

Análise de Complexidade de Tempo do Método Bucket 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

2.1	Complexidade de tempo do método Bucket Sort (Vetor Aleatório)	10
2.2	Complexidade de tempo do método Bucket Sort com mínimos quadrados (Vetor Aleatório)	11
2.3	Complexidade de tempo do método Bucket Sort (Vetor Ordenado Crescente)	11
2.4	Complexidade de tempo do método Bucket Sort com mínimos quadrados (Vetor Ordenado Crescente)	12
2.5	Complexidade de tempo do método Bucket Sort (Vetor Ordenado Decrescente)	12
2.6	Complexidade de tempo do método Bucket Sort com mínimos quadrados (Vetor Ordenado Decrescente)	13
2.7	Complexidade de tempo do método Bucket Sort (Vetor Parcialmente Ordenado Crescente)	13
2.8	Complexidade de tempo do método Bucket Sort com mínimos quadrados (Vetor Parcialmente Ordenado Crescente)	14
2.9	Complexidade de tempo do método Bucket Sort (Vetor Parcialmente Ordenado Decrescente)	14
2.10	Complexidade de tempo do método Bucket Sort com mínimos quadrados (Vetor Parcialmente Ordenado Decrescente)	15

Lista de Tabelas

3.1	Vetor Aleatório	16
3.2	Vetor Ordenado Crescente	16
3.3	Vetor Ordenado Decrescente	17
3.4	Vetor Parcialmente Ordenado Decrescente	17
3.5	Vetor Parcialmente Ordenado Crescente	17
3.6	Dados para Análise de Memória	18

Lista de Listagens

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

Sumário

Lista de Figuras	2
Lista de Tabelas	3
1 Introdução	6
1.1 Diretório	6
1.2 Códigos de programas	7
2 Gráficos	10
3 Tabelas	16
4 Análise	19
5 Citações e referências bibliográficas	20
Apêndice	21
A Códigos extensos	21
A.1 testdriver.py	21

Capítulo 1

Introdução

Este documento foi feito com o intuito de exibir uma análise do algoritmo Bucket 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 raiz:

```
tree --charset=ASCII -d
.
|-- Codigos
|   |-- Bubble
|   |   `-- __pycache__
|   |-- Bucket
|   |   `-- __pycache__
|   |-- Heap
|   |-- Insertion
|   |   `-- __pycache__
|   |-- Merge
|   |   `-- __pycache__
|   |-- Quick
|   |   `-- __pycache__
|   `-- Selection
|       `-- __pycache__
|-- Other
|-- Plot
|-- __pycache__
```

```

`-- relatorio
  |-- imagens
  |   |-- Bubble
  |   |-- Bucket
  |   |-- Insertion
  |   |-- Merge
  |   `-- Selection
  |-- Relatorio_Bubble
  |-- Relatorio_Bucket
  |-- Relatorio_Insertion
  |-- Relatorio_Merge
  |-- Relatorio_Selection
  `-- Resultados
      |-- Bubble
      |-- Bucket
      |-- Insertion
      |-- Merge
      |-- Quick
      `-- Selection

```

36 directories

1.2 Códigos de programas

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

1. BucketSort.py: Disponível na Listagem 1.1.

Listagem 1.1: BucketSort.py

```

1
2 @profile
3 def bucket_sort(floats):
4     buckets = [ [] for _ in range(len(floats)) ]
5     for num in floats:
6         i = int(len(floats) * num)
7         #print(buckets)
8         buckets[i].append(num)
9
10    result = []
11    for bucket in buckets:
12        _insertion_sort(bucket) # INSERTION_SORT(bucket)
13        result += bucket
14    return result
15
16
17 # colocamos a mesma versão do Insertion Sort, que já havíamos feito,
18 # aqui apenas
19 # para facilitar a análise de complexidade do Bucket Sort
20 def _insertion_sort(lista):
21     for j in range(1, len(lista)):
22         chave = lista[j]
23         i = j
24         while (i>0 and lista[i-1]>chave):

