Considere novamente a tarefa

→ *Encontrar o maior elemento de uma lista*

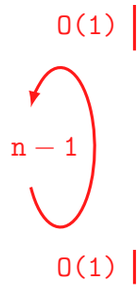
Na primeira aula nós apresentamos o seguinte algoritmo para esse problema

```
M  <--  L[1];      Mpos  <--  1
Para  i <-- 2  Até  N
 {
    Se ( L[i] > M )
     {
        M  <--  L[i];     Mpos  <--  i
 }   }
Imprime ("O maior elemento é:", M)
```

Daí, examinando o código nós fazemos duas observações simples

- o laço sempre dá $n-1$ voltas
- antes e depois do laço existem instruções que só são executadas uma vez

Nós podemos representar essa informação no código da seguinte maneira



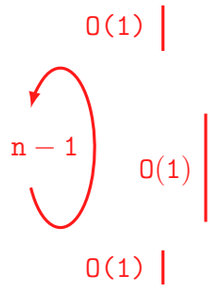

```
M  <--  L[1];      Mpos  <--  1
Para  i <-- 2  Até  N
 {
    Se ( L[i] > M )
     {
        M  <--  L[i];     Mpos  <--  i
 }   }
Imprime ("O maior elemento é:", M)
```

Certo.

Agora está na hora de examinar o que acontece dentro do laço.

E o que nós vemos ali é um trecho de código que é executado apenas uma vez — (*a cada volta do laço*)

Essa informação pode ser representada da seguinte maneira




```
M  <--  L[1];      Mpos  <--  1
Para  i <-- 2  Até  N
 {
    Se ( L[i] > M )
     {
        M  <--  L[i];     Mpos  <--  i
 }   }
Imprime ("O maior elemento é:", M)
```

Pronto.

Com base nessas anotações, nós podemos escrever a seguinte expressão para o tempo de execução do algoritmo

$$O(1) \quad + \quad (n-1) \times O(1) \quad + \quad O(1)$$

Finalmente, descartando os termos menores e concentrando a atenção no termo de maior ordem, nós chegamos ao seguinte

$$\cancel{O(1)} \quad + \quad \underbrace{(n-1) \times O(1)}_{O(n)} \quad + \quad \cancel{O(1)} \quad = \quad O(n)$$

Esse resultado está dizendo que o tempo de execução do algoritmo é da ordem de n

— (*ou mais ou menos igual a* n)

E isso faz sentido porque

→ *percorrer a lista examinando cada um dos seus elementos*

leva tempo proporcional ao tamanho da lista — (mais ou menos ...)

Legal, não é?