Universidade do Estado do Rio de Janeiro
Engenharia de Computação
Projeto e Análise de Algoritmos

Relatório

Algoritmos de ordenação Selection-Sort e Merge-Sort

Aluno:
Victor Pinheiro Feitosa

Professora:
Camila Martins Saporetti

Junho
2023

Conteúdo

1 Introdução

Um algoritmo de ordenação é um conjunto de passos ou instruções lógicas que são projetados para organizar os elementos de uma lista em uma ordem específica, geralmente em ordem crescente ou decrescente. O objetivo principal de um algoritmo de ordenação é rearranjar os elementos de forma que possam ser facilmente pesquisados, recuperados ou apresentados de maneira ordenada.
Os algoritmos de ordenação podem ser aplicados a vários tipos de dados, como números, strings, registros de banco de dados, entre outros. Eles são amplamente utilizados em ciência da computação e em muitos campos da tecnologia, uma vez que a ordenação é uma tarefa comum e fundamental em muitos problemas e aplicativos.
Neste trabalho, foi proposto a implementação dos algoritmos de ordenação de vetores Selection-Sort e Merge-Sort, com o intuito de comparação entre os mesmos, a fim de determinar qual algoritmo é mais eficiente em qual ocasião e também qual algoritmo possui maior complexidade.

2 Informações

2.1 Linguagem de Programação

A linguagem de programação utilizada foi Python e a IDE utilizada para os testes foi a Pycharm.

2.2 Computador

A configuração do computador utilizado para os testes é: Windows 11 Pro 64 bits,
Processador AMD Ryzen 5 3600, 16 GB de Memória RAM e Placa de Vídeo Geforce RTX 3060.

3 Selection-Sort

O Selection Sort é um algoritmo de ordenação que busca iterativamente o menor elemento não ordenado em uma lista e o coloca na posição correta. Ele percorre a lista várias vezes, dividindo-a em duas partes: a parte ordenada, localizada no início da lista, e a parte não ordenada, localizada no final.

O processo do Selection Sort pode ser descrito da seguinte forma:

Encontrar o menor elemento não ordenado: O algoritmo começa comparando o primeiro elemento da parte não ordenada com os elementos subsequentes. Ele busca o menor elemento e registra sua posição.

Troca: Uma vez encontrado o menor elemento não ordenado, ele é trocado com o primeiro elemento da parte não ordenada. Isso garante que o menor elemento esteja na posição correta e pertença à parte ordenada.

Expandir a parte ordenada: A cada iteração, a parte ordenada é expandida para incluir o próximo elemento mínimo da parte não ordenada. Esse elemento é encontrado através de comparações e trocas semelhantes às realizadas na primeira etapa.

Repetição: Os passos 1 a 3 são repetidos até que a parte não ordenada seja completamente percorrida. Isso garante que todos os elementos estejam na posição correta, resultando em uma lista ordenada.

O Selection Sort possui uma complexidade de tempo de $O(n^2)$, onde n é o tamanho da lista a ser ordenada. Isso significa que o tempo de execução aumenta quadraticamente com o tamanho da lista. Portanto, o Selection Sort é mais eficiente para listas pequenas ou quando a eficiência não é uma prioridade.

Uma característica importante do Selection Sort é que ele não é estável, ou seja, a ordem relativa dos elementos com chaves iguais pode ser alterada durante o processo de ordenação.

Apesar de não ser o algoritmo mais eficiente, o Selection Sort tem a vantagem de ser simples de implementar e requerer um número fixo de trocas. Também é útil quando há restrições de espaço, pois não requer espaço adicional além do necessário para armazenar a lista original.

