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 algoritmo quicksort baseia-se no paradigma de divisão e conquista, pois divide o vetor em dois subvetores da seguinte forma: escolhe um elemento como pivô e particiona o vetor de forma com que todos os elementos menores que o pivô sejam colocados à sua esquerda e todos os elementos maiores que o pivô sejam colocados à sua direta. Depois disso, ordena os dois subvetores recursivamente utilizando o quicksort.

O pior caso do Quicksort ocore quando a subrotina Particiona divide o vetor de forma desbalanceada, ou seja, divide o vetor em dois subvetores de tal forma que o tamanho de um deles é zero e o outro tem tamanho igual ao tamanho do vetor menos 1. Temos que cada etapa recursiva chama vetores de tamanho igual ao vetor anterior menos 1. Assim, teríamos a seguinte recorência representada por $T(n)$:

$T(n) = T(0) + T(n-1) + teta(n) = T(n-1) + teta(n)$

Dessa forma, ao somar os custos obtidos para todas as recursões, temos que a complexidade de tempo no pior caso é $T(n)$ pertence a $teta(n^2)$. É possível minimizar as chances do pior caso ocorrer escolhendo o pivô de forma aleatória.

O melhor caso do Quicksort ocorre quando a subrotina Particiona divide o vetor na metade, ou seja, o tamanho dos subvetores é igual ou ficam com diferença de um. Neste caso, o algoritmo é executado com mais eficiência. Assim, teríamos a seguinte recorrência representada por $T(n)$:

$T(n) = 2T(n/2) + teta(n)$

Essa recorrência pode ser facilmente provada pelo Teorema Master, $T(n)$ pertence a $teta(n * lgn)$. Além disso, no caso médio do Quicksort, a complexidade de tempo é mais próxima do melhor caso do que do pior caso.

Em termos de memória, esse algoritmo é muito eficiente devido ao fato de usar espaço constante, isto é, a sua complexidade de espaço é $teta(1)$.

Capítulo 2

Resultados

2.1 Tabelas

n	comparações	tempo(s)
32	41	0.000522
64	85	0.001133
128	171	0.002419
256	345	0.004859
512	683	0.011166
1024	1381	0.022620
2048	2777	0.048469
4096	5801	0.118723
8192	12791	0.267342

Tabela 2.1: *QuickSort com Vetores Aleatorio*

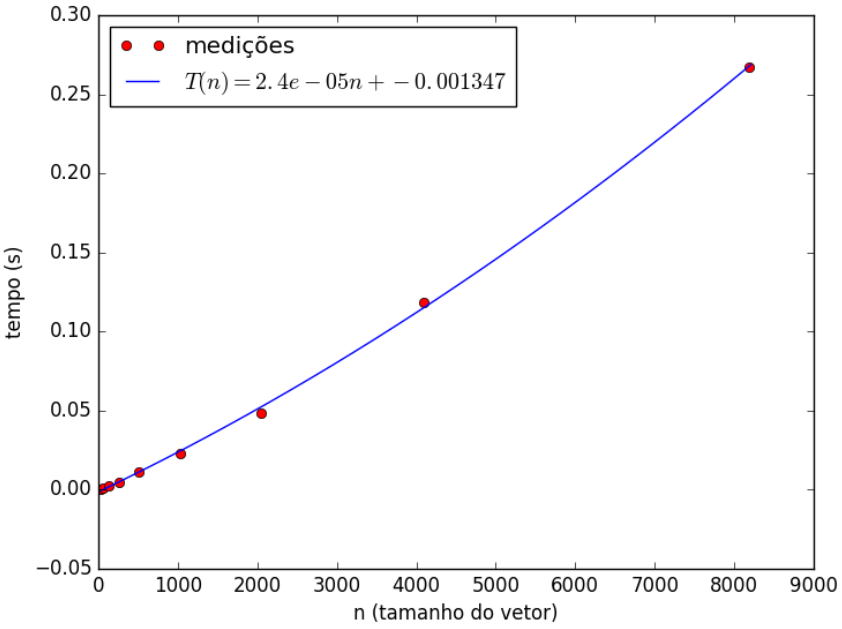

Figura 2.1: *Quicksort com relação ao tempo com vetor aleatório.*

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

$2.4 * 10^{-5} * n - 0.001347 = 103079.2151$

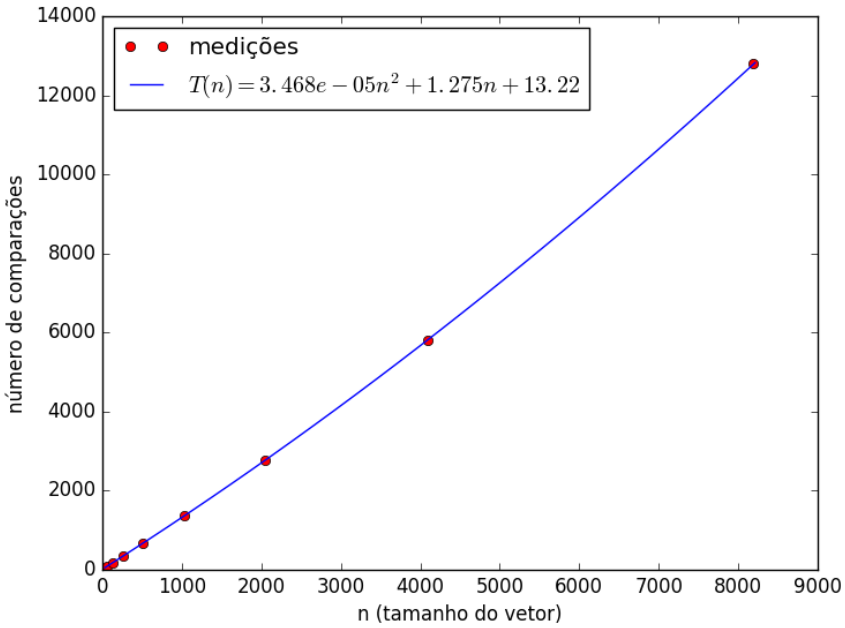

Figura 2.2: *Quicksort com relação ao número de comparações com vetor aleatório.*

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

$3.468 * 10^{-5} * n^2 + 1.275 * n + 13.22 = 6.397385e + 14$

n	comparações	tempo(s)
32	45	0.000506
64	81	0.001047
128	169	0.002155
256	339	0.004790
512	689	0.010673
1024	1385	0.022489
2048	2739	0.048894
4096	4739	0.105320
8192	12423	0.257012