```

```

25     lista[i] = lista[i-1]
26     i = i-1
27     lista[i] = chave
28
29
30 #print(bucket_sort([0.112, 0.3, 0.008, 0.07, 0.9, 0.8, 0.43, 0.29]))

```

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  sys.setrecursionlimit(200000)
9
10 from monitor import *
11 from memoria import *
12
13 from RadixSort import *
14 import argparse
15
16 parser = argparse.ArgumentParser()
17 parser.add_argument("n", type=int, help="número de elementos no vetor
   de teste")
18 args = parser.parse_args()
19
20 v = criavet(args.n)
21 radix(v)
22
23
24
25 ## A EXECUÇÃO DESSE ARQUIVO EH ASSIM
26 ## NA LINHA DE COMANDO VC MANDA O NOME DO ARQUIVO E O TAMANHO DO
   ELEMNTTO DO vetor
27 ##EXEMPLO testeBubble.py 10
28 ##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, inf=0, sup=0.999999999):
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()

```

Capítulo 2

Gráficos

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

1. Para um vetor aleatório

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

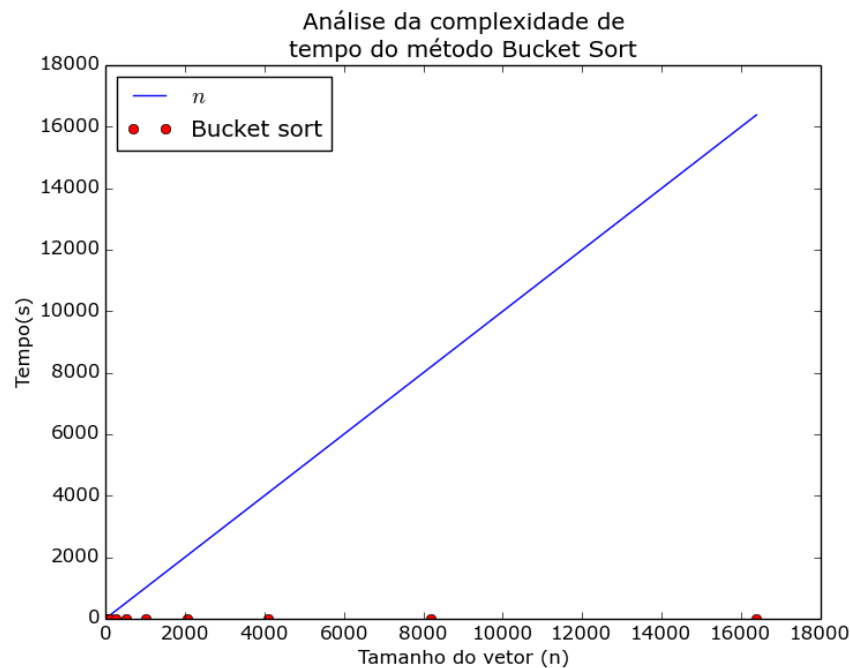


