Estrutura de Dados
Lista de Exercícios III
5 pontos

UNIVERSIADE FEDERAL DE LAVRAS (UFLA/ICTIN)
-GABARITO
Prof. Dr Johnatan Oliveira

22 de agosto de 2024

PRAZO DE ENTREGA: 04/08/2024 ATÉ 23:59

Exercícios sobre Notação Assintótica e Recorrência

INSTRUÇÕES:

- A lista deve ser resolvida por manuscrito, ou seja, a mão. Em seguida, o aluno deverá digitalizar (tirar uma foto legível) e enviar no UFLA Virtual.
- Qualquer resolução fora do padrão resultará em 0 pontos.

Questão 1

Instruções: Resolva os seguintes exercícios. Justifique todas as suas respostas.

1. f(n) + g(n) = θ(min(f(n), g(n)))

Demonstração: $f(n)+g(n) = \theta(\min(f(n), g(n)))$

Passo 1: Entendimento da Notação θ

A notação θ é usada para descrever o comportamento assintótico de uma função em termos de limites superiores e inferiores. Especificamente, dizemos que uma função $T(n)$ é $\theta(h(n))$ se existem constantes positivas c_1, c_2 e n_0 tais que:

$$c_1 \cdot h(n) \leq T(n) \leq c_2 \cdot h(n) \quad \text{para todo} \quad n \geq n_0$$

Aqui, precisamos provar que $f(n)+g(n)$ é assintoticamente equivalente ao mínimo entre $f(n)$ e $g(n)$.

Passo 2: Análise dos casos possíveis

Vamos analisar o comportamento de $f(n)+g(n)$ considerando diferentes relações entre $f(n)$ e $g(n)$.

Já sabemos que $f(n) + g(n)$ será maior ou igual a $f(n)$. Agora, para o limite superior, como $g(n) \geq f(n)$, podemos concluir que:

$$f(n) + g(n) \leq f(n) + f(n) = 2f(n)$$

Caso 1: $f(n) \leq g(n)$

Se $f(n)$ é menor ou igual a $g(n)$, o termo dominante em $f(n) + g(n)$ será $f(n)$, pois:

$$f(n) \leq f(n) + g(n) \leq 2f(n)$$

Isso significa que:

$$f(n) + g(n) = \theta(f(n))$$

E como $f(n) \leq g(n)$, temos que $f(n) = \min(f(n), g(n))$, logo:

$$f(n) + g(n) = \theta(\min(f(n), g(n)))$$

Caso 2: $g(n) \leq f(n)$

Este caso é simétrico ao anterior. Se $g(n)$ é menor ou igual a $f(n)$, então o termo dominante em $f(n) + g(n)$ será $g(n)$, pois:

$$g(n) \leq f(n) + g(n) \leq 2g(n)$$

E, portanto, podemos dizer que:

$$f(n) + g(n) = \theta(g(n))$$

Como $g(n) \leq f(n)$, temos que $g(n) = \min(f(n), g(n))$, logo:

$$f(n) + g(n) = \theta(\min(f(n), g(n)))$$

Passo 3: Conclusão

Em ambos os casos, $f(n) + g(n)$ é θ do menor entre $f(n)$ e $g(n)$. Isso prova que:

$$f(n) + g(n) = \theta(\min(f(n), g(n)))$$

2. f(n) + g(n) = θ(max(f(n), g(n)))

Demonstração: $f(n)+g(n) = \theta(\max(f(n), g(n)))$

Passo 1: Análise dos casos possíveis

Vamos considerar os dois casos possíveis para $f(n)$ e $g(n)$:

Caso 1: $f(n) \geq g(n)$

Se $f(n) \geq g(n)$, então $\max(f(n), g(n)) = f(n)$.

Neste caso, a soma $f(n)+g(n)$ será maior ou igual a $f(n)$, mas também será menor que $2f(n)$:

$$f(n) \leq f(n) + g(n) \leq 2f(n)$$

Isso significa que:

$$f(n) + g(n) = \theta(f(n)) = \theta(\max(f(n), g(n)))$$

Caso 2: $g(n) \geq f(n)$

Se $g(n) \geq f(n)$, então $\max(f(n), g(n)) = g(n)$.

Neste caso, a soma $f(n)+g(n)$ será maior ou igual a $g(n)$, mas também será menor que $2g(n)$:

$$g(n) \leq f(n) + g(n) \leq 2g(n)$$

Isso significa que:

$$f(n) + g(n) = \theta(g(n)) = \theta(\max(f(n), g(n)))$$

Conclusão

Nos dois casos, a soma $f(n) + g(n)$ é assintoticamente equivalente ao maior dos dois termos $f(n)$ e $g(n)$. Portanto, concluímos que:

$$f(n) + g(n) = \theta(\max(f(n), g(n)))$$

3. $O(n + n^2) = O(n^2)$

   Vamos provar que $O(n + n^2) = O(n^2)$ utilizando a definição formal de notação Big-O.

Passo 1: Definição formal de Big-O

Por definição, uma função $f(n)$ é $O(g(n))$ se existirem constantes positivas c e n_0 tais que:

$$f(n) \leq c \cdot g(n) \quad \text{para todo } n \geq n_0$$

No nosso caso, queremos provar que $f(n) = n + n^2$ é $O(g(n))$, onde $g(n) = n^2$.

