UNIVERSIDADE FEDERAL DO TOCANTINS
CÂMPUS UNIVERSITÁRIO DE PALMAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO DE PROJETO E ANÁLISE DE ALGORITMOS

ALGORITMOS DE ORDENAÇÃO, UM BREVE ESTUDO

JOÃO PEDRO SILVA CUNHA
JULIO CEZAR NOLASCO
LUIZ FERNANDO PIRES KOZAK
WILLIAN DOS SANTOS ALVES

PALMAS (TO)
2023

RESUMO

Este trabalho acadêmico apresenta uma análise comparativa de algoritmos de ordenação, com o objetivo de identificar as principais características de cada um e determinar seu desempenho em diferentes cenários. São abordados algoritmos clássicos, como Bubble Sort, Selection Sort, Insertion Sort, Merge Sort e Quick Sort, bem como algoritmos mais recentes, como Tim Sort e Radix Sort. Para a avaliação de desempenho, são utilizadas métricas como tempo de execução e uso de memória, considerando diferentes tamanhos de entrada e tipos de dados. Os resultados obtidos permitem uma compreensão mais profunda dos algoritmos de ordenação e ajudam a selecionar o mais adequado para cada situação.

Palavra-chave: LaTeX. UFTeX. Relatório da Disciplina. Algoritmos de Ordenação. Trabalho Academico.

ABSTRACT

This academic work presents a comparative analysis of sorting algorithms, with the objective of identifying the main characteristics of each one and determining their performance in different scenarios. Classic algorithms such as Bubble Sort, Selection Sort, Insertion Sort, Merge Sort and Quick Sort are covered, as well as more recent algorithms such as Tim Sort and Radix Sort. For performance evaluation, metrics such as execution time and memory usage are used, considering different input sizes and data types. The results obtained allow a deeper understanding of the sorting algorithms and help to select the most suitable one for each situation.

Keywords: LaTeX. UFTeX. Subject Report. Sorting Algorithms. Academic work.

SUMÁRIO

1 INTRODUÇÃO

A ordenação de dados é um processo fundamental em muitas áreas da ciência da computação e da engenharia. Existem vários algoritmos de ordenação disponíveis, cada um com diferentes desempenhos em termos de tempo e espaço. Neste trabalho, iremos realizar uma análise comparativa de diferentes algoritmos de ordenação, avaliando seu desempenho em relação ao número de elementos a serem ordenados, bem como a complexidade de tempo e espaço. Ao final deste trabalho, espera-se que o leitor tenha uma compreensão dos diferentes algoritmos de ordenação e seja capaz de perceber que alguns algoritmos são mais adequado que outros dependendo do seu caso específico de uso.

1.1 Métodos

Para realizar tal análise por meio comparações vamos levar alguns fatores para determinar resultados em nosso experimento. Para tanto vamos considerar:

- Tempo de execução.
- Quantidade de comparações.
- Quantidade de trocas.

Como aspectos para obter resultados que vão variar dependendo do algoritmo de ordenação em questão e o tamanho das entradas. Com isso temos de utilizar algumas ferramentas para melhor absorção dos resultados obtidos, à título de exemplo gnuplot para nós permitir uma melhor visualização das soluções encontradas tendo em vista os parâmetros tempo e tamanho da entrada que serão os eixos dos nossos gráficos.

1.2 Entradas

As nossas entradas serão vetores de valores inteiros gerados de forma aleátoria e de tamanhos variados com a finalidade de captar variados comportamentos dos algoritmos em situações distintas em que são submetidos. Desssa forma vamos ler esses vetores sendo os tamanhos definidos por:

- 10^3.
- 10^4.
- 10^5.
- $[2, ..., 9]$x10^5.
- 10^6.

E para cada tamno de entrada vamos ordenar esses vetores com os algoritmos de ordenação de forma:

- Crescente.
- Descrescente.
- Aleátoria.

Ou seja, para cada tamanho de entrada em apenas um algoritmos vamos gerar três gráficos como artefatos visto que serão feita três ordenações tendo cada gráfico aquela estrutura apresentada anteriormente de tempo X tamanho da entrada. **Observação:** No decorrer do relatório você pode se deparar com resultados indeterminados nas tabelas, isso acontece pois não foi possível mensurar as propriedas do algoritmo dado tamanha entrada, nesse caso os mesmos não aparecem na tabela de resultados/comparativos no fim de cada seção.

1.3 Estrutura do experimento

Visto que ja se tem conhecimento sobre as abordagens que serão feitas em particular de cada algoritmo de ordenação, vamos agora determinar uma estrutura a fim de padronizar de forma organizada os experimentos a serem feitos neste trabalho. Com isso vamos:

1. Aplicar os métodos e compreender cada caso prório de cada algoritmo.
2. Realizar comparações entre os mesmos.
3. Tomar conclusões e resultados

Tudo isso submetido em uma máquina q dispõe dê um processador Intel(R) Core(TM) i5-1035G1 e 8.00GB de memória RAM usando o sistema operacional Linux Mint.

2 ALGORITMOS DE ORDENAÇÃO

2.1 Bubble Sort

```
1  def bubble_sort(lista):
2      comp, trocas = 0, 0
3      num = len(lista)
4
5      for i in range(num):
6          for j in range(0, num - i - 1):
7              comp += 1
8              if lista[j] > lista[j + 1]:
9                  temp = lista[j]
10                 lista[j] = lista[j + 1]
11                 lista[j + 1] = temp
12                 trocas += 1
13
14
15     return comp, trocas
```

Essa função representa respectivamente o nosso algoritmo de ordenação por flutuação. A ideia é percorrer o vetor diversas vezes, e a cada passagem fazer flutuar para o topo o maior elemento da sequência. Agora vamos submeter esse algoritmo à vários tamanhos de entradas e considerando essas entradas ordenadas de forma crescente, decrescente e aleatória.

2.1.1 Ordenação Crescente

Para essa seção vamos assumir uma lista ja ordenada em ordem crescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 1 – Resultado.

Saída	
Tempo (Segundos)	0.03885
N° de Comparações	500500
N° de Trocas	0
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 2 – Resultado.

Saída	
Tempo (Segundos)	3.8275
N° de Comparações	50005000
N° de Trocas	0
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 3 – Resultado.

Saída	
Tempo (Segundos)	408.73872
N° de Comparações	5000050000
N° de Trocas	0
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 4 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	0
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 5 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	0
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 6 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	0
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 7 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	0
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 8 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	0
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 9 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	0
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 10 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	0
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 11 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	0
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 12 – Resultado.

Saída	
Tempo (Segundos)	Indeterminado
N° de Comparações	Indeterminado
N° de Trocas	0
Comprimento do Vetor	1 000 000

2.1.2 Ordenação Decrescente

Para essa seção vamos assumir uma lista ja ordenada em ordem decrescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 13 – Resultado.

Saída	
Tempo (Segundos)	0.0864
N° de Comparações	500500
N° de Trocas	500500
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 14 – Resultado.

Saída	
Tempo (Segundos)	8.9257
N° de Comparações	50005000
N° de Trocas	50005000
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 15 – Resultado.

Saída	
Tempo (Segundos)	932.0820
N° de Comparações	5000050000
N° de Trocas	5000050000
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 16 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 17 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 18 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 19 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 20 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 21 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 22 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 23 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 24 – Resultado.

Saída	
Tempo (Segundos)	Indeterminado
N° de Comparações	Indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	1 000 000

2.1.3 Ordenação Aleatória

Para essa seção vamos assumir uma lista de certa forma desordenada de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 25 – Resultado.

Saída	
Tempo (Segundos)	0.07875
N° de Comparações	500500
N° de Trocas	243683
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 26 – Resultado.

Saída	
Tempo (Segundos)	6.5352
N° de Comparações	50005000
N° de Trocas	24814760
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 27 – Resultado.

Saída	
Tempo (Segundos)	743.5727
N° de Comparações	5000050000
N° de Trocas	2504203449
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 28 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 29 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 30 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 31 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 32 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 33 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 34 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 35 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação bolha, obtemos como resultado:

Tabela 36 – Resultado.

Saída	
Tempo (Segundos)	Indeterminado
N° de Comparações	Indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	1 000 000

2.1.4 Comparativos

Tabela 37 – Resultados Comparativos.

Comparações				
Ordenação	Comprimento	Tempo (s)	N° Comparações	N° Trocas
Crescente	1000	0.03885	500500	0
	10000	3.8275	50005000	0
	100000	408.73872	5000050000	0
Decrescente	1000	0.0864	500500	500500
	10000	8.9257	50005000	50005000
	100000	932.0820	5000050000	5000050000
Aleatória	1000	0.0787	500500	243683
	10000	6.5352	50005000	24814760
	100000	743.5727	5000050000	2504203449

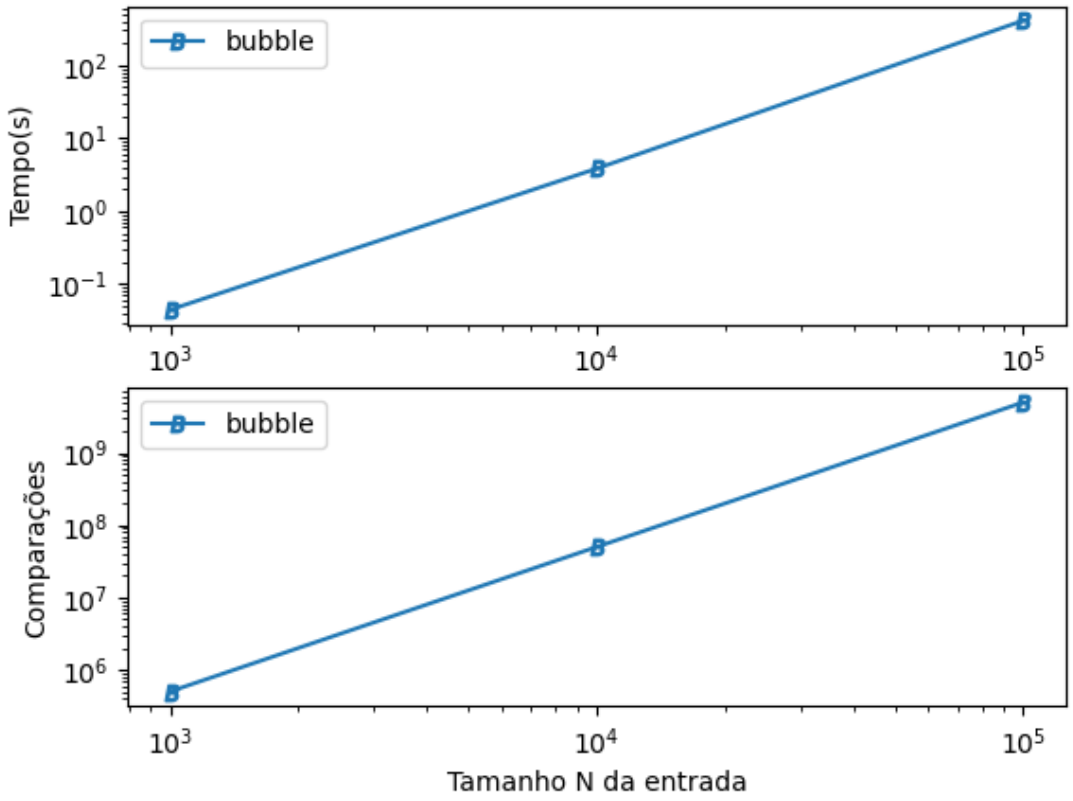

Figura 1 – Representação do comportamento do algoritmo de ordenação bolha com uma entradas de até 100000 ordenadas de forma crescente

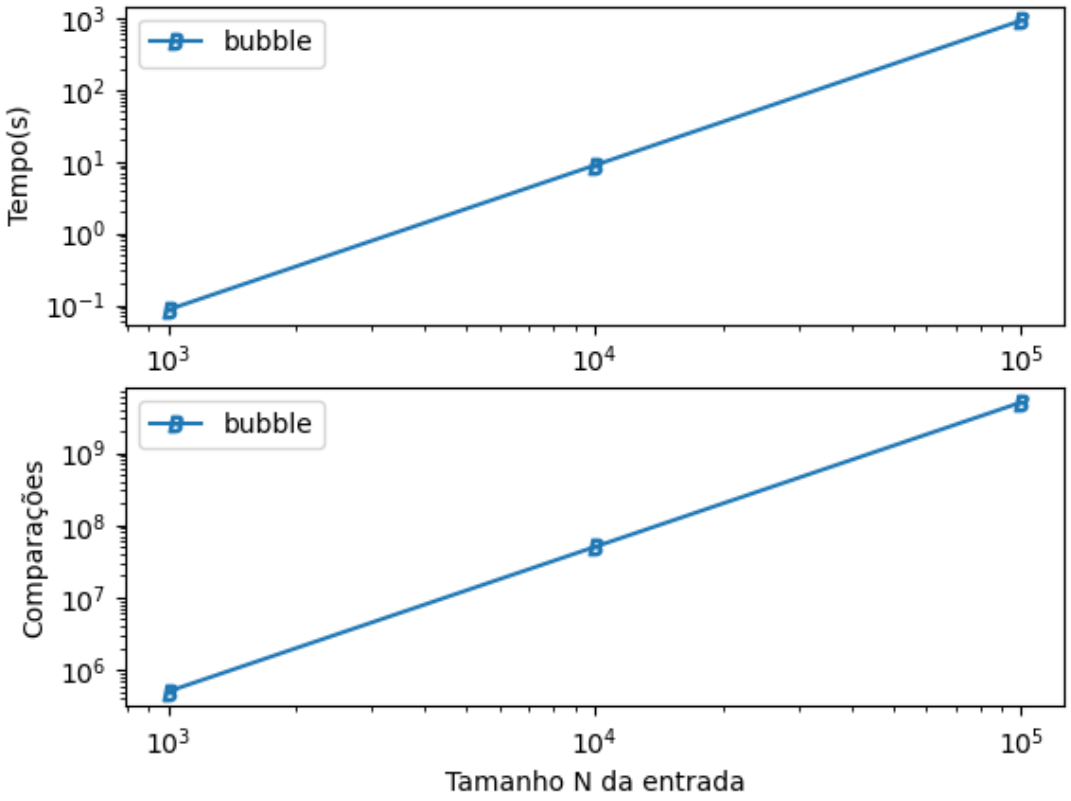

Figura 2 – Representação do comportamento do algoritmo de ordenação bolha com uma entradas de até 100000 ordenadas de forma decrescente

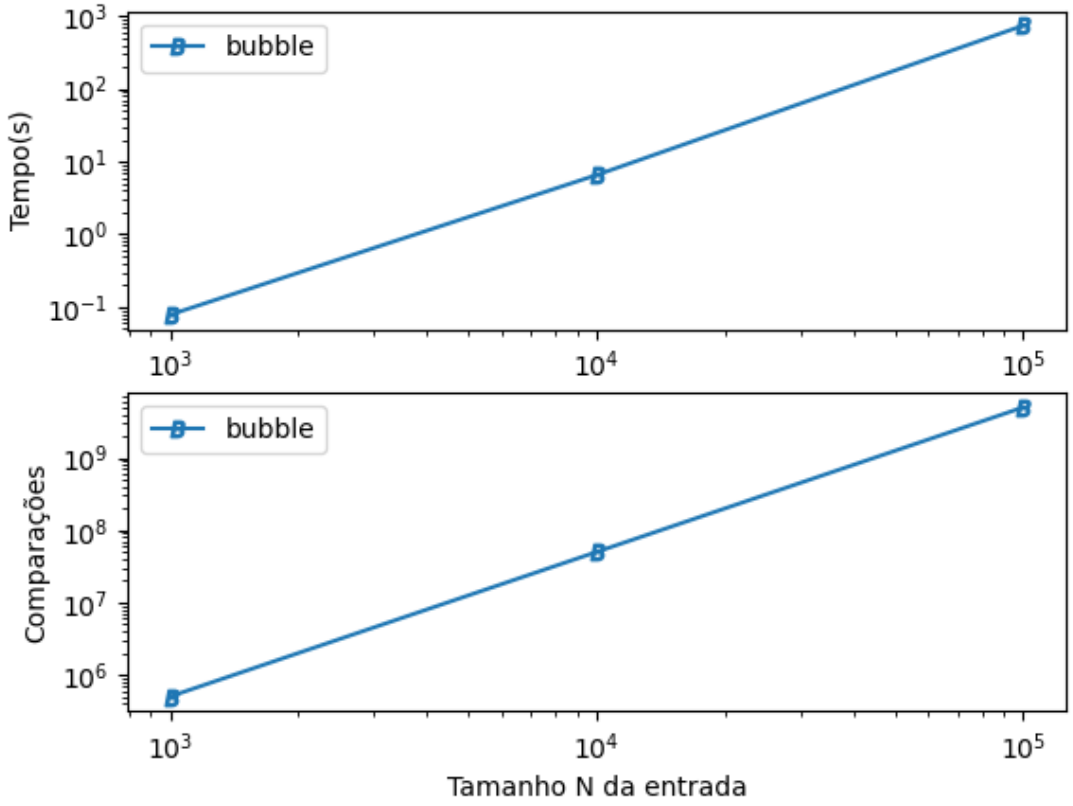

Figura 3 – Representação do comportamento do algoritmo de ordenação bolha com uma entradas de até 100000 desordenadas

2.2 Insert Sort

```
1  def insertion_sort(vetor):
2    comp, trocas = 0, 0
3    n = len(vetor)
4
5    for i in range(1, n):
6      marcado = vetor[i]
7      j = i - 1
8      while j >= 0 and marcado < vetor[j]:
9        comp += 1
10       vetor[j + 1] = vetor[j]
11       j -= 1
12       trocas += 1
13
14     vetor[j + 1] = marcado
15     trocas += 1
16
17   return  comp, trocas
```

Essa função representa respectivamente o nosso algoritmo de ordenação por inserção. O algoritmo de ordenação por inserção é um método simples de ordenação que percorre uma lista de elementos e, em cada passo, insere o elemento atual em sua posição correta dentro da porção já ordenada da lista. Ele continua esse processo até que todos os elementos estejam na posição correta.

