

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Instituto de Ciências Exatas e de Informática

Laboratório 06 – Ordenação – Algoritmos quadráticos

Gustavo Azi Prehl Gama

Resumo

Neste laboratório, buscamos aperfeiçoar a prática de implementação de algoritmos de ordenação, fazer a análise de complexidade e a medição de desempenho. O trabalho é dividido em quatro partes, sendo elas: 1- Execução dos algoritmos de ordenação 2- Aferição de desempenho de cada algoritmo na determinada máquina 3- Análise de desempenho dos algoritmos 4- Comparação entre a análise de complexidade e a de desempenho. A primeira parte é apenas implementar os algoritmos já estudados, será utilizada nas demais. Na segunda etapa será apresentado os resultados, os experimentos variam de máquina para máquina. No próximo passo, será comparado todas as informações coletadas, quais as vantagens e desvantagens, semelhanças e diferenças na análise de complexidade. E por fim, a conclusão do trabalho, que após análise de todos os pontos teremos o melhor, médio e pior desempenho, o número de comparações, memória gasta, e assim, chegando à ideia final, de qual seria o mais recomendado.

**Palavras-chave:** Algoritmos, complexidade, desempenho, resultados.

Abstract

In this lab, we seek to improve the pratice of implementing sorting algorithms, performing complexity analysis and measuring performance. The work is divided into four parts, namely: 1- Execution of the sorting algorithms 2- Performance measurement of each algorithm on a given machine 3- Algorithm performance analysis 4- Comparison between the complexity and performance analysis. The first part is just to implement the algorithms already studied, it will be used in the others. In the second stage, the results will be presented, the experiments vary from machine to machine. In the next step, all the information collected will be compared, what are the advantages and disadvantages, similarities and differences in the complexity analysis. And finally, the conclusion of the work, that after analyzing all the points we will have the best, average and worst performance, the number of comparisons, memory used, and thus, reaching the final idea, which would be the most recommended.

**Keywords:** Algorithm, complexity, performance, results.

1. INTRODUÇÃO

O objetivo do trabalho é realizar um experimento com os algoritmos de ordenação quadráticos, analisando seu desempenho e complexidade entre três casos, esses que serão considerados como melhor, pior e intermediário caso. Para avaliar o comportamento, será mostrado um gráfico para tempo e um para memória.

1. DESENVOLVIMENTO

O primeiro passo é a implementação dos três algoritmos, sendo eles: bolha, inserção e seleção. A partir disso começaremos uma análise em cima do tempo gasto de cada um, e também da memória. Logo em seguida, veremos qual a complexidade de cada caso, observando diferenças e semelhanças, e quais as vantagens e desvantagens.

2.1 Algoritmo bolha

Algorithm 1: Bubble sort

1: #include <stdio.h>

2:

3: void bolha (int \*array, int n) {

4: int i, j;

5: for (i = (n-1); i > 0; i--) {

6: for (j = 0; j < i; j++) {

7: if(array[j] > array[j+1] {

8: swap(&array[j], &array [j + 1];

9:}

10:}

11:}

12:}

13: FIM

2.1.1 Algoritmo inserção

Algorithm 2: Insetion sort

1: #include <stdio.h>

2:

3: void insercao (int \*array, int n) {

4: for (int i = 0; i < n; i++) {

5: int temp = array[i];

6: int j = i – 1;

7:

8: while ((j >= 0) && (array[j] > tmp)) {

9: array [j + 1] = array[j];

10: j--;

11:}

12:array [j + 1] = tmp;

13:}

14:}

15: FIM

2.1.1.1 Algoritmo seleção

Algorithm 3: Selection sort

1: #include <stdio.h>

2:

3: void selecao (int \*array, int n) {

4: for (int i = 0; i < (n – 1); i++) {

5: int menor = i;

6: for (int j = (i + 1); j < n; j++) {

7: if (array[menor] > array[j]) {

8: menor = j;

9:}

10:}

11: swap (&array[menor], &array[i]);

