Trabalho de análise de algoritmos de ordenação - Projeto e Análise de Algoritmos

Clarisse Midori Yoshimura Torres
Maria Carolina Tomain Rodrigues

Outubro, 2024

Instituto de Biociências, Letras e Ciências Exatas, Unesp - Universidade Estadual Paulista

Sumário

Lista de Figuras

1 Introdução

O trabalho tem como objetivo analisar comparativamente o desempenho entre diferentes algoritmos de ordenação, fazendo a avaliação de seus tempos de execução em diferentes cenários.

A pesquisa foi feita com uso da linguagem de programação Python para implementar 10 algoritmos distintos de ordenação com entradas que variam de dez a um milhão de valores. Estes foram distribuídos em um vetor de entrada de três formas diferentes, organizados de forma aleatória, ordenada e inversamente ordenada.

Neste relatório serão expostas comparações de valores esperados e reais de tempo para cada execução, comentários sobre o desempenho e funcionamento de cada algoritmo e também gráficos demonstrando de forma visual a evolução dos tempos de execuções e complexidades de cada algoritmo.

2 Bubble Sort

2.1 Funcionamento do Bubble Sort

Bubble sort é um algoritmo de classificação que compara dois elementos adjacentes e os troca até que estejam na ordem pretendida.

Assim como o movimento das bolhas de ar na água que sobem à superfície, cada elemento do array se move até o final em cada iteração. Portanto, é chamado de classificação por bolha em sua tradução literal.

2.2 Gráficos plotados do Bubble Sort

2.2.1 Evolução do tempo com entrada aleatória

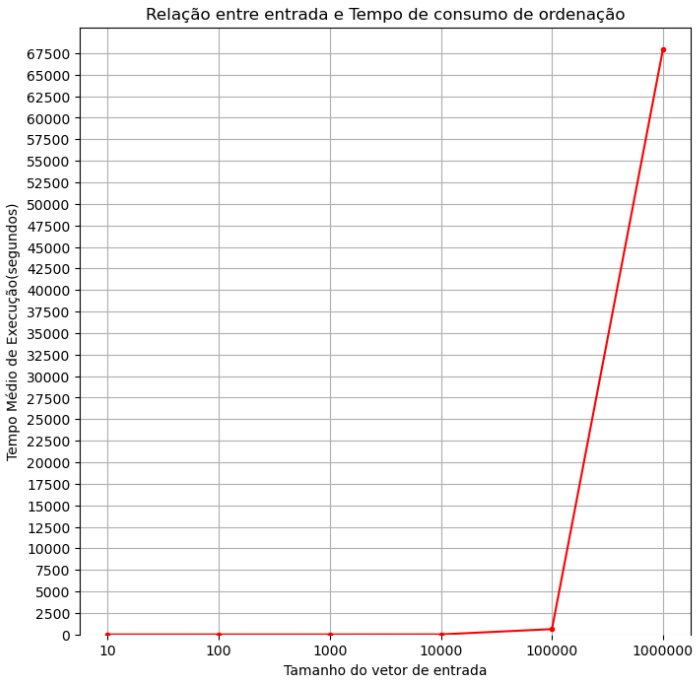

Figura 1: Plotagem do gráfico de evolução temporal com entradas aleatórias.

2.2.2 Evolução do tempo com entrada ordenada

Vale ressaltar que quando a entrada para o algoritmo Bubble Sort estiver ordenada, o mesmo estará no melhor caso, pois este lê o vetor inteiro a princípio para depois começar a ordenação, e neste caso, o mesmo verifica que o vetor já encontra-se ordenado.

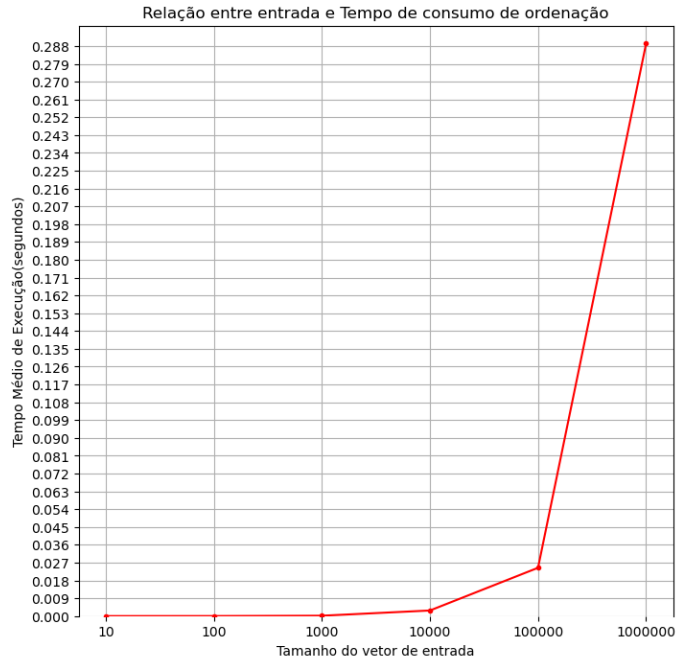

Figura 2: Plotagem do gráfico de evolução temporal com entradas ordenadas.

2.2.3 Evolução do tempo com entrada inversamente ordenada

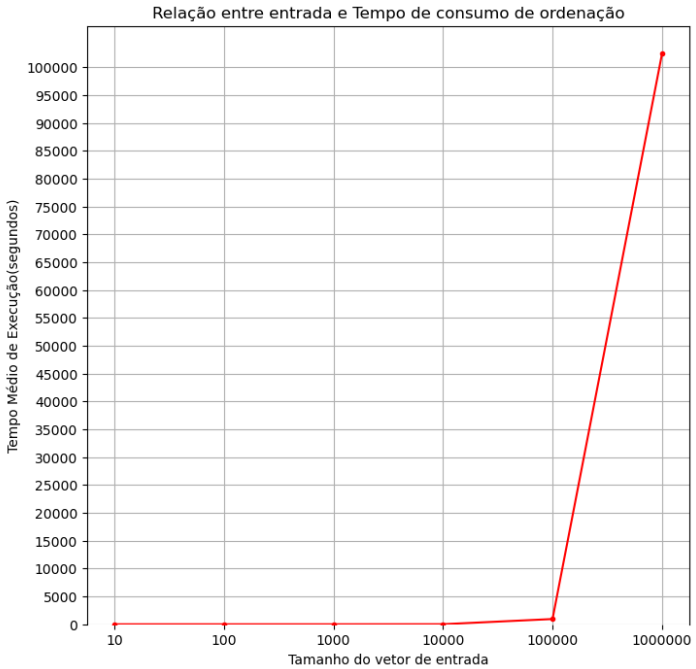

Figura 3: Plotagem do gráfico de evolução temporal com entradas inversamente ordenadas.

2.2.4 Comparação entre os três tipos de entradas

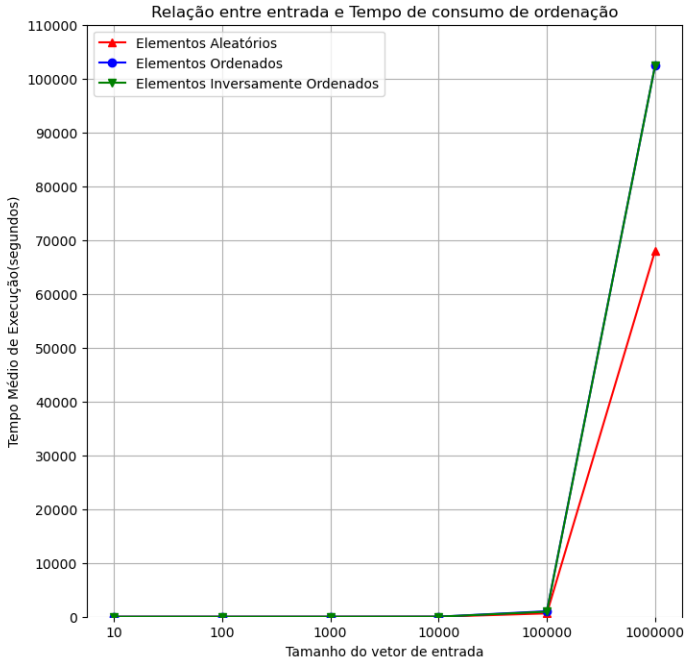

Figura 4: Plotagem do gráfico de comparação entre as diferentes entradas.

3 Bucket Sort

3.1 Funcionamento do Bucket Sort

Bucket Sort é um algoritmo de classificação que divide os elementos não classificados da matriz em vários grupos chamados buckets. Cada intervalo é então classificado usando qualquer um dos algoritmos de classificação adequados (neste caso foi utilizado o Counting Sort, que se mostrou mais rápido que os demais) ou aplicando recursivamente o mesmo algoritmo de intervalo.

Finalmente, os intervalos classificados são combinados para formar uma matriz classificada final.

O processo de classificação por bucket pode ser entendido como uma abordagem de coleta dispersa. Aqui, os elementos são primeiro espalhados em grupos e, em seguida, os elementos de cada grupo são classificados. Finalmente, os elementos são reunidos em ordem.

3.2 Gráficos plotados do Bucket Sort

3.2.1 Evolução do tempo com entrada aleatória

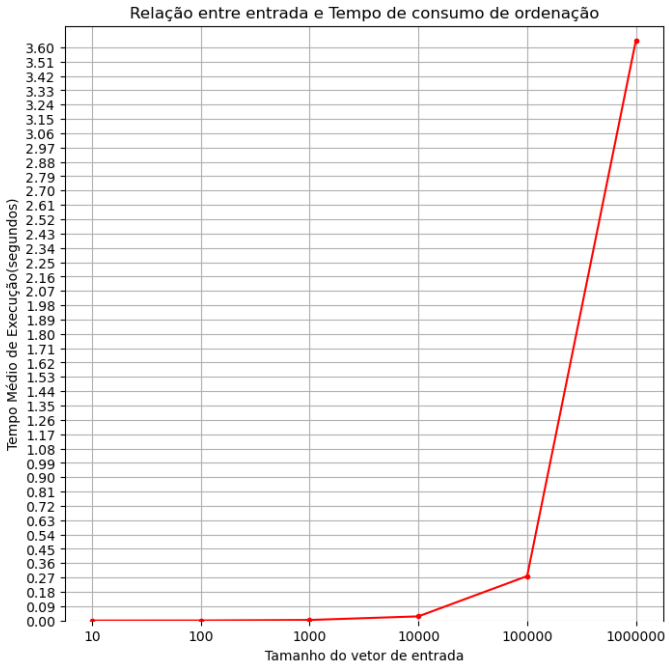

Figura 5: Plotagem do gráfico de evolução temporal com entradas aleatórias.

