Merge Sort

Prof. Ms. Déverson Rogério Rando

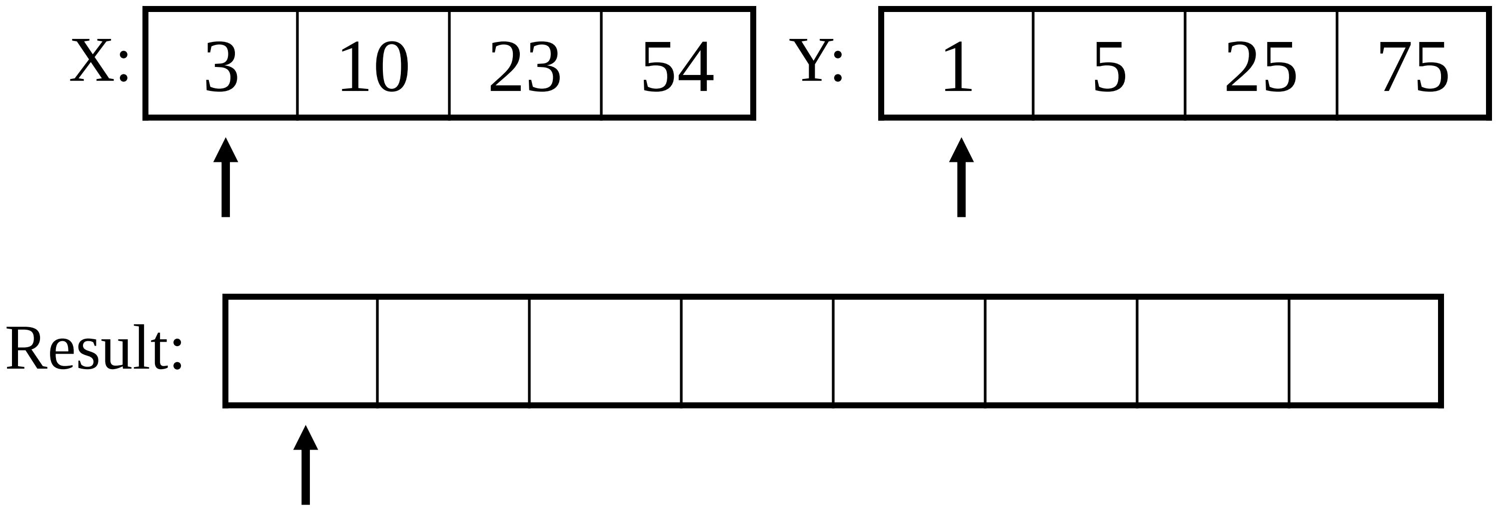
Merge

- A chave para o merge sort é a fusão de duas listas ordenadas em uma, de tal forma que se você tiver duas listas:
  - X ($x_1 \leq x_2 \leq \cdots \leq x_m$) e
  - Y($y_1 \leq y_2 \leq \cdots \leq y_n$)
  - o resultado da lista é Z($z_1 \leq z_2 \leq \cdots \leq z_{m+n}$)
- Exemplo:

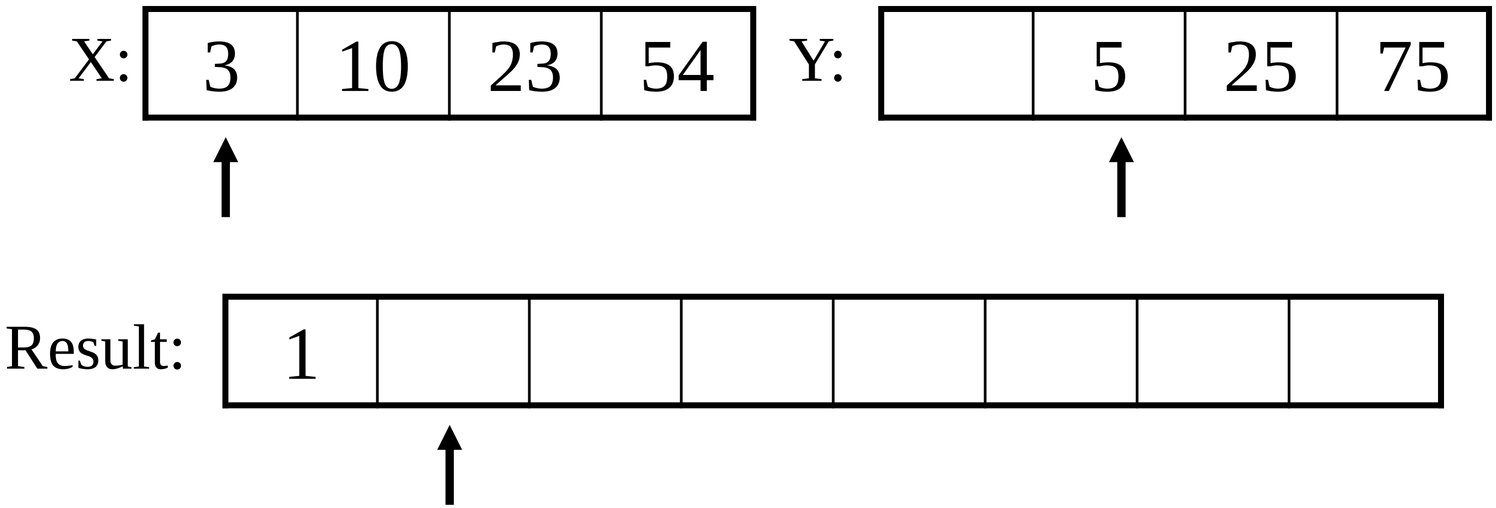
L_1 = { 3 8 9 } L_2 = { 1 5 7 }

merge(L_1, L_2) = { 1 3 5 7 8 9 }

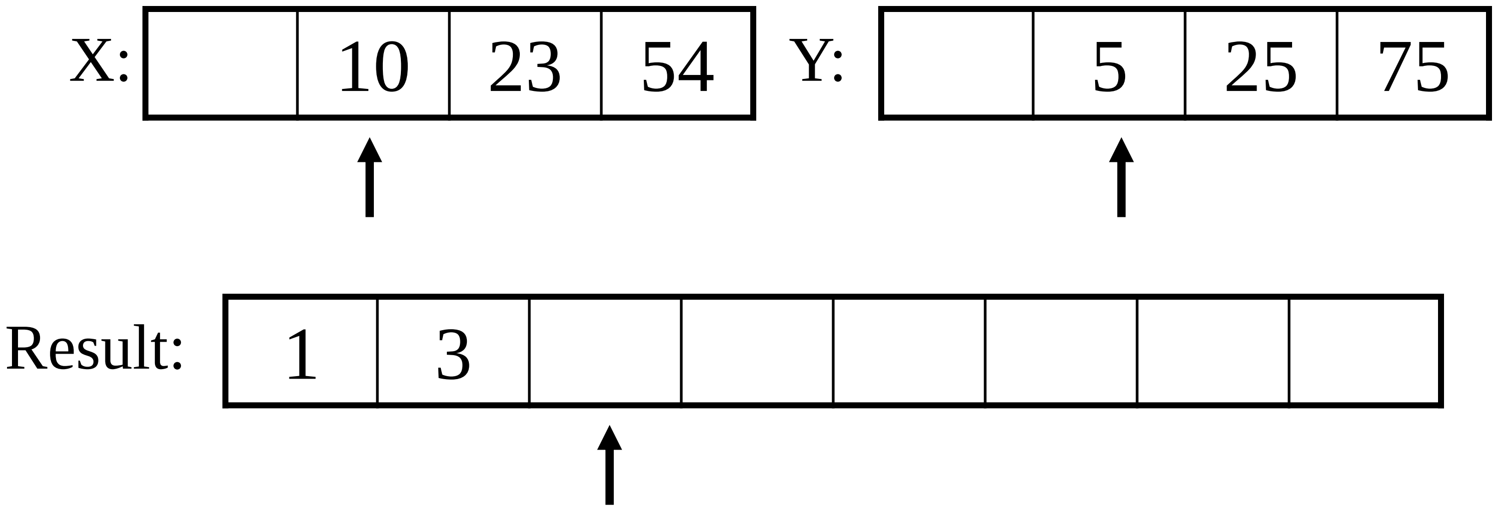
Merging (cont.)



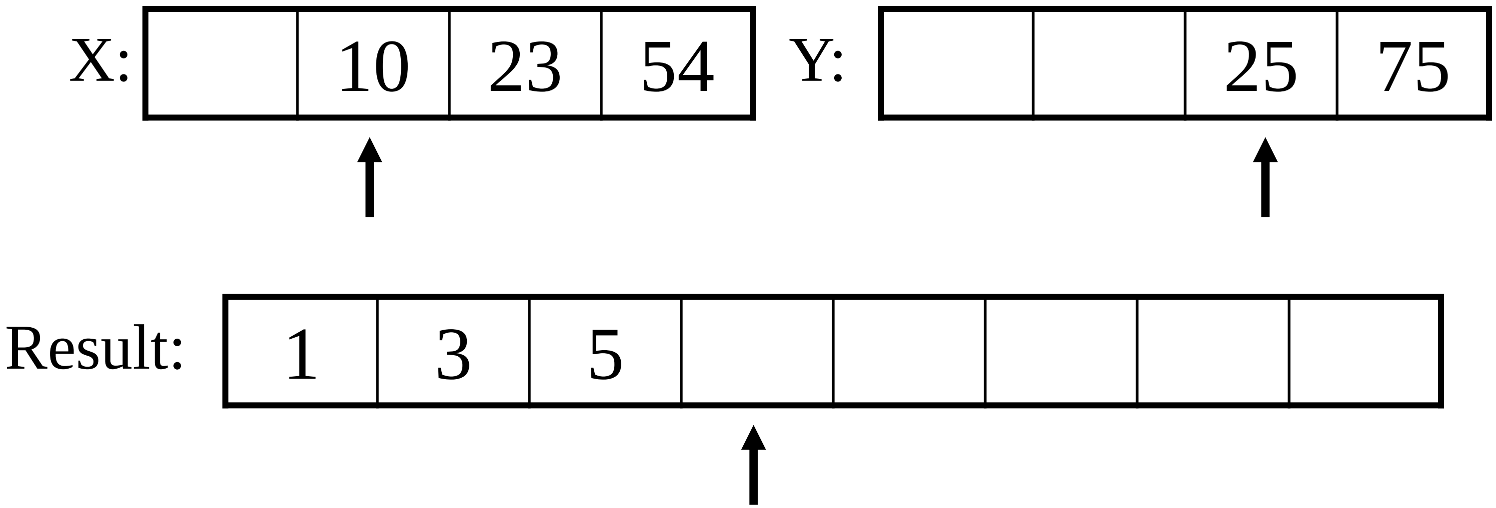
Merging (cont.)



