**ALGORITMOS DE ORDENAÇÃO BUBBLE, SELECTION E INSERTION SORT : UM ESTUDO DE CASO SOBRE O TEMPO DE EXECUÇÃO**

Pedro Henrique Resende Alves, Vitor de Azambuja Ribeiro Franco

**INTRODUÇÃO**

A ordenação de dados é um conjunto de métodos fundamentais presentes em diversas áreas da computação. Ela implica em facilitar algoritmos de buscas e pesquisas de ocorrências ordenadas e até a organização dos bancos de dados. As bases para a compreensão dos princípios de ordenação são utilizadas dentro de um módulo simples, e que constroi as pontes para outros algoritmos mais modernos. Esses métodos são o Bubble Sort, Selection Sort e Insertion Sort.

A ordenação é um dos requisitos mais comuns em aplicações. Um simples relatório, produzido por um programa, terá uma ordenação levando em conta algum critério. Para que os dados sejam ordenados, podem-se adotar duas abordagens: ao inserir um determinado elemento na lista, respeitar a ordenação da estrutura, ou aplicar algum algoritmo de ordenação a um conjunto de dados já criado. (Livro Estrutura de Dados 1ª Edição 2015, Ricardo Balieiro, p. 58)

Para realizar a análise comparativa entre os métodos de ordenação Bubble Sort, Selection Sort e Insertion Sort, foi utilizada a linguagem de programação Python devido à sua simplicidade e facilidade de implementação. Foram utilizados conjuntos de dados de três tamanhos diferentes: 100, 1000 e 10000 elementos. Cada conjunto de dados foi organizado em três ordens distintas: ordenado em ordem crescente (Ordem 1), ordenado em ordem decrescente (Ordem 2) e desordenado com números aleatórios (Ordem 3).

A linguagem de programação Python possui uma funcionalidade mais simples e acessível facilitando a implementação de dados. Para a análise comparativa entre os métodos de ordenação, três testes de ordens de listas com três tamanhos diferentes foram utilizados: 100, 1000 e 10000 como os elementos requeridos. Nesse teste prático, os conjuntos de dados são organizados em três ordens, sendo a Ordem 1 uma lista ordenada em ordem crescente, a Ordem 2 ordenada em ordem decrescente, e a Ordem 3 como uma lista desordenada com números aleatórios. Os resultados são inseridos em tabelas onde se destacam os tempos, as comparações e as movimentações de cada algoritmo.

**DESENVOLVIMENTO**

**METODOLOGIA**

A metodologia aplicada consiste em utilizar os três algoritmos de ordenação (Bubble Sort, Selection Sort e Insertion Sort) para executar os conjuntos de dados e então obter os resultados das três ordens específicas exigidas para a avaliação dos registros de tempo de execução, número de comparações e o número de movimentações.

As exigências implicam em calcular e executar conjuntos de dados através da escolha de uma linguagem de programação para a escolha dos praticantes. São exigidos a utilização dos três algoritmos de ordenação interna para que o estudo prático seja realizado.

Os testes são executados utilizando três ordens para os conjuntos de dados fornecidos que determinam os diferentes tamanhos das ordens de listas sob os valores de 100, 1000 e 10000. O principal objetivo desses testes é verificar o comportamento dos algoritmos em relação ao tempo, movimentações de trocas e comparações.

O trabalho prático foi desenvolvido com o uso da linguagem de programação Python através do compilador Visual Studio Code. Gerando os códigos de ordenação, uma tabela deve ser designada para inserir os resultados de cada ordem realizada distinguida pelos três métodos referenciados. Em última etapa, cada resultado é assemelhado pelos atributos alinhados totalizando nove tabelas para as comparações finais e as curvas de desempenho dos métodos são traçadas em gráficos pelo tempo de execução.

Essa metodologia permite elaborar a possibilidade de avaliar como os algoritmos se comportam em circunstâncias de pior e melhor caso, analisando também algoritmos de caso médio, tudo dentro da variação nos tamanhos dos conjuntos de dados e nas ordens de entrada. Os métodos de ordenação utilizados são importantes para os princípios básicos de ordenação e respectivos a simplicidade que todos desempenham quando postos em prática, tendo seus desempenhos comparados em relação aos resultados obtidos. A eficiência desses métodos foi proporcionada por insights dos resultados e podem auxiliar na indicação do método mais adequado para cada situação.

**REFERENCIAL TEÓRICO**

O Bubble Sort, ou ordenação por bolha, é um algoritmo de ordenação simples. O método percorre um conjunto de elementos diversas vezes, e cada passagem flutua para o topo do maior elemento da sequência. A movimentação se assemelha à forma como as bolhas em um tanque de água procuram seu próprio nível. O algoritmo executa *n* operações relevantes representando o número de elementos no vetor. Esse algoritmo é de ordem quadrática, não sendo recomendado para programas que precisam de velocidade com quantidade elevada de dados.

