Análise experimental de algoritmos usando Python

Patrícia Mariana Ramos Marcolino
pmrmarcolino@hotmail.com

Eduardo Pinheiro Barbosa
eduardptu@hotmail.com

Faculdade de Computação
Universidade Federal de Uberlândia

1 de julho de 2016

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Sumário

Capítulo 1

Análise

O algoritmo *Bucket Sort* possui tempo linear, desde que os valores a serem ordenados sejam distribuídos uniformemente sobre o intervalo $[0, 1)$.
O *Bucket Sort* divide o intervalo [0, 1) em n sub-intervalos iguais, denominados buckets (baldes), e então distribui os n números reais nos n buckets. Como a entrada é composta por dados distribuídos uniformemente, espera-se que cada balde possua, ao final deste processo, um número equivalente de elementos (usualmente 1).
Para obter o resultado, basta ordenar os elementos em cada bucket e então apresentá-los em ordem.
Ordena n números uniformemente distribuídos na faixa [0-1) em tempo médio O(n).
Sabendo que o bucket sort não trabalha com o método de comparação, todos os graficos respectivos a essa informação são constantes.

Capítulo 2

Resultados

2.1 Tabelas

n	comparações	tempo(s)
32	1	0.000494
64	1	0.001183
128	1	0.002272
256	1	0.005320
512	1	0.012027
1024	1	0.029675
2048	1	0.070583
4096	1	0.180627
8192	1	0.486444

Tabela 2.1: *Tabela com vetor teste aleatório: A linha te interesse analisada para este caso é a 13.*

n	comparações	tempo(s)
32	1	0.000412
64	1	0.000782
128	1	0.001439
256	1	0.002921
512	1	0.005770
1024	1	0.011244
2048	1	0.022422
4096	1	0.044494
8192	1	0.088436

Tabela 2.2: *Tabela com vetor teste crescente: A linha te interesse analisada para este caso é a 13.*

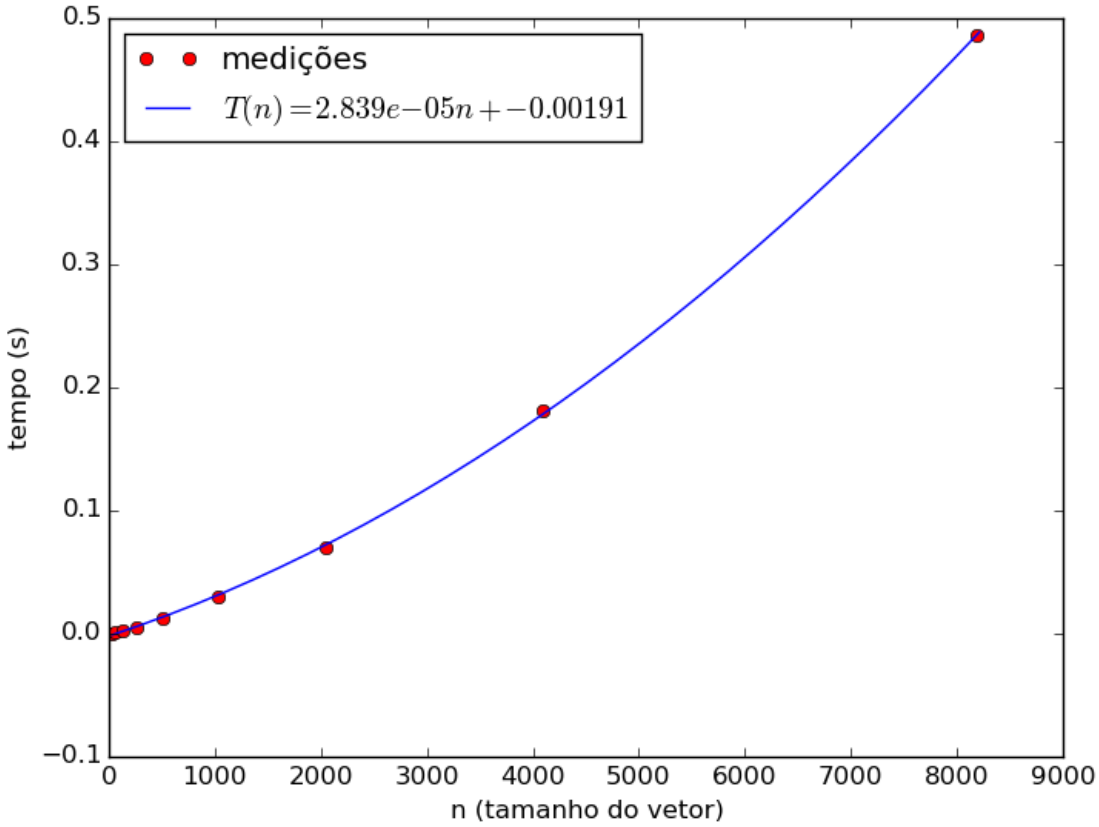

Figura 2.1: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 2.839\text{e} - 5 * n - 0.00191$ e para o $n = 2^{32}$,
$T(2^{32}) = 121934.11962344$

n	comparações	tempo(s)
32	1	0.000576
64	1	0.001213
128	1	0.002996
256	1	0.006872
512	1	0.018655
1024	1	0.044316
2048	1	0.127545
4096	1	0.306900
8192	1	0.830657

Tabela 2.3: *Tabela com vetor teste decrescente: A linha te interesse analisada para este caso é a 13*