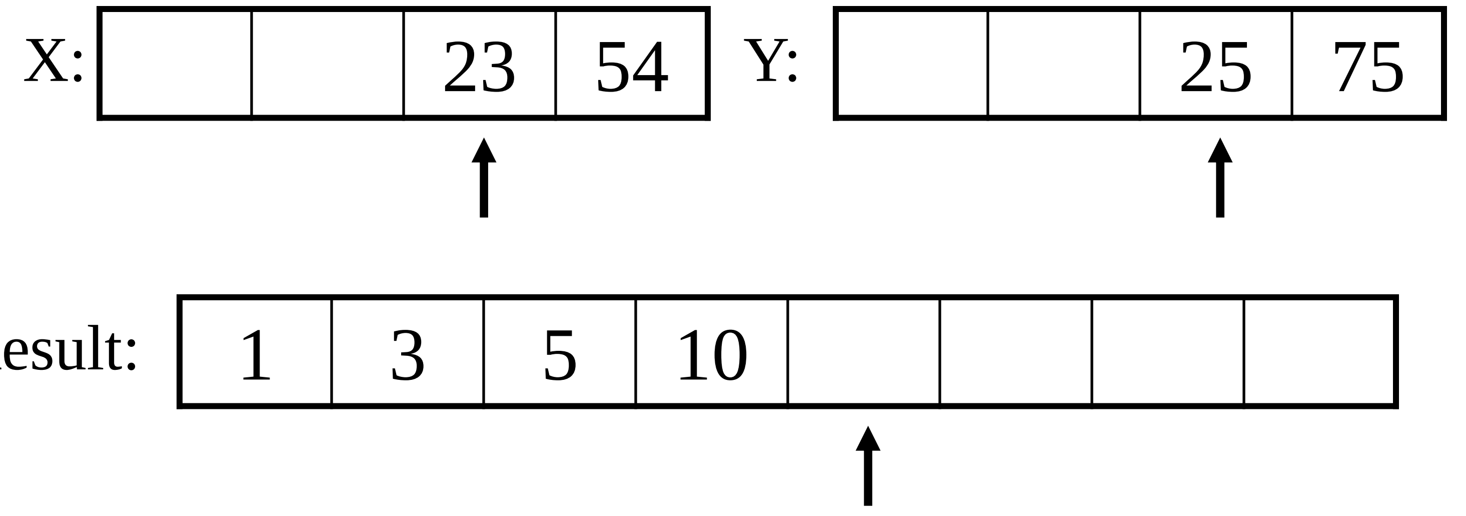
Merging (cont.)



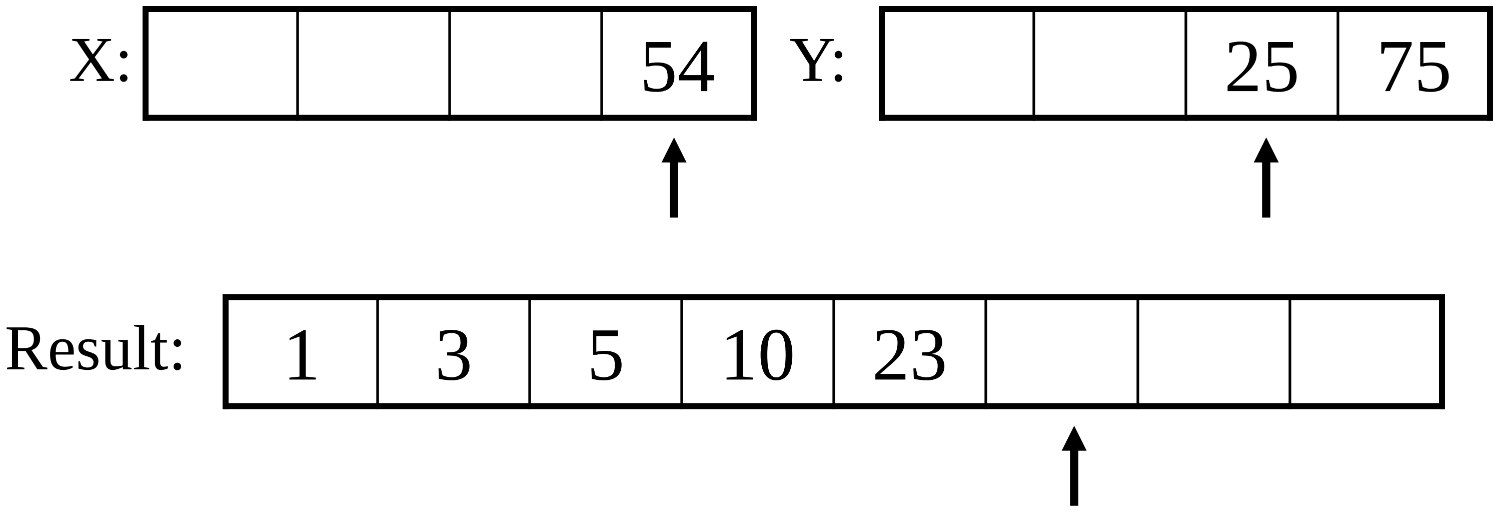
Merging (cont.)



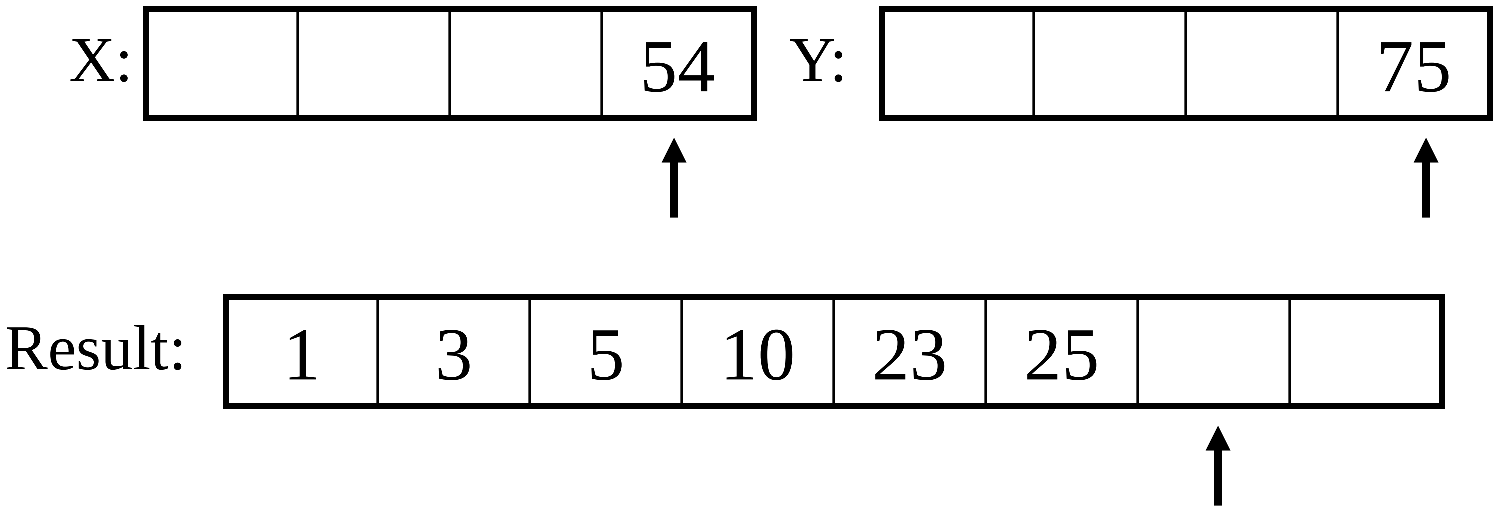
Merging (cont.)



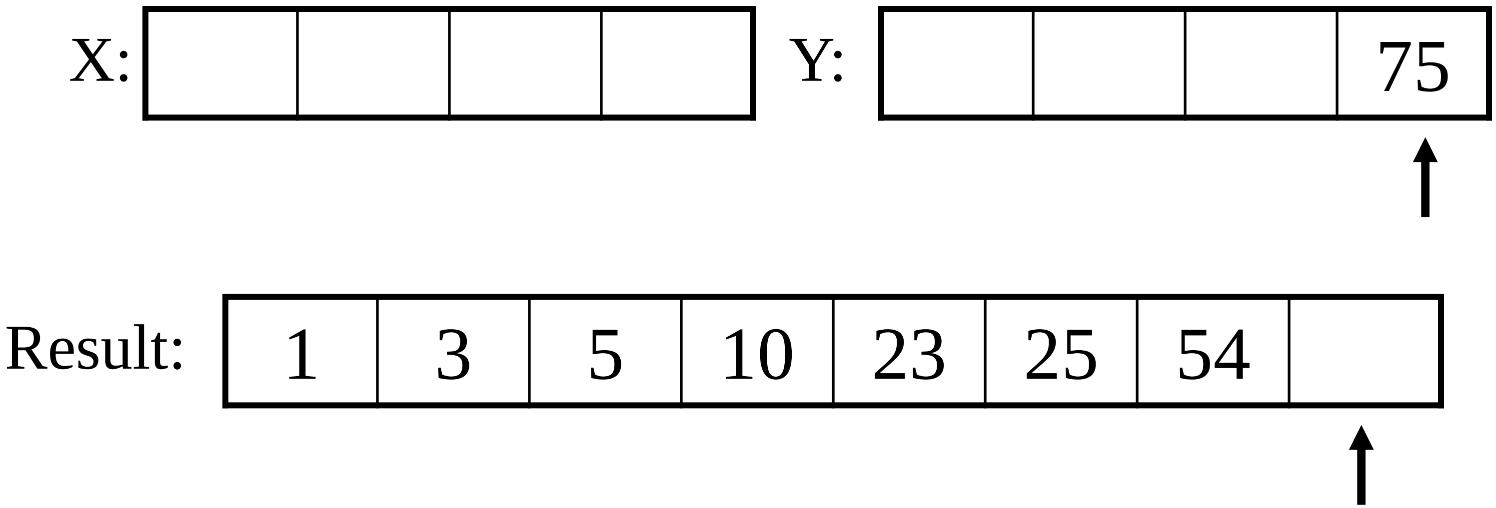
Merging (cont.)



