Análise experimental de algoritmos usando Python

Karen Catiguá Junqueira
karen@ufu.com

Matheus Prado Prandini Faria
matheus_prandini@ufu.com

Pedro Augusto Correa Braz
pedro_acbraz@hotmail.com

Faculdade de Computação
Universidade Federal de Uberlândia

16 de dezembro de 2016

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Sumário

Capítulo 1

Análise

O Bucket Sort é considerado um algoritmo de ordenação não baseado em comparações. Considera-se seu uso quando os elementos de entrada estão distribuídos uniformemente no intervalo $(0, 1]$. A ideia deste algoritmo é dividir o intervalo em n segmentos de mesmo tamanho, denominados bucket, e distribuir os n elementos nos respectivos segmentos. Como os elementos estão distribuídos de forma uniforme, então espera-se que o número de elementos em cada segmento seja aproximadamente o mesmo. Logo após essa etapa, os elementos são ordenados em cada segmento por um método qualquer, e, finalmente, esses segmentos são concatenados em ordem crescente.

Como espera-se que os n segmentos tenham aproximadamente a mesma quantidade de elementos, essa quantidade será pequena. E para vetores pequenos, temos que o uso do método de ordenação Insertion Sort é bastante eficiente. Dessa forma, como depende do tempo do algoritmo citado, espera-se que a complexidade de tempo do Bucket Sort seja linear, $teta(n)$. Entretanto, sua complexidade de tempo é $teta(n^2)$ no pior caso.

Em termos de memória, o Bucket Sort tem complexidade de espaço linear, $teta(n)$.

Capítulo 2

Resultados

2.1 Tabelas

n	comparações	tempo(s)
32	1	0.000305
64	1	0.000544
128	1	0.000994
256	1	0.001953
512	1	0.003745
1024	1	0.007638
2048	1	0.014962
4096	1	0.029871
8192	1	0.059436

Tabela 2.1: *Bucket com Vetores Aleatorio*

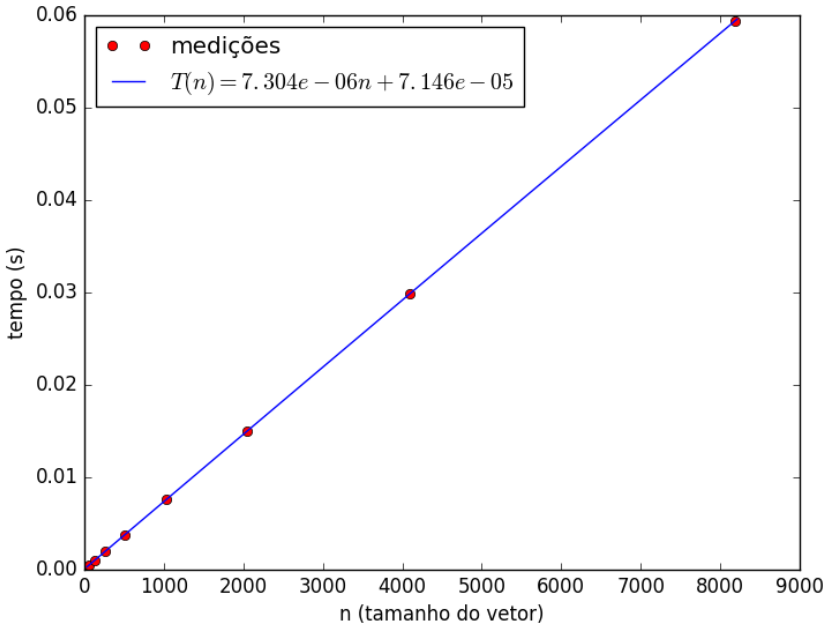

Figura 2.1: *Bucket com relação ao tempo com vetor aleatório.*

Análise com relação ao tamanho do vetor 2^{32}:

$7.304 * 10^{-6} * n + 7.146 * 10^{-5} = 31370.4412$

2.1

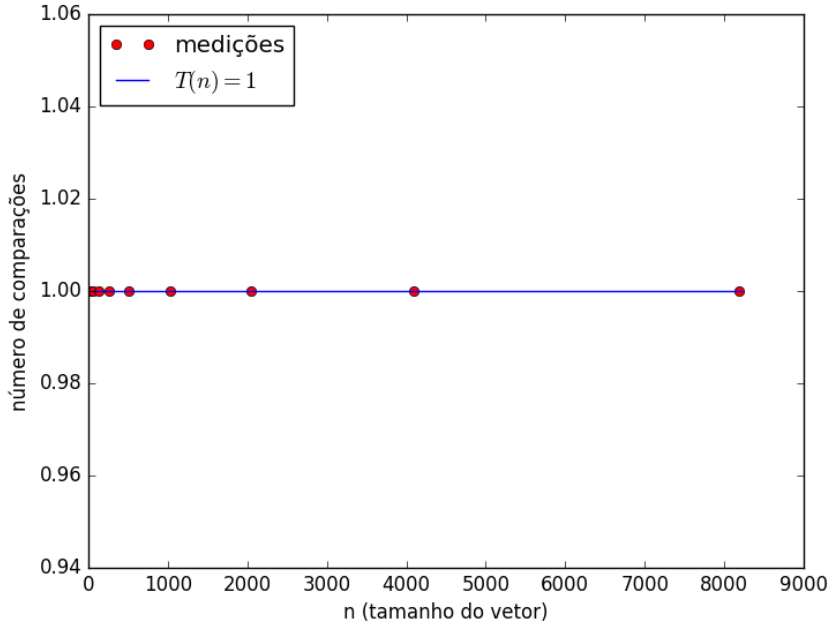

Figura 2.2: *Bucket com relação ao número de comparações com vetor aleatório.*

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000307
64	1	0.000525
128	1	0.001018
256	1	0.001962
512	1	0.003802
1024	1	0.007623
2048	1	0.014908
4096	1	0.029729
8192	1	0.060464

Tabela 2.2: *Bucket com Vetores Crescentes*

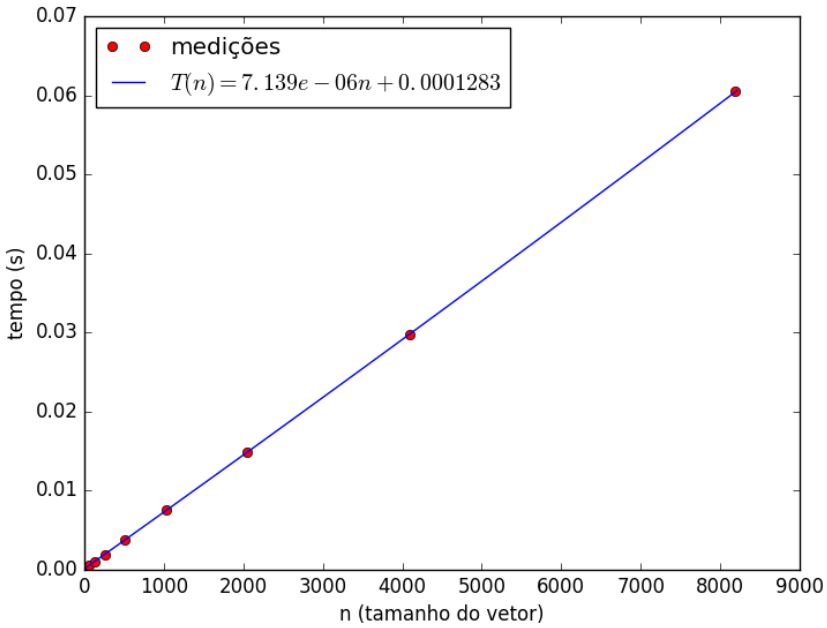

Figura 2.3: *Bucket com relação ao tempo com vetor crescente.*

Análise com relação ao tamanho do vetor 2^{32}:

$7.139 * 10^{-6} * n + 0.0001283 = 30661.77153$

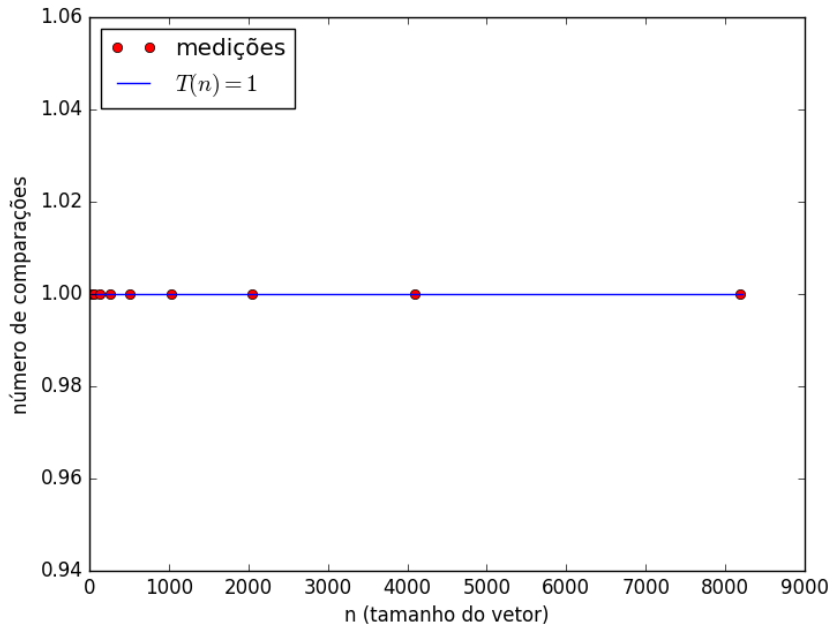

Figura 2.4: *Bucket com relação ao número de comparações com vetor crescente.*

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000296
64	1	0.000528
128	1	0.001017
256	1	0.001899
512	1	0.003788
1024	1	0.007836
2048	1	0.015064
4096	1	0.029404
8192	1	0.058275

Tabela 2.3: *Bucket com Vetores Decrescentes*

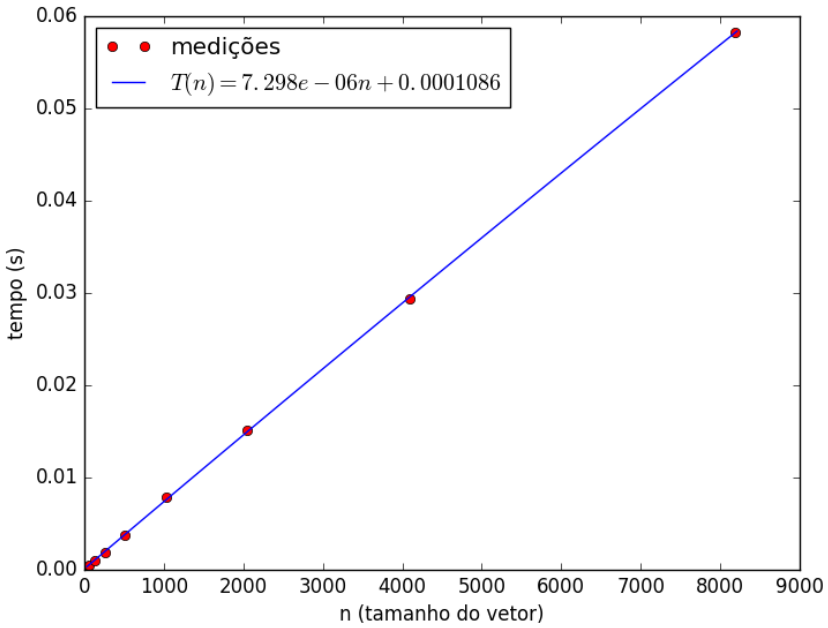

Figura 2.5: *Bucket com relação ao tempo com vetor decrescente.*

Análise com relação ao tamanho do vetor 2^{32}:

$7.298 * 10^{-6} * n + 0.0001086 = 31344.67133$

2.1

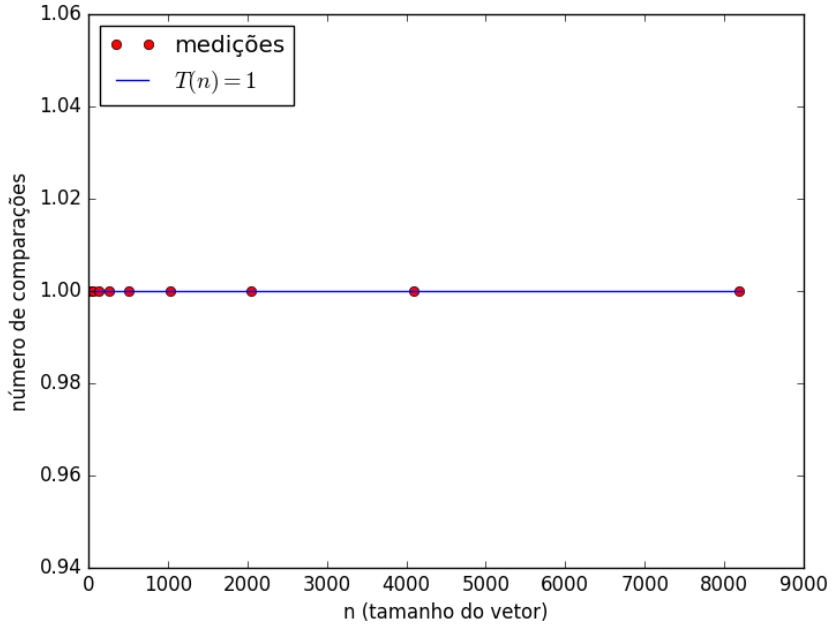

Figura 2.6: *Bucket com relação ao número de comparações com vetor decrescente.*

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000302
64	1	0.000554
128	1	0.000978
256	1	0.001915
512	1	0.003843
1024	1	0.007479
2048	1	0.015564
4096	1	0.029306
8192	1	0.058844

