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

1.0.1 Desempenho do quicksort no pior caso

Podemos supor que o *partition* não consome mais que $5n + 6$ unidades de tempo, sendo $n = r - p + 1$. Então o consumo de tempo do *quicksort* no pior caso, digamos $T(n)$, satisfaz a seguinte recorrência

$$T(n) = 5n + 7 + max_{0 \leq k \leq n}(T(k) + T(n - k)$$

onde o máximo é tomado sobre todos os possíveis valores de k no intervalos 1.. $n-1$. Podemos supor $T(1) = 1$. Assim, por exemplo, $T(2) \leq 10 + 7 + T(1) + T(1) = 19$. Em geral, vamos mostrar que

$$T(n) \leq 5n^2$$

para $n = 1, 2, 3, ...$ a desigualdade é certamente verdadeira quando $n = 1, 2, 3$. Agora suponha que $n > 3$ e suponha que a desigualdade vale para $T(i)$ sempre que $i < n$. Teremos então

$$\begin{aligned} T(n) &= 5n + 7 + max_k(T(k) + T(n - k)) \\ &\leq 5n + 7 + max_k(5 * k^2 + 5(n - k)^2) \\ &= 5n + 7 + 5(1 + T(n - 1)^2) \\ &= 5 * n^2 - 5 * n + 17 \\ &\leq 5 * n^2 \end{aligned}$$

O terceiro passo da prova segue da seguinte observação: para n fixo e k variando entre 1 e $n - 1$, a expressão $k^2 + (n - k)^2$ tem valor máximo nos pontos $k = 1$ e $k = n - 1$ (e atinge o mínimo quando k está próximo de $n/2$). O último passo da prova está correto pois quando $n > 3$ tem-se $-5 * n + 17 \leq 0$.

A recorrência pode ser reescrita como $T(n) = 5n + 7 + max_{0 \leq k \leq n}(T(k) + T(n - k) + O(n)$ daí se deduz que $T(n)$ está em $O(n^2)$. Conclusão: no pior caso, o algoritmo *quickSort* é quadratico.

1.0.2 Desempenho típico e melhor caso do quicksort

O melhor desempenho do Quicksort ocorre quando todas as invocações de *partition* dividem o vetor na proporção (1/2)-para-(1/2), ou seja, quando todas as invocações devolvem um índice que está a meio caminho entre p e r. O consumo de tempo do algoritmo nesse caso, digamos $S(n)$, satisfaz a recorrência

$$S(n) = S(\lceil n/2 \rceil) + S(\lfloor n/2 \rfloor) + 5 * n + 7$$

para todo natural $n \geq 2$, podemos mostrar que

$$S(n) \leq 12 * n \log_2 n$$

para todo natural $n \geq 3$

O comportamento não é muito diferente quando o vetor é dividido de maneira menos equilibrada. Suponha, por exemplo, que todas as invocações de Separe dividem o vetor na proporção (1/9)-para-(8/9). Nesse caso, o consumo de tempo, digamos $R(n)$, satisfaz recorrência da forma

$$R(n) = R(\lceil n/9 \rceil) + R(\lfloor (8 * n)/9 \rfloor + 5 * n + 7$$

para todo natural $n \geq 2$. Mostremos que

$$R(n) =\leq 7 * n \log_{9/8} n$$

para todo $n \geq 2$. Supondo $R(1) = 1$, é fácil verificar que a desigualdade vale quando $n = 2, 3$. Agora tome $n \geq 4$. Então, por hipótese de indução, e usando sempre log na base 9/8,

$$\begin{aligned} R(n) &= 7 * \lceil n/9 \rceil * \log \lceil n/9 \rceil + 7 * \lfloor (8 * n)/9 \rfloor * \log \lfloor (8 * n)/9 \rfloor + 5 * n + 7 \\ &\leq 7 * \lceil n/9 \rceil + \log \lfloor (8 * n)/9 \rfloor + 7 * \lfloor (8 * n)/9 \rfloor * \log \lfloor (8 * n)/9 \rfloor + 5 * n + 7 \\ &= 7 * n \log(n) + 7 * n \log(8/9) + 5 * n + 7 \\ &= 7 * n \log(n) - 7 * n + 5 * n + 7 \\ &= 7 * n \log(n) - 2 * n + 7 \\ &< 7 * n \log(n) \end{aligned}$$

A análise dos dois casos acima — a divisão na proporção (1/2)-para-(1/2) e a divisão na proporção (1/9)-para-(8/9) — sugere a seguinte conclusão: se Separe sempre divide o vetor em alguma proporção tn-para-$(1-t)n$, o consumo de tempo do Quicksort é $O(n \log_2(n))$

As conclusões valem mesmo que o valor de t seja diferente em cada invocação de *partition*. Essas observações sugerem que o consumo de tempo típico do *quicksort* está em $O(n \log_2(n))$.

Capítulo 2

Resultados

2.1 Tabelas

n	comparações	tempo(s)
32	45	0.000468
64	85	0.000964
128	167	0.002285
256	345	0.004648
512	683	0.010458
1024	1377	0.021494
2048	2763	0.047263
4096	5821	0.104322
8192	12783	0.238142

Tabela 2.1: *Tabela com vetor teste aleatório: A linha te interesse analisada para este caso é a 12.*

n	comparações	tempo(s)
32	45	0.000460
64	83	0.000982
128	169	0.002313
256	351	0.004588
512	679	0.010330
1024	1359	0.023083
2048	2743	0.047348
4096	4855	0.099813
8192	12431	0.220136

Tabela 2.2: *Tabela com vetor teste crescente: A linha te interesse analisada para este caso é a 12.*

Tendo a função $T(n) = 2.214\text{e}{-5} * n - 0.000609$ e para o $n = 2^{32}, T(2^{32}) = 95090.57532444$

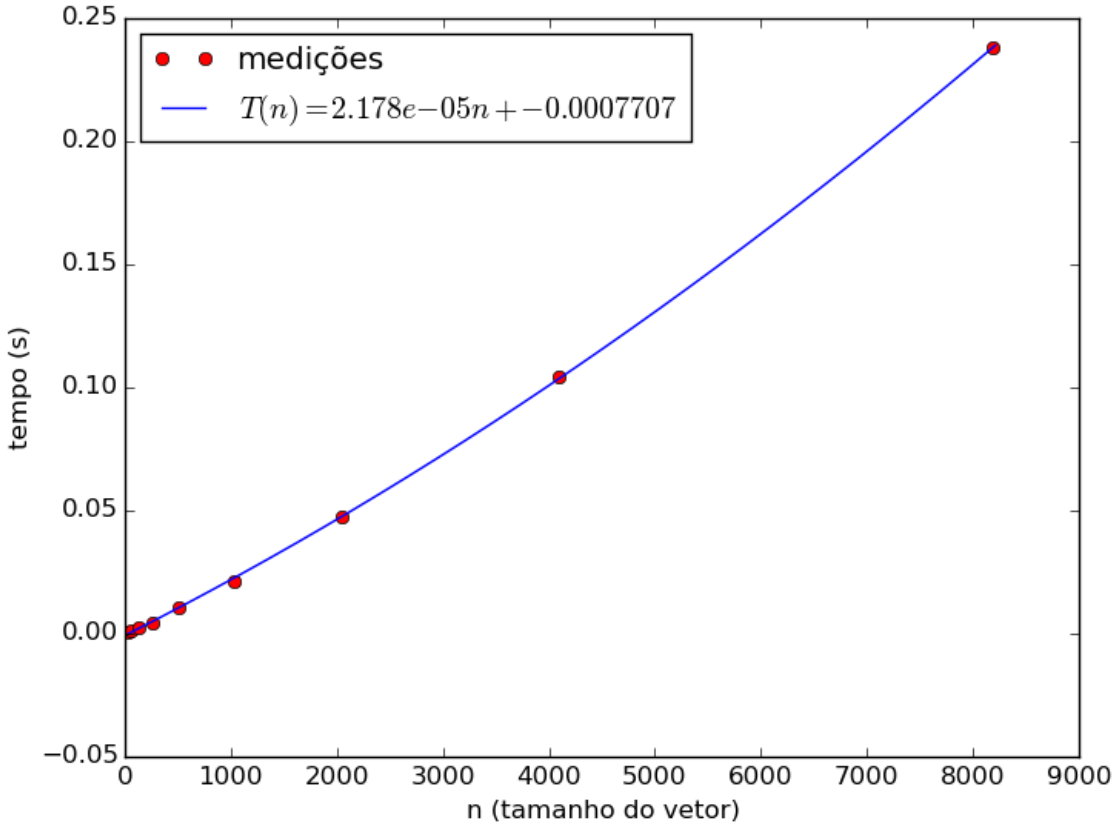

Figura 2.1: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor aleatório.* Tendo a função $T(n) = 2.178\text{e} - 5 * n - 0.0007707$ e para o $n = 2^{32}$, $T(2^{32}) = 93544.3869$

n	comparações	tempo(s)
32	41	0.000455
64	77	0.000949
128	169	0.002131
256	351	0.004956
512	685	0.009586
1024	1363	0.021693
2048	2753	0.045795
4096	4803	0.096253
8192	12439	0.230301

Tabela 2.3: *Tabela com vetor teste decrescente: A linha te interesse analisada para este caso é a 12.*

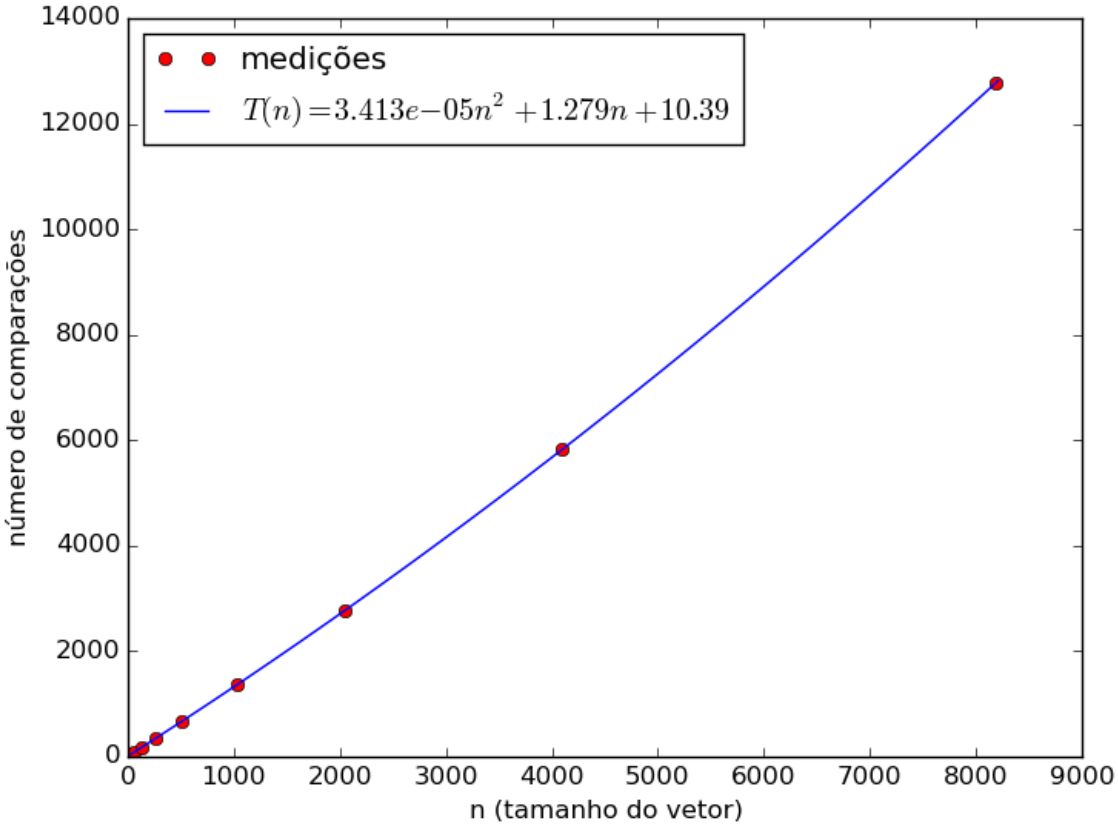

Figura 2.2: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor aleatório1.* Tendo a função $T(n) = 3.413\text{e} - 5 * n^2 + 1.279 * n + 10.39$ e para o $n = 2^{32}$, $T(2^{32}) = 6.1355 * 10^{303}$

n	comparações	tempo(s)
32	41	0.000451
64	83	0.001014
128	169	0.002203
256	341	0.004529
512	691	0.010736
1024	1387	0.022258
2048	2725	0.048822
4096	4773	0.096224
8192	12435	0.236025

