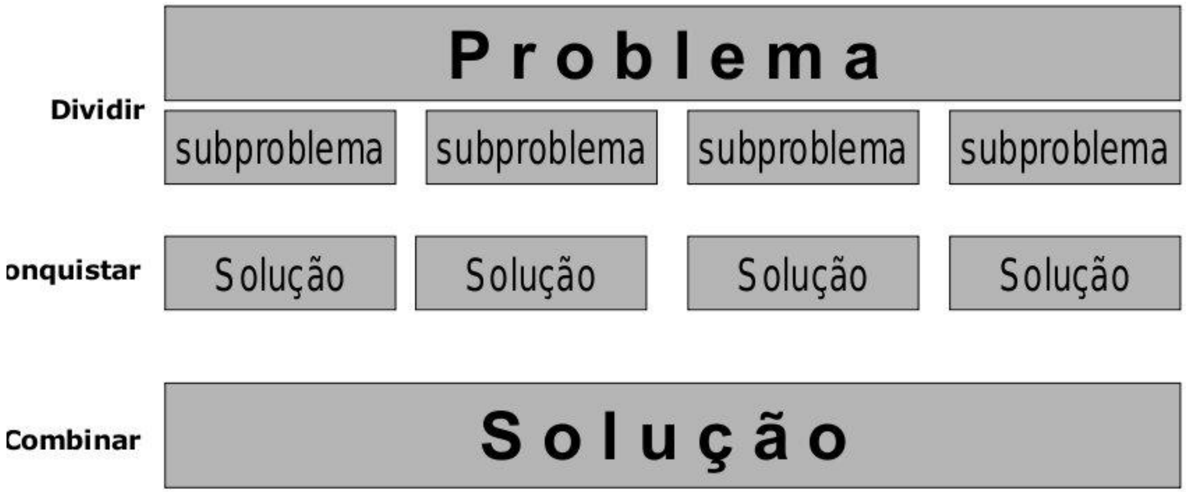
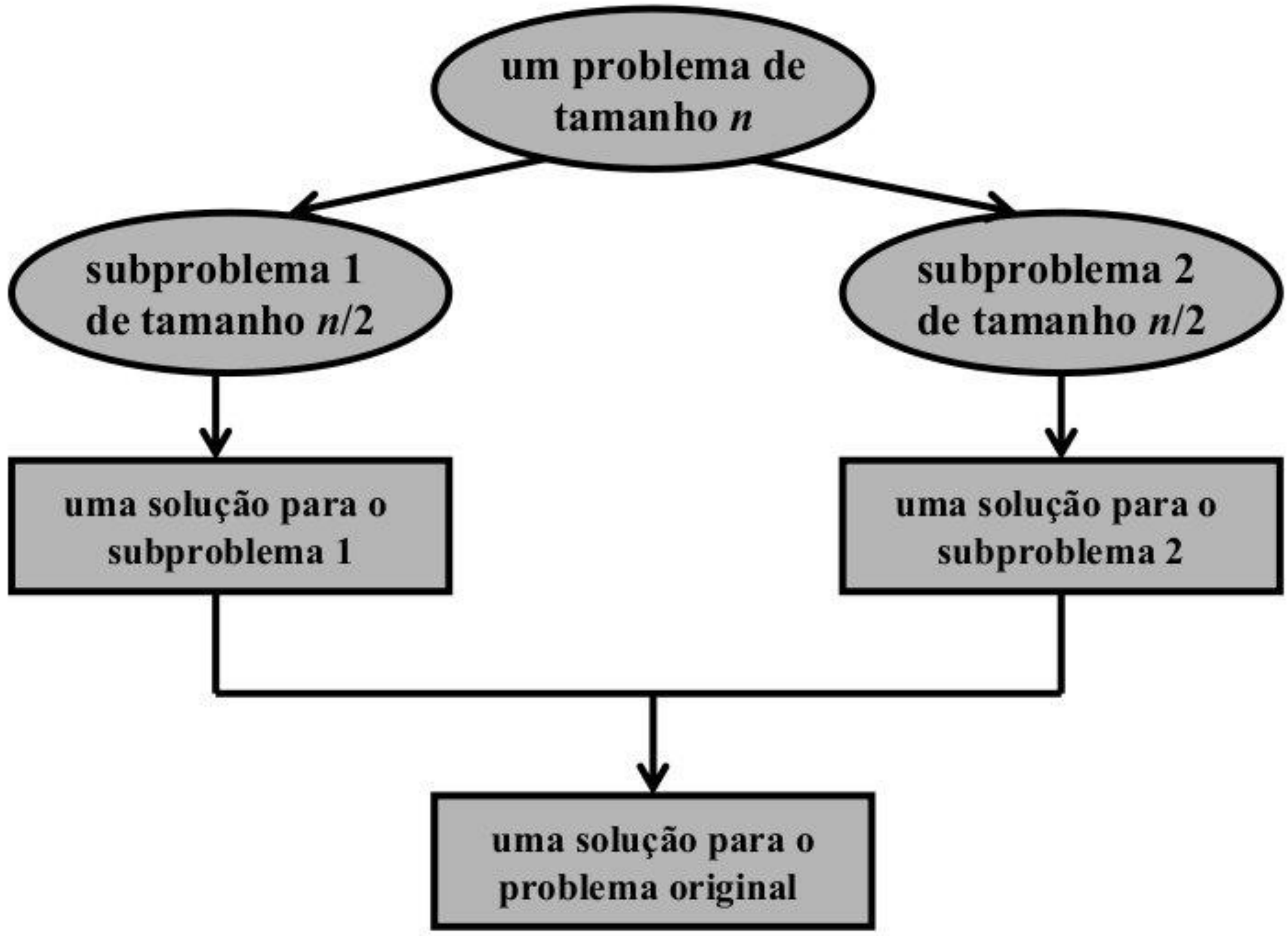
Merge Sort
Ordenação por Intercalação

