Análise de Complexidade de Tempo do Método Radix 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

18 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 Radix Sort com relação a tempo. Além disso, será feita uma comparação da curva de tempo do que se espera do algoritmo, ou seja, $O(n)$ 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
|   |   `-- __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# função countingsort adaptada
20 def _countingSort(A, k):
21     contador = [0] * 10  # Contador é o histograma
22     B = [0] * len(A)
23     n = len(A)
24     for i in range(0, n):
25         contador[(A[i] // k) % 10] += 1
26 
27     for i in range(1, len(contador)):
28         contador[i] += contador[i - 1]
29 
30     for j in range((n - 1), -1, -1):
31         B[contador[(A[j] // k) % 10] - 1] = A[j]
32         contador[(A[j] // k) % 10] -= 1
33 
34     for i in range(0, n):
35         A[i] = B[i]
36 
37 #@profile
38 def _contaDigitos(valor):
39     digitos = 0
40     while (valor != 0):
41         digitos += 1
42         valor //= 10
43     return digitos
44 
45 
46 
47 
48 
49 
50 
51 
52 #lista = [170, 45, 75, 90, 802, 24, 2, 66]
53 #radix(lista)
54 #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 
```

```
sys.setrecursionlimit(200000)

from monitor import *
from memoria import *

from RadixSort 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)
radix(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):
        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, inf=0, sup=0.9999999999):
    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.