2.2.1 Ordenação Crescente

Para essa seção vamos assumir uma lista ja ordenada em ordem decrescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 38 – Resultado.

Saída	
Tempo (Segundos)	0.0001
N° de Comparações	0
N° de Trocas	1000
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 39 – Resultado.

Saída	
Tempo (Segundos)	0.0012
N° de Comparações	0
N° de Trocas	10000
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 40 – Resultado.

Saída	
Tempo (Segundos)	0.0244
N° de Comparações	0
N° de Trocas	100000
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 41 – Resultado.

Saída	
Tempo (Segundos)	0.0250
N° de Comparações	0
N° de Trocas	200000
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 42 – Resultado.

Saída	
Tempo (Segundos)	0.03821
N° de Comparações	0
N° de Trocas	300000
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 43 – Resultado.

Saída	
Tempo (Segundos)	0.0493
N° de Comparações	0
N° de Trocas	400000
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 44 – Resultado.

Saída	
Tempo (Segundos)	0.0625
N° de Comparações	0
N° de Trocas	500000
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 45 – Resultado.

Saída	
Tempo (Segundos)	0.0762
N° de Comparações	0
N° de Trocas	600000
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 46 – Resultado.

Saída	
Tempo (Segundos)	0.0883
N° de Comparações	0
N° de Trocas	700000
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 47 – Resultado.

Saída	
Tempo (Segundos)	0.0991
N° de Comparações	0
N° de Trocas	800000
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 48 – Resultado.

Saída	
Tempo (Segundos)	0.1142
N° de Comparações	0
N° de Trocas	900000
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 49 – Resultado.

Saída	
Tempo (Segundos)	0.1243
N° de Comparações	0
N° de Trocas	1000000
Comprimento do Vetor	1 000 000

2.2.2 Ordenação Decrescente

Para essa seção vamos assumir uma lista ja ordenada em ordem decrescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 50 – Resultado.

Saída	
Tempo (Segundos)	0.0854
N° de Comparações	500500
N° de Trocas	501500
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 51 – Resultado.

Saída	
Tempo (Segundos)	6.6063
N° de Comparações	50005000
N° de Trocas	50015000
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 52 – Resultado.

Saída	
Tempo (Segundos)	688.6565
N° de Comparações	5000050000
N° de Trocas	5000150000
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 53 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 54 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 55 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 56 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 57 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 58 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 59 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 60 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 61 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	1 000 000

2.2.3 Ordenação Aleatória

- Vetor de 10^3 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 62 – Resultado.

Saída	
Tempo (Segundos)	0.0341
N° de Comparações	239119
N° de Trocas	240119
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 63 – Resultado.

Saída	
Tempo (Segundos)	3.5106
N° de Comparações	25089685
N° de Trocas	25099685
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 64 – Resultado.

Saída	
Tempo (Segundos)	335.4530
N° de Comparações	2494527726
N° de Trocas	2494627726
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 65 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 66 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 67 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 68 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 69 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 70 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 71 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 72 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação insert, obtemos como resultado:

Tabela 73 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	1 000 000

2.2.4 Comparatvios

Tabela 74 – Resultados Comparativos.

Comparações				
Ordenação	Comprimento	Tempo (s)	N° Comparações	N° Trocas
Crescente	1000	0.0001	0	1000
	10000	0.0012	0	10000
	100000	0.0244	0	100000
	200000	0.0250	0	200000
	300000	0.03821	0	300000
	400000	0.0493	0	400000
	500000	0.0625	0	500000
	600000	0.0762	0	600000
	700000	0.0883	0	700000
	800000	0.0991	0	800000
	900000	0.1142	0	900000
	1000000	0.1243	0	1000000
Decrescente	1000	0.0854	500500	501500
	10000	6.6063	50005000	50015000
	100000	688.6565	5000050000	5000150000
Aleatória	1000	0.0341	239119	240119
	10000	3.5106	25089685	25099685
	100000	335.4530	2494527726	2494627726

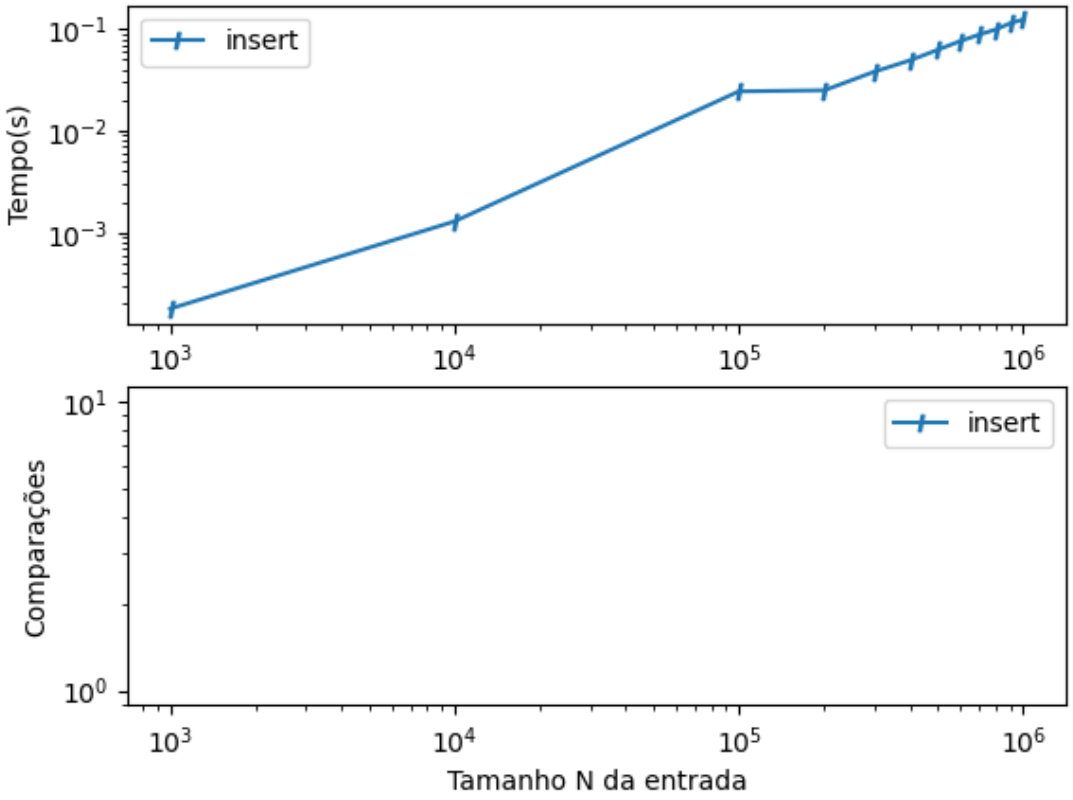

Figura 4 – Representação do comportamento do algoritmo de ordenação por inserção com uma entrada de tamanho 1000000 ordenadas de forma crescente

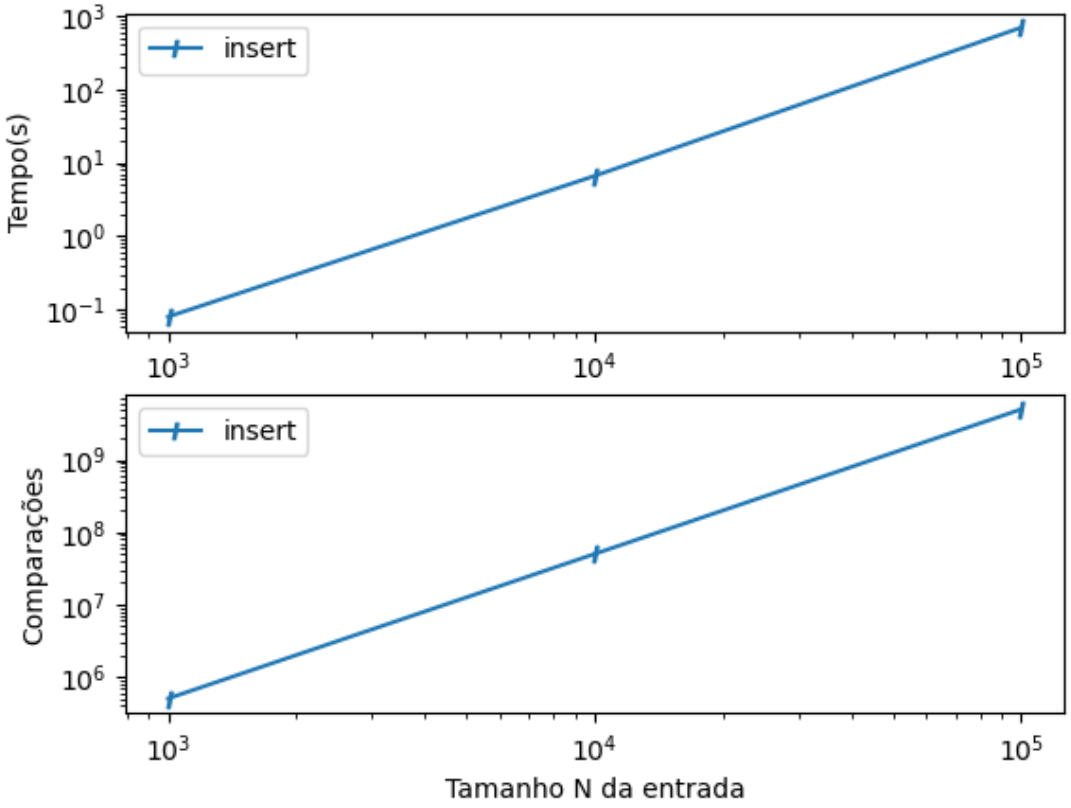

Figura 5 – Representação do comportamento do algoritmo de ordenação por inserção com uma entrada de tamanho 100000 ordenadas de forma decrescente

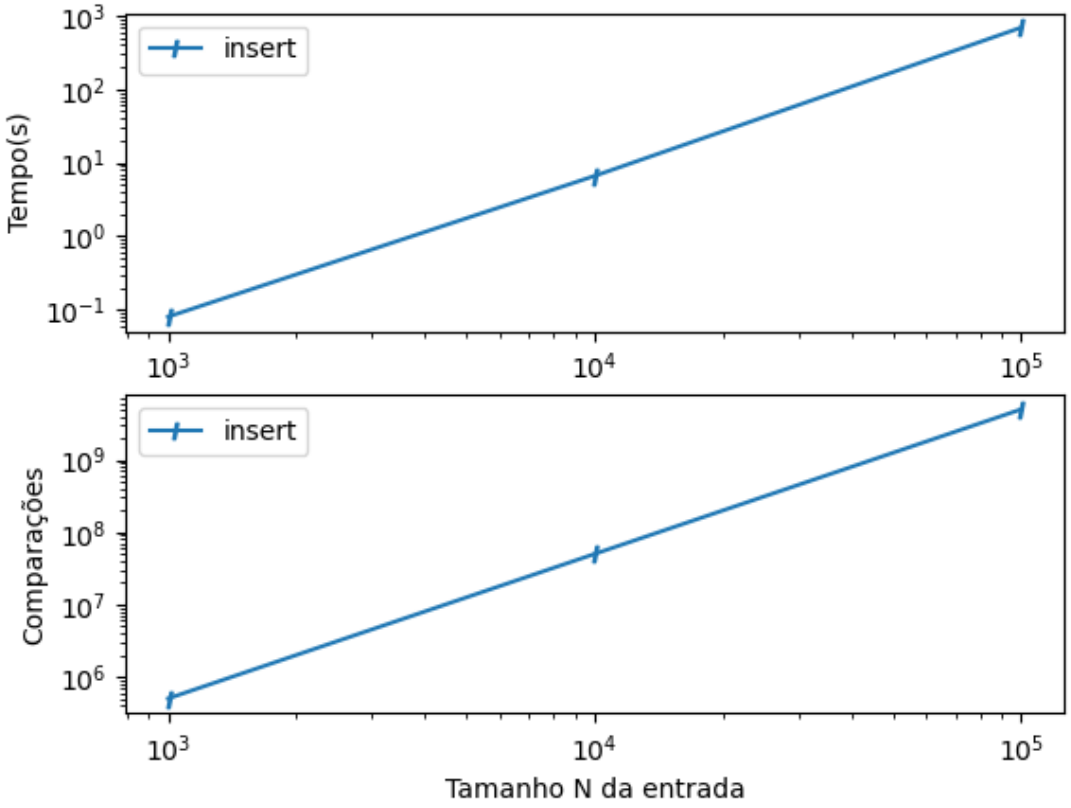

Figura 6 – Representação do comportamento do algoritmo de ordenação por inserção com uma entrada de tamanho 100000 desordenadas

2.3 Merge Sort

```
1  def merge_sort(vetor):
2    comp, trocas = 0, 0
3    if len(vetor) > 1:
4      divisao = len(vetor) // 2
5      esquerda = vetor[:divisao].copy()
6      direita = vetor[divisao:].copy()
7
8      merge_sort(esquerda)
9      merge_sort(direita)
10     i = j = k = 0
11
12     while i < len(esquerda) and j < len(direita):
13       if esquerda[i] < direita[j]:
14         comp += 1
15         vetor[k] = esquerda[i]
16         i += 1
17       else:
18         comp += 1
19         vetor[k] = direita[j]
20         j += 1
21       k += 1
22       trocas += 1
23
24     while i < len(esquerda):
25       comp += 1
26       vetor[k] = esquerda[i]
27       i += 1
28       k += 1
29     while j < len(direita):
30       comp += 1
31       vetor[k] = direita[j]
32       j += 1
33       k += 1
34       trocas += 1
35
36   return comp, trocas
```

Essa função representa respectivamente o nosso algoritmo de ordenação merge sort. O algoritmo ordenação Merge Sort é um método de ordenação que divide a lista em duas metades, recursivamente ordena cada metade e, em seguida, combina as duas metades em uma única lista ordenada. Ele usa o conceito de ”dividir para conquistar”para realizar a ordenação de forma eficiente.

2.3.1 Ordenação Crescente

Para essa seção vamos assumir uma lista ja ordenada em ordem crescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 75 – Resultado.

Saída	
Tempo (Segundos)	0.0024
N° de Comparações	1001
N° de Trocas	1001
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 76 – Resultado.

Saída	
Tempo (Segundos)	0.0404
N° de Comparações	10001
N° de Trocas	10001
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 77 – Resultado.

Saída	
Tempo (Segundos)	0.3087
N° de Comparações	100001
N° de Trocas	100001
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 78 – Resultado.

Saída	
Tempo (Segundos)	0.6490
N° de Comparações	200001
N° de Trocas	200001
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 79 – Resultado.

Saída	
Tempo (Segundos)	1.0007
N° de Comparações	300001
N° de Trocas	300001
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 80 – Resultado.

Saída	
Tempo (Segundos)	1.4187
N° de Comparações	400001
N° de Trocas	400001
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 81 – Resultado.

Saída	
Tempo (Segundos)	1.7866
N° de Comparações	500001
N° de Trocas	500001
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 82 – Resultado.

Saída	
Tempo (Segundos)	2.1255
N° de Comparações	600001
N° de Trocas	600001
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 83 – Resultado.

Saída	
Tempo (Segundos)	2.6611
N° de Comparações	700001
N° de Trocas	700001
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 84 – Resultado.

Saída	
Tempo (Segundos)	2.8802
N° de Comparações	800001
N° de Trocas	800001
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 85 – Resultado.

Saída	
Tempo (Segundos)	3.2981
N° de Comparações	900001
N° de Trocas	900001
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 86 – Resultado.

Saída	
Tempo (Segundos)	3.7101
N° de Comparações	1000001
N° de Trocas	1000001
Comprimento do Vetor	1 000 000

2.3.2 Ordenação Decrescente

Para essa seção vamos assumir uma lista ja ordenada em ordem decrescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 87 – Resultado.

Saída	
Tempo (Segundos)	0.0021
N° de Comparações	1001
N° de Trocas	501
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 88 – Resultado.

Saída	
Tempo (Segundos)	0.0302
N° de Comparações	10001
N° de Trocas	5001
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 89 – Resultado.

Saída	
Tempo (Segundos)	0.3260
N° de Comparações	100001
N° de Trocas	50001
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 90 – Resultado.

Saída	
Tempo (Segundos)	0.6452
N° de Comparações	200001
N° de Trocas	100001
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 91 – Resultado.

Saída	
Tempo (Segundos)	1.0389
N° de Comparações	300001
N° de Trocas	150001
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 92 – Resultado.

Saída	
Tempo (Segundos)	1.3160
N° de Comparações	400001
N° de Trocas	200001
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 93 – Resultado.

Saída	
Tempo (Segundos)	1.8045
N° de Comparações	500001
N° de Trocas	250001
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 94 – Resultado.

Saída	
Tempo (Segundos)	2.0270
N° de Comparações	600001
N° de Trocas	300001
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 95 – Resultado.

