

1. SOMA DE PRIMITIVAS

Operações fundamentais ou básicas que compõem um algoritmo.
Elas são as unidades básicas de execução que formam a base para a análise do desempenho de um algoritmo, como:

- * Adição e subtração de números inteiros ou reais
 $2 + 2$
- * Multiplicação e divisão de números inteiros ou reais
 $2 * 2$
- * Atribuição de valores a variáveis
 $idade = 16$
- * Comparação de valores
 $15 == 16$

Exemplo 1

```
fatorial(n)
    int fat = 1
    for (int i = 1; i ≤ n; i++)
        fat = fat * i
    return fat
```

$$\begin{array}{r} 2 \\ 2 + (n+1) + 2n = 3n + 3 \\ 2n \\ \hline 1 \\ 2 + 3n + 3 + 2n + 1 = 5n + 6 \end{array}$$

Exemplo 3

```
primo(n)
    for (int i = 2; i < n; i++)
        if n mod i = 0
            return false
    return true
```

$$\begin{array}{r} 2 + (n-1) + 2(n-2) = 3n - 3 \\ 2(n-2) = 2n - 4 \\ 1 \\ \hline 1 \\ 3n-3 + 2n-4 + 1 = 5n - 6 \end{array}$$

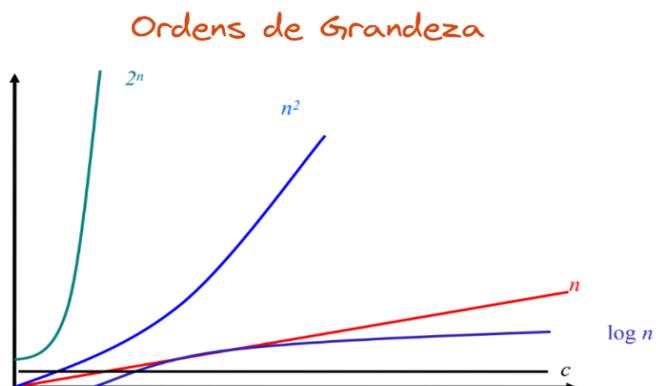
Exemplo 2

```
fibonacci(n)
    int num1 = 0
    int num2 = 1
    int soma = 0
    for (int i = 2; i ≤ n; i++)
        soma = num1 + num2
        num1 = num2
        num2 = soma
    return soma
```

$$\begin{array}{r} 2 \\ 2 \\ 2 \\ 2 + 2 + 2 = 6 \\ 2 + n + 2(n-1) = 3n \\ 2(n-1) = 2n-2 \\ n-1 \\ n-1 \\ n-1 \\ n-1 \\ 1 \\ 6 + 3n + 2n-2 + 2n-2 + 1 = 7n + 3 \end{array}$$

2. Análise Assintótica

Observações



Analizando ordens de grandeza

Em termos simplistas, $f(n) \in \Theta(g(n))$ significa dizer que o crescimento de $f(n)$ é igual ao de $g(n)$.

Limite inferior e superior (Θ)

$$0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n), \forall n \geq n_0$$

Limite inferior (Ω) Limite superior (O)

Determinando $g(n)$

Para determinar o $g(n)$ basta olhar para qual valor o n está sendo elevado na função e escolher o n cujo expoente seja o maior

Exemplos

$$3+3 \xrightarrow{n^0} g(n) = 1$$

$$3n+3 \xrightarrow{n^1} g(n) = n$$

$$3n+3n^2 \xrightarrow{n^2} g(n) = n^2$$

A notação Θ (Theta)

Para duas funções $f(n)$ e $g(n)$, dizemos que $f(n)$ é $\Theta(g(n))$ se

$$0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n), \forall n \geq n_0$$

Exemplo

$f(n)=3n+3$ é $\Theta(n)$?

1. Determinando parâmetros

$$\begin{aligned}f(n) &= 3n+3 \\g(n) &= n \\c_1 &= 1 \\c_2 &= 6\end{aligned}$$

3. Resolvendo inequação para $n_0=1$

$$\begin{aligned}n &\leq 3n+3 \leq 6n \\1 &\leq 3(1)+3 \leq 6(1) \\1 &\leq 6 \leq 6\end{aligned}$$



2. Resolvendo inequação $\Theta(n)$

$$\begin{aligned}c_1 * g(n) &\leq f(n) \leq c_2 * g(n) \\n &\leq 3n+3 \leq 6n \\n &\leq 3n+3 \leq 6n\end{aligned}$$

3. Resolvendo inequação para $n_0=10$

$$\begin{aligned}n &\leq 3n+3 \leq 6n \\10 &\leq 3(10)+3 \leq 6(10) \\10 &\leq 33 \leq 60\end{aligned}$$



A notação O (Big O notation)

Para duas funções $f(n)$ e $g(n)$, dizemos que $f(n)$ é $O(g(n))$ se

$$0 \leq f(n) \leq c * g(n), \forall n \geq n_0$$

Exemplo

$f(n)=n^2 + 1$ é $O(n)$?

1. Determinando parâmetros

$$\begin{aligned}f(n) &= n^2 + 1 \\g(n) &= n^2 \\c &= 2\end{aligned}$$

3. Resolvendo inequação para $n_0=1$

$$\begin{aligned}n^2 + 1 &\leq 2n^2 \\1^2 + 1 &\leq 2(1)^2 \\2 &\leq 2\end{aligned}$$



2. Resolvendo inequação $\Theta(n)$

$$\begin{aligned}f(n) &\leq c * g(n) \\n^2 + 1 &\leq 2 * n^2 \\n^2 + 1 &\leq 2n^2\end{aligned}$$

4. Resolvendo inequação para $n_0=10$

$$\begin{aligned}n^2 + 1 &\leq 2n^2 \\10^2 + 1 &\leq 2(10)^2 \\101 &\leq 200\end{aligned}$$



A notação Ω (Omega)

Para duas funções $f(n)$ e $g(n)$, dizemos que $f(n)$ é $\Omega(g(n))$ se

$$c * g(n) \leq f(n), \forall n \geq n_0$$

Exemplo

$f(n)=n^2 + 1$ é $\Omega(n^2)$?

1. Determinando parâmetros

$$\begin{aligned}f(n) &= n^2 + 1 \\g(n) &= n^2 \\c &= 1\end{aligned}$$

3. Resolvendo inequação para $n_0=1$

$$\begin{aligned}n^2 &\leq n^2 + 1 \\1^2 &\leq 1^2 + 1 \\1 &\leq 2\end{aligned}$$



2. Resolvendo inequação $\Theta(n)$

$$\begin{aligned}c * g(n) &\leq f(n) \\n^2 &\leq n^2 + 1 \\n^2 &\leq n^2 + 1\end{aligned}$$

4. Resolvendo inequação para $n_0=10$

$$\begin{aligned}n^2 &\leq n^2 + 1 \\10^2 &\leq 10^2 + 1 \\100 &\leq 101\end{aligned}$$



Exercícios

1. $6n^3 = O(n^2)$?

(1)
 $f(n) = 6n^3$
 $g(n) = n^2$

(2)
 $f(n) \leq c*g(n)$
 $6n^3 \leq c * n^2$
 ~~$6n^3 \leq c * n^2$~~
 $6n \leq c$

(3)
Dado o resultado $6n \leq c$ concluímos que $f(n)$ não pertence a $O(n^2)$ pois para qualquer valor de c vai sempre existir um n significativamente grande que vai invalidar a inequação

(4)

Exemplos

$$\begin{array}{ll} n_0 = 1 & 6n \leq c \\ c = 6 & 6 \leq 6 \end{array} \quad \checkmark$$

$$\begin{array}{ll} n_0 = 100 & 6n \leq c \\ c = 6 & 600 \leq 6 \end{array} \quad \checkmark$$

2. $n / 1000 = O(n)$?

