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 Counting Sort é considerado um algoritmo linear não baseado em comparações. Considera-se seu uso em casos onde os elementos são números inteiros "pequenos", mais precisamente, inteiros que pertencem a O(n), ou seja, a restrição para o uso desse algoritmo é que os elementos do vetor devem ser inteiros contidos em um determinado intervalo até k. O algoritmo funciona da seguinte maneira: cria-se um vetor auxiliar de tamanho k que, primeiramente, tem a função de um histograma. O índice representa o elemento, e seu conteúdo representa a quantidade de vezes que aparece no vetor. Após isso, transforma-se esse histograma em um vetor de frequência acumulada. Dessa forma, a tarefa de ordenar os elementos é facilitada pois sabe-se a posição, representada pelo conteúdo, na qual deve-se inserir o elemento, representado pelo índice. Como é um algoritmo que não faz uso de comparações, então a sua complexidade deve ser medida em função do número de operações. Para criar o histograma gasta-se tempo linear, para transformar esse histograma em um vetor de frequência acumulada, gasta-se um tempo teta(k) e o resto gasto para ordenação é também um tempo linear. Dessa forma, temos que a complexidade do Counting Sort é teta(n + k), e se k pertence a O(n), então sua complexidade é teta(n). Em termos de memória, o Counting Sort necessita criar um vetor auxiliar de tamanho k, ou seja, sua complexidade de espaço é teta(k).

Capítulo 2

Resultados

2.1 Tabelas

n	comparações	tempo(s)
32	33	0.001023
64	65	0.001057
128	129	0.001284
256	257	0.001908
512	513	0.002677
1024	1025	0.004362
2048	2049	0.007193
4096	4097	0.014430
8192	8193	0.026760

Tabela 2.1: *CountingSort com Vetores Aleatorio*

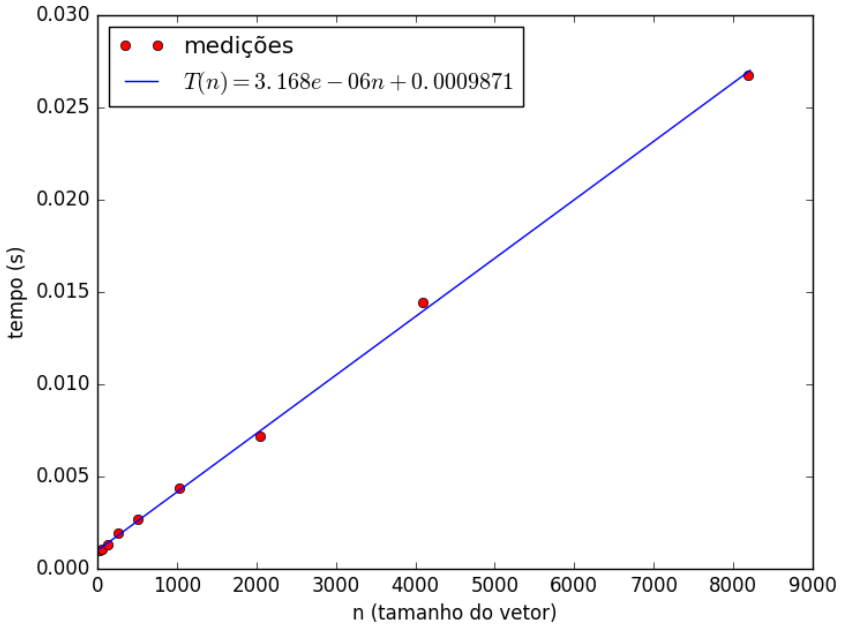

Figura 2.1: *Countingsort com relação ao tempo com vetor aleatório.*

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

$3.168 * 10^{-6} * n - 0.0009871 = 13606.45639$

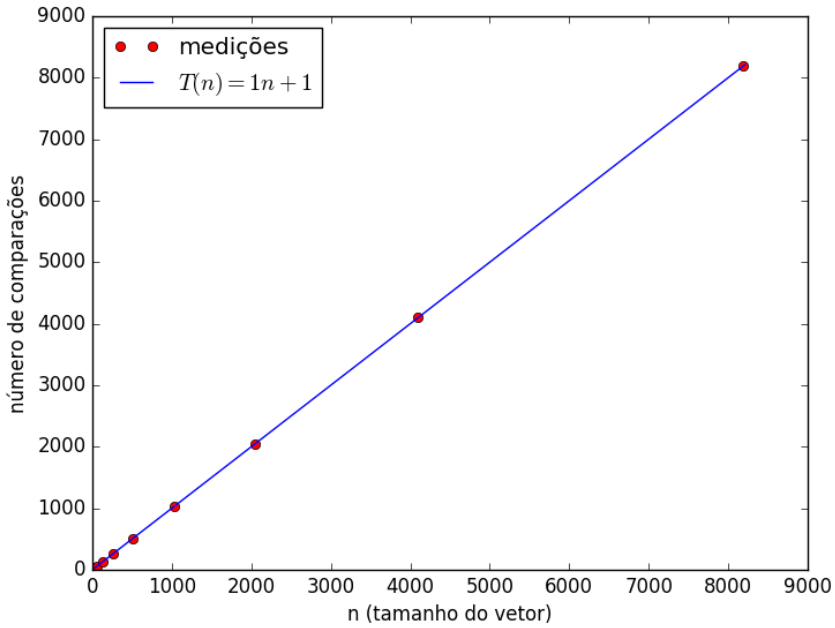

Figura 2.2: *Countingsort com relação ao número de comparações com vetor aleatório.*

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.001177
64	65	0.001273
128	129	0.001312
256	257	0.001704
512	513	0.002506
1024	1025	0.004144
2048	2049	0.007762
4096	4097	0.014016
8192	8193	0.025296

