Análise de Complexidade de Tempo do Método Quick 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 Quick Sort com relação a tempo. Além disso, será feita uma comparação da curva de tempo do que se espera do algoritmo, ou seja, $O(nlgn)$ com melhor caso e $\theta(n^2)$ no pior caso.

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__
|   |-- Quick
|   |   `-- __pycache__
|   |-- Quick
|   |   `-- __pycache__
|   `-- Selection
|       `-- __pycache__
|-- Other
```

```
|-- Plot
|-- __pycache__
`-- relatorio
    |-- imagens
    |   |-- Bubble
    |   |-- Bucket
    |   |-- Counting
    |   |-- Heap
    |   |-- Insertion
    |   |-- Quick
    |   |-- Quick
    |   |-- Radix
    |   `-- Selection
    |-- Relatorio_Bubble
    |-- Relatorio_Bucket
    |-- Relatorio_Counting
    |-- Relatorio_Heap
    |-- Relatorio_Insertion
    |-- Relatorio_Quick
    |-- Relatorio_Selection
    `-- Resultados
        |-- Bubble
        |-- Bucket
        |-- Counting
        |-- Heap
        |-- Insertion
        |-- Quick
        |-- Quick
        `-- Selection

47 directories
```

1.2 Códigos de programas

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

1. QuickSort.py: Disponível na Listagem 1.1.

Listagem 1.1: QuickSort.py

```
#@profile
def Particiona(A,p,r):
  x = A[r] #pivo é o último elemento
  i = p-1
  for j in range(p,r):
    if A[j] <= x:
      i = i + 1
      aux = A[i]
      A[i] = A[j]
      A[j] = aux
  temp = A[i+1]
  A[i+1] = A[r]
  A[r] = temp
  return i+1

```

```
#@profile
def quickSort(A,p,r):
  if(p < r):
    q = Particiona(A,p,r)
    quickSort(A,p,(q-1))
    quickSort(A,(q+1),r)
  return A

@profile
def quick(A):
  quickSort(A,0,(len(A)-1))




#A = [i for i in range(10)]
#quick(A)
#quickSort(A,0,6)
#print(A)
```

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

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
    if grau < 0.0:
        v = np.arange(sup, inf, -passo)
        if grau <= -1.0:
            return v
        else:
            return troca(-grau*n, v, n)
    elif grau > 0.0:
        v = np.arange(inf, sup, passo)
        if grau >= 1.0:
            return v
        else:
            return troca(grau*n, v, n)
    else:
        #return np.random.randint(inf, sup, size=n)
        return [random.random() for i in range(n)] # for bucket sort



#print(criavet(20))

#Tipo                                    grau
#aleatorio                                0
#ordenado_crescente                       1
#ordenado_decrescente                    -1
#parcialmente_ordenado_crescente          0.5
#parcialmente_ordenado_decrescente       -0.5

```

```
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 Quick Sort:

1. Para um vetor aleatório

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

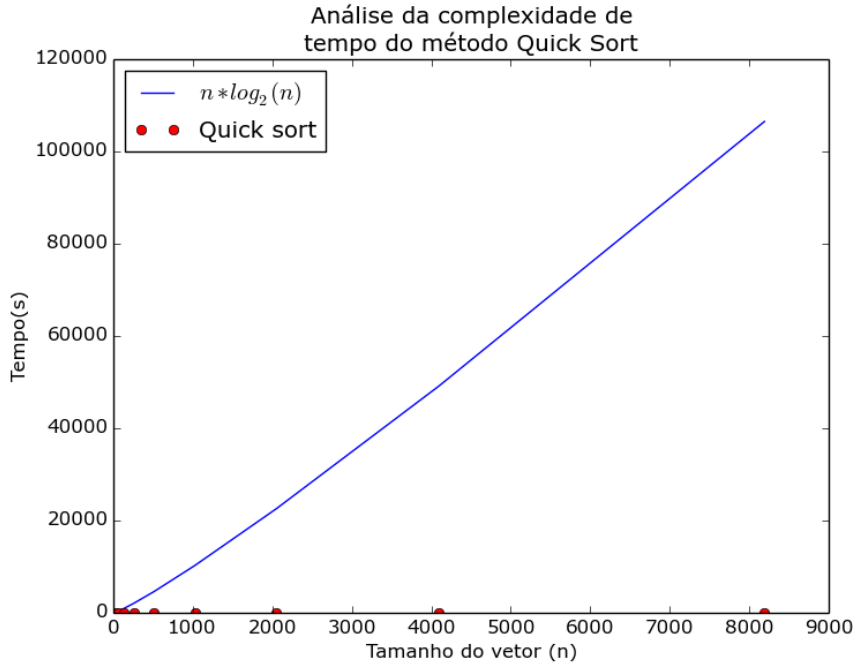

Figura 2.1: *Complexidade de tempo do método Quick Sort (Vetor Aleatório)*

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