Tabela 2.4: *Bucket com Vetores Quase Crescentes* 10%

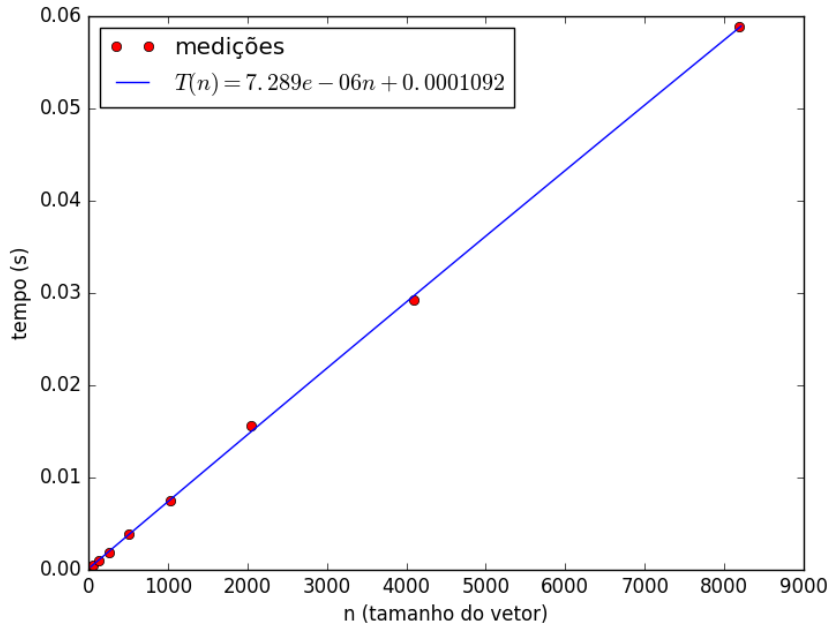

Figura 2.7: *bucket com relação ao tempo com vetor quase crescente* 10%.

Análise com relação ao tamanho do vetor 2^{32}:

$7.289 * 10^{-6} * n + 0.0001092 = 31306.01662$

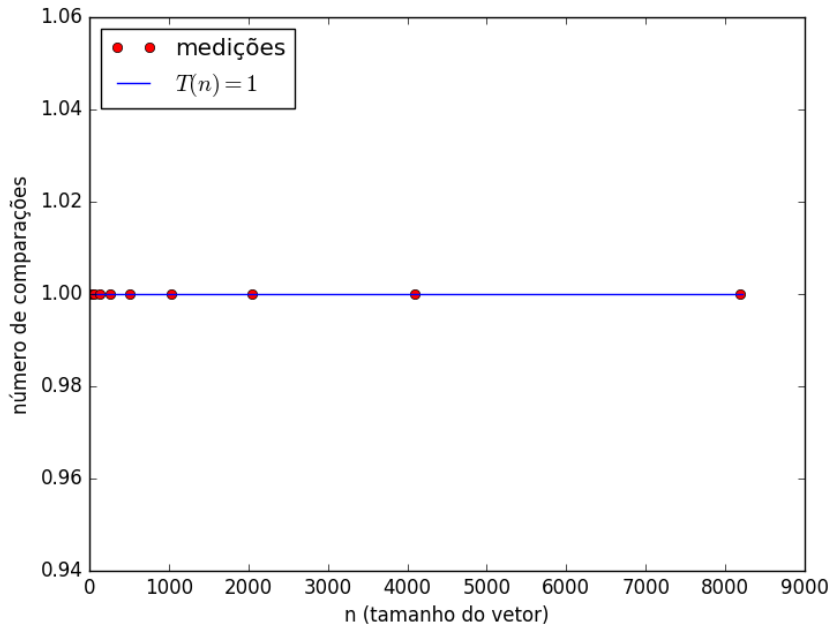

Figura 2.8: *Bucket com relação ao número de comparações com vetor quase crescente* 10%.

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000303
64	1	0.000576
128	1	0.001077

Tabela 2.5: *Bucket com Vetores Quase Crescentes* 20%

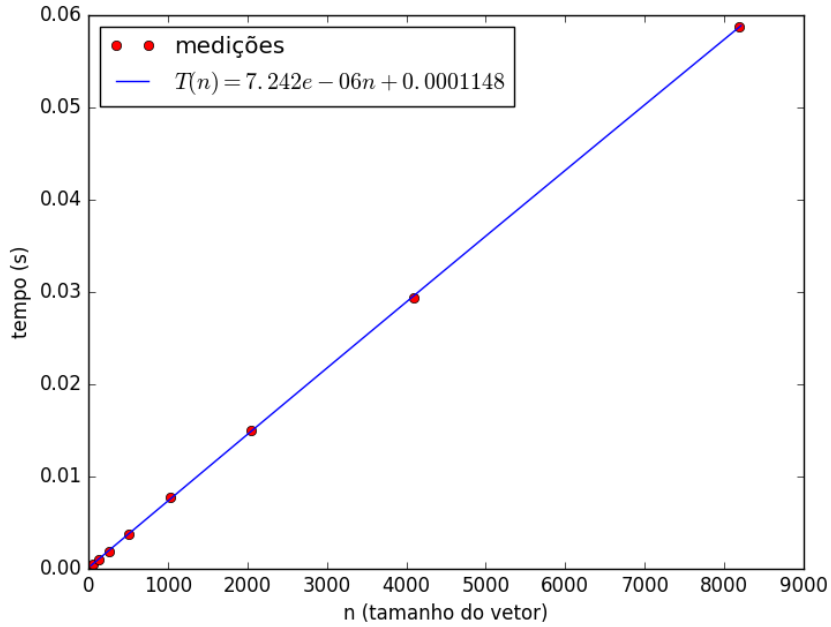

Figura 2.9: *Bucket com relação ao tempo com vetor quase crescente* 20%.