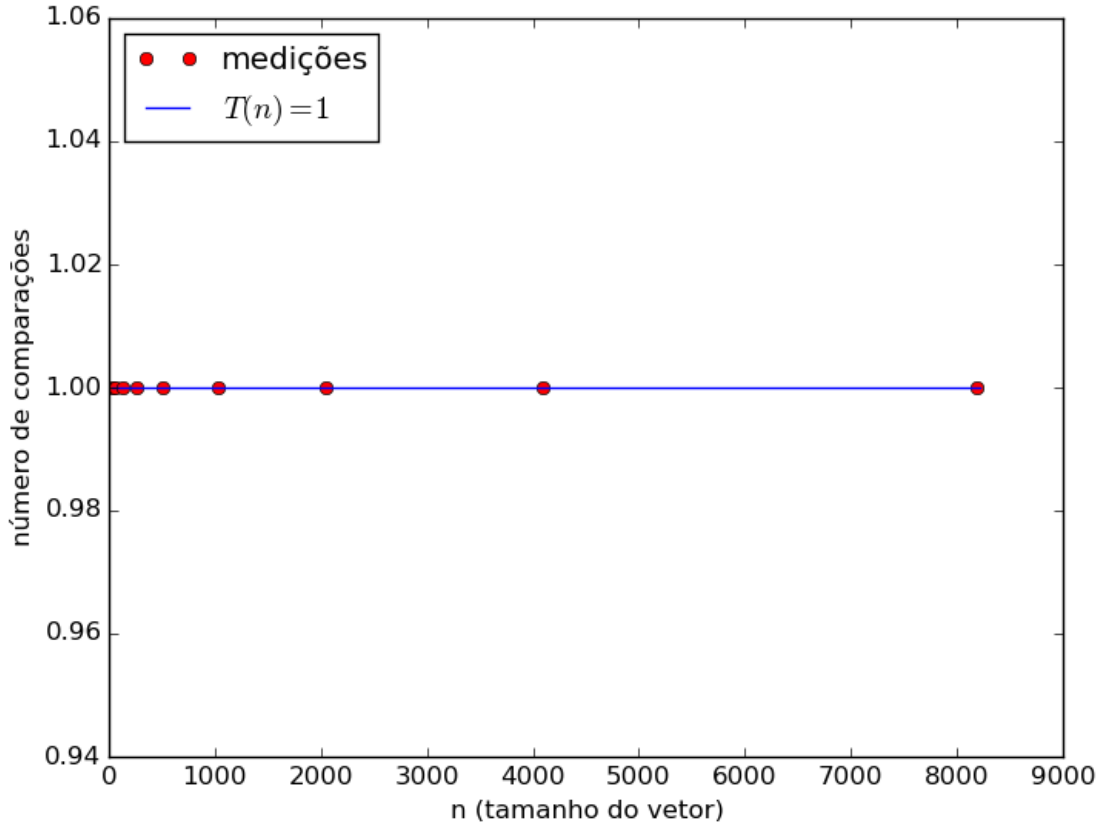

Figura 2.2: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 1$

n	comparações	tempo(s)
32	1	0.000420
64	1	0.000863
128	1	0.001523
256	1	0.003084
512	1	0.006400
1024	1	0.012465
2048	1	0.024938
4096	1	0.050542
8192	1	0.109208

Tabela 2.4: *Tabela com vetor teste quase crescente 10%: A linha te interesse analisada para este caso é a 13*

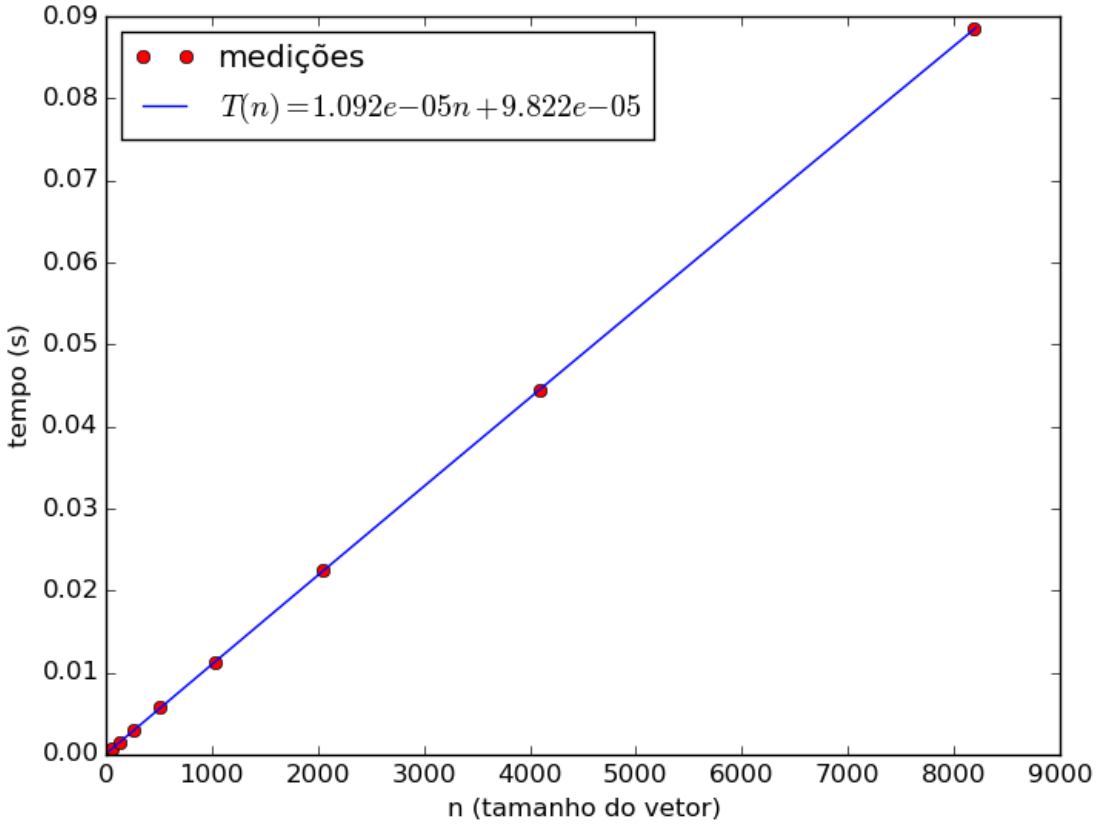

Figura 2.3: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 1.092\text{e} - 5 * n - 9.822\text{e} - 5$ e para o $n = 2^{32}$,
$T(2^{32}) = 46901.0427741$

n	comparações	tempo(s)
32	1	0.000424
64	1	0.000798
128	1	0.001573
256	1	0.003140
512	1	0.006396
1024	1	0.013214
2048	1	0.027804
4096	1	0.058344
8192	1	0.128207

Tabela 2.5: *Tabela com vetor teste quase crescente 20%: A linha te interesse analisada para este caso é a 13*