Tabela 2.2: *CountingSort com Vetores Crescente*

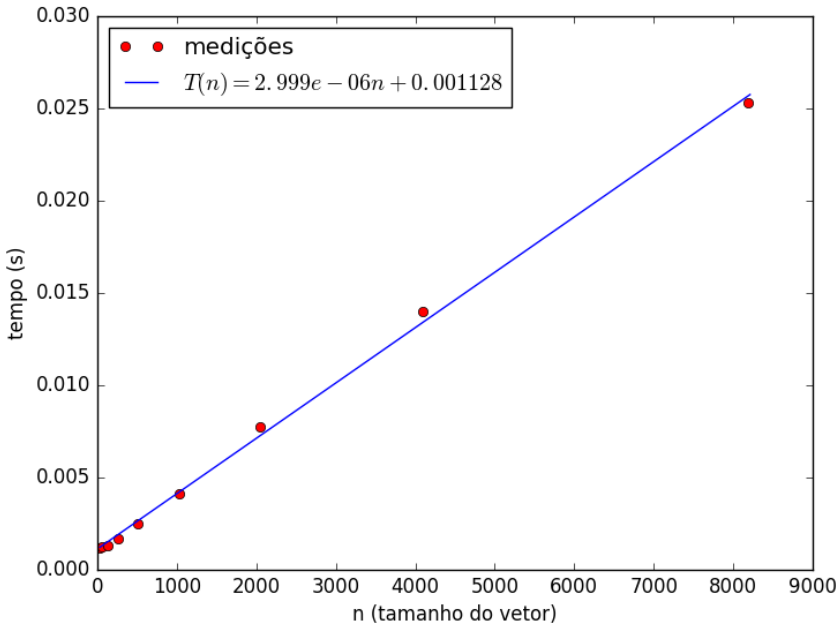

Figura 2.3: *Countingsort com relação ao tempo com vetor crescente.*

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

$2.999 * 10^{-6} * n + 0.001128 = 12880.60692$

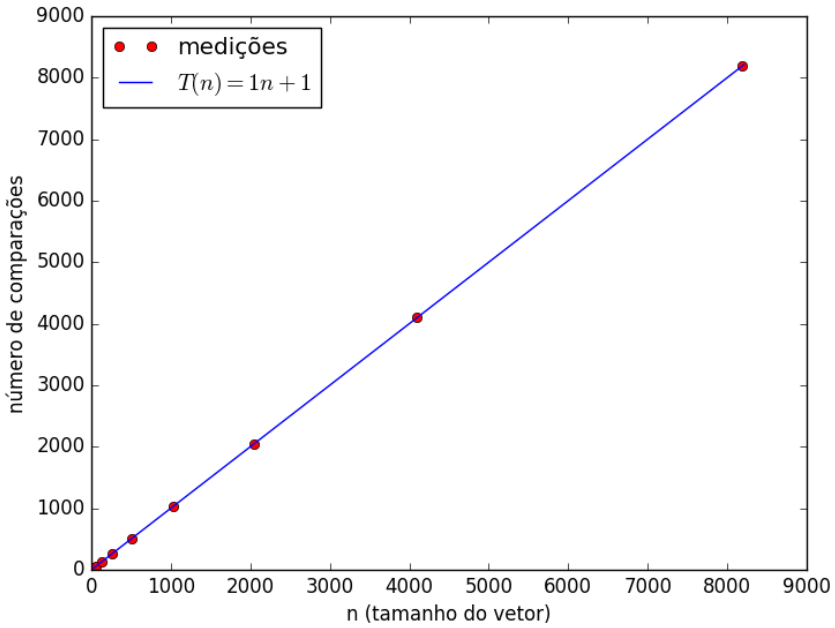

Figura 2.4: *Countingsort com relação ao número de comparações com vetor crescente.*

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.001049
64	65	0.001105
128	129	0.001415
256	257	0.001905
512	513	0.002688
1024	1025	0.004943
2048	2049	0.008366
4096	4097	0.014679
8192	8193	0.028053

Tabela 2.3: *CountingSort com Vetores Decrescente*

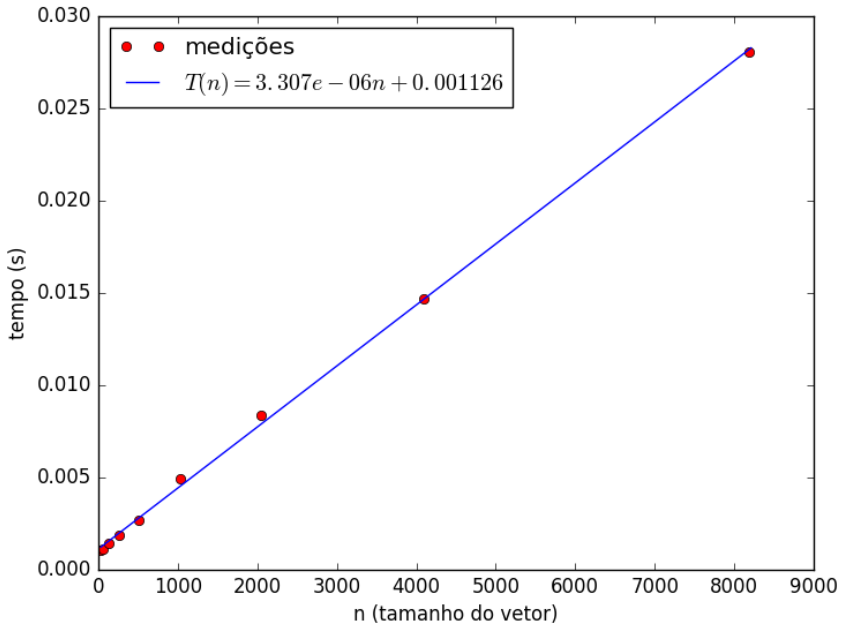

Figura 2.5: *Countingsort com relação ao tempo com vetor decrescente.*

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

$3.307 * 10^{-6} * n + 0.001126 = 14203.45685$

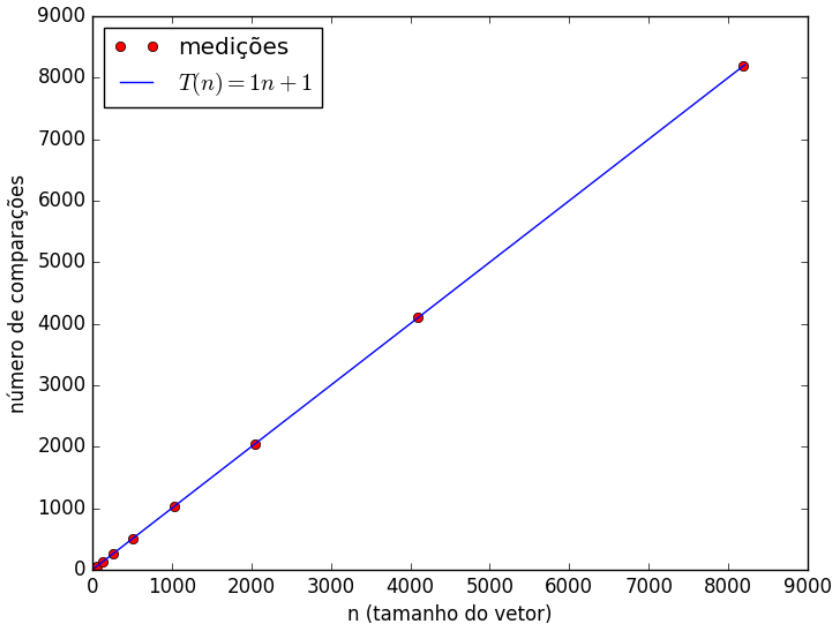

Figura 2.6: *Countingsort com relação ao número de comparações com vetor decrescente.*

