Universidade Federal do ABC
Centro de Matemática, Computação e Cognição

# Algoritmos para Ordenação

Monael Pinheiro Ribeiro, D.Sc.

# Algoritmos Estudados

- Bubble Sort
  - Consumo de Tempo no Pior Caso: $O(n^2)$
  - Consumo de Tempo no Melhor Caso: $O(n^2)$
- Selection Sort
  - Consumo de Tempo no Pior Caso: $O(n^2)$
  - Consumo de Tempo no Melhor Caso: $O(n^2)$
- Insertion Sort
  - Consumo de Tempo no Pior Caso: $O(n^2)$
  - Consumo de Tempo no Melhor Caso: $O(n)$

# Lower Bound do Problema de Ordenação

- Até agora, apresentamos algoritmos que ordenam *n* números em tempo $O(n^2)$. Por enquanto, esse é o nosso *upper bound* para o problema da ordenação baseado em comparações.

# Lower Bound do Problema de Ordenação

- Até agora, apresentamos algoritmos que ordenam $n$ números em tempo $O(n^2)$. Por enquanto, esse é o nosso *upper bound* para o problema da ordenação baseado em comparações.
- Seria possível calcular um *lower bound* para esse problema?

# Lower Bound do Problema de Ordenação

- Até agora, apresentamos algoritmos que ordenam *n* números em tempo $O(n^2)$. Por enquanto, esse é o nosso *upper bound* para o problema da ordenação baseado em comparações.
- Seria possível calcular um *lower bound* para esse problema?
- Em outras palavras, desejamos encontrar um limite inferior teórico para esse problema, isto é, a mínima complexidade de tempo de quaisquer de suas resoluções algorítmicas.

# Árvore de Comparações

- Qualquer algoritmo de ordenação baseado em comparações pode ser representado em uma árvore binária.

