UNIVERSIDADE DA BEIRA INTERIOR

Algoritmos e Estruturas de Dados **2º Semestre**

Exame – Época Normal (16 val) **2 h + 30 min** **23/06/2025**

A. [1.25 val = 0.75 + 0.50] 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");
}
```

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
…	…	…	…	…

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. [3.50 val = 1.50 + 2.00] Listas ligadas

Considere as seguintes declarações associadas a EAD Lista:

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

assim como o seguinte protótipo de uma função já implementada (operação básica sobre a EAD Lista):

```
PNodoLista libertarNodoLista (PNodoLista P); // liberta a memória apontada por P e devolve NULL
```

1. Implemente uma **função recursiva em C** que

- receba uma lista **L** com elementos do tipo **INFOLista** (inteiros) e dois inteiros **X** e **Y** (com X < Y), e

- devolva a quantidade de elementos de **L** cujos valores no campo **Elemento** estão em { X, …, Y }.

2. Implemente uma função iterativa (não recursiva) em C que

- **receba** uma lista **L** de elementos do tipo **INFOLista** (inteiros), e
- **remova** da lista **L** o **nodo** com o **maior valor** no campo **Elemento,** caso exista.

Nota: Otimizar os algoritmos, de forma que a lista L seja percorrida apenas uma vez.

C. [2.00 val] Pilhas

Considere as seguintes declarações associadas a EAD Pilha:

```
typedef int INFOPilha;      // INFOPilha = int
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 pop (PNodoPilha S); **INFOPilha topo (PNodoPilha S);**

PNodoPilha push (INFOPilha X, PNodoPilha S);

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

- **receba** uma **pilha S** com elementos do tipo INFOPilha (números inteiros), e
- **determine** e **devolva** a **quantidade** de elementos da pilha **S** com valores **iguais** ao elemento do **fundo da pilha S** e devolvendo a pilha S inalterada.

Ex: **S** = [3, -2, 4, -5, 4, 5, 4], em que topo(S) = 3 => resultados: **3** e **S** = [3, -2, 4, -5, 4, 5, 4]

D. [3.75 val = 1.50 + 2.25] Árvores Binárias de Pesquisa (ABP)

Considere as seguintes declarações associadas de EAD Árvore Binária de Pesquisa:

```
typedef int INFOABP;    // INFOABP = int
struct NodoABP {
   INFOABP  Elemento;
   struct  NodoABP  *Esquerda;
   struct  NodoABP  *Direita;
};
typedef struct NodoABP *PNodoABP;
```

1. Implemente uma **função recursiva** em C que **receba** uma ABP **T** e **devolva** a quantidade de elementos de **T** cujos valores no campo **Elemento são** números **positivos** (> 0).
2. Implemente uma **função iterativa** (**não recursiva**) em C que **receba** uma ABP **T** e um número **inteiro X**, e **devolva** um ponteiro para o **nodo-pai** do **nodo** com o **maior** valor no campo **Elemento** que seja **menor** que **X**; no caso em que **não exista o nodo-pai**, a função deve devolver **NULL**.

Nota: otimize os algoritmos criados (código escrito) tendo em conta a **definição de ABP**.

E. [2.25 val] ABP Balanceadas/Equilibradas (AVL)

Construa uma **ABP do tipo AVL** da seguinte forma:

- **insira** os nodos com os seguintes valores (um de cada vez e pela ordem apresentada):
  40, 60, 80, 70, 75 e 65. (**NOTA**: não é necessário colocar as alturas).

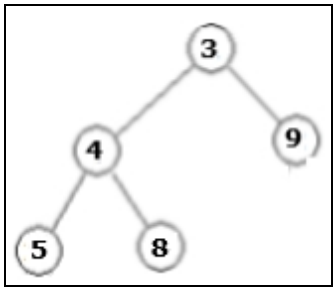
  Sempre que **inserir um nodo**, deve **acrescentar** o novo nodo à árvore e **verificar se continua balanceada/equilibrada**. Caso **não fique equilibrada**, deve

  1º) indicar o nodo onde se deteta o desequilíbrio e o tipo de rotação a aplicar para reequilibrá-la,

  2º) reequilibrar a árvore e redesenhá-la (apresentando todos os passos efetuados)
- **remova** o **nodo raiz** da ABP e **redesenhe** a ABP obtida, **verifique** se continua balanceada/equilibrada e **reequilibre-a**, caso seja necessário. Apresente a árvore final.

