

Número mecanográfico:

Nome completo: \_\_\_\_\_

### Grupo 1 - Fundamentos de Java (4%)

1.1. Escreva pequenos **excertos de código** para cada uma das seguintes alíneas:

- a) Extrair o algarismo das centenas de um inteiro  $n$ :
- b) Criar um array  $v$  com espaço para  $k$  caracteres:
- c) Expressão booleana para saber se matriz  $m[][]$  tem menos que  $k$  linhas:
- d) Ir buscar a última letra de uma string  $s$ :

### Grupo 2 - Implementação de uma classe simples (10%)

2.1. Escreva **código para definir uma classe** `Matrix` que representa representa uma matriz de inteiros. Inclua atributos para representar a matriz em si, bem como o número de linhas e de colunas. Adicione também um construtor que recebe o número de linhas e colunas, bem como um inteiro  $k$  e cria a respetiva matriz (inicialmente com todas as posições preenchidas com  $k$ ).

2.2. Escreva um **método transpose()** da classe `Matrix` que devolva uma nova matriz que é a transposta da original (trocar linhas com colunas). Por exemplo, a figura seguinte mostra uma matriz e como fica a sua transposta:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

### Grupo 3 - Tipos Abstractos de Dados (4%)

3.1. Explique o que é um **interface** e quais são as diferenças em relação a uma **classe**.

*(pode usar esta página para a continuação de respostas dos grupos 1, 2 e 3 que necessitem de mais espaço)*

### Grupo 4 - Complexidade Algorítmica (23%)

4.1. Para os pares de funções seguintes, escreva na caixa em branco  $\Theta$ ,  $\mathcal{O}$  ou  $\Omega$  para tornar a afirmação verdadeira. Note que se as funções em causa tiverem uma relação  $\Theta$ , não deverá escrever  $\mathcal{O}$  ou  $\Omega$ .

$n^4 = \square (4^n)$     
  $2^n = \square (2^{n+2})$     
  $42 = \square 24$     
  $\log(\log n^2) = \square (\log n^2)$     
  $\sqrt{n} = \square (\log_4 n)$

4.2. Descreva o pior caso em termos de tempo de execução para os seguintes pedaços de código, usando notação  $\mathcal{O}$  para a variável  $n$ . Use o limite mais "apertado" possível (ex: indicar todos como sendo  $\mathcal{O}(n!)$  resultará em 0 pontos).

a)

```
for (int i=0; i<n; i++)
    for (int j=0; j<i; j++)
        count++;
```

d)

```
int f1(int n) {
    if (n <= 0) return 0;
    return f1(n-1) + 1;
}
```

b)

```
for (int i=n/2; i>0; i--)
    for (int j=0; j<100; j++)
        count++;
```

e)

```
int f2(int n) {
    if (n <= 0) return 0;
    int count = 0;
    for (int i=0; i<n; i++) count++;
    return count + f2(n/2) + f2(n/2)
}
```

c)

```
for (int i=1; i<n; i=i*2)
    count++;
```

4.3. Justifique a sua resposta para a alínea (e) da pergunta anterior, **escrevendo a recorrência e desenhando a respectiva árvore de recorrência**, indicando porque dá origem à complexidade indicada. Indique um algoritmo que conheça cuja complexidade seja determinada pela mesma recorrência.

#### 4.4. Previsão de Tempo de Execução.

Complete a tabela seguinte usando a informação já preenchida em cada linha. Nos tempos, use uma previsão baseada no tempo já preenchido, tendo em conta a proporção entre  $n_1$  e  $n_2$  (ms = milissegundos).

Programa	Complexidade	Nome Comum	Tempo para $n_1 = 10$	Tempo para $n_2 = 20$
A		constante	10ms	
B		linear		50ms
C	$\Theta(n^2)$		20ms	
D		cúbica		40ms
E	$\Theta(2^n)$		10ms	

4.5. Suponha que tem uma implementação de conjuntos de palavras usa como base um array (desordenado) onde são guardadas as palavras (uma por cada posição). Seja  $n$  o número de palavras do conjunto. Indique a **melhor complexidade temporal** (notação  $\mathcal{O}$ ) que conseguiria obter para os seguintes métodos (justificando sucintamente):

a) Devolver número de palavras no conjunto:

b) Verificar se uma palavra está no conjunto:

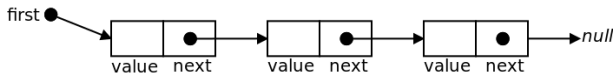
4.6. Explique como poderia obter uma **implementação mais eficiente para a verificar se a palavra está no conjunto**, indicando a nova complexidade e outras potenciais vantagens e desvantagens dessa outra implementação quando comparada com o array desordenado (ex: mais algum método é melhor ou pior?)

