

ANALISE DE ALGORITMOS DE ORDENAÇÃO

PROJETO E ANALISE DE ALGORITMOS

PROFº. DRº. DANILO MEDEIROS ELER

BRUNO SANTOS DE LIMA

LEANDRO UNGARI CAYRES

PRESIDENTE PRUDENTE

SETEMBRO - 2015

1. **Introdução**

A sociedade busca utilizar de critérios para organizar seus instrumentos de trabalho, sejam estes uma lista telefônica, uma lista de alunos de uma universidade, dados de aplicações em bancos de dados, a Ciência da Computação como disciplina ao longo de sua história idealizou diversos algoritmos capazes de realizar ordenações em um determinado de conjuntos de dados.

Nesta coletânea será realizado uma estudo considerando os algoritmos de ordenação mais utilizados e conhecidos afim de analisar e comparar o desempenho de cada um desses métodos sobre um determinado conjunto de dados em especifico. Este trabalho encontra-se organizado como segue: na Seção 2 será apresentado os algoritmos de estudo bem como a análise assintótica de cada um deles, na Seção 3 é descrita a análise experimental, a Seção 4 descreve as considerações finais do experimento.

1. **Algoritmos e Análise Assintótica**
   1. **Bubble Sort**

Este algoritmo enquadra-se na categoria dos baseados em troca, permutando elementos vizinhos se necessário, de forma que os maiores vão para o fim do arranjo a cada iteração, de forma análoga, em que as bolhas mais densas descem. Nesta versão são executadas iterações, porém é possível que o vetor já esteja ordenado antes do termino destas repetições. Desta forma, existe uma versão com uma pequena melhoria, a criação de uma flag que verifica se houve uma troca na última iteração, em caso negativo, então o vetor já se encontra ordenado e as demais iterações são desnecessárias [3].

Complexidade:

Melhor caso: O () vetor ordenado crescente somente para a implementação com melhoria.

Caso médio: O ().

Pior caso: O () vetor ordenado decrescente e vetor aleatório (consiste também no melhor para a versão sem melhoria).

* 1. **Quick Sort**

Este algoritmo foi criado por C.A.R. Hoare em 1962, o qual é extremamente eficiente na grande maioria dos casos, porém pode se tornar ineficiente como os algoritmos quadráticos em casos específicos do arranjo e escolha do pivô. Este método possui diversas estratégias de escolha do pivô, dentre as quais destacamos o pivô na posição inicial final [1].

Para ambos os casos, o passo a passo consiste em escolher um pivô e depois encontrar um elemento maior que o pivô a partir do início do vetor e um menor a partir do fim. Para obtenção do melhor desempenho deste algoritmo seria necessário conhecer o elemento com valor médio, desta forma, através da estratégia de divisão e conquista, o vetor seria dividido em sub-regiões de tamanho uniforme [1].

Analogamente, os piores caso deste algoritmo consistem na escolha de pivô como elemento mínimo ou máximo, o que pode acontecer mais frequentemente para a implementação do pivô inicial, para casos de vetor crescente e decrescente [1].

Complexidade:

Melhor caso:

Caso médio:

Pior caso:

* 1. **Insertion Sort**

A ordenação por inserção é um dos mais simples algoritmos, sendo eficiente se aplicado a um pequeno conjunto de elementos, sua estratégia consiste em percorrer o vetor da esquerda para a direita deixando os elementos que estão à esquerda ordenados. Um exemplo de aplicação consiste na inserção de cartas na mão de um jogador [4][5].

Complexidade:

Melhor caso: O () vetor ordenado crescente

Caso médio: O

Pior caso: O () vetor ordenado decrescente e vetor aleatório.

* 1. **Shell Sort**

O algoritmo Shell Sort criado em 1959 por Donald Shell é um dos métodos de ordenação mais eficientes dentre os quadráticos, baseia-se no Insertion Sort, realizando comparações com elementos não adjacentes, desta forma, solucionando problemas como o menor elemento na posição mais à direita do vetor, o que era um empecilho com o uso de trocas adjacentes, considera-se vários segmentos menores que o tamanho do vetor e aplica-se ordenação, em que o número desses segmentos é determinado pelo chamado número de saltos, que por sua vez é reduzido sucessivamente até chegar a 1, assim a cada interação torna-se mais ordenado [5].

Neste método o número de saltos é decrementado, sendo calculado de diversas formas, sendo assim a análise assintótica deste algoritmo é complexa para ser estabelecida, porém, em geral adota-se os seguintes valores [5].

Complexidade: , sendo igual para todos os casos.

* 1. **Selection Sort**

Este método trata-se de um algoritmo baseado em seleção, cuja técnica baseia-se em percorrer o vetor encontrando o menor elemento e pô-lo na primeira posição, depois encontrar o segundo menor e colocá-lo na segunda posição, e assim sucessivamente.

Tal método realiza comparação entre todos os elementos, o que o torna mais ineficiente, porém é o de mais simples implementação e menor número de trocas, no máximo , sendo o número de elementos [5].

