Ordenação por seleção

Invariante do tipo mais da saida

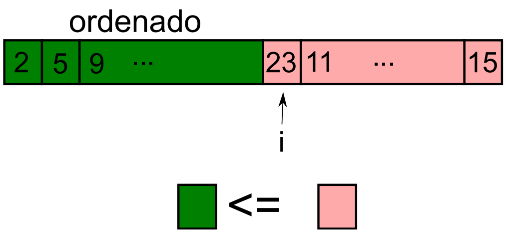
Invariante 1: A porção do início até a posição `i-1` está ordenada.

- `a[:i]` está ordenado

Invariante 2: Todos os elementos desta porção já estão na sua posição final.

- Para todo `j, k` tal que `i <= j < len(a) e 0 <= k < i` , `a[j] >= a[k]`

Invariante 3: `a` é uma permutação do `a` original



In [1]:

```
def ordenarSelecao(a):
    # mais da saída
    for i in range(len(a)-1):
        j = indiceDoMenorDesde(i, a)
        a[i], a[j] = a[j], a[i]
```

Para localizar o índice do menor elemento na porção `a[i:]` , usamos uma adaptação do algoritmo dos dois dedos

In [2]:

```
def indiceDoMenorDesde(i, a):
    m = i
    # mais da entrada
    for k in range(m+1, len(a)):
    # invariante: m aponta para o maior de a[i:k]
        if a[k] < a[m]:
            m = k
    return m

xs = [3,1,2,10,2, 5]
ordenarSelecao(xs)
xs
```

Out[2]:

```
[1, 2, 2, 3, 5, 10]
```

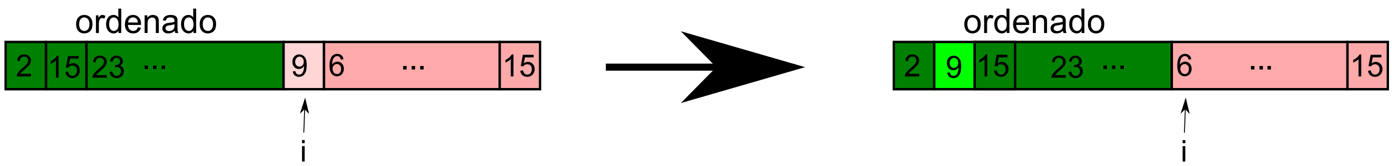
Ordenação por inserção

Método usado pelos jogadores de baralho.

Invariante do tipo mais da entrada

Invariante1: A porção `a[:i]` está ordenada

Invariante2: A porção `a[:i]` é uma permutação da parte original correspondente de `a[:i]`



In [3]:

```
def ordenarInsercao(a):
    # mais da entrada
    for i in range(1, len(a)):
        inserirEmOrdem(i, a)
```

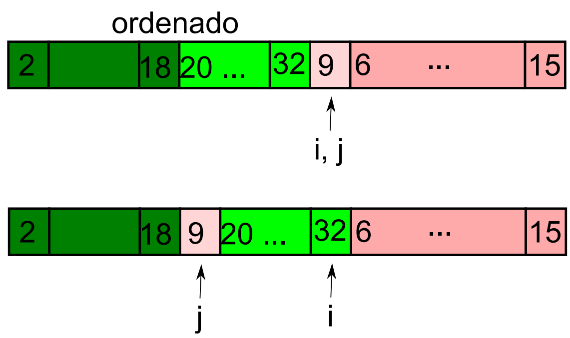
Para inserir mantendo a ordem o i-éssimo elemento na parte correspondente a `a[:i]` , que já está ordenada, usamos um algoritmo baseado em um invariante mais da saída

invariante1 : `a[j:i+1]` está ordenada

invariante2: `a[j+1:i+1]` é igual com a parte `a[j:i]` do original

invariante3: `a[j]` é igual com o original `a[i]`

invariante4: `a[:j]` está inalterado



In [4]:

```
def inserirEmOrdem(i, a):
    j = i
    while j > 0 and a[j-1] > a[j]:
        a[j-1], a[j] = a[j], a[j-1]
        j -= 1

xs = [3,1,2,10,2, 5]
ordenarInsercao(xs)
xs
```

Out[4]:

```
[1, 2, 2, 3, 5, 10]
```