Ordenação pelo método da bolha

Invariante do tipo **trabalho feito**

A **medida de progresso** é dada pela quantidade de pares fora da ordem.

- A condição de parada é que a lista esteja ordenada

Nesete caso particular, como a mera terminação implica na ordenação, o único cuidado que devemos ter é que a lista obtida é uma permutação da original, o que conseguimos com o invariante a seguir:

invariante: a lista é uma permutação da lista original

Em particular, podemos focar em pares contíguos fora da ordem para fazer trocas. Observe que a troca de ordem de um par contíguo fora da ordem vai decrementar em um a quantidade total de pares fora da ordem.

In [2]:

```
def ordenarBolha(a):
    while not ordenado(a):
        i = localizarParForaDaOrdem(a)
        a[i], a[i+1] = a[i+1], a[i]
```

Na implementação de `localizarParForaDaOrdem` , adotamos como convenção que

- se há um par fora da ordem, a função retorna o primeiro par contíguo fora da ordem. Em particular, retorna o índice do elemento esquerdo do par;
- se não há nenhum par fora da ordem, a função retorna o índice do último elemento da lista

O algoritmo se baseia no seguinte invariante mais da entrada

Invariante: em `a[:i]` não há pares fora da ordem

Um invariante similar, também mais da entrada, nos serve para implementar `ordenado`

In [3]:

```
def localizarParForaDaOrdem(a):
    i = 0
    while i < len(a)-1 and a[i] <= a[i+1]:
        i += 1
    return i

def ordenado(a):
    i = 0
    while i < len(a)-1 and a[i] <= a[i+1]:
        i += 1
    return i == len(a)-1

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

Out[3]:

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

Alternativamente, podemos implementar `localizarParForaDaOrdem` e `ordenado` usando laços `for`.

In [ ]:

```
def localizarParForaDaOrdem(a):
    for i in range(len(a)-1):
        if a[i] > a[i+1]:
            return i
    return len(a) - 1

def ordenado(a):
    for i in range(len(a)-1):
        if a[i] > a[i+1]:
            return False
    return True
```

Observe que `localizarParForaDaOrdem` e `ordenado` fazem basicamente o mesmo trabalho, o único que muda é o valor retornado. Podemos melhorar a eficiência do algoritmo eliminando a chamada a `ordenado`.

Observem que `localizarParForaDaOrdem` retorna o último índice da lista sse a lista está ordenada. Podemos então evitar a chamada a `ordenada` assim:

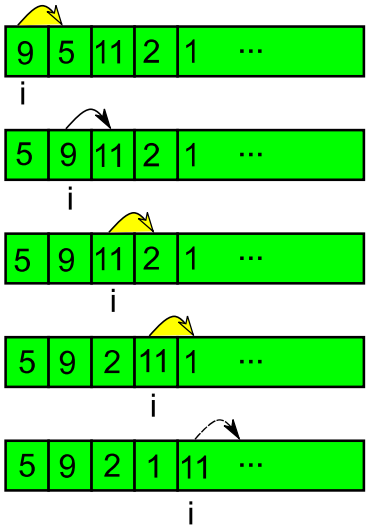
In [ ]:

```
def ordenarBolha(a):
    i = 0
    while i != len(a)-1 :
        i = localizarParForaDaOrdem(a)
        a[i], a[i+1] = a[i+1], a[i]
```

Algoritmo da bolha clássico

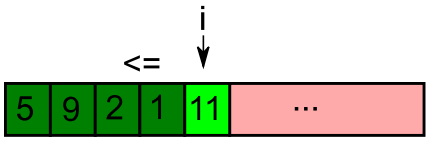
O algoritmo de ordenação pelo método da bolha que normalmente se apresenta na literatura não é que está acima, mas sim uma variante mais eficiente.

Dentro do laço while, ao invés de localizar um par fora da ordem e fazer a respectiva troca, podemos percorrer a lista de esquerda à direita analisando par a par se algum está fora da ordem e, se estiver, fazemos a respectiva troca. A função `borbulhar`, definida embaixo, faz este percurso.



Observem que temos como **invariante** que

> todos os elementos à esquerda de `i` são menores que `a[i]`.



In [ ]:

```
def borbulhar(a):
    for i in range(len(a)-1):
        if a[i] > a[i+1]:
            a[i], a[i+1] = a[i+1], a[i]
```

Em particular, a função `borbulhar`, após percorrer completamente a lista, puxará o maior elemento de `a` ao extremo direito. E, se repetirmos `borbulhar`, o segundo maior elemento será puxado até a penúltima posição. Podemos continuar assim de tal forma que

- se borbulhamos `n-1` vezes, onde `n` é o comprimento da lista, a lista ficará ordenada

In [ ]:

```
def ordenarBolhaClassico(a):
    for _ in range(len(a)-1):
        borbulhar(a)
```

Seja `n` o comprimento da lista.

Podemos obter duas melhoras para este último algoritmo observando que:

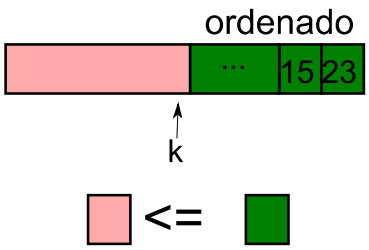
- se `borbulhar` não faz nenhuma troca, então a lista está ordenada e podemos parar;

- depois de chamar borbulhar `i` vezes, a porção relativa a `a[n-i:]` já estará resolvida (ordenada e com todos os elementos na posição definitiva),

Em outras palavras, a saída será construída pela direita. Se o progresso é controlado pela variável `k` que irá decrementando de um em um, temos os seguintes invariantes mais da saída

invariante1: `a[k+1:]` está ordenado

invariante2: todos os elementos de `a[k+1:]` são maiores ou iguais a qualquer elemento em `a[:k+1]`



In [2]:

```
def ordenarBolhaClassico1(a):
    k = len(a)-1
    while k > 0 and not borbulharAte(a, k):
        k -= 1

def borbulharAte(a, j):
    ordenado = True
    for i in range(j):
        if a[i] > a[i+1]:
            a[i], a[i+1] = a[i+1], a[i]
            ordenado = False
    return ordenado

xs = [3, 1, 2, 10, 2, 5]
ordenarBolhaClassico1(xs)
xs
```

Out[2]:

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