

**Justifique todas as suas respostas.**

Nome: *André Almeida Oliveira*  
 N. Mec.: *107637*

**Fórmulas:**

- $\sum_{k=1}^n 1 = n$
- $\sum_{k=1}^n k = \frac{n(n+1)}{2}$
- $\sum_{k=1}^n k^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{k=1}^n k^3 = \left(\frac{n(n+1)}{2}\right)^2$
- $\sum_{k=1}^n \frac{1}{k} \approx \log n$
- $n! \approx n^n e^{-n} \sqrt{2\pi n}$

**3.0** **1:** No seguinte código,

```
#include <stdio.h>

int f(int x) { return 2 * x + 3; }
int g(int x) { return x * x - 7; }

int main(void)
{
    for(int i = -5; i <= 5; i++)
        if( (f(i) > 0) || (g(i) > 0) )
            printf("%d\n", i);
    return 0;
}
```

1.5 a) para que valores da variável *i* é avaliada a função *g(x)*? *-5, -4, -3 e -2*

1.5 b) que valores de *i* são impressos? *-5, -4, -3, -1, 0, 1, 2, 3, 4 e 5*

**1.5** **2:** No seguinte código,

```
int a[10], *b = &a[7];
```

qual é o índice do elemento do array *a* que é referenciado por *b[-4]*? *b[-4] = a[7-4] = a[3]*

**4.0** **3:** A complexidade computacional de muitos algoritmos é expressa usando a notação “big Oh” (**O**) em vez da notação “Big Theta” (**Θ**). Porquê? (Nota: dois terços da cotação para uma boa explicação das duas notações, um terço para uma boa explicação do porquê.)

**3.0** **4:** Ordene as seguintes funções por ordem crescente de ritmo de crescimento. Responda nesta folha, usando o número das funções na sua resposta.

Número da função	função	termo dominante	ordem
1	$\frac{n!}{n^{100}} - 1$	$\frac{n!}{n^{100}}$	5
2	$n \log n + \sqrt{n}$	$n \log n$	2
3	$1.2^n + 17 + n^3$	$1.2^n$	4
4	$23 + \frac{\log n}{n}$	$\frac{\log n}{n}$	1
5	$n^4 + \frac{1000}{n}$	$n^4$	3

**Resposta:**

3.0 **5:** Para a seguinte função,

```
int f(int x)
{
    int i,j,r = 0;

    for(i = 0; i <= x; i++)
        for(j = i; j >= 0; j--)
            r += i - j;
    return r;
}
```

1.5 a) quantas vezes é executada a linha `r += i - j;`?  $\frac{x^2 + x + 2}{2}$

1.5 b) que valor é devolvido pela função?  $\sum_{i=0}^x \sum_{j=0}^i (i-j)$

4.0 **6:** O seguinte trecho de código reserva espaço para uma matriz com  $n$  linhas com uma determinada forma. Não é reservado espaço para os elementos da matriz fora dessa forma.

*mais não fazer*

```
int n;    // the number of rows of the matrix
int **a;  // the matrix
```

```
void init_a(void)
{
```

```
    int i,k,s,*p; // auxiliary variables
```

```
    // the total number of elements of the matrix
    s = ;
```

```
    // allocate memory for the array of pointers
```

```
    a = (int **)malloc((size_t)n * sizeof(int *));
```

```
    // the memory for ALL elements
```

```
    p = (int *)malloc((size_t)s * sizeof(int));
```

```
    for(i = 0; i < n; i++)
```

```
    {
```

```
        // the number of valid elements on the i-th line
```

```
        k = 2 * i + 1;
```

```
        // the pointer for the i-th line; this line uses p[0], p[1], ..., p[k-1];
```

```
        // the remaining elements of this line will never be used by a correct program
```

```
        a[i] = p - (n - 1) + i;
```

```
        // advance p
```

```
        p += k;
```

```
    }
```

```
}
```

1.5 a) Calcule o valor a dar à variável `s` de modo a que seja alocado o número exato de elementos da matriz.

1.5 b) Num acesso à matriz usando `a[i][j]`, qual é a gama de valores válidos para `j`?

1.0 c) Qual é a forma da matriz?

1.5 **7:** Dê um exemplo de uma função que tenha uma complexidade computacional de  $\Theta(n^2)$ .

- 3 - A notação "big Oh" é usada para expressar o limite superior da complexidade de um algoritmo, ou seja, o pior caso do mesmo. A notação "Big Theta" usa a média da complexidade do algoritmo, ou seja, generalidade da caso. A primeira é mais usada pois, sabendo o limite superior da complexidade de um algoritmo, é útil para comparar com outros algoritmos, escolhendo o mais adequado a usar.