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.000993
64	65	0.001112
128	129	0.001318
256	257	0.001767
512	513	0.002763
1024	1025	0.004712
2048	2049	0.008382
4096	4097	0.014972
8192	8193	0.028116

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

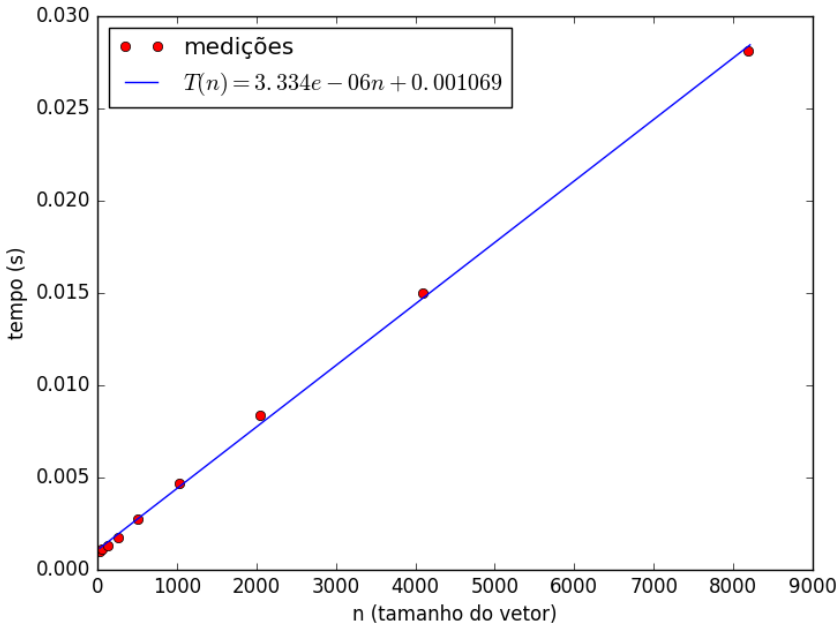

Figura 2.7: *countingsort com relação ao tempo com vetor quase crescente* 10%.

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

$3.334 * 10^{-6} * n + 0.001069 = 14345.19077$

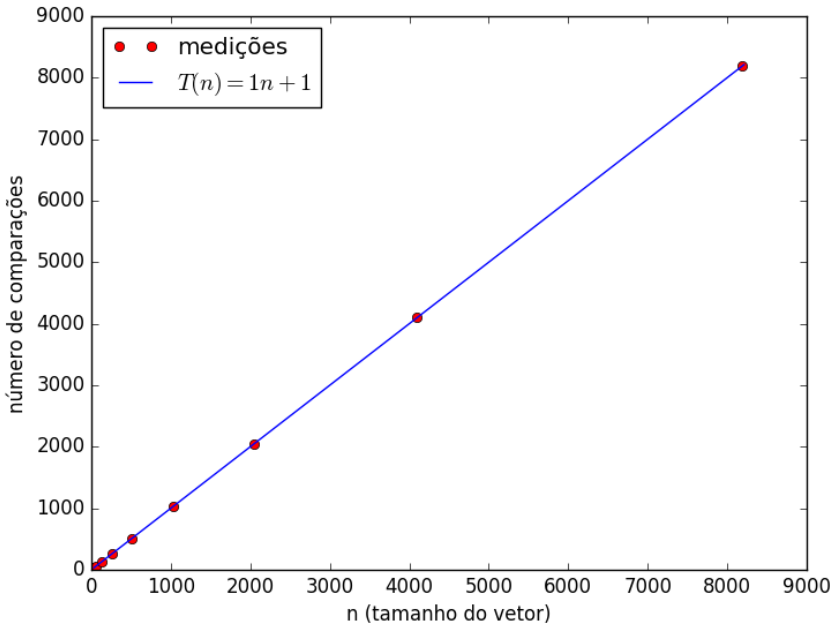

Figura 2.8: *Countingsort com relação ao número de comparações com vetor quase crescente* 10%.

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.001039
64	65	0.001215
128	129	0.001543
256	257	0.001536
512	513	0.002887
1024	1025	0.004920
2048	2049	0.008122
4096	4097	0.014585
8192	8193	0.027807

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

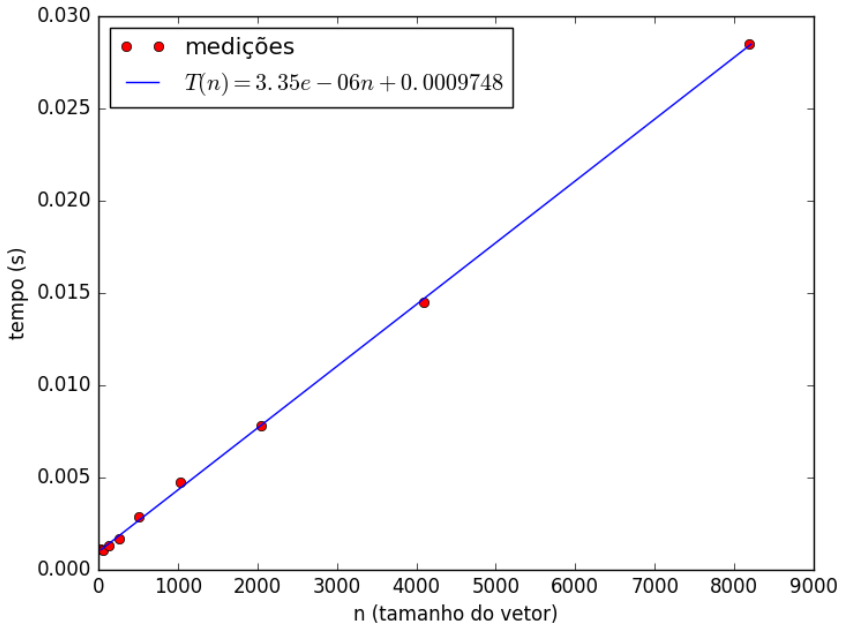

Figura 2.9: *Countingsort com relação ao tempo com vetor quase crescente* 20%.