Análise com relação ao tamanho do vetor 2^{32}:

$7.242 * 10^{-6} * n + 0.0001148 = 31104.15316$

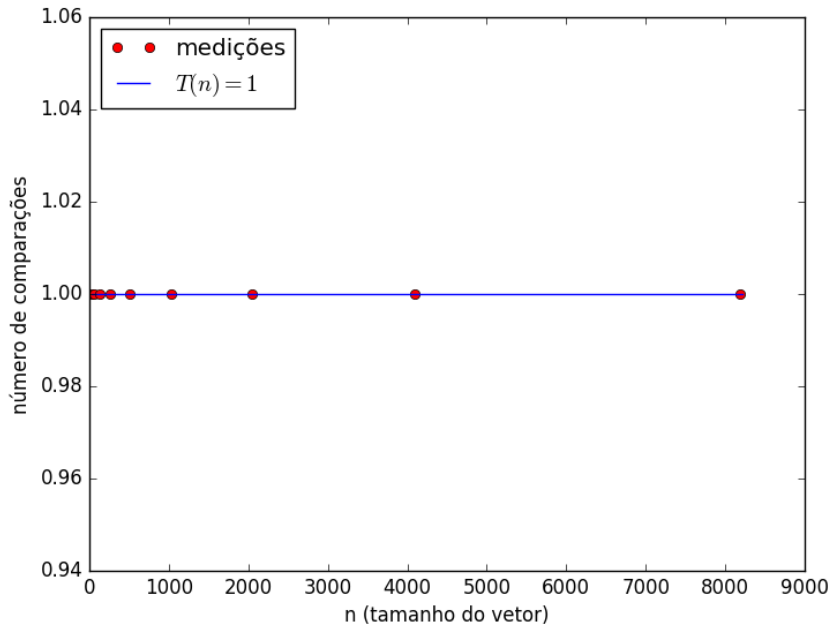

Figura 2.10: *Bucket com relação ao número de comparações com vetor quase crescente* 20%.

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000297
64	1	0.000533
128	1	0.000987
256	1	0.001978
512	1	0.003878
1024	1	0.007903
2048	1	0.015409
4096	1	0.029586
8192	1	0.059545

Tabela 2.6: *Bucket com Vetores Quase Crescentes* 30%

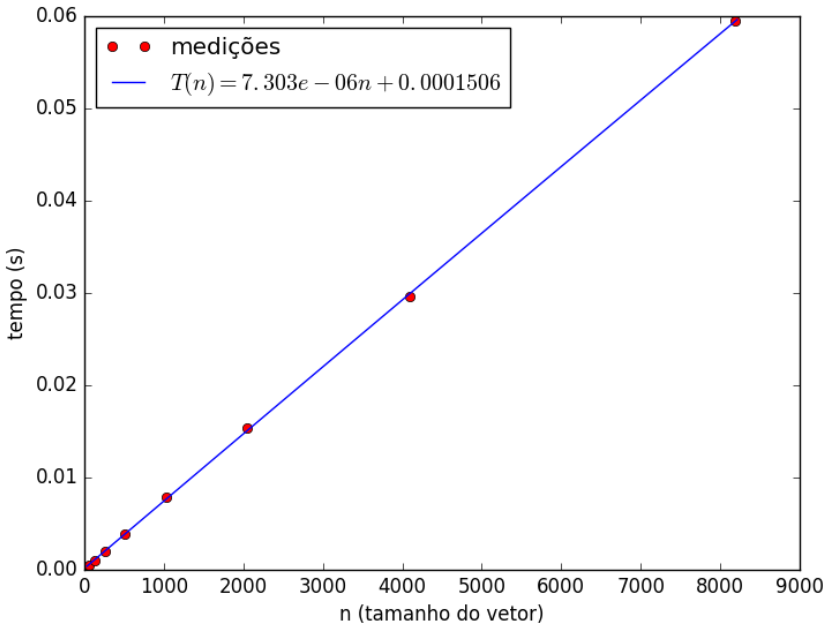

Figura 2.11: *Bucket com relação ao tempo com vetor quase crescente* 30%.

Análise com relação ao tamanho do vetor 2^{32}:

$7.303 * 10^{-6} * n + 0.0001506 = 31098.31823$

2.1

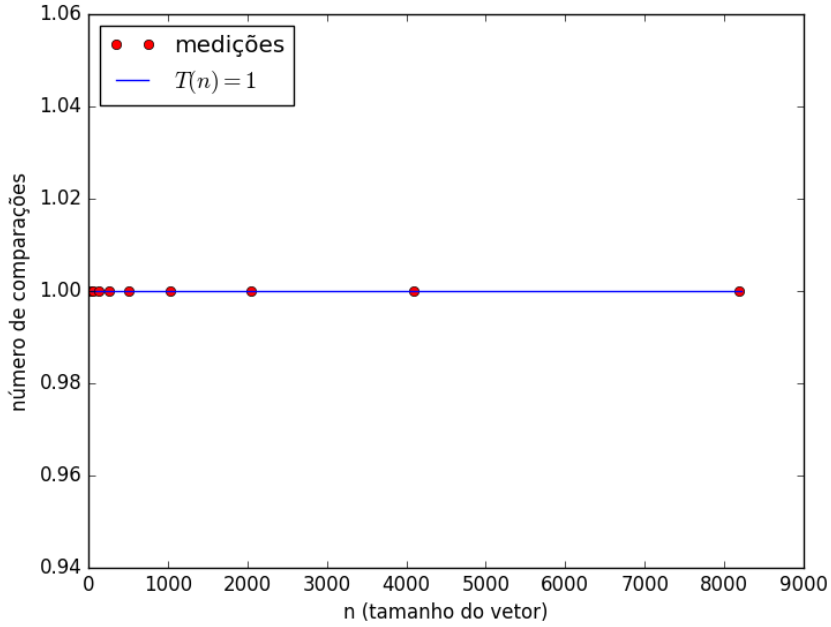

Figura 2.12: *Bucket com relação ao número de comparações com vetor quase crescente* 30%.

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000300
64	1	0.000525
128	1	0.000978
256	1	0.001901
512	1	0.003817
1024	1	0.007682
2048	1	0.015223
4096	1	0.029478
8192	1	0.060759

Tabela 2.7: *Bucket com Vetores Quase Crescentes* 40%

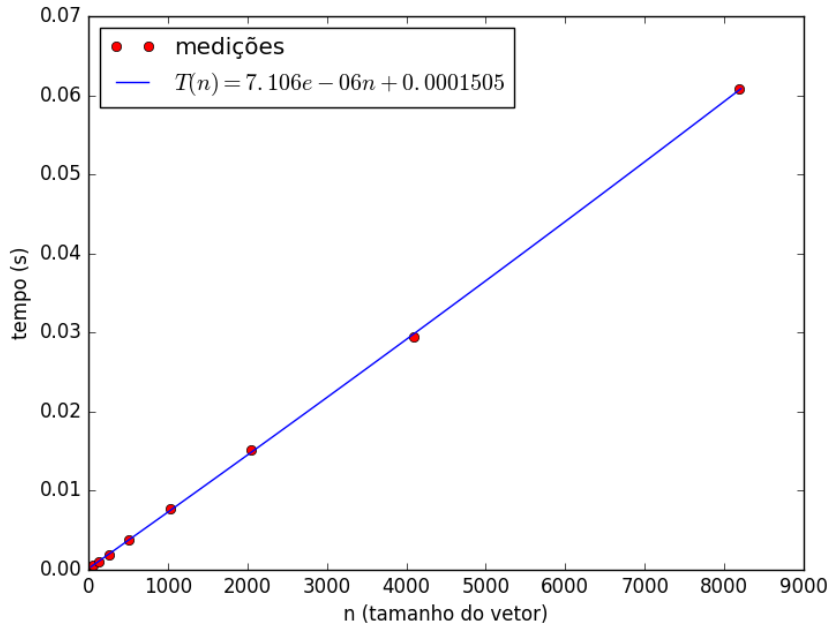

Figura 2.13: *Bucket com relação ao tempo com vetor quase crescente* 40%.

