A ideia mais natural para ordenar uma lista, provavelmente é a seguinte

- encontrar o maior elemento de todos, e colocá-lo na última posição
- encontrar o segundo maior elemento, e colocá-lo na penúltima posição
- encontrar o terceiro maior elemento, e colocá-lo na antepenúltima posição
- e por aí vai ...

A boa notícia é que é bem fácil encontrar o maior elemento da lista

```
M  <--  L[1];    posM  <--  1

Para  i <-- 2  Até  N
 {
     Se ( L[i] > M )
      {
         M  <--  L[i];    posM  <--  i
 }    }
```

E também é bem fácil colocar esse trecho de código dentro de uma função que encontra o maior elemento dentro de uma faixa da lista

```
func  posMaior ( L, inicio, fim )
 {
    M  <--  L[inicio];    posM  <--  inicio

    Para  i <-- inicio + 1  Até  fim
     {
         Se ( L[i] > M )
          {
             M  <--  L[i];    posM  <--  i
     }    }
    Retorna ( posM )
 }
```

Certo.

Agora imagine que nós também temos a função

```
func  Troca ( L, i, j )
```

— (*que troca os elementos das posições* `i, j` *de lugar*)

Então, é bem fácil ver que o seguinte par de instruções coloca o maior elemento na última posição

```
k  <--  posMaior ( L, 1, N)
Troca ( L, k, N )
```

E que depois disso, o seguinte par de instruções coloca o segundo maior elemento na penúltima posição

```
k  <--  posMaior ( L, 1, N-1)
Troca ( L, k, N-1 )
```

Daí que, para ordenar a lista completamente, basta botar esse negócio para repetir

```
Para  j <-- N Até  2
 {
    k  <--  posMaior ( L, 1, j)

    Troca ( L, k, j )
 }
```

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

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

Análise

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

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

```
                 func  posMaior ( L, inicio, fim )
                  {
O(1)                 M  <--  L[inicio];    posM  <--  inicio

                     Para  i <-- inicio + 1  Até  fim
                      {
?                         Se ( L[i] > M )
O(1)                       {
                              M  <--  L[i];    posM  <--  i
                      }    }
O(1)                 Retorna ( posM )
                  }
```

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

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

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

Certo.

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

```
O(n)                 Para  j <-- N Até  2
                      {
O(n) ←                   k  <--  posMaior ( L, 1, j)

O(1) ←                   Troca ( L, k, j )
                      }
```

— (*porque a função* `Troca( )` *foi anotada com* $O(1)$*?*

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

$$O(n) \cdot \Big(O(n) \quad + \quad O(1) \Big) \qquad = \qquad O(\, n^2 \,)$$

Legal.

Mas daí, alguém poderia dizer que a nossa análise não foi muito precisa.

Quer dizer, a gente assumiu que a função `posMaior( )` executa em tempo $O(n)$.

Mas, isso é apenas o tempo de execução no pior caso.

No melhor caso, a função executa em tempo $O(1)$ — (*e é isso o que acontece nas últimas iterações do algoritmo*)

Então, para levar isso em conta, nós vamos raciocinar com mais precisão.

Quer dizer, nós vamos considerar o tempo de execução de cada volta do laço em separado.

- na primeira volta, a chamada à função `posMaior()` leva em tempo n
- na segunda volta, a chamada à função `posMaior()` leva tempo $n - 1$
- na terceira volta, a chamada à função `posMaior()` leva tempo $n - 2$
- e por aí vai ...

Daí que, somando todos esses tempos a gente chega na expressão

$$n \quad + \quad (n-1) \quad + \quad \ldots \quad + \quad 3 \quad + \quad 2 \quad + \quad 1$$

Mas, todo mundo sabe que isso vale

$$\frac{n \cdot (n+1)}{2}$$

E nós temos que

$$\frac{n \cdot (n+1)}{2} \quad = \quad O(n^2)$$

— (*porque?*)

Quer dizer, no final das contas, a análise mais precisa não fez a menor diferença — (*ou só fez uma diferença menor ...*)

É por isso que, na maioria das vezes, a gente não faz uma análise precisa do tempo de execução de um algoritmo.