Análise de Complexidade de Tempo do Método Counting 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

15 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 Counting 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__
|   |-- Counting
|   |   `-- __pycache__
|   |-- Heap
|   |-- Insertion
|   |   `-- __pycache__
|   |-- Merge
|   |   `-- __pycache__
|   |-- Quick
|   |   `-- __pycache__
|   `-- Selection
|       `-- __pycache__
|-- Other
|-- Plot
|-- __pycache__
```

```
`-- relatorio
    |-- imagens
    |   |-- Bubble
    |   |-- Counting
    |   |-- Insertion
    |   |-- Merge
    |   `-- Selection
    |-- Relatorio_Bubble
    |-- Relatorio_Counting
    |-- Relatorio_Insertion
    |-- Relatorio_Merge
    |-- Relatorio_Selection
    `-- Resultados
        |-- Bubble
        |-- Counting
        |-- Insertion
        |-- Merge
        |-- Quick
        `-- Selection

36 directories
```

1.2 Códigos de programas

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

1. CountingSort.py: Disponível na Listagem 1.1.

Listagem 1.1: CountingSort.py

```
@profile
def countingSort(A):
  A = [ int(x) for x in A ]
  k = max(A)
  contador = [0] * (k+1) #Contador é o histograma
  B = [0] * len(A)
  n = len(A)
  for i in range(0, n):
      contador[A[i]] = contador[A[i]] + 1

  for i in range(1, len(contador)):
      contador[i] = contador[i] + contador[i-1]

  for j in range((n-1), -1, -1):
      B[contador[A[j]]-1] = A[j]
      contador[A[j]] = contador[A[j]]-1

  return B

  #lista = [2,5,3,0,2,3,0,3]
  #print (countingSort(lista,5))
```

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/Counting')
sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final')

from monitor import *
from memoria import *

from countingSort 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)
countingSort(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)
```

```
22          t = v[i]
23          v[i] = v[j]
24          v[j] = t
25      return v
26
27
28  def criavet(n, grau=-0.5, inf=0, sup=1000):
29      passo = (sup - inf)/n
30      if grau < 0.0:
31          v = np.arange(sup, inf, -passo)
32          if grau <= -1.0:
33              return v
34          else:
35              return troca(-grau*n, v, n)
36      elif grau > 0.0:
37          v = np.arange(inf, sup, passo)
38          if grau >= 1.0:
39              return v
40          else:
41              return troca(grau*n, v, n)
42      else:
43          return np.random.randint(inf, sup, size=n)
44
45  #print(criavet(20))
46
47  #Tipo                                    grau
48  #aleatorio                               0
49  #ordenado crescente                      1
50  #ordenado decresente                     -1
51  #parcialmente ordenado crescente         0.5
52  #parcialmente ordenado descrescente      -0.5
53
54
55  def executa(fn, v):
56      gc.disable()
57      with Tempo(True) as tempo:
58          fn(v)
59      gc.enable()
```

4. testdriver.py Referenciado no apêndice A.

Capítulo 2

Gráficos

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

1. Para um vetor aleatório

   (a) Complexidade de tempo do método Counting Sort disponível na lista de imagens 2.1.