3.2.2 Evolução do tempo com entrada ordenada

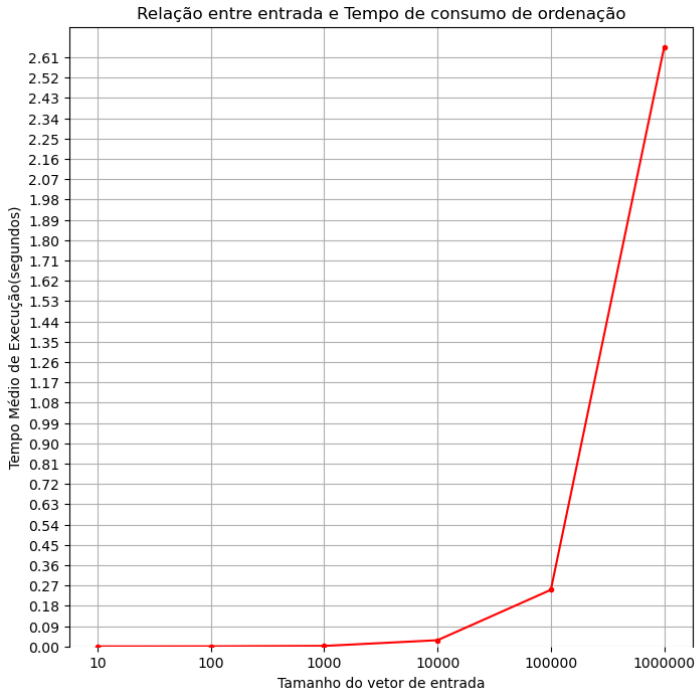

Figura 6: Plotagem do gráfico de evolução temporal com entradas ordenadas.

3.2.3 Evolução do tempo com entrada inversamente ordenada

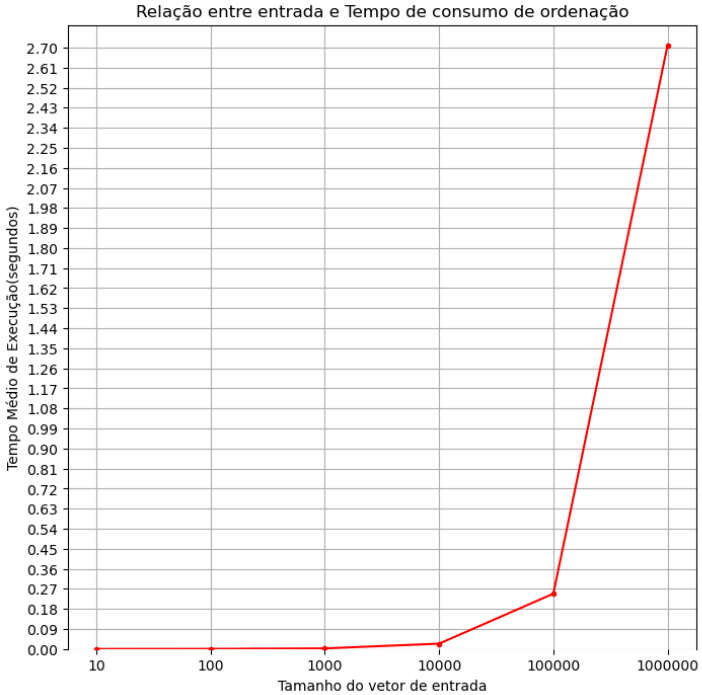

Figura 7: Plotagem do gráfico de evolução temporal com entradas inversamente ordenadas.

3.2.4 Comparação entre os três tipos de entradas

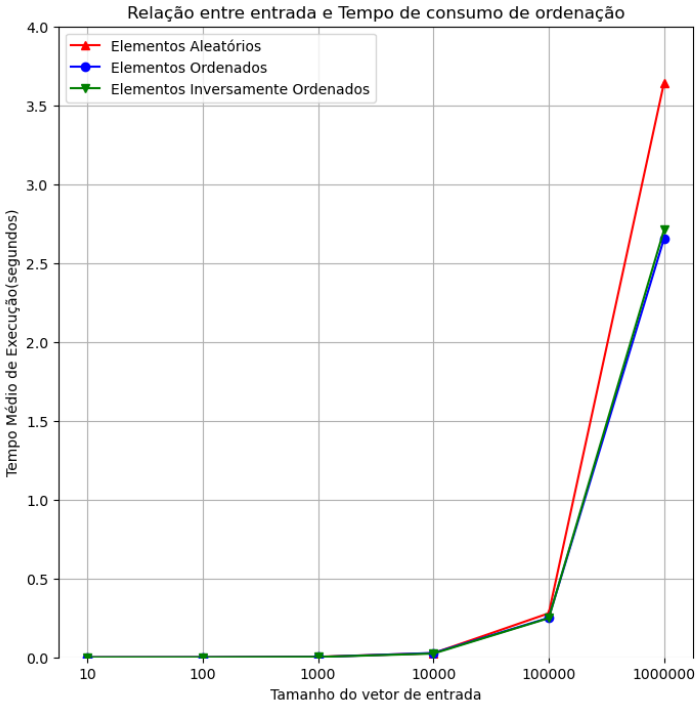

Figura 8: Plotagem do gráfico de comparação entre as diferentes entradas.

4 Counting Sort

4.1 Funcionamento do Counting Sort

O counting sort é um algoritmo de classificação que classifica os elementos de uma matriz contando o número de ocorrências de cada elemento exclusivo na matriz. A contagem é armazenada em um array auxiliar e a classificação é feita mapeando a contagem como um índice do array auxiliar.

4.2 Gráficos plotados do Counting Sort

4.2.1 Evolução do tempo com entrada aleatória

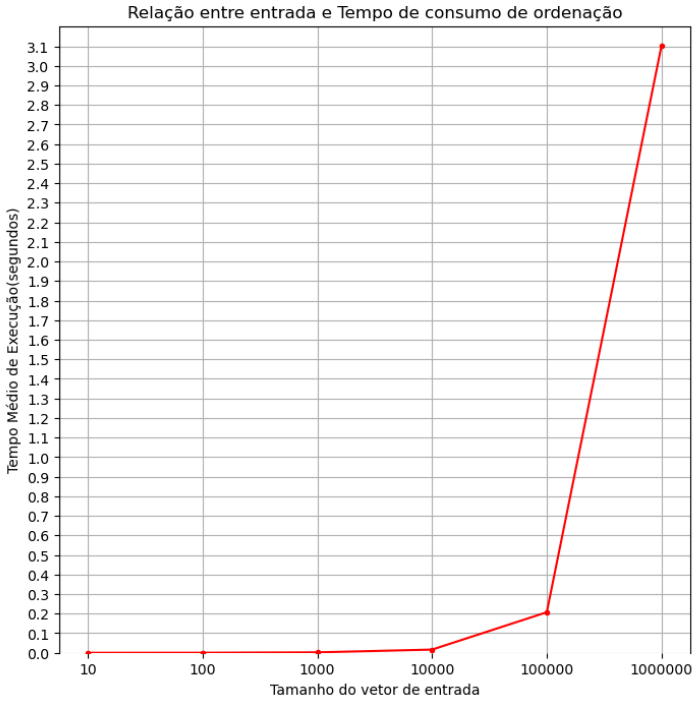

Figura 9: Plotagem do gráfico de evolução temporal com entradas aleatórias.

4.2.2 Evolução do tempo com entrada ordenada

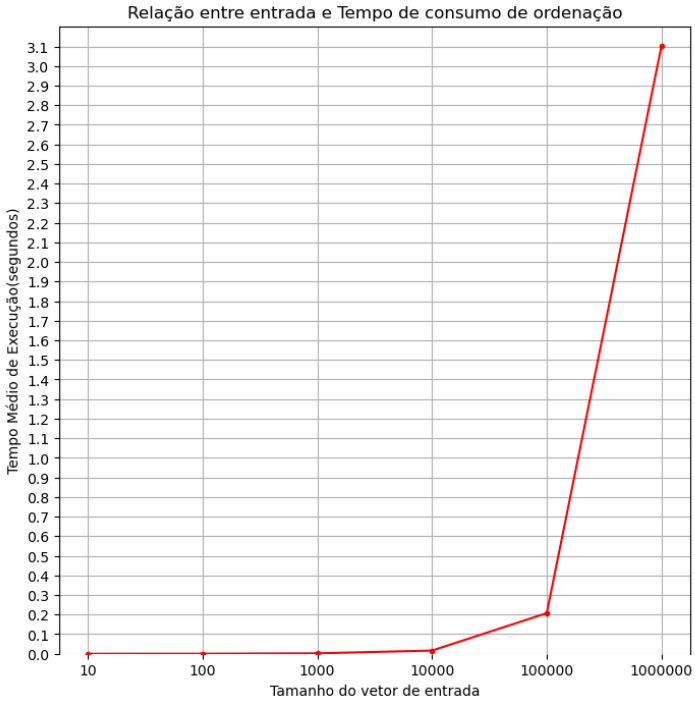

Figura 10: Plotagem do gráfico de evolução temporal com entradas ordenadas.

