Análise de Algoritmos Recursivos
Recorrências

José Elias Claudio Arroyo

Departamento de Informática
Universidade Federal de Viçosa

INF332 - 2022/2

Outline

Plano para Análisar **Algoritmos Recursivos**

- Identificar o **parâmetro de entrada** n.
- Identificar a **operação básica** do algoritmo.
- Verificar se é necessário fazer uma análise de **melhor caso** e **pior caso**.
- Escrever a **relação de recorrência** que indica o número de vezes que operação básica é executada.
- Identificar a **condição de parada** do algoritmo (**caso base**).
- **Resolver a recorrência** para obter uma fórmula fechada**.**
- Finalmente, indicar a classe de eficiência do algoritmo, através das notações assintóticas.

Exemplo #1: Calculando *n*! (**Recursivo**)

```
FATORIAL ( n )
{
if (n==0): return 1
return     n * FATORIAL(n-1)
}
```

- **Tamanho da entrada**: n
- **Operação básica**: multiplicação
- **Número de multiplicações realizadas pelo algoritmo**:
  $\Rightarrow$ Definir a Relação de Recorrência

Exemplo #1: Calculando $n!$ (**Recursivo**)

```
FATORIAL ( n )
{
if (n==0): return 1
return     n * FATORIAL(n-1)
}
```

- Já que $n! = n \times (n-1)!$
- $\Rightarrow$ $fatorial(n) = n \times fatorial(n-1)$
- $\Rightarrow$ o número de multiplicacoes é: $T(n) = 1 + T(n-1)$
- **Caso base** (condição de parada): $T(0) = 0$ (se $n = 0$ nenhuma multiplicação é feita).

Exemplo #1: Calculando $n!$ (**Recursivo**)

- **Relação de recorrência**: $\begin{cases} T(n) = 1 + T(n-1), & n > 0 \\ T(0) = 0 \end{cases}$

Exemplo #1: Calculando $n!$ (**Recursivo**)

- **Relação de recorrência**: $\begin{cases} T(n) = 1 + T(n-1), & n > 0 \\ T(0) = 0 \end{cases}$
- Resolução da recorrência:

$$
\begin{aligned}
T(n) &= 1 + T(n-1) \\
&= 1 + 1 + T(n-2) = 2 + T(n-2) \\
&= 2 + 1 + T(n-3) = 3 + T(n-3) \\
&\ldots \\
&= \qquad\qquad\qquad\qquad = n + T(0)\,.
\end{aligned}
$$

- **Relação de recorrência**: $\begin{cases} T(n) = 1 + T(n-1), & n > 0 \\ T(0) = 0 \end{cases}$
- Resolução da recorrência:

$$\begin{aligned} T(n) &= 1 + T(n-1) \\ &= 1 + 1 + T(n-2) = 2 + T(n-2) \\ &= 2 + 1 + T(n-3) = 3 + T(n-3) \\ &\dots \\ &= \qquad\qquad\qquad = n + T(0)\,. \end{aligned}$$

Como $T(0) = 0$, $\Rightarrow T(n) = n$ (fórmula fechada).

- Algoritmo recursivo FATORIAL é: $\Theta(n)$ (classe de eficiência)

Método da Substituição em Retrocesso (ou Substituições Sucessivas)

Se $T(n)$ é definido em função de $T(n-1)$:

- Determine $T(n-1)$ em função de $T(n-2)$, então $T(n-2)$ em função de $T(n-3)$, assim sucessivamente **expandir a recorrência** até que possa ser detectado o seu **comportamento no caso geral** $T(n-k)$.
- **Determine o valor de** k de forma que se atinja o **caso base**,
- Determine a fórmula fechada para $T(n)$ (solução da recorrência).

Exemplo: Determine a complexidade de tempo do seguinte algoritmo

```
ALGORITMO ( n ):
   if (n == 0): return
   else:
      ALGORITMO(n - 1)
      for i = 1 to n:
         print('*')
```

Exemplo: Determine a complexidade de tempo do seguinte algoritmo

```
ALGORITMO ( n ):
  if (n == 0): return
  else:
    ALGORITMO(n - 1)
    for i = 1 to n:
      print('*')
```

- **Operação básica**: print('*')
- $\Rightarrow$ o número de impressões é: $T(n) = T(n-1) + n$
- Condição de parada (**Caso base**): $T(0) = 0$ (se $n = 0$ nenhuma impressão).