- Técnica de Divisão e Conquista
  - Dividir a instância do problema em duas ou mais instâncias menores
  - Resolver as instâncias menores recursivamente
  - Obter a solução para as instâncias originais (maiores) através da combinação destas soluções



Estratégia:
Dividir & Conquistar



Dividir & Conquistar
Aplicação exemplo

- Computar a soma de n números $a_0, \ldots, a_{n-1}$
- Se $n>1$, podemos dividir o problema em duas instâncias do mesmo problema:
  - Soma dos primeiros $\lfloor n/2 \rfloor$ números
  - Soma dos $\lceil n/2 \rceil$ números restantes
- Uma vez estas duas somas computadas, adiciona-se seus valores para obter o resultado final:
- $a_0, \ldots, a_{n-1} = (a_0 + \ldots + a_{\lfloor n/2 \rfloor - 1}) + (a_{\lfloor n/2 \rfloor} + \ldots + a_{n-1})$

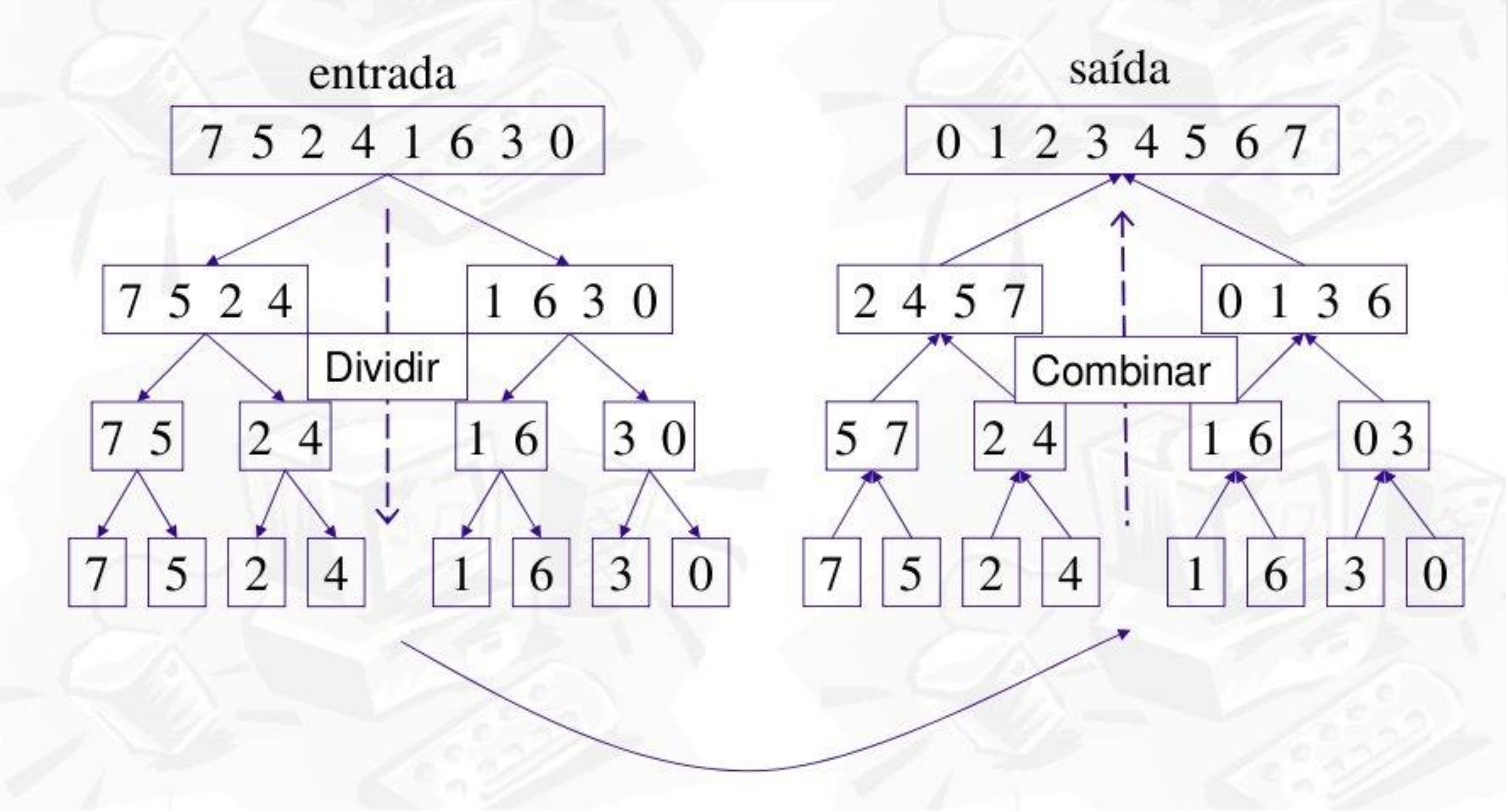
Dividir & Conquistar
Conclusões

- Usa-se, geralmente, chamadas recursivas ao mesmo algoritmo com instâncias menores (divididas)
  - Assim, obtem-se a seguinte recorrência:
    - $T(n) = aT(n/b) + f(n)$
- A estratégia D&C produz muitos algoritmos importantes e eficientes em ciência da computação, porém, nem todo algoritmo D&C é mais eficiente do que outra solução
- **A estratégia dividir & conquistar é idealmente adaptada a computação paralela**

Merge Sort
Estratégia D&C:

- Seja uma lista A de n elementos:
  - **Dividir** A em 2 sub-listas de tamanho $\approx n/2$
  - **Conquistar**: ordenar cada sub-lista chamando MergeSort recursivamente
  - **Combinar** as sub-listas ordenadas formando uma única lista ordenada (Merge)
  - **Caso base**: lista com um elemento

Merge Sort
Exemplo



Merge Sort
Exemplo animado

- Merge-sort with Transylvanian-saxon (German) folk dance
- https://youtu.be/XaqR3G_NVoo

