Projeto e Análise de Algoritmos

- Algoritmo da Inserção – insertion sort
- Algoritmo mergesort
- Análise assintótica

Análise de Algoritmos

Estudo sobre como os programas de computador se comportam em termos de desempenho e recursos de armazenamento utilizados

Outras características que podem ser mais importantes que o desempenho:

- modularidade
- Correção
- Manutenção
- Funcionalidade
- Robustez
- Amigável
- Custo
- Simplicidade
- Extensibilidade
- Confiabilidade

Motivos para se estudar o desempenho de algoritmos

- A análise de desempenho pode dizer o que possível e impossível fazer
- Um algoritmo bem projetado pode ter uma grande diferença de desempenho
- O desempenho pode ter um grande impacto nos custos

Problema da Ordenação

Entrada: uma sequencia $a_1, a_2, \ldots, a_n$ de números.

Saída: permutação $a'_1, a'_2, \ldots, a'_n$ tal que $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Exemplo:

Entrada: 8 2 4 9 3 6

Saída: 2 3 4 6 8 9

Algorítmo INSERTION-SORT

"pseudocódigo"

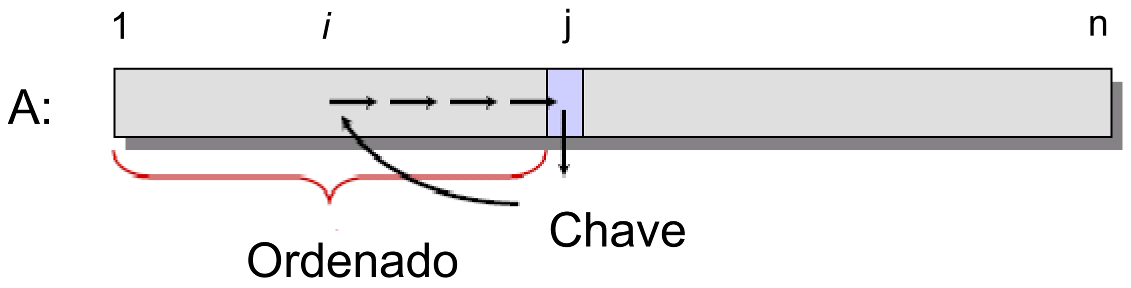
```
INSERTION-SORT (A, n)     A[1 . . n]
    for j ←2 to n  do
        chave ← A[ j]
        i ← j –1
        while i > 0 and A[i] > chave
            do A[i+1] ← A[i]
               i ← i –1
        A[i+1] = chave
```

Semelhante a ordenar cartas

INSERTION-SORT

"pseudocódigo"

```
INSERTION-SORT (A, n)        ⊳ A[1 . . n]
    for j ←2 to n  do
        chave ← A[ j]
        i ← j –1
        while i > 0 and A[i] > chave
            do A[i+1] ← A[i]
               i ← i –1
        A[i+1] = chave
```



Exemplo de insertion sort

8 2 4 9 3 6

Exemplo de insertion sort

8 2 4 9 3 6

Exemplo de insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

Exemplo de insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

Exemplo de insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

Exemplo de insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

Exemplo de insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

Exemplo de insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

Exemplo de insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

Exemplo de insertion sort

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6
2	3	4	8	9	6

Exemplo de insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

2 3 4 6 8 9 *Completo*

Tempo de Execução

- O tempo de execução depende do tamanho da entrada: uma lista que já foi ordenada é mais rápida
- Parametrizar o tempo de execução pelo tamanho da entrada: entradas maiores devem demorar mais tempo que as pequenas
- Normalmente procuramos limites superiores como limite total de tempo (como garantia do tempo)

Tipos de Análise

Pior Caso: (normalmente)

- $T(n)$ =tempo máximo que o algoritmo demora para qualquer entrada de tamanho n

Caso médio: (as vezes)

- $T(n)$ =tempo médio esperado para execução do algoritmo em todas as entradas de tamanho n
- Necessita de suposições estatísticas sobre a distribuição das entradas

Melhor caso: (problemático)

- Mesmo um algoritmo bem lento pode ser rápido para algum caso

Tempo independente da máquina

Qual é o pior caso para o algoritmo da inserção?

- Depende da velocidade do computador:
  - Velocidade relativa (no mesmo computador)
  - Velocidade absoluta(em diferentes máquinas)

Idéia:

- Ignorar as constantes que são dependentes da máquina
- Olhar o crescimento de $T(n)$ quando $n \to \infty$.

"Análise assintótica"

Notação Θ

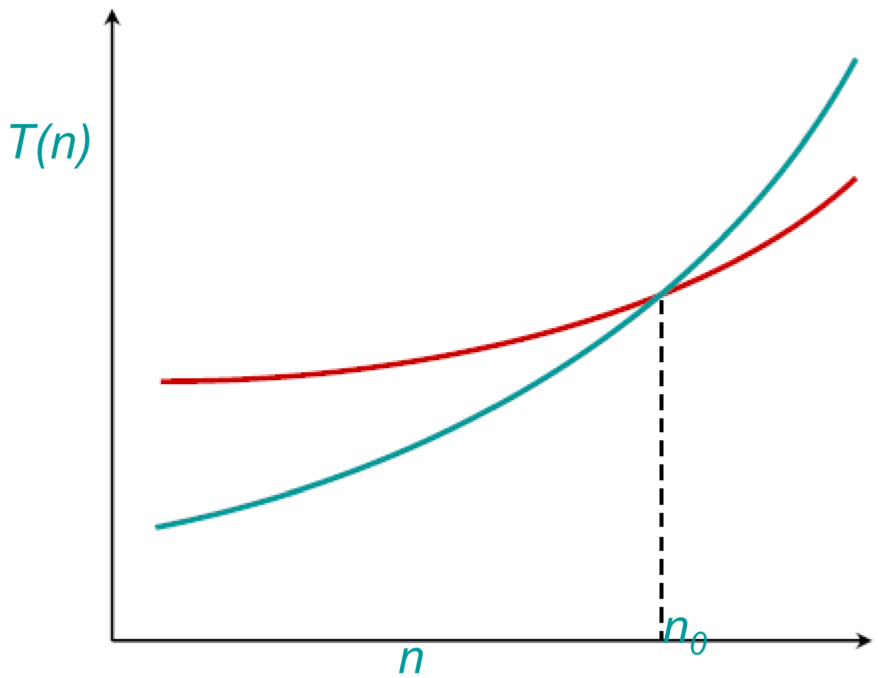
$\Theta(g(n)) = \{ f(n)$: existes constantes positivas c_1, c_2, n_0 tais que $\leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para todo $n \geq n_0\}$

Em geral:

- Pega o termo de maior ordem e ignora constantes
- Exemplo: $3n^3 + 90n^2 – 5n + 6046 = \Theta(n^3)$

Desempenho Assintótico

Quando o valorde n fica grande o suficiente uma algoritmo $\Theta(n^2)$ será mais rápido que um algoritmo $\Theta(n^3)$.



•Não se deve ignorar algoritmos assintoticamente mais lentos
•Eles podem ser úteis em situações reais
•Análise assintótica é uma ferramenta útil

Análise do insertion sort

Pior caso: entrada em ordem reversa.

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2)$$ [série aritmética]

Caso médio: Todas as permutações igualmente prováveis

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta(n^2)$$

Qual o desempenho do insertion-sort?

- Modesto, para um pequeno *n*.
- Ruim para um grande valor de *n*.