Tabela 2.4: *Tabela com vetor teste quase crescente 10%: A linha te interesse analisada para este caso é a 12.*

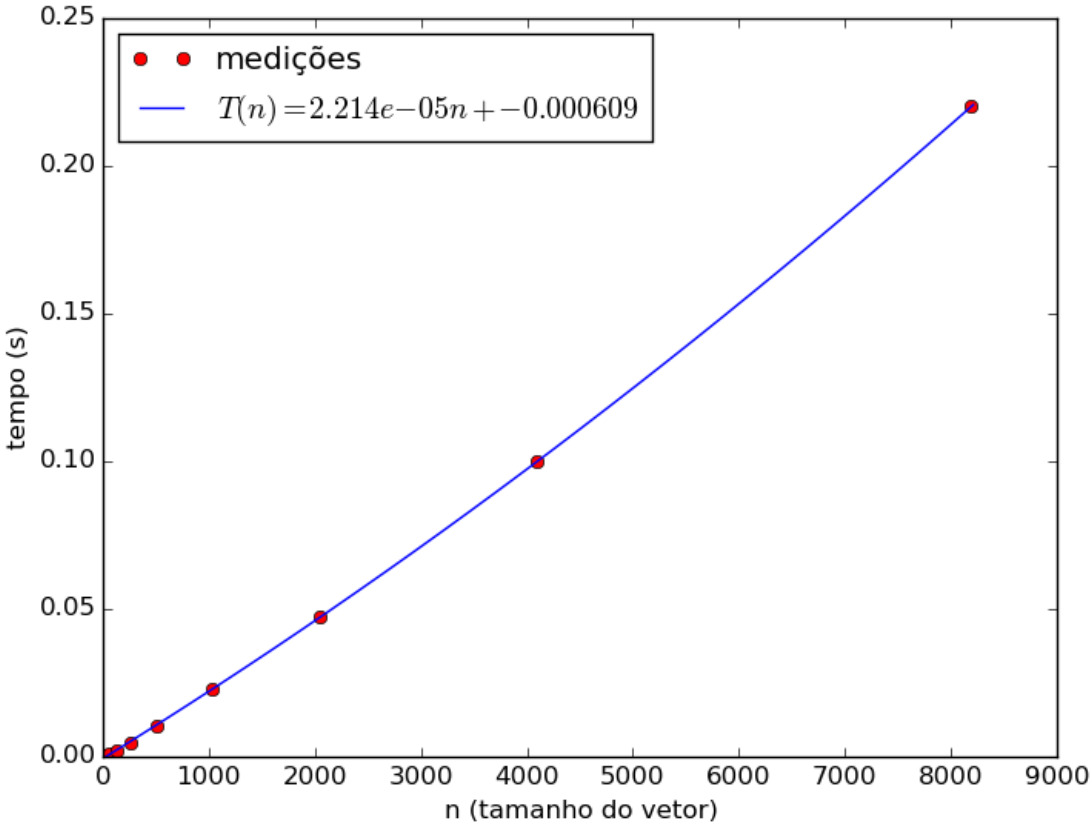

Figura 2.3: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente0*
Tendo a função $T(n) = 2.214\text{e} - 5 * n - 0.000609$ e para o $n = 2^{32}$,
$T(2^{32}) = 95090.57532444$

n	comparações	tempo(s)
32	41	0.000460
64	87	0.000899
128	165	0.002290
256	343	0.004409
512	691	0.010402
1024	1367	0.022453
2048	2735	0.048965
4096	4825	0.102515
8192	12433	0.236374

Tabela 2.5: *Tabela com vetor teste quase crescente 20%: A linha te interesse analisada para este caso é a 12.*

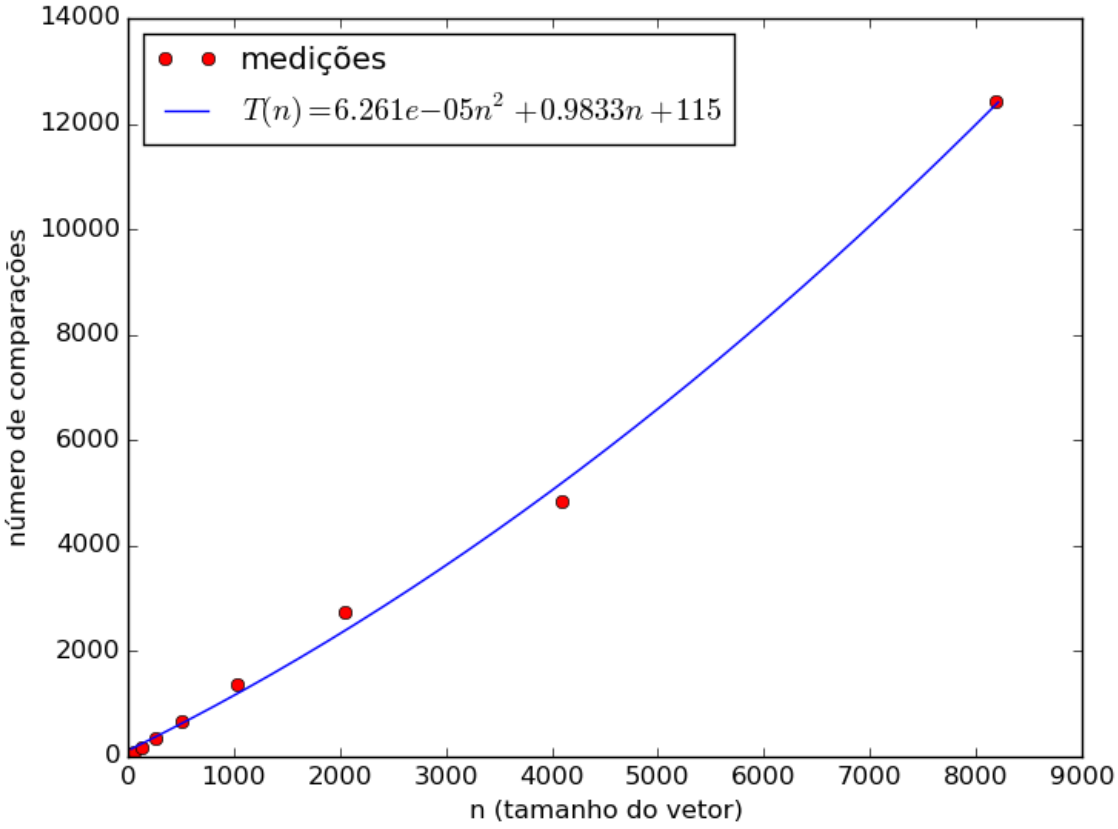

Figura 2.4: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente1*
Tendo a função $T(n) = 6.261\text{e} - 5 * n^2 + 0.9833 * n + 155$ e para o $n = 2^{32}$,
$T(2^{32}) = 1.1255 * 10^{304}$

2.1

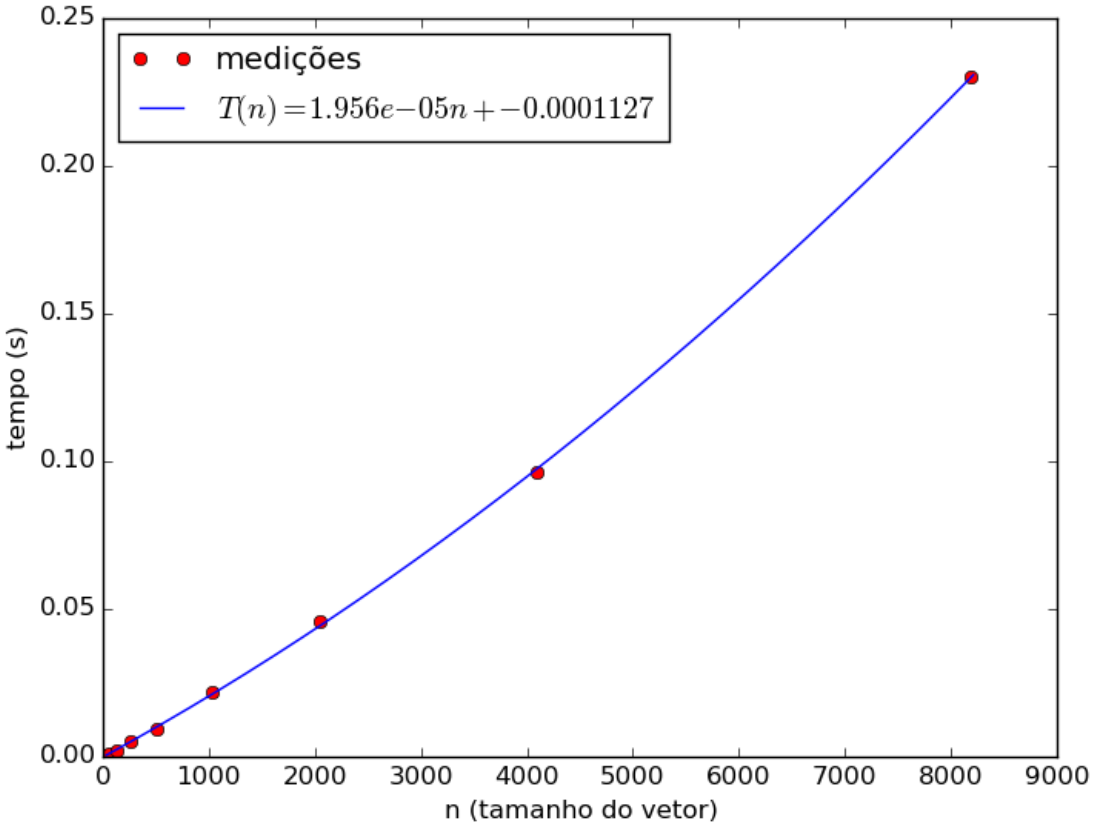

Figura 2.5: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente0*
Tendo a função $T(n) = 1.956\text{e} - 5 * n - 0.00001127$ e para o $n = 2^{32}$,
$T(2^{32}) = 84009.56029849$

