Tópico 4- Recorrências

Recursividade, divisão e conquista. Análise de recorrências. Prática e discussão com problemas computacionais relevantes.

Sumário

- Recursividade, Divisão e Conquista.
- Recorrências.
- Recorrências e Algoritmos.
- Hipótese “facilitadoras”.
- Método de Substituição.
- Método da Árvore de recursão.
- Método mestre.

Recursividade, Divisão e Conquista

- Algoritmos recursivos solucionam um problema computacional através de chamadas a eles mesmos (recursão), uma ou várias vezes, solucionando assim subproblemas relacionados.

- Esses algoritmos adotam um paradigma de projeto de algoritmos chamado divisão e conquista.

Recursividade, Divisão e Conquista

- Três passos da divisão e conquista:

1. **Dividir**: divida o problema em um número de subproblemas que representam instâncias menores do mesmo problema.

2. **Conquistar**: resolve os subproblemas recursivamente. Se o tamanho dos subproblemas é pequeno o suficiente, solucione de forma direta.

3. **Combinar:** combine as soluções dos subproblemas para formar a solução do problema original.

Recursividade, Divisão e Conquista

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

p r

1	2	3	4	5	6	7	8
50	20	40	70	10	30	20	60

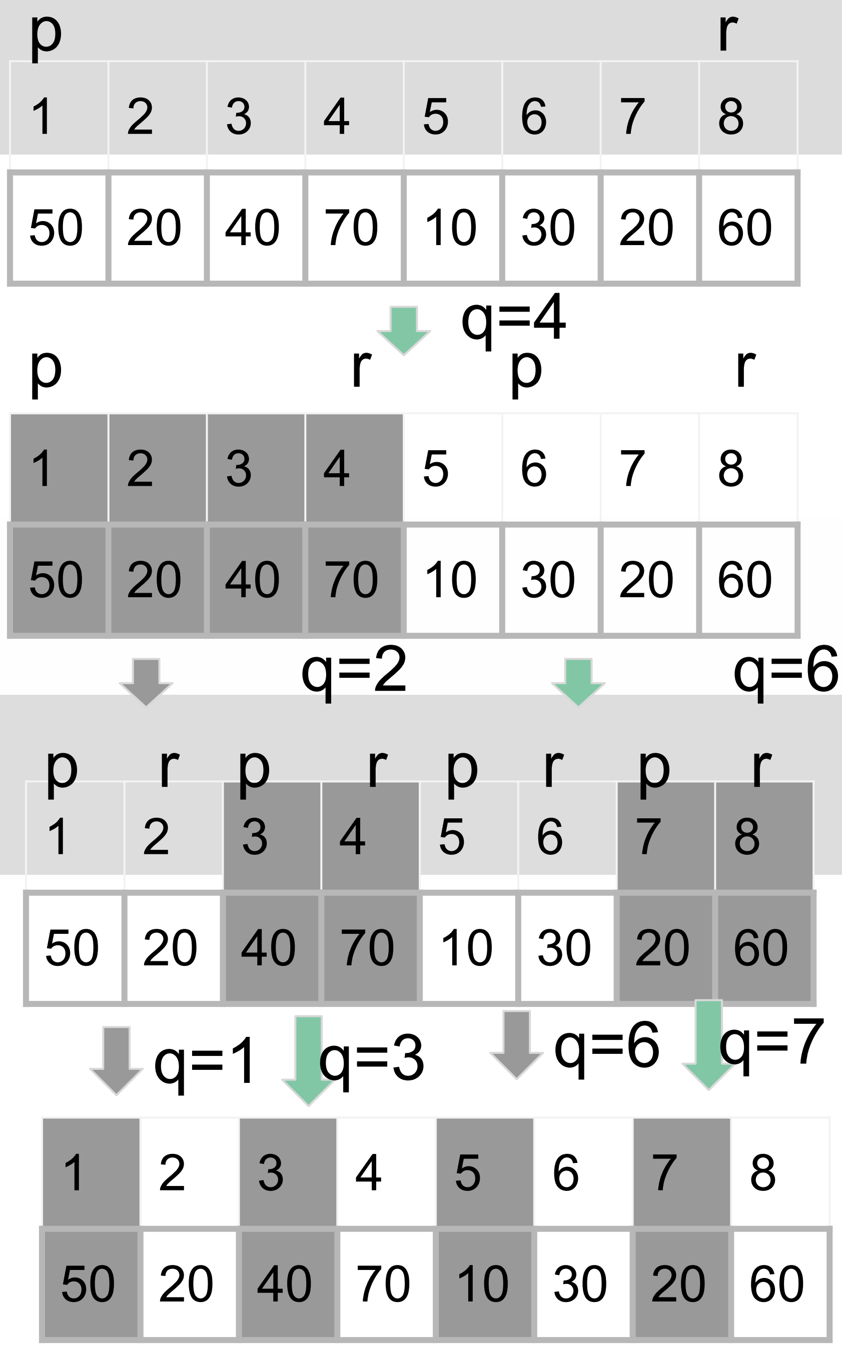
q=4

p r p r

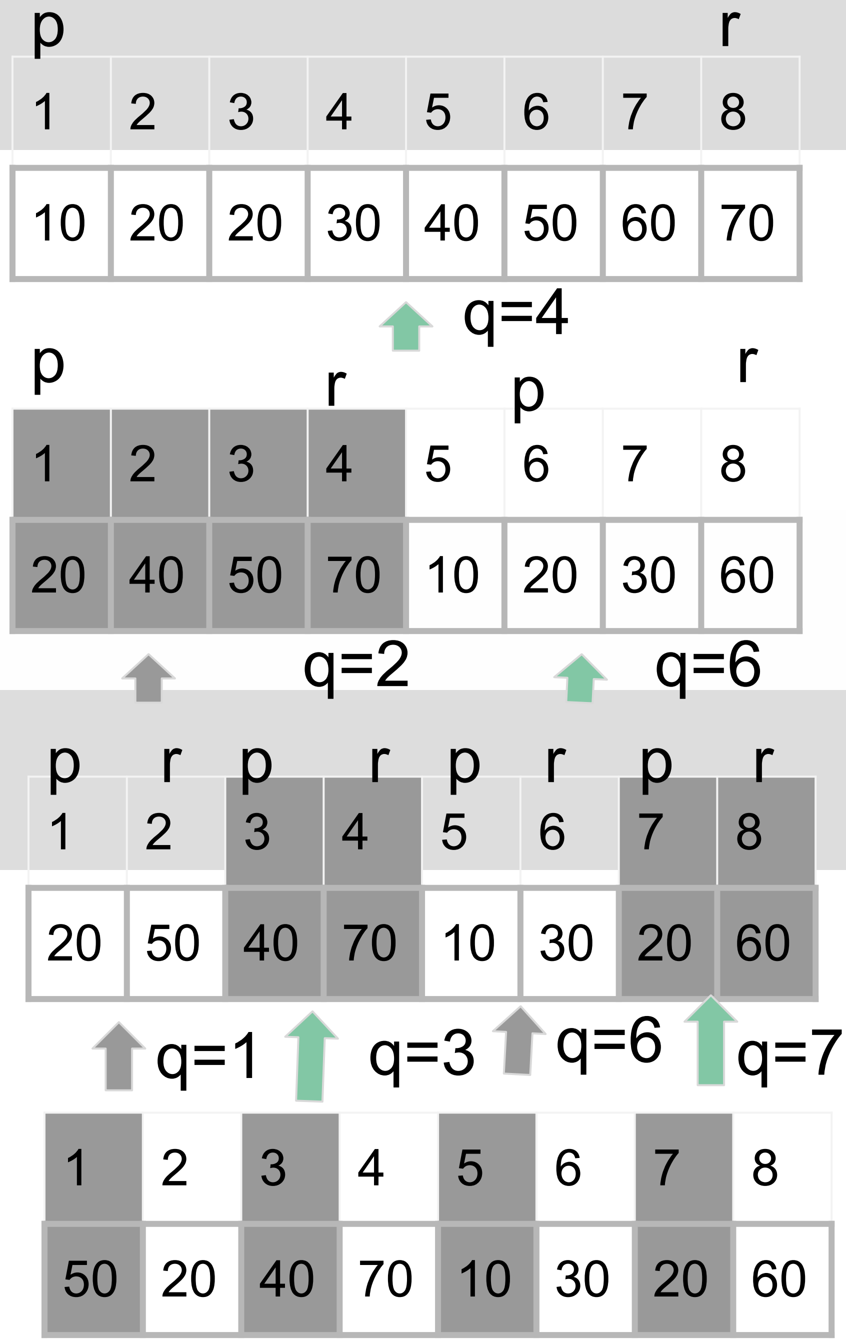
1	2	3	4	5	6	7	8
50	20	40	70	10	30	20	60

q=2 q=6

```
MERGE-SORT(A, p, r)
  if p < r
      q = ⌊(p + r)/2⌋
      MERGE-SORT(A, p, q)
      MERGE-SORT(A, q + 1, r)
      MERGE(A, p, q, r)
```



```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```



Recorrências

- O item sendo definido aparece como parte da definição.
- Chamadas definição recorrente, definição por recorrência ou definição por indução.
- Apresentam duas partes:
  1. **Condição básica**: casos simples do item a ser definido são explicitamente estabelecidos.
  2. **Passo Indutivo ou recorrente:** novos casos do item a ser definido são dados em função de casos anteriores.

Recorrências

- Exemplo: Relação de recorrência:

  $S(1)=2$

  $S(n)=2S(n-1)$

  S(n) pode ser calculada fazendo:

  $S(1)=2$

  $S(2)=2S(1)=4=2^2$.

  $S(3)=2S(2)=8=2^3$.

  $S(4)=2S(3)=16=2^4$.

  ….

