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 Radix Sort é considerado um algoritmo linear não baseado em comparações. Considera-se seu uso em casos onde os elementos são números inteiros de comprimento máximo constante, isto é, independentes do tamanho do vetor. Além disso, é possível utilizá-lo para ordenar certos tipos de elementos como datas, CEPs e palavras em ordem lexicográficas. Esse algoritmo resolve o problema de ordenar um vetor no qual os elementos podem ser representados com apenas d dígitos, sendo d uma constante. Ele utiliza um método de ordenação estável como subrotina de forma a ordenar o vetor do dígito menos significativo para o mais significativo, com o objetivo de poupar memória adicional.

A sua complexidade depende da complexidade do algoritmo estável utilizado. Temos os seguintes algoritmos estáveis vistos em sala: Insertion Sort, Merge Sort e Counting Sort. Comparando a complexidade dos três, temos que o primeiro é $teta(n^2)$, o segundo é $teta(n * lgn)$ e o terceiro é $teta(n + k)$. Dessa forma, temos que a complexidade total do Radix Sort é $teta(d * f(n))$, na qual $teta(f(n))$ é a complexidade do algoritmo estável. No caso de utilizarmos o Counting Sort, a complexidade total do Radix será $teta(d * (n + k))$. Se k pertencer $teta(n)$ a complexidade total do Radix será $teta(d * n)$, ou seja, complexidade linear em n.

Em termos de memória, temos que esse algoritmo necessita armazenar dois vetores auxiliares no caso de utilzar o Counting Sort como subrotina: um do tamanho do vetor a ser ordenado e outro do tamanho de k.

Capítulo 2

Resultados

2.1 Tabelas

n	comparações	tempo(s)
32	1	0.000309
64	1	0.000556
128	1	0.000987
256	1	0.001976
512	1	0.003803
1024	1	0.007793
2048	1	0.015191
4096	1	0.030633
8192	1	0.060610

Tabela 2.1: *RadixSort com Vetores Aleatorio*

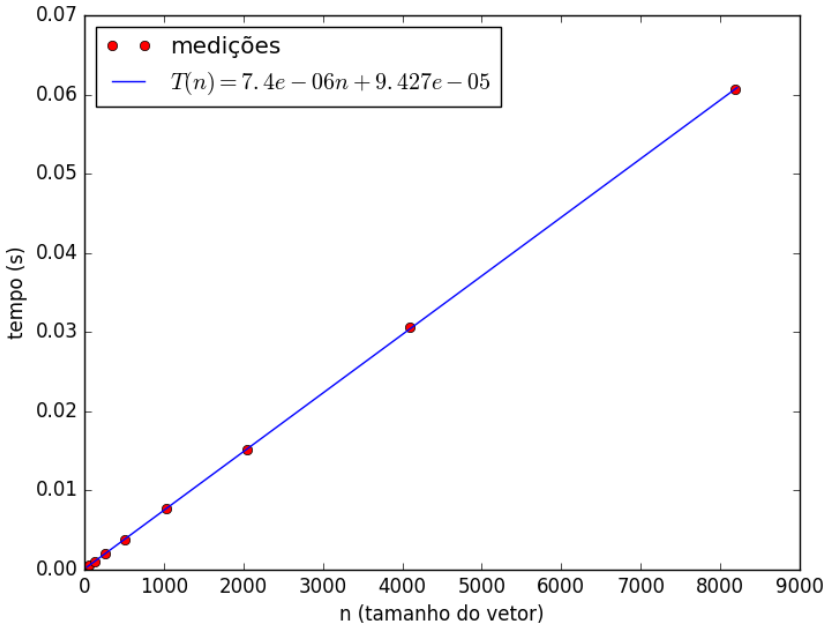

Figura 2.1: *Radix com relação ao tempo com vetor aleatório.*

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

$7.4 * 10^{-6} * n + 9.427 * 10^{-5} = 1.365059061e + 14$

2.1

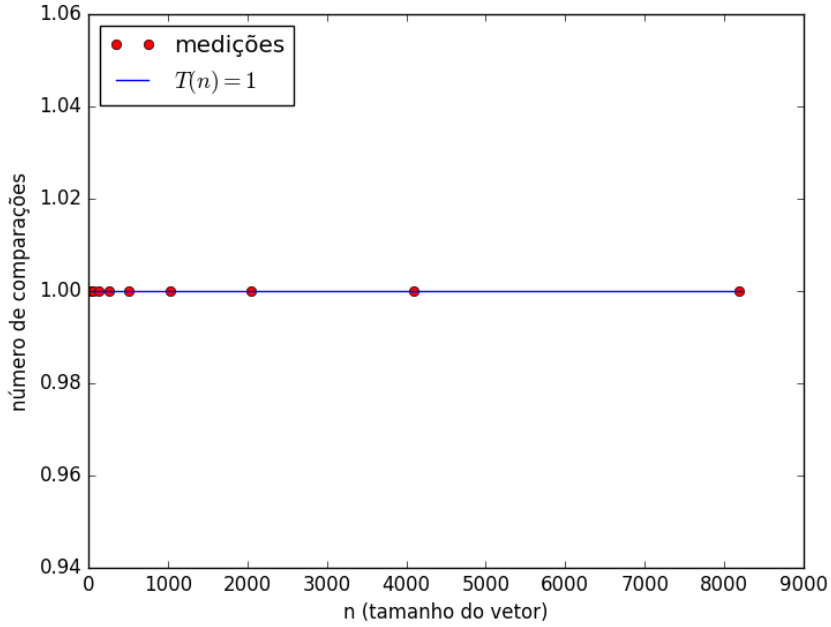

Figura 2.2: *Radix com relação ao número de comparações com vetor aleatório.*

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

1

n	comparações	tempo(s)
32	1	0.000306
64	1	0.000529
128	1	0.000999
256	1	0.001908
512	1	0.003865
1024	1	0.007595
2048	1	0.015088
4096	1	0.030361
8192	1	0.058781

