CENTRO DE CIÊNCIA E TECNOLOGIA
LABORATÓRIO DE CIÊNCIAS MATEMÁTICAS
UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE

Algoritmos de Ordenação
Mergesort

Disciplina: Estrutura de Dados II

Prof. Fermín Alfredo Tang Montané

Curso: Ciência da Computação

Algoritmo MergeSort

Princípio de Dividir e Conquistar

- O algoritmo MERGESORT(*V, p, r*) aplica o princípio de **divisão e conquista** para realizar a ordenação do arranjo *V*.
- O princípio de Dividir e Conquistar é típico de **algoritmos recursivos**, e compreende três etapas básicas**:**
- **Dividir.-** O problema é dividido em um determinado número de subproblemas.
- **Conquistar.-** Cada subproblema é resolvido de maneira recursiva. Ou seja, dividindo ele novamente em subproblemas e assim sucessivamente até que o tamanho do problema resulte na solução de maneira direta ou trivial.
- **Combinar.-** Utilizar a solução dos subproblemas menores para produzir a solução para um problema maior.

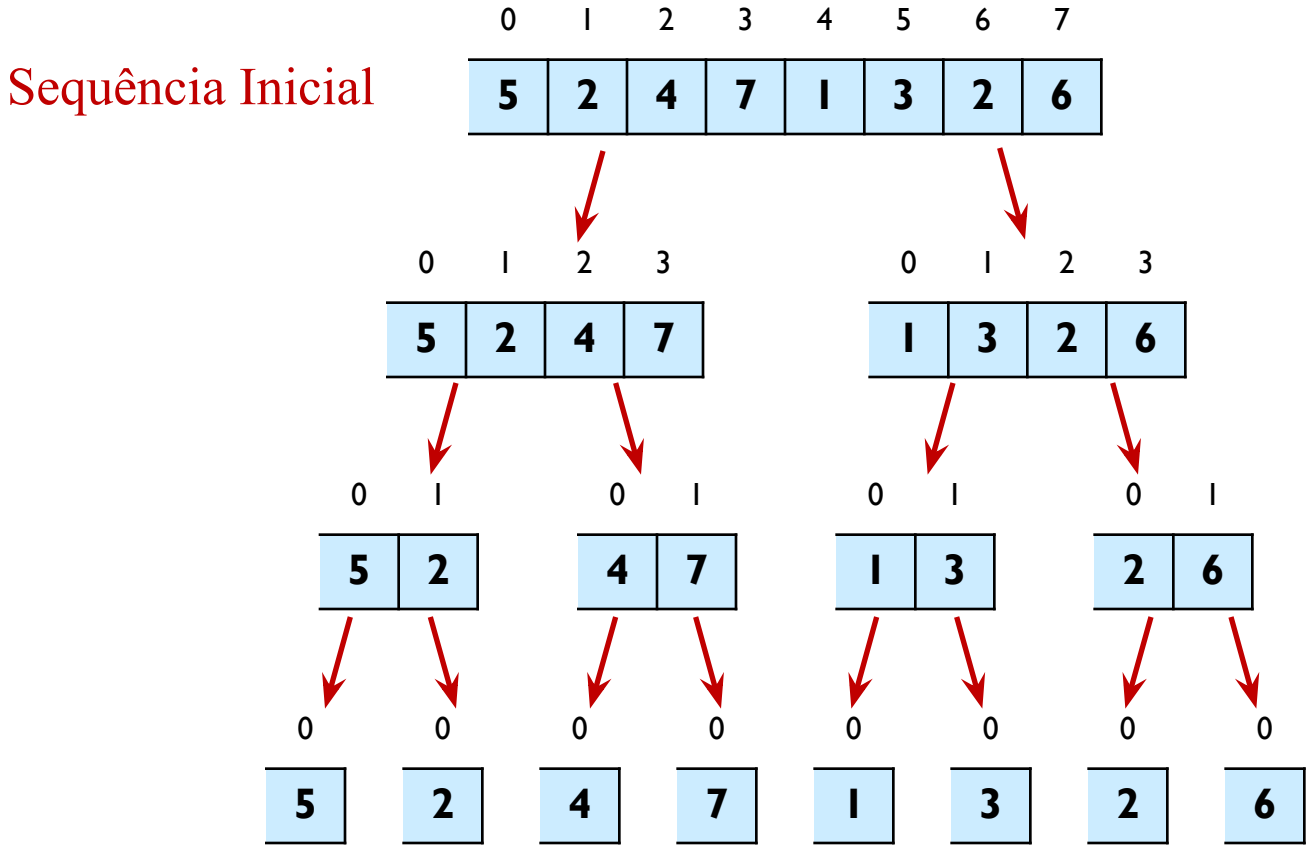
O Procedimento de Intercalação

Ordenação de um Subarranjo

- O procedimento MERGE(*V, p, q, r*), ou **intercalação**, é o procedimento chave do algoritmo Merge*sort*.
- Este é o procedimento realiza a ordenação de um subarranjo do vetor *V*, delimitado pelos índices p e r (extremos esquerdo e direito).
- O algoritmo utiliza o índice intermediário q, que serve para dividir o subarranjo em duas partes.
- O procedimento pressupõe que os subarranjos $V[p]..V[q]$ e $V[q+1]..V[r]$ já se encontram ordenados.

O Principio de Dividir e Conquistar

Dividindo o Problema



O Procedimento de Intercalação

Ordenação de um Subarranjo

- O procedimento compreende os seguintes passos:

```
MERGE (V, p, q, r)                { ordena V[p]..V[r] por intercalação de duas metades }
inicio
  { Divide o subarranjo V em duas metades V[p]..V[q] e V[q + 1]..V[r] }
  n1 ← q – p + 1;   n2 ← r – q;   { nro. de elementos em V[p]..V[q] e V[q + 1]..V[r] }
  para  i ←0 até (n1-1) faça        { copia V[p]..V[q] em L    }
       L[i] ← V [p + i ];
  fim-para
  para  j ← 0 até (n2-1) faça       { copia V[q + 1]..V[r] em R }
       R[j] ← V [q + j+1 ];
  fim-para
  L[n1] ← ∞;  R[n2] ← ∞;            { armazena ∞ no último elemento de L e R }
  { Intercala os elementos  de L e R e copia de volta em V[p]..V[r] }
    CONTINUA ...
```

O Procedimento de Intercalação

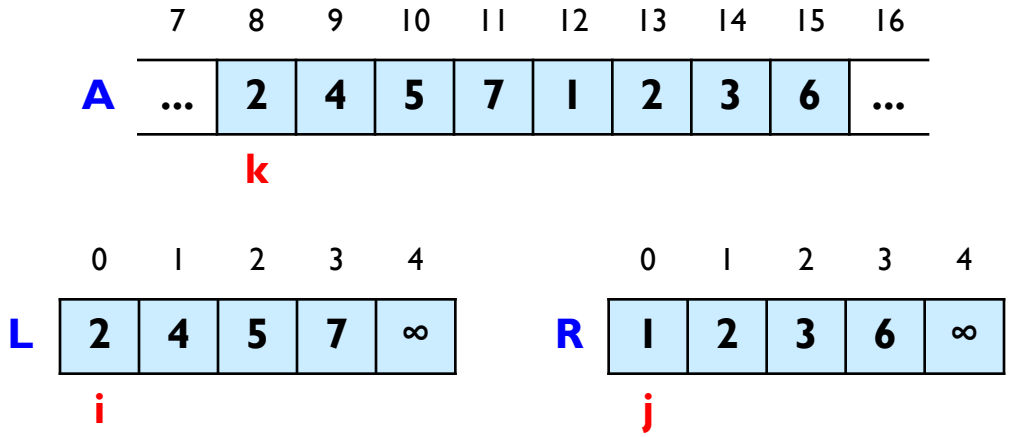
Ordenação de um Subarranjo

CONTINUAÇÃO:

```
{ Intercala os elementos de L e R e copia de volta em V[p]..V[r] }
i ←0; j ←0;                  { inicializa marcadores em L e R }
para k ← p até r faça        { define marcador em V }
    se L[i] ≤ R[j] então     { escolhe o menor elemento de L e R }
        V[k] ← L[ i];        { copia em V }
        i ←i+1;
    senão
        V[k] ← R[ j];        { copia em V }
        j ←j+1;
    fim-se
fim-para
fim
```