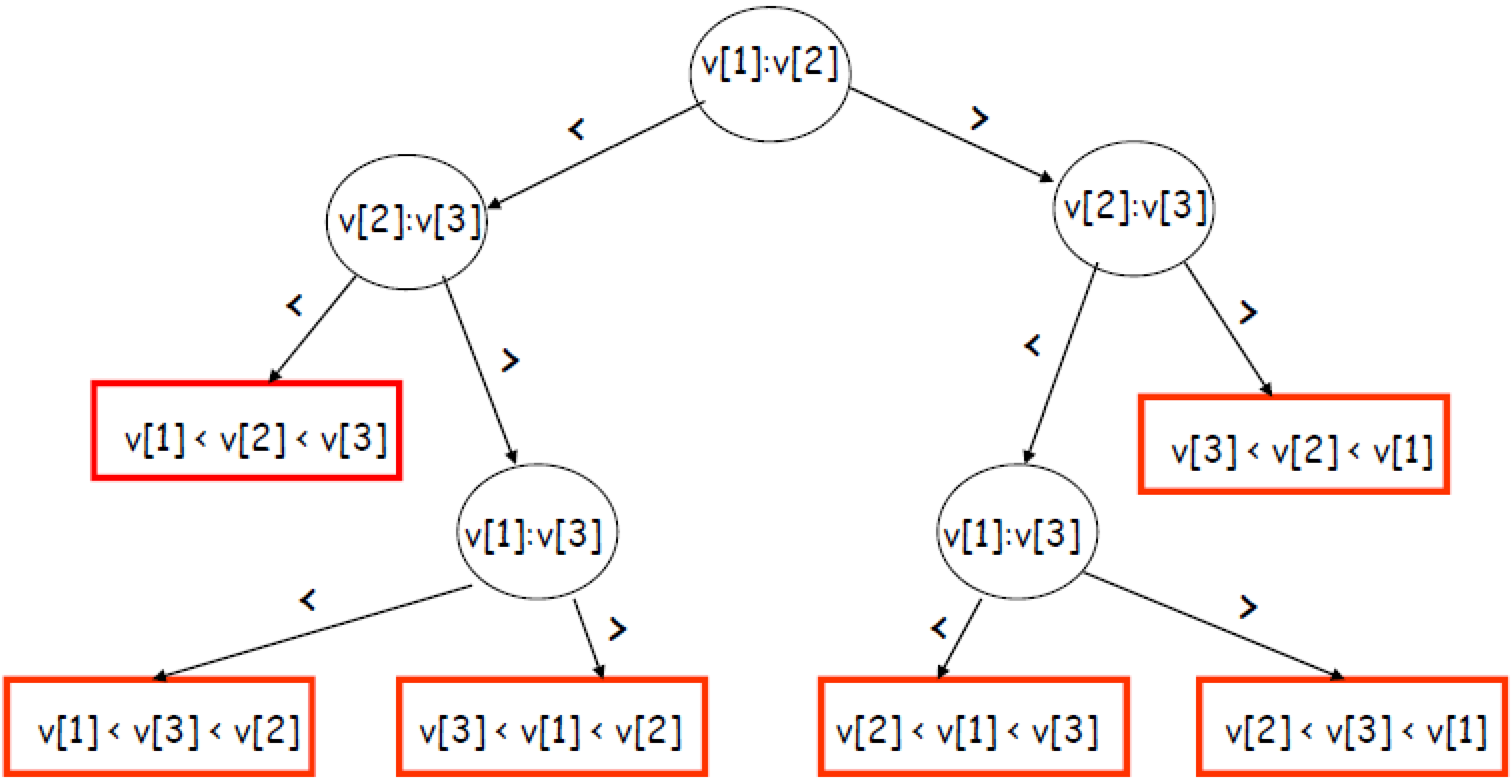
# Árvore de Comparações

- Qualquer algoritmo de ordenação baseado em comparações pode ser representado em uma árvore binária.
- Na raiz fica a primeira comparação realizada entre dois elementos; nos filhos, as comparações subsequentes.

