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 Heapsort transforma o vetor a ser ordenado em um max-heap chamando a função "Constrói Max Heap"e passando como parâmetro o vetor que ele recebeu. O primeiro elemento do vetor agora é o maior, ou seja, a raiz do "max heap", portanto basta trocá-lo com o elemento da última posição do vetor, pois este é o menor elemento do conjunto. Em cada iteração do laço "for"o tamanho do heap é decrementado, pois o maior item ele já está ordenado e não precisamos mais considerá-lo na ordenação. Devemos fazer a manutenção do heap para que o primeiro item seja novamente o maior, isso é feito chamando-se a função "Max Heapify"dentro do laço "for"do algoritmo Heapsort.

A complexidade da função "Constrói Max Heap"é $teta(n)$ e a complexidade da função "Max Heapify"é $teta(lgn)$, porém como esta se comporta dentro do laço "for"do Heapsort, então é executada n vezes, assim sua complexidade é $n * teta(lgn)$. Dessa forma, temos a seguinte recorrência representada por $T(n)$:

$T(n) = teta(n) + n * teta(n) = teta(n * lgn) + teta(n) = teta(n * lgn)$

Assim, a complexidade de tempo do algoritmo Heapsort é $teta(n * lgn)$.

Capítulo 2

Resultados

2.1 Tabelas

n	comparações	tempo(s)
32	31	0.000612
64	63	0.001465
128	127	0.003014
256	255	0.006684
512	511	0.015407
1024	1023	0.035079
2048	2047	0.077598
4096	4095	0.167752
8192	8191	0.364817

Tabela 2.1: *HeapSort com Vetores Aleatorio*

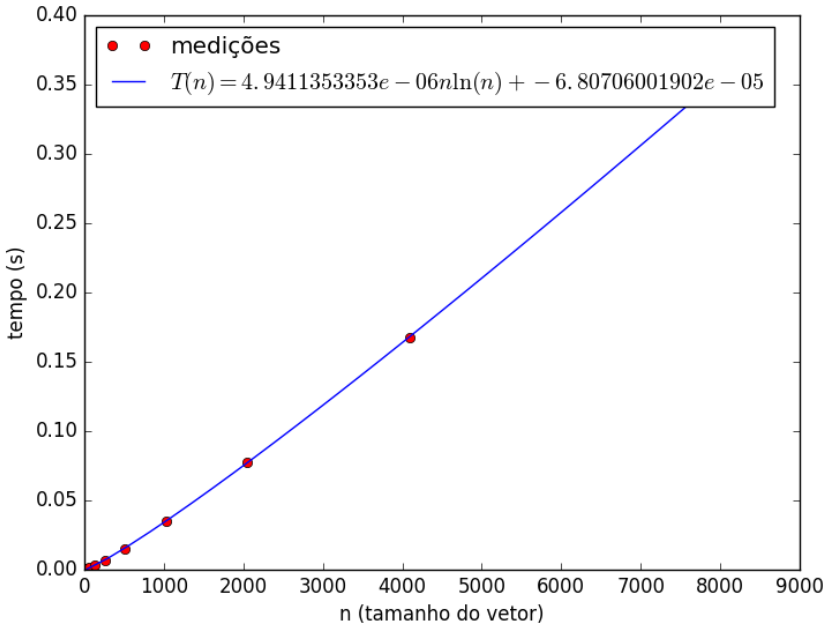

Figura 2.1: *Heapsort com relação ao tempo com vetor aleatório.*

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

$4.9411352252 * 10^{-6} * n * ln(n) - 6.80706001902 * 10^{-5} = 47898.4301$

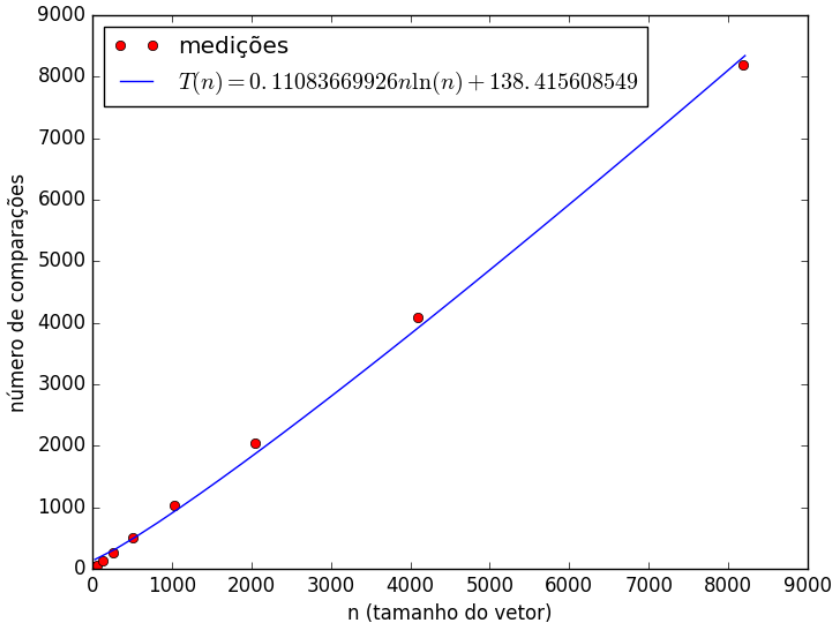

Figura 2.2: *Heapsort com relação ao número de comparações com vetor aleatório.*

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000596
64	63	0.001473
128	127	0.003232
256	255	0.007591
512	511	0.017123
1024	1023	0.039581
2048	2047	0.083045
4096	4095	0.170439
8192	8191	0.399824