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

$3.35 * 10^{-6} * n + 0.0009748 = 14388.14044$

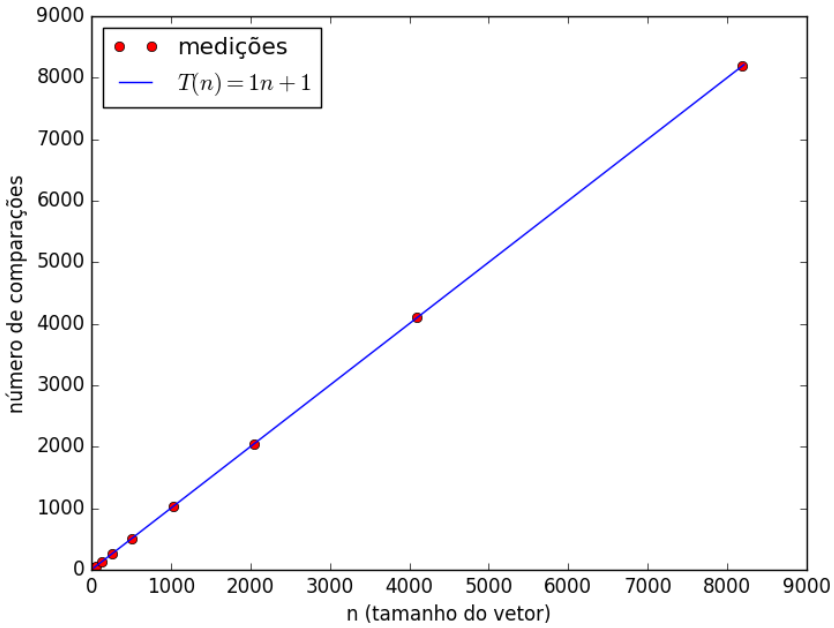

Figura 2.10: *Countingsort com relação ao número de comparações com vetor quase crescente* 20%.

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.001334
64	65	0.001228
128	129	0.001296
256	257	0.002040
512	513	0.002524
1024	1025	0.004625
2048	2049	0.007899
4096	4097	0.015035
8192	8193	0.030343

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

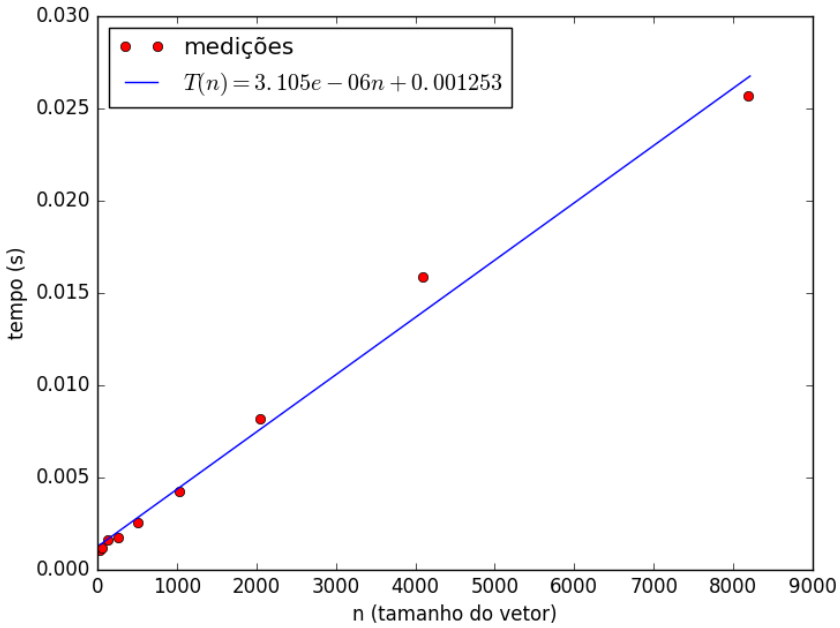

Figura 2.11: *Countingsort com relação ao tempo com vetor quase crescente* 30%.

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

$3.105 * 10^{-6} * n + 0.001253 = 13335.87345$

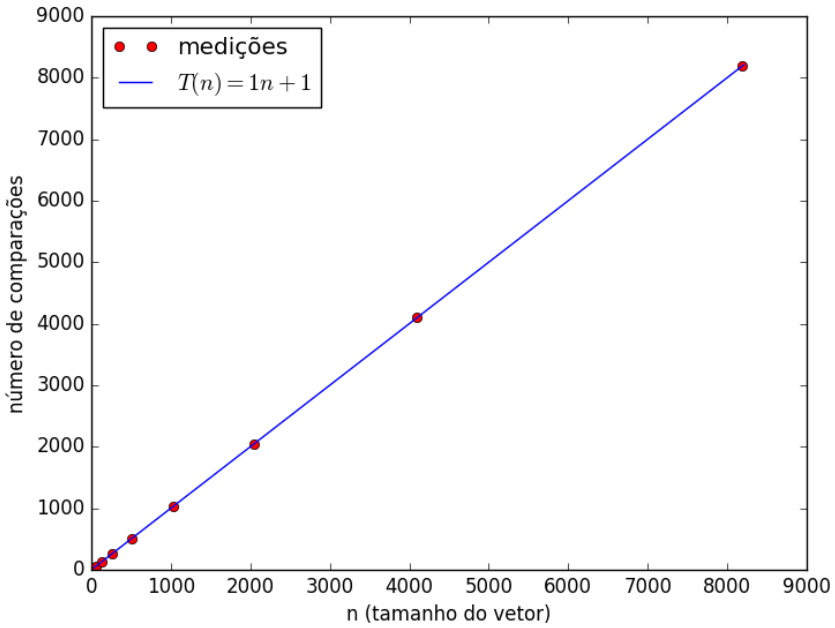

Figura 2.12: *Countingsort com relação ao número de comparações com vetor quase crescente* 30%.

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.001016
64	65	0.001223
128	129	0.001291
256	257	0.001715
512	513	0.002551
1024	1025	0.004466
2048	2049	0.007444
4096	4097	0.014642
8192	8193	0.030544

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

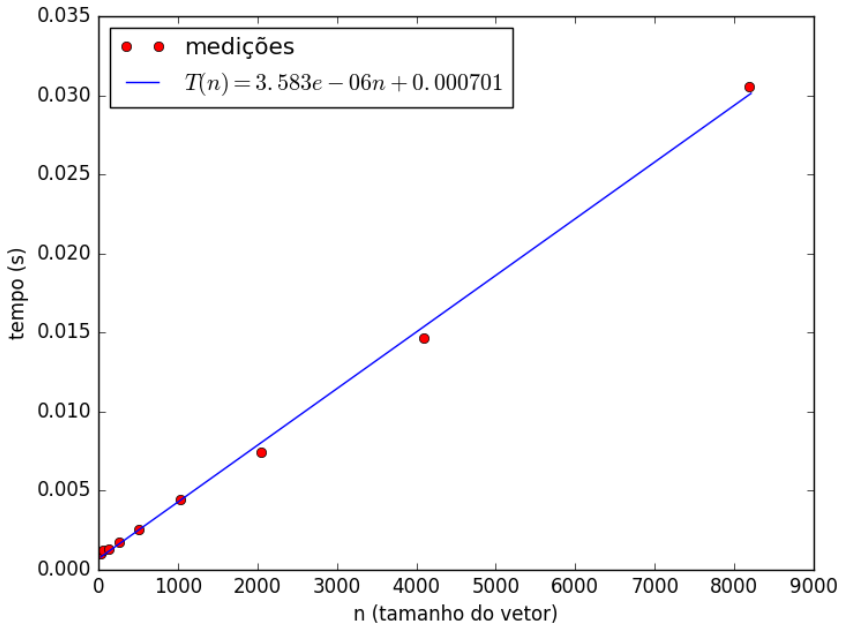

Figura 2.13: *Countingsort com relação ao tempo com vetor quase crescente* 40%.