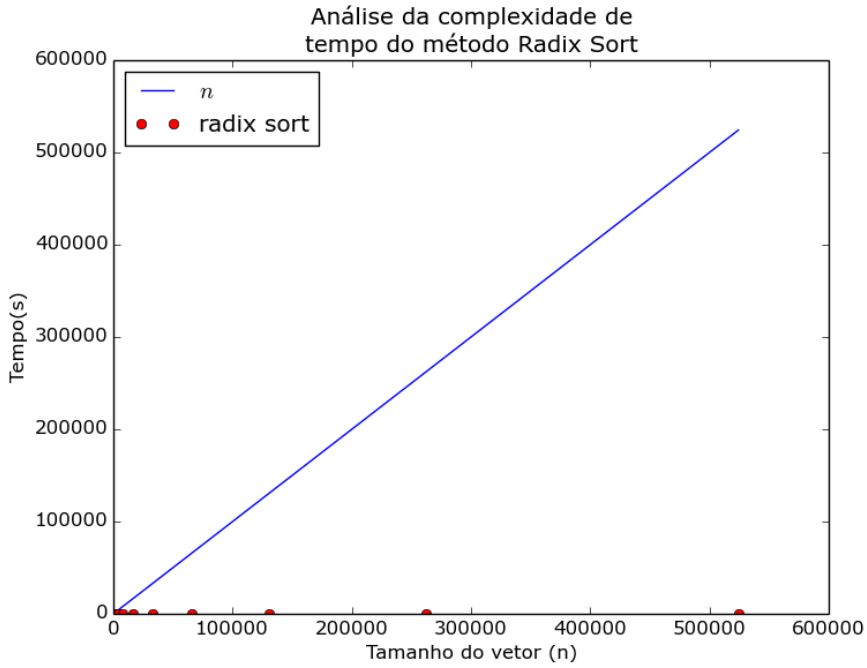

Figura 2.1: *Complexidade de tempo do método Radix Sort (Vetor Aleatório)*

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

2. Para um vetor ordenado crescente

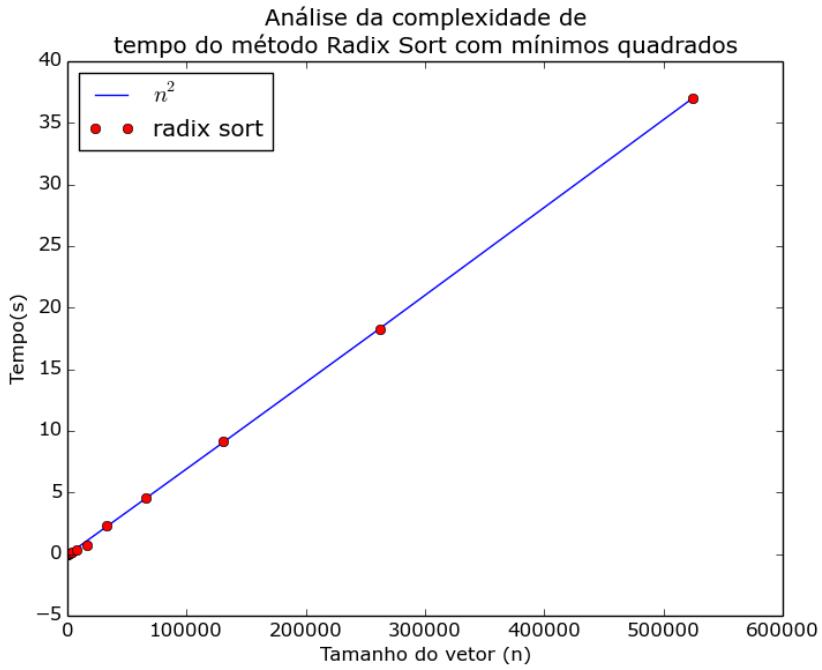

Figura 2.2: *Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Aleatório)*

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

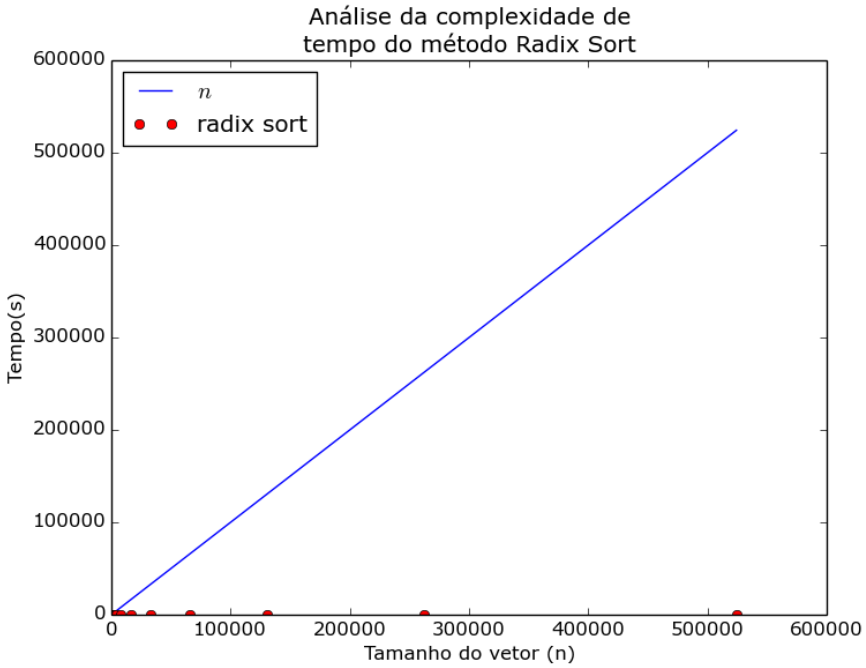

Figura 2.3: *Complexidade de tempo do método Radix Sort (Vetor Ordenado Crescente)*

