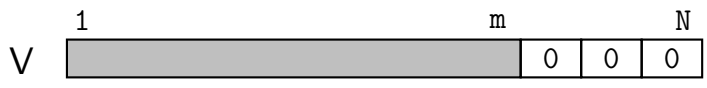
Imagine mais uma vez que nós temos uma lista com posições vazias no final



A tarefa é

→ *Remover o número* `k` *da lista*

Bom, se a lista não está ordenada, a tarefa é fácil

- basta colocar o último elemento na posição do `k`, e depois diminuir o valor de m

Mas antes disso, é preciso encontrar o `k`.

Abaixo nós temos o algoritmo completo (já anotado)

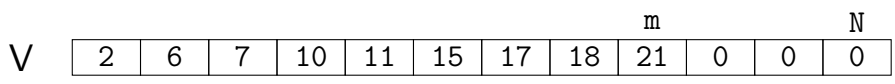
```
                    // 1a Etapa:  procurar o k

O(1)  | i = 1;    achei = 0
      | Enquanto ( i ≤ m )                    // percorre a lista
      |     | Se ( V[i] == k )                 // Você é o k?
O(n)  |O(1) |     achei = 1
      |     |     Quebra
      |     | i++

                    // 2a Etapa:  remover o k
      | Se ( achei == 1 )                      Se o k foi encontrado
O(1)  |     V[i] = V[m]
      |     m--
```

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

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

Legal.

Agora imagine que a lista está ordenada

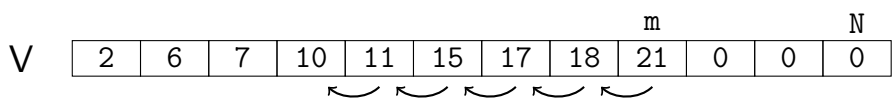


E imagine que `k` = 10.

Bom, dessa vez nós não podemos colocar o último elemento no lugar do `k` — (*porque a lista iria ficar desordenada*)

E nem podemos deixar a posição do `k` vazia — (*porque?*)

Então, a gente move todos os elementos depois do `k` uma posição para a esquerda



A coisa fica assim

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

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

                    // 2a Etapa:  deslocar elementos para a esquerda

      Se ( achei == 1)
          j = i+1                            // primeiro elemento depois do k
          Enquanto ( j ≥ m )                 // vai até o fim
O(n)      |O(1) | V[j-1] = V[j]              // deslocando para a esquerda
          |     | j++

                    // 3a Etapa:  apagar o último elemento
O(1)  | V[m] = 0
      | m--
```

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

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

◇