12:}

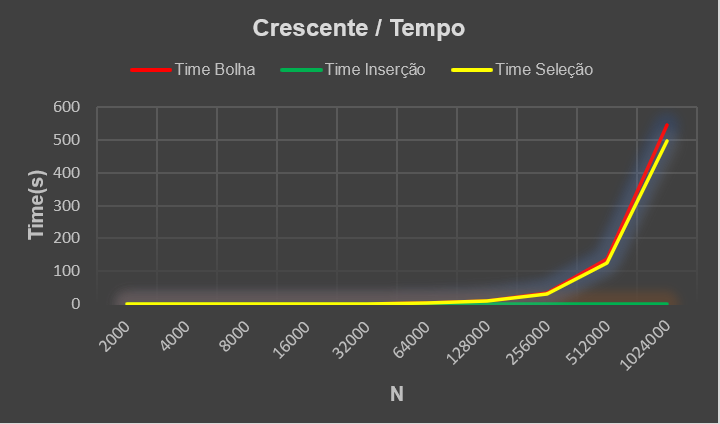
13:}

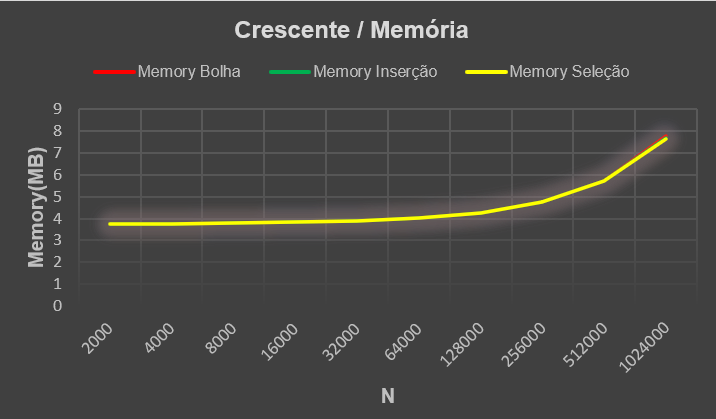
14: FIM

1. Análise de desempenho experimental

Após vermos quais foram os algoritmos utilizados, veremos os resultados obtidos após a implementação dos mesmos. Para tal análise foram construídos seis gráficos, dois para cada caso. Os gráficos ficaram divididos entre tempo e memória, a separação desses valores tem o intuito de melhor visualização da atividade.

* 1. Gráficos do melhor caso



Este primeiro gráfico têm a comparação entre o tempo dos três algoritmos, o tempo está em segundos. Nota-se que o bolha levou aproximadamente 550s para atingir os 10 casos testes, já o seleção se aproximou de 500s e o inserção se manteve com 0s para realizar todos.

Agora nós temos os valores da memória, são números próximos, mas o bolha ainda continua na frente, alcançando aproximadamente 7,76MB, em segundo temos o de inserção com valor próximo à 7,68MB, e por último o seleção com 7,66MB. Os picos dos valores da memória são bem parecidos, ainda mais se compararmos com o gráfico de tempo.

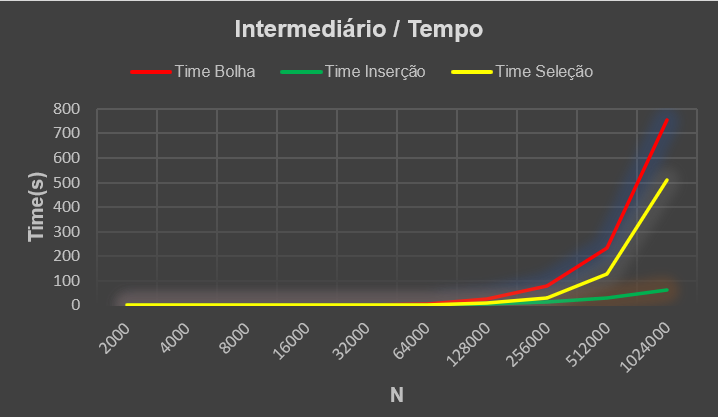
Vamos falar sobre a complexidade destes algoritmos, mas antes, o que é a complexidade de um algoritmo? É a quantidade de trabalho necessário para executar uma tarefa. Isto é medido em cima das funções fundamentais que o algoritmo é capaz de fazer, o volume de dados, e a maneira de como ele chega no resultado.