Tabela 2.2: *HeapSort com Vetores Crescentes*

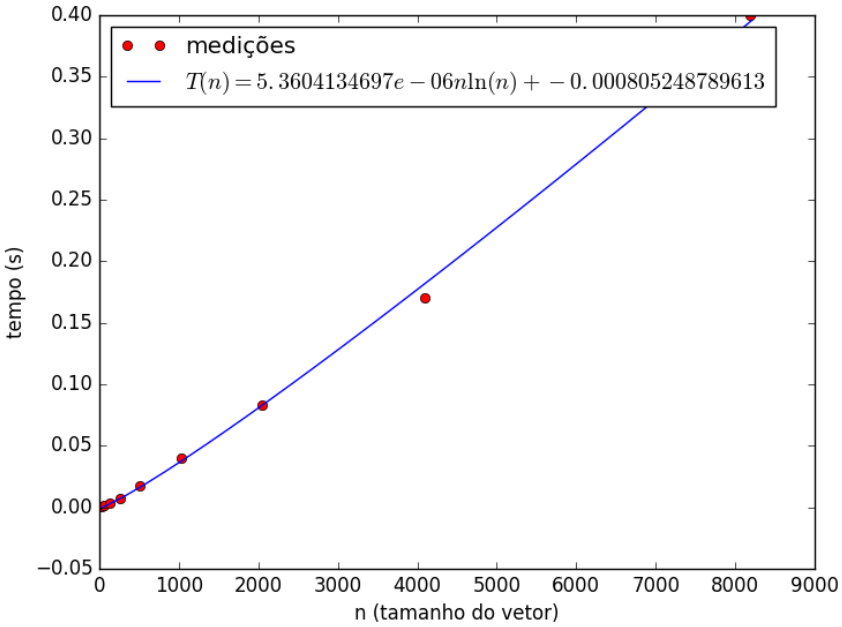

Figura 2.3: *Heapsort com relação ao tempo com vetor crescente.*

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

$5.36041134687 * 10^{-6} * n * ln(n) - 0.008052487 = 50712.7619$

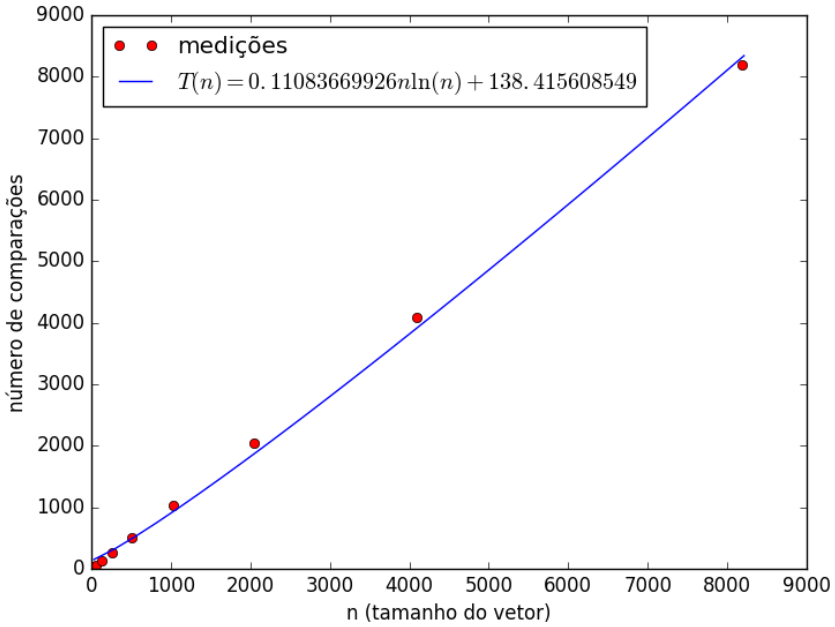

Figura 2.4: *Heapsort com relação ao número de comparações com vetor crescente.*

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000520
64	63	0.001198
128	127	0.002727
256	255	0.006985
512	511	0.014583
1024	1023	0.034831
2048	2047	0.073848
4096	4095	0.171473
8192	8191	0.344057

Tabela 2.3: *HeapSort com Vetores Decrescentes*

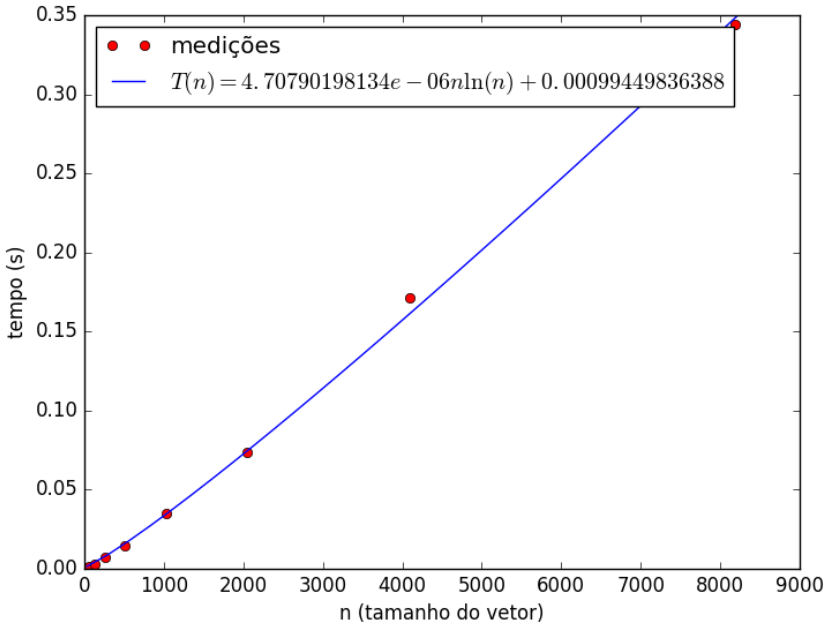

Figura 2.5: *Heapsort com relação ao tempo com vetor decrescente.*

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

$4.70790198134 * 10^{-6} * n * ln(n) - 0.0099449836388 = 46781.51753$

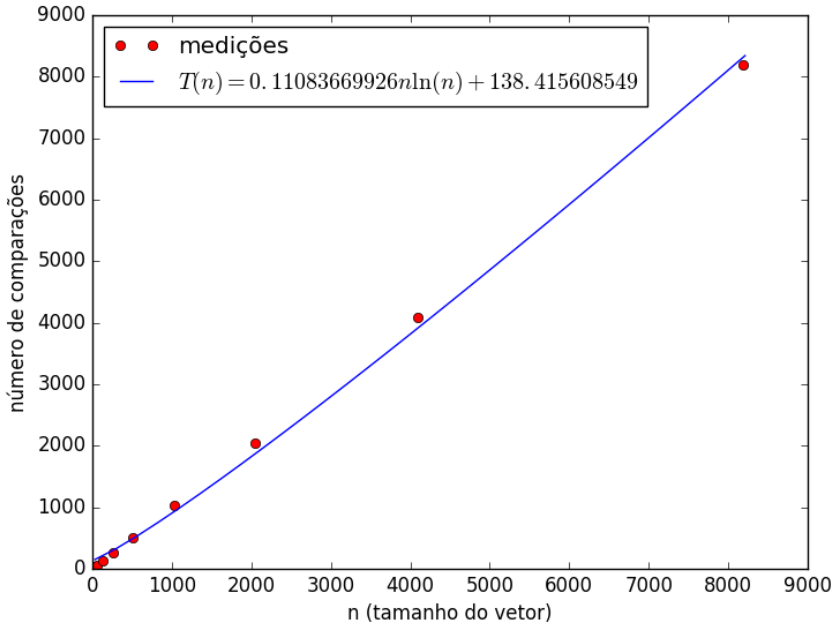

Figura 2.6: *Heapsort com relação ao número de comparações com vetor decrescente.*

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000668
64	63	0.001541
128	127	0.003424