Complexidade: , invariável para todos os casos.

* 1. **Heap Sort**

Consiste em um algoritmo de ordenação extremamente sofisticado cuja implementação baseia-se em uma fila de prioridades utilizando a estrutura de dados Heap, o que consiste em uma árvore binária em vetor. Foi inventado por J.W.J. Williams em 1964. O algoritmo consiste em construir um *max heap*, em seguida trocar o primeiro elemento com o último, e por fim rearranjar o *max heap*, para todos os elementos exceto para o último, que já é o maior elemento do vetor [2].

Complexidade: para quaisquer casos.

* 1. **Merge Sort**

Consiste em um algoritmo que utiliza o princípio de divisão e conquista, que gera dois conjuntos de tamanho , sendo seu melhor caso quando nunca é necessário realizar trocas após comparações, pior caso quando sempre é necessário realizar trocas após comparações e caso médio quando nem sempre é necessário realizar trocas após comparações, entretanto sua complexidade é a mesma para todos os casos. Como ponto negativo, a complexidade de espaço é de , sendo o número de elementos do vetor.

Complexidade: para quaisquer casos.

1. **Análise Experimental**

Após a realização da análise assintótica, foi realizada a análise experimental dos mesmos algoritmos, tais testes envolveram diferentes tamanhos de entrada, as dimensões utilizadas para o vetor foram: 10, 100, 1 000, 10 000, 100 000 e 1 000 0000 de elementos, em todos os casos citados acima foram utilizados como teste um vetor crescente, decrescente e elementos dispersos pelo vetor de forma aleatória, ressaltando que para este último foi utilizado um arquivo que armazena os elementos, para que esta sequência aleatória seja sempre a mesma para execução em todos os algoritmos, para todos os testes o tempo é dado em milissegundos (*ms*).

Para a realização dos testes foi utilizado um computador com as seguintes especificações: processador Intel Core i3 - 3110M CPU @ 2.40 GHz, memória RAM 4 GB, memória caches L1 de 128 KB, L2 de 512 KB e L3 de 3 MB.

* 1. **Comparativos entre casos**
     1. **Selection Sort**

Através do gráfico acima, pode-se concluir que o algoritmo de ordenação Selection Sort, apesar de ser indiferente quanto a organização prévia dos elementos pertencentes ao arranjo e ter uma das mais simples implementações, torna-se pior a medida que o número de elementos cresce, cuja curva se assemelha às quadráticas.

* + 1. **Bubble Sort**

A análise experimental realizada com o método Bubble Sort em sua versão clássica revelou que independente do caso em que os elementos estão distribuídos no vetor, seja de modo aleatório, crescente ou decrescente o comportamento expresso no gráfico acima é semelhante, considerado de ordem quadrática, tendo para a organização inicial um vetor crescente seu desempenho foi levemente superior aos demais devido ao número reduzido de trocas entre elementos.

* + 1. **Bubble Sort Melhorado**

Os resultados da análise experimental revelaram que para o método de ordenação Bubble Sort com melhorias apresenta como seu melhor caso quando o vetor já está organizado de modo crescente tendo um desempenho linear, além disso tanto para conjuntos organizados de modo decrescente quanto para conjuntos organizados de forma aleatória temos um comportamento que se enquadra em seu pior caso com desempenho considerado quadrático, que é visivelmente observado em conjuntos de tamanho maior que cem mil.

* + 1. **Insertion Sort**

Conclui-se, por meio do gráfico anterior, que o Insertion Sort consiste em um excelente algoritmo para arranjos quase ordenados, aproximando de uma complexidade linear, o que pode ser observado pela curva do arranjo crescente, porém para os outros dois casos este torna-se próximo da complexidade quadrática, não consistindo na melhor escolha para tais situações.

* + 1. **Shell Sort**

O gráfico acima mostra uma representação do método de ordenação Shell Sort, onde este método apresenta para vetores com dimensões de dez até um milhão elementos para os três casos o desempenho apresenta poucas mudanças com relação ao seu tempo de execução, contudo quando o tamanho do conjunto ultrapassa a casa dos cem mil elementos tem-se que para a organização de forma aleatória temos um o pior desempenho se comparado com a organização crescente e decrescente dos elementos no vetor, quanto para esses dois últimos casos o melhor desempenho está quando o vetor já está ordenado.

* + 1. **Merge Sort**

Em suma, a sua implementação baseada em divisão e conquista, permite um remanejamento dos elementos quase indiferente para os casos de conjunto crescente, decrescente e aleatório, colocando como um dos melhores algoritmos, apesar de seu maior consumo de memória.

* + 1. **Quick Sort com pivô inicial**

O experimento demonstrou que o algoritmo Quick Sort com pivô inicial tem um bom desempenho para um conjunto organizado em ordem aleatória, contudo para conjuntos em ordem crescente e decrescente o teste resultou em um baixo desempenho aumentando de forma considerável o tempo de execução.

