

Nome: André Almeida Oliveira

N. Mec.: 107637

3.0 **1:** No seguinte código,

```
#include <stdio.h>
```

```
int f(int x) { return x % 3; }
int g(int x) { return x % 2; }
```

```
int main(void)
{
    int c = 0;
    for(int i = -1000; i <= 1000; i++)
        if( f(i) || g(i) )
        {
            printf("i = %d\n", i);
            (i > 0) && c++;
        }
    printf("c = %d\n", c);
}
```

- 1.0 a) para que valores da variável  $i$  é avaliada a função  $g(x)$ ?
- 1.0 b) que valores de  $i$  são impressos?
- 1.0 c) que valor de  $c$  é impresso?

Respostas:

- a) A função  $g(x)$  apenas é avaliada quando  $f(x)$  é falso (retorna 0), logo quando  $i$  é múltiplo de 3.
- b) Serão impressos todos os valores que não são múltiplos de 2 ou que não são múltiplos de 3 ou que não são múltiplos de nenhum.
- c) Entre 1 e 1000 há 333 múltiplos de 3, dos quais 166 são múltiplos de 2, logo  $c = 1000 - 166 = 834$ .

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 **2:** Ordene as seguintes funções por ordem crescente de ritmo de crescimento. Responda nas duas colunas da direita da tabela. Na coluna da ordem, coloque o número 1 na função com o ritmo de crescimento menor (e, obviamente, coloque o número 5 na com o ritmo de crescimento maior).

Número da função	função	termo dominante	ordem
1	$n^{99} + 1.1^n$	$1.1^n$	3
2	$\frac{n!}{42^n}$	$\frac{n!}{42^n}$	5
3	$n^2 \log n^{99} + n^2 \sqrt[3]{n}$	$n^2 \sqrt[3]{n}$	1
4	$1.2^n + n^2$	$1.2^n$	4
5	$\sum_{k=1}^n (k^2 + k)$	$\sum_{k=1}^n k^2 = \frac{n(n+1)(2n+1)}{6}$	2

- 4.0 **3:** Um programador inexperiente escreveu a seguinte função para copiar uma zona de memória com `size` bytes que começa no endereço `src` para uma outra zona de memória que começa no endereço `dest`.

```
void mem_copy(char *src, char *dest, size_t size)
{
    for(size_t i = 0; i < size; i++)
        dest[i] = src[i];
}
```

Responda às seguintes perguntas, considerando que para cada uma das duas primeiras o conteúdo **inicial** do array `c` é `char c[10] = { 0,1,2,3,4,5,6,7,8,9 }`;

- 1.3 a) qual o conteúdo do array `c` depois de `mem_copy(&c[3], &c[5], 4)`; ter sido executado?

Resposta:

0	1	2	3	4	3	4	3	4	9
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]

- 1.3 b) qual o conteúdo do array `c` depois de `mem_copy(&c[5], &c[3], 4)`; ter sido executado?

Resposta:

0	1	2	5	6	7	8	7	8	9
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]

- 1.4 c) num dos casos anteriores a cópia do conteúdo de parte do array não foi feita corretamente; sugira uma maneira de corrigir este problema (não é obrigatório escrever código).

Resposta:

```
void mem_copy(char *src, char *dest, size_t size)
{
    for (size_t i = size-1; i >= 0; i--)
        dest[i] = src[i];
}
```

- 3.0 **4:** A notação “big Oh” é usualmente usada para descrever a complexidade computacional do pior caso de um algoritmo. Porquê?

Resposta (tente não exceder as 100 palavras):

A notação big Oh é usada para indicar o número máximo de instruções que o computador vai ter que executar para completar o código.  
Entendendo este valor, podemos saber uma ideia do tempo máximo que o programa vai demorar, pois o tempo é diretamente proporcional à quantidade de instruções.