Passo 2: Analisar a função $f(n) = n + n^2$

Vamos expressar $f(n)$ em termos de $g(n)$:

$$f(n) = n + n^2 = n^2 + n$$

Queremos encontrar constantes c e n_0 tal que:

$$n + n^2 \leq c \cdot n^2 \quad \text{para todo } n \geq n_0$$

Passo 3: Dividir ambos os lados por n^2

Para simplificar, dividimos ambos os lados da inequação por n^2:

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

Isso nos dá:

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

$$\frac{1}{n} + 1 \leq c$$

Passo 4: Encontrar o valor de c

Observe que conforme n aumenta, $\frac{1}{n}$ se aproxima de 0. Então, podemos considerar o termo $\frac{1}{n}$ insignificante para valores grandes de n.

Para qualquer valor de $n \geq 1$, temos:

$$\frac{1}{n} \leq 1$$

Portanto:

$$1 + 1 \leq c$$

$$c \geq 2$$

Passo 5: Determinar n_0

Para garantir que a inequação $\frac{1}{n} + 1 \leq c$ seja válida, escolhemos $n_0 = 1$. Isso significa que para $n \geq 1$ e $c = 2$, a inequação é satisfeita.

Passo 6: Conclusão

Com base na definição de Big-O, mostramos que:

$$n + n^2 \leq 2 \cdot n^2 \quad \text{para todo } n \geq 1$$

Portanto, $n + n^2$ é $O(n^2)$, ou seja:

$$O(n + n^2) = O(n^2)$$

4. $O(n \cdot \log n) = O(n) \cdot O(\log n)$

   Vamos provar matematicamente que $O(n \cdot \log n) = O(n) \cdot O(\log n)$ utilizando a definição formal de Big-O.

Passo 1: Definição Formal de Big-O

A notação $O(f(n))$ é definida como um conjunto de funções. Uma função $f(n)$ pertence ao conjunto $O(g(n))$ se existirem constantes positivas c e n_0 tais que:

$$f(n) \leq c \cdot g(n) \quad \text{para todo } n \geq n_0$$

Passo 2: Análise da Expressão $O(n \cdot \log n)$

Vamos provar que:

$$O(n \cdot \log n) = O(n) \cdot O(\log n)$$

Primeiro, devemos provar que $O(n \cdot \log n)$ está contido em $O(n) \cdot O(\log n)$. Isso significa que precisamos encontrar constantes c_1, c_2, e n_0 tais que:

$$n \cdot \log n \leq c_1 \cdot n \cdot c_2 \cdot \log n$$

Passo 3: Prova Matemática

Vamos assumir que:

$$f(n) = n \cdot \log n$$
$$g_1(n) = n \quad \text{e} \quad g_2(n) = \log n$$

De acordo com a propriedade multiplicativa, temos:

$$O(f(n)) = O(g_1(n)) \cdot O(g_2(n))$$

Agora, precisamos mostrar que:

$$n \cdot \log n \leq c_1 \cdot n \cdot \log n \quad \text{para algum } c_1 > 0 \text{ e para todo } n \geq n_0$$

Escolha $c_1 = 1$. Então, temos:

$$n \cdot \log n \leq 1 \cdot n \cdot \log n$$

Esta inequação é obviamente verdadeira para todo $n \geq 1$.

Passo 4: Conclusão

Portanto, mostramos que:

$$O(n \cdot \log n) = O(n) \cdot O(\log n)$$

5. Se $g = O(f)$ e $h = \Theta(f)$, então $g = O(h)$ Vamos provar que se $g = O(f)$ e $h = \Theta(f)$, então $g = O(h)$.

Passo 1: Definição de Big-O e Theta

Lembre-se das definições:

- $g = O(f)$ significa que existe uma constante $c_1 > 0$ e um n_0 tal que:
$$g(n) \leq c_1 \cdot f(n) \quad \text{para todo } n \geq n_0$$
- $h = \Theta(f)$ significa que existem constantes $c_2, c_3 > 0$ e um n_1 tal que:
$$c_2 \cdot f(n) \leq h(n) \leq c_3 \cdot f(n) \quad \text{para todo } n \geq n_1$$

Passo 2: Relacionar $g(n)$ e $h(n)$

Queremos mostrar que $g = O(h)$, ou seja, precisamos provar que existe uma constante $c_4 > 0$ e um n_2 tal que:

$$g(n) \leq c_4 \cdot h(n) \quad \text{para todo } n \geq n_2$$

Passo 3: Utilizar as Definições

Sabemos que $g(n) \leq c_1 \cdot f(n)$ (definição de $g = O(f)$) e que $h(n) \geq c_2 \cdot f(n)$ (definição de $h = \Theta(f)$). Vamos usar essas duas desigualdades.

Especificamente, da definição de $h = \Theta(f)$, sabemos que $h(n)$ é sempre maior ou igual a $c_2 \cdot f(n)$:

$$h(n) \geq c_2 \cdot f(n)$$

Agora, substituímos $f(n)$ da definição de $g = O(f)$ por $\frac{h(n)}{c_2}$ (da desigualdade acima):

$$g(n) \leq c_1 \cdot f(n) = c_1 \cdot \frac{h(n)}{c_2}$$

Simplificando essa expressão:

$$g(n) \leq \frac{c_1}{c_2} \cdot h(n)$$

Passo 4: Conclusão

Podemos escolher $c_4 = \frac{c_1}{c_2}$, então temos:

$$g(n) \leq c_4 \cdot h(n) \quad \text{para todo } n \geq \max(n_0, n_1)$$

Assim, $g(n)$ é $O(h(n))$, o que prova que $g = O(h)$.

6. $f(n) = O(g(n))$ implica que $g(n) = \Theta(f(n))$

   Vamos analisar a afirmação e provar se $f(n) = O(g(n))$ implica que $g(n) = \Theta(f(n))$.