4.2.3 Evolução do tempo com entrada inversamente ordenada

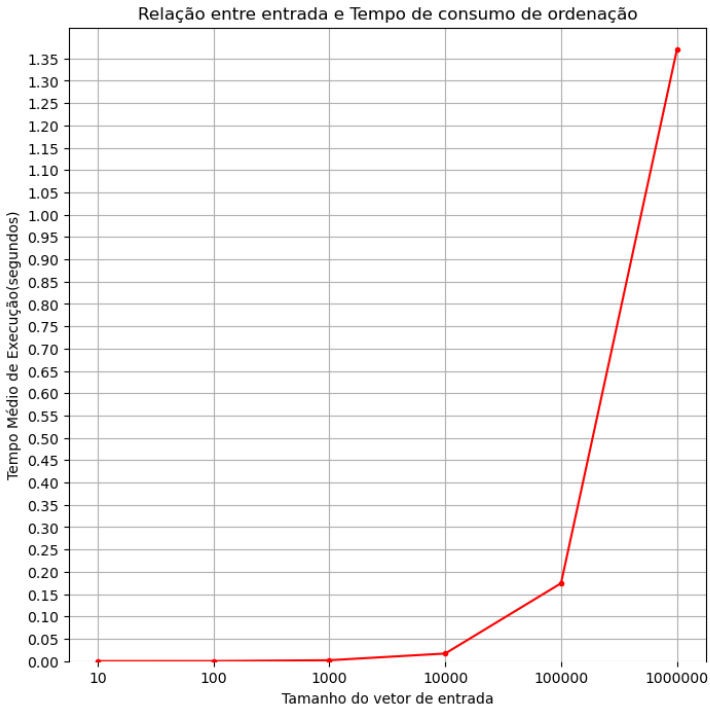

Figura 11: Plotagem do gráfico de evolução temporal com entradas inversamente ordenadas.

4.2.4 Comparação entre os três tipos de entradas

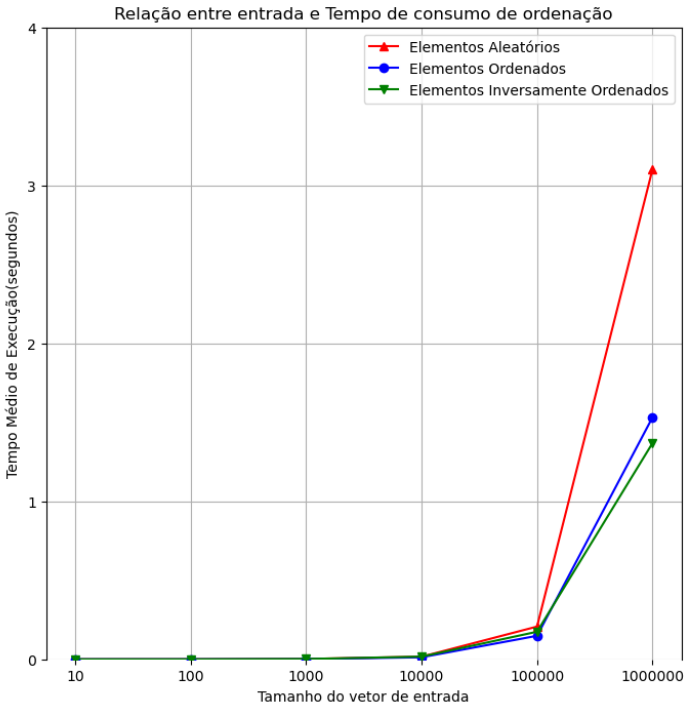

Figura 12: Plotagem do gráfico de comparação entre as diferentes entradas.

5 Radix Sort

5.1 Funcionamento do Radix Sort

Radix sort é um algoritmo de classificação que classifica os elementos agrupando primeiro os dígitos individuais do mesmo valor posicional. Em seguida, classifica os elementos de acordo com sua ordem crescente/decrescente.

5.2 Gráficos plotados do Radix Sort

5.2.1 Evolução do tempo com entrada aleatória

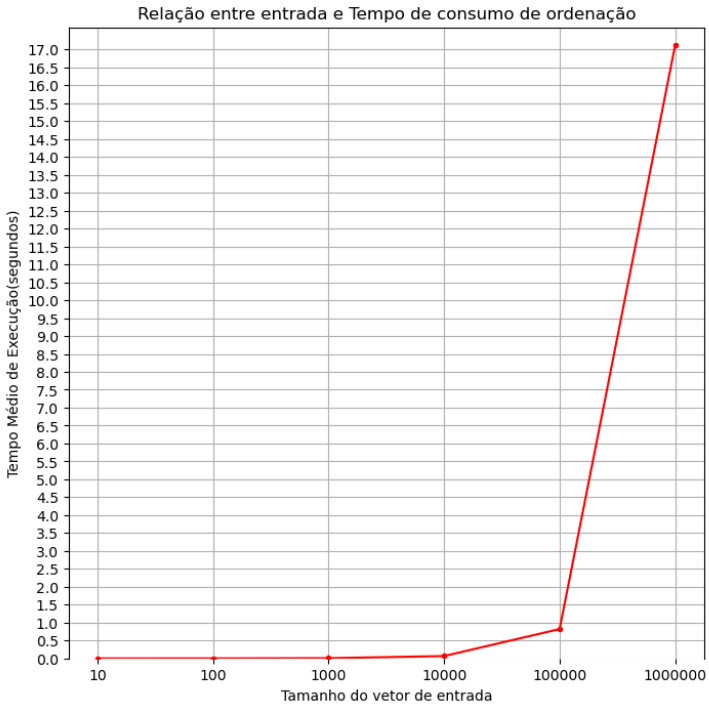

Figura 13: Plotagem do gráfico de evolução temporal com entradas aleatórias.

5.2.2 Evolução do tempo com entrada ordenada

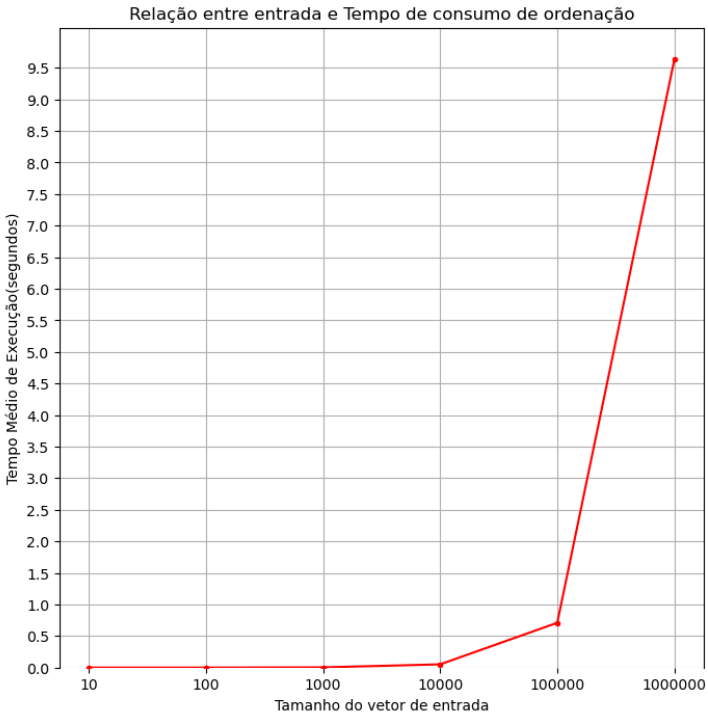

Figura 14: Plotagem do gráfico de evolução temporal com entradas ordenadas.

5.2.3 Evolução do tempo com entrada inversamente ordenada

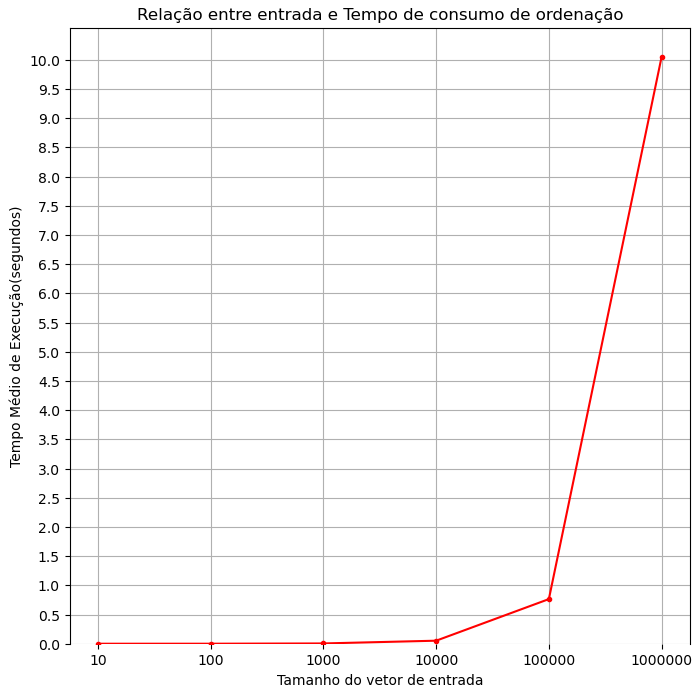

Figura 15: Plotagem do gráfico de evolução temporal com entradas inversamente ordenadas.

5.2.4 Comparação entre os três tipos de entradas

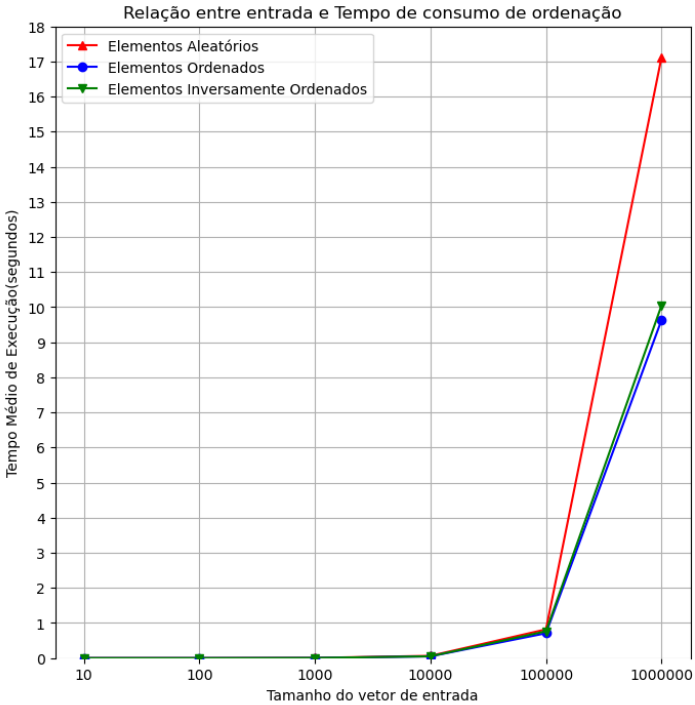

Figura 16: Plotagem do gráfico de comparação entre as diferentes entradas.

