





Aluno:

(não vale ponto)

## QUESTION

- a) Defina notação **O** discorrendo também sobre funções assintóticas. Dê exemplos de funções de complexidade com suas respectivas ordens de complexidade em termos de **O**
- b) Indique se as alternativas abaixo são verdadeiras ou falsas justificando suas respostas por meio da definição da notação **O**
- $2^{n-1} = \mathbf{O}(2^n)$
  - $2^{n+1} = \mathbf{O}(2^n)$
  - $2^{3n} = \mathbf{O}(4^n)$
  - $10000n^2 + 10000n + n\log n = \mathbf{O}(n^3)$
  - $10000n^2 + 10000n + n\log n = \mathbf{O}(n\log n)$
- c) Prove que:
- Seja  $h_1(n) = \mathbf{O}(f(n))$  e  $h_2(n) = \mathbf{O}(g(n))$ . Mostre que  $h_1(n) + h_2(n) = \mathbf{O}(\max\{f(n), g(n)\})$
  - Seja  $h_1(n) = \mathbf{O}(f(n))$  e  $h_2(n) = \mathbf{O}(g(n))$ . Mostre que  $h_1(n) \times h_2(n) = \mathbf{O}(f(n) \times g(n))$
  - Seja  $h_1(n) = \mathbf{O}(f(n))$  e  $c$  uma constante positiva. Mostre que  $h_1(n) \times c = \mathbf{O}(f(n))$
- d) Analise as assertivas a seguir, assinalando V, se verdadeiras, ou F, se falsas.

- ( ) Seja um vetor com  $n$  elementos. Seja um algoritmo A para encontrar um elemento neste vetor usando uma estratégia de busca de binária. A relação de recorrência deste algoritmo será

$$T(n) = \begin{cases} T(n/2) + c, & \text{se } n > 1 \\ c, & \text{se } n = 1 \end{cases}$$

- ( ) Seja um vetor com  $n$  elementos. Seja um algoritmo A para encontrar o maior elemento do vetor. Este algoritmo A, com ordem de complexidade  $\mathbf{O}(n)$ , pode possuir a seguinte a relação de recorrência

$$T(n) = \begin{cases} 2 \times T(n/2) + c, & \text{se } n > 1 \\ c, & \text{se } n = 1 \end{cases}$$

- ( ) Seja um vetor com  $n$  elementos. Seja um algoritmo A para encontrar o maior elemento do vetor. Este algoritmo A, com ordem de complexidade  $\mathbf{O}(\log n)$ , pode possuir a seguinte a relação de recorrência

$$T(n) = \begin{cases} 2 \times T(n/2) + c, & \text{se } n > 1 \\ c, & \text{se } n = 1 \end{cases}$$

- ( ) Sejam dois vetores A e B contendo  $m$  e  $n$  elementos, respectivamente. Seja o seguinte algoritmo: (i) selecione o menor elemento de A e compare com o menor elemento de B; (ii) copie o menor elemento encontrado para um vetor C; (iii) repita os procedimentos (i) e (ii) enquanto houver elemento ainda não copiado para C. O custo computacional deste algoritmo é  $\mathbf{O}(mn)$ .

- ( ) Se  $f'(n) = \mathcal{O}(g'(n))$  e  $f''(n) = \mathcal{O}(g''(n))$ , então  $f'(n) + f''(n) = \mathcal{O}(g'(n) + g''(n))$ .

- ( )  $f(n) = \mathcal{O}(g(n))$  se  $\exists c > 0, n_0 \geq 0$  tal que  $g(n) \leq cf(n), \forall n \geq n_0$ .

a) A notação  $\mathcal{O}$  se dá quando, para uma constante  $C$  maior que 0, e uma variável inicial  $M_0$  maior ou igual a 0, a função  $f(n)$  é menor ou igual à constante vezes  $g(n)$ , para todos os valores de  $n$  maiores que  $M_0$ .