(1)
 $f(n) = n/1000$
 $g(n) = n$

(2)
 $f(n) \leq c*g(n)$
 $n/1000 \leq c*n$
 $n \leq c*n*1000$

(3)
Dado o resultado $n \leq c*n*1000$ concluímos que $f(n)$ pertence a $O(n)$ pois assumindo $c=1$ teremos $n \leq n*1000$ e com isso não importando o valor de n a inequação sempre será satisfeita

(4)

Exemplos

$$\begin{array}{ll} n_0 = 1 & n \leq c*n*1000 \\ c = 1 & 1 \leq 1001 \end{array} \quad \checkmark$$

$$\begin{array}{ll} n_0 = 100 & n \leq c*n*1000 \\ c = 1 & 100 \leq 1100 \end{array} \quad \checkmark$$

3. Seja $T(n) = 2n^2 + 3n$. Quais das seguintes afirmações são verdadeiras?

$T(n) = O(n)$.

$T(n) = \Omega(n)$.

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

$T(n) = O(n^3)$.

Sem utilizar valores, podemos observar as anotações $T(n) = 2n^2 + 3n$.

Para a primeira afirmação, $T(n) = O(n)$, a notação $O(n)$ indica que $T(n)$ deve crescer na mesma taxa ou mais lentamente do que n . No entanto, a função $T(n)$ cresce mais rapidamente do que n , pois o termo dominante é $2n^2$, enquanto o termo dominante de n é n . Portanto, a afirmação é falsa.

Para a segunda afirmação, $T(n) = \Omega(n)$, a notação $\Omega(n)$ indica que $T(n)$ deve crescer na mesma taxa ou mais rapidamente do que n . Como o termo dominante de $T(n)$ é $2n^2$, que cresce mais rapidamente do que n , a afirmação é verdadeira.

Para a terceira afirmação, $T(n) = \Theta(n^2)$, a notação $\Theta(n^2)$ indica que $T(n)$ deve crescer na mesma taxa que n^2 . Como o termo dominante de $T(n)$ é $2n^2$, que cresce na mesma taxa que n^2 , a afirmação é verdadeira.

Para a quarta afirmação, $T(n) = O(n^3)$, a notação $O(n^3)$ indica que $T(n)$ deve crescer na mesma taxa ou mais lentamente do que n^3 . Como o termo dominante de $T(n)$ é $2n^2$, que cresce mais lentamente do que n^3 , a afirmação é verdadeira.

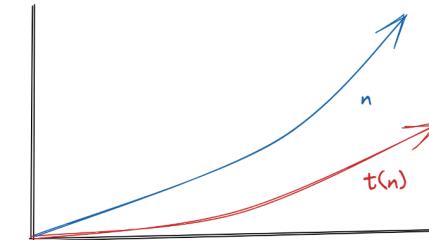
Portanto, sem utilizar valores, podemos concluir que as afirmações verdadeiras são:

- $T(n) = \Omega(n)$.
- $T(n) = \Theta(n^2)$.
- $T(n) = O(n^3)$.

 Regenerate response

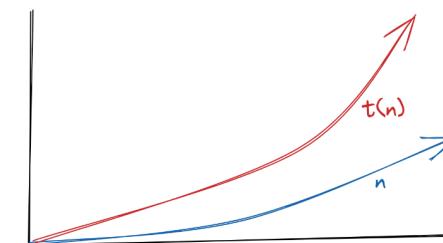
$T(n) = O(n)$

- * $O(n) = n$ é igual ou superior a $T(n)$
- * $T(n)$ deve crescer na mesma taxa ou mais lentamente do que n



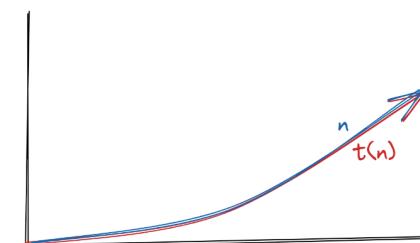
$T(n) = \Omega(n)$

- * $\Omega(n) = n$ é igual ou inferior a $T(n)$
- * $T(n)$ deve crescer na mesma taxa ou mais rapidamente do que n



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

- * $\Theta(n) = n$ é igual a $T(n)$
- * $T(n)$ deve crescer na mesma taxa do que n

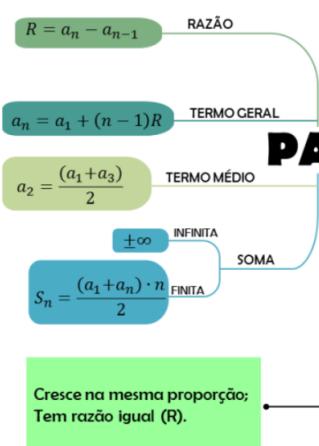
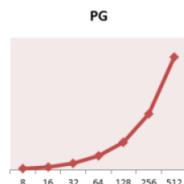
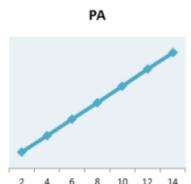


3. Análise de Algoritmos Recursivos

Material Auxiliar

Propriedades	$n, m, k \in \mathbb{R}$	$x, y \in \mathbb{R}^+$	$a, b \in \mathbb{R}^+$	$p \in \mathbb{R} \setminus \{0\}$
potências de expoente real				
• $a^0 = 1$				
• $a^n \times a^m = a^{n+m}$				
• $\frac{a^n}{a^m} = a^{n-m}$				
• $a^n \times b^n = (a \times b)^n$				
• $\frac{a^n}{b^n} = \left(\frac{a}{b}\right)^n$				
• $a^{-n} = \frac{1}{a^n}$				
• $(a^n)^m = a^{n \times m}$				
• $\sqrt[n]{a^m} = a^{\frac{m}{n}}$				
logaritmos $a \neq 1$ $b \neq 1$				
$a^{\log_a x} = x$				
$\log_a a^k = k$				
• $\log_a 1 = 0$				
• $\log_a a = 1$				
• $\log_a x^k = k \log_a x$				
• $\log_a(x \times y) = \log_a x + \log_a y$				
• $\log_a \left(\frac{x}{y}\right) = \log_a x - \log_a y$				
• $\log_a x = \frac{\log_b x}{\log_b a}$				
• $\log_b a \times \log_a b = 1$				
• $\log_a x = \log_a^p x^p$				

Fórmulas	P.A	P.G
Termo Geral:	$a_n = a_1 + (n - 1) \cdot r$	$a_n = a_1 \cdot q^{(n-1)}$
Soma dos termos:		$S_n = \frac{a_1(q^n - 1)}{q - 1}$ $S_\infty = \frac{a_1}{1 - q}$ Condição: $-1 < q < 1$
Equivalência:	$a_n + a_m = a_p + a_t$ Onde: $n + m = p + t$	$a_n \cdot a_m = a_p \cdot a_t$ Onde: $n + m = p + t$



Soma dos termos de uma:

PA PG finita PG infinita decrescente

$$S_n = \frac{n(a_1 + a_n)}{2}$$

$$S_n = \frac{a_1(q^n - 1)}{q - 1}$$

$$\cdot S_n \frac{a_1}{1 - q}$$

RAZÃO $q = \frac{a_n}{a_{n-1}}$

TERMO GERAL $a_n = a_1 \cdot q^{n-1}$

TERMO MÉDIO $a_2 = \sqrt{a_1 \cdot a_3}$

SOMA $S_\infty = \frac{a_1}{1 - q}$

INFINITA

$S_n = \frac{a_1(q^n - 1)}{q - 1}$ FINITA

Cada termo, a partir do 2º, é igual à multiplicação do termo anterior por uma constante (q).