3.1 Resultado e Análise

Ordem Inicial	Tamanho dos vetores	Tempo de execução (segundos)
Decrescente	50	0
Crescente	50	0
Aleatorio	50	0
Decrescente	500	0,00400018692016601
Crescente	500	0,00400042533874511
Aleatorio	500	0,0040009021759033 20
Decrescente	5000	0,402108907699584000
Crescente	5000	0,392980813980102000
Aleatorio	5000	0,383604049682617000
Decrescente	50000	41,01695823669430000
Crescente	50000	38,97694492340080000
Aleatorio	50000	40,40640139579770000

Figura 1: Tabela do tempo de execução utilizando o Selection-Sort

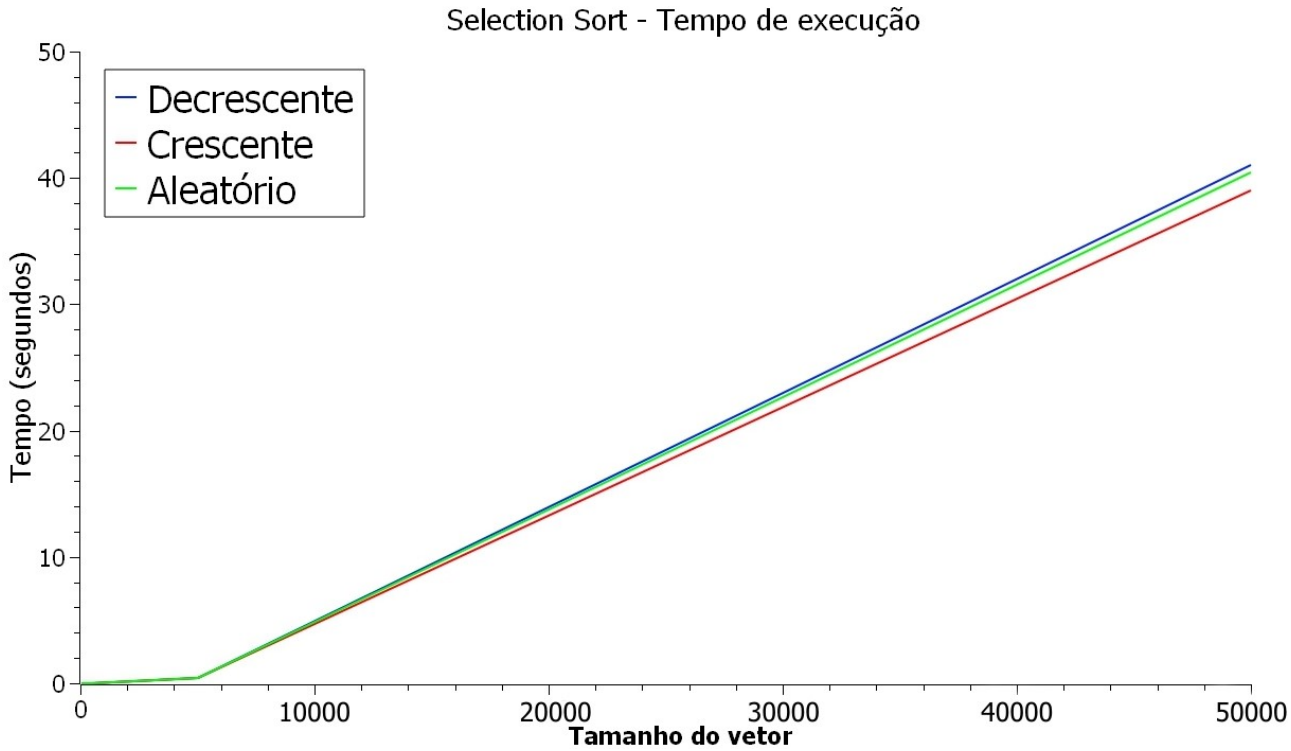

Figura 2: Gráfico Selection-Sort

A figura 1 retrata a ordem, o tamanho dos vetores e o tempo de execução em segundos que o algoritmo Selection-Sort levou para executar a ordenação.

Já a figura 2 apresenta o gráfico do tempo de execução onde x é o tamanho do vetor e y é o tempo de execução do algoritmo de ordenação Selection-Sort em segundos, dado o tamanho do vetor. No gráfico, podemos observar que o algortimo leva mais tempo para realizar a ordenação dos vetores de números decrescentes, um tempo mediano para os vetores de números aleatórios e um tempo consideralmente menor que os outros dois para vetores de números crescentes.
Analisando os vetores de tamanho 50, podemos observar que foram ordenados de maneira imediata, levando 0 segundos.
Nos vetores de tamanho 500, observamos um tempo de execução parecido entre os vetores de ordem decrescente, crescente e aleatória.
Nos vetores de tamanho 5000, temos um tempo de execução maior para o vetor de ordem decrescente, quando comporado com o tempo de execução dos vetores de ordem crescente e aleatória.
Já nos vetores de tamanho 50000, podemos analisar um cenário similar, visto que o vetor de ordem decrescente demora ligeiramente mais tempo para ser executado do que os de ordem crescente e aleatória.

