Pontifícia Universidade Católica do Paraná
Escola Politécnica / Bacharelado em Ciência da Computação
Disciplina de Complexidade de Algoritmos / Prof. Edson Emilio Scalabrin
Revisão/simulado: 25/04/2024

Exercício 01: Assumindo que a equação de recorrência a seguir está correta para os números de Fibonacci:

$$T(n) = \begin{cases} c_0 & se\ n = 0\ ou\ 1 \\ T(n-1) + T(n-2) + c_1 & se\ n > 1 \end{cases}$$

Podemos simplificar o processo de resolução da recorrência, considerando que o que nos interessa é um limite superior:

$$T(n) \leq 2T(n-1) + c_1 \quad se\ n > 1$$

Desdobrando T(n) em:

$$\begin{aligned} T(n) &\leq 2T(n-1) + c_1 \\ &\leq 2(2T(n-2) + c_1) + c_1 \\ &\leq 2(2(2T(n-3) + c_1) + c_1) + c_1 \\ &\leq 2^3T(n-3) + 2^2c_1 + 2^1c_1 + 2^0c_1 \\ &\ldots \\ &\leq 2^kT(n-k) + \sum_{i=0}^{k-1} 2^i c_1 \\ &\leq 2^kT(n-k) + c_1 \sum_{i=0}^{k-1} 2^i \end{aligned}$$

Assumindo que chamadas recursivas terminam quando n é igual 0 ou 1. E como se está procurando um limite superior, considera-se 0. Tem-se a seguinte relação: $n - k = 0 \leftrightarrow k = n$

Pode-se reescrever:

$$T(n) \leq 2^kT(n-k) + c_1 \sum_{i=0}^{k-1} 2^i$$

$$= 2^nT(n-n) + c_1 \sum_{i=0}^{n-1} 2^i$$

Sabe-se que:

$$\sum_{i-0}^{n-1} r^i = \frac{r^n - 1}{r - 1} \qquad r \neq 1$$

Resolva a recorrência a seguir:

$$a)\ T(n) = 2^nT(n-n) + c_1 \sum_{i=0}^{n-1} 2^i \in O(2^n)$$

$$T(n) = 2^nT(0) + c_1\left(\frac{2^n - 1}{2 - 1}\right)$$

$$T(n) = 2^nc_0 + c_1(2^n - 1)$$

$$T(n) = 2^nc_0 + 2^nc_1 - c_1$$

$$Resposta: T(n) = 2^n(c_0 + c_1) - c_1$$

Exercício 02: Dado o programa em Python fictício a seguir, encontre a sua equação de recorrência. Caso você faça alguma simplificação ou consideração, descreva-a

```python
def fatorial( n ):
    if n == 0:
        return 1
    else:
        return (n * fatorial(n - 1))

def A2(A):
    n = len(A)
    m = n/5
    if (n <= 0): return m
    else:
        x = 0
        for b in A:
            x = x + fatorial(b)
        print(x)
        A2(A[0 : m])
        A2(A[m : 2 * m])
        A2(A[2 * m : 3 * m])
        A2(A[3 * m : n])
```

Fatorial

$$T(n) = \begin{cases} 1 & se\ n = 0 \\ T(n-1) + 1 & se\ n \geq 1 \end{cases}$$

Equação de recorrência de A2:

$$T(n) = \begin{cases} 4 & se\ n = 0 \\ 3T\left(\frac{n}{5}\right) + T\left(\frac{2n}{5}\right) + n(b+1) + 5 & se\ n \geq 1 \end{cases}$$

Deve-se notar que "b" representa um valor constante na lista A e não se tem nenhuma precisão sobre a sua magnitude.

Exercício 03: Para cada equação de recorrência a seguir, calcule a altura máxima—ou o ramo mais longo—da árvore de recursão.

a) $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$

A altura máxima da árvore é $\log_{\frac{3}{2}} n$

b) $T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{11n}{10}\right) + n^3$

A altura máxima da árvore é ∞ e não há solução, pois o tamanho do problema nunca diminui.

