# Capítulo 1

# Ordenação por inserção recursivo

Nesta seção estudaremos o algoritmo de ordenação por inserção recursivo. A estrutura de dados utilizada é a de listas, e para simplificar trabalharemos com números naturais, mas as ideias são as mesmas para ordenarmos qualquer estrutura que possua uma ordem total. Vimos no capítulo anterior que as listas de naturais possuem dois construtores: $nil$ para representar a lista vazia, e o operador :: que nos permite construir uma nova lista a partir de um número natural e de uma lista já construída. Assim, a lista unitária contendo apenas o natural 5 é representada por $5 :: nil$, enquanto que a lista $1 :: (5 :: nil)$, ou simplesmente $1 :: 5 :: nil$, representa a lista que tem 1 como primeiro elemento, e a lista $5 :: nil$ como cauda.

A operação que dá nome ao algoritmo é a operação de inserção porque a cada passo queremos inserir um novo elemento em uma lista já ordenada. Suponha, por exemplo, que queiramos inserir o número 3 na lista $1 :: 5 :: nil$, isto é, o nosso objetivo final é obter a lista ordenada $1 :: 3 :: 5 :: nil$. Para isto, precisamos inicialmente comparar o 3 com o primeiro elemento da lista, e o resultado desta comparação nos diz que o 3 deve ser inserido depois do 1, ou seja, em algum lugar da cauda $5 :: nil$. Em seguida, comparamos o 3 com o primeiro elemento da cauda, ou seja, com 5, e como $3 < 5$, sabemos que ele deve ser inserido antes do 5. Esta ideia está implementada na função $ins$ definida a seguir:

**Definição 1.** *Sejam $x$ um número natural, e $l$ uma lista de números naturais. A função $(ins\ x\ l)$, que insere o natural $x$ na lista $l$, é definida recursivamente como a seguir:*

$$ins\ x\ l = \begin{cases} x :: nil, & se\ l = nil \\ x :: l, & se\ l = h :: tl\ e\ x \leq h \\ h :: (ins\ x\ tl), & se\ l = h :: tl\ e\ x > h \end{cases}$$

O algoritmo de ordenação por inserção então consiste em recursivamente, dada uma lista não vazia $h :: tl$, inserir o primeiro elemento $h$ na versão ordenada da cauda $tl$. Ou seja, o algoritmo de ordenação por inserção que será implementado pela função $is$ (*insertion sort*) que recebe uma lista $l$ como argumento. Se $l$ for a lista vazia não há nada a fazer, e caso contrário, recursivamente ordenamos a cauda da lista para então inserir o novo elemento:

**Definição 2.** *Seja $l$ uma lista de números naturais. A função $is$ é definida recursivamente como a seguir:*

$$is\ l = \begin{cases} nil, & se\ l = nil \\ ins\ h\ (is\ tl), & se\ l = h :: tl \end{cases}$$

Vejamos como este algoritmo funciona na prática. Suponha que queiramos ordenar a lista $3 :: 2 :: 1 :: nil$. Ao fornecermos esta lista como argumento para a função $is$, temos:

$is\ (3 :: 2 :: 1 :: nil) =$ (def. $is$)
$ins\ 3\ (is\ (2 :: 1 :: nil)) =$ (def. $is$)
$ins\ 3\ (ins\ 2\ (is\ (1 :: nil))) =$ (def. $is$)
$ins\ 3\ (ins\ 2\ (ins\ 1\ (is\ nil))) =$ (def. $is$)
$ins\ 3\ (ins\ 2\ (ins\ 1\ nil)) =$ (def. $ins$)
$ins\ 3\ (ins\ 2\ (1 :: nil)) =$ (def. $ins$)
$ins\ 3\ (1 :: (ins\ 2\ nil)) =$ (def. $ins$)
$ins\ 3\ (1 :: 2 :: nil) =$ (def. $ins$)
$1 :: (ins\ 3\ (2 :: nil)) =$ (def. $ins$)
$1 :: 2 :: (ins\ 3\ nil) =$ (def. $ins$)
$1 :: 2 :: 3 :: nil$

Veja que o algoritmo ordenou corretamente a lista $3 :: 2 :: 1 :: nil$, mas será que ele ordena corretamente qualquer lista de números naturais? Para responder esta pergunta, vamos analisar se o algoritmo é correto ou não.