Além, a análise da definição da notação big-O é assintótica, já que analisa uma condição à medida que as funções crescem indefinidamente (indefinidamente) a partir de um valor de  $M_0$  para  $n$ .

Um exemplo de função de complexidade e ordem de complexidade é a função  $n^2 + L$ , que possui ordem de complexidade  $\mathcal{O}(n^2)$ , pois:

$$n^2 + L \leq C \cdot n^2 ? \rightarrow \frac{n^2 + L}{n^2} \leq C$$

$$\frac{n^2}{n^2} + \frac{L}{n^2} \leq C$$

assim temos  $C = 2$

existe uma constante  $C$  finita para um valor de  $M_0$  que permite que a condição continue verdadeira assintoticamente e permite a ordem de complexidade  $\mathcal{O}(n^2)$ .

$$1 + \frac{L}{n^2} \leq C \leftarrow$$

$$1 + \frac{L}{n^2} \leq 2$$

$$1 + L \leq 2 \checkmark$$

b)

- a)  $2^{n-1} = \mathcal{O}(2^n)$
- b)  $2^{n+1} = \mathcal{O}(2^n)$
- c)  $2^{3n} = \mathcal{O}(4^n)$
- d)  $10000n^2 + 10000n + n\log n = \mathcal{O}(n^3)$
- e)  $10000n^2 + 10000n + n\log n = \mathcal{O}(n\log n)$

$$a) 2^{n-1} = O(2^n) \rightarrow 2^{n-1} \leq c \cdot 2^n$$

$$\frac{2^{n-1}}{2^n} \leq c$$

$$\frac{2^n \cdot 2^{-1}}{2^n} \leq c \\ \frac{1}{2} \leq c \quad \checkmark$$

$$b) 2^{n+1} = O(2^n) \rightarrow 2^{n+1} \leq c \cdot 2^n$$

$$\frac{2^{n+1}}{2^n} \leq c \cdot \frac{2^n}{2^n}$$

$$2^1 \leq c \quad \checkmark$$

$$c) 2^{3n} = O(4^n) \rightarrow 2^{3n} \leq c \cdot 4^n$$

$$\frac{2^{3n}}{2^{2n}} \leq c \cdot \frac{2^n}{2^n}$$

$$\lim_{n \rightarrow \infty} 2^n \Rightarrow \infty$$

*\* sempre que o valor de n mudar, a constante também mudará, o que prova que não há uma constante finita que satisfaça a definição.*

*ou seja, C constante, n é indefinidamente maior (uma constante)*

$$d) 100000n^2 + 10000n + n\log n = O(n^3)$$



$$\frac{100000n^2 + 10000n + n\log n}{n^2} \leq \frac{c \cdot n^3}{n^3}$$

$$\frac{10000n + 10000 + \log n}{n^2} \leq c$$

$$\lim \frac{10^n + 10^n + \log n}{n^2}$$

e)  $100000n^2 + 10000n + n \log n = O(n \log n)$

X

$$100000n^2 + 10000n + n \log n \leq c \cdot n \log n$$

$$\frac{100000n^2 + 10000n + n \log n}{n \log n} \leq \frac{c \cdot n \log n}{n \log n}$$

$$\frac{10000n^2 + 10000n}{n \log n} \leq c$$

c) Prove que:

- Seja  $h_1(n) = O(f(n))$  e  $h_2(n) = O(g(n))$ . Mostre que  $h_1(n) + h_2(n) = O(\max\{f(n), g(n)\})$
- Seja  $h_1(n) = O(f(n))$  e  $h_2(n) = O(g(n))$ . Mostre que  $h_1(n) \times h_2(n) = O(f(n) \times g(n))$
- Seja  $h_1(n) = O(f(n))$  e  $c$  uma constante positiva. Mostre que  $h_1(n) \times c = O(f(n))$

3) Analise os seguintes argumentos para comprovar que  $V$  é um algoritmo em  $E$  ou falso.