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

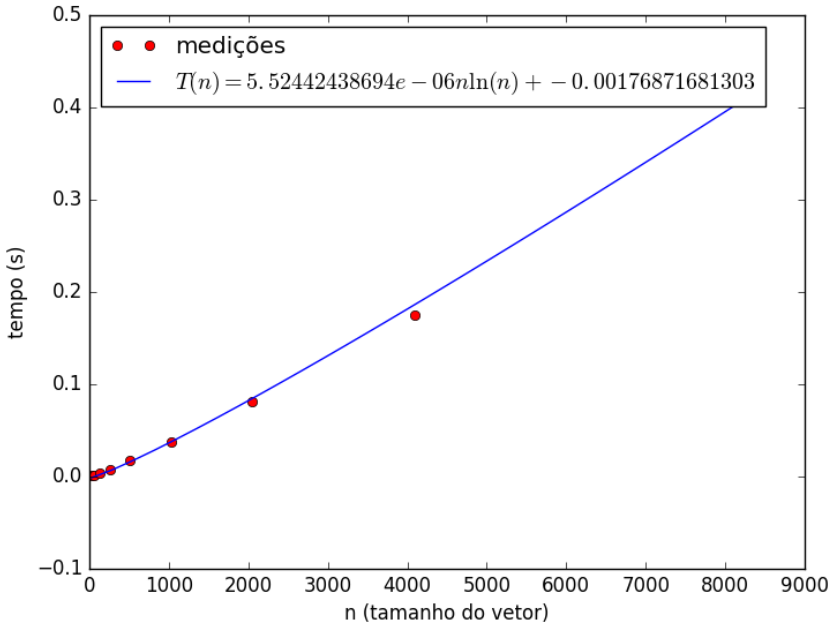

Figura 2.7: *heapsort com relação ao tempo com vetor quase crescente* 10%.

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

$5.52442438694 * 10^{-6} * n * ln(n) - 0.001768716 = 51819.42964$

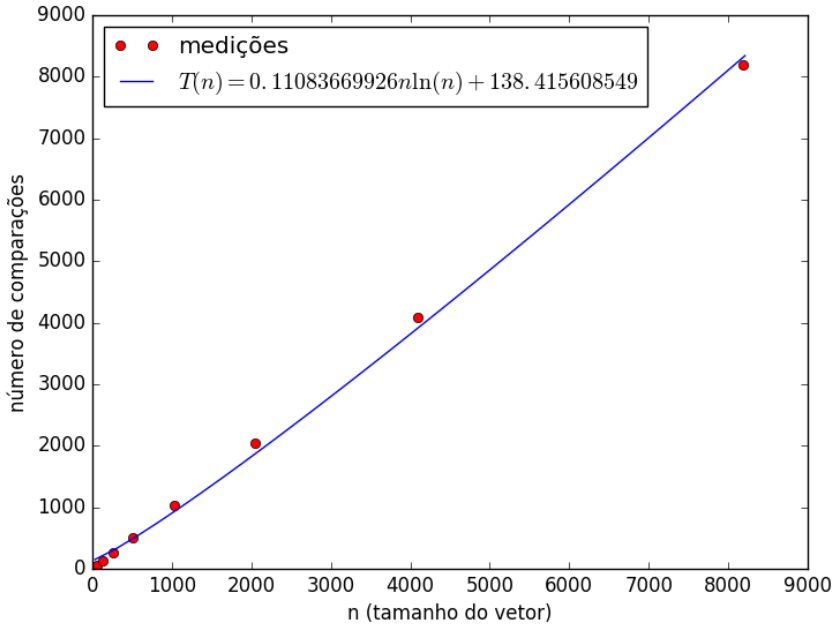

Figura 2.8: *Heapsort com relação ao número de comparações com vetor quase crescente* 10%.

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000621
64	63	0.001409
128	127	0.003303

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

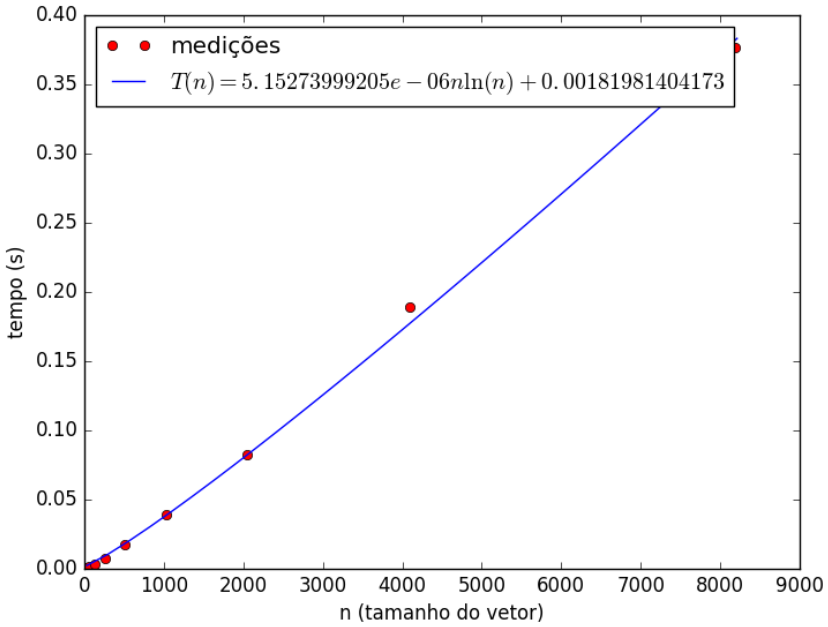

Figura 2.9: *Heapsort com relação ao tempo com vetor quase crescente* 20%.

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

$5.15273999205 * 10^{-6} * n * ln(n) - 0.00181981404173 = 50971.3694$

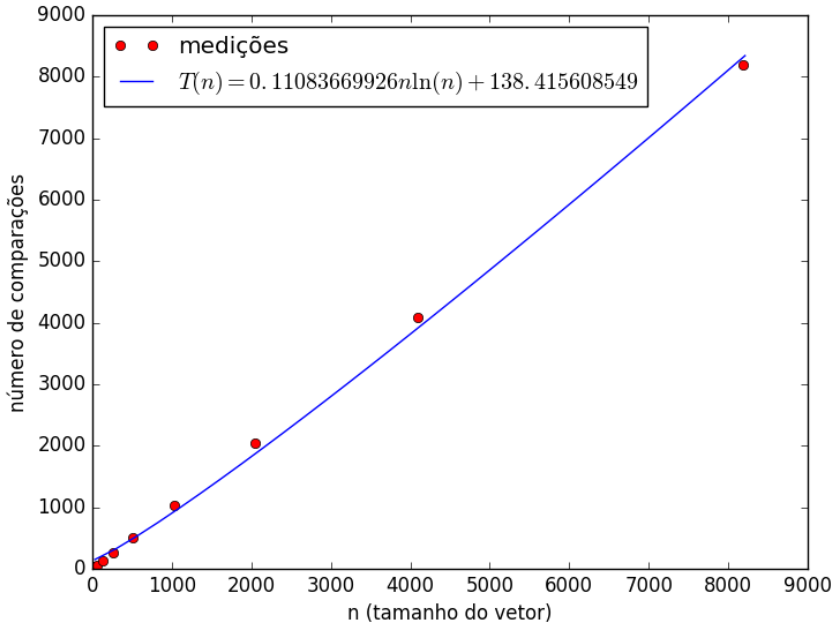

Figura 2.10: *Heapsort com relação ao número de comparações com vetor quase crescente* 20%.

