RELATÓRIO – PBJL4 ALGORITMOS DE ORDENAÇÃO

Luan Estevinho, Erick Marlon, Vinicius Garcia

Sumário (+0,5 ponto na média)

Esse relatório apresenta uma análise objetiva do comportamento dos 3 algoritmos de ordenação passados em sala, dentre eles: Bubble Sort, Insertion Sort, Quick Sort. Para essa analise de desempenho foram utilizados 9 arquivos de listas numéricas em tamanhos e organizações diferentes. O objetivo foi comprar o tempo de execução e eficiência dos algoritmos em cada cenário, buscando identificar possíveis padrões e entender o porque cada um se comportou da maneira observada.

1. Estrutura

O Código foi desenvolvido em java e possui as classes:

- class ordenacao:

Essa é a classe responsável por armazenar os algoritmos de ordenação (Bubble, Insertion, Quick) além da estrutura e metodos adicionais que eu prefer usar para otimizar e facilitar a visualização do código. Dentre os metodos auxiliaries estão: método swap(), para troca de itens de posição, é amplamente util e utilizado nos algoritmos de Bubble Sort e Quick Sort. Fiz também o método mostrarLista(), usado para imprimir a lista ordenada com todos os elementos presentes. Por fim tem o método de inserir(), que como o próprio nome sugere, ele serve para inserir elementos na lista.

- class main:

Esta é responsável pela leitura dos arquivos, instanciação de objetos da classe ordenacao, e chama aos métodos principais da classe (Bubble, Insertion, Quick).

2. Explicação prévia dos algoritmos

Lembre-se que essas são explicações teoricas, portanto podem acabar sendo de certa forma confusas.

- Bubble Sort:

Esse algoritmo consiste principalmente na verificação por pares, ou seja, ele verifica um par de elementos da lista e confere se o da esquerda é maior que o da direita, e se for troca eles de posição. Essa estratégia usada pelo Bubble Sort, faz com que a cada passagem pela lista o maior element vá para o final, e por conta disso as iterações são feitas até que o Contador seja >= (tamanho – i).

- Insertion Sort:

Esse algoritmo basicamente aponta para um element alvo que sera ordenado, e vai fazendo validações de se o element anterior ao "atual" for maior que ele, empurra o elemento anterior para a direita e volta uma posição para poder inserir o elemento "atual".

- Quick Sort:

Esse algoritmo usa a técnica de "dividir para conquistar", ou seja, ele seleciona um elemento pivot que será usado como referência para definir elementos maiores e menores. Ele começa varrendo a lista do inicio e vai verificando se o elemento atual < pivot, se for menor que o pivot, troca o elemento atual analisado com o antecessor dele (o antecessor é guardado por um Contador do tipo inicio – 1, e vai incrementado o Contador nas execuções), até que chege no fim – 1 da lista, depois ele troca de lugar o pivot com a proxima posição depois da partição de menores gerada, que pode ser descoberta por meio do Contador (i + 1). Guarda o indice do pivot e chama recursivamente o metodo para a particao de menores e de maiores.

3. Resultados obtidos em ns

Arquivo	Bubble Sort (ns)	Insertion Sort (ns)	Quick Sort (ns)
aleatorio_100.csv	313700	70100	44100
aleatorio_1000.csv	6501300	2380900	471700
aleatorio_10000.csv	95253100	21172300	1244900
crescente_100.csv	9400	122800	50300
crescente_1000.csv	422200	57700	3392000
crescente_10000.csv	36676100	90400	85294500
decrescente_100.csv	11100	12600	34000
decrescente_1000.csv	1013200	893700	542900
decrescente_10000.csv	89437600	42188200	25478300

4. Analise dos resultados

aqui vamos entender por que cada algoritmo teve o desempenho mostrado na tabela:

- Arquivos Aleatórios:

Nos 3 conjuntos aleatorios (100, 1000 e 10000), o Quick Sorte teve de longe o melhor desempenho de todos, tendo em Segundo lugar o Insertion Sort, e por ultimo o Bubble Sort. Isso era bastante previsivel, pois o Quick Sort, por ter esse esquema de particionamento, faz com que ele acabe organizando muito mais elementos do que apenas um por iteração. Quando temos listas com posições aleatorias, a localização do pivot tende a ficar mais no “meio” da distribuição, gerando assim partições mais equilibradas e com partições mais equilibradas gera menos profundidade de recursão nas partições e menos comparações/trocas.

- Arquivos Aleatórios:

Nesse cenário aqui, o Bubble Sort acabou se destacando como o algoritmo mais eficiente, isso também era muito previsível por uma questão de lógica. Se a lista já está ordenada o algoritmo basicamente não faz trocas, ele so varre a lista e nunca entra na condição de o elemento da esquerda ser maior que o da direita. O Insertion Sort também tem um desempenho muito parecido nesse caso, pois ele só valida se um é maior que outro e como nunca é, ele não entra no while interno... O Quick Sort teve um desempenho tão terrivel, pois o pivot que eu escolhi foi o ultimo element da lista, isso faz a partição de menores ter 9999 elementos (no caso do arquivo de 10000), fazendo com que na proxima iteração, tenha 9998 e assim por Diante. Ele cai literalmente no seu PIOR caso.

- Arquivos Decrescentes:

Aqui é um cenário ruim para todos os algoritmos, o que teve seu destaque foi o Quick Sort, mas ainda assim levou um tempo pior que nos outros casos. Isso acontece pois, como a lista está invertida, todos os elementos que deveriam estar para a direita estão na esquerda fazendo com que o algoritmo tenha que trocar os pares varias vezes (Bubble Sort), empurrar o maior elemento para o final da fila e tendo que passar por todas as posições (Insertion Sort), ou como o pivot é o menor elemento, a partição de maiores fica cheia, sendo similar ao arquivo crescent (Quick Sort).

5. Conclusão

De forma geral, os testes confirmaram tudo aquilo que foi visto na teoria:

- Bubble Sort é o mais simples e também o mais lento, principalmente em listas grandes.
- Insertion Sort é eficiente em listas pequenas e principalmente quando os dados já estao mais ou menos ordenados.
- Quick Sort é o mais eficiente na maioria dos casos, mas pode cair no pior caso se escolher um pivô ruim (como eu que escolhi o ultimo elemento, nessas listas ordenadas isso é péssimo).

Em resumo, os resultados estão bastante condizentes com o comportamento esperado dos algoritmos. O Quick Sort foi o mais rapido no geral, o Insertion Sort teve um ótimo desempenho nas listas pequenas ou quase ordenadas, e o Bubble Sort teve o pior desempenho, mas ainda assim ele consegue cumprir a sua função.

Obrigado!

*Obs: não usei IA, por isso não tem nada referênciado aqui. Tamo Junto, Boas Férias.