Saída	
Tempo (Segundos)	2.5137
N° de Comparações	700001
N° de Trocas	350001
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 96 – Resultado.

Saída	
Tempo (Segundos)	2.7485
N° de Comparações	800001
N° de Trocas	400001
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 97 – Resultado.

Saída	
Tempo (Segundos)	3.1883
N° de Comparações	900001
N° de Trocas	450001
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 98 – Resultado.

Saída	
Tempo (Segundos)	3.5724
N° de Comparações	1000001
N° de Trocas	500001
Comprimento do Vetor	1 000 000

2.3.3 Ordenação Aleatória

Para essa seção vamos assumir uma lista desordenada de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 99 – Resultado.

Saída	
Tempo (Segundos)	0.0024
N° de Comparações	1001
N° de Trocas	996
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 100 – Resultado.

Saída	
Tempo (Segundos)	0.0333
N° de Comparações	10001
N° de Trocas	10001
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 101 – Resultado.

Saída	
Tempo (Segundos)	0.3566
N° de Comparações	100001
N° de Trocas	100001
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 102 – Resultado.

Saída	
Tempo (Segundos)	0.7831
N° de Comparações	200001
N° de Trocas	199999
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 103 – Resultado.

Saída	
Tempo (Segundos)	1.2462
N° de Comparações	300001
N° de Trocas	300000
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 104 – Resultado.

Saída	
Tempo (Segundos)	1.8121
N° de Comparações	400001
N° de Trocas	400001
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 105 – Resultado.

Saída	
Tempo (Segundos)	2.2109
N° de Comparações	500001
N° de Trocas	500001
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 106 – Resultado.

Saída	
Tempo (Segundos)	2.5679
N° de Comparações	600001
N° de Trocas	600001
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 107 – Resultado.

Saída	
Tempo (Segundos)	3.2420
N° de Comparações	700001
N° de Trocas	700001
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 108 – Resultado.

Saída	
Tempo (Segundos)	3.6515
N° de Comparações	800001
N° de Trocas	800000
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 109 – Resultado.

Saída	
Tempo (Segundos)	4.0694
N° de Comparações	900001
N° de Trocas	899995
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação merge, obtemos como resultado:

Tabela 110 – Resultado.

Saída	
Tempo (Segundos)	4.5969
N° de Comparações	1000001
N° de Trocas	1000001
Comprimento do Vetor	1 000 000

2.3.4 Comparativos

Tabela 111 – Resultados Comparativos.

Comparações				
Ordenação	Comprimento	Tempo (s)	N° Comparações	N° Trocas
Crescente	1000	0.0024	1001	1001
	10000	0.0404	10001	10001
	100000	0.3087	100001	100001
	200000	0.6490	200001	200001
	300000	1.0007	300001	300001
	400000	1.4187	400001	400001
	500000	1.7866	500001	500001
	600000	2.1255	600001	600001
	700000	2.6611	700001	700001
	800000	2.8802	800001	800001
	900000	3.2981	900001	900001
	1000000	3.7101	1000001	1000001
Decrescente	1000	0.0021	1001	501
	10000	0.0302	10001	5001
	100000	0.3260	100001	50001
	200000	0.6452	200001	100001
	300000	1.0389	300001	150001
	400000	1.3160	400001	200001
	500000	1.8045	500001	250001
	600000	2.0270	600001	300001
	700000	2.5137	700001	350001
	800000	2.7485	800001	400001
	900000	3.1883	900001	450001
	1000000	3.5724	1000001	500001
Aleatória	1000	0.0024	1001	996
	10000	0.0333	10001	10001
	100000	0.3566	100001	100001
	200000	0.7831	200001	199999
	300000	1.2462	300001	300000
	400000	1.8121	400001	400001
	500000	2.2109	500001	500001
	600000	2.5679	600001	600001
	700000	3.2420	700001	700001
	800000	3.6515	800001	800000
	900000	4.0694	900001	899995
	1000000	4.5969	1000001	1000001

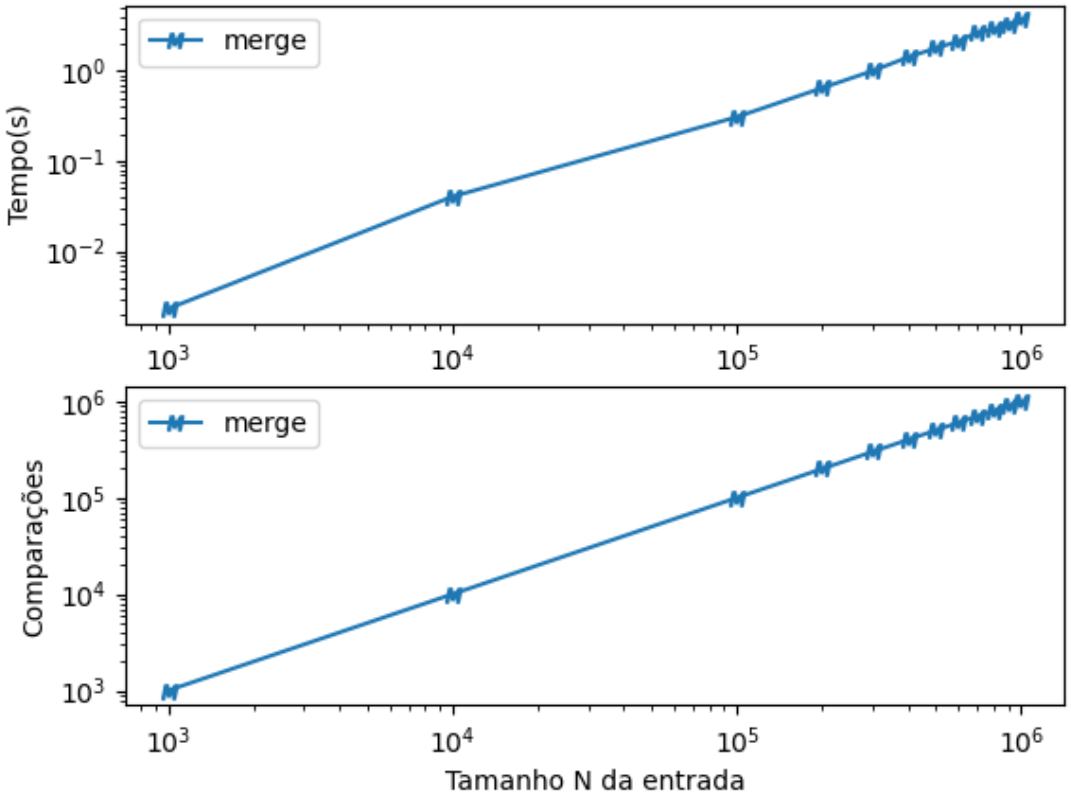

Figura 7 – Representação do comportamento do algoritmo de ordenação merge sort com uma entrada de tamanho 1000000 ordenadas de forma crescente

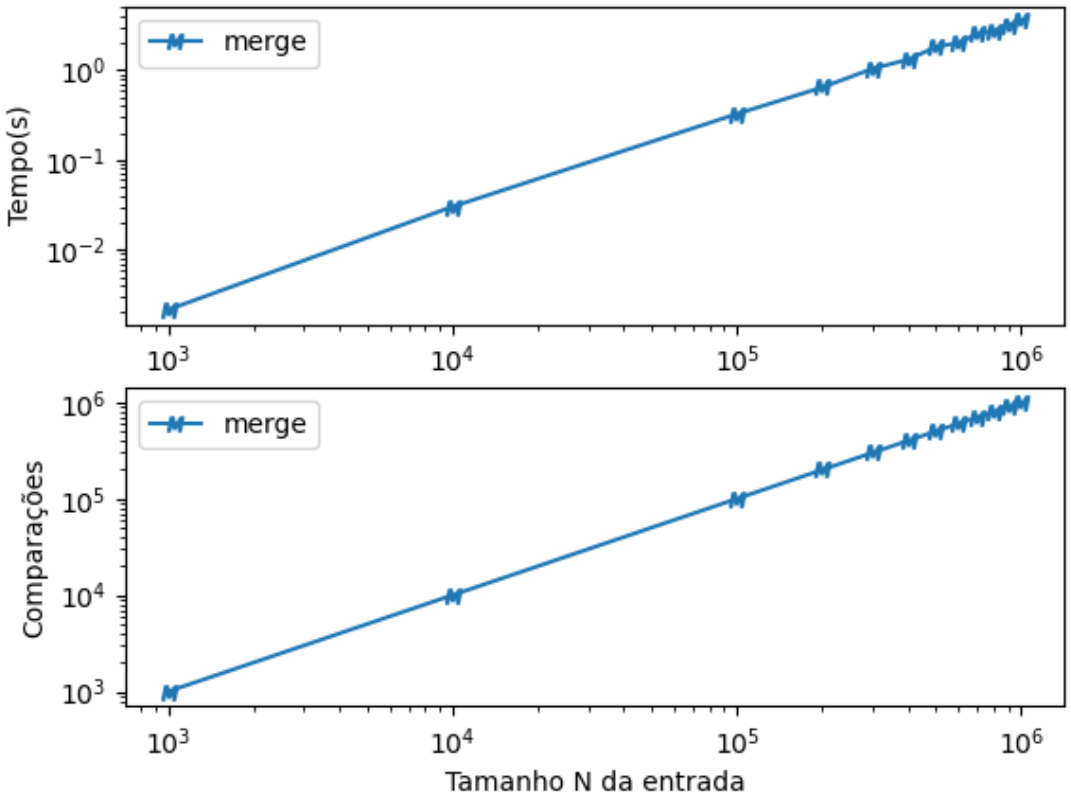

Figura 8 – Representação do comportamento do algoritmo de ordenação merge sort com uma entrada de tamanho 1000000 ordenadas de forma decrescente

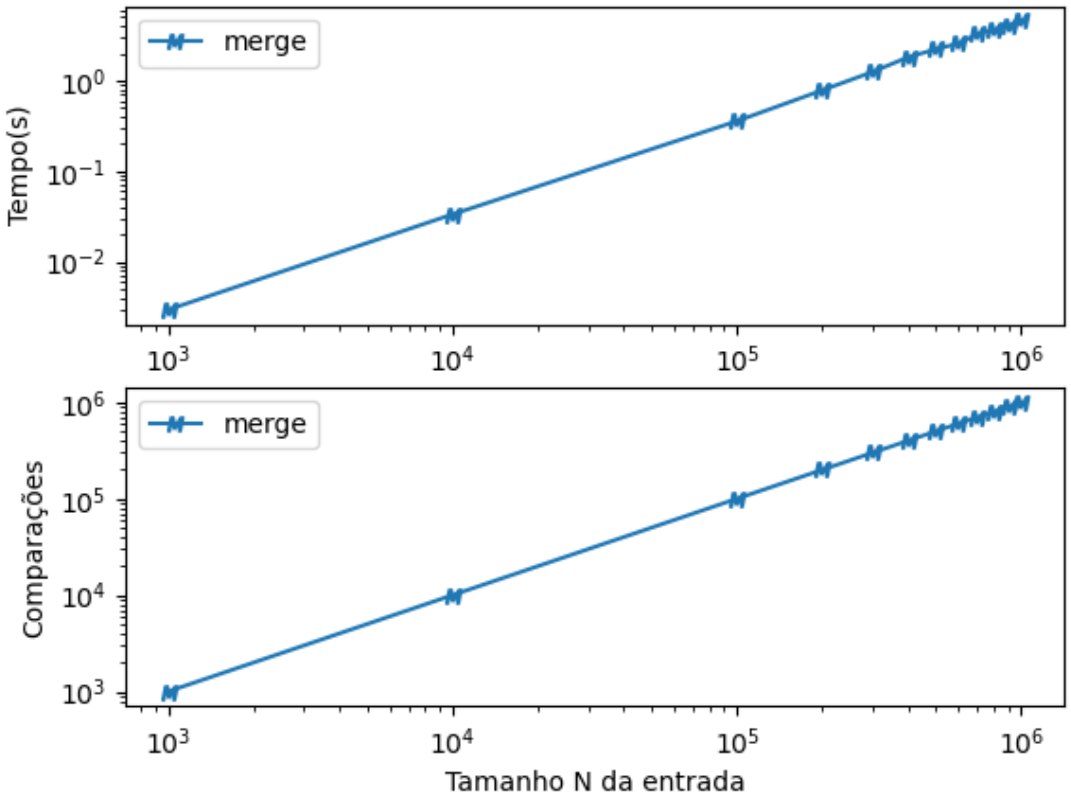

Figura 9 – Representação do comportamento do algoritmo de ordenação merge sort com uma entrada de tamanho 1000000 desordenadas

2.4 Quick Sort

```
1  def particao(vetor, inicio, final):
2    comp, trocas = 0, 0
3    pivo = vetor[final]
4    i = inicio - 1
5
6    for j in range(inicio, final):
7      if vetor[j] <= pivo:
8        comp += 1
9        i += 1
10       vetor[i], vetor[j] = vetor[j], vetor[i]
11       trocas += 1
12   vetor[i + 1], vetor[final] = vetor[final], vetor[i + 1]
13   trocas += 1
14   return i + 1, comp, trocas
15
16 def quick_sort(vetor, inicio, final):
17   comp, trocas = 0, 0
18
19   if inicio < final:
20     posicao, comp, trocas = particao(vetor, inicio, final)
21     # Esquerda
22     quick_sort(vetor, inicio, posicao - 1)
23     # Direito
24     quick_sort(vetor, posicao + 1, final)
25   return comp, trocas
```

Essa duas funções representa respectivamente o nosso algoritmo de ordenação quick sort sendo uma função para auxílio e a outra o algoritmo propriamente dito. O algoritmo de ordenação Quick Sort é um método de ordenação que escolhe um elemento pivô da lista e rearranja os elementos de forma que os elementos menores que o pivô estejam antes dele e os elementos maiores estejam depois. Em seguida, o algoritmo é aplicado recursivamente nas sublistas antes e depois do pivô, até que a lista esteja completamente ordenada. É um método eficiente que também utiliza o conceito de ”dividir para conquistar”.

2.4.1 Ordenação Crescente

Para essa seção vamos assumir uma lista ja ordenada em ordem crescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 112 – Resultado.

Saída	
Tempo (Segundos)	0.0702
N° de Comparações	1000
N° de Trocas	1001
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 113 – Resultado.

Saída	
Tempo (Segundos)	7.1435
N° de Comparações	10000
N° de Trocas	10001
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 114 – Resultado.

Saída	
Tempo (Segundos)	390.5776
N° de Comparações	100001
N° de Trocas	100001
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 115 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 116 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 117 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 118 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 119 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 120 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 121 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 122 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 123 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	1 000 000

2.4.2 Ordenação Decrescente

Para essa seção vamos assumir uma lista ja ordenada em ordem decrescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 124 – Resultado.

Saída	
Tempo (Segundos)	0.0462
N° de Comparações	0
N° de Trocas	1
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 125 – Resultado.

Saída	
Tempo (Segundos)	4.6015
N° de Comparações	0
N° de Trocas	1
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 126 – Resultado.

Saída	
Tempo (Segundos)	210.0367
N° de Comparações	0
N° de Trocas	1
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 127 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 128 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 129 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 130 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 131 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 132 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 133 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 134 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 135 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	1 000 000

2.4.3 Ordenação Aleatória

Para essa seção vamos assumir uma lista desordenada de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 136 – Resultado.

Saída	
Tempo (Segundos)	0.0015
N° de Comparações	0.2424
N° de Trocas	267
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 137 – Resultado.

Saída	
Tempo (Segundos)	0.0215
N° de Comparações	7434
N° de Trocas	7434
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 138 – Resultado.

Saída	
Tempo (Segundos)	0.2649
N° de Comparações	73089
N° de Trocas	73090
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 139 – Resultado.

Saída	
Tempo (Segundos)	0.5209
N° de Comparações	79937
N° de Trocas	79938
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 140 – Resultado.

Saída	
Tempo (Segundos)	0.8243
N° de Comparações	18924
N° de Trocas	18925
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 141 – Resultado.

Saída	
Tempo (Segundos)	1.2690
N° de Comparações	122636
N° de Trocas	122637
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 142 – Resultado.

Saída	
Tempo (Segundos)	1.4828
N° de Comparações	250224
N° de Trocas	250225
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 143 – Resultado.

Saída	
Tempo (Segundos)	1.8320
N° de Comparações	331620
N° de Trocas	331621
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 144 – Resultado.

Saída	
Tempo (Segundos)	2.4601
N° de Comparações	683072
N° de Trocas	683073
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 145 – Resultado.

Saída	
Tempo (Segundos)	2.7642
N° de Comparações	413336
N° de Trocas	413337
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 146 – Resultado.

Saída	
Tempo (Segundos)	2.8631
N° de Comparações	513634
N° de Trocas	513635
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação quick, obtemos como resultado:

Tabela 147 – Resultado.

Saída	
Tempo (Segundos)	3.2134
N° de Comparações	286312
N° de Trocas	286313
Comprimento do Vetor	1 000 000

2.4.4 Comparativos

Tabela 148 – Resultados Comparativos.