Revisão das Definições

- **Definição de Big-O**:

  $f(n) = O(g(n))$ significa que existe uma constante $c_1 > 0$ e um valor n_0 tal que:

  $$f(n) \leq c_1 \cdot g(n) \quad \text{para todo } n \geq n_0$$

- **Definição de Theta (Θ)**:

  $g(n) = \Theta(f(n))$ significa que existem constantes $c_2 > 0$ e $c_3 > 0$ e um valor n_1 tal q

  $$c_2 \cdot f(n) \leq g(n) \leq c_3 \cdot f(n) \quad \text{para todo } n \geq n_1$$

Prova da Afirmação

Vamos agora analisar se $f(n) = O(g(n))$ implica que $g(n) = \Theta(f(n))$.

Passo 1: Assumir que $f(n) = O(g(n))$

Se $f(n) = O(g(n))$, então existe uma constante $c_1 > 0$ tal que:

$$f(n) \leq c_1 \cdot g(n) \quad \text{para todo } n \geq n_0$$

Passo 2: Testar a Implicação

Para que $g(n) = \Theta(f(n))$ seja verdadeiro, precisamos encontrar constantes c_2 e c_3 e um n_1 tal que:

$$c_2 \cdot f(n) \leq g(n) \leq c_3 \cdot f(n) \quad \text{para todo } n \geq n_1$$

A desigualdade $g(n) \leq c_3 \cdot f(n)$ pode ser satisfeita, pois podemos escolher $c_3 = \frac{1}{c_1}$ para a constante superior. Porém, a desigualdade $c_2 \cdot f(n) \leq g(n)$ não necessariamente se mantém para qualquer função $f(n)$ e $g(n)$.

Passo 3: Contraexemplo

Considere um exemplo simples onde $f(n) = 1$ e $g(n) = n$. Nesse caso, claramente $f(n) = O(g(n))$, pois $1 \leq c_1 \cdot n$ para qualquer $c_1 > 0$.

Agora, testemos se $g(n) = \Theta(f(n))$:

- $g(n) = n$, mas $f(n) = 1$, então a desigualdade $c_2 \cdot f(n) \leq g(n)$ implica $c_2 \leq n$, que não é possível satisfazer para um c_2 constante quando n é grande.

Este contraexemplo demonstra que $f(n) = O(g(n))$ não implica que $g(n) = \Theta(f(n))$.

Conclusão

Portanto, $f(n) = O(g(n))$ **não** implica que $g(n) = \Theta(f(n))$. O exemplo com $f(n) = 1$ e $g(n) = n$ serve para mostrar que, embora $f(n)$ esteja limitado por $g(n)$, $g(n)$ não está simetricamente limitado por $f(n)$ de maneira que satisfaça a definição de $\Theta(f(n))$.

7. $f(n) = O(g(n))$ não implica que $g(n) = O(f(n))$

Vamos provar que $f(n) = O(g(n))$ **não** implica que $g(n) = O(f(n))$.

Revisão das Definições

- **Definição de Big-O**:

$$f(n) = O(g(n)) \quad \text{significa que existe uma constante } c_1 > 0 \text{ e um valor } n_0 \text{ tal que:}$$

$$f(n) \leq c_1 \cdot g(n) \quad \text{para todo } n \geq n_0$$

- **Implicação que Queremos Testar**:

$$g(n) = O(f(n)) \quad \text{significa que existe uma constante } c_2 > 0 \text{ e um valor } n_1 \text{ tal que:}$$

$$g(n) \leq c_2 \cdot f(n) \quad \text{para todo } n \geq n_1$$

Prova com um Contraexemplo

Vamos considerar $f(n) = 1$ e $g(n) = n$.

1. Verificar se $f(n) = O(g(n))$:

- $f(n) = 1$ é uma função constante.
- $g(n) = n$ é uma função linear.
- Podemos afirmar que $f(n) = 1 \leq c_1 \cdot n$ para qualquer $c_1 \geq 1$ e $n \geq 1$.

Portanto, $f(n) = O(g(n))$ com $c_1 = 1$ e $n_0 = 1$.

2. Verificar se $g(n) = O(f(n))$:

- Para que $g(n) = O(f(n))$, precisaríamos de:

$$n \leq c_2 \cdot 1$$

  para algum $c_2 > 0$ e para todo $n \geq n_1$.
- No entanto, isso não é possível porque a função n cresce indefinidamente e não pode ser limitada por uma constante c_2 multiplicada por 1.

Esse exemplo mostra que, embora $f(n) = O(g(n))$, o inverso não é verdadeiro: $g(n)$ não é $O(f(n))$.

Conclusão

Portanto, $f(n) = O(g(n))$ **não** implica que $g(n) = O(f(n))$. O exemplo com $f(n) = 1$ e $g(n) = n$ demonstra claramente essa falha na implicação.

8. Se $g(n) = \Theta(f(n))$, então $f(n) + g(n) = \Theta(f(n))$

Vamos provar que se $g(n) = \Theta(f(n))$, então $f(n) + g(n) = \Theta(f(n))$.

Revisão das Definições

- **Definição de Θ (Theta)**:

  $$g(n) = \Theta(f(n)) \quad \text{significa que existem constantes positivas } c_1, c_2 \text{ e um valor } n_0 \text{ tal q}$$

  $$c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n) \quad \text{para todo } n \geq n_0$$

- Isso implica que $g(n)$ está assintoticamente limitado acima e abaixo por $f(n)$.