Figura 2.1: Complexidade de tempo do método Bucket Sort (Vetor Aleatório)

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

2. Para um vetor ordenado crescente

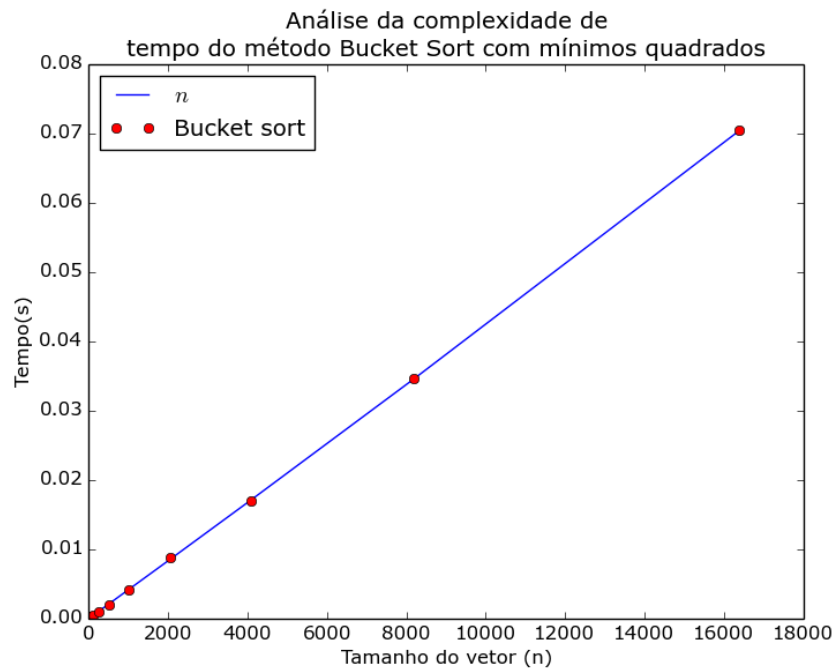


Figura 2.2: Complexidade de tempo do método Bucket Sort com mínimos quadrados (Vetor Aleatório)

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

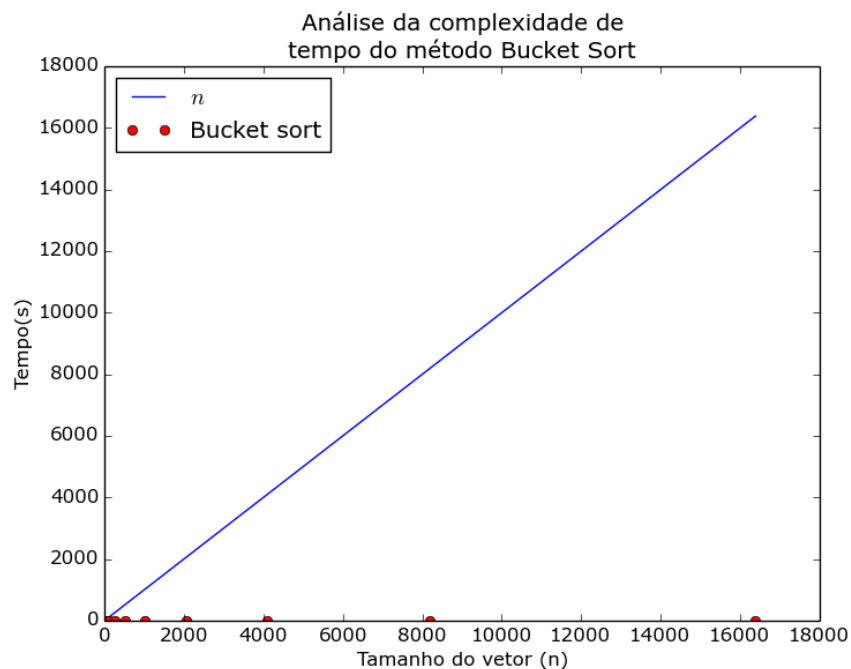


Figura 2.3: Complexidade de tempo do método Bucket Sort (Vetor Ordenado Crescente)

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

3. Para um vetor ordenado decrescente

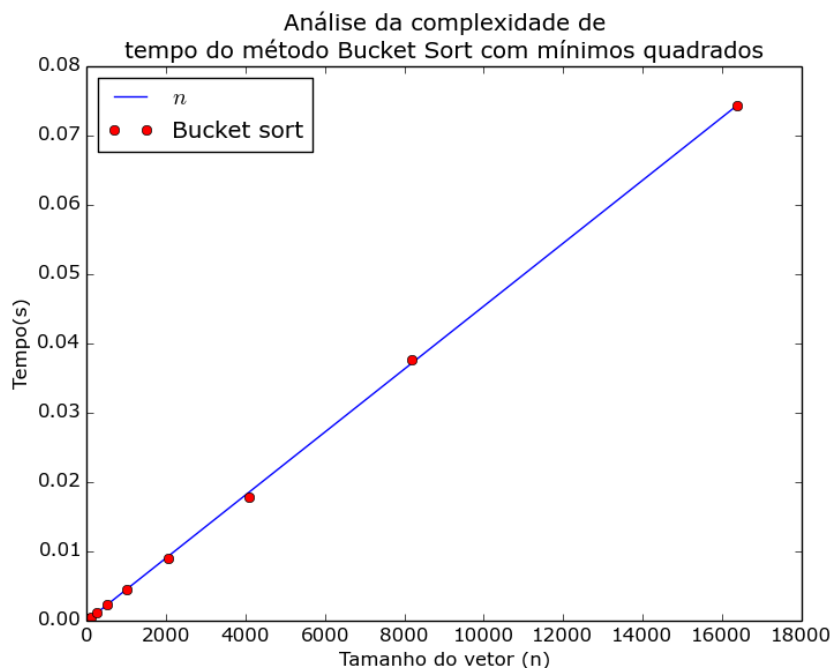


Figura 2.4: Complexidade de tempo do método Bucket Sort com mínimos quadrados (Vetor Ordenado Crescente)

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

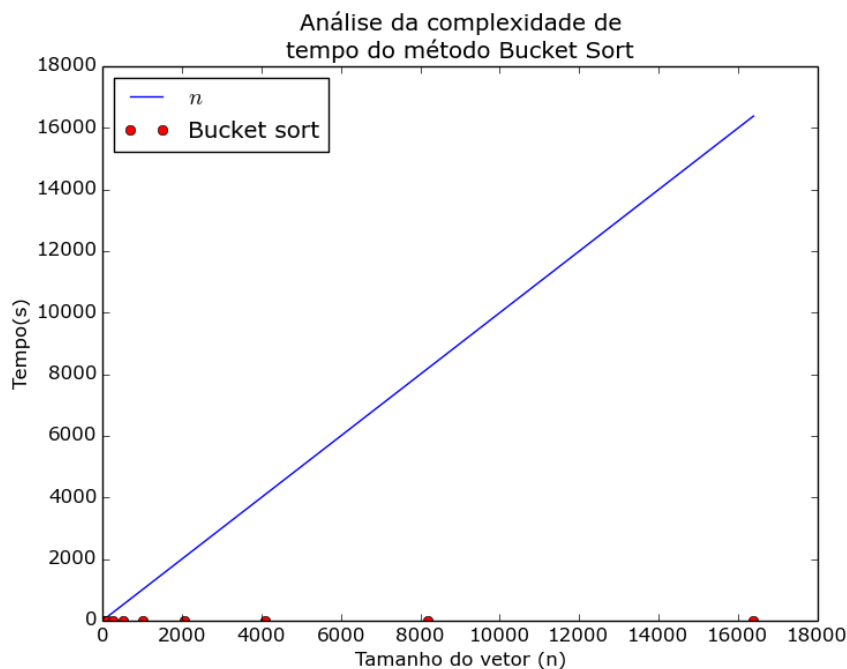


Figura 2.5: Complexidade de tempo do método Bucket Sort (Vetor Ordenado Decrescente)

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

4. Para um vetor parcialmente ordenado crescente

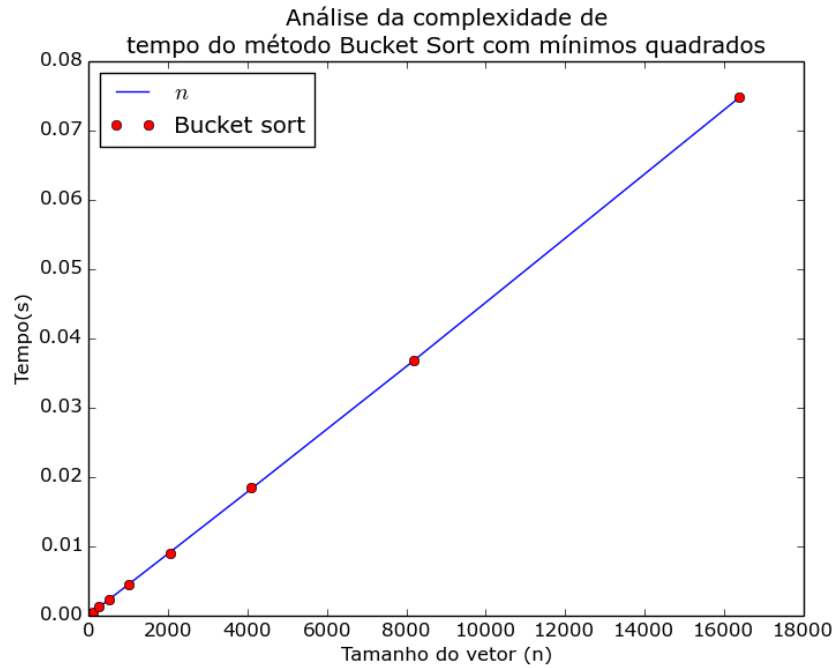


Figura 2.6: Complexidade de tempo do método Bucket Sort com mínimos quadrados (Vetor Ordenado Decrescente)