- 2.0 **5:** Escreva o código de uma função que tenha uma complexidade computacional de  $\Theta(\sqrt{n})$ . Como alternativa, pode optar por escrever o código de uma função de tenha uma complexidade computacional de  $\Theta(\log n)$ . (Pode usar pseudo-código, se bem que uma função em C será mais valorizada.)

Resposta:

```
for (int i = 0; i^2 < n; i++)  
    printf("%d\n", i);
```

5.0 **6:** Para a seguinte função,

```
int f(int n)
{
    int r = -1;

    for(int i = -2; i <= n; i++)
        for(int j = i; j >= -3; j--)
            r += i - j;
    return r;
}
```

$$\sum_{i=-2}^n \sum_{j=-3}^i 1 = \sum_{i=-2}^n \sum_{j=1}^{i+4} 1 = \sum_{i=1}^{n+3} (i+4-3) = \sum_{i=1}^{n+3} (i+1)$$

$$= \sum_{i=1}^{n+3} i + \sum_{i=1}^{n+3} 1 = \frac{(n+3)(n+4)}{2} + n+3$$

=

- 2.0 a) quantas vezes é executada a linha `r += i - j`?  $\frac{n^2}{2} + \frac{9}{2}n + 9$
- 2.0 b) que valor é devolvido pela função?  $-1 + \sum_{i=-2}^n \sum_{j=-3}^i (i \cdot j)$
- 1.0 c) qual é a complexidade computacional da função?  $O(n^2)$

Respostas:

a)  $\sum_{i=-2}^n \sum_{j=-3}^i 1 = \sum_{i=-2}^n \sum_{j=1}^{i+4} 1 = \sum_{i=-2}^n (i+4) = \sum_{i=1}^{n+3} (i-3+4) = \sum_{i=1}^{n+3} (i+1) = \sum_{i=1}^{n+3} i + \sum_{i=1}^{n+3} 1 = \frac{(n+3)(n+4)}{2} + n+3 =$

$$= \frac{n^2 + 4n + 3n + 12}{2} + n + 3 = \frac{n^2}{2} + \frac{7}{2}n + 6 + n + 3 = \frac{n^2}{2} + \frac{9}{2}n + 9$$

## Segundo teste de Algoritmos e Estruturas de Dados

18 de Novembro de 2019

14h10m – 15h00m

Responda a todas as perguntas no enunciado do teste. Justifique todas as suas respostas.

O teste é composto por 5 grupos de perguntas.

Nome: André Almeida Oliveira

N. Mec.: 107637

- 4.0 **1:** Pretende-se que a seguinte função implemente uma pesquisa binária. Complete-a (isto é, preencha as caixas).

```
int binary_search(int n,int a[n],int v)
{
    int low = ;
    int high = ;
    while(high  low)
    {
        int middle = ;
        if(a[middle] == v)
            return middle;
        if(a[middle]  v)
             = middle - 1;
        else
             = middle + 1;
    }
    return ;
}
```

Indique (não é preciso justificar) qual é a complexidade computacional desta função.

Resposta:  $O(\log n)$

- 4.0 **2:** Explique como está organizado um *max-heap*. Para o *max-heap* apresentado a seguir, insira o número 8. Não apresente apenas o resultado final; mostre, passo a passo, o que acontece ao *array* durante a inserção. Em cada linha, basta escrever as entradas do *array* que foram alteradas.

Respostas:

9	6	7	3	1	4	5	2	8
			8					3
	8		6					
9	8	7	6	1	4	5	2	3
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

- 3.0 **3:** Pretende-se implementar uma fila (*queue*) usando um *array* circular. O código seguinte define a estrutura de dados a usar (para simplificar, vamos usar variáveis globais).

```
#define array_size 1024

int array[array_size];
int read_pos = 1; // incremented after reading
int write_pos = 0; // incremented before writing
int count = 0;    // equal to (read_pos - write_pos - 1) % array_size
```

Responda às seguintes perguntas:

- 1.5 a) Por que é que neste caso é vantajoso usar um *array* circular?