## 1.1 A complexidade do algoritmo de ordenação por inserção

Em algoritmos de ordenação sobre listas, o tamanho da entrada consiste no tamanho da lista a ser ordenada. Vamos iniciar nossa análise com a função $ins\ x\ l$ (Definição 1). A operação básica no caso do algoritmo de ordenação por inserção é a comparação entre chaves. Note que quando $l$ é a lista vazia, a lista unitária $x :: nil$ é retornada, e nenhuma comparação é feita. Quando $l$ é uma lista da forma $h :: tl$ então comparamos $x$ com $h$, e quando $x \leq h$ a lista $x :: h :: tl$ é retornada e o algoritmo termina. Por outro lado, se $x > h$, o algoritmo continua buscando recursivamente a posição correta para inserir $x$. Denotaremos por $T_{ins}\ x\ l$ a função que computa o número de operações básicas realizadas pela função $ins$ para inserir o elemento $x$ na lista $l$:

$$T_{ins}\ x\ l = \begin{cases} 0, & \text{se } l = nil \\ 1, & \text{se } l = h :: tl \text{ e } x \leq h \\ 1 + (T_{ins}\ x\ tl), & \text{se } l = h :: tl \text{ e } x > h \end{cases}$$

Normalmente estamos interessados na análise do pior caso, e a função acima não computa necessariamente o número de comparações do pior caso para inserir um elemento qualquer em uma lista com $n$ elementos. De fato, considerando $n = 3$, temos que $T_{ins}\ 1\ (2 :: 3 :: 4 :: nil) = 1$, enquanto que $T_{ins}\ 4\ (1 :: 2 :: 3 :: nil) = 3$. Assim, para inserirmos um elemento em uma lista com $n$ elementos, no pior caso, precisamos comparar o elemento a ser inserido com todos os elementos da lista. A função $T^w_{ins}(n)$ modela esta situação ao receber o natural $n$ como argumento (que corresponde ao tamanho da lista a ser ordenada) e faz o número máximo de comparações possíveis:

$$T^w_{ins}(n) = \begin{cases} 0, & \text{se } n = 0 \\ 1 + T^w_{ins}(n-1), & \text{se } n > 0 \end{cases}$$

Assim, se a lista não possui elementos, nenhuma comparação é feita ($n = 0$), e caso contrário, uma comparação é feita para cada elemento da lista ($n > 0$). Observe que o número de comparações feitas pela função $T^w_{ins}(n)$ não pode ser maior do que o tamanho da lista:

**Exercício 1.** *Prove que $T^w_{ins}(n) = n$, para todo $n \geq 0$.*

Assim, a relação entre as funções $T_{ins}$ e $T^w_{ins}$ é dada pelo lema a seguir:

**Exercício 2.** *Sejam $x$ um número natural, e $l$ uma lista de números naturais. Prove que $T_{ins}\ x\ l \leq T^w_{ins}(|l|)$, onde $|l|$ denota o tamanho da lista $l$.*

Os dois últimos exercícios nos permitem concluir que $T_{ins}\ x\ l \leq |l|$, ou seja, que a função $ins$ tem complexidade linear.

### 1.1.1 A complexidade do algoritmo $is$

Qual é o número de comparações realizadas pelo algoritmo de ordenação por inserção, isto é, pela função $is$, para ordenar uma lista $l$? Vamos denotar por $T_{is}()$ a função que faz esta contagem. Se $l$ for a lista vazia então nenhuma comparação é feita, ou seja, $T_{is}(nil) = 0$. Se $l = h :: tl$ então é feita uma chamada à função $ins$, além da chamada recursiva à função $is$:

$$T_{is}(l) = \begin{cases} 0, & \text{se } l = nil \\ T_{is}(tl) + T_{ins}\ h\ (is\ tl), & \text{se } l = h :: tl \end{cases}$$