Merge sort

MERGE-SORT *A*[1 . . n]

1. Se n= 1, fim.
2. Ordena recursivamente A[1 . . .n/2.] e A[[n/2]+1 . . n] .
3. Faz “Merge” (junção) de duas listas ordenadas

Sub-rotina: MERGE

Fusão de duas listas ordenadas

20	12
13	11
7	9
2	1

Fusão de duas listas ordenadas

20	12
13	11
7	9
2	1

1

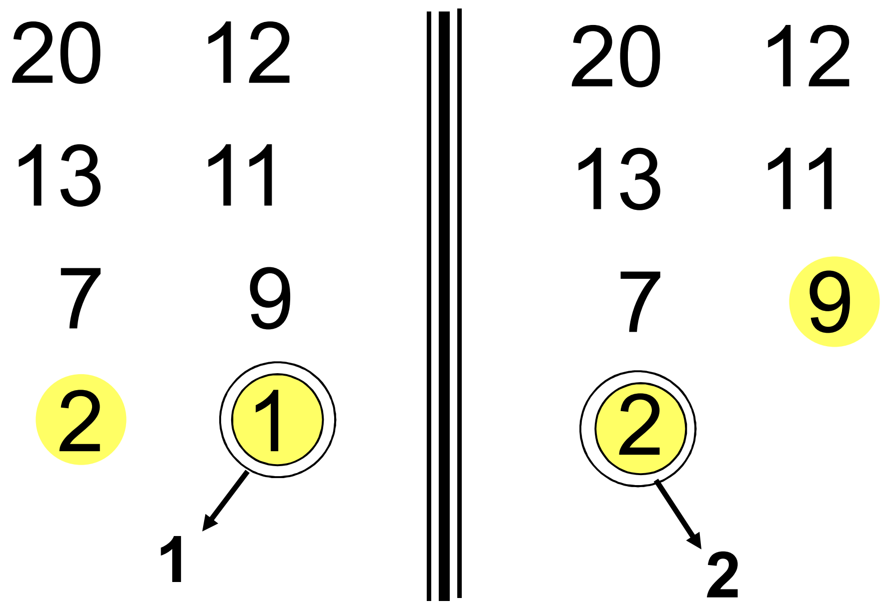
Fusão de duas listas ordenadas

20	12
13	11
7	9
2	1

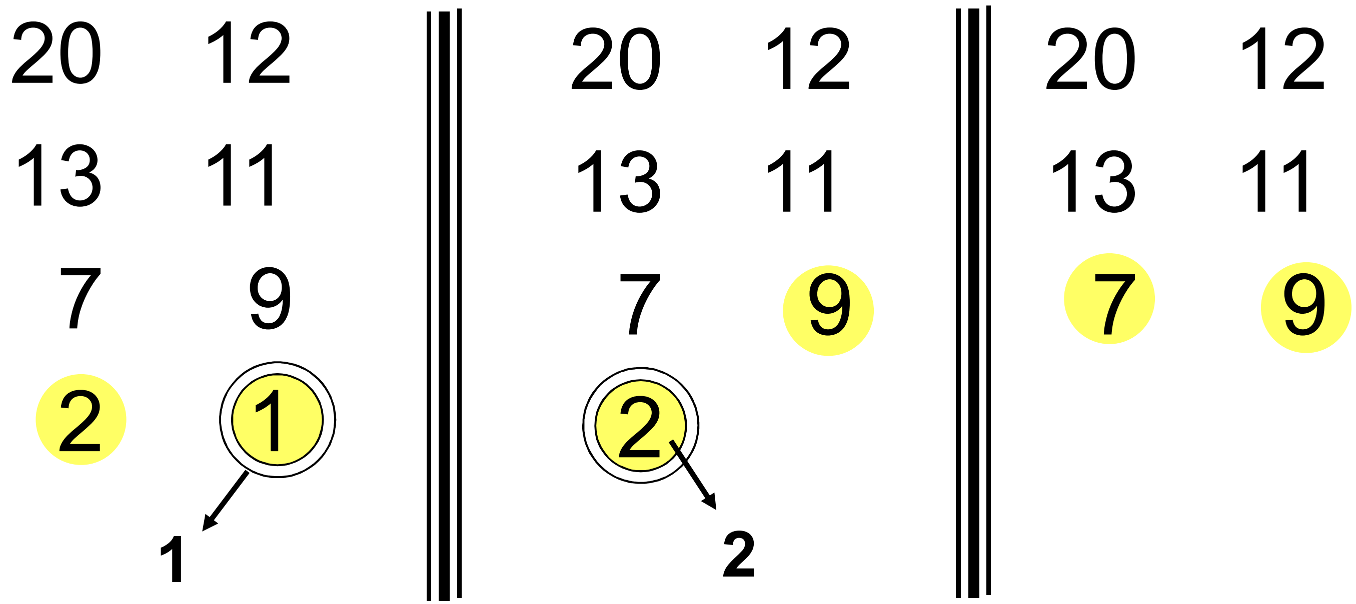
1

20	12
13	11
7	9
2	

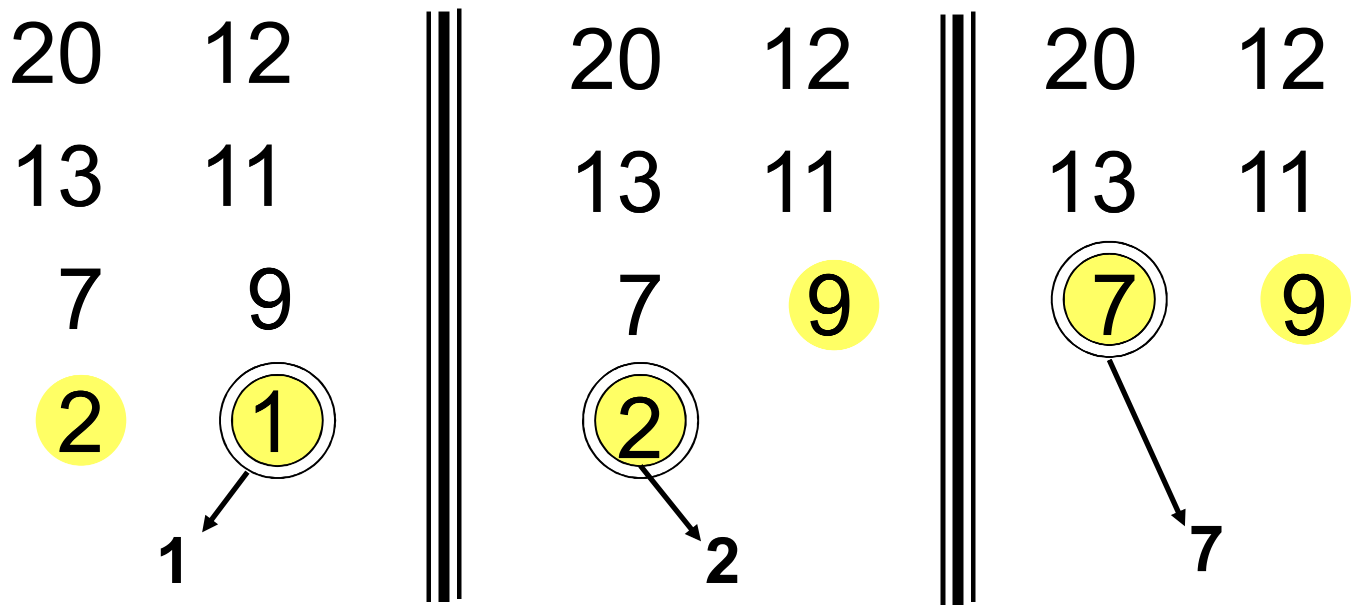
