Complexidade de Algoritmos

Mariana Kolberg

Projeto de Algoritmos
Divisão e conquista

Complexidade de Algoritmos

Divisão e conquista

- Divide um problema em subproblemas independentes, resolve-os e combina as soluções obtidas em uma solução para o problema original.
  - resulta em um **processo recursivo** de decomposições e recombinações.

- Pode ser aplicado em problemas de:
  - buscas em tabelas, como buscas seqüencial e binária;
  - classificação, como classificação por seleção (selectionsort), por intercalação (mergesort) e por particionamento (quicksort);
  - multiplicação (de matrizes e de números binários, por exemplo);
  - seleção (para determinar máximo e mínimo, etc.).

Complexidade de Algoritmos

Somatório dos elementos de uma lista.

- Considere uma lista L de elementos do tipo inteiro.
  - Se L tem no máximo 1 elemento, a soma de seus elementos é trivial.
  - Caso contrário, este somatório pode ser visto como sendo **a soma dos elementos da primeira metade** de L, chamada L_1, com **os elementos da segunda metade**, chamada L_2.

somatorio(L):= se curta(L) // Tem comprimento de no máximo 1 ?

então retorna (L) // retona zero se L é vazia, ou o próprio elemento.

senão retorna (soma(somatorio(sublist1(L)), somatorio(sublist2(L)))).

Onde soma(r_1, r_2) = r_1+ r_2

Complexidade de Algoritmos

Classificação de listas por intercalação de sublistas

- O algoritmo recebe como entrada uma lista L e devolve esta lista classificada.
  - Se **L** tem comprimento no máximo 1, então **L** já está classificada.
  - Caso contrário, a lista **L** é dividida em duas listas *aproximadamente* do mesmo tamanho, $\mathbf{L_1}$ e $\mathbf{L_2}$ correspondendo a primeira e a segunda metades de **L**, respectivamente.

- $\mathbf{L_1}$ e $\mathbf{L_2}$ são recursivamente **classificadas** e **intercaladas** a fim de obter uma versão classificada da lista de entrada.

Complexidade de Algoritmos

```
Função Mergesort(d:D) → R

n ← compr(d);
se n ≤ 1
   então
   r ← d;
   retorne-saída(r);
   fim-então
   senão
   d1 ← Prim(d); d2 ← Fin(d);
   r1 ← Mergesort(d1); r2 ← Mergesort(d2);
   r ← Intercl(r1, r2);
   retorne-saída(r);
   fim-senão
fim-se
fim-Função
```

Complexidade de Algoritmos

Divisão e conquista

- se a entrada é simples, a saida é obtida diretamente;
- caso contrário, a entrada é decomposta e aplicado o mesmo processo.
- os resultados parciais são combinados para gerar uma saída para a entrada original.

Baseado nisto, podemos descrever a divisão e conquista binária como

Função: Div_Conq_2 $(d : D) \rightarrow R$

$$\text{Div\_Conq\_2}(d) = \begin{cases} \text{drt}(d) & \text{se smpl}(d) \\ \text{combn\_2}(\text{Div\_Conq\_2}(\text{part}_1(d)), \text{Div\_Conq\_2}(\text{part}_2(d))) & \text{caso contrário} \end{cases}$$

Onde

- as funções part1 e part2 decompõem a entrada;

- a função cmbn_2 combina as saídas;

- o procedimento smpl testa se a entrada é simples ou não; e

- a função drt dá a saída para entradas simples.

Complexidade de Algoritmos

Projeto e Análise de Algoritmos

O algoritmo Mergesort pode ser obtido especializando a formulação recursiva da divisão e conquista binária

```
Função Mergesort(d:D)→R
n←compr(d);
se n≤1
   então
   r←d;
   retorne-saída(r);
   fim-então
   senão
   d1←Prim(d); d2←Fin(d);
   r1←Mergesort(d1); r2←Mergesort(d2);
   r←Intercl(r1,r2);
   retorne-saída(r);
   fim-senão
fim-se
fim-Função
```

Função: Div_Conq_2 (d:D)→R

$$\text{Div\_Conq\_2}(d) = \begin{cases} \text{drt}(d) & \text{se smpl}(d) \\ \text{combn\_2}(\text{Div\_Conq\_2}(\text{part}_1(d)), \text{Div\_Conq\_2}(\text{part}_2(d))) & \text{caso contrário} \end{cases}$$

Mergesort(d) := **Div_Conq**_2 (d)

smpl(d) := compr(d) ≤ 1 {teste smpl} ;

drt (d) := d {operação drt} ;

part1(d) := Prim(d) {primeira parte} ;

part2(d) := Fin(d) {segunda parte} ;

cmbn_2(r1 , r2) := Intercl(r1 , r2) {combinação}

Complexidade de Algoritmos

O algoritmo Somatório pode ser visto como uma especialização da formulação recursiva da divisão e conquista

somatorio(L):= **se** simples(L)

então retorna (L)

senão retorna (soma(somatorio(sublist1(L)), somatorio(sublist2(L)))).