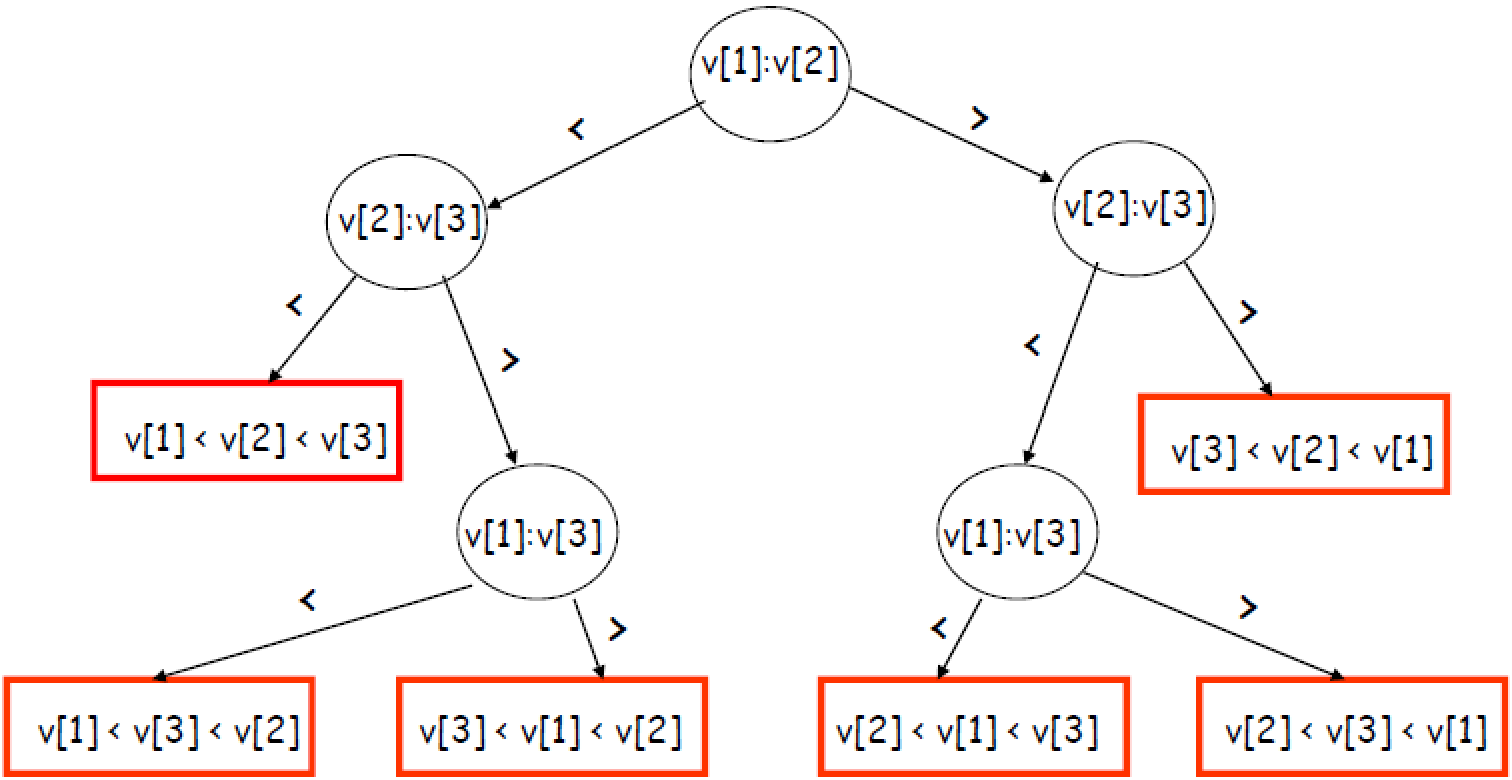
# Árvore de Comparações

- Qualquer algoritmo de ordenação baseado em comparações pode ser representado em uma árvore binária.
- Na raiz fica a primeira comparação realizada entre dois elementos; nos filhos, as comparações subsequentes.
- Assim, as folhas dessa árvore representam as possíveis soluções do problema.

# Árvore de Comparação (n=3)



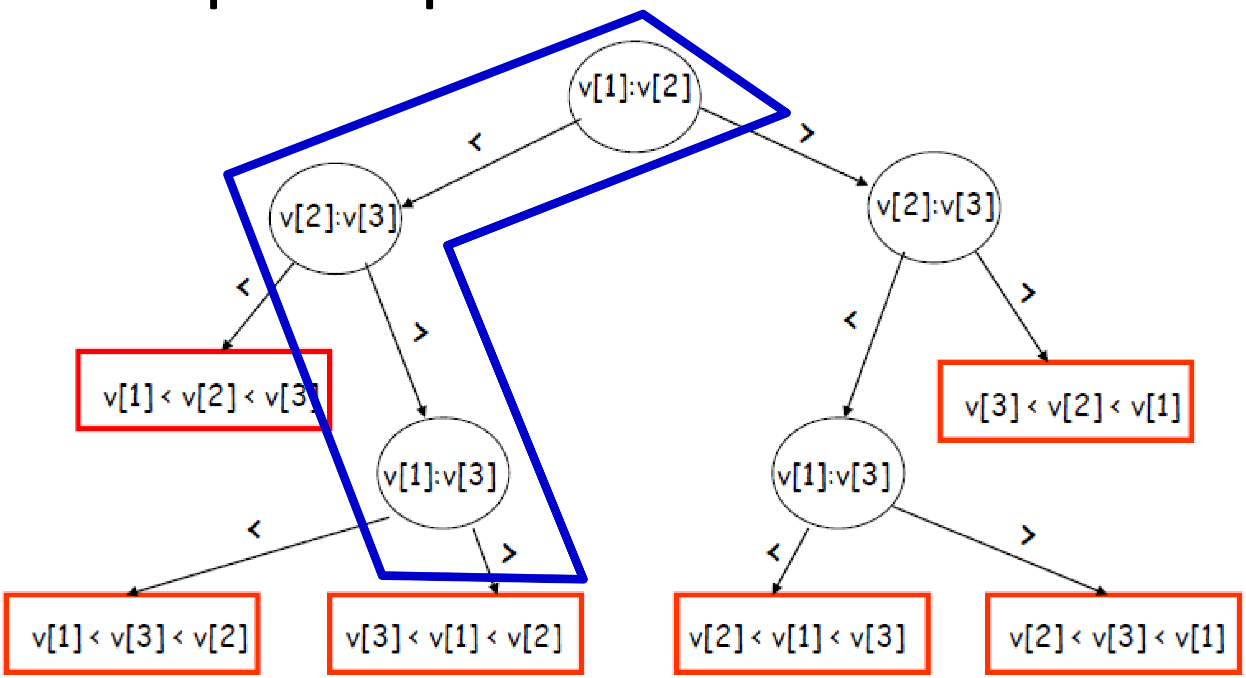
# Árvore de Comparação (n=3)




Como estamos ordenando 3 elementos, há 3! possíveis resultados.