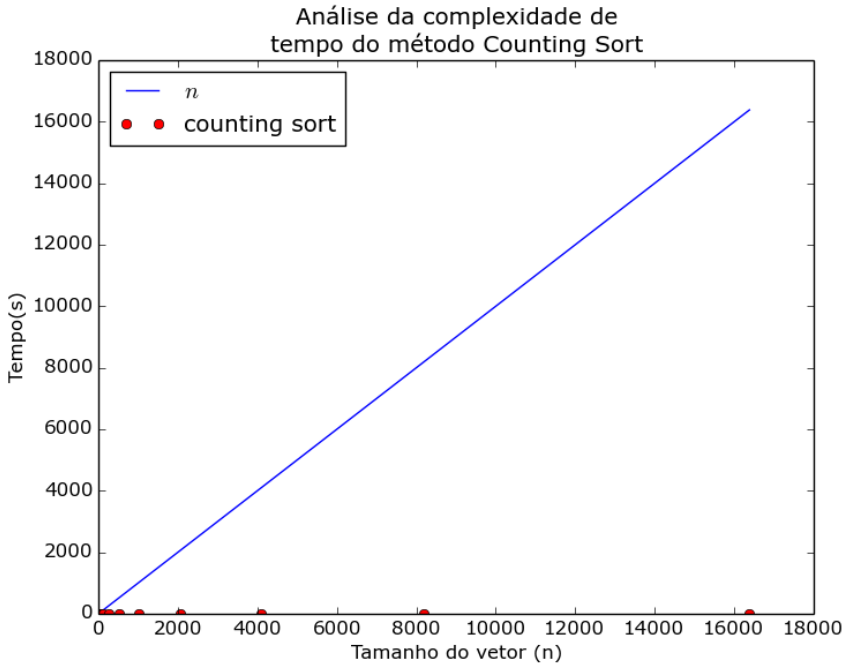

Figura 2.1: *Complexidade de tempo do método Counting Sort (Vetor Aleatório)*

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

2. Para um vetor ordenado crescente

2.0

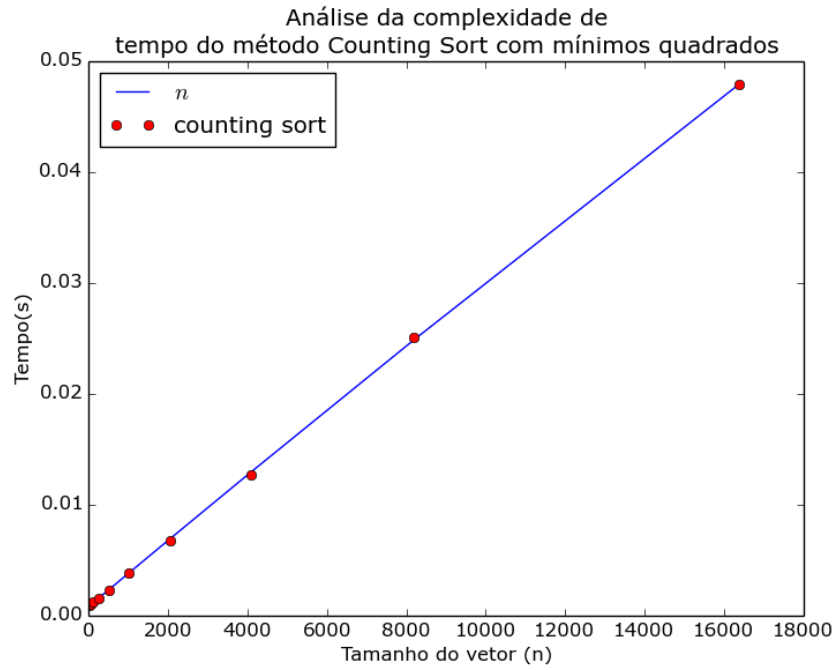

Figura 2.2: *Complexidade de tempo do método Counting Sort com mínimos quadrados (Vetor Aleatório)*

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

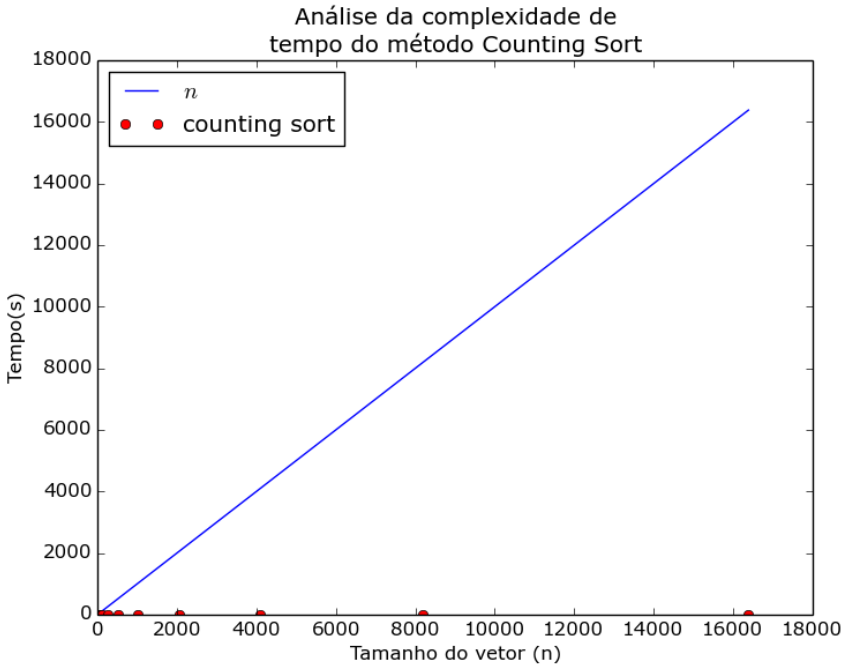

Figura 2.3: *Complexidade de tempo do método Counting Sort (Vetor Ordenado Crescente)*