Comparações				
Ordenação	Comprimento	Tempo (s)	N° Comparações	N° Trocas
Crescente	1000	0.0702	1000	1001
	10000	7.1435	10000	10001
	100000	390.5776	100001	100001
Decrescente	1000	0.0462	0	1
	10000	4.6015	0	1
	100000	210.0367	0	1
Aleatória	1000	0.0015	325	326
	10000	0.0215	7434	7434
	100000	0.2649	73089	73090
	200000	0.5209	79937	79938
	300000	0.8243	18924	18925
	400000	1.2690	122636	122637
	500000	1.4828	250224	250225
	600000	1.8320	331620	331621
	700000	2.4601	683072	683073
	800000	2.7642	413336	413337
	900000	2.8631	513634	513635
	1000000	3.2134	286312	286313

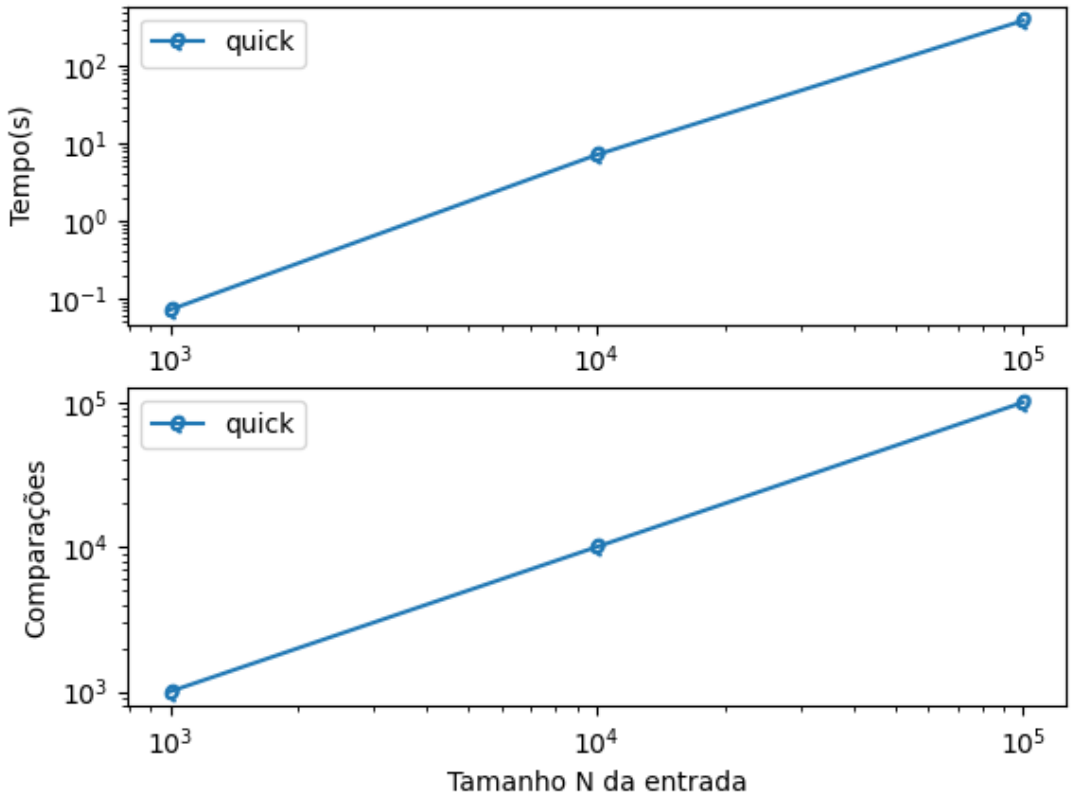

Figura 10 – Representação do comportamento do algoritmo de ordenação quick sort com uma entrada de tamanho até 100000 ordenadas de forma crescente

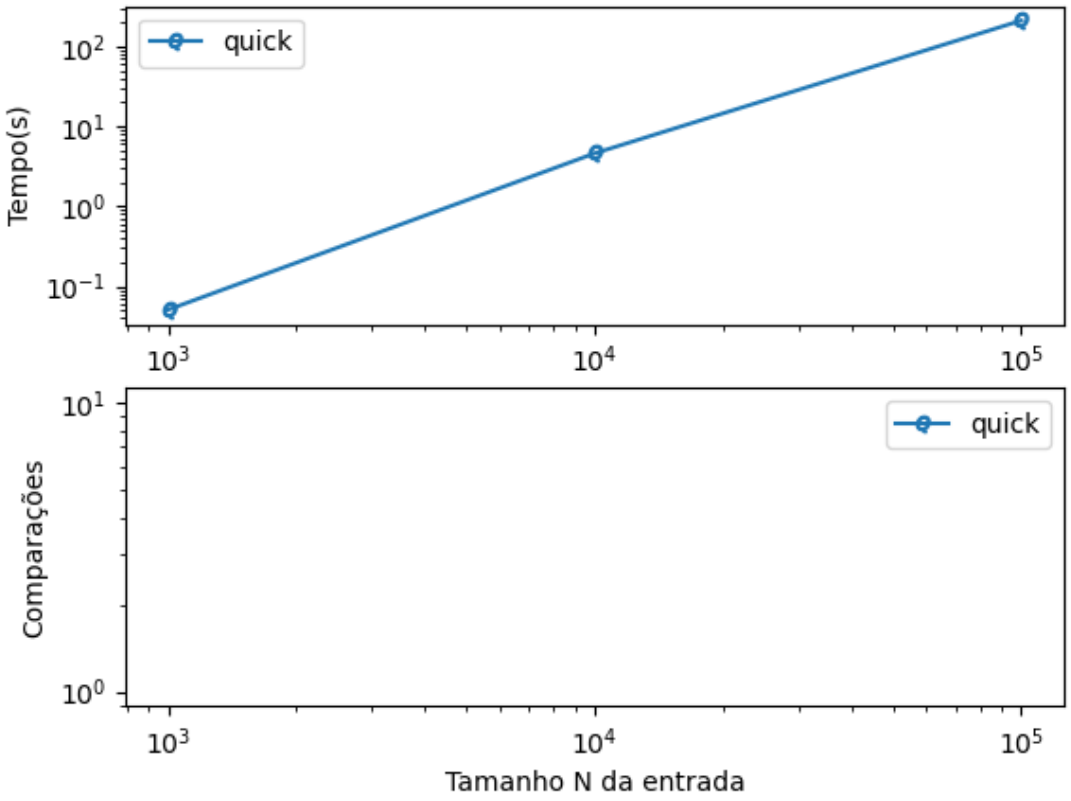

Figura 11 – Representação do comportamento do algoritmo de ordenação quick sort com uma entrada de tamanho até 100000 ordenadas de forma decrescente

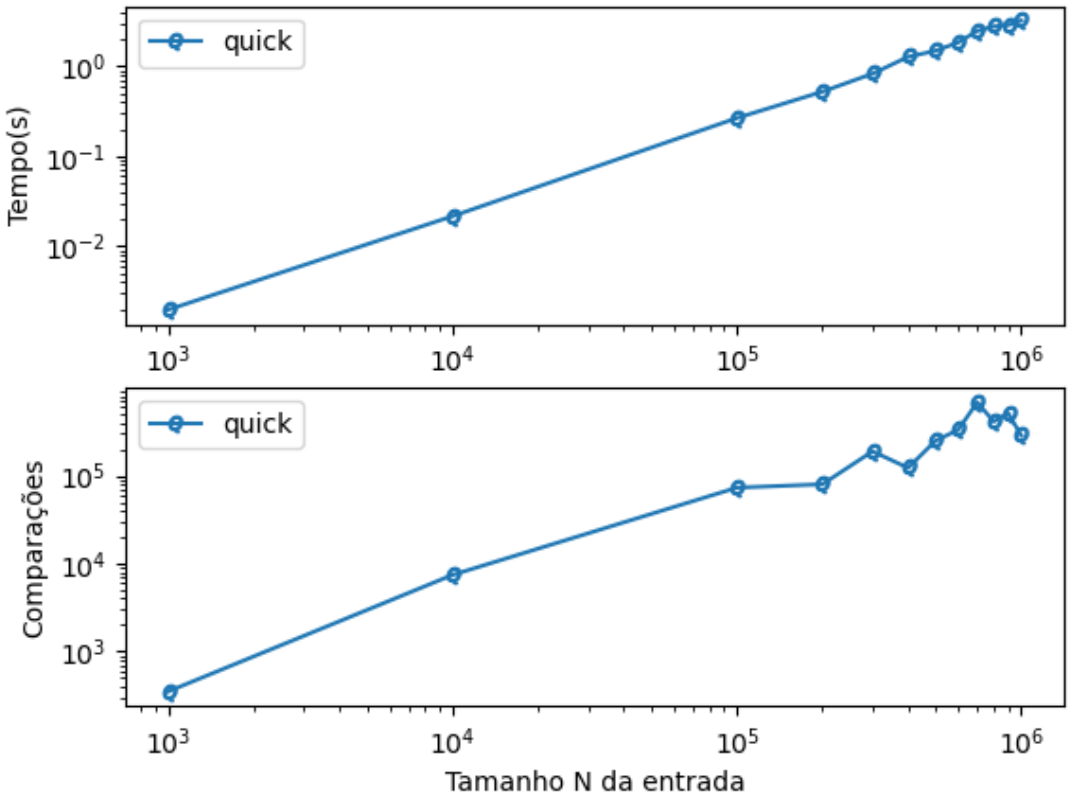

Figura 12 – Representação do comportamento do algoritmo de ordenação quick sort com uma entrada de tamanho até 1000000 desordenadas

2.5 Select Sort

```
1 def selection_sort(vetor):
2   comp, trocas = 0, 0
3   n = len(vetor)
4
5   for i in range(n):
6     id_minimo = i
7     for j in range(i + 1, n):
8       if vetor[id_minimo] > vetor[j]:
9         comp += 1
10        id_minimo = j
11    temp = vetor[i]
12    vetor[i] = vetor[id_minimo]
13    vetor[id_minimo] = temp
14    trocas += 1
15
16  return comp, trocas
```

Essa função representa respectivamente o nosso algoritmo de ordenação select sort. O algoritmo de ordenação Select Sort é um método de ordenação que percorre repetidamente a lista em busca do menor elemento e o coloca na posição correta. Ele divide a lista em uma parte ordenada e outra não ordenada, selecionando o menor elemento da parte não ordenada e inserindo-o na parte ordenada. Esse processo é repetido até que todos os elementos estejam na posição correta.

2.5.1 Ordenação Crescente

Para essa seção vamos assumir uma lista ja ordenada em ordem decrescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 149 – Resultado.

Saída	
Tempo (Segundos)	0.0220
N° de Comparações	0
N° de Trocas	1001
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 150 – Resultado.

Saída	
Tempo (Segundos)	2.1851
N° de Comparações	0
N° de Trocas	10001
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 151 – Resultado.

Saída	
Tempo (Segundos)	205.8524
N° de Comparações	0
N° de Trocas	100001
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 152 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 153 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 154 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 155 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 156 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 157 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 158 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 159 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 160 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	1 000 000

2.5.2 Ordenação Decrescente

Para essa seção vamos assumir uma lista ja ordenada em ordem crescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 161 – Resultado.

Saída	
Tempo (Segundos)	0.0293
N° de Comparações	250500
N° de Trocas	1001
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 162 – Resultado.

Saída	
Tempo (Segundos)	2.8245
N° de Comparações	25005000
N° de Trocas	10001
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 163 – Resultado.

Saída	
Tempo (Segundos)	274.3685
N° de Comparações	2500050000
N° de Trocas	100001
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 164 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 165 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 166 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 167 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 168 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 169 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 170 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 171 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 172 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	1 000 000

2.5.3 Ordenação Aleatória

Para essa seção vamos assumir uma lista desordenadas de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 173 – Resultado.

Saída	
Tempo (Segundos)	0.0233
N° de Comparações	5386
N° de Trocas	1001
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 174 – Resultado.

Saída	
Tempo (Segundos)	2.1981
N° de Comparações	77911
N° de Trocas	10001
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 175 – Resultado.

Saída	
Tempo (Segundos)	207.4811
N° de Comparações	1007976
N° de Trocas	100001
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 176 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 177 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 178 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 179 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 180 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 181 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 182 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 183 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação select, obtemos como resultado:

Tabela 184 – Resultado.

Saída	
Tempo (Segundos)	indeterminado
N° de Comparações	indeterminado
N° de Trocas	indeterminado
Comprimento do Vetor	1 000 000

2.5.4 Comparativos

Tabela 185 – Resultados Comparativos.

Comparações				
Ordenação	Comprimento	Tempo (s)	N° Comparações	N° Trocas
Crescente	1000	0.0220	0	1001
	10000	2.1851	0	10001
	100000	205.8524	0	100001
Decrescente	1000	0.0293	250500	1001
	10000	2.8245	25005000	10001
	100000	274.3685	2500050000	100001
Aleatória	1000	0.0233	5386	1001
	10000	2.1981	77911	10001
	100000	207.4811	1007976	100001

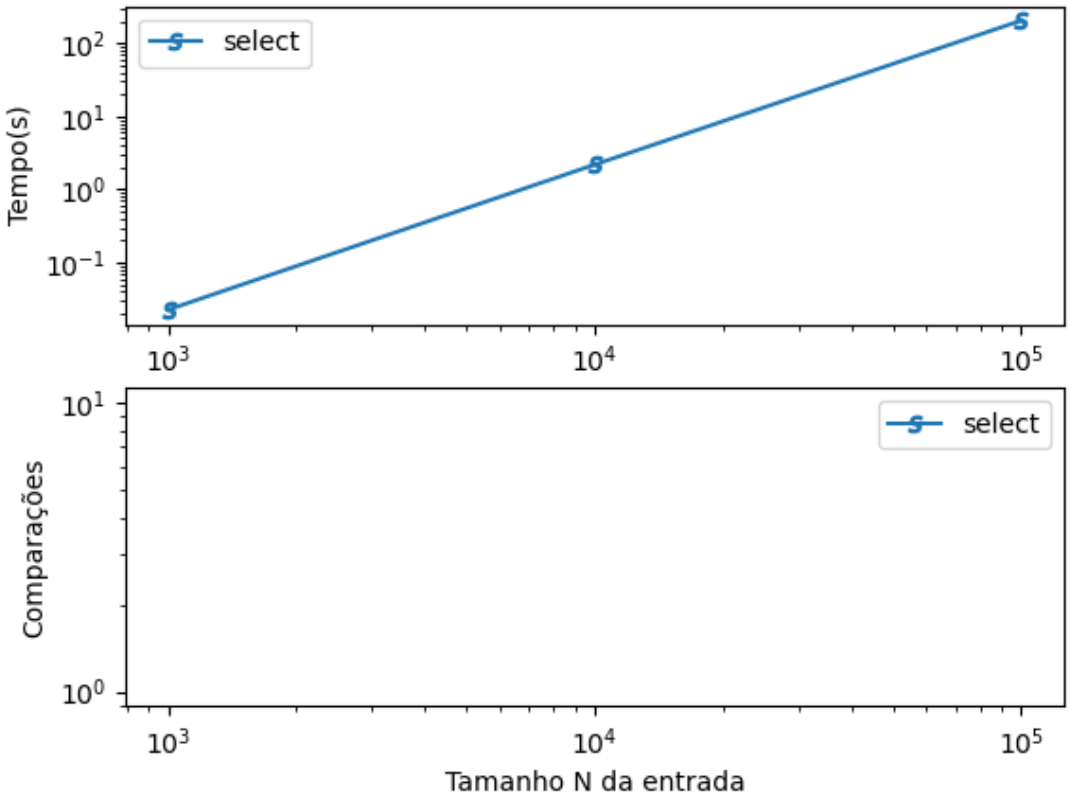

Figura 13 – Representação do comportamento do algoritmo de ordenação select sort com entradas de tamanho até 100000 ordenadas de forma crescente

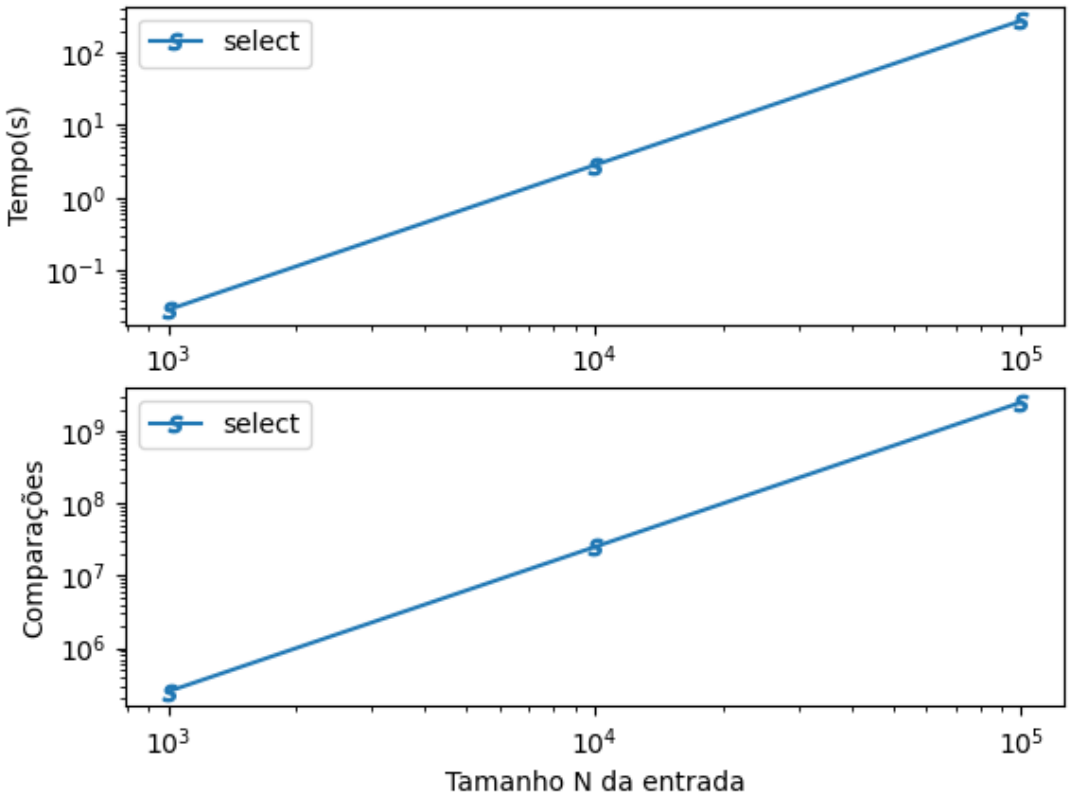

Figura 14 – Representação do comportamento do algoritmo de ordenação select sort com entradas de tamanho até 100000 ordenadas de forma decrescente