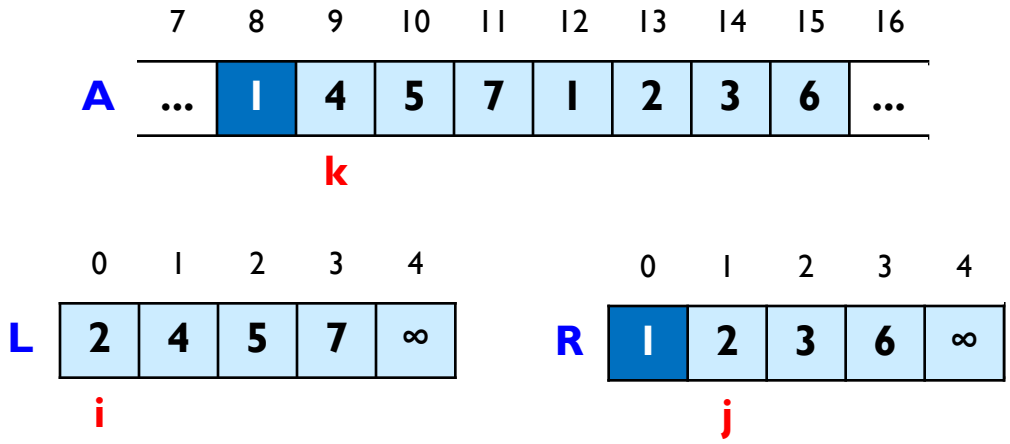
O Procedimento de Intercalação

Ordenação de um Subarranjo



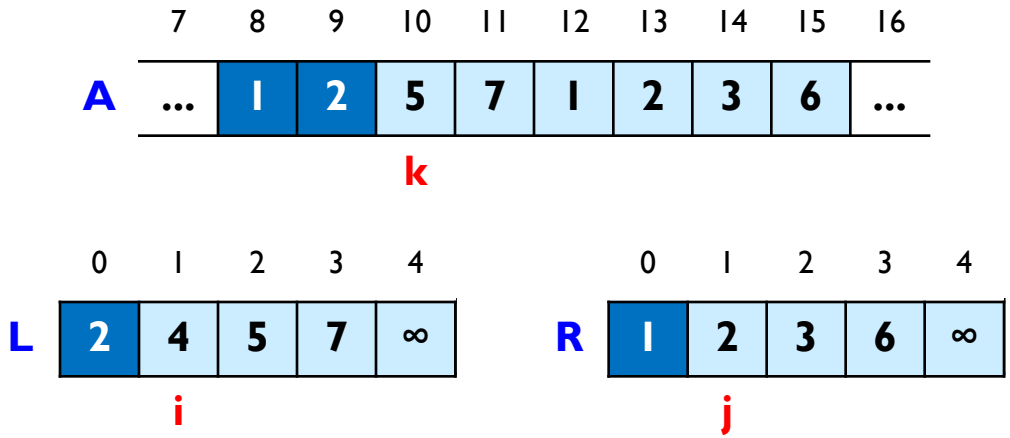
O Procedimento de Intercalação

Ordenação de um Subarranjo



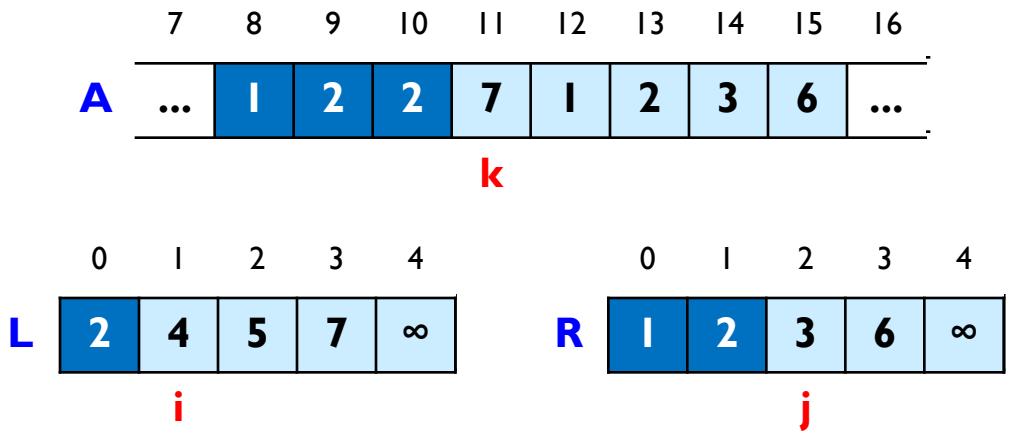
O Procedimento de Intercalação

Ordenação de um Subarranjo



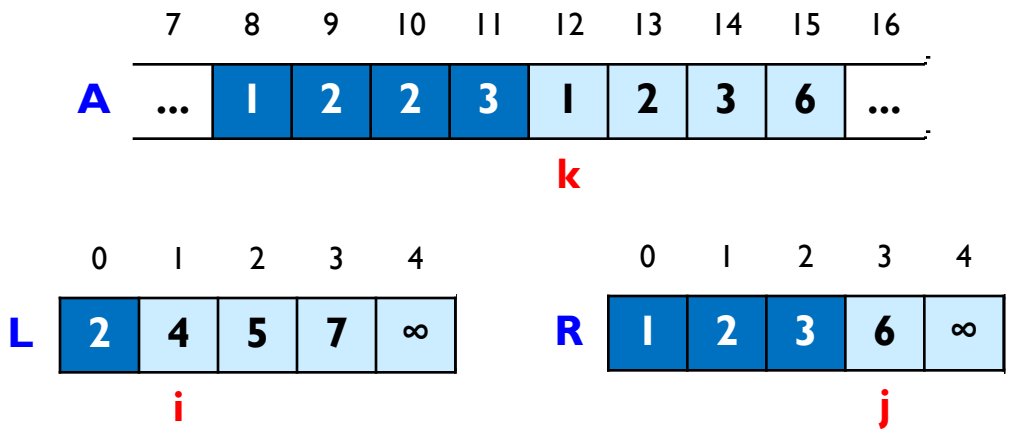
O Procedimento de Intercalação

Ordenação de um Subarranjo



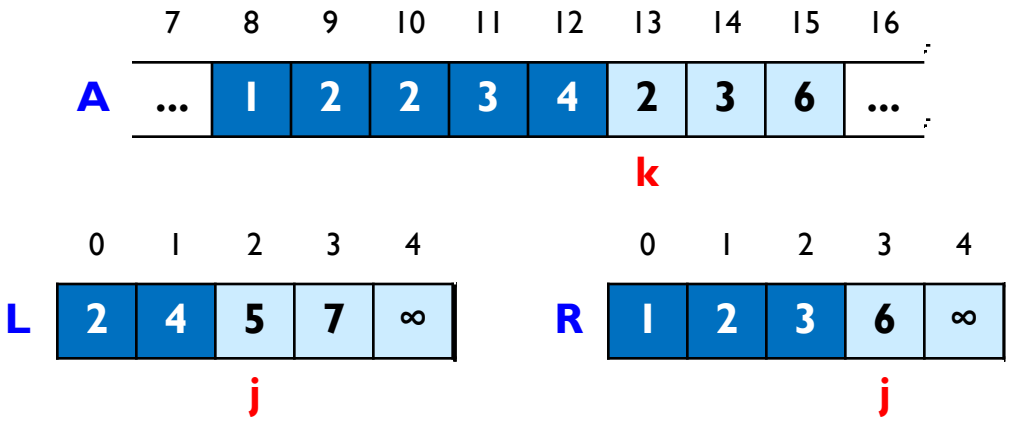
O Procedimento de Intercalação

Ordenação de um Subarranjo



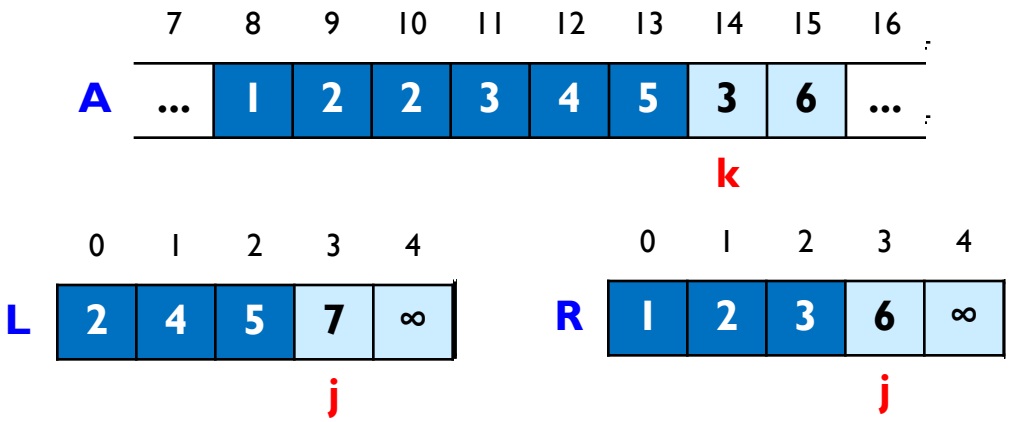
O Procedimento de Intercalação

Ordenação de um Subarranjo



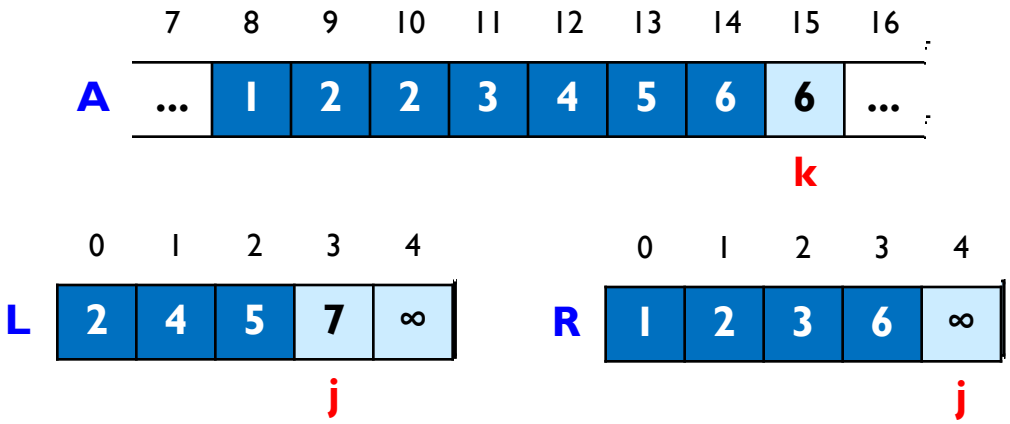
O Procedimento de Intercalação

Ordenação de um Subarranjo



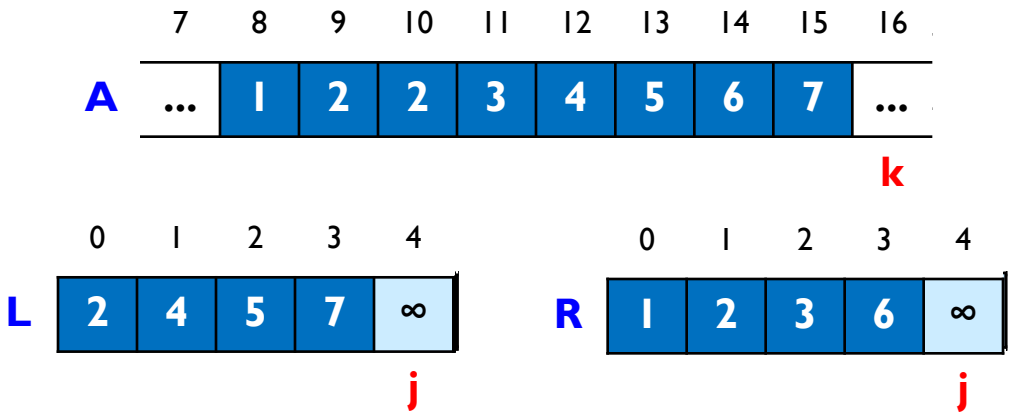
O Procedimento de Intercalação

Ordenação de um Subarranjo



O Procedimento de Intercalação

Ordenação de um Subarranjo