Fusão de duas listas ordenadas



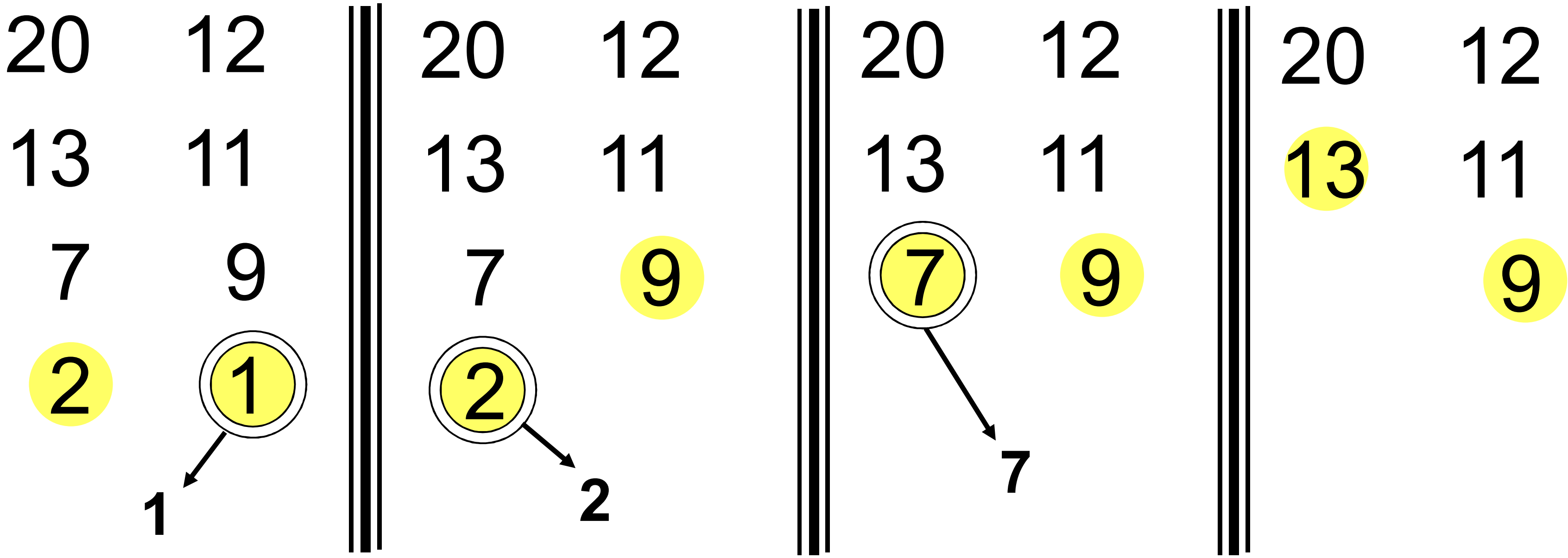
Fusão de duas listas ordenadas



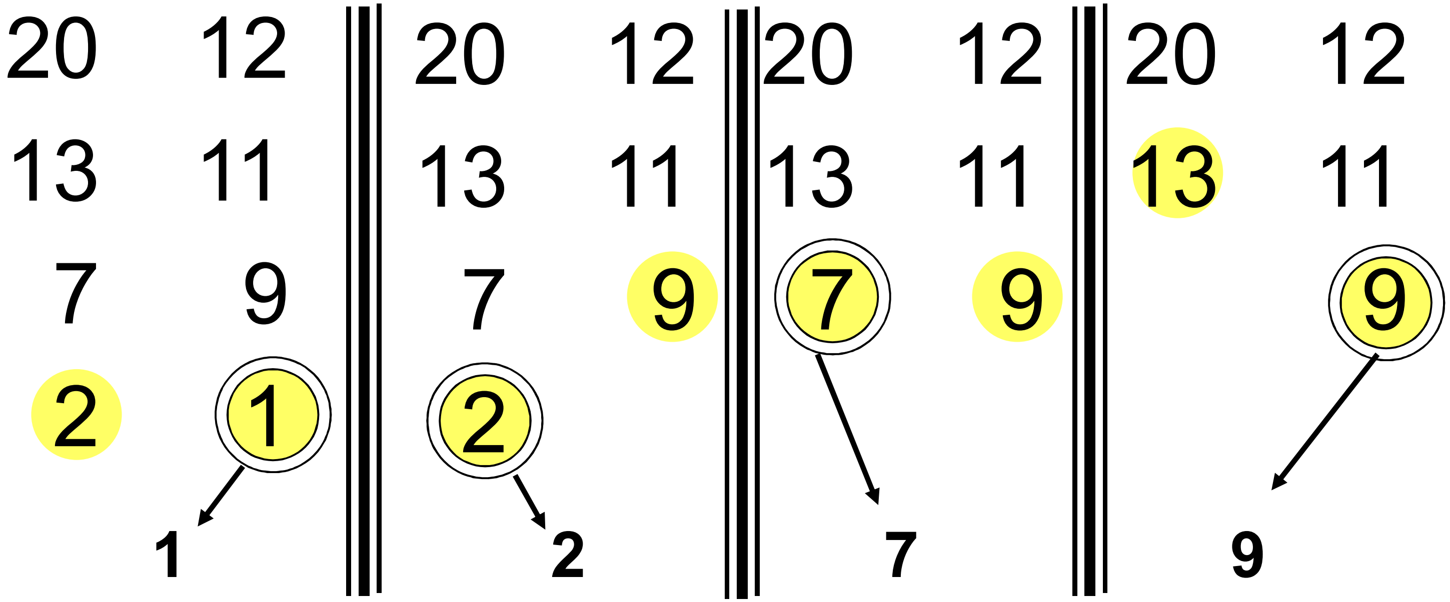
Fusão de duas listas ordenadas



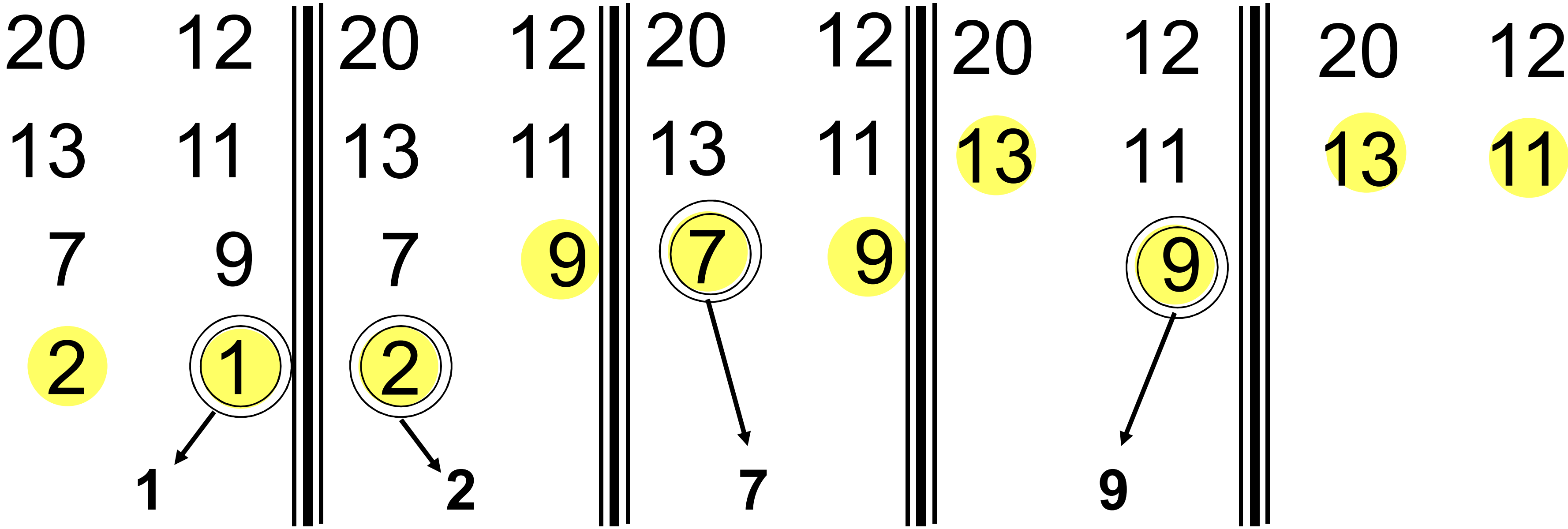
Fusão de duas listas ordenadas



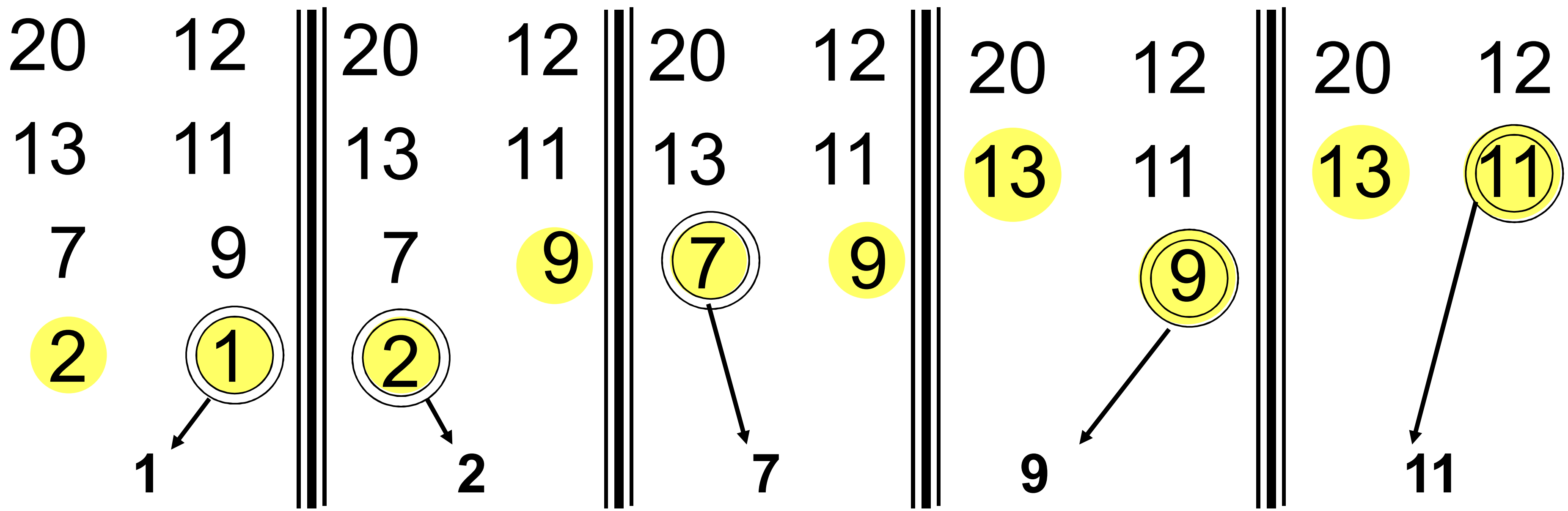
Fusão de duas listas ordenadas



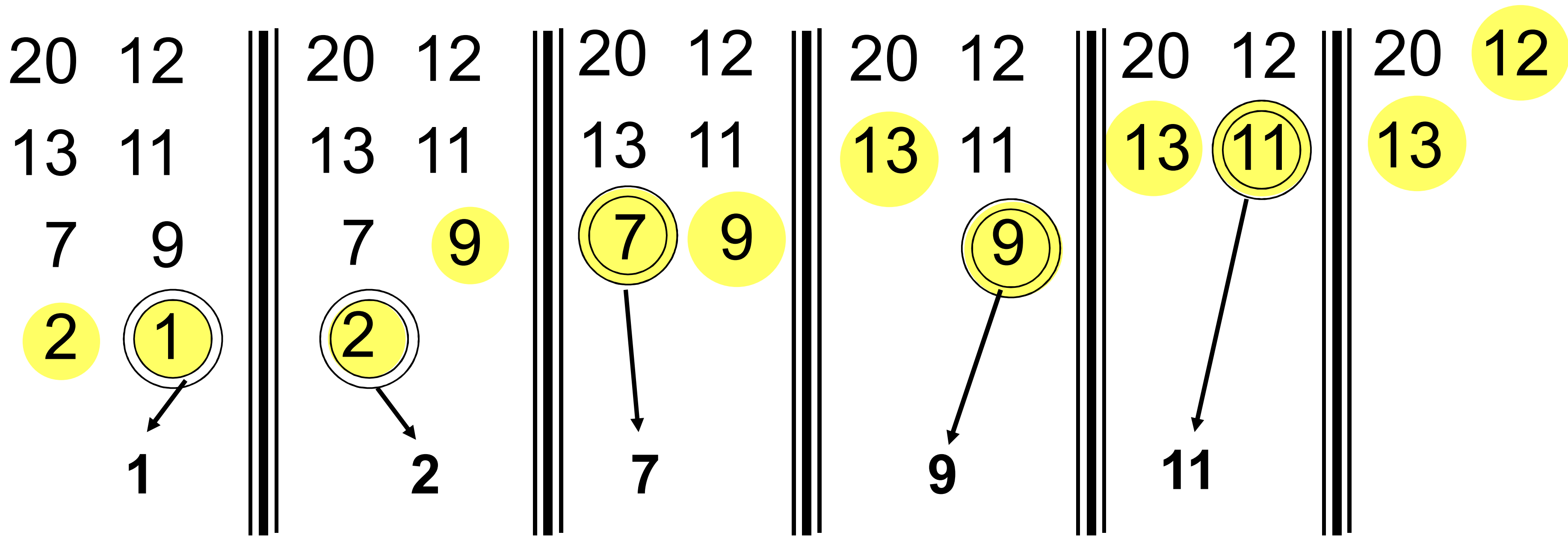
Fusão de duas listas ordenadas



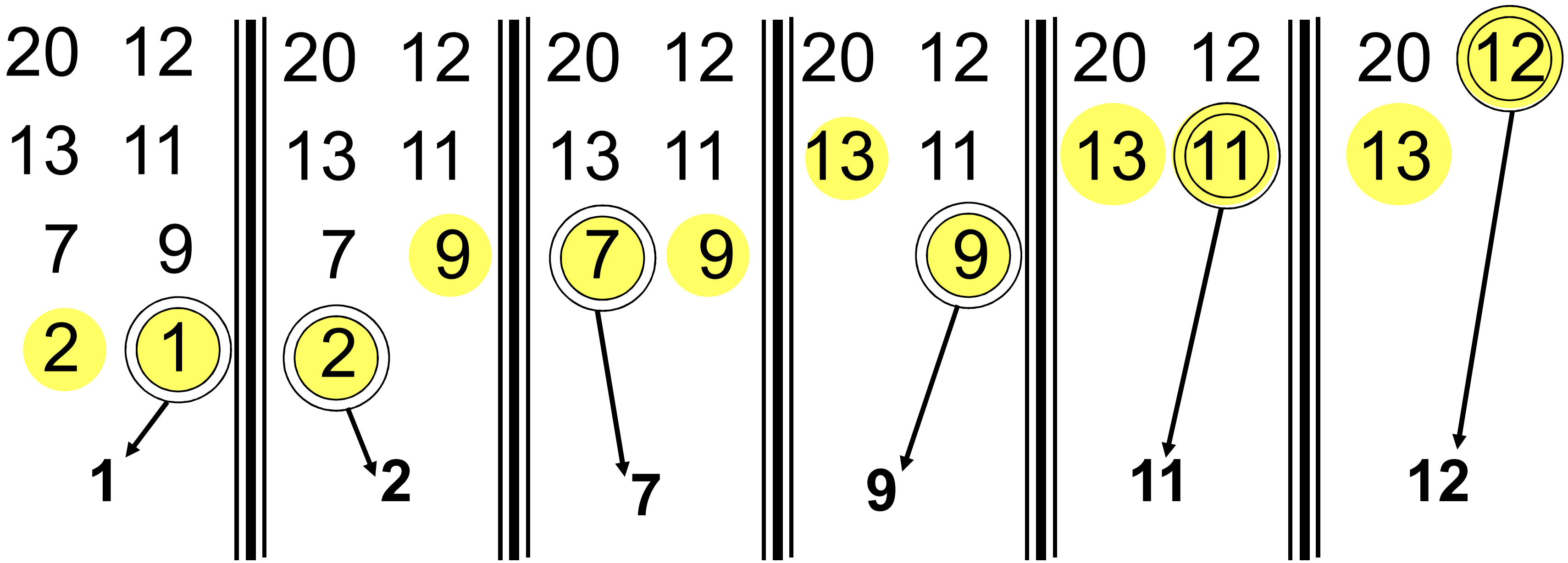
Fusão de duas listas ordenadas



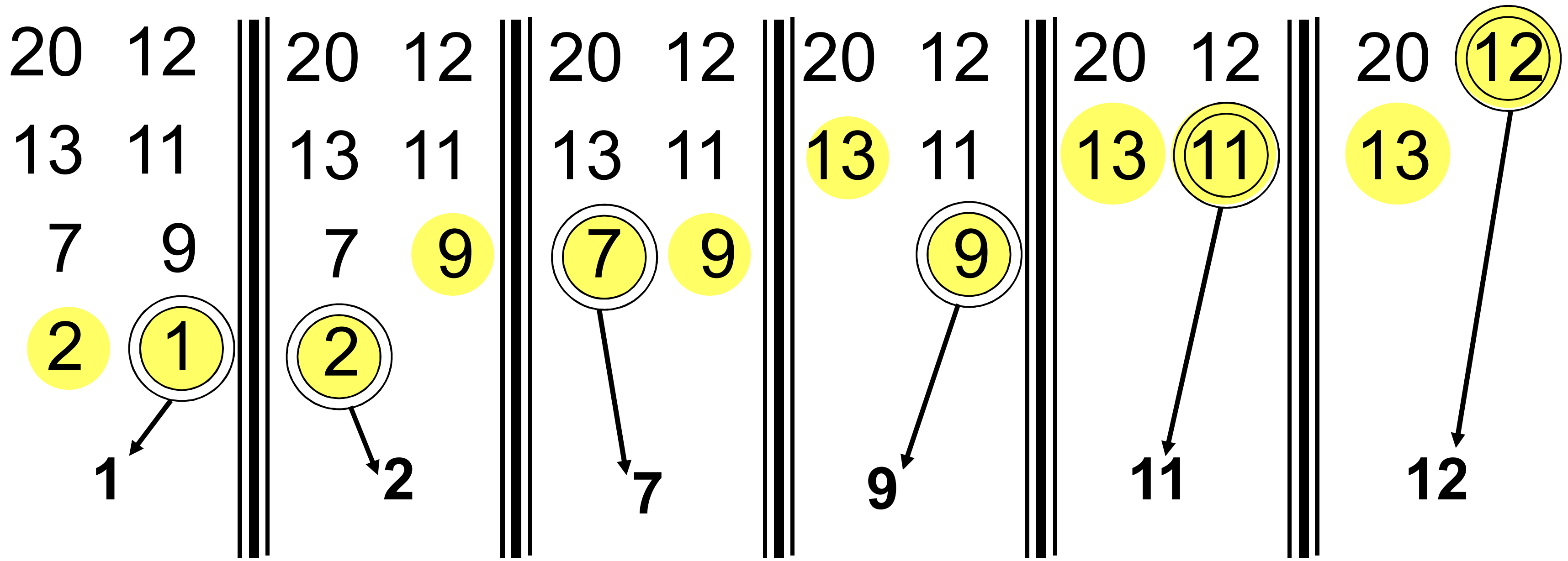