Tabela 2.2: *InsertionSort com Vetores Crescentes*

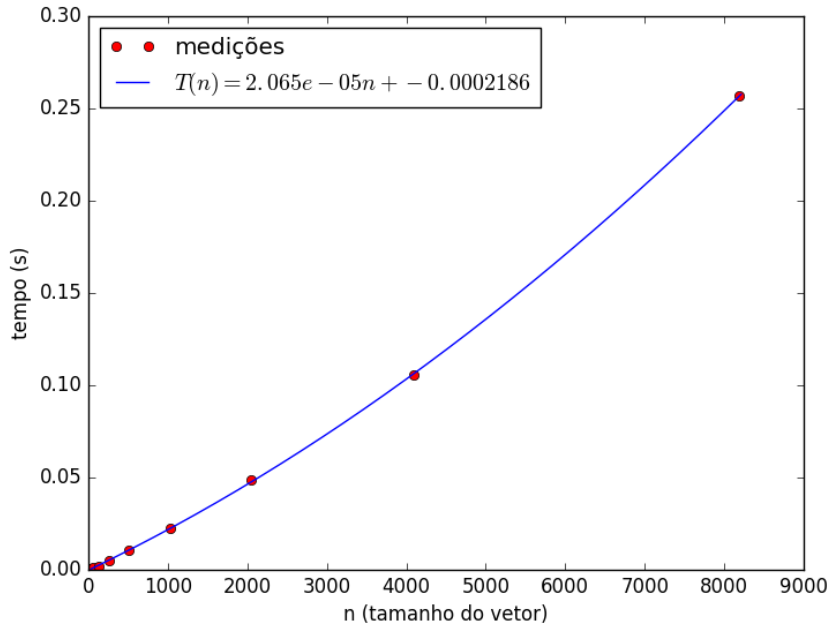

Figura 2.3: *Quicksort com relação ao tempo com vetor crescente.*

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

$2.065 * 10^{-5} * n - 0.002186 = 88691.07466$

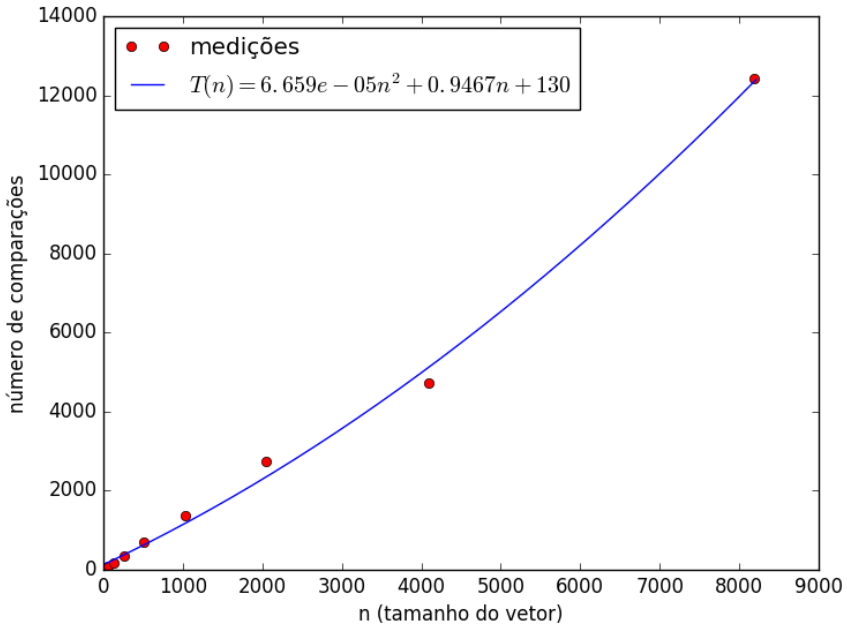

Figura 2.4: *Quicksort com relação ao número de comparações com vetor crescente.*

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

$6.659 * 10^{-5} * n^2 + 0.9467 * n + 130 = 1.228409e + 14$

n	comparações	tempo(s)
32	43	0.000503
64	89	0.000991
128	177	0.002216
256	341	0.004712
512	673	0.010046
1024	1359	0.022621
2048	2741	0.047068
4096	4783	0.101985
8192	12429	0.242304

Tabela 2.3: *InsertionSort com Vetores Decrescentes*

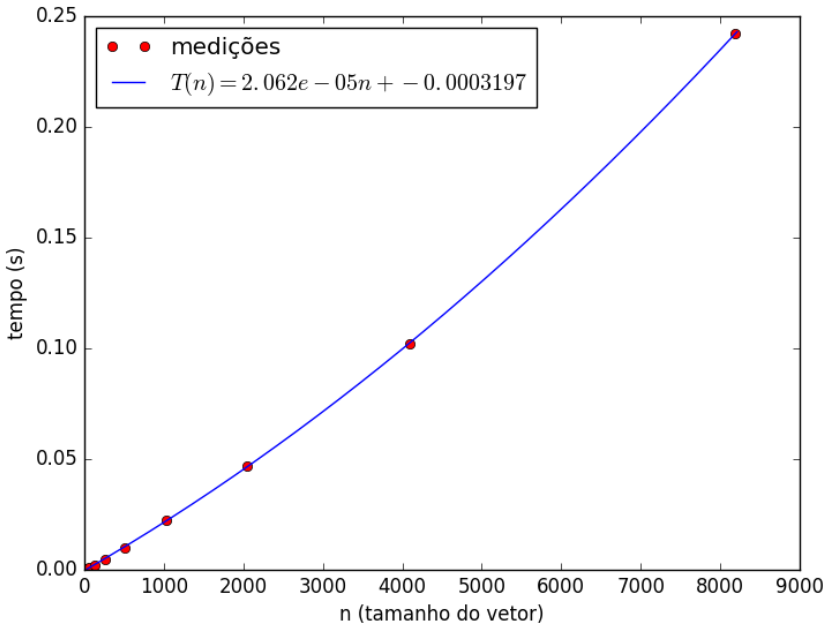

Figura 2.5: *Quicksort com relação ao tempo com vetor decrescente.*

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

$2.062 * 10^{-5} * n - 0.0003197 = 88562.22564$

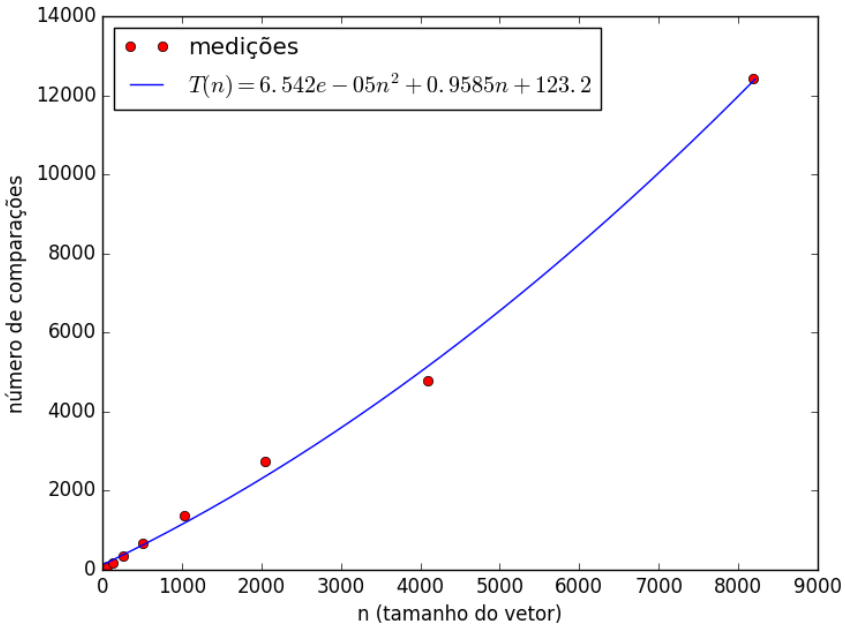

Figura 2.6: *Quicksort com relação ao número de comparações com vetor decrescente.*