Div_Conq_2 (d) :=Somatório(L)

smpl(d) := simples(L)

drt (d) := retorna(L)

part1(d) := sublist1(L)

part2(d) := sublist2(L)

cmbn_2(r1 , r2) := soma(r1 , r2)

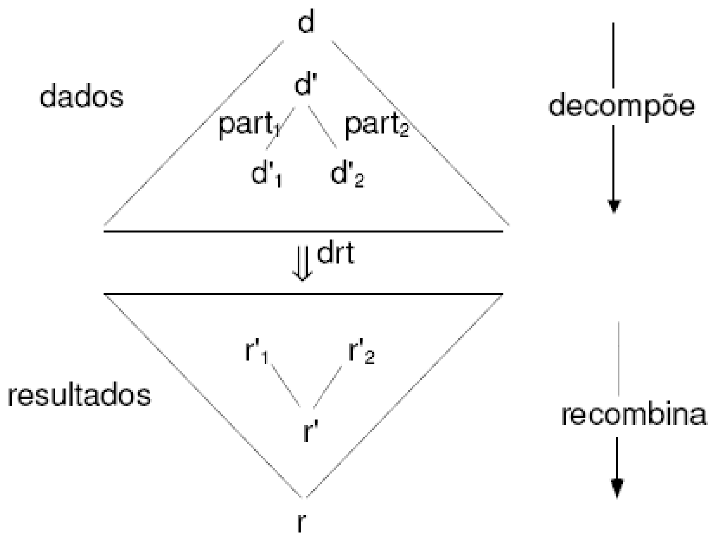
Complexidade de Algoritmos

A recursão na divisão e conquista pode ser visualizada, em três estágios :

1. Construção da árvore de dados: é construída da raiz em direção à folhas (por decomposições repetidas das instâncias de dados) até que as folhas sejam simples.

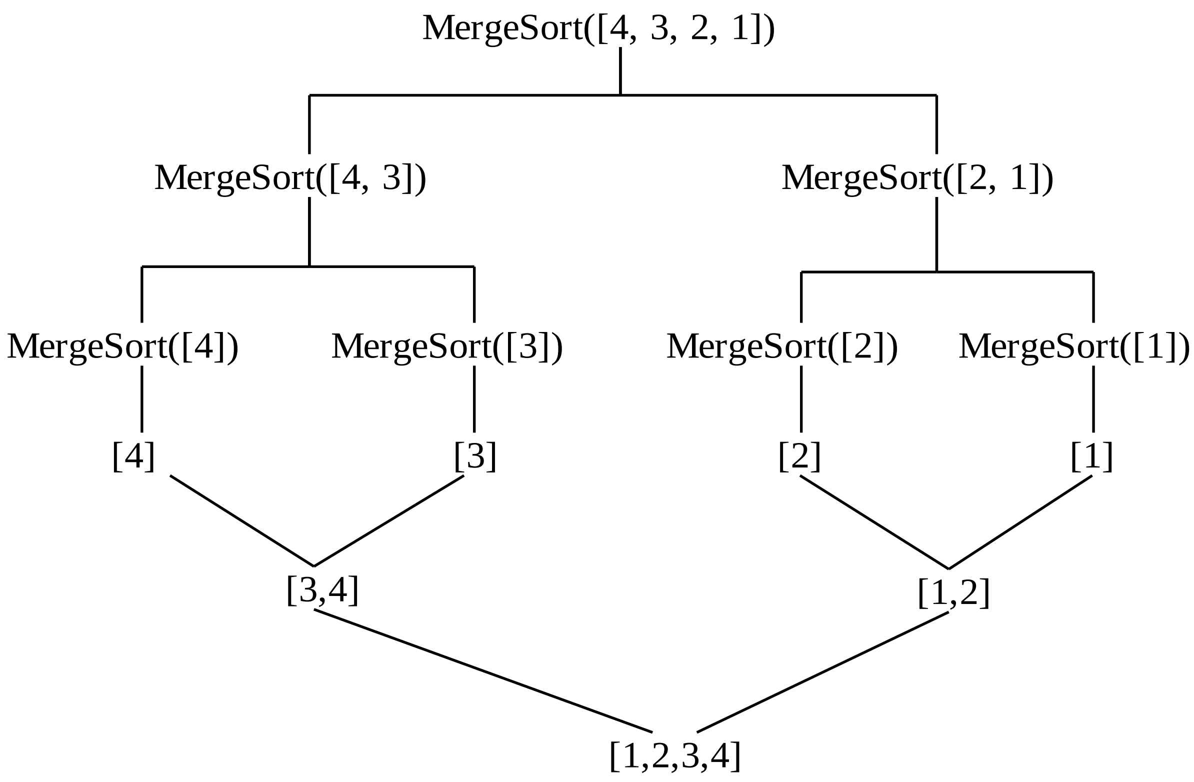
2. Aplicação da função drt para transferir as folhas para o estágio de resultados.