a)  $h_1(n) + h_2(n) = O(\max\{f(n), g(n)\})$

FAZER

DEPOIS

V Seja um vetor com  $n$  elementos. Seja um algoritmo A para encontrar um elemento neste vetor usando uma estratégia de busca binária. A relação de recorrência deste algoritmo será

$$T(n) = \begin{cases} T(n/2) + c, & \text{se } n > 1 \\ c, & \text{se } n = 1 \end{cases}$$

$$\begin{aligned} T(n) &= T(n/2) + c \\ T(n/2) &= T(n/4) + c + c \\ T(n/4) &= T(n/8) + c + c + c \\ &\vdots \\ T(n/2^i) &= T(n/2^{i+1}) + c^{i+1} \end{aligned}$$

$$\begin{aligned} T(2) &= \\ T(1) &= c \end{aligned} \quad i = [0, \log_2 n - 1]$$

$$\begin{aligned} \frac{n}{2^i} &= n & 2^i &= \frac{n}{n} \\ 2^i &= 1 & 2^0 &= 1 \\ 2^0 &= 1 & i &= 0 \end{aligned}$$

$$\begin{aligned} \frac{n}{2^i} &= 2 \\ n &= 2 \cdot 2^i \end{aligned}$$

$$\log_2 n/2 = i$$

$$i = \log_2 n - \underbrace{\log_2 2}_1$$

$$i = \log_2 n - 1$$

$$\frac{C}{\log n} \left( \frac{\log n}{\log 2} + 1 \right)$$

$$\frac{C \cdot \log n}{\log n \cdot \log 2} + \frac{C}{\log n}$$

$$\begin{aligned} \frac{C}{\log 2} + \frac{C}{\log n} \\ C \left( \frac{1}{\log 2} + \frac{1}{\log n} \right) \end{aligned}$$

$$\sum_{i=0}^{\log_2 n - 1} C^i + C$$

(caso base)

variação  $0 < \underbrace{\log_2 n}_{(\log_2 n \leq n)} \leq \log_2 n - 1$

$$T(n) = C (\log_2 n + 1) + C$$

$$C (\log_2 n) + C \leq C_1 \cdot \log n$$

$$C \left( \frac{\log_{10} 2}{\log_{10} n} + 1 \right) \leq C_1 \cdot \log n$$

$$\frac{C \left( \frac{\log_{10} 2}{\log_{10} n} + 1 \right)}{\log_{10} n} \leq \frac{C_1 \cdot \log n}{\log n}$$

$$C \left( \frac{1}{\log 2} + \frac{1}{\log n} \right) \leq C_1$$

$$C_1 = C \left( \frac{1}{\log 2} + 1 \right)$$

$$\lim_{n \rightarrow \infty} \frac{1}{\log_{10} n} = 1$$

depois de 1

( ) Seja um vetor com  $n$  elementos. Seja um algoritmo A para encontrar o maior elemento do vetor. Este algoritmo A, com ordem de complexidade  $O(n)$ , pode possuir a seguinte a relação de recorrência

$$T(n) = \begin{cases} 2 \times T(n/2) + c, & \text{se } n > 1 \\ c, & \text{se } n = 1 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(n/2) + C \\ 2T(n/2) &= 2 \cdot 2T(n/4) + C \\ 2 \cdot 2T(n/4) &= 2 \cdot 2 \cdot 2T(n/8) + C \\ &\vdots \\ 2^i T(n/2^i) &= 2^{i+1} T(n/2^{i+1}) + C \\ &\vdots \\ T(L) &= C \end{aligned}$$

$$\left. \begin{aligned} \frac{n}{2^i} &= n \Rightarrow i = 0 \\ \frac{n}{2^i} &= 2 \Rightarrow i = \log_2 n - 1 \end{aligned} \right\} \sum_{i=0}^{\log_2 n - 1} 2^i + C \quad \text{case base}$$

fórmula  
da P.G.