# Árvore de Comparação

- A altura h da árvore é o número máximo de comparações que o algoritmo realiza, ou seja, o seu tempo de pior caso



# Generalização

- Na ordenação de *n* elementos, há então *n!* possíveis resultados, que correspondem às permutações desses elementos.

# Generalização

- Na ordenação de *n* elementos, há então *n!* possíveis resultados, que correspondem às permutações desses elementos.
- Portanto, qualquer árvore binária de comparações terá no mínimo *n!* folhas.

# Generalização

- Na ordenação de *n* elementos, há então *n!* possíveis resultados, que correspondem às permutações desses elementos.
- Portanto, qualquer árvore binária de comparações terá no mínimo *n!* folhas.
- A árvore mínima de comparações tem exatamente *n!* folhas.

# Generalização

- Na ordenação de *n* elementos, há então *n!* possíveis resultados, que correspondem às permutações desses elementos.
- Portanto, qualquer árvore binária de comparações terá no mínimo *n!* folhas.
- A árvore mínima de comparações tem exatamente *n!* folhas.
- Supondo que a altura dessa árvore seja *h*, então *LB(n) = h*, onde *LB(n)* é o *lower bound* de tempo para a ordenação de *n* elementos.

# Generalização

- Sabemos que a quantidade de folhas de uma árvore binária de altura $h$ é $\leq 2^h$.
- Portando, $n! \leq 2^h$.
- Ou seja, $h \geq \log_2 n!$
- Conclui-se que $\underline{LB(n) \geq \log_2 n!}$

# Aproximação de Stirling

- O valor numérico de $n!$ pode ser calculado por multiplicação repetida se $n$ não for grande demais. É isto que as calculadoras fazem. O maior fatorial, que a maioria das calculadoras suportam é 69!, porque $70! > 10^{100}$.
- Quando $n$ é grande demais, $n!$ pode ser calculado com uma boa precisão usando a **aproximação de Stirling**:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^{n}$$

# Cálculo do Lower Bound

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

# Cálculo do Lower Bound

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$n! \approx (2\pi n)^{\frac{1}{2}} n^n e^{-n}$$

# Cálculo do Lower Bound

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$n! \approx (2\pi n)^{\frac{1}{2}} n^n e^{-n}$$

$$\log_2 n! \approx \log_2 (2\pi)^{\frac{1}{2}} + \log_2 (n)^{\frac{1}{2}} + \log_2 n^n + \log_2 e^{-n}$$

# Cálculo do Lower Bound

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$n! \approx (2\pi n)^{\frac{1}{2}} n^n e^{-n}$$

$$\log_2 n! \approx \log_2 (2\pi)^{\frac{1}{2}} + \log_2 (n)^{\frac{1}{2}} + \log_2 n^n + \log_2 e^{-n}$$

$$\log_2 n! \approx \left(\frac{\log_2(2\pi)}{2}\right) + \left(\frac{\log_2(n)}{2}\right) + \log_2 n^n + \log_2 e^{-n}$$

# Cálculo do Lower Bound

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^{n}$$

$$n! \approx (2\pi n)^{\frac{1}{2}} n^{n} e^{-n}$$

$$\log_2 n! \approx \log_2 (2\pi)^{\frac{1}{2}} + \log_2 (n)^{\frac{1}{2}} + \log_2 n^{n} + \log_2 e^{-n}$$

$$\log_2 n! \approx \left(\frac{\log_2(2\pi)}{2}\right) + \left(\frac{\log_2(n)}{2}\right) + \log_2 n^{n} + \log_2 e^{-n}$$

$$\log_2 n! \approx \left(\frac{\log_2(2\pi)}{2}\right) + \left(\frac{\log_2(n)}{2}\right) + (n \cdot \log_2 n) - n \cdot \log_2 e$$