Descobrindo Relação de Recorrência

(1) Exemplo com fatorial

```
int fatorial(n)
if n == 0
    return 1
else
    return n * fatorial(n - 1)
```

Caso base
recursão

$$t(n) = t(n-1) + \Theta(1) + \Theta(1)$$

Relação de Recorrência

$t(n)$

$$\begin{cases} t(0) = \Theta(1) \\ t(n) = t(n-1) + \Theta(1) \end{cases}$$

(2) Exemplo com fibonacci

```
int fib(n)
int resp = 0
if(n == 1 || n == 2)
    resp = 1
else
    resp = fib(n-1) + fib(n-2)
return resp
```

Caso base
recursão

$$t(n) = t(n-1) + t(n-2) + \Theta(1) + \Theta(1)$$

Relação de Recorrência

$t(n)$

$$\begin{cases} t(1) = \Theta(1) \\ t(2) = \Theta(1) \\ t(n) = t(n-1) + t(n-2) + \Theta(1) \end{cases}$$

(3) Exemplo com MergeSort

```
MergeSort (A, inicio, fim)
if (inicio < fim) then
    meio = (inicio + fim)/2
    MergeSort(A, inicio, meio)
    MergeSort(A, meio + 1, fim)
    Merge(A, inicio, meio, fim)
```

custo $\Theta(n)$

recursão

$$t(n) = t(n/2) + t(n/2) + \Theta(n) + \Theta(1)$$

$$t(n) = 2t(n/2) + \Theta(n)$$

Nesse tipo de soma
prevalece o com o
maior custo

Relação de Recorrência

$$t(n) \left\{ \begin{array}{l} t(1) = \Theta(1) \\ t(n) = 2t(n/2) + \Theta(n) \end{array} \right.$$

(4) Exemplo com algoritmo qualquer

```
xpto(v, ini, fim)
if(ini < fin)
    meio = (ini + fim)/2
    xpto(v, ini, meio)
    xpto(v, meio+1, fim)
    for(int i = ini; i ≤ fim; i++)
        for(int j=ini+1; j ≤ fim; j++)
            print("processado")
```

custo $\Theta(n^2)$

recursão

$$t(n) = t(n/2) + t(n/2) + \Theta(n^2) + \Theta(1)$$

$$t(n) = 2t(n/2) + \Theta(n^2)$$

Relação de Recorrência

$$t(n) \left\{ \begin{array}{l} t(1) = \Theta(1) \\ t(n) = 2t(n/2) + \Theta(n^2) \end{array} \right.$$

Exemplos

Exemplo com fatorial

```
int factorial(n)
    if n == 0
        return 1
    else
        return n * factorial(n - 1)
```

(1) Relação de Recorrência

$$t(n) \leftarrow \begin{array}{l} t(0) = \Theta(1) \text{ (Caso base da recursão)} \\ t(n) = t(n-1) + \Theta(1) \text{ (Chamada da recursão)} \end{array}$$

(2) Calculando custo com o método interativo

(1)

$$\begin{aligned} 1 & \quad t(n) = t(n-1) + 1 \\ 2 & \quad \text{substituição} \\ & \quad t(n-1) = t(n-2) + 1 \Rightarrow t(n) = t(n-1) + 1 \\ & \quad t(n) = [t(n-2) + 1] + 1 \\ & \quad t(n) = t(n-2) + 2 \\ 3 & \quad \text{substituição} \\ & \quad t(n-2) = t(n-3) + 1 \Rightarrow t(n) = t(n-2) + 2 \\ & \quad t(n) = [t(n-3) + 1] + 2 \\ & \quad t(n) = t(n-3) + 3 \end{aligned}$$