Merging (cont.)



Merging (cont.)



Merging (cont.)

X:

Y:

Result:

1	3	5	10	23	25	54	75

Dividir e Conquistar

- O paradigma de dividir e conquistar envolve três passos em cada nível da recursão:
  - **Dividir** o problema em um determinado número de subproblemas.
  - **Conquistar** os subproblemas, resolvendo-os recursivamente. Porém, se os tamanhos dos subproblemas forem pequenos o bastante, basta resolver os subproblemas de maneira direta.
  - **Combinar** as soluções dadas aos subproblemas, a fim de formar a solução para o problema original.

Dividir e Conquistar

- O algoritmo de ordenação por intercalação a seguir obedece ao paradigma de dividir e conquistar. Intuitivamente, ele opera do modo ilustrado a seguir.
  - **Dividir:** Divide a sequência de ***n*** elementos a serem ordenados em duas subsequências de ***n/2*** elementos cada uma.
  - **Conquistar:** Classifica as duas subsequências recursivamente, utilizando a ordenação por intercalação.
  - **Combinar: Faz** a intercalação das duas sequências ordenadas, de modo a produzir a resposta ordenada.
- A recursão "não funciona" quando a sequência a ser ordenada tem comprimento 1, pois nesse caso não há nenhum trabalho a ser feito, tendo em vista que toda sequência de comprimento 1 já está ordenada

Merge Sort Algoritmo

Dado uma lista L com um comprimento k:

- If k == 1 → a lista está ordenada
- Senão:
  – Merge Sort o lado direito (1 até k/2)
  – Merge Sort o lado esquerdo (k/2+1 até k)
  – Intercala o lado direito com o lado esquerdo.