F. [1.00 val] Heaps

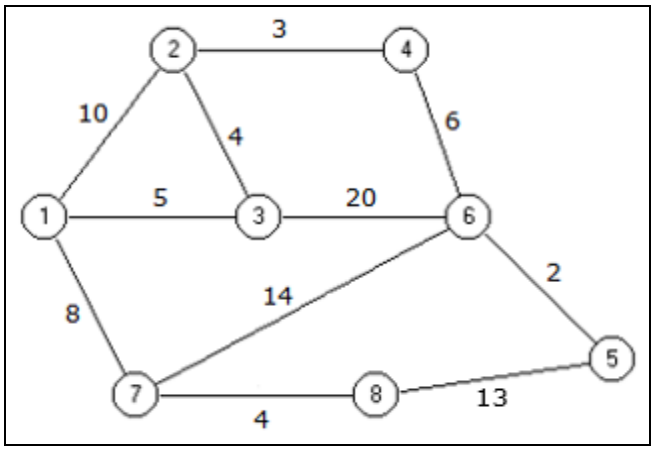
Considere a seguinte **minHeap**:



1. Insira um nodo com o valor **7** na **minHeap apresentada ao lado** e **redesenhe** a **minHeap obtida**. Justifique, apresentando a evolução da minHeap desde a estrutura inicial até à estrutura final.
2. Aplique a operação **remover** à **minHeap apresentada ao lado** e **redesenhe a minHeap obtida**. Justifique, apresentando a evolução da minHeap desde a estrutura inicial até à estrutura final.

G. [2.25 val] Grafos

Considere a rede G = (A, N, C), onde o valor de cada arco corresponde ao seu custo/comprimento ($\mathbf{C_{ij}}$):



1. Determine a **Árvore dos Caminhos Mais Curtos (ACMC)** do **nó 5 (S = 5) para todos os outros nós**, simulando o algoritmo de **Dijkstra**. Apresente os dados da simulação: todos os valores das estruturas de dados **R**, **Pai**, **Permanentes** e **Temporários**, e das variáveis **k** e **j**. Desenhe a ACMC.
   Esquema para apresentar os dados da simulação:

	1	2	3	4	5	6	7	8
R								
Pai								

Permanentes ←

Temporários ←

k ← ?
  j ← ?
k ← ?
  j ← ?
...

2. A **partir dos resultados da simulação**, **indique** qual o caminho mais curto do nó **5** ao nó **1**, assim como o respetivo **custo/comprimento**. Justifique com os dados da simulação.

Algoritmo de Dijkstra:

Passo 1.
  $R_S \leftarrow 0$
  $Pai_S \leftarrow S$
  $R_i \leftarrow C_{Si}$, se $(S, i) \in A$
  $Pai_i \leftarrow S$, se $(S, i) \in A$
  $R_i \leftarrow \infty$, se $(S, i) \notin A$
  $Pai_i \leftarrow S$, se $(S, i) \notin A$
  Permanentes = { S }
  Temporários = N - { S }

Passo 2.
  se (Temporários = ∅) **então**
    PARAR *(todos os nós têm rótulos permenentes – determinada ACMC)*
  fim_se
  k ← nó de Temporários em que R_k é mínimo ($k : R_k = \min \{R_X , X \in Temporários\}$)
  Permanentes ← Permanentes ∪ { k }
  Temporários ← Temporários - { k }

Passo 3.
  para (todo o **j** ∈ **N** tal que $(k, j) \in A$ **e** $j \in$ Temporários) **repetir**
    se ($R_k + C_{kj} < R_j$) **então**
      $R_j \leftarrow R_k + C_{kj}$
      $Pai_j \leftarrow k$
    fim_se
  fim_para
  regressar ao **Passo 2**