- (a) Complexidade de tempo do método Bucket Sort disponível na lista de imagens [2.7](#).

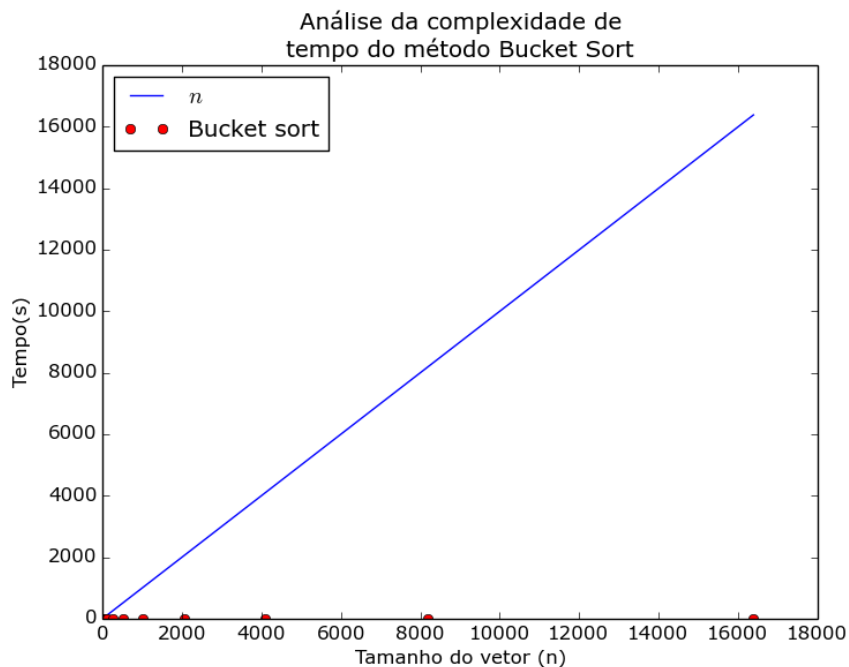


Figura 2.7: Complexidade de tempo do método Bucket Sort (Vetor Parcialmente Ordenado Crescente)

- (b) Complexidade de tempo do método Bucket Sort com mínimos quadrados disponível na lista de imagens [2.8](#).

5. Para um vetor parcialmente ordenado decrescente

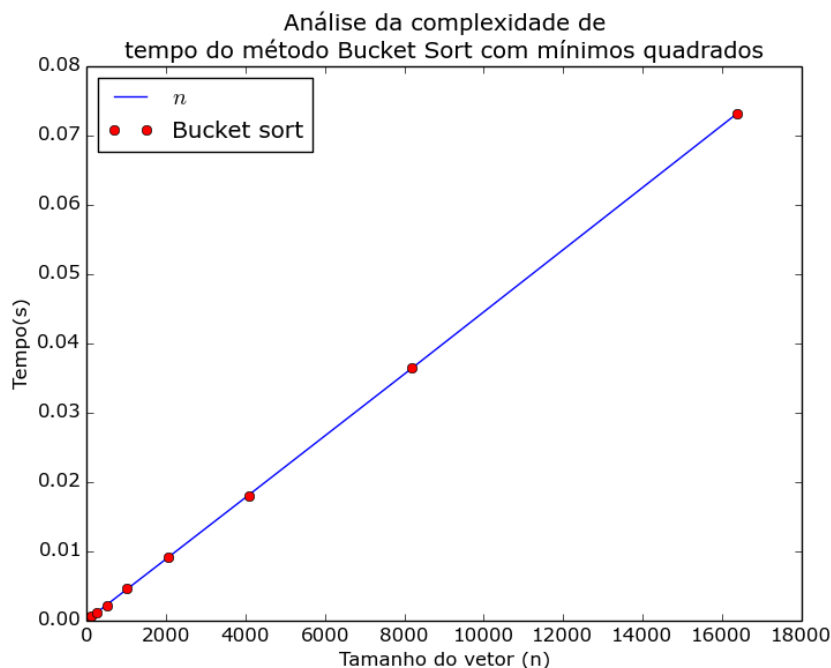


Figura 2.8: Complexidade de tempo do método Bucket Sort com mínimos quadrados (Vetor Parcialmente Ordenado Crescente)

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

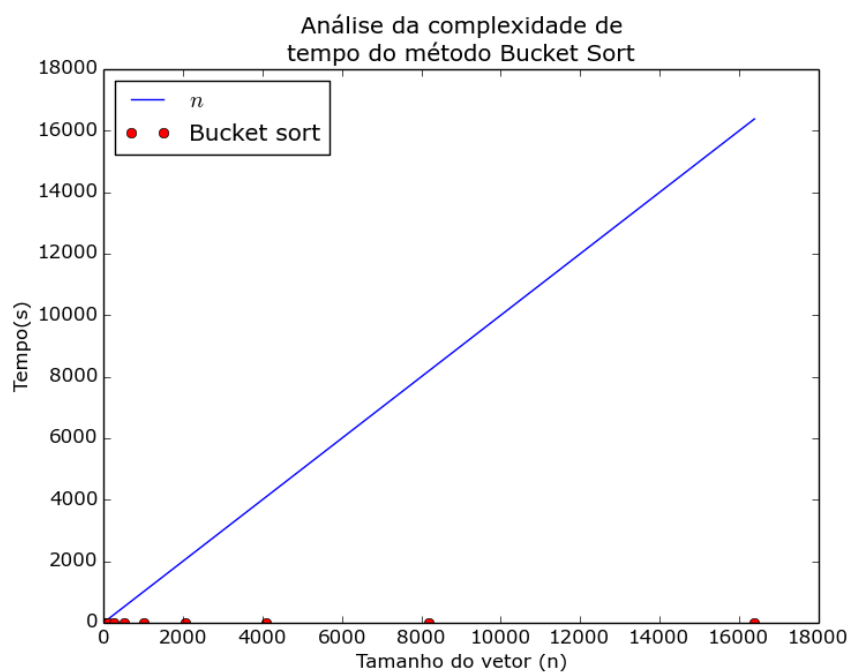


Figura 2.9: Complexidade de tempo do método Bucket Sort (Vetor Parcialmente Ordenado Decrescente)

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

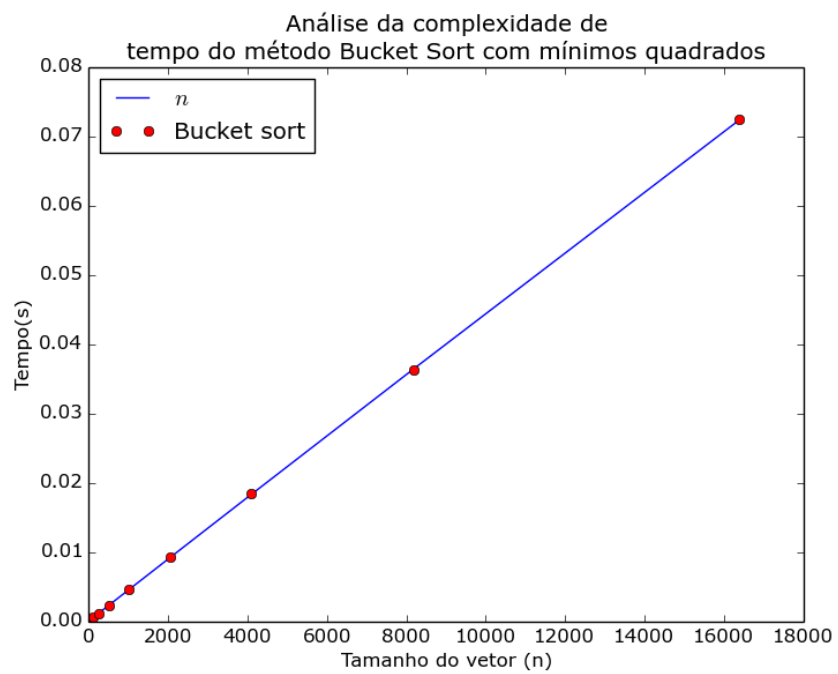


Figura 2.10: Complexidade de tempo do método Bucket Sort com mínimos quadrados (Vetor Parcialmente Ordenado Decrescente)

Capítulo 3

Tabelas

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