Resposta:

Uma queue é uma estrutura do tipo FIFO (first in, first out) onde o ponteiro read - pos aponta para o primeiro elemento e write - pos para o último. Usar um array circular é vantajoso, pois poderíamos simplesmente aceder ao índice de um elemento que queremos remover em vez de percorrer uma lista ligada por inteiro.

- 1.5 b) Use algumas das seguintes linhas de código para implementar a função `enqueue` (que coloca um item de informação na fila). Risque as linhas que estão a mais.

```
int enqueue(int v)
void enqueue(int v)
{
if(count == 0) exit(1); // underflow
if(count == array_size) exit(1); // overflow
array[write_pos] = v;
v = array[write_pos];
write_pos = (write_pos + 1 == array_size) ? 0 : write_pos + 1;
write_pos = (write_pos > 0) ? write_pos - 1 : array_size - 1;
array[write_pos] = v;
v = array[write_pos];
count--;
count++;
return v;
}
```

- 5.0 4: Um programador pretende utilizar uma *hash table* (tabela de dispersão, dicionário) para contar o número de ocorrências de palavras num ficheiro de texto. O programador está à espera que o ficheiro tenha cerca de 6000 palavras distintas, pelo que usou uma *hash table* do tipo *separate chaining* com 10007 entradas, e usou a seguinte *hash function*:

```
unsigned int hash_function(unsigned char *s, unsigned int hash_table_size)
{
    unsigned int sum;

    for(sum = 0; *s != '\0'; s++)
        sum += (unsigned int)(*s);
    return sum % hash_table_size;
}
```

Infelizmente, as expetativas do programador estavam erradas, e o ficheiro de texto era muito maior que o esperado, tendo cerca de 1000000 palavras distintas. Responda às seguintes perguntas:

- 2.0 a) A *hash function* apresentada acima é muito má. Porquê? Sugira uma outra que seja bem melhor.
- 3.0 b) Uma implementação do tipo *separate chaining* usa habitualmente uma lista ligada para armazenar todas as chaves (neste caso, as palavras) para as quais a *hash function* tem o mesmo valor. Que vantagens/desvantagens teria uma implementação que usa uma árvore binária ordenada em vez da lista ligada? E se for uma árvore binária ordenada e balanceada?

Respostas:

a) Por um lado, esta função não retorna valores uniformes pela *hash table*, pois apenas retorna uma gama bastante baixa de valores. Por outro lado, não é capaz de distinguir anagramas.

b) A busca, inserção e remoção de elementos seria mais eficiente em termos de tempo, pois a árvore binária ordenada é uma estrutura de dados auto-balanceada e tem uma complexidade  $O(\log n)$ , enquanto a lista ligada tem complexidade  $O(n)$ . Também ocuparia menos espaço, pois as árvores binárias têm menos overhead de dados do que as listas ligadas.

A implementação seria mais complexa, pois é necessário manter a árvore binária ordenada, enquanto a lista ligada é mais simples de implementar.

Se a árvore binária for balanceada, pode melhorar ainda mais a eficiência da busca, inserção e remoção de elementos, pois o tempo de acesso seria garantido  $O(\log n)$  independentemente da estrutura de dados. No entanto, a manutenção de uma árvore balanceada é mais complexa do que uma árvore binária ordenada não balanceada.

4.0 **5:** Apresentam-se a seguir várias funções que visitam todos os nós de uma árvore binária, e mostram-se várias ordens pelas quais a função `visit` foi chamada para cada um dos nós (1 significa que o nó correspondente foi o primeiro a chamar a função `visit`, 2 que foi o segundo, e assim por diante). Para cada uma das ordens apresentadas, indique que função, ou funções, deram origem a essa ordem.