  $S(n)=2^n$. **SOLUÇÃO EM FORMA FECHADA!!!**

Recorrências

- Relações de recorrência podem ser resolvidas aplicando a técnica “expanda, suponha e verifique”.

- Exemplo 0: $S(1)=2$

  $S(n)=2S(n-1)$

Expandir: $S(n)=\mathbf{2S(n-1)} = 2.[2S(n-2)]=\mathbf{2^2S(n-2)}=$

$= 2^2.[2S(n-3)]=\mathbf{2^3S(n-3)}=$

$= 2^3.[2S(n-4)]=\mathbf{2^4S(n-4)}$

Logo, $S(n)=2^kS(n-k)$ e para $k=n-1$, teremos:

$S(n) = 2^{n-1}S(n-(n-1))= 2^{n-1}S(1)=2^n$.

Suponha: $S(n)=2^n$ para $n\geq1$.

Recorrências

- Exemplo 0: $S(1)=2$
  $S(n)=2S(n-1)$

Verifique: $S(n)=2^n$ para $n\geq 1$ considerando a Recorrência.

Provando por indução:

$n=1 \Longrightarrow 2^1 = S(1)=2$ Ok

Para $n=k$, suponha que $S(k)=2^k$.

Para $n = k+1$, pela relação de recorrência:

$S(k+1)=2S(k)$

$=2.2^k$, pela H.I.

$=2^{k+1}$.

Logo, a forma fechada proposta está correta.

Recorrências

- Exemplo 1: Encontre a forma fechada para

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

- Exemplo 2: Encontre a forma fechada para

$$T(1)=1$$
$$T(n)=nT(n-1) \quad n \geq 2$$

Recorrências e Algoritmos

Exemplo:

```
int exp1(int a, int b) {
1.  if (b == 1)
2.      return a;
    else
3.      return a*exp1(a, b-1);
}
```

- Sejam:
  - T (b): função de complexidade
  - b: número de vezes que teremos de multiplicar a base para obter a exponenciação.

- Custos:
  - Custo das linhas 1 e 2: O (1) .
  - Custo da linha 3: Número de chamadas recursivas.

Recorrências e Algoritmos

Exemplo:

```
int exp1(int a, int b) {
1.  if (b == 1)
2.      return a;
    else
3.      return a*exp1(a, b-1);
}
```

T(1) = 2.

T (b) = 4 + T(b-1).

Expandindo: T (b) = 4 + T(b-1)=

= 4+(4+T(b-2))=

=**2.4+T(b-2)**=

=2.4+4+T(b-3)=

= **3.4+T(b-3)**

T(b)=4.k+T(b-k), para k=b-1 temos

T(b) = 4(b-1)+T(1)=4b-4+2=**4b-2**

Supondo: T(b)=4b-2.

Recorrências e Algoritmos

Exemplo:

```
int exp1(int a, int b) {
1.  if (b == 1)
2.      return a;
    else
3.      return a*exp1(a, b-1);
}
```

$T(1) = 2.$

$T(b) = 4 + T(b-1).$

Verificando:

$b=1 \Longrightarrow 4.(1)-2=2=T(1)$ Ok

Suponha que para $b=k$, temos

$T(k)=4k-2.$

Para $b=k+1 \Longrightarrow$

$T(k+1)=4+T(k)=$

$=4+4k-2=4(k+1)-2$

Logo, $T(k+1) = 4(k+1)-2.$

Recorrências e Algoritmos

Exemplo:

```
int exp2(int a, int b) {
    if (b == 1)
        return a;
    if ((b % 2) == 0)
        return exp2(a*a, b/2);
    else
        return a*exp2(a, b-1);
}
```

- Para b=2m (par).
  - 2 comparações
  - 1 módulo
  - 1 produto
  - 1 divisão
  - 1 retorno
  - 1 chamada recursiva
- Logo,
  - $T(b) = 6 + T(b/2)$ para b=2m.

Recorrências e Algoritmos

Exemplo:

```
int exp2(int a, int b) {
if (b == 1)
        return a;
if ((b % 2) == 0)
    return exp2(a*a, b/2);
else
    return a*exp2(a, b-1);
}
```

- Para b=2m+1 (ímpar).
  - 2 comparações
  - 1 módulo
  - 1 produto
  - 1 subtração
  - 1 retorno
  - 1 chamada recursiva
- Logo,
  - $T(b) = 6 + T(b-1)$ para $b=2m+1$.

Recorrências e Algoritmos

Exemplo:

```
int exp2(int a, int b) {
    if (b == 1)
        return a;
    if ((b % 2) == 0)
        return exp2(a*a, b/2);
    else
        return a*exp2(a, b-1);
}
```

- Porém, se b=2m+1 (ímpar), b-1 é par.
  - $T(b) = 6 + T(b-1)$ para $b=2m+1$.

    $= 6+(6+T((b-1)/2))=$

    $= 12 + T((b-1)/2)$

    $\approx 12 + T(b/2)$
- Temos $T(b) = 12+T(b/2)$

  Expandir

$T(b)=12+T(b/2)=12+(12+T(b/4))$

$=$ $\mathbf{2.12+T(b/2^2)}$$=2.12+(12+T(b/8))$

$=$ $\mathbf{3.12+T(b/2^3)}$

Logo, $T(b)=12.k +T(b/2^k)$

Recorrências e Algoritmos

Exemplo:

```
int exp2(int a, int b) {
    if (b == 1)
        return a;
    if ((b % 2) == 0)
        return exp2(a*a, b/2);
    else
        return a*exp2(a, b-1);
}
```

$T(b)=12.k + T(b/2^k)$

A expansão termina quando

$T(1) = T(b/2^k)$,

temos:

$b/2^k=1 \Longrightarrow 2^k=b \Longrightarrow k = \log_2 b.$

Logo,

$T(b) = 12\log_2 b + T(1)$

Recorrências e Algoritmos

Exemplo:

```
int exp2(int a, int b) {
    if (b == 1)
        return a;
    if ((b % 2) == 0)
        return exp2(a*a, b/2);
    else
        return a*exp2(a, b-1);
}
```

- $T(b) = 12+T(b/2)$

  Suponha

  $T(b) = 12\log_2 b + T(1)$

  Verifique

b=1 $T(1)=\log_2 1 + T(1) = T(1)$ Ok

Suponha para $0 \leq r \leq k$

$T(r)=12\log_2 r + T(1)$

Para $b = k+1$

$T(k+1)=12+T((k+1)/2)$

$T(k+1)=12+12\log_2[(k+1)/2] + T(1)$

$T(k+1) =12+12[\log_2(k+1)-\log_2 2]+T(1)$

$T(k+1)=12+12\log_2(k+1) -12 +T(1)$

$\mathbf{T(k+1)=12\log_2(k+1) +T(1)}$

Recorrências e Algoritmos

- Algoritmo Exp1:

$$T(b)=4b-2 \Longrightarrow T(b) = O(b)$$

- Algoritmo Exp2:

$$T(b) = 12\log_2 b + T(1) \Longrightarrow T(b) = O(\log_2 b)$$

Recorrências e Algoritmos

- T (n) : tempo no pior caso (n = r − p + 1).

MERGE-SORT(A, p, r)	
if $p < r$	c_0
$\quad q = \lfloor (p + r)/2 \rfloor$	c_1
$\quad$ MERGE-SORT(A, p, q)	$T(n/2)$
$\quad$ MERGE-SORT$(A, q + 1, r)$	$T(n/2)$
$\quad$ MERGE(A, p, q, r)	$c_3 n$

- Relação de recorrência:
  1. $T(1) = \Theta(1)$ $\qquad$ n=1.
  2. $T(n) = 2.T(n/2)+c_0+c_1+n.c_3$ $\qquad$ n>1.

Recorrências

- Relação de recorrência:
  1. $T(n) = \Theta(1)$ n=1.
  2. $T(n) = 2.T(n/2)+c_0+c_1+n.c_3$ n>1.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n=1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n>1 \end{cases}$$

Recorrências

- Definida uma relação de recorrência, precisamos encontrar g(n) tal que $T(n) \in \Theta(g(n))$.

- Há métodos que auxiliam na determinação de
  $T(n) \in \Theta(g(n))$.

Hipóteses “facilitadoras”

- Ignoramos os arredondamentos para cima ou para baixo. Assim, valores inteiros são assumidos para as variáveis.
- Casos base da recursão podem ser ignorados. Assume-se que T(n) tenha valor constante e baixo quando n é baixo.
- Por exemplo, no caso do algoritmo MERGE-SORT temos:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n=1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n>1 \end{cases}$$

- Todavia, podemos considerar apenas:

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

Métoda da Substituição

- **Passo1**: Suponha uma possível solução.

- **Passo 2**: Prove por indução matemática que a solução proposta é válida.

Métoda da Substituição

Exemplo1: $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Passo 1: Suponha que $T(n) = O(n \lg n)$ **(Pq?)**

Passo 2: Provar por indução que

$T(n) \leq cn \lg n$ para algum c>0.