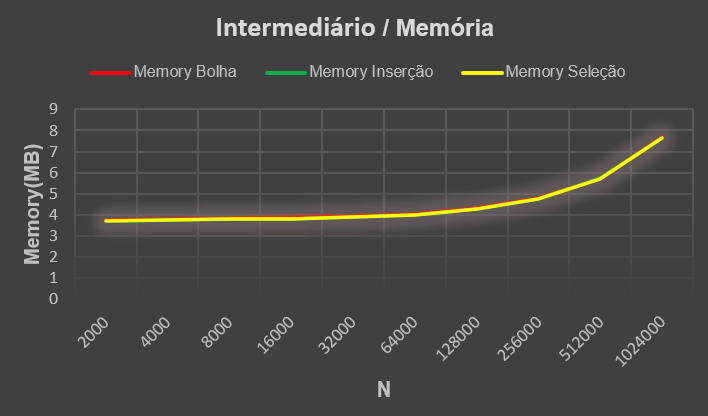
O melhor caso do bubble sort(bolha) ocorre quando o vetor já está ordenado, pois o laço será executado apenas uma única vez, quando i = 1. Assim, a comparação será realizada n – 1 vezes. Logo, a complexidade do bubble sort no seu melhor caso é Ο(n).

O melhor caso do insertion sort(inserção) também ocorre quando o vetor já está ordenado, e ele também é Ο(n).

O selection sort(seleção) compara a cada interação um elemento com os outros, visando encontrar o menor. Dessa forma, podemos entender que não existe um melhor caso mesmo que o vetor esteja ordenado ou em ordem inversa serão executados os dois laços do algoritmo, o externo e o interno. A complexidade deste algoritmo será sempre Ο ().

* 1. Gráficos do caso intermediário



Novamente vamos analisar um gráfico de tempo, agora de caso intermediário. O bolha continua liderando os algoritmos atingindo o maior tempo, dessa vez o intervalo dele para os outros é maior, se aproximando de 755s, o seleção teve uma diferença de 242s para o anterior, atingiu por volta de 510s e por último o de inserção, que teve uma pequena diferença, desta vez o seu tempo foi quase 62s.

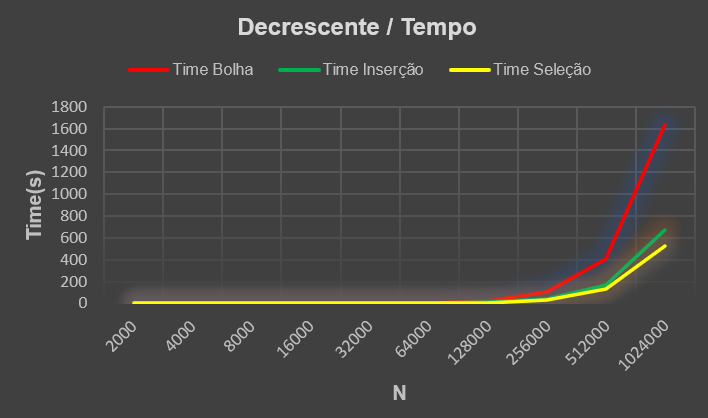
O gráfico de memória conta com números bem parecidos, visto que o bolha está um pouco a frente dos outros com o gasto a cerca de 7,71MB, no caso do algoritmo de seleção e inserção ambos estão com valores em torno de 7,66MB, fazendo com que uma linha fique em cima da outra.

Analisando a complexidade dos algoritmos em seu caso intermediário, com valores aleatórios e sem ordem de classificação (crescente ou decrescente).