Tabela 2.2: *RadixSort com Vetores Crescentes*

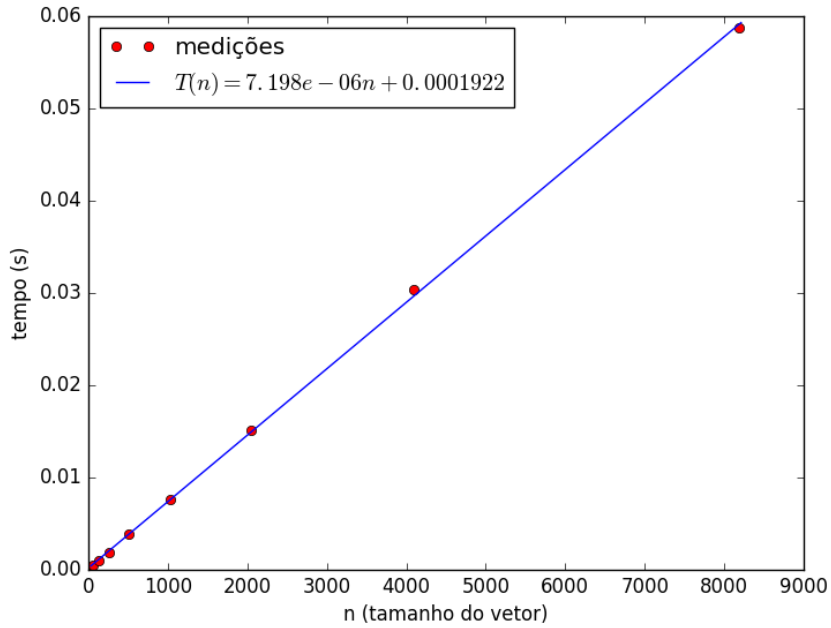

Figura 2.3: *Radix com relação ao tempo com vetor crescente.*

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

$7.198 * 10^{-6} * n + 0.0001922 = 30915.17479$

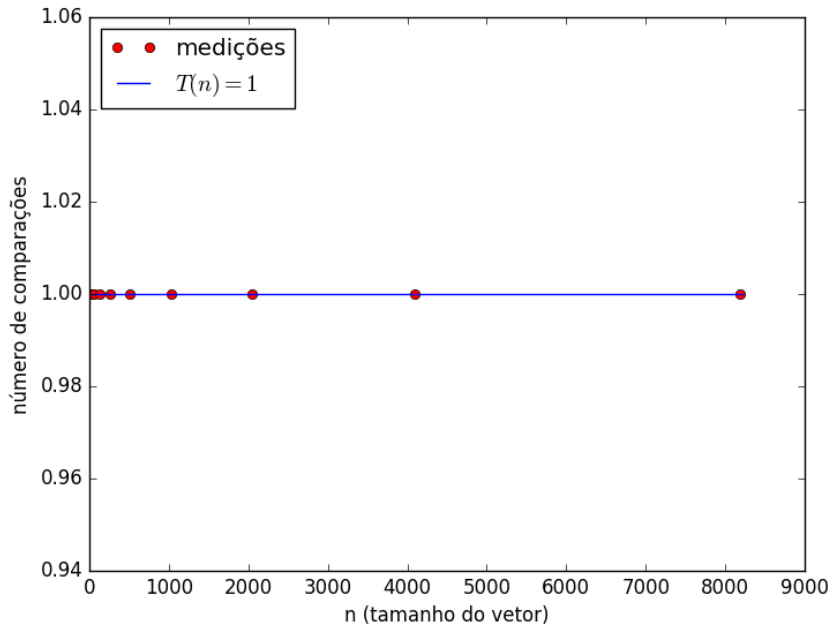

Figura 2.4: *Radix com relação ao número de comparações com vetor crescente.*

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

1

n	comparações	tempo(s)
32	1	0.000295
64	1	0.000525
128	1	0.001001
256	1	0.001945
512	1	0.003881
1024	1	0.007522
2048	1	0.015137
4096	1	0.029712
8192	1	0.058699

Tabela 2.3: *RadixSort com Vetores Decrescentes*

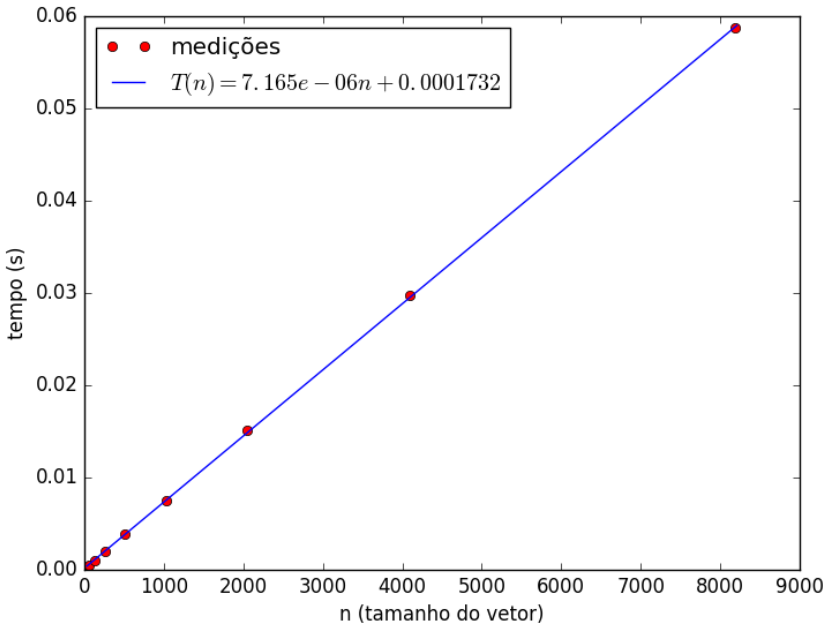

Figura 2.5: *Radix com relação ao tempo com vetor decrescente.*

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

$7.165 * 10^{-6} * n + 0.0001732 = 30773.44085$

2.1

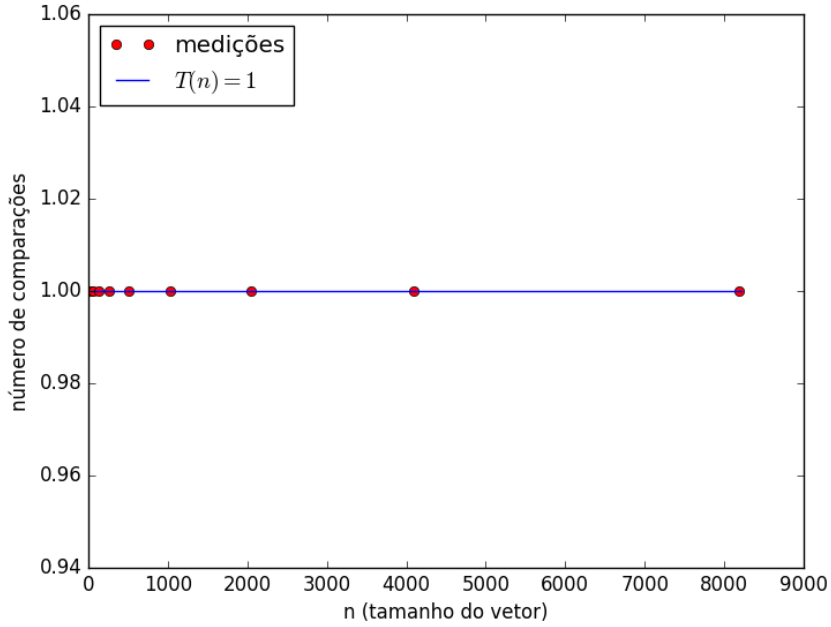

Figura 2.6: *Radix com relação ao número de comparações com vetor decrescente.*

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

1

n	comparações	tempo(s)
32	1	0.000305
64	1	0.000525
128	1	0.000981
256	1	0.001916
512	1	0.003801
1024	1	0.007496
2048	1	0.015655
4096	1	0.029552
8192	1	0.058935