Análise com relação ao tamanho do vetor 2^{32}:

$7.106 * 10^{-6} * n + 0.0001505 = 30692.57120$

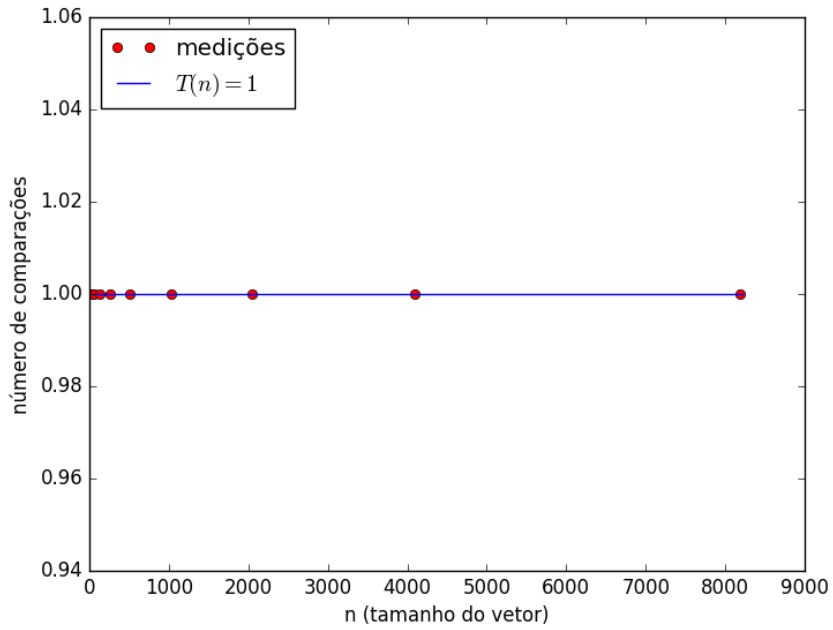

Figura 2.14: *Bucket com relação ao número de comparações com vetor quase crescente* 40%.

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000339
64	1	0.000537
128	1	0.000982

Tabela 2.8: *Bucket com Vetores Quase Crescentes* 50%

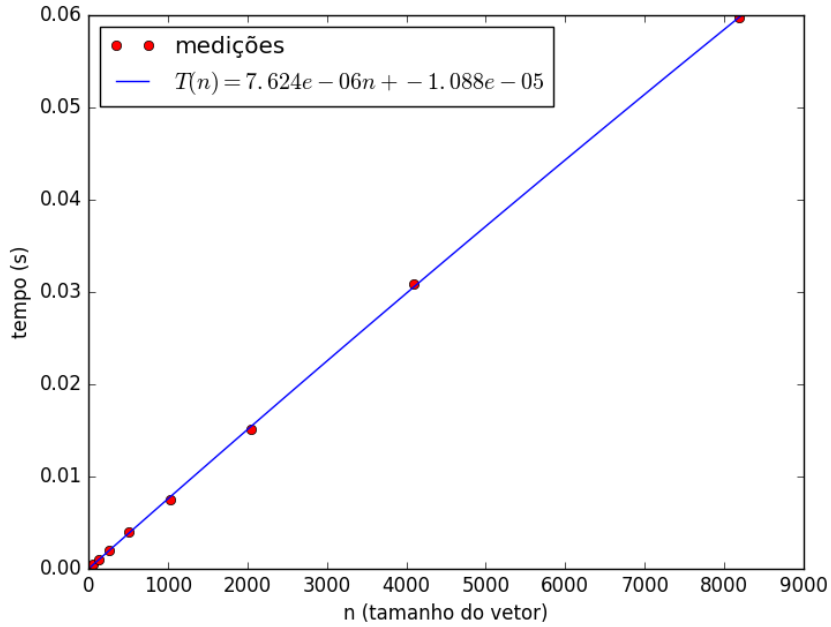

Figura 2.15: *Bucket com relação ao tempo com vetor quase crescente* 50%.

Análise com relação ao tamanho do vetor 2^{32}:

$7.642 * 10^{-6} * n - 1.088 * 10^{-5} = 32822.14007$

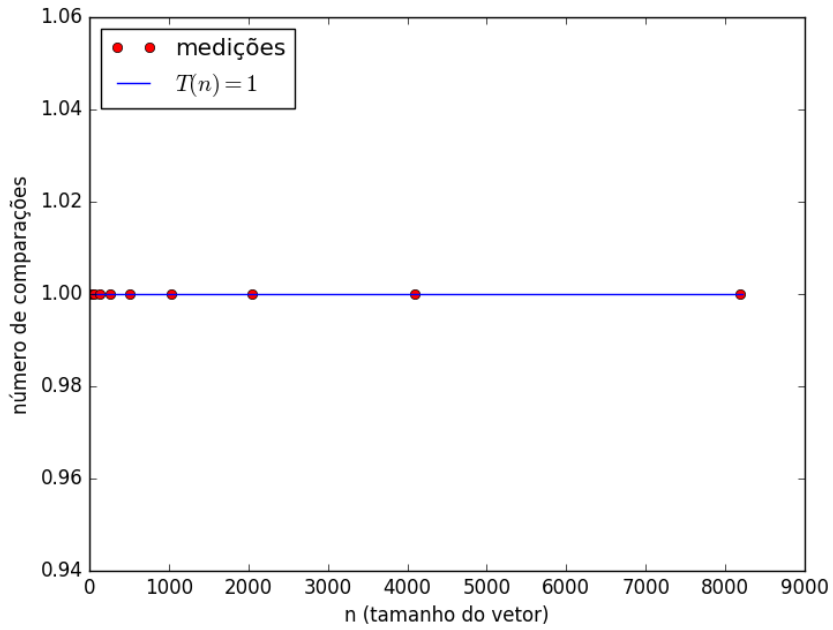

Figura 2.16: *Bucket com relação ao número de comparações com vetor quase crescente* 50%.