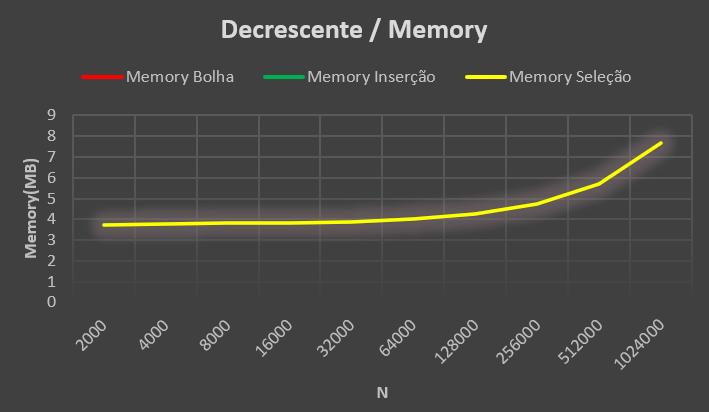
No caso intermediário do bubble sort(bolha) temos complexidade Ο ().

No caso intermediário do insertion sort(inserção) temos complexidade Ο ().

No caso intermediário do selection sort(seleção) temos complexidade Ο ().

* 1. Gráficos do pior caso

Vamos para o último e pior caso, o bolha continua como o algoritmo mais demorado, alcançando quase 1630s, o diferencial deste gráfico para os outros é que o inserção ultrapassou o seleção no quesito tempo, ele se aproxima de 670s enquanto o seleção está próximo de 530s.



Neste gráfico, a memória gasta é idêntica. Todos tiveram um gasto aproximado de 7,66MB.

Analisando a complexidade dos algoritmos em seu pior caso, ou seja, os valores na ordem inversa daquele que se deve retornar.

No pior caso do bubble sort(bolha) temos complexidade Ο ().

No pior caso do insertion sort(inserção) também temos complexidade Ο ().

E, o selection sort(seleção) como dito anteriormente sempre será Ο ().

1. Vantagens e desvantagens

Bubble sort(bolha):

* Simples de entender e implementar.
* Uma desvantagem é que na prática ele tem execução lenta mesmo quando comparado a outros algoritmos quadráticos ().
* Tem um número muito grande de movimentação de elementos, assim não deve ser usado se a estrutura a ser ordenada for complexa.

Insertion sort(inserção):

* É o método a ser utilizado quando o arquivo está "quase" ordenado
* É um bom método quando se desejar adicionar poucos elementos em um arquivo já ordenado, pois seu custo é linear.
* O algoritmo de ordenação por inserção é estável.
* Alto custo de movimentação de elementos no vetor.

Selection sort(seleção):

* Uma vantagem do Selection Sort é que entre os algoritmos de ordenação ele apresenta uma das menores quantidades de movimentos entre os elementos, assim pode haver algum ganho quando se necessita ordenar estruturas complexas.
* Uma desvantagem é que o número de comparações é igual para o melhor caso, caso médio e o pior caso. Assim, mesmo que o vetor esteja ordenado o custo continua quadrático ().
* Não é estável (depende da implementação).

5.0 Especificação da máquina e links de redirecionamento

Processador: 11th Gen Intel(R) Core (TM) i7-11700 @ 2.50GHz 2.50 GHz

Memória RAM: 16gb

<https://github.com/gustavoprehl/Ordenation_Code.git>

6.0 Conclusão

Após a análise de todos os casos, com esses valores a melhor escolha seria pelo algoritmo de inserção, mas em outras situações ele pode acabar sendo uma escolha não muito correta, temos esse exemplo no pior caso, onde o algoritmo de seleção mostrou-se mais eficiente, mantendo a mesma memória e com um menor tempo. Vendo a análise experimental e a de complexidade, percebe-se que entre os três o algoritmo de seleção se manteve bem rente nos testes já que todos mantém a mesma complexidade, o algoritmo do bolha é perceptível a crescente curva entre os três, tanto é que seu tempo variou entre 500 e 1700 segundos, e no caso do algoritmo de inserção têm uma crescente, visto que ele realizou o melhor caso em 0s, o intermediário em aproximadamente 1 minuto e o pior caso quase em 12 minutos. Percebe-se que quando foi realizado os casos testes a memória variou mais, a partir do quarto ela se estabilizou até o fim de todos.