Algoritmos de Ordenação

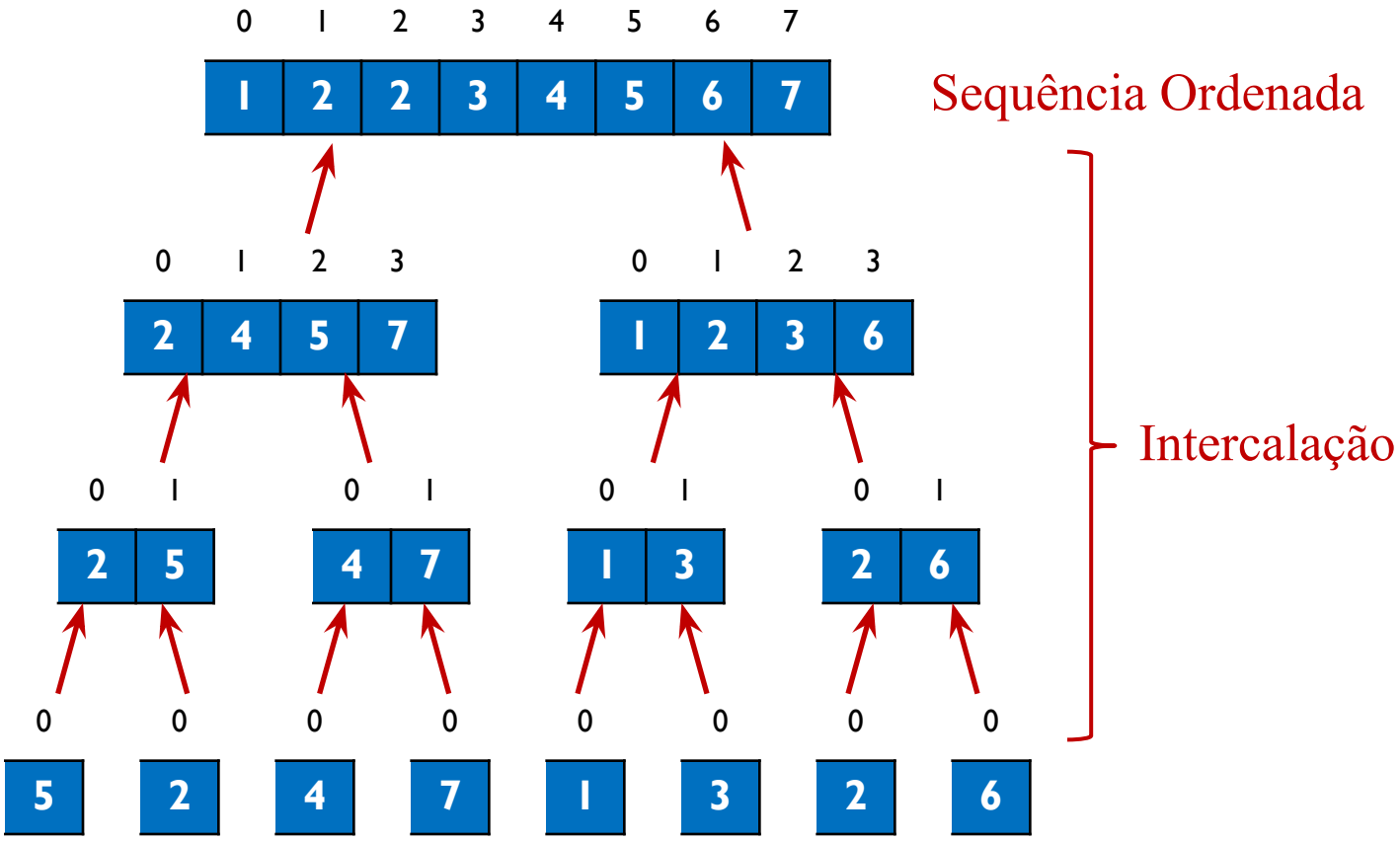
Algoritmo MergeSort

- O algoritmo MERGE-SORT(V, p, r) aplica o princípio de **divisão e conquista** para realizar a ordenação do arranjo V. Para isso **divide** o arranjo $V[p] \ldots V[r]$ em duas metades: $V[p] \ldots V[q]$ e $V[q+1] \ldots V[r]$.
- Ordena cada metade **recursivamente** e calcula o arranjo ordenado V, mediante o procedimento de intercalação MERGE(V, p, q, r) descrito anteriormente.

```
MERGE-SORT (V, p, r)          { ordena o arranjo V[p]..V[r] por intercalação }
inicio
    se (p<r) então
        q ← ⌊(p+r)/2⌋;        { calcula a metade do arranjo V }
        MERGE-SORT (V, p, q);  { ordena a primeira metade de V por intercalação }
        MERGE-SORT (V, q+1, r); { ordena a segunda metade de V por intercalação }
        MERGE (V, p, q, r);    { intercala elementos das duas metades de V }
    fim-se
fim
```

