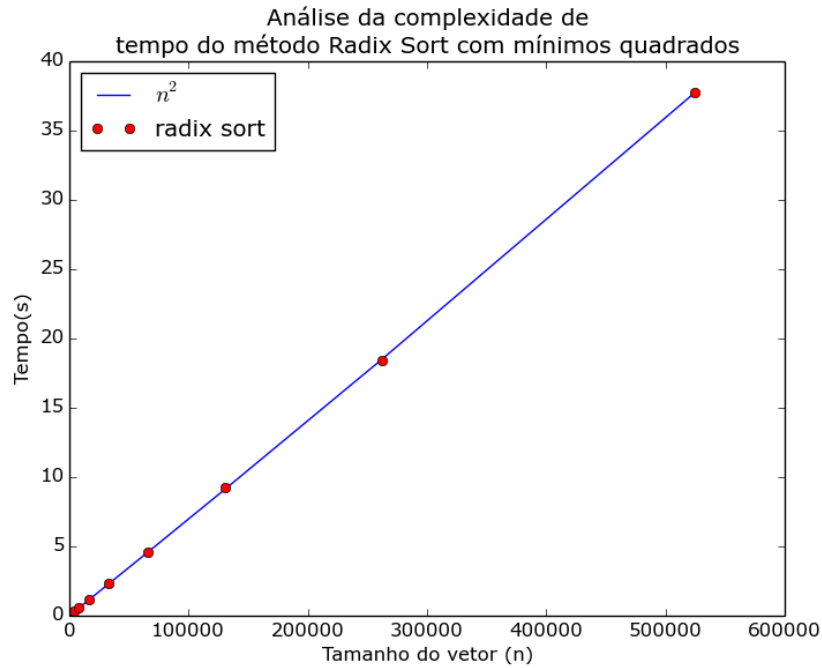
- (a) Complexidade de tempo do método Radix Sort disponível na lista de imagens 2.7.



**Figura 2.7:** Complexidade de tempo do método Radix Sort (Vetor Parcialmente Ordenado Crescente)

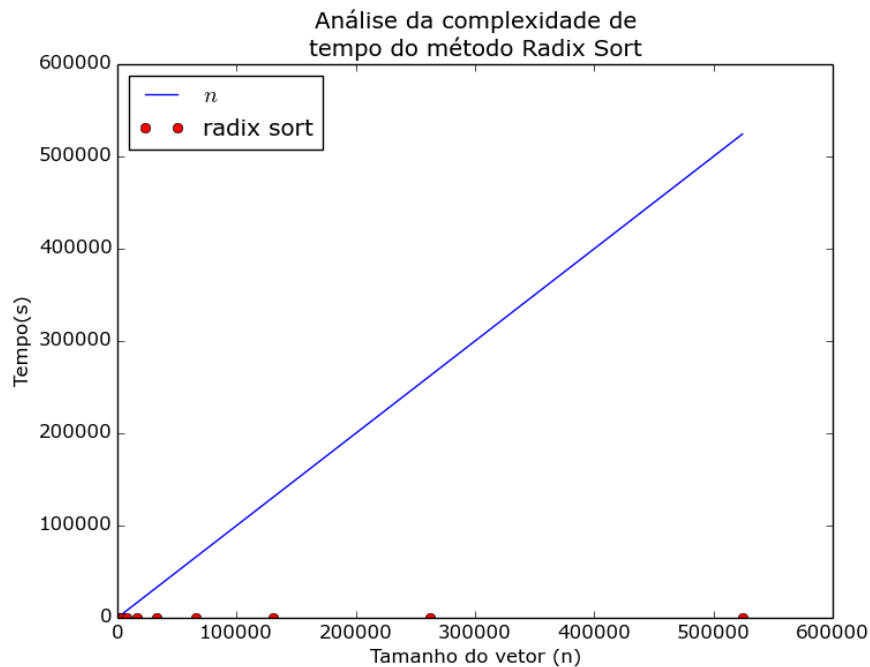
- (b) Complexidade de tempo do método Radix Sort com mínimos quadrados disponível na lista de imagens 2.8.