3. Construção da árvore de resultados: é construída das folhas em direção à raiz.



Complexidade de Algoritmos

A execução do algoritmo Mergesort sobre a entrada d := [4, 3, 2, 1]



Complexidade de Algoritmos

A formulação binária pode ser generalizada para uma versão m-ária.

Função: Div_Conq_m $(d:D) \rightarrow R$

$$\text{Div\_Conq\_m}(d) = \begin{cases} \text{drt}(d) & \text{se smpl}(d) \\ \text{combn\_m}(\text{Div\_Conq\_m}(\text{part}_1(d)), \ldots, \text{Div\_Conq\_m}(\text{part}_m(d))) & \text{caso contrário} \end{cases}$$

Um exemplo imediato de divisão e conquista ternária (m = 3) pode ser obtido, modificando o algoritmo Quicksort.

Essa nova versão divide a entrada em três partes a partir do pivô:

- a primeira com os elementos menores do que o pivô,
- a segunda com os elementos iguais ao pivô, e
- a terceira com os elementos maiores do que o pivô.

A combinação é feita através do processo de concatenação.

Complexidade de Algoritmos

Podemos ter o caso de divisão e conquista unária (m = 1).

Função: Div_Conq_1 (d:D)$\rightarrow$R {Divisão e Conquista unária}

$$\text{Div\_Conq\_1}(d) = \begin{cases} \text{drt}(d) & \text{se smpl}(d) \\ \text{combn\_1}(\text{Div\_Conq\_1}(\text{part}_1(d))) & \text{caso contrário} \end{cases}$$

Neste caso , o termo divisão é substituido por redução

Exemplos: algoritmos de busca

- Na busca seqüencial, a pesquisa é direcionada a uma tabela com um elemento a menos, caso o elemento procurado ainda não tenha sido encontrado.

- Na busca binária, a pesquisa é direcionada a uma das metades da tabela, dependendo da comparação com o elemento procurado.

Complexidade de Algoritmos

Projeto de **Algoritmos por Divisão e Conquista**

Considere a versão binária

Função: Div_Conq_2 (d:D)→R {Divisão e Conquista binária}
{Sortes D (entrada); R (resultados)}com a seguinte formulação recursiva.

$$\text{Div\_Conq\_2}(d) = \begin{cases} \text{drt}(d) & \text{se smpl}(d) \\ \text{combn\_2}(\text{Div\_Conq\_2}(\text{part}_1(d)), \text{Div\_Conq\_2}(\text{part}_2(d))) & \text{caso contrário} \end{cases}$$

Podemos ajustar esta função para gerar diferentes algoritmos de classificação.

Considere que smpl e a função unária drt correspondam, respectivamente, a

$$\text{Smpl}(d) = \text{compr}(d) \leq 1$$

$$\text{Drt}(d) := d$$

Complexidade de Algoritmos

As decomposições e recombinações podem ser definidas da seguinte maneira

Versão 1

$Part_1$ e $Part_2$: a primeira e a segunda metades da entrada d e

$Cmbn\_2(r_1, r_2)$: a intercalação de r_1 e r_2 .

Dá origem ao algoritmo Mergesort

Versão 2

$Part_1$ pode conter o menor valor da entrada d e $Part_2$ pode ser a entrada d restante.