k

$$t(n) = t(n-k) + k \text{ (Chamada da recursão)}$$

$$t(0) = 1 \text{ (Caso base da recursão)}$$

$$n-k = 0$$

$$k = n$$

$$t(n) = t(n-n) + n$$

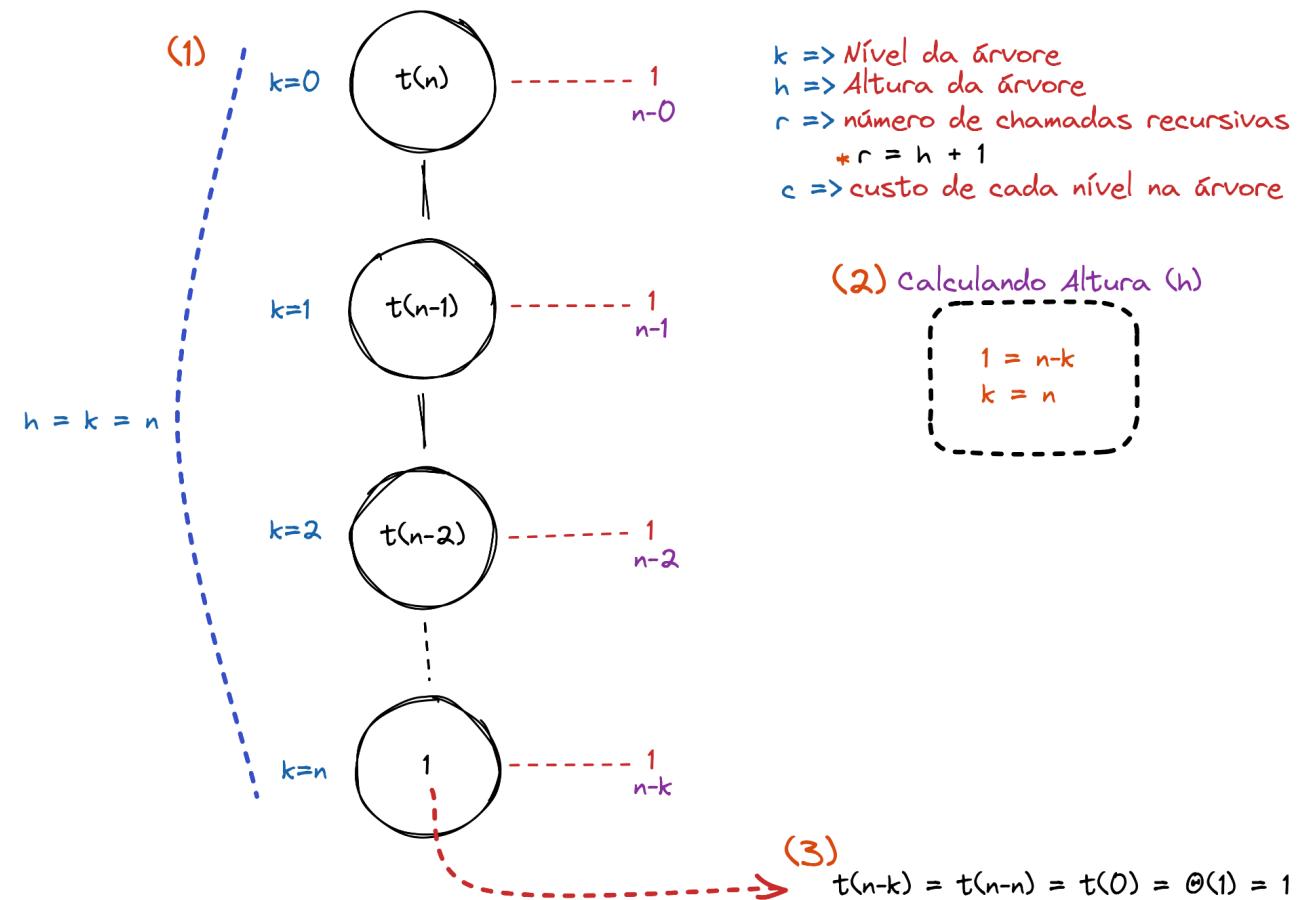
$$t(n) = t(0) + n$$

$$t(n) = 1 + n$$

$$t(n) = \Theta(n)$$

$$(2) t(n) = \Theta(n)$$

(2) Calculando custo com o método da árvore de recursão



(4) Custo

$$c = 1$$

$$r = n + 1$$

$$t(n) = r * c$$

$$t(n) = (n+1) * 1$$

$$t(n) = \Theta(n)$$

(5) $t(n) = \Theta(n)$

Exemplo com MergeSort

```
MergeSort (A, inicio, fim)
    if (inicio < fim) then
        meio = (inicio + fim)/2
        MergeSort(A, inicio, meio)
        MergeSort(A, meio + 1, fim)
        Merge(A, inicio, meio, fim)
```

(1) Relação de Recorrência

$$t(n) \left\{ \begin{array}{l} t(1) = \Theta(1) \text{ (Caso base da recursão)} \\ t(n) = 2t(n/2) + \Theta(n) \text{ (Chamada da recursão)} \end{array} \right.$$

(2) Calculando custo com o método interativo

(1)



$$t(n) = 2t(n/2) + n$$



substituição

$$\begin{aligned} t(n/2) &= 2t(n/4) + n & \Rightarrow t(n) &= 2t(n/2) + n \\ t(n) &= 2[2t(n/4) + n] + n & t(n) &= 4t(n/4) + 2n \\ t(n) &= 4t(n/4) + 2n \end{aligned}$$



substituição

$$\begin{aligned} t(n/4) &= 2t(n/8) + n & \Rightarrow t(n) &= 4t(n/4) + 2n \\ t(n) &= 4[2t(n/8) + n] + 2n & t(n) &= 8t(n/8) + 3n \\ t(n) &= 8t(n/8) + 3n \end{aligned}$$



$$t(n) = 2^k t(n/2^k) + kn \text{ (Chamada da recursão)}$$

$$t(1) = 1 \text{ (Caso base da recursão)}$$

$$n/2^k = 1$$

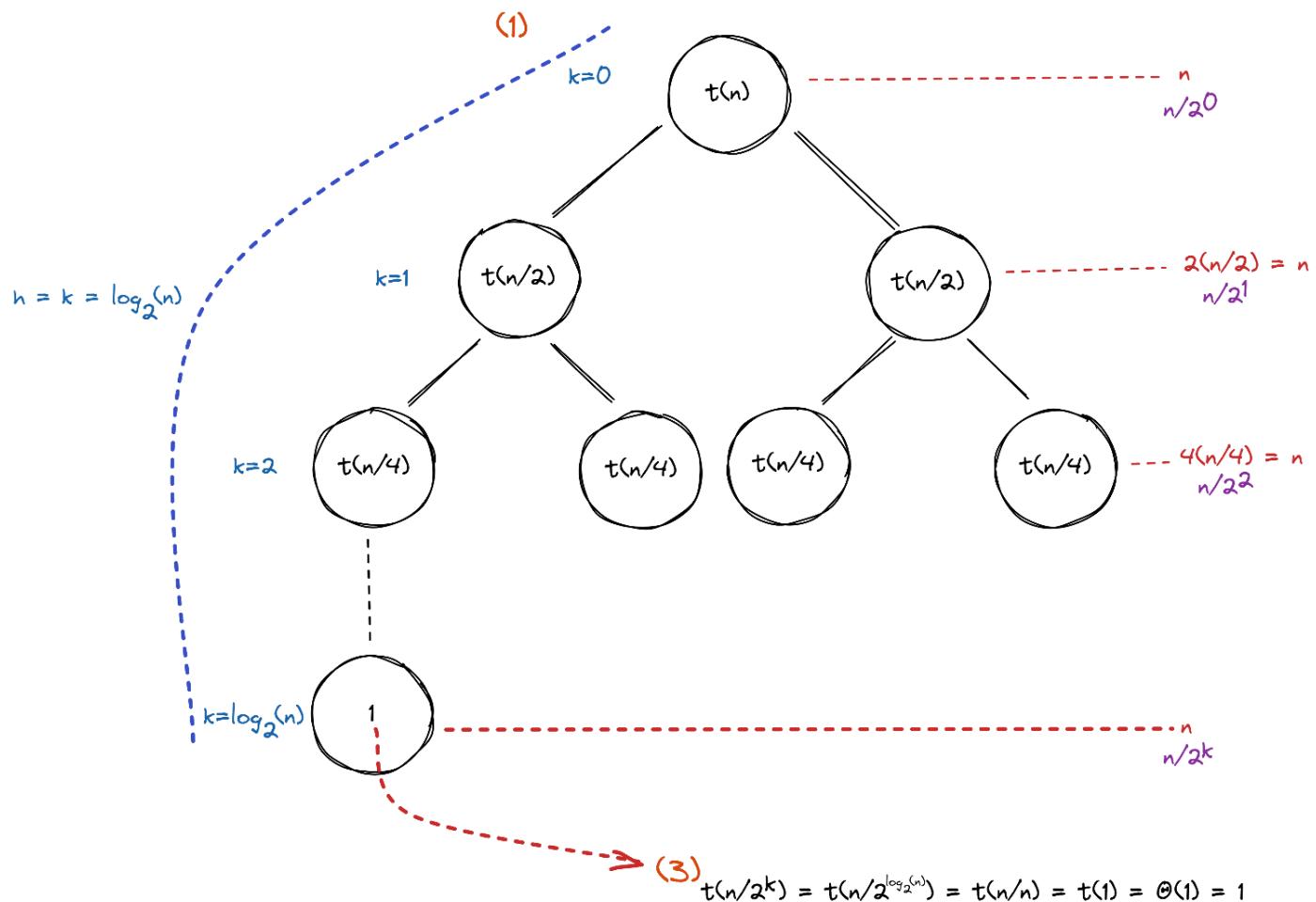
$$2^k = n$$

$$k = \log_2(n)$$

$$\begin{aligned} t(n) &= 2^{\log_2(n)} t(n/2^{\log_2(n)}) + \log_2(n) * n \\ t(n) &= nt(1) + \log_2(n) * n \\ t(n) &= n + \log_2(n) * n \\ t(n) &= \Theta(\log_2(n) * n) \end{aligned}$$

$$(2) t(n) = \Theta(n * \log_2(n))$$

(2) Calculando custo com o método da árvore de recursão



(3) *Custo*

caso

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

$$t^{(n)} = c$$

$$t(n) = (\log_2 n + 1) * n$$

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

$$t(n) = n + \log_2(n) + n$$

$$t(n) = \Theta(n + \log_2(n))$$

$$O(n) = O(n + \log_2(n))$$

$$(4) t(n) = \Theta(n + \log_2(n))$$

- $k \Rightarrow$ Nível da árvore
- $h \Rightarrow$ Altura da árvore
- $r \Rightarrow$ número de chamadas recursivas
 - * $r = h + 1$
- $c \Rightarrow$ custo de cada nível na árvore