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

3. Para um vetor ordenado decrescente

2.0

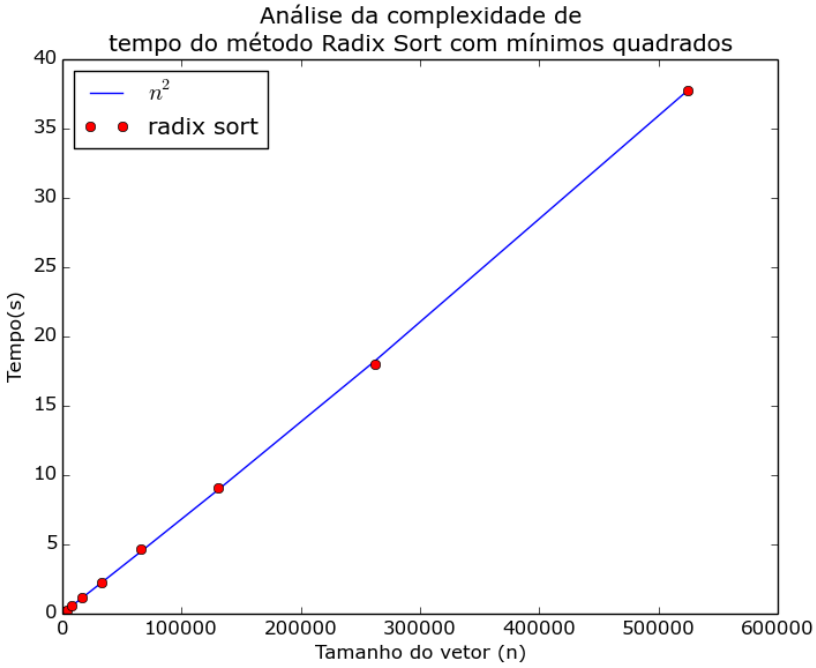

Figura 2.4: *Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Ordenado Crescente)*

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

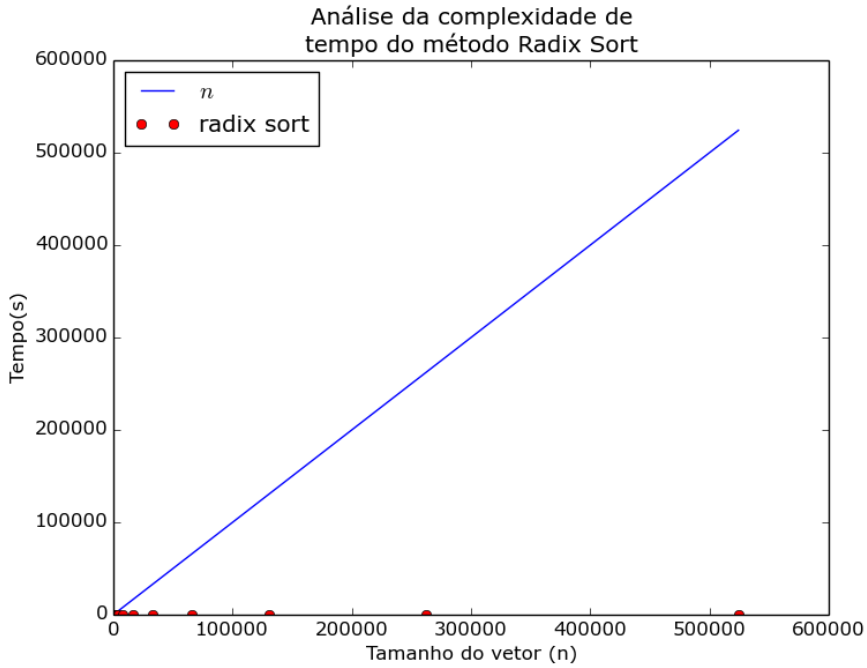

Figura 2.5: *Complexidade de tempo do método Radix Sort (Vetor Ordenado Decrescente)*