Merge Sort
Algoritmo principal

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

Merge Sort
Algoritmo *MERGE*

- Combina (intercala ou mescla) dois arranjos ordenadas
  - $MERGE(A,p,q,r)$
  - A – arranjo
  - p, q e r são índices de enumeração de A, tal que:
    $p \leq q < r$
  - Sendo: $A[p..q]$ e $A[q+1..r]$ ordenados
- Analogia com o jogo de cartas: duas pilhas ordenadas com as cartas de menor valor para cima...
  - Juntar as 2 pilhas em uma pilha única de saída ordenada
    - Carta sentinela na parte inferior de cada pilha: $+\infty$
  - $\Theta(n)$ onde $n = r - p + 1$

Merge Sort

Algoritmo *MERGE*

(combina)

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
for i ← 1 to n1
        do L[i] ← A[p + i − 1]
for j ← 1 to n2
        do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
        do if L[i] ≤ R[j]
              then A[k] ← L[i]
                   i ← i + 1
              else A[k] ← R[j]
                   j ← j + 1
```

Algoritmo *MERGE*
Entendendo o Algoritmo

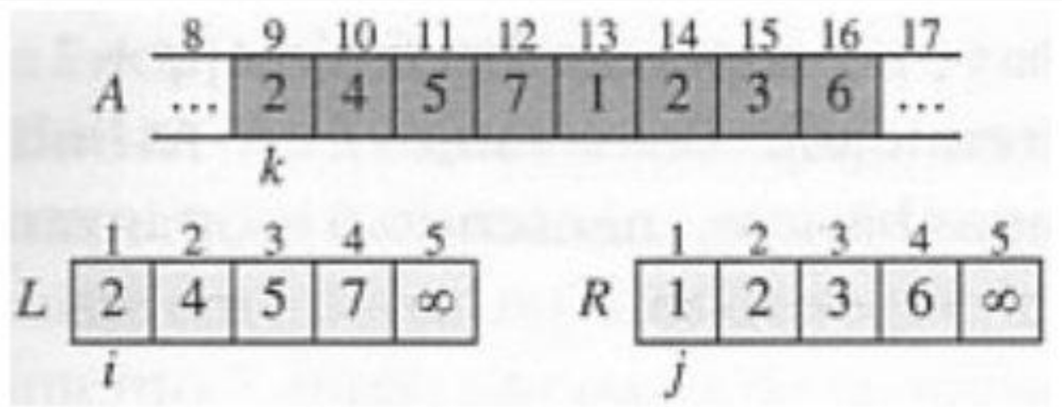
- Linhas 1 a 2: Comprimento n_1 e n_2 do subarranjo $A[p..q]$ e $A[q+1..r]$
- Linha 3: Cria subarranjos L e R de tamanho n_1+1 e n_2+1
- Linhas 4 a 7: Copia os subarranjos $A[p..q]$ e $A[q+1..r]$ em $L[1..n_1]$ e $R[1..n_2]$
- Linhas 8 e 9: Coloca sentinelas nas extremidades