6 Heap Sort

6.1 Funcionamento do Heap Sort

Heap Sort é um algoritmo de classificação popular e eficiente em programação de computadores. Aprender como escrever o algoritmo de classificação de heap requer conhecimento de dois tipos de estruturas de dados, arrays e árvores.

O conjunto inicial de números que queremos classificar é armazenado em uma matriz, por exemplo. [10, 3, 76, 34, 23, 32] e após a classificação, obtemos um array classificado [3,10,23,32,34,76].

A classificação de heap funciona visualizando os elementos do array como um tipo especial de árvore binária completa chamada heap.

6.2 Gráficos plotados do Heap Sort

6.2.1 Evolução do tempo com entrada aleatória

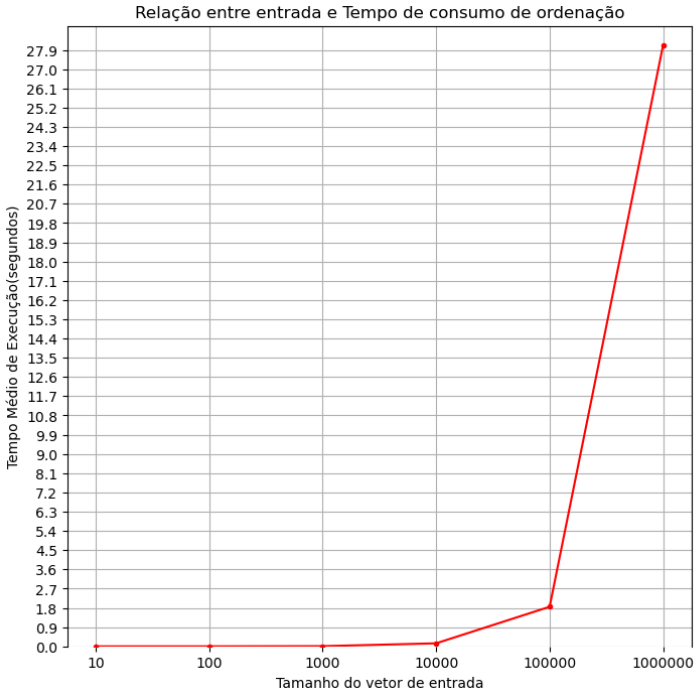

Figura 17: Plotagem do gráfico de evolução temporal com entradas aleatórias.

6.2.2 Evolução do tempo com entrada ordenada

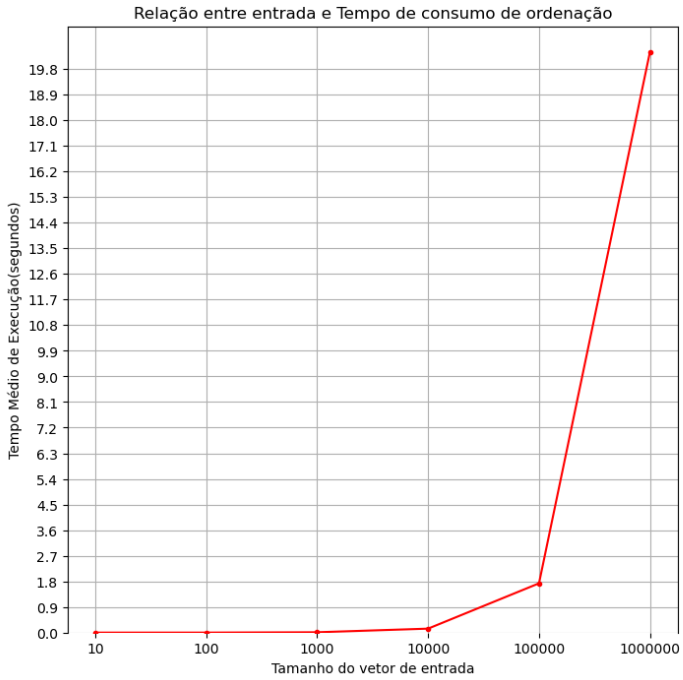

Figura 18: Plotagem do gráfico de evolução temporal com entradas ordenadas.

6.2.3 Evolução do tempo com entrada inversamente ordenada

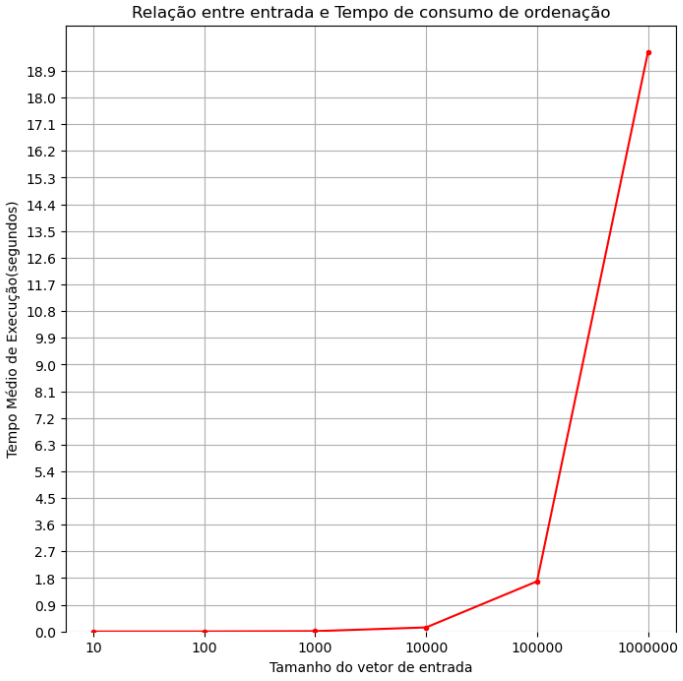

Figura 19: Plotagem do gráfico de evolução temporal com entradas inversamente ordenadas.

6.2.4 Comparação entre os três tipos de entradas

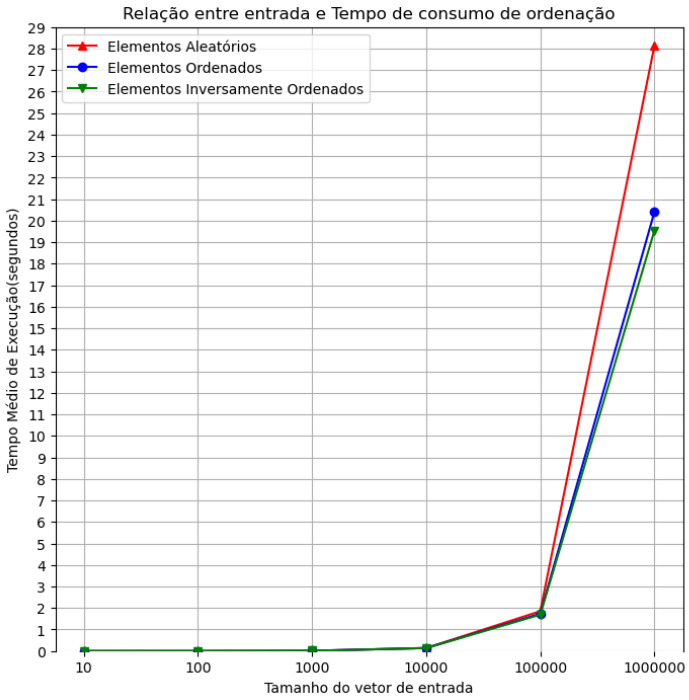

Figura 20: Plotagem do gráfico de comparação entre as diferentes entradas.

7 Insertion Sort

7.1 Funcionamento do Insertion Sort

A classificação por inserção é um algoritmo de classificação que coloca um elemento não classificado em seu local adequado em cada iteração.

A classificação por inserção funciona de maneira semelhante à que classificamos as cartas em nossas mãos em um jogo de cartas.

Assumimos que o primeiro cartão já está classificado e então selecionamos um cartão não classificado. Se a carta não ordenada for maior que a carta em mãos, ela é colocada à direita, caso contrário, à esquerda. Da mesma forma, outras cartas não classificadas são retiradas e colocadas em seus devidos lugares.

7.2 Gráficos plotados do Insertion Sort

7.2.1 Evolução do tempo com entrada aleatória

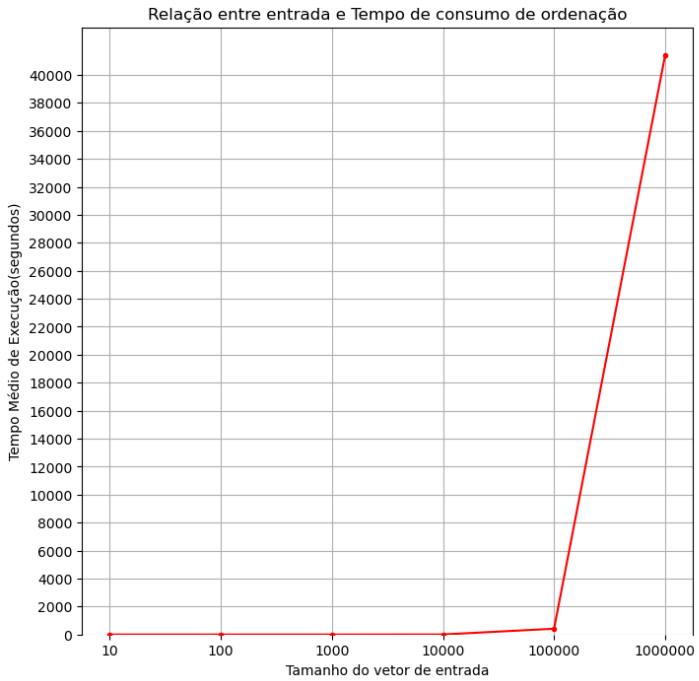

Figura 21: Plotagem do gráfico de evolução temporal com entradas aleatórias.

7.2.2 Evolução do tempo com entrada ordenada

É interessante notar que este, assim como ocorre no Bubble Sort, é o melhor caso de ordenação por insertion também, já que o algoritmo percorre o vetor da mesma maneira que nós lemos um texto e verifica rapidamente que o mesmo já está ordenado.