4 Merge-Sort

O Merge Sort é um algoritmo de ordenação que divide repetidamente uma lista em partes menores até que cada parte contenha apenas um elemento. Em seguida, ele combina essas partes de forma ordenada, repetindo o processo até que todos os elementos estejam unidos em uma única lista ordenada, a ideia principal por trás do Merge Sort é que é mais fácil e rápido ordenar pequenas partes de uma lista do que a lista inteira de uma só vez. Ao dividir a lista em partes menores, o Merge Sort simplifica o problema da ordenação e o torna mais gerenciável.

O algoritmo funciona da seguinte maneira:
Primeiro, ele divide a lista pela metade até que cada parte contenha apenas um elemento. Isso é feito recursivamente até que a lista esteja completamente dividida.
Depois, combina as partes ordenadamente. Ele compara os elementos das partes e os mescla em ordem crescente, criando uma nova lista ordenada. Esse processo é repetido até que todas as partes tenham sido mescladas e uma única lista ordenada seja obtida.

O Merge Sort é eficiente porque divide a lista em partes menores logaritmicamente, o que resulta em um tempo de execução rápido e previsível.
A complexidade de tempo do Merge Sort é O(n log n), o que significa que seu desempenho é proporcional ao tamanho da lista a ser ordenada multiplicado pelo logaritmo desse tamanho, uma vantagem importante do Merge Sort é que ele é um algoritmo estável, o que significa que ele preserva a ordem relativa de elementos com chaves iguais.
Além disso, o Merge Sort pode ser implementado para ordenar listas encadeadas e não apenas vetores. No entanto, o Merge Sort requer espaço adicional para armazenar as partes e a lista temporária durante o processo de mesclagem. Isso pode ser uma desvantagem em situações em que a memória é limitada.

4.1 Resultado e Análise

A figura 3 abaixo retrata a ordem, o tamanho dos vetores e o tempo de execução em segundos que o algoritmo Merge-Sort levou para executar a ordenação.
Já a figura 4, também abaixo, apresenta o gráfico do tempo de execução onde x é o tamanho do vetor e y é o tempo de execução do algoritmo de ordenação Merge-Sort em segundos, dado o tamanho do vetor. No gráfico, podemos observar que o algortimo leva mais tempo para realizar a ordenação dos vetores de números aleatórios, enquanto leva um tempo menor e similar para os vetores de números decrescentes e crescentes.
Analisando os vetores de tamanho 50, podemos observar que, assim como no Selection-Sort, foram ordenados de maneira imediata, levando 0 segundos.

Nos vetores de tamanho 500, observamos um tempo de execução maior para os vetores de ordem crescente, enquanto os de ordem decrescente e aleatória obtiveram resultados similares. Nos vetores de tamanho 5000, temos um tempo de execução maior para os vetores de ordem aleatória, enquanto o de ordem decrescente leva um tempo menor, tendo o de ordem crescente sendo executado de maneira mais rápida entre as três possíveis ordens, sendo o mais otimizado. Já nos vetores de tamanho 50000, podemos analisar que os vetores de ordem aleatória levaram muito mais tempo para ser executado em comparação com os vetores de ordem crescente e decrescente, que foram executados de maneira extremamente otimizada.

