1. **Introdução**

Este trabalho consiste na aplicação de alguns algoritmos de ordenação comumente utilizados, sendo eles: Bubble Sort, Insertion Sort, Merge Sort, QuickSort, HeapSort; em dois tipos de listas (ordenadas e aleatórias) com tamanho distintos, a fim de se avaliar a complexidade temporal dos mesmos.

Para tanto, as seções foram divididas em: Funções Auxiliares, a qual contém algumas funções utilizadas para facilitar a criação e manipulação das listas; Implementações dos Algoritmos, em que. como o próprio nome diz, traz a implementação na linguagem Python 3 de cada um dos algoritmos de ordenação; Experimentações, cujo intuito está em trazer o tempo médio de execução dos algoritmos tendo em vista os dois tipos de listas com tamanhos distintos; Resultados e Conclusões, na qual traz os resultados obtidos e as considerações finais acerca do tema.

Dessa forma, utilizou-se de dois código-fonte separados: um denominado de “T1\_Ordenacao.py” e outro de “T1\_Ordenacao\_Testes.py”. O primeiro contém uma classe principal denominada “Ordenacao”, a qual estão contidas as implementações dos algoritmos de ordenação; enquanto que o segundo contém as funções auxiliares e a realização dos testes em si. A escolha para trabalhar-se com uma classe separada se deu por conta do conceito de reuso de código e facilidade de implementação posterior, bastando apenas realizar a chamada instância da classe.

Por fim, necessitou-se de três bibliotecas para a melhor implementação dos algoritmos de ordenação e das medições de tempo.

Especificamente, abaixo tem-se a importação da biblioteca “heapq”, a qual possui funções primárias de construção e manipulação de uma árvore do tipo Heap (sendo utilizado no HeapSort), e está contida no arquivo “T1\_Ordenacao.py” junto da classe principal “Ordenacao”.

Já as bibliotecas “random” e “time” são utilizadas no arquivo de testes, sendo fundamentais para a geração de listas e demarcação do tempo de execução de cada algoritmo de ordenação, respectivamente.

1. **Funções Auxiliares**

Para facilitar na criação e manipulação de listas, bem como na medição e exibição dos tempos de processamento dos algoritmos, foram desenvolvidas três funções auxiliares no código-fonte “T1\_Ordenacao\_Testes.py”.

A primeira, denominada de “geraLista”, é responsável por gerar e retornar uma lista de tamanho previamente especificado e que contenha elementos de 0 ao número equivalente a esse tamanho não incluso (por exemplo: considere um tamanho = 10, a lista a ser gerada conterá os elementos de 0 a 9, ordenados ou não). Além disso, possui um parâmetro denominado “ordenado”, o qual representa se a lista deve ser gerada de maneira ordenada (ordenado=True) ou de maneira aleatória (ordenado=False).

A segunda, denominada de “printaLista”, como o próprio nome sugere, é responsável por exibir os valores de uma lista (sendo restrito aos cinco primeiros e cinco últimos) e o tempo de processamento do respectivo algoritmo de ordenação aplicado.

A terceira e última, denominada de “executaOrdenacao”, é responsável por gerenciar as duas funções auxiliares anteriores bem como aplicar um algoritmo de ordenação (especificado por parâmetro) e calcular o seu respectivo tempo de processamento (em milissegundos), exibindo as listas antes e após a ordenação.

1. **Implementação dos Caralhos**

Essa seção se restringe a exibição separada dos algoritmos de ordenação analisados neste trabalho e na respectiva classe “Ordenacao”. Para tanto, utilizou-se as melhores implementações de cada um para a linguagem Python 3.

1. **Experimentações**

Os experimentos foram divididos da seguinte forma: primeiramente, é definido um tamanho para a geração de listas, sendo utilizado tamanhos variando de 10, 100, 1000 e 10000 para a análise neste trabalho. Após isso, é definido o tipo de lista a ser gerada: aleatória ou ordenada, sendo o primeiro utilizado a função “shuffle” da biblioteca “random” para embaralhar os elementos. Por fim, é aplicado um algoritmo de ordenação e exibido o respectivo tempo de processamento (em milissegundos). Desta forma, pode-se mensurar, em termos de complexidade temporal, as principais diferenças entre os algoritmos de ordenação utilizados neste trabalho.

1. **Conclusão**

Texto