Método de Substituição

- ✓ Por hipótese de indução, para n<k, temos

  $T(n) \leq cn \lg n$

- ✓ Vamos provar para n =k com $T(k) = 2T(\lfloor k/2 \rfloor) + k$
- ✓ Como k/2 < k, por indução, temos $T(k/2) \leq c\,(k/2) \lg (k/2)$
- ✓ Dada a natureza não decrescente das funções envolvidas, ignoramos a função piso $\lfloor k/2 \rfloor$. Logo,

$$T(k) = 2T(k/2) + k \leq 2c\,(k/2) \lg (k/2) + k$$

$$T(k) \leq c\,k \lg k - c\,k \lg 2 + k$$

$$T(k) \leq c\,k \lg k - c\,k + k$$

$$T(k) \leq c\,k \lg k - (c-1)k$$

$$T(k) \leq c\,k \lg k, \quad para\ c \geq 1$$

Método de Substituição

E o passo base?

- ✓ $T(1)=1$ e $T(n) \leq cn \lg n$

  $T(1) \leq c(1)\ (\lg 1\ ) = 0$

  ?????
- ✓ Pela notação assintótica, devemo encontrar $n \geq n_0$
- ✓ $T(1) = 1$ Caso base da Relação de Recorrência (RR).
- ✓ Também temos pela RR:

  $T(2) = 2T(2/2)+2 = 2.1+2=4$

  $T(3) = 2T(3/2)+3 = 2.1+3=5$
- ✓ Logo, devemos achar uma constante c tal que:

  $T(2) \leq c(2)\ (\lg 2\ )$ e $T(3) \leq c(3)\ (\lg 3\ )$
- ✓ A constante $c \geq 2$, satisfaz a condição acima.

  $T(2) \leq c(2)\ (\lg 2\ ) \Longrightarrow 4 = (2)(2)\ (\lg 2\ )$

  $T(3) \leq c(3)\ (\lg 3\ ) \Longrightarrow 5 \leq (2)(3)\ (\lg 3\ ) = 6.(1{,}58)$

Método de Substituição

Logo, foi provado por indução que a relação de recorrência:

$$T(1) = 1 \text{ e } T(n) = 2T(\lfloor n/2 \rfloor) + n \text{ , } n > 1$$

Pode ser expressa como $T(n) = O(n \lg n)$ dado que

$T(\mathrm{n}) \leq cn \lg n$ para $\mathrm{c} \geq 2$, $\mathrm{n} \geq 2$ ($\mathrm{n}_0 = 2$)

Método da Substituição

Exemplo 2: $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

Passo1: Suponha que $T(n) = O(n)$

Passo2: Provar por indução que $T(n) \leq cn$ para $c > 0$

Dem:

✓Provando para $n = k$, temos que verificar $T(k) \leq ck$

✓$T(k) = T(\lfloor k/2 \rfloor) + T(\lceil k/2 \rceil) + 1 \leq ck/2 + ck/2 + 1$

✓$T(k) \leq ck+1 \nRightarrow T(k) \leq ck$ PROBLEMA!!

Métoda da Substituição

Passo 1: Suponha que $T(n) = O(n)$ (MANTIDO)

Passo 2: Provar por indução que $T(n) \leq cn - d$ para $c, d > 0$

Dem: Provando para n = k, temos que verificar

$T(k) \leq ck - d$

$T(k) = T(\lfloor k/2 \rfloor) + T(\lceil k/2 \rceil) + 1 \leq ck/2 - d + ck/2 - d + 1$

$T(k) \leq ck - 2d + 1$

$T(k) \leq ck - d$, para $d \geq 1$ (Pq????)

$T(k) \leq ck$

Método da Substituição

Exemplo 3: $T(n) = 2T(\lfloor \mathrm{SQRT}(n) \rfloor) + \lg n$

Mudança de variável:

- ✓Seja m=lg n para SQRT(n) inteiro.
  $n=2^m$ e $\mathrm{SQRT}(n) = 2^{m/2}$.
- ✓$T(n) = 2T(\lfloor \mathrm{SQRT}(n) \rfloor) + \lg n \Rightarrow T(2^m) = 2T(2^{m/2}) + m$
- ✓Seja $S(m) = T(2^m)$, temos:
  $T(2^m) = 2T(2^{m/2}) + m \Rightarrow S(m) = 2S(m/2) + m$
- ✓Mesma recorrência do exemplo 1, logo
  $S(m) = O(m \lg m) = O(\lg n(\lg \lg n))$

Árvore de Recorrência

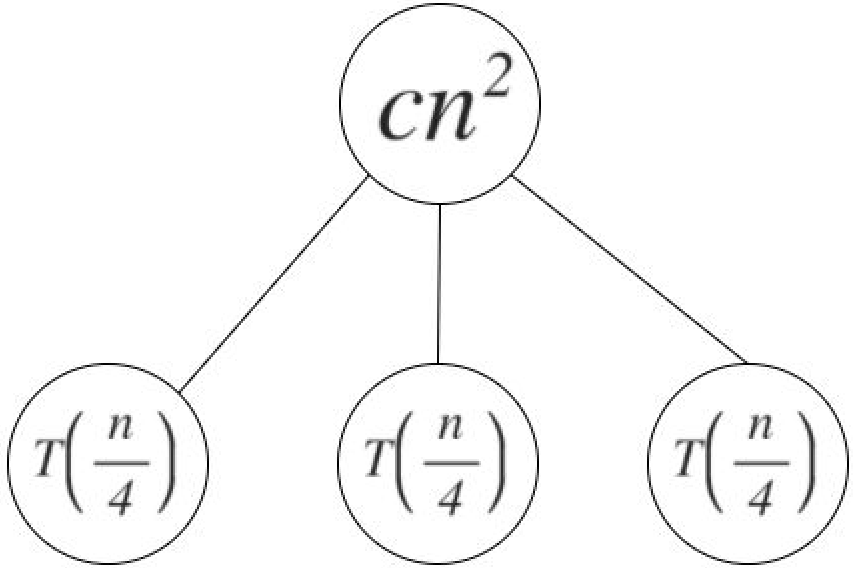
- Trata-se de uma representação visual da hierarquia das chamadas recursivas.
- Cada nó representa o custo de um subproblema.
- Auxilia na definição da função utilizada no Passo 1 do Método de Substituição (MS).

Árvore de Recorrência

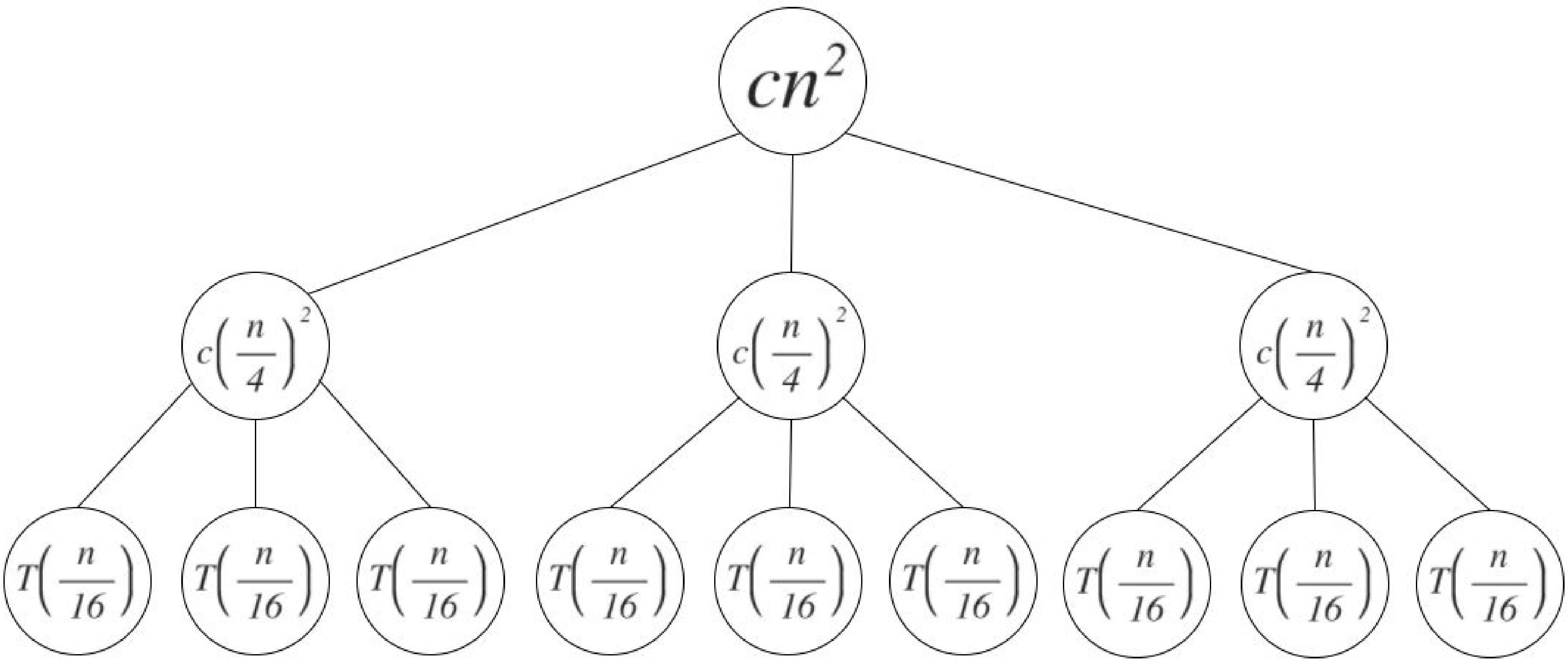
Exemplo1: $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$

- Hipóteses "facilitadoras":
  - A função piso (assim como a teto) não costuma ter relevância quando resolvendo recorrências.
  - Por conveniência, vamos assumir que n é uma potência do número 4. Assim, todo subproblema terá um tamanho inteiro.

Árvore de Recorrência

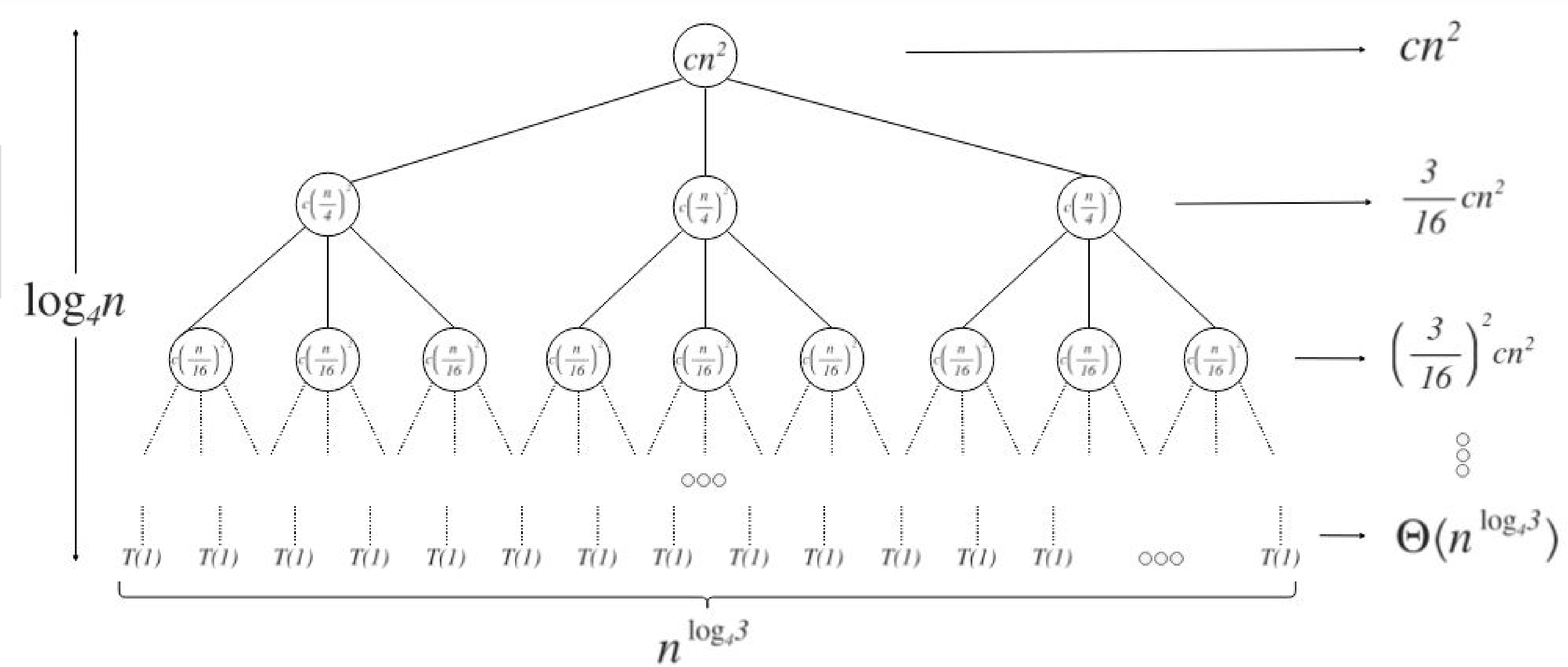


Árvore de Recorrência



Árvore de Recorrência

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$



Árvore de Recorrência

- O tamanho dos subproblemas decrescem a um fator 4 cada vez que descemos um nível na árvore.
- O tamanho de um subproblema para um nó na profundidade i é $n/4^i$.
- No último nível da árvore:

$$n/4^i = 1 \Rightarrow i=\log_4 n$$

- A árvore terá $\log_4 n + 1$ níveis.

Árvore de Recorrência

- No nível i, o número de nós será 3^i.
- O custo associado a cada nó será $c(n/4^i)^2$.
- Logo, o custo no nível i será

$$3^i c(n/4^i)^2 = (3/16)^i cn^2.$$

Árvore de Recorrência

- No último nível ($i=\log_4 n$), temos $3^{\log_4 n}$ nós.
- Assumindo custo constante T(1) **(Pq???).**
- O custo total no último nível será

$$3^{\log_4 n}T(1) = \Theta(3^{\log_4 n})=\Theta(n^{\log_4 3})$$

Árvore de Recorrência

O custo total considerando todos os níveis será:

$$T(n) = \sum_{i=0}^{\log_4 n - 1} (3/16)^i cn^2 + \Theta(n^{\log_4 3})$$

$$T(n) \leq \sum_{i=0}^{\infty} (3/16)^i cn^2 + \Theta(n^{\log_4 3})$$

$$T(n) \leq (1/(1-3/16))cn^2 + \Theta(n^{\log_4 3})$$

$$T(n) \leq (16/13)cn^2 + \Theta(n^{\log_4 3})$$

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

!!!NÃO TÃO RÁPIDO!!!

Árvore de Recorrência

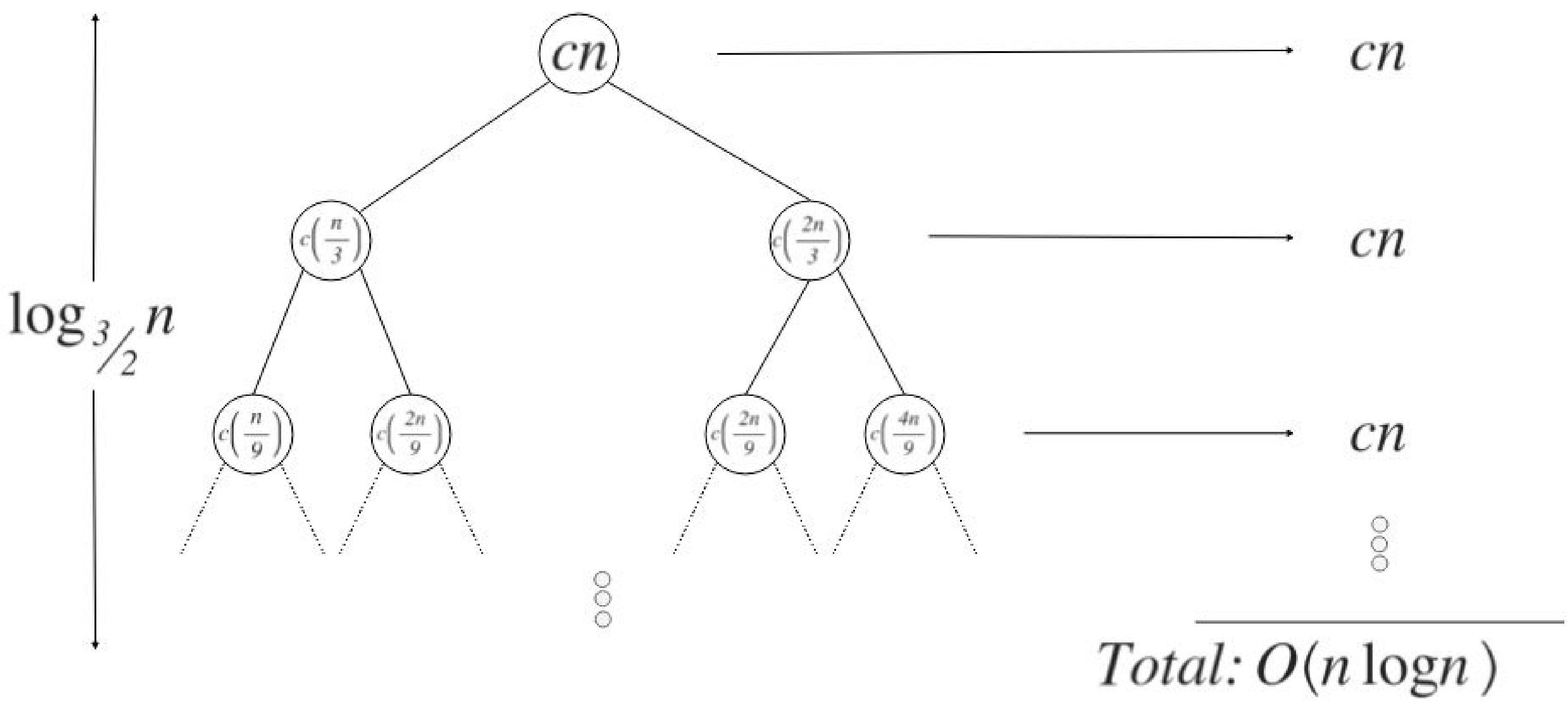
- Via árvore de recursão, temos uma suposição para o Passo 1 do Método de Substituição: $T(n) \leq dn^2$.