(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

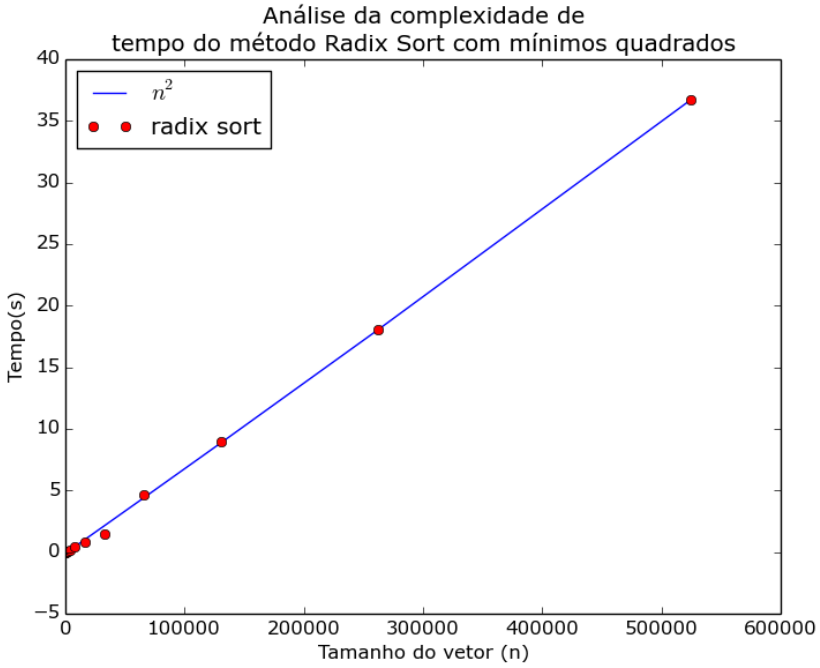

Figura 2.6: *Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Ordenado Decrescente)*

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

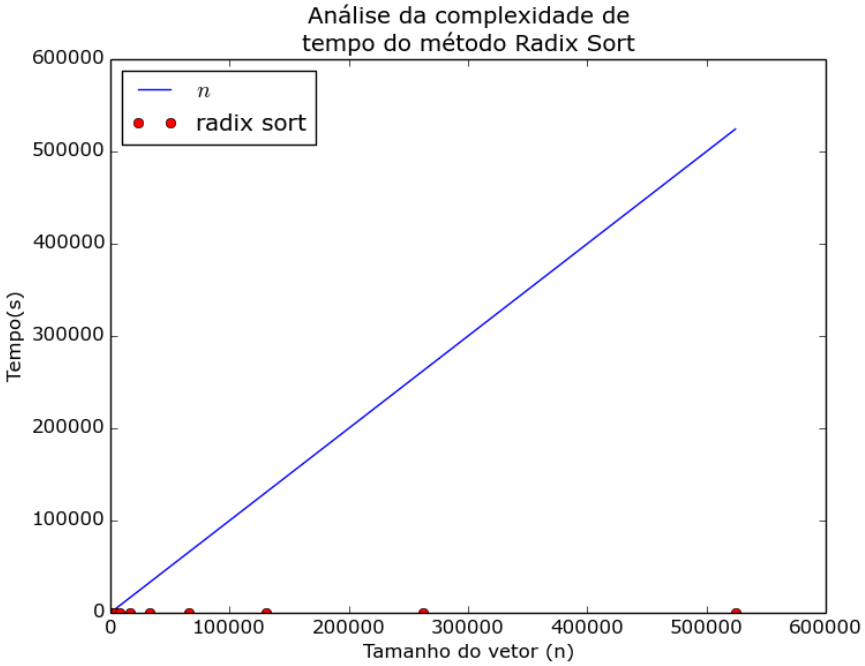

Figura 2.7: *Complexidade de tempo do método Radix Sort (Vetor Parcialmente Ordenado Crescente)*