---

*(pode usar o resto desta página para a continuação de respostas do grupo 4 que necessitem de mais espaço)*

**Grupo 5 - Listas, Pilhas e Filas (20%)**

5.1. Considere uma classe `SinglyLinkedList<T>` representando uma lista ligada simples genérica tal como dada nas aulas com atributos `size` e `first`, e uma classe `Node<T>` representando um nó com atributos `value` e `next`.



a) Implemente um **método** `contains(x)` da classe `SinglyLinkedList<T>` devolve `true` se o elemento  $x$  está na lista e `false` caso contrário. Não pode usar qualquer outro método já existente da classe. Indique a **complexidade temporal** do método que descreveu.

b) Implemente um **método** `remove(i)` da classe `SinglyLinkedList<T>` que remove o  $i$ -ésimo elemento da lista (elemento na posição  $i$ ). As posições começam em 0. Se a posição  $i$  não existir, o método não deve fazer nada. Não pode usar qualquer outro método já existente da classe. Indique a **complexidade temporal** do método que descreveu.

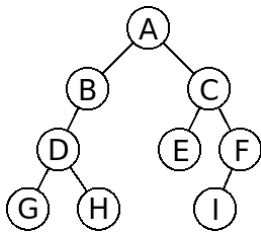
5.2. Indique, justificando, uma situação onde **faz mais sentido usar uma lista ligada** ao invés de um simples array.

5.3. Explique (não é preciso código) como **inverter o conteúdo de uma pilha**, sendo que para interagir com a pilha apenas pode usar a sua API (`push`, `pop` e `size`). Como estruturas de dados auxiliares apenas pode usar pilhas ou filas (também apenas através da API). Indique a **complexidade temporal** do método que descreveu.

*(pode usar esta página para a continuação de respostas do grupo 5 que necessitem de mais espaço)*

**Grupo 6 - Árvores (16%)**

6.1. Considerando a **árvore da figura** seguinte, responda às seguintes alíneas:



- a) Os nós da árvore em **preorder**:
- b) Os nós da árvore em **inorder**:
- c) Os nós numa **pesquisa em largura**:

6.2. Considere uma classe `BTree<T>` de uma árvore binária genérica tal como dada nas aulas com um atributo `root` a apontar para a raiz que é um `BNode<T>` representando um nó com atributos `value`, `left` e `right`.

a) Implemente um **método** `depth()` da classe `BTree<T>` que devolve a profundidade da árvore. Não pode usar qualquer outro método já existente da classe. Indique a **complexidade temporal** do método que descreveu.

b) Implemente um **método** `level(x)` da classe `BTree<T>` que devolve o número de nós da árvore que estão a profundidade `x`. Não pode usar qualquer outro método já existente da classe. Indique a **complexidade temporal** do método que descreveu.

**Grupo 7 - Árvores Binárias de Pesquisa (15%)**

7.1. Vamos inserir (por esta ordem) os seguintes números numa árvore binária de pesquisa: 42, 20, 24, 50, 53, 11. Para as 3 alíneas de baixo **desenhe a árvore** correspondente (depois de completa a operação respetiva):

a) Como fica a **árvore original** depois de inseridos os 6 números?

b) Como fica a árvore (original) depois de **adicionarmos um 51**?

c) Como fica a árvore (original) depois de **removermos o 42**?

7.2. Em que nó fica guardado o **elemento mínimo** numa árvore binária de pesquisa? Justifique sucintamente.

7.3. Suponha que quer inserir os 15 inteiros de 1 a 15 numa árvore binária de pesquisa. Qual a **mínima altura possível** da árvore? Indique por **qual ordem deve inserir os elementos** para garantir essa altura mínima.

Altura mínima:  Ordem de Inserção:

7.4. Explique (por palavras) um algoritmo para **procurar um elemento numa árvore binária de pesquisa**. Indique, justificando, a **complexidade temporal** do método que descreveu supondo que a árvore está equilibrada.

### Grupo 8 - Filas de Prioridade e Heaps (8%)

8.1. Imagine que tem uma **minHeap** descrita pelo seguinte array 

3	4	5	8	7	10
---	---	---	---	---	----

Para as 3 alíneas de baixo **desenhe a árvore** (a heap) correspondente (sempre já com o invariante reposto):

a) Qual a heap representada pelo **array original**?

b) Como fica a heap (original) depois de **adicionarmos um 2**?

c) Como fica a heap (original) depois de **removermos o mínimo**?

8.2. Suponha que lê um conjunto de  $n$  números, inserindo-os um a um numa fila de prioridade implementada com uma *minHeap*. Depois retira-os um a um da heap e vai escrevendo os valores à medida que vão sendo removidos.

a) Qual a **ordem** em que aparecem os números no output?

b) Indique, justificando, a **complexidade temporal e espacial** (memória) de todo o processo descrito.

*(pode usar o resto desta página para a continuação de respostas dos grupos 6, 7 e 8 que necessitem de mais espaço)*