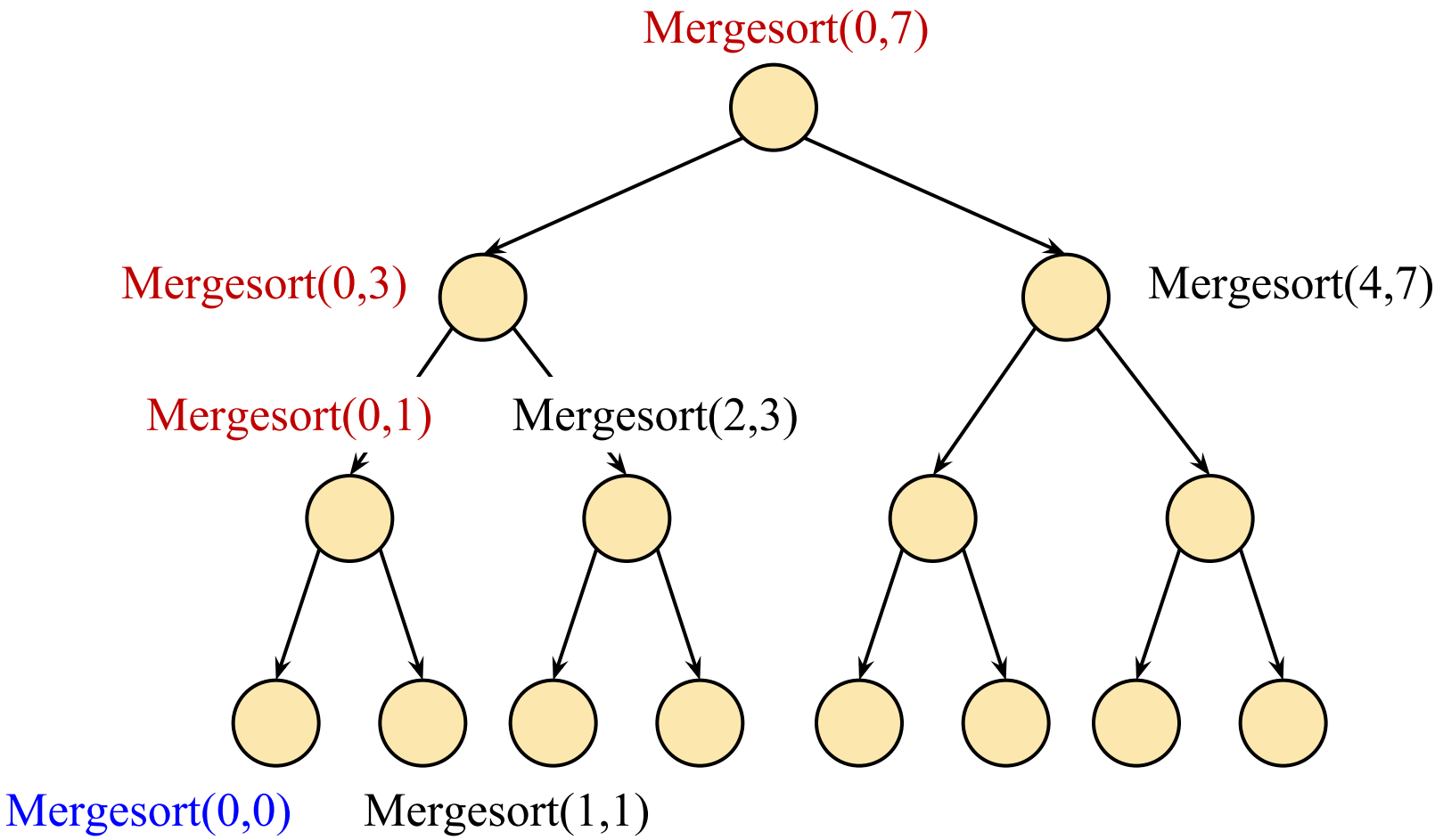
O Principio de Dividir e Conquistar

Combinando a solução dos subproblemas



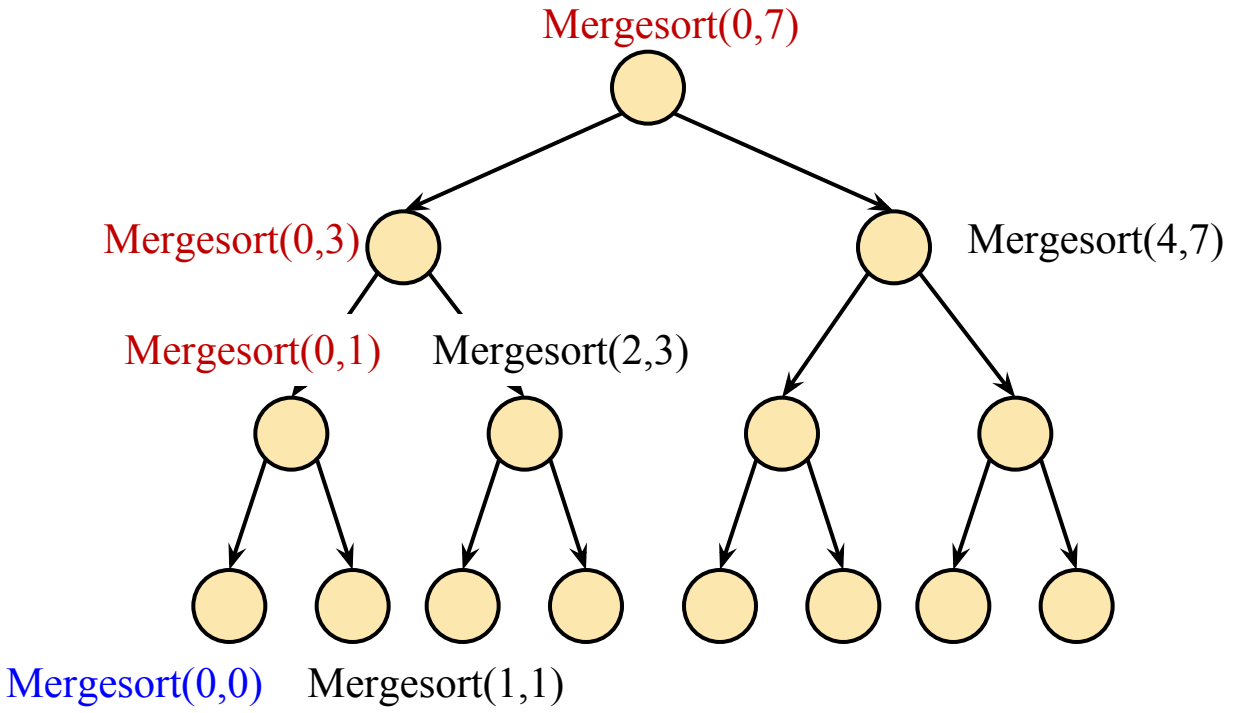
Algoritmo Mergesort

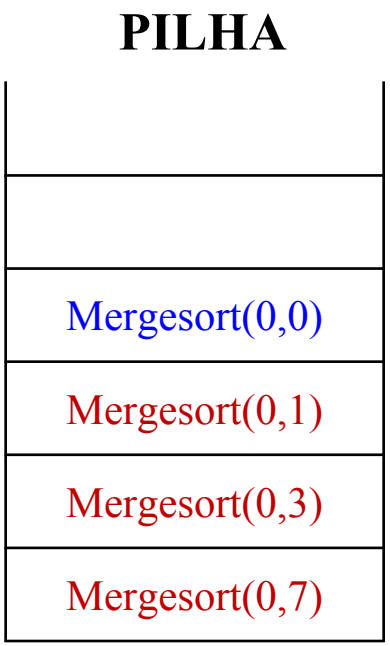
Sequência de Chamadas Recursivas



Algoritmo Mergesort