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

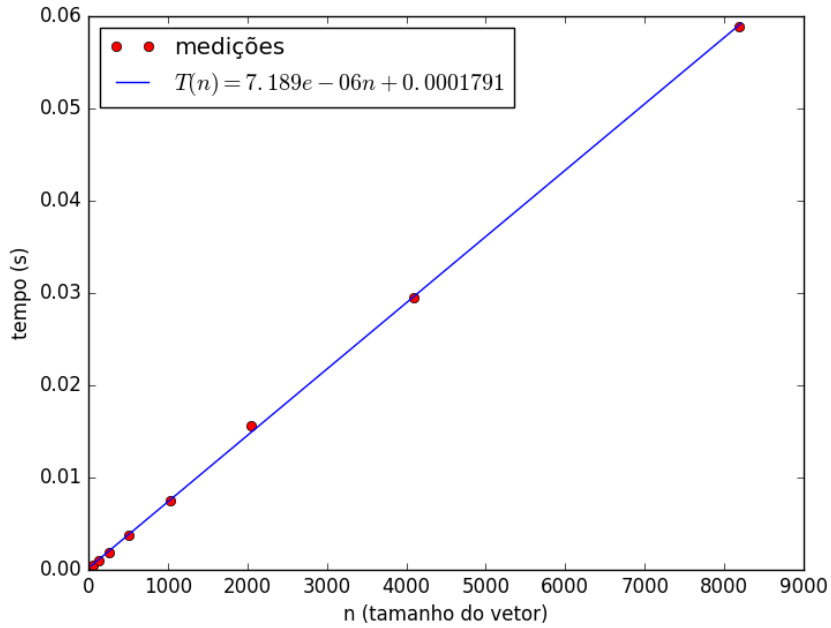

Figura 2.7: *radix com relação ao tempo com vetor quase crescente* 10%.

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

$7.189 * 10^{-6} * n + 0.0001791 = 30876.51989$

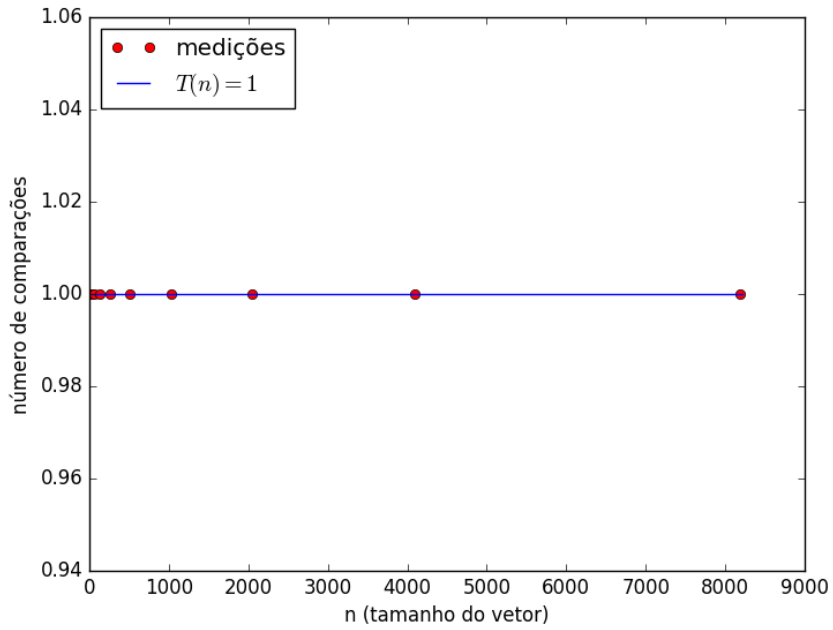

Figura 2.8: *Radix com relação ao número de comparações com vetor quase crescente* 10%.

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

1

n	comparações	tempo(s)
32	1	0.000305
64	1	0.000530
128	1	0.000981
256	1	0.001906
512	1	0.003752
1024	1	0.007721
2048	1	0.015039
4096	1	0.029841
8192	1	0.058711

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

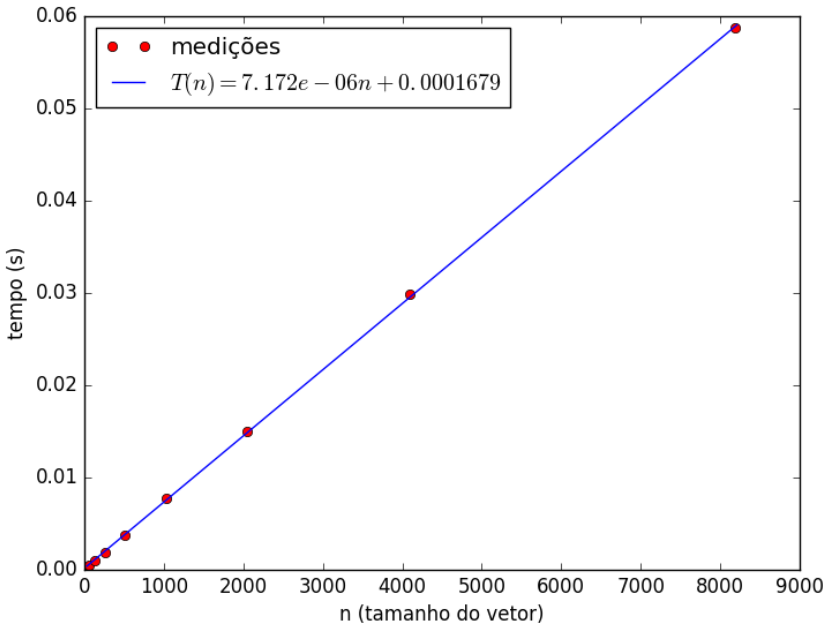

Figura 2.9: *Radix com relação ao tempo com vetor quase crescente* 20%.