5. Para um vetor parcialmente ordenado decrescente



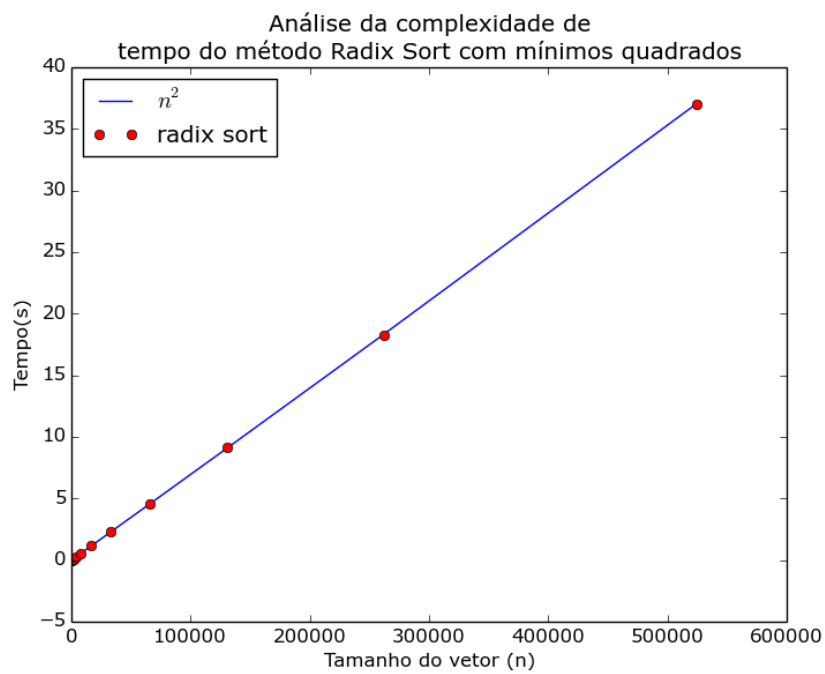
**Figura 2.8:** Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Parcialmente Ordenado Crescente)

- (a) Complexidade de tempo do método Radix Sort disponível na lista de imagens [2.9](#).



**Figura 2.9:** Complexidade de tempo do método Radix Sort (Vetor Parcialmente Ordenado Decrescente)

- (b) Complexidade de tempo do método Radix Sort com mínimos quadrados disponível na lista de imagens [2.10](#).



**Figura 2.10:** Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Parcialmente Ordenado Decrescente)



# Capítulo 3

## Tabelas

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

**Tabela 3.1:** *Vetor Aleatório*

Tamanho do Vetor	Tempo(s)
32	0.000924
64	0.001048
128	0.001275
256	0.001602
512	0.002296
1024	0.003892
2048	0.006807
4096	0.012659
8192	0.025144
16384	0.047915

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

Tamanho do Vetor	Tempo(s)
32	0.001087
64	0.001281
128	0.001409
256	0.001953
512	0.002287
1024	0.004282
2048	0.007226
4096	0.013225
8192	0.025477
16384	0.050537

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

Tamanho do Vetor	Tempo(s)
32	0.001238
64	0.001347
128	0.001692
256	0.001601
512	0.002410
1024	0.004015
2048	0.007246
4096	0.012888
8192	0.025990
16384	0.048788

**Tabela 3.4:** *Vetor Parcialmente Ordenado Decrescente*

Tamanho do Vetor	Tempo(s)
32	0.001269
64	0.001236
128	0.001449
256	0.001712
512	0.002419
1024	0.003892
2048	0.006918
4096	0.013007
8192	0.025498
16384	0.048699

**Tabela 3.5:** *Vetor Parcialmente Ordenado Crescente*

Tamanho do Vetor	Tempo(s)
32	0.001263
64	0.001182
128	0.001514
256	0.001999
512	0.002636
1024	0.004079
2048	0.006980
4096	0.013248
8192	0.025285
16384	0.050265

# 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 2.1, 2.3, 2.5, 2.7, 2.9), apresentaram uma correspondência forte com a curva da função  $F(x) = x$ , o que nos permite concluir que, dada a complexidade de tempo do algoritmo Radix 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 Radix Sort é  $O(n)$ .