(2) Calculando Altura (h)

$$1 = n/2^k$$

$$2^k = n$$

$$k = \log_2(n)$$

Exercícios

1

$$t(n) \left\{ \begin{array}{l} t(1) = \Theta(1) \\ t(n) = 3t(n/3) + \Theta(n) \end{array} \right.$$

1

$$t(n) = 3t(n/3) + n$$

2

$$t(n/3) = 3t(n/9) + n/3 \xrightarrow{\text{substituição.}} \begin{aligned} t(n) &= 3t(n/3) + n \\ &= 3[3t(n/9) + n/3] + n \\ &= 9t(n/9) + 2n \end{aligned}$$

3

$$t(n/9) = 3t(n/27) + n/9 \xrightarrow{\text{substituição.}} \begin{aligned} t(n) &= 9t(n/9) + 2n \\ &= 9[3t(n/27) + n/9] + 2n \\ &= 27t(n/27) + 3n \end{aligned}$$

k

$$t(n) = t3^k(n/3^k) + kn \quad (\text{Chamada da recursão})$$

$$T(1) = 1 \quad (\text{Caso base da recursão})$$

$$\begin{aligned} n/3^k &= 1 \\ n &= 3^k \\ k &= \log_3(n) \end{aligned}$$

$$\begin{aligned} t(n) &= t3^{\log_3(n)}(n/3^{\log_3(n)}) + \log_3(n) * n \\ t(n) &= \log_3(n) * n \\ t(n) &= \Theta(\log_3(n) * n) \end{aligned}$$

2

$$t(n) \left\{ \begin{array}{l} t(1) = \Theta(1) \\ t(n) = 5t(n/2) + \Theta(n^3) \end{array} \right.$$

1

$$t(n) = 5t(n/2) + n^3$$

2

$$t(n/2) = 5t(n/4) + n^3/8 \xrightarrow{\text{substituição.}} \begin{aligned} t(n) &= 5t(n/2) + n^3 \\ &= 5[5t(n/4) + n^3/8] + n^3 \\ &= 25t(n/4) + 13n^3/8 \end{aligned}$$

3

$$t(n/4) = 5t(n/8) + n^3/64 \xrightarrow{\text{substituição.}} \begin{aligned} t(n) &= 25t(n/4) + 13n^3/8 \\ &= 25[5t(n/8) + n^3/64] + 13n^3/8 \\ &= 125t(n/8) + 25n^3/64 + 13n^3/8 \\ t(n) &= 125t(n/8) + 129n^3/64 \end{aligned}$$

k

$$t(n) = 5^k t(n/2^k) + \Theta(n^3) \quad (\text{Chamada da recursão})$$

$$T(1) = 1 \quad (\text{Caso base da recursão})$$

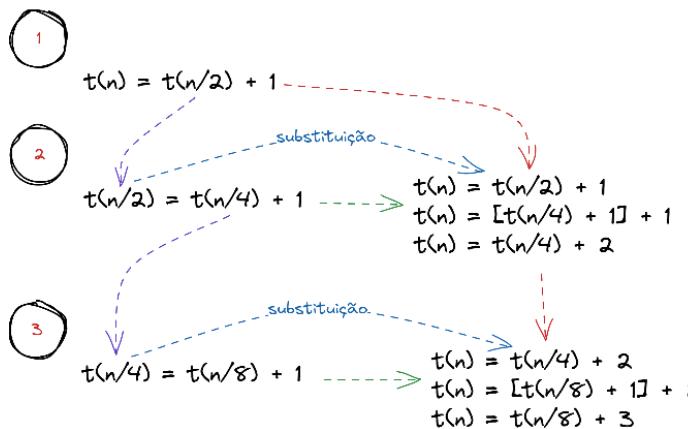
$$\begin{aligned} n/2^k &= 1 \\ n &= 2^k \\ k &= \log_2(n) \end{aligned}$$

$$\begin{aligned} t(n) &= 5^{\log_2(n)} t(n/2^{\log_2(n)}) + \Theta(n^3) \\ t(n) &= \log_2(n) + \Theta(n^3) \\ t(n) &= \Theta(\log_2(n)) + \Theta(n^3) \\ t(n) &= \Theta(n^3) \end{aligned}$$

3

$$t(n) = t(n/2) + \theta(1)$$

Algoritmo de busca binária



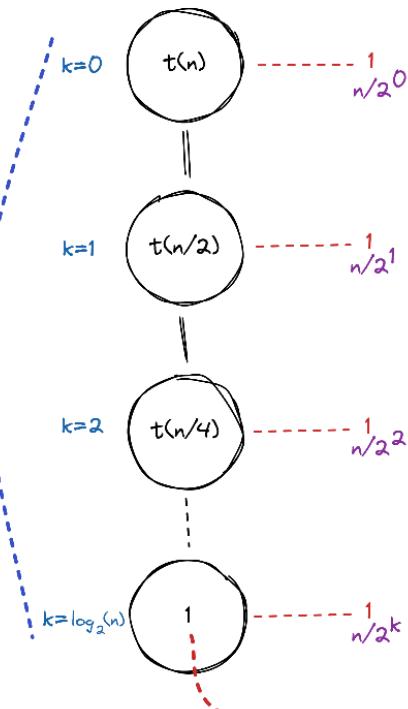
$$t(n) = t(n/2^k) + k \text{ (Chamada da recursão)}$$

$T(1) = 1$ (Caso base da recursão)

$$\begin{aligned} \downarrow \\ n/2^k &= 1 \\ n &= 2^k \\ k &= \log_2(n) \end{aligned}$$

$$\begin{aligned}t(n) &= t(n/2^{\log_2(n)}) + \log_2(n) \\t(n) &= t(1) + \log_2(n) \\t(n) &= 1 + \log_2(n) \\t(n) &= \Theta(\log_2(n))\end{aligned}$$

$$h = k = \log_2(n)$$



- => Nível da árvore
- => Altura da árvore
- => número de chamadas recursivas
 - * $r = h + 1$
- => custo de cada nível na árvore

Calculando Altura (h)

$$k = \log_2(n)$$

Custo

$$c = 1$$

$$\begin{aligned}t(n) &= r * c \\t(n) &= (\log_2(n) + 1) * 1 \\t(n) &= \log_2(n) + 1 \\t(n) &= \Theta(\log_2(n))\end{aligned}$$