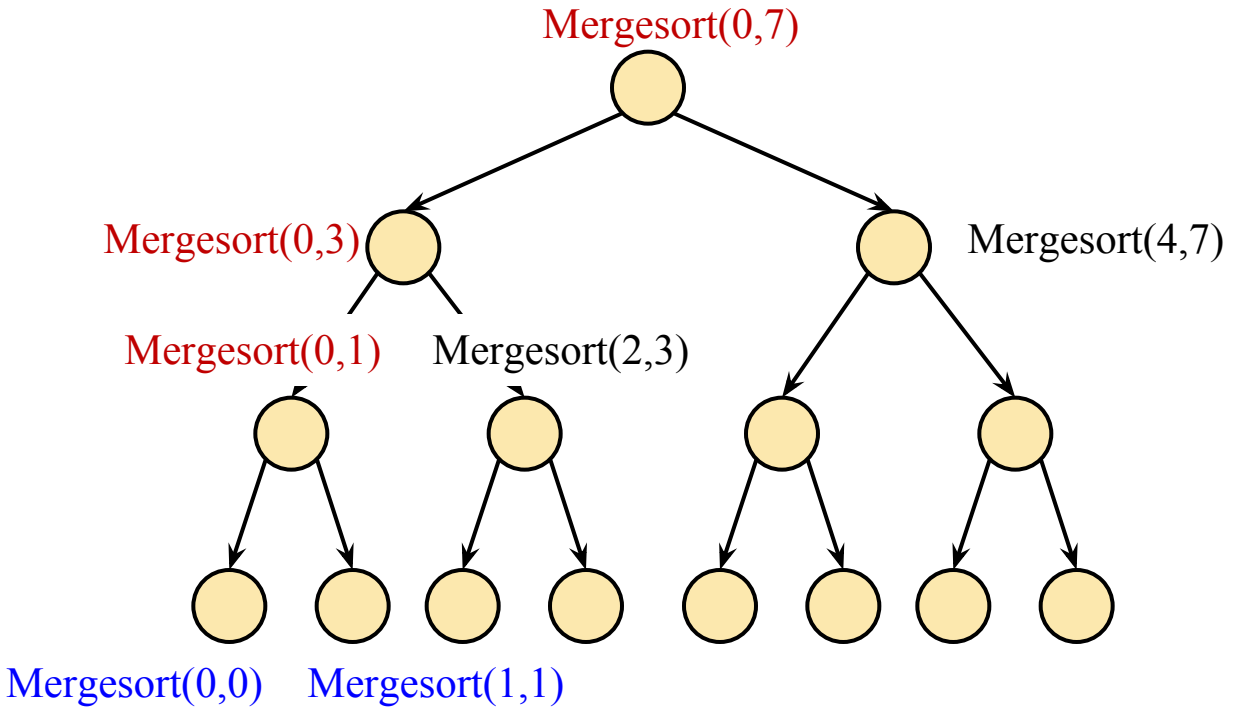
Sequência de Chamadas Recursivas





Algoritmo Mergesort

Sequência de Chamadas Recursivas

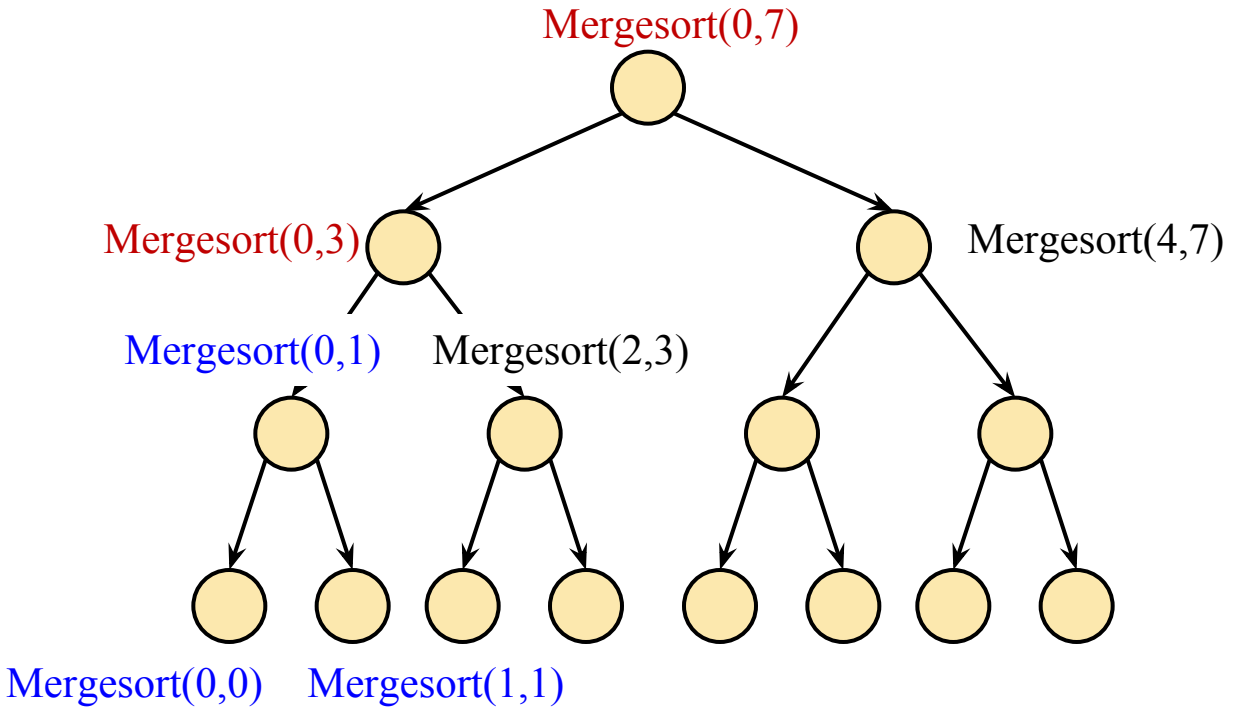


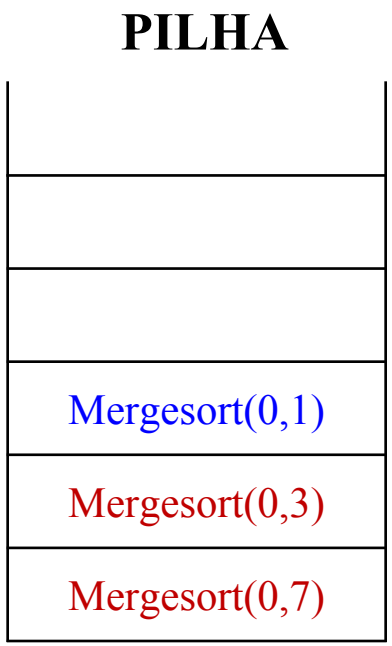
PILHA

Mergesort(1,1)
Mergesort(0,1)
Mergesort(0,3)
Mergesort(0,7)