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

$6.542 * 10^{-5} * n^2 + 0.9585 * n + 123.2 = 1.206790e + 14$

n	comparações	tempo(s)
32	41	0.000506
64	87	0.001027
128	169	0.002360
256	343	0.005228
512	685	0.010188
1024	1383	0.023080
2048	2703	0.048103
4096	4867	0.103046
8192	12417	0.235479

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

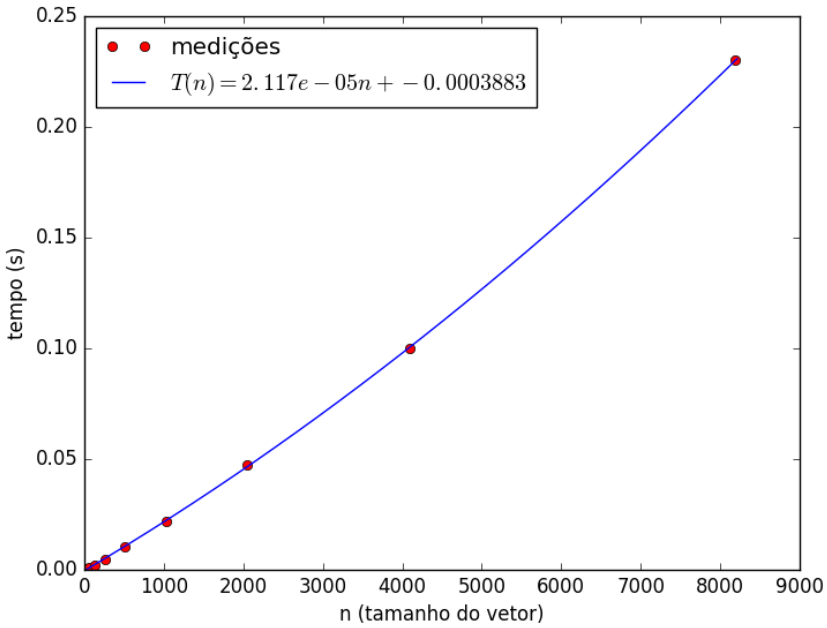

Figura 2.7: *quicksort com relação ao tempo com vetor quase crescente* 10%.

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

$2.117 * 10^{-5} * n - 0.0003883 = 90924.45766$

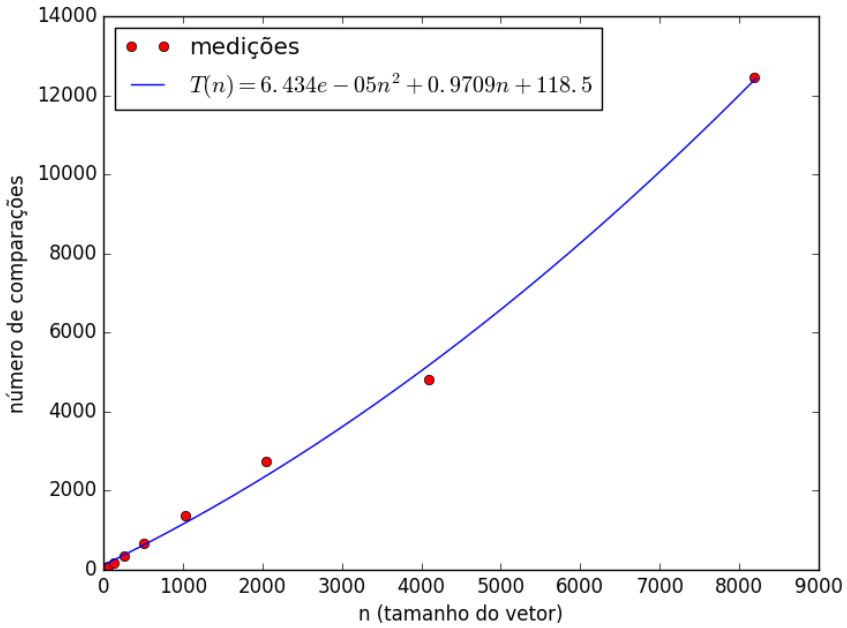

Figura 2.8: *Quicksort com relação ao número de comparações com vetor quase crescente* 10%.

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

$6.434 * 10^{-5} * n^2 + 0.9709 * n + 118.5 = 1.196852e + 14$

n	comparações	tempo(s)
32	43	0.000476
64	85	0.001082
128	169	0.002173
256	333	0.004879
512	695	0.010162
1024	1375	0.022948
2048	2745	0.052480
4096	4841	0.099375
8192	12419	0.232312

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

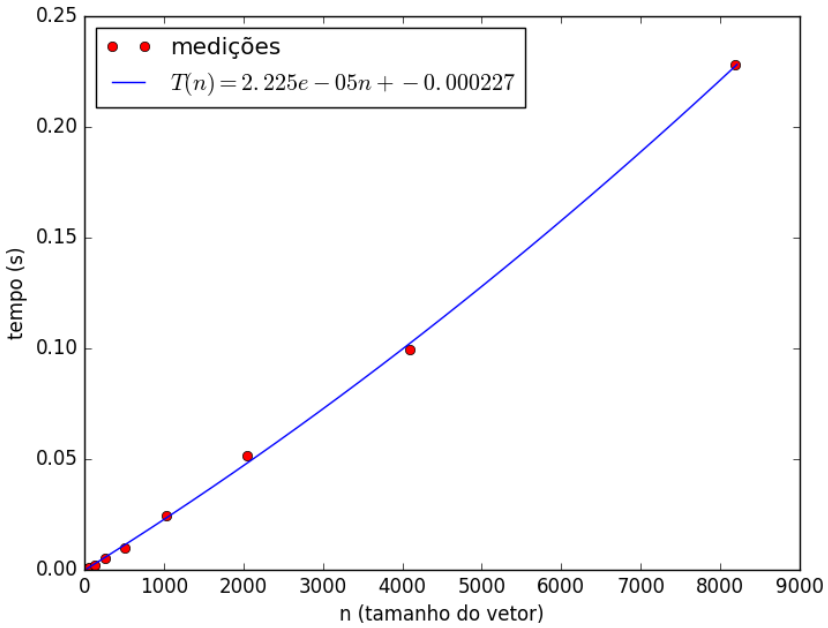

Figura 2.9: *Quicksort com relação ao tempo com vetor quase crescente* 20%.