2.1

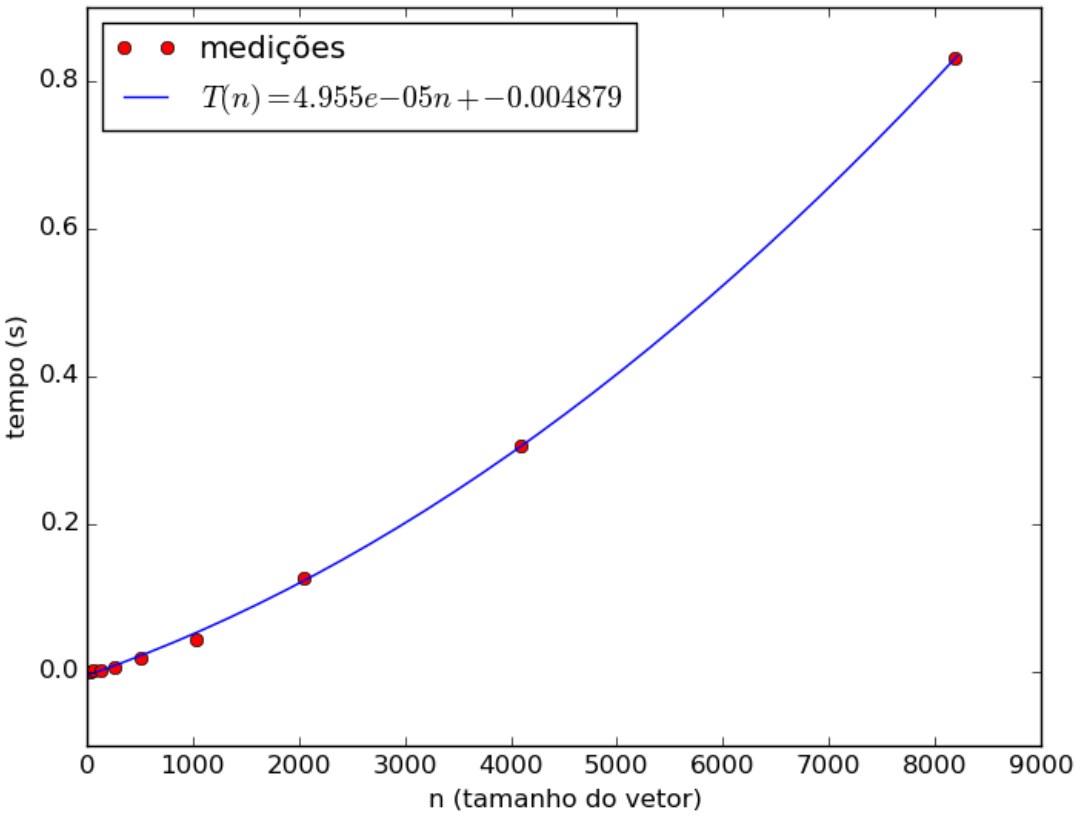

Figura 2.4: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 4.955\text{e} - 5 * n - 0.004879$ e para o $n = 2^{32}$,
$T(2^{32}) = 212815.6246378$

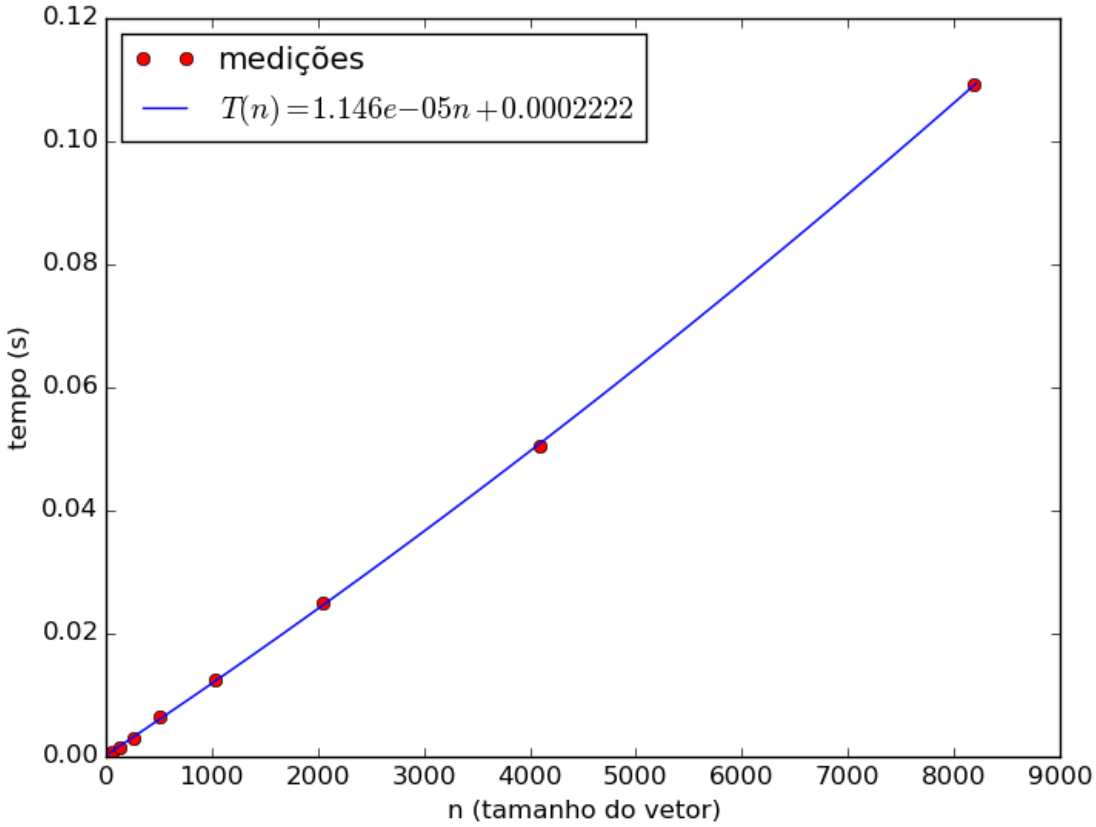

Figura 2.5: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 1.146\text{e} - 5 * n + 0.0002222$ e para o $n = 2^{32}$,
$T(2^{32}) = 49220.32543436$