Algoritmo *MERGE*
Entendendo o Algoritmo (II)

- Linhas 10 a 17: Executam as $r-p+1$ etapas básicas, mantendo o laço invariante a seguir:
  - No início de cada iteração do laço for, o subarranjo $A[p...k-1]$ contém os $k-p$ menores elementos de $L[1...n_1+1]$ e $R[1...n_2+1]$ em seqüência ordenadas
  - $L[i]$ e $R[j]$ são os menores elementos de seus arranjos que não foram copiados de volta em A
  -

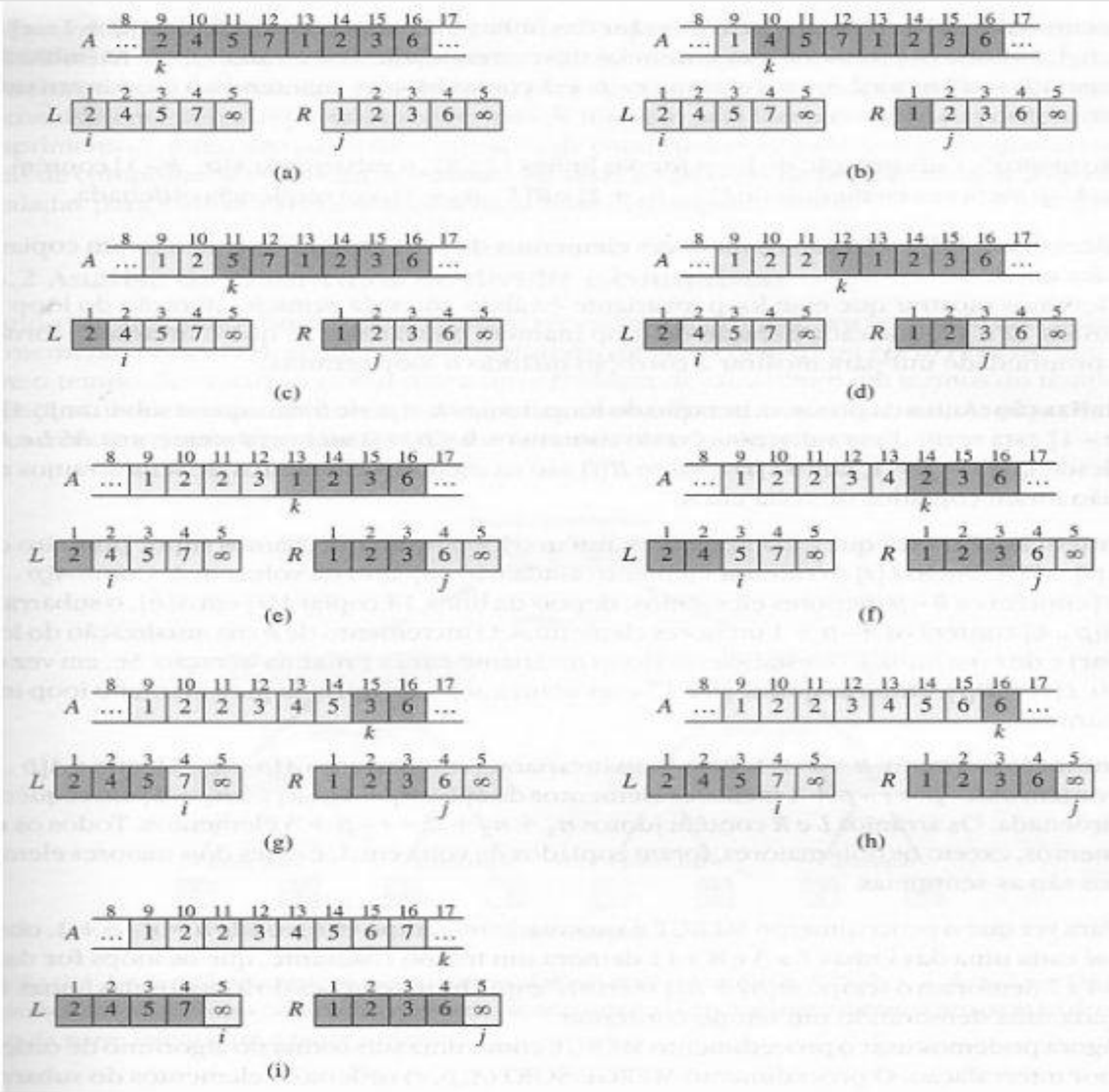
Algoritmo *MERGE*
Exemplo de mesclagem

- Aplicando para o arranjo a seguir:



Algoritmo *MERGE*

Exemplo de mesclagem (II)



Merge Sort
Análise do algoritmo *MERGE*

- É executado no tempo $\Theta(n)$, onde $n = r-p+1$
  - Linhas 1-3 e 8-11: Tempo constante
  - Linhas 4-7: laço *for*: Tempo $\Theta(n_1+n_2) = \Theta(n)$
  - Linhas 12-17: laço *for*: n iterações com tempo constante

- Podemos usar o procedimento Merge como uma subrotina no algoritmo de ordenação por intercalação

Merge Sort
Análise do algoritmo

- Tempo de execução escrito por uma recorrência
  - Recorrência: descreve o tempo sobre n elementos em função do tempo sobre entradas menores que n
    - Ex.: $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + cn$
  - Em geral:

$$T(n) = \begin{cases} \Theta(1) & se \quad n < c \\ aT(n/b) + D(n) + C(n) & caso \quad contrário \end{cases}$$

