# Análise experimental de algoritmos usando Python

Patricia 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 Introdução

O mergesort, é um algoritmo de ordenação que pode ser obtido especializando a formulação recursiva da divisão e onquista binária, onde divide todo o vetor usando recursividade até chegar ao caso base. após chegar ao caso base ele vem desempilhando comparando menores massas de dados.
Ideia básica:
Dividir: dividir a lista em duas listas com cerca da metade do tamanho.
Conquistar: dividir cada uma das d uas sublistas recursivamente até que tenham tamanho um.
Combinar: fundir as duas sublistas de volta em uma lista ordenada.

### 1.0.2 Desempenho do mergesort

Primeiramente vamos definir o que é melhor, médio e pior caso para o MergeSort.
Melhor Caso – nunca é necessário trocar após comparações.
Médio Caso – há necessidade de haver troca após comparações.
Pior Caso – sempre é necessário trocar após comparações.
P ara o MergeSort não tem tanta importância se o vetor está no melhor, médio ou pior caso, porque para qualquer que seja o caso ele sempre terá a complexidade de ordem

$$n * logn$$

# Capítulo 2

# Resultados

## 2.1 Tabelas

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000519 |
| 64 | 63 | 0.001137 |
| 128 | 127 | 0.002478 |
| 256 | 255 | 0.005270 |
| 512 | 511 | 0.012029 |
| 1024 | 1023 | 0.026283 |
| 2048 | 2047 | 0.060780 |
| 4096 | 4095 | 0.146784 |
| 8192 | 8191 | 0.398959 |

**Tabela 2.1:** *Tabela com vetor teste quase aletorio: a linha de interesse analisada para este caso é a 16*

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000513 |
| 64 | 63 | 0.001113 |
| 128 | 127 | 0.002482 |
| 256 | 255 | 0.005269 |
| 512 | 511 | 0.011844 |
| 1024 | 1023 | 0.026106 |
| 2048 | 2047 | 0.060101 |
| 4096 | 4095 | 0.145868 |
| 8192 | 8191 | 0.391886 |

**Tabela 2.2:** *Tabela com vetor teste crescente : a linha de interesse analisada para este caso é a 16*

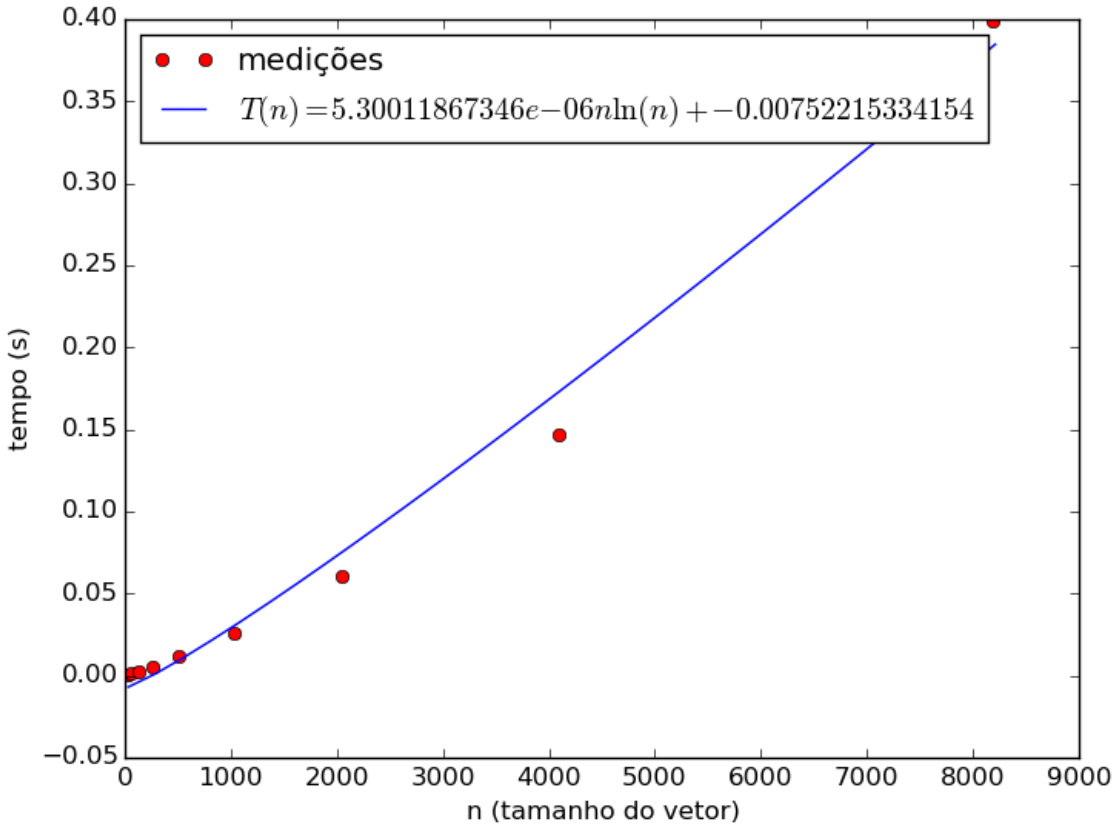


**Figura 2.1:** *A análise do grafico para $2^{32}$ segue abaixo para mergesort de vetor aleatorio. Tendo a função $T(n) = 5.3001 * n * ln(n) - 0.0075$ e para o $n = 2^{32}$, $T(2^{32}) = 5.04916 * 10^{11}$*

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000527 |
| 64 | 63 | 0.001117 |
| 128 | 127 | 0.002405 |
| 256 | 255 | 0.005217 |
| 512 | 511 | 0.011778 |
| 1024 | 1023 | 0.026174 |
| 2048 | 2047 | 0.060149 |
| 4096 | 4095 | 0.148273 |
| 8192 | 8191 | 0.391649 |

**Tabela 2.3:** *Tabela com vetor teste decrescente: a linha de interesse analisada para este caso é a 16*

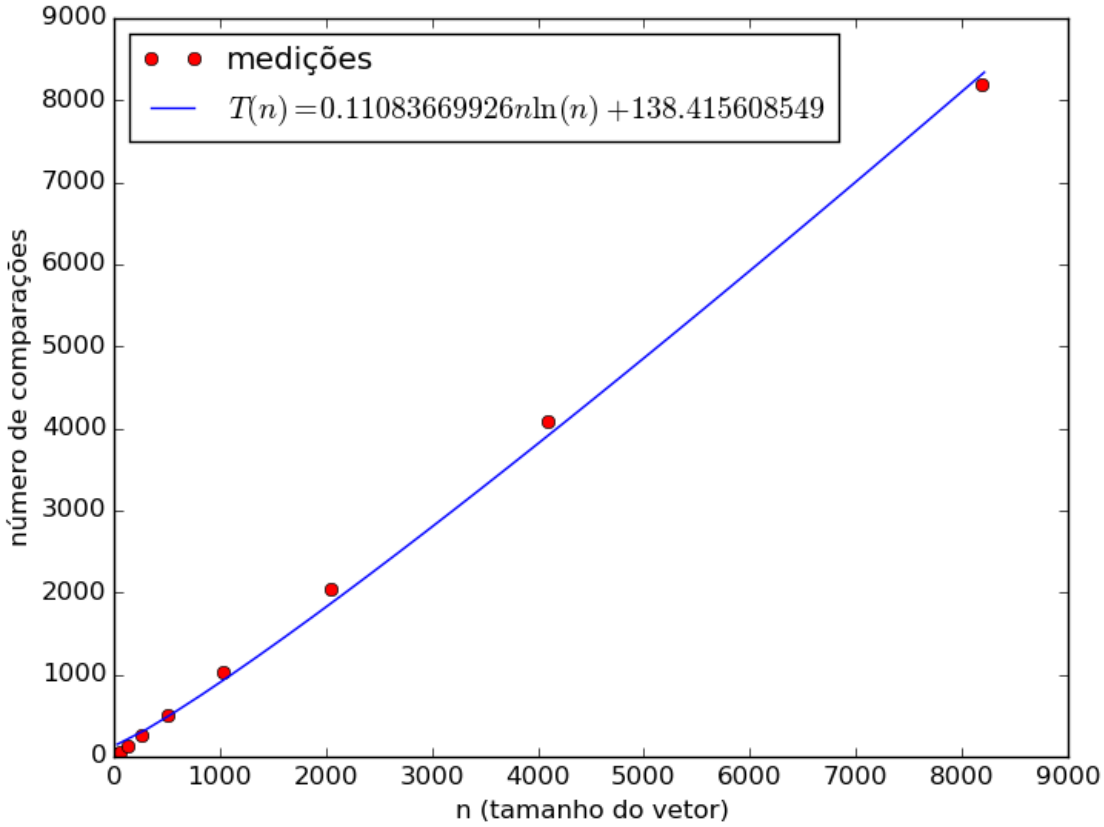


**Figura 2.2:** *A análise do grafico para $2^{32}$ segue abaixo para mergesort de vetor aleatorio. Tendo a função $T(n) = 0.1108 * n * ln(n) + 138.4156$ e para o $n = 2^{32}$, $T(2^{32}) = 1.05554 * 10^{10}$*

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000506 |
| 64 | 63 | 0.001116 |
| 128 | 127 | 0.002468 |
| 256 | 255 | 0.005274 |
| 512 | 511 | 0.011579 |
| 1024 | 1023 | 0.026290 |
| 2048 | 2047 | 0.059959 |
| 4096 | 4095 | 0.146191 |
| 8192 | 8191 | 0.393734 |

**Tabela 2.4:** *Tabela com vetor teste quase crescente 10%: a linha de interesse analisada para este caso é a 16*

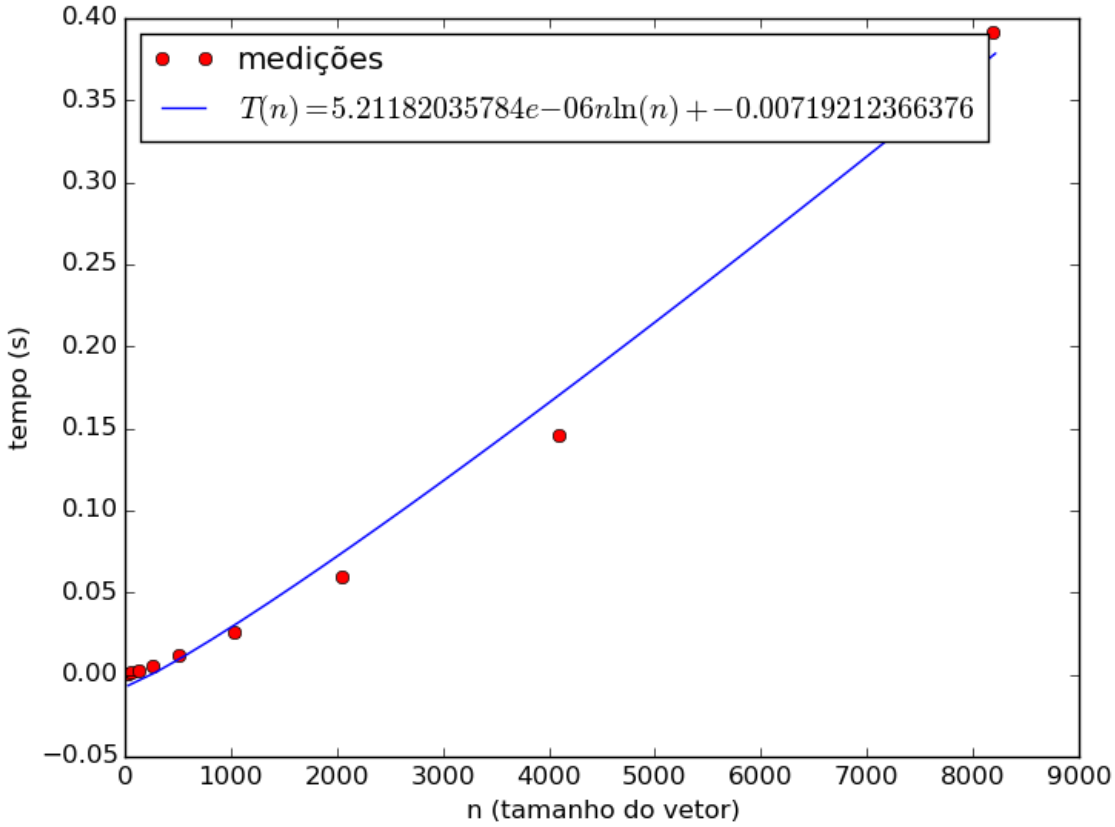


**Figura 2.3:** *A análise do grafico para $2^{32}$ segue abaixo para mergesort de vetor crescente. Tendo a função $T(n) = 5.2118\text{e} - 6 * n * ln(n) - 0.00719$ e para o $n = 2^{32}$, $T(2^{32}) = 4.96504 * 10^5$*