Algoritmo Mergesort

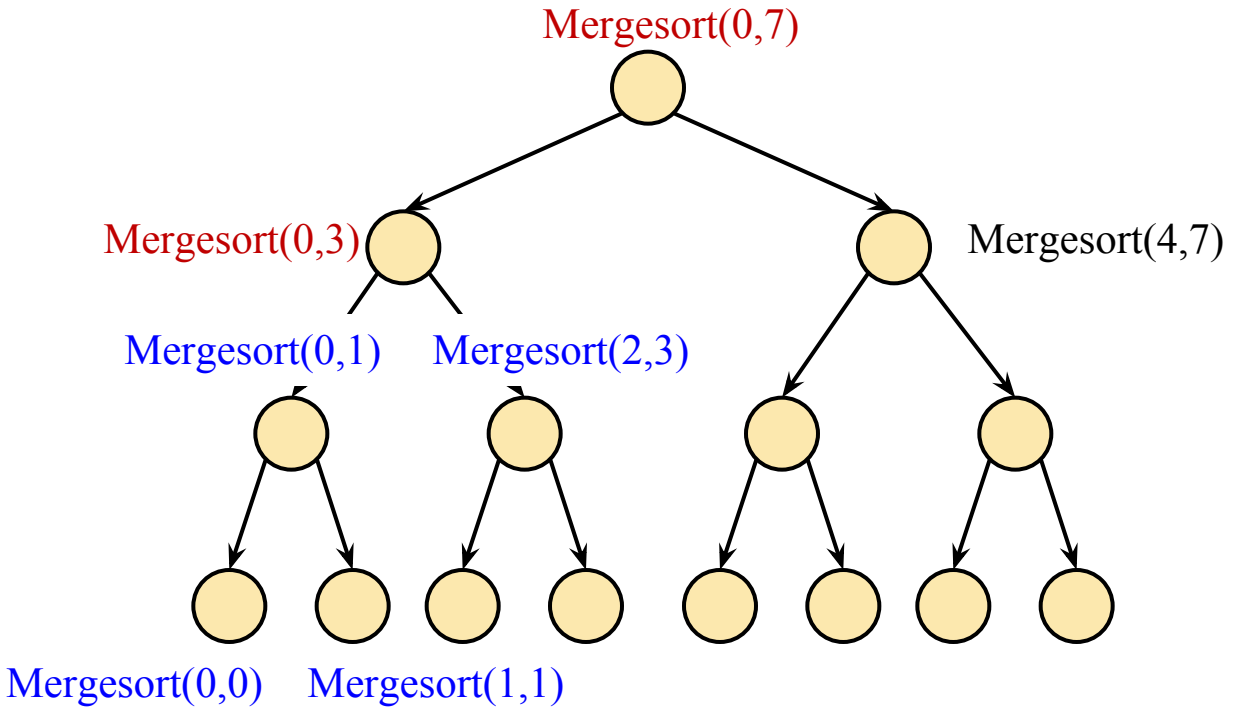
Sequência de Chamadas Recursivas





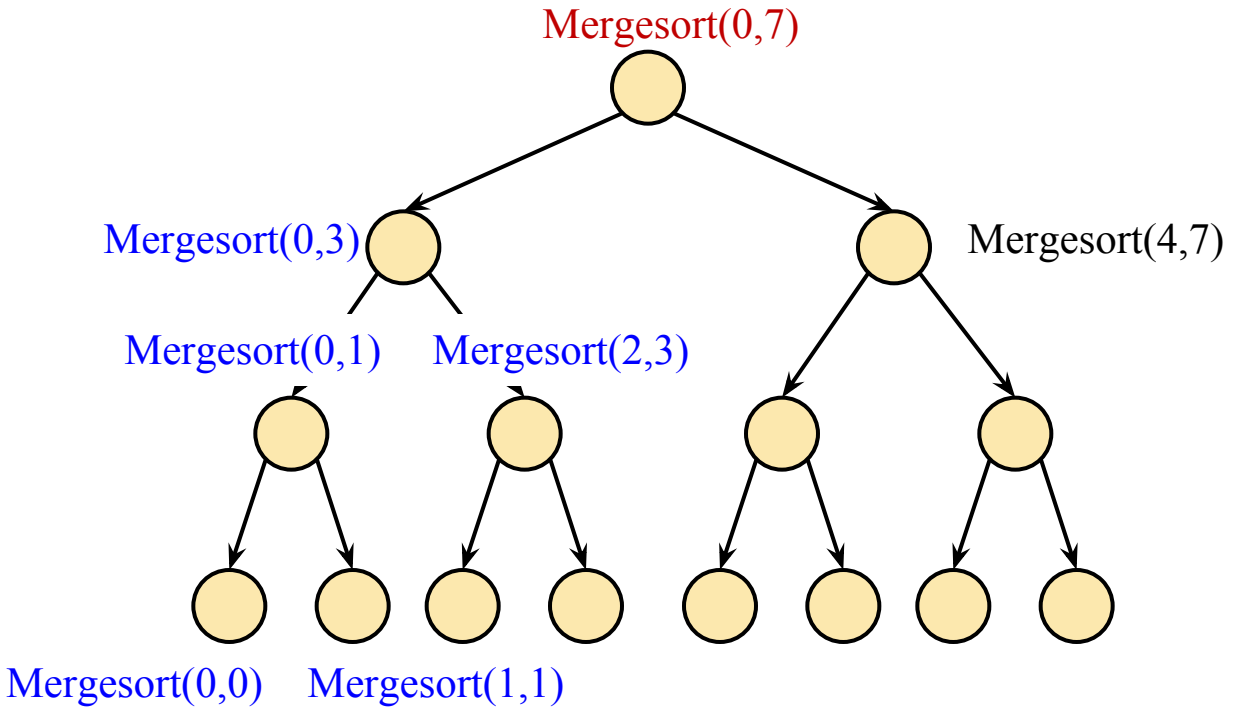
Algoritmo Mergesort

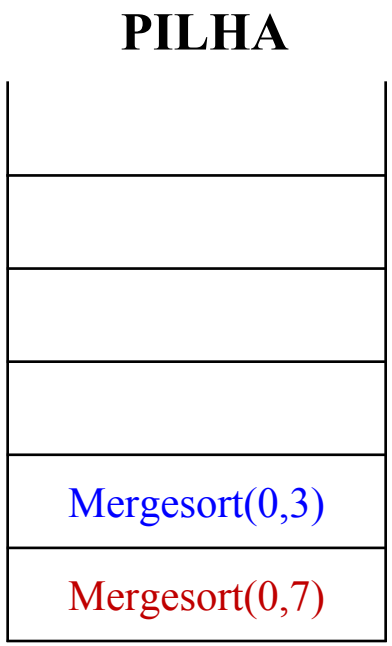
Sequência de Chamadas Recursivas



Algoritmo Mergesort

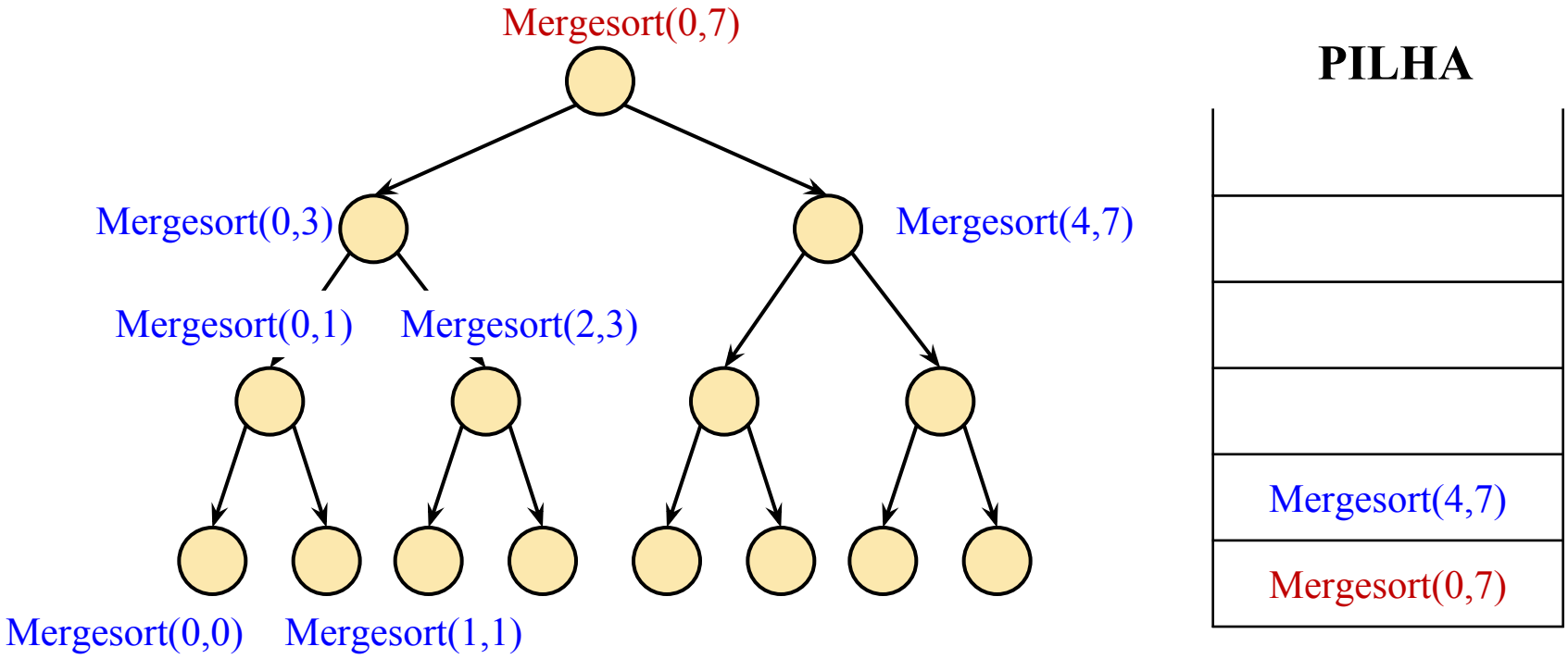
Sequência de Chamadas Recursivas





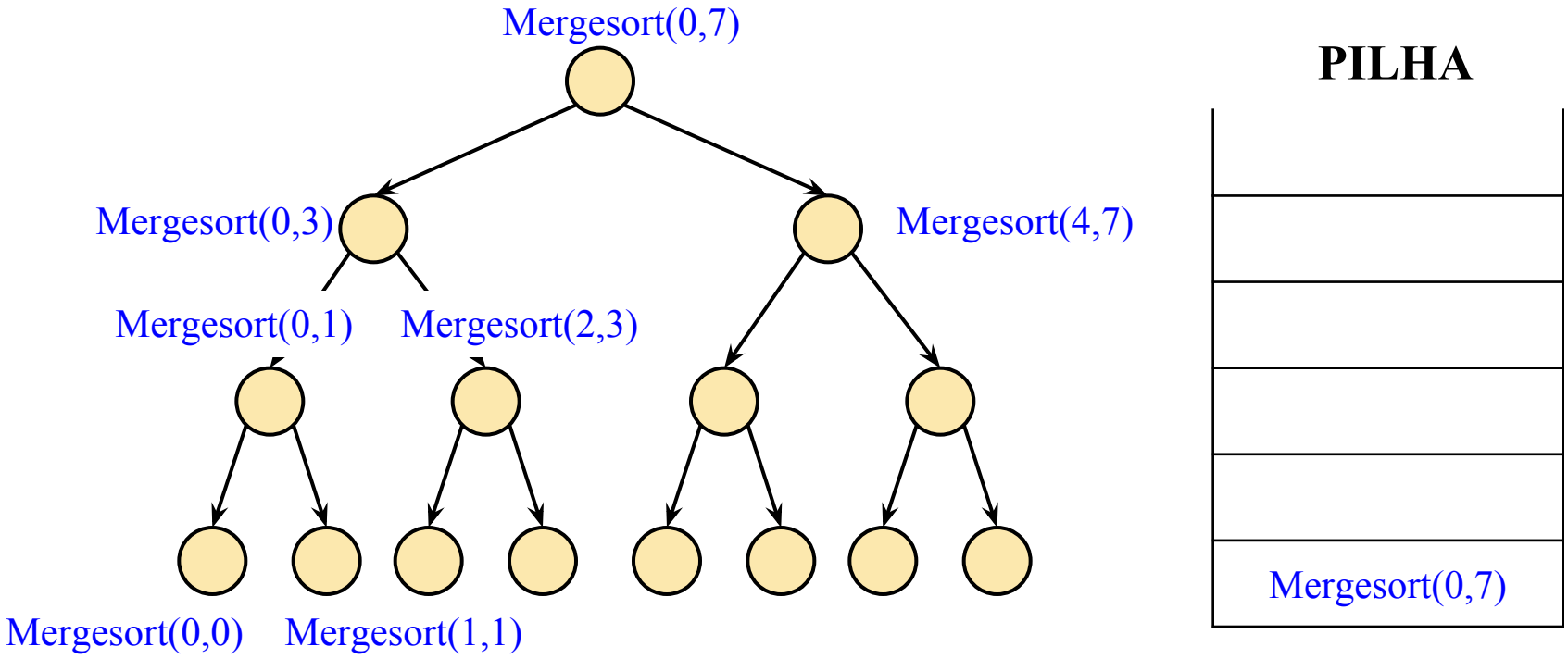
Algoritmo Mergesort

Sequência de Chamadas Recursivas



Algoritmo Mergesort

Sequência de Chamadas Recursivas



Algoritmo MergeSort

Análise de Complexidade

- A abordagem de **dividir e conquistar** consiste em:
  - Desmembrar o problema em vários subproblemas de menor tamanho **(divisão)** de maneira que eles possam ser resolvidos **(conquista)** com maior facilidade.
  - Os subproblemas devem ter a mesma estrutura do problema original.

- Para analisar a complexidade do algoritmo, devemos representar o problema mediante uma relação de recorrência.

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

- Considerando a complexidade da intercalação temos a seguinte relação:

$$T(n) = \begin{cases} O(1), & \text{se } n=1 \ \text{(Intercalação)} \\ 2.T(n/2) + O(n), & \text{se } n>1 \ \text{(Divisão + Intercalação)} \end{cases}$$

- Ainda precisamos eliminar a recorrência.