    - a: quantidade de subproblemas ($a=2$)
    - n/b: tamanho dos subproblemas ($a=b=2$)
    - $D(n)$: tempo gasto na etapa de divisão
    - $C(n)$: tempo gasto na etapa de combinação

Merge Sort
Análise do algoritmo (II)

- A ordenação por intercalação sobre um único elemento ($n=1$) demora um tempo constante
- Quando $n > 1$ elementos, tem-se:
  - **Dividir**: Simplesmente calcula o ponto médio do subarranjo, o que demora um tempo constante: $D(n) = \Theta(1)$
  - **Conquistar**: Resolvesse recursivamente dois subproblemas, cada um tem tamanho $n/2$, contribuindo com $2T(n/2)$ no custo de execução
  - **Combinar**: procedimento *Merge* leva o tempo $\Theta(n)$, assim: $C(n) = \Theta(n)$

Merge Sort
Análise do algoritmo (III)

- Somando-se as funções $D(n)$ e $C(n)$ tem-se:
  - $\Theta(n) + \Theta(1) \Rightarrow \Theta(n)$
- Adicionando-se esta função ao termo $2T(n/2)$ da etapa de **conquistar** tem-se a recorrência para o tempo de execução do pior caso $T(n)$ da ordenação por intercalação:

$$T(n) = \begin{cases} \Theta(1) & se\ n = 1 \\ 2T(n/2) + \Theta(n) & se\ n > 1 \end{cases}$$

Merge Sort
Análise do algoritmo (IV)

- Reescrevendo a recorrência anterior como:

$$T(n)=\begin{cases} c & se\ n=1 \\ 2T(n/2)+cn & se\ n>1 \end{cases}$$

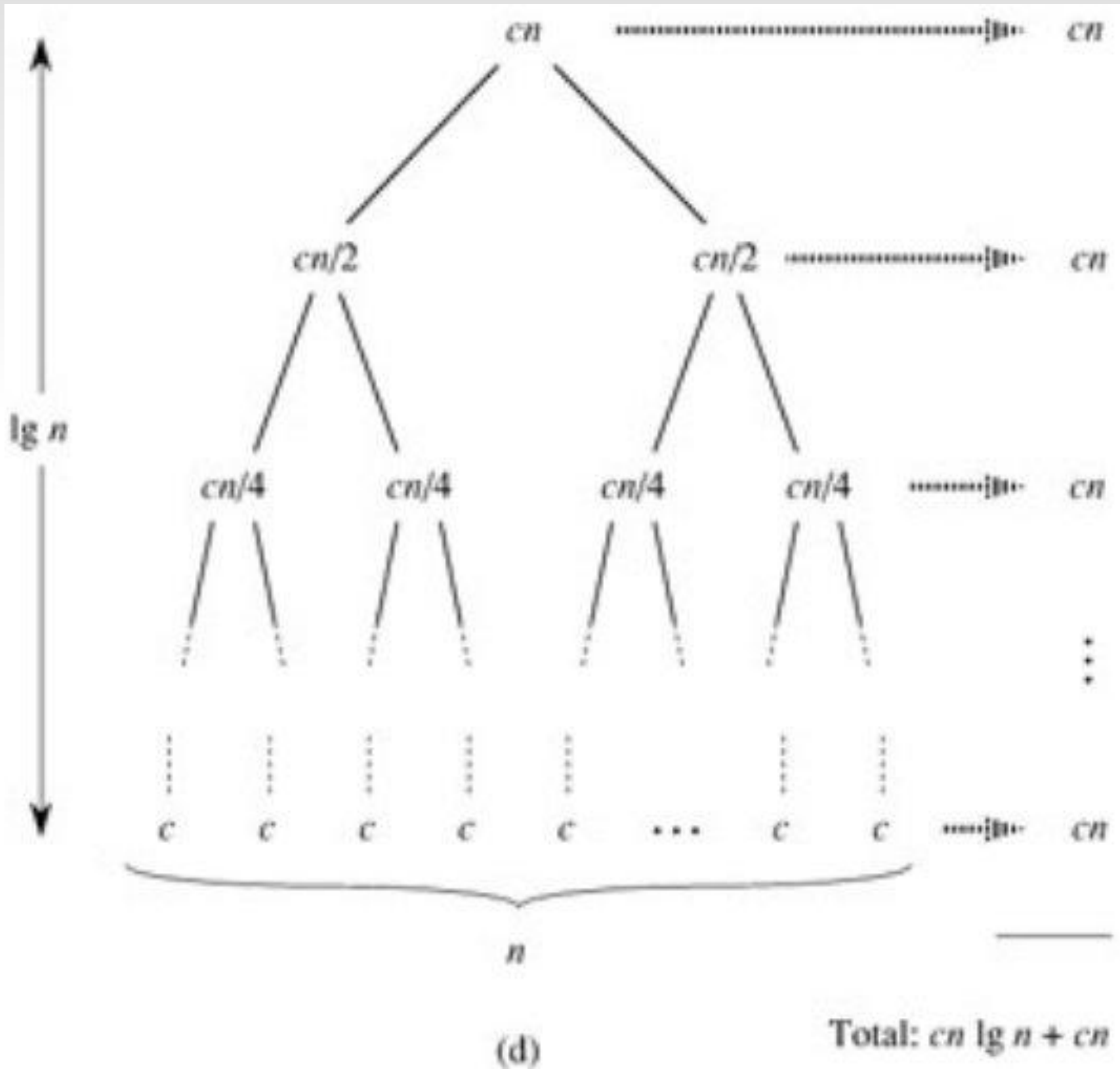