* + 1. **Quick Sort com pivô central**

Considerando a implementação do algoritmo Quick Sort utilizando um o pivô central é notório que nos casos em que o vetor está inicialmente ordenado de modo crescente e de modo decrescente o desempenho são semelhantes e melhores se comparados com um vetor com elementos dispostos aleatoriamente.

* + 1. **Heap Sort**

Por fim, o gráfico acima demonstra que o algoritmo Heap Sort mediante a uma implementação sofisticada promove resultados invariantes para todos os casos, sem o uso adicional de recursos.

* 1. **Comparação quanto a organização do vetor**

As analises a seguir realizam uma comparação entre todos os métodos de ordenação estudados para cada um dos casos de organização do vetor, sendo ele de ordenado de modo crescente, decrescente e aleatório.

* + 1. **Crescente**

Primeiramente podemos observar que as estratégias de ordenação dos algoritmos Bubble Sort Clássico, Quick Sort com pivô inicial e o Selection Sort apresentaram desempenhos inferiores aos demais porem semelhantes entre si. Aos demais algoritmos seus comportamentos se assemelham a algoritmos lineares sendo extremamente eficiente para este caso devido a presença de estratégias que reduzem o número de operações sobre os elementos, com destaque ao Insertion Sort, um algoritmo inicialmente quadrático, torna-se o mais eficiente entre os analisados.

* + 1. **Decrescente**

Analisando o gráfico acima é possível concluir que para agregados homogêneos decrescentes os métodos com melhores resultados consistem em Heap Sort, Merge Sort, Shell Sort e Quick Sort com pivô central, também é perceptível que as duas implementações do algoritmo Bubble Sort tiveram um desempenho consideravelmente inferior com relação aos demais métodos aplicados.

* + 1. **Aleatório**

O experimento revelou que para esta organização do conjunto os algoritmos Heap Sort, Merge Sort, Shell e Quick Sort para ambas implementações tiveram um desempenho considerado superior com relação aos demais. Os algoritmos de pior rendimento consistem nas duas implementações do Bubble Sort.

* 1. **Comparativos intra-categorias**

Todas as comparações abaixo foram realizadas utilizando os casos para conjunto de dados aleatório e também com as melhores implementações de cada algoritmo, de forma a não favorecer determinados algoritmos e consequentemente propor conclusões equivocadas.

* + 1. **Algoritmos baseados em troca**

Observando o gráfico acima que realiza uma comparação entre os dois casos de estudo para algoritmos de ordenação baseados em troca, é notório que para organização do arranjo com mais de cem mil elementos o algoritmo Bubble Sort com melhoria tem desempenho inferior ao Quick Sort com Pivô Central, visto que este último tem um desempenho semelhante para quase todas as dimensões com um pequeno acréscimo em seu tempo conforme o tamanho do conjunto é aumentado.

* + 1. **Algoritmos baseados em inserção**

Realizando uma comparação entre os algoritmos que são baseados em inserção podemos ressaltar que o tempo de execução entre eles permanecem semelhantes tendo vetores de tamanhos que podem assumir valores de dez até cem mil elementos, onde a partir desta última dimensão o método de ordenação Shell Sort passa a se sobressair mantendo uma linha um pouco menor se comparado com o método Insertion Sort onde para conjuntos de tamanho maiores que cem mil ocorre um crescimento levemente maior no tempo de execução deste algoritmo perdendo desempenho.

* + 1. **Algoritmos baseados em seleção**

Por meio do gráfico acima, é possível observar a grande diferença de tempo de execução entre algoritmos baseados em seleção; o Selection Sort, o qual utiliza a estratégia força bruta, em detrimento ao Heap Sort, um dos mais sofisticados algoritmos de ordenação, cuja curva de complexidade de tempo praticamente inexiste.

1. **Considerações finais**

Deste modo, os algoritmos de ordenação são de grande importância quando nos referimos a organização de dados e é importante analisá-los e compará-los afim de ter um entendimento de quais são os melhores e piores casos de cada método.

1. **Referências Bibliográficas**

[1] ZIVIANI, Nivio. Projeto de Algoritmos com implementações em Java e C++. São Paulo. Thomson Learning, 2007.

[2] FEOFILOFF, Paulo. Quicksort. Disponível em: <http://www.ime.usp.br/~pf/algoritmos/ aulas/quick.html>. Acesso em: 27 nov. 2015.

[3] FEOFILOFE. Paulo. Heapsort. Disponível em: <http://www.ime.usp.br/~pf/algoritmos /aulas/hpsrt.html>. Acesso em: 27 nov. 2015.

[4] ALMEIDA, Jussara. Ordenação pelo Método da Bolha Bubblesort. Disponível em: <http://www2.dcc.ufmg.br/disciplinas/aeds2\_turmaA1/bubblesort.pdf>. Acesso em: 27 nov. 2015.

[5] KONISHI, Mari. Métodos de Ordenação. Disponível em: <https://marikonishi.wordp ress.com/category/metodos-de-ordenacao/>. Acesso em: 27 nov. 2015.