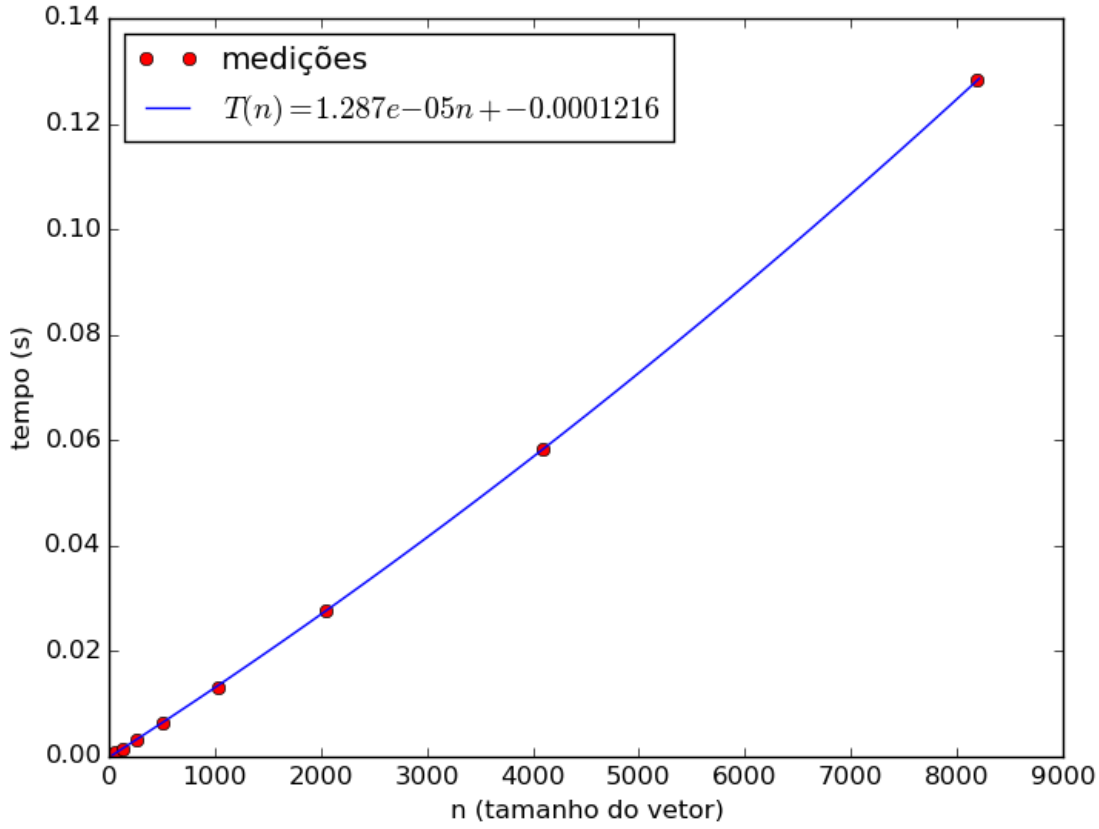

Figura 2.6: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 1.287\text{e} - 5 * n - 0.0001216$ e para o $n = 2^{32}$,
$T(2^{32}) = 55276.22897792$

n	comparações	tempo(s)
32	1	0.000424
64	1	0.000809
128	1	0.001623
256	1	0.003317
512	1	0.006570
1024	1	0.014242
2048	1	0.031286
4096	1	0.064335
8192	1	0.147849

Tabela 2.6: *Tabela com vetor teste quase crescente 30%: A linha te interesse analisada para este caso é a 13*

n	comparações	tempo(s)
32	1	0.000430
64	1	0.000783
128	1	0.001645
256	1	0.003366
512	1	0.007152
1024	1	0.014522
2048	1	0.033981
4096	1	0.073264
8192	1	0.165654

Tabela 2.7: *Tabela com vetor teste quase crescente 40%: A linha te interesse analisada para este caso é a 13*

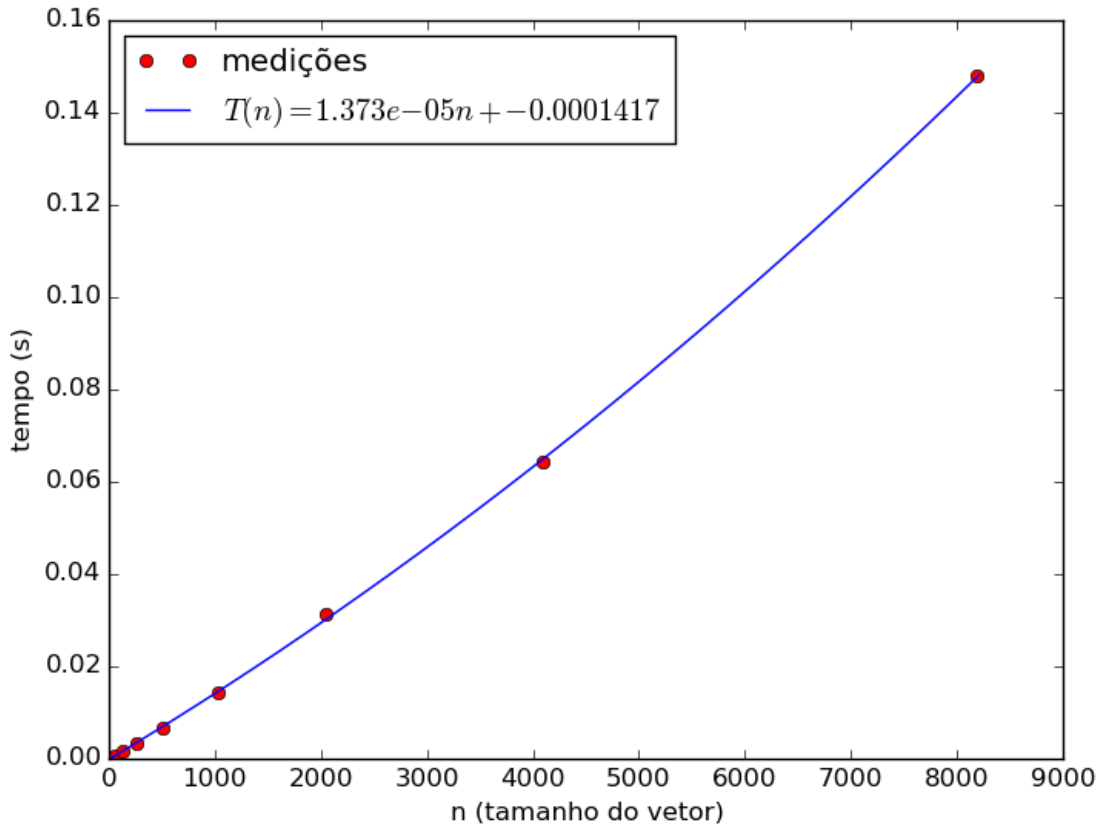

Figura 2.7: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 1.373\text{e} - 5 * n - 0.0001417$ e para o $n = 2^{32}$,
$T(2^{32}) = 58969.90083238$

n	comparações	tempo(s)
32	1	0.000449
64	1	0.000842
128	1	0.001675
256	1	0.003452
512	1	0.007282
1024	1	0.016236
2048	1	0.034841
4096	1	0.078142
8192	1	0.184326