- Agora, vamos para o Passo 2:
- A H.I. forte ocorre para $r \leq k$, para $n=k+1$:

$$
\begin{aligned}
T(k+1) &= 3T(\lfloor (k+1)/4 \rfloor) + \Theta((k+1)^2) \\
&\leq 3T(\lfloor (k+1)/4 \rfloor) + c(k+1)^2 \\
&\leq 3d((k+1)/4)^2 + c(k+1)^2 \\
&= (3/16)d(k+1)^2 + c(k+1)^2 \\
&= [(3/16)d + c](k+1)^2 \\
&\leq d(k+1)^2 \text{ para } d \geq (16/13)c
\end{aligned}
$$

(Pq???)

Árvore de Recorrência

Exemplo 2: $T(n) = T(n/3) + T(2n/3) + O(n)$



Árvore de Recorrência

- Novamente, vamos desconsiderar funções piso e teto.
- O maior caminho do nó raiz até um nó folha passa pelos nós n, (2/3)n, $(2/3)^2n,\ldots,(2/3)^kn,\ldots,1$.
- Logo, $(2/3)^kn=1 \Rightarrow k = \log_{3/2}n$ (altura!!)
- Nem todo nível da árvore contribui com um custo *cn*.

Árvore de Recorrência

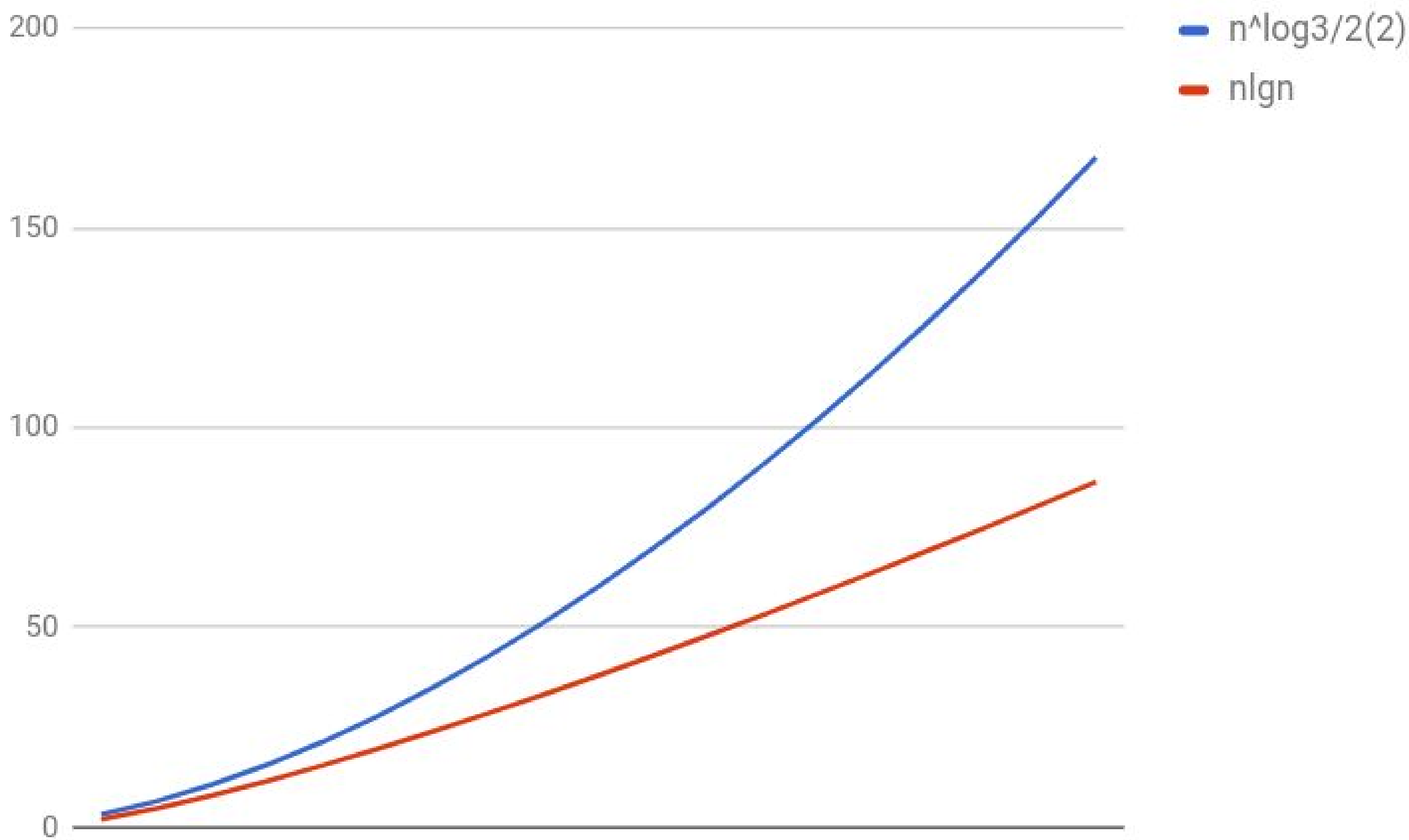
- Considerando o custo nos nós folhas em uma árvore binária completa com altura $\log_{3/2} n$ (altura!!), teremos:

$$2^{\log_{3/2} n} = n^{\log_{3/2} 2} \text{ folhas.}$$

- Desde que o custo de cada folha é constante, o custo total no último nível poderia ser $\Theta(n^{\log_{3/2} 2})$

- Como $\log_{3/2} 2 > 1$, tal custo será $\omega(n \lg n)$

Árvore de Recorrência



Árvore de Recorrência

- Relembrando que a altura da árvore será no máximo:

$$h = 2^{\log 3/2 n} = n^{\log_{3/2} 2} \text{ folhas.}$$

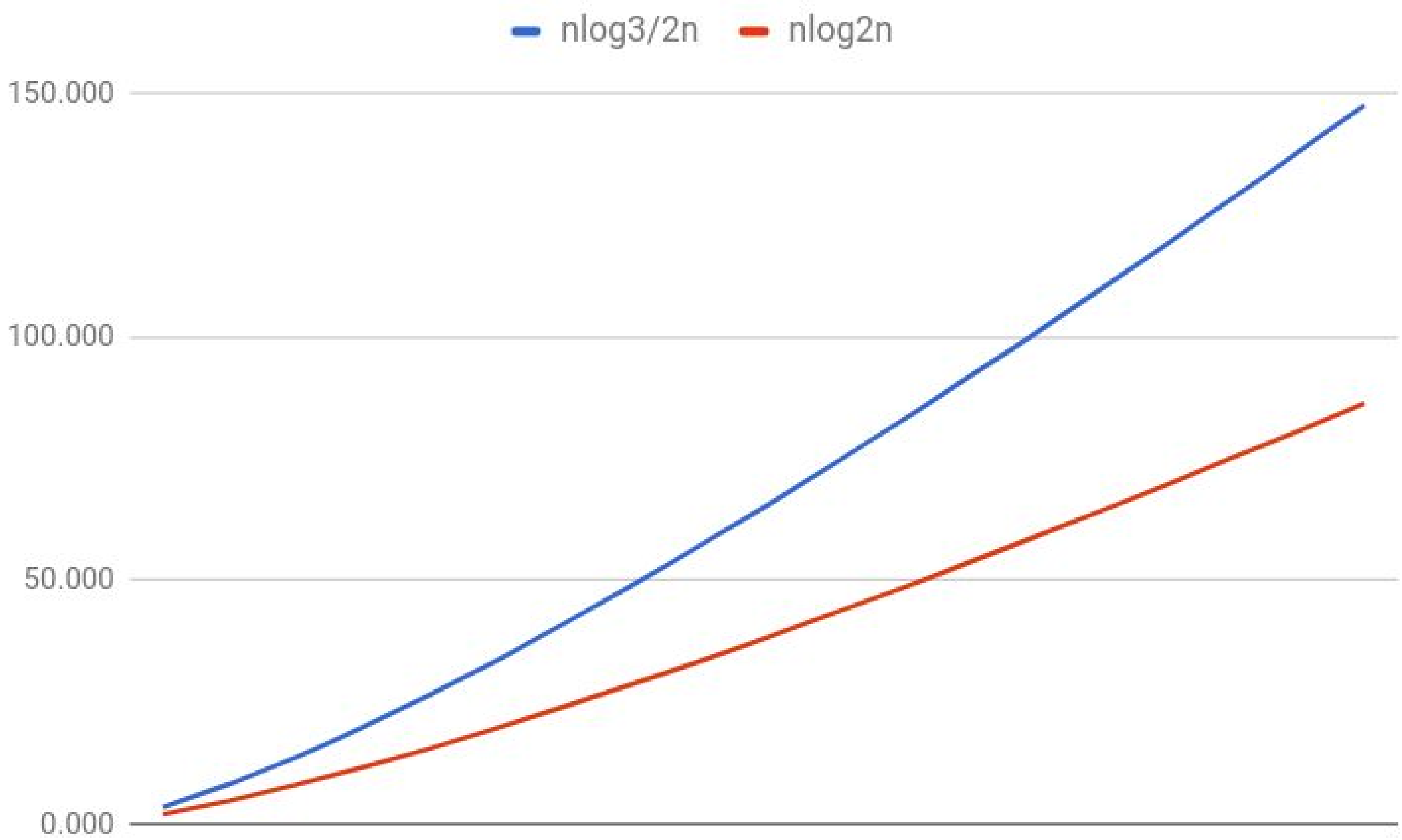
- O custo em cada nível da árvore é

$$\text{custo} = cn.$$

- Espera-se que o custo total de uma árvore com altura h e custo *cn* será

$$O(cn\log_{3/2} n) = O(n\lg n)$$

Árvore de Recorrência



Árvore de Recursão

- Custo total superestimado:

$$T(n) = \sum_{i=0}^{\log_{3/2} n - 1} cn + \Theta(n^{\log_{3/2} 2})$$

- Podemos chegar em:

$$T(n) = O(n \lg n) + \omega(n \lg n)$$

- Vamos supor:

$$T(n) = O(n \lg n)$$

Árvore de Recorrência

- Podemos assumir

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

- Supondo no Passo 1 do M.S.:

$$T(n) = O(n \lg n)$$

- Provaremos no Passo 2 do M.S.:

$$T(n) \le dn \lg n \text{ para } \quad d>0$$

Árvore de Recorrência

- Novamente, vamos ignorar o caso base

- . Pela H.I. temos que para $n < k$ a desigualdade $T(n) \leq dn \lg n$ é válida.

Árvore de Recorrência

- Pela H.I, temos:

$$T(k/3) \le d\, k/3 \lg k/3$$

$$T(2k/3) \le d\, 2k/3 \lg 2k/3$$

- A relação de recorrência é:

$$T(k) = T(k/3) + T(2k/3) + ck$$