(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

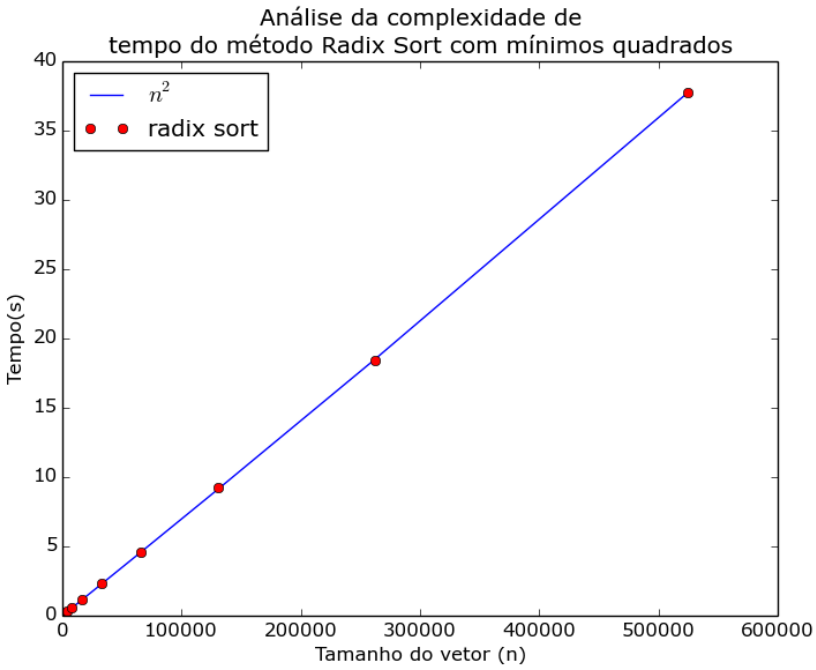

Figura 2.8: *Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Parcialmente Ordenado Crescente)*

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

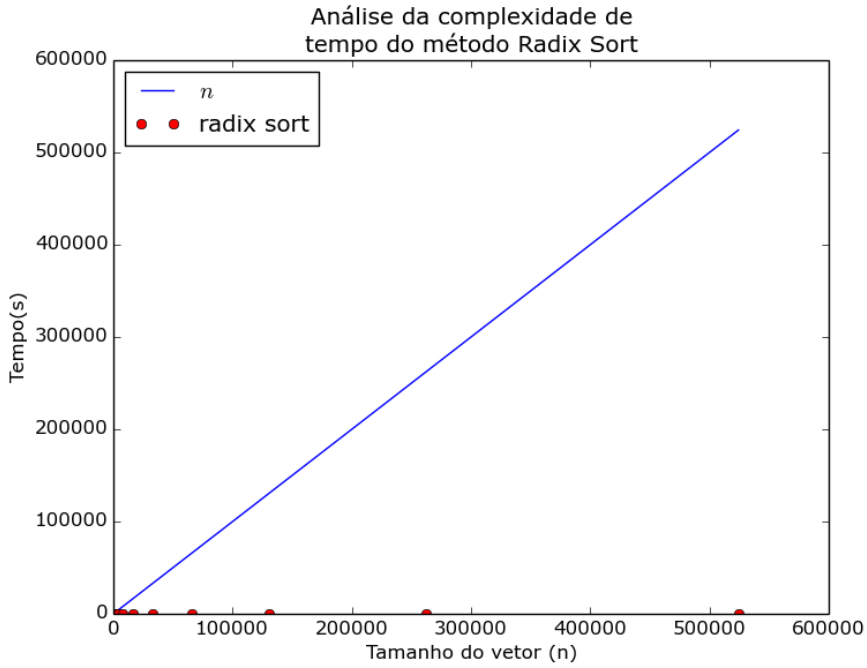

Figura 2.9: *Complexidade de tempo do método Radix Sort (Vetor Parcialmente Ordenado Decrescente)*