4

$$t(n) \leftarrow t(1) = \Theta(1)$$

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

1

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

2

substituição

$$t(n/2) = 8t(n/4) + n^2/4 \rightarrow t(n) = 8t(n/2) + n^2$$

$$t(n) = 8[8t(n/4) + n^2/4] + n^2$$

$$t(n) = 64t(n/4) + 3n^2$$

3

substituição

$$t(n/4) = 8t(n/8) + n^2/16 \rightarrow t(n) = 64t(n/4) + 3n^2$$

$$t(n) = 64[8t(n/8) + n^2/16] + 3n^2$$

$$t(n) = 512t(n/8) + 7n^2$$

k

$$t(n) = 8^k t(n/2^k) + (2^k - 1) * n^2 \quad (\text{Chamada da recursão})$$

$$T(1) = 1 \quad (\text{Caso base da recursão})$$

$$\begin{aligned} n/2^k &= 1 \\ n &= 2^k \\ k &= \log_2(n) \end{aligned}$$

$$\begin{aligned} t(n) &= 8^{\log_2(n)} t(n/2^{\log_2(n)}) + (2^{\log_2(n)} - 1) * n^2 \\ t(n) &= n^3 t(n/n) + (n-1) * n^2 \\ t(n) &= n^3 + n^3 + n^2 \\ t(n) &= 2n^3 + n^2 \\ t(n) &= \Theta(n^3) \end{aligned}$$

$2^k - 1$
sequência de números ímpares
que aumenta em um valor
cada vez maior.
Seja: 1..3..7..15..31

Calculando Logaritmo $8^{\log_2(n)}$

$$\begin{aligned} 8^{\log_2(n)} &= (2^3)^{\log_2(n)} \\ &= 2^{3 \cdot \log_2(n)} \\ &= 2^{\log_2(2^3)} \\ &= 2^{\log_2(n^3)} \\ &= n^3 \end{aligned}$$

4. Análise de Algoritmos Recursivos com o Método Mestre

(1)

O que é?

É uma técnica que permite identificar a classe de complexidade de um algoritmo aplicando apenas algumas operações matemáticas e comparando ordem de complexidade de funções.

(2)

Em quais algoritmos posso utilizar?

Primeiramente, é preciso que a relação de recorrência do algoritmo tenha determinadas propriedades.
Vamos analisar concretamente essas propriedades:

Exemplo

MergeSort

$$t(n) = 2t(n/2) + \Theta(n)$$

$$\begin{array}{c} a \\ \parallel \\ 2 \geq 1 \end{array} \quad \begin{array}{c} b \\ \parallel \\ 2 > 1 \end{array} \quad \begin{array}{c} f(n) \\ | \\ n \text{ é positivo} \end{array}$$



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

$a \geq 1$
 $b > 1$
 $f(n)$ é positivo

(3)

Como utilizamos?

O método mestre estabelece que:

(1) Se $f(n) < n^{\log_b(a-\epsilon)}$ então $t(n) = \Theta(n^{\log_b(a)})$

(2) Se $f(n) = n^{\log_b(a)}$ então $t(n) = \Theta(f(n) * \log_b(n))$

(3) Se $f(n) > n^{\log_b(a+\epsilon)}$ então $t(n) = \Theta(f(n))$

com $\epsilon > 0$

(4) Exemplos

1

$$T(n) = 8T(n/2) + 1000 * n^2$$

1 Atribuindo os valores

$$\begin{aligned} a &= 8 \\ b &= 2 \\ f(n) &= 1000 * n^2 \end{aligned}$$

\star

$a \geq 1$ ✓
 $b > 1$ ✓
 $f(n)$ é positivo ✓

2 Calculando logaritmo

$$n^{\log_b(a)} = n^{\log_2(8)} = n^3$$



$$\begin{aligned} \log_2(8) &= \log_2(2^3) \\ &= 3 * \log_2(2) \\ &= 3 * \log_2(2) \\ &= 3 * 1 \\ &= 3 \end{aligned}$$

3 Fazendo comparação

$$(1) f(n) < n^{\log_b(a)} \quad \checkmark$$

$$(2) f(n) = n^{\log_b(a)} \quad \times$$

$$(3) f(n) > n^{\log_b(a)} \quad \times$$

1º Regra

$$1000 * n^2 < n^3$$

4 Concluindo

Pela aplicação da 1º regra do teorema mestre concluímos que:

$$\begin{aligned} T(n) &= \Theta(n^{\log_b(a)}) \\ T(n) &= \Theta(n^3) \end{aligned}$$

2

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

1 Atribuindo os valores

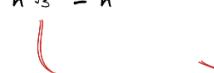
$$\begin{aligned} a &= 9 \\ b &= 3 \\ f(n) &= n \end{aligned}$$

\star

$a \geq 1$ ✓
 $b > 1$ ✓
 $f(n)$ é positivo ✓

2 Calculando logaritmo

$$n^{\log_b(a)} = n^{\log_3(9)} = n^2$$



$$\begin{aligned} \log_3(9) &= \log_3(3^2) \\ &= 2 * \log_3(3) \\ &= 2 * \log_3(3) \\ &= 2 * 1 \\ &= 2 \end{aligned}$$

3 Fazendo comparação

$$(1) f(n) < n^{\log_b(a)} \quad \checkmark$$

$$(2) f(n) = n^{\log_b(a)} \quad \times$$

$$(3) f(n) > n^{\log_b(a)} \quad \times$$

1º Regra

$$n < n^2$$

4 Concluindo

Pela aplicação da 1º regra do teorema mestre concluímos que:

$$\begin{aligned} T(n) &= \Theta(n^{\log_b(a)}) \\ T(n) &= \Theta(n^2) \end{aligned}$$

3

$$T(n) = 5T(n/2) + n^3$$

4

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

1 Atribuindo os valores

$$\begin{aligned} a &= 5 \\ b &= 2 \\ f(n) &= n^3 \end{aligned}$$

$a \geq 1$ ✓
 $b > 1$ ✓
 $f(n)$ é positivo ✓

2 Calculando logaritmo

$$n^{\log_b(a)} = n^{\log_2(5)} = n^{2,3}$$

$$\log_2(5) \approx 2,3219$$

3 Fazendo comparação

$$(1) f(n) < n^{\log_b(a)} \times$$

$$(2) f(n) = n^{\log_b(a)} \times$$

$$(3) f(n) > n^{\log_b(a)} \checkmark$$

$n^3 > n^{2,3}$
 $3^{\text{a}} \text{ Regra}$

4 Concluindo

Pela aplicação da 3^o regra do teorema mestre
concluímos que:

$$\begin{aligned} T(n) &= \Theta(f(n)) \\ T(n) &= \Theta(n^3) \end{aligned}$$

1 Atribuindo os valores

$$\begin{aligned} a &= 1 \\ b &= 3/2 = 1,5 \\ f(n) &= 1 \end{aligned}$$

$a \geq 1$ ✓
 $b > 1$ ✓
 $f(n)$ é positivo ✓

2 Calculando logaritmo

$$n^{\log_b(a)} = n^{\log_{1,5}(1)} = n^0 = 1$$

3 Fazendo comparação

$$(1) f(n) < n^{\log_b(a)} \times$$

$$(2) f(n) = n^{\log_b(a)} \checkmark \quad \xrightarrow{2^{\text{o}} \text{ Regra}} 1 = 1$$

$$(3) f(n) > n^{\log_b(a)} \times$$

4 Concluindo

Pela aplicação da 2^o regra do teorema mestre
concluímos que:

$$\begin{aligned} T(n) &= \Theta(f(n) * \log_b(n)) \\ T(n) &= \Theta(1 * \log_{1,5}(n)) \\ T(n) &= \Theta(\log_{1,5}(n)) \end{aligned}$$

5

$$T(n) = 0,5T(n/2) + 1/n$$

1 Atribuindo os valores

$$a = 0,5$$

$$b = 2$$

$$f(n) = 1/n$$

~~a ≥ 1~~ ✓
~~b > 1~~
f(n) é positivo ~~x~~

2 Concluindo

A relação de recorrência não atende os critérios necessários para a aplicação do método mestre.