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

$2.225 * 10^{-5} * n - 0.000227 = 95563.02234$

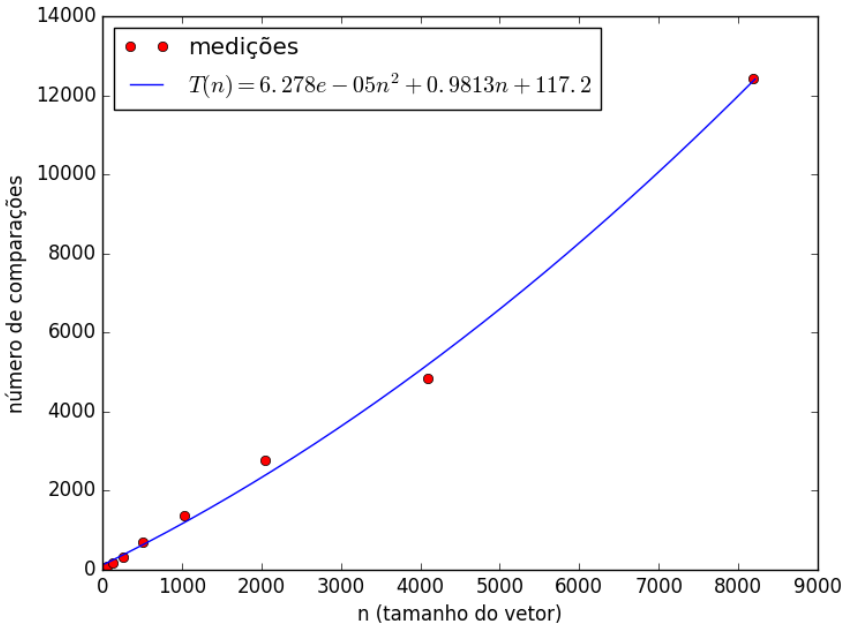

Figura 2.10: *Quicksort com relação ao número de comparações com vetor quase crescente* 20%.

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

$6.278 * 10^{-5} * n^2 + 0.9813 * n + 117.2 = 1.1580908e + 15$

n	comparações	tempo(s)
32	49	0.000547
64	83	0.001047
128	175	0.002234
256	345	0.005267
512	691	0.010421
1024	1331	0.022471
2048	2735	0.050717
4096	4859	0.106128
8192	12443	0.247242

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

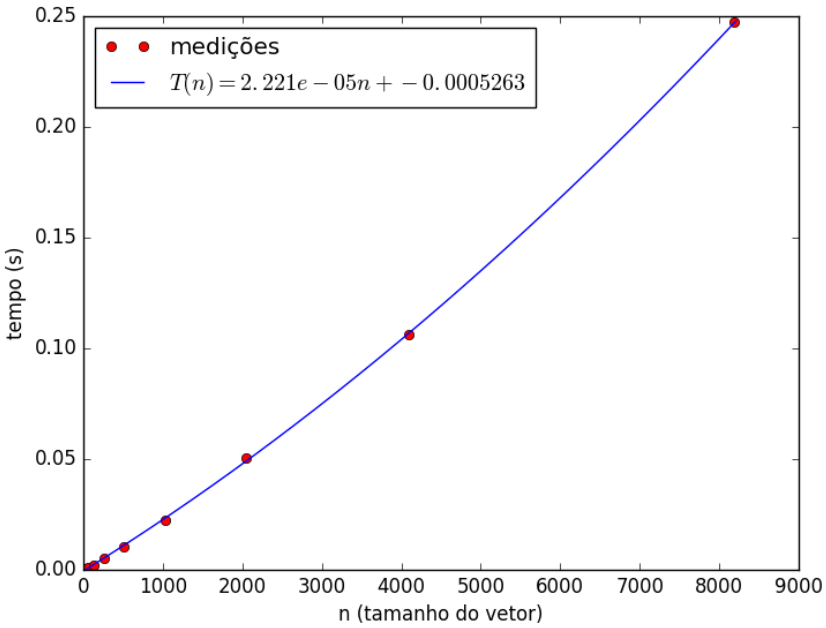

Figura 2.11: *Quicksort com relação ao tempo com vetor quase crescente* 30%.

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

$2.221 * 10^{-5} * n - 0.0005263 = 95291.22364$

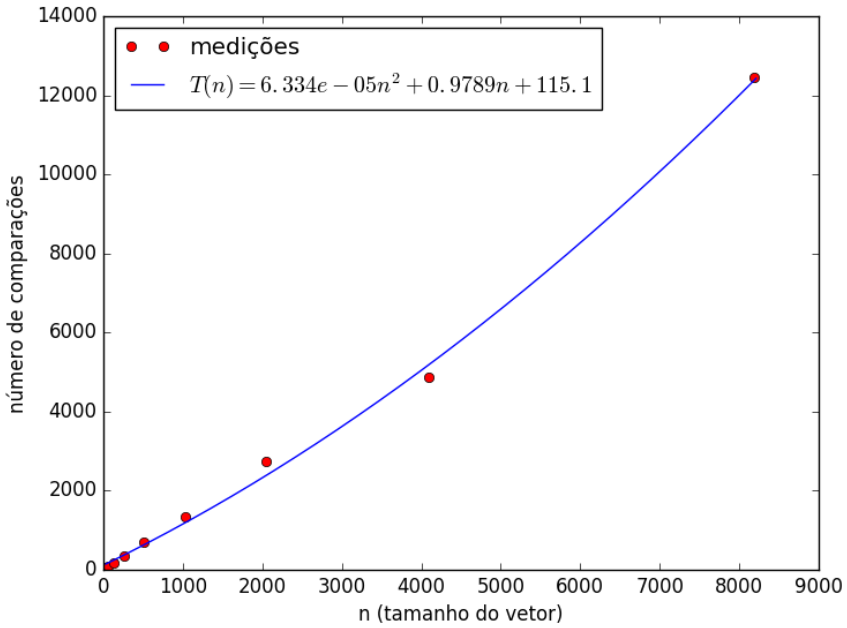

Figura 2.12: *Quicksort com relação ao número de comparações com vetor quase crescente* 30%.

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

$6.334 * 10^{-5} * n^2 + 0.9789 * n + 115.1 = 1.1684209e + 15$

n	comparações	tempo(s)
32	45	0.000485
64	85	0.001019
128	173	0.002363
256	357	0.004942
512	693	0.010198
1024	1365	0.023385
2048	2719	0.049659
4096	4817	0.093524
8192	12441	0.242838

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

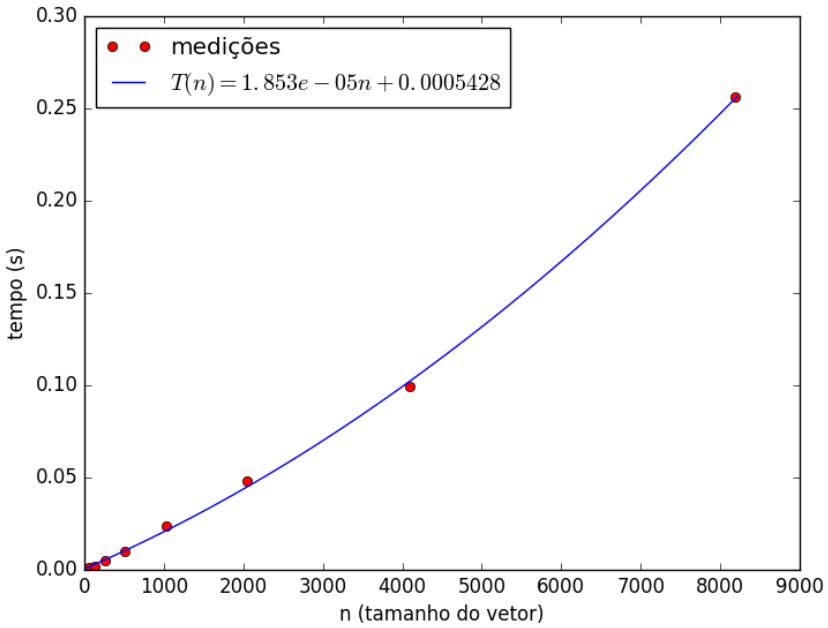

Figura 2.13: *Quicksort com relação ao tempo com vetor quase crescente* 40%.