Observe que, $T_{is}(1 :: 2 :: 3 :: nil) = 2$, $T_{is}(3 :: 2 :: 1 :: nil) = 3$, $T_{is}(1 :: 2 :: 3 :: 4 :: nil) = 3$ e $T_{is}(4 :: 3 :: 2 :: 1 :: nil) = 6$, etc. Portanto o número de comparações pode ser diferente para listas de mesmo tamanho, o que é esperado pelas chamadas feitas à função $ins$. Como então definir a função $T^w_{is}(n)$ que nos dá um limite superior para o número de comparações feitas pelo algoritmo de ordenação por inserção para uma lista qualquer de tamanho $n$? Em outras palavras, qual a complexidade do pior caso para o algoritmo de ordenação por inserção? Sabemos que quando $n = 0$, nenhuma comparação é feita. Quando $n > 0$, o algoritmo é aplicado recursivamente na cauda da lista, isto é, em uma lista de tamanho $n - 1$, e é feita uma chamada à função $ins$ cuja complexidade já conhecemos. Isto nos permite escrever a função $T^w_{is}(n)$ como a seguir:

$$T^w_{is}(n) = \begin{cases} 0, & \text{se } n = 0 \\ T^w_{is}(n-1) + T^w_{ins}(n-1), & \text{se } n > 0 \end{cases}$$

que pode ser simplificada como a seguir, já que $T^w_{ins}(n) = n$:

$$T^w_{is}(n) = \begin{cases} 0, & \text{se } n = 0 \\ T^w_{is}(n-1) + (n-1), & \text{se } n > 0 \end{cases}$$

Podemos usar o método da substituição para encontrarmos uma solução para esta recorrência, e em seguida utilizar indução para verificarmos se a solução está correta. Pelo método da substituição, podemos ir aplicando a definição da recorrência, assumindo que $n > 0$:

$$\begin{aligned} T^w_{is}(n) &= T^w_{is}(n-1) + (n-1) \\ &= T^w_{is}(n-2) + (n-2) + (n-1) \\ &= T^w_{is}(n-3) + (n-3) + (n-2) + (n-1) \\ &= \ldots \end{aligned}$$

Podemos continuar este processo de substituição até chegarmos em $T^w_{is}(1)$ que é igual a 0:

$$\begin{aligned} T^w_{is}(n) &= T^w_{is}(n-1) + (n-1) \\ &= T^w_{is}(n-2) + (n-2) + (n-1) \\ &= T^w_{is}(n-3) + (n-3) + (n-2) + (n-1) \\ &= \ldots \\ &= T^w_{is}(1) + 1 + 2 + \ldots + (n-3) + (n-2) + (n-1) \\ &= 0 + 1 + 2 + \ldots + (n-3) + (n-2) + (n-1) \\ &= \textstyle\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \end{aligned}$$

Para finalizar, precisamos utilizar indução em $n$ para provar que $T^w_{is}(n) = \frac{n(n-1)}{2}$. Se $n = 0$, o resultado é trivial. Se $n > 0$ então, por definição, $T^w_{is}(n) = T^w_{is}(n-1) + (n-1)$. A hipótese de indução, nos dá que $T^w_{is}(n-1) = \frac{(n-1)(n-2)}{2}$, e portanto, $T^w_{is}(n) = T^w_{is}(n-1) + (n-1) \stackrel{h.i.}{=} \frac{(n-1)(n-2)}{2} + (n-1) = \frac{n(n-1)}{2}$.

Nossa conclusão, portanto, é que o algoritmo de ordenação por inserção recursivo é correto, e sua complexidade no pior caso é quadrática, assim como na versão não-recursiva.

**Exercício 3.** *Resolva as seguintes relações de recorrência:*

1. $T(1) = 1$, $T(n) = 2T(n-1) + 1$, $n \geq 2$

2. $T(1) \in \Theta(1)$, $T(n) = T(n-1) + 1/n$

3. $T(1) \in \Theta(1)$, $T(n) = T(n-1) + ln(n)$