Passo 1: Assumir que $g(n) = \Theta(f(n))$

Se $g(n) = \Theta(f(n))$, então existem constantes $c_1 > 0$, $c_2 > 0$ e um valor n_0 tal que:

$$c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n) \quad \text{para todo } n \geq n_0$$

Passo 2: Analisar $f(n) + g(n)$

Queremos provar que $f(n) + g(n) = \Theta(f(n))$. Vamos considerar os limites inferior e superior de $f(n) + g(n)$.

1. Limite Inferior

Sabemos que $g(n) \geq c_1 \cdot f(n)$, então:

$$f(n) + g(n) \geq f(n) + c_1 \cdot f(n) = (1 + c_1) \cdot f(n)$$

Portanto, $f(n) + g(n)$ é limitado inferiormente por $(1 + c_1) \cdot f(n)$.

2. Limite Superior

Sabemos que $g(n) \leq c_2 \cdot f(n)$, então:

$$f(n) + g(n) \leq f(n) + c_2 \cdot f(n) = (1 + c_2) \cdot f(n)$$

Portanto, $f(n) + g(n)$ é limitado superiormente por $(1 + c_2) \cdot f(n)$.

Passo 3: Conclusão

Combinando os limites inferior e superior, temos:

$$(1 + c_1) \cdot f(n) \leq f(n) + g(n) \leq (1 + c_2) \cdot f(n) \quad \text{para todo } n \geq n_0$$

Isso mostra que $f(n) + g(n)$ está assintoticamente limitado acima e abaixo por uma constante múltipla de $f(n)$, o que significa que:

$$f(n) + g(n) = \Theta(f(n))$$

9. Se $f(n) = \Theta(n^2)$ e $g(n) = O(n)$, então $f(n) + g(n) = \Theta(n^2)$

   Vamos provar que se $f(n) = \Theta(n^2)$ e $g(n) = O(n)$, então $f(n) + g(n) = \Theta(n^2)$.

Revisão das Definições

- **Definição de Θ (Theta)**:

  $f(n) = \Theta(n^2)$ significa que existem constantes positivas c_1, c_2 e um valor n_0 tal qu

  $$c_1 \cdot n^2 \leq f(n) \leq c_2 \cdot n^2 \quad \text{para todo } n \geq n_0$$

- **Definição de $O(g(n))$ (Big-O)**:

  $g(n) = O(n)$ significa que existe uma constante positiva c_3 e um valor n_1 tal que:

  $$g(n) \leq c_3 \cdot n \quad \text{para todo } n \geq n_1$$

Passo 1: Analisar $f(n) + g(n)$

Queremos provar que $f(n) + g(n) = \Theta(n^2)$. Para isso, vamos analisar os limites inferior e superior de $f(n) + g(n)$.

1. Limite Inferior

Sabemos que $f(n) \geq c_1 \cdot n^2$. Como $g(n)$ é $O(n)$, podemos afirmar que, para valores grandes de n, o termo $f(n)$ domina $g(n)$.

Portanto:

$$f(n) + g(n) \geq f(n) \geq c_1 \cdot n^2$$

2. Limite Superior

Sabemos que $f(n) \leq c_2 \cdot n^2$ e que $g(n) \leq c_3 \cdot n$. Logo:

$$f(n) + g(n) \leq c_2 \cdot n^2 + c_3 \cdot n$$

Para valores grandes de n, o termo $c_2 \cdot n^2$ domina $c_3 \cdot n$, então a expressão pode ser limitada por:

$$f(n) + g(n) \leq c_2 \cdot n^2 + c_3 \cdot n \leq (c_2 + \epsilon) \cdot n^2$$

Onde ϵ é uma constante pequena que absorve o impacto de $c_3 \cdot n$ em comparação com $c_2 \cdot n^2$ quando n é grande.

Passo 2: Conclusão

Combinando os limites inferior e superior, temos:

$$c_1 \cdot n^2 \leq f(n) + g(n) \leq (c_2 + \epsilon) \cdot n^2 \quad \text{para todo } n \geq \max(n_0, n_1)$$

Isso mostra que $f(n) + g(n)$ está assintoticamente limitado acima e abaixo por uma constante múltipla de n^2, o que significa que:

$$f(n) + g(n) = \Theta(n^2)$$

10. Se $f(n) = \omega(g(n))$, então $f(n) = \Omega(g(n))$

    Vamos provar que se $f(n) = \omega(g(n))$, então $f(n) = \Omega(g(n))$.

Revisão das Definições

- **Definição de $\omega(g(n))$ (little-omega)**:

  $$f(n) = \omega(g(n)) \quad \text{significa que para qualquer constante positiva } c > 0, \text{ existe um val}$$

  $$f(n) > c \cdot g(n) \quad \text{para todo } n \geq n_0$$

- Em outras palavras, $f(n)$ cresce mais rapidamente que $g(n)$ e nunca é limitado superiormente por qualquer múltiplo constante de $g(n)$.
- **Definição de $\Omega(g(n))$ (Big-Omega)**:

  $$f(n) = \Omega(g(n)) \quad \text{significa que existe uma constante positiva } c > 0 \text{ e um valor } n_0 \text{ tal}$$

  $$f(n) \geq c \cdot g(n) \quad \text{para todo } n \geq n_0$$

- Isso implica que $f(n)$ é limitado inferiormente por um múltiplo constante de $g(n)$.