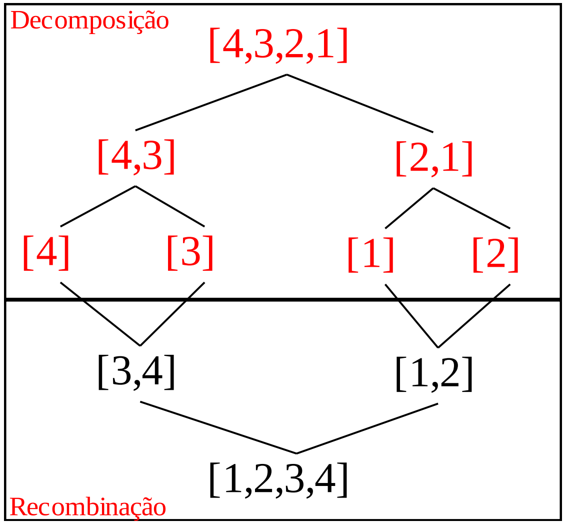
$Cmbn\_2(r_1, r_2)$ pode ser a concatenação de r_1 seguida de r_2 .

Dá origem ao algoritmo de Seleção

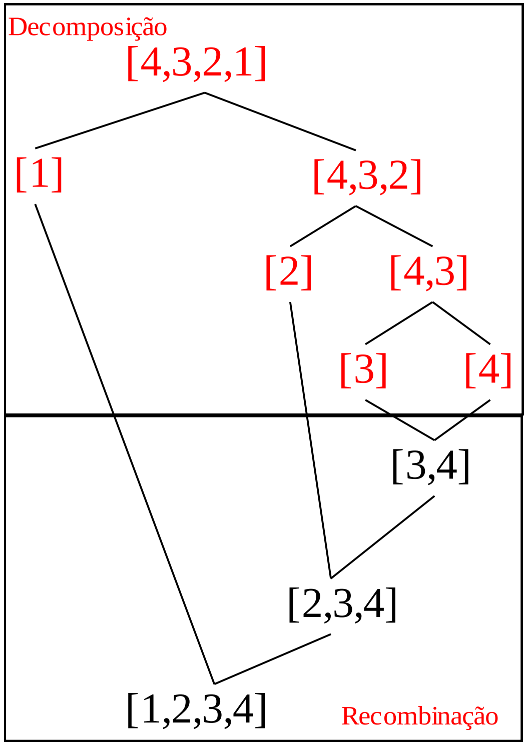
Complexidade de Algoritmos

Quais são as árvores de execução para a entrada [4,3,2,1]?

Algoritmo Mergesort



Algoritmo Seleção



Complexidade de Algoritmos

Quais são as árvores de execução para a entrada [2,1,0,3,5,4]?