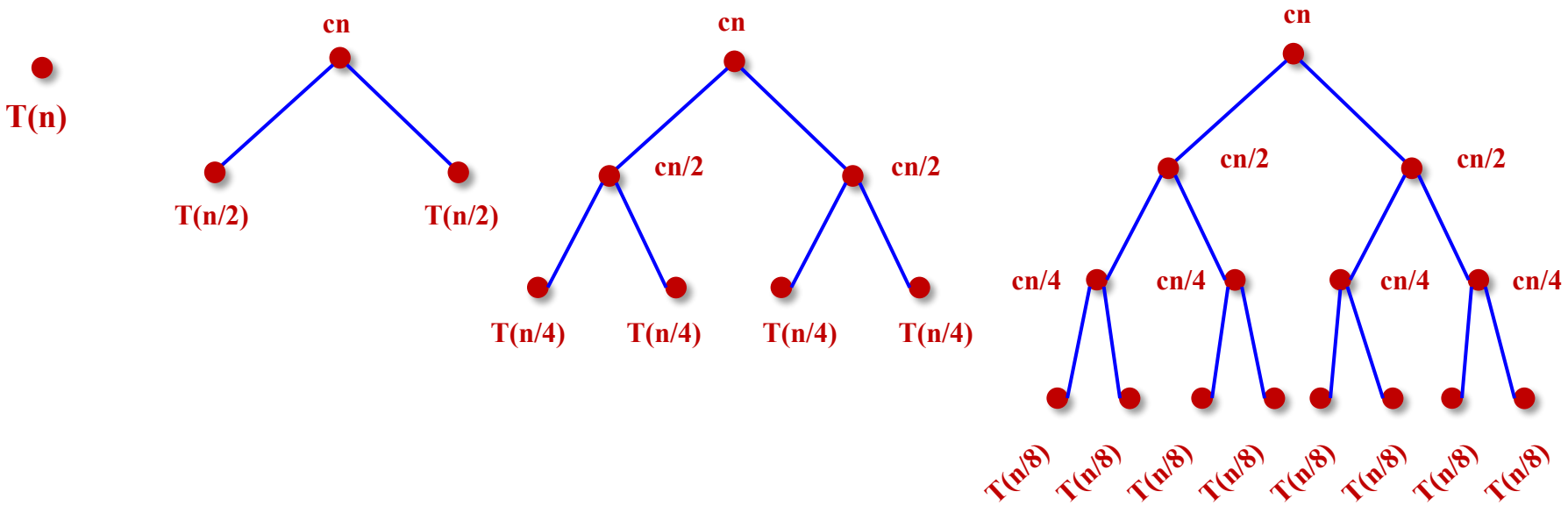
Algoritmo MergeSort

Análise de Complexidade – Usando uma árvore

- Utilizando a relação de recorrência:

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

- Podemos calcular o custo computacional do algoritmo para um problema de tamanho n, considerando o número de subproblemas gerados e as operações envolvidas.



Algoritmo MergeSort

Análise de Complexidade – Usando uma árvore

- Considere que o tamanho do problema n é uma potência de 2. Por exemplo, considere que $n = 8$ elementos.
- Interessa considerar a altura da árvore (ou número de níveis) para calcular o número operações por subproblema.

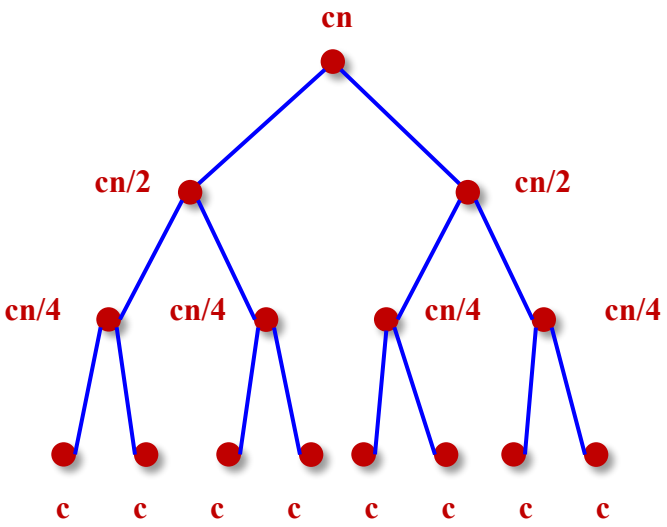
Subproblemas:

Nível 0: 2^0 **=1 nós**

Nível 1: 2^1 **= 2 nós**

Nível 2: 2^2 **= 4 nós**

Nível 3: 2^3 **= 8 nós**



$$H = (\log_2 n) + 1$$

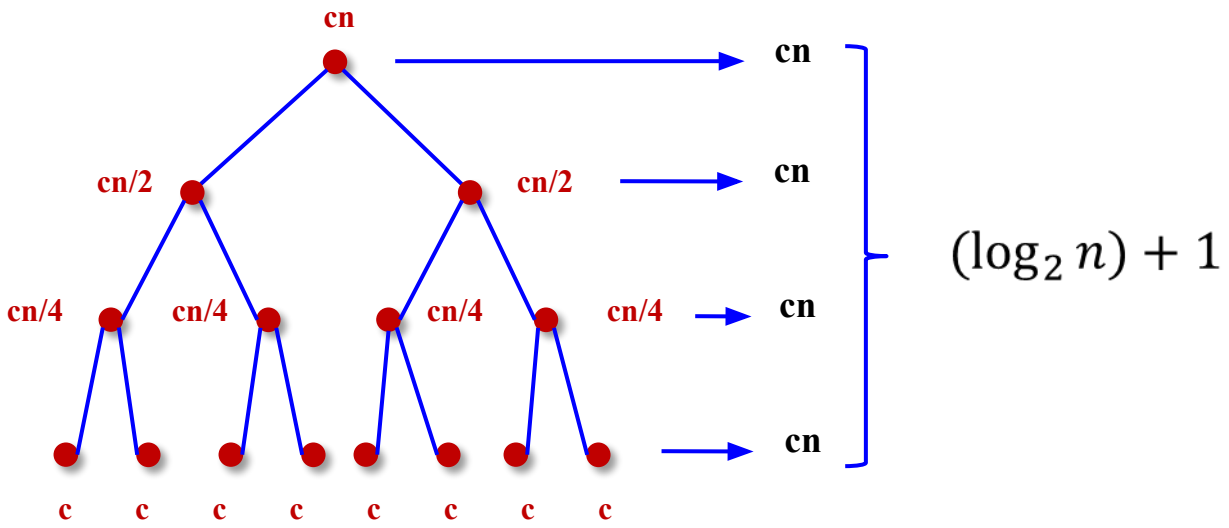
$$H = \log_2 8 + 1 = 3 + 1 = 4$$

$n = 8 \Rightarrow T(n/8) = T(1) = c$

Algoritmo MergeSort

Análise de Complexidade – Usando uma árvore

- Se calculamos o custo por nível, observamos que temos um custo computacional de cn operações. Logo, multiplicamos isso pelo número de níveis (ou altura da árvore) que é $(\log n) + 1$ para obter complexidade de $O(n \log n)$.



$$cn \times [(\log_2 n) + 1]$$

- Onde:

$$cn \times [(\log_2 n) + 1] = cn(\log_2 n) + cn = O(n \log n)$$

Algoritmo MergeSort

Análise de Complexidade – Método de Desdobramento

- **Método do Desdobramento:**
- Consiste em aplicar repetidamente a função de recorrência no termo de menor ordem, fazendo substituições sucessivas até:
  - atingir o caso base, ou
  - conseguir identificar a forma geral

$$T(n) = \begin{cases} c, & \text{se } n=1 \quad \text{(Intercalação)} \\ 2.T(n/2) + cn, & \text{se } n>1 \quad \text{(Divisão + Intercalação)} \end{cases}$$

- Para n>0:

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

- Substituindo na função de recorrência T(n/2), temos:

$$T(n) = 2.(2.T(n/4)+c.n/2) + cn$$

Algoritmo MergeSort

Análise de Complexidade – Método de Desdobramento

- Para n>0:

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

- Substituindo na função de recorrência T(n/2), temos:

$$T(n) = 2.(2.T(n/4)+c.n/2) + cn \qquad T(n) = 2^2.T(n/4) + 2cn$$

- Substituindo na função de recorrência T(n/4), temos:

$$T(n) = 2^2.(2.T(n/8)+c.n/4) + 2cn \qquad T(n) = 2^3.T(n/8)+ 3cn$$

- Generalizando para k<n, temos:

$$T(n) = 2^k.T(n/2^k)+ kcn$$

- Considerando k, tal que $n=2^k$, temos:

$$T(n) = n.T(1)+ (\log n).cn \qquad T(n) = c.n + c.n\ (\log n) \qquad T(n) = O(n \log n)$$

Algoritmo MergeSort
Complexidade do Algoritmo

- A complexidade de tempo de **pior caso** do algoritmo Mergesort é comparada aos outros algoritmos descritos previamente.
- A principal desvantagem do algoritmo Mergesort é que ela requer um vetor adicional de igual tamanho ao que está sendo ordenado.