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

$3.583 * 10^{-6} * n + 0.000701 = 15388.86782$

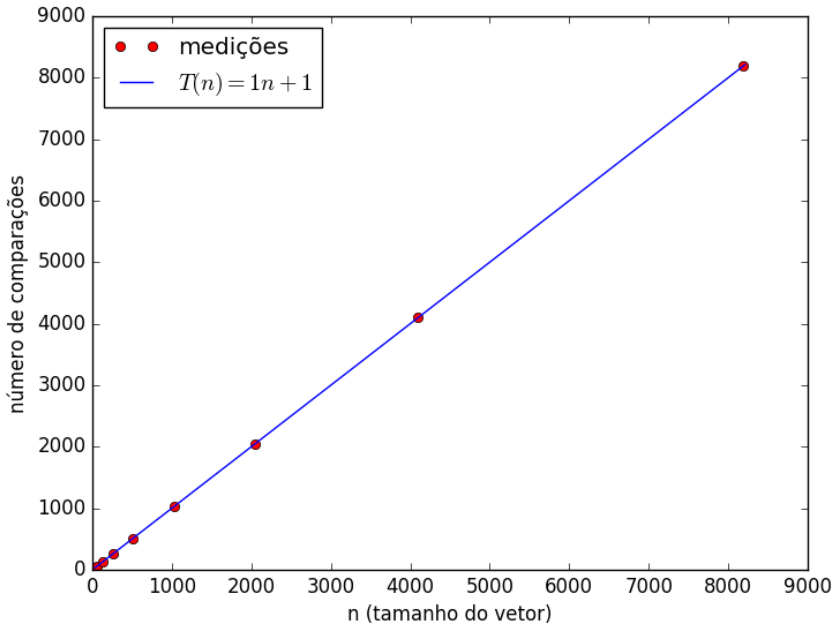

Figura 2.14: *Countingsort com relação ao número de comparações com vetor quase crescente* 40%.

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.001108
64	65	0.001117
128	129	0.001379
256	257	0.001879
512	513	0.002411
1024	1025	0.004170
2048	2049	0.008500
4096	4097	0.016349
8192	8193	0.029093

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

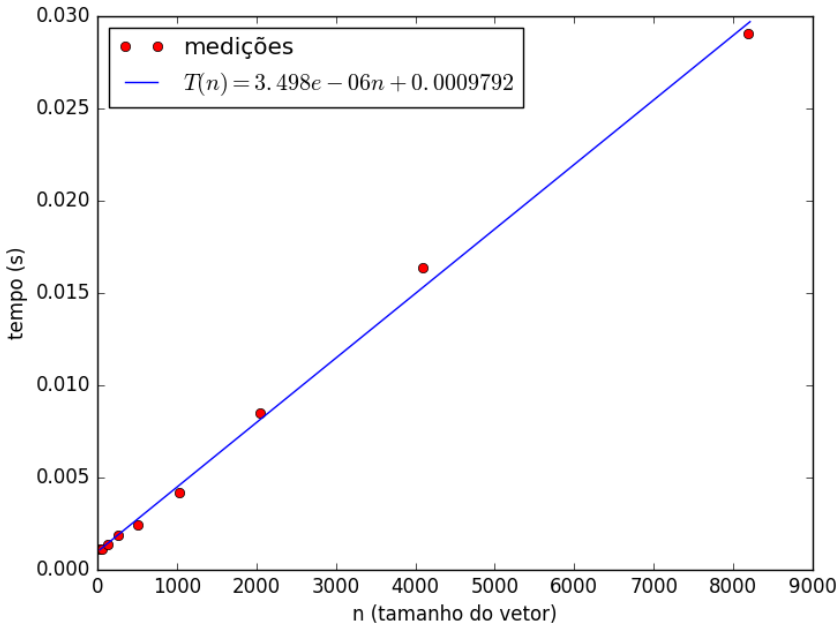

Figura 2.15: *Countingsort com relação ao tempo com vetor quase crescente* 50%.

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

$3.498 * 10^{-6} * n + 0.0009792 = 15023.7956$

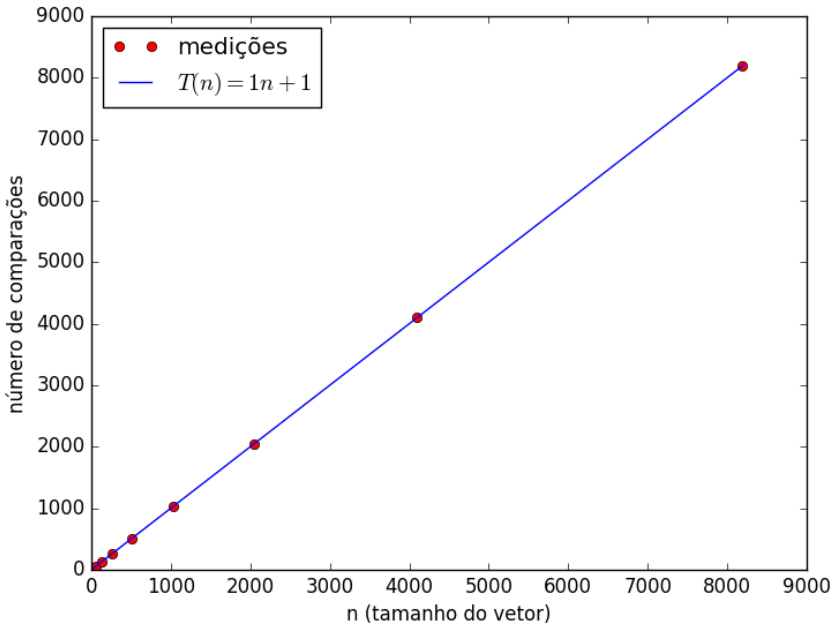

Figura 2.16: *Countingsort com relação ao número de comparações com vetor quase crescente* 50%.

