**UNIVERSIDADE FEDERAL DE OURO PRETO - ICEA**

JOSÉ MARTINS DA ROSA JÚNIOR

**TP – Algoritmo e Estrutura de Dados**

Análise Comparativa: algoritmos de ordenação

João Monlevade

2018

Exercício I**3**

Exercício II4

Exercício III5

Exercício IV**9**

Exercício V **11**

Exercício VI**12**

Exercício VII**13**

Exercício VIII**18**

Exercício IX**20**

Exercício X**21**

Exercício XI**23**

Exercício XII**24**

Exercício XIII**26**

Introdução

Ao longo dos tempos, com o avanço tecnológico, as vezes fica um pouco esquecido o custo de memória e tempo de alguns algoritmos, devido aos altos índices de memória primária e velocidade dos processadores atuais.

Esse trabalho consiste em fazer uma análise comparativa do tempo de execução, número de trocas e número de comparações dos algoritmos de ordenação *Insertion Sort, Selection Sort, Shell Sort* e *Quick Sort.*

Implementação

Os algoritmos de ordenação foram implementados na linguagem C. Para contagem do tempo de execução do algoritmo, usou-se a função “clock()”, inclusa na biblioteca “<time.h>”.

A contagem de comparações e número de trocas que cada algoritmo realizou, será feita com contadores dentro das funções nas posições que correspondem ao pedido.

*Insertion Sort*

Algoritmo estável que ordena os itens por inserção. É um ótimo algoritmo para uma baixa quantidade de dados para ordenação, porém para altas quantidades, ele não é efetivo, pois o seu tempo gasto é alto. Segue implementação utilizada:

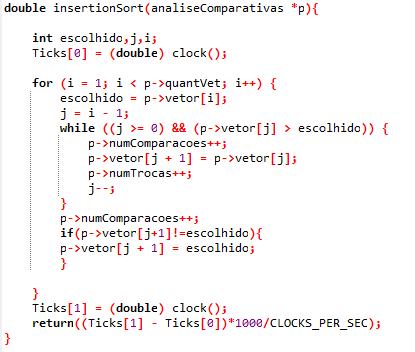


Figura 1: Implementação do *Insertion Sort.*

*Selection Sort*

Algoritmo de ordenação por seleção. Ele é um algoritmo eficiente para pequenas cargas, porém por sua complexidade ser O(n²) para qualquer caso, ele não é muito efetivo para alta quantidade de dados. Segue implementação utilizada:

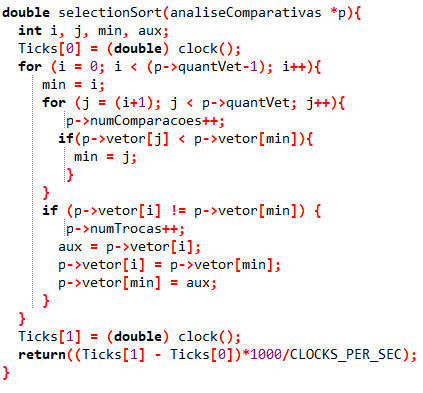


Figura 2: Implementação do *Selection Sort.*

*Shell Sort*

É o algoritmo mais eficiente dentre os de complexidade quadrática. Ele divide o grupo maior do vetor, em outros menores e aplica o algoritmo de inserção, fazendo assim, ser um algoritmo mais eficiente que os demais citados. Segue implementação utilizada:

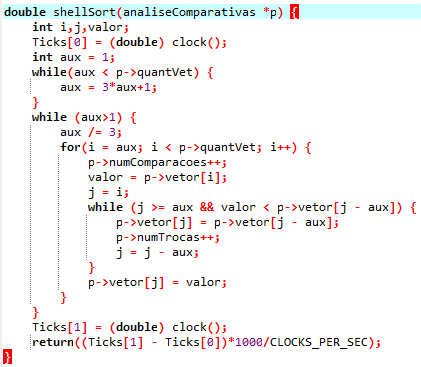


Figura 3: Implementação do *Shell Sort.*

*Quick Sort*

Algoritmo de divisão e conquista. Ele rearranja as chaves de forma que as chaves à esquerda de seu pivô sejam menores que ele e as chaves à direita do pivô sejam maiores. É um algoritmo mais eficiente e o mais utilizado profissionalmente. Sem dúvidas, é o melhor algoritmo de ordenação. Segue implementação:

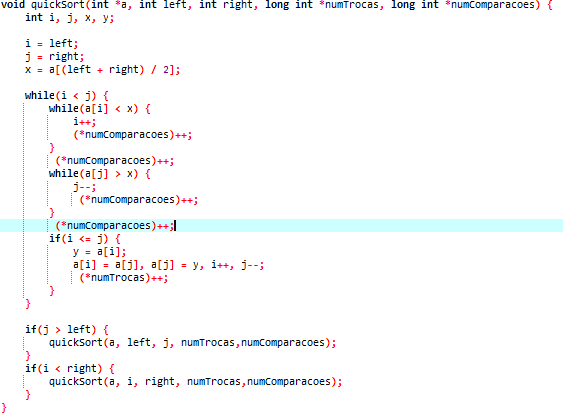


Figura 4: Implementação do *Quick Sort.*

Metodologia

Para realização da análise, o computador teve todas as suas aplicações fechadas, exceto o compilador do programa para execução dos testes.

Os testes foram executados no computador com as seguintes configurações:

* Processador Core I3 CPU 540 @ 3.07GHz – 3.06 GHz;
* Memória RAM 2GB DDR 3;
* HD Sata Samsung 1 TB(teraByte);
* Placa de Vídeo GT 220

A análise foi feita da seguinte forma:

* Leitura do arquivo “*.dat*” com os dados, incluindo-os em um vetor, sendo que cada posição do vetor inclui um elemento do tipo (int);
* Aplicação dos algoritmos de ordenação no vetor com os itens do arquivo;
* Leitura dos resultados em outro arquivo “.*dat*”.

