UNIVERSIDADE DA BEIRA INTERIOR

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

Exame Época Normal (16 val) **2 h + 30 min** **18/06/2026**

Nota: nas perguntas que impliquem a criação de código, os algoritmos que lhes estão associadas devem ser otimizados (executarem o menor número possível de operações/ações).

A. [0.75 val + 0.50 val] Análise de complexidade dos algoritmos

Considere o seguinte bloco de código em linguagem C:

```
...
for (k = 1; k < 4; k++) {
    for (j = k; j < n; j++)
        printf("OLA.");
}
...
```

1. Simule o bloco de código dado (apresente apenas as 2 primeiras e as 2 últimas iterações de cada um dos ciclos), considerando que **n** é um número inteiro **muito grande**.

Apresente os dados da simulação numa tabela com a seguinte estrutura:

k	k < 4 (S/N)	j	j < n (S/N)	printf()
1	1 < 4 (S)	...	...	…
...	...	...	...	...

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

B. [1.50 val] Algoritmos de ordenação e de pesquisa

Implementar uma **função** em C que dado um array **A** com **N** números **inteiros positivos** (pode haver repetidos) **ordenados** por **ordem crescente** (da primeira para a última posição), **devolva o terceiro maior valor** do array **A**, se existir; se não existir, devolver **-1**.

C. [1.25 val + 2.00 val] 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;
```

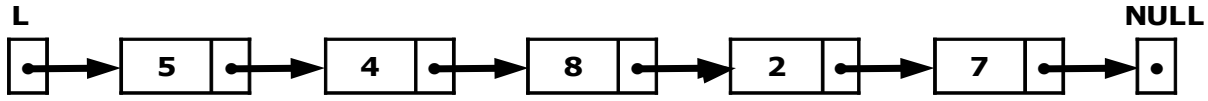
assim como os seguintes protótipos de funções já implementadas (operações básicas sobre a EAD Lista):

```
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. **Implemente** uma **função recursiva em C** que
   - **receba** uma lista **L** com elementos do tipo **DadosLista** (números inteiros) e dois números inteiros **A** e **B** (com A < B),
   - **devolva** a **quantidade** de elementos de **L** com valores no campo **Elemento** entre A e B.
2. **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** (números inteiros) e um número inteiro **X**,
   - **devolva** a lista **L** após **inserir** um nodo com o elemento **X** como **antecessor** de um nodo com o **maior** elemento da lista **L**; se a lista **L** estiver vazia, o nodo com o **X** será o único elemento da lista.

   Ex: o elemento **X** será inserido entre o nodo com **4** e o nodo com **8** (**8** é o maior da lista).



