GOVERNO DO ESTADO DO PIAUÍ - UNIVERSIDADE ESTADUAL DO PIAUÍ
CAMPUS PROFESSOR ANTÔNIO GIOVANNI ALVES DE SOUSA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO
DISCIPLINA: Estrutura de Dados
ALUNOS: Samuel Sousa Lustosa e Eduardo Alves de Sousa

Link do GitHub para acesso do código: https://github.com/Eduardo-Alves-de-Sousa/Avalia-o_ED

1. Introdução:

O problema abordado neste trabalho é a comparação de desempenho de algoritmos de ordenação, tendo como foco principal na implementação e análise do algoritmo **MergeSort**. O objetivo é analisar como o algoritmo se comporta em diferentes cenários, considerando tamanhos e tipos variados de vetores de entrada. O programa desenvolvido gera vetores do tipo **aleatórios, ordenados e inversamente ordenados**, aplicando o **MergeSort** e registrando métricas como **tempo de execução, número de comparações e movimentações.**

2. Implementação:

A estrutura do programa é baseada em **linguagem C** e consiste em uma implementação do **algoritmo MergeSort** adaptado para contar o número de comparações e movimentações. A estrutura de dados principal é um vetor de inteiros. Abaixo estão os principais detalhes da implementação:

- **merge_sort:** Função principal que implementa o algoritmo MergeSort.
- **merge:** Função auxiliar para mesclar dois sub-vetores ordenados.
- **testa_merge_short:** Função para testar o mergesort com diferentes tipos e tamanhos de arrays.
- **main:** Função principal que gera e ordena vetores de diferentes tamanhos e tipos, calculando métricas de desempenho.

O formato de entrada é especificado pelo usuário escolhendo o tamanho do vetor (100, 1000, 10000 e 100000) e qual tipo de vetores (aleatórios, ordenados e inversamente ordenados). A saída inclui informações sobre o tamanho do vetor, o tipo aleatório (Tipo: r), ordenado (Tipo: s) ou inversamente ordenado (Tipo: d), o tempo de execução, o número de comparações e o número de movimentos.

3. Listagem de testes executados:

Os testes foram **executados com vetores de tamanhos 100, 1000, 10000 e 100000**, utilizando **três tipos de vetores: aleatórios, ordenados e inversamente ordenados.** Cada teste foi repetido pelo menos 10 vezes para calcular médias de

tempo de execução, comparações e movimentações. Logo abaixo se encontra a tabela onde para valores **aleatórios** foram calculados o seu tempo de execução para

Todas as entradas propostas e logo depois as tabelas de **Vetores Ordenado, e Inversamente Ordenados** com resultados de 3 testes em cada tamanho de execução. Vale ressaltar que em todas as entradas foram feitas requeridos 10 testes de tamanho e tipo.

Tabela de Desempenho para Vetores Aleatórios, EXEMPLO (Tamanho: 100):

Tamanho do vetor	Tipo	Tempo de Execução	Número de Comparações	Número de Movimentos
100	Aleatório	0.000013s	541	672
100	Aleatório	0.000014s	541	672
100	Aleatório	0.000014s	542	672
100	Aleatório	0.000015s	541	672
100	Aleatório	0.000012s	540	672
100	Aleatório	0.000017s	538	672
100	Aleatório	0.000009s	540	672
100	Aleatório	0.000017s	540	672
100	Aleatório	0.000010s	543	672
100	Aleatório	0.000013s	544	672
Média T. Execução	0.00022s			
Número médio de comparações	540			

Tabela de Desempenho para Vetores Aleatórios, EXEMPLO (Tamanho: 1000):

Tamanho do vetor	Tipo	Tempo de Execução	Número de Comparações	Número de Movimentos
1000	Aleatório	0.000143s	8688	9976
1000	Aleatório	0.000156s	8717	9976
1000	Aleatório	0.000201s	8686	9976

1000	Aleatório	0.000157s	8665	9976
1000	Aleatório	0.000174s	8701	9976
1000	Aleatório	0.000254s	8703	9976
1000	Aleatório	0.000193s	8701	9976
1000	Aleatório	0.000194s	8726	9976
1000	Aleatório	0.000184s	8697	9976
1000	Aleatório	0.000183s	8693	9976
Média T. Execução	0.000184s			
Número médio de comparações	8700			

Tabela de Desempenho para Vetores Aleatórios, EXEMPLO (Tamanho: 10000):