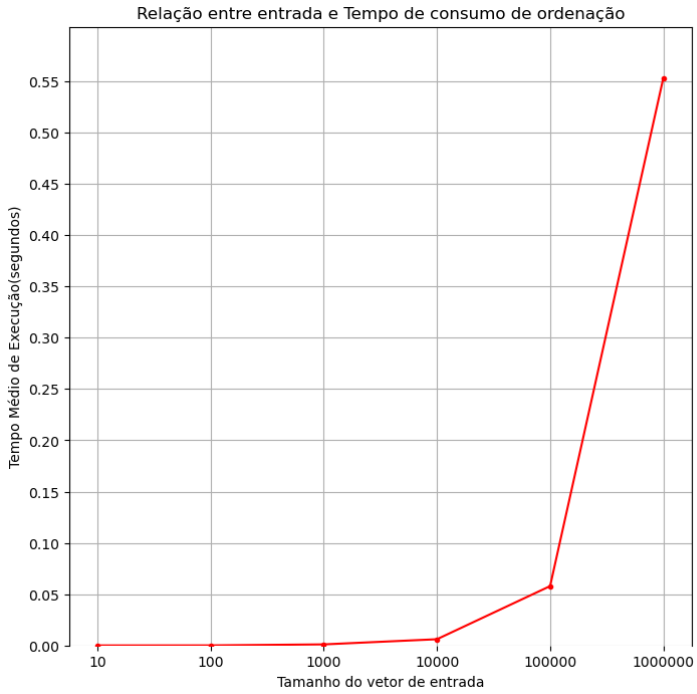

Figura 22: Plotagem do gráfico de evolução temporal com entradas ordenadas.

7.2.3 Evolução do tempo com entrada inversamente ordenada

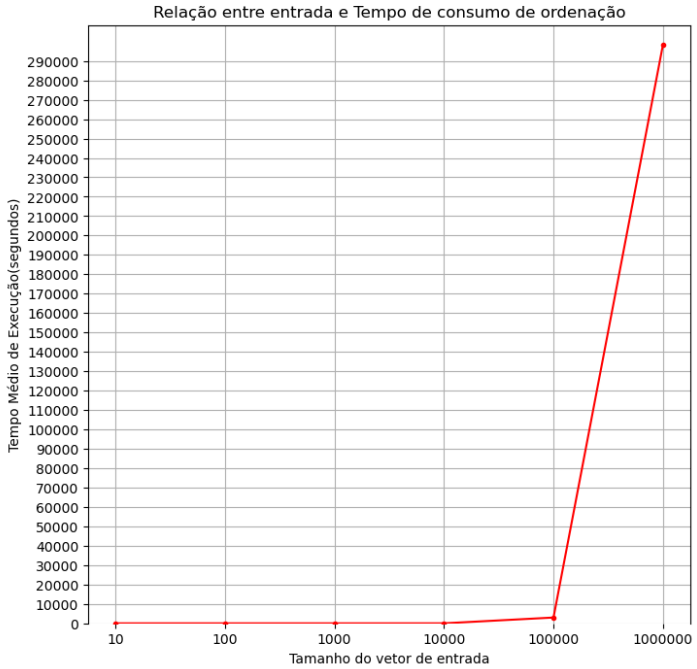

Figura 23: Plotagem do gráfico de evolução temporal com entradas inversamente ordenadas.

7.2.4 Comparação entre os três tipos de entradas

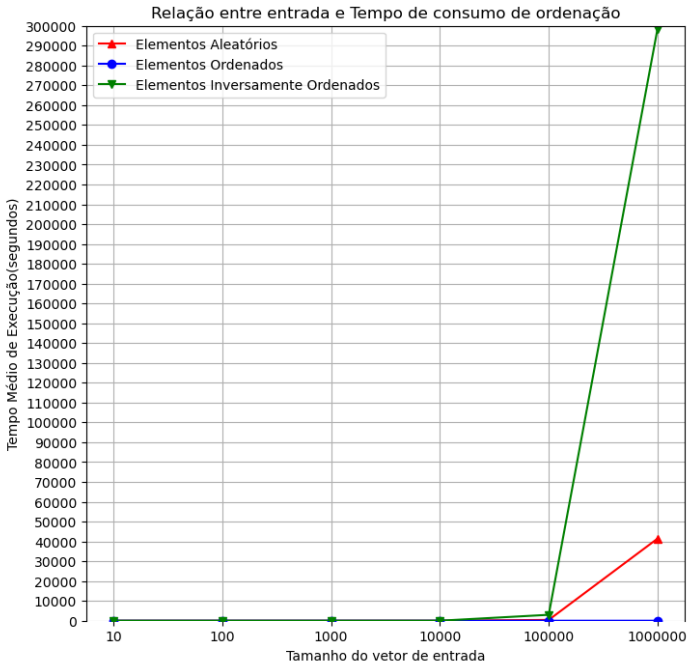

Figura 24: Plotagem do gráfico de comparação entre as diferentes entradas.

8 Merge Sort

8.1 Funcionamento do Merge Sort

Merge Sort é um dos algoritmos de classificação mais populares, baseado no princípio do algoritmo Dividir e Conquistar.

Neste, um problema é dividido em vários subproblemas. Cada subproblema é resolvido individualmente. Finalmente, os subproblemas são combinados para formar a solução final.

8.2 Gráficos plotados do Merge Sort

8.2.1 Evolução do tempo com entrada aleatória

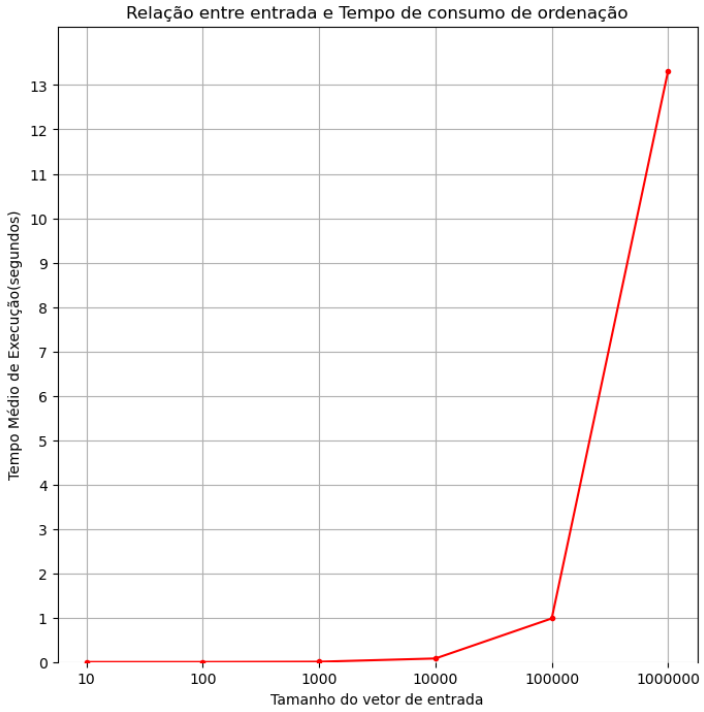

Figura 25: Plotagem do gráfico de evolução temporal com entradas aleatórias.

8.2.2 Evolução do tempo com entrada ordenada

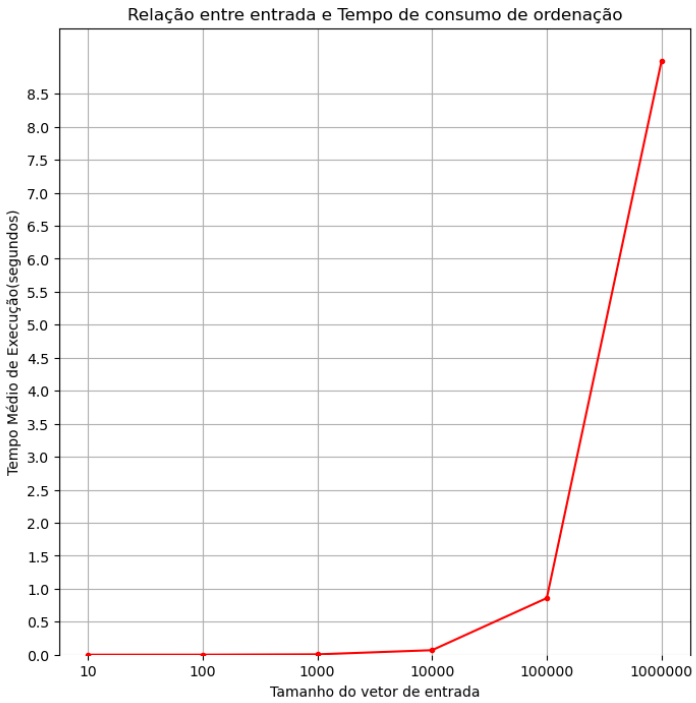

Figura 26: Plotagem do gráfico de evolução temporal com entradas ordenadas.

8.2.3 Evolução do tempo com entrada inversamente ordenada

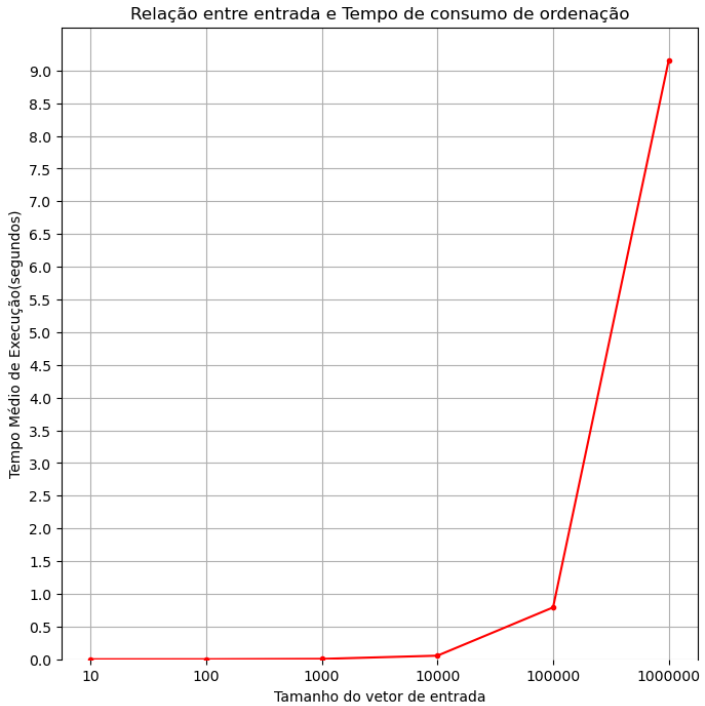

Figura 27: Plotagem do gráfico de evolução temporal com entradas inversamente ordenadas.