Resolução de Relações de Recorrência

Resolvendo a recorrência

$$T(n) = \begin{cases} T(n-1) + n, & \text{para } n > 0. \\ 0, & \text{para } n = 0. \end{cases}$$

Resolução de Relações de Recorrência

Resolvendo a recorrência

$$T(n) = \begin{cases} T(n-1) + n, & \text{para } n > 0. \\ 0, & \text{para } n = 0. \end{cases}$$

Solução

$$T(n) = \mathbf{T(n\text{-}1)} + n$$
$$= \mathbf{T(n\text{-}2)} + (n-1) + n$$
$$= \mathbf{T(n\text{-}3)} + (n-2) + (n-1) + n$$

Generalizando após k passos:

$$= T(n-k) + (n-(k-1)) + \cdots + (n-1) + n$$

Para $k = n$, temos $T(n-k) = T(0)$

$$= 0 + 1 + 2 + \cdots + n = \frac{n(n+1)}{2}, (f.fechada).$$

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

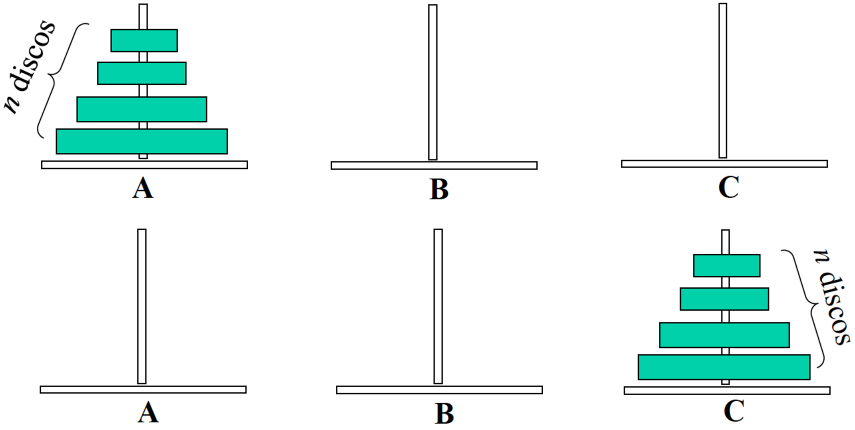
Exercício: Resolva a recorrência a seguir

$$T(n) = \begin{cases} T(n/2) + 3, & \text{para } n > 0. \\ 1, & \text{para } n = 1. \end{cases}$$

Algoritmos Recursivos - Exemplo #2: Torres de Hanoi

- Existem três pinos (A, B e C) e n discos de tamanhos diferentes (disco 1 é o menor e disco n é o maior).
- Todos os discos estão empilhados no pino A, em ordem decrescente de tamanho, de baixo para cima.
- O objetivo é mover todos os n discos do pino A para o pino C, utilizando o pino B (pino auxiliar).
- Deve-se mover apenas um disco por vez.
- Um disco nunca pode ficar em cima de um disco menor.
- Realizar o menor número de movimentos.

Algoritmos Recursivos - Exemplo #2: Torres de Hanoi

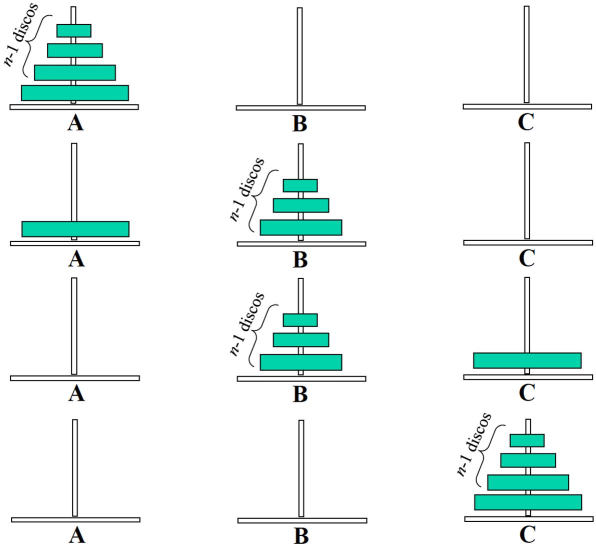


Algoritmos Recursivos - Exemplo #2: Torres de Hanoi

Passo 1

Passo 2

Passo 3



Algoritmos Recursivos - Exemplo #2: Torres de Hanoi

Algoritmo para resolver o problema das torres de Hanoi:

Se $n = 1$: mova o único disco de A para C.
Senão (i.e. $n > 1$):

1. Mova recursivamente $n - 1$ discos de A para B;
2. Mova o maior disco (disco n) de A para C;
3. Mova recursivamente $n - 1$ discos de B para C;

Algoritmos Recursivos - Exemplo #2: Torres de Hanoi

Suponha que os discos são **enumerados**: o menor é 1,..., o maior é *n*.

Algoritmo para mover os *n* discos do pino **A** para o pino **C** usando o pino **B**.

```
HANOI(n, A, C, B)
{
   if (n==1): Mova o disco 1 de A para C
   else{
       HANOI(n-1, A, B, C)
       Mova o disco n de A para C
       HANOI(n-1, B, C, A)
   }
}
```

Algoritmos Recursivos - Exemplo #2: Torres de Hanoi

- **Tamanho da entrada**: n (número de discos)
- **Operação básica**: mover um disco
- **Número de movimentos**:

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

- **Relação de recorrência**:

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

Resolvendo a recorrência:

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

Resolvendo a recorrência:

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

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

Algoritmos Recursivos - Exemplo #2: Torres de Hanoi

Resolvendo a recorrência:

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

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

Resolvendo a recorrência:

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

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

Resolvendo a recorrência:

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

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

....

Algoritmos Recursivos - Exemplo #2: Torres de Hanoi

Resolvendo a recorrência:

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

$$\begin{aligned} T(n) &= 2 \times T(n-1) + 1 \\ &= 2(2 \times T(n-2) + 1) + 1 \\ &= 2^2 \times T(n-2) + 2 + 1 \\ &= 2^2(2 \times T(n-3) + 1) + 2 + 1 \\ &= 2^3 \times T(n-3) + 2^2 + 2 + 1 \\ &\dots\dots\dots\dots\dots\dots \\ &= 2^k \times T(n-k) + 2^{k-1} + \cdots 2 + 1 \quad \text{(generalizando)} \end{aligned}$$

Resolvendo a recorrência:

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

$$\begin{aligned}
T(n) &= 2 \times T(n-1) + 1 \\
&= 2(2 \times T(n-2) + 1) + 1 \\
&= 2^2 \times T(n-2) + 2 + 1 \\
&= 2^2(2 \times T(n-3) + 1) + 2 + 1 \\
&= 2^3 \times T(n-3) + 2^2 + 2 + 1 \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&= 2^k \times T(n-k) + 2^{k-1} + \cdots 2 + 1 \quad \text{(generalizando)} \\
&= 2^{n-1} \times T(1) + 2^{n-2} + \cdots + 2 + 1 \quad (k = n-1)
\end{aligned}$$

Algoritmos Recursivos - Exemplo #2: Torres de Hanoi

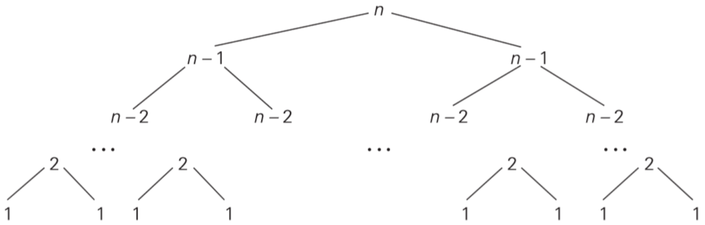
Resolvendo a recorrência:

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

$$\begin{aligned}
T(n) &= 2 \times T(n-1) + 1 \\
&= 2(2 \times T(n-2) + 1) + 1 \\
&= 2^2 \times T(n-2) + 2 + 1 \\
&= 2^2(2 \times T(n-3) + 1) + 2 + 1 \\
&= 2^3 \times T(n-3) + 2^2 + 2 + 1 \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&= 2^k \times T(n-k) + 2^{k-1} + \cdots 2 + 1 \quad \text{(generalizando)} \\
&= 2^{n-1} \times T(1) + 2^{n-2} + \cdots + 2 + 1 \quad (k = n-1) \\
&= 2^{n-1} + 2^{n-2} + \cdots + 2^1 + 2^0 = \sum_{i=0}^{n-1} 2^i = 2^n - 1 \Rightarrow \Theta(2^n).
\end{aligned}$$

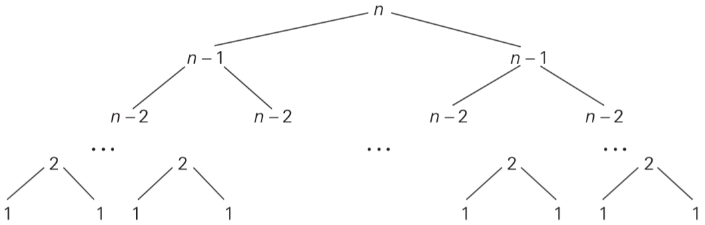
- A **análise de algoritmos recursivos** (e resolução das relações de recorrências) também pode ser feita através da **Árvore de Chamadas Recursivas**.

Árvore de chamadas recursivas do algoritmo Hanoi



Árvore de Chamadas Recursivas

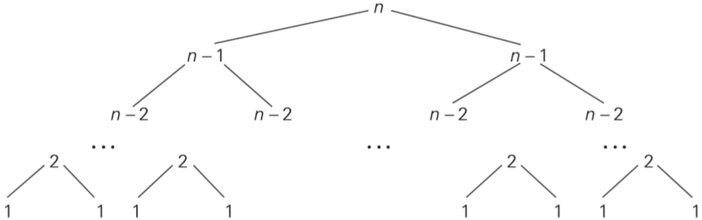
Árvore de chamadas recursivas do algoritmo Hanoi



- Cada nó da árvore corresponde a uma chamada recursiva.
- Para cada chamada, o algoritmo faz **1 movimento**.
- Nível 0 $\Longrightarrow 1 = 2^0$ movimento;
- Nível 1 $\Longrightarrow 2 = 2^1$ movimentos;
- Nível 2 $\Longrightarrow 4 = 2^2$ movimentos;
- ...
- Nível h $\Longrightarrow 2^h$ movimentos.

Árvore de Chamadas Recursivas

Árvore de chamadas recursivas do algoritmo Hanoi

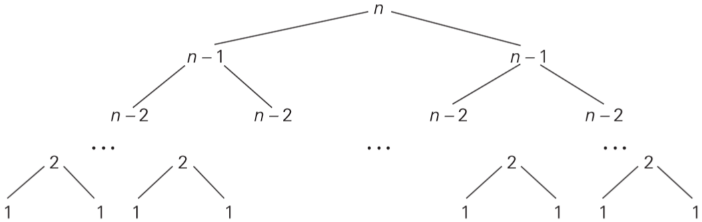


- Cada nó da árvore corresponde a uma chamada recursiva.
- Para cada chamada, o algoritmo faz **1 movimento.**
- Nível 0 $\Longrightarrow 1 = 2^0$ movimento;
- Nível 1 $\Longrightarrow 2 = 2^1$ movimentos;
- Nível 2 $\Longrightarrow 4 = 2^2$ movimentos;
- ...
- Nível h $\Longrightarrow 2^h$ movimentos.

Note que a **altura da árvore** é: $h = n - 1$.