Tabela 2.8: *Tabela com vetor teste quase crescente 50%: A linha te interesse analisada para este caso é a 13*

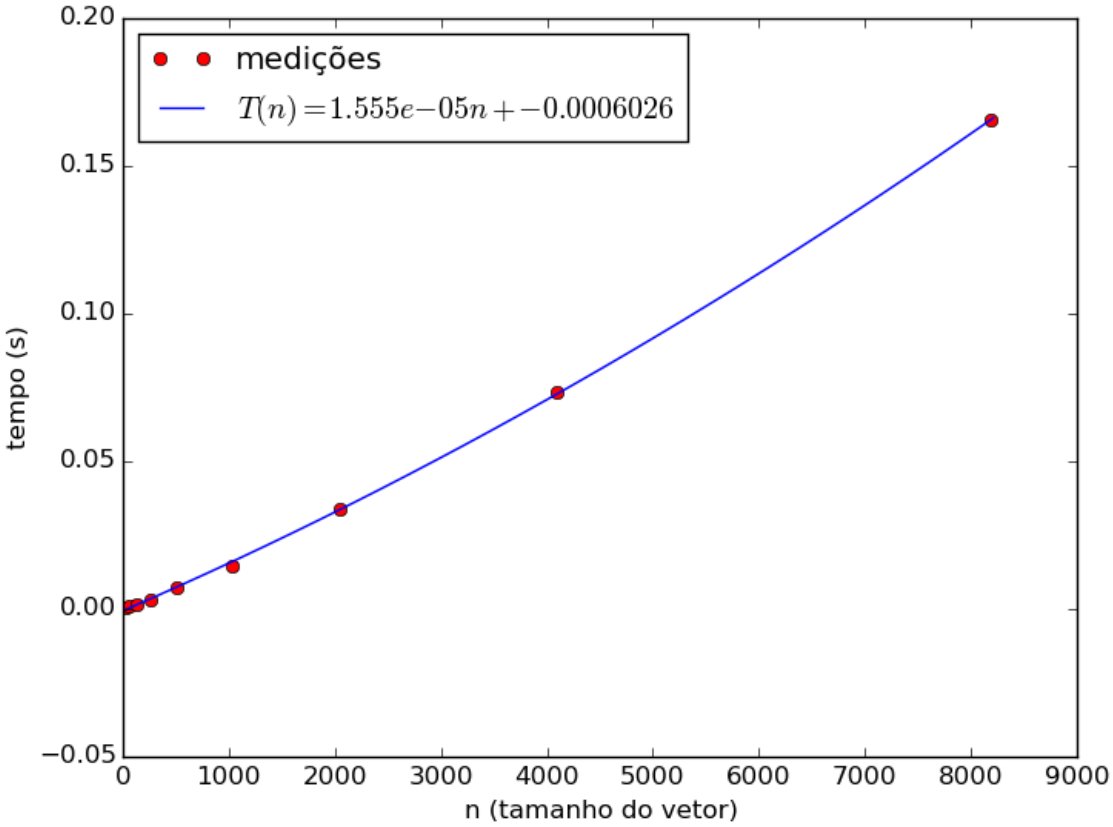

Figura 2.8: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 1.555\text{e} - 5 * n - 0.0006026$ e para o $n = 2^{32}$,
$T(2^{32}) = 66786.7408502$

n	comparações	tempo(s)
32	1	0.000571
64	1	0.001247
128	1	0.002872
256	1	0.006947
512	1	0.017190
1024	1	0.045480
2048	1	0.119944
4096	1	0.312863
8192	1	0.835964

Tabela 2.9: *Tabela com vetor teste quase decrescente 10%: A linha te interesse analisada para este caso é a 13*

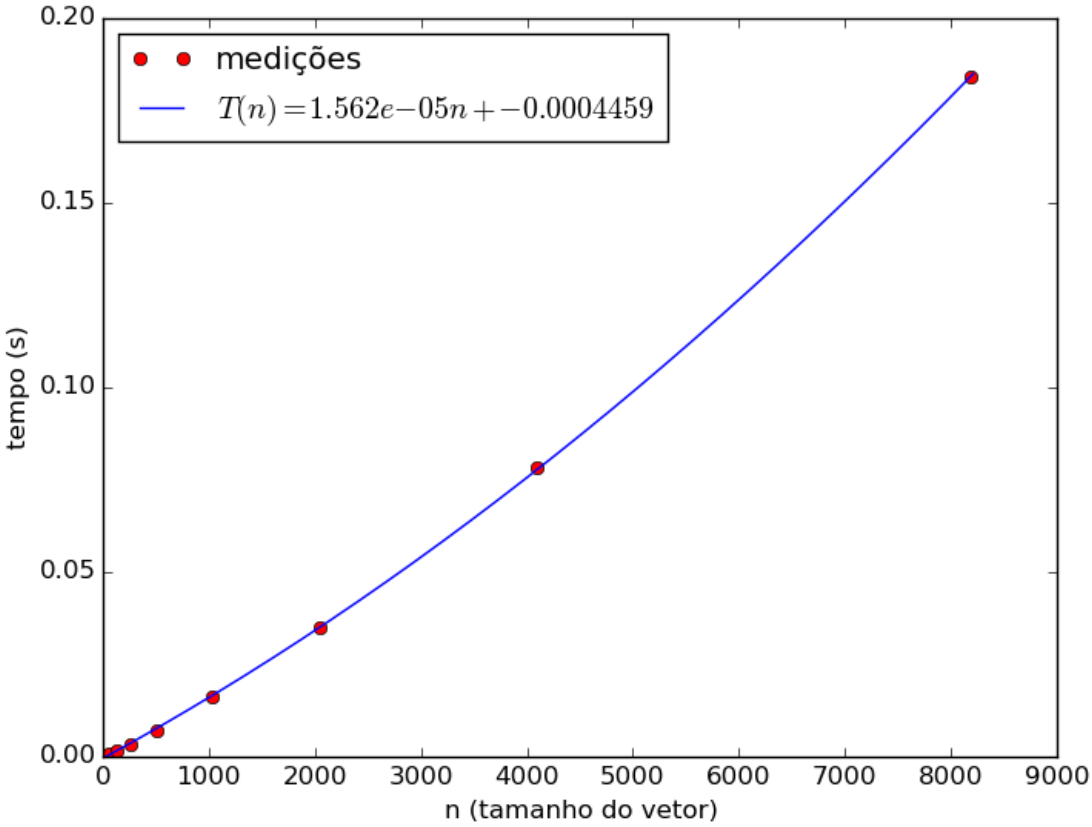

Figura 2.9: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 1.562\text{e} - 5 * n - 0.0004459$ e para o $n = 2^{32}$,
$T(2^{32}) = 67087.38871762$

n	comparações	tempo(s)
32	1	0.000553
64	1	0.001224
128	1	0.002883
256	1	0.006726
512	1	0.017088
1024	1	0.043727
2048	1	0.118664
4096	1	0.309639
8192	1	0.813655