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000296
64	1	0.000532
128	1	0.001012
256	1	0.001966
512	1	0.003752
1024	1	0.007560
2048	1	0.015235
4096	1	0.030476
8192	1	0.059235

Tabela 2.9: *Bucket com Vetores Quase Decrescentes* 10%

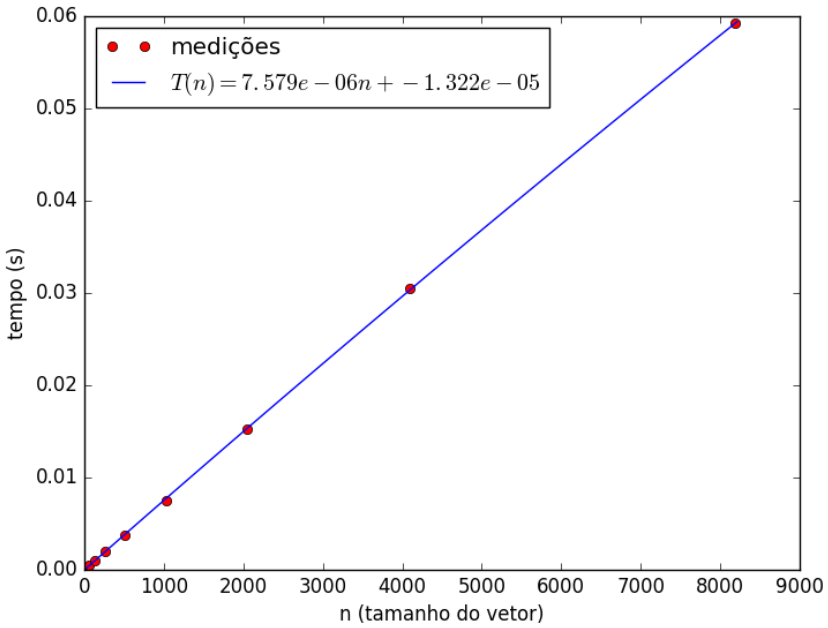

Figura 2.17: *Bucket com relação ao tempo com vetor quase decrescente* 10%.

Análise com relação ao tamanho do vetor 2^{32}:

$7.589 * 10^{-6} * n - 1.322 * 10^{-5} = 32594.5068$

2.1

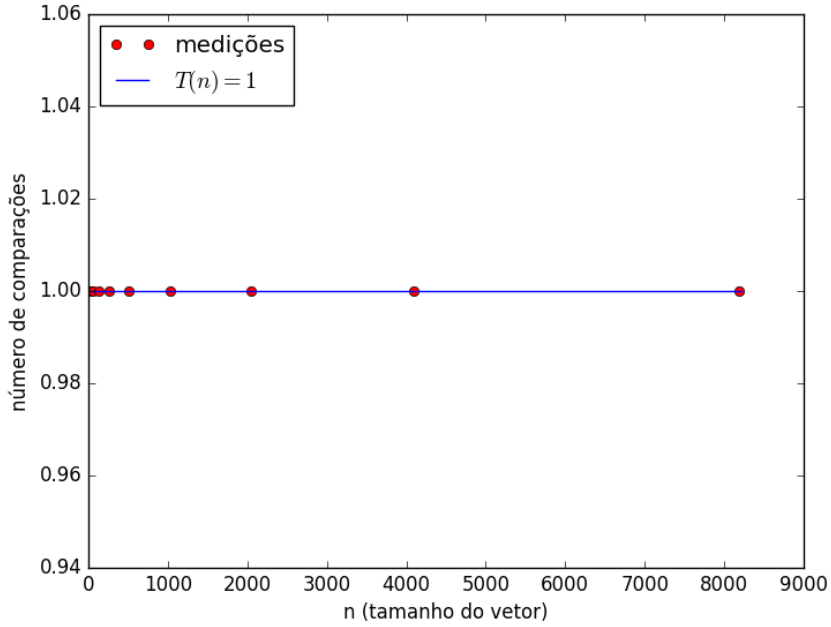

Figura 2.18: *Bucket com relação ao número de comparações com vetor quase decrescente* 10%.

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000299
64	1	0.000525
128	1	0.001007
256	1	0.001906
512	1	0.003722
1024	1	0.007548
2048	1	0.015134
4096	1	0.029732
8192	1	0.058867

Tabela 2.10: *Bucket com Vetores Quase Decrescentes* 20%

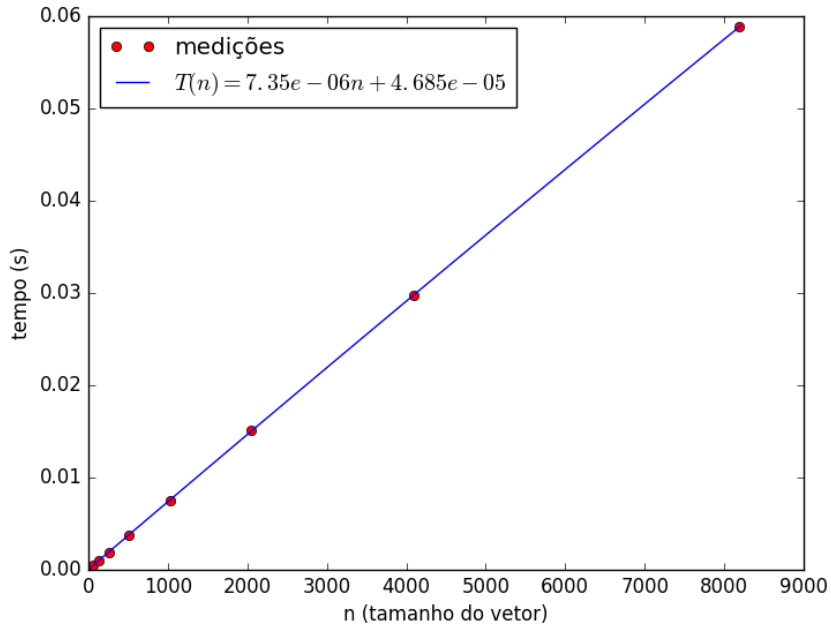

Figura 2.19: *Bucket com relação ao tempo com vetor quase decrescente* 20%.

Análise com relação ao tamanho do vetor 2^{32}:

$7.35 * 10^{-6} * n + 4.685 * 10^{-5} = 31568.00967$

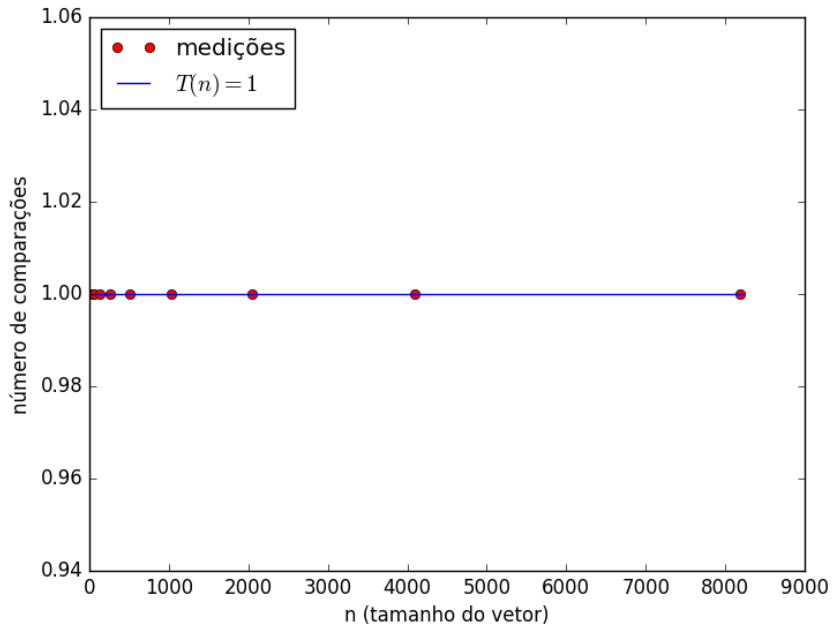

Figura 2.20: *Bucket com relação ao número de comparações com vetor quase decrescente* 20%.