Árvore de Chamadas Recursivas

Árvore de chamadas recursivas do algoritmo Hanoi



$\Longrightarrow$ Número total de movimentos:

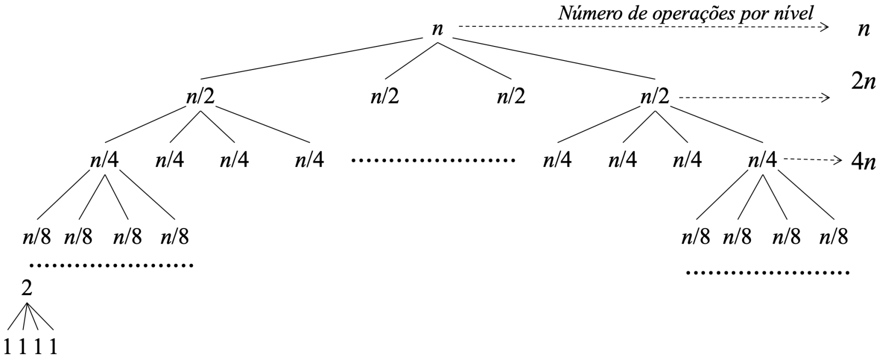
- $T(n) = 2^0 + 2^1 + ... + 2^{n-1} = \sum_{i=0}^{n-1} 2^i = 2^n - 1$.
- $\Rightarrow$ o algoritmo é $\Theta(2^n)$.

Árvore de Chamadas Recursivas

Resolva a recorrência: $T(1) = 1$, $T(n) = 4T(n/2) + n$, $n > 1$.

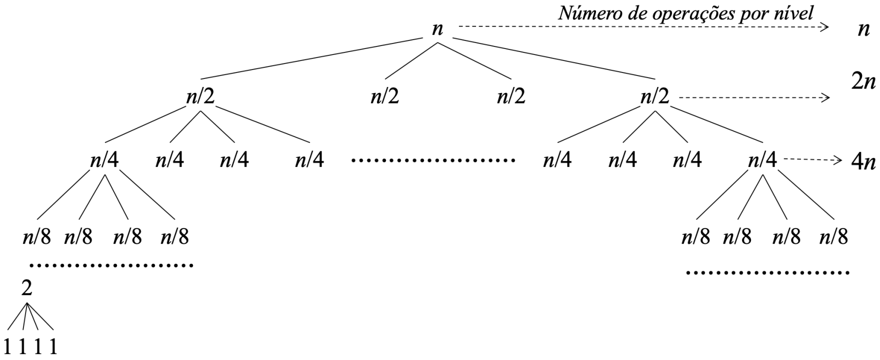
Árvore de Chamadas Recursivas

Resolva a recorrência: $T(1) = 1$, $T(n) = 4T(n/2) + n$, $n > 1$.



Árvore de Chamadas Recursivas

Resolva a recorrência: $T(1) = 1$, $T(n) = 4T(n/2) + n$, $n > 1$.



- O número total de operações é:
  $T(n) = n + 2n + 4n + 8n + ... + 2^h n$
- $T(n) = 2^0 n + 2^1 n + 2^2 n + ... + 2^h n = n \sum_{i=0}^{h} 2^i = n(2^{h+1} - 1)$,
  h = altura da árvore.

Árvore de Chamadas Recursivas

Determinando altura h da árvore:

- Caminho da raiz até uma folha: $n, \quad n/2, \quad n/4, ..., 1$
- $\Leftrightarrow (1/2)^0 n, \quad (1/2)^1 n, \quad (1/2)^2 n, ..., (1/2)^h n$
- $\Rightarrow (1/2)^h n = 1$
- $\Rightarrow (1/2)^h = 1/n$
- $\Rightarrow 2^h = n$
- $\Rightarrow log_2 2^h = log_2 n$
- $\Rightarrow h = log_2 n$

Árvore de Chamadas Recursivas

Determinando altura h da árvore:

- Caminho da raiz até uma folha: $n, \quad n/2, \quad n/4, ..., 1$
- $\Leftrightarrow (1/2)^0 n, \quad (1/2)^1 n, \quad (1/2)^2 n, ..., (1/2)^h n$
- $\Rightarrow (1/2)^h n = 1$
- $\Rightarrow (1/2)^h = 1/n$
- $\Rightarrow 2^h = n$
- $\Rightarrow log_2 2^h = log_2 n$
- $\Rightarrow h = log_2 n$

Substituindo h em $T(n) = n(2^{h+1} - 1)$ = $n(2 \times 2^h - 1)$

- $T(n) = n(2 \times 2^{log_2 n} - 1)$
- $T(n) = n(2 \times n - 1)$
- $T(n) = 2n^2 - n, \quad$ (total de operações).
- $T(n) \in \Theta(n^2)$.