Foi realizado o teste para vetores com 1000, 10000 e 100000 elementos. Não menos que 1000, pois não teríamos informações com um mínimo de volume, devido as configurações do computador e não mais que 100000 pela ineficiência em relação ao tempo de uns algoritmos utilizados nesse experimento para vetores com muitos elementos, fazendo que dê uma demora mais que considerável para execução de um teste.

Foram realizados 33 experimentos para cada caso, sendo eles:

* Vetor com itens em ordem crescente.
* Vetor com itens em ordem decrescente.
* Vetor com itens de forma randômica.

Resultados

Os resultados serão divididos em três partes: 1 – Vetor com 1000 elementos. 2 – Vetor com 10000 elementos. 3 – Vetor com 100000 elementos.

1. – VETOR COM 1000 ITENS:

* Ordem crescente



Tabela 1: Vetor com 1000 itens em ordem crescente.

Vemos que o Insertion Sort, apresenta maior eficiência para vetores ordenados de forma crescente, por ter o menor número de comparações e seu número de trocas zerado. Observe que o Selection Sort ainda realiza muitas comparações, pois a sua complexidade é sempre O(n²), independente da situação. O Shell Sort por sua vez, apresenta uma vantagem ao Quick Sort, pois o pior caso do Quick Sort é quando as chaves estão ordenadas de forma crescente.

* Ordem decrescente



Tabela 2: Vetor com 1000 itens em ordem decrescente

Nesse caso, encontramos a ineficiência do Insertion Sort para vetores ordenados de forma decrescente. Observe que ele tem o maior número de trocas, o maior número de comparações e o maior tempo gasto. O Selection Sort continua com o seu mesmo número de comparações de quando o vetor estava em ordem crescente, porém faz menos trocas que os seus concorrentes. Em contrapartida, tem um tempo gasto superior ao Quick Sort e ao Shell Sort, que tiveram seus tempos zerados.

* De forma randômica



Tabela 3: Vetor com 1000 itens de forma randômica

Dessa vez, vemos um alto número de comparações e trocas do Insertion Sort e um tempo considerável, quando comparado ao Quick Sort e ao Shell Sort. O Selection Sort segue tendo seu número de comparações grande, mas ainda manteve o número de trocas inferior. Entretanto, teve seu tempo gasto superior aos outros. O Shell Sort e o Quick Sort, ainda mantém valores normais e tempo gasto zero.

2 – VETOR COM 10000 ITENS:

* Ordem crescente



Tabela 4: Vetor com 10000 itens em ordem crescente.

Novamente, está clara a eficiência do Insertion Sort para vetores com ordenação crescente. Além de realizar o menor número de comparações, teve tempo gasto zero. Nessa situação o pior algoritmo foi o Selection Sort, pois realizou uma quantidade altíssima de comparações e teve um tempo gasto absurdamente superior aos outros. Outro fato interessante é a ineficiência do Quick Sort para vetores ordenados de forma crescente, pois realizou mais comparações que o Shell Sort e teve um tempo relativamente baixo, porém acima de zero.

* Ordem decrescente



Tabela 5: Vetor com 10000 itens em ordem decrescente.

O Insertion Sort foi o menos efetivo para vetores com itens em ordem decrescente, como previsto, tendo o número de trocas, comparações e tempo gasto superior aos outros. Logo atrás, vem o Selection Sort com um número muito alto de comparações e um tempo gasto bem próximo do Insertion Sort. Em casos de vetores de ordenação decrescente, o Shell Sort leva vantagem em cima do Quick Sort pois seu número de trocas é inferior, mesmo o tempo gasto ser quase igual.

* De forma randômica



Tabela 6: Vetor com 10000 itens de forma randômica.

Começamos a notar a superioridade do Insertion Sort comparado ao Selection Sort, mesmo realizando mais trocas, realiza muito menos comparações e o tempo gasto é inferior ao Selection Sort. A partir de uma quantidade maior de dados de forma randômica, começamos a perceber a superioridade do Quick Sort comparado ao Shell Sort, realizando menos trocas, mais comparações, porém com um tempo menor.

3 – VETOR COM 100000 ITENS:

* Ordem crescente



Tabela 7: Vetor com 100000 itens em ordem crescente.

Mais uma vez é posto em prova a ineficiência do Selection Sort e do Quick Sort para vetor com itens em ordem crescente, realizando alto número de comparações, entretanto o Quick Sort é abundantemente mais eficiente que o Selection Sort, a diferença de tempo gasto, prova isso. Um fato interessante é que mesmo o Quick Sort realizando muito mais comparações que o Shell, seu tempo gasto é inferior mais de 3ms.

* Ordem decrescente



Tabela 8: Vetor com 100000 itens em ordem decrescente.

O Quick Sort nessa situação, prova a sua maior eficiência entre esses algoritmos: Realizou menos trocas e comparações que todos exceto o Selection Sort, e seu tempo gasto foi muito baixo comparado a qualquer outro. Mais uma vez, o Selection Sort, mostra ser nada eficiente para quantidade de itens maiores e gasta muito tempo de execução.

* De forma randômica



Tabela 9: Vetor com 100000 itens de forma randômica.

Esses resultados refletem tudo aquilo que já foi dito, e deixa claro mais uma vez, que o Quick Sort é o melhor algoritmo para ordenação de vetor de itens não ordenados.

Conclusão