Tabela 3.1: *Vetor Aleatório*

Tamanho do Vetor	Tempo(s)
32	0.000139
64	0.000276
128	0.000572
256	0.001063
512	0.002014
1024	0.004126
2048	0.008782
4096	0.016970
8192	0.034664
16384	0.070438

Tabela 3.2: *Vetor Ordenado Crescente*

Tamanho do Vetor	Tempo(s)
32	0.000161
64	0.000301
128	0.000695
256	0.001160
512	0.002488
1024	0.004466
2048	0.009608
4096	0.018747
8192	0.036629
16384	0.072537

Tabela 3.3: *Vetor Ordenado Decrescente*

Tamanho do Vetor	Tempo(s)
32	0.000151
64	0.000289
128	0.000565
256	0.001332
512	0.002330
1024	0.004517
2048	0.009023
4096	0.018452
8192	0.036754
16384	0.074803

Tabela 3.4: *Vetor Parcialmente Ordenado Decrescente*

Tamanho do Vetor	Tempo(s)
32	0.000177
64	0.000294
128	0.000609
256	0.001150
512	0.002376
1024	0.004624
2048	0.009367
4096	0.018495
8192	0.036341
16384	0.072448

Tabela 3.5: *Vetor Parcialmente Ordenado Crescente*

Tamanho do Vetor	Tempo(s)
32	0.000186
64	0.000301
128	0.000589
256	0.001151
512	0.002246
1024	0.004608
2048	0.009222
4096	0.017981
8192	0.036525
16384	0.073222

Tabela 3.6: *Dados para Análise de Memória*

Tamanho do Vetor	Memória (MiB)