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.001106
64	65	0.001007
128	129	0.001341
256	257	0.001880
512	513	0.002791
1024	1025	0.004188
2048	2049	0.008545
4096	4097	0.015407
8192	8193	0.027908

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

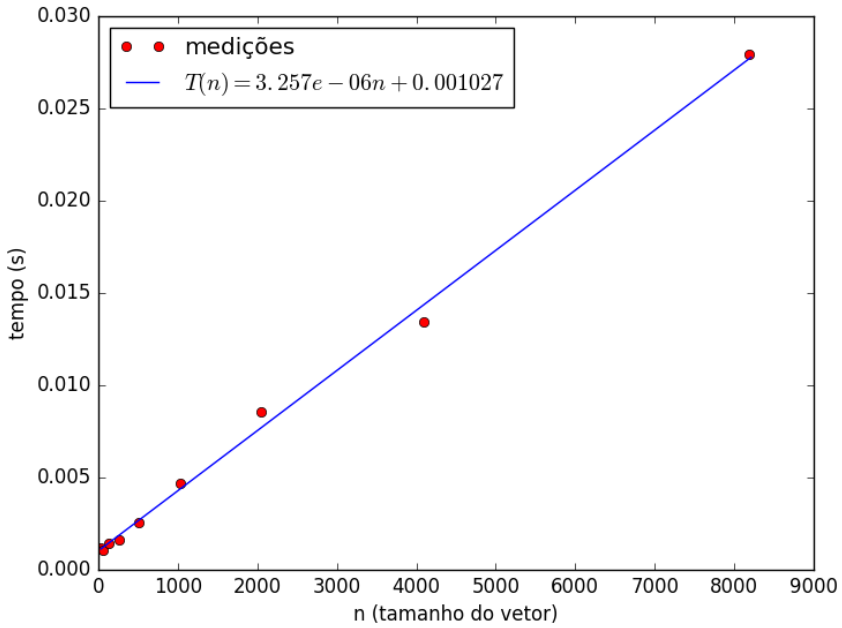

Figura 2.17: *Countingsort com relação ao tempo com vetor quase decrescente* 10%.

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

$3.257 * 10^{-6} * n + 0.001027 = 13988.70848$

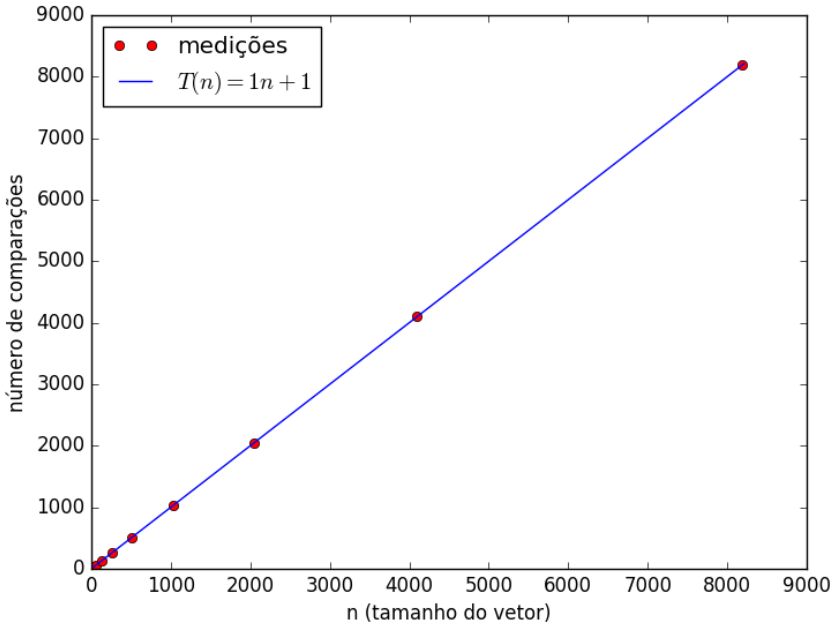

Figura 2.18: *Countingsort com relação ao número de comparações com vetor quase decrescente* 10%.

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.001010
64	65	0.001298
128	129	0.001329
256	257	0.001975
512	513	0.002940
1024	1025	0.004370
2048	2049	0.007550
4096	4097	0.015701
8192	8193	0.027275

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

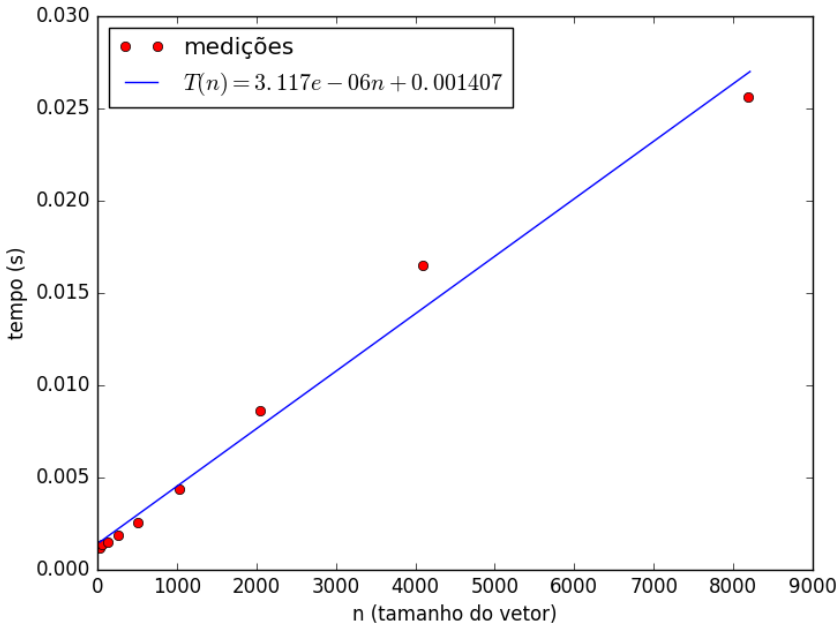

Figura 2.19: *Countingsort com relação ao tempo com vetor quasede decrescente* 20%.

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

$3.117 * 10^{-6} * n + 0.001407 = 13874.41306$

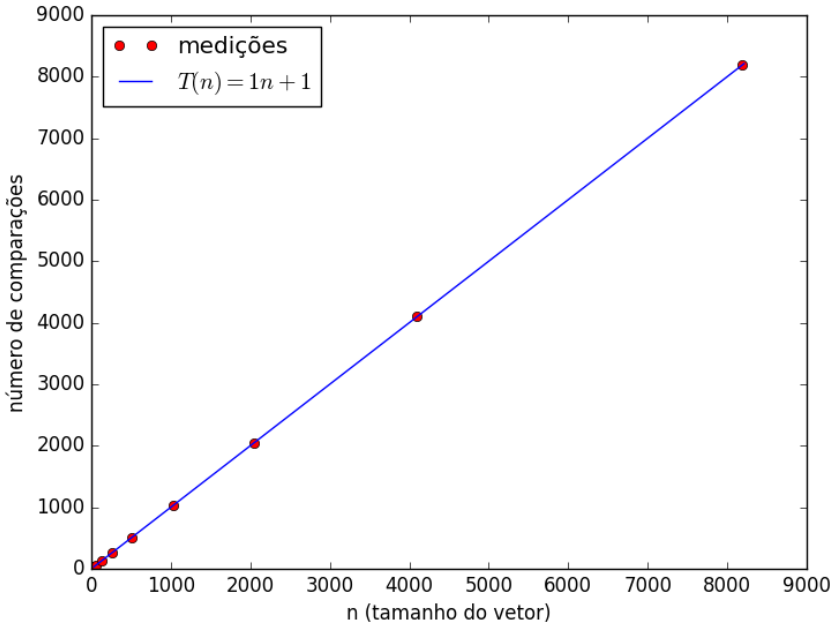

Figura 2.20: *Countingsort com relação ao número de comparações com vetor quase decrescente* 20%.