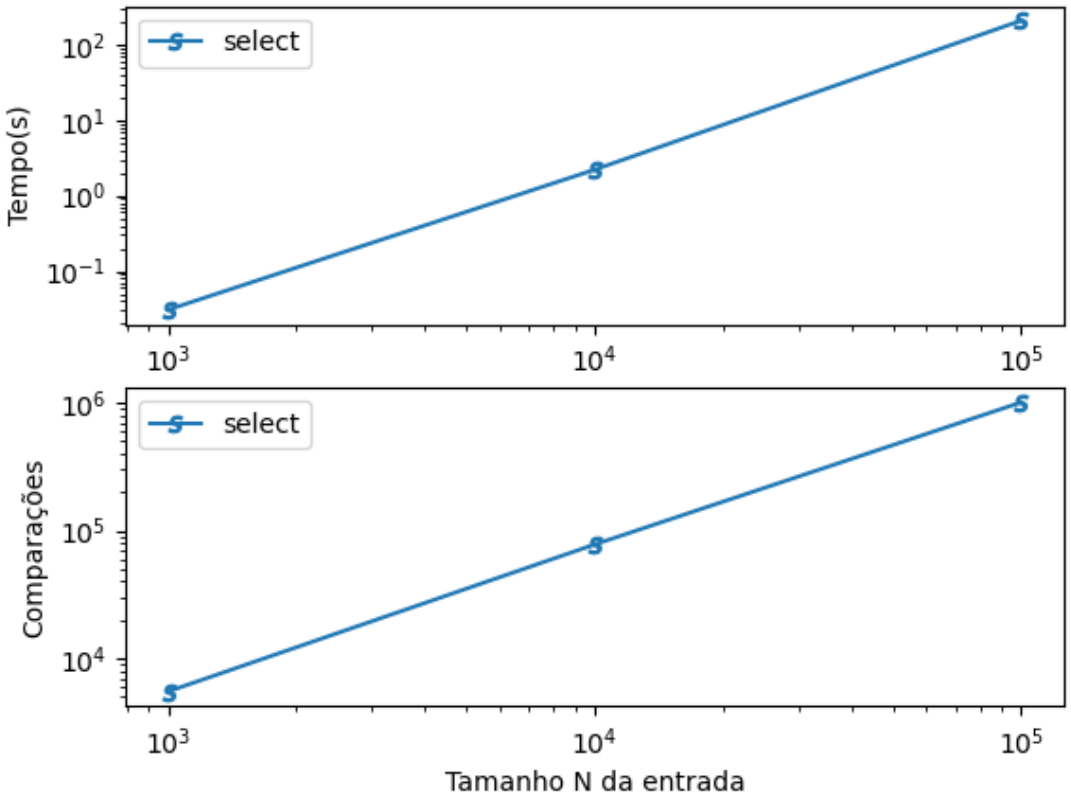

Figura 15 – Representação do comportamento do algoritmo de ordenação select sort com entradas de tamanho até 100000 desordenadas

2.6 Counting Sort

```
def counting_sort(arr):
    max_value = max(arr)
    count = [0] * (max_value + 1)

    comp = 0
    for num in arr:
        count[num] += 1
        comp += 1

    sorted_arr = []
    trocas = 0
    for i in range(len(count)):
        for j in range(count[i]):
            sorted_arr.append(i)


    return comp, trocas
```

Essa função representa respectivamente o nosso algoritmo de ordenação counting sort. O algoritmo de ordenação Counting Sort é um método de ordenação eficiente para listas com um intervalo conhecido de valores. Ele cria um array de contagem para contar o número de ocorrências de cada valor na lista original e, em seguida, usa essas contagens para posicionar os elementos na ordem correta. É um algoritmo estável, ou seja, preserva a ordem relativa de elementos com valores iguais.

2.6.1 Ordenação Crescente

Para essa seção vamos assumir uma lista ja ordenada em ordem decrescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 186 – Resultado.

Saída	
Tempo (Segundos)	0.0002
N° de Comparações	1001
N° de Trocas	0
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 187 – Resultado.

Saída	
Tempo (Segundos)	0.0044
N° de Comparações	10001
N° de Trocas	0
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 188 – Resultado.

Saída	
Tempo (Segundos)	0.0217
N° de Comparações	100001
N° de Trocas	0
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 189 – Resultado.

Saída	
Tempo (Segundos)	0.0436
N° de Comparações	200001
N° de Trocas	0
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 190 – Resultado.

Saída	
Tempo (Segundos)	0.0656
N° de Comparações	300001
N° de Trocas	0
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 191 – Resultado.

Saída	
Tempo (Segundos)	0.0889
N° de Comparações	400001
N° de Trocas	0
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 192 – Resultado.

Saída	
Tempo (Segundos)	0.1108
N° de Comparações	500001
N° de Trocas	0
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 193 – Resultado.

Saída	
Tempo (Segundos)	0.1400
N° de Comparações	600001
N° de Trocas	0
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 194 – Resultado.

Saída	
Tempo (Segundos)	0.1512
N° de Comparações	700001
N° de Trocas	0
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 195 – Resultado.

Saída	
Tempo (Segundos)	0.1769
N° de Comparações	800001
N° de Trocas	0
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 196 – Resultado.

Saída	
Tempo (Segundos)	0.2115
N° de Comparações	900001
N° de Trocas	0
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 197 – Resultado.

Saída	
Tempo (Segundos)	0.2217
N° de Comparações	1000001
N° de Trocas	0
Comprimento do Vetor	1 000 000

2.6.2 Ordenação Decrescente

Para essa seção vamos assumir uma lista ja ordenada em ordem decrescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 198 – Resultado.

Saída	
Tempo (Segundos)	0.0002
N° de Comparações	1001
N° de Trocas	0
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 199 – Resultado.

Saída	
Tempo (Segundos)	0.0021
N° de Comparações	10001
N° de Trocas	0
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 200 – Resultado.

Saída	
Tempo (Segundos)	0.0228
N° de Comparações	100001
N° de Trocas	0
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 201 – Resultado.

Saída	
Tempo (Segundos)	0.0431
N° de Comparações	200001
N° de Trocas	0
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 202 – Resultado.

Saída	
Tempo (Segundos)	0.0685
N° de Comparações	300001
N° de Trocas	0
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 203 – Resultado.

Saída	
Tempo (Segundos)	0.0914
N° de Comparações	400001
N° de Trocas	0
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 204 – Resultado.

Saída	
Tempo (Segundos)	0.1093
N° de Comparações	500001
N° de Trocas	0
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 205 – Resultado.

Saída	
Tempo (Segundos)	0.1317
N° de Comparações	600001
N° de Trocas	0
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 206 – Resultado.

Saída	
Tempo (Segundos)	0.1542
N° de Comparações	700001
N° de Trocas	0
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 207 – Resultado.

Saída	
Tempo (Segundos)	0.1816
N° de Comparações	800001
N° de Trocas	0
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 208 – Resultado.

Saída	
Tempo (Segundos)	0.2053
N° de Comparações	900001
N° de Trocas	0
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 209 – Resultado.

Saída	
Tempo (Segundos)	0.2461
N° de Comparações	1000001
N° de Trocas	0
Comprimento do Vetor	1 000 000

2.6.3 Ordenação Aleatória

Para essa seção vamos assumir uma lista desordenada de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 210 – Resultado.

Saída	
Tempo (Segundos)	0.0006
N° de Comparações	1001
N° de Trocas	0
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 211 – Resultado.

Saída	
Tempo (Segundos)	0.0022
N° de Comparações	10001
N° de Trocas	0
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 212 – Resultado.

Saída	
Tempo (Segundos)	0.0213
N° de Comparações	100001
N° de Trocas	0
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 213 – Resultado.

Saída	
Tempo (Segundos)	0.0456
N° de Comparações	200001
N° de Trocas	0
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 214 – Resultado.

Saída	
Tempo (Segundos)	0.0726
N° de Comparações	300001
N° de Trocas	0
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 215 – Resultado.

Saída	
Tempo (Segundos)	0.0907
N° de Comparações	400001
N° de Trocas	0
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 216 – Resultado.

Saída	
Tempo (Segundos)	0.1165
N° de Comparações	500001
N° de Trocas	0
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 217 – Resultado.

Saída	
Tempo (Segundos)	0.1625
N° de Comparações	600001
N° de Trocas	0
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 218 – Resultado.

Saída	
Tempo (Segundos)	0.1967
N° de Comparações	700001
N° de Trocas	0
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 219 – Resultado.

Saída	
Tempo (Segundos)	0.2268
N° de Comparações	800001
N° de Trocas	0
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 220 – Resultado.

Saída	
Tempo (Segundos)	0.2571
N° de Comparações	900001
N° de Trocas	0
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação counting, obtemos como resultado:

Tabela 221 – Resultado.

Saída	
Tempo (Segundos)	0.2652
N° de Comparações	1000001
N° de Trocas	0
Comprimento do Vetor	1 000 000

2.6.4 Comparativos

Tabela 222 – Resultados Comparativos.

Comparações				
Ordenação	Comprimento	Tempo (s)	N° Comparações	N° Trocas
Crescente	1000	0.0002	1001	0
	10000	0.0044	10001	0
	100000	0.0217	100001	0
	200000	0.0436	200001	0
	300000	0.0656	300001	0
	400000	0.0889	400001	0
	500000	0.1108	500001	0
	600000	0.1400	600001	0
	700000	0.1512	700001	0
	800000	0.1769	800001	0
	900000	0.2115	900001	0
	1000000	0.2217	1000001	0
Decrescente	1000	0.0002	1001	0
	10000	0.0021	10001	0
	100000	0.0228	100001	0
	200000	0.0431	200001	0
	300000	0.0685	300001	0
	400000	0.0914	400001	0
	500000	0.1093	500001	0
	600000	0.1317	600001	0
	700000	0.1542	700001	0
	800000	0.1816	800001	0
	900000	0.2053	900001	0
	1000000	0.2461	1000001	0
Aleatória	1000	0.0006	1001	0
	10000	0.0022	10001	0
	100000	0.0213	100001	0
	200000	0.0456	200001	0
	300000	0.0726	300001	0
	400000	0.0907	400001	0
	500000	0.1165	500001	0
	600000	0.1625	600001	0
	700000	0.1967	700001	0
	800000	0.2268	800001	0
	900000	0.2571	900001	0
	1000000	0.2652	1000001	0

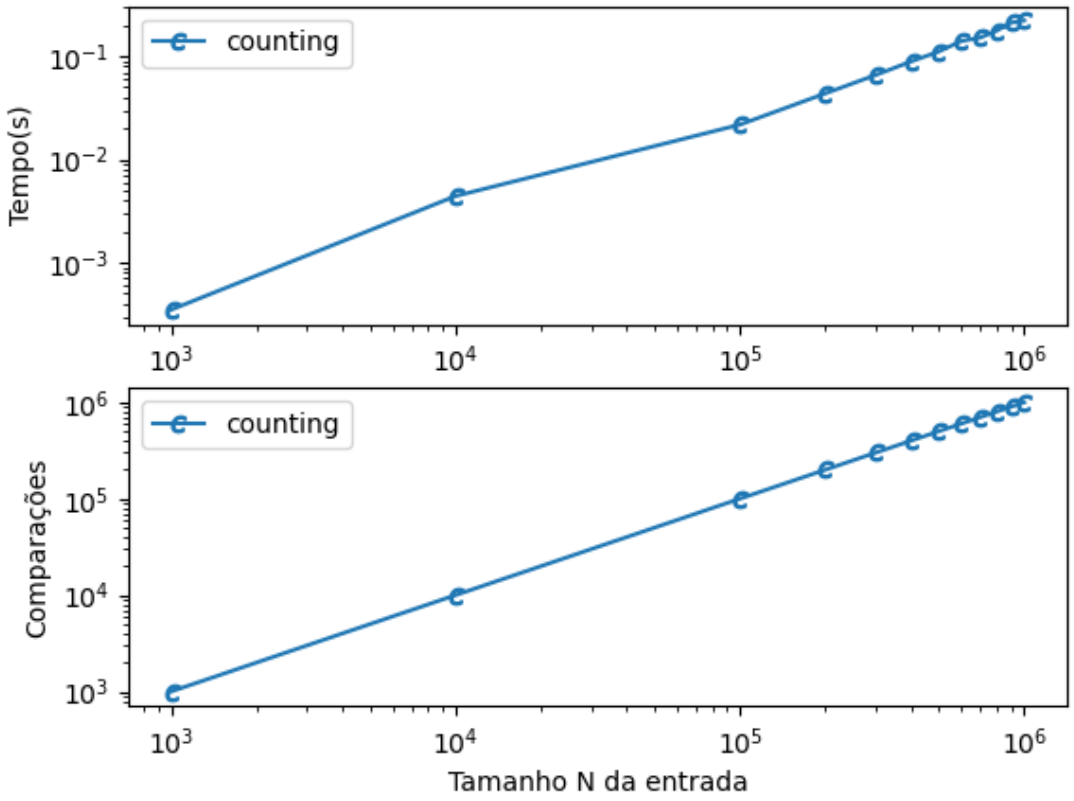

Figura 16 – Representação do comportamento do algoritmo de ordenação counting sort com entradas de tamanho até 1000000 ordenadas de forma crescente

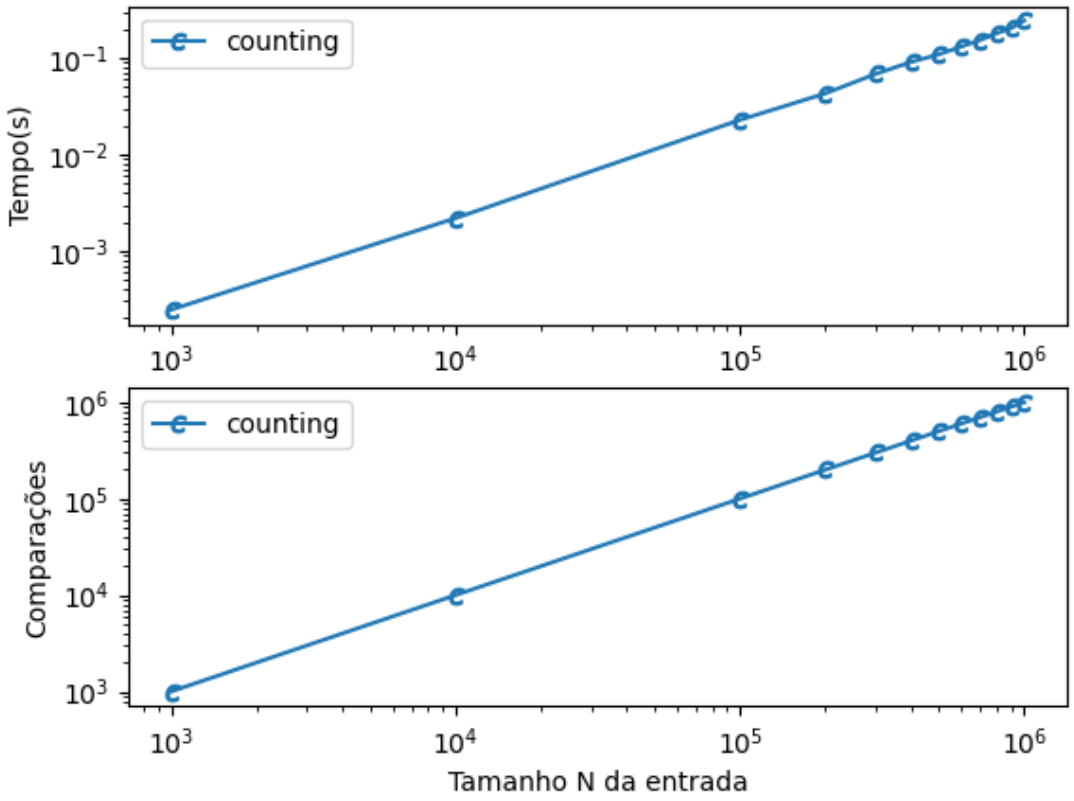

Figura 17 – Representação do comportamento do algoritmo de ordenação counting sort com entradas de tamanho até 1000000 ordenadas de forma decrescente