Passo 1: Assumir que $f(n) = \omega(g(n))$

Se $f(n) = \omega(g(n))$, então, por definição, para qualquer constante positiva c, existe um n_0 tal que:

$$f(n) > c \cdot g(n) \quad \text{para todo } n \geq n_0$$

Passo 2: Provar que $f(n) = \Omega(g(n))$

Para provar que $f(n) = \Omega(g(n))$, precisamos encontrar uma constante $c_1 > 0$ e um n_1 tal que:

$$f(n) \geq c_1 \cdot g(n) \quad \text{para todo } n \geq n_1$$

A partir da definição de $\omega(g(n))$, sabemos que $f(n) > c \cdot g(n)$ para qualquer constante $c > 0$. Em particular, podemos escolher $c = c_1$ para qualquer valor positivo de c_1, e haverá um valor $n_1 \geq n_0$ tal que:

$$f(n) > c_1 \cdot g(n) \quad \text{para todo } n \geq n_1$$

Isso satisfaz a condição para $f(n) = \Omega(g(n))$.

Passo 3: Conclusão

Portanto, se $f(n) = \omega(g(n))$, então $f(n) = \Omega(g(n))$. Isso ocorre porque a definição de $\omega(g(n))$ implica que $f(n)$ não apenas cresce mais rapidamente que $g(n)$, mas também é limitado inferiormente por um múltiplo constante de $g(n)$ para valores suficientemente grandes de n.

Questão 2

Resolva as equações de recorrência a seguir. Resolva usando Método da substituição e Teorema Mestre.

1. $$T(n) \leq T(\frac{n}{4}) + 2n$$

   Vamos resolver a equação de recorrência $T(n) \leq T\left(\frac{n}{4}\right) + 2n$ usando o Método da Substituição.

Passo 1: Suposição

Vamos supor que $T(n) \leq c \cdot n$, onde c é uma constante a ser determinada.

Passo 2: Substituição na Recorrência

Substituímos a suposição $T(n) \leq c \cdot n$ na recorrência:

$$T\left(\frac{n}{4}\right) \leq c \cdot \frac{n}{4}$$

Substituindo isso na equação original, obtemos:

$$T(n) \leq c \cdot \frac{n}{4} + 2n$$

Passo 3: Simplificação da Expressão

Agora, simplificamos a expressão:

$$T(n) \leq \frac{c}{4} \cdot n + 2n$$

Queremos garantir que nossa suposição $T(n) \leq c \cdot n$ seja verdadeira. Assim, precisamos que:

$$\frac{c}{4} \cdot n + 2n \leq c \cdot n$$

Dividimos ambos os lados por n (assumindo $n > 0$):

$$\frac{c}{4} + 2 \leq c$$

Passo 4: Isolando c

Agora, isolamos c:

$$2 \leq c - \frac{c}{4}$$

Para simplificar, escrevemos:

$$c - \frac{c}{4} = \frac{4c}{4} - \frac{c}{4} = \frac{3c}{4}$$

Então, temos:

$$2 \leq \frac{3c}{4}$$

Multiplicamos ambos os lados por 4:

$$8 \leq 3c$$

Finalmente, dividimos ambos os lados por 3:

$$c \geq \frac{8}{3}$$

Conclusão

Portanto, para $c = \frac{8}{3}$, a suposição $T(n) \leq c \cdot n$ é válida, o que implica que:

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

Vamos resolver a recorrência $T(n) \leq T\left(\frac{n}{4}\right) + 2n$ usando o Teorema Mestre.

Passo 1: Identificar os parâmetros da recorrência

A recorrência está na forma padrão para aplicação do Teorema Mestre:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Comparando com a recorrência dada:

$$T(n) \leq T\left(\frac{n}{4}\right) + 2n$$

Podemos identificar os valores dos parâmetros:

- $a = 1$ (pois há um único termo $T\left(\frac{n}{4}\right)$)
- $b = 4$ (porque estamos dividindo n por 4)
- $f(n) = 2n$

Passo 2: Calcular $n^{\log_b a}$

Para aplicar o Teorema Mestre, precisamos calcular $n^{\log_b a}$:

$$n^{\log_b a} = n^{\log_4 1}$$

Como $\log_4 1 = 0$ (qualquer número na base de logaritmo elevado a 0 é igual a 1):

$$n^{\log_4 1} = n^0 = 1$$

Passo 3: Comparar $f(n)$ com $n^{\log_b a}$

Agora, comparamos $f(n) = 2n$ com $n^{\log_b a} = 1$:

- $f(n) = 2n$ é uma função polinomial de ordem maior que $n^0 = 1$, ou seja, $f(n) = \Theta(n)$.

Passo 4: Aplicar o Teorema Mestre

O Teorema Mestre nos dá três casos para considerar:

- **Caso 1**: Se $f(n) = O(n^{\log_b a - \epsilon})$ para algum $\epsilon > 0$, então $T(n) = \Theta(n^{\log_b a})$.

- **Caso 2**: Se $f(n) = \Theta(n^{\log_b a} \cdot \log^k n)$, para algum $k \geq 0$, então $T(n) = \Theta(n^{\log_b a} \cdot \log^{k+1} n)$.
- **Caso 3**: Se $f(n) = \Omega(n^{\log_b a+\epsilon})$ para algum $\epsilon > 0$, e se $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$.

No nosso caso:

- $f(n) = \Theta(n)$ e $n^{\log_b a} = 1$.
- Como $\Theta(n)$ é maior que $n^0 = 1$, estamos no **Caso 3** do Teorema Mestre.

Portanto, a solução é:

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

Conclusão

Usando o Teorema Mestre, verificamos que a solução da recorrência $T(n) \leq T\left(\frac{n}{4}\right) + 2n$ é:

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

2.

$$4T(\frac{n}{2}) + cn$$

Vamos resolver a recorrência $T(n) = 4T\left(\frac{n}{2}\right) + cn$ usando o Teorema Mestre e o Método da Substituição.