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000600
64	63	0.001371
128	127	0.003106
256	255	0.007629
512	511	0.017727
1024	1023	0.035558
2048	2047	0.080876
4096	4095	0.184995
8192	8191	0.401693

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

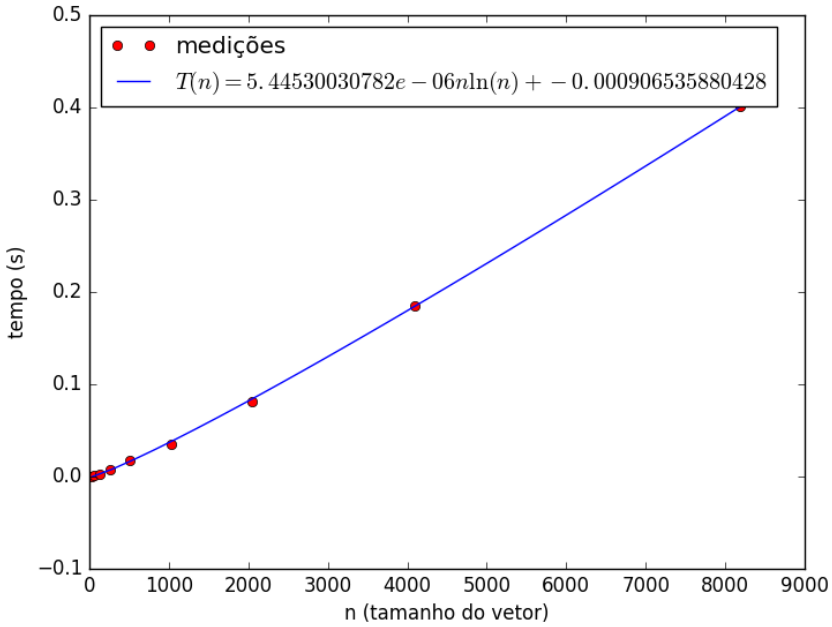

Figura 2.11: *Heapsort com relação ao tempo com vetor quase crescente* 30%.

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

$5.44530030782 * 10^{-6} * n * ln(n) - 0.00090653588 = 51347.63192$

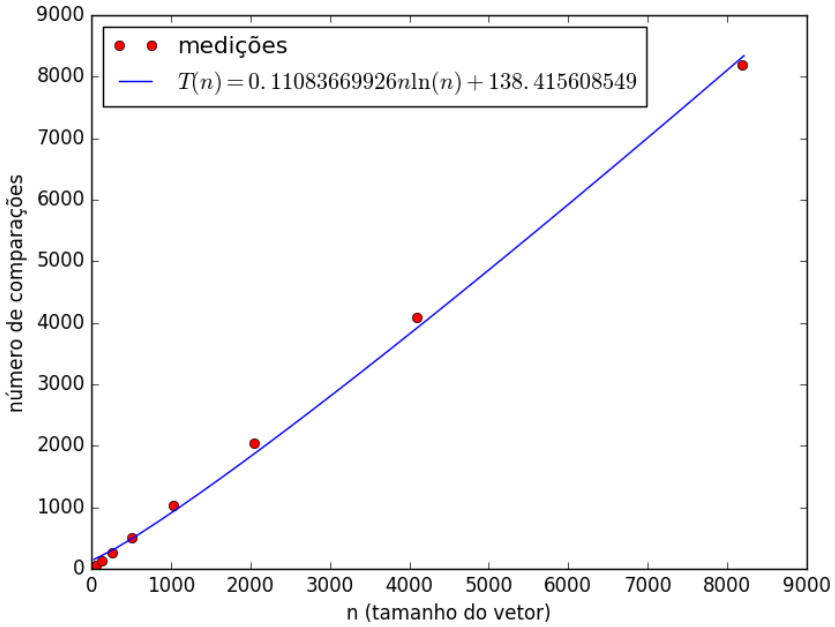

Figura 2.12: *Heapsort com relação ao número de comparações com vetor quase crescente* 30%.

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000596
64	63	0.001341
128	127	0.003476
256	255	0.007358
512	511	0.017912
1024	1023	0.035751
2048	2047	0.079975
4096	4095	0.187078
8192	8191	0.395067

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

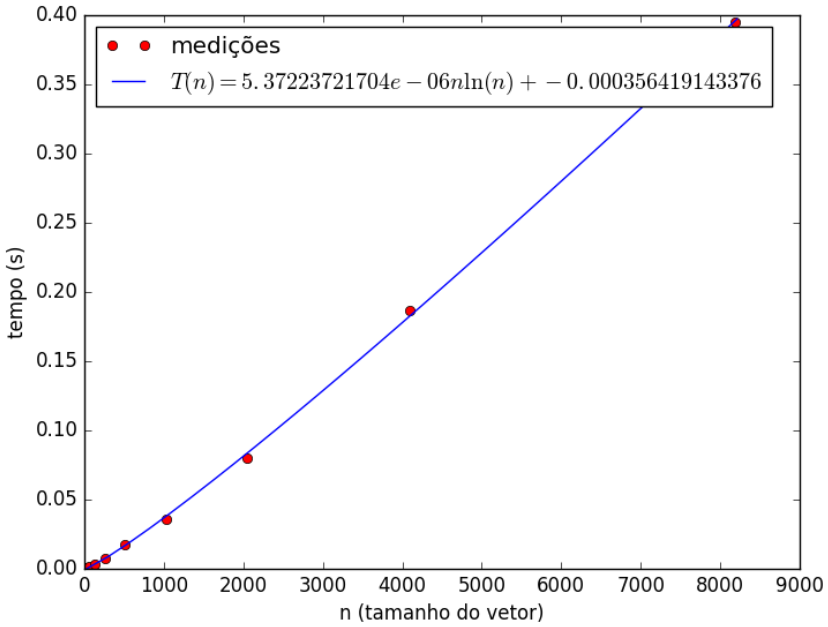

Figura 2.13: *Heapsort com relação ao tempo com vetor quase crescente* 40%.