Tabela 2.10: *Tabela com vetor teste quase decrescente 20%: A linha te interesse analisada para este caso é a 13*

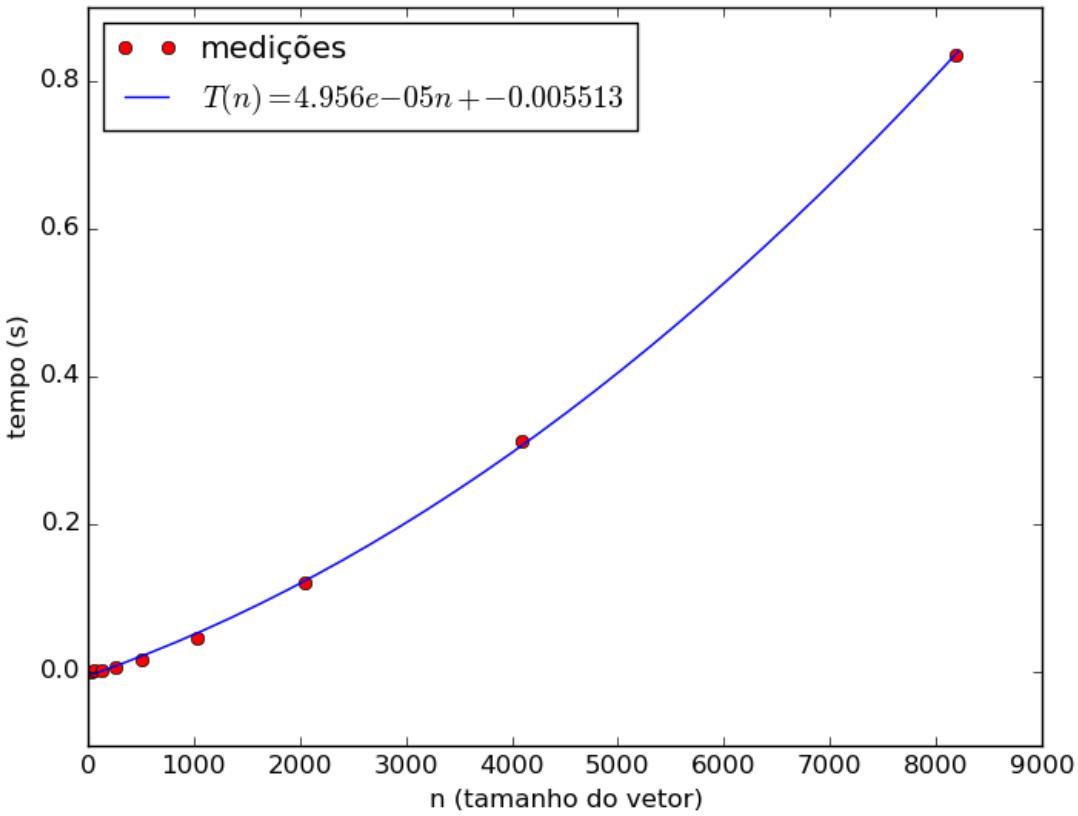

Figura 2.10: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 4.956\text{e} - 5 * n - 0.005513$ e para o $n = 2^{32}$,
$T(2^{32}) = 212858.57367676$

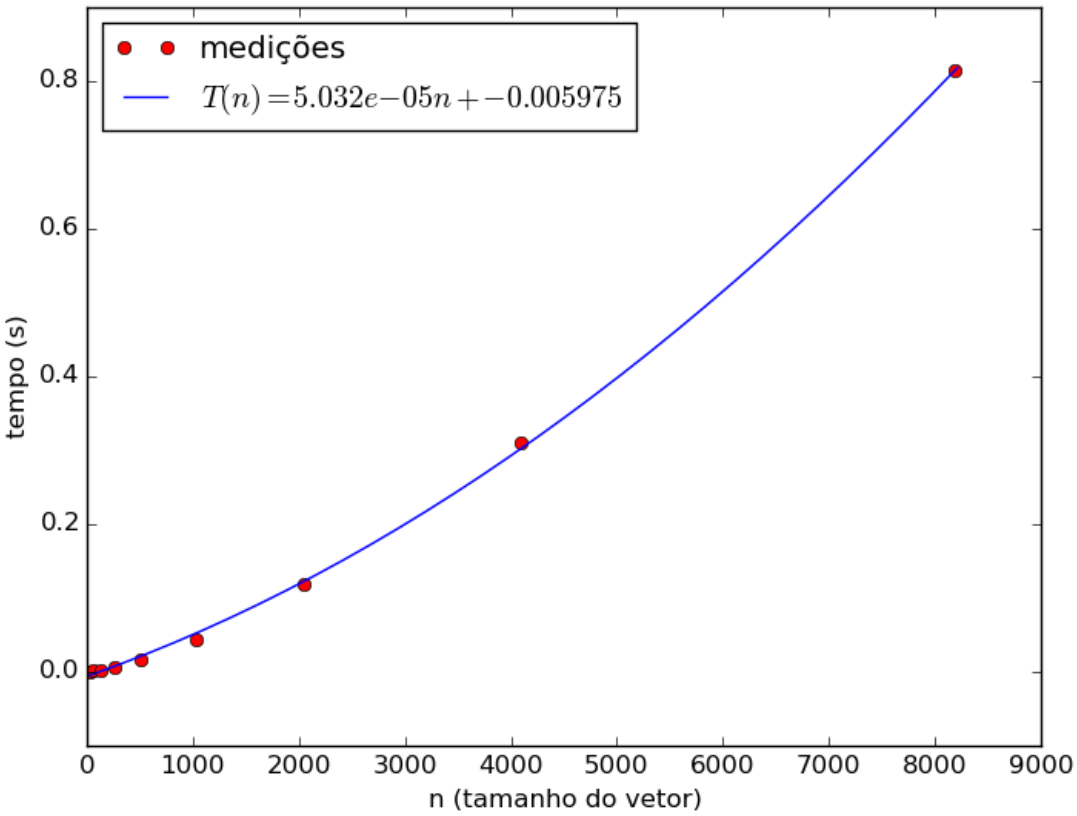

Figura 2.11: *EA análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 5.032\text{e} - 5 * n - 0.005975$ e para o $n = 2^{32}$,
$T(2^{32}) = 216122.74835972$

n	comparações	tempo(s)
32	1	0.000574
64	1	0.001189
128	1	0.002789
256	1	0.006847
512	1	0.016691
1024	1	0.044169
2048	1	0.112430
4096	1	0.294344
8192	1	0.805679