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

$7.172 * 10^{-6} * n + 0.0001679 = 30803.50545$

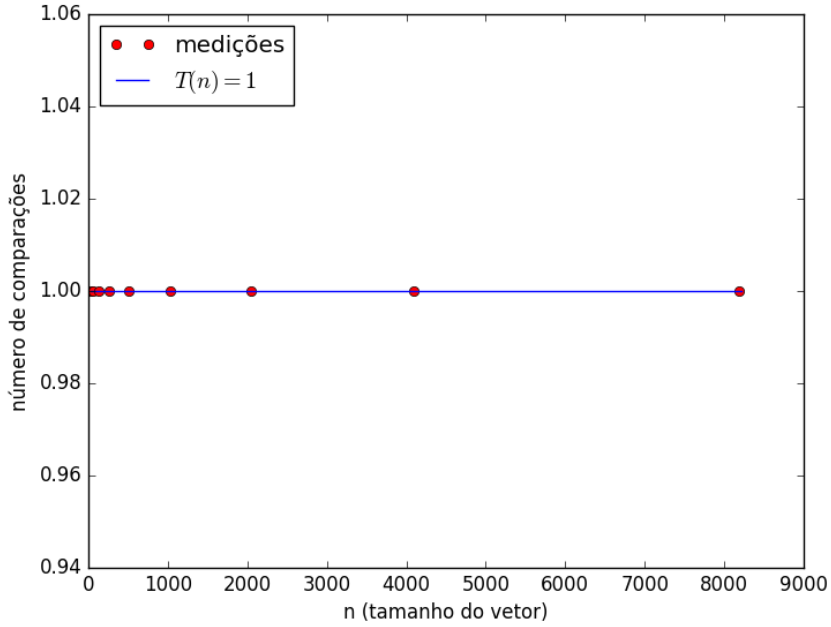

Figura 2.10: *Radix com relação ao número de comparações com vetor quase crescente* 20%.

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

1

n	comparações	tempo(s)
32	1	0.000300
64	1	0.000541
128	1	0.001082

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

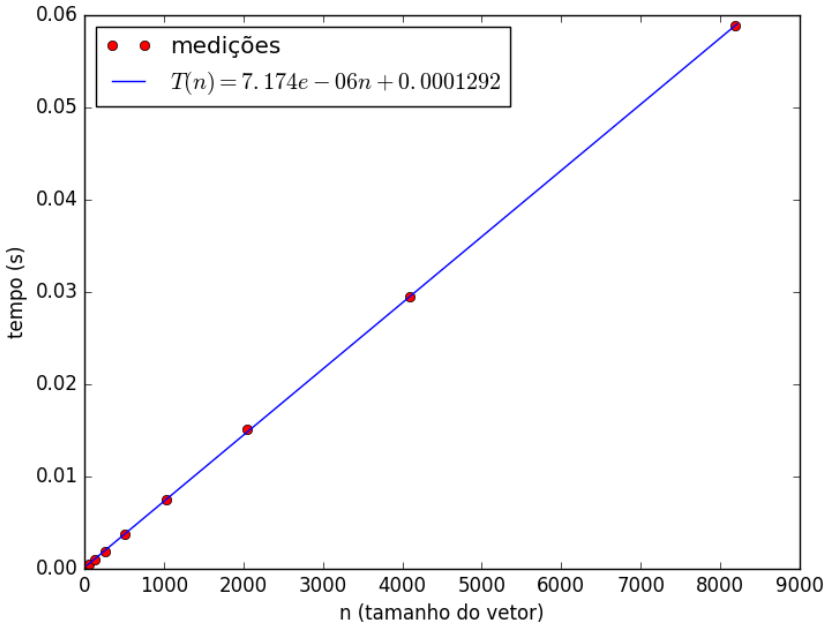

Figura 2.11: *Radix com relação ao tempo com vetor quase crescente* 30%.

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

$7.174 * 10^{-6} * n + 0.0001292 = 30812.09538$

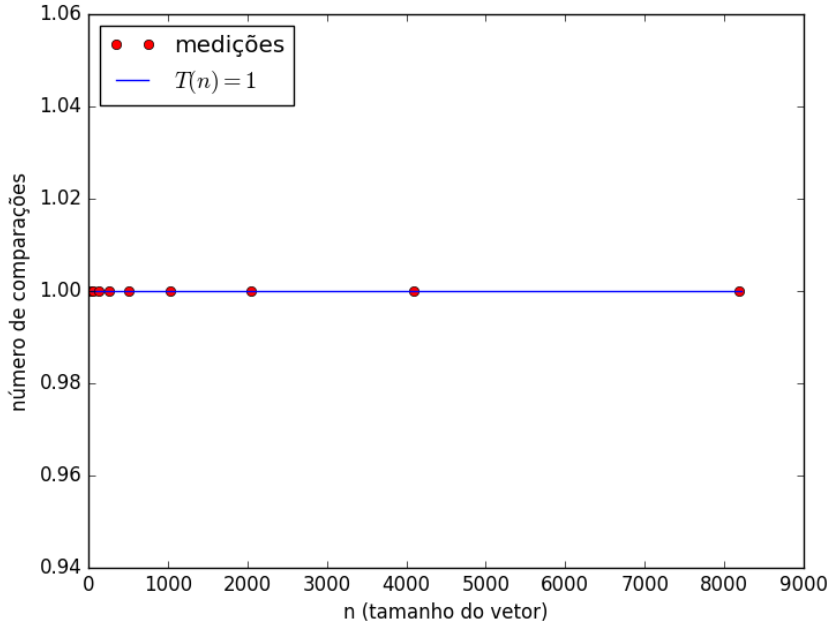

Figura 2.12: *Radix com relação ao número de comparações com vetor quase crescente* 30%.

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

1

n	comparações	tempo(s)
32	1	0.000296
64	1	0.000537
128	1	0.001016
256	1	0.001940
512	1	0.003804
1024	1	0.007566
2048	1	0.014913
4096	1	0.029861
8192	1	0.061097

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

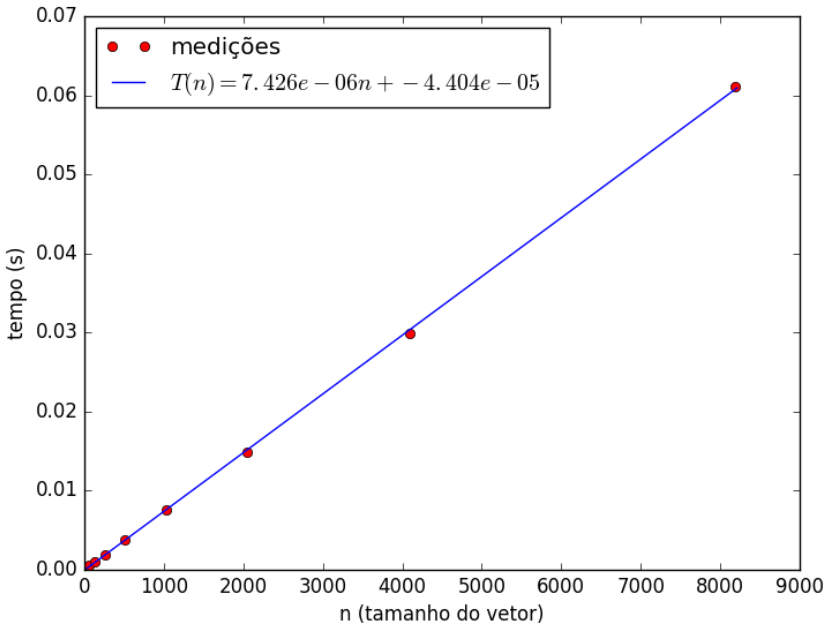

Figura 2.13: *Radix com relação ao tempo com vetor quase crescente* 40%.