D. [2.00 val] Pilhas

Considere as seguintes declarações associadas a EAD Pilha:

```
typedef int DadosPilha;    // DadosPilha = 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 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 números **inteiros positivos** (parâmetro da função),
- **devolva** a **quantidade** de elementos da pilha **S** com **valores maiores** que o valor do elemento que **está no fundo da pilha**, mas devolvendo a pilha **S** como no início; se S está vazia, devolver **-1**.

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

E. [2.25 val] Árvores Binárias de Pesquisa (ABP)

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

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

Implemente **uma** função em C que

- **receba** uma ABP **T** e um número **inteiro X**, e
- **devolva** o **maior** valor em **T** que seja **par** e **maior** que **X**, se existe; se não existe, devolver **-1**.

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

Equilibrar uma **ABP do tipo AVL**, consiste em aplicar uma das 4 operações de **rotação**: simples à esquerda, simples à direita, dupla à esquerda e dupla à direita.

Construa uma **ABP do tipo AVL** da seguinte forma (pela ordem indicada):

a) **insira** os nodos com os seguintes valores (um de cada vez e pela ordem apresentada):

40, 50, 80, 30, 35 e 20 (**NOTA**: ao desenhar as ABP não é necessário colocar as alturas)

Sempre que **inserir um nodo**, deve-o **acrescentar** à ABP e **verificar se continua 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,

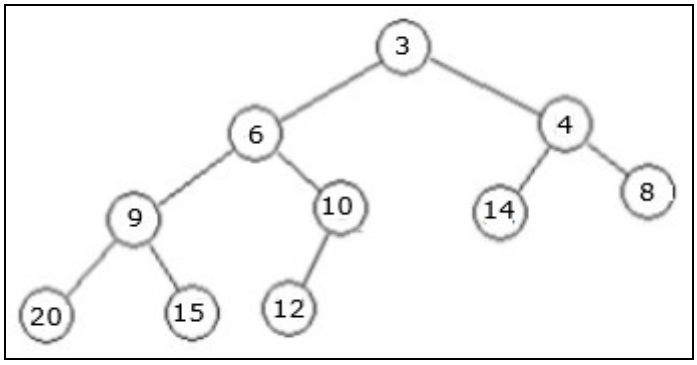
2º) reequilibrar a árvore e redesenhá-la (apresentando todos os passos efetuados)

b) **remova** a **raiz** da ABP obtida em **a)** e **redesenhe** a ABP obtida com a remoção, **verifique** se continua equilibrada e **reequilibre-a**, caso seja necessário. Apresente a árvore final.

NOTA: em caso de rotação dupla, apresentar (desenhar) as duas rotações que lhe estão associadas.

G. [1.25 val] Heaps (Filas de Prioridade)

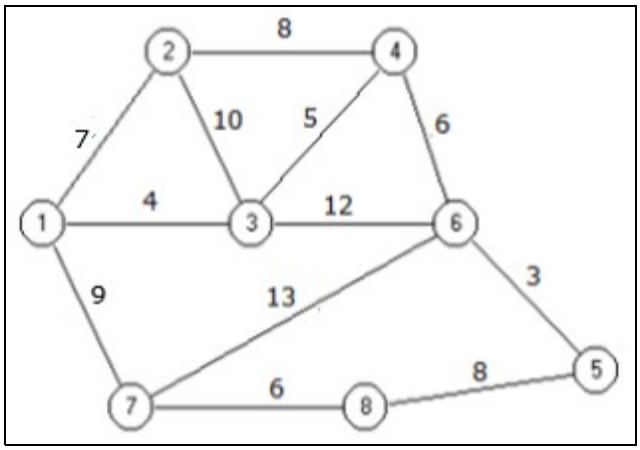
Considere o **minHeap binário** que se segue e responda às seguintes questões:



1. Simule a operação **remover** no minHeap dado (ao lado) e desenhe o minHeap obtido. Justifique, apresentando a evolução do minHeap desde a estrutura inicial até à final.
2. Simule a operação **inserir** um elemento com o valor **5** no minHeap dado (ao lado) e desenhe o minHeap obtido. Justifique, apresentando a evolução do minHeap desde a estrutura inicial até à final.

H. [2.00 val + 0.25 val] Grafos

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



1. Determine a **Árvore Abrangente Mínima (AAM)**, simulando o algoritmo de **PRIM** e tomando como **nó inicial o nó 7 (S = 7)**. Apresente os dados da simulação: evolução das estruturas de dados (rótulo **R**, **Pai**, **Permanentes** e **Temporários**) e de todos os valores de **k** e de **j** (usar esquema apresentado em baixo).
2. **Desenhe** a **AAM** e calcule o **custo** da **AAM** determinada em 1? Justifique usando os dados obtidos na simulação.

Esquema para apresentar os dados da simulação (questão 1):

	1	2	3	4	5	6	7	8
R								
Pai								

apresentar a evolução dos rótulos (R e Pai)

Temporários ←
Permanentes ←
k ← ?
  j ← …
…

Algoritmo de PRIM:

Passo 1.
  Tome-se arbitrariamente um nó S
  Atribuir ao nó S um rótulo permanente:
    $R_S \leftarrow 0$
    $Pai_S \leftarrow 0$
  Atribuir aos restantes nós rótulos temporários:
    $R_j \leftarrow C_{Sj}$, se (S, j) $\in$ A
    $Pai_j \leftarrow$ S , se (S, j) $\in$ A
    $R_j \leftarrow \infty$, se (S, j) $\notin$ A
    $Pai_j \leftarrow$ NULO , se (S, j) $\notin$ A
  Permanentes = { S }
  Temporários = N - { S }

Passo 2.
  k ← nó com rótulo temporário contendo o **menor valor**
  Temporários ← Temporários - { k }
  Permanentes ← Permanentes $\cup$ { k }
  (o arco (Pai_k, k) pertence à árvore abrangente mínima, com valor R_k)
  se (Temporários = $\varnothing$) **então**
    PARAR *(foi determinada uma árvore abrangente mínima)*
  fim_se
Passo 3.
  para (todo o **j** $\in$ **N** com (k, j) $\in$ A **e** j $\in$ Temporários) **repetir**
    se ($C_{kj} < R_j$) **então**
      $R_j \leftarrow C_{kj}$
      $Pai_j \leftarrow$ k
    fim_se
  fim_para
  regressar ao Passo 2