Fusão de duas listas ordenadas



Fusão de duas listas ordenadas



Fusão de duas listas ordenadas



Tempo = Θ(n) para fundir os
n elementos (tempo linear).

Análise do merge sort

$T(n)$	MERGE-SORTA[1 . . n]
$\Theta(1)$	1.Se n= 1, fim.
$2T(n/2)$	2.Recursivamente ordene A[1.. n/2] e A[n/2 +1 .. n] .
$\Theta(n)$	3.“Merge” duas listas ordenadas

Deveria ser $T(\lceil n/2 \rfloor) + T(\lceil n/2 \rfloor)$,

Mas não faz diferença assintoticamente

Recorrência do merge sort

$$T(n) = \begin{cases} \Theta(1) \text{ se } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

- Podemos ignorar ocaso inicial quando $T(n) = \Theta(1)$ para um *n pequeno*, mas só quando não afeta a solução assintótica da recorrência

Árvore de recursão

Resolver $T(n) = 2T(n/2) + cn$, onde $c > 0$ é constante.

Árvore de recursão

Resolver $T(n)=2T(n/2)+cn$, onde $c > 0$ é constante

$T(n)$

Árvore de recursão

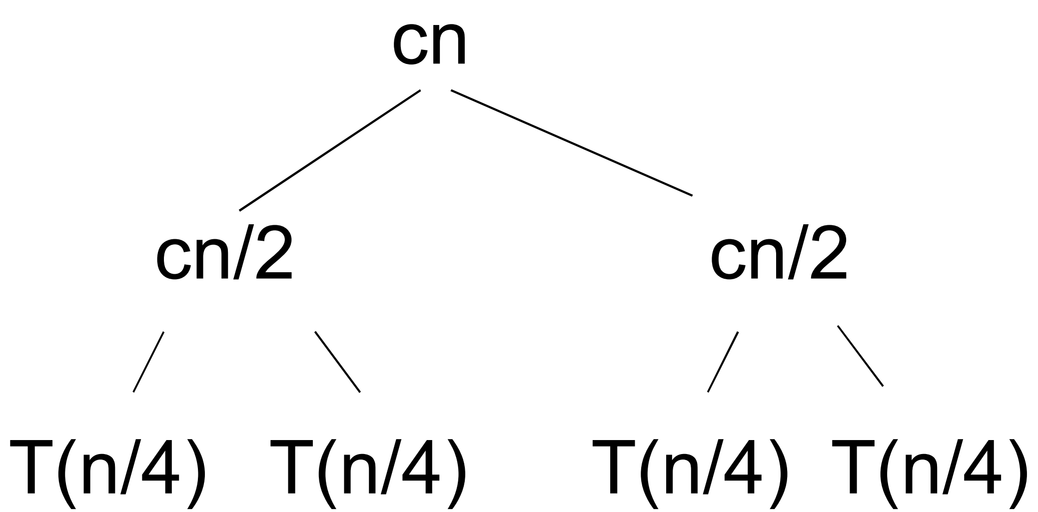
Resolver $T(n)=2T(n/2)+cn$, onde $c > 0$ é constante