Algoritmos Recursivos - Exemplo #3: Conta bits

O seguinte algoritmo determina o **número de dígitos binários** na **representação binária** de um inteiro decimal positivo.

```
BINARIO( n ) {
    if(n == 1): return  1
    return BINARIO(n/2) + 1
}
```

Por exemplo:
$n = 8$ (1000) 4 dígitos
$n = 4$ (100) 3 dígitos
$n = 2$ (10) 2 dígitos

Algoritmos Recursivos - Exemplo #3: Conta bits

```
BINARIO( n ) {
if(n == 1): return  1
return BINARIO(n/2) + 1
}
```

- **Tamanho da entrada**: n
- **Operação básica**: return
- **Relação de recorrência**:

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + 1, & n > 1 \\ 1, & n = 1 \end{cases}$$

O número inteiro n pode ser **par** ou **impar**. Para tratar qualque entrada n, usamos $\lfloor n/2 \rfloor$. Note que, n terá um bit a mais que o número de bits de $\lfloor n/2 \rfloor$.

Algoritmos Recursivos - Exemplo #3: Conta bits

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + 1, & n > 1 \\ 1, & n = 1 \end{cases}$$

Para facilitar os cálculos, considerar entradas que são potência de 2: $n = 2^k$ (sempre número **par**).

A regra da suavização[1] mostra que a classe de complexidade do algoritmo para entradas não múltiplas de 2 é igual àquela para entradas potência de 2.

[1] Vejam Apêndice B do livro texto:Levitin.

Algoritmos Recursivos - Exemplo #3: Conta bits

Considerando n potência de 2 ($n = 2^k$), temos $\lfloor n/2 \rfloor = n/2$. Então:

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

Substituição em retrocesso:

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= T(n/4) + 2 \\ &= T(n/8) + 3 \\ &\ldots \\ &= T(n/2^k) + k \quad (generalizando). \\ &= T(1) + k, \quad pois \quad n = 2^k \\ &= 1 + log_2 n, \quad pois \quad k = log_2, \quad T(1) = 1\ . \end{aligned}$$

$T(n) \in \Theta(log_2 n)$.

Tipos importantes de recorrência

- $\begin{cases} T(n) = T(n-1) + c \\ T(1) = d \end{cases}$
  Solução: $T(n) = c(n-1) + d \Rightarrow T(n) \in O(n)$ (linear).

Tipos importantes de recorrência

- $\begin{cases} T(n) = T(n-1) + c \\ T(1) = d \end{cases}$
  Solução: $T(n) = c(n-1) + d \Rightarrow T(n) \in O(n)$ (linear).
- $\begin{cases} T(n) = T(n/2) + c \\ T(1) = d \end{cases}$
  Solução: $T(n) = c\log_2 n + d \Rightarrow T(n) \in O(\log_2 n)$ (logarítmica).

Tipos importantes de recorrência

- $\begin{cases} T(n) = T(n-1) + c \\ T(1) = d \end{cases}$
  Solução: $T(n) = c(n-1) + d \Rightarrow T(n) \in O(n)$ (linear).
- $\begin{cases} T(n) = T(n/2) + c \\ T(1) = d \end{cases}$
  Solução: $T(n) = c \log_2 n + d \Rightarrow T(n) \in O(\log_2 n)$ (logarítmica).
- $\begin{cases} T(n) = T(n-1) + cn \\ T(1) = d \end{cases}$
  Solução: $T(n) = c(\frac{n(n+1)}{2} - 1) + d \Rightarrow T(n) \in O(n^2)$ (quadrática).

Tipos importantes de recorrência

- $\begin{cases} T(n) = T(n-1) + c \\ T(1) = d \end{cases}$
  Solução: $T(n) = c(n-1) + d \Rightarrow T(n) \in O(n)$ (linear).
- $\begin{cases} T(n) = T(n/2) + c \\ T(1) = d \end{cases}$
  Solução: $T(n) = c\log_2 n + d \Rightarrow T(n) \in O(\log_2 n)$ (logarítmica).
- $\begin{cases} T(n) = T(n-1) + cn \\ T(1) = d \end{cases}$
  Solução: $T(n) = c(\frac{n(n+1)}{2} - 1) + d \Rightarrow T(n) \in O(n^2)$ (quadrática).
- $\begin{cases} T(n) = 2T(n/2) + cn \\ T(1) = d \end{cases}$
  Solução: $T(n) = cn\log_2 n + dn \Rightarrow T(n) \in O(n\log_2 n)$ (linearítmica).