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000570 |
| 64 | 63 | 0.001104 |
| 128 | 127 | 0.002437 |
| 256 | 255 | 0.005351 |
| 512 | 511 | 0.011737 |
| 1024 | 1023 | 0.026590 |
| 2048 | 2047 | 0.061359 |
| 4096 | 4095 | 0.148785 |
| 8192 | 8191 | 0.388701 |

**Tabela 2.5:** *Tabela com vetor teste quase crescente 20%: a linha de interesse analisada para este caso é a 16*

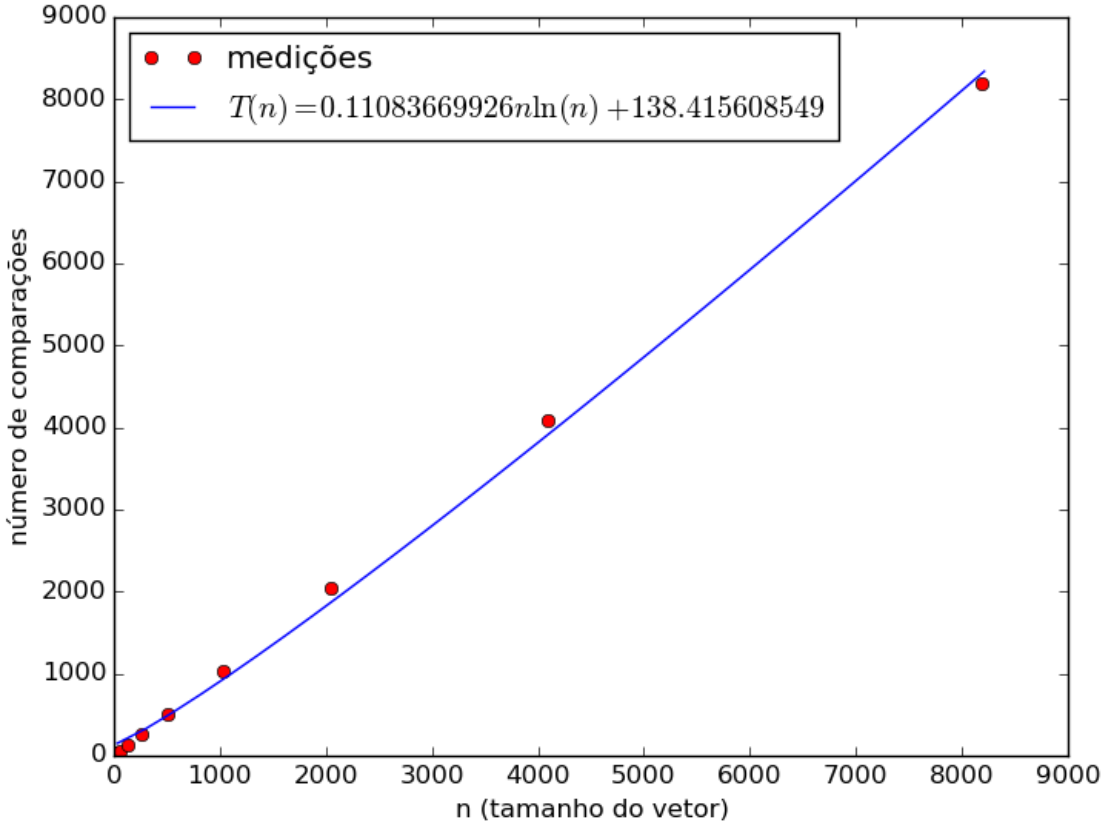


**Figura 2.4:** *A análise do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor crescente. Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

2.1

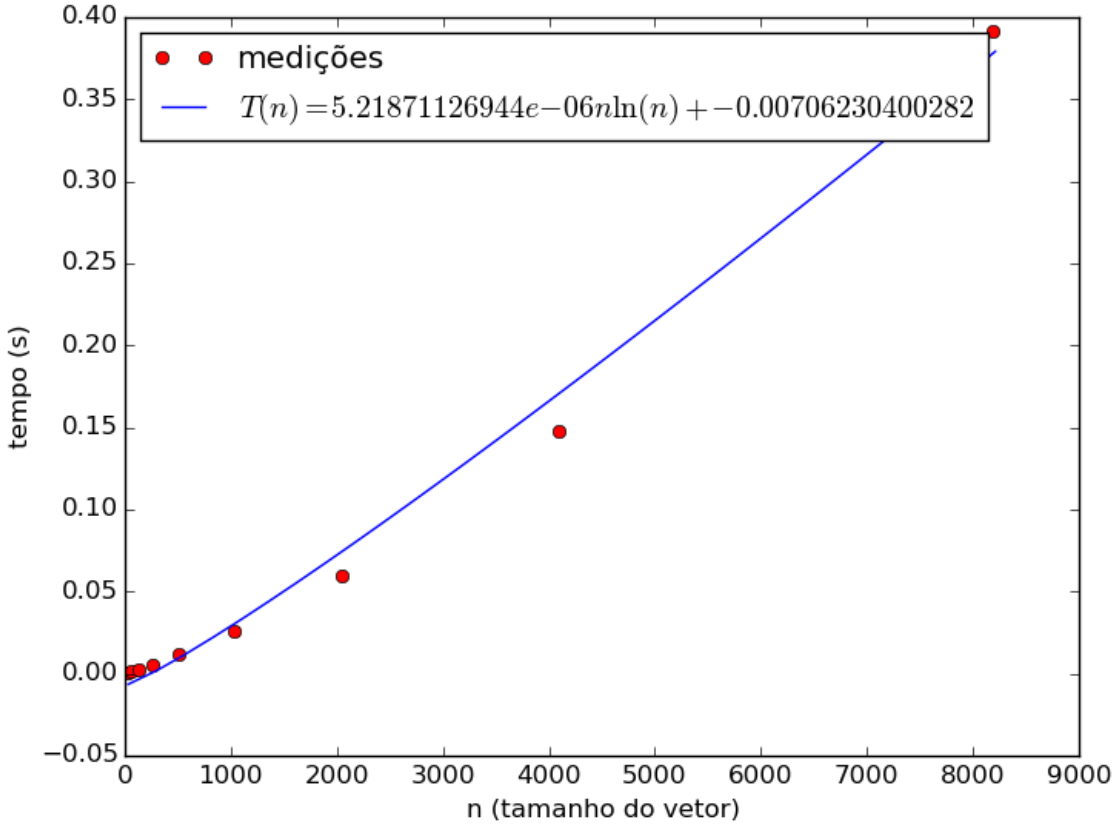


**Figura 2.5:** *A análise do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor decrescente. Tendo a função* $T(n) = 5.2187\text{e} - 6 * n * ln(n) - 0.00706$ *e para o* $n = 2^{32}$, $T(2^{32}) = 4.97162 * 10^5$

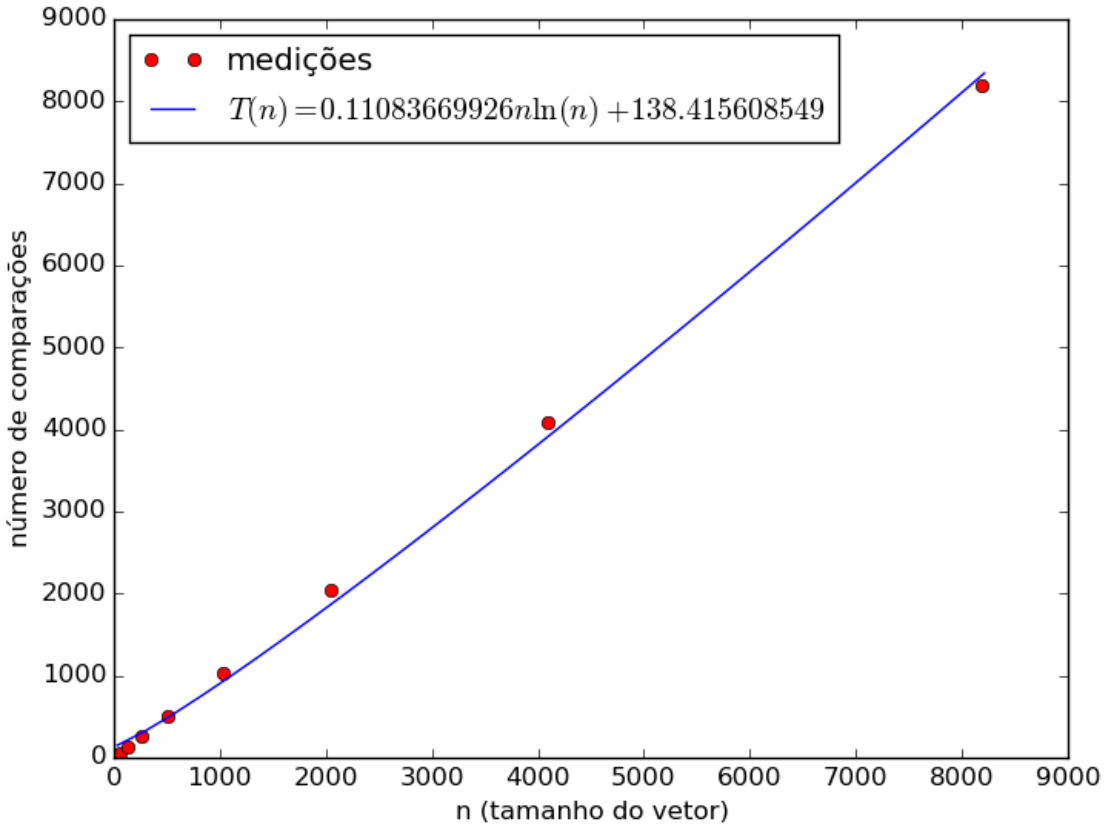


**Figura 2.6:** *A análise numero de comparações do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor crescente.*
*Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