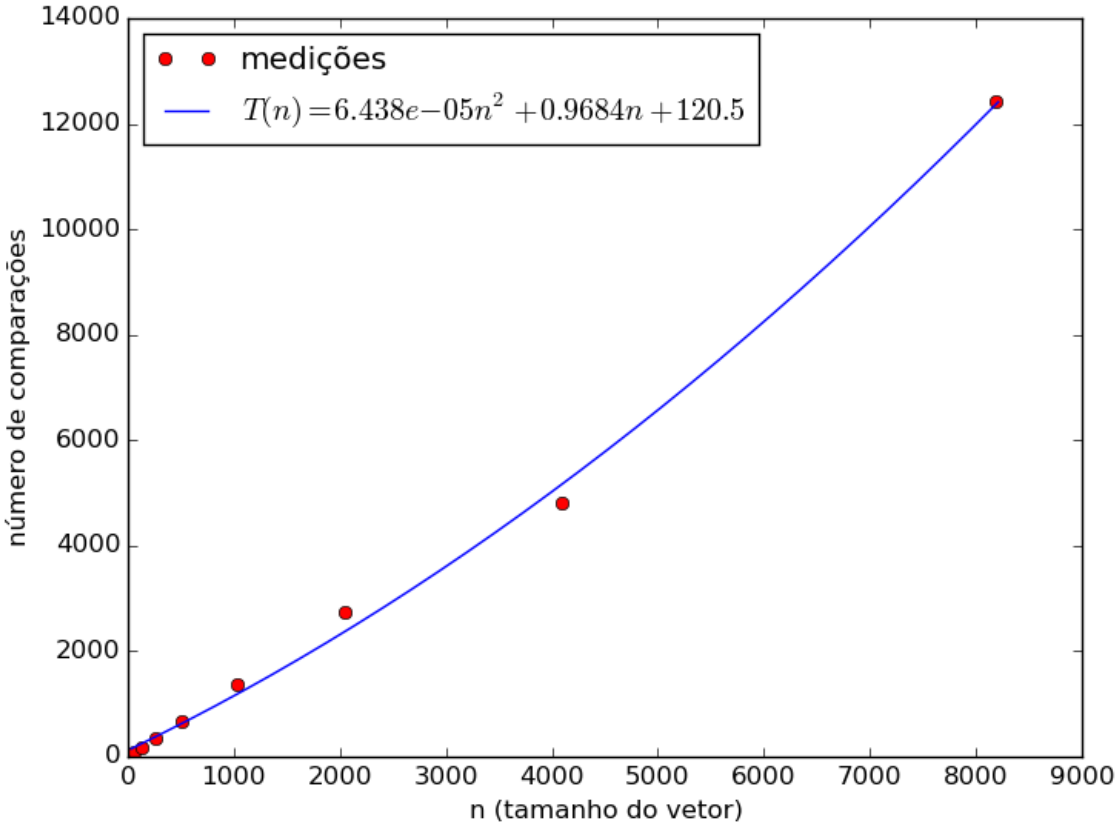

Figura 2.6: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente1*
Tendo a função $T(n) = 6.438\text{e} - 5 * n^2 + 0.9684 * n + 120.5$ e para o $n = 2^{32}$,
$T(2^{32}) = 1.15735 * 10^{304}$

2.1

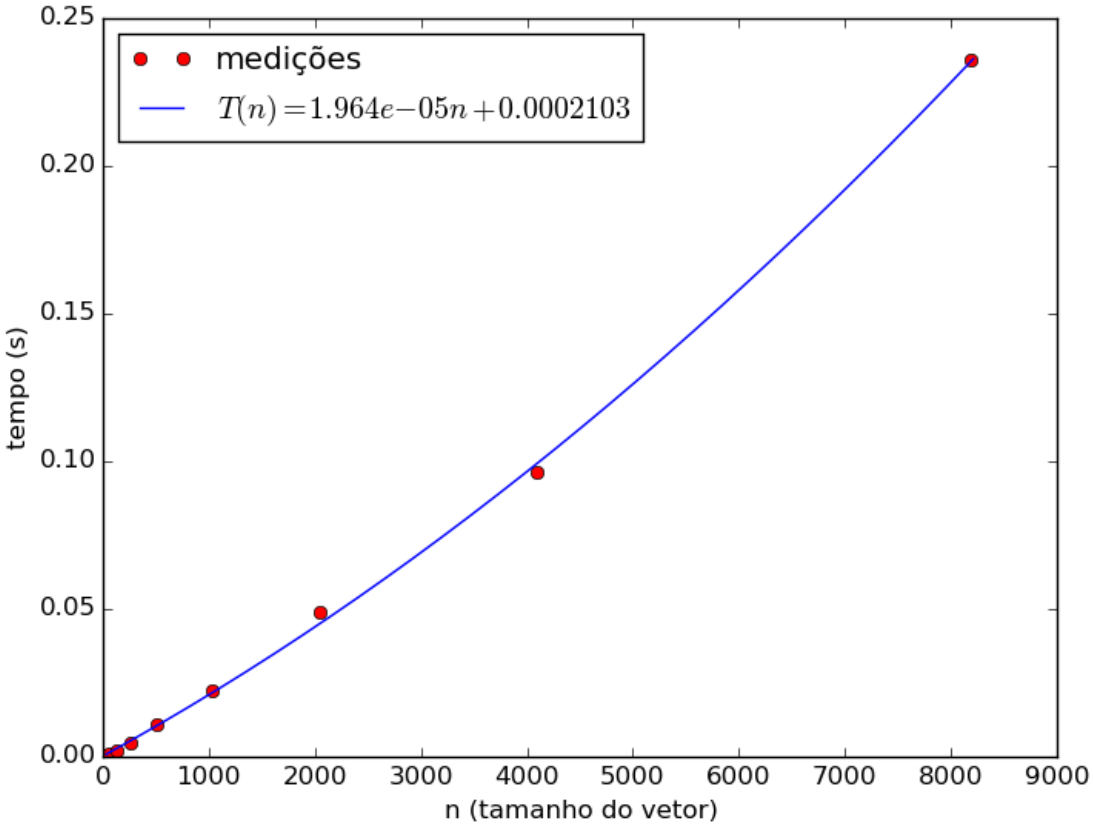

Figura 2.7: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente100*
Tendo a função $T(n) = 1.964\text{e} - 5 * n + 0.0002103$ e para o $n = 2^{32}$,
$T(2^{32}) = 84353.15790374$

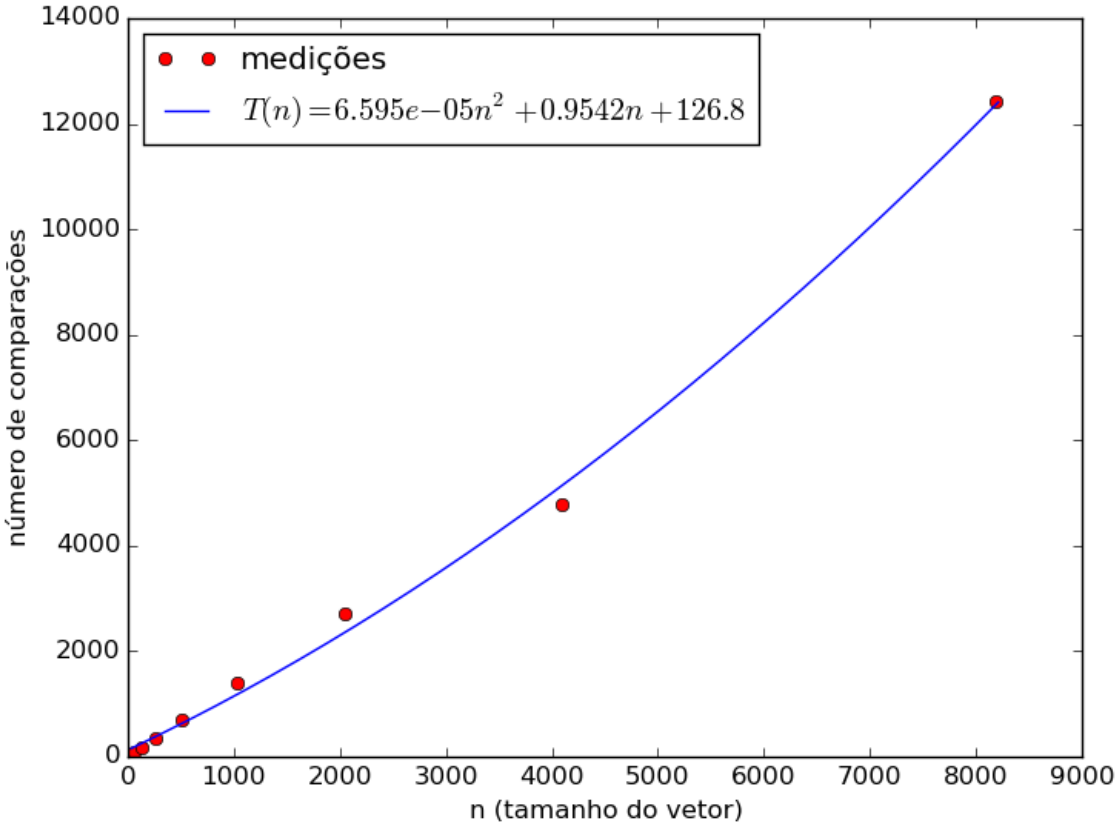

Figura 2.8: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente101*
Tendo a função $T(n) = 6.595\text{e} - 5 * n^2 + 0.9542 * n + 126.8$ e para o $n = 2^{32}$,
$T(2^{32}) = 1.1855 * 10^{304}$

2.1

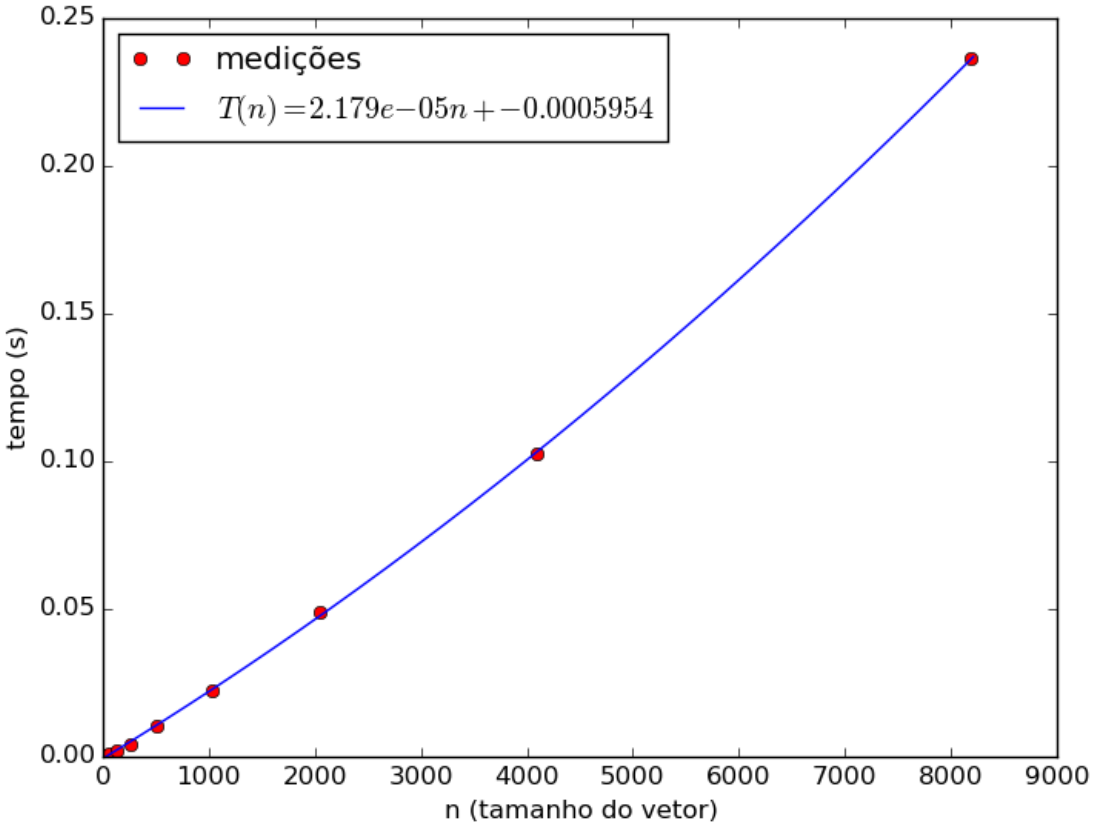

Figura 2.9: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente200*
Tendo a função $T(n) = 2.179\text{e} - 5 * n - 0.0005954$ e para o $n = 2^{32}$,
$T(2^{32}) = 93587.33678444$