6

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

1 Concluindo

A relação de recorrência não segue o formato de relação que pode ser utilizado pelo método mestre.

5. Análise de Algoritmos Recursivos com o Método da Substituição

(1)

O que é?

O método da substituição envolve dois passos:

1. Pressupor a solução da recorrência
2. Provar que a suposição é correta por indução

(2)

Revisão de indução matemática

A indução matemática é um método de prova usado para demonstrar que uma afirmação é verdadeira para todos os números naturais, mostrando que a afirmação é verdadeira para o número 1 e que, se for verdadeira para um número qualquer k , então também será verdadeira para o número $k+1$.

Exemplo

Somatório dos primeiros n inteiros

$$\begin{array}{ll} 1 & = 1 \\ 1 + 2 & = 3 \\ 1 + 2 + 3 & = 6 \\ 1 + 2 + 3 + 4 & = 10 \\ 1 + 2 + 3 + 4 + 5 & = 15 \end{array}$$



$$\begin{array}{ll} 1 & = 1 = 1 * 2 / 2 \\ 1 + 2 & = 3 = 1 * 3 / 2 \\ 1 + 2 + 3 & = 6 = 1 * 4 / 2 \\ 1 + 2 + 3 + 4 & = 10 = 1 * 5 / 2 \\ 1 + 2 + 3 + 4 + 5 & = 15 = 1 * 6 / 2 \\ 1 + 2 + 3 + 4 + \dots + n & = n * (n + 1) / 2 \end{array}$$

1 Queremos provar que

$$1 + 2 + 3 + 4 + \dots + n = n * (n + 1) / 2$$

$$\forall n \in \mathbb{N}, n \geq 1$$

2 Provando para $n = 1$

$$n = n * (n+1)/2$$

$$1 = 1 * (1+1)/2$$

$$1 = 2/2$$

$$1 = 1 \text{ OK}$$

3 Definindo a hipótese indutiva

$$1 + 2 + 3 + 4 + \dots + k = k * (k + 1) / 2$$

$$\forall k \in \mathbb{N}, k \geq 1$$

4 Provando para $k+1$

$$1 + 2 + 3 + 4 + \dots + k = k * (k + 1) / 2 \text{ Hipótese Indutiva}$$

$$1 + 2 + 3 + 4 + \dots + k + k + 1 = (k + 1) * (k + 1 + 1) / 2$$

$$(k^2 + k)/2 + k + 1 = (k + 1) * (k + 2)/2$$

$$(k^2 + k)/2 + k + 1 = (k^2 + 3k + 2)/2$$

$$(k^2 + k) + 2(k + 1)/2 = (k^2 + 3k + 2)/2$$

$$(k^2 + k + 2k + 2)/2 = (k^2 + 3k + 2)/2$$

$$(k^2 + 3k + 2)/2 = (k^2 + 3k + 2)/2 \text{ OK}$$

4 Concluindo

Portanto, pelo princípio da matemática indutiva,

$1+2+3+\dots+n = n(n+1)/2$ é verdade para $\forall n \in \mathbb{N}, n \geq 1$

(3) Exemplos

1

$$t(n) \left(\begin{array}{l} t(1) = \Theta(1) \\ t(n) = 2t(n-1) + \Theta(1) \end{array} \right)$$

1 Descobrindo a forma fechada da recorrência

1

$$t(n) = 2t(n-1) + 1$$

2

$$t(n-1) = 2t(n-2) + 1 \longrightarrow t(n) = 2t(n-1) + 1$$

$$t(n) = 2[2t(n-2) + 1] + 1$$

$$t(n) = 4t(n-2) + 3$$

3

$$t(n-2) = 2t(n-3) + 1 \longrightarrow t(n) = 4t(n-2) + 2$$

$$t(n) = 4[2t(n-3) + 1] + 2$$

$$t(n) = 8t(n-3) + 7$$

k

$$t(n) = 2^k t(n-k) + 2^k - 1$$

$$t(1) = 1$$

$$n-k = 1$$

$$k = n-1$$

$$t(n) = 2^{n-1} t(n-(n-1)) + 2^{n-1} - 1$$

$$t(n) = 2^{n-1} t(1) + 2^{n-1} - 1$$

$$t(n) = 2^{n-1} + 2^{n-1} - 1$$

$$t(n) = 2^n - 1$$

Calculando $2^{n-1} + 2^{n-1}$

$$\begin{aligned} 2^{n-1} + 2^{n-1} &= 2 + 2^{n-1} \\ &= 2^1 * 2^{n-1} \\ &= 2^{n-1+1} \\ &= 2^n \end{aligned}$$

2 Utilizando o método da substituição

1 Queremos provar que

$$t(n) = 2^n - 1$$

2 Provando para $n = 1$ (Caso Base)

$$\begin{aligned} t(1) &= 2^1 - 1 \\ t(1) &= 2 - 1 \\ t(1) &= 1 \text{ OK} \end{aligned}$$

3 Definindo a hipótese indutiva

$$t(k) = 2^k - 1$$

4 Provando para $k+1$ (Caso Indutivo)

$$t(k+1) = 2^{k+1} - 1$$

Temos

$$\begin{aligned} t(k) &= 2^k - 1 \text{ Hipótese Indutiva} \\ t(k) &= 2t(k-1) + 1 \text{ relação de recorrência} \end{aligned}$$

Realizando prova

$$\begin{aligned} t(k) &= 2t(k-1) + 1 \\ t(k+1) &= 2t((k+1)-1) + 1 \\ t(k+1) &= 2t(k) + 1 \\ t(k+1) &= 2(2^k - 1) + 1 \\ t(k+1) &= 2^{k+1} - 2 + 1 \\ t(k+1) &= 2^{k+1} - 1 \text{ OK} \end{aligned}$$

4 Concluindo

Portanto, pelo princípio da matemática indutiva, provamos que:

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

2

$$t(n) \leftarrow t(1) = \Theta(0)$$

$$t(n) = 4t(n/2) + n^2$$

1

Descobrindo a forma fechada da recorrência com método mestre

1 Atribuindo os valores

$$a = 4$$

$$b = 2$$

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

$a \geq 1$ ✓
 $b > 1$ ✓
 $f(n)$ é positivo ✓

2 Calculando logaritmo

$$n^{\log_b(a)} = n^{\log_2(4)} = n^2$$

$$\begin{aligned} \log_2(4) &= \log_2(2^2) \\ &= 2 + \log_2(2) \\ &= 2 + \log_2(2) \\ &= 2 + 1 \\ &= 2 \end{aligned}$$

3 Fazendo comparação

$$(1) f(n) < n^{\log_b(a)} \times$$

$$(2) f(n) = n^{\log_b(a)} \checkmark \quad \text{1º Regra} \rightarrow n^2 = n^2$$

$$(3) f(n) > n^{\log_b(a)} \times$$

4 Concluindo

Pela aplicação da 1º regra do teorema mestre
concluímos que:

$$\begin{aligned} T(n) &= f(n) * \log_b(n) \\ T(n) &= n^2 * \log_2(n) \end{aligned}$$