Resolução pelo Teorema Mestre

A equação de recorrência está na forma padrão para aplicação do Teorema Mestre:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Comparando, temos:

- $a = 4$
- $b = 2$
- $f(n) = cn$, onde c é uma constante.

Para aplicar o Teorema Mestre, precisamos calcular o valor de $n^{\log_b a}$:

$$n^{\log_b a} = n^{\log_2 4}$$

Sabemos que $\log_2 4 = 2$, então:

$$n^{\log_2 4} = n^2$$

Agora, comparamos $f(n) = cn$ com $n^{\log_b a} = n^2$:

- $f(n) = cn$ é de ordem inferior a n^2, ou seja, $f(n) = O(n)$.

Aplicar o Teorema Mestre

O Teorema Mestre nos dá três casos para considerar:

- **Caso 1**: Se $f(n) = O(n^{\log_b a - \epsilon})$ para algum $\epsilon > 0$, então $T(n) = \Theta(n^{\log_b a})$.
- **Caso 2**: Se $f(n) = \Theta(n^{\log_b a} \cdot \log^k n)$, para algum $k \geq 0$, então $T(n) = \Theta(n^{\log_b a} \cdot \log^{k+1} n)$.
- **Caso 3**: Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ para algum $\epsilon > 0$, e se $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$.

No nosso caso:

- $f(n) = cn$ e $n^{\log_b a} = n^2$.

- Como $f(n) = O(n)$, que é de ordem inferior a n^2, estamos no **Caso 1** do Teorema Mestre.

Portanto, a solução é:

$$T(n) = \Theta(n^2)$$

Resolução pelo Método da Substituição

Vamos usar o Método da Substituição para verificar a solução obtida com o Teorema Mestre.

Passo 1: Supor a Forma da Solução

Vamos supor que $T(n) = O(n^2)$. Mais precisamente, vamos supor $T(n) = c \cdot n^2$, onde c é uma constante a ser determinada.

Passo 2: Substituir na Recorrência

Substituímos na recorrência:

$$T\left(\frac{n}{2}\right) = c \cdot \left(\frac{n}{2}\right)^2 = c \cdot \frac{n^2}{4}$$

Então, a recorrência se torna:

$$T(n) = 4 \cdot \frac{c \cdot n^2}{4} + cn = c \cdot n^2 + cn$$

Passo 3: Simplificar e Verificar

A expressão se simplifica para:

$$T(n) = c \cdot n^2 + cn$$

Podemos observar que o termo cn é de ordem inferior a n^2, então $T(n)$ é dominado por $c \cdot n^2$, confirmando que:

$$T(n) = \Theta(n^2)$$

Conclusão

Tanto o Teorema Mestre quanto o Método da Substituição nos dão a mesma solução:

$$T(n) = \Theta(n^2)$$

3.
$$T(n) = T(n-1) + N$$

Vamos resolver a recorrência $T(n) = T(n-1) + n$ usando o Método da Substituição.

Passo 1: Expandir a Recorrência

Começamos expandindo a recorrência para observar o padrão:

$$\begin{aligned}
T(n) &= T(n-1) + n \\
T(n-1) &= T(n-2) + (n-1) \\
T(n-2) &= T(n-3) + (n-2) \\
&\vdots \\
T(2) &= T(1) + 2
\end{aligned}$$

Somando todas as expressões, temos:

$$T(n) = T(1) + 2 + 3 + \ldots + (n-1) + n$$

Podemos ver que a soma dos termos $2 + 3 + \ldots + n$ é uma soma aritmética.

Passo 2: Soma Aritmética

A soma dos primeiros n números naturais $1 + 2 + 3 + \ldots + n$ é dada por:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

No nosso caso, a soma começa de 2 até n, portanto:

$$\sum_{k=2}^{n} k = \frac{n(n+1)}{2} - 1$$

Isso nos dá:

$$T(n) = T(1) + \frac{n(n+1)}{2} - 1$$

Passo 3: Considerando $T(1)$

Geralmente, $T(1)$ é uma constante que pode ser simplificada como $T(1) = c$. Assim, podemos simplificar a equação para:

$$T(n) = c + \frac{n(n+1)}{2} - 1$$

$$T(n) = \frac{n(n+1)}{2} + (c-1)$$

Como estamos interessados na ordem de crescimento assintótico, podemos ignorar as constantes e os termos de menor ordem:

$$T(n) = \Theta\left(\frac{n^2}{2}\right) = \Theta(n^2)$$

Conclusão

A solução da recorrência $T(n) = T(n-1) + n$ é:

$$T(n) = \Theta(n^2)$$

4.
$$T(n) = T(n-1)^2$$

Vamos resolver a recorrência $T(n) = T(n-1)^2$ usando o método da substituição e análise detalhada.

Análise Inicial

A recorrência dada, $T(n) = T(n-1)^2$, sugere um crescimento exponencial, uma vez que cada termo é o quadrado do termo anterior. Vamos expandir a recorrência para identificar um padrão e determinar a solução.