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

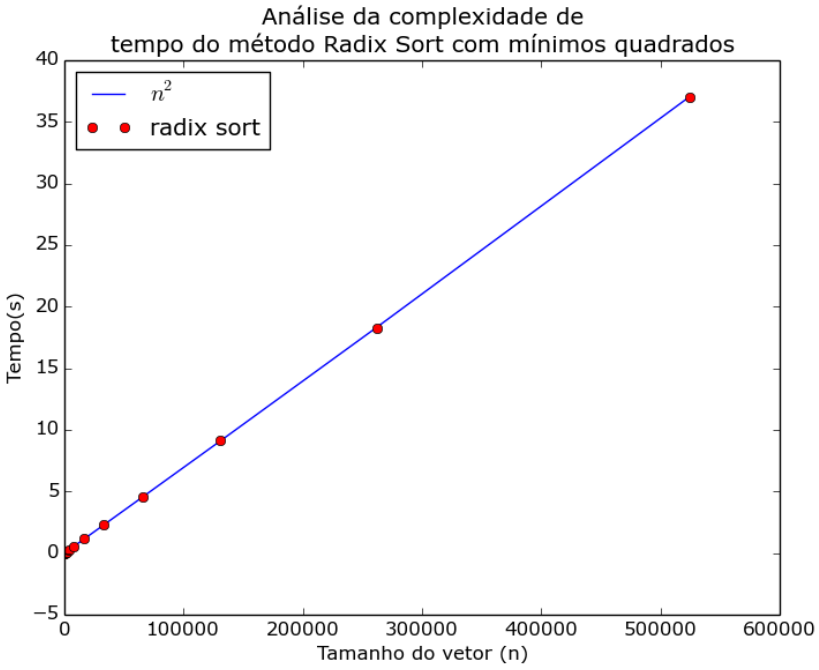

Figura 2.10: *Complexidade de tempo do método Radix Sort com mínimos quadrados (Vetor Parcialmente Ordenado Decrescente)*

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	0.001969
64	0.003053
128	0.005992
256	0.012797
512	0.023158
1024	0.051994
2048	0.092159
4096	0.202499
8192	0.370421
16384	0.758604
32768	1.619927
65536	4.523400
131072	9.818129
262144	18.306709
524288	36.800410

Tabela 3.2: *Vetor Ordenado Crescente*

Tamanho do Vetor	Tempo(s)
32	0.002569
64	0.004878
128	0.009286
256	0.018059
512	0.035540
1024	0.071870
2048	0.139780
4096	0.286223
8192	0.576995
16384	1.154050
32768	2.276046
65536	4.633520
131072	9.113568
26214	18.039597
524288	37.771921

Tabela 3.3: *Vetor Ordenado Decrescente*

Tamanho do Vetor	Tempo(s)
32	0.001840
64	0.003180
128	0.006559
256	0.011725
512	0.023004
1024	0.050099
2048	0.092232
4096	0.199064
8192	0.399744
16384	0.797877
32768	1.505112
65536	4.643434
131072	8.969211
262144	18.060512
524288	36.680364

Tabela 3.4: *Vetor Parcialmente Ordenado Decrescente*

Tamanho do Vetor	Tempo(s)
32	0.001891
64	0.003064
128	0.006604
256	0.011961
512	0.025210
1024	0.048299
2048	0.106676
4096	0.279154
8192	0.557848
16384	1.147800
32768	2.322026
65536	4.530593
131072	9.178682
262144	18.287574
524288	37.046162

Tabela 3.5: *Vetor Parcialmente Ordenado Crescente*

Tamanho do Vetor	Tempo(s)
32	0.001702
64	0.003069
128	0.009290
256	0.017940
512	0.035721
1024	0.070035
2048	0.143542
4096	0.294574
8192	0.571618
16384	1.151897
32768	2.321207
65536	4.564890
131072	9.270520
262144	18.408352
524288	37.733941

Capítulo 4

Análise

O Radix Sort é um algoritmo que dado um vetor, todos os elementos deste podem ser representados com d dígitos, onde d é uma constante. A ordenação é feita aplicando o algoritmo Counting Sort em cada dígito, o uso do Counting Sort se da pelo fato que este algoritmo é estável, além de possuir complexidade $O(n)$.

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

[1] Algoritmos: Teoria e Prática. Thomas H. Cormen Today

Apêndice A

Códigos extensos

A.1 testdriver.py

Listagem A.1: testdriver.py

```
# coding = utf-8
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 METODO 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/Radix')  ## adicionei o código de ordenação
sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
    relatorio/Resultados/Radix') ## adicionei o resultado do executa_teste


def executa_teste(arqteste, arqsaida,  intervalo,tempo=[3,17,31,56]):
    """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:
       intervalo: tamanhos dos vetores: Ex: 2 ** np.arange(5,10)
    """
    f = open(arqsaida,mode='w', encoding='utf-8')
    f.write('#      n       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')
        #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
```

```
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
            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/Radix/radix_plot_3_ordenado_decrescente
            .png')
158     plt.show()
159 
160 #plota_teste1("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Radix/tRadix_vetor_ordenado_decrescente.dat")
161 plota_teste2("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Radix/tRadix_vetor_ordenado_decrescente.dat")
162 plota_teste3("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Radix/tRadix_vetor_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()
```