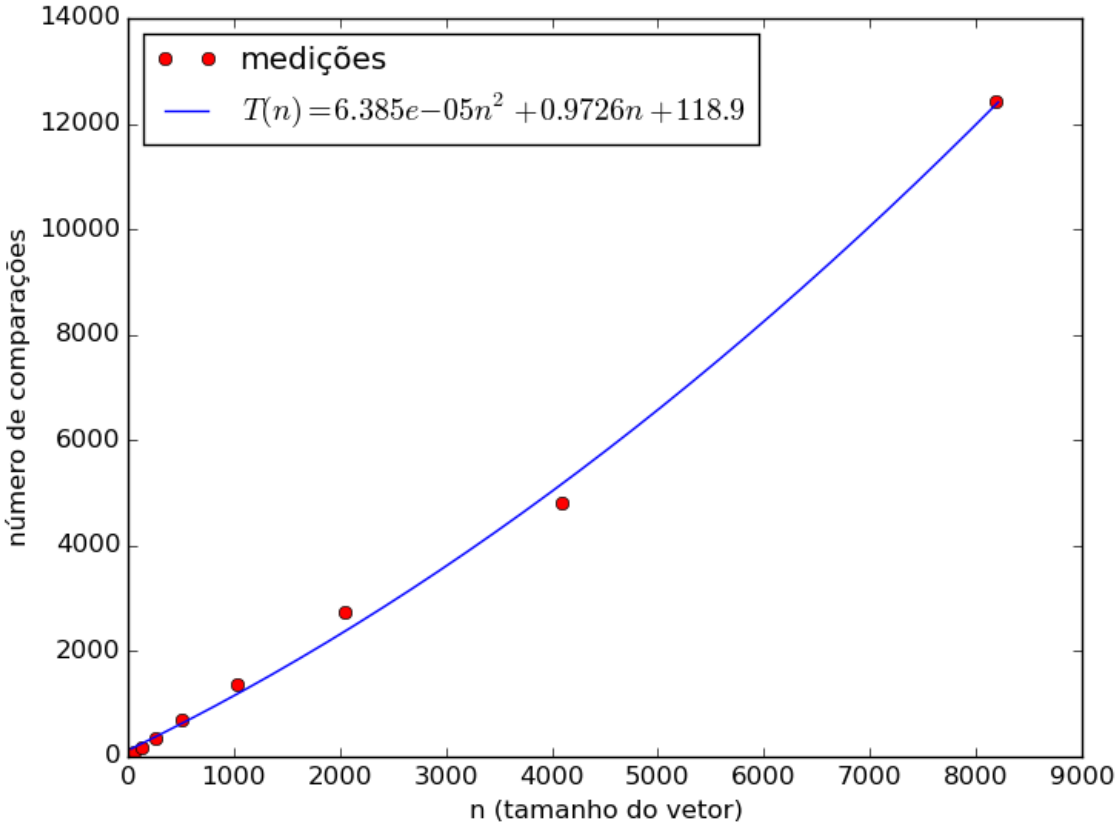

Figura 2.10: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente201*
Tendo a função $T(n) = 6.385\text{e} - 5 * n^2 + 0.9726 * n + 111.9$ e para o $n = 2^{32}$,
$T(2^{32}) = 1.14782 * 10^{304}$

n	comparações	tempo(s)
32	41	0.000418
64	81	0.000956
128	169	0.002140
256	349	0.004458
512	687	0.009880
1024	1371	0.020995
2048	2737	0.051357
4096	4875	0.101185
8192	12439	0.238561

Tabela 2.6: *Tabela com vetor teste quase crescente 30%: A linha te interesse analisada para este caso é a 12.*

n	comparações	tempo(s)
32	45	0.000475
64	91	0.001001
128	175	0.002174
256	341	0.004848
512	679	0.010063
1024	1391	0.021854
2048	2737	0.046297
4096	4797	0.097154
8192	12429	0.247491

Tabela 2.7: *Tabela com vetor teste quase crescente 40%: A linha te interesse analisada para este caso é a 12.*

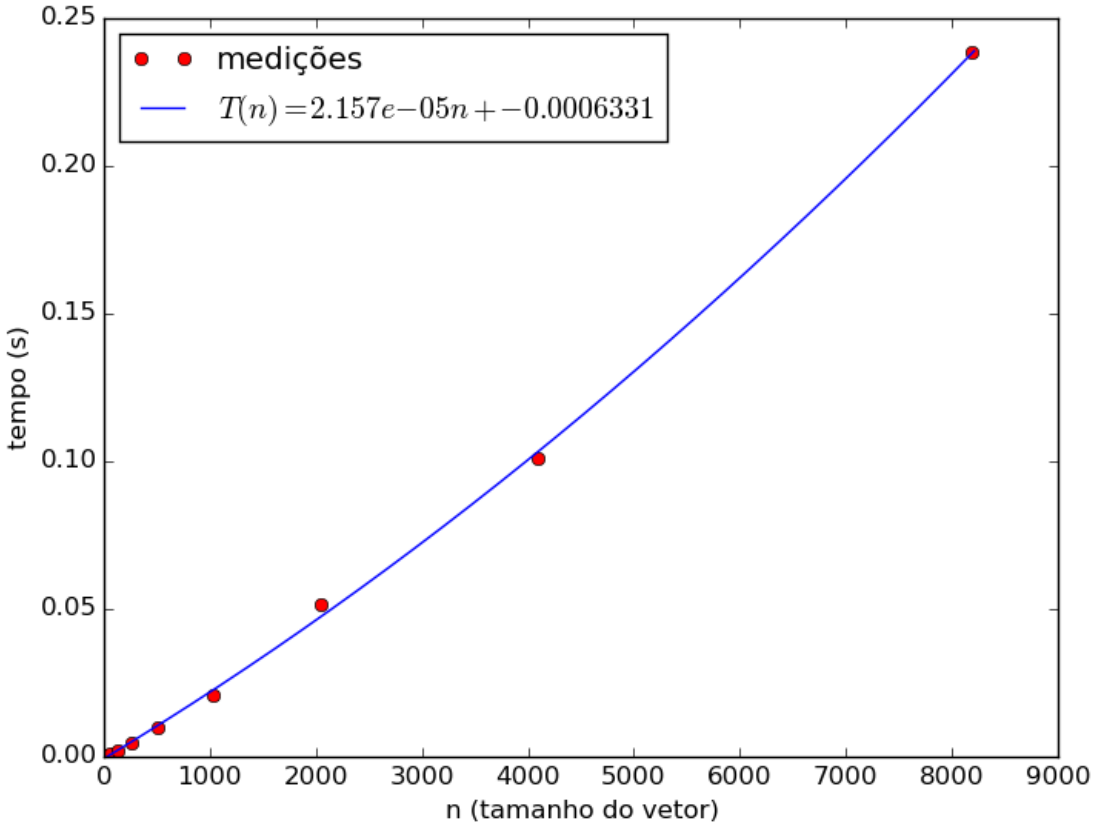

Figura 2.11: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente300*
Tendo a função $T(n) = 2.157\text{e} - 5 * n - 0.0006331$ e para o $n = 2^{32}$,
$T(2^{32}) = 92642.44394162$

n	comparações	tempo(s)
32	47	0.000495
64	89	0.001033
128	175	0.002108
256	357	0.004774
512	671	0.009816
1024	1357	0.023627
2048	2743	0.046188
4096	4851	0.098774
8192	12439	0.234423

Tabela 2.8: *Tabela com vetor teste quase crescente 50%: A linha te interesse analisada para este caso é a 12.*

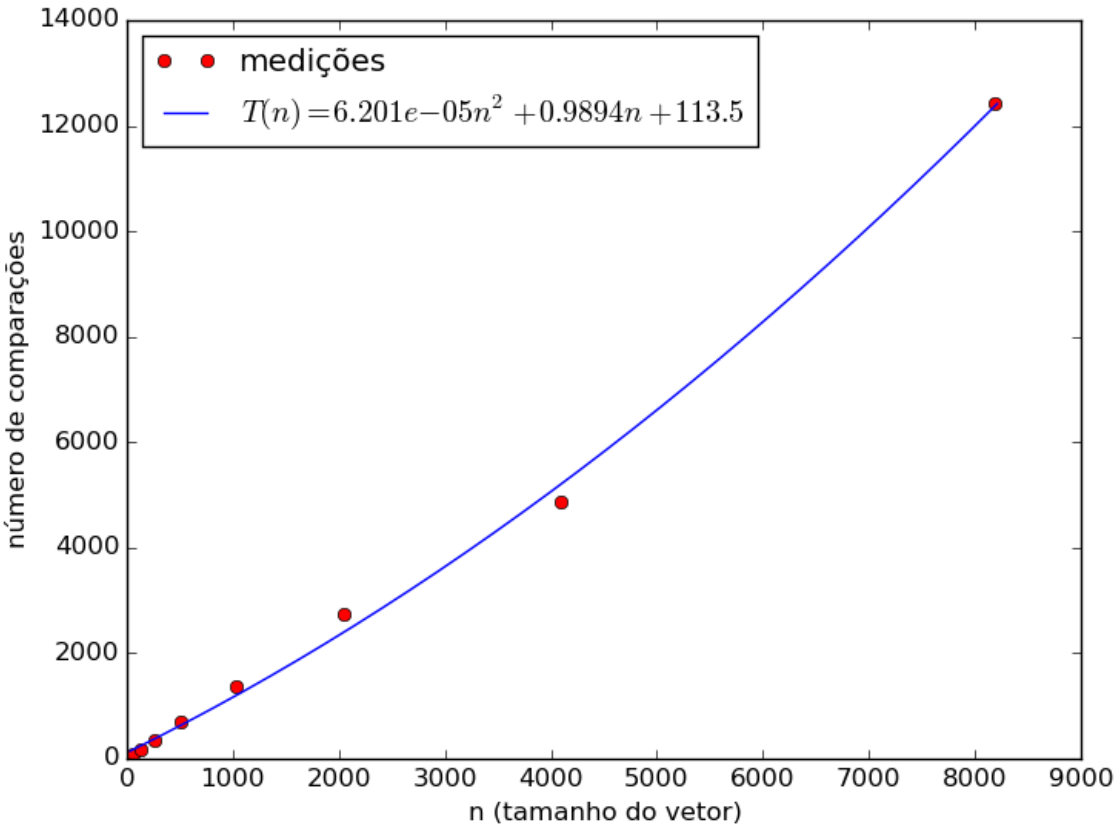

Figura 2.12: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente301*
Tendo a função $T(n) = 6.201\text{e} - 5 * n^2 + 0.9894 * n + 113.5$ e para o $n = 2^{32}$,
$T(2^{32}) = 1.1438 \times 10^{15}$

n	comparações	tempo(s)
32	43	0.000497
64	87	0.000969
128	169	0.002212
256	343	0.005239
512	687	0.009943
1024	1387	0.023199
2048	2761	0.048857
4096	4829	0.100816
8192	12431	0.230520