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

3. Para um vetor ordenado decrescente

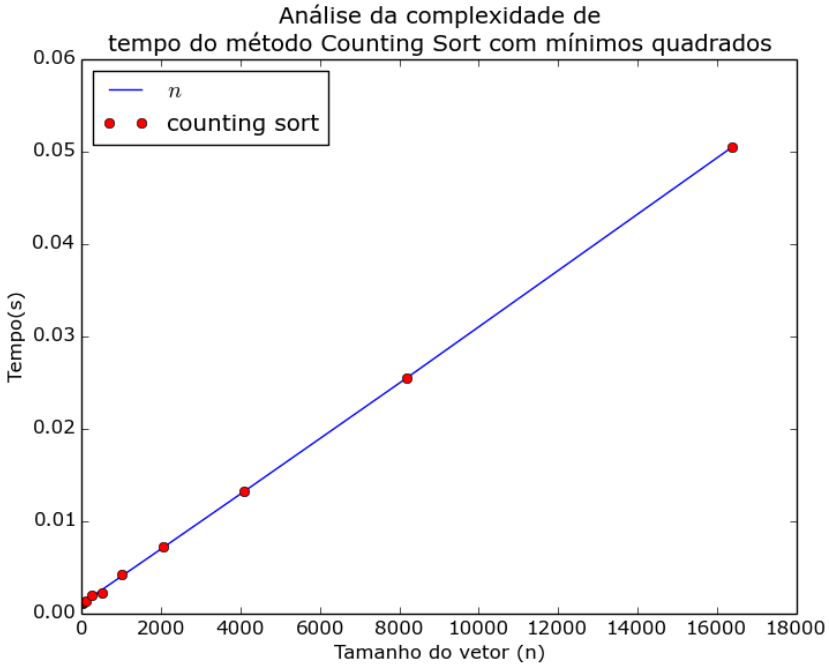

Figura 2.4: *Complexidade de tempo do método Counting Sort com mínimos quadrados (Vetor Ordenado Crescente)*

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

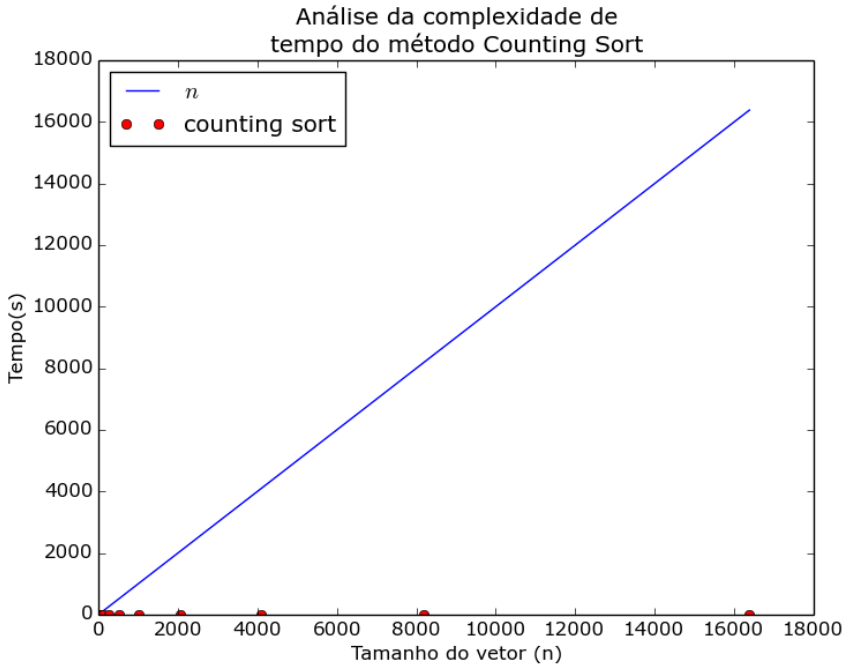

Figura 2.5: *Complexidade de tempo do método Counting Sort (Vetor Ordenado Decrescente)*