## 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/Radix') ## adicionei o código de ordenação
16 sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
    relatorio/Resultados/Radix') ## adicionei o resultado do executa_teste
17
18
19 def executa_teste(arqteste, arqsaida, intervalo, tempo=[3,17,31,56]):
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          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]) +
        float(linhas[tempo[3]].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[tempo[0]].split()[2])
45     #print (linhas[tempo[1]].split()[2])
46     #print (linhas[tempo[2]].split()[2])
47     #print (linhas[tempo[3]].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}  {:13.6f}'.format(n ,tempo_total)
55     print(str_res)
56     f.write(str_res + '\n')
57     lcomp = 0
58     f.close()
59     shutil.move("tRadix_vetor_ordenado_decrescente.dat", "/home/gmarson/
        Git/AnaliseDeAlgoritmos/Trabalho_Final/relatorio/Resultados/Radix/
        tRadix_vetor_ordenado_decrescente.dat")
60
61     executa_teste("testeGeneric.py", "tRadix_vetor_ordenado_decrescente.dat",
        2 ** np.arange(5,20))
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

```

## A.1

```
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("tRadix_memoria.dat", "/home/gmarson/Git/
    AnaliseDeAlgoritmos/Trabalho_Final/relatorio/Resultados/Radix/
    tRadix_memoria.dat")
91
92 #executa_teste_memoria("testeGeneric.py", "tRadix_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 , label='$n$') ## custo esperado bubble Sort
99     plt.plot(n, c, 'ro', label='radix sort')
100     # Posiciona a legenda
101     plt.legend(loc='upper left')
102
103     # Posiciona o título
104     plt.title('Análise de comparações do método Radix 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/Radix/radix_plot_1_ordenado_decrescente
        .png')
111     plt.show()
112
113
114
115 def plota_teste2(arqsaida):
116     n, t = np.loadtxt(arqsaida, unpack=True)
117     plt.plot(n, n, label='$n $')
118     plt.plot(n, t, 'ro', label='radix 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 Radix Sort')
125
126     # Rotula os eixos
127     plt.xlabel('Tamanho do vetor (n)')
128     plt.ylabel('Tempo(s)')
129
130     plt.savefig('relatorio/imagens/Radix/radix_plot_2_ordenado_decrescente
        .png')
131     plt.show()
132
133
134
135
136 def plota_teste3(arqsaida):
137     n, 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='radix 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 Radix 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/Radix/radix_plot_3_ordenado_decrescente
159             .png')
160     plt.show()
161
162 #plota_teste1("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
163     relatorio/Resultados/Radix/tRadix_vetor_ordenado_decrescente.dat")
164
165 #plota_teste2("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
166     relatorio/Resultados/Radix/tRadix_vetor_ordenado_decrescente.dat")
167
168 #plota_teste3("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
169     relatorio/Resultados/Radix/tRadix_vetor_ordenado_decrescente.dat")
170
171 def plota_teste4(arqsaida):
172     n, c, t = np.loadtxt(arqsaida, unpack=True)
173
174     # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
175     # o método dos mínimos quadrados
176     coefs = np.polyfit(n, c, 2)
177     p = np.poly1d(coefs)
178
179     plt.plot(n, p(n), label='$n^2$')
180     plt.plot(n, c, 'ro', label='bubble sort')
181
182     # Posiciona a legenda
183     plt.legend(loc='upper left')
184
185     # Posiciona o título
186     plt.title('Análise da complexidade de \ntempo do método da bolha')
187
188     # Rotula os eixos
189     plt.xlabel('Tamanho do vetor (n)')
190     plt.ylabel('Número de comparações')
191
192     plt.savefig('bubble4.png')
193     plt.show()
194
195 def plota_teste5(arqsaida):
196     n, c, t = np.loadtxt(arqsaida, unpack=True)
197
198     # 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.polyld(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()
```

---