2.1

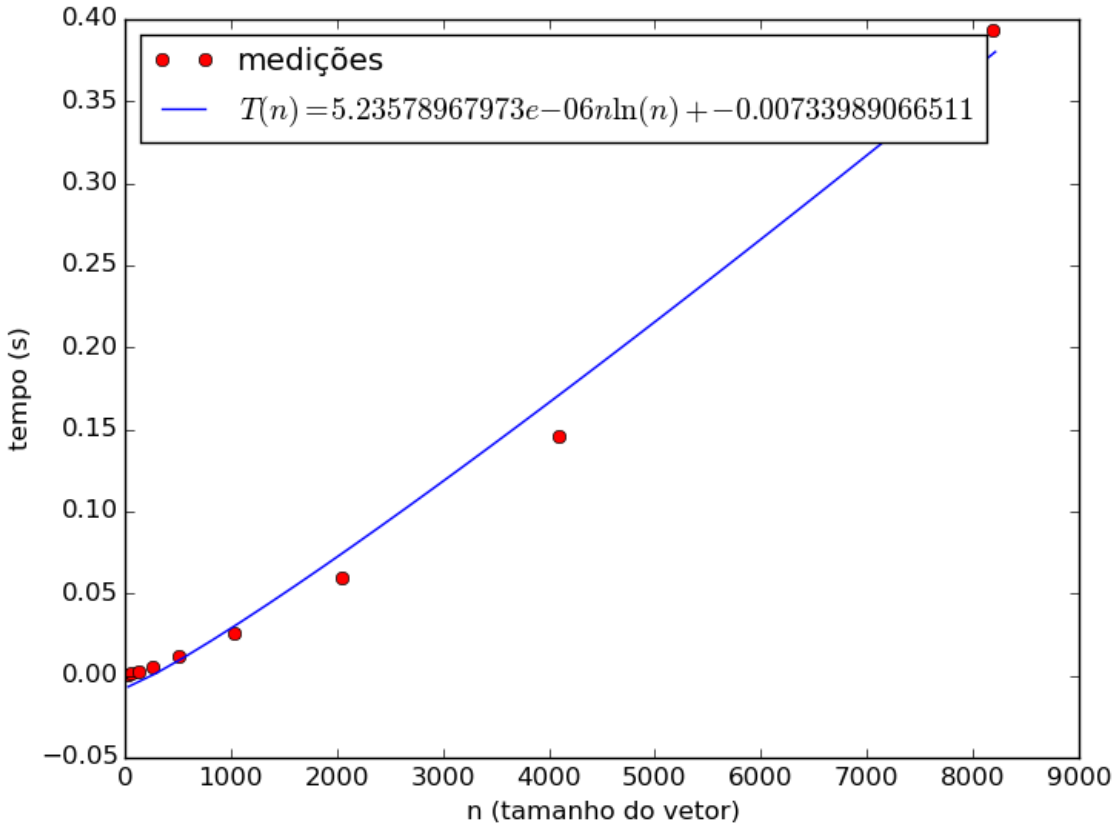


**Figura 2.7:** *A análise do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase crescente 10%.*
*Tendo a função* $T(n) = 5.2357\text{e} - 6 * n * ln(n) - 0.0073$ *e para o* $n = 2^{32}$, $T(2^{32}) = 4.98781 * 10^5$

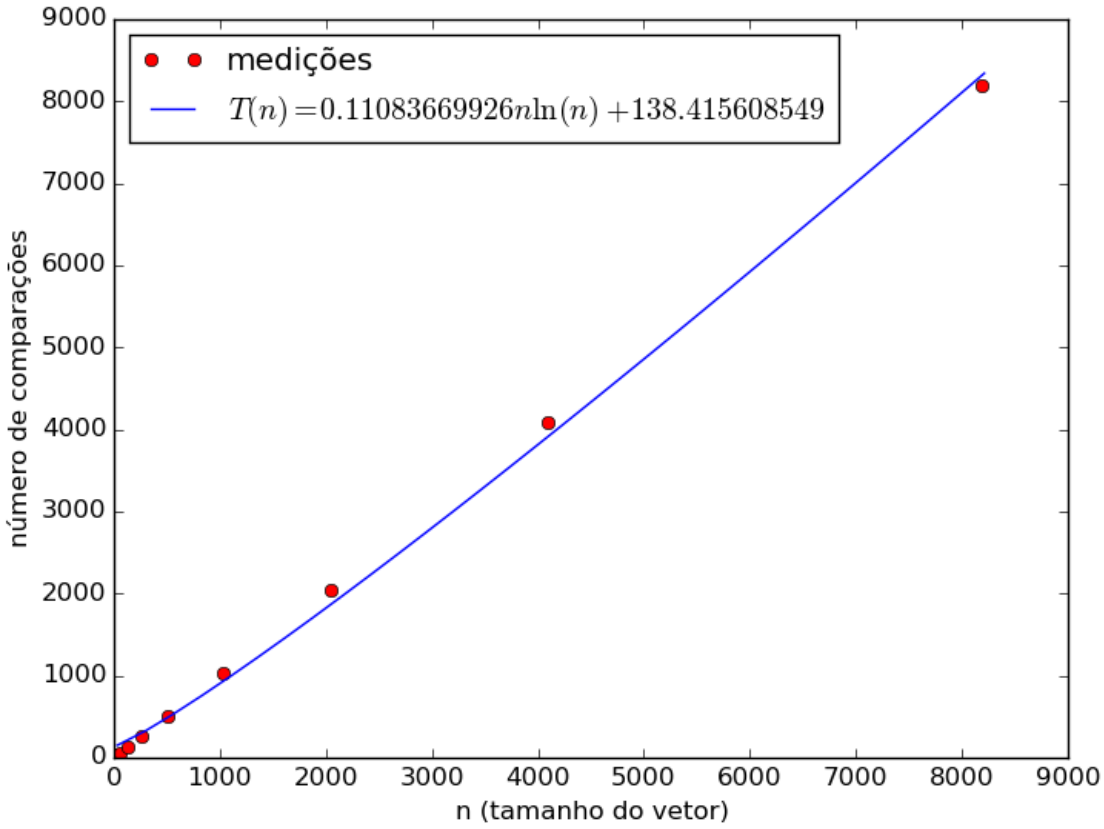


**Figura 2.8:** *A análise numero de comparações do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase crescente 10%.*
*Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

2.1

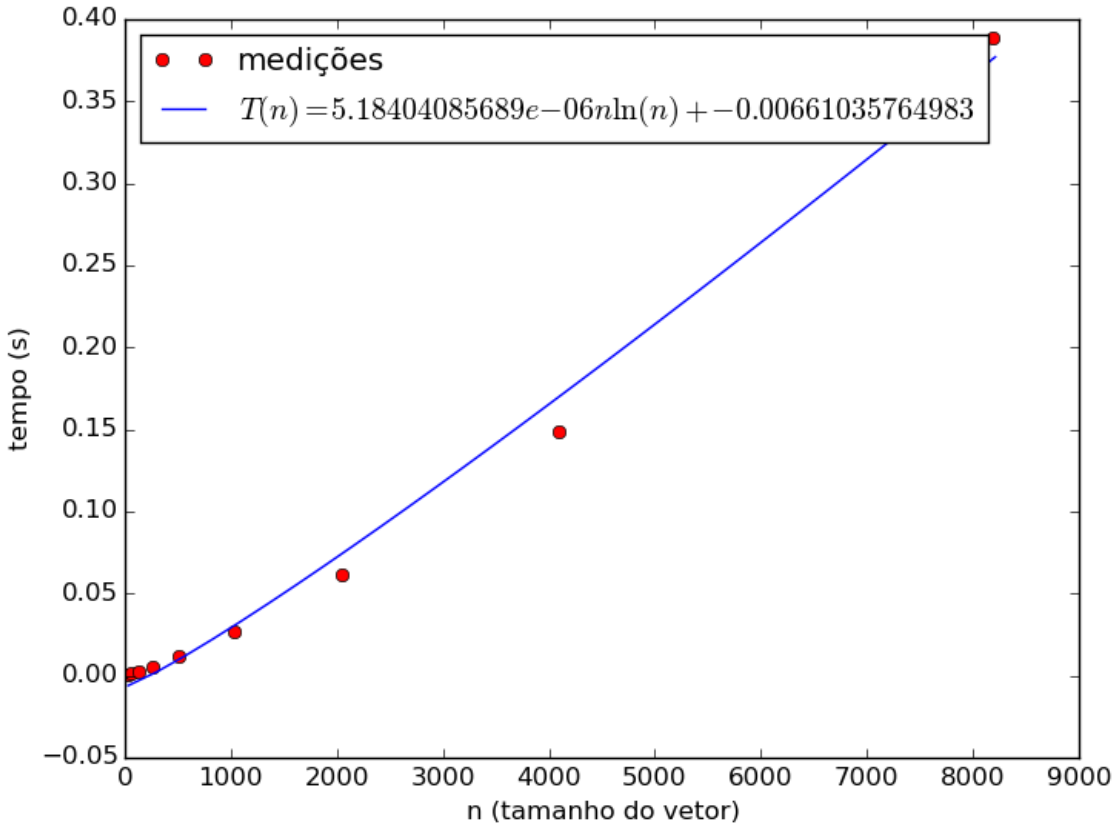


**Figura 2.9:** *A análise do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase crescente 20%.*
*Tendo a função* $T(n) = 5.1840\text{e} - 6 * n * ln(n) - 0.0066$ *e para o* $n = 2^{32}$, $T(2^{32}) = 4.93856 * 10^5$

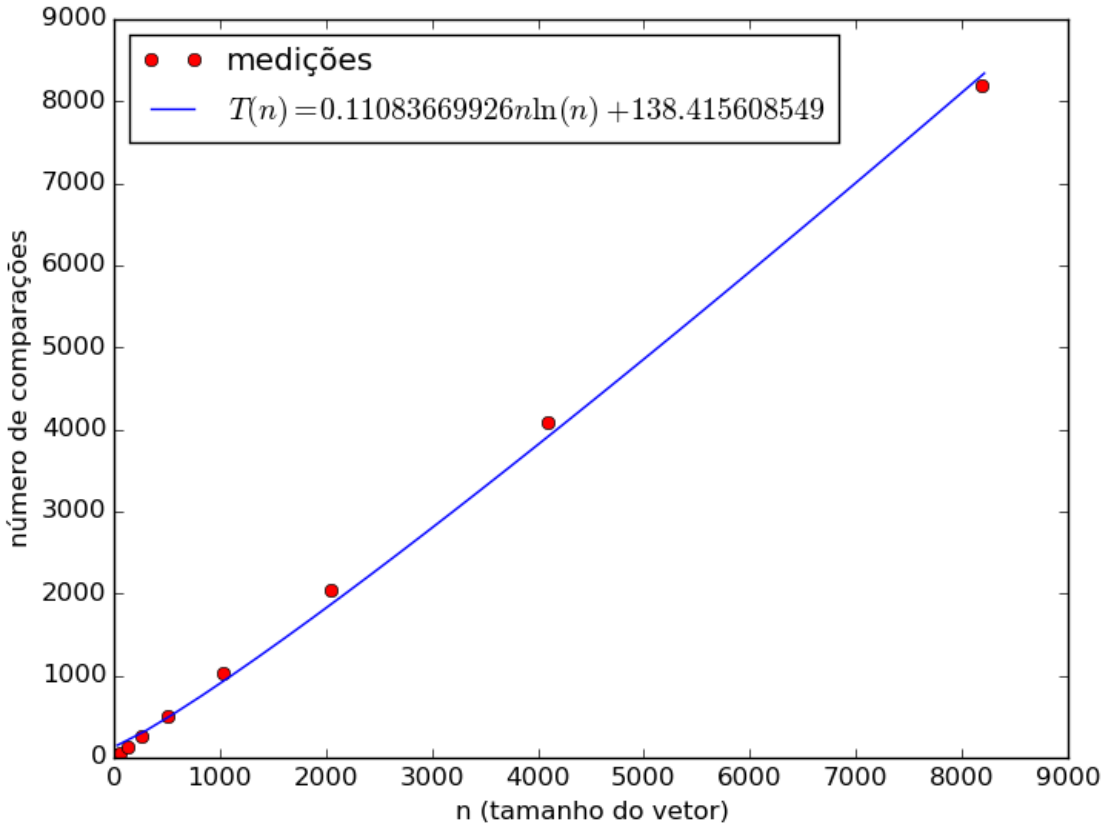


**Figura 2.10:** *A análise numero de comparações do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase crescente 20%.*
*Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000530 |
| 64 | 63 | 0.001109 |
| 128 | 127 | 0.002468 |
| 256 | 255 | 0.005251 |
| 512 | 511 | 0.011655 |
| 1024 | 1023 | 0.025999 |
| 2048 | 2047 | 0.060406 |
| 4096 | 4095 | 0.148614 |
| 8192 | 8191 | 0.391055 |