Exercício 04: Dados os somatórios abaixo, encontre as fórmulas suas fechadas aplicando a propriedade **telescópica.**

$$(a) \sum_{i=3}^{\infty} \left(\frac{3}{i^2} - \frac{3}{(i+1)^2}\right)$$

Soma parcial de 3 até N

$$S_n = \sum_{i=3}^{N} \left(\frac{3}{i^2} - \frac{3}{(i+1)^2}\right)$$

$$S_n = \left(\frac{3}{3^2} - \frac{3}{4^2}\right) + \left(\frac{3}{4^2} - \frac{3}{5^2}\right) + \left(\frac{3}{5^2} - \frac{3}{6^2}\right) + \cdots + \left(\frac{3}{(N-2)^2} - \frac{3}{(N-1)^2}\right) + \left(\frac{3}{(N-1)^2} - \frac{3}{N^2}\right) + \left(\frac{3}{N^2} - \frac{3}{(N+1)^2}\right)$$

$$S_n = \left(\frac{3}{3^2} - \frac{3}{(N+1)^2}\right) \qquad \lim_{N\to\infty} \left(\frac{3}{3^2} - \frac{3}{(N+1)^2}\right) = \frac{1}{3}$$

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

Reescrever na forma de uma diferença

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

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

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

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

$$A=1$$

$$\sum_{n=1}^{\infty}\left(\frac{1}{(n+1)}-\frac{1}{(n+2)}\right)$$

Soma parcial de 1 até N

$$S_n=\sum_{n=1}^{N}\left(\frac{1}{(n+1)}-\frac{1}{(n+2)}\right)$$

$$S_n=\left(\frac{1}{2}-\frac{1}{3}\right)+\left(\frac{1}{3}-\frac{1}{4}\right)+\left(\frac{1}{4}-\frac{1}{5}\right)+\cdots+\left(\frac{1}{(N-1)}-\frac{1}{N}\right)+\left(\frac{1}{N}-\frac{1}{(N+1)}\right)+\left(\frac{1}{(N+1)}-\frac{1}{(N+2)}\right)$$

$$S_n=\left(\frac{1}{2}-\frac{1}{(N+2)}\right)\qquad \lim_{N\to\infty}\left(\frac{1}{2}-\frac{1}{(N+2)}\right)=\frac{1}{2}$$

Exercício 05: O método da substituição pode ser usado para estabelecer limites superiores ou inferiores em uma recorrência. Como exemplo, vamos determinar um limite superior na recorrência

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

Vamos supor que a solução é $T(n) = O(n \lg n)$. O método consiste em provar que $T(n) \leq cn \lg n$ para uma escolha apropriada da constante $c > 0$. Assume-se que este limite vale para $\frac{n}{2}$, ou seja, que $T(n/2) \leq c\frac{n}{2}\lg(\frac{n}{2})$. Substituindo na recorrência, resolva:

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

$$T(n) = 2c\frac{n}{2}\log_2 n - 2c\frac{n}{2}\log_2 2 + n$$

$cn\log_2 n \geq cn\log_2 n - cn + n$ substituir $T(n)$ pela solução $cn\log_2 n$ que

