UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS DE RIO PARANAÍBA
SISTEMAS DE INFORMAÇÃO

GLEIDSON VINÍCIUS GOMES BARBOSA – 6331

ANÁLISE DE UM ALGORITMO DE ORDENAÇÃO - INSERTION SORT

RIO PARANAÍBA
29 de setembro de 2022

GLEIDSON VINÍCIUS GOMES BARBOSA – 6331

ANÁLISE DE UM ALGORITMO DE ORDENAÇÃO - INSERTION SORT

Trabalho apresentado para obtenção de créditos na disciplina SIN213 - Projeto de Algoritmos da Universidade Federal de Viçosa - Campus de Rio Paraníba, ministrada pelo Professor Pedro Moisés de Souza.

Orientador: Pedro Moisés de Souza

RIO PARANAÍBA

29 de setembro de 2022

Resumo

Os algoritmos de ordenação apresentam grande importância no estudo teórico de algoritmos, bem como em aplicações rotineiras e práticas do cotidiano. Diante dessa grande importância ao longo dos anos surgem alguns algoritmos de ordenação na literatura, onde são analisados casos em que cada algoritmo se enquadra. Sendo assim se torna necessário um estudo para analisar o comportamento desses algoritmos, bem como suas complexidades para indicar qual é o mais adequado para cada tipo de problema. Neste trabalho será estudado o algoritmo Insertion Sort ou ordenação por inserção que como descrito em (WIKIPEDIA, 2022) é um algoritmo de ordenação quadrático, onde são realizados testes de elemento por elemento onde testamos uma condição para entrar em um loop que irá se repetir até que o elemento seja posicionado em ordem comparando com o próximo elemento e com os anteriores.

Palavras-chaves: Insertion Sort, algoritmos de ordenação, projeto de algoritmos.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Neste trabalho, objetiva-se analisar o algoritmo insertion sort e fazer testes com entradas para melhor caso, pior caso e caso médio e comparar seus desempenhos analisando sua complexidade e observar se os tresultados práticos condizem com a teoria apresentada. Apesar de existirem diversas ferramentas na web como (TOPTAL, 2022) e (USF, 2022) onde podemos ver um pouco da teoria além de um exemplo do funcionamento do algoritmo, para este trabalho desenvolvemos um software em linguagem C conforme a teoria apresentada em sala de aula, vista também em (WIKIPEDIA, 2022) e no já citado Toptal. Neste software geramos as entradas compatíveis com cada caso para 10, 100, 1.000, 10.000, 100.000 e 1.000.000 de instâncias, executamos testes e gravamos os dados em arquivos, tanto as entradas quanto saída destes dados ordenados e o tempo de execução do algoritmo e analisamos os resultados com base na teoria.

2 Análise e Complexidade do Algoritmo

Conforme visto em sala de aula, o algoritmo usa é apresentado utilizando uma sequência A de n elementos, observamos isso na tabela a seguir:

Insertion Sort(A,n)	**Custo**	**Vezes**
1 for j <- 2 to comprimento	C1	n
2 do chave < - A[j]	C2	n-1
3 "Inserir A[j] na sequência ordenada A[1.2.....j-1]"	C3 = 0	n-1
4 i <- j-1	C4	n-1
5 while i>0 e A[i] >chave	C5	$\sum_{t=2}^{n} tj$
6 do A[i+1] <- A[i]	C6	$\sum_{t=2}^{n}(tj-1)$
7 i <- i-1	C7	$\sum_{t=2}^{n}(tj-1)$
8 A[i+1] <- chave	C8	n-1

Fonte: Algoritmo visto em sala de aula

Ao observar o algoritmo apresentado, nota-se que é possível calcular sua complexidade multiplicando o custo de cada linha de sua execução por pela quantidade de vezes que esta é executada, assim chegamos na seguinte equação:

$$C1n+C2(n-1)+C4(n+1)+C5[\sum_{t=02}^{n} tj]+C6[\sum_{t=02}^{n}(tj-1)]+C7[\sum_{t=02}^{n}(tj-1)]+C8(n-1) \quad (2.1)$$

Sendo tj o número de vezes que o loop foi executado.Observa-se que a linha 3 tem custo 0 visto que apenas apresenta um comentário. Para chegar ao melhor caso, pior caso e caso médio, basta desenvolver tal equação utilizando os seguintes valores:

2.1 Melhor Caso

$$tj = 1 \quad (2.2)$$

Neste caso teremos que o nosso tj = 1 e na linha 5 a condição do loop é falsa, logo as linhas 6 e 7 não são executadas. Assim ao desenvolver a equação da complexidade do algoritmo temos:

$$(C1 + C2 + C3 + C4 + C5 + C8)n - (C2 + C4 + C5 + C8) \quad (2.3)$$

que podemos simplificar para:

$$An - B \tag{2.4}$$

chegando assim a uma função linear de n que é o melhor caso.