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

$1.853 * 10^{-5} * n + 0.0005428 = 79585.74399$

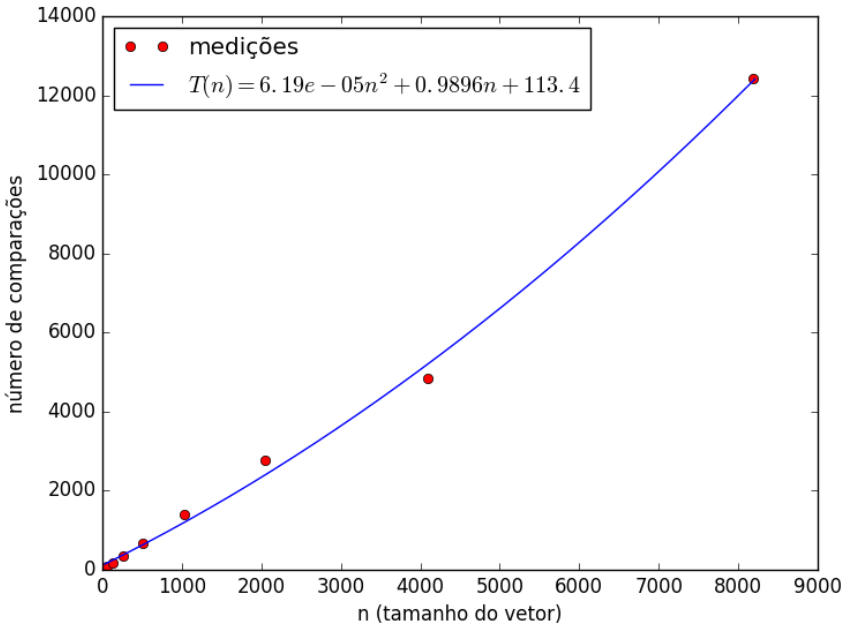

Figura 2.14: *Quicksort com relação ao número de comparações com vetor quase crescente* 40%.

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

$6.19 * 10^{-5} * n^2 + 0.9896 * n + 113.4 = 1.1418577e + 15$

n	comparações	tempo(s)
32	45	0.000522
64	85	0.001082
128	169	0.002092
256	345	0.004797
512	693	0.010744
1024	1375	0.022203
2048	2753	0.050512
4096	4809	0.105051
8192	12451	0.248645

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

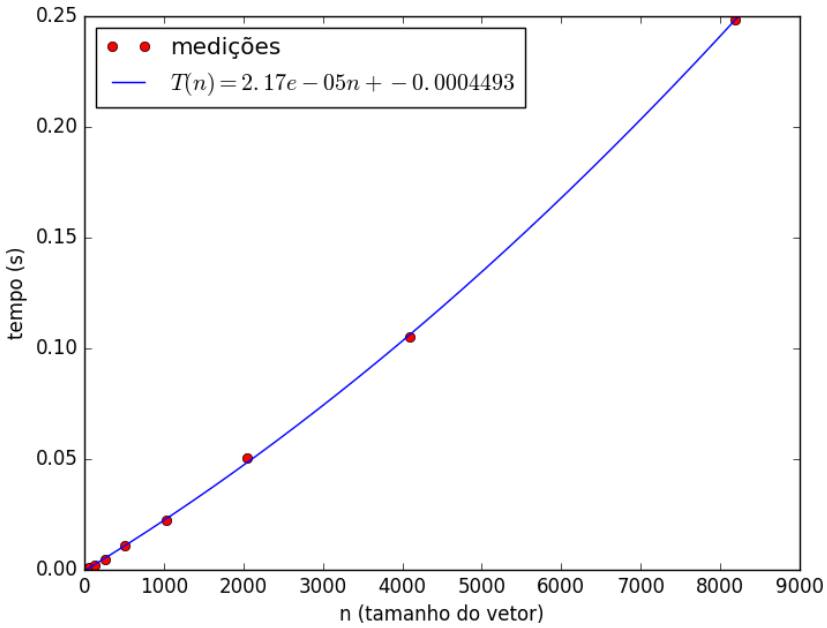

Figura 2.15: *Quicksort com relação ao tempo com vetor quase crescente* 50%.

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

$2.17 * 10^{-5} * n + 0.0004493 = 93200.79032$

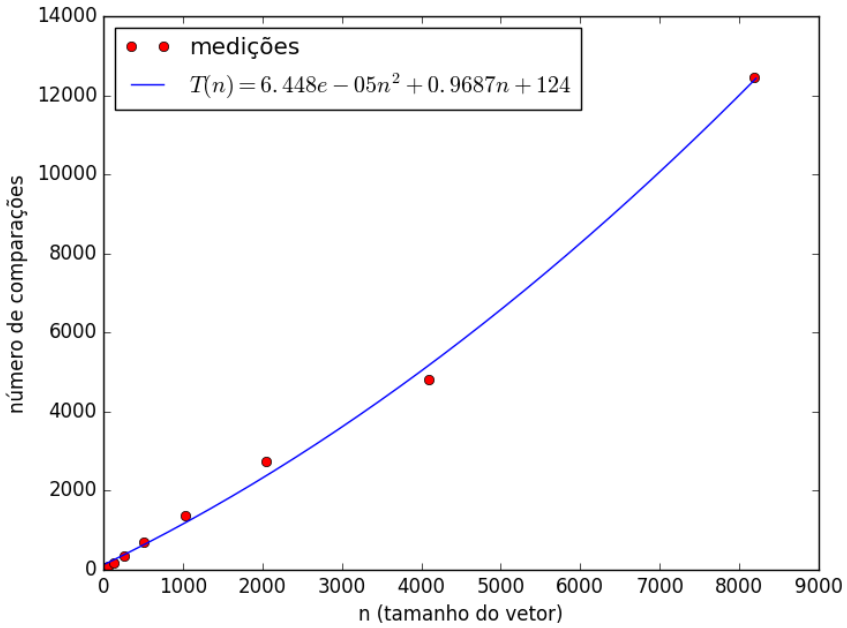

Figura 2.16: *Quicksort com relação ao número de comparações com vetor quase crescente* 50%.