$cn\log_2 n - cn\log_2 n \geq -cn + n$

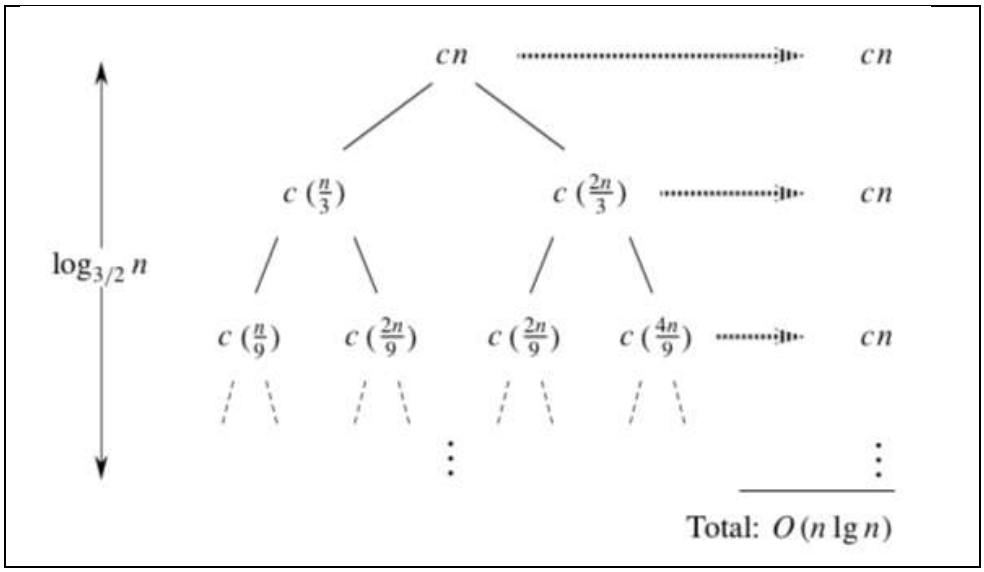
$0 \geq -cn + n$

$cn \geq n$

$c \geq 1$ é verdade para c maior igual 1

Exercício 06: Considerando a árvore resultante da equação a seguir que é desbalanceada.

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$$



Espera-se que a solução para a recorrência seja, no máximo, o resultado do número de níveis multiplicado pelo custo de cada nível, ou seja, O(cn $\log_{3/2}$ n) = O(n log n). No entanto, é importante notar que esta árvore de recursão não é uma árvore binária completa, o que implica que possui menos do que $n^{\log_{3/2} 2}$ folhas. Além disso, à medida que se navega a partir da raiz, torna-se evidente que mais e mais nós internos estão ausentes. Isso significa que nem todos os níveis contribuem com um custo exato de *cn*; os níveis mais baixos contribuem menos. Embora seja possível calcular uma contabilidade precisa de todos os custos, para efeito de exercício estamos simplesmente tentando criar uma estimativa para usar no método da substituição. Em outras palavras, vamos tolerar uma certa imprecisão e tentar demonstrar que um palpite de O(n log n) para o limite superior está correto. Pode-se usar o método de substituição para verificar que O(n lg n) é um limite superior para a solução da recorrência.

Mostre que T (n) ≤ **d**n lg n, onde d é uma constante positiva adequada. Resolva:

$$T(n) = d\frac{n}{3}\log\frac{n}{3} + d\frac{2n}{3}\log\frac{2n}{3} + cn$$

$$T(n) = d\left(\frac{n}{3}\log n - \frac{n}{3}\log 3\right) + d\left(\frac{2n}{3}\log n + \frac{2n}{3}\log 2 - \frac{2n}{3}\log 3\right) + cn$$

$$T(n) = d\left(\frac{n}{3} + \frac{2n}{3}\right)\log n + d\left(-\frac{n}{3} - \frac{2n}{3}\right)\log 3 + d\frac{2n}{3}\log 2 + cn$$

$$T(n) = dn\log n - dn\log 3 + d\frac{2n}{3}\log_2 2 + cn$$

$$T(n) = dn\log n - dn\log 3 + d\frac{2n}{3} + cn = dn\log n - dn\left(\log 3 - \frac{2}{3}\right) + cn$$