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

$5.37223721704 * 10^{-6} * n * ln(n) - 0.00035641914 = 50892.30175$

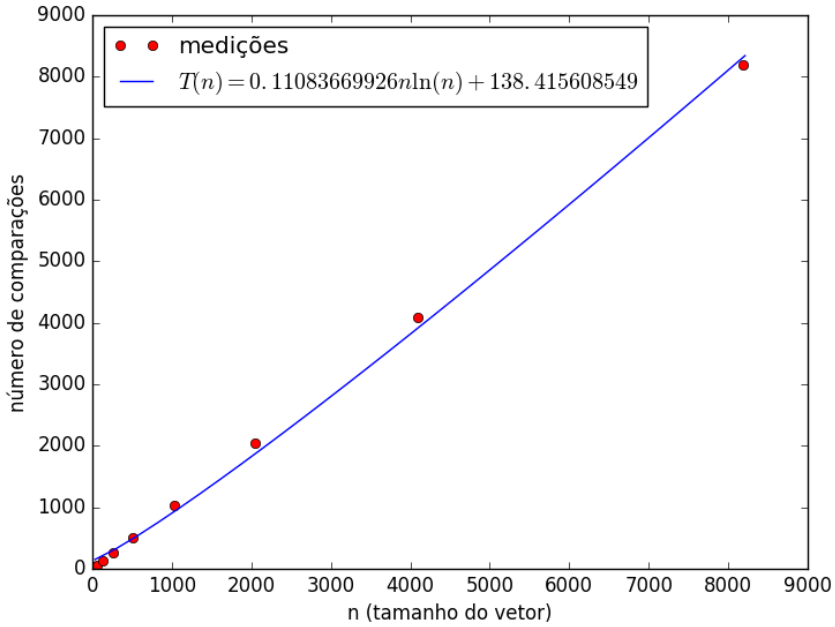

Figura 2.14: *Heapsort com relação ao número de comparações com vetor quase crescente* 40%.

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000589
64	63	0.001354
128	127	0.003127
256	255	0.007598
512	511	0.015462
1024	1023	0.041802
2048	2047	0.085861
4096	4095	0.169554
8192	8191	0.401085

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

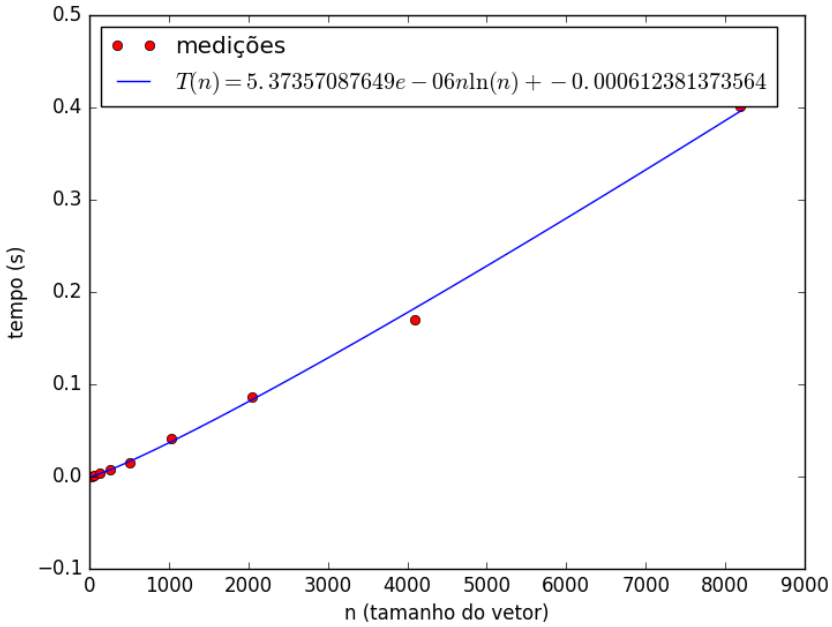

Figura 2.15: *Heapsort com relação ao tempo com vetor quase crescente* 50%.

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

$5.37357087649 * 10^{-6} * n * ln(n) - 0.000612381373 = 50716.91741$

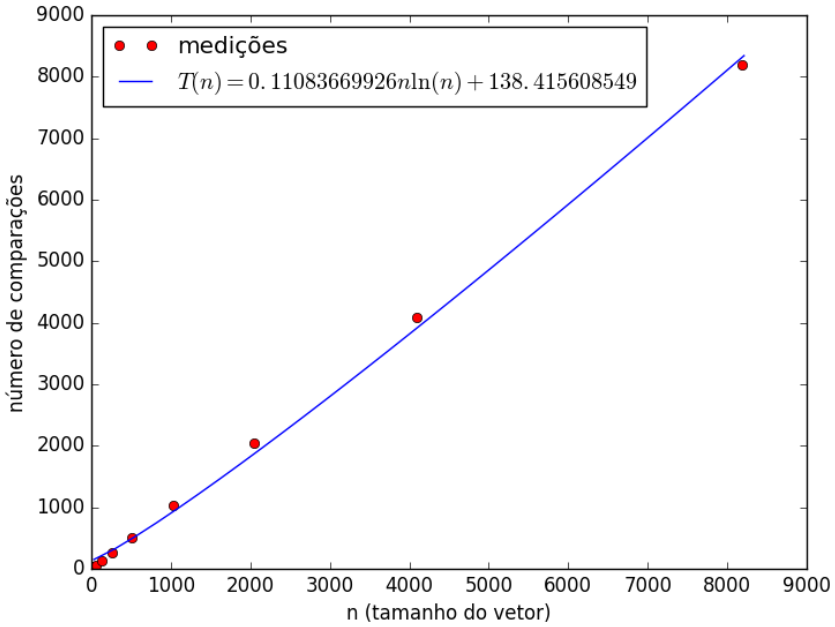

Figura 2.16: *Heapsort com relação ao número de comparações com vetor quase crescente* 50%.