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

$6.448 * 10^{-5} * n^2 + 0.9687 * n + 124 = 1.189450e + 15$

n	comparações	tempo(s)
32	39	0.000518
64	83	0.001028
128	177	0.002340
256	355	0.004682
512	689	0.011057
1024	1373	0.023594
2048	2731	0.051619
4096	4829	0.101065
8192	12425	0.232339

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

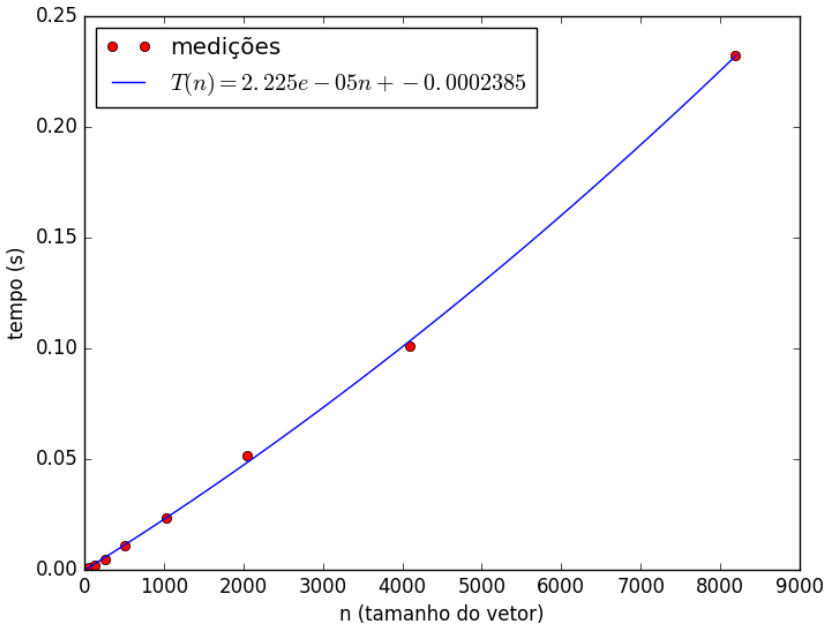

Figura 2.17: *Quicksort com relação ao tempo com vetor quase decrescente* 10%.

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

$2.225 * 10^{-5} * n + 0.0002385 = 95563.02234$

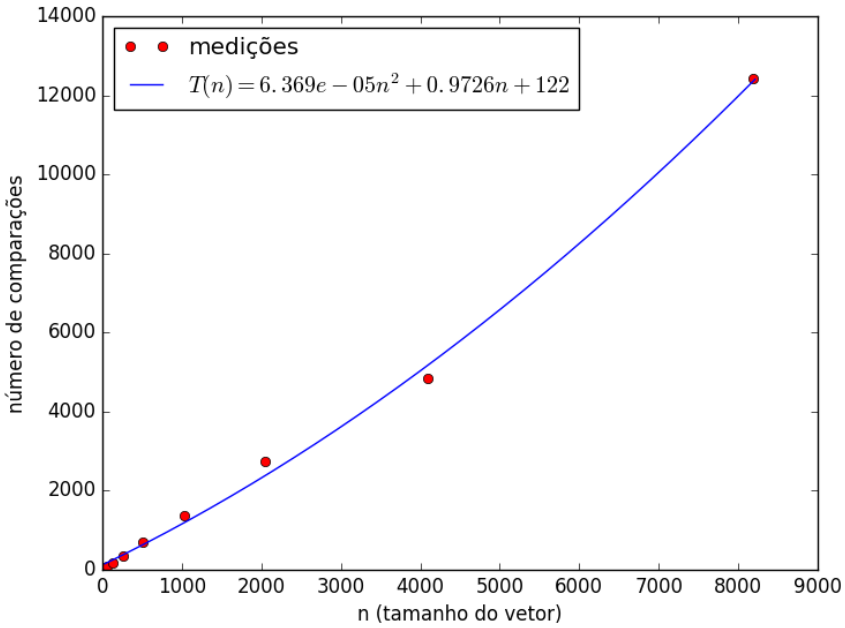

Figura 2.18: *Quicksort com relação ao número de comparações com vetor quase decrescente* 10%.

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

$6.369 * 10^{-5} * n^2 + 0.9726 * n + 122 = 1.1748773e + 15$

n	comparações	tempo(s)
32	39	0.000511
64	81	0.001074
128	179	0.002347
256	337	0.004851
512	679	0.011101
1024	1347	0.023786
2048	2727	0.049848
4096	4831	0.099260
8192	12447	0.231600

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

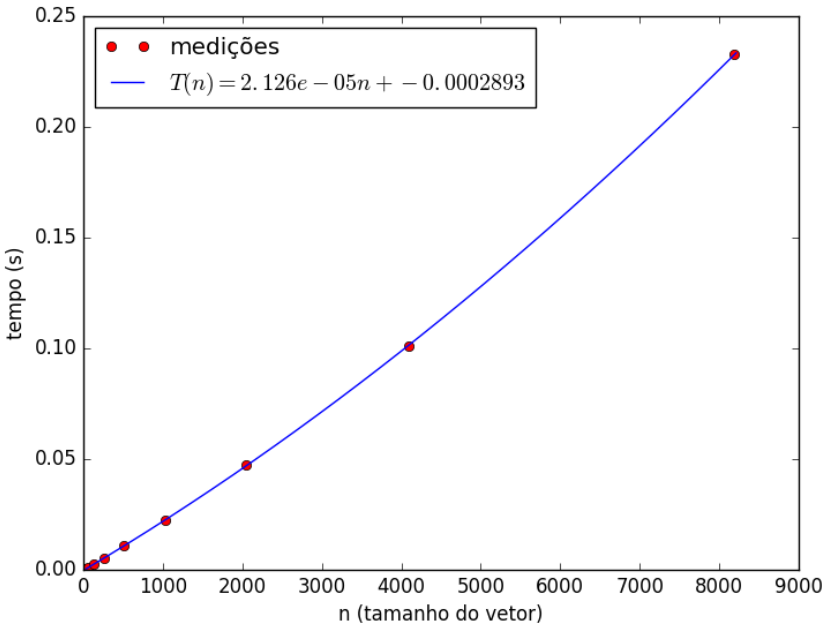

Figura 2.19: *Quicksort com relação ao tempo com vetor quase decrescente* 20%.

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

$2.126 * 10^{-5} * n + 0.0002893 = 91311.00471$

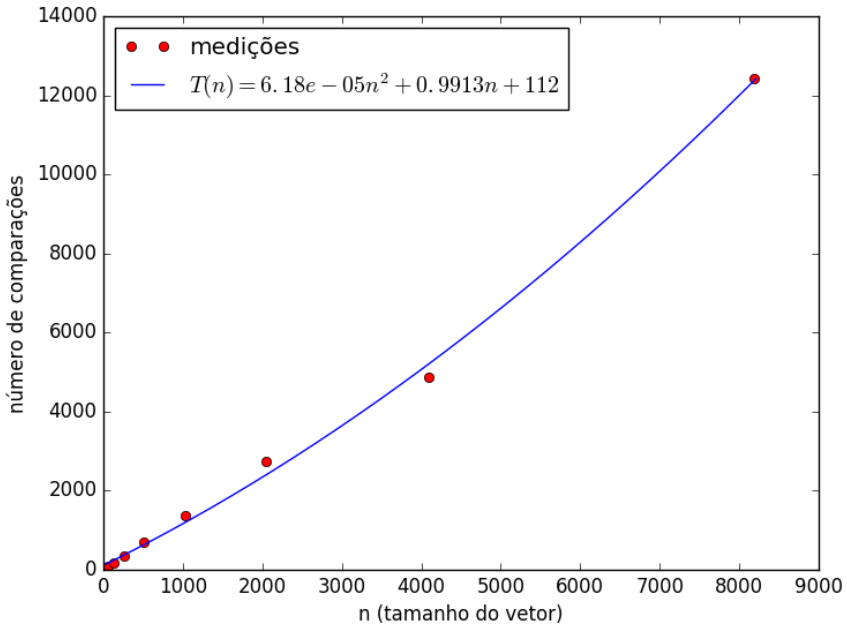

Figura 2.20: *Quicksort com relação ao número de comparações com vetor quase decrescente* 20%.