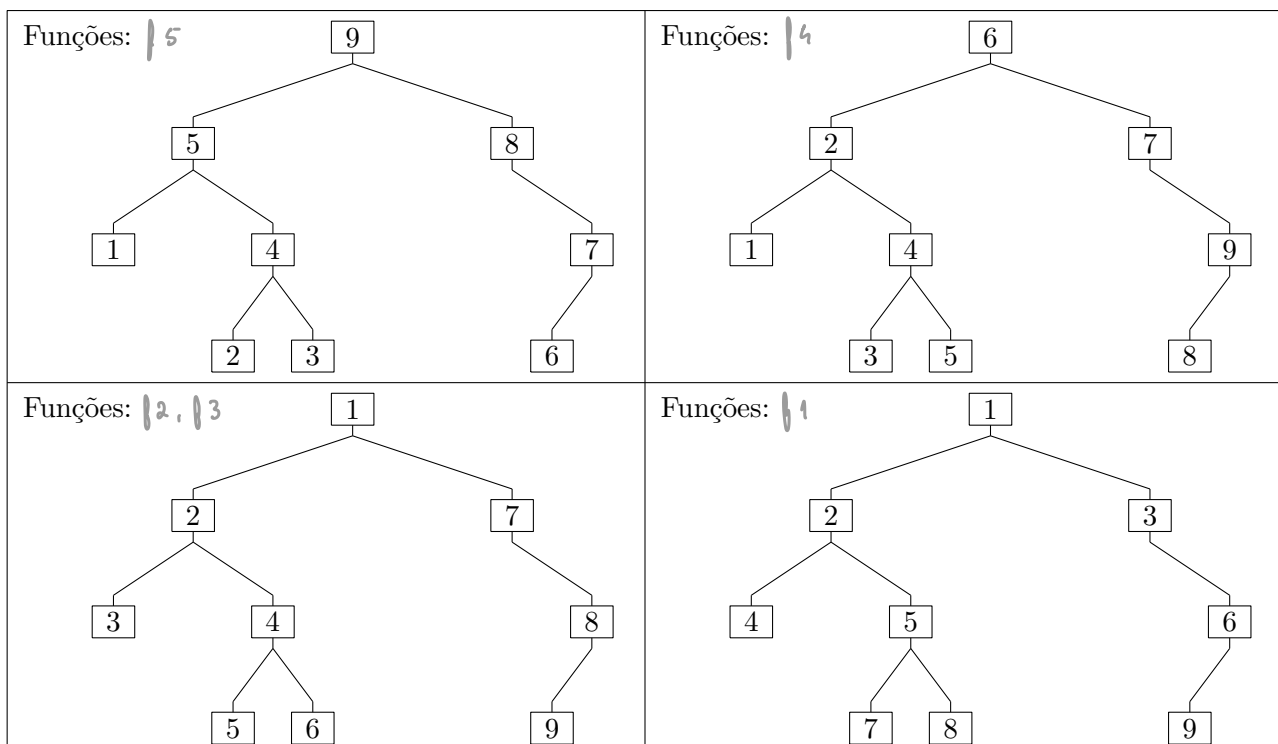
```
void f1(tree_node *link) breadth_search
{
    queue *q = new_queue();
    enqueue(q, link);
    while(is_empty(q) == 0)
    {
        link = dequeue(q);
        if(link != NULL)
        {
            visit(link);
            enqueue(q, link->left);
            enqueue(q, link->right);
        }
    }
    free_queue(q);
}
```

```
depth_search
void f2(tree_node *link)
{
    stack *s = new_stack();
    push(s, link);
    while(is_empty(s) == 0)
    {
        link = pop(s);
        if(link != NULL)
        {
            visit(link);
            push(s, link->right);
            push(s, link->left);
        }
    }
    free_stack(s);
}
```

```
void f3(tree_node *link)
{
    if(link != NULL)
    {
        visit(link);
        f3(link->left);
        f3(link->right);
    }
} pre-order
```

```
void f4(tree_node *link)
{
    if(link != NULL)
    {
        f4(link->left);
        visit(link);
        f4(link->right);
    }
} in-order
```

```
void f5(tree_node *link)
{
    if(link != NULL)
    {
        f5(link->left);
        f5(link->right);
        visit(link);
    }
} post-order
```





## Terceiro teste de Algoritmos e Estruturas de Dados

9 de Dezembro de 2019

14h10m – 15h00m

Responda a todas as perguntas no enunciado do teste. Justifique todas as suas respostas.  
O teste é composto por 5 grupos de perguntas.