8.2.4 Comparação entre os três tipos de entradas

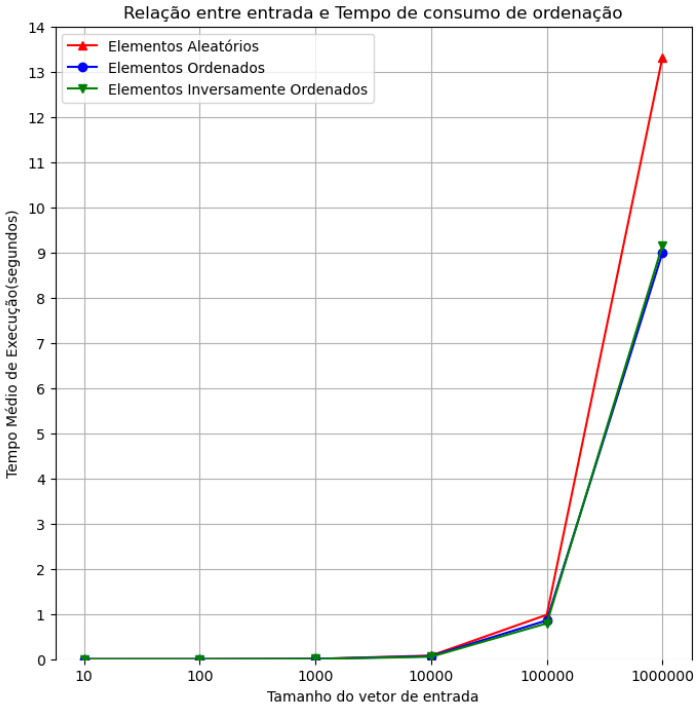

Figura 28: Plotagem do gráfico de comparação entre as diferentes entradas.

9 Quick Sort

9.1 Funcionamento do Quick Sort

Quicksort é um algoritmo de classificação baseado na abordagem de dividir e conquistar para obter as mesmas vantagens do Merge Sort, No Quick Sort uma matriz é dividida em submatrizes selecionando um elemento pivô (elemento selecionado da matriz).

Ao dividir a matriz, o elemento pivô deve ser posicionado de tal forma que os elementos menores que o pivô sejam mantidos no lado esquerdo e os elementos maiores que o pivô fiquem no lado direito do pivô.

As submatrizes esquerda e direita também são divididas usando a mesma abordagem. Este processo continua até que cada submatriz contenha um único elemento.

Neste ponto, os elementos já estão classificados. Finalmente, os elementos são combinados para formar uma matriz classificada(ordenada).

9.2 Gráficos plotados do Quick Sort

9.2.1 Evolução do tempo com entrada aleatória

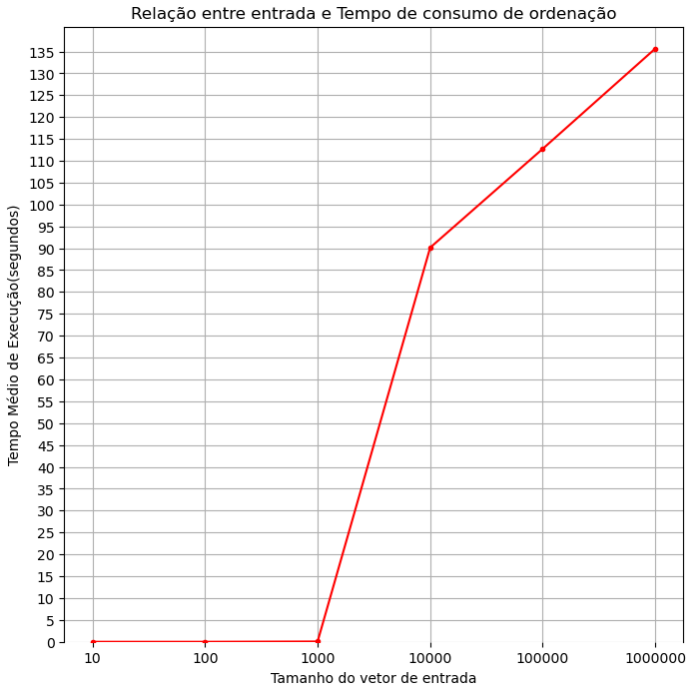

Figura 29: Plotagem do gráfico de evolução temporal com entradas aleatórias.

9.2.2 Evolução do tempo com entrada ordenada

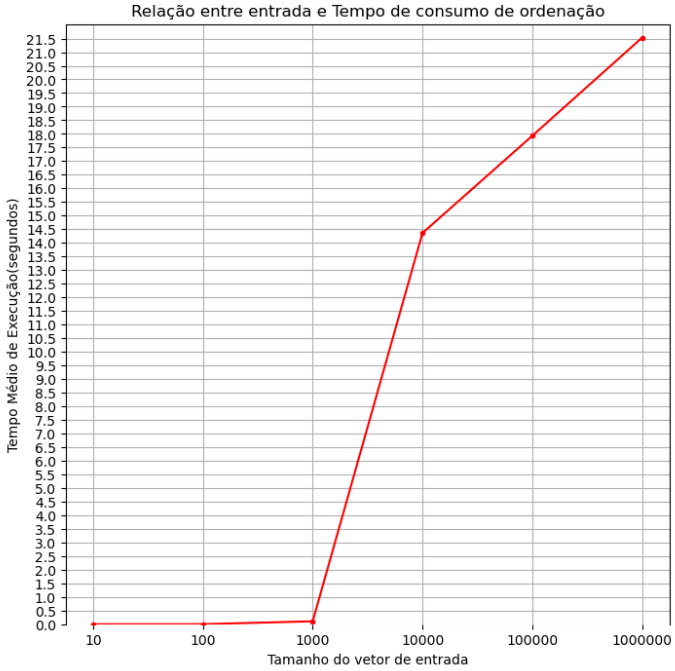

Figura 30: Plotagem do gráfico de evolução temporal com entradas ordenadas.

9.2.3 Evolução do tempo com entrada inversamente ordenada

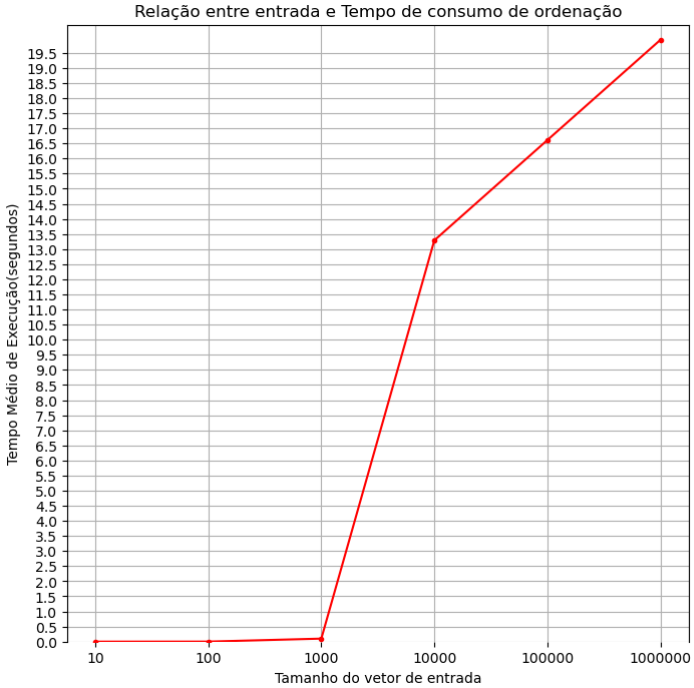

Figura 31: Plotagem do gráfico de evolução temporal com entradas inversamente ordenadas.

9.2.4 Comparação entre os três tipos de entradas

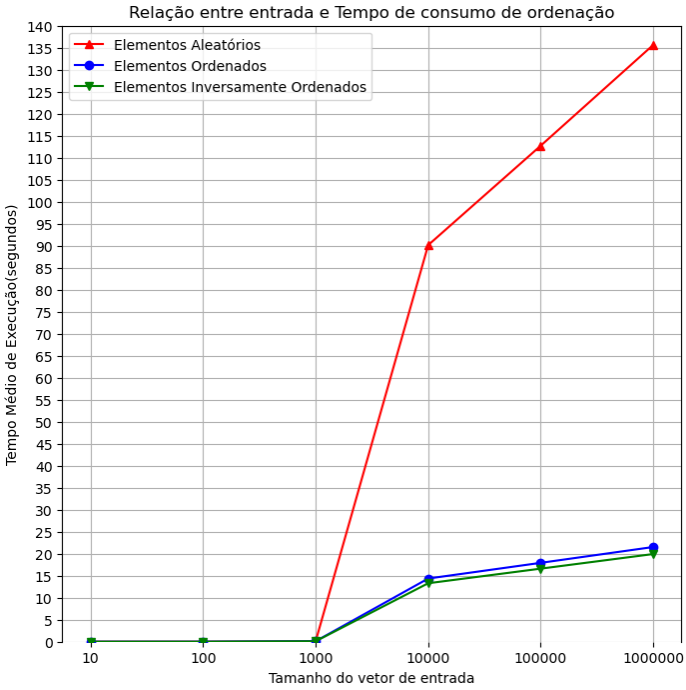

Figura 32: Plotagem do gráfico de comparação entre as diferentes entradas.

10 Selection Sort

10.1 Funcionamento do Selection Sort

O Selection Sort é um algoritmo de classificação que seleciona o menor elemento de uma lista não classificada em cada iteração e coloca esse elemento no início da lista não classificada.