# Cálculo do Lower Bound

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^{n}$$

$$n! \approx (2\pi n)^{\frac{1}{2}} n^{n} e^{-n}$$

$$\log_2 n! \approx \log_2 (2\pi)^{\frac{1}{2}} + \log_2 (n)^{\frac{1}{2}} + \log_2 n^{n} + \log_2 e^{-n}$$

$$\log_2 n! \approx \left(\frac{\log_2(2\pi)}{2}\right) + \left(\frac{\log_2(n)}{2}\right) + \log_2 n^{n} + \log_2 e^{-n}$$

$$\log_2 n! \approx \left(\frac{\log_2(2\pi)}{2}\right) + \left(\frac{\log_2(n)}{2}\right) + (n \cdot \log_2 n) - n \cdot \log_2 e$$

$$\log_2 n! \approx O(1) + O(\log_2 n) + O(n \cdot \log_2 n) - O(n)$$

# Cálculo do Lower Bound

$$\log_2 n! \approx O(1) + O(\log_2 n) + O(n \cdot \log_2 n) - O(n)$$

$$LB(n) \geq \log_2 n!$$

*então*

$$LB(n) = \Omega(n \cdot \log_2 n)$$

# Cálculo do Lower Bound

$$\log_2 n! \approx O(1) + O(\log_2 n) + O(n \cdot \log_2 n) - O(n)$$

$$LB(n) \geq \log_2 n!$$

*então*

$$LB(n) = \Omega(n \cdot \log_2 n)$$

Desta forma, se encontrarmos um algoritmo que resolva a ordenação em tempo $\boldsymbol{O(n \cdot \log_2 n)}$, ele será ótimo, e esse problema estará computacionalmente resolvido.

# Intercalação

- Lembrando do problema da intercalação
  - Usando duas string.

# Intercalação

- Lembrando do problema da intercalação
  - Usando duas string.
  - Usando dois vetores ordenados.

# Intercalação

- Lembrando do problema da intercalação
  - Usando duas string.
  - Usando dois vetores ordenados.
  - Usando um vetor bipartido e ordenados.

# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 6 | 8 | 9 | 10 | 32 | 65 | 88 | 0 | 1 | 4 | 7 | 12 | 45 | 63 | 69 | 71 | 80 |

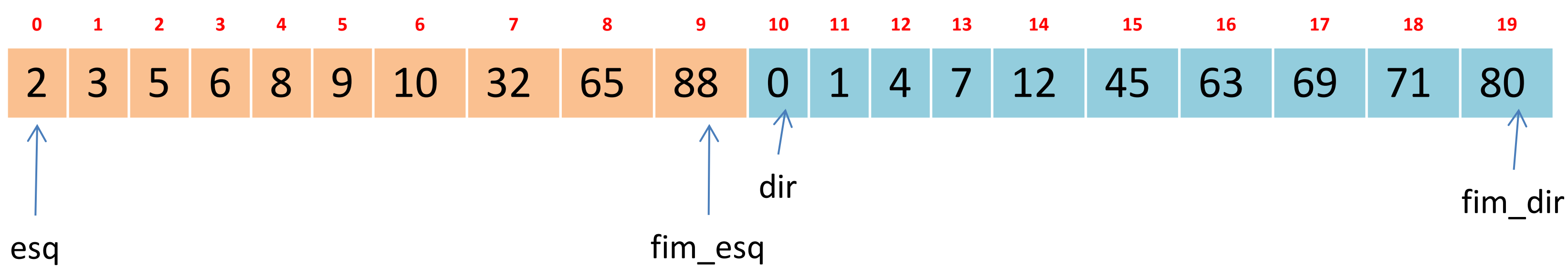
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 6 | 8 | 9 | 10 | 32 | 65 | 88 | 0 | 1 | 4 | 7 | 12 | 45 | 63 | 69 | 71 | 80 |