2.2 Pior Caso

$$tj = j \tag{2.5}$$

Neste caso, o vetor inicial estará em ordem decrescente, o que nos fará entrar em nosso loop e comparar cada elemento do nosso vetor principal com cada elemento do subarranjo ordenado, neste caso teremos que o nosso tj = j para todo jogo de 2 até n. O que nos leva a desenvolver utilizando propriedades matemáticas os nossos somatórios, chegando a seguintes igualdades:

$$\sum_{t=02}^{n} tj = \sum_{t=2}^{n} j = \frac{n(1+n)}{2} - 1 \tag{2.6}$$

$$\sum_{t=02}^{n} (tj - 1) = \sum_{t=2}^{n} tj - \sum_{t=2}^{n} 1 = \frac{n(1+n)}{2} - 1 - (n-1) = \frac{n(n-1)}{2} \tag{2.7}$$

Assim, ao desenvolver a equação de complexidade do algoritmo aplicando esta condição temos:

$$(\frac{C5 + C6 + C7}{2})n^2 + (C1 + C2 + C4 + C8 + \frac{C5 - C6 - C7}{2})n + (-C2 - C4 - C5 - C8) \tag{2.8}$$

que podemos simplificar para:

$$An^2 + Bn + C \tag{2.9}$$

Chegando assim a uma função quadrática de n que é o pior caso.

2.3 Caso Médio

$$tj = \frac{j}{2} \tag{2.10}$$

No caso médio, assumimos que teremos nossa condição para entrar no loop cumprida em média 50% das vezes, logo temos tj = j/2, utilizando as propriedades matemáticas chegamos nas seguintes igualdades:

$$\sum_{t=02}^{n} tj = \sum_{t=2}^{n} \frac{j}{2} = \frac{1}{2}(\frac{n(1+n)}{2} - 1) = \frac{n(1+n)}{4} - \frac{1}{2} \tag{2.11}$$

$$\sum_{t=02}^{n}(tj-1)=\sum_{t=2}^{n}tj-\sum_{t=2}^{n}1=\frac{n(1+n)}{4}-\frac{1}{2}-(n-1)=\frac{n(1+n)}{4}-n+\frac{1}{2} \tag{2.12}$$

Assim, ao desenvolver a equação de complexidade do algoritmo aplicando esta condição temos:

$$(\frac{C5+C6+C7}{4})n^2+(C1+C2+C4+C8-C6-C7+\frac{C5-C6-C7}{4})n+(-C2-C4-C5-C8-\frac{C5-C6} \tag{2.13}$$

que podemos simplificar para:

$$An^2+Bn+C \tag{2.14}$$

Chegando assim a uma função quadrática de n que é o caso médio, observe que é igual ao nosso pior caso.

3 Tabela e Gráfico do Algoritmo

O Algoritmo foi implementado conforme apresentado em sala de aula, apenas com adaptações para a linguagem solicitada conforme a imagem a seguir:

```
void insertionSort(int *vetor, int tamanho)
{
    int i, j;
    for(i = 1; i < tamanho; i++)
    {
        int x = vetor[i];
        j = (i-1);
        while(j >= 0 && vetor[j] > x)
        {
            vetor[j+1] = vetor[j];
            vetor[j] = x;
            j--;
        }
    }
}
```