**Tabela 2.6:** *Tabela com vetor teste quase crescente 30%: a linha de interesse analisada para este caso é a 16*

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000511 |
| 64 | 63 | 0.001165 |
| 128 | 127 | 0.002398 |
| 256 | 255 | 0.005383 |
| 512 | 511 | 0.011723 |
| 1024 | 1023 | 0.026041 |
| 2048 | 2047 | 0.061550 |
| 4096 | 4095 | 0.147485 |
| 8192 | 8191 | 0.386246 |

**Tabela 2.7:** *Tabela com vetor teste quase crescente 40%: a linha de interesse analisada para este caso é a 15*

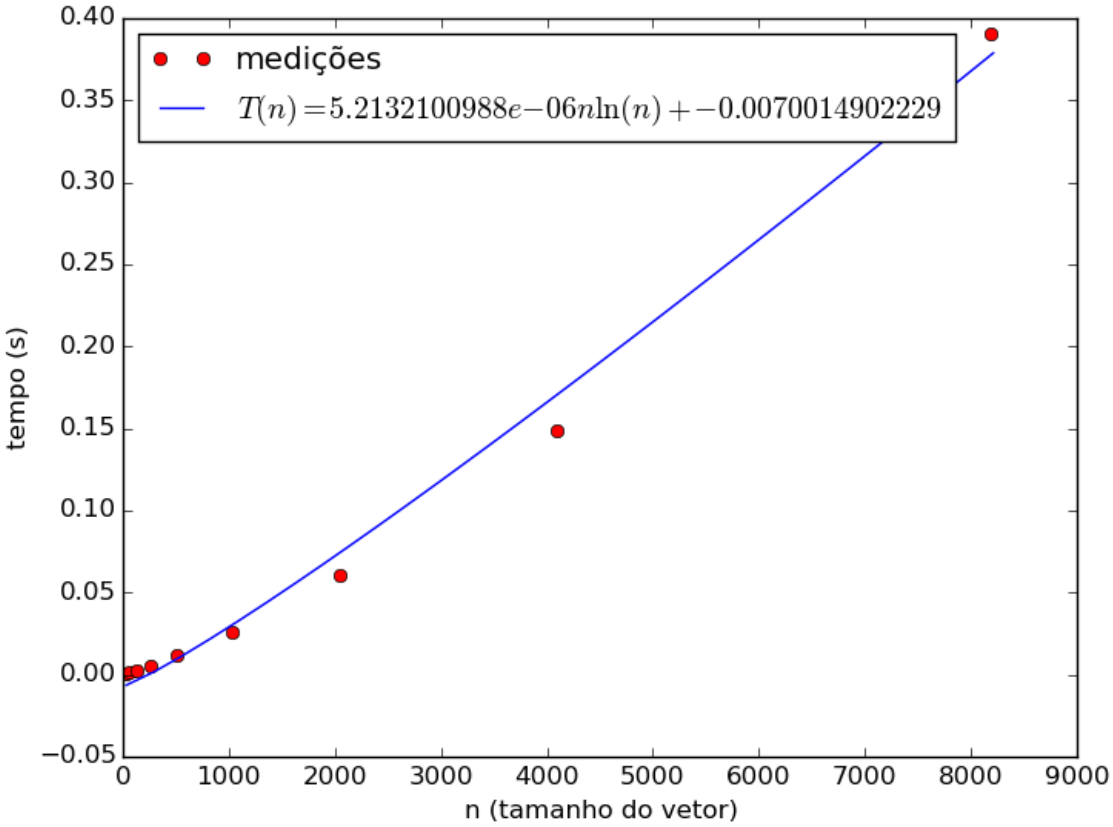


**Figura 2.11:** *A análise do grafico para $2^{32}$ segue abaixo para mergesort de vetor quase crescente 30%.*
*Tendo a função $T(n) = 5.2132\text{e} - 6 * n * ln(n) - 0.007$ e para o $n = 2^{32}$, $T(2^{32}) = 4.96638 * 10^5$*

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000517 |
| 64 | 63 | 0.001173 |
| 128 | 127 | 0.002439 |
| 256 | 255 | 0.005322 |
| 512 | 511 | 0.011837 |
| 1024 | 1023 | 0.026087 |
| 2048 | 2047 | 0.060615 |
| 4096 | 4095 | 0.147344 |
| 8192 | 8191 | 0.392809 |

**Tabela 2.8:** *Tabela com vetor teste quase crescente 50%: a linha de interesse analisada para este caso é a 16*

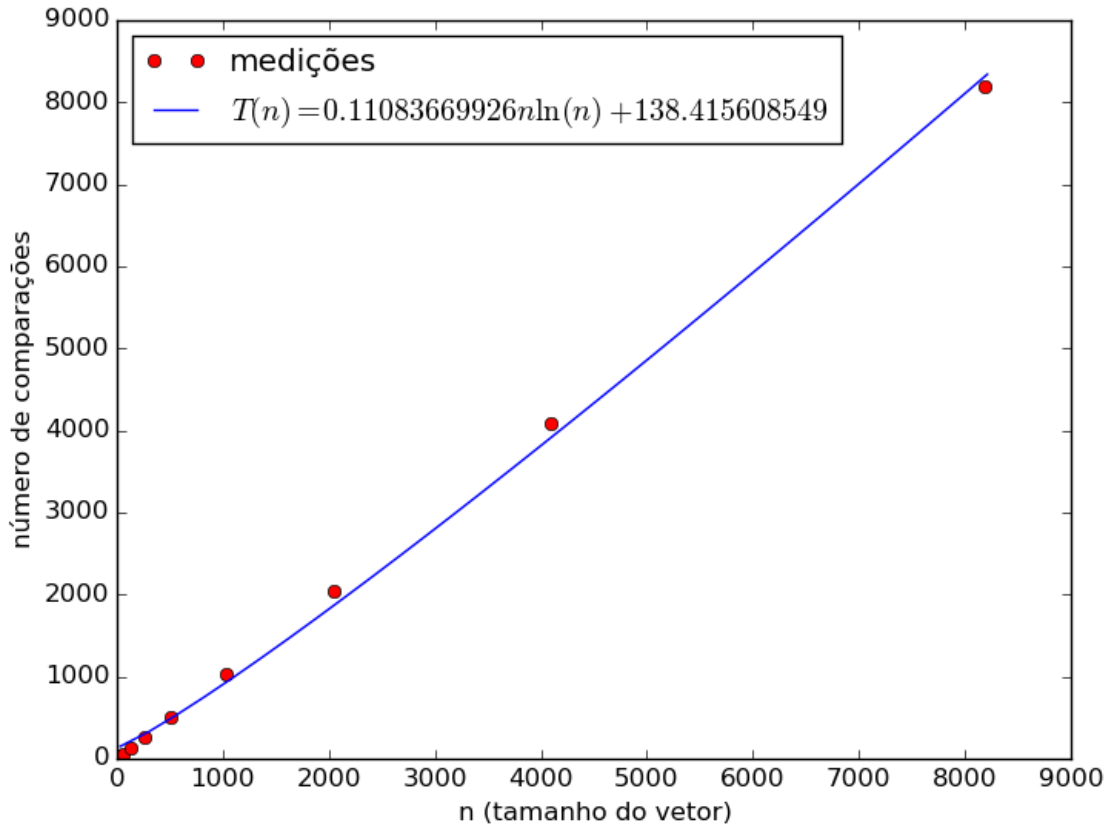


**Figura 2.12:** *A análise numero de comparações do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase crescente 30%.*
*Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000538 |
| 64 | 63 | 0.001107 |
| 128 | 127 | 0.002456 |
| 256 | 255 | 0.005264 |
| 512 | 511 | 0.011877 |
| 1024 | 1023 | 0.026144 |
| 2048 | 2047 | 0.060374 |
| 4096 | 4095 | 0.147950 |
| 8192 | 8191 | 0.388941 |

**Tabela 2.9:** *Tabela com vetor teste quase decrescente 10%: a linha de interesse analisada para este caso é a 16*

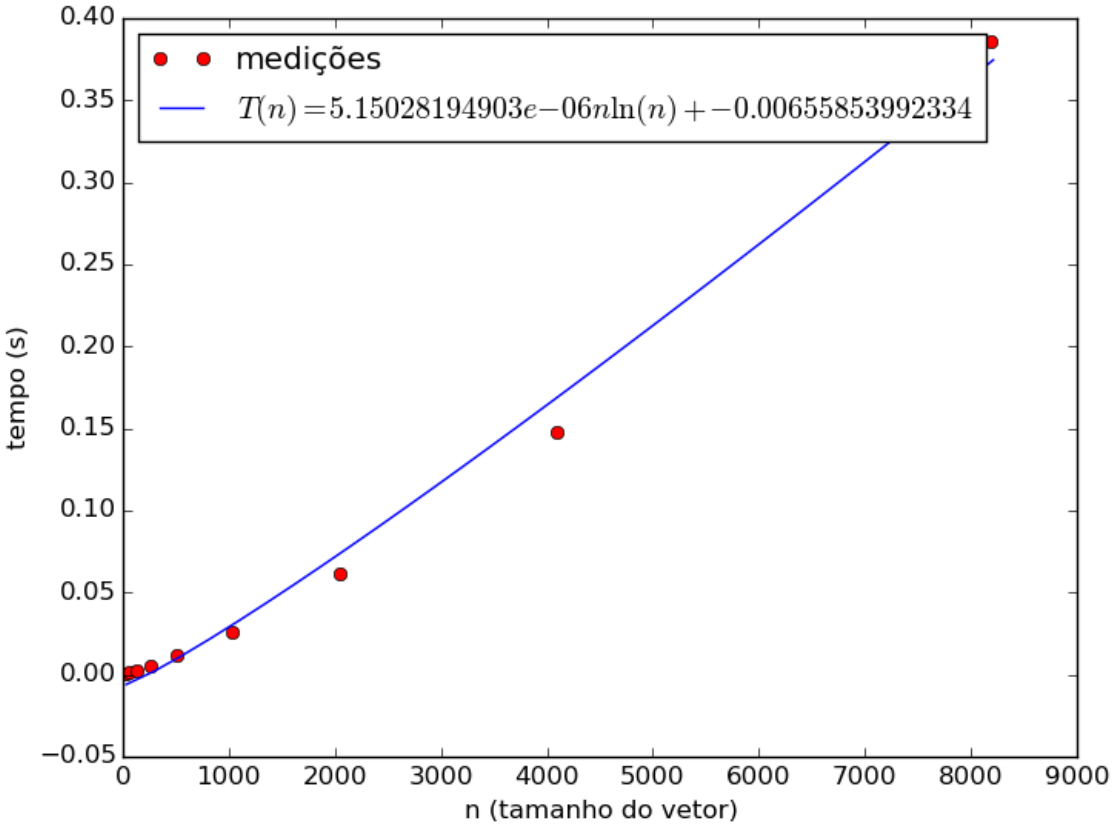


**Figura 2.13:** *A análise do grafico para $2^{32}$ segue abaixo para mergesort de vetor quase crescente 40%.*
*Tendo a função $T(n) = 5.1550\text{e} - 6 * n * ln(n) - 0.0065$ e para o $n = 2^{32}$, $T(2^{32}) = 4.91093 * 10^5$*

