# UNIVERSIDADE DA BEIRA INTERIOR

**Algoritmos e Estruturas de Dados** **1º Semestre**

**Frequência 1** **2025/2026**

## A. Análise de complexidade dos algoritmos

Considere o seguinte bloco de código em linguagem C:

```
...
for (k = 1; k < n; k++) {
   for (j = 1; j < k; j++)
      printf("OLA.\n");
}
...
```

**1.** Simule o bloco de código dado, considerando que **n** é um número inteiro **muito grande**. Apresente os dados da simulação numa tabela cujo cabeçalho é o seguinte:

| k | k < n (S/N) | j | j < k (S/N) | printf |
|---|---|---|---|---|
| 1 | 1 < n (S) | ... | ... | ... |
| ... | ... | ... | ... | ... |

**2.** A partir dos resultados da simulação obtidos em 1, determine a **ordem de complexidade** do algoritmo associado ao bloco de código dado. Justifique, apresentando os cálculos efetuados.

## B. Algoritmos de ordenação e de pesquisa

Considere a seguinte função já implementada, que devolve o **menor índice** de um elemento do array **A** (de tamanho **N**) com valor igual a **E** (se E existe em A) ou **-1** (se E não existe em A):

```
int pesquisaSequencial (int E, int A[], int N);
```

Pretende-se implementar uma **função** em C que dados um array **A** com **N** elementos do tipo inteiro **ordenado** por ordem **crescente** e um inteiro **E**, **determine** e **devolva** a quantidade de elementos do array **A** com valores iguais a **E**.

Desta forma, complete a função seguinte para que consiga o objetivo proposto.

```
int  quantidadeIguais  (int A[], int N, int E)
{
   int  k, cont = 0;
   k = pesquisaSequencial(E, A, N);
      .
      .
      .
   return cont;
}
```

**NOTA:** A função **pesquisaSequencial** é para ser usada de forma a otimizar o algoritmo (realizar o menor número de operações possível).

## C. Listas ligadas

Considere as seguintes declarações associadas a EAD Lista:

```
typedef  int  DadosLista;   //  DadosLista = int
struct NodoLista {
    DadosLista  Elemento;
    struct NodoLista *Prox;
};
typedef struct NodoLista *PNodoLista;
```

assim como os seguintes protótipos de funções já implementadas (operações básicas sobre a EAD Lista):

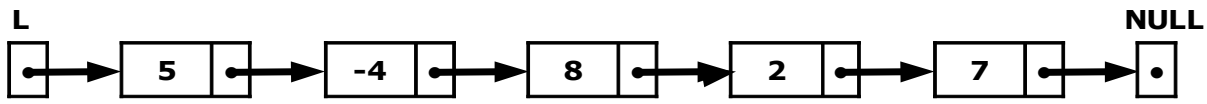
```
int listaVazia (PNodoLista L);  // devolve 1 se lista L está vazia e 0 caso contrário
PNodoLista criarNodoLista (DadosLista X);  // devolve um ponteiro para um novo nodo com o X
```

Usando as definições e as funções dadas (protótipos), resolva as seguintes questões:

1. **Usando** ou **não** as funções fornecidas em cima, mas **não podendo** usar outras funções, **implemente uma função iterativa (não recursiva) em C** que
   - **receba** uma lista **L** com elementos do tipo **DadosLista** (**int**) e um inteiro **X** (parâmetros da função),
   - **devolva** a lista **L** após **inserir** um nodo com o elemento **X** como **sucessor** de um nodo com o **menor** elemento da lista **L**; se a lista **L** estiver vazia, o nodo com o **X** será o único elemento da lista.

   **Nota:** Otimizar o algoritmo, de forma que seja realizado o menor número de operações possível.

   Exemplo: o elemento **X** será inserido entre o nodo com **-4** e o nodo com **8** (**-4** é o menor da lista).




2. **Implemente** uma **função recursiva em C** que
   - **receba** uma lista **L**, com elementos do tipo **DadosLista** (números inteiros) e um número inteiro **K** (parâmetros da função),
   - **determine** e **devolva** a soma dos valores do campo **Elemento** dos elementos de **L** a partir da **K-ésima posição** da lista (o elemento da K-ésima posição não entra na soma).

## D. Pilhas

Considere as seguintes declarações associadas a EAD Pilha:

```
typedef  int  DadosPilha;
struct NodoPilha { … };
typedef struct NodoPilha *PNodoPilha;
```

assim como os seguintes protótipos de funções já implementadas (operações básicas sobre a EAD Pilha):

```
PNodoPilha criarPilha ();
int pilhaVazia (PNodoPilha S);
PNodoPilha push (DadosPilha X, PNodoPilha S);
PNodoPilha pop (PNodoPilha S);
DadosPilha topo (PNodoPilha S);
```

Usando as operações básicas sobre uma EAD Pilha, implemente uma **função em C** que

- **receba** uma **pilha S** com valores inteiros positivos e sem repetições (parâmetro da função),
- **determine** e **devolva** a soma dos elementos que **estão acima do maior elemento da pilha**, mas devolvendo a pilha **S** como no início; se o maior elemento está no topo da pilha **S**, então devolver 0.

Ex: S = [ 3, 7, 2, 9, 5, 1 ], com topo(S) = 3 => resultados: 12 (3+7+2) e S = [ 3, 7, 2, 9, 5, 1 ]