$$cn$$

$$T(n/2) \qquad T(n/2)$$

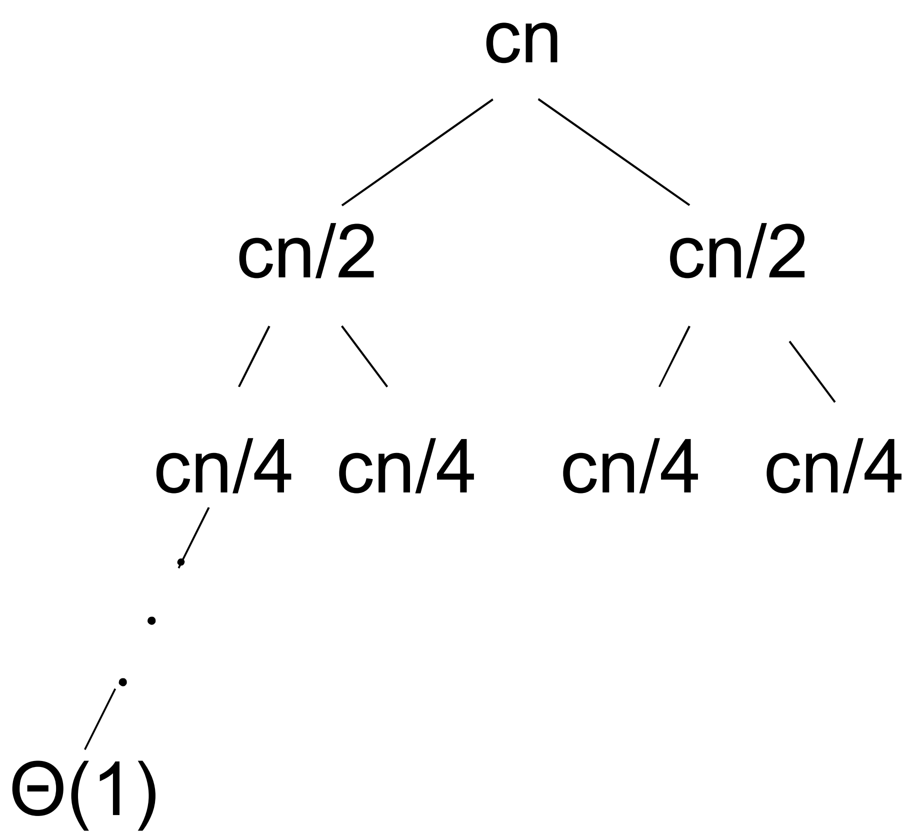
Árvore de recursão

Resolver $T(n)=2T(n/2)+cn$, onde $c > 0$ é constante



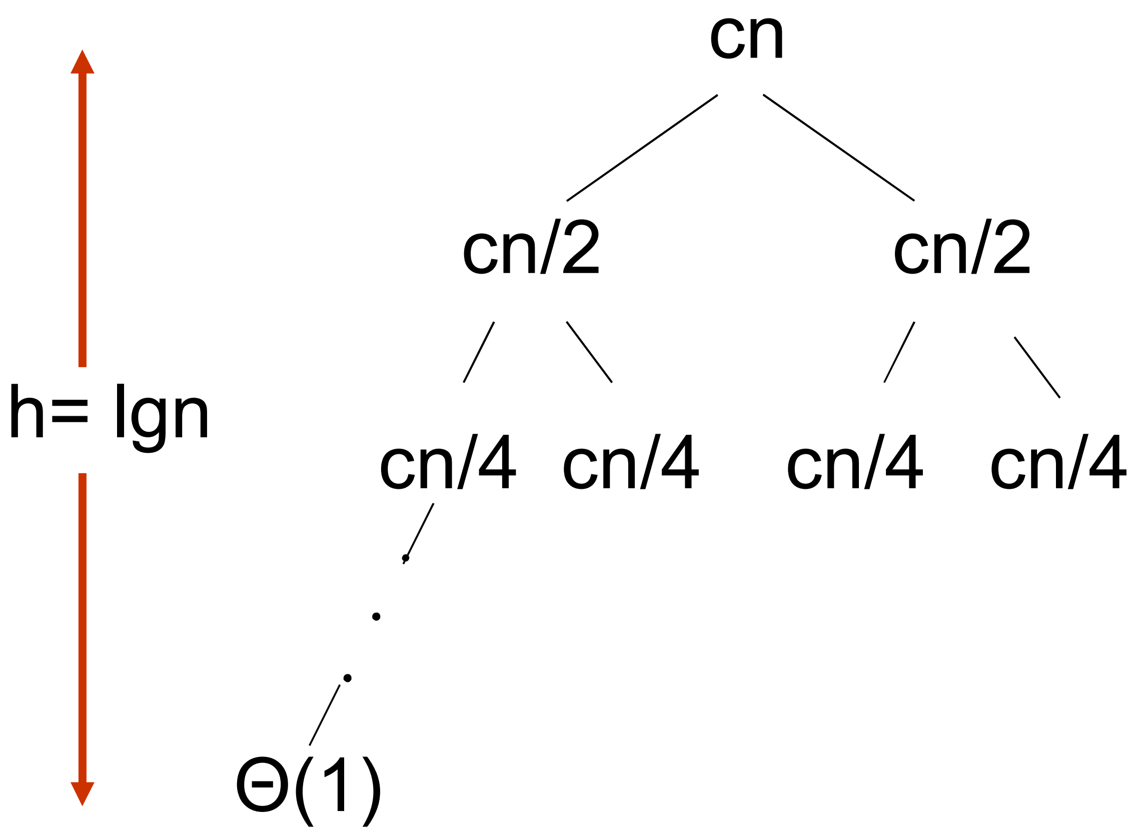
Árvore de recursão

Resolver $T(n)=2T(n/2)+cn$, onde $c > 0$ é constante



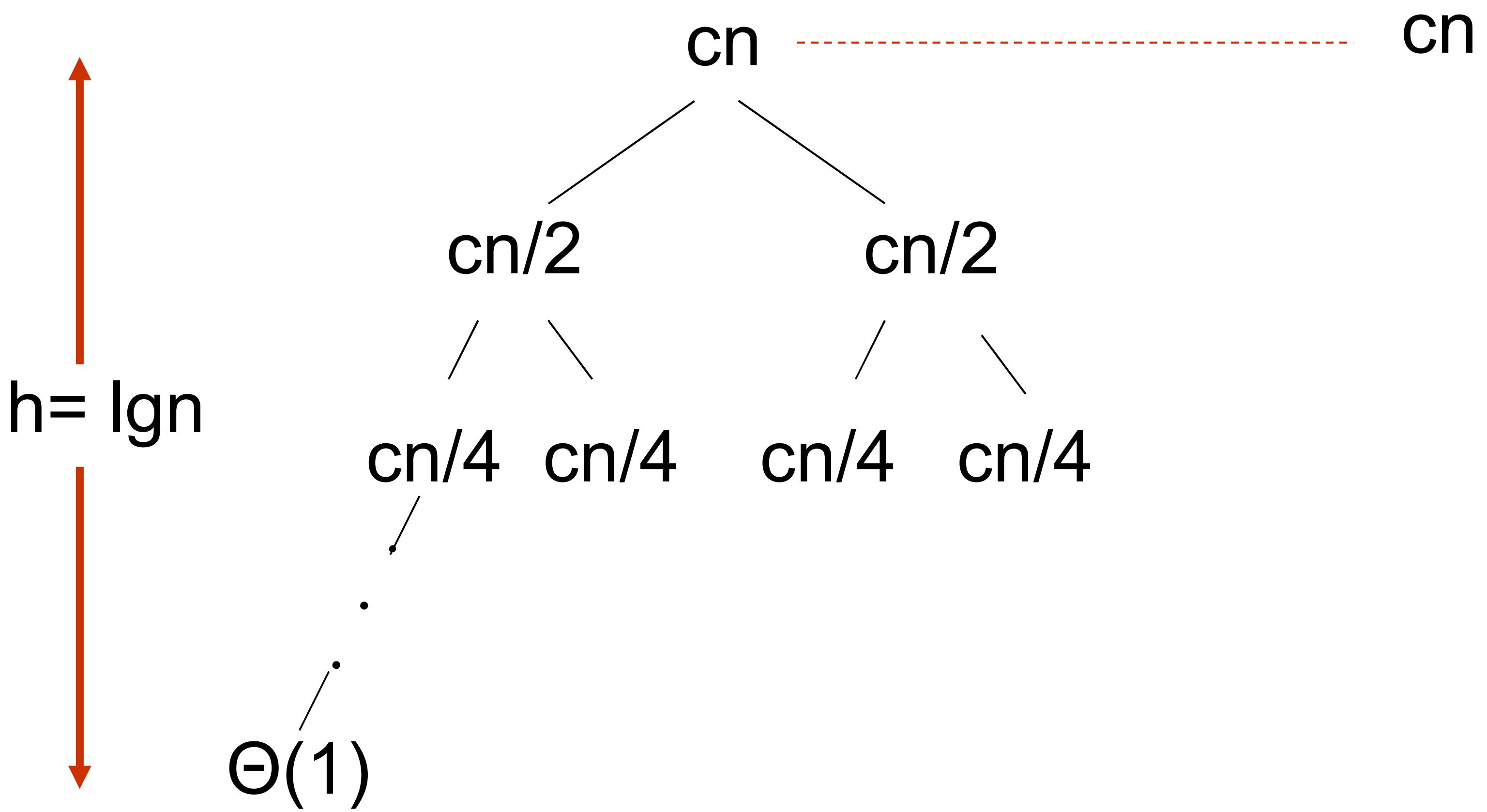
Árvore de recursão

Resolver $T(n)=2T(n/2)+cn$, onde $c > 0$ é constante