Tabela 2.9: *Tabela com vetor teste quase decrescente 10%: A linha te interesse analisada para este caso é a 12.*

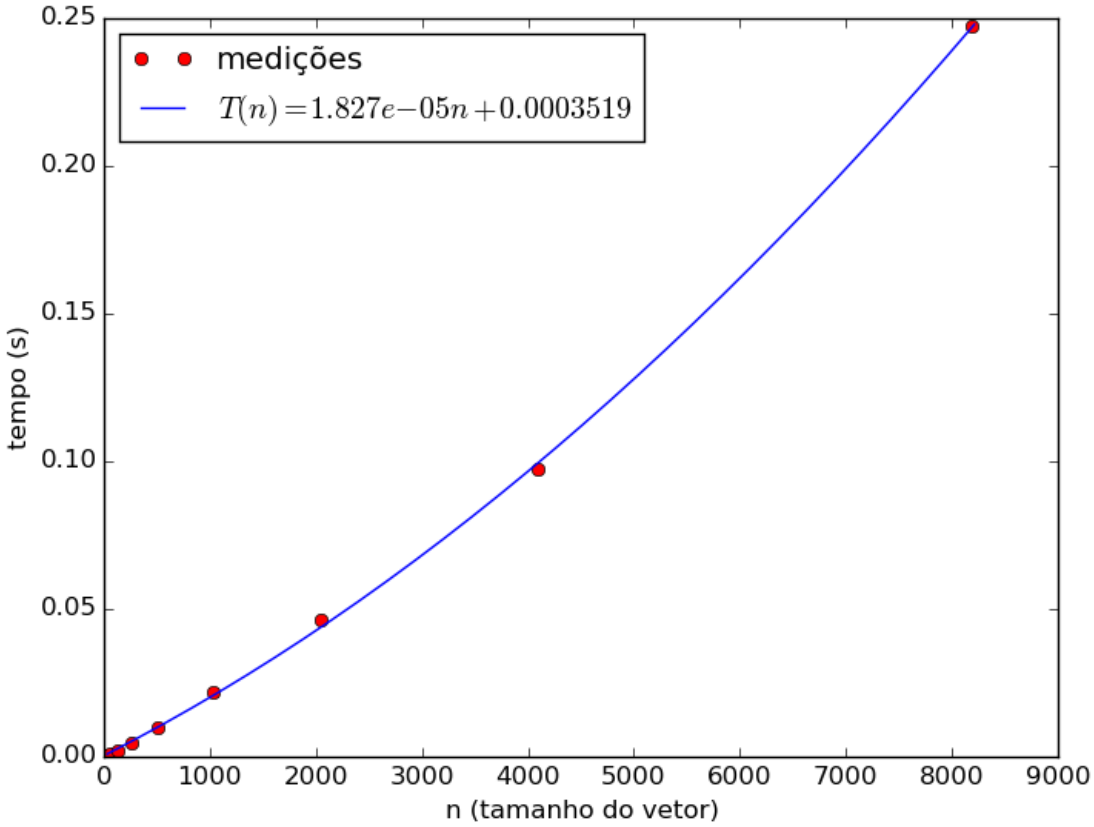

Figura 2.13: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente400*
Tendo a função $T(n) = 1.827\text{e} - 5 * n + 0.0003519$ e para o $n = 2^{32}$,
$T(2^{32}) = 78469.05284982$

n	comparações	tempo(s)
32	45	0.000498
64	87	0.001026
128	165	0.002288
256	341	0.004596
512	687	0.010886
1024	1339	0.025004
2048	2713	0.050401
4096	4825	0.095543
8192	12433	0.237404

Tabela 2.10: *Tabela com vetor teste quase decrescente 20%: A linha te interesse analisada para este caso é a 12.*

2.1

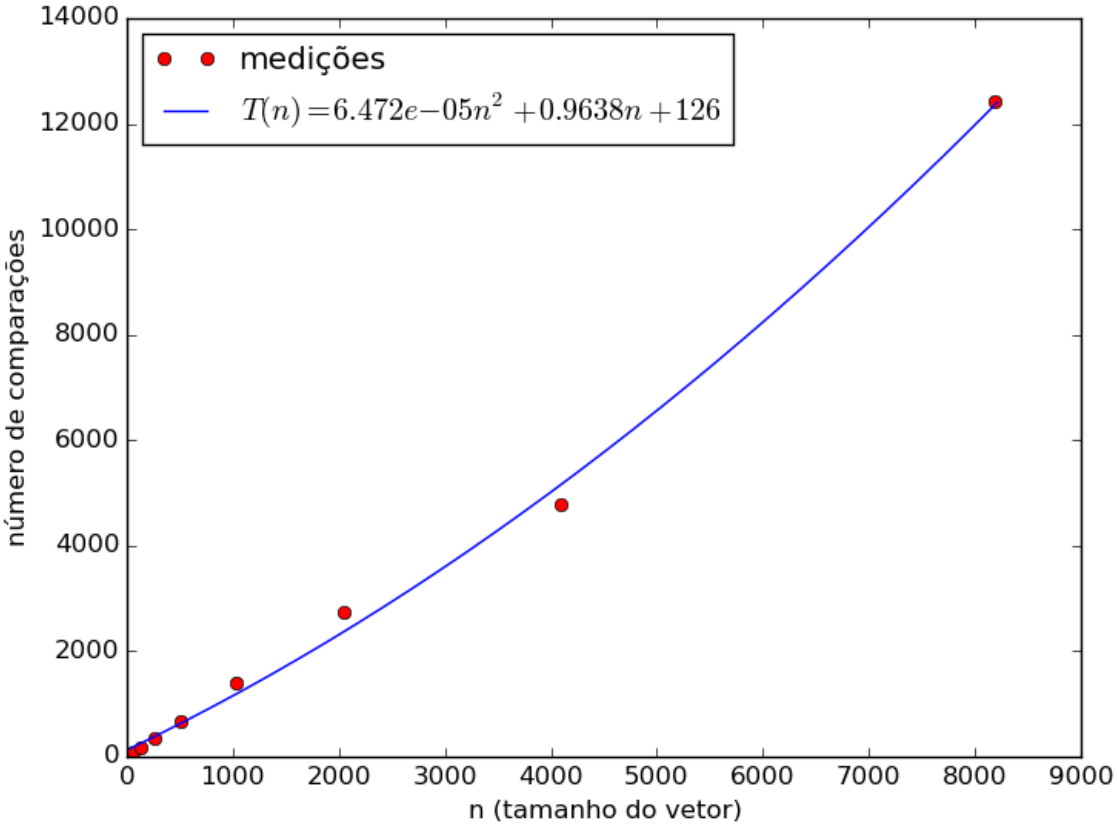

Figura 2.14: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente401*
Tendo a função $T(n) = 6.472\text{e} - 5 * n^2 + 0.9638 * n + 126$ e para o $n = 2^{32}$,
$T(2^{32}) = 1.1634 \times 10^{304}$

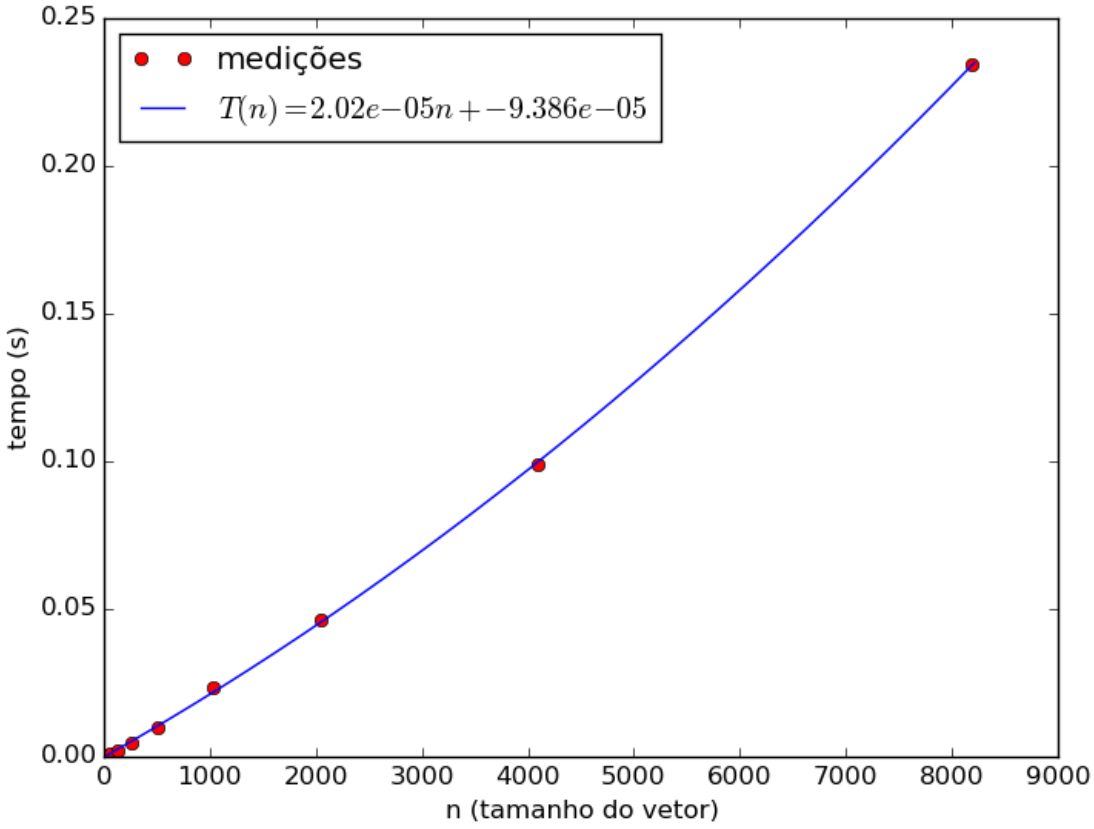

Figura 2.15: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente500*
Tendo a função $T(n) = 2.02\text{e} - 5 * n - 9.386\text{e} - 05$ e para o $n = 2^{32}$,
$T(2^{32}) = 86758.33928534$