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000518 |
| 64 | 63 | 0.001109 |
| 128 | 127 | 0.002432 |
| 256 | 255 | 0.005340 |
| 512 | 511 | 0.011881 |
| 1024 | 1023 | 0.027017 |
| 2048 | 2047 | 0.062077 |
| 4096 | 4095 | 0.150054 |
| 8192 | 8191 | 0.407999 |

**Tabela 2.10:** *Tabela com vetor teste quase decrescente 20%: a linha de interesse analisada para este caso é a 16*

2.1

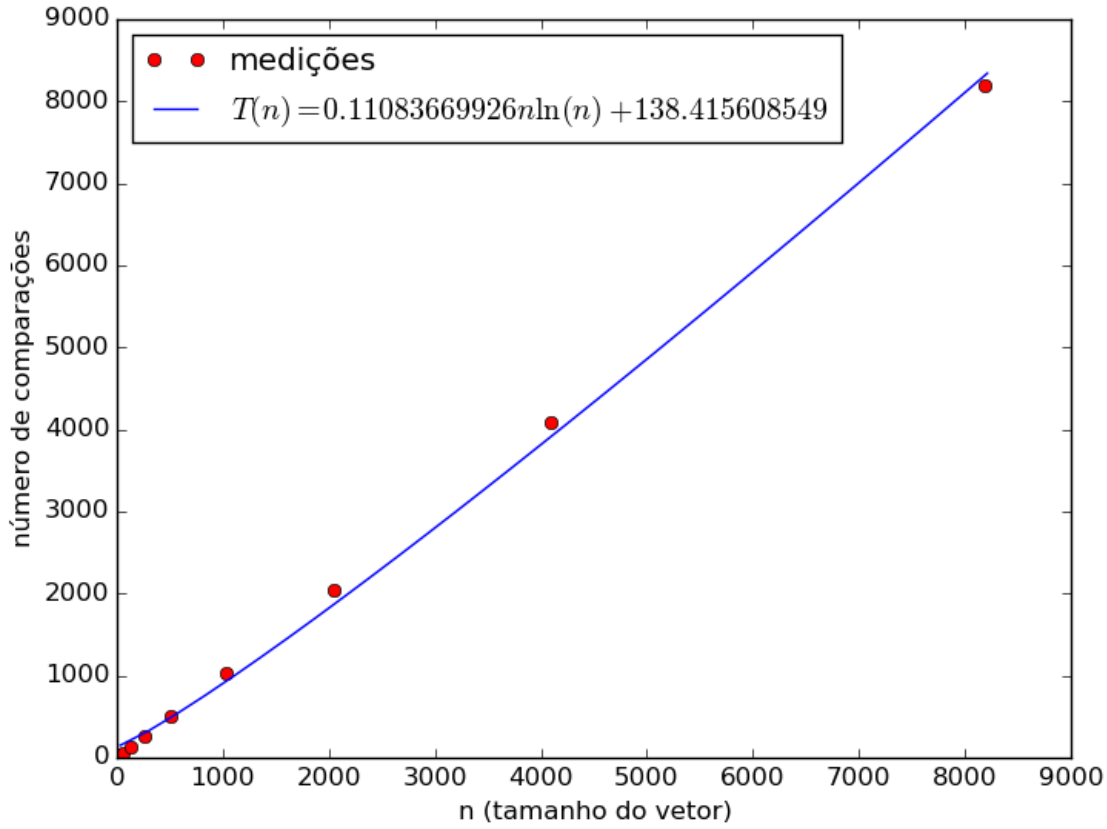


**Figura 2.14:** *A análise numero de comparações do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase crescente 40%.*
*Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

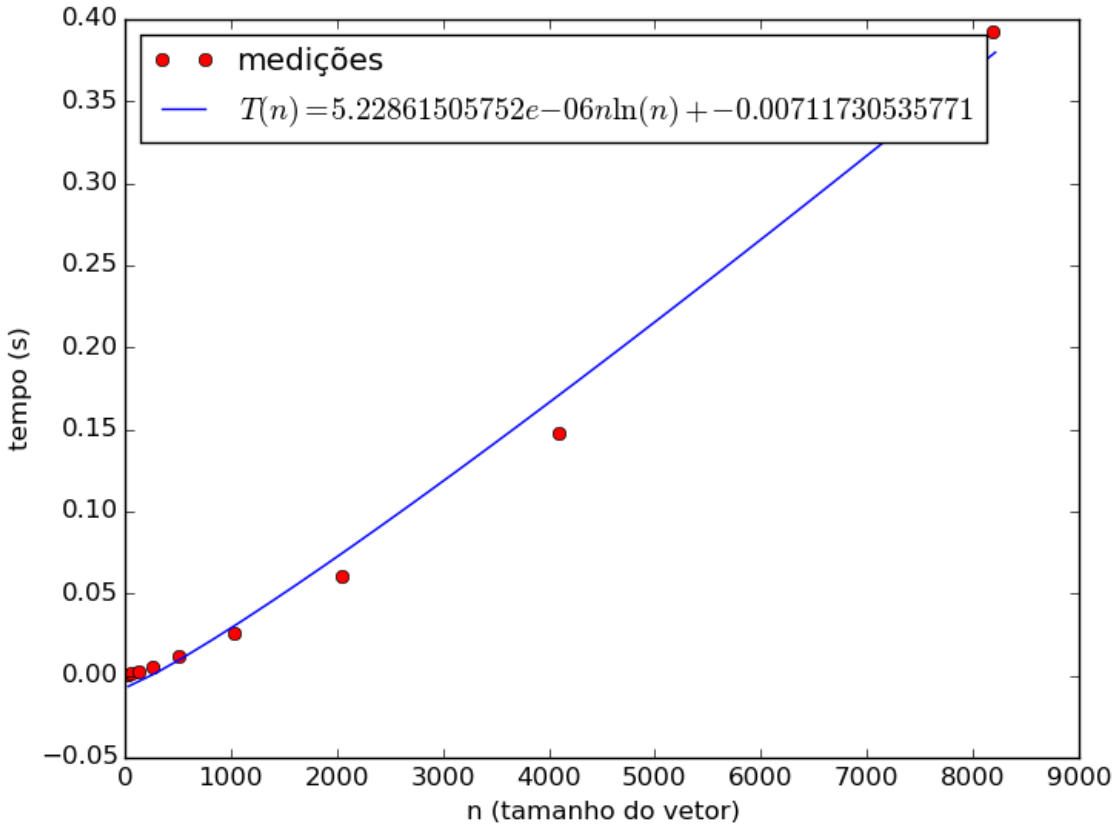


**Figura 2.15:** *A análise do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase crescente 50%.*
*Tendo a função* $T(n) = 5.2286\text{e} - 6 * n * ln(n) - 0.0071$ *e para o* $n = 2^{32}$, $T(2^{32}) = 498105$

2.1

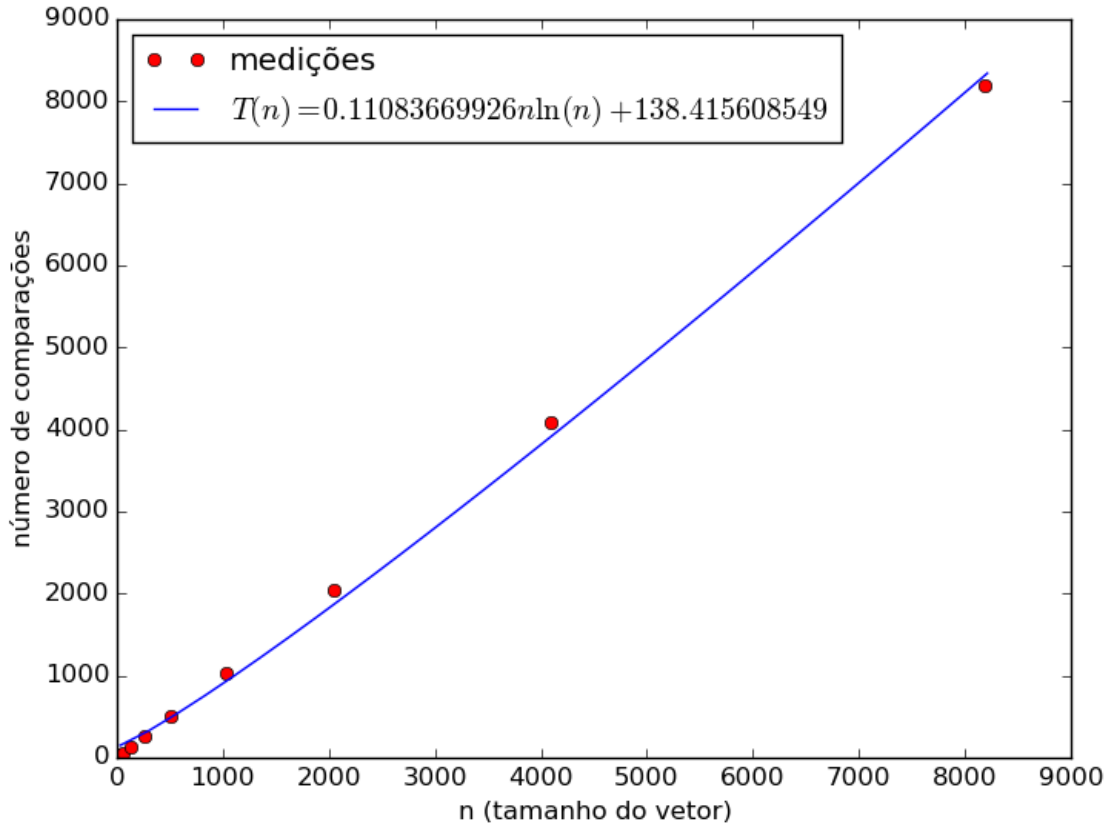


**Figura 2.16:** *A análise numero de comparações do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase crescente 50%.*
*Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