Tamanho do vetor	Tipo	Tempo de Execução	Número de Comparações	Número de Movimentos
10000	Aleatório	0.002012s	120422	133616
10000	Aleatório	0.001734s	120410	133616
10000	Aleatório	0.001447s	120483	133616
10000	Aleatório	0.001466s	120500	133616
10000	Aleatório	0.003764s	120443	133616
10000	Aleatório	0.004101s	120481	133616
10000	Aleatório	0.001701s	120363	133616
10000	Aleatório	0.001642s	120416	133616
10000	Aleatório	0.001576s	120454	133616
10000	Aleatório	0.002998s	120498	133616
Média T. Execução	0.002237s			
Número médio de comparações	120447			

Tabela de Desempenho para Vetores Aleatórios, EXEMPLO (Tamanho: 100000):

Tamanho do vetor	Tipo	Tempo de Execução	Número de Comparações	Número de Movimentos
100000	Aleatório	0.023114s	1536024	1668928
100000	Aleatório	0.024852s	1536475	1668928
100000	Aleatório	0.022440s	1536233	1668928
100000	Aleatório	0.024027s	1536215	1668928
100000	Aleatório	0.024450s	1536101	1668928
100000	Aleatório	0.023582s	1535812	1668928
100000	Aleatório	0.022037s	1536215	1668928
100000	Aleatório	0.023420s	1535799	1668928
100000	Aleatório	0.022695s	1535985	1668928
10000	Aleatório	0.024103s	1536095	1668928
Média T. Execução	0.023472s			
Número médio de comparações	1536095			

Tabela de Desempenho para Vetores Ordenado, EXEMPLO (Tamanho: todos):

Tamanho do vetor	Tipo	Tempo de Execução	Número de Comparações	Número de Movimentos
100	Ordenado	0.000006s	316	672
100	Ordenado	0.000011s	316	672
100	Ordenado	0.000009s	316	672
1000	Ordenado	0.000201s	8690	9976
1000	Ordenado	0.000133s	8683	9976
1000	Ordenado	0.000182s	8725	9976
10000	Ordenado	0.001868s	120458	133616

10000	Ordenado	0.001436s	120418	133616
10000	Ordenado	0.002020s	120310	133616
100000	Ordenado	0.024809s	1536143	1668928
100000	Ordenado	0.024920	1536397	1668928
100000	Ordenado	0.025611	1536175	1668928

Tabela de Desempenho para Vetores Inversamente Ordenado, EXEMPLO (Tamanho: todos):

Tamanho do vetor	Tipo	Tempo de Execução	Número de Comparações	Número de Movimentos
100	Inversamente Ordenado	0.000013s	547	672
100	Inversamente Ordenado	0.000034s	534	672
100	Inversamente Ordenado	0.000031s	542	672
1000	Inversamente Ordenado	0.000206s	8676	9976
1000	Inversamente Ordenado	0.000132s	8708	9976
1000	Inversamente Ordenado	0.000123s	8694	9976
10000	Inversamente Ordenado	0.003083s	120394	133616
10000	Inversamente Ordenado	0.001567s	120514	133616
10000	Inversamente Ordenado	0.003171s	120416	133616
100000	Inversamente Ordenado	0.019137s	1536004	1668928
100000	Inversamente Ordenado	0.018175s	1536205	1668928
100000	Inversamente Ordenado	0.018586s	153672	1668928

4. Conclusão:

O trabalho permitiu uma compreensão aprofundada do algoritmo MergeSort e sua **eficiência em diferentes cenários**. As principais dificuldades encontradas foram relacionadas à criação e adaptação do código para contar comparações e movimentações de maneira precisa e de maneira clara e objetiva para assim conseguir cumprir com todos os pontos propostos no trabalho em questão.

5. Referencias:

- Merge Sort - data structure and algorithms tutorials. (2013, March 15). GeeksforGeeks. https://www.geeksforgeeks.org/merge-sort/
- C program for merge sort. (2013, March 15). GeeksforGeeks. https://www.geeksforgeeks.org/c-program-for-merge-sort/
- Merge sort (with code in Python/C++/Java/C). (n.d.). Programiz.com. Retrieved November 17, 2023, from https://www.programiz.com/dsa/merge-sort
- Jha, M. (2022, October 31). C program for merge sort. Scaler Topics. https://www.scaler.com/topics/merge-sort-in-c/