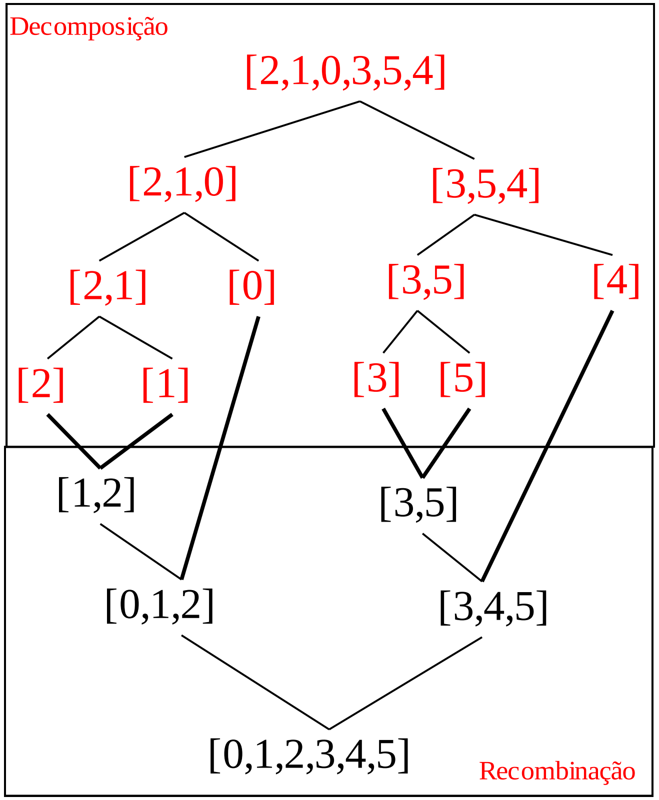
Algoritmo Mergesort

Algoritmo Seleção

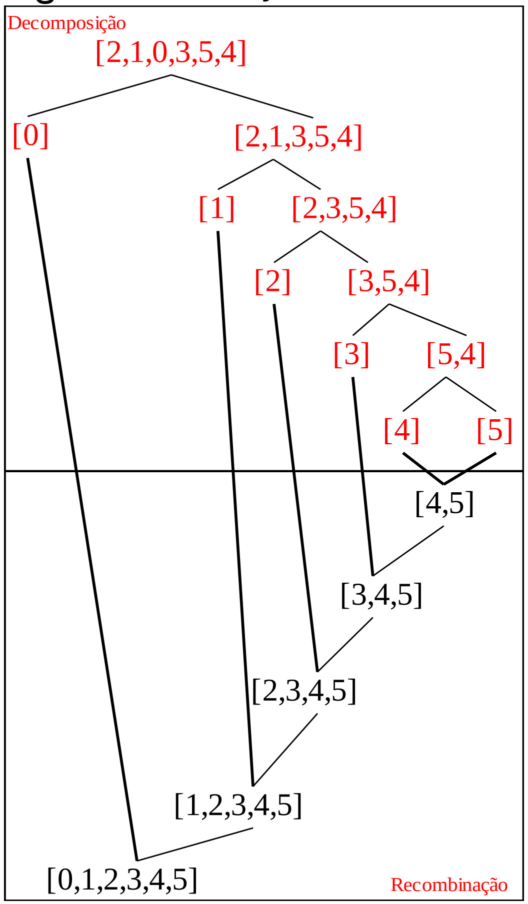
Complexidade de Algoritmos

Quais são as árvores de execução para a entrada [2,1,0,3,5,4]?

Algoritmo Mergesort



Algoritmo Seleção



Complexidade de Algoritmos

O que podemos concluir ?

Quanto mais balanceado for o particionamento do(s) dado(s) de entrada menores serão as árvores de execução!

De acordo com o princípio da Equipartição, o desempenho do algoritmo é dependente do balanceamento do particionamento.

Ele tende a melhorar a medida que o particionamento se torna equilibrado.

Divisão e Conquista

- Algoritmos de divisão e conquista envolve a resolução recursiva de subproblemas
- O tempo de execução dos algoritmos recursivos pode ser descrito por uma recorrência.
  - Equação ou desigualdade que descreve uma função em termos de seu valor em entradas menores

Recursão natural

- Seja $d(n)$ o tempo para a divisão.
- Seja $s(n)$ o tempo para computar a solução final.
- Podemos somar: $f(n) = d(n) + s(n)$ e obtemos

$$T(n) = \begin{cases} \Theta(1) & \text{para } n < n_0 \\ \sum_{1 \le i \le k} T(n_i) + f(n) & \text{caso contrário.} \end{cases}$$

Recursão natural: caso balanceado

$$T(n) = \begin{cases} \Theta(1) & \text{para } n < n_0 \\ kT(\lceil n/m \rceil) + f(n) & \text{caso contrário.} \end{cases}$$

Divisão e Conquista

MERGESORT

Entrada Índices p, r e um vetor A com elementos $A_p, \ldots, A_r$

Saída A com elementos em ordem não-decrescente, i.e. para $i < j$ temos $A_i \leq A_j$.

```
if p < r then
    q = ⌊(p+r)/2⌋
    MergeSort(A,p,q);
    MergeSort(A,q+1,r);
    Merge(A,p,q,r)
end if
```

Recorrências simplificadas

Formalmente, a equação de recorrência do Mergesort é

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

Em vez de

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

Para simplificar a equação, sem afetar a análise de complexidade correspondente, em geral:

- supõe-se argumentos inteiros para funções (omitindo pisos e tetos)
- omite-se a condição limite da recorrência

Divisão e Conquista

Recorrências: caso do Mergesort

A equação de recorrência do Mergesort é

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

Sendo que:

- T(n) representa o tempo da chamada recursiva da função para um problema de tamanho n.
- $2T(\frac{n}{2})$ indica que, a cada iteração, duas chamadas recursivas (2T) serão executadas para entradas de tamanho $\frac{n}{2}$.
- Os resultados das duas chamadas recursivas serão combinados (*merged*) com um algoritmo com complexidade de pior caso $\Theta(n)$.