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.001137
64	65	0.001065
128	129	0.001428
256	257	0.001763
512	513	0.002535
1024	1025	0.003972
2048	2049	0.007760
4096	4097	0.015620
8192	8193	0.027310

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

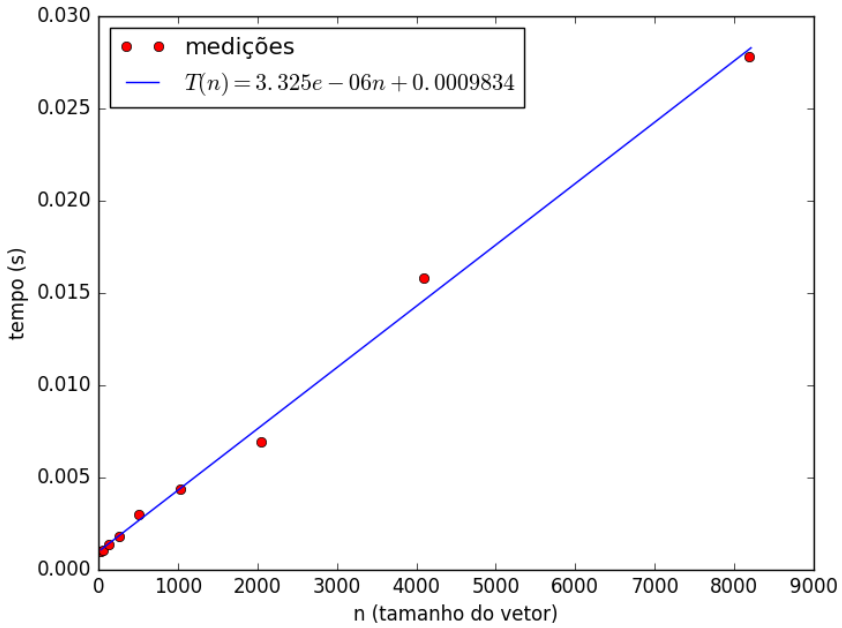

Figura 2.21: *Countingsort com relação ao tempo com vetor quase decrescente* 30%.

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

$3.25 * 10^{-6} * n + 0.0009834 = 13958.64371$

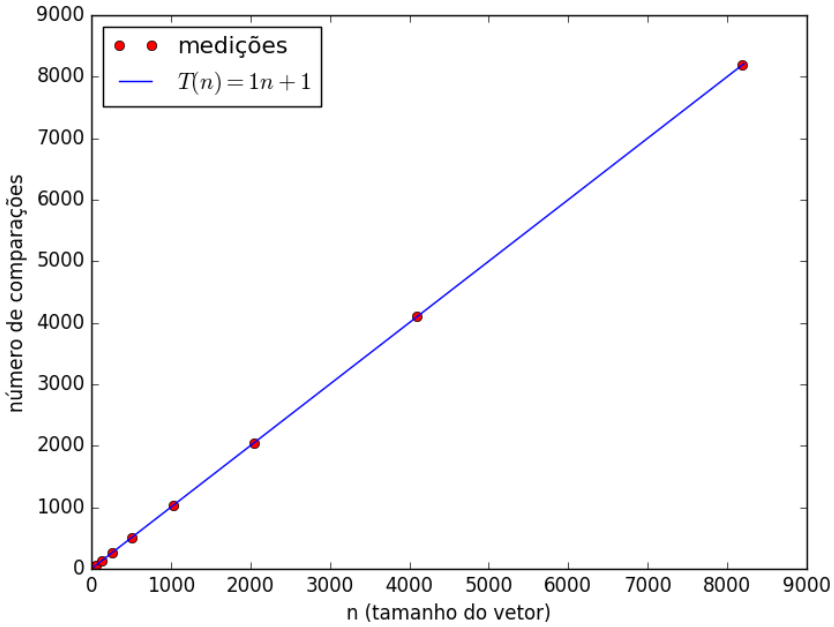

Figura 2.22: *Countingsort com relação ao número de comparações com vetor quase decrescente* 30%.

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.001035
64	65	0.001387
128	129	0.001393
256	257	0.001726
512	513	0.002881
1024	1025	0.004266
2048	2049	0.007691
4096	4097	0.014473
8192	8193	0.029091

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

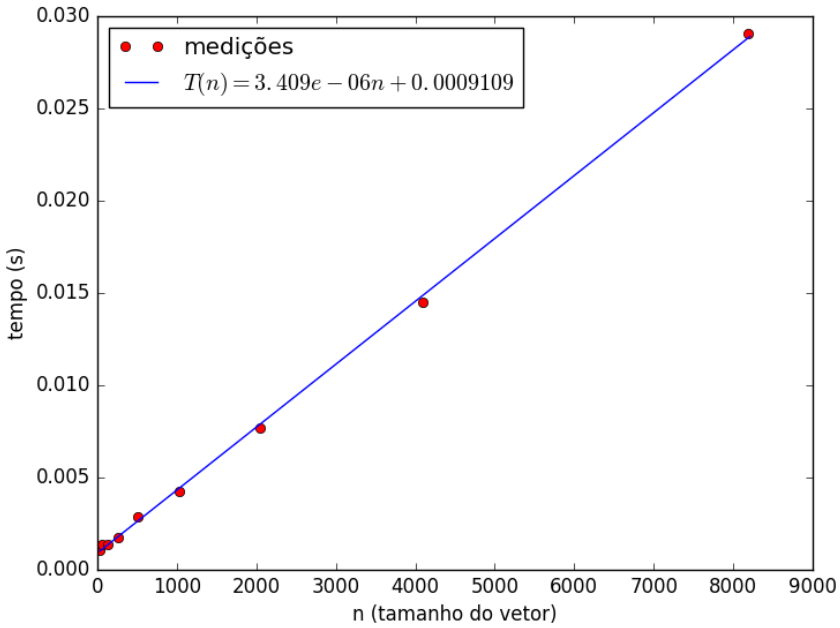

Figura 2.23: *Countingsort com relação ao tempo com vetor quase decrescente* 40%.