32	24.129000
64	24.121000
128	23.102000
256	24.141000
512	24.051000
1024	24.230000
2048	23.996000
4096	24.230000
8192	24.203000
16384	25.723000

Capítulo 4

Análise

O Bucket Sort é um dos algoritmos analisados em nosso trabalho que não utiliza de comparações para atingir seu objetivo. No entanto, ele possui algumas restrições para poder ser utilizado. Uma dessas restrições é de que os elementos do arranjo devem ser números reais no intervalo $[0,1)$. Além disso, como se pode observar nos dados coletados, ele não é um algoritmo in-place, ou seja, utiliza memória adicional da ordem de $\theta(n)$.

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 Bucket 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 Bucket 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

```
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/Bucket') ## adicionei o código de ordenação
16 sys.path.append('/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
    relatorio/Resultados/Bucket') ## adicionei o resultado do executa_teste
17
18
19 def executa_teste(arqteste, arqsaida, nlin, intervalo):
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: 14
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     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} {: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))

```

A.1

```
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_descrescente.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_decrecente.png')
148     plt.show()
149
150 #plota_teste1("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
               relatorio/Resultados/Selection/tSelection_vetor_ordenado_descrescente.
               dat")
151 plota_teste2("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
               relatorio/Resultados/Bucket/
               tBucket_vetor_parcialmente_ordenado_decrecente.dat")
152 plota_teste3("/home/gmarson/Git/AnaliseDeAlgoritmos/Trabalho_Final/
               relatorio/Resultados/Bucket/
               tBucket_vetor_parcialmente_ordenado_decrecente.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')

```


A.1

```
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()
```