Merge Sort Exemplo

99	6	86	15	58	35	86	4	0

Merge Sort Exemplo

99	6	86	15	58	35	86	4	0

99	6	86	15

58	35	86	4	0

Merge Sort Exemplo

99	6	86	15	58	35	86	4	0

99	6	86	15

58	35	86	4	0

99	6

86	15

58	35

86	4	0

Merge Sort Exemplo

99	6	86	15	58	35	86	4	0

99	6	86	15

58	35	86	4	0

99	6

86	15

58	35

86	4	0

99

6

86

15

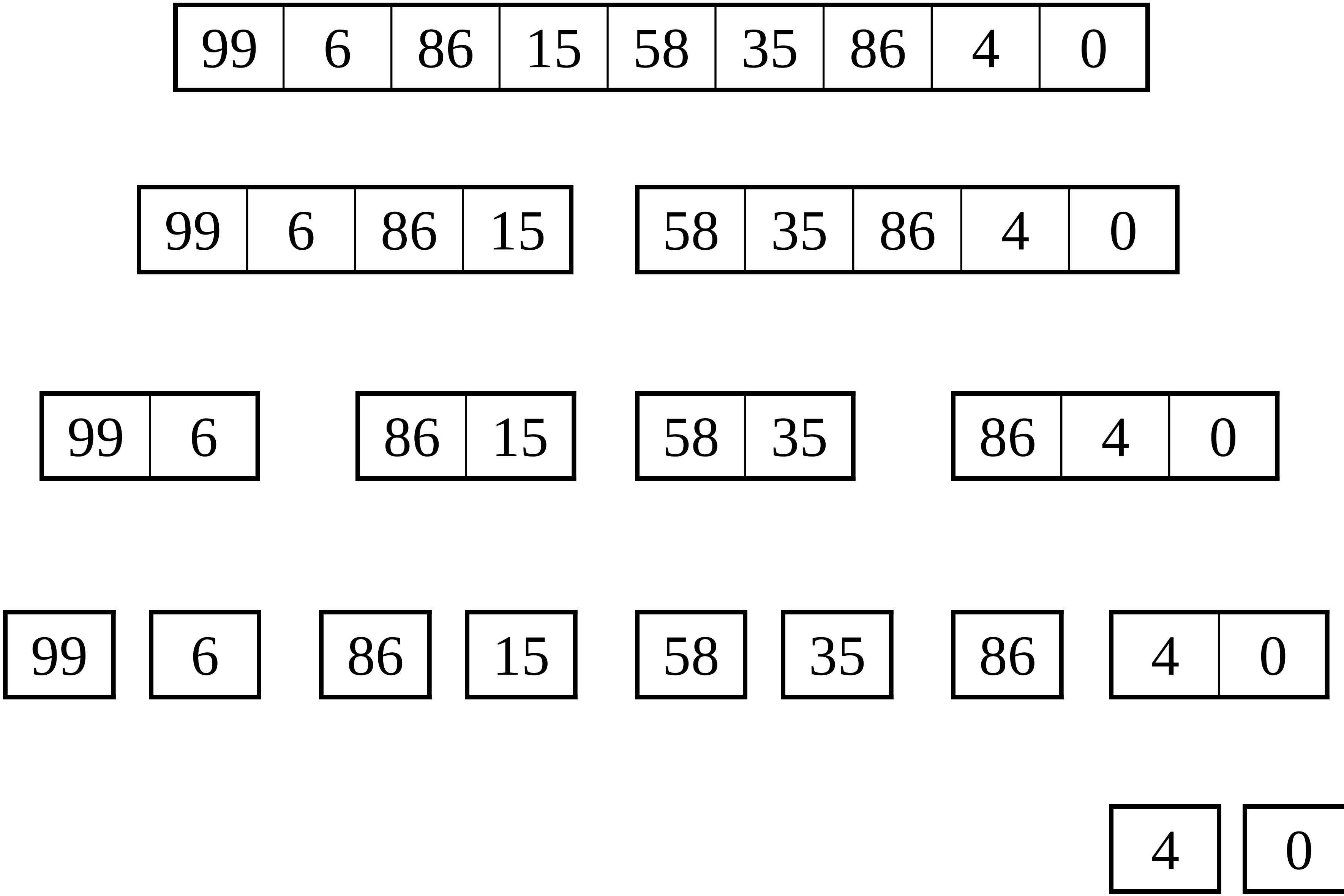
58

35

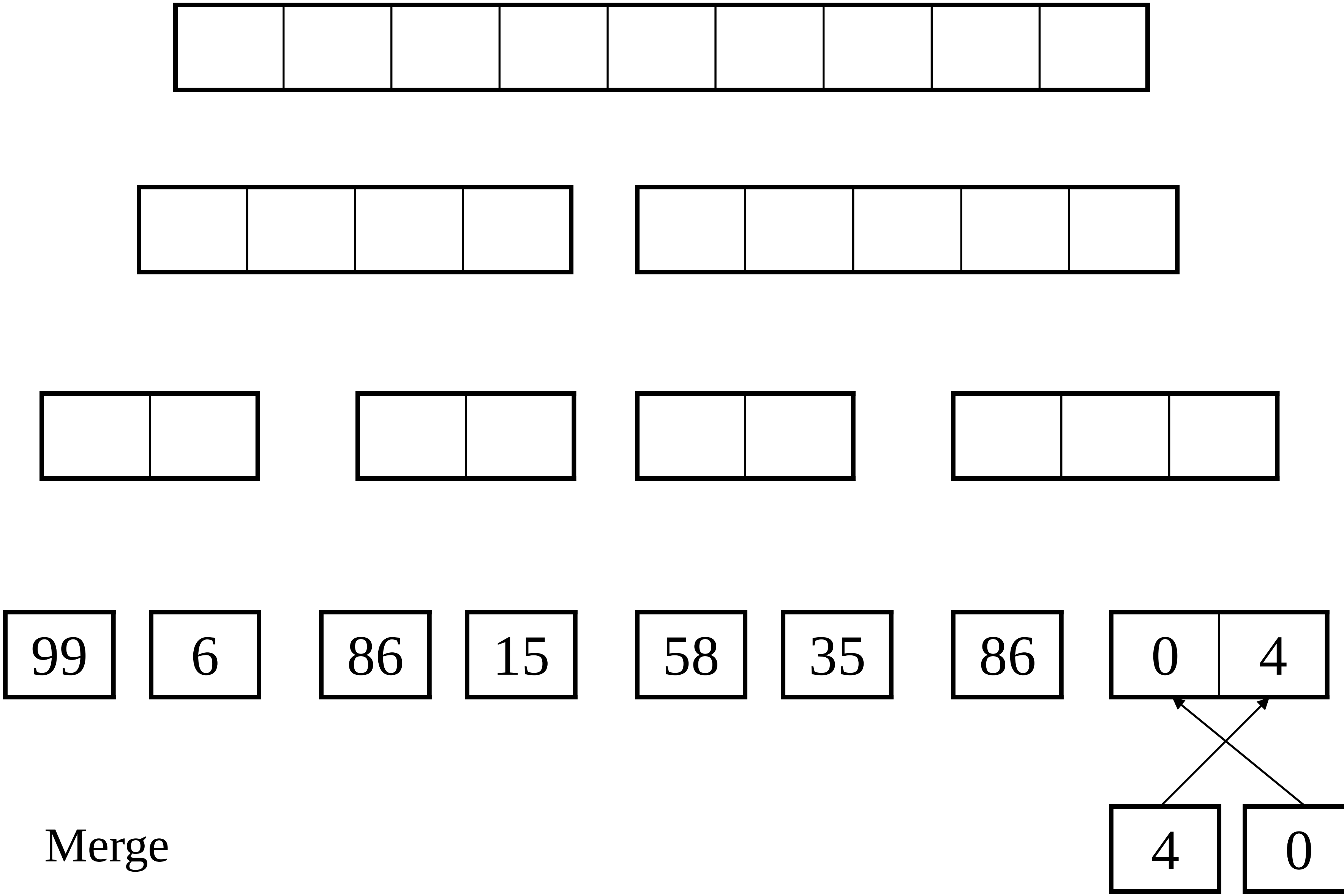
86

4	0

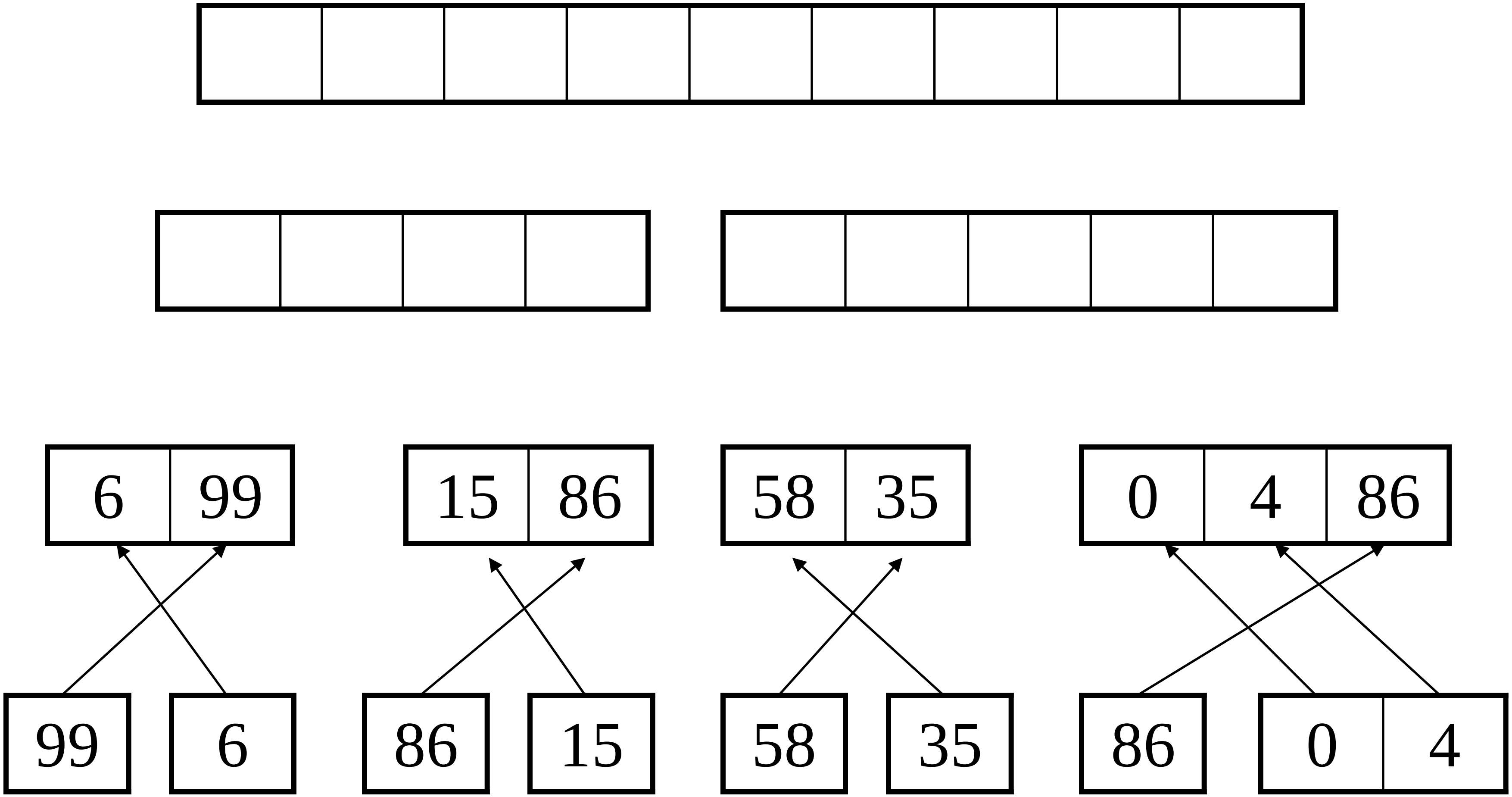
Merge Sort Exemplo



Merge Sort Exemplo

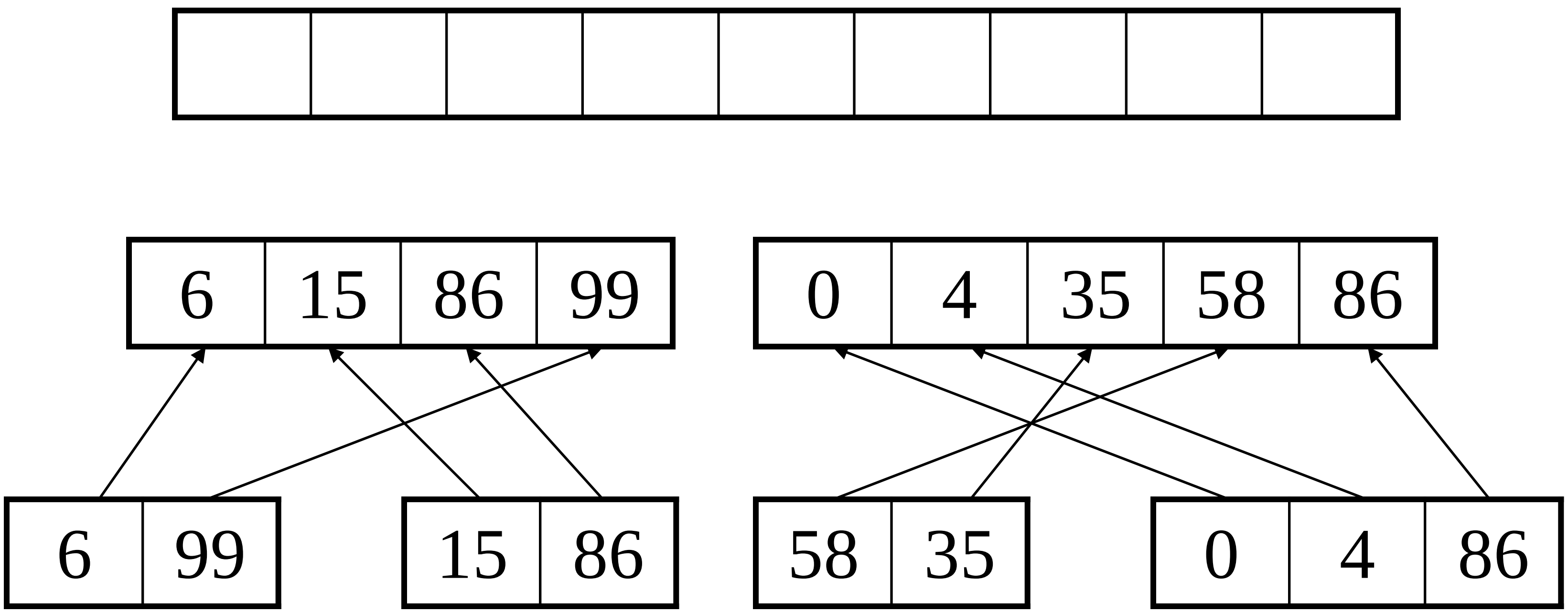


Merge Sort Exemplo



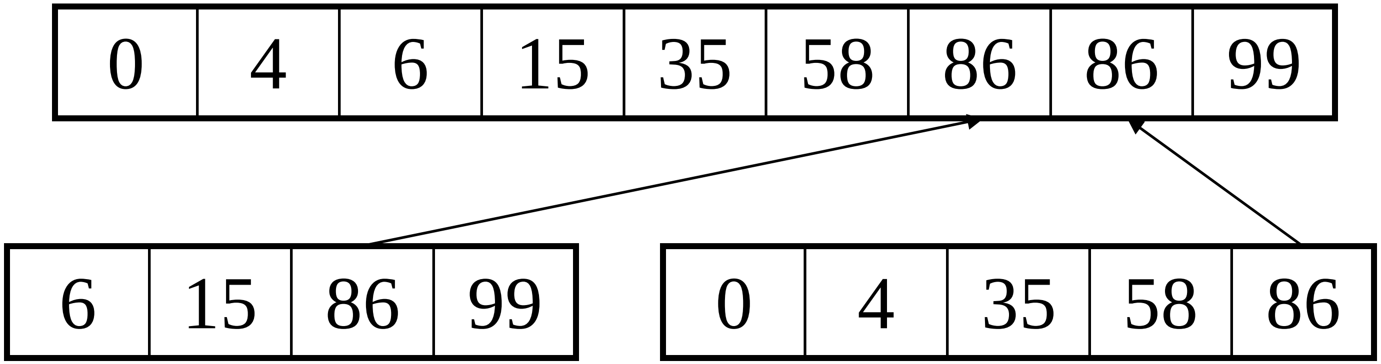
Merge

Merge Sort Exemplo



Merge

Merge Sort Exemplo



Merge

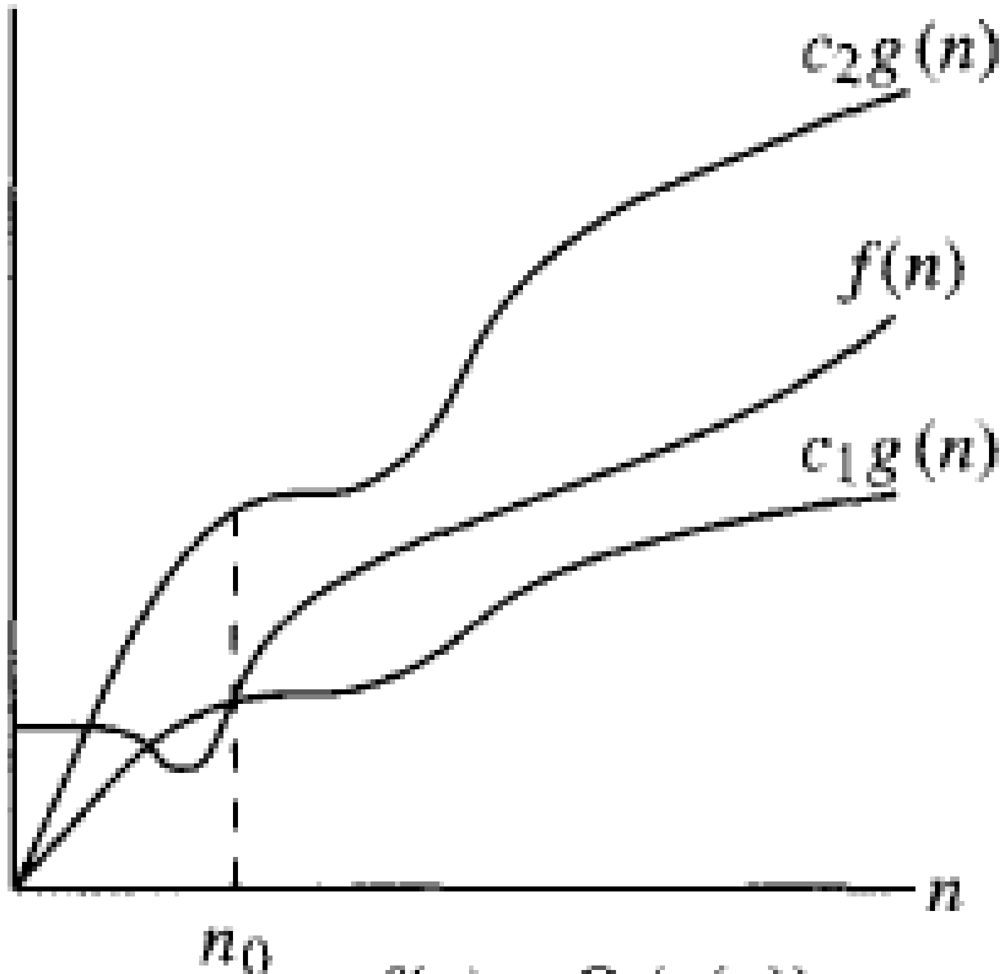
Merge Sort Exemplo

0	4	6	15	35	58	86	86	99

Crescimento de Funções

Notação Θ

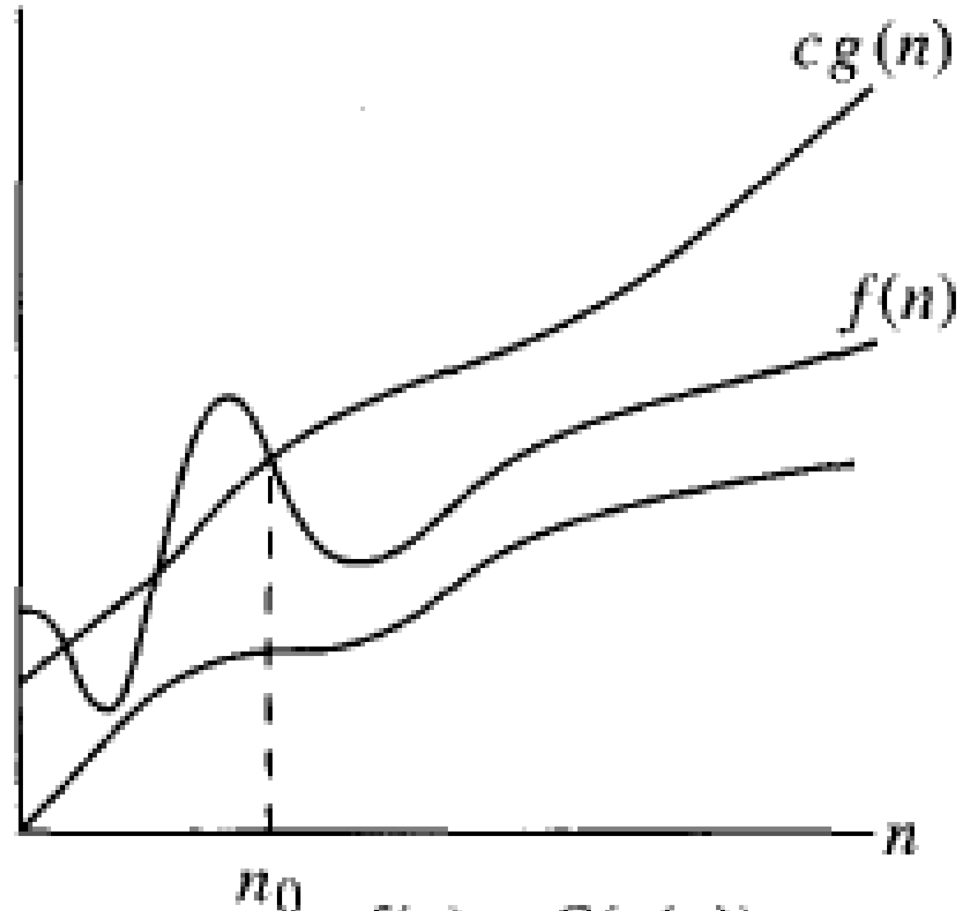
- A notação Θ limita assintoticamente uma função acima e abaixo.
- $\Theta(g(n)) = \{f(n)$: existem constantes positivas c_1, c_2, e n_0 tais que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para todo $n \geq n_0\}$.
- Uma função f(n) pertence ao conjunto $\Theta(g(n))$ se existem constantes positivas c_1, c_2 tais que ela possa ser "imprensada" entre $c_1 g(n)$ e $c_2 g(n)$ para um valor de n suficientemente grande.



$f(n) = \Theta(g(n))$

Notação *O*

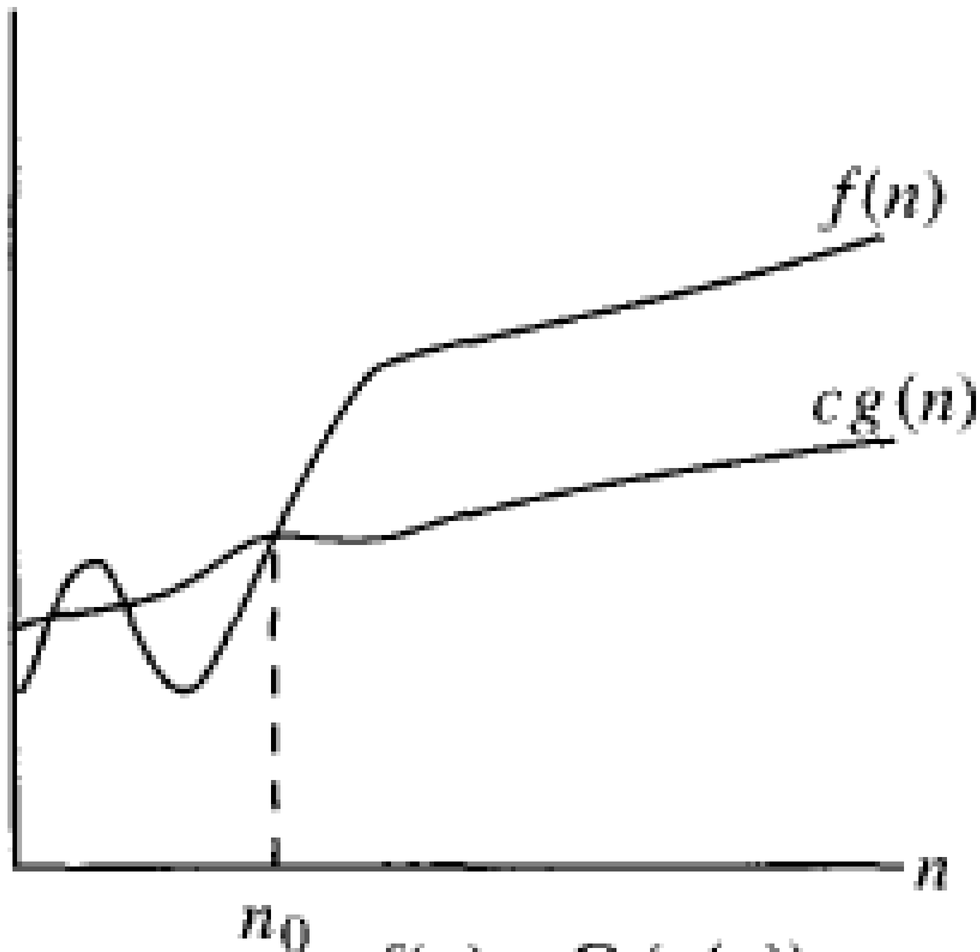
- Quando temos apenas um limite assintótico superior, usamos a notação 0.
- $O(g(n)) = \{f(n)$: existem constantes positivas c, e n_0 tais que $0 \leq f(n) \leq cg(n)$ para todo $n \geq n_0\}$.



$f(n) = O(g(n))$

Notação Ω