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

$7.426 * 10^{-6} * n - 4.404 * 10^{-5} = 31894.4271$

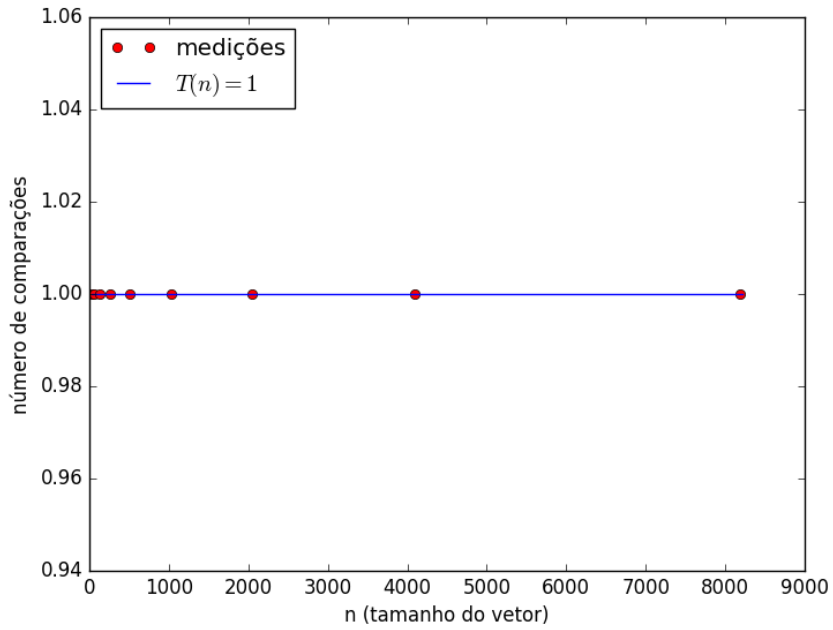

Figura 2.14: *Radix com relação ao número de comparações com vetor quase crescente* 40%.

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

1

n	comparações	tempo(s)
32	1	0.000297
64	1	0.000524
128	1	0.001011
256	1	0.001923
512	1	0.003738
1024	1	0.007561
2048	1	0.015872
4096	1	0.029650
8192	1	0.061698

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

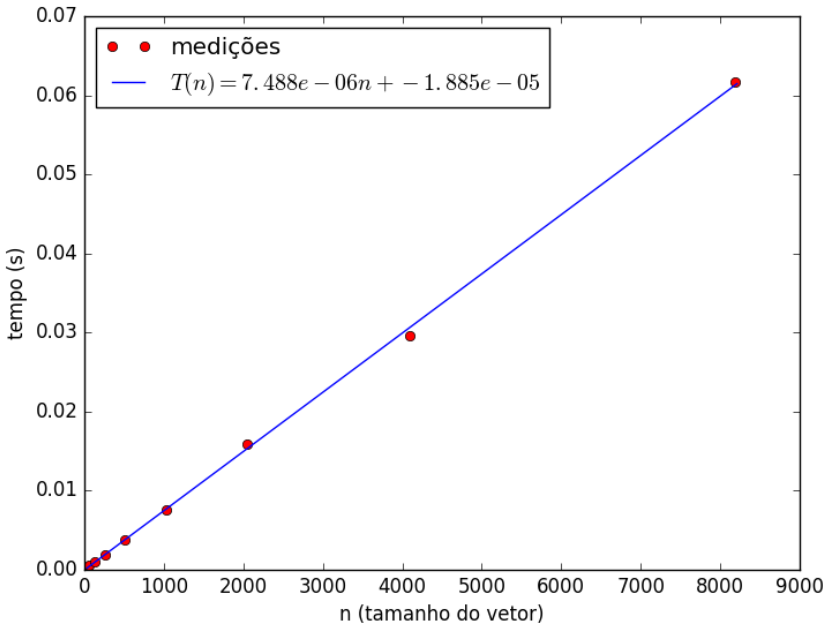

Figura 2.15: *Radix com relação ao tempo com vetor quase crescente* 50%.

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

$7.488 * 10^{-6} * n - 1.885 * 10^{-5} = 32160.71509$

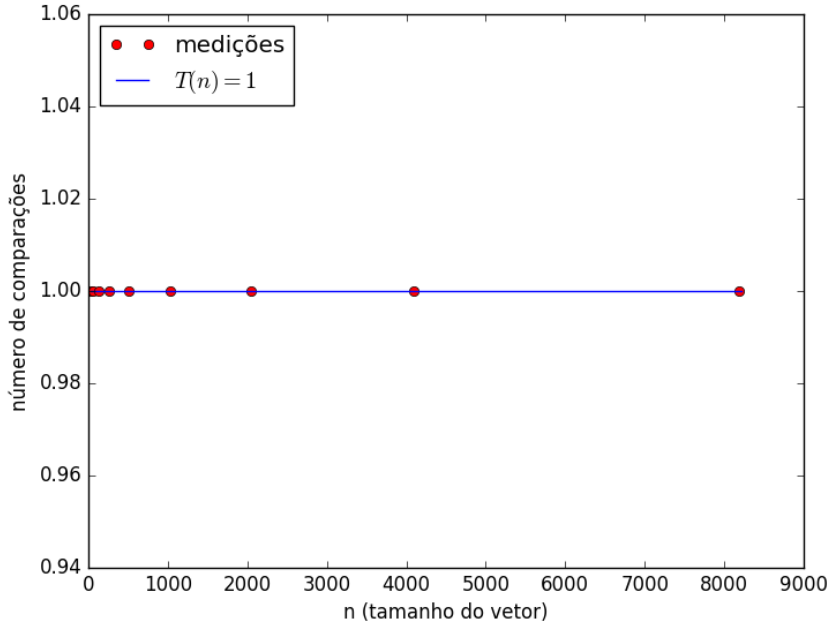

Figura 2.16: *Radix com relação ao número de comparações com vetor quase crescente* 50%.