Verificar se está $O(n \log n)$

$$T(n) \leq dn\log n$$

$$dn\log n - dn\left(\log 3 - \frac{2}{3}\right) + cn \leq dn\log n$$

$$cn \leq dn\log n - dn\log n + dn\left(\log 3 - \frac{2}{3}\right)$$

$$cn \leq dn\left(\log 3 - \frac{2}{3}\right)$$

$$\frac{cn}{n\left(\log 3 - \frac{2}{3}\right)} \leq d$$

É verdade para $d \geq \frac{c}{\left(\log 3 - \frac{2}{3}\right)}$

Exercício 07: Use o método mestre para fornecer limites assintóticos precisos para as seguintes recorrências.

a. $T(n) = 4T(\frac{n}{2}) + n$

$a = 4, b = 2, k = 1$

log **a** / log **b** = **2**

Caso 1: se log **a** / log **b** > k então T está em $\Theta(n^{\log a / \log b})$

Resposta: $\Theta(n^2)$

b. $T(n) = 4T(\frac{n}{2}) + n^2$

$a = 4, b = 2, k = 2$

log **a** / log **b** = **2**

Caso 2: se log **a** / log **b** = k então T está em $\Theta(n^2 \log n)$

Resposta: $\Theta(n^2 \log n)$

(c) $T(n) = 4T(\frac{n}{2}) + n^3$

$a = 4, b = 2, k = 3$

log **a** / log **b** = **2**

Caso 3: se log **a** / log **b** < k então T está em $\Theta(n^2 \log n)$

Resposta: $\Theta(n^3)$

Como se trata da aplicação do Caso 3, precisa-se ainda mostrar que a condição de regularidade é válida para f(n). E se **a**f (n / **b**) ≤ **c**f (n) para uma constante **c** < 1 e para todo n suficientemente grande, então T(n) ∈ Θ f (n))

$$af(\frac{n}{b}) \leq c_2 f(n)$$

$$4f(c) \leq c_2 f(n)$$

$$4\left(\frac{n}{2}\right)^3 \leq c_2 n^3$$

$$4\left(\frac{n^3}{2^3}\right) \leq c_1 n^3$$

$$4\left(\frac{1}{8}\right) \leq c_2$$

$$\frac{4}{8} \leq c_2$$

$$\frac{1}{2} + \text{delta1} \leq c_2 \quad \text{onde delta1} \geq 0$$

$$\frac{1}{2} \geq c_1 - \text{delta2} \quad \text{onde delta2} \; [0, \frac{1}{2})$$

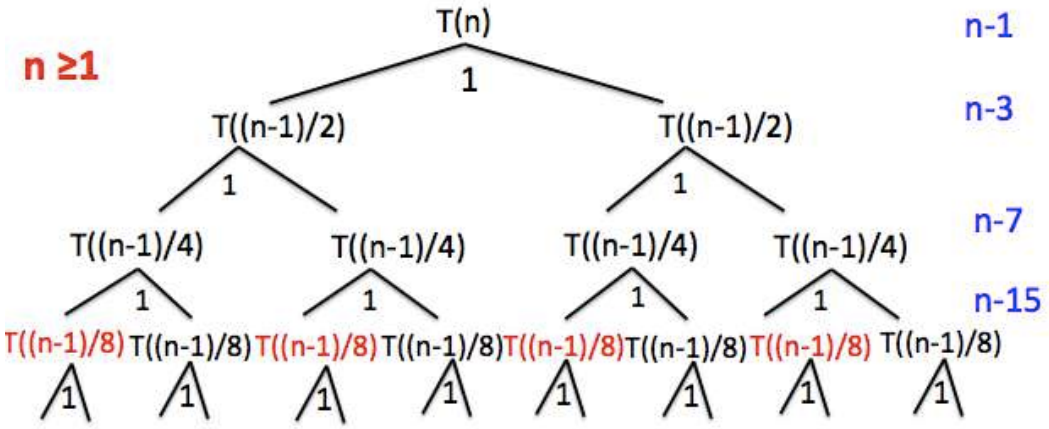
$$n_0 = 1$$

$$c_1 f(n) \leq f(c) \leq c_2 f(n)$$

Exercício 08: Dados os algoritmos a seguir, encontre para cada um deles a sua equação de recorrência para o cálculo de tempo de execução.

```
def qs(array):
    if len(array) <= 1:
        return array
    else:
        pivot = array[0]
        menores = [x for x in array[1:] if x <= pivot]
        maiores = [x for x in array[1:] if x > pivot]
        return qs(menores) + [pivot] + qs(maiores)
```