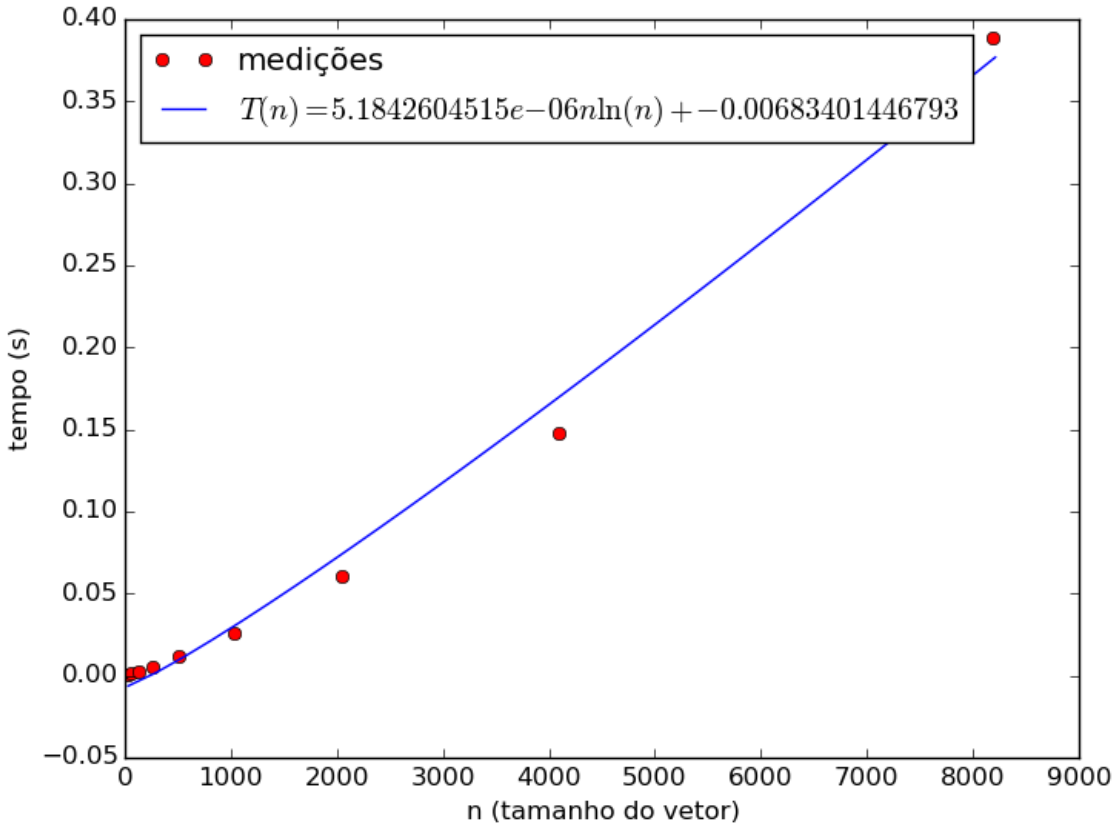


**Figura 2.17:** *A análise do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase decrescente 10%.*
*Tendo a função* $T(n) = 5.1842\text{e} - 6 * n * ln(n) - 0.0068$ *e para o* $n = 2^{32}$, $T(2^{32}) = 4.93875 * 10^5$

2.1

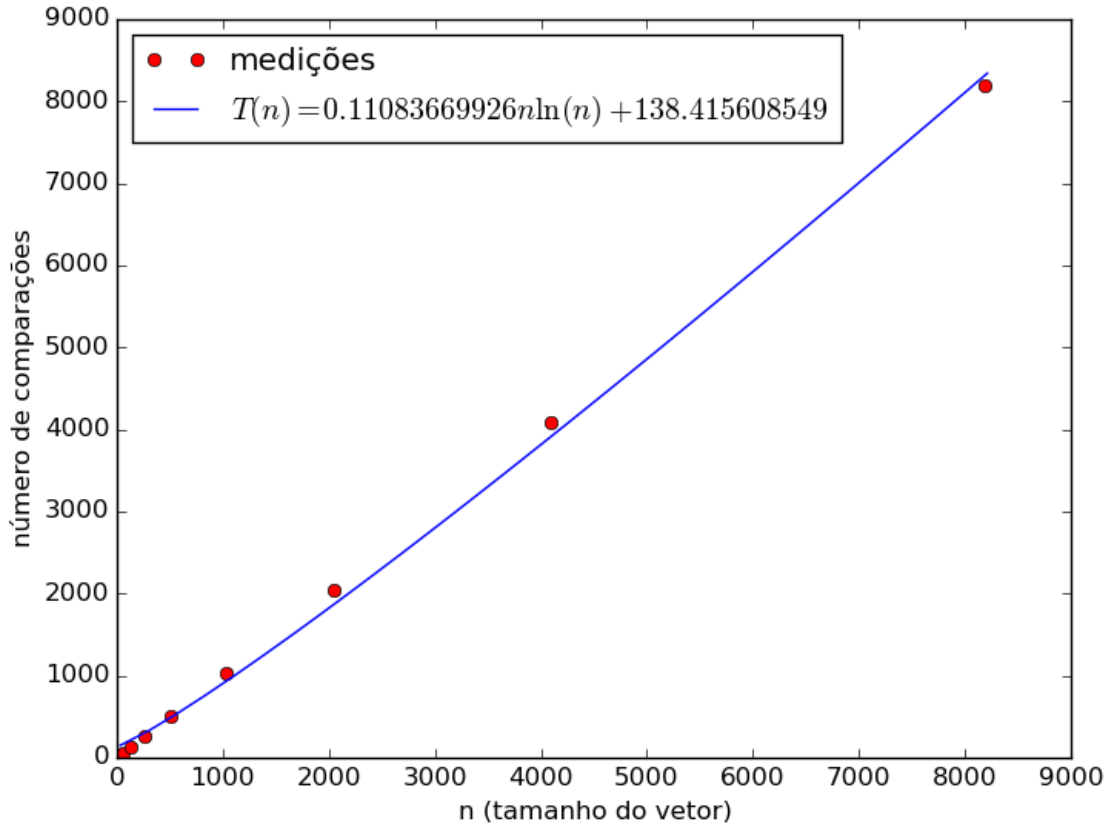


**Figura 2.18:** *A análise numero de comparações do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase decrescente 10%.*
*Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

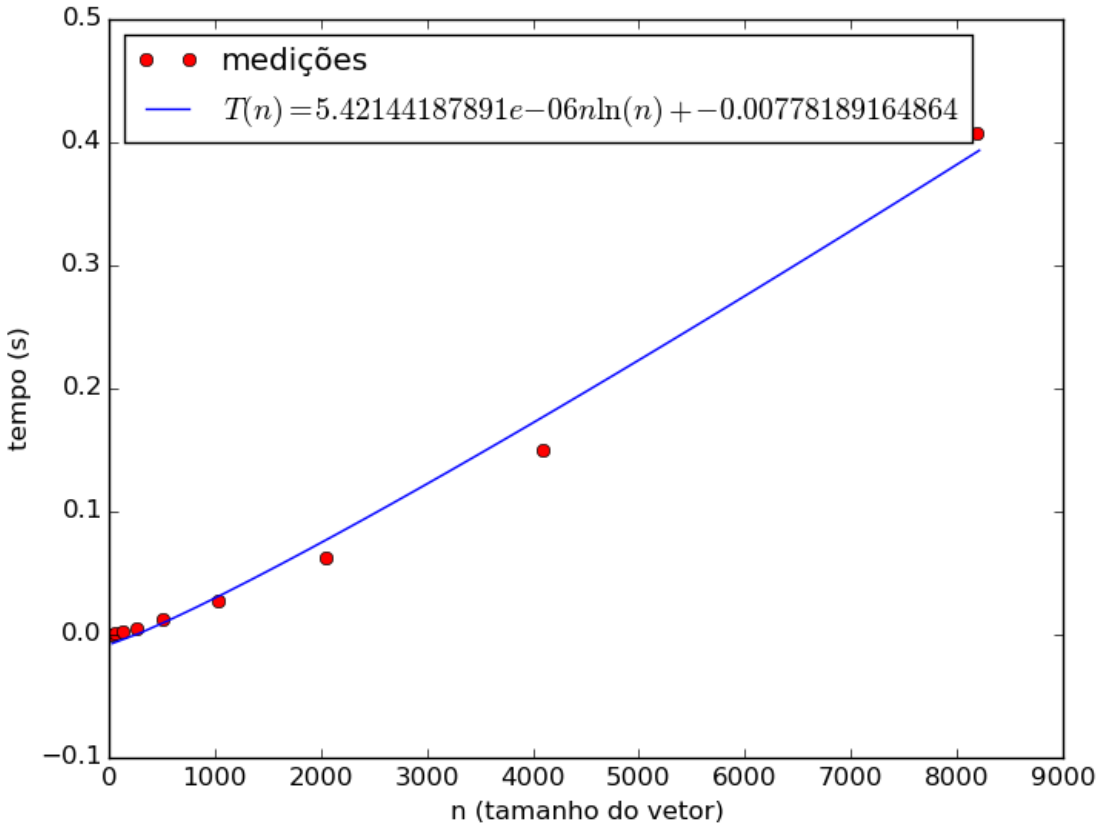


**Figura 2.19:** *A análise do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase decrescente 20%.*
*Tendo a função* $T(n) = 5.4214\text{e} - 6 * n * ln(n) - 0.0077$ *e para o* $n = 2^{32}$, $T(2^{32}) = 5.16472 * 10^5$

2.1

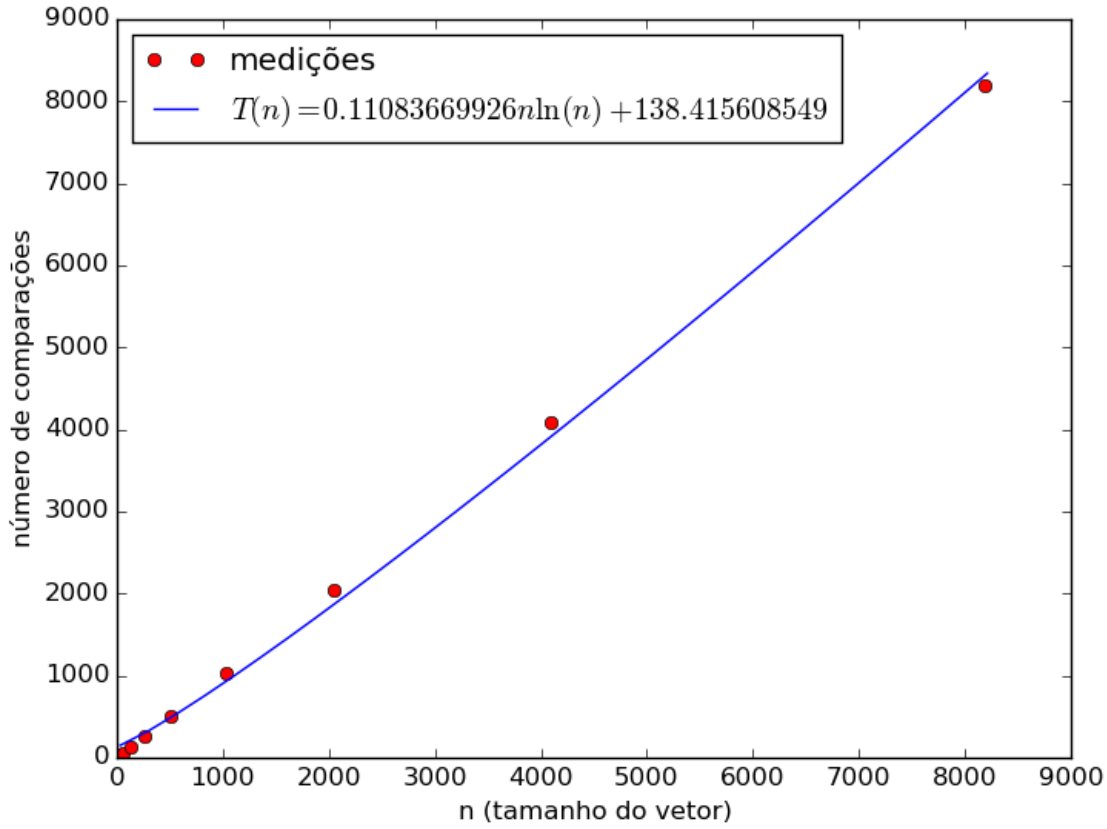


**Figura 2.20:** *A análise numero de comparações do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase decrescente 20%.*
*Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000535 |
| 64 | 63 | 0.001124 |
| 128 | 127 | 0.002435 |
| 256 | 255 | 0.005314 |
| 512 | 511 | 0.011761 |
| 1024 | 1023 | 0.026343 |
| 2048 | 2047 | 0.061023 |
| 4096 | 4095 | 0.147302 |
| 8192 | 8191 | 0.388036 |