10.2 Gráficos plotados do Selection Sort

10.2.1 Evolução do tempo com entrada aleatória

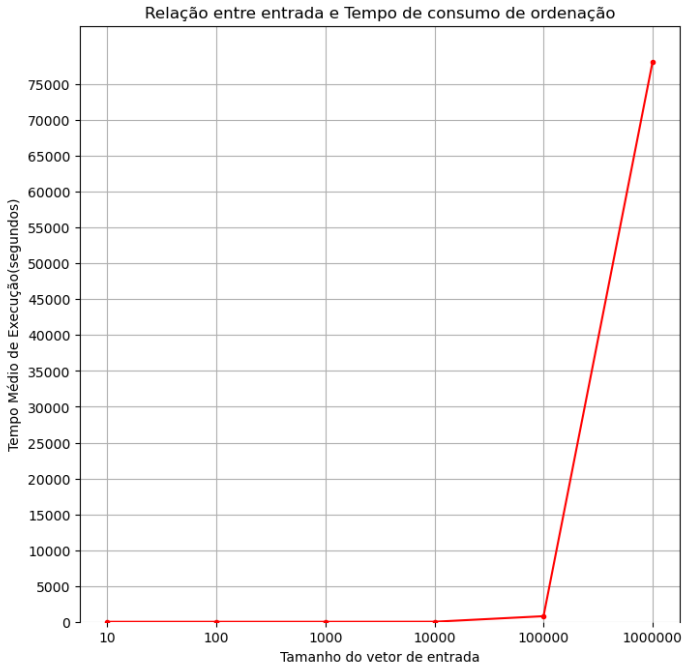

Figura 33: Plotagem do gráfico de evolução temporal com entradas aleatórias.

10.2.2 Evolução do tempo com entrada ordenada

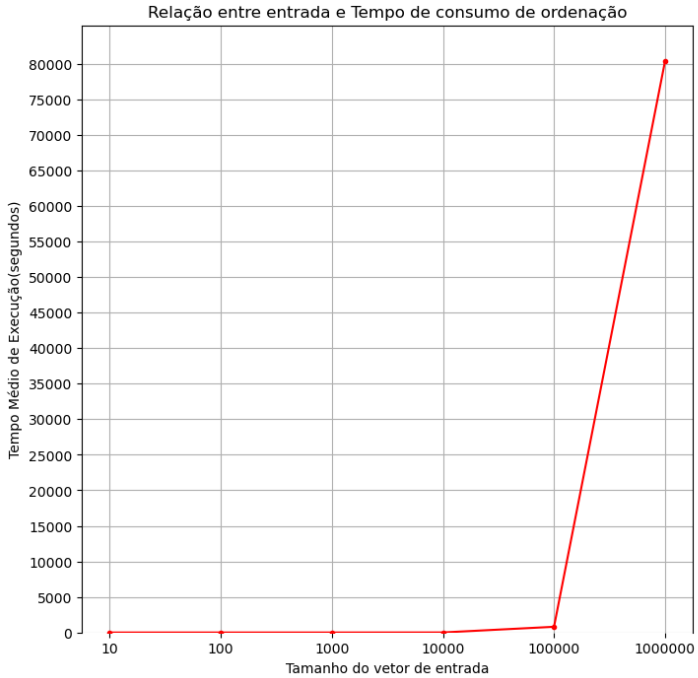

Figura 34: Plotagem do gráfico de evolução temporal com entradas ordenadas.

10.2.3 Evolução do tempo com entrada inversamente ordenada

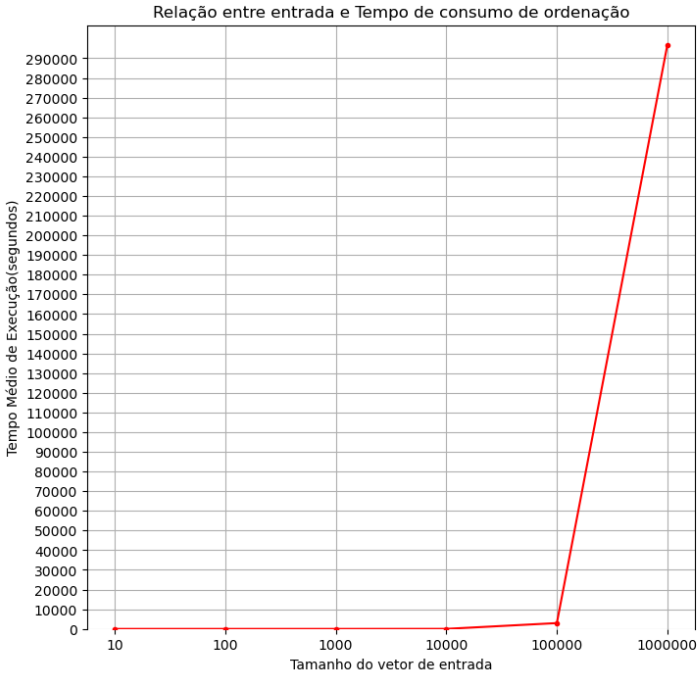

Figura 35: Plotagem do gráfico de evolução temporal com entradas inversamente ordenadas.

10.2.4 Comparação entre os três tipos de entradas

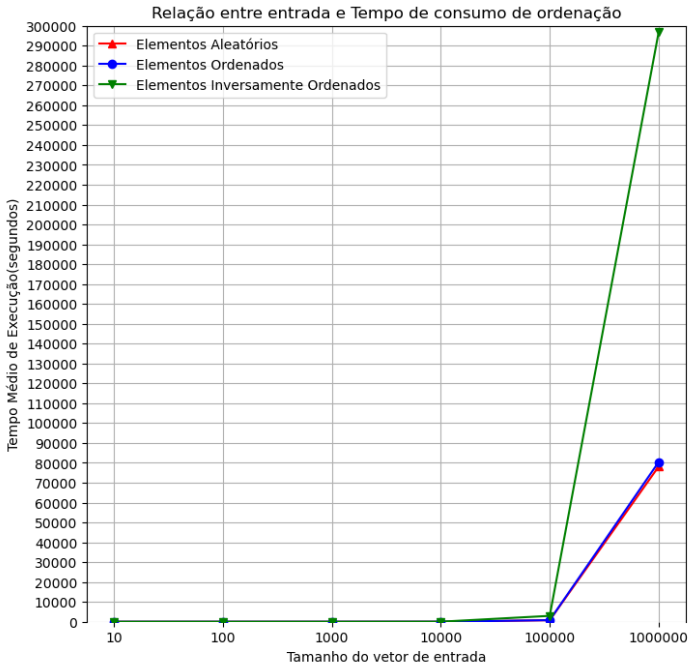

Figura 36: Plotagem do gráfico de comparação entre as diferentes entradas.

11 Shell Sort

11.1 Funcionamento do Shell Sort

A classificação Shell é uma versão generalizada do algoritmo de classificação por inserção. Ele primeiro classifica os elementos distantes entre si e reduz sucessivamente o intervalo entre os elementos a serem classificados.

O intervalo entre os elementos é reduzido com base na sequência utilizada. Algumas das sequências ideais que podem ser usadas no algoritmo de classificação de shell são:

- Sequência original do Shell: N/2 , N/4 , ..., 1
- Incrementos de Knuth: 1, 4, 13, ..., (3k – 1) / 2
- Incrementos de Sedgewick: 1, 8, 23, 77, 281, 1073, 4193, 16577, ..., 4j+1+ $3 \cdot 2$j+ 1
- Incrementos de Hibbard: 1, 3, 7, 15, 31, 63, 127, 255, 511...

- Incremento de Papernov e Stasevich: 1, 3, 5, 9, 17, 33, 65,...
- Pratt: 1, 2, 3, 4, 6, 9, 8, 12, 18, 27, 16, 24, 36, 54, 81....

É importante notar que a performance do Shell Sort depende do tipo de sequência utilizado para o vetor dado de entrada.

11.2 Gráficos plotados do Shell Sort

11.2.1 Evolução do tempo com entrada aleatória

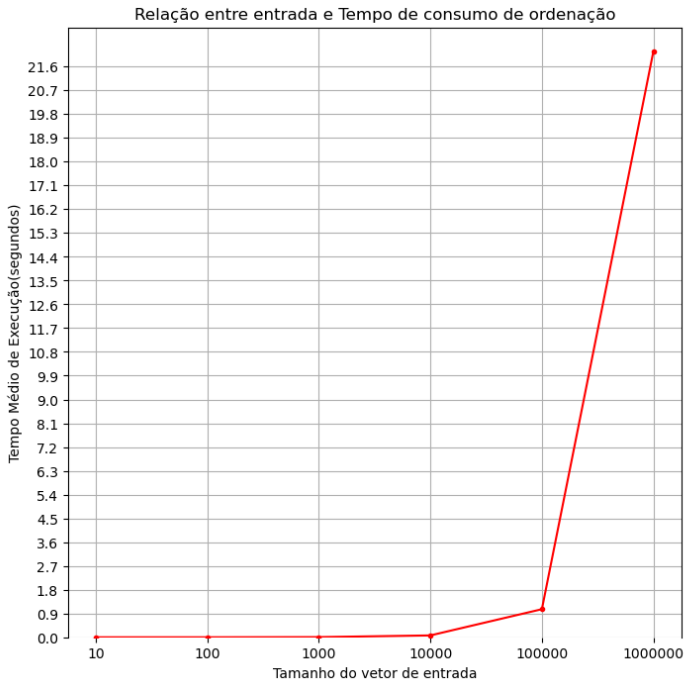

Figura 37: Plotagem do gráfico de evolução temporal com entradas aleatórias.

11.2.2 Evolução do tempo com entrada ordenada

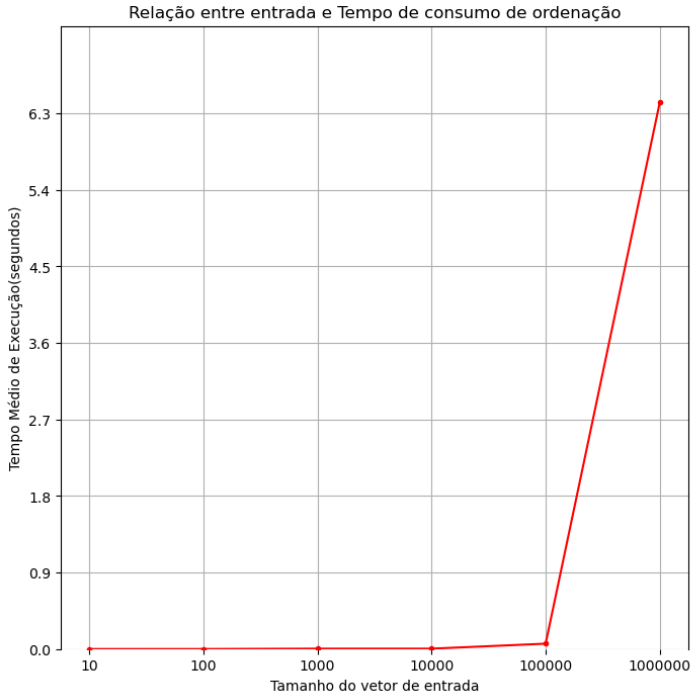

Figura 38: Plotagem do gráfico de evolução temporal com entradas ordenadas.