Nome: André Almeida Oliveira

N. Mec.: 107637

4.0 **1:** O algoritmo *merge sort* divide o *array* a ser ordenado ao meio, ordena (recursivamente) cada uma das duas partes, e depois junta-as. A sua complexidade computacional é  $\Theta(n \log n)$ . Um aluno está convencido que se em vez de se dividir o *array* em duas partes se se dividir em três partes (todas mais ou menos do mesmo tamanho), então a complexidade computacional desta variante do *merge sort* será ainda mais baixa. Responda às seguintes perguntas:

- 1.0 a) Que estratégia algorítmica usa o *merge sort*? *Divide and Conquer*
- 2.0 b) O aluno tem razão? Justifique.
- 1.0 c) Indique uma desvantagem do *merge sort*, quando comparado com o *quicksort*.

Respostas:

b)  $a = 3 \rightarrow$  operações na 3 arrays

$b = 3 \rightarrow$  divisão do array em 3 partes

$$T(n) = O(n^{\log_3 3} \log n) = O(n \log n), \text{ pois } \log_3 3 = 1$$

O aluno não tem razão, pois a complexidade mantém-se

c) O *merge sort* precisa de mais memória, como por exemplo, um buffer que é usado para a união de 2 subarrays

O *master theorem* afirma que se  $T(n) = aT(n/b) + f(n)$  então

- se  $f(n) = O(n^{\log_b a - \epsilon})$  para um  $\epsilon > 0$ , então  $T(n) = \Theta(n^{\log_b a})$ ,
- se  $f(n) = \Theta(n^{\log_b a})$ , então  $T(n) = O(n^{\log_b a} \log n)$ ,
- se  $f(n) = \Omega(n^{\log_b a + \epsilon})$  para um  $\epsilon > 0$  e se  $af(\frac{n}{b}) \leq cf(n)$  para  $c < 1$  e  $n$  suficientemente grande, então  $T(n) = \Theta(f(n))$ .

- 6.0 **2:** Num tabuleiro de xadrez, pretende-se ir do canto inferior esquerdo  $(0, 0)$  para o canto superior direito  $(7, 7)$  fazendo movimentos apenas para a direita e para cima. Quando se está em  $(x, y)$ , o custo de ir para a direita é dado por  $R_{xy}$  e o custo de ir para cima é dado por  $U_{xy}$ . O custo total é a soma dos custos dos 14 movimentos efetuados (7 para a direita e 7 para cima). Sabe-se que  $4 \leq R_{xy} \leq 10$  e que  $6 \leq U_{xy} \leq 20$ . No programa que foi usado para calcular o **custo mínimo** para ir de  $(0, 0)$  até  $(7, 7)$ , os valores de  $R_{xy}$  e  $U_{xy}$  estão guardados nas matrizes

```
int R[7][8], U[8][7]; // initialized elsewhere
```

Responda às seguintes perguntas:

- 3.0 a) Complete o seguinte código, que resolve o problema usando a técnica *branch-and-bound*.

