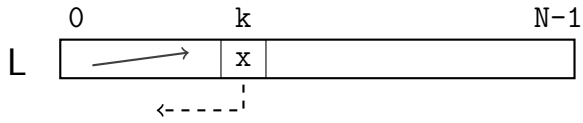
O nosso próximo algoritmo vai ordenar a lista do começo para o fim.

Quer dizer, imagine que a porção `L[0..k-1]` da lista já está ordenada.

E imagine que nós queremos inserir o elemento da posição `k` no lugar correto nessa ordem



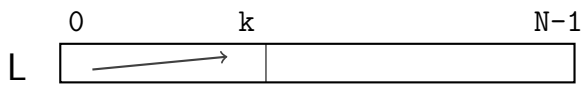
Como é que a gente faz isso?

Bom, uma ideia simples consiste em

→ *Trocar esse elemento de lugar com o elemento anterior*

enquanto ele for menor do que o anterior

Depois que a gente faz isso, a porção `L[0..k]` da lista fica ordenada



E se a gente continuar fazendo isso, a lista eventualmente fica completamente ordenada.

Legal.

Para implementar essa ideia, o primeiro passo é escrever a função `Inserção()`

```
func  Inserção ( L, k )
 {
    i  <--  k

    Enquanto ( i > 0  &&  L[i] < L[i-1] )
     {
         Troca ( L, i, i-1 )
         i--
 }   }
```

E daí, basta botar esse negócio para repetir

```
Para  k <-- 2  Até  N
 {
    Inserção ( L, k )
 }
```

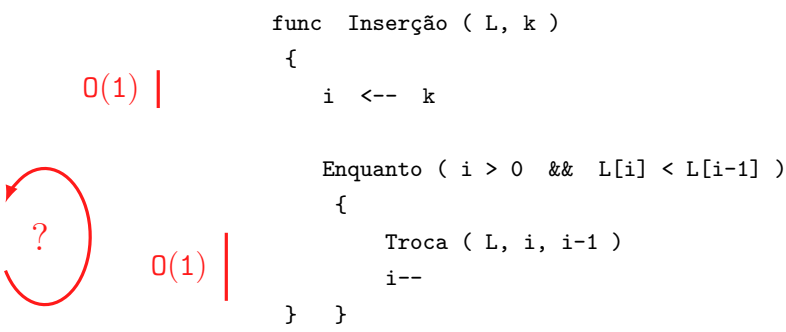
Esse algoritmo é conhecido como a *ordenação por inserção*.

E agora, nós vamos analisar o seu tempo de execução.

Análise

Para fazer isso, o primeiro passo é analisar a função `Inserção()`.

E para isso, nós anotamos o seu código da seguinte maneira



A observação fácil aqui é que, no pior caso, o laço dá $O(n)$ voltas — (*porque?*)

Daí que, no pior caso, a função executa em tempo

$$O(1) \quad + \quad O(n) \cdot O(1) \qquad = \qquad O(n)$$

Certo.

Daí, sabendo disso, a gente anota o código principal assim

Daí que, o tempo de execução da ordenação por inserção é

$$O(n) \cdot O(n) \qquad = \qquad O(n^2)$$