2.1

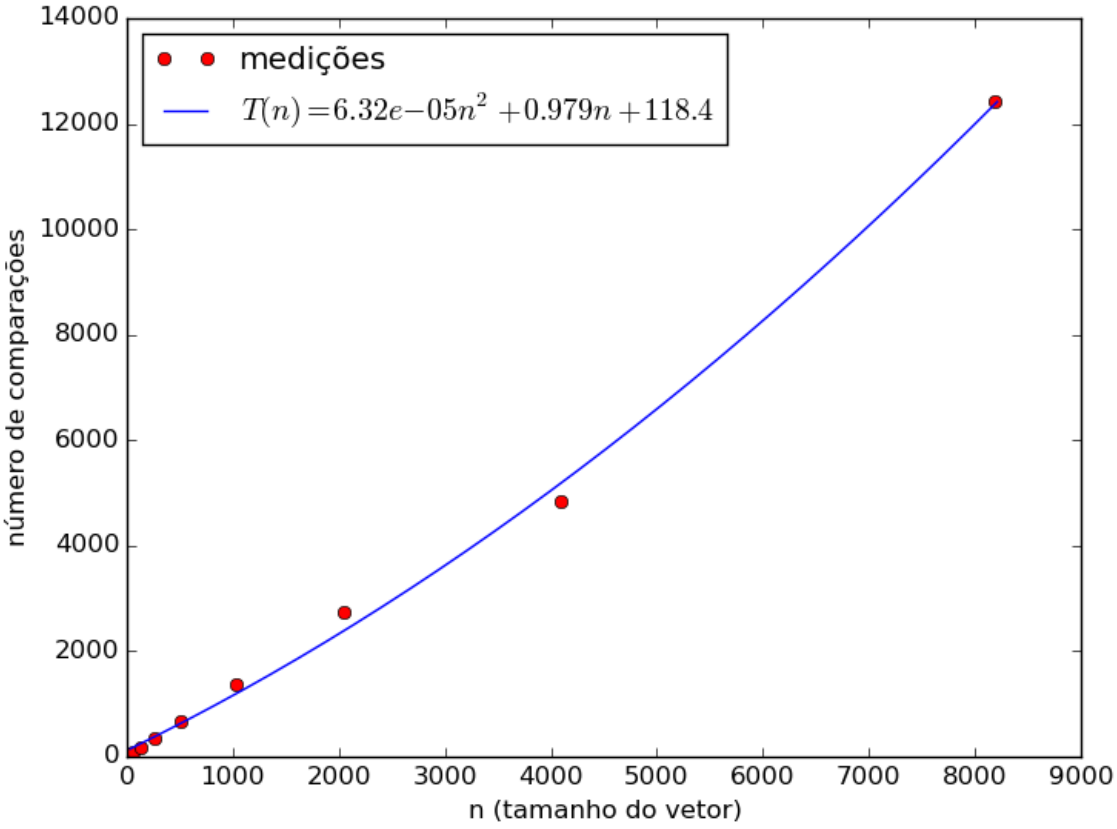

Figura 2.16: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor crescente501*
Tendo a função $T(n) = 6.32\text{e} - 5 * n^2 + 0.979 * n + 118.4$ e para o $n = 2^{32}$,
$T(2^{32}) = 1.13614 * 10^{304}$

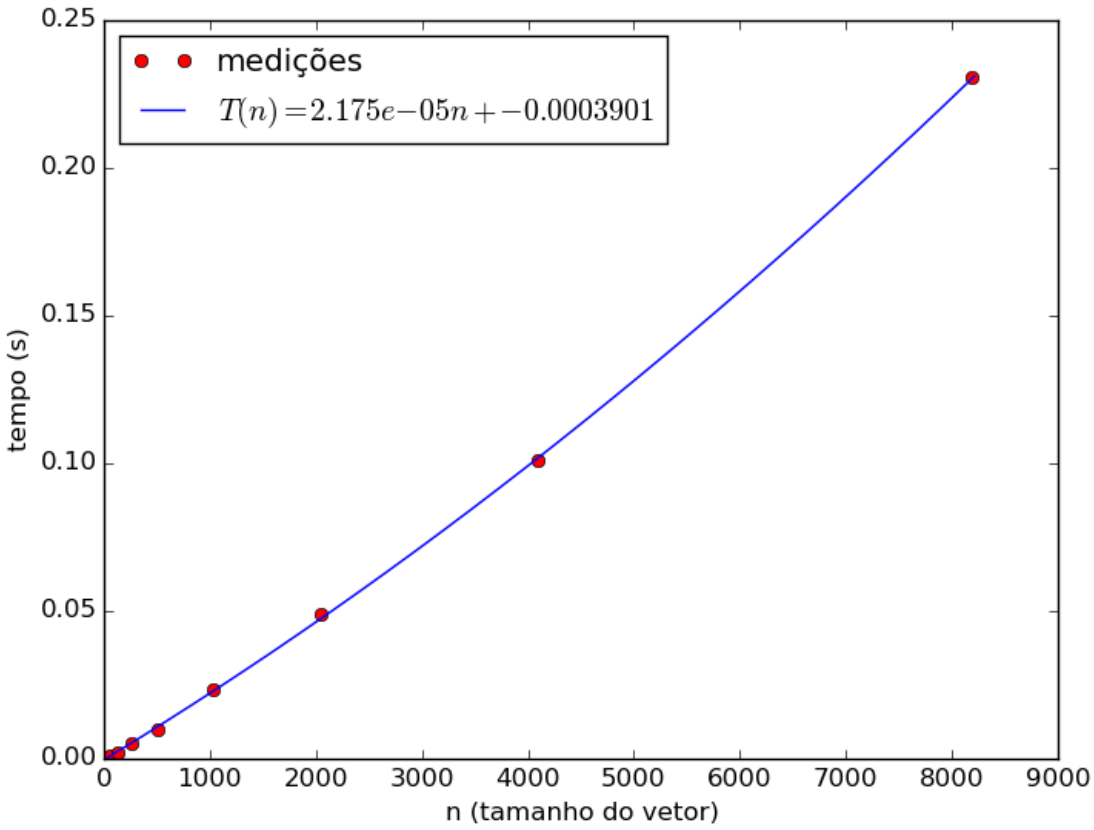

Figura 2.17: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente100*
Tendo a função $T(n) = 2.175\text{e} - 5 * n - 0.0003901$ e para o $n = 2^{32}$, $T(2^{32}) = 93415.5382$

2.1

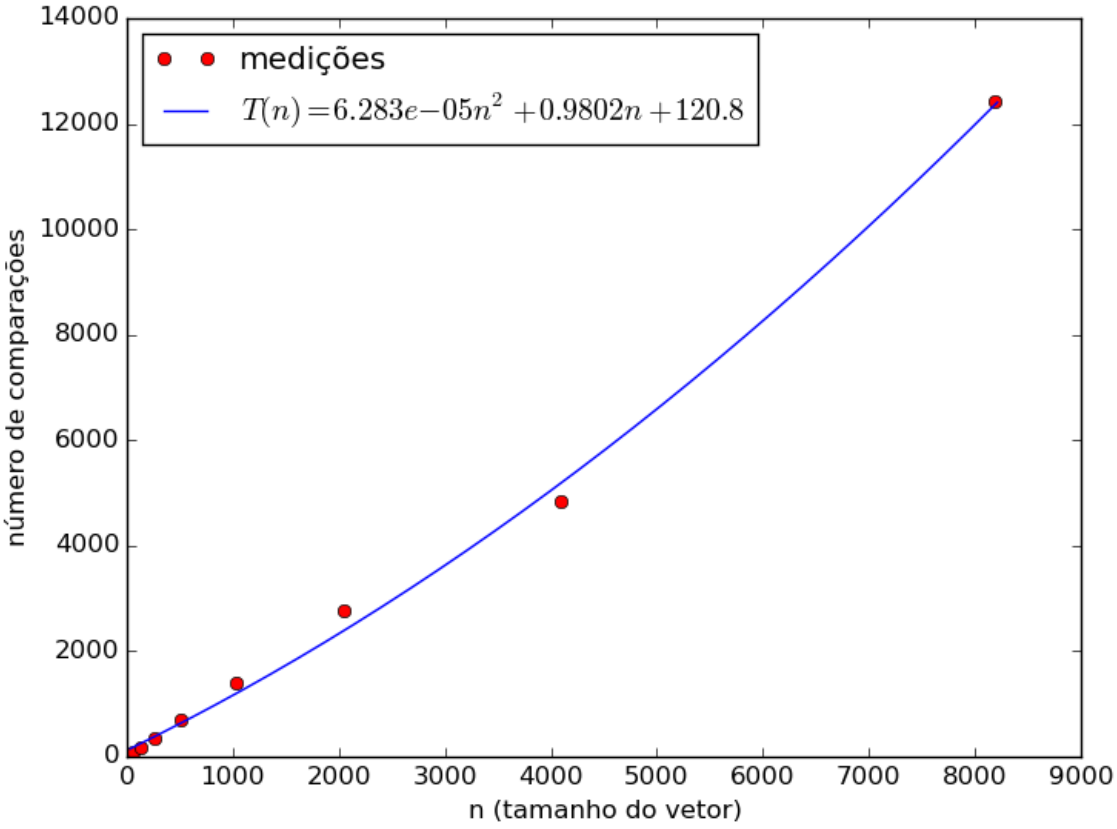

Figura 2.18: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente101*
Tendo a função $T(n) = 6.283\text{e} - 5 * n^2 + 0.9802 * n + 120.8$ e para o $n = 2^{32}$,
$T(2^{32}) = 4.75507 * 10^{313}$

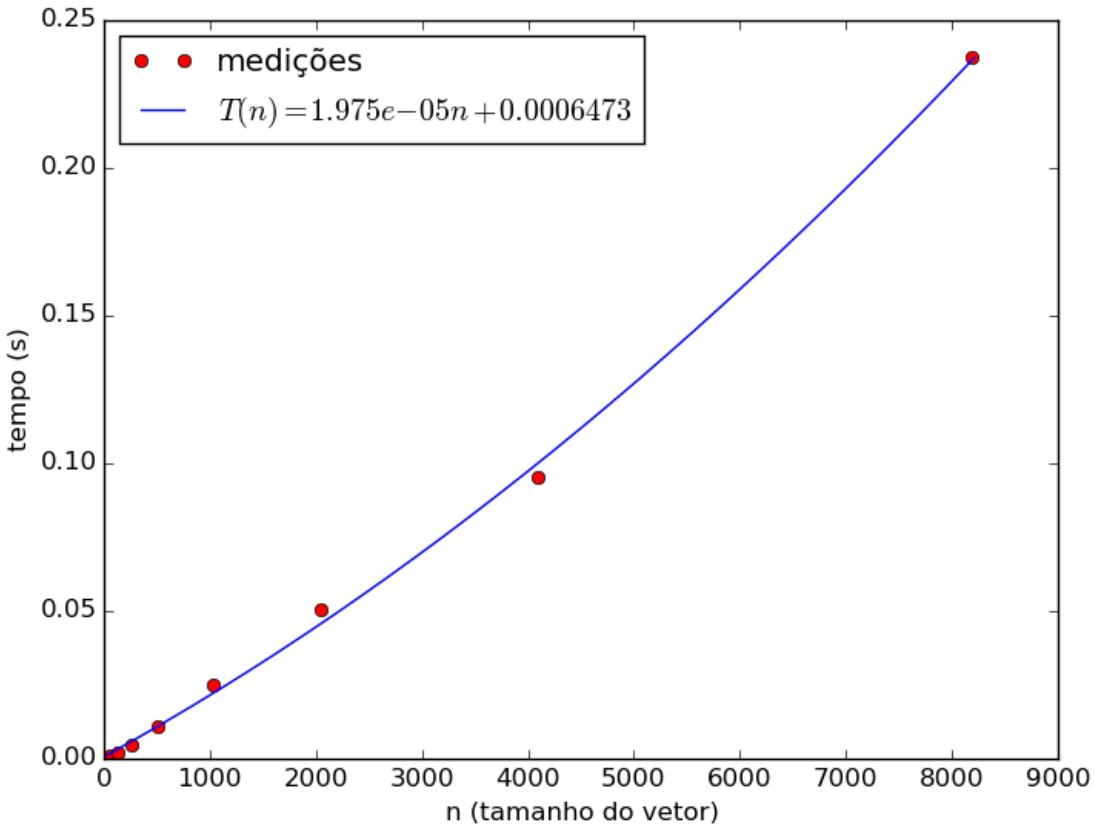

Figura 2.19: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente101*
Tendo a função $T(n) = 1.975\text{e} - 5 * n + 0.0006473$ e para o $n = 2^{32}$, $T(2^{32}) = 84825.6047433$