5 - a) 
$$\sum_{i=0}^x \sum_{j=0}^i 1 = \sum_{i=0}^x \sum_{j=1}^{i+1} 1 = \sum_{i=0}^x (i+1) = \sum_{i=1}^{x+1} i = \frac{(x+1)(x+2)}{2} = \frac{x^2 + 3x + 2}{2}$$

7 - 

```
for (int i = 0; i < m; i++)  
    for (int j = 0; j < m; j++)  
        printf("da");
```

Nome: *André Almeida Oliveira*

N. Mec.: 107637

---

Nas perguntas sobre árvores binárias, cada nó da árvore usa a seguinte estrutura de dados:

```
typedef struct tree_node
{
    struct tree_node *left;    // pointer to the left branch (a sub-tree)
    struct tree_node *right;   // pointer to the right branch (a sub-tree)
    struct tree_node *parent;  // pointer to the parent node (NULL for the root node)
    int data;                  // the data (only one node can have a given data value)
}
```

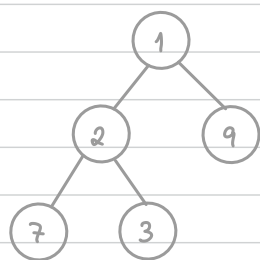
---

```
tree_node;
```

- 2.0 **1:** Escreva uma função recursiva que, dada a raiz de uma árvore binária **não ordenada**, e um valor  $v$ , conta o número de nós da árvore que armazenam valores menores ou iguais a  $v$ . Qual é a complexidade computacional da sua função?
- 3.0 **2:** Escreva uma função recursiva **eficiente** que, dada a raiz de uma árvore binária **ordenada**, e um valor  $v$ , conta o número de nós da árvore que armazenam valores menores ou iguais a  $v$ . Qual é a complexidade computacional da sua função?
- 2.0 **3:** Explique como está organizada a informação num *min-heap*. Ilustre a sua exposição inserindo os números, por esta ordem, num *min-heap* inicialmente vazio: **7, 3, 9, 1, 2**.
- 3.0 **4:** É possível implementar eficientemente uma fila (*queue*) usando uma lista simplesmente ligada. Como?
- 2.0 **5:** Dos vários algoritmos de ordenação que conhece, existem alguns que funcionam naturalmente de uma forma recursiva. Explique o funcionamento de um deles (recursivo!).
- 2.0 **6:** Compare dois algoritmos de ordenação à sua escolha no que diz respeito a i) complexidade computacional, ii) melhor caso, iii) pior caso.
- 3.0 **7:** Explique como pode procurar informação numa lista biligada. Explique também como pode tornar a procura mais eficiente quando a informação de que se está à procura não estiver uniformemente distribuída.
- 3.0 **8:** Explique como funciona uma *hash table*. Indique as vantagens e desvantagens das implementações de *hash tables* usando *open-addressing* e *chaining*.

- 3 - Um min - heap é uma árvore binária em que cada nó é menor que a sua soma.

7			
7	3		
3	7		
3	7	9	
3	7	9	1
1	3	9	7
1	3	9	7 2
1	2	9	7 3



- 4 - Uma queue é uma estrutura de dados do tipo FIFO (first in, first out). Para ela ser implementada eficientemente por uma linked list teremos que usar dois ponteiros, um aponta para o primeiro elemento da linked list (head) e outro aponta para o último (tail). Para adicionar um elemento, usaremos o ponteiro tail para adicionar o elemento no fim e para remover um elemento, usaremos o ponteiro head para remover o elemento de início.

- 5 - Um exemplo é o quick sort que escolhe um pivot e divide o array em dois arrays, um com valores maiores que o pivot e outro com menores. Fazemos o processo anterior recursivamente até obtermos apenas um elemento e, de seguida, juntamos tudo de forma a formar um array ordenado.

- 6 - O algoritmo merge sort tem uma complexidade computacional  $O(n \log n)$ , o seu melhor caso é a ordenação de um array que está na ordem inversa e o melhor caso é de um que já está ordenado.

O algoritmo quick sort tem uma complexidade computacional  $O(n^2)$ , o seu melhor caso é a ordenação de um array que está na ordem inversa e o melhor caso é de um que já está ordenado.

- 7 - Sendo a lista não ordenada, teremos de percorrer todos os elementos da lista e, para cada um, avaliar se é igual ao pretendido ou não. Este algoritmo tem complexidade  $O(n)$ , pois no fim da conta terá que percorrer a lista toda.

Uma maneira de tornar esta procura mais eficiente, é colocar a informação com maior procura no início da lista.

- 8 - Uma hash table, é uma estrutura de dados que armazena pares key-value e permite o acesso rápido aos valores a partir das chaves. Estas chaves são obtidas através de hash functions que, tomando um input, retornam um valor correspondente que, em princípio, será o índice onde o valor será armazenado na hash table. Existem duas formas de implementar hash tables. A primeira é o método de direct addressing em que todos os elementos são armazenados diretamente na hash table, ou seja, quando existem colisões será necessário encontrar um novo índice para o valor (geralmente fica no índice mais próximo vazio). Uma vantagem é que é mais rápido acessar os elementos, mas uma desvantagem é que a memória alocada pode não ser suficiente. O segundo método é o separate chaining em que os elementos são armazenados em linked lists, ou seja, colisões já não são um problema. Uma vantagem é que não há limite de elementos armazenados, mas uma desvantagem é que o acesso a elementos é mais lento.