2. Complexidade de tempo do método Quick Sort disponível na lista de imagens 2.3.

3. Complexidade de tempo do método Quick Sort com mínimos quadrados disponível na lista de imagens 2.4.

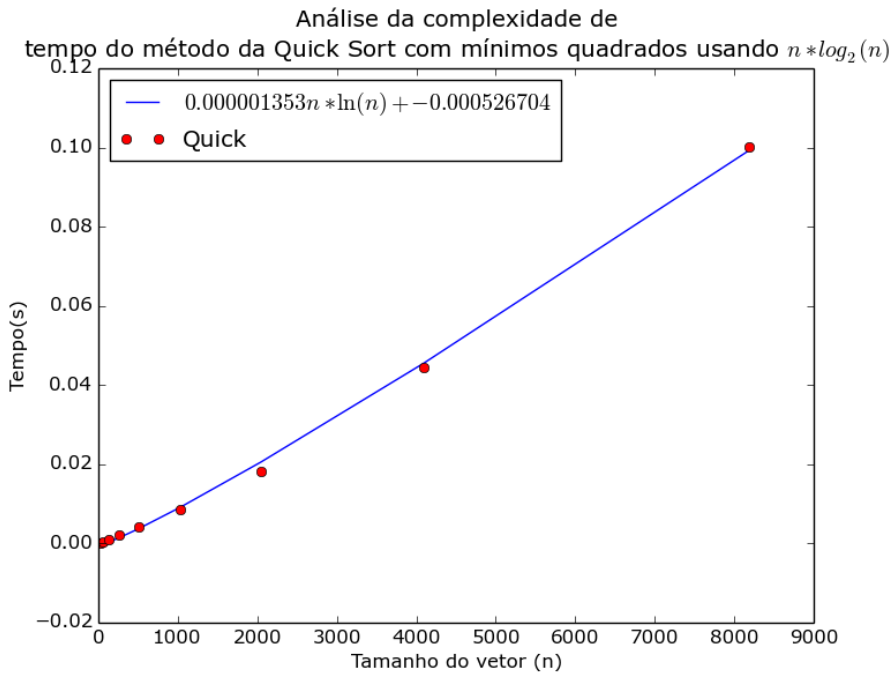

Figura 2.2: *Complexidade de tempo do método Quick Sort com mínimos quadrados (Vetor Aleatório)*

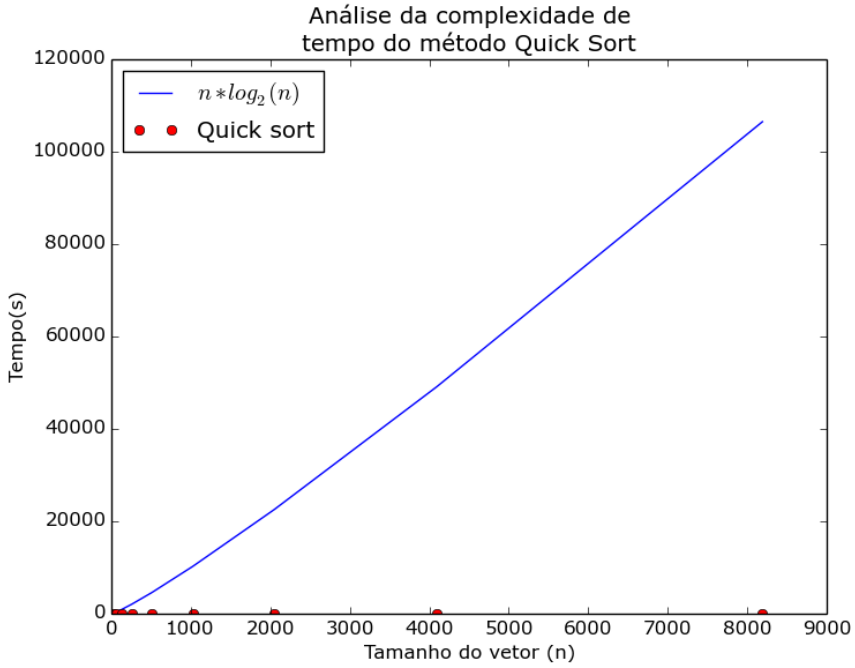

Figura 2.3: *Complexidade de tempo do método Quick Sort (Vetor Ordenado Crescente)*

