

Projeto e Análise de Algoritmos (2025-1)

Flávio L. C. de Moura*

3 de abril de 2025

Ordenação por inserção

Nesta seção estudaremos o algoritmo *insertion sort* (ordenação por inserção), que tem como etapa principal inserir um elemento em um vetor ordenado. Assim, queremos ordenar $n > 0$ números naturais em ordem crescente, e para isto vamos supor que estes números estão armazenados no vetor $A[0..n-1]$. Ao final do processo queremos obter uma permutação de $A[0..n-1]$, digamos $A'[0..n-1]$ tal que $A'[i-1] \leq A'[i]$, para todo $1 \leq i < n$. O algoritmo de ordenação por inserção (*insertion sort*) é dado pelo pseudocódigo a seguir:

Algorithm 1: InsertionSort($A[0..n-1]$)

```
1 for  $j = 1$  to  $n - 1$  do
2    $key \leftarrow A[j]$ ;
3    $i \leftarrow j - 1$ ;
4   while  $i \geq 0$  and  $A[i] > key$  do
5      $A[i+1] \leftarrow A[i]$ ;
6      $i \leftarrow i - 1$ ;
7   end
8    $A[i+1] \leftarrow key$ ;
9 end
```

A primeira pergunta que precisamos responder é: este algoritmo é correto? Isto é, ele satisfaz as especificações do problema que propõe resolver?

A correção do algoritmo de ordenação por inserção

Em algoritmos iterativos utilizamos as invariantes de laço para estabelecer a correção do algoritmo. Dada a dinâmica do algoritmo InsertionSort, considere a seguinte invariante de laço:

Antes da j -ésima iteração do laço **for** (linhas 1-9), o subvetor $A[0..j-1]$ está ordenado e contém

*flaviomoura@unb.br

os mesmos elementos do vetor original $A[0..j-1]$.

Assim, se esta propriedade for válida ao final da execução do laço **for**, *i.e.* antes da $n+1$ -ésima iteração, teremos que o vetor gerado consiste dos elementos do vetor original $A[0..n-1]$ ordenado. Isto corresponde a dizer que InsertionSort é correto.

Como então provar esta invariante para InsertionSort? A prova é por indução no número de iterações do laço **for**:

- **Inicialização** (Base da indução):

Antes da primeira iteração do laço **for**, temos que $j = 1$ (condição necessária para iniciar o laço), e portanto a invariante é trivial porque o subvetor unitário $A[0]$ está ordenado por definição.

- **Manutenção** (Passo indutivo):

Considere a k -ésima iteração, isto é, $j = k$ ($1 < k < n$). Temos como hipótese que "Antes da k -ésima iteração do laço **for** o subvetor $A[0..k-1]$ é uma permutação que está ordenada do subvetor original $A[0..k-1]$." Assim, durante a k -ésima iteração, o laço **while** vai deslocar cada elemento maior do que $A[k]$, *i.e.* *key*, uma posição para a direita até encontrar a posição correta onde o elemento $A[k]$ deve ser inserido, de forma que neste momento o subvetor $A[0..k]$ está ordenado e possui os mesmos elementos do subvetor $A[0..k]$ original. A incrementarmos o valor de k para a próxima iteração, a invariante é reestabelecida. Informalmente estamos dizendo que o laço **while** encontra a posição correta para inserir $A[j]$ (que está armazenado na variável *key*). Provaremos este fato com a ajuda de uma invariante para o laço **while**:

Antes de cada iteração do laço **while**, o subvetor $A[i+1..j]$ possui elementos que são maiores ou iguais a *key*.

A prova é também por indução no número de iterações do laço **while**:

1. **Inicialização**: Antes da primeira iteração do **while** temos que $i+1 = j = k$, e como $key = A[j]$ a invariante está satisfeita.
2. **Manutenção**: Por hipótese de indução temos que o subvetor $A[i+1..j]$ possui elementos que são maiores ou iguais a *key*. Durante uma iteração do laço, o elemento $A[i]$ é copiado na posição $i+1$ do vetor A , e portanto a invariante continua valendo.
3. **Finalização**: Ao final da execução do laço, temos que i é, de fato, a posição correta para inserir o elemento $A[k]$ já que todos os elementos do subvetor $A[i+1..j]$ são maiores ou iguais a *key*. É importante observar que a inserção do elemento $A[k]$ na posição i não elimina nenhum elemento do vetor original porque o elemento que está na posição i foi

copiado para a posição $i + 1$, se o laço **while** foi executado pelo menos uma vez, ou ele é o próprio elemento armazenado em *key*, quando o laço não é executado.

- **Finalização:** Ao final da execução do laço **for**, temos $j = n$, e portanto a invariante corresponde a dizer que o vetor $A[0..n - 1]$ obtido ao final da execução do algoritmo está ordenado, e é uma permutação do vetor original $A[0..n - 1]$. Assim, concluímos a prova da correção do algoritmo InsertionSort.

Exercício 0.1. Prove que o algoritmo BubbleSort a seguir é correto.

Algorithm 2: BubbleSort($A[0..n - 1]$)

```
1 for  $i = 0$  to  $n - 2$  do
2   for  $j = 0$  to  $n - 2 - i$  do
3     if  $A[j + 1] < A[j]$  then
4       swap  $A[j]$  and  $A[j + 1]$ ;
5     end
6   end
7 end
```

Exercício 0.2. Prove que o algoritmo BubbleSort2 [1] a seguir é correto.

Algorithm 3: BubbleSort2($A[0..n - 1]$)

```
1 for  $i = 0$  to  $n - 2$  do
2   for  $j = n - 1$  downto  $i + 1$  do
3     if  $A[j] < A[j - 1]$  then
4       swap  $A[j]$  and  $A[j - 1]$ ;
5     end
6   end
7 end
```

Referências

- [1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 4 edition, April 2022.