2.1

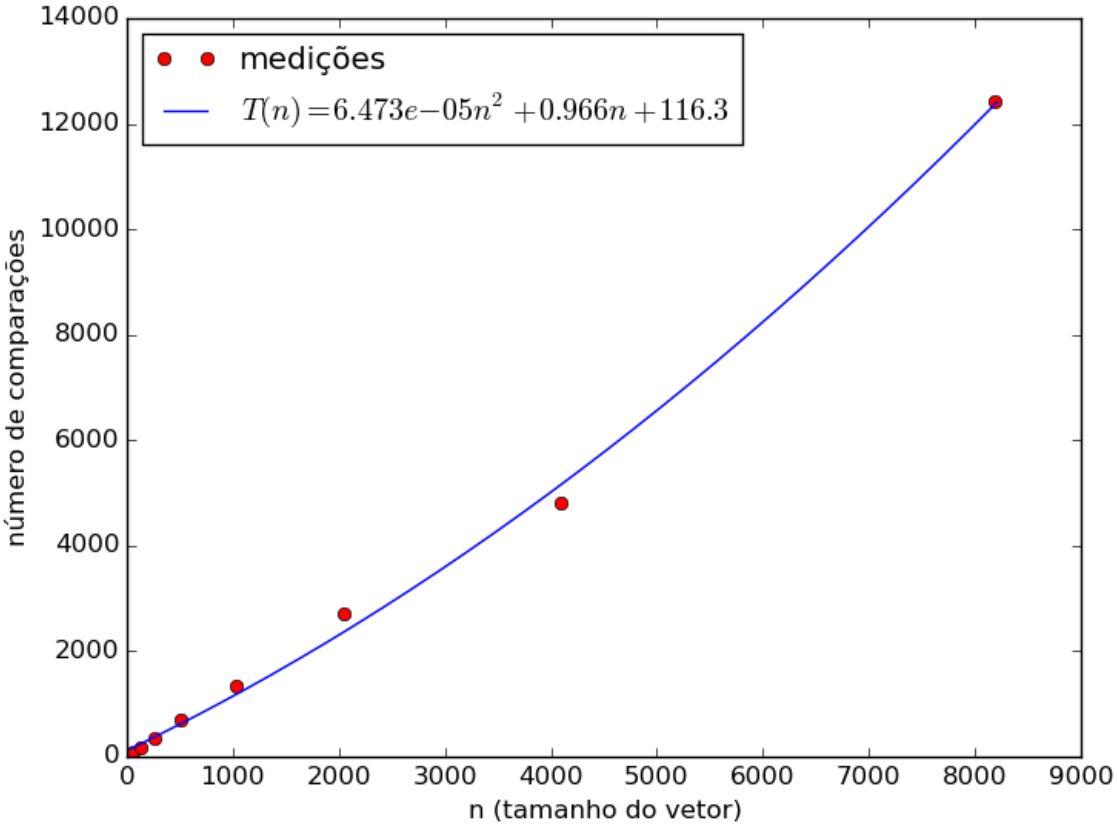

Figura 2.20: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente201*
Tendo a função $T(n) = 6.473\text{e} - 5 * n^2 + 0.966 * n + 116.3$ e para o $n = 2^{32}$,
$T(2^{32}) = 84825.6047433$

n	comparações	tempo(s)
32	45	0.000490
64	81	0.000922
128	179	0.002251
256	347	0.004792
512	695	0.009644
1024	1361	0.022657
2048	2735	0.047662
4096	4841	0.095222
8192	12441	0.236857

Tabela 2.11: *Tabela com vetor teste quase decrescente 30%: A linha te interesse analisada para este caso é a 12.*

n	comparações	tempo(s)
32	41	0.000448
64	89	0.000929
128	171	0.002126
256	331	0.004763
512	689	0.010562
1024	1367	0.023329
2048	2697	0.047045
4096	4859	0.094538
8192	12435	0.232884

Tabela 2.12: *Tabela com vetor teste quase decrescente 40%: A linha te interesse analisada para este caso é a 12.*

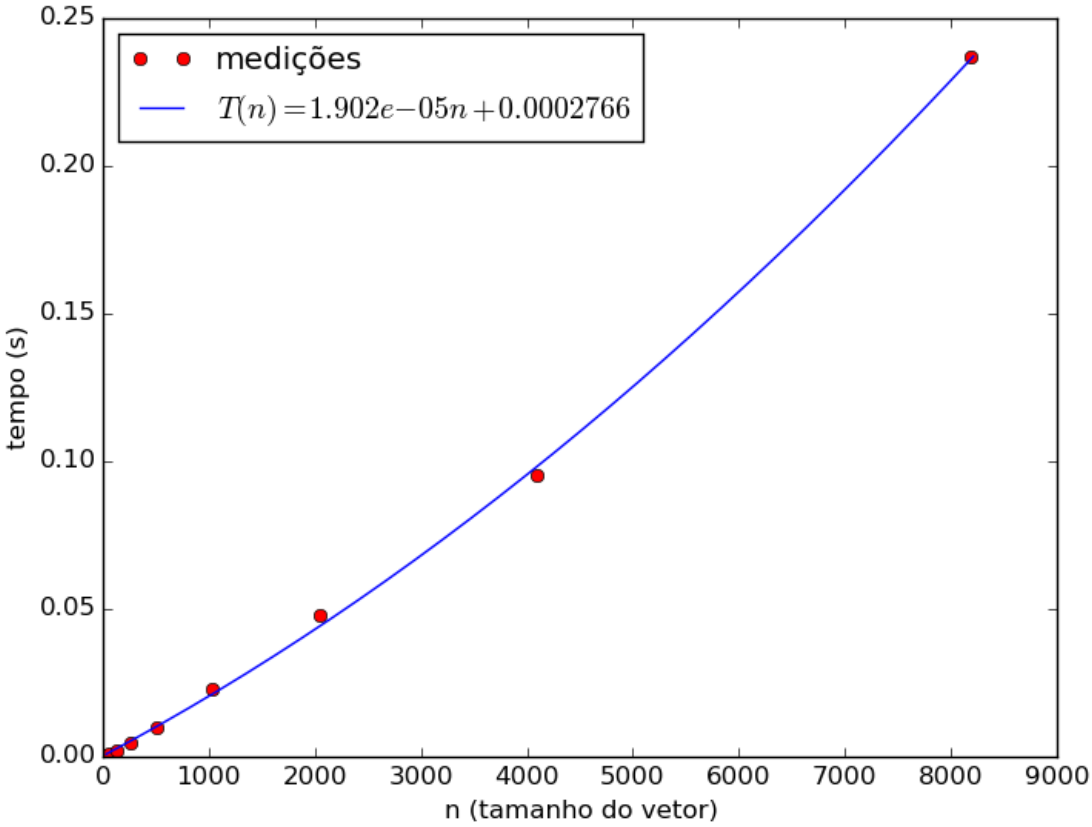

Figura 2.21: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente300*
Tendo a função $T(n) = 1.902\text{e} - 5 * n + 0.0002766$ e para o $n = 2^{32}$,
$T(2^{32}) = 81690.27824652$

n	comparações	tempo(s)
32	47	0.000458
64	87	0.001033
128	177	0.002219
256	337	0.004370
512	675	0.010310
1024	1363	0.022362
2048	2737	0.048213
4096	4839	0.101969
8192	12435	0.230613

Tabela 2.13: *Tabela com vetor teste quase decrescente 50%: A linha te interesse analisada para este caso é a 12.*

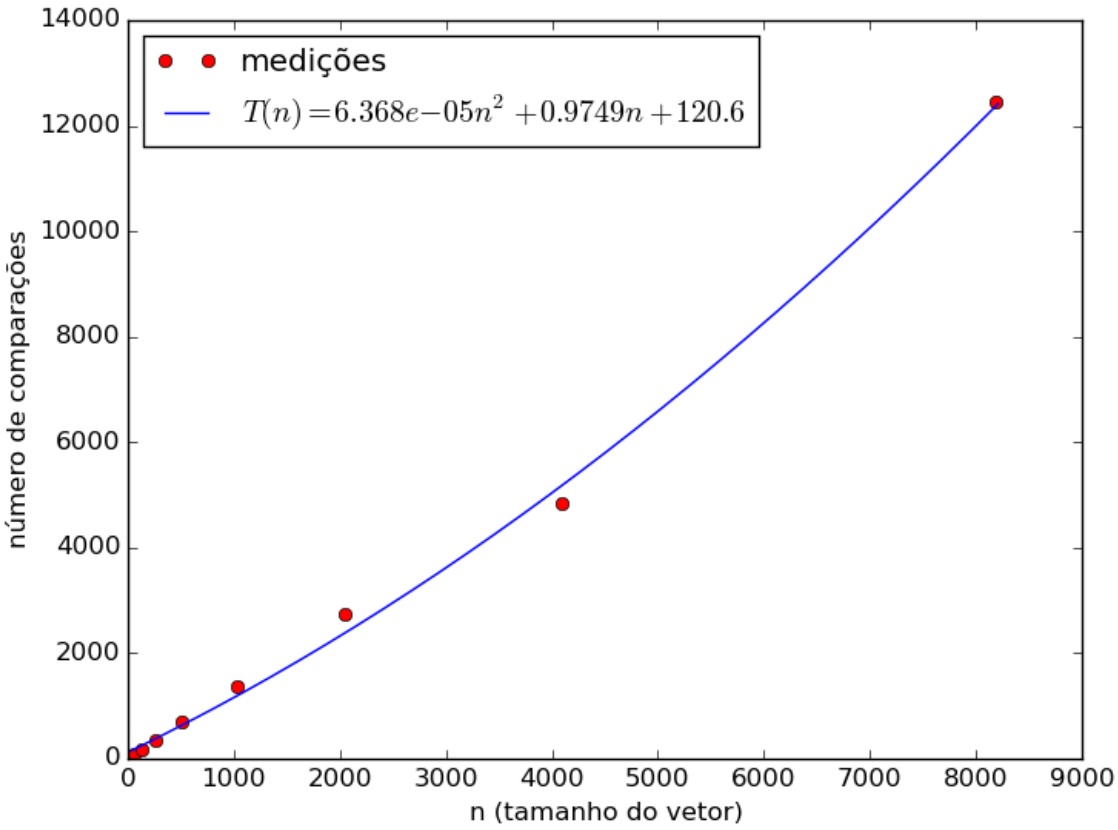

Figura 2.22: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente301*
Tendo a função $T(n) = 6.368\text{e} - 5 * n + 120.6$ e para o $n = 2^{32}$, $T(2^{32}) = 273624.11740928$