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

4. Para um vetor parcialmente ordenado crescente

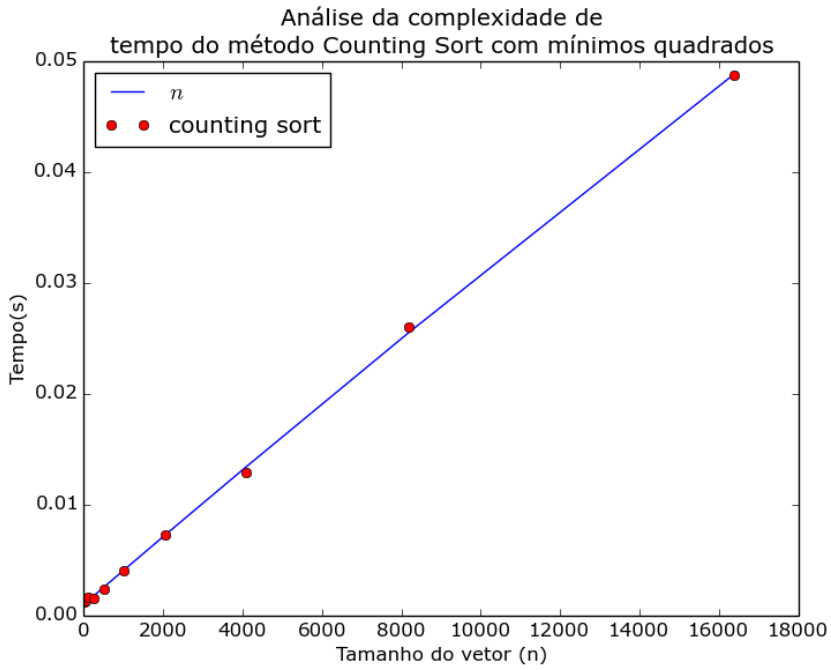

Figura 2.6: *Complexidade de tempo do método Counting Sort com mínimos quadrados (Vetor Ordenado Decrescente)*

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

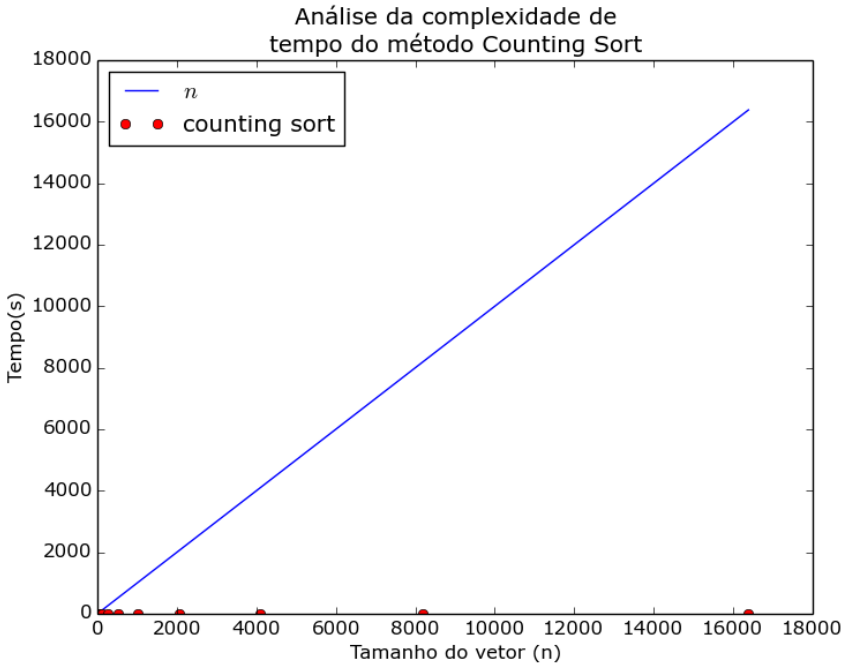

Figura 2.7: *Complexidade de tempo do método Counting Sort (Vetor Parcialmente Ordenado Crescente)*