11.2.3 Evolução do tempo com entrada inversamente ordenada

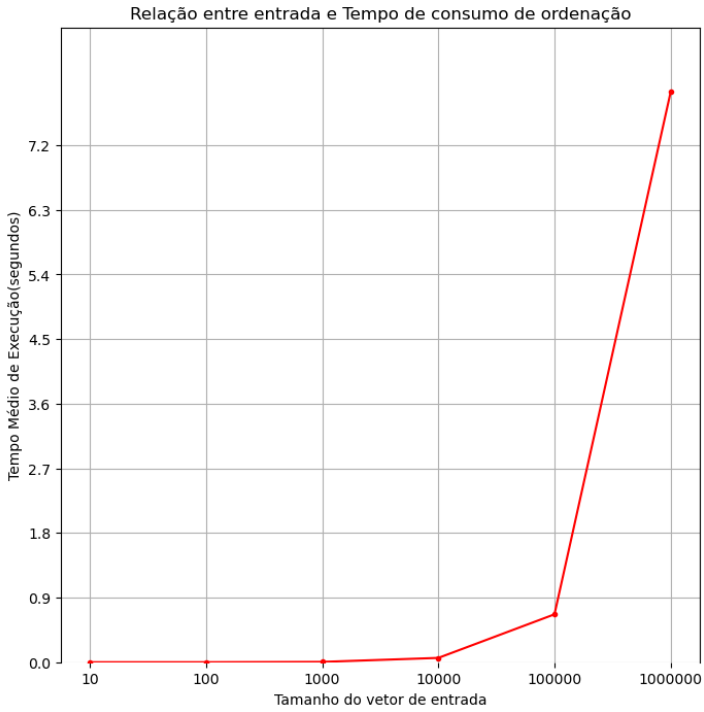

Figura 39: Plotagem do gráfico de evolução temporal com entradas inversamente ordenadas.

11.2.4 Comparação entre os três tipos de entradas

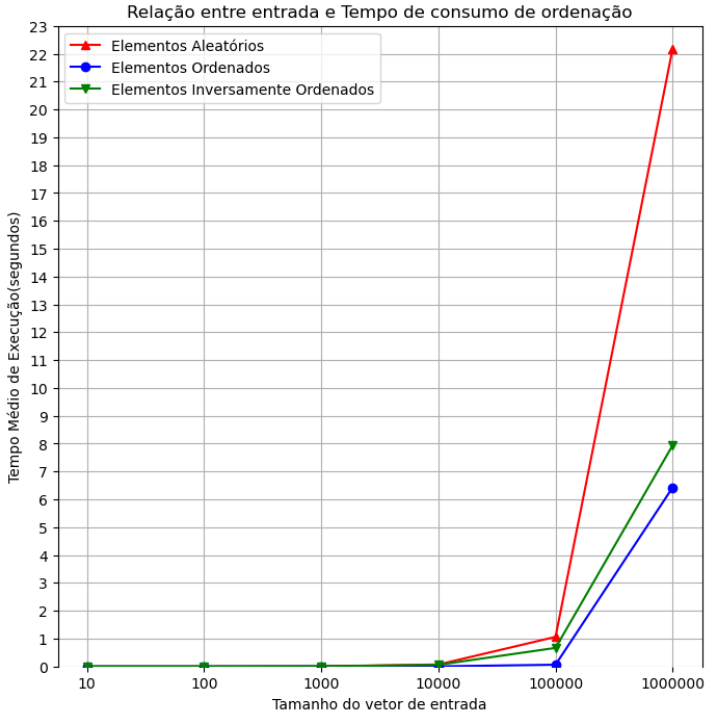

Figura 40: Plotagem do gráfico de comparação entre as diferentes entradas.

12 Comparação entre os diferentes algoritmos

12.1 Comparação entre os tempos de execução dos algoritmos quando suas entradas são vetores ordenados aleatóriamente

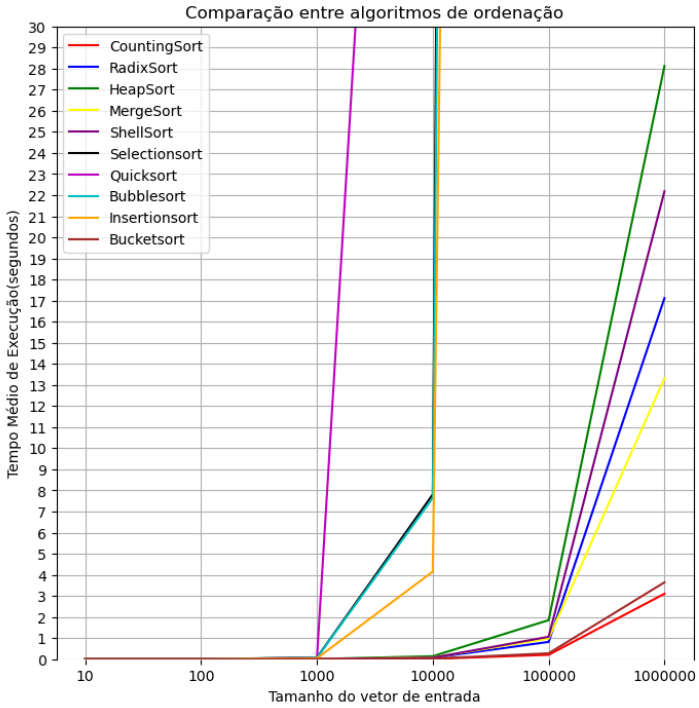

Figura 41: Plotagem do gráfico de comparação entre os diferentes algoritmos.

12.2 Comparação entre os tempos de execução dos algoritmos quando suas entradas são vetores ordenados

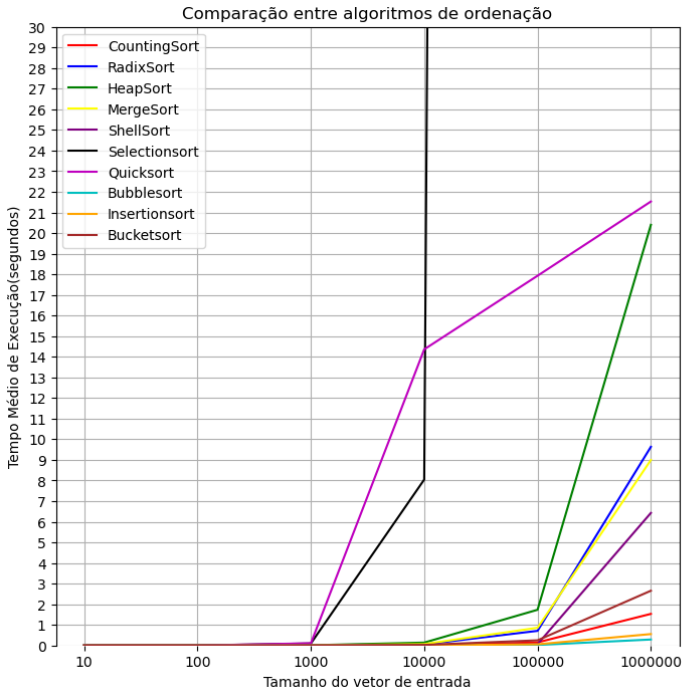

Figura 42: Plotagem do gráfico de comparação entre os diferentes algoritmos.

12.3 Comparação entre os tempos de execução dos algoritmos quando suas entradas são vetores inversamente ordenados

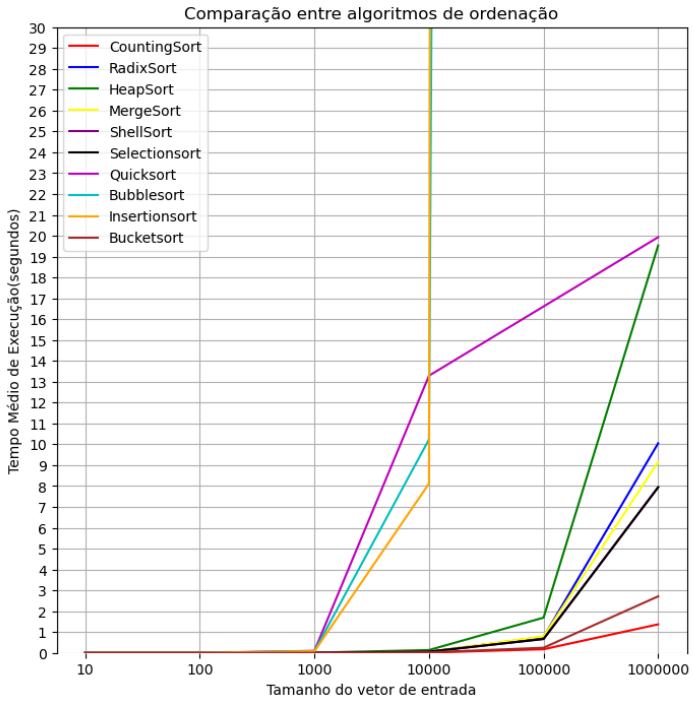

Figura 43: Plotagem do gráfico de comparação entre os diferentes algoritmos.

13 Conclusão

Verificamos então, após observar de perto a execução dos diversos algoritmos em diferentes condições, sejam elas por variação do tamanho de entrada ou pela organização de tal entrada.

É possível classificar uma lista apontando os "melhores"e os "piores"algoritmos no geral, de acordo com seus tempos de execução nas condições propostas:

1. Counting Sort;
2. Bucket Sort;
3. Merge Sort;
4. Radix Sort;
5. Shell Sort;

6. Heap Sort;
7. Quick Sort;
8. Selection Sort;
9. Insertion Sort;
10. Bubble Sort.

14 Bibliografia

1. Slides disponibilizados pelo professor.

2. Miller, Brad; Ranum, David. Apostila "Resolução de Problemas com Algoritmos e Estruturas de Dados usando Python", IME.
Disponível em: <https://panda.ime.usp.br/panda/static/pythonds_pt/>. **Acesso em: 24 de Outubro de 2024**.