$$T(k) \le [d\,(k/3) \lg (k/3)] + [d\,(2k/3) \lg (2k/3)] + ck$$

Árvore de Recorrência

$$T(k) \le [d\,(k/3)\lg k - d\,(k/3)\lg 3] + [d\,(2k/3)\lg k - d\,(2k/3)\lg(3/2)] + ck$$

$$T(k) \le dk\lg k - d[(k/3)\lg 3 + (2k/3)\lg(3/2)] + ck$$

$$T(k) \le dk\lg k - dk[\lg 3 - 2/3] + ck$$

$$T(k) \le dk\lg k \text{ para } d \ge c/(\lg 3 - (2/3))$$

Método Mestre

- Fornece uma receita para resolver recorrências que devem estar na forma:

$$T(n) = aT(n/b) + f(n)$$

com $a \geq 1$ e $b > 1$ sendo constantes e $f(n)$ uma função assintoticamente positiva.

Método Mestre

Teorema: Seja T(n)=aT(n/b)+f(n), com a>0 e b>0 constantes e f(n) assintoticamente positiva. Temos que:

1. Se $f(n) = O(n^{\log_b a - \varepsilon})$ para alguma constante $\epsilon > 0$, então $T(n) = \Theta(n^{\log_b a})$
2. Se $f(n) = \Theta(n^{\log_b a})$, entao $T(n) = \Theta(n^{\log_b a} \lg n)$
3. Se $f(n) = \Omega(n^{\log_b a + \varepsilon})$ para alguma constatnte $\varepsilon > 0$ e se $af(n/b) \leq cf(n)$ para alguma constante $c < 1$ e todo n suficientemente grande, então $T(n) = \Theta(f(n))$

Método Mestre

Exemplo 1: $T(n)=9T(n/3) + n$

Temos, $a=9$, $b=3$, $f(n)=n$. Pelo teorema anterior, $n^{logb(a)}=n^{log3(9)}=\Theta(n^2)$.

Como $f(n)=O(n^{log3(9-\epsilon)})$, onde $\epsilon=1$, podemos aplicar o caso 1 do teorema mestre, concluindo que $T(n)=\Theta(n^2)$

Método Mestre

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

Temos, $a=1$, $b=3/2$, $f(n)=1$ com $n^{\log b(a)}=n^{\log 3/2(1)}=n^0=1$.

O caso 2 se aplica aqui, desde que $f(n)=\Theta(1)$, então a solução para a relação de recorrência será $T(n)=\Theta(\lg n)$

Método Mestre

Exemplo 3: $T(n)=3T(n/4) + n\lg n$

Temos, a=3, b=4, $f(n)=n\lg n$ com $n^{\log b(a)}=n^{\log 4(3)}=O(n^{0.793})$. Desde que $f(n)=\Omega(n^{\log 4(3+\epsilon)})$, onde $\epsilon \approx 0.2$, o caso 3 se aplica se as condições de regularidade ocorrem.

Para n suficientemente grande, temos

$af(n/b)= 3(n/4)\lg(n/4) \leq (3/4)n\lg n = cf(n)$ para $c=3/4$.

Logo, pelo caso 3, a solução para a relação de recorrência será $T(n)=\Theta(n\lg n)$

Método Mestre

Exemplo 4: $T(n)=2T(n/2) + nlgn$

Temos, $a=2$, $b=2$, $f(n)=nlgn$ com $n^{logb(a)}=n$.

A função $f(n)=nlgn$ é assintoticamente maior que $n^{logb(a)}=n$, mas não é polinomialmente maior:

$f(n)/n^{logb(a)}= (nlgn)/n = lgn$

onde lgn é assintoticamente menor que n^{ϵ} para qualquer constante positiva ϵ. Logo, o caso 3 não se aplica.