Expansão da Recorrência

Começamos expandindo a recorrência para os primeiros valores de n:

$$\begin{aligned}
T(n) &= T(n-1)^2 \\
T(n-1) &= T(n-2)^2 \\
T(n-2) &= T(n-3)^2 \\
&\vdots \\
T(2) &= T(1)^2
\end{aligned}$$

Substituindo cada expressão na anterior, obtemos:

$$T(n) = \left(T(n-2)^2\right)^2 = T(n-2)^{2^2} = T(n-2)^4$$

$$T(n) = \left(T(n-3)^2\right)^4 = T(n-3)^{2^3} = T(n-3)^8$$
$$\vdots$$
$$T(n) = T(1)^{2^{n-1}}$$

Conclusão

A partir da expansão, vemos que a solução geral para $T(n)$ é:

$$T(n) = T(1)^{2^{n-1}}$$

Essa expressão mostra que $T(n)$ cresce de forma extremamente rápida (exponencial em uma torre de expoentes). A solução final é:

$$T(n) = \Theta\left(2^{2^{n-1}}\right)$$

5.
$$T(n) = T(n/2) + 1$$

Vamos resolver a recorrência $T(n) = T\left(\frac{n}{2}\right) + 1$ usando o Teorema Mestre. Em seguida, vamos verificar a solução expandindo a recorrência para identificar o padrão.

Aplicação do Teorema Mestre

A equação de recorrência está no formato padrão para aplicação do Teorema Mestre:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Comparando com a recorrência dada:

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

Podemos identificar os valores dos parâmetros:

- $a = 1$ (pois há um único termo $T\left(\frac{n}{2}\right)$)
- $b = 2$ (porque estamos dividindo n por 2)
- $f(n) = 1$ (uma constante)

Para aplicar o Teorema Mestre, precisamos calcular $n^{\log_b a}$:

$$n^{\log_b a} = n^{\log_2 1}$$

Sabemos que $\log_2 1 = 0$, então:

$$n^{\log_2 1} = n^0 = 1$$

Agora, comparamos $f(n) = 1$ com $n^{\log_b a} = 1$:

- $f(n) = 1$ é equivalente a $n^0 = 1$.

Identificação do Caso do Teorema Mestre

O Teorema Mestre nos dá três casos:

- **Caso 1**: Se $f(n) = O(n^{\log_b a - \epsilon})$ para algum $\epsilon > 0$, então $T(n) = \Theta(n^{\log_b a})$.
- **Caso 2**: Se $f(n) = \Theta(n^{\log_b a} \cdot \log^k n)$, para algum $k \geq 0$, então $T(n) = \Theta(n^{\log_b a} \cdot \log^{k+1} n)$.
- **Caso 3**: Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ para algum $\epsilon > 0$, e se $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$.

No nosso caso:

- $f(n) = 1 = \Theta(1)$, e $n^{\log_b a} = 1$.
- Como $f(n) = \Theta(n^0 \cdot \log^0 n)$, estamos no **Caso 2** do Teorema Mestre.

Portanto, a solução é:

$$T(n) = \Theta(\log n)$$

Verificação por Expansão

Vamos expandir a recorrência para verificar o resultado.

$$
\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + 1 \\
T\left(\frac{n}{2}\right) &= T\left(\frac{n}{4}\right) + 1 \\
T\left(\frac{n}{4}\right) &= T\left(\frac{n}{8}\right) + 1 \\
&\vdots
\end{aligned}
$$

Após k expansões, temos:

$$T(n) = T\left(\frac{n}{2^k}\right) + k$$

Quando $\frac{n}{2^k} = 1$, temos $k = \log_2 n$, e portanto:

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

Como $T(1)$ é uma constante, $T(n)$ cresce de acordo com $\log n$, confirmando que:

$$T(n) = \Theta(\log n)$$

Conclusão

A solução da recorrência $T(n) = T\left(\frac{n}{2}\right) + 1$ é:

$$T(n) = \Theta(\log n)$$

6.

$$T(n) = T(n/2) + nlogn$$

Vamos resolver a recorrência $T(n) = T\left(\frac{n}{2}\right) + n \log n$ usando o Teorema Mestre.

Aplicação do Teorema Mestre

A equação de recorrência está na forma padrão para aplicação do Teorema Mestre:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Comparando com a recorrência dada:

$$T(n) = T\left(\frac{n}{2}\right) + n \log n$$

Podemos identificar os valores dos parâmetros:

- $a = 1$ (pois há um único termo $T\left(\frac{n}{2}\right)$)
- $b = 2$ (porque estamos dividindo n por 2)
- $f(n) = n \log n$

Para aplicar o Teorema Mestre, precisamos calcular $n^{\log_b a}$:

$$n^{\log_b a} = n^{\log_2 1}$$

Sabemos que $\log_2 1 = 0$, então:

$$n^{\log_2 1} = n^0 = 1$$

Agora, comparamos $f(n) = n \log n$ com $n^{\log_b a} = 1$:

- $f(n) = n \log n$ é maior que $n^0 = 1$, especificamente $f(n) = \Theta(n \log n)$.

Identificação do Caso do Teorema Mestre

O Teorema Mestre nos dá três casos:

- **Caso 1**: Se $f(n) = O(n^{\log_b a - \epsilon})$ para algum $\epsilon > 0$, então $T(n) = \Theta(n^{\log_b a})$.

- **Caso 2**: Se $f(n) = \Theta(n^{\log_b a} \cdot \log^k n)$, para algum $k \geq 0$, então $T(n) = \Theta(n^{\log_b a} \cdot \log^{k+1} n)$.
- **Caso 3**: Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ para algum $\epsilon > 0$, e se $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$.

No nosso caso:

- $f(n) = n \log n$ é maior que $n^{\log_b a} = 1$.
- Como $f(n) = \Theta(n \log n)$, estamos no **Caso 3** do Teorema Mestre.

