Universidade Estadual da Paraíba - UEPB

Centro de Ciências e Tecnologia - CCT

Curso de Ciências da Computação - CC

Roteiro comparativo dos algoritmos elementares de ordenação

Laryssa Finizola Costa da Silva

Campina Grande

Março - 2024

Universidade Estadual da Paraíba - UEPB

Centro de Ciências e Tecnologia - CCT

Curso de Ciências da Computação - CC

Projeto unidade 1 - parte 1

Laboratório de Estrutura de Dados

Laryssa Finizola Costa da Silva

Projeto desenvolvido junto ao curso de Ciências da Computação da Universidade Estadual da Paraíba, na cadeira de Laboratório de Estrutura de Dados, como requisito parcial à obtenção do título de Bacharel.

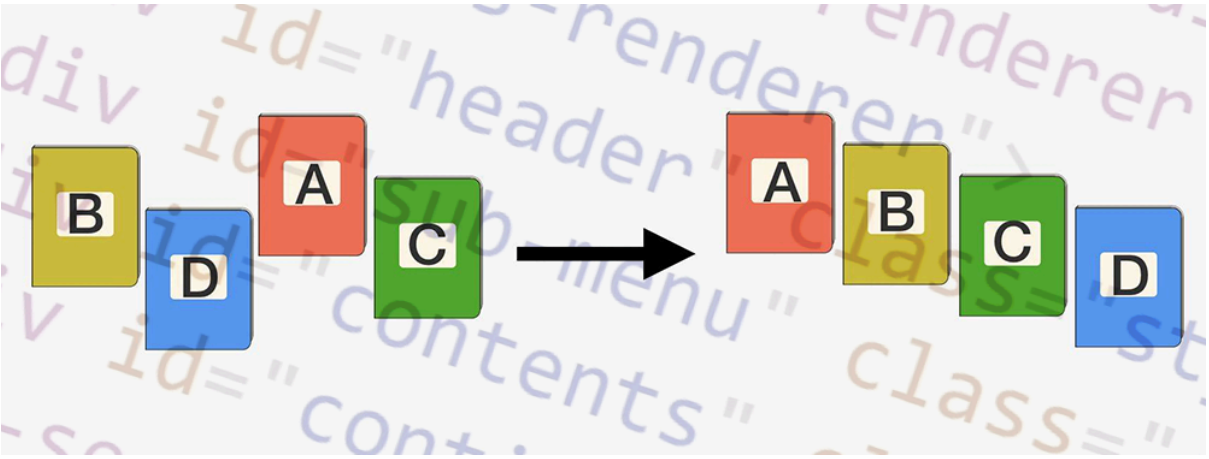
Orientador: Prof. Ms. Fábio Luiz Leite Júnior

SUMÁRIO

1.Introdução:

1.1. Objetivo:

Este relatório tem como objetivo fazer uma análise comparativa acerca de algoritmos de ordenação. Por sua vez, esses correspondem ao processo de reorganizar um conjunto de dados em uma ordem específica, como crescente ou decrescente.



Desse modo, esse tipo de reorganização oferece diversos benefícios como a facilidade na hora da busca, na compreensão e análise de dados, entre outros.
Sendo assim, o relatório baseia-se pelo estudo dos algoritmos Insertion Sort, Selection Sort e Bubble Sort, fazendo a comparação de medidas de complexidade a seguir:

a) Quantidade de Trocas realizadas;
b) Quantidade de comparações feitas.
c) Melhor tempo de execução;

Dessa maneira, sendo uma das principais finalidades desse tipo de algoritmo a ordenação de vetores, a análise proposta vai ser descrita com 3 tipos de massas de testes diferentes:

a) vetor de 1.000 elementos, os quais vão ser números aleatórios de 0 a 999.

b) vetor de 10.000 elementos, os quais vão ser números aleatórios de 0 a 9.999.
c) vetor de 100.000 elementos, os quais vão ser números aleatórios de 0 a 99.999.

Então, para cada algoritmo de ordenação foi passado um vetor de com esses elementos, de forma que esses números se organizassem para formar uma ordem crescente Sendo assim, ao fim, foram armazenadas as medidas de complexidade. Assim, foi possível organizar cada um e verificar o respectivo comportamento dos mesmos.

1.2. Metodologia:

A metodologia utilizada para alcançar os objetivos do trabalho foram:

a) O experimento foi todo realizado em uma máquina dedicada com as seguintes configurações: Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz, com 16,00 GB de memória RAM a fim de analisar o desempenho do algoritmo. Ademais, todos os códigos encontram-se na linguagem Java.

b) Realizou-se uma análise detalhada do comportamento dos algoritmos de ordenação como Selection Sort, Insertion Sort e Bubble Sort. Assim, sendo possível realizar a análise das formas que a configuração do tempo de execução, número de trocas e de comparações, após a inserção de dados, se ajustaram. Então, todos os dados foram computados e analisados por meio de tabelas e gráficos.

