InsertionSort(A[1..n])

Para $i \leftarrow 1$ até n

$j \leftarrow i$

Enquanto $j > 1$ e $A[j] < A[j-1]$

$A[j] \rightarrow A[j-1]$

$j \leftarrow j - 1$

Corretude (total)

↳ Garantir que algoritmo retorna resposta correta e que sempre termina de executar

Saber como provar nos ajuda a desenvolver

↳ projetar algoritmo com prova na cabeça

↳ evidencia erros

Prova formal (matemática) que algoritmo funciona corretamente: baseia-se em duas estruturas de repetição

Invariantes de laço: Afirmações verdadeiras no início e no fim de cada iteração de um laço.

Provar a invariante em 3 partes:

1) Inicialização: A invariante é válida antes da primeira iteração

2) Manutenção: Supondo que é válida no início da iteração, prova que a informação continua verdadeira no fim da mesma iteração

3) Término: Quando o laço termina, a invariante fornece uma propriedade útil que ajuda a provar que o algoritmo é correto.

```
InsertionSort(A[1..n])
  Para i ← 1 até n
    j ← i
    Enquanto j > 1 e A[j] < A[j-1]
      A[j] ↔ A[j-1]
      j ← j - 1
```

Invariante do laço interno:

Para $1 \leq k < k' \leq i$, se $k' \neq j$, então $A[k] < A[k']$.

Prova:

1) Inicialização:
   - Recebe vetor ordenado até $i-1$

2) Manutenção:
   - Seja x o elemento da posição $A[j-1]$ e y o elemento da pos. j. Como a invariante é válida antes da iteração do laço, x é maior que todos os elementos que vem antes dele no vetor, e menor que os elementos $A[j+1..i]$

```
InsertionSort(A[1..n])
  Para i ← 1 até n
    j ← i
    Enquanto j > 1 e A[j] < A[j-1]
      A[j] ↔ A[j-1]
      j ← j - 1
```

2) continuação

Dentro do laço, acontece apenas a troca de x com o elemento da posição j(y), e portanto x continua sendo maior que todos os elementos que vem antes dele (a troca só acontece se x > y) e menor que os elementos de A[j+1..i].

Observe que a nova posição de y é j-1. Mas, depois da troca, j ← j-1, e, portanto, o elemento da posição j continua sendo o único em A[1..i] que está possivelmente fora de ordem.

3) Término:

- j decresce e no pior caso chega no valor 1.

```
InsertionSort(A[1..n])
  Para i ← 1 até n
    j ← i
    Enquanto j > 1 e A[j] < A[j-1]
      A[j] ↔ A[j-1]
      j ← j - 1
```

---

* Quando o loop interno termina, temos uma propriedade, que vai ajudar na corretude do algoritmo, que passa a ser válida. Qual seria ela?

Subvetor $A[1..i]$ estará ordenado.

Exercício: Assumindo a invariante do loop interno e a propriedade que ela fornece, provar a invariante do loop externo que implica na corretude do algoritmo.