Teorema 3.23. $\lg n = o(n^{\alpha}), \forall \alpha > 0$. *Ou seja, a função logaritmo cresce mais lentamente do que qualquer potência de n (incluindo potências fracionárias)*

Teorema 3.24. $n^k = o(2^n), \forall k > 0$. *Ou seja, potências de n crescem mais lentamente que a função exponencial 2^n. Mais ainda, potências de n crescem mais lentamente do que qualquer função exponencial $c^n, c > 1$.*

Exercício 3.25. *Mostre que* $\frac{n^2}{2} - 3n = \Theta(n^2)$.

Exercício 3.26. *Mostre que* $6n^3 \neq \Theta(n^2)$.

Exercício 3.27. *Sejam $f(n)$, $g(n)$ e $h(n)$ funções não-negativas tais que $f(n) = O(h(n))$ e $g(n) = O(h(n))$. Prove que $f(n) + g(n) = O(h(n))$.*

3.4 A complexidade de algoritmos recursivos: busca sequencial

Considere novamente o pseudocódigo da busca sequencial recursiva:

$$seq\_search\ x\ l := \begin{cases} FALSE, & \text{se } l = nil; \\ TRUE, & \text{se } l = h :: l' \text{ e } h = x; \\ seq\_search\ x\ l', & \text{se } l = h :: l' \text{ e } h \neq x. \end{cases}$$

Denote por $T_{ss}(x, l)$ o número de comparações realizadas pelo algoritmo *seq_search* x l, que pode ser definida como a seguir:

$$T_{ss}(x, l) := \begin{cases} 0, & \text{se } l = nil; \\ 1, & \text{se } l = h :: l' \text{ e } h = x; \\ 1 + T_{ss}(x, l'), & \text{se } l = h :: l' \text{ e } h \neq x. \end{cases}$$

Assim, se l é a lista vazia, nenhuma comparação é feita. Se l é uma lista não-vazia, digamos $h :: l'$, e x é igual a h (primeiro elemento da lista) então apenas 1 comparação é feita e o algoritmo para. Caso x não seja igual a h então recursivamente continuamos contando o número de comparações. Por exemplo, $T_{ss}(1, 1 :: 3 :: 5 :: nil) = 1$, $T_{ss}(2, 1 :: 3 :: 5 :: nil) = 3$, $T_{ss}(3, 1 :: 3 :: 5 :: nil) = 2$, etc. Ou seja, a função $T_{ss}(x, l)$ retorna o número exato de comparações realizadas pelo algoritmo *seq_search* durante a busca do elemento x na lista l.

Para fazermos a análise assintótica do algoritmo *seq_search* no pior caso, construiremos uma recorrência análoga à função T_{ss} que utiliza o tamanho da lista l como parâmetro.

$$T^w_{ss}(|l|) := \begin{cases} 0, & \text{se } l = nil; \\ 1 + T^w_{ss}(|l'|), & \text{se } l = h :: l'. \end{cases}$$

onde $|l|$ denota o número de elementos da lista l.

Assim, a função $T^w_{ss}(n)$ vai retornar o número de comparações necessárias para realizar a busca em uma lista com n elementos no pior caso:

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

Esta recorrência pode ser resolvida utilizando o *método da substituição* que consiste na aplicação sucessiva da definição da recorrência até que sejamos capazes de inferir uma solução. A verificação da correção da solução pode ser feita por indução, como veremos a seguir. Assumindo que $n > 0$, temos

$$\begin{aligned} T^w_{ss}(n) &= T^w_{ss}(n-1) + 1 \\ &= T^w_{ss}(n-2) + 2 \\ &= T^w_{ss}(n-3) + 3 \\ &= \ldots \\ &= T^w_{ss}(n-n) + n = 0 + n = n. \end{aligned}$$

Logo, $T^w_{ss}(n) = n$. Agora, podemos utilizar indução para verificar que esta é, de fato, uma solução da recorrência. A base da indução é trivial porque quando $n = 0$ temos $T^w_{ss}(0) = 0$. Quando $n > 0$, temos que $T^w_{ss}(n) \stackrel{def.}{=} 1 + T^w_{ss}(n-1) \stackrel{h.i.}{=} 1 + (n-1) = n$ como queríamos mostrar. Em notação assintótica, acabamos de mostrar que $T^w_{ss}(n) = \Theta(n)$.

3.5 A complexidade de algoritmos recursivos: ordenação por inserção

Considere o pseudocódigo do algoritmo de ordenação por inserção recursivo:

$is\ nil = nil$

$is\ (h :: tl) = insert\ h\ (is\ tl)$

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

onde

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

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 $n > 0$:

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

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, $T^w_{is}(n) = T^w_{is}(n-1) + (n-1) \overset{h.i.}{=} \frac{(n-1)(n-2)}{2} + (n-1) = \frac{n(n-1)}{2}$.

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

3.6 Mergesort

Algoritmos recursivos desempenham um papel fundamental em Computação. O algoritmo de ordenação *mergesort* é um exemplo de algoritmo recursivo, que se caracteriza por dividir o problema original em subproblemas que, por sua vez, são resolvidos recursivamente. As soluções dos subproblema são então