  - onde a constante *c* representa o tempo exigido para resolver problemas de tamanho 1, como também o tempo por elemento do arranjo para as etapas de dividir e combinar

Merge Sort
Análise do algoritmo (V)

- Supondo que n é uma potência exata de 2
- $T(n)$ pode ser **expandido em uma árvore**, por exemplo, representando a recorrência, onde cn é a raiz (custo do nível superior da recursão) e $T(n/2)$ são as duas recorrências menores
  - Cada nó na árvore deve ser expandido até os tamanhos dos problemas se reduzirem a 1 (com o custo c)
  - Somando os custos de cada nível da árvore:
    - Nível superior: cn
    - Próximo nível: $c(n/2) + c(n/2) = cn$
    - Próximo nível: $c(n/4) + c(n/4) + c(n/4) + c(n/4) = cn$

Merge Sort

Árvore de recursão



Merge Sort
Árvore de recursão (II)

- O nível "*i*" abaixo do topo tem 2^i nós, cada qual contribuindo com um custo $c(n/2^i)$, de forma que o i–ésimo nível abaixo do topo tem custo total $2^i c(n/2^i) = cn$
- O número total de níveis da árvore de recursão é $lg\ n + 1$
  - n=1 => só um nível
    - pois: lg1 = 0

Merge Sort
Árvore de recursão (III)

- Para calcular o custo total representado pela recorrência deve-se, simplesmente, somar o custo de todos os níveis
- Há $lg\ n + 1$ níveis, cada qual com o custo cn, o que nos dá o custo total $cn(lg\ n + 1) = cn\ lg\ n + cn$
- Assim: $\boldsymbol{\Theta(n\ lg\ n)}$