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000311
64	1	0.000523
128	1	0.001047
256	1	0.001965
512	1	0.003893
1024	1	0.007608
2048	1	0.015381
4096	1	0.029609
8192	1	0.059256

Tabela 2.11: *Bucket com Vetores Quase Decrescentes* 30%

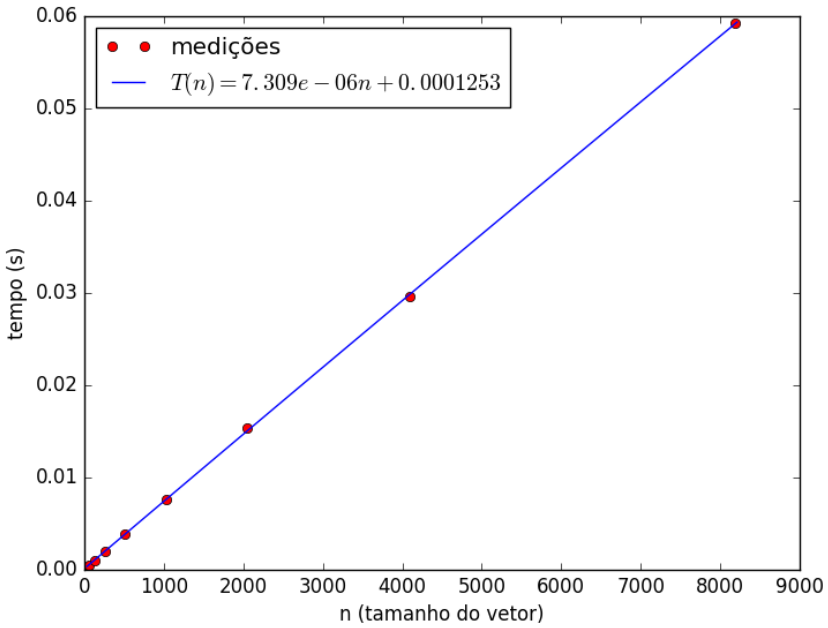

Figura 2.21: *Bucket com relação ao tempo com vetor quase decrescente* 30%.

Análise com relação ao tamanho do vetor 2^{32}:

$7.309 * 10^{-6} * n + 0.0001253 = 31391.91597$

2.1

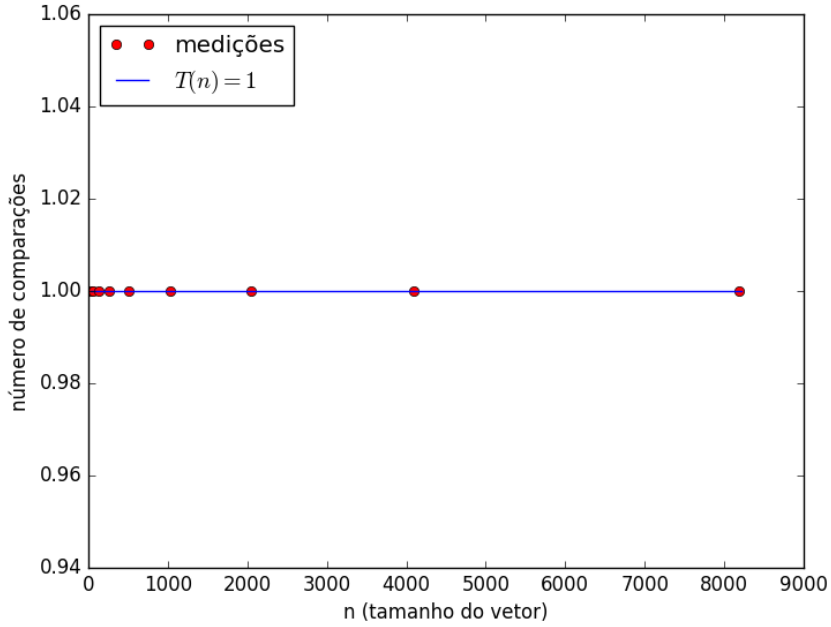

Figura 2.22: *Bucket com relação ao número de comparações com vetor quase decrescente* 30%.

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000326
64	1	0.000558
128	1	0.001042
256	1	0.002045
512	1	0.004139
1024	1	0.007858
2048	1	0.016271
4096	1	0.030146
8192	1	0.059188

Tabela 2.12: *Bucket com Vetores Quase Decrescentes* 40%

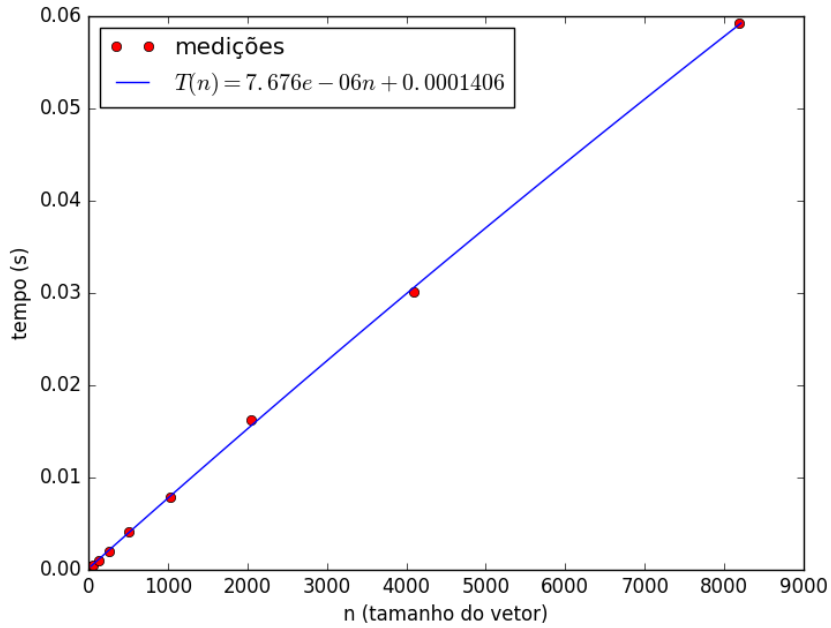

Figura 2.23: *Bucket com relação ao tempo com vetor quase decrescente* 40%.