O Selection Sort, ou ordenação por seleção, é um algoritmo de ordenação baseado em passar o menor valor do vetor para a primeira posição, ou dependendo da ordem requisitada, pode passar também o maior valor. Em sequência, o segundo menor valor passa para a segunda posição e assim segue sucessivamente com os elementos restantes através de *n - 1*até os dois últimos elementos. Esse método compara cada interação de um elemento com os outros, visando sempre encontrar o menor valor.

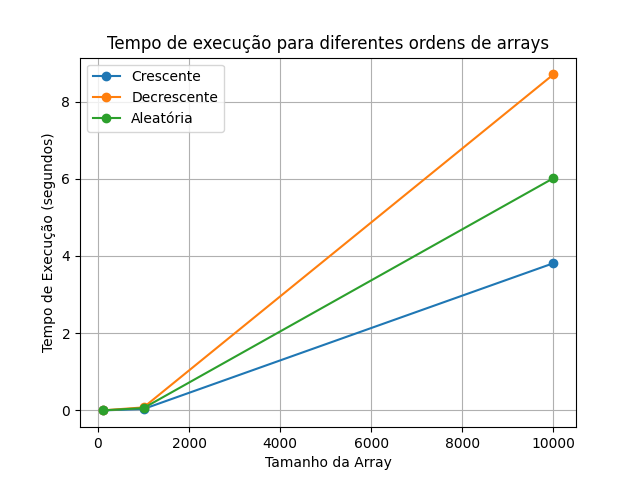
O Insertion Sort, ou ordenação por inserção, é um algoritmo de ordenação que através de uma estrutura de dados, desenvolve uma matriz final com um elemento de cada vez. É o algoritmo mais eficiente entre os três métodos de ordenação apresentados. Esse algoritmo pode ser comparado do mesmo modo que as pessoas organizam um baralho em um jogo de cartas. As cartas são posicionadas e ordenadas, e as cartas recebidas devem ser colocadas na posição correta na mão de cartas, de uma forma que haja uma ordenação flexível de cartas. A cada nova carta adicionada na mão, o valor dela pode ser menor ou maior em relação a outras cartas da mesma mão, criando assim uma forma de comparação entre as cartas e suas posições. A nova carta é inserida na posição correta, compondo uma nova ordenação de cartas, e assim, sucessivamente, se repete o procedimento até não receber mais cartas. Por essa ideia, as posições do array, começando com o índice 1, cada nova posição é como uma nova carta recebida, sendo inserida na posição correta no subarray ordenado à esquerda daquela posição.

**RESULTADOS**

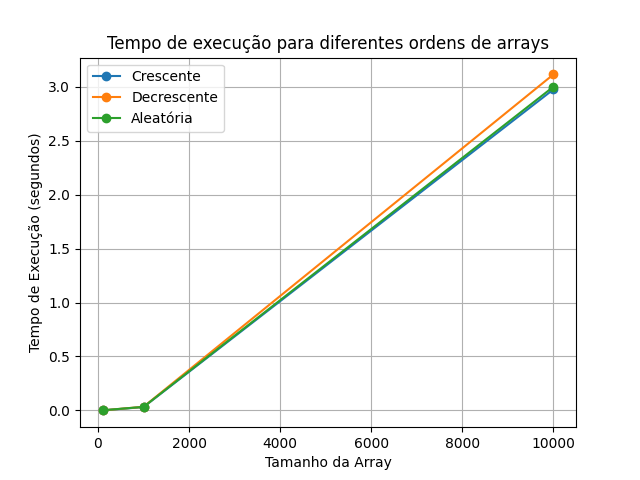
Portanto, utilizando-se da linguagem Python de programação, obtiveram-se os seguintes gráficos e tabelas ao aferir o tempo de execução de cada algoritmo ao ordenar uma array de números, sendo Ordem 1: uma lista ordenada em ordem crescente, Ordem 2: uma lista ordenada em ordem decrescente e Ordem 3: uma lista desordenada com números aleatórios. Foram usados também 3 tamanhos de arrays numéricas: 100, 1000 e 10000. Obtiveram-se os seguintes dados:

**Gráficos:**

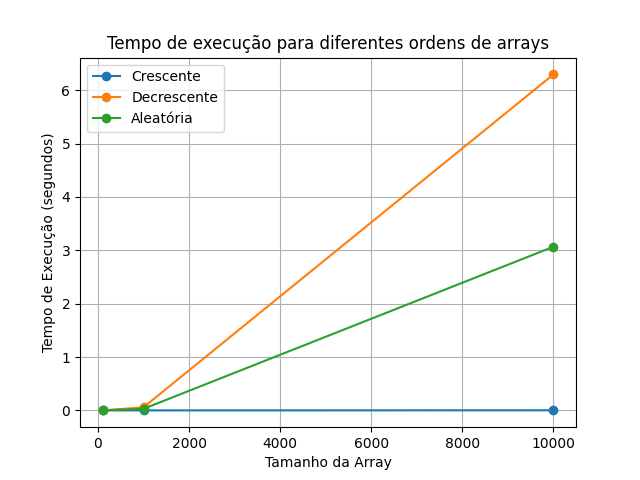
**Bubble Sort:**

****

**Selection Sort:**

****

**Insertion Sort:**

****

**Tabelas:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem 1** | **Tamanho de conjunto de dados = 100** | | |
| **Algoritmo** | **Tempo (ms)** | **Comparações** | **Movimentações** |
| **Bubble Sort** | 1 | 4950 | 0 |
| **Selection Sort** | 0,3 | 4950 | 0 |
| **Insertion Sort** | 0 | 0 | 99 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem 1** | **Tamanho de conjunto de dados = 1000** | | |
| **Algoritmo** | **Tempo (ms)** | **Comparações** | **Movimentações** |
| **Bubble Sort** | 34,5 | 499500 | 0 |
| **Selection Sort** | 31,51 | 499500 | 0 |
| **Insertion Sort** | 0 | 0 | 999 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem 1** | **Tamanho de conjunto de dados = 10000** | | |
| **Algoritmo** | **Tempo (ms)** | **Comparações** | **Movimentações** |
| **Bubble Sort** | 3700 | 49995000 | 0 |
| **Selection Sort** | 3010 | 49995000 | 0 |
| **Insertion Sort** | 1 | 0 | 9999 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem 2** | **Tamanho de conjunto de dados = 100** | | |
| **Algoritmo** | **Tempo (ms)** | **Comparações** | **Movimentações** |
| **Bubble Sort** | 1 | 4950 | 4950 |
| **Selection Sort** | 1,1 | 4950 | 50 |
| **Insertion Sort** | 0 | 4950 | 5049 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem 2** | **Tamanho de conjunto de dados = 1000** | | |
| **Algoritmo** | **Tempo (ms)** | **Comparações** | **Movimentações** |
| **Bubble Sort** | 72 | 499500 | 499500 |
| **Selection Sort** | 32 | 499500 | 500 |
| **Insertion Sort** | 50 | 499500 | 500499 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem 2** | **Tamanho de conjunto de dados = 10000** | | |
| **Algoritmo** | **Tempo (ms)** | **Comparações** | **Movimentações** |
| **Bubble Sort** | 7870 | 49995000 | 49995000 |
| **Selection Sort** | 3150 | 49995000 | 5000 |
| **Insertion Sort** | 6040 | 49995000 | 50004999 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem 3** | **Tamanho de conjunto de dados = 100** | | |
| **Algoritmo** | **Tempo (ms)** | **Comparações** | **Movimentações** |
| **Bubble Sort** | 1,2 | 4950 | 2773 |
| **Selection Sort** | 0,12 | 4950 | 96 |
| **Insertion Sort** | 1 | 2486 | 2585 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem 3** | **Tamanho de conjunto de dados = 1000** | | |
| **Algoritmo** | **Tempo (ms)** | **Comparações** | **Movimentações** |
| **Bubble Sort** | 58,3 | 499500 | 256362 |
| **Selection Sort** | 29,8 | 499500 | 995 |
| **Insertion Sort** | 30 | 248326 | 249325 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem 3** | **Tamanho de conjunto de dados = 10000** | | |
| **Algoritmo** | **Tempo (ms)** | **Comparações** | **Movimentações** |
| **Bubble Sort** | 6030 | 49995000 | 25321395 |
| **Selection Sort** | 3000 | 49995000 | 9994 |
| **Insertion Sort** | 3030 | 24725655 | 24735654 |

Ao analisar os dados obtidos, pela característica de todos os métodos serem métodos simples, percebe-se a clara dificuldade encontrada ao ordenar uma array ordenada decrescentemente, seguida da array de números aleatórios e da array ordenada crescentemente, sendo esta a de menor complexidade de ordenação. Também é interessante evidenciar a semelhança entre o número de comparações e movimentações iguais em algumas ordens em determinados algoritmos. Nota-se a estabilidade do método de Insertion Sort devido ao espaçamento equivalente entre os tempos de execução e o aumento do tempo de proporcional ao número de elementos nas arrays.

**CONCLUSÃO**

Logo, conclui-se que apesar do método de Insertion Sort possuir o tempo de execução da array ordenada crescente de 0 segundos, o algoritmo de ordenação de Selection Sort mostrou-se o mais eficiente para ordenar todas as arrays devido ao seu tempo de execução muito menor em todas as ordens e em quase todos os tamanhos de arrays, com exceção da array crescente, mostrando-se como o maior tempo de ordenação. Logo, o ranking por tempo de ordenação seria Selection Sort, seguido de Insertion Sort e Bubble Sort, de maneira geral, tendo o Bubble Sort como método mais ineficiente.

**REFERÊNCIAS**

Conheça os principais algoritmos de ordenação. Disponível em: <https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao#:~:text=Um%20m%C3%A9todo%20de%20ordena%C3%A7%C3%A3o%20%C3%A9>. Acesso em: 1 abr. 2024.

BALIEIRO, Ricardo. **Estrutura de Dados**. 1. ed. Rio de Janeiro: SESES, 2015. 176 p. ISBN 978-85-60923-34-2.