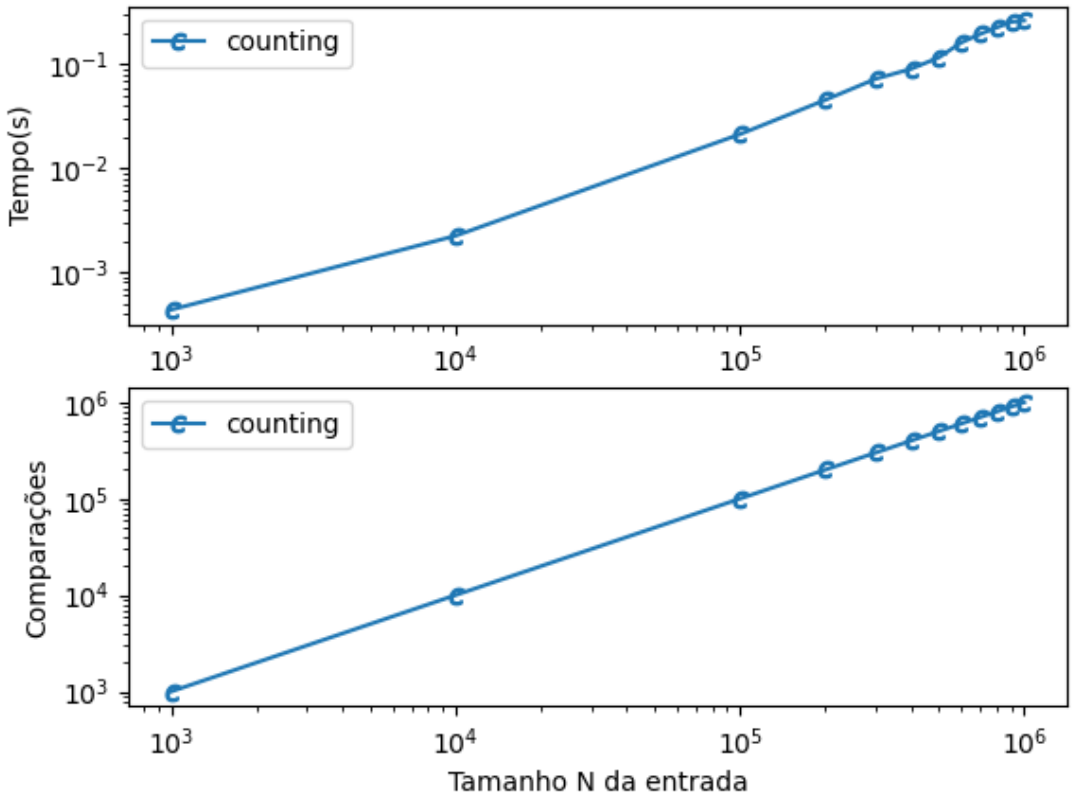

Figura 18 – Representação do comportamento do algoritmo de ordenação counting sort com entradas de tamanho até 1000000 desordenadas

2.7 Shell Sort

```
def shell_sort(arr):
    n = len(arr)
    gap = n // 2
    comp = 0
    trocas = 0
    while gap > 0:
        for i in range(gap, n):
            temp = arr[i]
            j = i
            while j >= gap and arr[j - gap] > temp:
                arr[j] = arr[j - gap]
                j -= gap
                trocas += 1
                comp += 1
            arr[j] = temp
            comp += 1
        gap //= 2
    return comp, trocas
```

Essa função representa respectivamente o nosso algoritmo de ordenação shell sort. O algoritmo de ordenação Shell Sort é um algoritmo de ordenação baseado na inserção direta que divide a lista em subgrupos menores e, em seguida, aplica a inserção direta em cada subgrupo. Essa abordagem permite que elementos distantes um do outro sejam comparados e trocados, resultando em uma maior eficiência do que a inserção direta tradicional. O algoritmo repete esse processo de dividir e ordenar os subgrupos com um tamanho de intervalo decrescente até que todos os elementos estejam ordenados.

2.7.1 Ordenação Crescente

Para essa seção vamos assumir uma lista ja ordenada em ordem crescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 223 – Resultado.

Saída	
Tempo (Segundos)	0.0010
N° de Comparações	8015
N° de Trocas	0
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 224 – Resultado.

Saída	
Tempo (Segundos)	0.0264
N° de Comparações	120018
N° de Trocas	0
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 225 – Resultado.

Saída	
Tempo (Segundos)	0.1612
N° de Comparações	1500022
N° de Trocas	0
Comprimento do Vetor	100 000

- Vetor de $2*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 226 – Resultado.

Saída	
Tempo (Segundos)	0.3423
N° de Comparações	3200023
N° de Trocas	0
Comprimento do Vetor	200 000

- Vetor de $3*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 227 – Resultado.

Saída	
Tempo (Segundos)	0.5526
N° de Comparações	5100026
N° de Trocas	0
Comprimento do Vetor	300 000

- Vetor de $4*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 228 – Resultado.

Saída	
Tempo (Segundos)	0.7508
N° de Comparações	6800024
N° de Trocas	0
Comprimento do Vetor	400 000

- Vetor de $5*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 229 – Resultado.

Saída	
Tempo (Segundos)	0.9573
N° de Comparações	8500025
N° de Trocas	0
Comprimento do Vetor	500 000

- Vetor de $6*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 230 – Resultado.

Saída	
Tempo (Segundos)	1.2261
N° de Comparações	10800027
N° de Trocas	0
Comprimento do Vetor	600 000

- Vetor de $7*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 231 – Resultado.

Saída	
Tempo (Segundos)	1.4106
N° de Comparações	12600028
N° de Trocas	0
Comprimento do Vetor	700 000

- Vetor de $8*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 232 – Resultado.

Saída	
Tempo (Segundos)	1.5973
N° de Comparações	14400025
N° de Trocas	0
Comprimento do Vetor	800 000

- Vetor de $9*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 233 – Resultado.

Saída	
Tempo (Segundos)	1.8699
N° de Comparações	16200030
N° de Trocas	0
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 234 – Resultado.

Saída	
Tempo (Segundos)	2.0871
N° de Comparações	18000026
N° de Trocas	0
Comprimento do Vetor	1 000 000

2.7.2 Ordenação Decrescente

Para essa seção vamos assumir uma lista ja ordenada em ordem decrescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 235 – Resultado.

Saída	
Tempo (Segundos)	0.0017
N° de Comparações	12391
N° de Trocas	4376
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 236 – Resultado.

Saída	
Tempo (Segundos)	0.0232
N° de Comparações	178150
N° de Trocas	58132
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 237 – Resultado.

Saída	
Tempo (Segundos)	0.2611
N° de Comparações	2249322
N° de Trocas	749300
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 238 – Resultado.

Saída	
Tempo (Segundos)	0.5672
N° de Comparações	4798619
N° de Trocas	1598596
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 239 – Resultado.

Saída	
Tempo (Segundos)	0.9045
N° de Comparações	7717934
N° de Trocas	2617908
Comprimento do Vetor	300 000

- Vetor de $4*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 240 – Resultado.

Saída	
Tempo (Segundos)	1.2244
N° de Comparações	10197212
N° de Trocas	3397188
Comprimento do Vetor	400 000

- Vetor de $5*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 241 – Resultado.

Saída	
Tempo (Segundos)	1.5927
N° de Comparações	13118317
N° de Trocas	4618292
Comprimento do Vetor	500 000

- Vetor de $6*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 242 – Resultado.

Saída	
Tempo (Segundos)	2.0064
N° de Comparações	16335839
N° de Trocas	5535812
Comprimento do Vetor	600 000

- Vetor de $7*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 243 – Resultado.

Saída	
Tempo (Segundos)	2.3611
N° de Comparações	19014608
N° de Trocas	6414580
Comprimento do Vetor	700 000

- Vetor de $8*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 244 – Resultado.

Saída	
Tempo (Segundos)	2.6097
N° de Comparações	21594397
N° de Trocas	7194372
Comprimento do Vetor	800 000

- Vetor de $9*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 245 – Resultado.

Saída	
Tempo (Segundos)	3.0482
N° de Comparações	25001298
N° de Trocas	8801268
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 246 – Resultado.

Saída	
Tempo (Segundos)	3.3044
N° de Comparações	27736606
N° de Trocas	9736580
Comprimento do Vetor	1 000 000

2.7.3 Ordenação Aleatória

Para essa seção vamos assumir uma lista desordenada de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 247 – Resultado.

Saída	
Tempo (Segundos)	0.0022
N° de Comparações	15355
N° de Trocas	7340
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 248 – Resultado.

Saída	
Tempo (Segundos)	0.0424
N° de Comparações	283012
N° de Trocas	162994
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 249 – Resultado.

Saída	
Tempo (Segundos)	0.6174
N° de Comparações	4442047
N° de Trocas	2942025
Comprimento do Vetor	100 000

- Vetor de $2*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 250 – Resultado.

Saída	
Tempo (Segundos)	1.6673
N° de Comparações	10258263
N° de Trocas	7058240
Comprimento do Vetor	200 000

- Vetor de $3*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 251 – Resultado.

Saída	
Tempo (Segundos)	2.3748
N° de Comparações	14655470
N° de Trocas	9555444
Comprimento do Vetor	300 000

- Vetor de $4*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 252 – Resultado.

Saída	
Tempo (Segundos)	3.9487
N° de Comparações	23612383
N° de Trocas	16812359
Comprimento do Vetor	400 000

- Vetor de $5*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 253 – Resultado.

Saída	
Tempo (Segundos)	4.5935
N° de Comparações	29728106
N° de Trocas	21228081
Comprimento do Vetor	500 000

- Vetor de $6*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 254 – Resultado.

Saída	
Tempo (Segundos)	5.8534
N° de Comparações	35041695
N° de Trocas	24241668
Comprimento do Vetor	600 000

- Vetor de $7*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 255 – Resultado.

Saída	
Tempo (Segundos)	7.0302
N° de Comparações	42171168
N° de Trocas	29571140
Comprimento do Vetor	700 000

- Vetor de $8*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 256 – Resultado.

Saída	
Tempo (Segundos)	8.7403
N° de Comparações	52990282
N° de Trocas	38590257
Comprimento do Vetor	800 000

- Vetor de $9*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 257 – Resultado.

Saída	
Tempo (Segundos)	9.3743
N° de Comparações	58555948
N° de Trocas	42355918
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 258 – Resultado.

Saída	
Tempo (Segundos)	10.5518
N° de Comparações	66036620
N° de Trocas	48036594
Comprimento do Vetor	1 000 000

2.7.4 Comparativos

Tabela 259 – Resultados Comparativos.

Comparações				
Ordenação	Comprimento	Tempo (s)	N° Comparações	N° Trocas
Crescente	1000	0.0010	8015	0
	10000	0.0264	120018	0
	100000	0.1612	1500022	0
	200000	0.3423	3200023	0
	300000	0.5526	5100026	0
	400000	0.7508	6800024	0
	500000	0.9573	8500025	0
	600000	1.2261	10800027	0
	700000	1.4106	12600028	0
	800000	1.5973	14400025	0
	900000	1.8699	16200030	0
	1000000	2.0871	18000026	0
Decrescente	1000	0.0.0017	12391	4376
	10000	0.0232	178150	58132
	100000	0.2611	2249322	749300
	200000	0.5672	4798619	1598596
	300000	0.9045	7717934	2617908
	400000	1.2244	10197212	3397188
	500000	1.5927	13118317	4618292
	600000	2.0064	16335839	5535812
	700000	2.3611	19014608	6414580
	800000	2.6097	21594397	7194372
	900000	3.0482	25001298	8801268
	1000000	3.3044	27736606	9736580
Aleatória	1000	0.0022	15355	7340
	10000	0.0424	283012	162994
	100000	0.6174	4442047	2942025
	200000	1.6673	10258263	7058240
	300000	2.3748	14655470	9555444
	400000	3.9487	23612383	16812359
	500000	4.5935	29728106	21228081
	600000	5.8534	35041695	24241668
	700000	7.0302	42171168	29571140
	800000	8.7403	52990282	38590257
	900000	9.3743	58555948	42355918
	1000000	10.5518	66036620	48036594

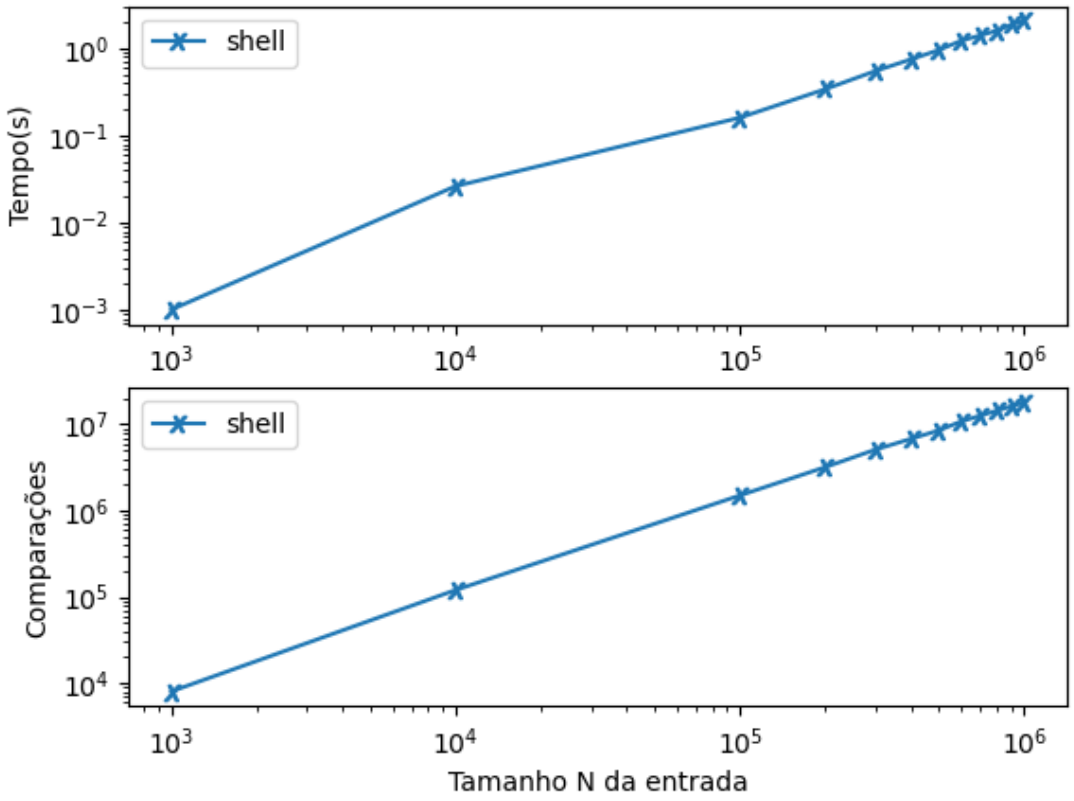

Figura 19 – Representação do comportamento do algoritmo de ordenação shell com uma entrada de tamanho 1000000 ordenadas de forma crescente

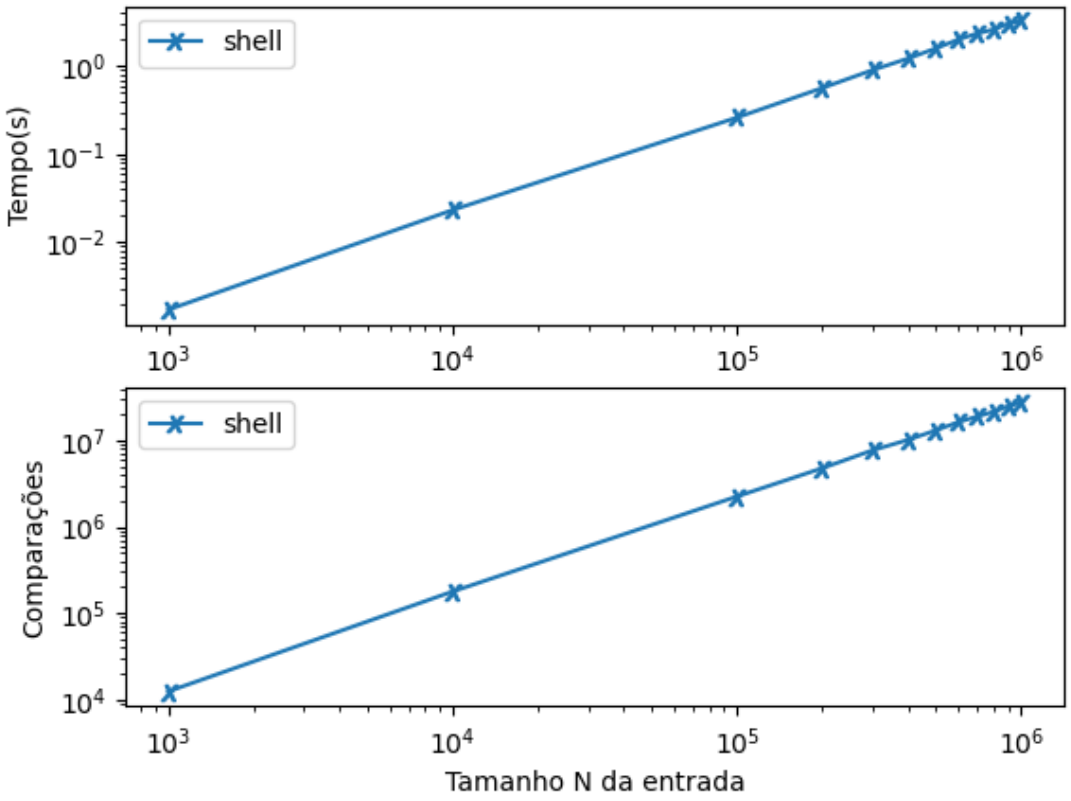

Figura 20 – Representação do comportamento do algoritmo de ordenação shell com uma entrada de tamanho 1000000 ordenadas de forma decrescente

