Implementação Algorítmica

Análise do Insertion Sort

Carlos H. A. Higa

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

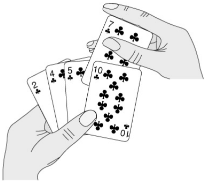
FACOM-UFMS

UFMS

Insertion Sort

Ordenação
Entrada: Uma sequência de n números $\langle a_1, a_2, \ldots, a_n \rangle$.
Saída: Uma permutação $\langle a'_1, a'_2, \ldots, a'_n \rangle$ da sequência de entrada tal que $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

O algoritmo *insertion sort* (ordenação por inserção) é eficiente para ordenar um número pequeno de elementos.



Insertion Sort

INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1..j-1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i+1] = A[i]
        i = i - 1
    A[i+1] = key
```

Insertion Sort

O algoritmo está correto?

Invariante

- No início de cada iteração do laço **for** das linhas 1 - 8, o sub-vetor $A[1\,..\,j-1]$ consiste dos elementos originais em $A[1\,..\,j-1]$, mas de maneira ordenada.

Este invariante é verdadeiro antes, durante e após a execução do algoritmo.

Insertion Sort

Análise do Insertion Sort

- Geralmente, o tempo de execução de um algoritmo cresce de acordo com o tamanho da entrada;
- O tamanho da entrada depende do problema. Pode ser *o número de itens da entrada*, por exemplo, o tamanho n de um vetor. Em outros problemas, como a multiplicação de dois inteiros, a melhor medida para o tamanho da entrada é o *número total de bits* para representar a entrada na notação binária;
- O tempo de execução de um algoritmo em uma entrada em particular é o número de passos executados.

Insertion Sort

Quanto tempo leva para executar o INSERTION-SORT em uma entrada de tamanho n?

Uma quantidade constante de tempo é necessária para executar cada linha do código.

Uma linha pode levar uma quantidade de tempo diferente da outra; assumimos que a execução da i-ésima linha leva tempo c_i, onde c_i é uma constante.

Também denotaremos por t_j o número de vezes que o teste laço **while** na linha 5 é executado para um dado valor de j.

Insertion Sort

Linha	Custo	Vezes
1	c_1	n
2	c_2	$n-1$
3	0	$n-1$
4	c_4	$n-1$
5	c_5	$\sum_{j=2}^{n} t_j$
6	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	c_8	$n-1$

FACOM-UFMS

Insertion Sort

O custo de executar o INSERTION-SORT é

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\ &\quad + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) \end{aligned}$$

Insertion Sort

O melhor caso acontece quando o vetor de entrada já está ordenado.

Na linha 5, a condição $A[i] > key$ é sempre falsa, e as linhas 6 e 7 nunca são executadas.

Assim,

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

que pode ser expresso por $an + b$, para constantes a e b que dependem dos custos c_i. Assim, $T(n)$ é linear em n.

Insertion Sort

O pior caso ocorre quando o vetor de entrada está ordenado em ordem decrescente.

Devemos comparar $A[j]$ com cada elemento em $A[1\,..\,j-1]$.

Assim, $t_j = j$ para $j = 2, 3, \ldots, n$.

Insertion Sort

O pior caso ocorre quando o vetor de entrada está ordenado em ordem decrescente.

Devemos comparar $A[j]$ com cada elemento em $A[1\,..\,j-1]$.

Assim, $t_j = j$ para $j = 2, 3, \ldots, n$.

Note que

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

e

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

Insertion Sort

No pior caso,

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 \\ &\quad + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

$T(n)$ pode ser expresso como $an^2 + bn + c$ para constantes a, b e c que dependem de c_i. Assim, $T(n)$ é quadrática em n.

FACOM-UFMS