Divisão e Conquista

- Métodos para resolver recorrências:
  - Método da árvore de recursão
  - Método da substituição
  - Método mestre

Recorrências - Método da árvore de recursão

- Bem intuitiva para a análise de complexidade
  - Numa árvore de recursão cada nó representa o custo de um único subproblema da respectiva chamada recursiva
  - Somam-se os custos de todos os nós de um mesmo nível, para obter o custo daquele nível
  - Somam-se os custos de todos os níveis para obter o custo da árvore

Recorrências - Método da árvore de recursão

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

- Em que nível da árvore o tamanho do problema é 1?
- Quantos níveis tem a árvore?
- Quantos nós têm cada nível?
- Qual o tamanho do problema em cada nível?
- Qual o custo de cada nível i da árvore?
- Quantos nós tem o último nível?
- Qual o custo da árvore?

Recorrências - Método da árvore de recursão

nível					nós
0			$\Theta(n^2)$		3^0
1		$\Theta((n/4)^2)$	$\Theta((n/4)^2)$	$\Theta((n/4)^2)$	3^1
2	$\Theta((n/4^2)^2)$	$\Theta((n/4^2)^2)$	$\Theta((n/4^2)^2)$		3^2
			$\cdots$		
$h = \lceil \log_4 n \rceil$		$\cdots$	$\Theta((n/4^h)^2)$	$\cdots$	3^h

Recorrências - Método da árvore de recursão

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

- Em que nível da árvore o tamanho do problema é 1? No nível $i = \log_4 n = \frac{\log_2 n}{2}$.
- Quantos níveis tem a árvore? A árvore tem $\log_4 n + 1$ níveis (0, 1, 2, 3, ..., $\log_4 n$).
- Quantos nós têm cada nível? 3^i.
- Qual o tamanho do problema em cada nível? $\frac{n}{4^i}$.
- Qual o custo de cada nível i da árvore? $3^i c(\frac{n}{4^i})^2$.
- Quantos nós tem o último nível? $\Theta(n^{\log_4 3})$.
- Qual o custo da árvore? $\sum_{i=0}^{\log_4(n)} n^2 \cdot (3/16)^i = O(n^2)$.

Recorrências - Método da árvore de recursão

Dada a recorrência $T(n) = 3T(n/2) + cn$

- Em que nível da árvore o tamanho do problema é 1?

- Quantos níveis tem a árvore?

- Quantos nós têm cada nível?
- Qual o tamanho do problema em cada nível?
- Qual o custo de cada nível i da árvore?
- Quantos nós tem o último nível?
- Qual o custo da árvore?

Recorrências - Método da árvore de recursão

Resumindo o método

1. Desenha a árvore de recursão
2. Determina
   - o número de níveis
   - o número de nós e o custo por nível
   - o número de folhas
3. Soma os custos dos níveis e o custo das folhas
4. (Eventualmente) Verifica por substituição

Recorrências - Método mestre

Para aplicar o método mestre deve ter a recorrência na seguinte forma:

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

onde $a \geq 1$, $b > 1$ e $f(n)$ é uma função assintoticamente positiva. Se a recorrência estiver no formato acima, então $T(n)$ é limitada assintoticamente como:

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

Considerações

- Nos casos 1 e 3 $f(n)$ deve ser polinomialmente menor, resp. maior que $n^{\log_b a}$, ou seja, $f(n)$ difere assintoticamente por um fator n^{ϵ} para um $\epsilon > 0$.
- Os três casos não abrangem todas as possibilidades

Recorrência – Método mestre simplificado

- Algoritmos de divisão e conquista seguem um padrão genérico:
  - Resolvem um problema de tamanho n solucionando recursivamente a subproblemas de tamanho n/b e então combinando essas resposta em tempo $O(n^d)$, para certos a,b,d>0.
  - Seus tempos podem ser capturados pela equação $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$
- Teorema mestre:
  - Se $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ para constantes a>0, b>1 e d≥0, então

$$T(n) = \begin{cases} O(n^d) & se\ d > log_b a \\ O(n^d log n) & se\ d = log_b a \\ O(n^{log_b a}) & se\ d < log_b a \end{cases}$$

  - Esse teorema nos dá o tempo de execução da maioria dos problemas de divisão e conquista