*min dumb*

```
void go_to(int x,int y,int partial_cost,int *min_cost) {  
    if(partial_cost +  * (7 - x) + 6 * ()  *min_cost) return;  
    if(x < 7) go_to(x + 1,y,partial_cost + ,min_cost);  
    if(y < 7) go_to(x,y + 1,partial_cost + ,min_cost);  
    if(x == 7 && y == 7) *min_cost = partial_cost;  
}  
  
int compute_min_cost_bb(void) {  
    int min_cost = ; go_to(0,0,,&min_cost); return min_cost; }
```

- 3.0 b) Pretende-se também resolver este problema usando programação dinâmica. Para isso, o custo mínimo para ir de  $(0, 0)$  até  $(x, y)$  é guardado na matriz

```
int C[8][8];
```

Complete o seguinte código. (No fim, estamos interessados no valor de  $C[7][7]$ , mas, para o calcular, dá jeito conhecer os outros valores.)

```
int update_C(int x,int y,int C[8][8]) {  
    if(x  0 || y < 0) return 1000000000;  
    if(C[x][y]  0)  
    {  
        int Cx = update_C(x - 1,y,C) + ;  
        int Cy = update_C(x,y - 1,C) + ;  
        C[x][y] = (Cx < Cy) ? Cx : Cy;  
    }  
    return C[x][y]; }  
  
int compute_min_cost_dp(void) {  
    int C[8][8]; // -1 means not yet known  
    for(int x = 0;x < 8;x++)  
        for(int y = 0;y < ;y++)  
            C[x][y] = (x == 0 && y == 0) ? 0 : ;  
    update_C(,,C);  
    return C[7][7]; }
```

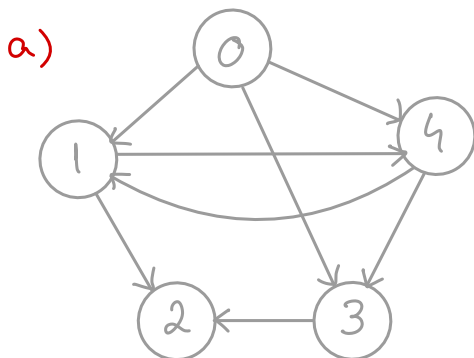
3.0 **3:** Os 5 vértices, numerados de 0 a 4, de um grafo têm as seguintes listas de adjacência:

0	→ 1 → 3 → 4 → NULL
1	→ 4 → 2 → NULL
2	→ NULL
3	→ 2 → NULL
4	→ 3 → 1 → NULL

Responda às seguintes perguntas:

- 1.0 a) Desenhe o grafo.
- 1.0 b) Represente o grafo usando uma matriz de adjacência.
- 1.0 c) É possível representar este grafo usando um único inteiro de **32 bits**? Se sim, como? Se não, por que não?

Respostas:



b)

$$\begin{matrix}
 & 0 & 1 & 2 & 3 & 4 \\
 \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}
 \end{matrix}$$

b) Será possível, pois no máximo temos  $5 \times 5 = 25$  arestas, logo 25 bits necessários que é menor que 32

3.0 **4:** Explique para que serve e como funciona o algoritmo *union find*.

O algoritmo union find pode encontrar a qual conjunto um certo elemento pertence e unir conjuntos existentes. Este algoritmo é altamente utilizado em problemas relacionados com grafos, como por exemplo, unir dois componentes conexos.

- 4.0 **5:** Considere um labirinto desenhado na superfície de uma esfera. Pretende-se ir, a andar, do pólo norte até ao pólo sul. (Considere que a esfera tem um raio relativamente pequeno, pelo que ir a pé não demora meses, mas pode demorar dias.) Responda às seguintes perguntas:
- 1.0 a) Que algoritmo usaria para encontrar uma solução? *depth first search*
- 1.5 b) Que material levaria consigo para o ajudar?
- 1.5 c) Acha que é possível encontrar a solução mais curta de uma forma eficiente? (Não se esqueça que tem de fazer todo o caminho a pé.) Porquê?

Answers:

*b) algum objeto para marcar a rota por onde já tenha passado*

*c) Não, a única maneira "mais eficiente" neste caso seria usar breadth first search, mas teríamos de percorrer uma distância muito grande. Se já conhecessemos o labirinto na totalidade, podíamos usar o algoritmo dijkstra calculando o caminho mais curto*