Ordem Inicial	Tamanho dos vetores	Tempo de execução (segundos)
Decrescente	50	0
Crescente	50	0
Aleatorio	50	0
Decrescente	500	0,00099992752075195
Crescente	500	0,00100016593933105
Aleatorio	500	0,000999212265014648
Decrescente	5000	0,012287855148315400
Crescente	5000	0,010005235671997000
Aleatorio	5000	0,015003442764282200
Decrescente	50000	0,08752465248107910
Crescente	50000	0,08652901649475090
Aleatorio	50000	0,13153457641601500

Figura 3: Tabela do tempo de execução utilizando o Merge-Sort

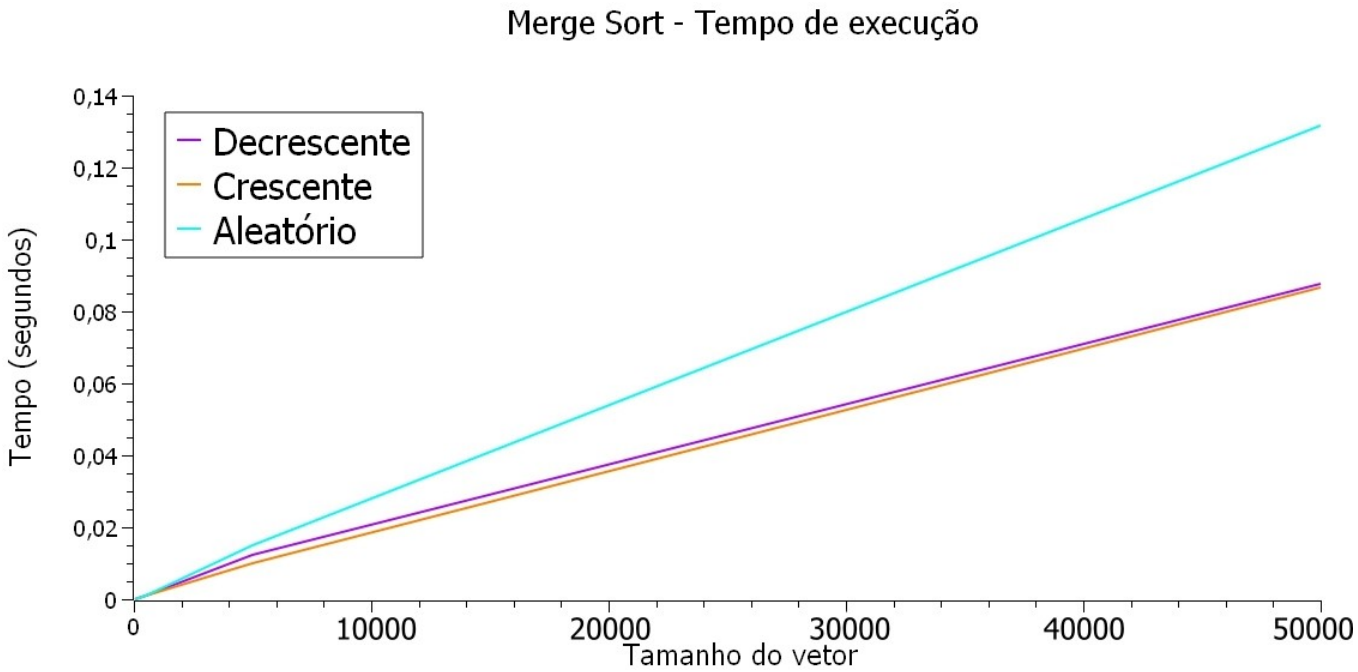

Figura 4: Gráfico Merge-Sort

5 Comparação e Conclusão

Comparando ambos os resultados, temos que, para os testes realizados, o algoritmo de ordenação Merge-Sort mostrou ser mais eficiente do que o Selection-Sort em todos os cenários. Neste cenário, o algoritmo Merge-Sort provou ser melhor que o Selection-Sort a medida que o tamanho do vetor aumenta.
Concluindo, notei em meus testes uma grande diferença de tempo quando se tratava principalmente de vetores de tamanho 50000, visto que o algortimo de ordenação Merge-Sort ordenou o vetor de maneira rápida e o Selection-Sort levou um tempo considerável para finalizar a ordenação, deixando claro que mesmo com o Selection-Sort possuindo maior complexidade ($O(n^2)$) do que o Merge-Sort ($O(nlogn)$), o algoritmo de ordenação Merge-Sort se mostrou superior.

6 Referências

https://www.javatpoint.com/sorting-algorithms
https://www.programiz.com/dsa/selection-sort
https://www.programiz.com/dsa/merge-sort