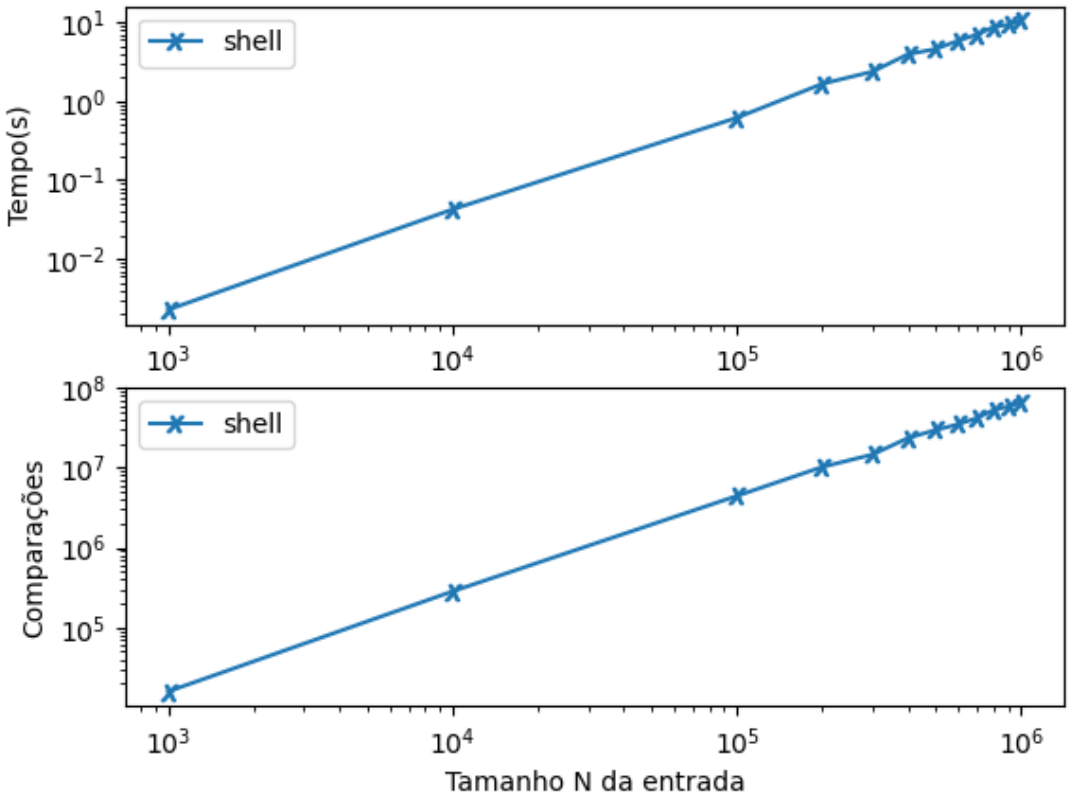

Figura 21 – Representação do comportamento do algoritmo de ordenação shell com uma entrada de tamanho 1000000 desordenadas

2.8 Radix Sort

```
1  def radix_sort(arr):
2      max_num = max(arr)
3      exp = 1
4      comp = 0
5      trocas = 0
6
7      while max_num // exp > 0:
8          trocas += counting_sort(arr, exp)
9          comp += 1
10         exp *= 10
11
12     return comp, trocas
```

Essa função representa respectivamente o nosso algoritmo de ordenação radix sort. O algoritmo de ordenação Radix Sort é um método de ordenação que ordena os elementos com base nos dígitos individuais. Ele classifica os elementos em baldes ou filas com base nos dígitos menos significativos primeiro e, em seguida, avança para os dígitos mais significativos. Esse processo é repetido até que todos os dígitos tenham sido considerados, resultando em uma lista ordenada. O Radix Sort é especialmente eficiente para ordenar números inteiros não negativos.

2.8.1 Ordenação Crescente

Para essa seção vamos assumir uma lista ja ordenada em ordem crescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 260 – Resultado.

Saída	
Tempo (Segundos)	0.0036
N° de Comparações	4
N° de Trocas	4004
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 261 – Resultado.

Saída	
Tempo (Segundos)	0.0237
N° de Comparações	5
N° de Trocas	50005
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 262 – Resultado.

Saída	
Tempo (Segundos)	0.2039
N° de Comparações	6
N° de Trocas	600006
Comprimento do Vetor	100 000

- Vetor de $2*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 263 – Resultado.

Saída	
Tempo (Segundos)	0.4274
N° de Comparações	6
N° de Trocas	1200006
Comprimento do Vetor	200 000

- Vetor de $3*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 264 – Resultado.

Saída	
Tempo (Segundos)	0.6722
N° de Comparações	6
N° de Trocas	1800006
Comprimento do Vetor	300 000

- Vetor de $4*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 265 – Resultado.

Saída	
Tempo (Segundos)	0.8897
N° de Comparações	6
N° de Trocas	2400006
Comprimento do Vetor	400 000

- Vetor de $5*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 266 – Resultado.

Saída	
Tempo (Segundos)	1.1325
N° de Comparações	6
N° de Trocas	3000006
Comprimento do Vetor	500 000

- Vetor de $6*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 267 – Resultado.

Saída	
Tempo (Segundos)	1.4048
N° de Comparações	6
N° de Trocas	3600006
Comprimento do Vetor	600 000

- Vetor de $7*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 268 – Resultado.

Saída	
Tempo (Segundos)	1.7977
N° de Comparações	6
N° de Trocas	4200006
Comprimento do Vetor	700 000

- Vetor de $8*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 269 – Resultado.

Saída	
Tempo (Segundos)	1.9558
N° de Comparações	6
N° de Trocas	4800006
Comprimento do Vetor	800 000

- Vetor de $9*10^5$ posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 270 – Resultado.

Saída	
Tempo (Segundos)	2.1013
N° de Comparações	6
N° de Trocas	5400006
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação shell, obtemos como resultado:

Tabela 271 – Resultado.

Saída	
Tempo (Segundos)	2.9429
N° de Comparações	7
N° de Trocas	7000007
Comprimento do Vetor	1 000 000

2.8.2 Ordenação Decrescente

Para essa seção vamos assumir uma lista ja ordenada em ordem decrescente de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 272 – Resultado.

Saída	
Tempo (Segundos)	0.0036
N° de Comparações	4
N° de Trocas	4004
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 273 – Resultado.

Saída	
Tempo (Segundos)	0.0285
N° de Comparações	5
N° de Trocas	50005
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 274 – Resultado.

Saída	
Tempo (Segundos)	0.2132
N° de Comparações	6
N° de Trocas	600006
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 275 – Resultado.

Saída	
Tempo (Segundos)	0.4298
N° de Comparações	6
N° de Trocas	1200006
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 276 – Resultado.

Saída	
Tempo (Segundos)	0.6655
N° de Comparações	6
N° de Trocas	1800006
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 277 – Resultado.

Saída	
Tempo (Segundos)	0.9059
N° de Comparações	6
N° de Trocas	2400006
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 278 – Resultado.

Saída	
Tempo (Segundos)	1.1941
N° de Comparações	6
N° de Trocas	3000006
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 279 – Resultado.

Saída	
Tempo (Segundos)	1.4038
N° de Comparações	6
N° de Trocas	3600006
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 280 – Resultado.

Saída	
Tempo (Segundos)	1.6313
N° de Comparações	6
N° de Trocas	4200006
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 281 – Resultado.

Saída	
Tempo (Segundos)	1.9599
N° de Comparações	6
N° de Trocas	4800006
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 282 – Resultado.

Saída	
Tempo (Segundos)	2.2028
N° de Comparações	6
N° de Trocas	5400006
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 283 – Resultado.

Saída	
Tempo (Segundos)	2.8295
N° de Comparações	7
N° de Trocas	7000007
Comprimento do Vetor	1 000 000

2.8.3 Ordenação Aleatória

Para essa seção vamos assumir uma lista desordenada de valores dado cada tamanho proposto.

- Vetor de 10^3 posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 284 – Resultado.

Saída	
Tempo (Segundos)	0.0037
N° de Comparações	4
N° de Trocas	4004
Comprimento do Vetor	1 000

- Vetor de 10^4 posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 285 – Resultado.

Saída	
Tempo (Segundos)	0.0275
N° de Comparações	5
N° de Trocas	50005
Comprimento do Vetor	10 000

- Vetor de 10^5 posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 286 – Resultado.

Saída	
Tempo (Segundos)	0.2137
N° de Comparações	6
N° de Trocas	600006
Comprimento do Vetor	100 000

- Vetor de $2 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 287 – Resultado.

Saída	
Tempo (Segundos)	0.5052
N° de Comparações	6
N° de Trocas	1200006
Comprimento do Vetor	200 000

- Vetor de $3 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 288 – Resultado.

Saída	
Tempo (Segundos)	0.8909
N° de Comparações	6
N° de Trocas	1800006
Comprimento do Vetor	300 000

- Vetor de $4 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 289 – Resultado.

Saída	
Tempo (Segundos)	1.1645
N° de Comparações	6
N° de Trocas	2400006
Comprimento do Vetor	400 000

- Vetor de $5 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 290 – Resultado.

Saída	
Tempo (Segundos)	1.6165
N° de Comparações	6
N° de Trocas	3000006
Comprimento do Vetor	500 000

- Vetor de $6 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 291 – Resultado.

Saída	
Tempo (Segundos)	1.9171
N° de Comparações	6
N° de Trocas	3600006
Comprimento do Vetor	600 000

- Vetor de $7 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 292 – Resultado.

Saída	
Tempo (Segundos)	2.2288
N° de Comparações	6
N° de Trocas	4200006
Comprimento do Vetor	700 000

- Vetor de $8 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 293 – Resultado.

Saída	
Tempo (Segundos)	2.5409
N° de Comparações	6
N° de Trocas	4800006
Comprimento do Vetor	800 000

- Vetor de $9 * 10^5$ posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 294 – Resultado.

Saída	
Tempo (Segundos)	2.9309
N° de Comparações	6
N° de Trocas	5400006
Comprimento do Vetor	900 000

- Vetor de 10^6 posições. Executando o algoritmo de ordenação radix, obtemos como resultado:

Tabela 295 – Resultado.

Saída	
Tempo (Segundos)	4.0856
N° de Comparações	7
N° de Trocas	7000007
Comprimento do Vetor	1 000 000

2.8.4 Comparativos

Tabela 296 – Resultados Comparativos.

Comparações				
Ordenação	Comprimento	Tempo (s)	N° Comparações	N° Trocas
Crescente	1000	0.0036	4	4004
	10000	0.0237	5	50005
	100000	0.2039	6	600006
	200000	0.4274	6	1200006
	300000	0.6722	6	1800006
	400000	0.8897	6	2400006
	500000	1.1325	6	3000006
	600000	1.4048	6	3600006
	700000	1.7977	6	4200006
	800000	1.9558	6	4800006
	900000	2.1013	6	5400006
	1000000	2.9429	7	7000007
Decrescente	1000	0.0036	4	4004
	10000	0.0285	5	50005
	100000	0.2132	6	600006
	200000	0.4298	6	1200006
	300000	0.6655	6	1800006
	400000	0.9059	6	2400006
	500000	1.1941	6	3000006
	600000	1.4038	6	3600006
	700000	1.6313	6	4200006
	800000	1.9599	6	4800006
	900000	2.2028	6	5400006
	1000000	2.8295	7	7000007
Aleatória	1000	0.0037	4	4004
	10000	0.0275	5	50005
	100000	0.2137	6	600006
	200000	0.5052	6	1200006
	300000	0.8909	6	1800006
	400000	1.1645	6	2400006
	500000	1.6165	6	3000006
	600000	1.9171	6	3600006
	700000	2.2288	6	4200006
	800000	2.5409	6	4800006
	900000	2.9309	6	5400006
	1000000	4.0856	7	7000007

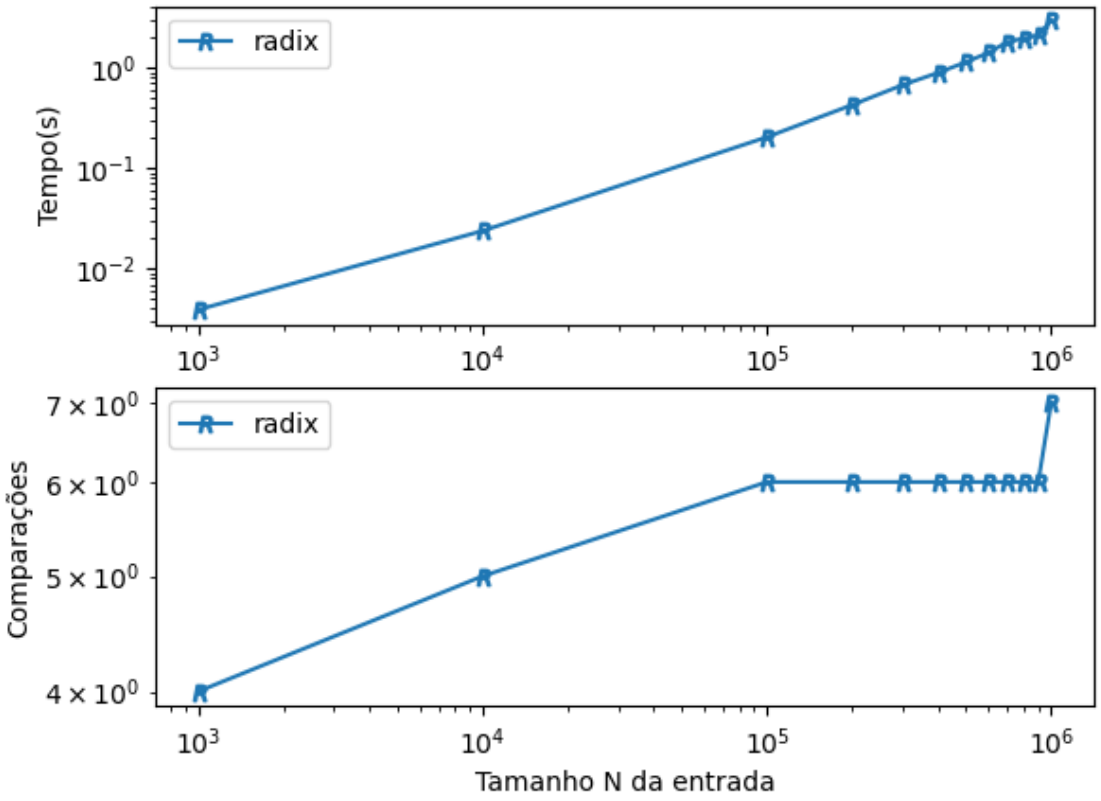

Figura 22 – Representação do comportamento do algoritmo de ordenação radix com uma entrada de tamanho 1000000 ordenadas de forma crescente

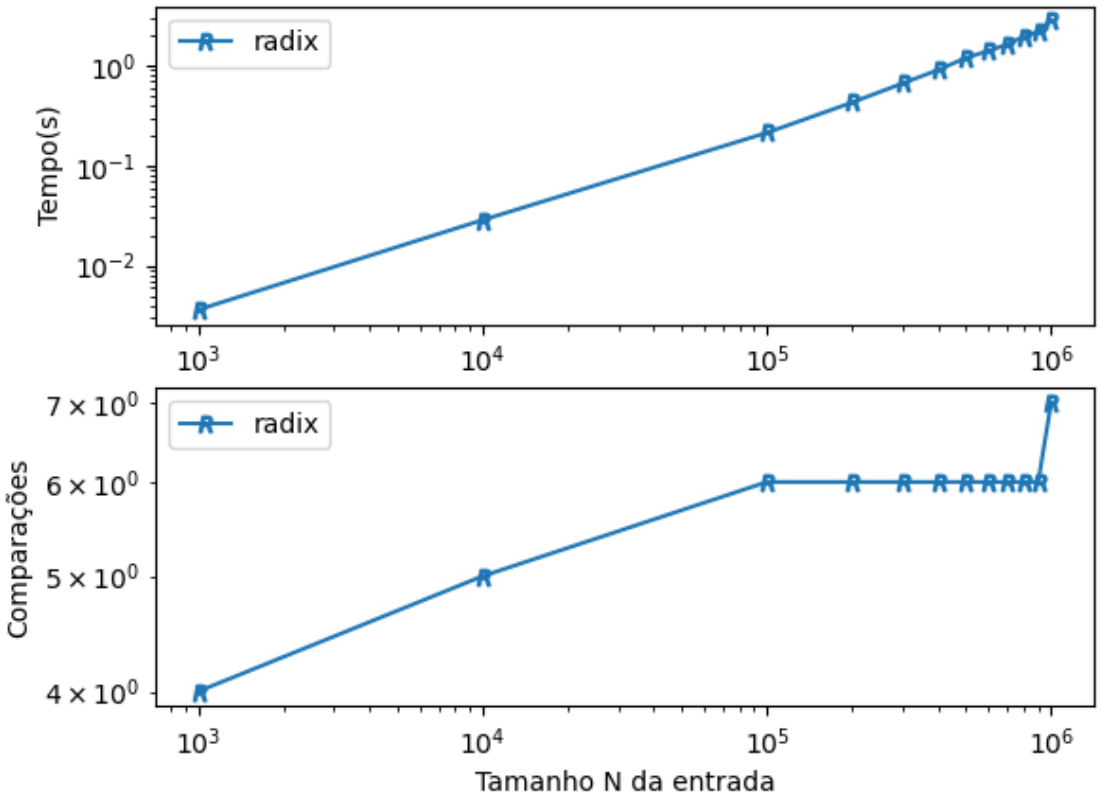

Figura 23 – Representação do comportamento do algoritmo de ordenação radix com uma entrada de tamanho 1000000 ordenadas de forma decrescente