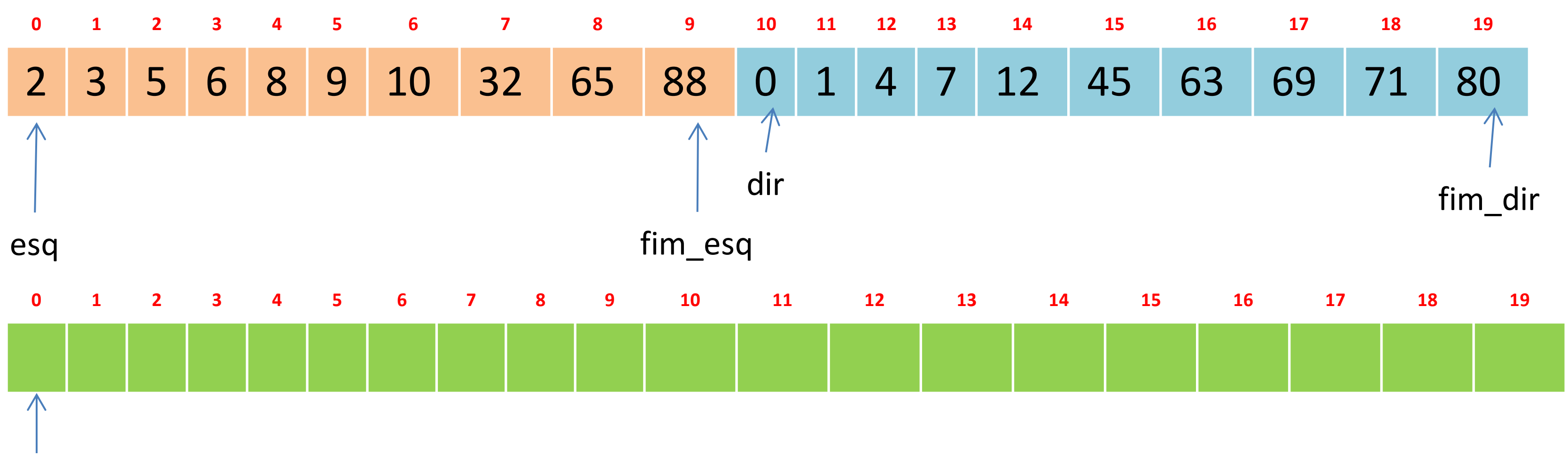
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



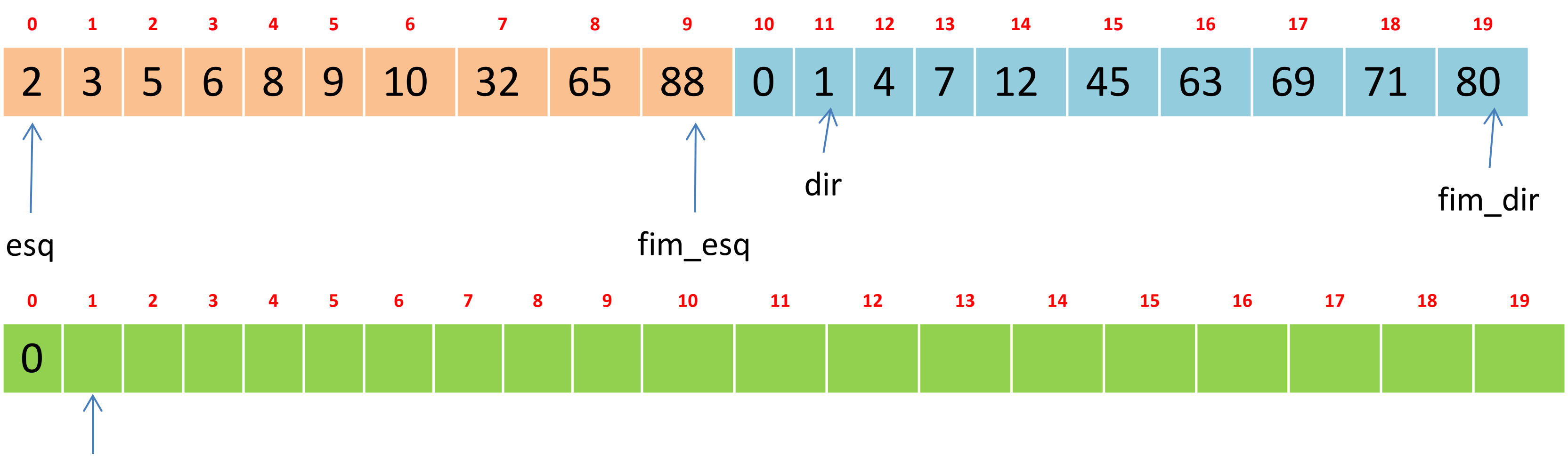
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