**Tabela 2.11:** *Tabela com vetor teste quase decrescente 30%: a linha de interesse analisada para este caso é a 16*

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000504 |
| 64 | 63 | 0.001126 |
| 128 | 127 | 0.002481 |
| 256 | 255 | 0.005277 |
| 512 | 511 | 0.011759 |
| 1024 | 1023 | 0.025893 |
| 2048 | 2047 | 0.060078 |
| 4096 | 4095 | 0.146398 |
| 8192 | 8191 | 0.387558 |

**Tabela 2.12:** *Tabela com vetor teste quase decrescente 40%: a linha de interesse analisada para este caso é a 16*

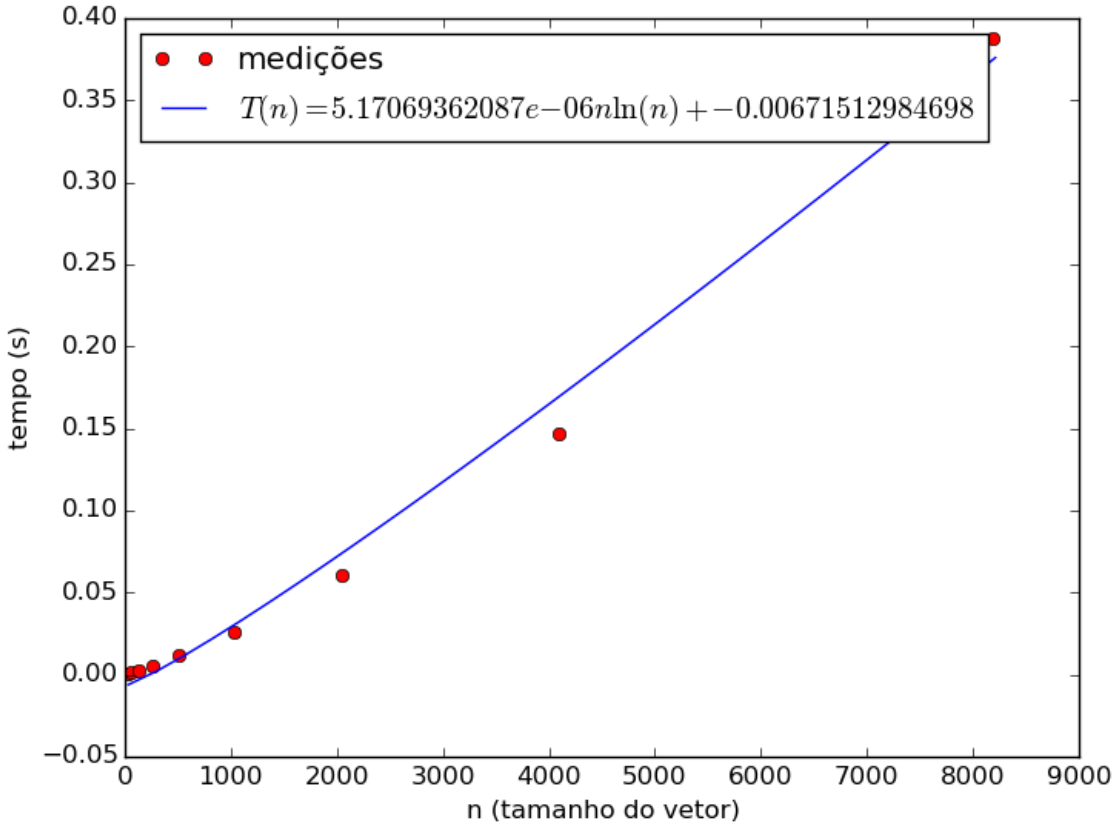


**Figura 2.21:** *A análise do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase decrescente 30%.*
*Tendo a função* $T(n) = 5.1706\text{e} - 6 * n * ln(n) - 0.0067$ *e para o* $n = 2^{32}$, $T(2^{32}) = 4.92579 * 10^5$

| n | comparações | tempo(s) |
|---|---|---|
| 32 | 31 | 0.000520 |
| 64 | 63 | 0.001125 |
| 128 | 127 | 0.002412 |
| 256 | 255 | 0.005348 |
| 512 | 511 | 0.011810 |
| 1024 | 1023 | 0.026291 |
| 2048 | 2047 | 0.060189 |
| 4096 | 4095 | 0.147712 |
| 8192 | 8191 | 0.398088 |

**Tabela 2.13:** *Tabela com vetor teste quase decrescente 50%: a linha de interesse analisada para este caso é a 16*

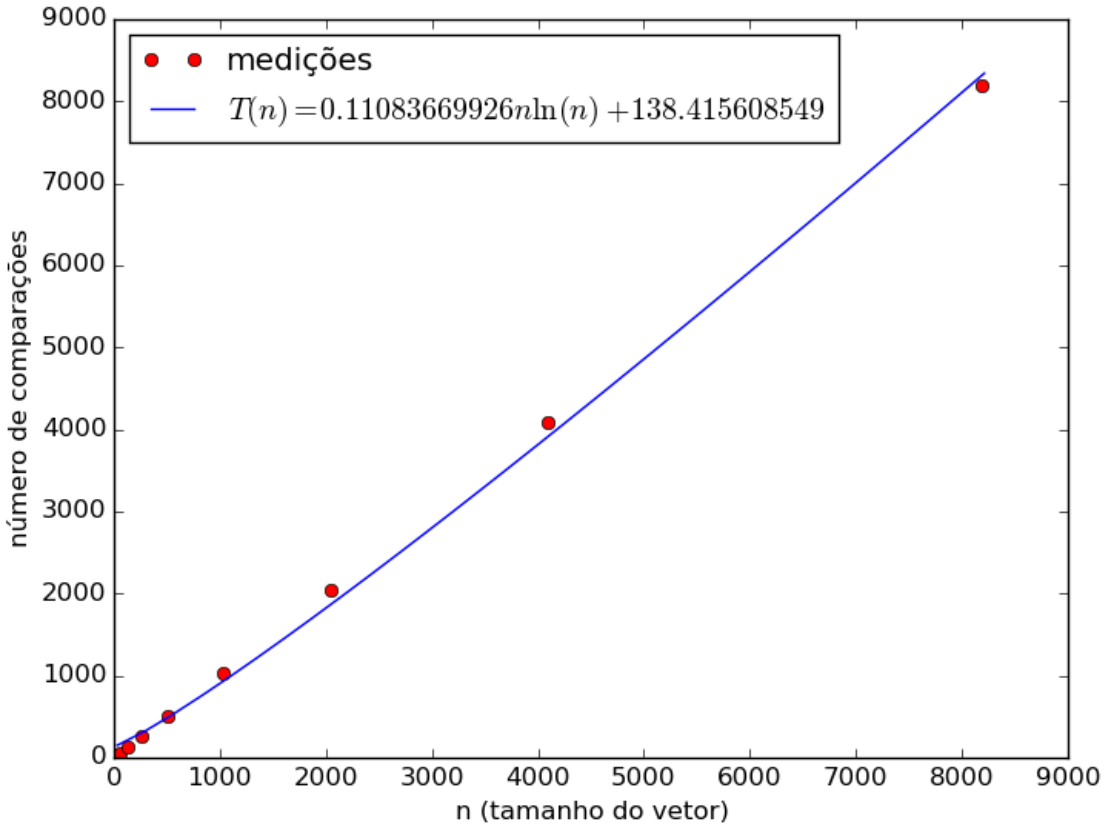


**Figura 2.22:** *A análise numero de comparações do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase decrescente 30%.*
*Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

2.1

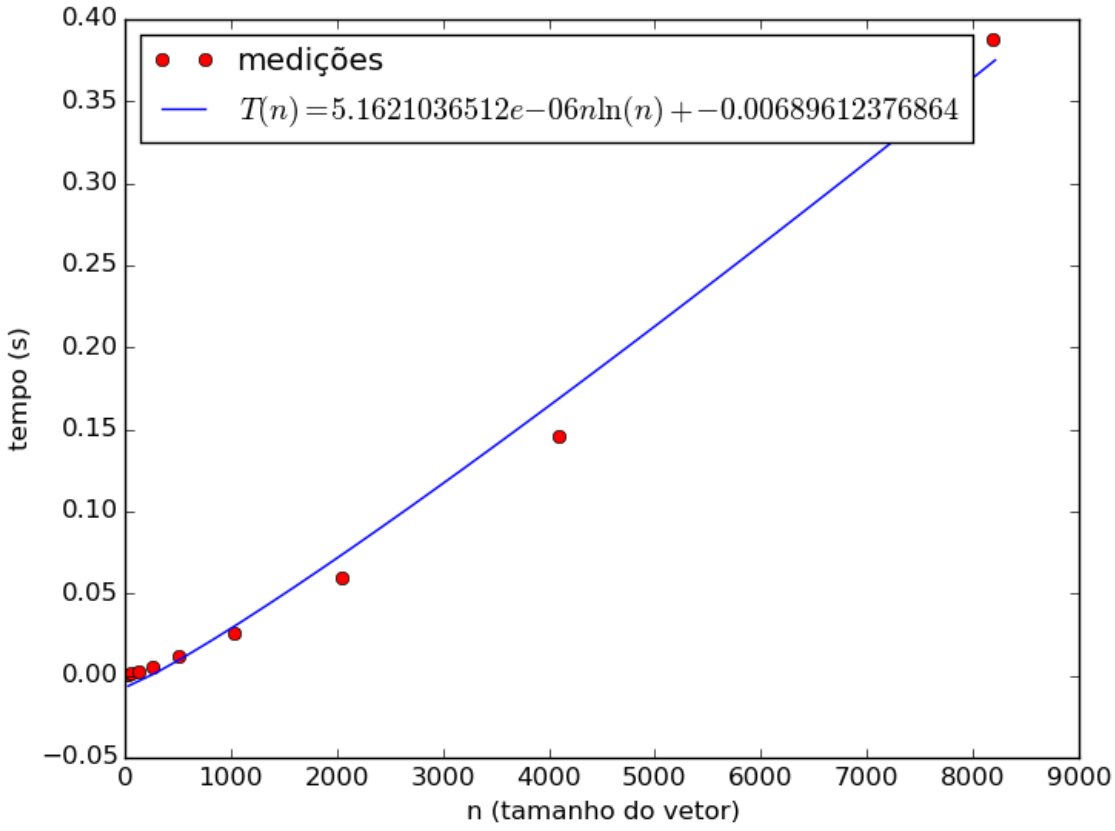


**Figura 2.23:** *A análise do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase decrescente 30%.*
*Tendo a função* $T(n) = 5.1621\text{e} - 6 * n * ln(n) - 0.0089$ *e para o* $n = 2^{32}$, $T(2^{32}) = 4.91770 * 10^5$