1. Complexidade de tempo do método Quick Sort disponível na lista de imagens 2.5.
2. Complexidade de tempo do método Quick Sort com mínimos quadrados disponível na lista de imagens 2.6.

1. Complexidade de tempo do método Quick Sort disponível na lista de imagens 2.7.

2.0

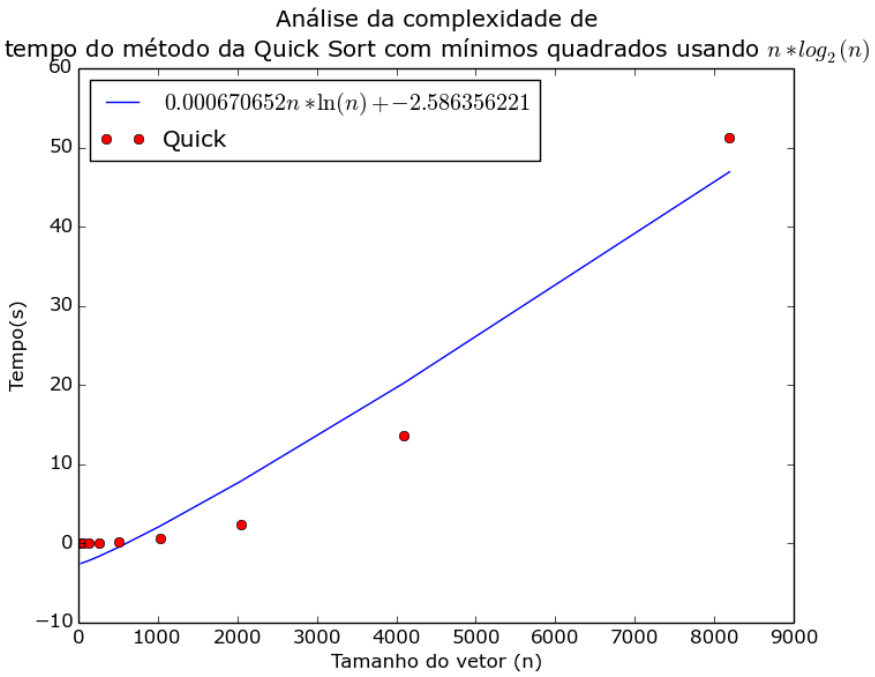

Figura 2.4: *Complexidade de tempo do método Quick Sort com mínimos quadrados (Vetor Ordenado Crescente)*

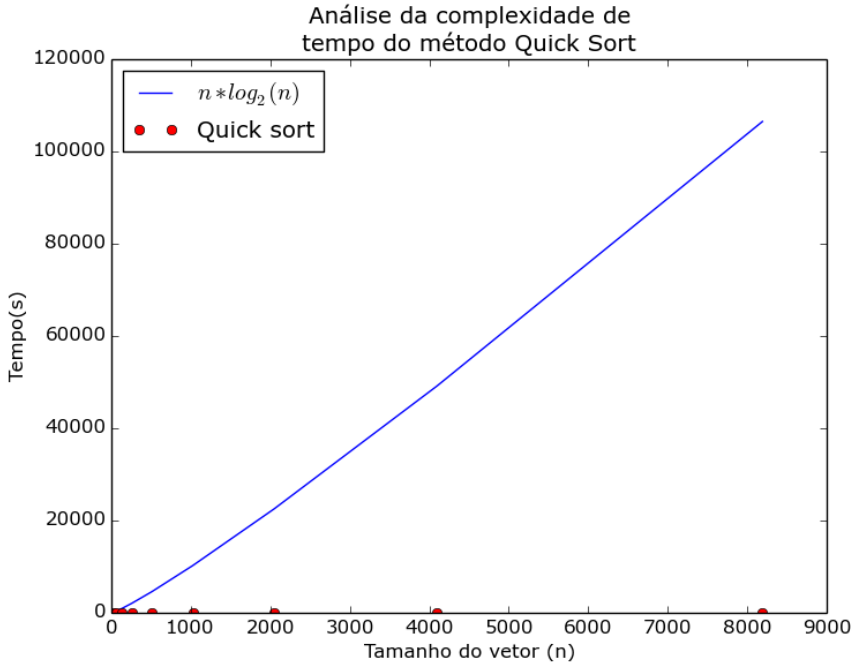

Figura 2.5: *Complexidade de tempo do método Quick Sort (Vetor Ordenado Decrescente)*