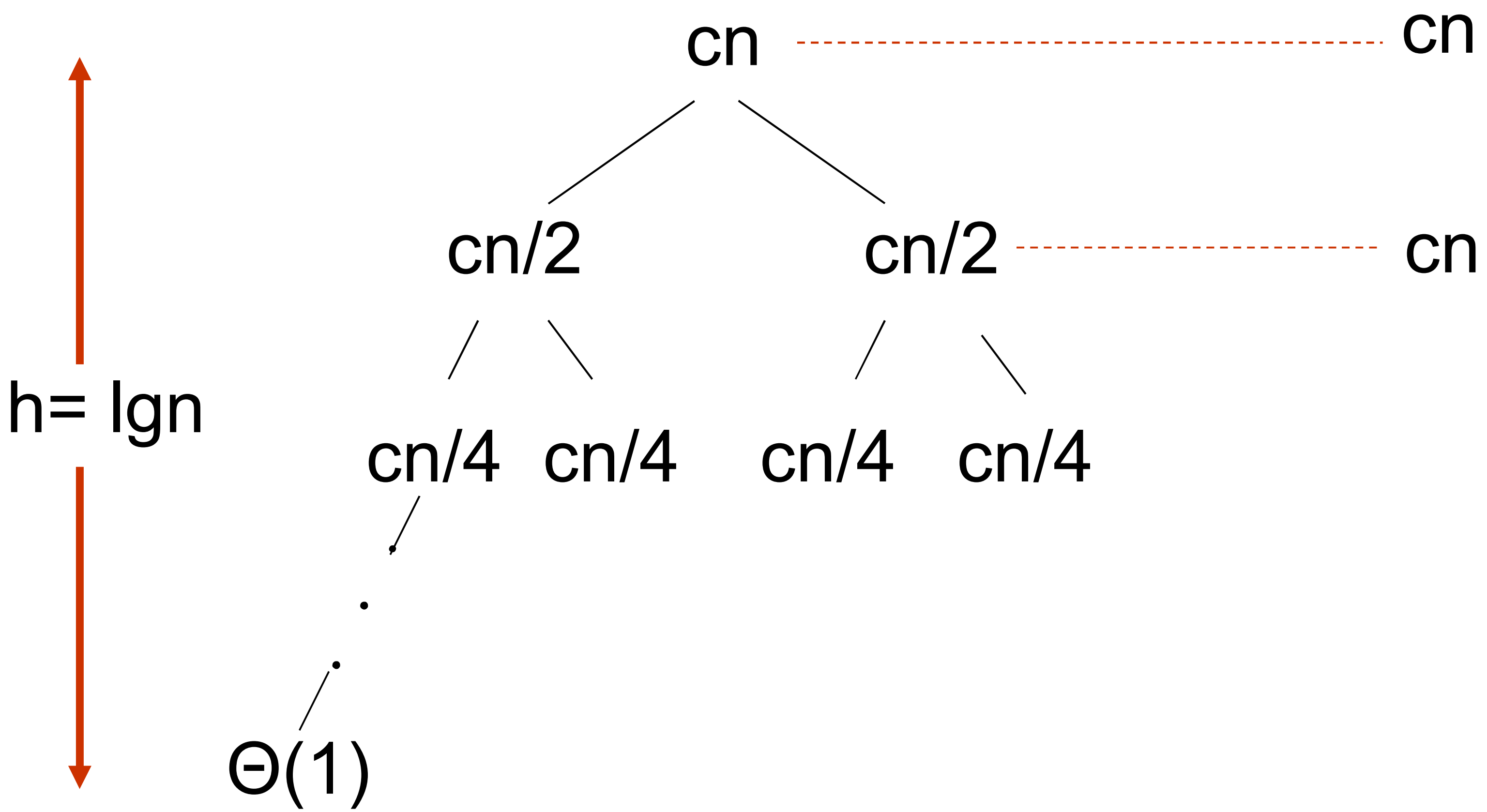
Árvore de recursão

Resolver $T(n)=2T(n/2)+cn$, onde $c > 0$ é constante



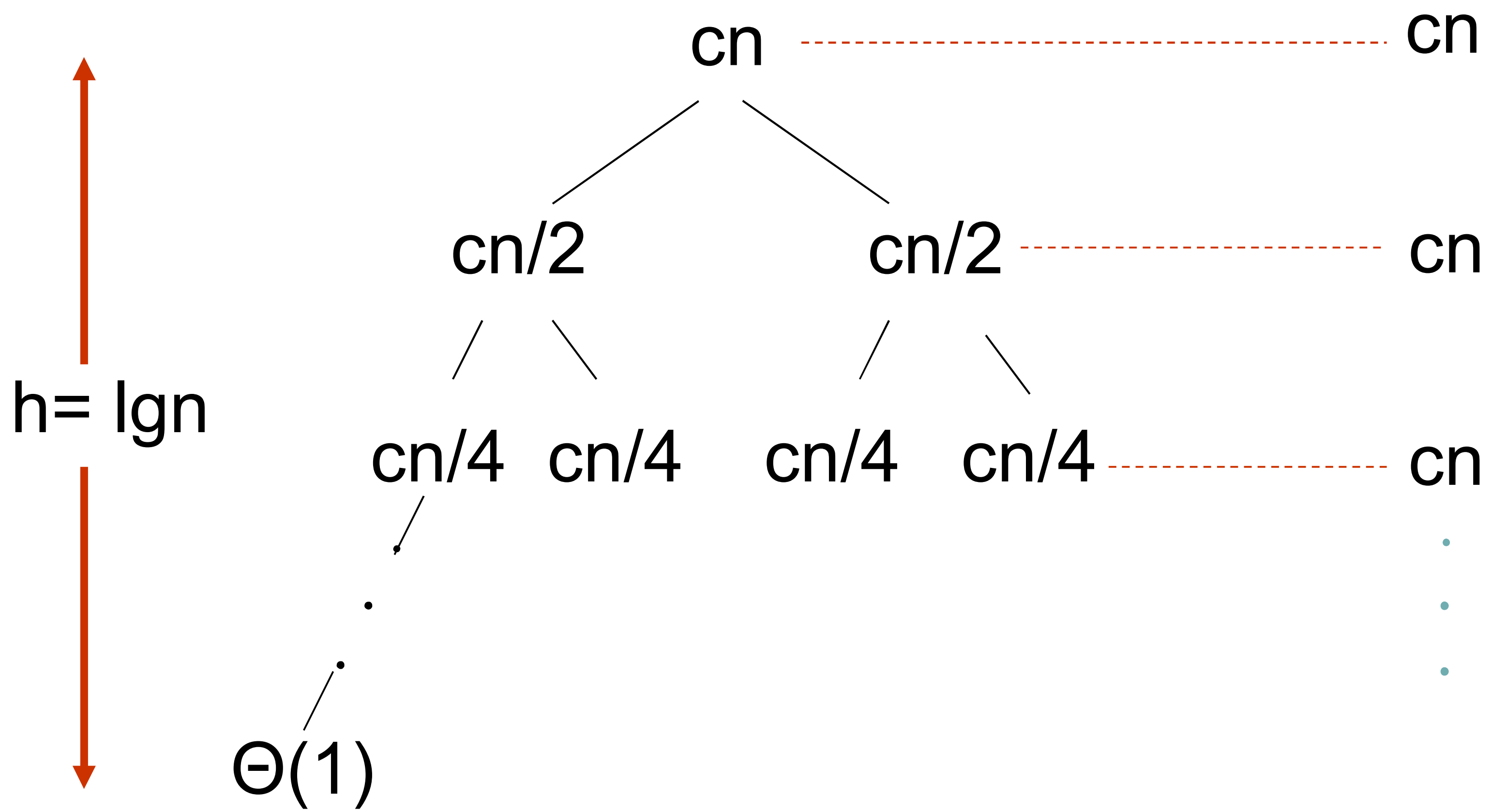
Árvore de recursão

Resolver $T(n)=2T(n/2)+cn$, onde $c > 0$ é constante



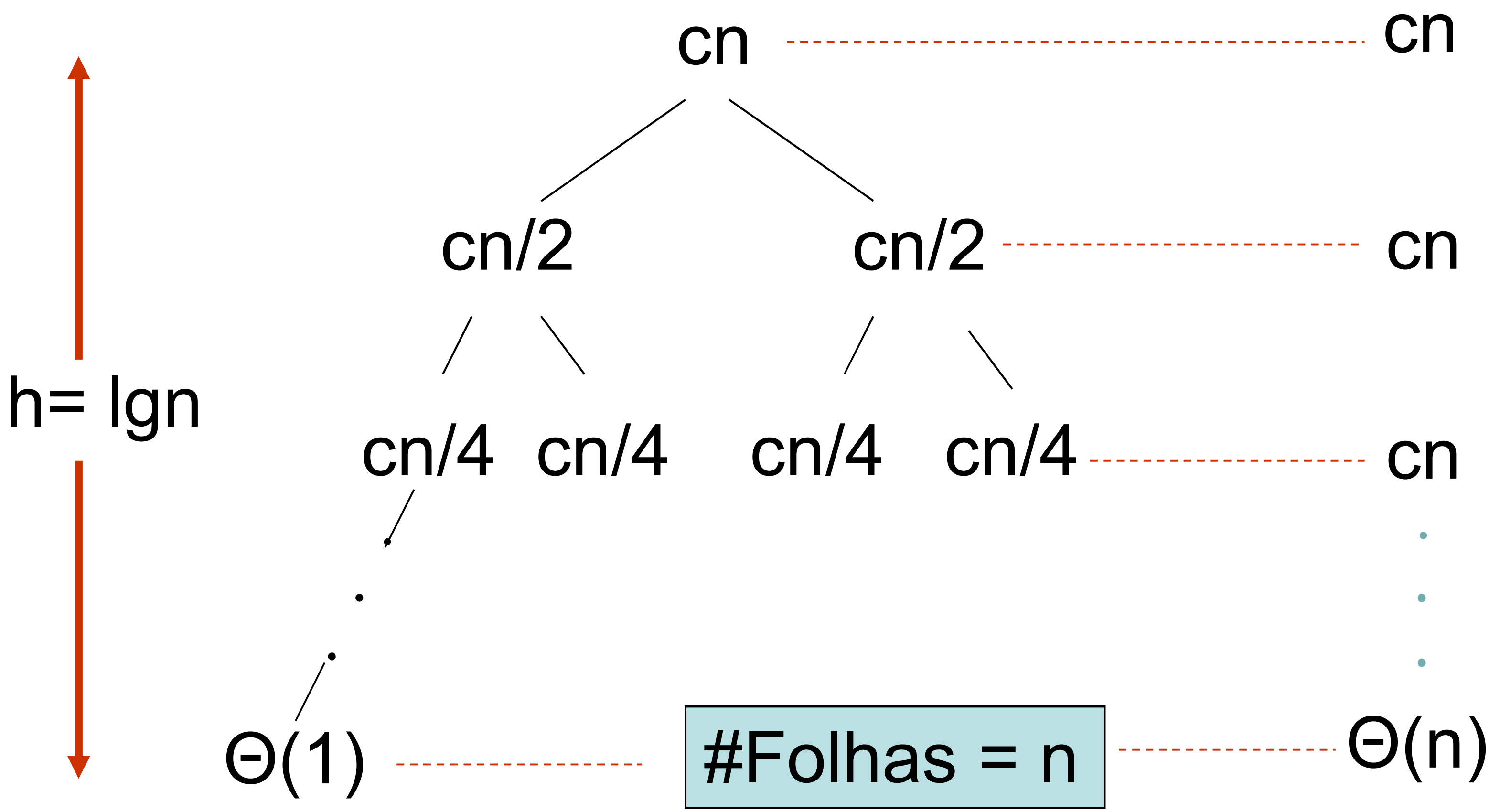
Árvore de recursão

Resolver $T(n)=2T(n/2)+cn$, onde $c > 0$ é constante



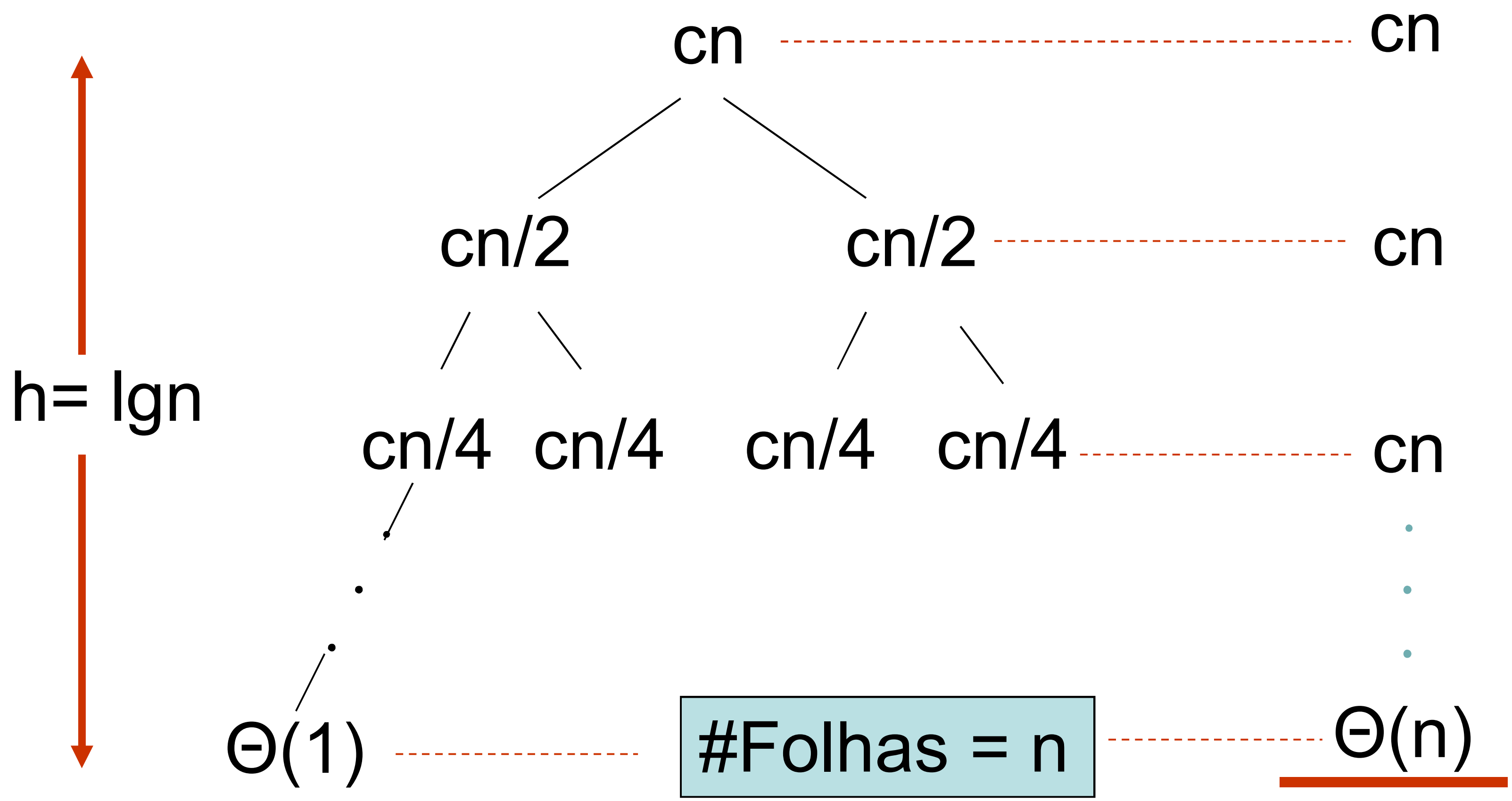
Árvore de recursão

Resolver $T(n)=2T(n/2)+cn$, onde $c > 0$ é constante



Árvore de recursão

Resolver $T(n)=2T(n/2)+cn$, onde $c > 0$ é constante



Total= Θ(n lg n)

Conclusões

- $\Theta(n \lg n)$ cresce mais lentamente que $\Theta(n^2)$.
- Portanto o merge sort é mais rápido que o insertion sort no pior caso para um n suficientemente grande.
- Na prática o merge sort supera o insertion sort para n> 30