$$\begin{aligned} P_G &= \frac{q_1(r^n - 1)}{r - 1} = \frac{1(2^{\log_2 n} - 1)}{2 - 1} = \\ &= \frac{2^{\log_2 n}}{2} = \frac{n}{2} \end{aligned}$$

utimos quantificando

$$T(m) = 2T(m/2) + C$$

curto de tempo do seu algoritmo baseado na entrada  $m$

$$2 \cdot T(m/2) = 2 \cdot T(m/4) + 2C$$

⋮

$$T(2) = 2T(1) + C$$

$$T(1) = C$$

Temos que resolver  $T(m)$  → é a última tarefa é resolver todos os  $T(m)$ , multiplicando os dois lados por constante

$$d = v \cdot t \Rightarrow \text{fórmula fechada}$$

$$\sum_{i=0}^{\log_2 n} f(i) \Rightarrow \text{não é uma fórmula fechada}$$

(Temos que resolver os anteriores p/ tornar a fechada)

ignorando  $i$  à primeira e à ultima

$$\frac{m}{2^i} = 2 \Rightarrow \log_2 n - 1$$

instância

$$\frac{m}{2^i} = m \Rightarrow 0$$

2^{\log\_2 n} = n

chegamos numa fórmula fechada!

$$T(m) = \frac{3}{2}C \cdot m - C$$

$$T(m) = O(m)$$

$$\exists c > 0, m_0 > 0 \mid \frac{3}{2}m \leq c \cdot m, \forall m > m_0$$

$$\frac{3}{2}m \leq c \cdot m$$

c \geq \frac{3}{2}

satisfaz

$$\log_3 m \leq \log_2 m \cdot c$$

$$\frac{\log_{10} m}{\log_{10} 3} \leq c \cdot \frac{\log_{10} m}{\log_{10} 2}$$

$$c > \frac{\log_2 2}{\log_{10} 3}$$

Prove que:

- Seja  $h_1(n) = \mathbf{O}(f(n))$  e  $h_2(n) = \mathbf{O}(g(n))$ . Mostre que  $h_1(n) + h_2(n) = \mathbf{O}(\max\{f(n), g(n)\})$
- Seja  $h_1(n) = \mathbf{O}(f(n))$  e  $h_2(n) = \mathbf{O}(g(n))$ . Mostre que  $h_1(n) \times h_2(n) = \mathbf{O}(f(n) \times g(n))$
- Seja  $h_1(n) = \mathbf{O}(f(n))$  e  $c$  uma constante positiva. Mostre que  $h_1(n) \times c = \mathbf{O}(f(n))$

a)  $n_1 \leq c_1 \cdot f(n)$ ,  $n \geq n_1$   
 $n_2 \leq c_2 \cdot g(n)$ ,  $n \geq n_2$  + }  $n_1 + n_2 \leq c_1 \cdot f(n) + c_2 \cdot g(n)$ ,  
 $n \geq \max(n_1, n_2)$

$$h_1 + h_2 \leq C \cdot \max(g(n), f(n))$$

Se  $f > g \Rightarrow \max(f, g) = f$

$$c_1 f + c_2 g \leq c_1 f + c_2 f \\ \leq (c_1 + c_2) f$$

$$h_1 + h_2 \leq c_3 f$$

b)  $h_1 = f(n)$   
 $h_2 = g(n)$

$$f(n) \cdot g(n) \leq c_1 f(n) \cdot c_2 g(n)$$

$$f \cdot g \leq c_3 f \cdot g$$

como  $c$  é maior que  
0, se a desigualdade é  
a equação é igual  
(p/  $c_3 = L$ ) ou maior  
que  $f \cdot g$  p/ todo valor  
de  $n$  menor ou igual a  $M_0$

p/ todo  
 $n \geq \max(M_1, M_2)$   
(p/ garantir que  
 $n_2$  é menor ou igual a  $n_1$ )  
 $n_2$  é menor ou igual a  $n_1$  ou menor

$$c) h_1 = O(f(m)) \quad f(n) \leq K \cdot f(m)$$
$$c \cdot h_1(n) \leq K \cdot f(n) \cdot c$$