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

1

n	comparações	tempo(s)
32	1	0.000304
64	1	0.000523
128	1	0.001040

Tabela 2.9: *RadixSort com Vetores Quase Decrescente* 10%

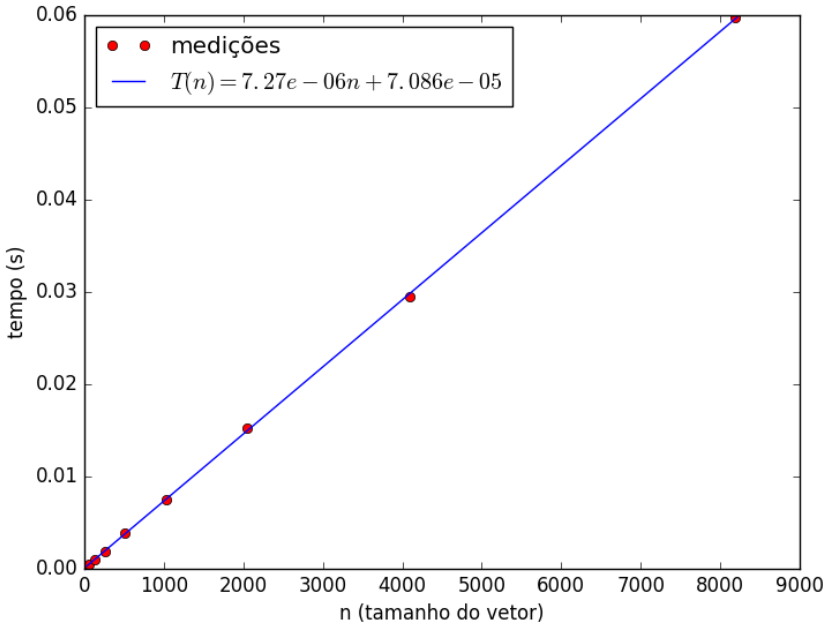

Figura 2.17: *Radix com relação ao tempo com vetor quase decrescente* 10%.

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

$7.27 * 10^{-6} * n + 7.086 * 10^{-5} = 31224.41231$

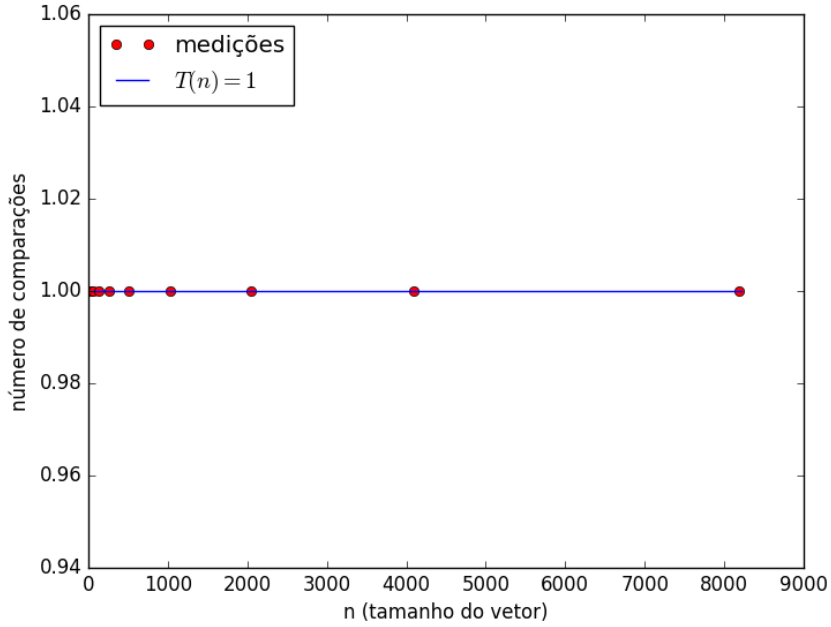

Figura 2.18: *Radix com relação ao número de comparações com vetor quase decrescente* 10%.

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

1

n	comparações	tempo(s)
32	1	0.000306
64	1	0.000557
128	1	0.001022
256	1	0.001940
512	1	0.003776
1024	1	0.007690
2048	1	0.015499
4096	1	0.030006
8192	1	0.060569

Tabela 2.10: *RadixSort com Vetores Quase Decrescente* 20%

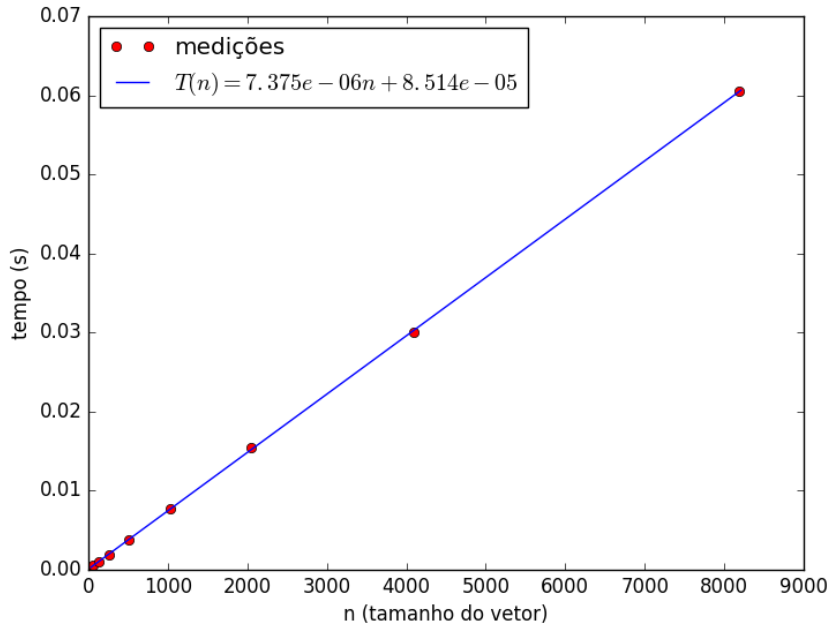

Figura 2.19: *Radix com relação ao tempo com vetor quase decrescente* 20%.

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

$7.375 * 10^{-6} * n + 8.514 * 10^{-5} = 31615.38389$

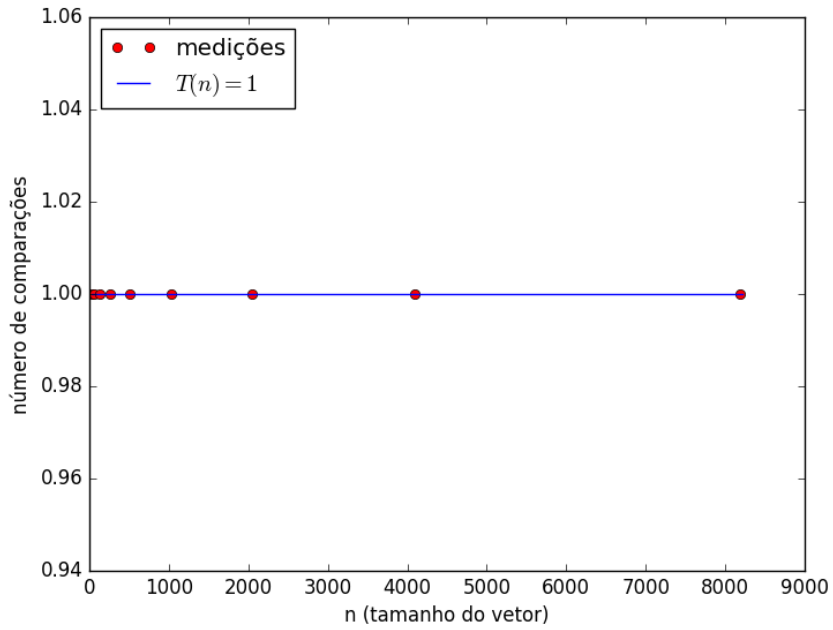

Figura 2.20: *Radix com relação ao número de comparações com vetor quase decrescente* 20%.

