

Nome: 

N. Mec.: \_\_\_\_\_

- 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 cinco 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) Nesta variante do *merge sort*, a fase de *merge* é mais fácil ou mais complicada que a do algoritmo original?

b) Não, a complexidade é baseada no número de operações para dividir, ordenar e fundir  $x$  listas. Embora seja mais rápido ordenar a sublista, fazer a junção de 5 listas será um processo mais demorado.

c) Mais complicada, pois haverá um maior número de listas a juntar

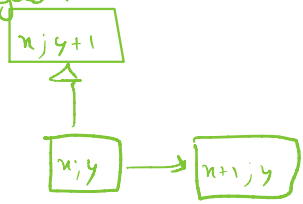
$$\begin{aligned}
 a &= 5 \\
 b &= 5 \\
 T_m &= O\left(m^{\log_5 5} \log(m)\right) \\
 &= O(\log(m))
 \end{aligned}$$

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  $T(n) = \Theta(f(n))$ .

- 4.0 **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. O seguinte código apresenta uma maneira de calcular o número de maneiras de fazer isso.

```
1  long eval(int x,int y)
   {
2    if(x < 0 || x > 7 || y < 0 || y > 7) { ver se a posição está fora do
3      return 0L;                          tabuleiro
4    else if(x == 7 && y == 7) { se chegarmos à posição desejada return
3      return 1L;
4    else
4      return eval(x + 1,y) + eval(x,y + 1); → avançar uma
                                           casa
   }
5  long count_paths(void)
   {
6    return eval(0,0); → calcular a partir de (0;0)
   }
```



Responda às seguintes perguntas:

- 1.0 a) Que estratégia algorítmica é usada por este código? *Programação dinâmica*
- 3.0 b) Explique por palavras qual é o objetivo de cada uma das partes numeradas do código.

- 4.0 **3:** Tal como no problema anterior, 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. O seguinte código apresenta uma maneira de calcular o número de maneiras de fazer isso.

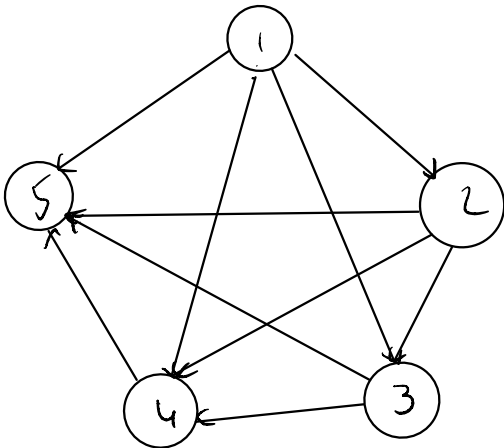
```
1  long count_data[8][8];
2  long eval(int x,int y)
   {
3    if(x < 0 || x > 7 || y < 0 || y > 7) { ver se excede os limites do tabuleiro
3      return 0L;
4    if(count_data[x][y] < 0L)
4      count_data[x][y] = eval(x - 1,y) + eval(x,y - 1); { Se ainda não tiver sido
5    return count_data[x][y];                             calculado, fazemos as contas
                                                           e retornamos o valor
   }
6  long count_paths(void)
   {
7    for(int x = 0;x < 8;x++)
7      for(int y = 0;y < 8;y++) { inicializar uma matriz com todas as
7        count_data[x][y] = -1L;                             posições a -1
8    count_data[0][0] = 1L;
9    return eval(7,7); }
                           } Posição final
                           ↳ retorna o número de possibilidades de chegar lá
```

Responda às seguintes perguntas:

- 1.0 a) Que estratégia algorítmica é usada por este código? *Programação dinâmica*
- 3.0 b) Explique por palavras qual é o objetivo de cada uma das partes numeradas do código.

4.0 **4:** Um grafo com 5 vértices, numerados de 1 a 5, tem uma aresta entre o vértice número  $i$  e o número  $j$  se e só se  $i < j$ . Responda às seguintes perguntas:

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



	1	2	3	4	5
1	0	0	0	0	0
2	1	0	0	0	0
3	1	1	0	0	0
4	1	1	1	0	0
5	1	1	1	1	0

d) Sendo que são precisos apenas no máximo  $5 \times 5 = 25$  bits, sim é possível

```

1 | 2 → 3 → 4 → 5 → NULL
2 | 3 → 4 → 5 → NULL
3 | 4 → 5 → NULL
4 | 5 → NULL
5 | NULL
  
```

4.0 **5:** Você foi capturado pelos Borg e levado para um cubo Borg para ser assimilado.<sup>1</sup> Você conseguiu fugir mas está perdido dentro do cubo Borg, que pode ser considerado um labirinto tridimensional. Responda às seguintes perguntas:

- 2.0 a) Para encontrar uma saída do cubo, usaria *depth search* ou *breadth search*?
- 2.0 b) Que material levaria consigo para o ajudar implementar a estratégia algorítmica que escolheu na alínea anterior?

a) usaria o *depth search*, uma vez que a pé seria a maneira de andar menos

b) Alguma coisa para assimilar pontos chave onde já estive

<sup>1</sup>Os Borg são uma raça alienígena do universo *Star Trek*. As suas maiores naves são os cubos Borg, que têm o volume de **27** quilômetros cúbicos. A resistência é fútil!