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

$6.18 * 10^{-5} * n^2 + 0.9913 * n + 112 = 1.140013e + 15$

n	comparações	tempo(s)
32	41	0.000483
64	87	0.001082
128	165	0.002275
256	343	0.004844
512	695	0.010762
1024	1361	0.022350
2048	2727	0.047655
4096	4847	0.098957
8192	12445	0.248575

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

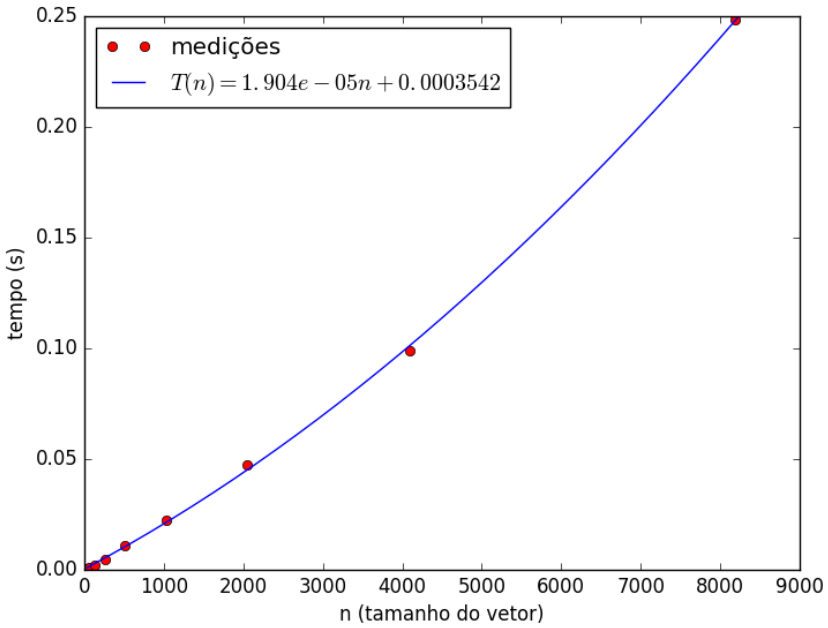

Figura 2.21: *Quicksort com relação ao tempo com vetor quase decrescente* 30%.

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

$1.904 * 10^{-5} * n + 0.0003542 = 81776.17732$

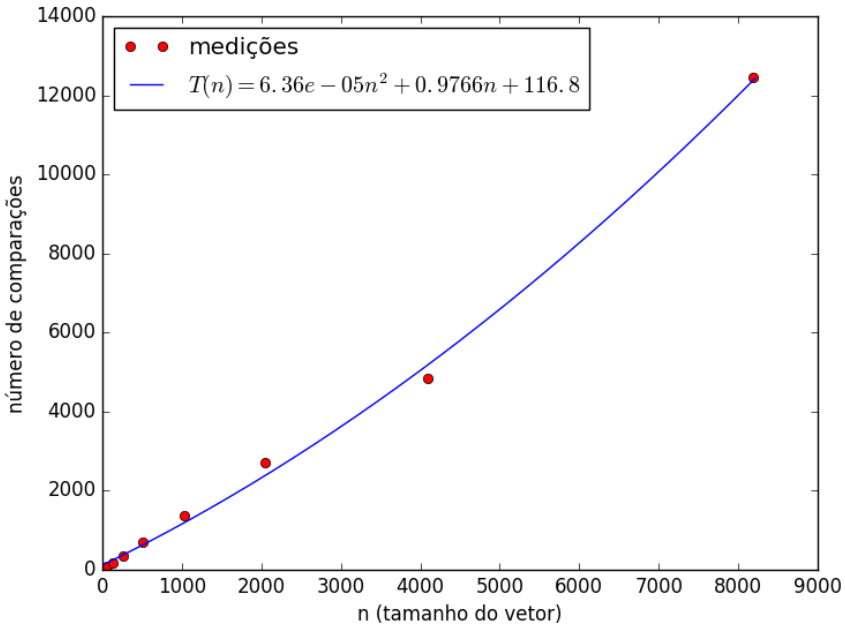

Figura 2.22: *Quicksort com relação ao número de comparações com vetor quase decrescente* 30%.

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

$6.36 * 10^{-5} * n^2 + 0.9766 * n + 116.8 = 1.1732171e + 15$

n	comparações	tempo(s)
32	41	0.000550
64	87	0.001082
128	173	0.002341
256	339	0.005133
512	691	0.010205
1024	1369	0.022400
2048	2687	0.048879
4096	4821	0.103010
8192	12425	0.241535

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

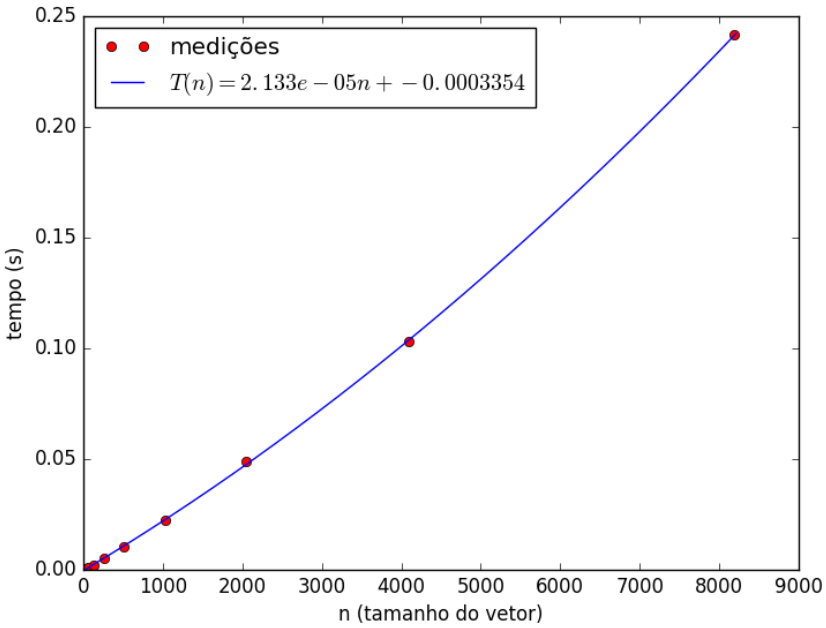

Figura 2.23: *Quicksort com relação ao tempo com vetor quase decrescente* 40%.

