Comparação de algoritmos de ordenação

Objetivo:

O objetivo deste projeto é comparar o desempenho de diferentes algoritmos de ordenação em termos de tempo e espaço de execução, utilizando uma variedade de conjuntos de dados.

Público-alvo:

Este projeto é direcionado a estudantes de Análise de Complexidade de Algoritmos que desejam aprofundar seus conhecimentos sobre o desempenho de algoritmos de ordenação e suas aplicações práticas.

Materiais necessários:

- Linguagem de programação (ex: Python, Java, C++)
- Compilador ou interpretador
- Ferramentas de medição de tempo (bibliotecas de medida de tempo)
- Conjuntos de dados de diferentes tamanhos e características:
    - Aleatórios
    - Ordenados
    - Inversamente ordenados

Etapas do projeto:

1. Seleção de algoritmos:

- Algoritmos de ordenação a serem comparados, considerando diferentes abordagens e complexidades:
    - $O(n^2)$:
        - Bubble Sort,
        - Insertion Sort,
        - Selection Sort,
    - O(n log n):
        - Merge Sort,
        - Quick Sort,
        - Heap Sort.
    - O(n+k) ou O(nk):
        - Counting Sort

        - Radix sort

2. Implementação dos algoritmos:

- Implementar cada algoritmo de ordenação na linguagem de programação escolhida, garantindo que as implementações sejam eficientes e adequadas para a comparação.

3. Preparação dos conjuntos de dados:

- Criar ou coletar conjuntos de dados de diferentes tamanhos e características, como:
    - **Ordenados:** para analisar o desempenho em casos "ideais".
    - **Inversamente ordenados:** para analisar o pior caso dos algoritmos.
    - **Aleatórios:** para analisar o desempenho em casos mais gerais.
- Para garantir que todos os algoritmos sejam submetidos ao mesmo conjunto de testes, os dados devem ser gravados em arquivos e lidos em cada execução de teste.
- Considerar os seguintes N, para cada caso teste (Ordenados, Inversamente ordenados e Aleatórios):
    - 10000
    - 20000
    - 40000
    - 80000
    - 100000

4. Realização dos testes:

- Executar cada algoritmo de ordenação em cada conjunto de dados, medindo o tempo de execução (segundos).
- Medir a quantidade de comparações entre elementos do conjunto de dados executadas durante a ordenação.
- Medir a quantidade de trocas entre elementos do conjunto de dados executadas durante a ordenação
- Variar o tamanho dos conjuntos de dados para observar como o tempo de execução e o espaço de memória mudam com o aumento da quantidade de elementos.
    - Dobre o tamanho do conjunto de dados a cada variação de tamanho.
- Repetir os testes várias vezes para garantir a confiabilidade dos resultados.

5. Análise dos resultados:

- Organizar os resultados em tabelas e gráficos para facilitar a visualização e comparação.
- Analisar os resultados para identificar os algoritmos com melhor desempenho em termos de tempo e espaço de execução para cada tipo de conjunto de dados.
- Discutir os resultados obtidos, considerando a complexidade computacional de cada algoritmo e suas características.

6. Conclusões:

- Apresentar as conclusões do projeto, destacando os algoritmos mais eficientes para diferentes tipos de conjuntos de dados.
- Discutir as implicações práticas dos resultados obtidos, considerando aplicações reais de ordenação.
- Sugerir futuras pesquisas e aprimoramentos para o projeto.