

Projeto e Análise de Algoritmos (2025-1)

Flávio L. C. de Moura*

21 de maio de 2025

Prova 1 - gabarito

Teorema 0.1. *Seja $T(n)$ uma função eventualmente não-decrescente que satisfaz a recorrência $T(n) = a.T(n/b) + f(n)$, para $n = b^k, k = 1, 2, 3, \dots$
 $T(1) = c$*

onde $a \geq 1, b \geq 2$ e $c \geq 0$. Se $f(n) = \Theta(n^d)$, onde $d \geq 0$, então

$$T(n) = \begin{cases} \Theta(n^{\log_b a}), & \text{se } a > b^d \\ \Theta(n^d \cdot \lg n), & \text{se } a = b^d \\ \Theta(n^d), & \text{se } a < b^d \end{cases}$$

Teorema 0.2. *Sejam $a \geq 1$ e $b > 1$ constantes, $f(n)$ uma função assintoticamente positiva, e $T(n)$ definida nos inteiros não-negativos pela recorrência:*

$$T(n) = a.T(n/b) + f(n)$$

onde n/b deve ser interpretado como $\lfloor n/b \rfloor$ ou $\lceil n/b \rceil$. Então $T(n)$ tem as seguintes cotas assintóticas:

1. *Se $f(n) = O(n^{\log_b a - \epsilon})$ para alguma constante $\epsilon > 0$, então $T(n) = \Theta(n^{\log_b a})$.*
 2. *Se $f(n) = \Theta(n^{\log_b a})$, então $T(n) = \Theta(n^{\log_b a} \cdot \lg n)$.*
 3. *Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ para alguma constante $\epsilon > 0$, e se $a.f(n/b) \leq c.f(n)$ para alguma constante $c < 1$, então para todo n suficientemente grande, temos que $T(n) = \Theta(f(n))$.*
-
1. (2.5 pontos) Sejam $f(n)$ e $g(n)$ funções (assintoticamente) não-negativas. Prove, utilizando as definições de notação assintótica, que $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$.

*flaviomoura@unb.br

Solução Inicialmente, mostraremos que $\max f(n), g(n) = O(f(n) + g(n))$, isto é, mostraremos que existem constantes positivas c_1 e n_1 tais que $\max(f(n), g(n)) \leq c_1 \cdot (f(n) + g(n)), \forall n \geq n_1$. De fato, $\max(f(n), g(n)) \leq f(n) + g(n), \forall n$. Então podemos tomar $c_1 = 1$ e n_1 qualquer. Para mostrarmos que $\max\{f(n), g(n)\} = \Omega(f(n) + g(n))$, precisamos encontrar constantes positivas c_2 e n_2 tais que $\max\{f(n), g(n)\} \geq c_2 \cdot (f(n) + g(n)), \forall n \geq n_2$. Sabemos que $\max\{f(n), g(n)\} \geq f(n), \forall n$ e $\max\{f(n), g(n)\} \geq g(n), \forall n$. Somando as duas desigualdades, temos que $2 \cdot \max\{f(n), g(n)\} \geq f(n) + g(n), \forall n$, ou seja, $\max\{f(n), g(n)\} \geq \frac{1}{2} \cdot (f(n) + g(n)), \forall n$. Então basta tomarmos, $c_2 = \frac{1}{2}$ e n_2 qualquer. Por fim, como $\max\{f(n), g(n)\} = O(f(n) + g(n))$ e $\max\{f(n), g(n)\} = \Omega(f(n) + g(n))$, concluímos que $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$.

2. (2.5 pontos) Mostre que $\sum_{i=1}^n (\lg i) = \Theta(n \cdot \lg n)$.

Solução Observe que $\sum_{i=1}^n (\lg i) \leq n \cdot \lg(n), \forall n \geq 1$, e portanto $\sum_{i=1}^n (\lg i) = O(n \cdot \lg(n))$. Por outro lado, $\sum_{i=1}^n (\lg i) \geq \frac{n}{2} \cdot \lg(\frac{n}{2}), \forall n$, ou seja, $\sum_{i=1}^n (\lg i) = \Omega(n \cdot \lg(n))$. Assim, temos $\sum_{i=1}^n (\lg i) = \Theta(n \cdot \lg n)$.

3. Considere um algoritmo recursivo hipotético que resolve uma classe de problemas da seguinte forma:

- (a) Cada instância do problema de tamanho n recebida como entrada é dividida em um subproblema de tamanho $2n/3$, que por sua vez é resolvida recursivamente;
- (b) As soluções dos subproblemas são combinadas em tempo constante, para que seja possível formar uma solução do problema original.

Responda os itens a seguir:

- (a) (1.0 ponto) Escreva a recorrência que modela a complexidade deste algoritmo.

Solução $T(n) = T(2n/3) + \Theta(1)$

- (b) (1.5 pontos) Determine a complexidade assintótica deste algoritmo.

Solução Esta recorrência tem a forma dada no Teorema Mestre com $a = 1$, $b = 3/2$ e $d = 0$. Como $a = 1 = (\frac{3}{2})^0 = b^d$, temos pelo caso 2 que $T(n) = \Theta(\lg n)$.

4. (2.5 pontos) Considere o pseudocódigo a seguir:

Algorithm 1: *algoritmo*($A[1..n]$)

```
1  $x \leftarrow A[1]$ ;  
2 for  $i = 2$  to  $n$  do  
3   | if  $A[i] < x$  then  
4   |   |  $x \leftarrow A[i]$ ;  
5   | end  
6 end  
7 return  $x$ ;
```

O que faz esse algoritmo? Mostre que ele é correto baseado na sua resposta.

Solução O algoritmo retorna o menor elemento no vetor $A[1..n]$. Mostraremos que ele é correto baseado na seguinte invariante:

Antes de cada iteração indexada por i , a variável x contém o menor elemento do subvetor $A[1..i - 1]$.

Inicialização: Antes da primeira iteração, temos $i = 2$, e $x = A[1]$ (linha 1) contém o menor elemento do subvetor $A[1]$.

Manutenção: Assuma que antes da k -ésima iteração, isto é, quando $i = k + 1$, x contém o menor elemento do subvetor $A[1..k]$. Ao longo da k -ésima iteração o elemento $A[i] = A[k + 1]$ será comparado com x (linha 3). Se $A[k + 1]$ for menor do que x então encontramos um valor menor do que o atual no subvetor $A[1..k + 1]$, e este novo valor é armazenado em x (linha 4). Assim, antes da $(k + 1)$ -ésima iteração x contém o menor elemento do subvetor $A[1..k + 1]$, como queríamos provar.

Finalização: Ao final da execução do laço, isto é, quando $i = n + 1$, temos que x contém o menor elemento do vetor $A[1..n]$, e portanto o algoritmo retorna o menor elemento do vetor $A[1..n]$.