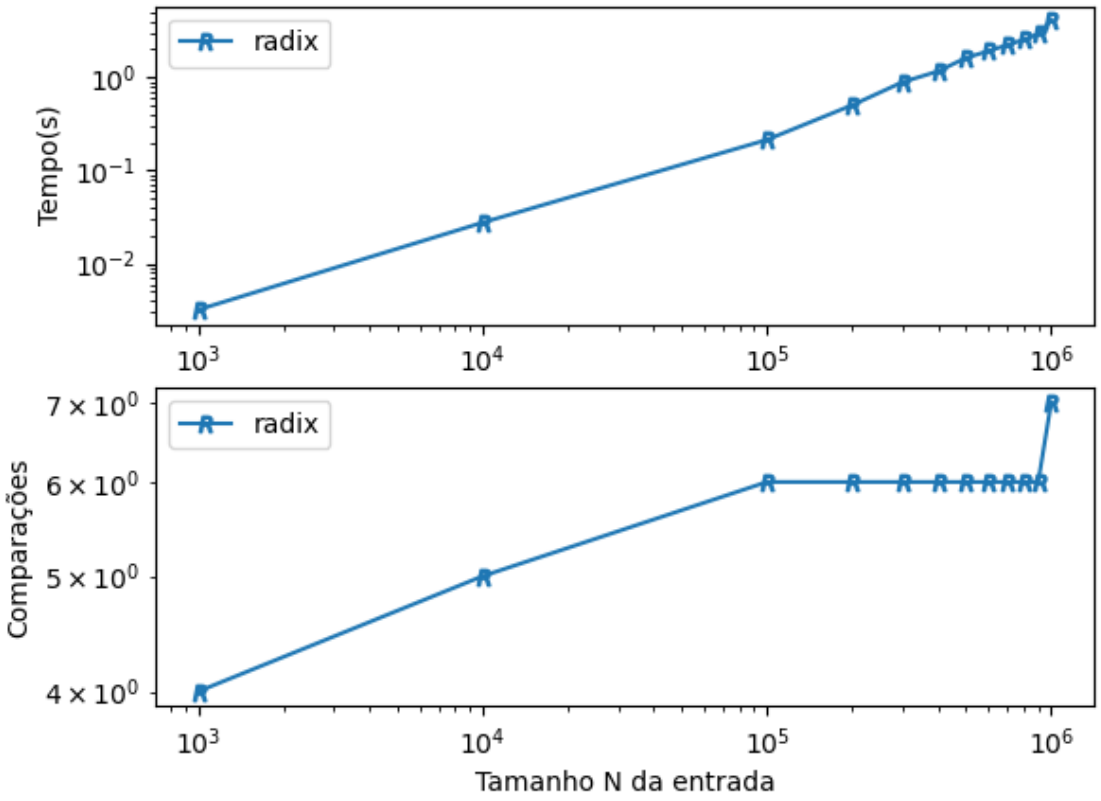

Figura 24 – Representação do comportamento do algoritmo de ordenação radix com uma entrada de tamanho 1000000 desordenadas

3 RESULTADOS

3.1 Comparação Final

Feito todo o experimento avaliando cada algoritmo e suas propriedades podemos realizar uma comparação melhor entre os algoritmos para melhor perceber em que aspecto cada um se destaca mehor. **Observação:** Alguns algoritmos não foram possível coletar algumas informações devido ao alto custo computacional nesse caso eles vão assumir uma posição linear em algum momento do gráfico pois foi repetido o último resultado obtido.

3.1.1 Ordenado Crescente

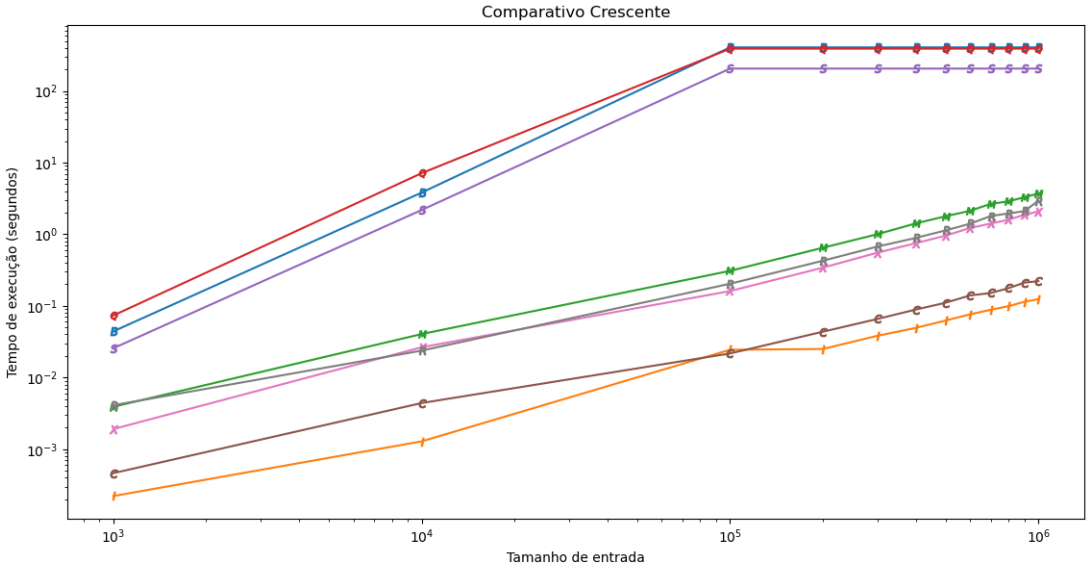

Figura 25 – Representação do comportamento dos algoritmos estudados dado entradas de tamanho até 1000000 em função do tempo

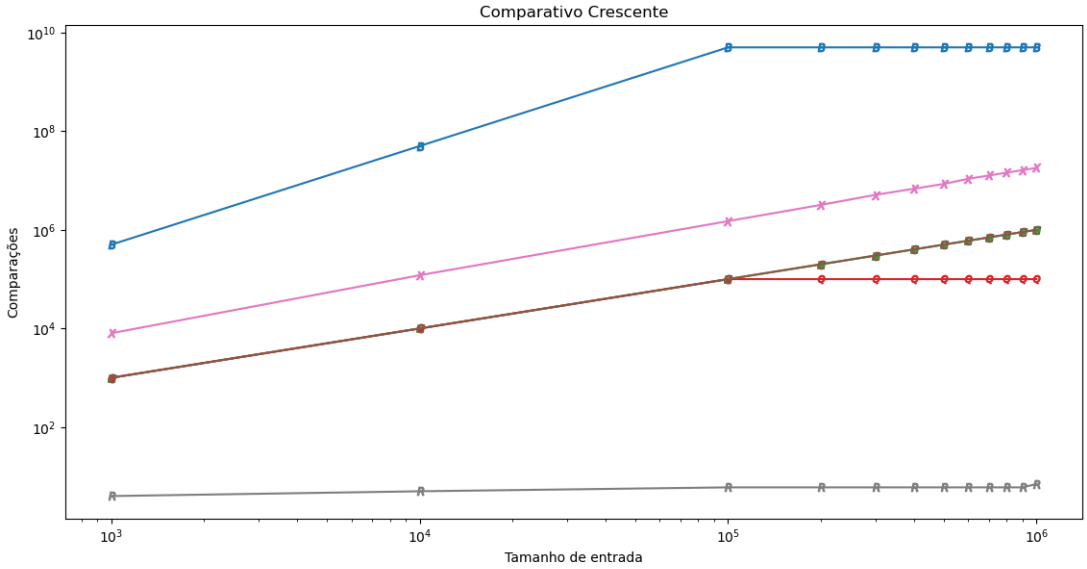

Figura 26 – Representação do comportamento dos algoritmos estudados dado entradas de tamanho até 1000000 em função das comparações

3.1.2 Ordenado Decrescente

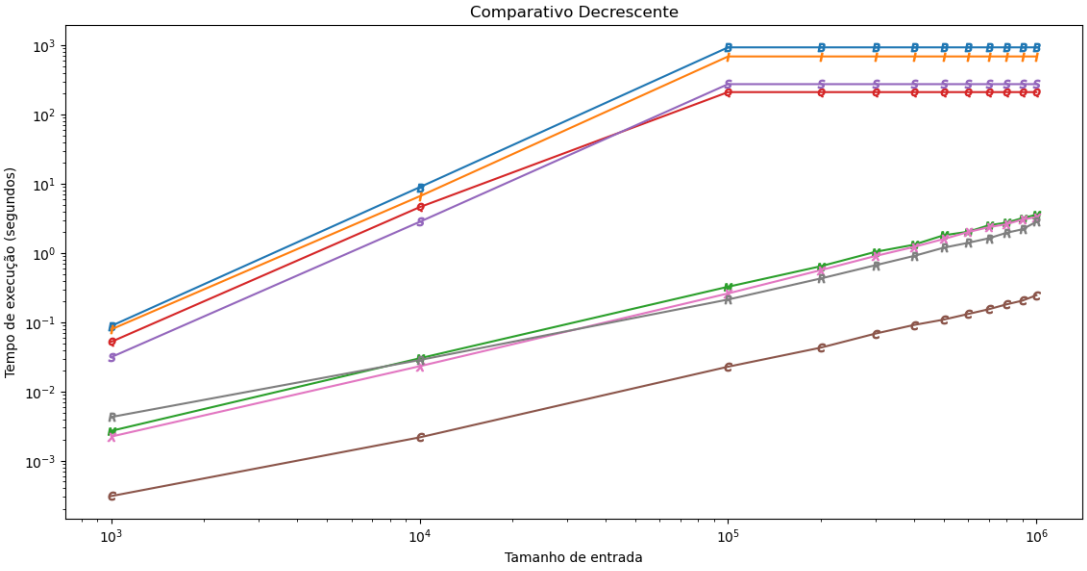

Figura 27 – Representação do comportamento dos algoritmos estudados dado entradas de tamanho até 1000000 em função do tempo

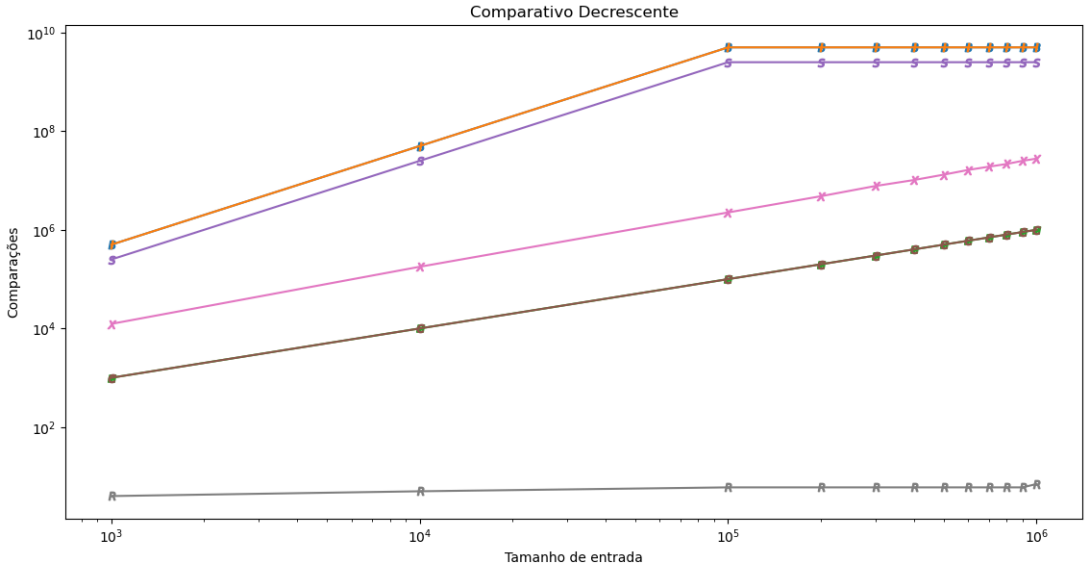

Figura 28 – Representação do comportamento dos algoritmos estudados dado entradas de tamanho até 1000000 em função das comparações

3.1.3 Desordenado

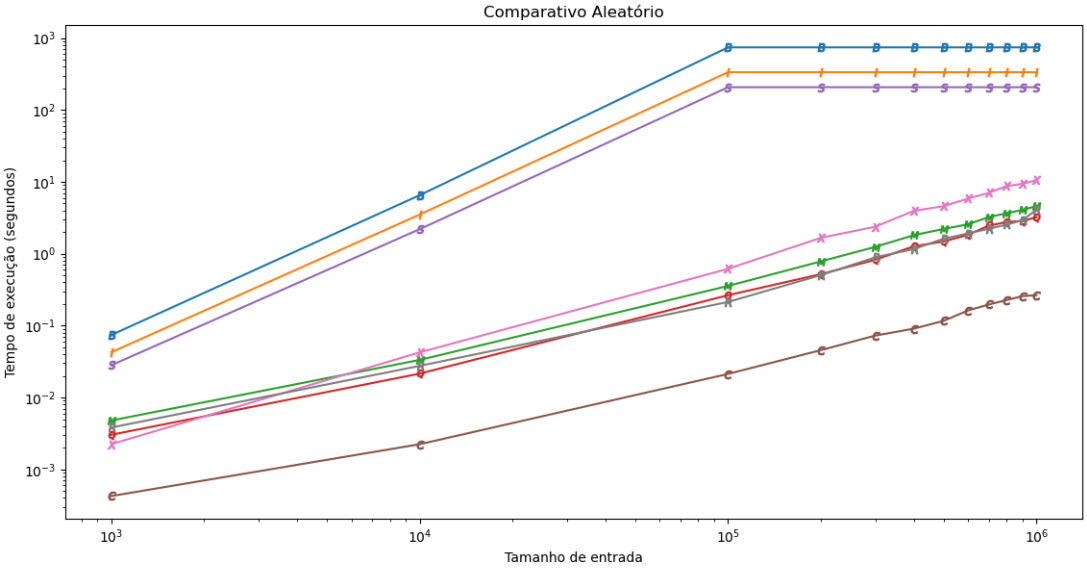

Figura 29 – Representação do comportamento dos algoritmos estudados dado entradas de tamanho até 1000000 em função do tempo

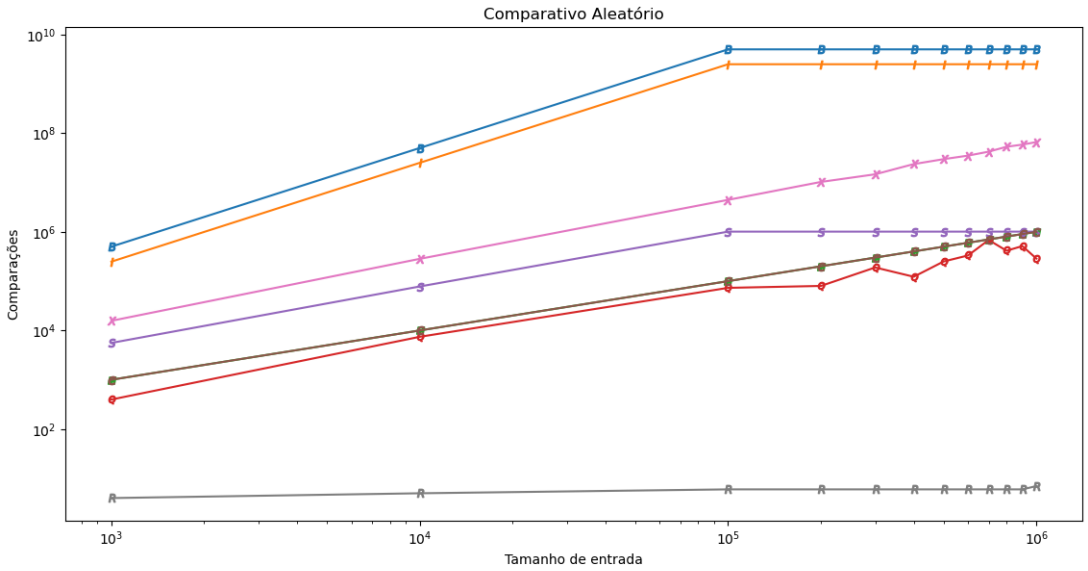

Figura 30 – Representação do comportamento dos algoritmos estudados dado entradas de tamanho até 1000000 em função das comparações

3.1.4 Considerações

Este trabalho serve para compreender o comportamento dos algoritmos de modo a perceber como ele trabalha se realiza comparações ou trocas até mesmo a combinação entre, com isso foi possivel observar que cada um se comporta de uma forma alguns trabalham melhor em diferentes condições, sendo elas as ordenações dessa forma podemos dizer que Quick sort se destaca quando está usando os valores embaralhados, já o Bubble sort em pior de todos em qualquer situação, passando-se horas sem obter resultado. E por fim é possivel perceber que dos mais estudados e famosos algoritmos de ordenação temos o Merge Sort que mais se destaca em qualquer ambiente pois usa-se de um conceito muito usado que é conquistar e dividir no qual tem o artificio de transformar os problemas menos, subdividindo tarefas de modo que reduza o trabalho nas comparações de maneira que em todos os seus casos se é esperado o suficiente a $\mathcal{O}(n \log n)$.