- Da mesma maneira que a notação O fornece um limite assintótico superior sobre uma função, a notação Ω fornece um ***limite assintótico inferior.***
- $\Omega(g(n)) = \{f(n)$: existem constantes positivas c, e n_0 tais que $0 \leq cg(n) \leq f(n)$ para todo $n \geq n_0\}$



$f(n) = \Omega(g(n))$

Recorrência

- Uma ***recorrência*** é uma equação ou desigualdade que descreve uma função em termos de seu valorem entradas menores.
- **Métodos para solução**(Capítulo 4)
  - Método da Substituição
  - Método árvore recursão.
  - Método Mestre
- Relações de recorrência surgem quando analisamos o tempo de execução de algoritmos iterativos ou recursivos.

Método Mestre

- O método mestre fornece um processo de "livro de receitas" para resolver recorrências da forma:
- $T(n) = aT(n/b) + f(n)$
  - $a \geq 1$, $b > 1$ são constantes.
  - $f(n)$ é assintoticamente positiva.
  - n/b
- Necessita da memorização de três casos

Método Mestre

Sejam $a \geq 1$ e $b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definida sobre os inteiros não negativos pela recorrência $T(n) = aT(n/b) + f(n)$, onde interpretamos n/b pelo $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. $T(n)$ pode ser limitado assintoticamente como a seguir:

1. Se $f(n) = O(n^{\log_b a - \varepsilon})$ para alguma constante $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. Se $f(n) = \Theta(n^{\log_b a})$, Então $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. Se $f(n) = \Omega(n^{\log_b a + \varepsilon})$ para alguma constante $\varepsilon > 0$, para alguma constante $c < 1$ e para todo n suficientemente grande, então $T(n) = \Theta(f(n))$.