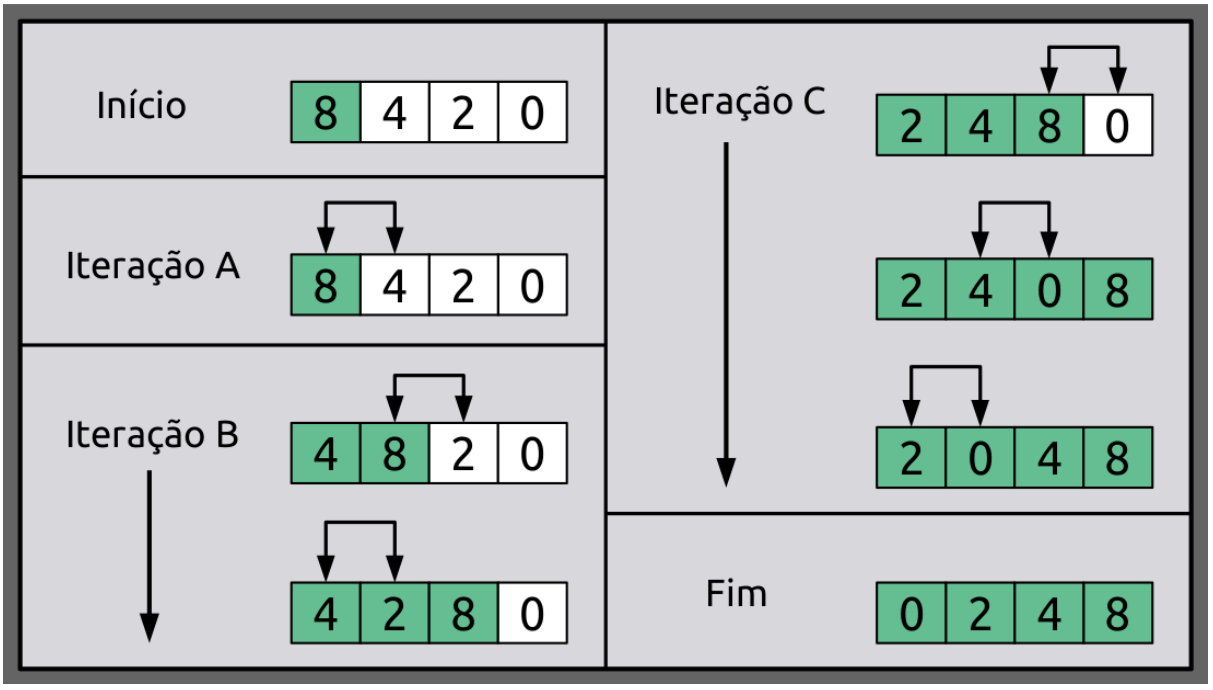
2. Insertion Sort:

2.1. Conceito:

Esse é um algoritmo de ordenação simples que cria o array final ordenando um item por vez.

Verifique como ocorre o funcionamento desse algoritmo de ordenação:

1. O primeiro passo envolve a comparação do elemento em questão com seu elemento adjacente.
2. Se a cada comparação for revelado que o elemento em questão pode ser inserido em uma posição específica, é criado espaço para ele deslocando os outros elementos uma posição para a direita e inserindo o elemento na posição adequada.
3. O procedimento acima é repetido até que todos os elementos no array estejam na posição apropriada.



Por esse motivo, a ordenação torna-se simples e seu funcionamento se dá por comparação e inserção direta

2.2: Análise Insertion:

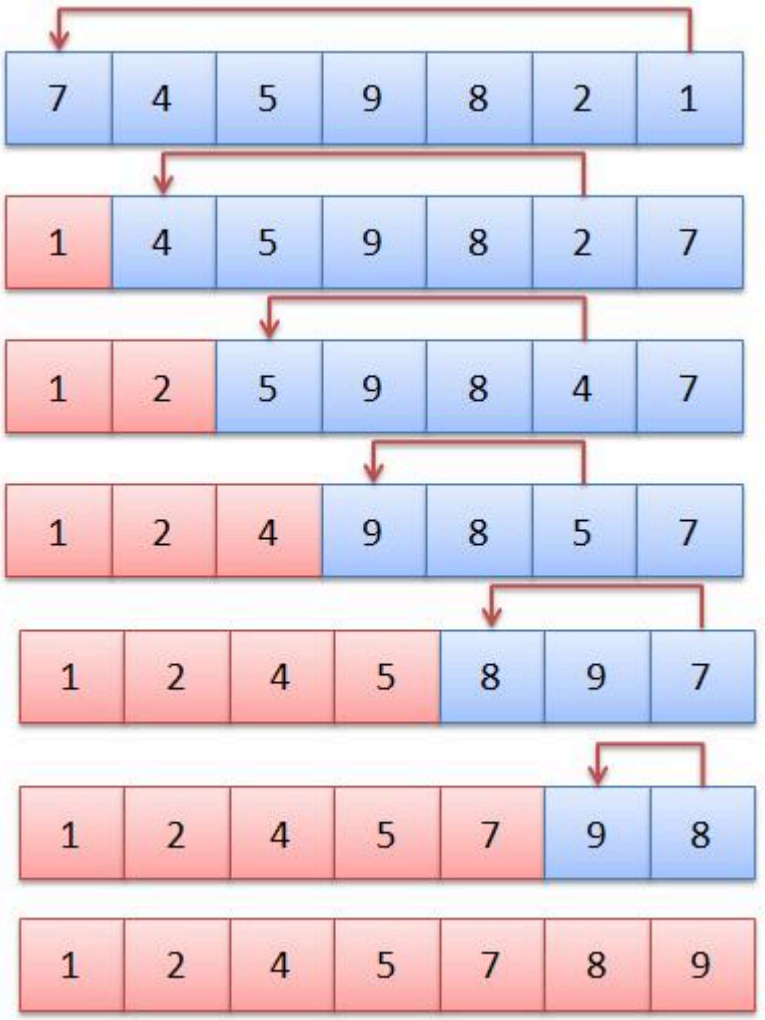
Desse modo, ao implementar o código, o Insertion Sort obteve esse resultado quando foram inseridos os dados e sendo ordenados em **ordem crescente:**

MASSAS DE TESTE	INSERTION SORT 1000 dados inseridos		
	1.000	10.000	100.000
TEMPO DE EXECUÇÃO (em milissegundos)	2 ms	47 ms	3528 ms
QUANTIDADE DE TROCAS	249579	25494322	1800655738
QUANTIDADE DE COMPARAÇÕES	250578	25504321	1800555739

2. Selection Sort:

3.1. Conceito:

Um dos mais simples e utilizados, o Selection Sort tem como principal finalidade passar o menor valor para a primeira posição, o segundo menor para a segunda posição, e assim sucessivamente, para os n valores nas n posições, onde o valor à esquerda é sempre menor que o valor à direita (valoresquerda < valordireita).



Ou seja, ele sempre vai em busca do menor valor dentro de um array.

3.2. Análise Selection:

Desse modo, ao implementar o código, o Selection Sort obteve esse resultado quando foram inseridos os dados e sendo ordenados em **ordem crescente:**

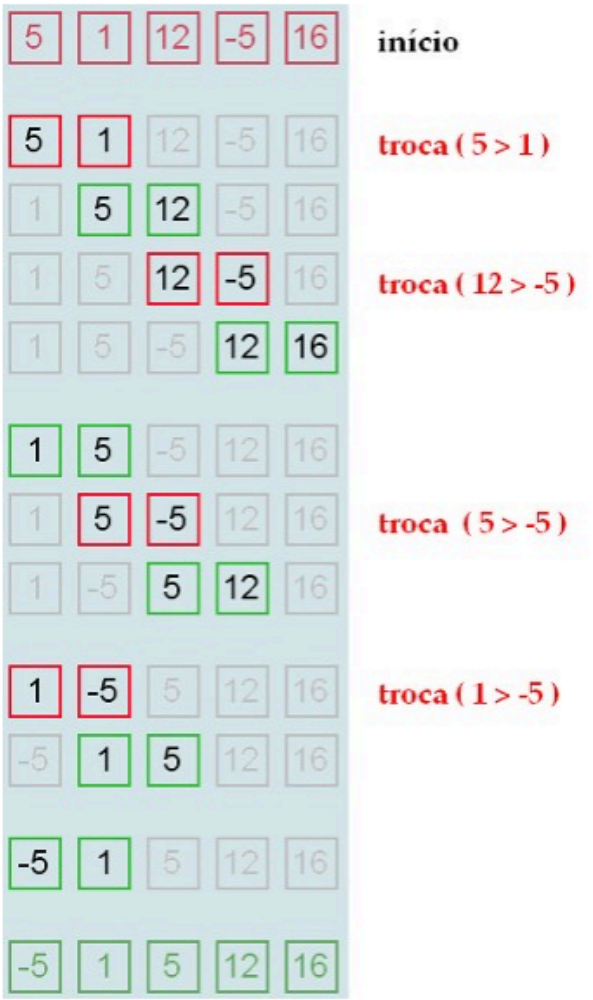
MASSAS DE TESTE	SELECTION SORT 1000 dados inseridos		
TEMPO DE EXECUÇÃO (em milissegundos)	1.000	10.000	100.000
	2 ms	48 ms	3913 ms
QUANTIDADE DE TROCAS	990	9994	99987
QUANTIDADE DE COMPARAÇÕES	499500	49995000	704982704

4. Bubble Sort:

4.1. Conceito:

O método da Bolha pode ser resumido em:

1. Comparar dois elementos adjacentes: se o primeiro é maior do que o segundo, ambos são trocados.
2. Realizar a comparação/troca definida no item 1 para todos os pares de elementos adjacentes, começando com os dois primeiros e terminando com os dois últimos. Neste ponto o último elemento é o maior.
3. Repetir o passo 2 para todos os elementos, com exceção do último sucessivamente.



