Professor
Marcio Feitosa

CURSO: Estruturas de Dados

ALGORITMOS DE ORDENAÇÃO

-- TEORIA --

Introdução

O objetivo de um algoritmo de ordenação é dispor em ordem, seja numérica, alfanumérica, etc., um conjunto de elementos de mesma natureza.

Exemplo

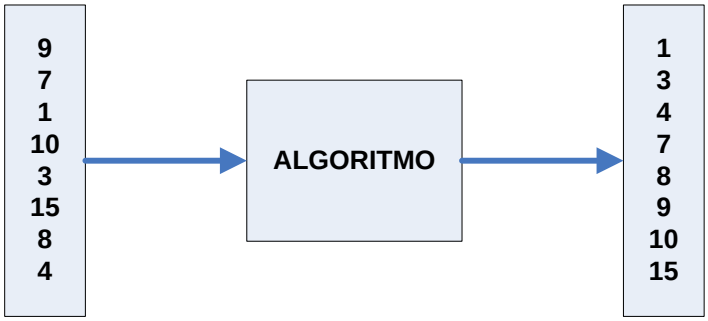

Figura 1

Essa ordenação pode ser em ordem crescente ou decrescente, a depender da necessidade do algoritmo.

Existem diversos algoritmos de ordenação. Aqui veremos os mais utilizados. Dentre os algoritmos temos aqueles mais eficientes, ou seja, dada uma quantidade de elementos a organizar, despendem menos esforço computacional, enquanto que outros já apresentam um custo mais elevado, consequentemente consumindo maiores tempos para concluir o trabalho.

Para pequenas quantidades de elementos (p.ex. 100) o tempo de processamento, para um usuário humano, é praticamente imperceptível, mas quantidades de 100.000 já podem inviabilizar o uso de algoritmos menos eficientes.

Ora, por que existem, então, esses menos eficientes? Não poderiam ser aposentados e se esquecer dele?

Ocorre, porém, que esses menos eficientes são muito mais simples de se implementar e para pequenas quantidades de elementos, em alguns casos, podem até ser mais eficientes. Então eles ainda têm o seu lugar no mundo dos ordenadores.

Vamos começar com os mais simples de compreender e implementar.

Bubble Sort

Esse é o mais conhecido dos algoritmos de ordenação, mas também dos menos eficientes. É de complexidade $O(n^2)$ no pior caso, ou seja, o tempo de processamento aumenta com o quadrado da quantidade de elementos do conjunto a ser ordenado. A complexidade chega a $O(n)$ para o caso em que o conjunto, ao ser submetido ao algoritmo, já se encontra totalmente ordenado[1].

A estratégia utilizada é simples (supondo-se ordenação no sentido menor → maior):

1. Carregam-se os dados em um vetor.
2. Corre-se o vetor o início ao fim verificando se o elemento seguinte é maior que o elemento atual.
    a. Se for maior, nada faz.
    b. Caso contrário trocam-se os dois de posição (swap).
3. Chegado ao final do vetor:
    a. Ocorreu pelo menos uma troca → retorna ao passo 2.
    b. Não ocorreu nenhuma troca → conjunto está ordenado → FIM

Insertion Sort

A forma como este algoritmo organiza os dados se assemelha à organização de cartas de um baralho logo após serem recebidas.

1. Carregam-se os dados em um vetor.
2. Corre-se o vetor do início ao fim verificando se o próximo elemento é maior que o atual.
    a. Se maior, não faz nada.
    b. Caso contrário, volta-se no vetor procurando a sua posição no grupo de elemento que ficou acima; deslocam-se todos uma posição abaixo (ou já se vai deslocando à medida em que se sobe) e deposita-se o elemento na sua posição.
3. Continua-se o curso do vetor na posição do último elemento.
4. Chegando ao final do vetor → conjunto está ordenado → FIM.

[1] Na verdade a notação **O** de complexidade se baseia sempre no pior caso. O melhor caso é analisado na chamada notação **Ω** (ômega). O caso médio é tratado pela notação **Θ** (teta).

Da mesma forma que o Bubble Sort, a complexidade também é $O(n^2)$.

Selection Sort

Outra alternativa, com conceito semelhante ao do algoritmo Insertion Sort, em que se evolui a ordenação na parte superior do vetor, se baseia na ideia de procurar o menor valor na parte não ordenada (inferior) e fazer o *swap* com o primeiro dessa parte, ampliando, assim, a parte ordenada em um elemento e, consequentemente, reduzindo a parte não ordenada.

Embora pareça mais eficiente que o Bubble, pois não precisa ficar correndo o vetor do início ao fim em cada iteração, mas sim do ponto inicial do espaço não ordenado, também é considerado de complexidade $O(n^2)$.

Vamos ao código!!

-o-o-o-o-o-o-o-o-o-o-