2.1

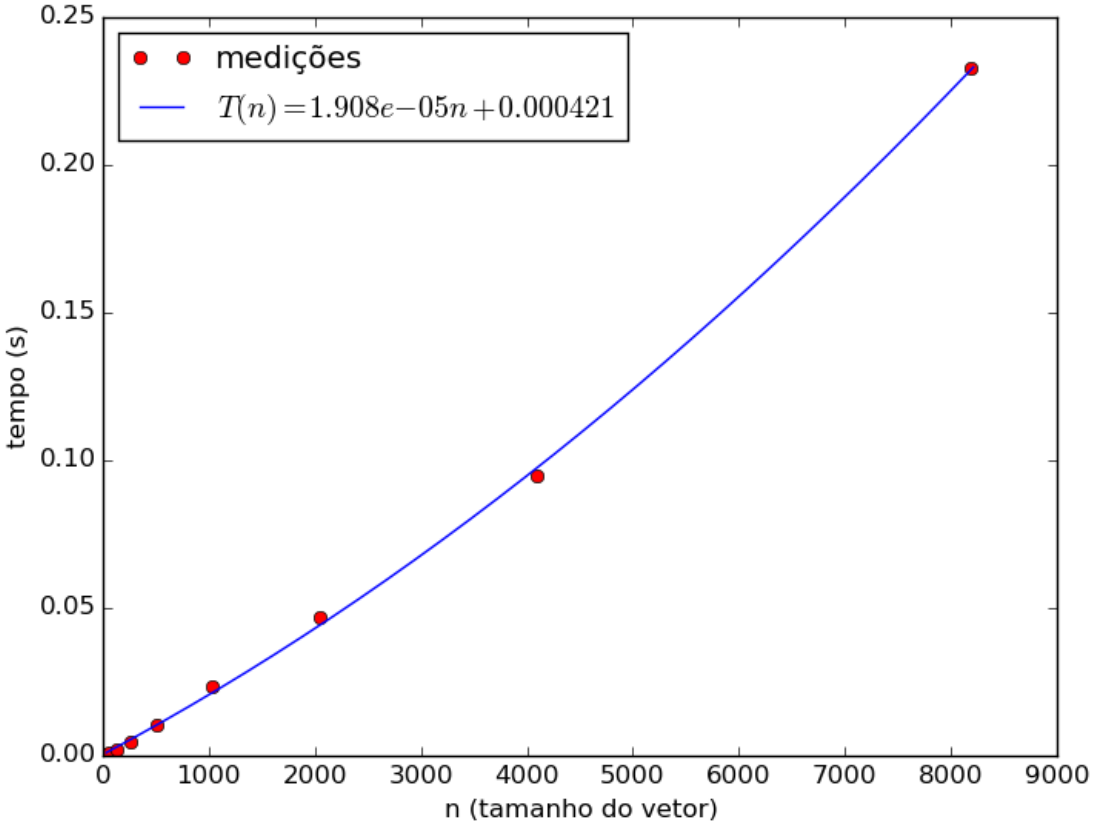

Figura 2.23: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente400*
Tendo a função $T(n) = 1.908\text{e} - 5 * n + 0.000421$ e para o $n = 2^{32}$, $T(2^{32}) = 81947.97642868$

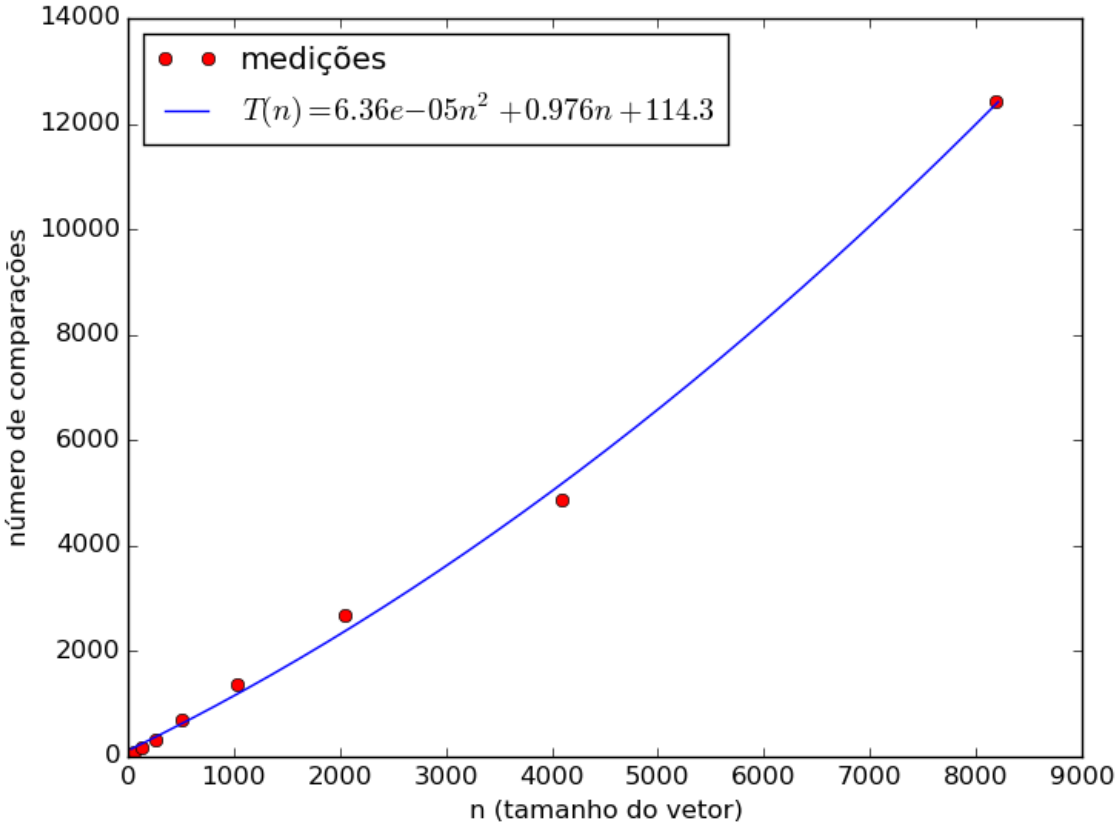

Figura 2.24: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente401*
Tendo a função $T(n) = 6.36\text{e} - 5 * n + 0.976 * n + 114.3$ e para o $n = 2^{32}$,
$T(2^{32}) = 4.1921 * 10^9$

2.1

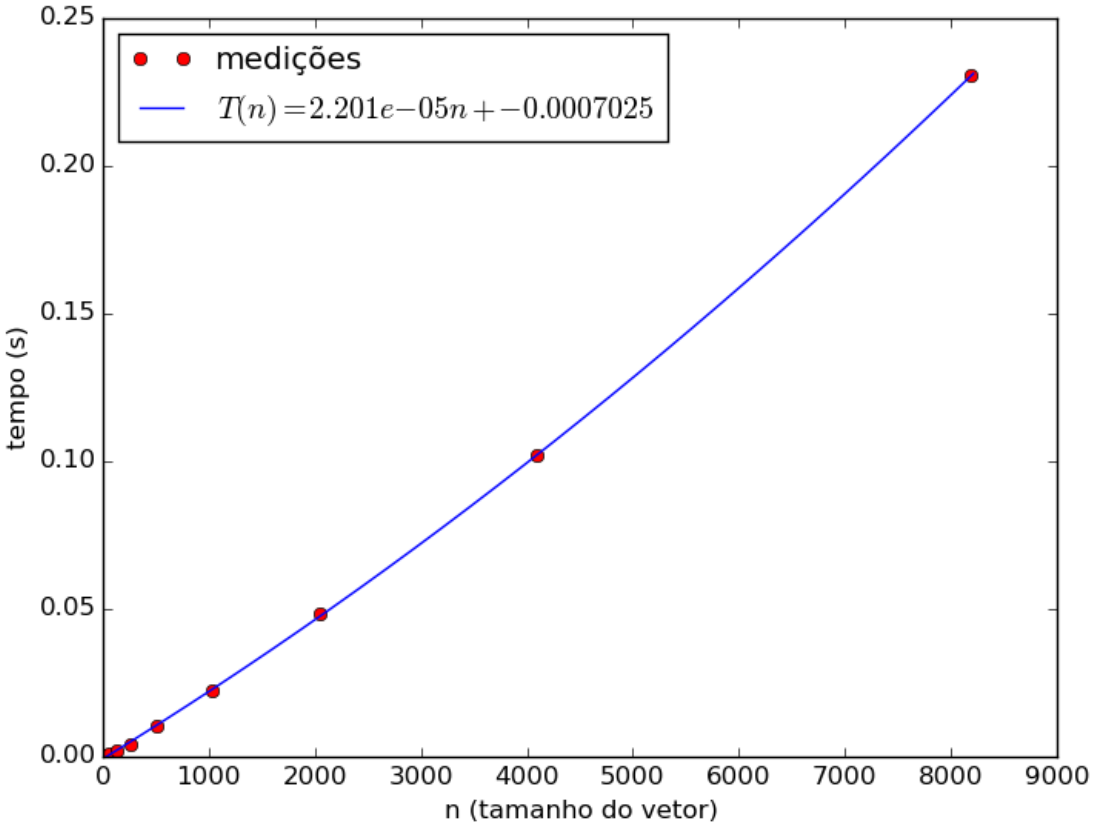

Figura 2.25: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente500*
Tendo a função $T(n) = 2.201\text{e} - 5 * n - 0.0007025$ e para o $n = 2^{32}$,
$T(2^{32}) = 94532.22948246$

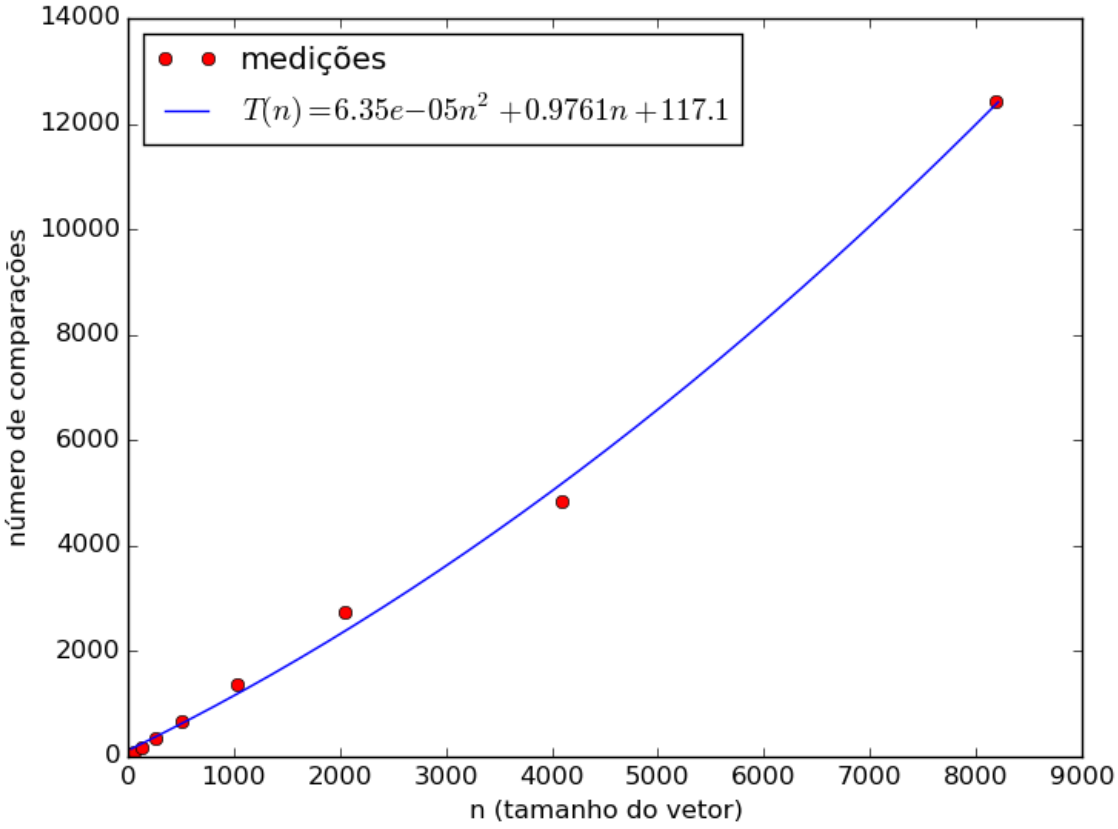

Figura 2.26: *A análise do grafico para* 2^{32} *segue abaixo para quicksort de vetor decrescente501*
Tendo a função $T(n) = 6.35\text{e} - 5 * n^2 + 0.9761 * n + 117.1$ e para o $n = 2^{32}$,
$T(2^{32}) = 1.141535 * 10^{304}$

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)
```