Traduzindo

- Case 1:
- Se f(n) é dominada por $n^{\log_b a}$:
- $T(n) = \Theta(n^{\log_b n})$

- Case 3:
- Se f(n) domina $n^{\log_b a}$:
- $T(n) = \Theta(f(n))$

- Case 2:
- Se $f(n) = \Theta(n^{\log_b a})$:
- $T(n) = \Theta(n^{\log_b a} \log n)$

- Vamos resolver a equação de recorrência:
- $T(n) = 4T(n/2) + n$
- Para saber se o Teorema Mestre pode ser aplicado ou não, temos que ter as constantes $a \geq 1$ e $b > 1$ e a função $f(n)$ assintoticamente positiva.
- Temos $a = 4$, $b = 2$, $f(n) = n$ e as três condições são satisfeitas.

- Deve-se analisar se a equação de recorrência cai em um dos três casos previstos do teorema ou não.
- Para isso, devemos comparar a função f(n) com a função $n^{\log_b a}$, ou seja:
- $f(n) : n^{\log_b a}$
- $n : n^{\log_2 4}$
- $n : n^2$.
- A função $n^{\log_b a}$ domina a função f(n) por um fator polinomial n^1.
- Assim, o caso 1 do Teorema Mestre se aplica e temos que $T(n) = \Theta(n^2)$.

- Vamos resolver a resolver a seguinte equação de recorrência:
- $T(n) = 4T(n/2) + n^2$.
- Para saber se o Teorema Mestre pode ser aplicado ou não, temos que ter as constantes $a \geq 1$ e $b > 1$ e a função $f(n)$ assintoticamente positiva.
- Temos $a = 4$, $b = 2$, $f(n) = n^2$ e as três condições são satisfeitas.
- Deve-se analisar se a equação de recorrência cai em um dos três casos previstos do teorema ou não.

- Para isso, devemos comparar a função f(n) com a função $n^{\log_b a}$, ou seja:
- f(n) : $n^{\log_b a}$
- n 2 : $n^{\log_2 4}$
- $n^2 : n^2$.
- As duas funções f(n) e $n^{\log_b a}$ têm a mesma taxa de crescimento.
- Assim, o caso 2 do Teorema Mestre se aplica e temos que $T(n) = \Theta(n^2 \log n)$.

- Use o Teorema Mestre para resolver a seguinte equação de recorrência:
- $T(n) = 4T(n\ 2\ ) + n^3$.
- Para saber se o Teorema Mestre pode ser aplicado ou não, temos que ter as constantes $a \geq 1$ e $b > 1$ e a função f(n) assintoticamente positiva.
- Temos $a = 4$, $b = 2$, $f(n) = n\ 3$ e as três condições são satisfeitas.
- Deve-se analisar se a equação de recorrência cai em um dos três casos previstos do teorema ou não.

- Para isso, devemos comparar a função f(n) com a função $n^{\log_b a}$, ou seja:
- $f(n) : n^{\log_b a}$
- $n\ 3 : n^{\log_2 4}$
- $n^3 : n^2$.
- A função f(n) domina a função $n^{\log_b a}$ por um fator polinomial n^1 .

- Assim, o caso 3 do Teorema Mestre pode ser aplicado se a condição de “regularidade” for satisfeita, ou seja,
- $af(n/b) \leq cf(n)$ para uma constante $c < 1$.
- $af(n/b) \leq cf(n)$
- $4(n/2)^3 \leq cn^3$
- $4n^3/8 \leq cn^3$
- $1/2n^3 \leq cn^3$
- A inequação é satisfeita para $c = 1/2$.
- Portanto, o caso 3 do Teorema Mestre se aplica e $T(n) = \Theta(n^3)$.

Método Mestre Merge Sort

- O algoritmo recursivo possui três chamadas de função, sendo que as duas primeiras são chamadas recursivas e recebem metade dos elementos do vetor passado, e a outra é chamada para a função que realiza a intercalação das duas metades.
- Os valores necessários para a resolução do método mestre são:
- a=2, b=2, f(n) = n
- A expressão de recorrência do algoritmo Merge Sort é dada por $T(n)=2T(n/2) + n$
- $f(n) = \Theta(n^{\log_b a})$
- $n = \Theta(n^{\log_2 2})$

$n = \Theta(n^1)$

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

Exercícios

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

b) $T(n)=16T(n/4)+n^2$

c) $T(n)=7T(n/3)+n^2$

d) $T(n)=27T(n/3)+n^3$

e) $T(n)=64T(n/4)+n^2$

Exercícios

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

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

$n^3 = \Theta(n^{\log_2 2})$

$n^3 = \Theta(n^1)$

$T(n)= \Theta(n^3)$

Exercícios

b) $T(n)=16T(n/4)+n^2$

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

$n^2 = \Theta(n^{\log_4 16})$

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

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

Exercícios

c) $T(n)=7T(n/3)+n^2$

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

$n^2 = \Theta(n^{\log_3 7})$

$n^2 = \Theta(n^{1,77})$

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

Exercícios

d) $T(n)=27T(n/3)+n^3$

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

$n^3 = \Theta(n^{\log_3 27})$

$n^3 = \Theta(n^3)$

$T(n)= \Theta(n^3 \log n)$

Exercícios

e) $T(n)=64T(n/4)+n^2$

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

$n^2 = \Theta(n^{\log_4 64})$

$n^2 = \Theta(n^3)$

$T(n)= \Theta(n^3)$

Método Árvore de Recursão

- Fazer um bom palpite é por vezes difícil com o método de substituição.
- Use árvores de recursão para elaborar bons palpites.
- Mostrar expansões sucessivas de recorrências usando árvores.
  - Mantem o controle do tempo gasto nas subproblemas de um algoritmo de dividir e conquistar.
  - Ajuda a organizar a contabilidade algébrica necessário para resolver uma recorrência .

Árvore de Recursão – Exemplo

- Verificando o tempo do Merge Sort:

$$T(n) = \Theta(1) \quad \text{if } n = 1$$

$$T(n) = 2T(n/2) + \Theta(n) \quad \text{if } n > 1$$

- Reescrevendo a recorrência como:

$$T(n) = \boldsymbol{c} \quad \text{if } n = 1$$

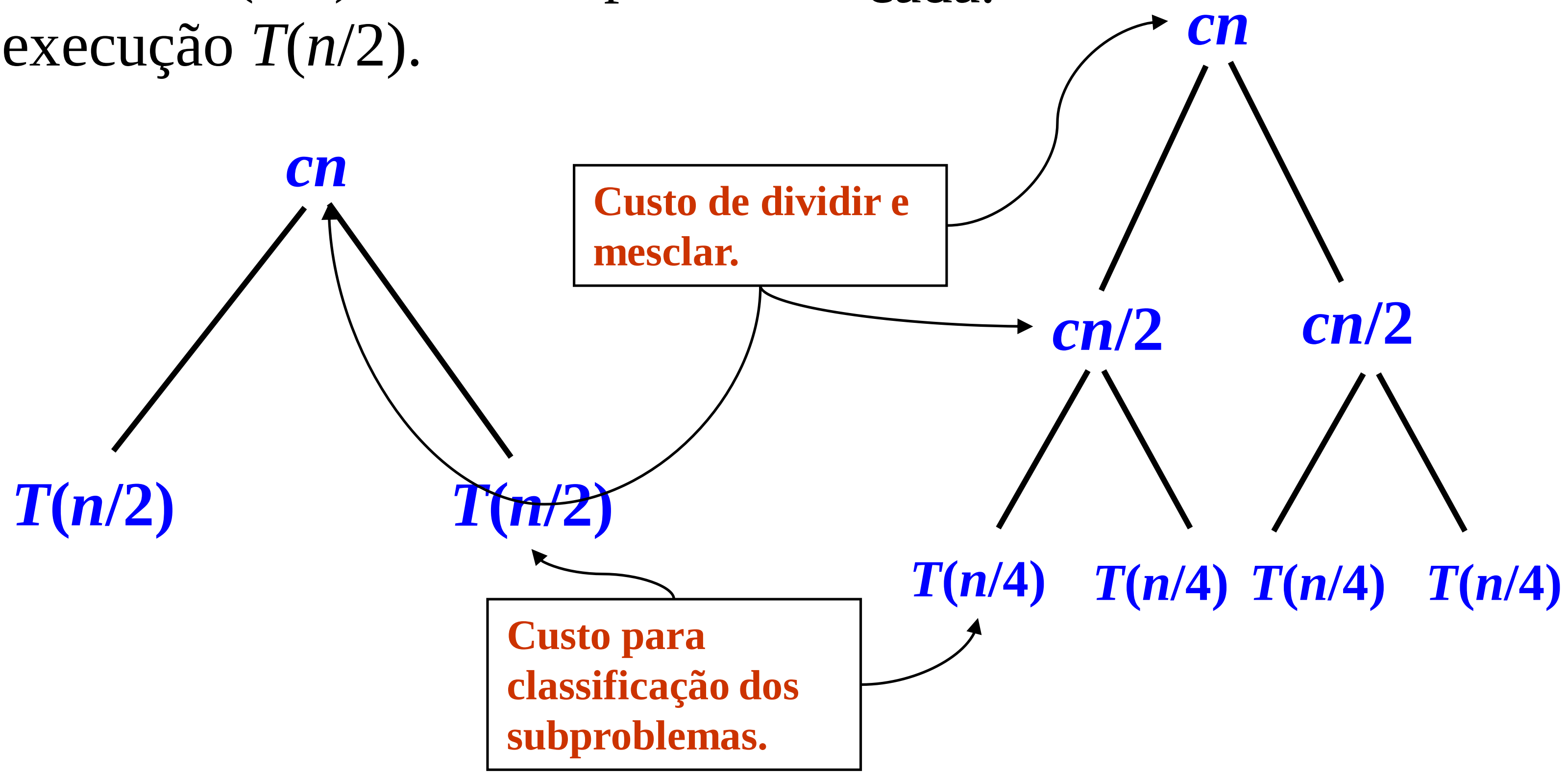
$$T(n) = 2T(n/2) + \boldsymbol{cn} \quad \text{if } n > 1$$