Análise com relação ao tamanho do vetor 2^{32}:

$7.676 * 10^{-6} * n + 0.0001406 = 32968.16896$

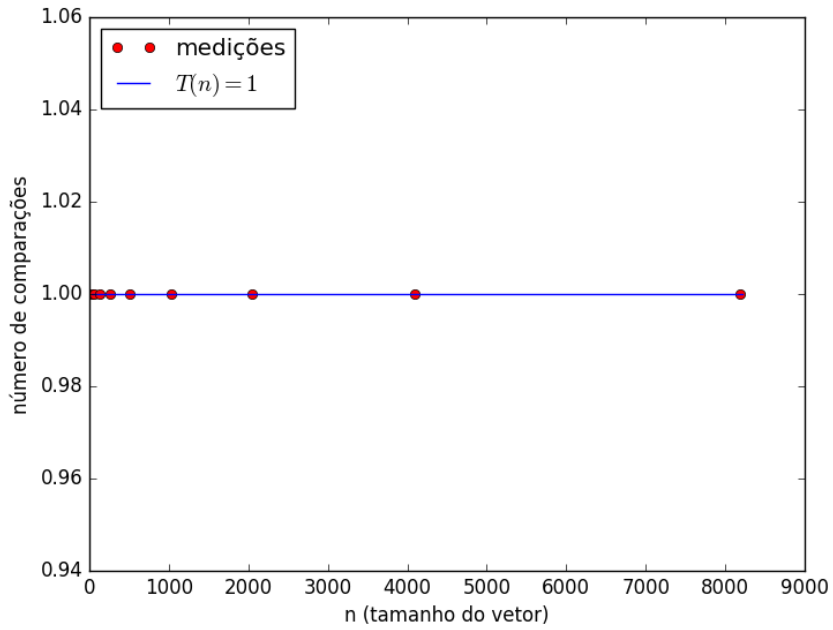

Figura 2.24: *Bucket com relação ao número de comparações com vetor quase decrescente* 40%.

Análise com relação ao tamanho do vetor 2^{32}:

1

n	comparações	tempo(s)
32	1	0.000305
64	1	0.000539
128	1	0.000979
256	1	0.002032
512	1	0.003806
1024	1	0.007525
2048	1	0.015194
4096	1	0.029464
8192	1	0.059404

Tabela 2.13: *Bucket com Vetores Quase Decrescentes* 50%

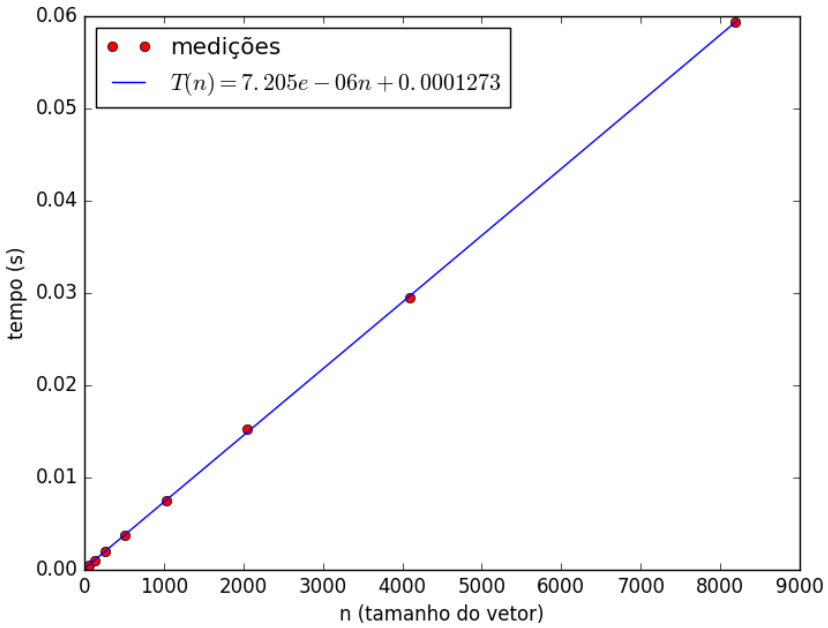

Figura 2.25: *Bucket com relação ao tempo com vetor quase decrescente* 50%.

Análise com relação ao tamanho do vetor 2^{32}:

$7.205 * 10^{-6} * n + 0.0001273 = 30945.23937$

2.1

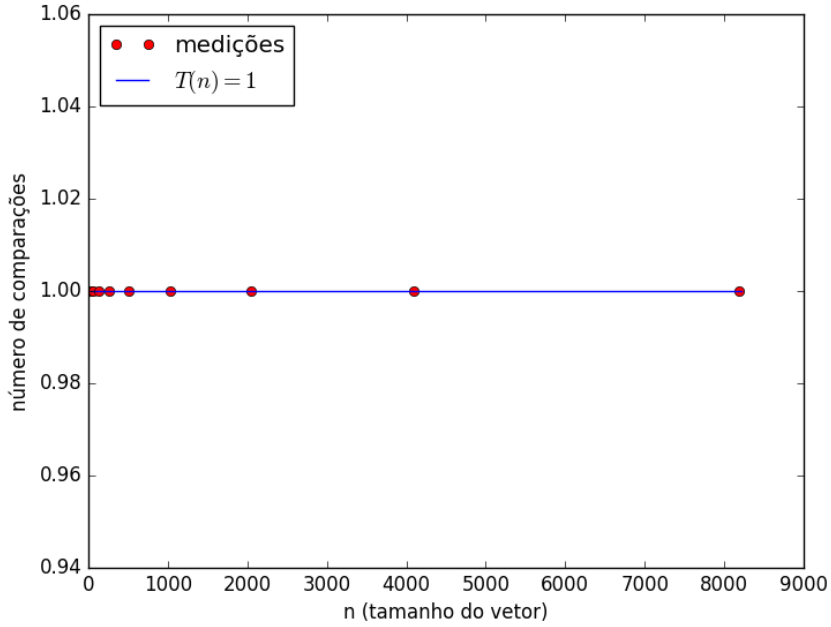

Figura 2.26: *Bucket com relação ao número de comparações com vetor quase decrescente* 50%.

Análise com relação ao tamanho do vetor 2^{32}:

1

Apêndice A

Arquivo ../bucket/bucket.py

Listagem A.1: ../bucket/bucket.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, aqui
     apenas
18 # para facilitar a análise de complexidade do Bucket Sort
19
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]))
```

Apêndice B

Arquivo ../bucket/ensaio.py

Listagem B.1: ../bucket/ensaio.py

```
1  import numpy as np
2  import argparse
3
4  from radixsort import *
5
6  parser = argparse.ArgumentParser()
7  parser.add_argument("arq_vetor",
8                      help="nome do arquivo contendo o vetor de teste")
9  args = parser.parse_args()
10
11 # Lê o arquivo contendo o vetor e passado na linha de comando como um
12 # vetor do Numpy.
13
14 vet = np.loadtxt(args.arq_vetor)
15 radix(vet)
```