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

5. Para um vetor parcialmente ordenado decrescente

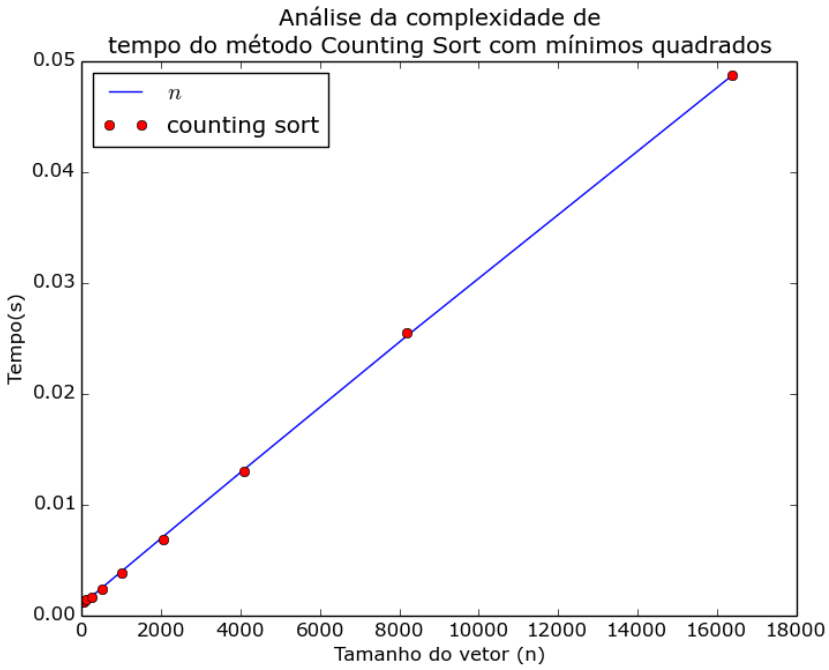

Figura 2.8: *Complexidade de tempo do método Counting Sort com mínimos quadrados (Vetor Parcialmente Ordenado Crescente)*

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

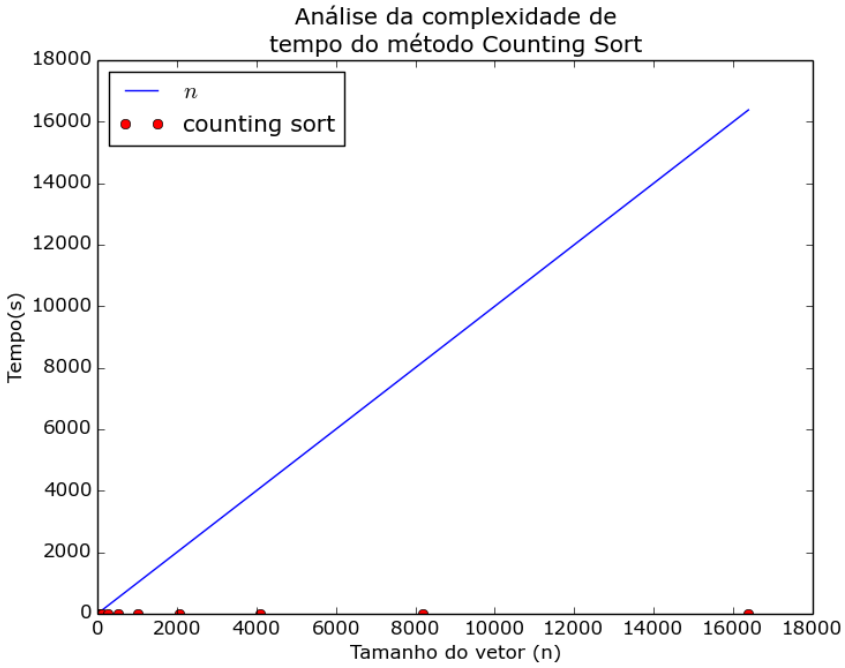

Figura 2.9: *Complexidade de tempo do método Counting Sort (Vetor Parcialmente Ordenado Decrescente)*