$\boldsymbol{c > 0}$: O tempo no caso base e o tempo para dividir os elementos da matriz e combinar as etapas

Árvore de recursão Merge Sort

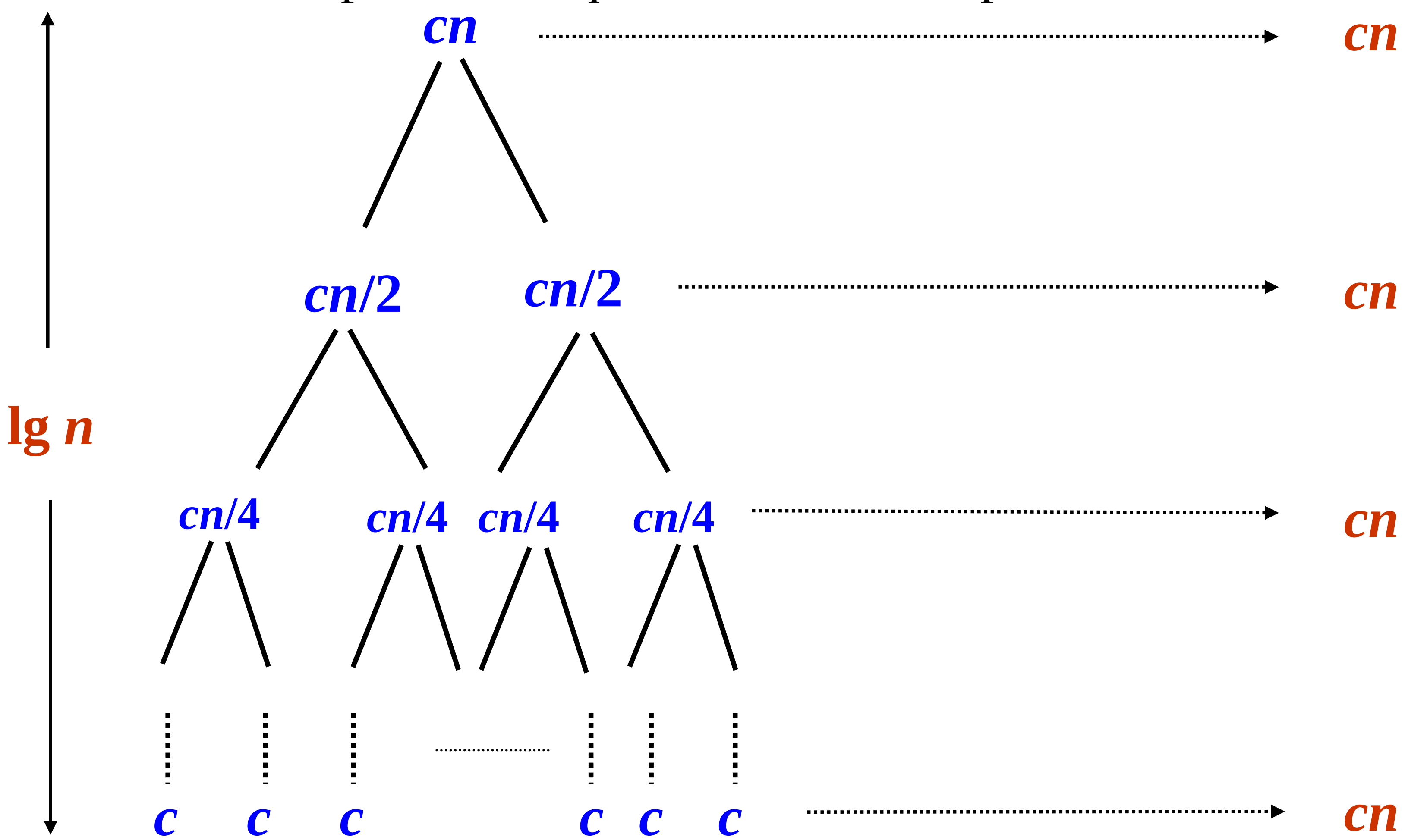
Para o problema original, nós temos um custo *cn*, além de dois subproblemas com tamanho (*n*/2) e um tempo de execução *T*(*n*/2).

Cada problema de tamanho *n*/2 tem um custo *cn*/2 além de dois subproblemas com custo *T*(*n*/4) cada.



Árvore de recursão Merge Sort

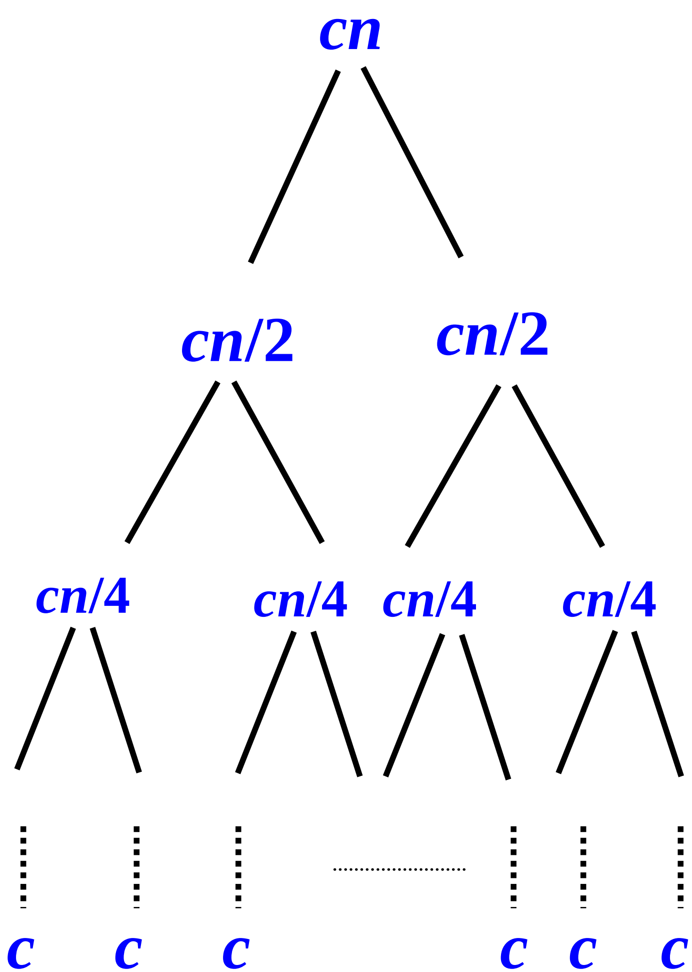
Continuar a expandir até que o tamanho do problema reduza a 1



Total : $cn\lg n + cn$

Árvore de recursão Merge Sort

Continua expandido o problema até 1.



- Cada nível tem um custo total ***cn***.
- Cada vez que vamos descer um nível, o número de subproblemas dobra, mas o custo por subproblema divide $\Rightarrow$ *Custo por nível permance o mesmo*.
- Existe lg n + 1 níveis, a altura é lg n. (Assumindo que n é uma potência de dois.)
- Custo total= soma dos custos em cada nível = $(\lg n + 1)cn = cn\lg n + cn = \Theta(n \lg n)$.