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

1

n	comparações	tempo(s)
32	1	0.000298
64	1	0.000567
128	1	0.000981

Tabela 2.11: *RadixSort com Vetores Quase Decrescente* 30%

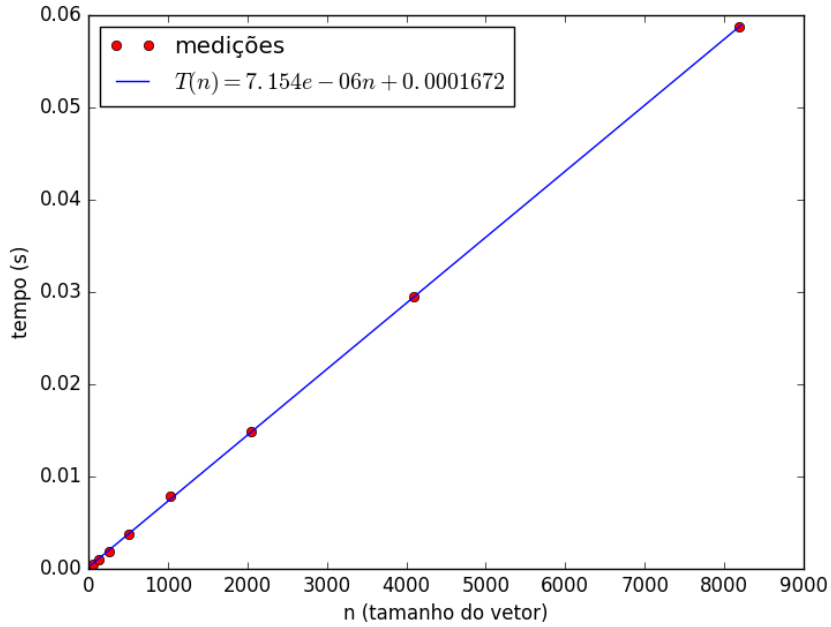

Figura 2.21: *Radix com relação ao tempo com vetor quase decrescente* 30%.

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

$7.154 * 10^{-6} * n + 0.0001672 = 30726.19604$

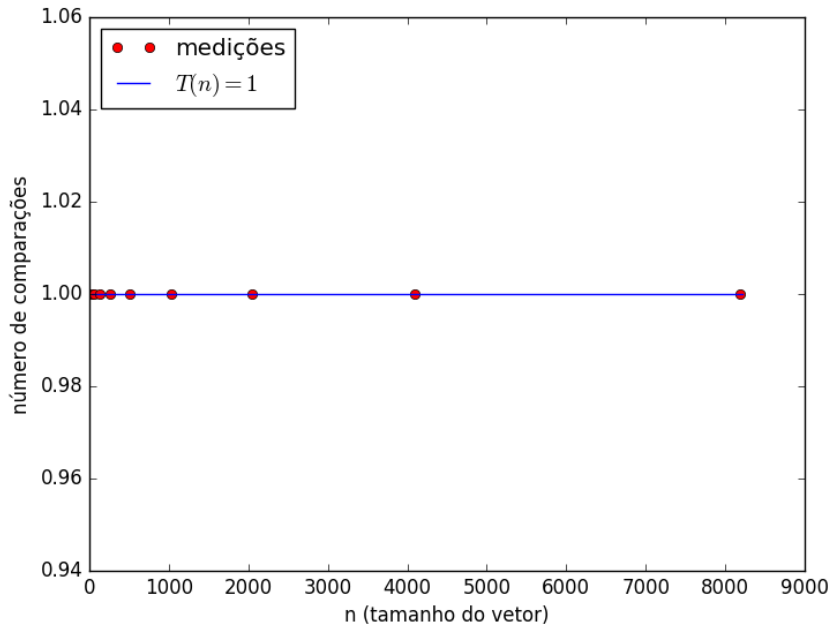

Figura 2.22: *Radix com relação ao número de comparações com vetor quase decrescente* 30%.

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

1

n	comparações	tempo(s)
32	1	0.000301
64	1	0.000532
128	1	0.000992

Tabela 2.12: *RadixSort com Vetores Quase Decrescente* 40%

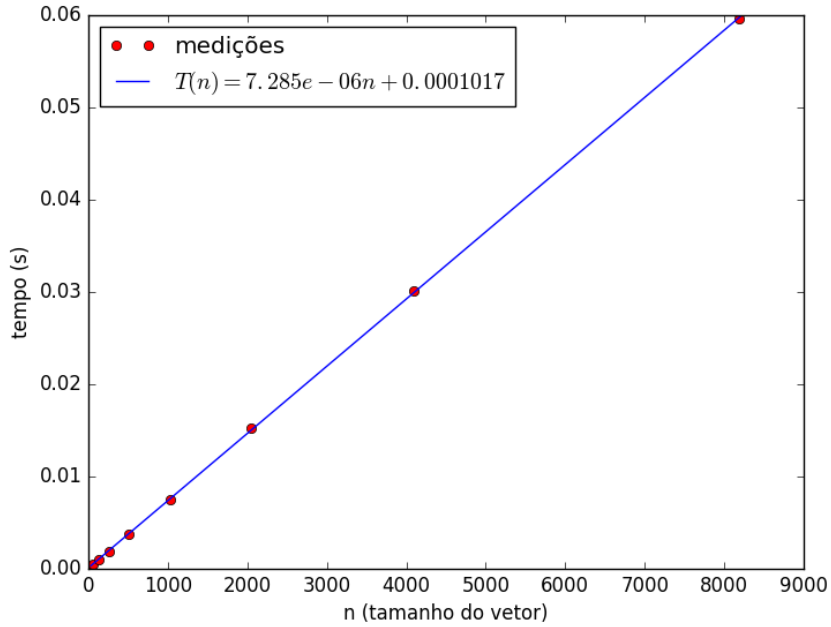

Figura 2.23: *Radix com relação ao tempo com vetor quase decrescente* 40%.