Tabela 2.11: *Tabela com vetor teste quase decrescente 30%: A linha te interesse analisada para este caso é a 13*

n	comparações	tempo(s)
32	1	0.000568
64	1	0.001186
128	1	0.002779
256	1	0.006866
512	1	0.017219
1024	1	0.042049
2048	1	0.110336
4096	1	0.303495
8192	1	0.806721

Tabela 2.12: *Tabela com vetor teste quase decrescente 40%: A linha te interesse analisada para este caso é a 13*

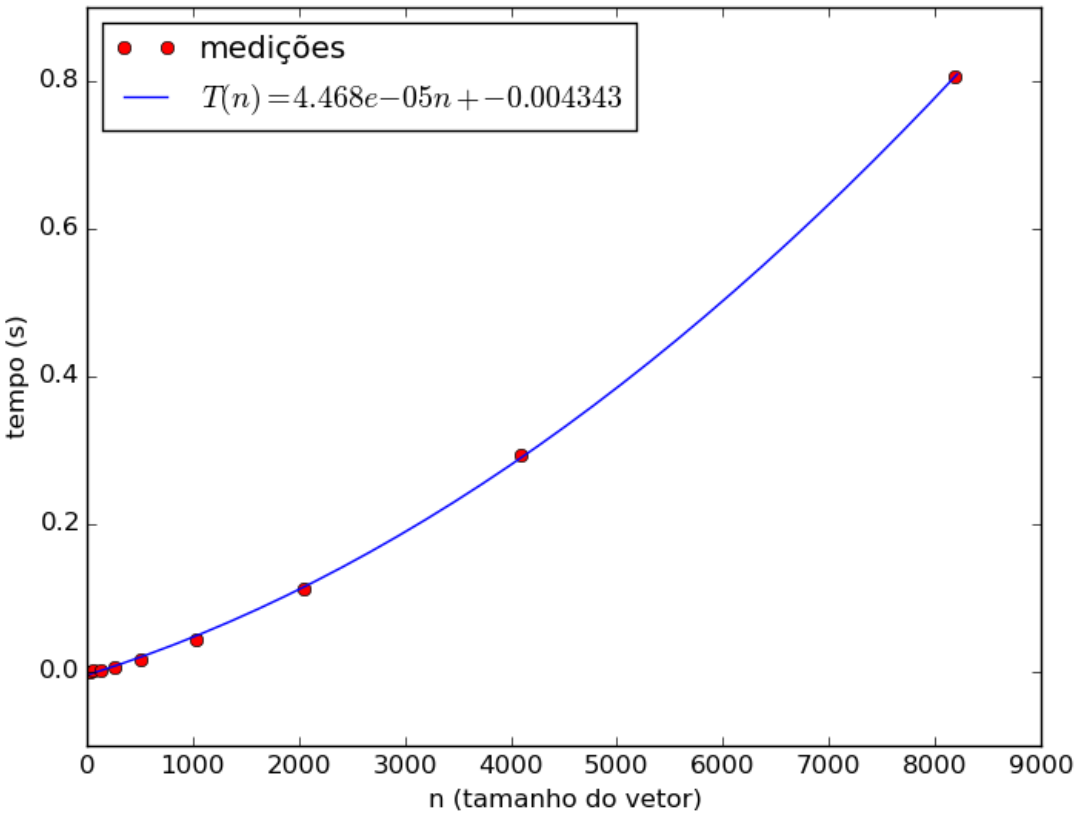

Figura 2.12: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 4.468\text{e} - 5 * n - 0.004343$ e para o $n = 2^{32}$,
$T(2^{32}) = 191899.13444228$

n	comparações	tempo(s)
32	1	0.000550
64	1	0.001277
128	1	0.002749
256	1	0.006829
512	1	0.016530
1024	1	0.041242
2048	1	0.112440
4096	1	0.280583
8192	1	0.776396

Tabela 2.13: *Tabela com vetor teste quase decrescente 50%: A linha te interesse analisada para este caso é a 13*