2. Complexidade de tempo do método Quick Sort com mínimos quadrados disponível na lista de imagens 2.8.

1. Complexidade de tempo do método Quick Sort disponível na lista de imagens 2.9.

2. Complexidade de tempo do método Quick Sort com mínimos quadrados disponível na lista de imagens 2.10.

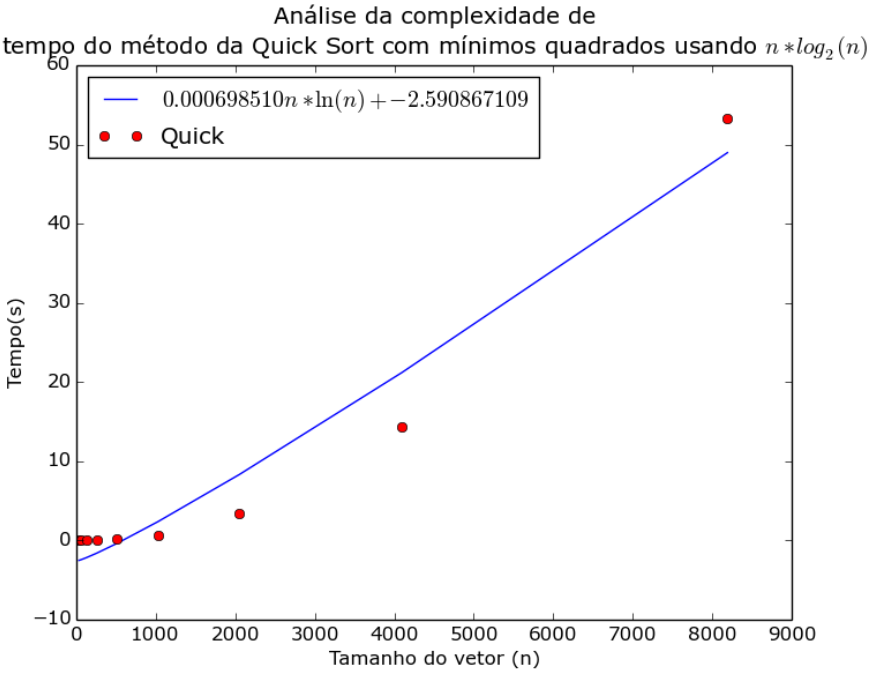

Figura 2.6: *Complexidade de tempo do método Quick Sort com mínimos quadrados (Vetor Ordenado Decrescente)*

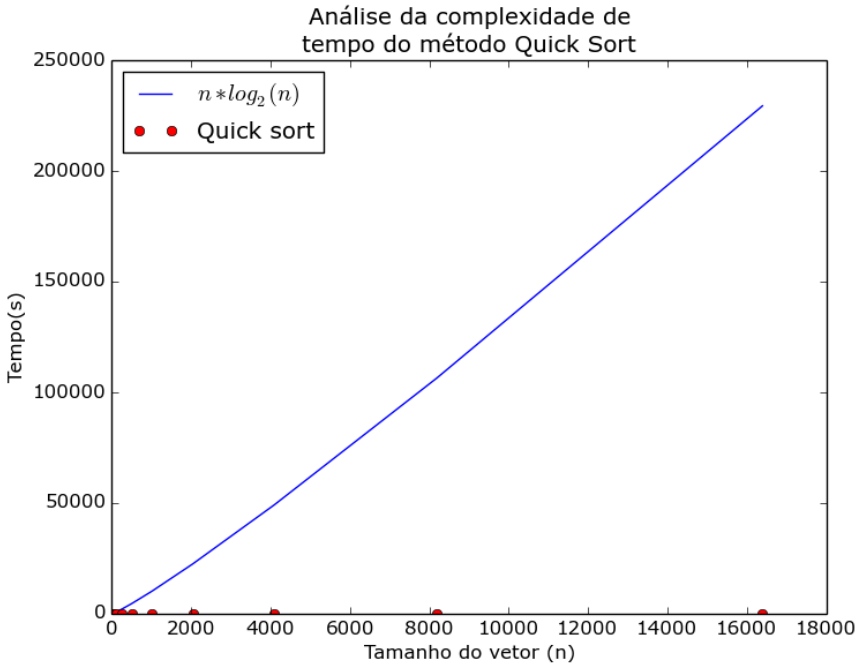

Figura 2.7: *Complexidade de tempo do método Quick Sort (Vetor Parcialmente Ordenado Crescente)*

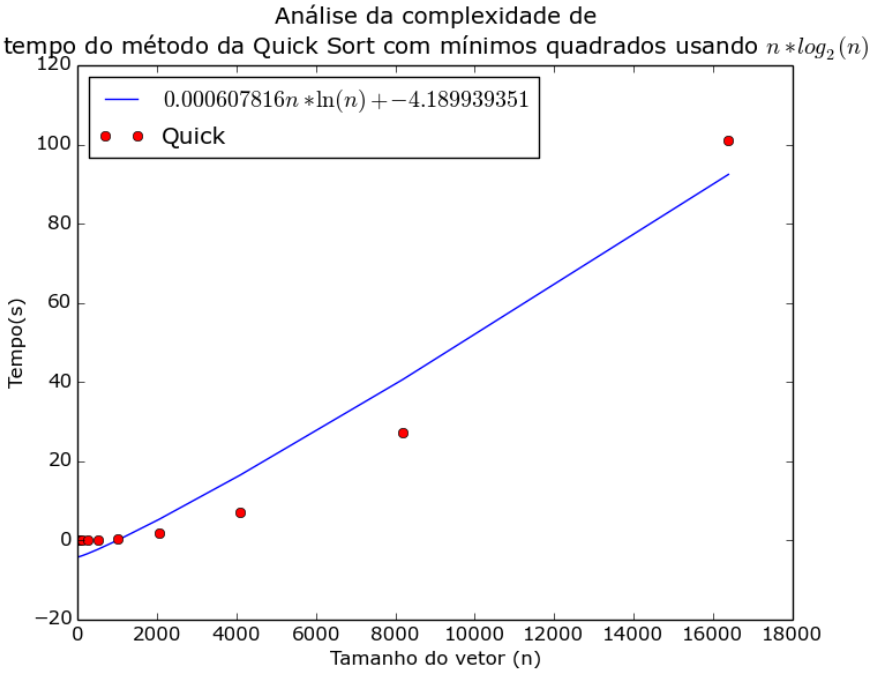

Figura 2.8: *Complexidade de tempo do método Quick Sort com mínimos quadrados (Vetor Parcialmente Ordenado Crescente)*

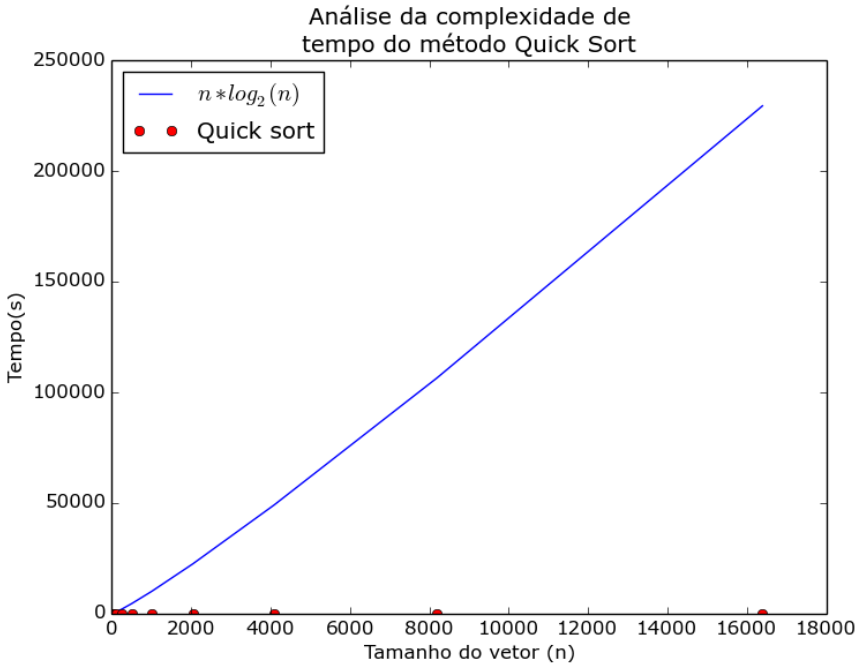

Figura 2.9: *Complexidade de tempo do método Quick Sort (Vetor Parcialmente Ordenado Decrescente)*

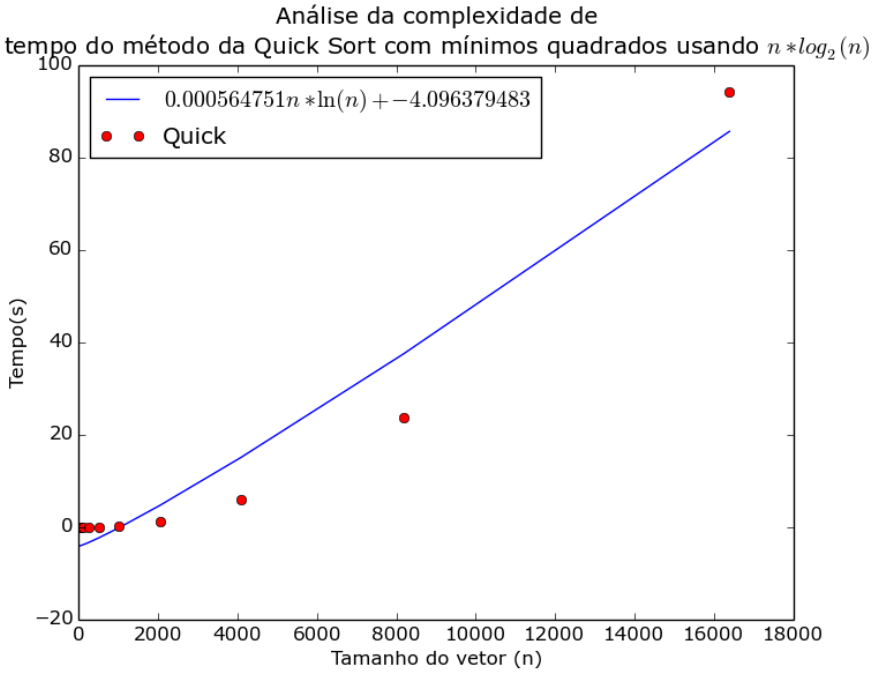

Figura 2.10: *Complexidade de tempo do método Quick Sort com mínimos quadrados (Vetor Parcialmente Ordenado Decrescente)*

Capítulo 3

Tabelas

Seguem as tabelas utilizadas para a análise do método Quick 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*

