RESULTADOS

O algoritmo Comb Sort criado para realizar técnicas de ordenação, foi implementado primeiramente como exemplo na linguagem Python. Com isso, utilizando a biblioteca “time” para fazer a contagem do tempo de ordenação, foi obtido uma análise da execução do algoritmo tempo bem próximos, mesmo com listas de tamanhos diferentes. No entanto, a função utilizada para fazer a contagem do tempo, não é a única da biblioteca “time” que pode ser utilizada para a contagem de tempo, existem outras formas de se fazer o mesmo.

Feito isso, como exemplo do funcionamento e do tempo de execução do algoritmo foi realizado diversos testes com entradas diferentes para analisar o tempo de resposta para a ordenação de cada lista. Como o algoritmo teve entradas diferentemente ordenadas para a execução do teste, os testes foram iniciados com elementos ordenados em ordem crescente, depois o mesmo foi realizado com uma entrada ordenada decrescente e por último, com entradas aleatórias. Dessa forma, fez-se necessário tabelar os dados encontrados para melhor ilustrar o funcionamento do programa. A Figura 1 demonstra um exemplo do teste feito com uma entrada pequena de 40 elementos em uma lista e o tempo de execução gasto na ordenação.

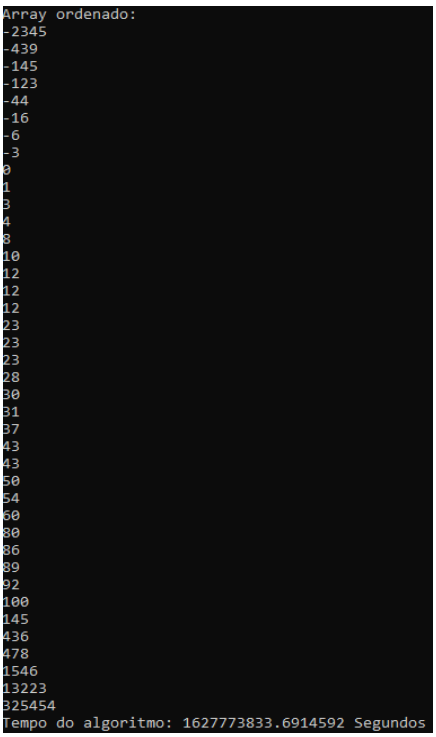

Figura 1 - Vetor de elementos aleatórios já ordenados pelo algoritmo Comb Sort e tempo gasto na ordenação.

Da mesma forma, foi coletado os dados de tempo que o algoritmo gastou na ordenação de listas de tamanhos e organizações diferentes. As tabelas 1, 2 e 3 mostram os tempos gastos na ordenação de cada entrada.

Tabela 1 - Tempo em segundos gastos na execução de algoritmos em ordem crescente

Tamanhos da Lista	**40**	**80**	**300**	**1000000**
Tempo gasto na ordenação	1627851837.8	1627851924.5	1627852091.9	1627852157.3

Fonte: do autor

Tabela 2 - Tempo em segundos gastos na execução do algoritmo em ordem decrescente

Tamanhos da Lista	**40**	**80**	**300**	**1000000**
Tempo gasto na ordenação	1627852351.0	1627852414.2	1627852432.8	1627852486.2

Fonte: do autor

Tabela 3 - Tempo em segundos gastos na execução do algoritmo com elementos aleatórios e repetidos

Tamanhos da Lista	**40**	**80**	**300**	**1000000**
Tempo gasto na ordenação	1627773833.6	1627774011.2	1627774261.2	1627774706.2

Fonte: do autor

Observando as tabelas, nota-se que os tempos de ordenação com o mesmo tipo de organização dos dados são bem próximos. Além disso, a ordenação de números aleatórios e repetidos pode ser muitas vezes mais rápida que a ordenação dos outros exemplos de entrada. Pelo falo de que, caso o número seja repetido não é necessário fazer a troca e pode - se verificar também, que com o uso do intervalo (gap) a ordenação de listas com valores aleatórios pode ser feita mais rapidamente.

Com base neste trabalho e pesquisas feitas sobre os algoritmos existentes, pode-se notar a eficácia no algoritmo Comb Sort, mesmo sendo parecido com o algoritmo

Bubble Sort, o Comb Sort consegue ordenar valores aleatórios mais rapidamente que os outros, uma amostra dos tempos é demonstrada na Tabela 3.

Como sugerido para o trabalho uma criação do algoritmo Comb Sort, não foi apresentado testes e amostras sobre os outros algoritmos. Outro problema foi obter os dados sobre a quantidade de memória gasta por cada algoritmo na ordenação das listas, pois foi realizado testes com a biblioteca “psutil” da linguagem python para isso, mas não foi obtido sucesso nesse âmbito.