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

2.0

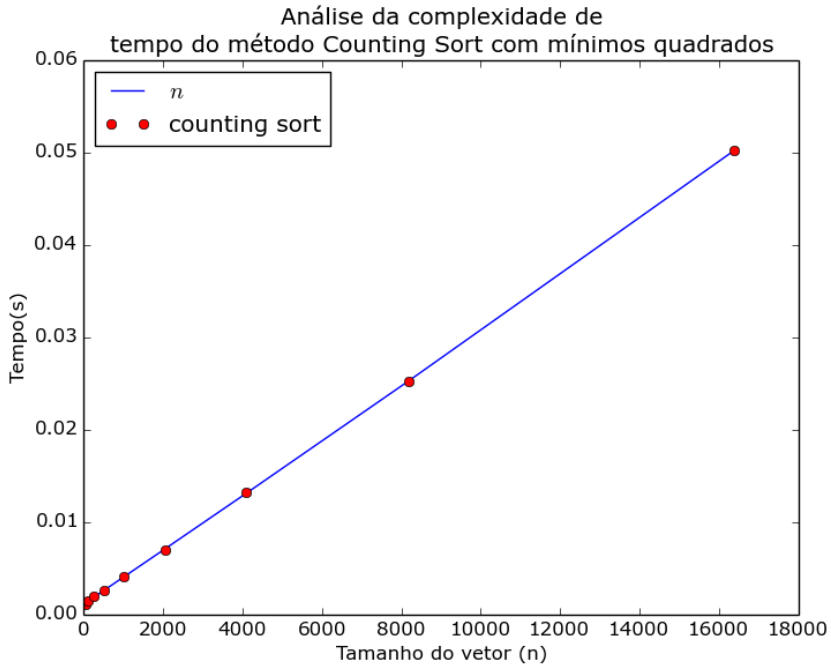

Figura 2.10: *Complexidade de tempo do método Counting Sort com mínimos quadrados (Vetor Parcialmente Ordenado Decrescente)*

Capítulo 3

Tabelas

Seguem as tabelas utilizadas para a análise do método Counting 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 Counting 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 Counting Sort é $O(n)$.

Capítulo 5

Citações e referências bibliográficas

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/Counting')  ## adicionei o código de ordenação
sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
    relatorio/Resultados/Counting') ## 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      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')
```

```
33          #for i in linhas:
34          #    print(i)
35          #print (linhas)
36          unidade_tempo = float(linhas[1].split()[2])
37          tempo_total = float(linhas[3].split()[2])
38          #lcomp = linhas[nlin].split()
39
40          #print ("unidade tempo: ",unidade_tempo )
41          #print("lcomp: ",lcomp)
42          #print("tempo total",tempo_total)
43
44          #num_comps = int(lcomp[1])
45          str_res = '{:>8} {:13.6f}'.format(n, tempo_total)
46          print(str_res)
47          f.write(str_res + '\n')
48      f.close()
49      shutil.move("tCounting_vetor_parcialmente_ordenado_decrescente.dat", "
            /home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/relatorio/
            Resultados/Counting/
            tCounting_vetor_parcialmente_ordenado_decrescente.dat")
50
51  executa_teste("testeGeneric.py", "
        tCounting_vetor_parcialmente_ordenado_decrescente.dat", 46, 2 ** np.
        arange(5,15))
52
53  def executa_teste_memoria(arqteste, arqsaida, nlin, intervalo):
54      """Executa uma sequência de testes contidos em arqteste, com:
55         arqsaida: nome do arquivo de saída, ex: tBolha.dat
56         nlin: número da linha no arquivo gerado pelo line_profiler contendo
57               os dados de interesse. Ex: 14
58         intervalo: tamanhos dos vetores: Ex: 2 ** np.arange(5,10)
59      """
60      f = open(arqsaida,mode='w', encoding='utf-8')
61      f.write('#       n    comparações       tempo(s)\n')
62
63      for n in intervalo:
64          cmd = ' '.join(["kernprof -l -v ", "testeGeneric.py", str(n)])
65
66          str_saida = subprocess.check_output(cmd, shell=True).decode('utf-8
                ')
67
68          linhas = str_saida.split('\n')
69          for i in linhas:
70              print(i)
71
72          print ("Linhas:",linhas[1])
73
74          unidade_tempo = float(linhas[1].split()[2])
75
76
77          str_res = '{:>8} {:>13} {:13.6f}'.format(n, n, n)
78          print(str_res)
79          f.write(str_res + '\n')
80      f.close()
81      #shutil.move("tSelection_memoria.dat", "/home/gmarson/Git/
            AnaliseDeAlgoritmos/Trabalho_Final/relatorio/Resultados/Selection/
            tSelection_memoria.dat")
82
```