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

$7.285 * 10^{-6} * n + 0.0001017 = 31288.83675$

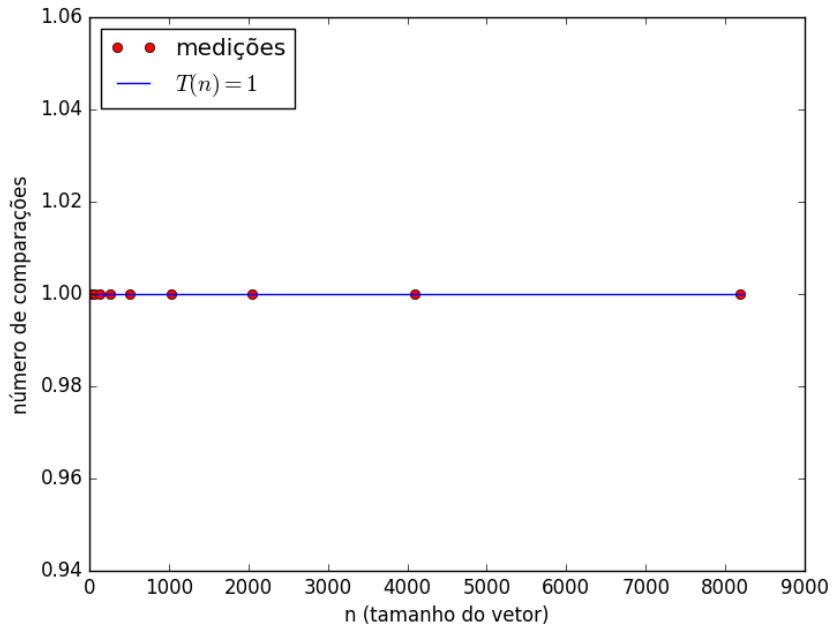

Figura 2.24: *Radix com relação ao número de comparações com vetor quase decrescente* 40%.

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

1

n	comparações	tempo(s)
32	1	0.000304
64	1	0.000569
128	1	0.001037
256	1	0.001989
512	1	0.003917
1024	1	0.007490
2048	1	0.015944
4096	1	0.029561
8192	1	0.059619

Tabela 2.13: *RadixSort com Vetores Quase Decrescente* 50%

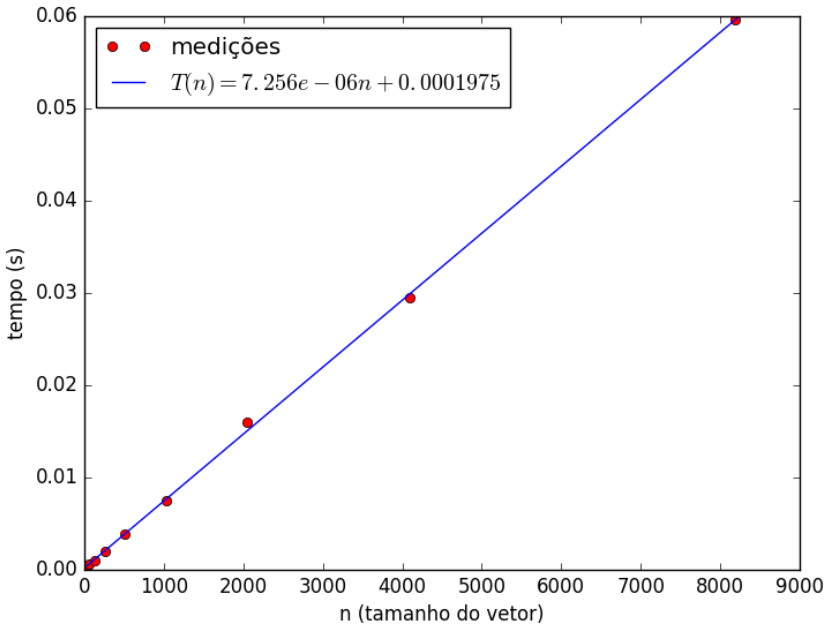

Figura 2.25: *Radix com relação ao tempo com vetor quase decrescente* 50%.

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

$7.256 * 10^{-6} * n + 0.0001975 = 31164.2827$

2.1

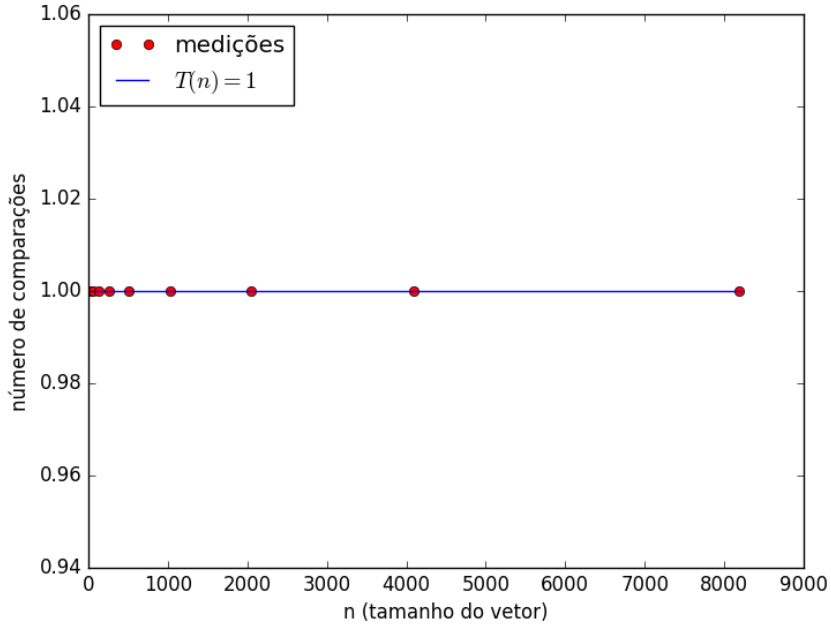

Figura 2.26: *Radix com relação ao número de comparações com vetor quase decrescente* 50%.

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

1

Apêndice A

Arquivo ../radix/radix.py

Listagem A.1: ../radix/radix.py

```
1
2 @profile
3 def radix(A):
4     A = [ int(x) for x in A ]
5     n = len(A)
6     maior = max(A)
7     d = _contaDigitos(maior)
8     radixSort(A, n, d)
9
10 #@profile
11 def radixSort(A, n, d):
12     exp = 1
13     maior = max(A)
14     while exp < maior:
15         _countingSort(A, exp)
16         exp *= 10
17
18
19 #@profile# função countingsort adaptada
20 def _countingSort(A, k):
21     contador = [0] * 10  # Contador é o histograma
22     B = [0] * len(A)
23     n = len(A)
24     for i in range(0, n):
25         contador[(A[i] // k) % 10] += 1
26
27     for i in range(1, len(contador)):
28         contador[i] += contador[i - 1]
29
30     for j in range((n - 1), -1, -1):
31         B[contador[(A[j] // k) % 10] - 1] = A[j]
32         contador[(A[j] // k) % 10] -= 1
33
34     for i in range(0, n):
35         A[i] = B[i]
36
37 #@profile
38 def _contaDigitos(valor):
39     digitos = 0
40     while (valor != 0):
41         digitos += 1
```

```
42          valor //= 10
43      return digitos
44
45
46
47
48
49
50
51
52 #lista = [170, 45, 75, 90, 802, 24, 2, 66]
53 #radix(lista)
54 #print(lista)
```

Apêndice B

Arquivo ../radix/ensaio.py

Listagem B.1: ../radix/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)
```