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

$3.409 * 10^{-6} * n + 0.0009109 = 14641.54351$

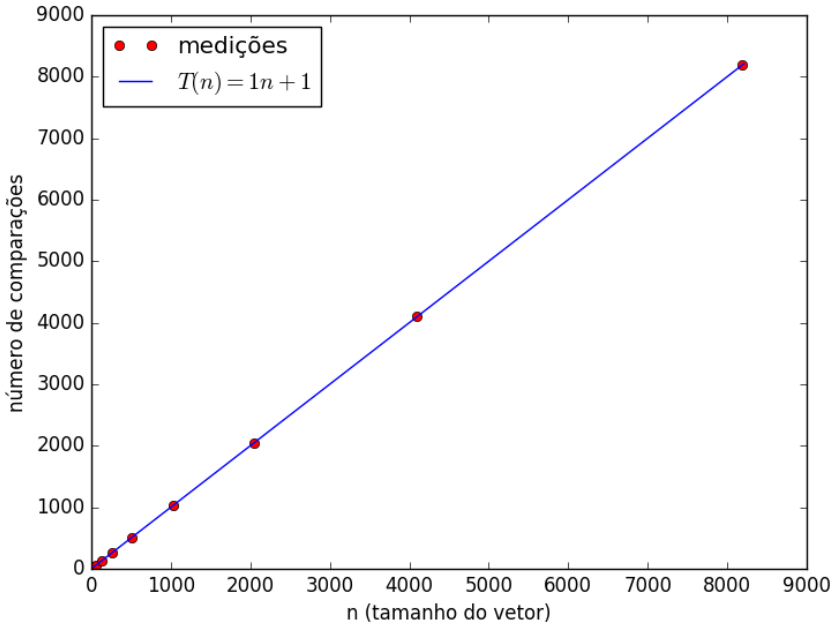

Figura 2.24: *Countingsort com relação ao número de comparações com vetor quase decrescente* 40%.

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

$n + 1 = 4294967297$

n	comparações	tempo(s)
32	33	0.001078
64	65	0.001191
128	129	0.001364
256	257	0.001821
512	513	0.002499
1024	1025	0.003832
2048	2049	0.007402
4096	4097	0.015834
8192	8193	0.027802

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

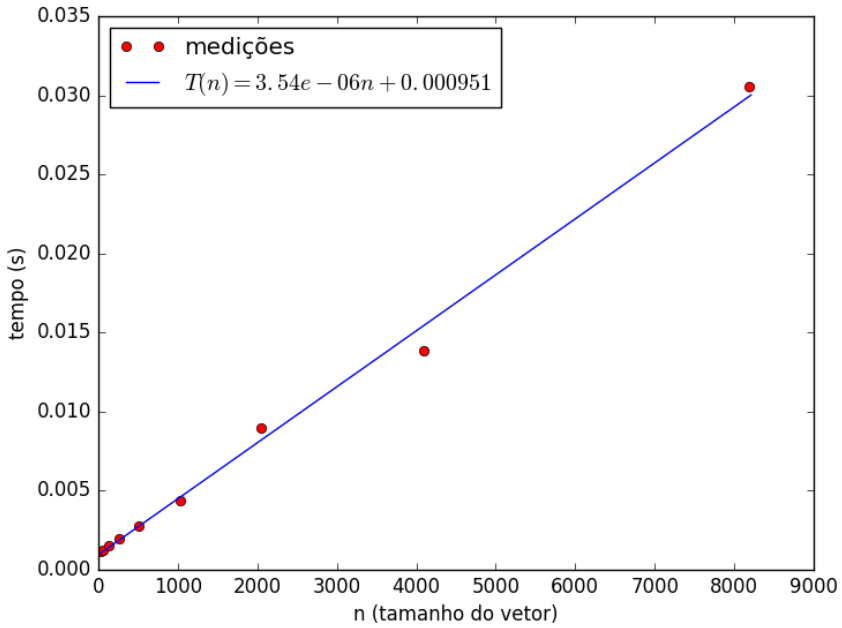

Figura 2.25: *Countingsort com relação ao tempo com vetor quase decrescente* 50%.

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

$3.54 * 10^{-6} * n + 0.000951 = 15204.18423$

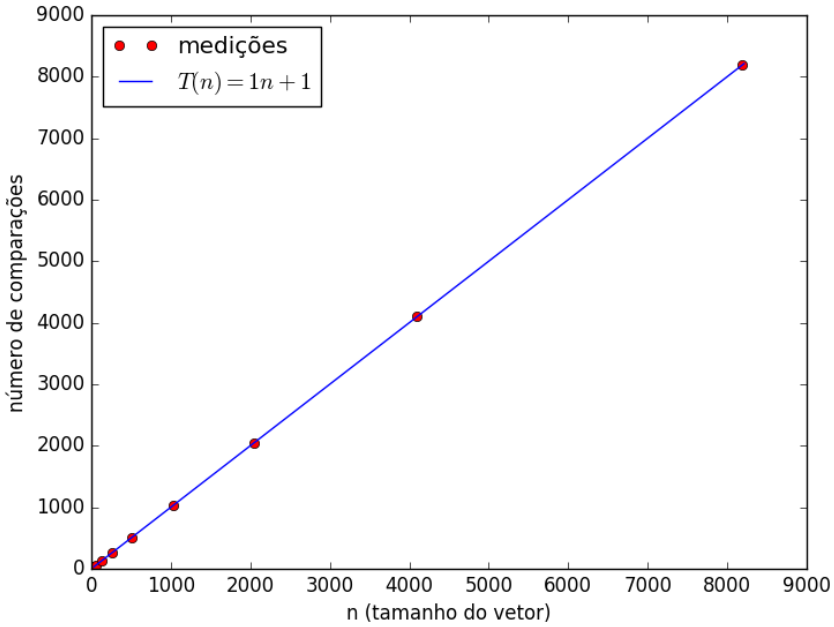

Figura 2.26: *Countingsort com relação ao número de comparações com vetor quase decrescente* 50%.

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

$n + 1 = 4294967297$

Apêndice A

Arquivo ../countingsort/countingsort.py

Listagem A.1: ../countingsort/countingsort.py

```
@profile
def countingsort(aList, k):
    counter = [0] * ( k + 1 )
    for i in aList:
        counter[int(i)] += 1
    ndx = 0;
    for i in range( len( counter ) ):
        while 0 < counter[i]:
            aList[ndx] = i
            ndx += 1
            counter[i] -= 1
```

Apêndice B

Arquivo ../countingsort/ensaio.py

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

```
1  import numpy as np
2  import argparse
3
4  from countingsort 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 countingsort(vet,1000)
```