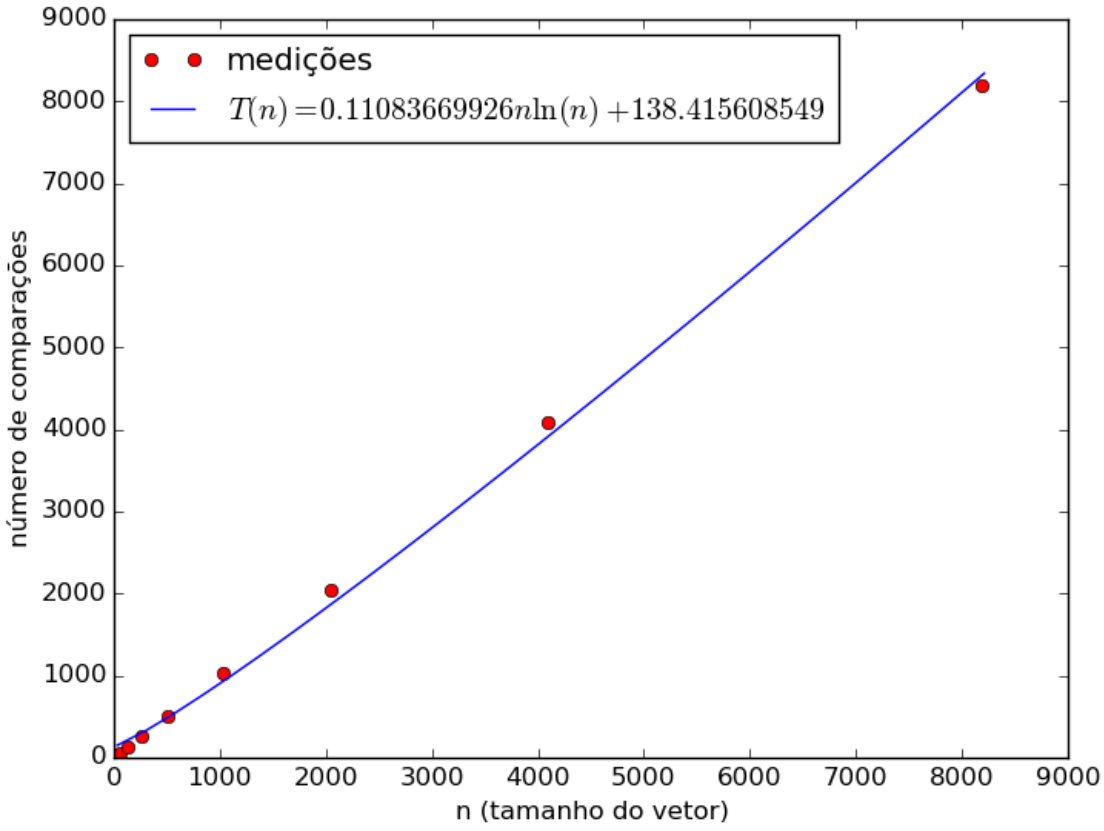


**Figura 2.24:** *A análise numero de comparações do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase decrescente 40%.*
*Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

2.1

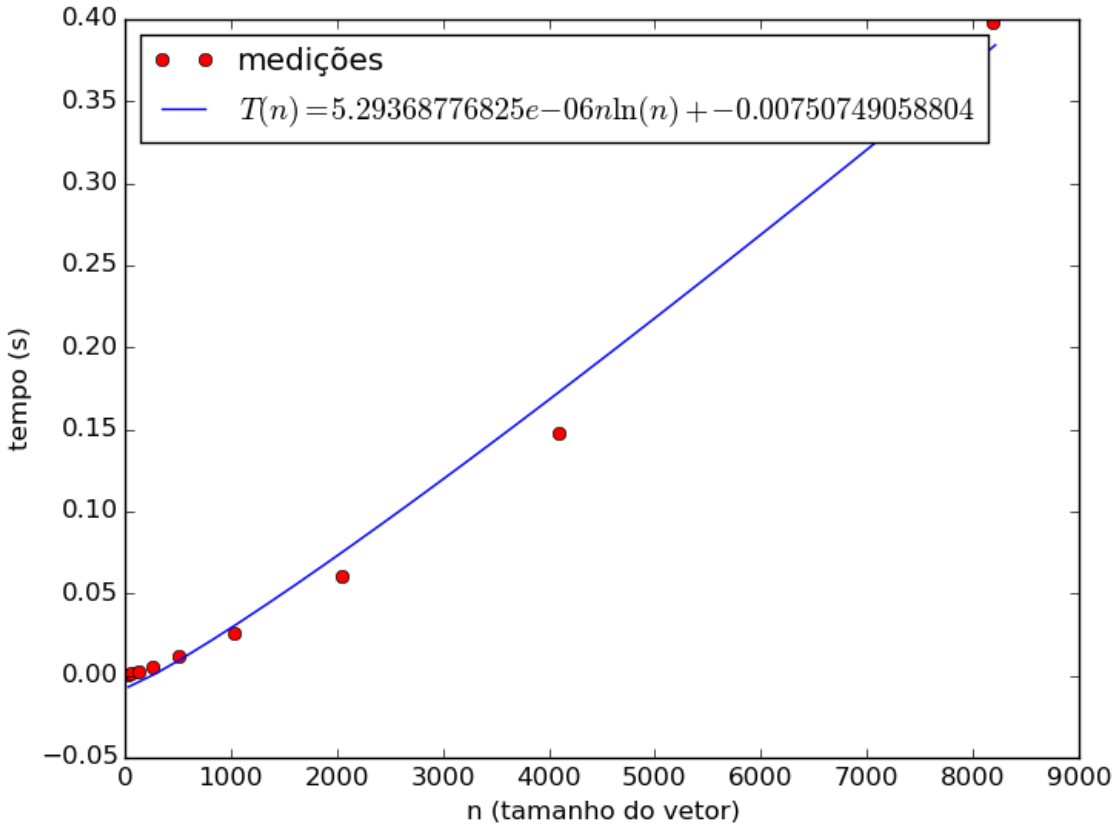


**Figura 2.25:** *A análise do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase decrescente 30%.*
*Tendo a função* $T(n) = 5.2936\text{e} - 6 * n * ln(n) - 0.0075$ *e para o* $n = 2^{32}$, $T(2^{32}) = 5.04297 * 10^5$

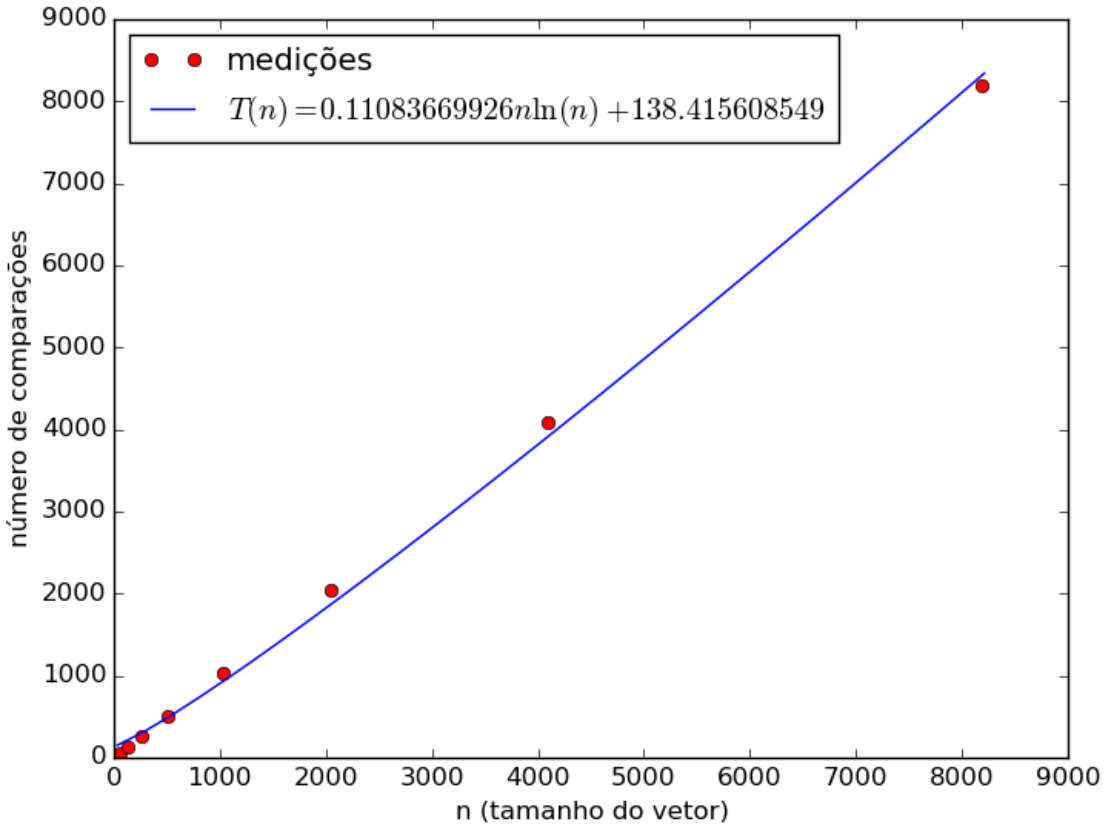


**Figura 2.26:** *A análise numero de comparações do grafico para* $2^{32}$ *segue abaixo para mergesort de vetor quase decrescente 50%.*
*Tendo a função* $T(n) = 0.11083 * n * ln(n) - 138.4156$ *e para o* $n = 2^{32}$, $T(2^{32}) = 1.05583 * 10^{10}$

# Apêndice A

# Arquivo ../mergesort/mergesort.py

Listagem A.1: ../mergesort/mergesort.py

```
1  import math
2
3  def intercala(A,p,q,r):
4    B = [0] *len(A)
5    for i in range(p,(q+1)):
6      B[i] = A[i]
7    for j in range(q+1,(r+1)):
8      B[r+q+1-j] = A[j]
9    i = p
10   j = r
11   for k in range (p,(r+1)):
12     if(B[i] <= B[j]):
13       A[k] = B[i]
14       i = i+1
15     else:
16       A[k] = B[j]
17       j = j-1
18 def merge(A):
19   mergesort(A,0,len(A)-1)
20
21 @profile
22 def mergesort(A,esquerda,direita):
23   if(esquerda<direita):
24     meio = math.floor((esquerda+direita)/2)
25     mergesort(A,esquerda,meio)
26     mergesort(A,meio+1,direita)
27     intercala(A,esquerda,meio,direita)
```

# Apêndice B

# Arquivo ../mergesort/ensaio.py

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

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