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000530
64	63	0.001288
128	127	0.002709
256	255	0.006940
512	511	0.015185
1024	1023	0.034707
2048	2047	0.070914
4096	4095	0.173265
8192	8191	0.345929

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

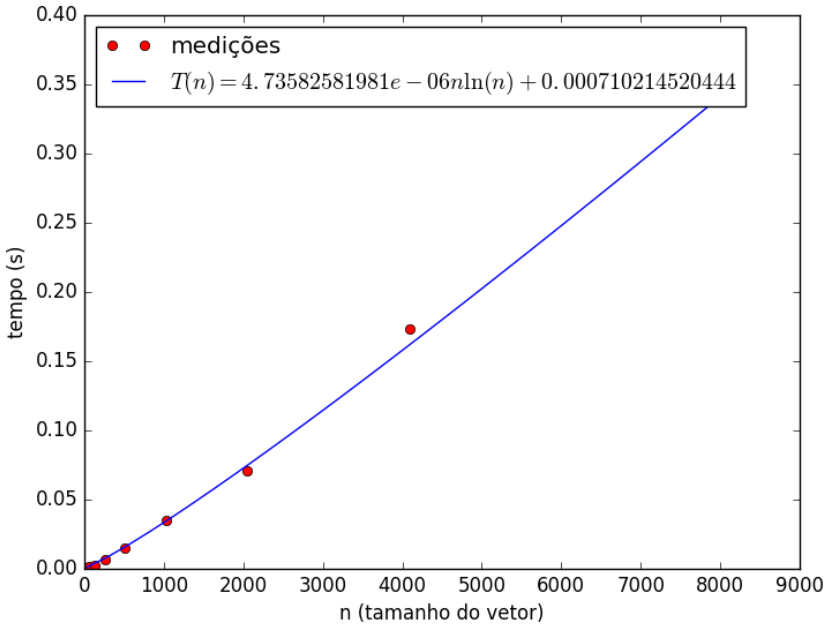

Figura 2.17: *Heapsort com relação ao tempo com vetor quase decrescente* 10%.

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

$4.73582581981 * 10^{-6} * n * ln(n) + 0.00071021452044 = 47293.10524$

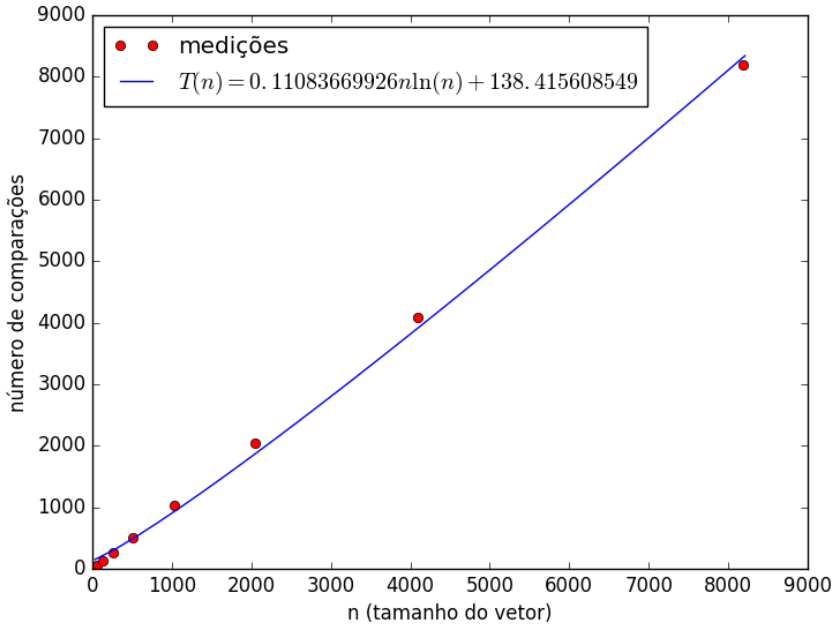

Figura 2.18: *Heapsort com relação ao número de comparações com vetor quase decrescente* 10%.

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000545
64	63	0.001250
128	127	0.002927

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

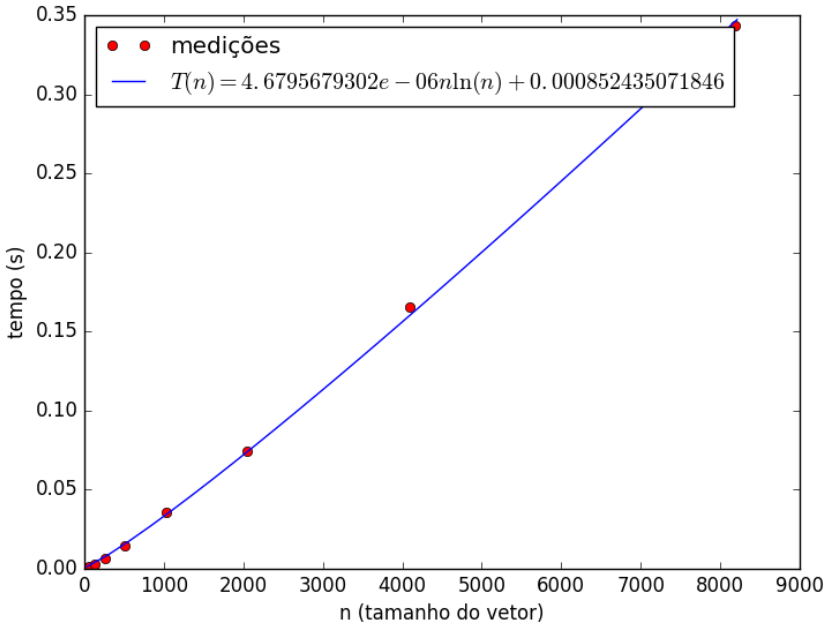

Figura 2.19: *Heapsort com relação ao tempo com vetor quase decrescente* 20%.

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

$4.6795679302 * 10^{-6} * n * ln(n) + 0.00085243507 = 46685.33901$

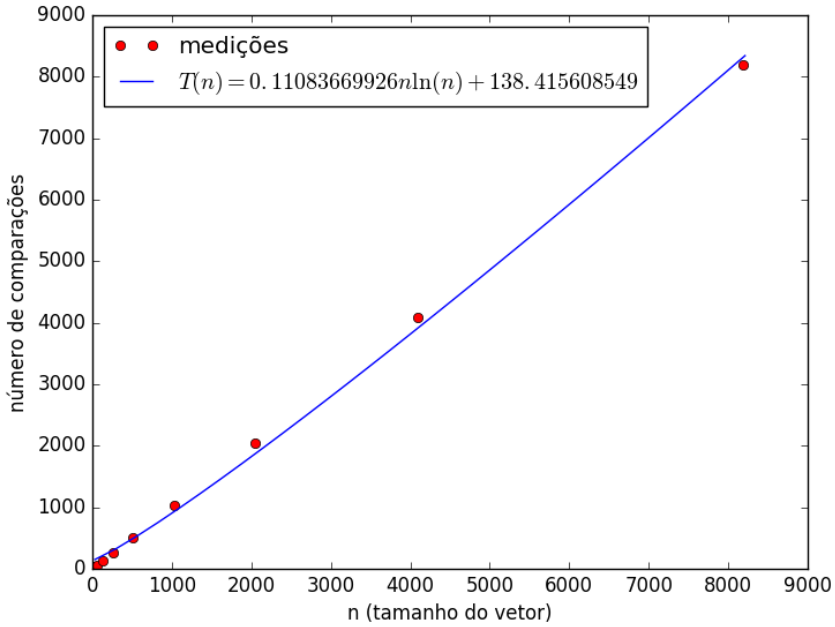

Figura 2.20: *Heapsort com relação ao número de comparações com vetor quase decrescente* 20%.