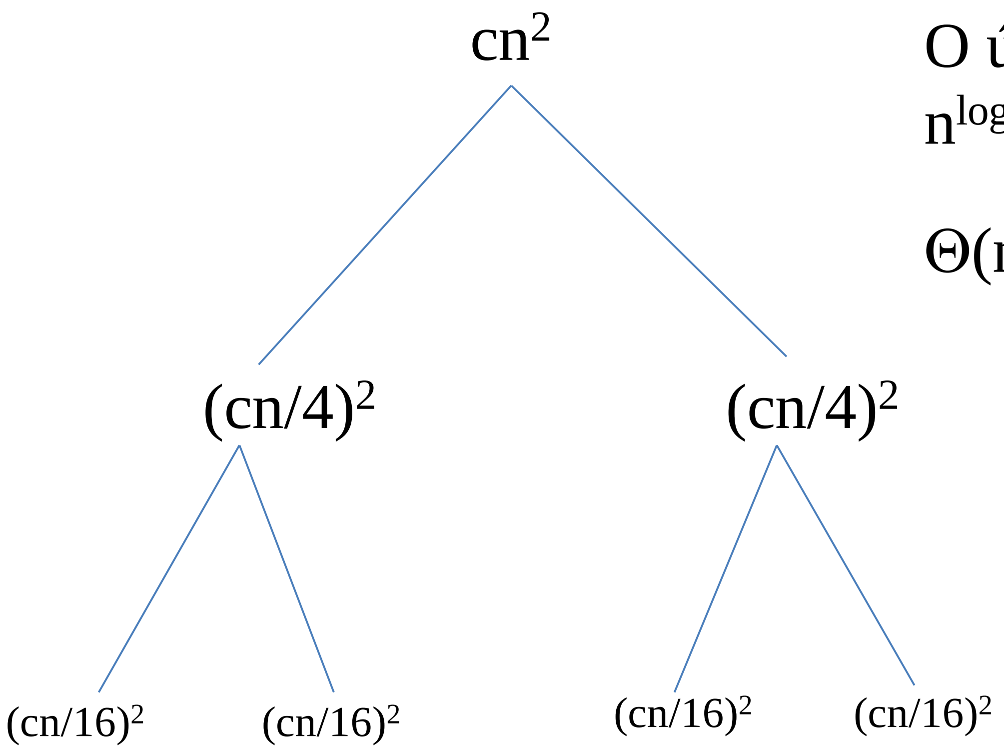
Exercícios

a) $T(n) = 2T(n/4) + \Theta(n^2)$

b) $T(n) = 2T(n/3) + \Theta(n)$

Exercícios

a) $T(n) = 2T(n/4) + \Theta(n^2)$



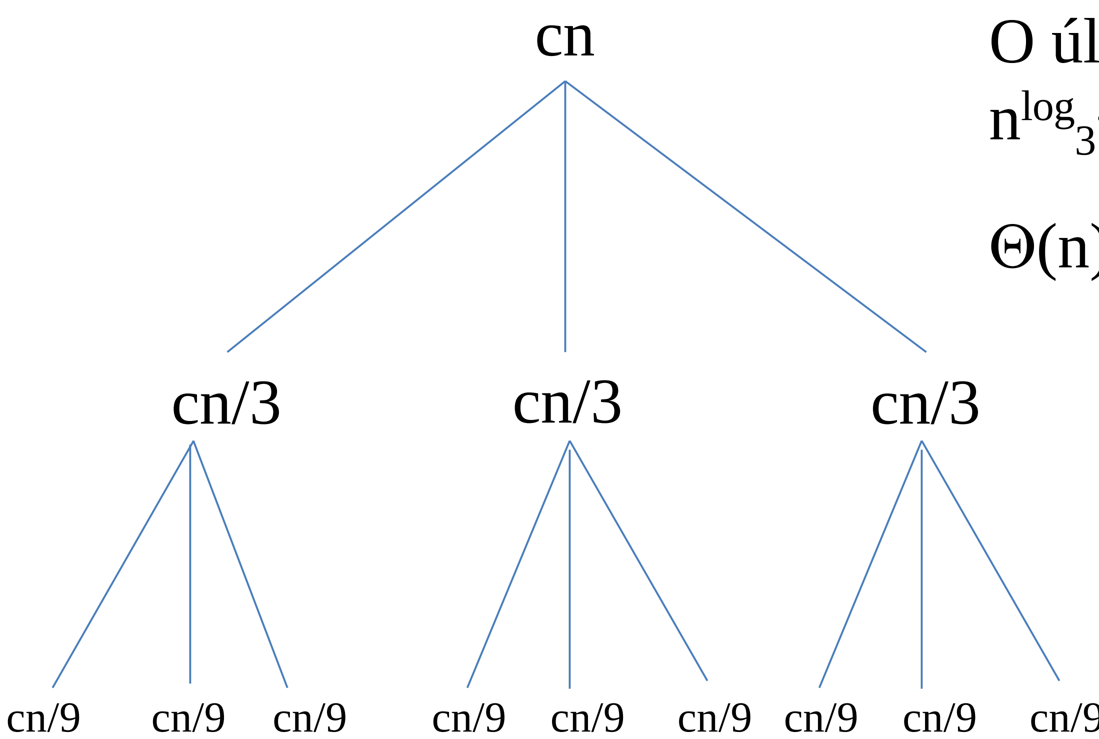
O último nível tem

$n^{\log_4 2} = n^{0,5}$

$\Theta(n^2)$

Exercícios

b) $T(n) = 2T(n/3) + \Theta(n)$



O último nível tem

$n^{\log_3 2} = n^{0,63}$

$\Theta(n)$

Quick sort (Custo)

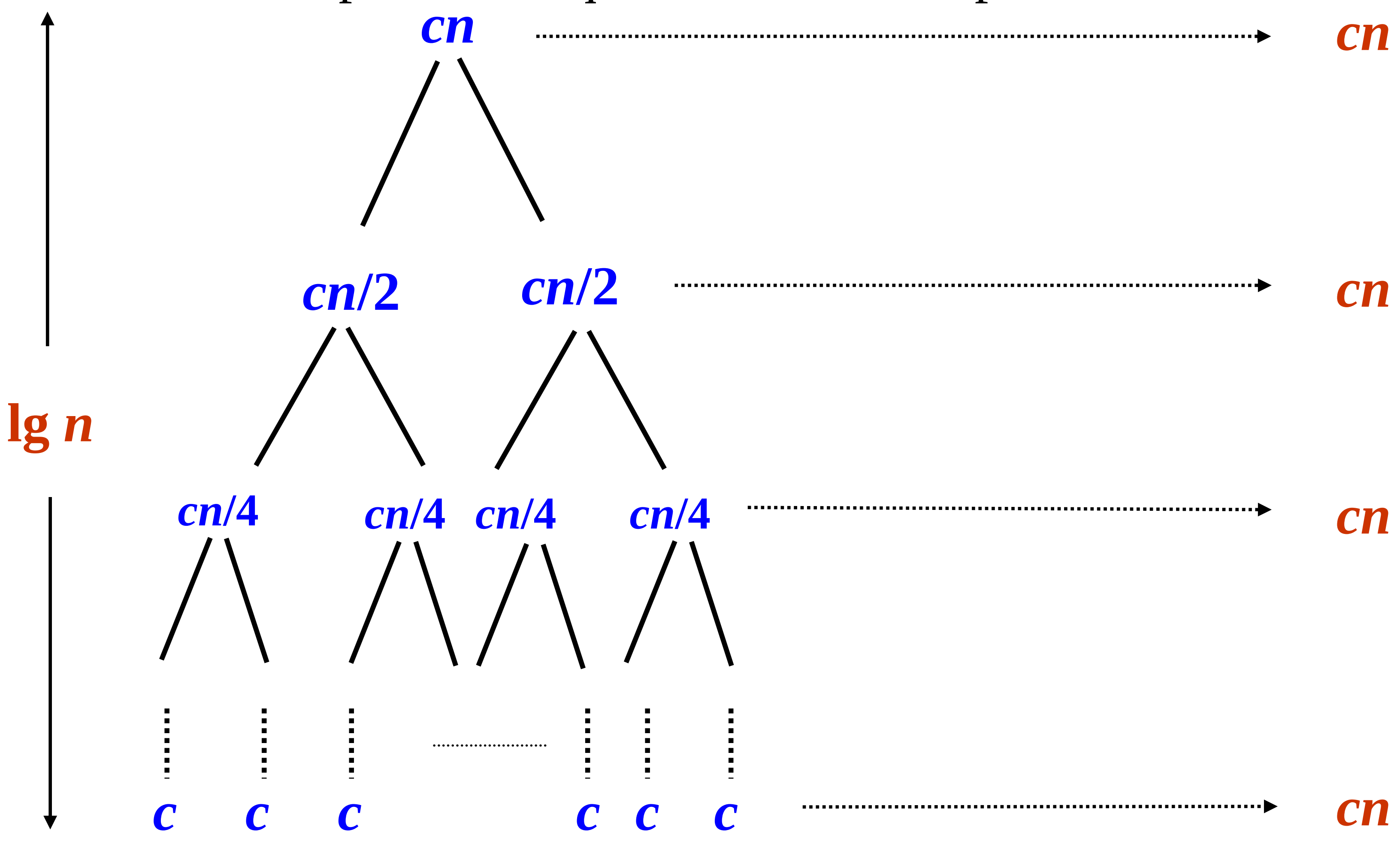
- O algoritmo recursivo possui três chamadas de função, sendo que as duas primeiras são chamadas recursivas e recebem metade dos elementos do vetor passado, e a outra é chamada para a função que realiza a intercalação das duas metades.
- Os valores necessários para a resolução do método mestre são:
- $a=2$, $b=2$, $f(n) = n$
- A expressão de recorrência do algoritmo QuickSort é dada por $T(n)=2T(n/2) + n$
- $f(n) = \Theta(n^{\log_b a})$
- $n = \Theta(n^{\log_2 2})$

$n = \Theta(n^1)$

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

Árvore de recursão Quick Sort

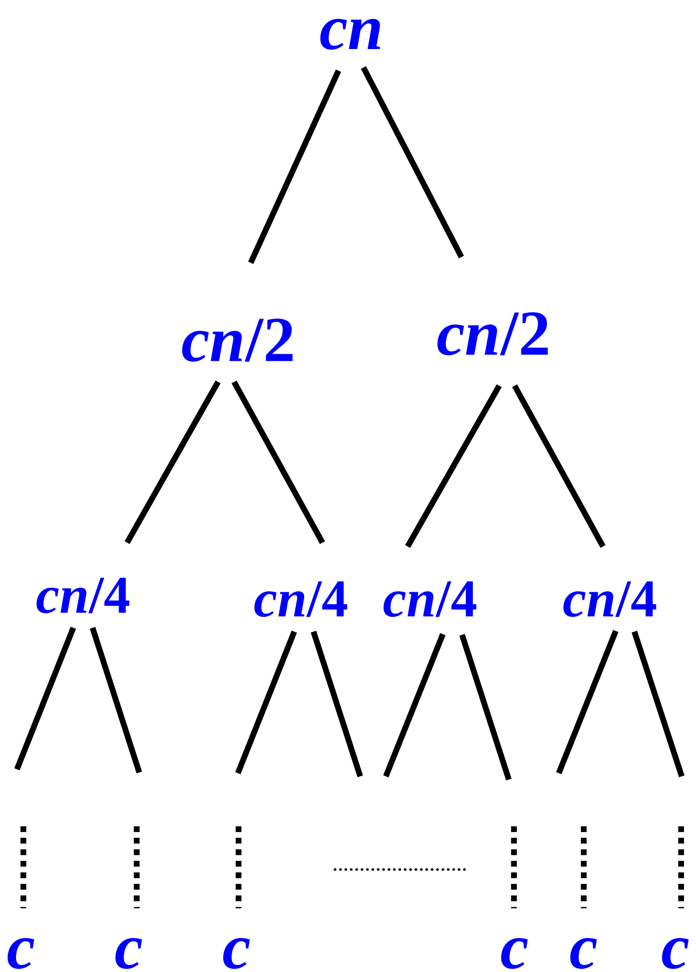
Continuar a expandir até que o tamanho do problema reduza a 1



Total : $cn\lg n + cn$

Árvore de recursão Quick Sort

Continua expandido o problema até 1.



•Cada nível tem um custo total ***cn***.

• Cada vez que vamos descer um nível, o número de subproblemas dobra, mas o custo por subproblema divide $\Rightarrow$ *Custo por nível permance o mesmo*.

•Existe lg n + 1 níveis, a altura é lg n. (Assumindo que n é uma potência de dois.)

•Custo total= soma dos custos em cada nível = (lg n + 1)cn = cnlg$n$ + cn = $\Theta(n \lg n)$.

Exercícios

Referência

CORMEN, T., LEISERSON, C., RIVEST, R.L., STEIN, C. Introduction to Algorithms., 2. ed. New York: MIT Press, 2001.