Figura 1 – Código Insertion Sort

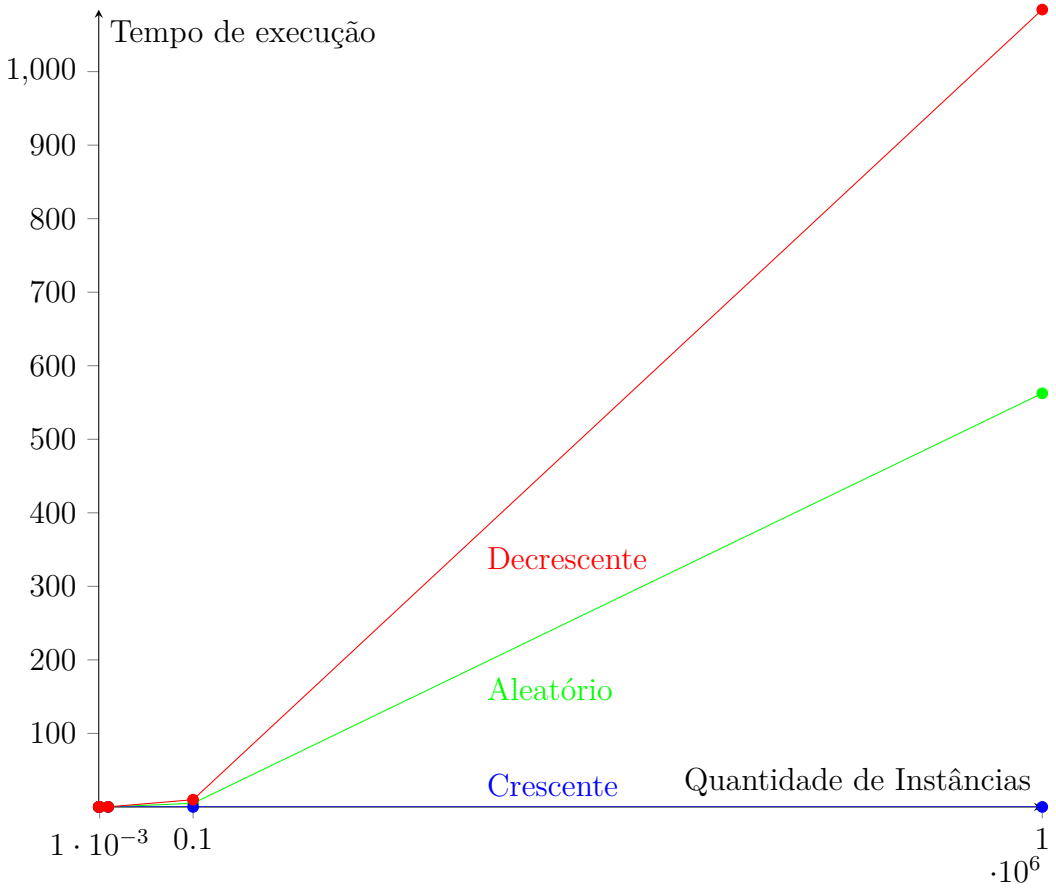
Fonte: Autoria própria

Foram realizados testes para cada entrada que variam entre 10, 100, 1.000. 10.000, 100.000, 1.000.000 instâncias e são ordenados de três formas: Crescente, Decrescente e Aleatório. Para cada combinação de quantidades de instâncias e tipo ordenação foi realizada uma execução do algoritmo. Os resultados da ordenação e o tempo de execução são gravados em arquivos diferentes para que se possa observar o resultado da ordenação e o tempo de execução do algoritmo. Os resultados destes testes são apresentados na tabela e gráfico a seguir:

Tabela 1 – Tempos por tipo e tamanho das entradas.

	10	**100**	**1.000**	**10.000**	**100.000**	**1.000.000**
Aleatório	0.0140	0.0140	0.0160	0.0760	4.9940	562.4980
Crescente	0.0130	0.0150	0.0140	0.0150	0.0140	0.0140
Decrescente	0.0170	0.0140	0.0150	0.1310	9.8410	1084.3570

Fonte: Autoria própria



Assim é possível observar que conforme a teoria nos propõe com a análise da complexidade, os resultados dos testes apresentam desempenho compatível com aquilo que foi proposto.

4 Conclusão

Portanto, levando em consideração os estudos e os testes feitos método de Ordenação Insertion Sort ao longo deste trabalho pode-se perceber que em uma ordenação com 10, 100 e 1.000 instâncias, o algoritmo possui o mesmo desempenho, tanto para crescente, decrescente e randômico, a partir de 10.000 instâncias já se pode perceber uma pequena variação no tempo de execução que logo se torna uma grande diferença com 100.000 e 1.000.000 de instâncias. Assim confirma-se que conforme a teoria os desempenhos alcançam desempenho compatível a complexidade apresentada para todos os casos.

Referências

TOPTAL. **Toptal**. 2022. Disponível em: <https://www.toptal.com/developers/sorting-algorithms/insertion-sort>. Acesso em: 28 sep. 2022. Citado na página 6.

USF. **USF Computer Science Department**. 2022. Disponível em: <https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html>. Acesso em: 28 sep. 2022. Citado na página 6.

WIKIPEDIA. **Wikipedia, the free encyclopedia**. 2022. Disponível em: <https://en.wikipedia.org/wiki/Insertion_sort>. Acesso em: 28 sep. 2022. Citado 2 vezes nas páginas 2 e 6.