Tamanho do Vetor	Comparações	Tempo(s)
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*

Tamanho do Vetor	Comparações	Tempo(s)
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

Tabela 3.6: *Dados para Análise de Memória*

Tamanho do Vetor	Memória (MiB)
32	24.102000
64	24.199000
128	23.121000
256	24.078000
512	24.242000
1024	24.215000
2048	24.10864
4096	24.230000
8192	24.168000
16384	24.738000

Capítulo 4

Análise

O algoritmo Quick Sort é um algoritmo de divisão e conquista que utiliza a função particiona, $\theta(n)$, que tem como objetivo ordenar as partições. O algoritmo quicksort possui o melhor caso em $\theta(nlgn)$ e pior caso em $\theta(n^2)$. O pior caso ocorre quando o vetor já está ordenado. 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 $F(x) = x^2$, o que nos permite concluir que, dada a complexidade de tempo do algoritmo Quick 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 Quick Sort é $\theta(nlgn)$.

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/Bucket')  ## adicionei o código de ordenação
sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
    relatorio/Resultados/Bucket') ## 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')
        #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("tBucket_vetor_parcialmente_ordenado_decrescente.dat", "/
            home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/relatorio/
            Resultados/Bucket/tBucket_vetor_parcialmente_ordenado_decrescente.
            dat")
50
51  executa_teste("testeGeneric.py", "
        tBucket_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='bucket 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 Bucket Sort')
117
118    # Rotula os eixos
119    plt.xlabel('Tamanho do vetor (n)')
120    plt.ylabel('Tempo(s)')
121
122    plt.savefig('relatorio/imagens/Bucket/
           bucket_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='bucket 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 Bucket 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/Bucket/
            bucket_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/Bucket/
        tBucket_vetor_parcialmente_ordenado_decrescente.dat")
152 plota_teste3("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
        relatorio/Resultados/Bucket/
        tBucket_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()
```