Teorema Mestre

Considere a recorrência:

$$T(n) = aT(\frac{n}{b}) + f(n),$$

onde, $a \geq 1$ e $b > 1$ são constantes, $f(n) \in \Theta(n^k)$ para $k \geq 0$, e supondo que n é potencia de b.

$$\Rightarrow T(n) \in \begin{cases} \Theta(n^k), & \text{se} \quad a < b^k. \quad \text{(caso 1)} \\ \Theta(n^k \log_b n), & \text{se} \quad a = b^k. \quad \text{(caso 2)} \\ \Theta(n^{\log_b a}), & \text{se} \quad a > b^k. \quad \text{(caso 3)} \end{cases}$$

Teorema Mestre

Considere a recorrência:

$$T(n) = aT(\frac{n}{b}) + f(n),$$

onde, $a \geq 1$ e $b > 1$ são constantes, $f(n) \in \Theta(n^k)$ para $k \geq 0$, e supondo que n é potencia de b.

$$\Rightarrow T(n) \in \begin{cases} \Theta(n^k), & \text{se} \quad a < b^k. \quad \text{(caso 1)} \\ \Theta(n^k \log_b n), & \text{se} \quad a = b^k. \quad \text{(caso 2)} \\ \Theta(n^{\log_b a}), & \text{se} \quad a > b^k. \quad \text{(caso 3)} \end{cases}$$

O teorema também é válido para as notações Ω e O.

Teorema Mestre

Encontre a complexidade das recorrências abaixo.

Exemplo 1

$$T(n) = 9T(n/3) + n$$

Teorema Mestre

Encontre a complexidade das recorrências abaixo.

Exemplo 1

$$T(n) = 9T(n/3) + n$$

Como $a = 9, b = 3, k = 1$ $(9 > 3^1)$, aplica-se o Caso 3 do Teorema.
Portanto, $T(n) \in \Theta(n^{\log_3^9}) = \Theta(n^2)$.

Encontre a complexidade das recorrências abaixo.

Exemplo 1

$$T(n) = 9T(n/3) + n$$

Como $a = 9, b = 3, k = 1$ $(9 > 3^1)$, aplica-se o Caso 3 do Teorema.
Portanto, $T(n) \in \Theta(n^{\log_3^9}) = \Theta(n^2)$.

Exemplo 2

$$T(n) = T(2n/3) + 1$$

Teorema Mestre

Encontre a complexidade das recorrências abaixo.

Exemplo 1

$$T(n) = 9T(n/3) + n$$

Como $a = 9, b = 3, k = 1$ $(9 > 3^1)$, aplica-se o Caso 3 do Teorema. Portanto, $T(n) \in \Theta(n^{\log_3^9}) = \Theta(n^2)$.

Exemplo 2

$$T(n) = T(2n/3) + 1$$

Como $a = 1, b = 3/2, k = 0$, $(a = b^0)$ aplica-se caso 2 do Teorema Mestre. Portanto, $T(n) \in \Theta(n^0 \cdot \log_{3/2} n) = \Theta(\log_{3/2} n) = \Theta(\log n)$.

Exemplo 3

$$T(n) = T(n/3) + 6n - 1$$

Teorema Mestre

Exemplo 3

$$T(n) = T(n/3) + 6n - 1$$

Como $a = 1, b = 3, k = 1$ $(1 < 3^1)$, aplica-se o Caso 1 do Teorema. Portanto, $T(n) \in \Theta(n)$.

Teorema Mestre

Exemplo 3

$$T(n) = T(n/3) + 6n - 1$$

Como $a = 1, b = 3, k = 1$ $(1 < 3^1)$, aplica-se o Caso 1 do Teorema. Portanto, $T(n) \in \Theta(n)$.

Exemplo 4

$$T(n) = 8T(n/2) + n^2$$

Exercício: determinar a ordem Θ a qual pertence esta função de recorrência.