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000519
64	63	0.001182
128	127	0.002964

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

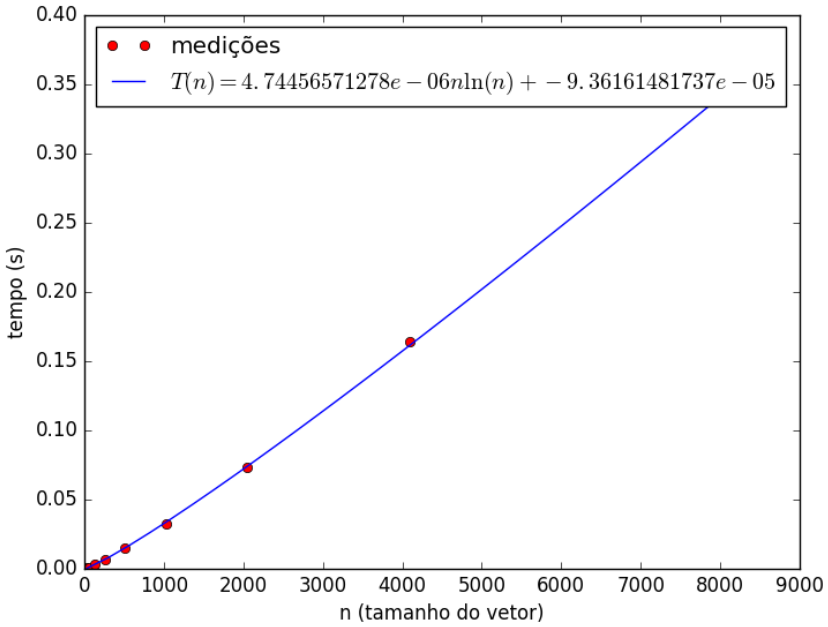

Figura 2.21: *Heapsort com relação ao tempo com vetor quase decrescente* 30%.

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

$4.7445667127 * 10^{-6} * n * ln(n) - 9.361481713 * 10^{-5} = 45481.15823$

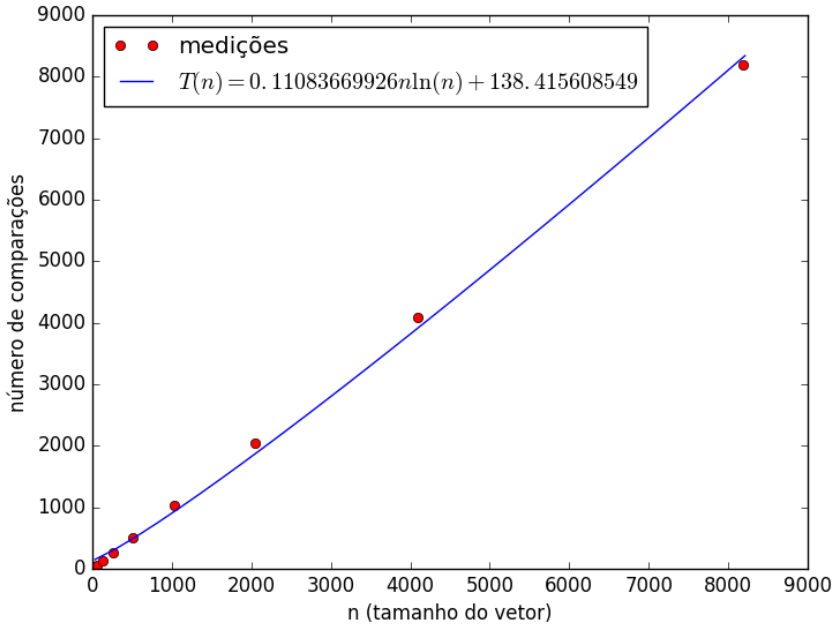

Figura 2.22: *Heapsort com relação ao número de comparações com vetor quase decrescente* 30%.

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000504
64	63	0.001196
128	127	0.002764
256	255	0.006809
512	511	0.013986
1024	1023	0.032507
2048	2047	0.071842
4096	4095	0.161676
8192	8191	0.332576

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

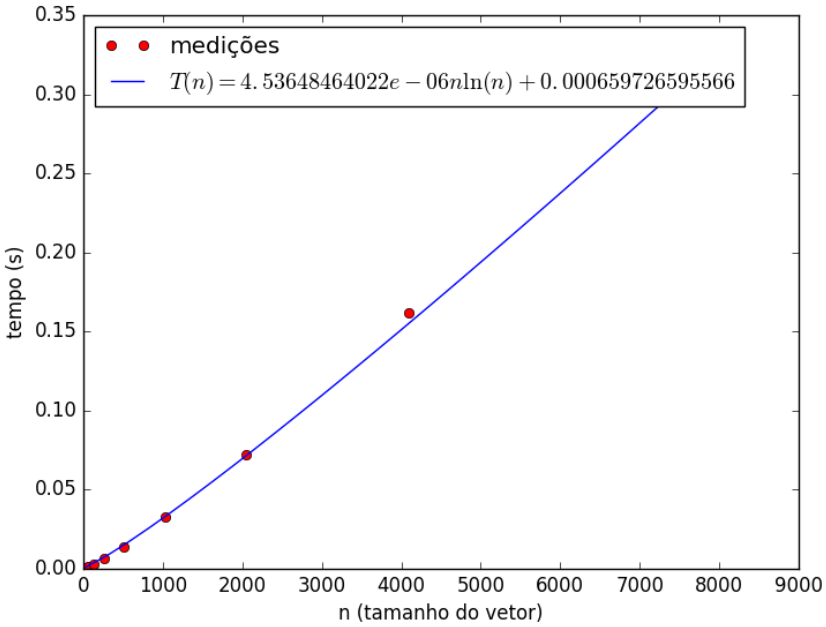

Figura 2.23: *Heapsort com relação ao tempo com vetor quase decrescente* 40%.