Tamanho	Método de Inserção	Método Shellsort	Método MergeSort
n	$O(n^2)$	Est. Pessimista $O(n^{3/2})$	$O(n \log n)$
10	100	32	33
100	10.000	1.000	664
1.000	1.000.000	31.623	9.966
10.000	100.000.000	1.000.000	132.877

Ordenação Mergesort

Complexidade do Algoritmo

- A complexidade da ordenação independe dos valores contidos no vetor:
  - **Melhor Caso**.- A árvore fica balanceada, a complexidade depende da altura da árvore. Se o vetor estiver ordenado, o número de comparações cai pela metade, mais o numero de copias determina a complexidade. **Executa em** $O(nlog\,n)$.
  - **Pior caso**.- A árvore fica ligeiramente desbalanceada se o número de elementos não é potencia de 2. Se o vetor estiver invertido, o número de comparações poderá ser máximo, mas ainda o número de copias determinada a complexidade. **Executa em tempo ligeiramente superior a** $\mathrm{O}(\mathrm{nlog}\,n)$.

Algoritmo MergeSort

Complexidade do Algoritmo – Comparações

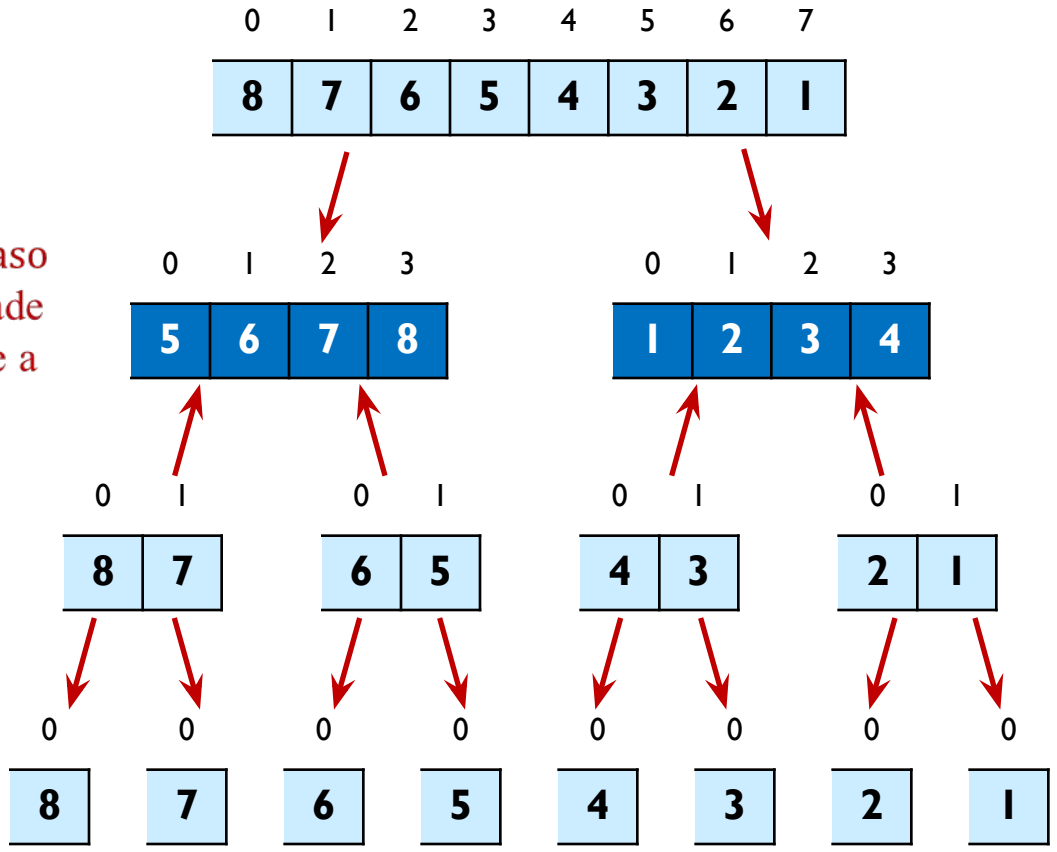
- O número de comparações depende exclusivamente do processo de intercalação. Se consideramos duas metades que totalizam n elementos. **No pior caso,** para intercalar as duas metades são necessárias $(n-1)$ comparações. **No melhor caso**, para intercalar as duas metades são necessárias $(n/2)$ comparações.

n	Elementos Intercalados *n log n*	Comp. Máx ?	Comp. Min ?
2	2	1	1
4	8	5	4
8	24	17	12
16	64	49	32
32	160	129	80
64	384	321	192
128	896	769	448

O Principio de Dividir e Conquistar

Exemplo – Número de Comparações

Melhor Caso
Uma metade maior que a outra



$1 \times 4 = 4\ comparações$

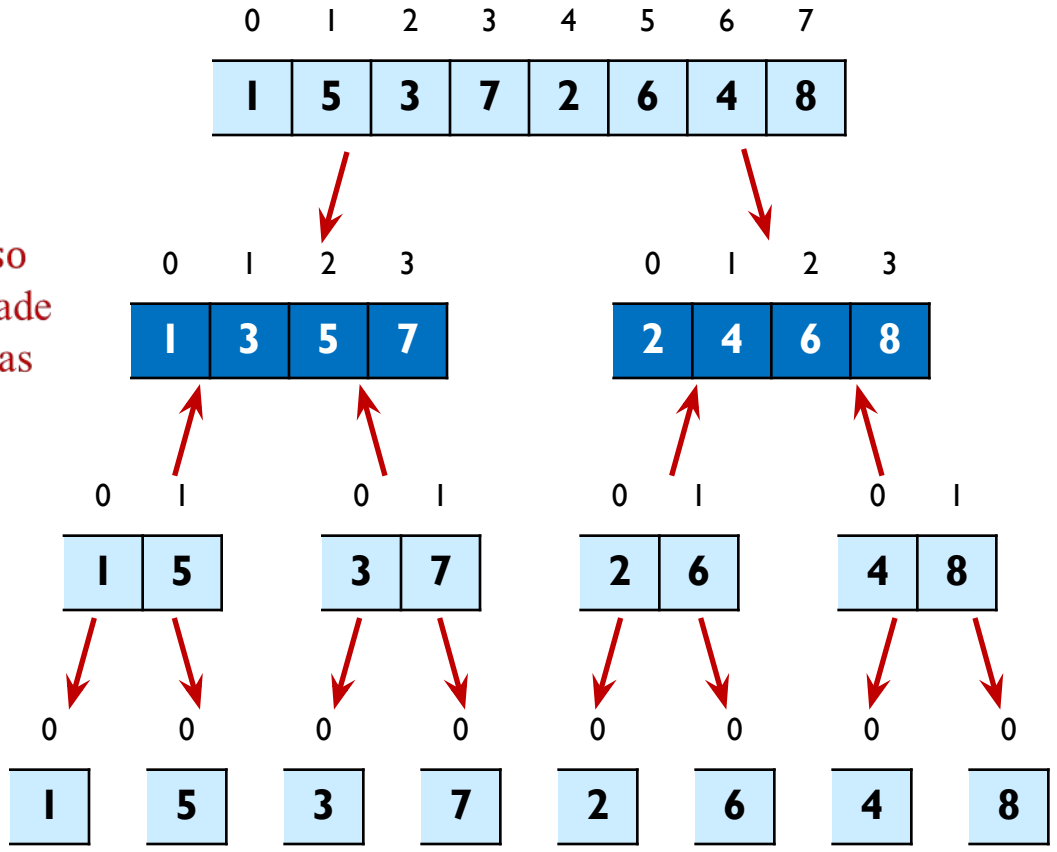
$2 \times 2 = 4\ comparações$

$4 \times 1 = 4\ comparações$

$Total\colon\ 12\ comparações$

O Principio de Dividir e Conquistar

Exemplo – Número de Comparações



Pior Caso
Duas metade misturadas

$1 \times 7 = 7\ comparações$

$2 \times 3 = 6\ comparações$

$4 \times 1 = 4\ comparações$

$Total\colon\ 17\ comparações$

Algoritmo MergeSort

Complexidade do Algoritmo – Comparações e copias

- O número de elementos copiados é $n \log n$. Os elementos de cada subproblema são copiados 2 vezes, primeiro no vetor auxiliar, depois no vetor original.

n	Elementos Intercalados *n log n*	Comp. Máx	Comp. Min	Copias *2(n log n)*
2	2	1	1	4
4	8	5	4	16
8	24	17	12	48
16	64	49	32	128
32	160	129	80	320
64	384	321	192	768
128	896	769	448	1792

Referências

- Thomas **Cormen**, Charles **Leiserson,** et al.**.** Algoritmos. Teoria e Prática. 2ª Edição. 2002.
- Robert **Lafore**. Estruturas de Dados e Algoritmos em Java. Editora Ciencia Moderna. 2ª Edição. 2004.