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

$2.133 * 10^{-5} * n + 0.0003354 = 91611.65242$

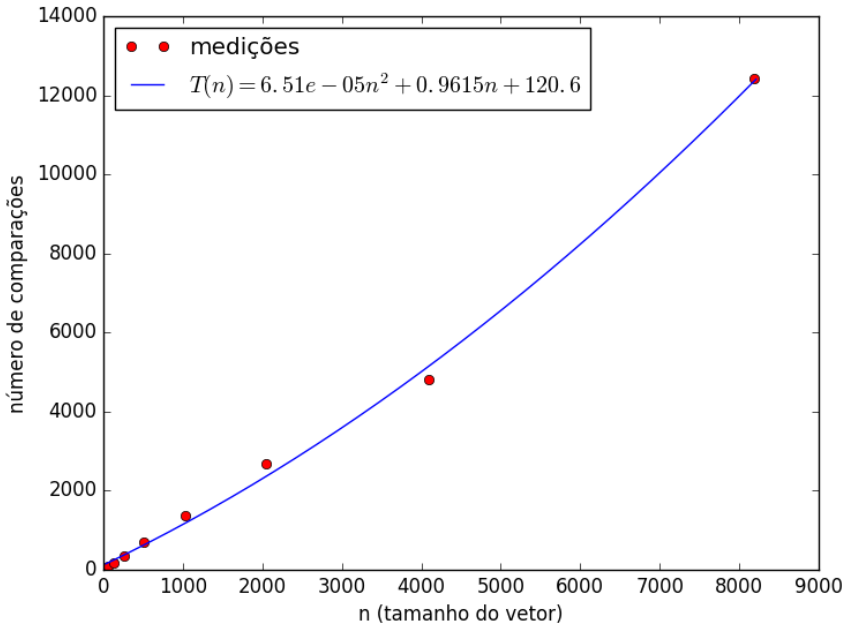

Figura 2.24: *Quicksort com relação ao número de comparações com vetor quase decrescente* 40%.

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

$6.51 * 10^{-5} * n^2 + 0.9615 * n + 120.6 = 1.2008871e + 15$

n	comparações	tempo(s)
32	45	0.000524
64	85	0.001129
128	169	0.002331
256	339	0.004671
512	681	0.010102
1024	1367	0.023496
2048	2745	0.048454
4096	4923	0.099790
8192	12431	0.243797

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

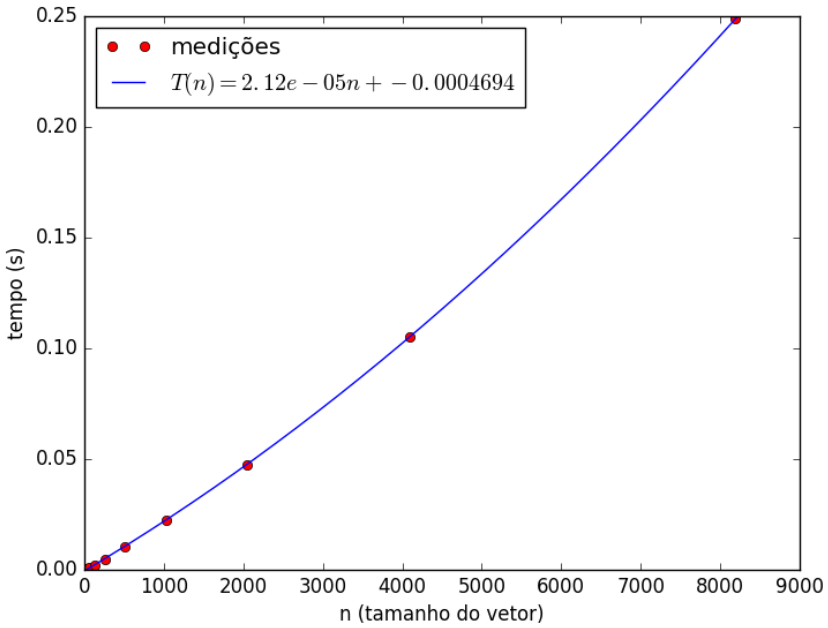

Figura 2.25: *Quicksort com relação ao tempo com vetor quase decrescente* 50%.

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

$2.12 * 10^{-5} * n + 0.0004694 = 91053.30668$

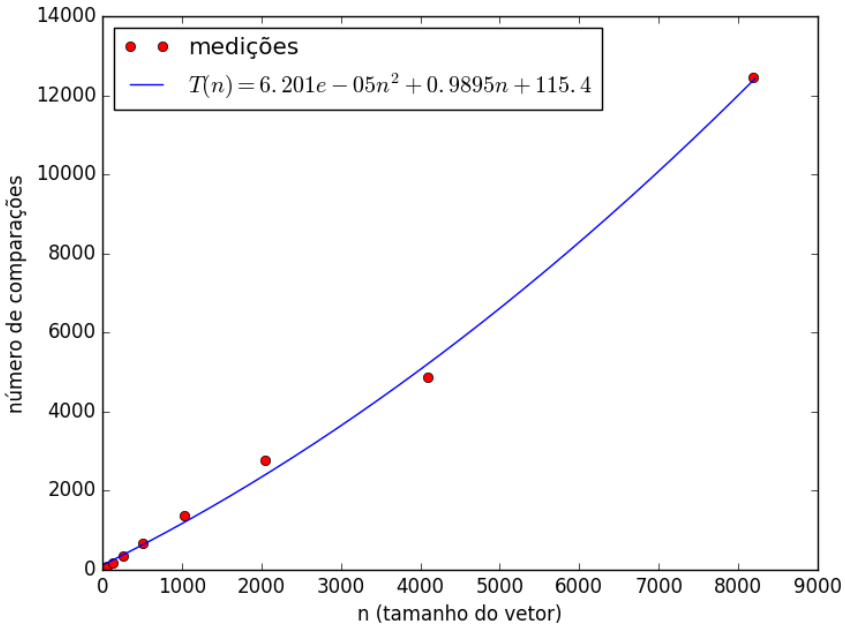

Figura 2.26: *Quicksort com relação ao número de comparações com vetor quase decrescente* 50%.

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

$6.201 * 10^{-5} * n^2 + 0.9895 * n + 115.4 = 1.143888685e + 15$

Apêndice A

Arquivo ../quicksort/quicksort.py

Listagem A.1: ../quicksort/quicksort.py

```
1  import random
2
3  def quicksort( aList ):
4    _quicksort( aList, 0, len( aList ) - 1 )
5
6  @profile
7  def _quicksort( aList, first, last ):
8      if first < last:
9        pivot = partition( aList, first, last )
10       _quicksort( aList, first, pivot - 1 )
11       _quicksort( aList, pivot + 1, last )
12
13 def partition( aList, first, last ) :
14     pivot = first + random.randrange( last - first + 1 )
15     swap( aList, pivot, last )
16     for i in range( first, last ):
17       if aList[i] <= aList[last]:
18         swap( aList, i, first )
19         first += 1
20
21     swap( aList, first, last )
22     return first
23
24
25 def swap( A, x, y ):
26   A[x],A[y]=A[y],A[x]
```

Apêndice B

Arquivo ../quicksort/ensaio.py

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

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