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

$4.536484022 * 10^{-6} * n * ln(n) + 0.00065972565 = 43793.40918$

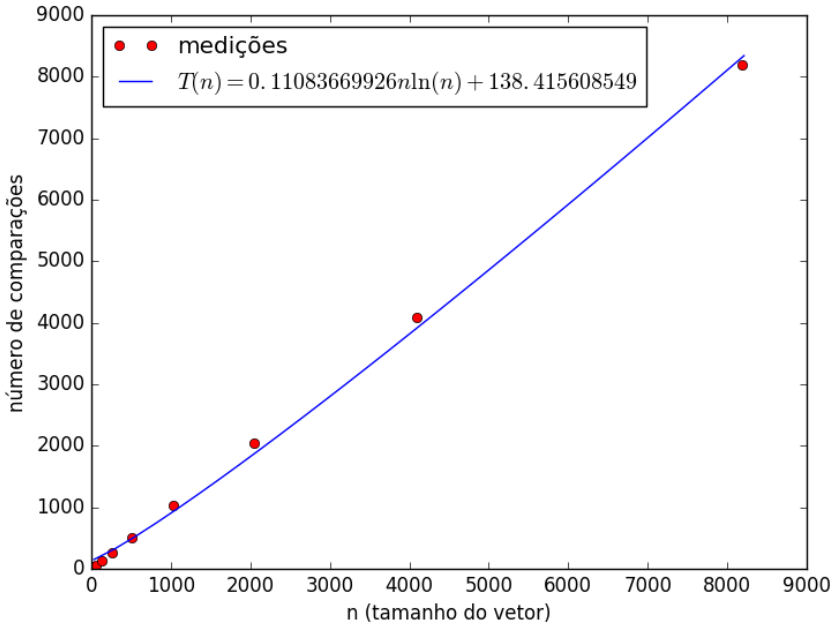

Figura 2.24: *Heapsort com relação ao número de comparações com vetor quase decrescente* 40%.

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

n	comparações	tempo(s)
32	31	0.000539
64	63	0.001257
128	127	0.002960
256	255	0.006867
512	511	0.016310
1024	1023	0.032047
2048	2047	0.081482
4096	4095	0.172628
8192	8191	0.372216

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

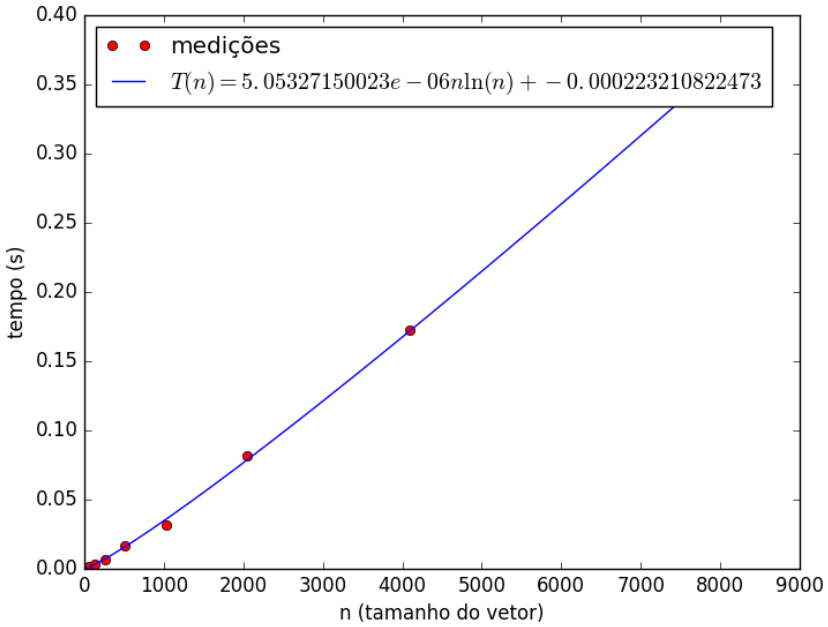

Figura 2.25: *Heapsort com relação ao tempo com vetor quase decrescente* 50%.

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

$5.05327150023 * 10^{-6} * n * ln(n) + 0.00022321082 = 45981.76319$

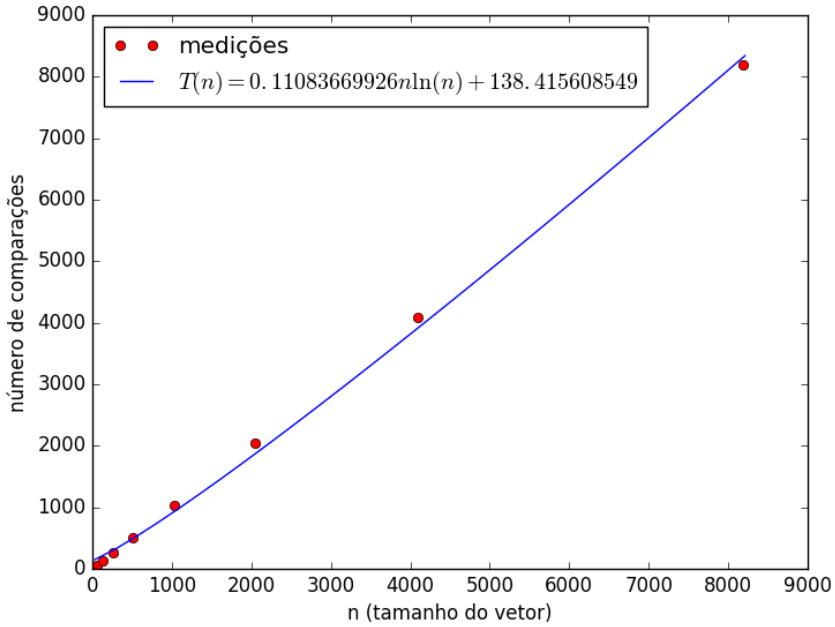

Figura 2.26: *Heapsort com relação ao número de comparações com vetor quase decrescente* 50%.

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

$0.1108367 * n * ln(n) + 138.415608549 = 10558905163$

Apêndice A

Arquivo ../heapsort/heapsort.py

Listagem A.1: ../heapsort/heapsort.py

```
#@profile
def trocaElementos(A,x,y):
    aux = A[y]
    A[y] = A[x]
    A[x] = aux

#@profile
def maxHeapify(A,n,i):
    esquerda = 2*i + 1 #Pq o indice começa de 0
    direita = 2*i +2 #Pq o indice começa de 0

    if esquerda < n and A[esquerda] > A[i]:
        maior = esquerda
    else:
        maior = i
    if direita < n and A[direita] > A[maior]:
        maior = direita
    if maior!=i:
        trocaElementos(A,i,maior)
        maxHeapify(A,n,maior)

#@profile
def constroiMaxHeap(A,n):
    for i in range(n // 2, -1, -1):
        maxHeapify(A,n,i)


@profile
def heapSort(A):
    n = len(A)
    constroiMaxHeap(A,n)
    m = n
    for i in range((n-1), 0, -1):
        trocaElementos(A,0,i)
        m = m - 1
        maxHeapify(A,m,0)

#lista = [13,46,17,34,41,15,14,23,30,21,10,12,21]
#heapSort(lista, 13)
#print(lista)

```

```

#heapSort([i for i in range(524288)])
```

Apêndice B

Arquivo ../heapsort/ensaio.py

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

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