Ou seja, se o objetivo é ordenar os valores em forma *crescente*, então, a **posição atual é comparada com a próxima posição e, se a posição atual for menor que a posição posterior, é realizada a troca.** Caso contrário, a troca não é feita e passa-se para o próximo par de comparação.

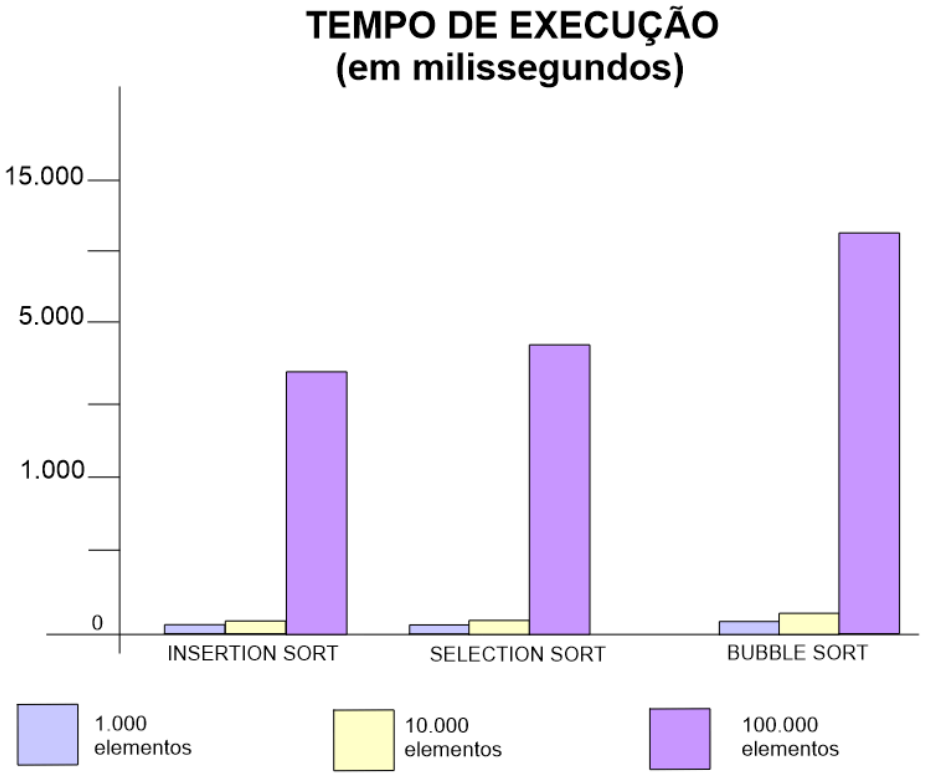
4.2. Análise Bubble:

Desse modo, ao implementar o código, o Bubble Sort obteve esse resultado quando foram inseridos os dados e sendo ordenados em **ordem crescente:**

MASSAS DE TESTE	BUBBLE SORT 1000 dados inseridos		
TEMPO DE EXECUÇÃO (em milissegundos)	1.000	10.000	100.000
	6 ms	127 ms	13862 ms
QUANTIDADE DE TROCAS	242105	25017046	1795882378
QUANTIDADE DE COMPARAÇÕES	933066	99090090	1369265815

5. Conclusão:

Diante dos testes realizados, conseguimos perceber que para o nosso cenário de testes (números crescentes) o insertion sort teve um melhor desempenho em todas as medidas que coletamos, de acordo com o gráfico:



Então, isso se dá pois o insertion roda em **O(n)** no melhor caso. Assim, ele acaba sendo um algoritmo melhor para propósitos gerais. Assim, para o insertion, temos:

- Complexidade de tempo:
- Pior caso: O(n^2) comparações e trocas.
- Melhor caso (quando a lista já está ordenada): O(n) comparações e O(1) trocas.

Por outro lado, de acordo com os testes realizados, o Bubble Sort foi aquele com o pior resultado na análise de medidas, isso ocorreu de forma que o Bubble se mostrou bem mais lento do que o Insertion e o Selection sort, além de que os números de trocas e de comparações foram bem maiores.

Ao falar apenas nos números de trocas, o selection obteve os melhores resultados, uma vez que por conta do seu funcionamento de percorrer a lista atrás do menor elemento, o número de trocas realmente tende a ser menor.