2

Utilizando o método da substituição

1 Queremos provar que

$$t(n) = n^2 * \log_2(n)$$

2 Provando para $n = 1$ (Caso Base)

$$t(1) = 1^2 * \log_2(1)$$

$$t(1) = 1 * 0$$

$$t(1) = 0 \text{ OK}$$

3 Definindo a hipótese indutiva

$$t(k/2) = k^2/4 * \log_2(k/2)$$

4 Provando para k (Caso Indutivo)

$$t(k) = k^2 * \log_2(k)$$

Temos

$$t(k/2) = k^2/4 * \log_2(k/2) \text{ Hipótese Indutiva}$$

$$t(k) = 4t(k/2) + k^2 \text{ relação de recorrência}$$

Realizando prova

$$\begin{aligned} t(k) &= 4t(k/2) + k^2 \\ &= 4[k^2/4 * \log_2(k/2)] + k^2 \\ &= k^2 * \log_2(k/2) + k^2 \\ &= k^2[\log_2(k) - \log_2(2)] + k^2 \\ &= k^2[\log_2(k) - 1] + k^2 \\ &= k^2 * \log_2(k) - k^2 + k^2 \\ &= k^2 * \log_2(k) \text{ OK} \end{aligned}$$

5 Concluindo

Portanto, pelo princípio da matemática indutiva,
provamos que:

$$t(n) = \Theta(n^2 * \log_2(n))$$

6. Análise Amortizada

(1) O que é?

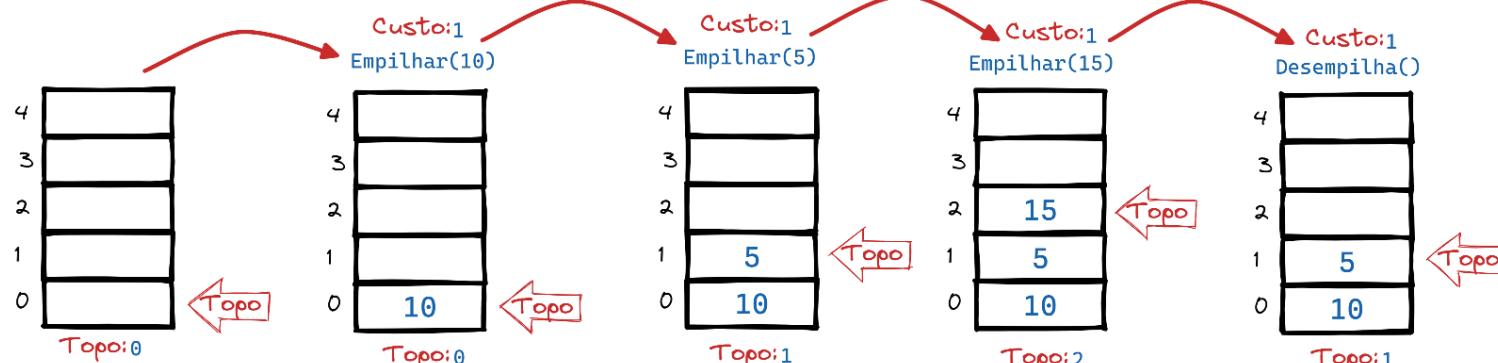
- * Analisa uma sequência de operações
- * Custo médio por operação

(2) Método Agregado para Análise Amortizada

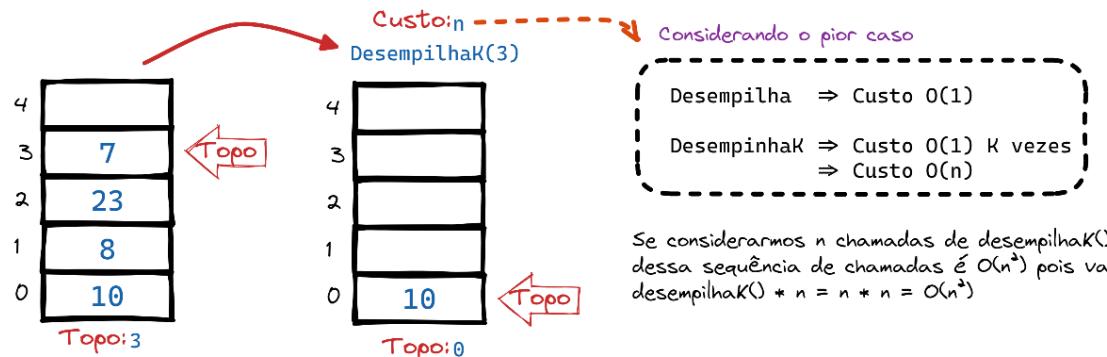
Seja $T(n)$ o custo de uma sequência de n operações
O custo amortizado por operação é $T(n)/n$

(3) Exemplo usando uma pilha

1 Observando custo dos métodos da pilha



2 Observando o custo do método desempilhaK()



Se considerarmos n chamadas de desempilhaK() o custo dessa sequência de chamadas é $O(n^2)$ pois vai ser o custo de desempilhaK() * $n = n * n = O(n^2)$

3 Usando o método agregado

Qualquer sequência de n operações envolvendo empilhar(), desempilhar() e desempilhaK() tem custo máximo $O(n)$

Custo amortizado de cada operação: $O(n)/n = O(1)$

7. Prova de corretude de algoritmos iterativos

(1) Exemplo

1

Provando a corretude do algoritmo que calcula a soma dos n primeiros números

```
soma(n)
    sum = 0
    for i = 1 to n do
        sum = sum + i
    return sum
```

1 Derivando um invariante de laço

Após k iterações, onde $0 \leq k \leq n$

$$\text{sum} = \sum_{x=1}^k x$$

2 Inicialização: antes da 1º iteração ($k = 0$)

$$\text{sum} = \sum_{x=1}^k x = \sum_{x=1}^0 x = 0$$

3 Manutenção

Hipótese indutiva: Após k iterações $\text{sum} = \sum_{x=1}^k x$

Provar: O invariante é mantido após $\text{sum} = \sum_{x=1}^{k+1} x$ $k+1$ iterações

Após a iteração 1: $\text{sum}_1 = \text{sum}_0 + 1 = 1$

Após a iteração 2: $\text{sum}_2 = \text{sum}_1 + 2 = 3$

Após a iteração 3: $\text{sum}_3 = \text{sum}_2 + 3 = 6$

Após a iteração k : $\text{sum}_k = \text{sum}_{k-1} + k$

$$\text{sum}_{k+1} = \text{sum}_k + (k+1) = \sum_{x=1}^k x + (k+1) = \sum_{x=1}^{k+1} x$$

hipótese indutiva

3 Terminação

Mostrar que quando o laço termina: $\text{sum} = \sum_{x=1}^n x$

Após n iterações, o algoritmo sai do laço.

De acordo com o invariante de laço após k iterações, $\text{sum} = \sum_{x=1}^k x$

$$\text{Logo, sum} = \sum_{x=1}^n x$$

8. Prova de corretude de algoritmos recursivos

(1) Exemplo

1

Provando a corretude do algoritmo que calcula o fatorial

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

1

Conjectura

$fatorial(n) = n!$, para $n \geq 0$

2

Caso base 

Se $n = 0$, então $fatorial(n) = 1$, pois $0! = 1$

Pelo algoritmo temos que:

$fatorial(0) = 1$ 

3

Hipótese indutiva

$fatorial(k) = k!$, para $k \geq 0$

4

Devemos provar que

$fatorial(k+1) = (k+1)!$

5

Passo indutivo 

$$\begin{aligned}fatorial(k+1) &= fatorial(k) * (k+1) \\&= k! * (k+1) \\&= (k+1)!\end{aligned}$$

Pela definição de fatorial