$$T(n) = \begin{cases} 1 & se\ n \leq 1 \\ 2T\left(\frac{n-1}{2}\right) + n - 1 & se\ n > 1 \end{cases}$$



$$soma = (n-1) + (n-3) + (n-7) + (n-15) + \cdots$$

$$soma = \sum_{i=0}^{\log_2 n} (n - (2^i - 1))$$

$$soma = n\log_2 n + \log_2 n - \sum_{i=0}^{\log_2 n} 2^i$$

Não é infinito

Sabe-se que:

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1}$$

$$\sum_{i=0}^{\log_2 n} 2^i = \frac{2^{\log_2 n + 1} - 1}{2 - 1} = \frac{2^{\log_2 n + 1} - 2^0}{2 - 1} = \frac{2.2^{\log_2 n} - 1}{1} = 2n - 1$$

$$soma = n\log_2 n + \log_2 n - (2n - 1) = O(n\log_2 n)$$

```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

$$T(n) = \begin{cases} 1 & se\ n \leq 1 \\ T(n-1) + T(n-2) + 1 & se\ n > 1 \end{cases}$$

```
def potencia(base, expoente):
    if expoente == 0:
        return 1
    else:
        return base * potencia(base, expoente - 1)
```

$$T(n) = \begin{cases} 1 & se\ n \leq 1 \\ T(n-1) + 1 & se\ n > 1 \end{cases}$$

```
def torres_hanoi(n, origem, destino, auxiliar):
    if n == 1:
        print("Mova o disco 1 de", origem, "para", destino)
        return
    torres_hanoi(n-1, origem, auxiliar, destino)
    print("Mova o disco", n, "de", origem, "para", destino)
    torres_hanoi(n-1, auxiliar, destino, origem)
```

$$T(n) = \begin{cases} 3 & se\ n = 1 \\ 2T(n-1) + 2 & se\ n > 1 \end{cases}$$

$$\begin{aligned} T(n) &\leq 2T(n-1) + 2 \\ &\leq 2(2T(n-2) + 2) + 2 \\ &\leq 2(2(2T(n-3) + 2) + 2) + 2 \\ &\leq 2^3 T(n-3) + 2^2 2 + 2^1 2 + 2^0 2 \\ &\dots \\ &\leq 2^k T(n-k) + \sum_{i=0}^{k-1} 2^i 2 \\ &\leq 2^k T(n-k) + 2\sum_{i=0}^{k-1} 2^i \end{aligned}$$

Assumindo que chamadas recursivas terminam quando n é igual 0 ou 1. E como se está procurando um limite superior, considera-se 0. Tem-se a seguinte relação: $n - k = 0 \leftrightarrow k = n$

Pode-se reescrever:

$$T(n) \leq 2^k T(n-k) + 2\sum_{i=0}^{k-1} 2^i$$

$$T(n) = 2^n T(n-n) + 2\sum_{i=0}^{n-1} 2^i$$

$$T(n) = 2^n T(0) + 2\left(\frac{2^n - 1}{2 - 1}\right)$$

$T(n) = 3.2^n + 2(2^n - 1)$ onde o ponto entre 3 e 2 é multiplicação.

$T(n) = 3.2^n + 2.2^n - 2$ onde o ponto entre 3 e 2 é multiplicação.

$Resposta: T(n) = 2^n(3 + 2) - 2 = 5.2^n - 2$ onde o ponto entre 5 e 2 é multiplicação.

Sabe-se que:

$$\sum_{i-0}^{n-1} r^i = \frac{r^n - 1}{r - 1} \qquad r \neq 1$$