Para verificar a condição de regularidade, precisamos verificar se:

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$$

Substituindo os valores:

$$f\left(\frac{n}{2}\right) = \frac{n}{2} \log \frac{n}{2} = \frac{n}{2}(\log n - 1)$$

$$f\left(\frac{n}{2}\right) = \frac{n \log n}{2} - \frac{n}{2}$$

Agora, comparando:

$$f\left(\frac{n}{2}\right) \approx \frac{n \log n}{2}$$

Isso confirma que:

$$f\left(\frac{n}{2}\right) \leq \frac{1}{2} f(n)$$

Portanto, a condição de regularidade é satisfeita, e podemos aplicar o **Caso 3** do Teorema Mestre.

A solução é:

$$T(n) = \Theta(n \log n)$$

Conclusão

A solução da recorrência $T(n) = T\left(\frac{n}{2}\right) + n \log n$ é:

$$T(n) = \Theta(n \log n)$$

Questão 3

Encontre a equação de recorrência dos algoritmos a seguir:

Algorithm 1 FIND-MAX

```
Procedure FIND-MAX(a[], first, last)
if first == last then
    return a[first]
end if
mid ← first + (last − first)/2
left ← FIND-MAX(a, first, mid)
right ← FIND-MAX(a, mid + 1, last)
return MAX(left, right)
```

Algorithm 2 SELECTION-SORT

```
Procedure SELECTION-SORT(a[], first, last)
if first ≥ last then
    return
end if
min_index ← FIND-MIN(a, first, last)
SWAP(a[min_index], a[first])
SELECTION-SORT(a, first + 1, last)
```

Equações de Recorrência dos Algoritmos

Vamos encontrar as equações de recorrência para os dois algoritmos apresentados: `FIND-MAX` e `SELECTION-SORT`. Explicaremos detalhadamente como chegamos a cada resultado.

1. Algoritmo `FIND-MAX`

Análise do Algoritmo

O algoritmo `FIND-MAX` é um exemplo clássico de divisão e conquista (divide-and-conquer). A ideia central é dividir o problema de encontrar o máximo de um array em duas partes, encontrar o máximo de cada parte recursivamente, e depois comparar os dois resultados.

Vamos analisar o algoritmo passo a passo:

- **Caso Base**: Quando o subarray tem um único elemento (ou seja, first == last), o algoritmo retorna esse elemento. Isso é uma operação constante, $O(1)$.
- **Divisão**: O array é dividido em duas metades. O ponto médio é calculado com:
$$\text{mid} \leftarrow \text{first} + \frac{\text{last} - \text{first}}{2}$$
- **Conquista**: O algoritmo é chamado recursivamente para as duas metades:
$$\texttt{FIND-MAX}(a, \text{first}, \text{mid})$$
$$\texttt{FIND-MAX}(a, \text{mid} + 1, \text{last})$$
- **Combinação**: Finalmente, o algoritmo compara os máximos das duas metades com uma operação $\text{MAX}(\text{left}, \text{right})$, que é uma operação constante, $O(1)$.

Definição da Recorrência

Seja $T(n)$ o tempo necessário para encontrar o máximo de um array de tamanho n.

- **Divisão**: O array é dividido em duas partes de aproximadamente $n/2$ cada. Portanto, o tempo de divisão é dado por:
$$T\left(\frac{n}{2}\right)$$
- **Conquista**: O tempo total é a soma do tempo necessário para encontrar o máximo em cada metade:
$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right)$$

- **Combinação**: Comparar os dois valores máximos de cada metade é uma operação $O(1)$.

Portanto, a equação de recorrência para o algoritmo `FIND-MAX` é:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$$

2. Algoritmo `SELECTION-SORT`

Análise do Algoritmo

O algoritmo `SELECTION-SORT` é um algoritmo de ordenação que, a cada iteração, seleciona o menor elemento de uma sublista não ordenada e o move para o início. O algoritmo funciona recursivamente, reduzindo o tamanho da sublista a cada chamada recursiva.

Vamos analisar o algoritmo passo a passo:

- **Caso Base**: Quando o subarray tem tamanho 0 ou 1 (ou seja, first $\geq$ last), o algoritmo retorna, pois não há nada para ordenar. Isso é uma operação constante, $O(1)$.
- **Encontrar o Mínimo**: O algoritmo usa uma função `FIND-MIN` para encontrar o índice do menor elemento no subarray. O tempo para encontrar o mínimo em um subarray de tamanho n é $O(n)$.
- **Troca**: Depois de encontrar o mínimo, o algoritmo troca o menor elemento com o primeiro elemento do subarray. A operação de troca é $O(1)$.
- **Recursão**: O algoritmo chama a si mesmo para ordenar o restante do subarray, que tem tamanho $n - 1$:

$$\texttt{SELECTION-SORT}(a, \text{first} + 1, \text{last})$$

Definição da Recorrência

Seja $T(n)$ o tempo necessário para ordenar um array de tamanho n usando `SELECTION-SORT`.

- **Encontrar o Mínimo**: O tempo necessário para encontrar o mínimo é $O(n)$.

- **Troca**: A operação de troca é $O(1)$.
- **Recursão**: O tempo total necessário para ordenar o restante do array é $T(n-1)$.

Portanto, a equação de recorrência para o algoritmo `SELECTION-SORT` é:

$$T(n) = T(n-1) + O(n)$$

Resumo

As equações de recorrência para os algoritmos são:

- `FIND-MAX`: $T(n) = 2T\left(\frac{n}{2}\right) + O(1)$
- `SELECTION-SORT`: $T(n) = T(n-1) + O(n)$