Considere a seguinte tarefa

→ *Encontrar o segundo maior elemento da lista* L

Abaixo nós temos o primeiro algoritmo apresentado na aula 00.1 para esse problema, já anotado

```
O(1) |
                 // 1a ETAPA: localizar o maior
                 M  <--  L[1];   Mpos  <--  1
                 Para  i <-- 2  Até  n
                  {
n      O(1) |        Se ( L[i] > M )
                      {
                         M  <--  L[i];   Mpos  <--  i
                  }   }

                 // 2a ETAPA: trocar maior c/ primeiro
O(1) |           Aux  <--  L[1];    L[1]  <--  L[Mpos];   L[Mpos]  <--  Aux

                 // 3a ETAPA: encontrar segundo maior
O(1) |           M2  <--  L[2];
                 Para  i <-- 3  Até  n
                  {
n      O(1) |        Se ( L[i] > M )
                      {
                         M2  <--  L[i];
                  }   }
O(1) |           Imprime ("O segundo maior é:", M2)
```

Observe que, dessa vez, coisas como $n-1$ e $n-2$ foram anotadas simplesmente como n — (*porque a análise é só mais ou menos ...*)

Agora com base nas anotações, nós obtemos a seguinte expressão para o tempo de execução do algoritmo

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

Daí, descartando os termos menores nós chegamos ao seguinte resultado

$$\cancel{O(1)} + \quad n \times O(1) \quad + \quad \cancel{O(1)} \quad + \quad \cancel{O(1)} \quad + \quad n \times O(1) \quad + \quad \cancel{O(1)} \quad = \quad 2 \cdot O(n)$$

Legal.

Agora nós vamos analisar a segunda versão do algoritmo

```
O(1) |           Se ( L[1] > L[2] )
                    {  M1  <--  L[1];   M2  <--  L[2]  }
                 Senão
                    {  M1  <--  L[2];   M2  <--  L[1]  }

                 Para  i <-- 3  Até  n
                  {
n      O(1) |        Se ( L[i] > M1 )
                      {
                          M2  <--  M1;    M1  <--  L[i]
                      }
                     Senão Se ( L[i] > M2 )
                      {
                          M2  <--  L[i]
                  }   }
O(1) |           Imprime ("O segundo maior é:", M2)
```

Daí com base nas anotações, nós obtemos a seguinte estimativa para o tempo de execução do algoritmo

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

Quer dizer, as análises nos deram os seguintes resultados

- `versão 1:` $2 \cdot O(n)$
- `versão 2:` $O(n)$

E isso parece indicar que a segunda versão é mais eficiente do que a primeira.

Mas aqui é preciso cuidado.

Porque, examinando outra vez os códigos, nós vemos que

→ a `versão 1` contém dois laços, com um teste dentro de cada um

→ a `versão 2` contém um laço, com dois testes dentro dele

E isso é basicamente a mesma coisa.

Quer dizer, para levar esse detalhe em conta, nós deveríamos anotar a `versão 2` assim

```
                 ( . . . )
                 Para  i <-- 3  Até  n
                  {
                     Se ( L[i] > M1 )
n      O(2) |        {
                          M2  <--  M1;    M1  <--  L[i]
                      }
                     Senão Se ( L[i] > M2 )
                      {
                          M2  <--  L[i]
                  }   }
                 ( . . . )
```

Mas, para não complicar as coisas, nós vamos assumir que coisas como $2O(n)$ e $3O(n)$ são a mesma coisa que $O(n)$.

Dessa maneira, o resultado final da análise é

- `versão 1:` $O(n)$
- `versão 2:` $O(n)$

De modo que os dois algoritmos são igualmente eficientes — (*do ponto de vista da nossa análise mais ou menos*)

Isso faz sentido, porque os dois algoritmos de fato executam em tempo proporcional ao tamanho da lista.

◇