**UNIVERSIDADE FEDERAL DO ABC**
**CENTRO DE MATEMÁTICA, COMPUTAÇÃO E COGNIÇÃO**
Prof. Monael Pinheiro Ribeiro
Prof. Paulo Henrique Pisani

# INSERTION SORT
# ANÁLISE DE CORRETUDE

`InsertionCorretude.[ c | cpp | java | cs ]`

Em Ciência da Computação é possível provar formalmente que um algoritmo é correto através de uma invariante. A invariante de um laço define uma propriedade que se deseja manter; tal prova é dividida nas seguintes partes:

- Inicialização: É uma proposição verdadeira antes do laço iniciar;
- Manutenção: Se for verdadeira antes de uma iteração do laço, então se mantem verdadeira antes da próxima iteração do laço; e
- Término: Quando o laço termina, a invariante resulta em uma propriedade que leva à prova de correção do algoritmo.

No algoritmo de ordenação Insertion Sort a invariante que prova sua corretude está no seguinte laço:

```
for(i=1; i < TAMANHO DA LISTA L; i++)
```

É possível definir a seguinte invariante para este laço:

- No início de cada iteração deste laço, a sublista até Li é formada pelos elementos que já estavam na lista inicial, porém ordenados.

Para provar que a ordenação Insertion Sort é correta, as três etapas da prova são enunciadas da seguinte forma:

- Inicialização: A sublista inicia com o primeiro elemento da lista L, que por sua vez, já está ordenada.
- Manutenção: A cada iteração do laço, a sublista expande-se mantendo a propriedade de estar ordenada. Um novo elemento é inserido na sublista somente quando for

maior que o elemento a sua esquerda. Com os elementos a sua esquerda mantendo a mesma propriedade; isso significa que o novo elemento é maior que todos os elementos a sua esquerda, desta maneira se garante que a sublista permaneça ordenada.

- Término: O código termina quando o laço chega ao último elemento da lista inicial, o que significa que a sublista ordenada é expandida até o tamanho total da lista inicial, o que faz com que a lista inicial esteja completamente ordenada.

Neste exercício você deverá executar o algoritmo de ordenação Insertion Sort e a cada nova inserção de elementos na sublista ordenada, você deverá imprimir e discriminar a sublista ordenada e a sublista não ordenada.

## Entrada

A entrada é composta de apenas um caso de teste com 2 linhas.

Na primeira linha há um inteiro **N**, que consiste no comprimento da lista **L**.

A segunda linha terá uma sucessão de **N** valores inteiros separados por um espaço em branco cada, representando os **N** elementos da lista **L**.

Restrições

- $1 \leq \mathbf{N} \leq 1000$
- $-10000 \leq \mathbf{L_i} \leq 1000$

## Saída

Seu programa deve gerar **N** pares de linhas com sublista de **L** ordenada e desordenada no momento que o laço da invariante inicia uma iteração. Após imprimir cada uma das sublistas de **L**, salte uma linha. Siga o exemplo a seguir.

# Exemplos

| Entrada | Saída |
| --- | --- |
| 6<br>1 4 3 5 6 2 | Sublista Ordenada: 1<br>Sublista Desordenada: 4 3 5 6 2<br>Sublista Ordenada: 1 4<br>Sublista Desordenada: 3 5 6 2<br>Sublista Ordenada: 1 3 4<br>Sublista Desordenada: 5 6 2<br>Sublista Ordenada: 1 3 4 5<br>Sublista Desordenada: 6 2<br>Sublista Ordenada: 1 3 4 5 6<br>Sublista Desordenada: 2<br>Sublista Ordenada: 1 2 3 4 5 6<br>Sublista Desordenada: |