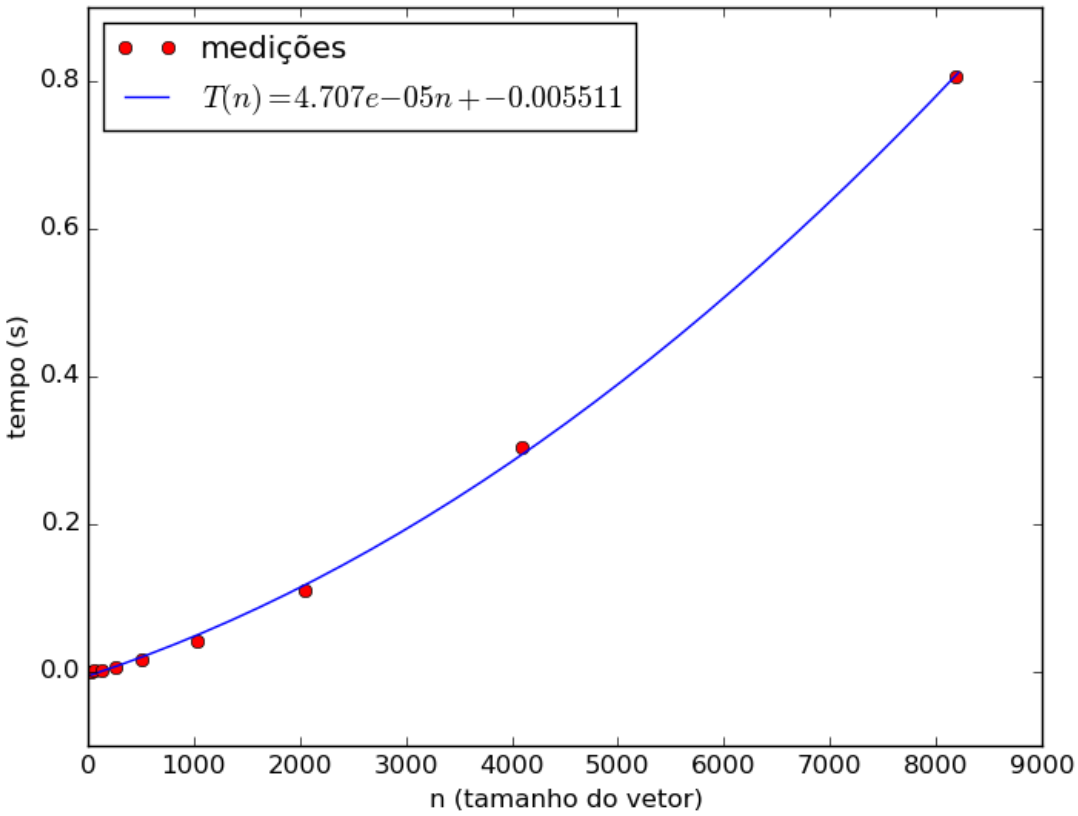

Figura 2.13: *A análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 4.707\text{e} - 5 * n - 0.005511$ e para o $n = 2^{32}$,
$T(2^{32}) = 202164.10511172$

2.1

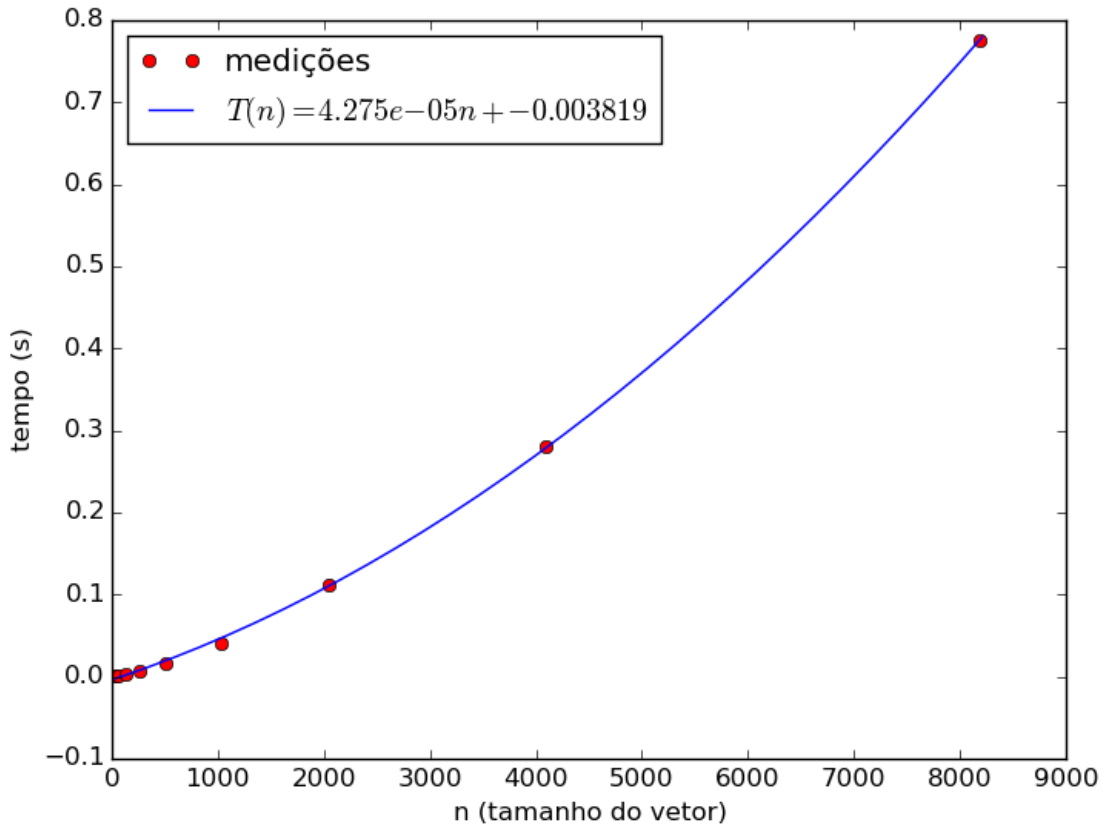

Figura 2.14: *EA análise do grafico para* 2^{32} *segue abaixo para bucketsort*
Tendo a função $T(n) = 4.275\text{e} - 5 * n - 0.003819$ e para o $n = 2^{32}$, $T(2^{32}) = 183609.848085$

Apêndice A

Arquivo ../bucketsort/bucketsort.py

Listagem A.1: ../bucketsort/bucketsort.py

```
1  import numpy as np
2
3  from insertionsort import insertionsort
4
5  @profile
6  def bucketsort( A ):
7      # get hash codes
8      code = hashing( A )
9      buckets = [list() for _ in range( code[1] )]
10
11     # distribute data into buckets: O(n)
12     for i in A:
13         x = re_hashing( i, code )
14         buck = buckets[x]
15         buck.append( i )
16
17         # Sort each bucket: O(n).
18         # I mentioned above that the worst case for bucket sort is
               counting
19         # sort. That's because in the worst case, bucket sort may end up
20         # with one bucket per key. In such case, sorting each bucket would
21         # take 1^2 = O(1). Even after allowing for some probabilistic
22         # variance, to sort each bucket would still take 2-1/n, which is
23         # still a constant. Hence, sorting all the buckets takes O(n).
24     for bucket in buckets:
25         insertionsort( bucket )
26
27     ndx = 0
28     # merge the buckets: O(n)
29     for b in range( len( buckets ) ):
30         for v in buckets[b]:
31             A[ndx] = v
32             ndx += 1
33
34 import math
35
36 def hashing( A ):
37     m = A[0]
38     for i in range( 1, len( A ) ):
39         if ( m < A[i] ):
```

```
            m = A[i]
    result = [m, int( math.sqrt( len( A ) ) )]
    return result

def re_hashing( i, code ):
    return int( i / code[0] * ( code[1] - 1 ) )
```

Apêndice B

Arquivo ../bucketsort/ensaio.py

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

```
1 import numpy as np
2 import argparse
3
4 from bucketsort 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 bucketsort(vet)
```