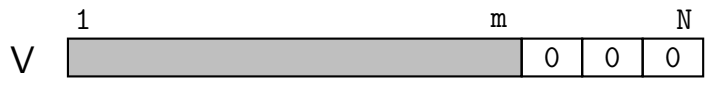
Imagine mais uma vez que nós temos uma lista com posições vazias no final



A tarefa é

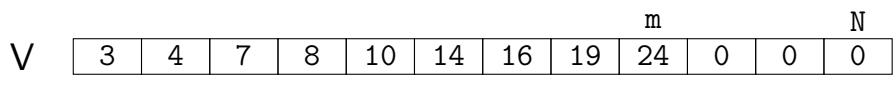
→ *Encontrar o número* `x` *e atualizar o seu valor para* `y`

Dessa vez, a versão desordenada não tem muita graça.

Quer dizer, primeiro a gente faz a busca, e depois atualiza o valor do elemento.

E isso leva tempo $O(n)$ — (*porque?*)

Então, a gente imagina direto que a lista está ordenada



E imagina que `x` = `24`.

Daí nós temos dois casos.

Quer dizer,

- Se `y` for menor que `x`, então após a atualização pode ser necessário arrastar o elemento para a esquerda
- Se `y` for maior que `x`, então após a atualização pode ser necessário arrastar o elemento para a direita

A coisa fica assim

```
// 1a Etapa:  procurar o x

O(n) | (...  a mesma coisa ...)

// 2a Etapa:  deslocar elementos para a esquerda

Se ( achei == 1)
O(1) | V[i] = y                                  // atualiza o elemento
     | Se (y < x)
          Enquanto ( i > 1 )                     // arrasta para a esquerda
O(n)           | Se ( V[i] < V[i-1])
O(1)           |      aux = V[i]
               |      V[i] = V[i-1]
               |      V[i-1] = aux
               | Senão
               |      Quebra
               | i--

Se (y < x)
     Enquanto ( i < m )                          // arrasta para a esquerda
O(n)           | Se ( V[i] > V[i+1])
O(1)           |      aux = V[i]
               |      V[i] = V[i+1]
               |      V[i+1] = aux
               | Senão
               |      Quebra
               | i++
```

Agora basta examinar as anotações para obter o tempo de execução do algoritmo

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

◇