```
83 #executa_teste_memoria("testeGeneric.py", "tSelection_memoria.dat", 14, 2
       ** np.arange(5,15))
84
85 def plota_teste1(arqsaida):
86     n, c, t = np.loadtxt(arqsaida, unpack=True)
87     #print("n: ",n,"\nc: ",c,"\nt: ",t)
88     #n eh o tamanho da entrada , c eh o tanto de comparações e t eh o
           tempo gasto
89     plt.plot(n, n ** 2, label='$n^2$')  ## custo esperado bubble Sort
90     plt.plot(n, c, 'ro', label='selection sort')
91
92     # Posiciona a legenda
93     plt.legend(loc='upper left')
94
95     # Posiciona o título
96     plt.title('Análise de comparações do método da seleção')
97
98     # Rotula os eixos
99     plt.xlabel('Tamanho do vetor (n)')
100    plt.ylabel('Número de comparações')
101
102    plt.savefig('relatorio/imagens/Selection/
           selection_plot_1_ordenado_descresente.png')
103    plt.show()
104
105
106
107 def plota_teste2(arqsaida):
108    n, t = np.loadtxt(arqsaida, unpack=True)
109    plt.plot(n, n , label='$n$')
110    plt.plot(n, t, 'ro', label='counting sort')
111
112    # Posiciona a legenda
113    plt.legend(loc='upper left')
114
115    # Posiciona o título
116    plt.title('Análise da complexidade de \ntempo do método Counting Sort'
           )
117
118    # Rotula os eixos
119    plt.xlabel('Tamanho do vetor (n)')
120    plt.ylabel('Tempo(s)')
121
122    plt.savefig('relatorio/imagens/Counting/
           counting_plot_2_parcialmente_ordenado_decrescente.png')
123    plt.show()
124
125
126 def plota_teste3(arqsaida):
127    n, t = np.loadtxt(arqsaida, unpack=True)
128
129    # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
130    # o método dos mínimos quadrados
131    coefs = np.polyfit(n, t, 2)
132    p = np.poly1d(coefs)
133
134    plt.plot(n, p(n), label='$n$')
135    plt.plot(n, t, 'ro', label='counting sort')
136
```

```
137     # Posiciona a legenda
138     plt.legend(loc='upper left')
139 
140     # Posiciona o título
141     plt.title('Análise da complexidade de \ntempo do método Counting Sort
            com mínimos quadrados')
142 
143     # Rotula os eixos
144     plt.xlabel('Tamanho do vetor (n)')
145     plt.ylabel('Tempo(s)')
146 
147     plt.savefig('relatorio/imagens/Counting/
            counting_plot_3_parcialmente_ordenado_decrescente.png')
148     plt.show()
149 
150 #plota_teste1("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Selection/tSelection_vetor_ordenado_descresente.
        dat")
151 plota_teste2("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Counting/
        tCounting_vetor_parcialmente_ordenado_decrescente.dat")
152 plota_teste3("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Counting/
        tCounting_vetor_parcialmente_ordenado_decrescente.dat")
153 
154 
155 def plota_teste4(arqsaida):
156     n, c, t = np.loadtxt(arqsaida, unpack=True)
157 
158     # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
159     # o método dos mínimos quadrados
160     coefs = np.polyfit(n, c, 2)
161     p = np.poly1d(coefs)
162 
163     plt.plot(n, p(n), label='$n^2$')
164     plt.plot(n, c, 'ro', label='bubble sort')
165 
166     # Posiciona a legenda
167     plt.legend(loc='upper left')
168 
169     # Posiciona o título
170     plt.title('Análise da complexidade de \ntempo do método da bolha')
171 
172     # Rotula os eixos
173     plt.xlabel('Tamanho do vetor (n)')
174     plt.ylabel('Número de comparações')
175 
176     plt.savefig('bubble4.png')
177     plt.show()
178 
179 def plota_teste5(arqsaida):
180     n, c, t = np.loadtxt(arqsaida, unpack=True)
181 
182     # Calcula os coeficientes de um ajuste a um polinômio de grau 2 usando
183     # o método dos mínimos quadrados
184     coefs = np.polyfit(n, c, 2)
185     p = np.poly1d(coefs)
186 
187     # set_yscale('log')
```

```
188     # set_yscale('log')
189     plt.semilogy(n, p(n), label='$n^2$')
190     plt.semilogy(n, c, 'ro', label='bubble sort')
191 
192     # Posiciona a legenda
193     plt.legend(loc='upper left')
194 
195     # Posiciona o título
196     plt.title('Análise da complexidade de \ntempo do método da bolha')
197 
198     # Rotula os eixos
199     plt.xlabel('Tamanho do vetor (n)')
200     plt.ylabel('Número de comparações')
201 
202     plt.savefig('bubble5.png')
203     plt.show()
```