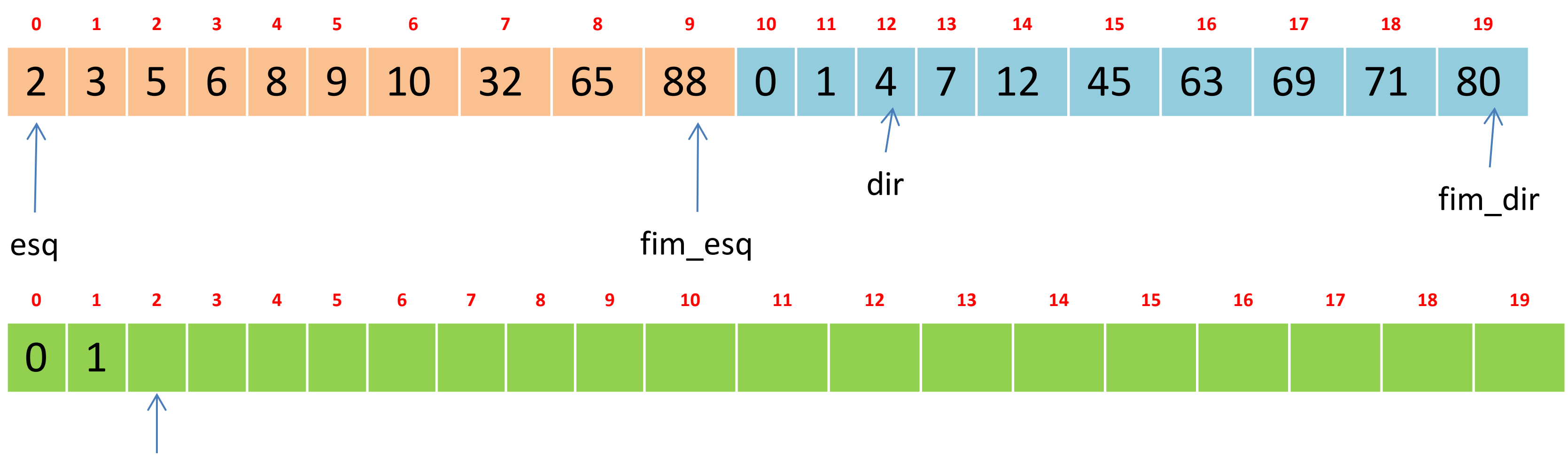
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



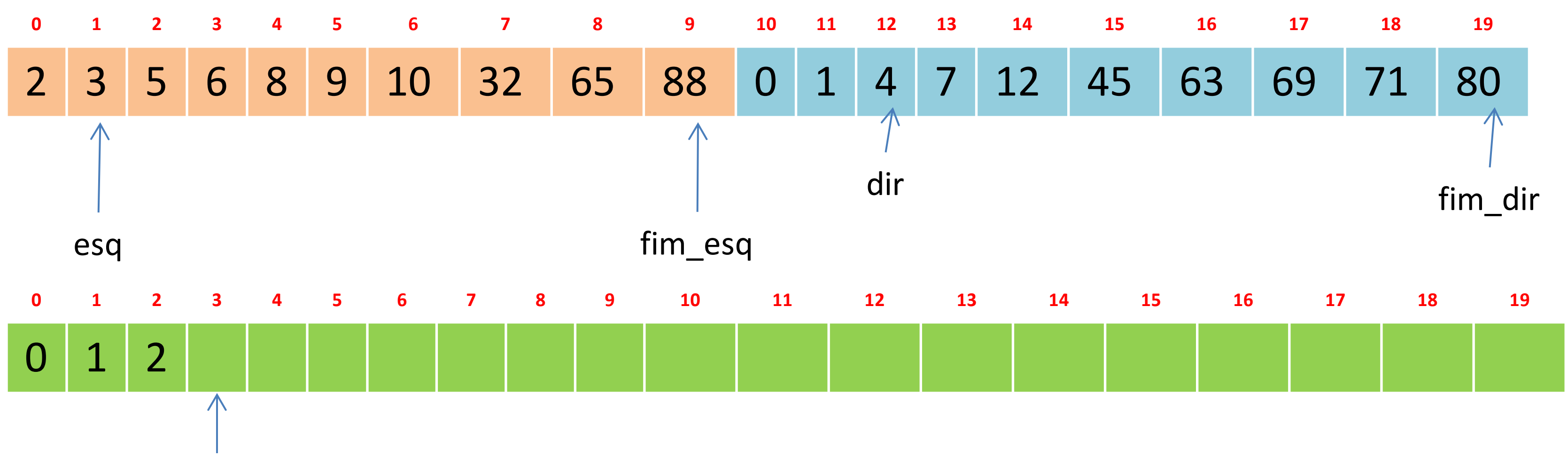
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



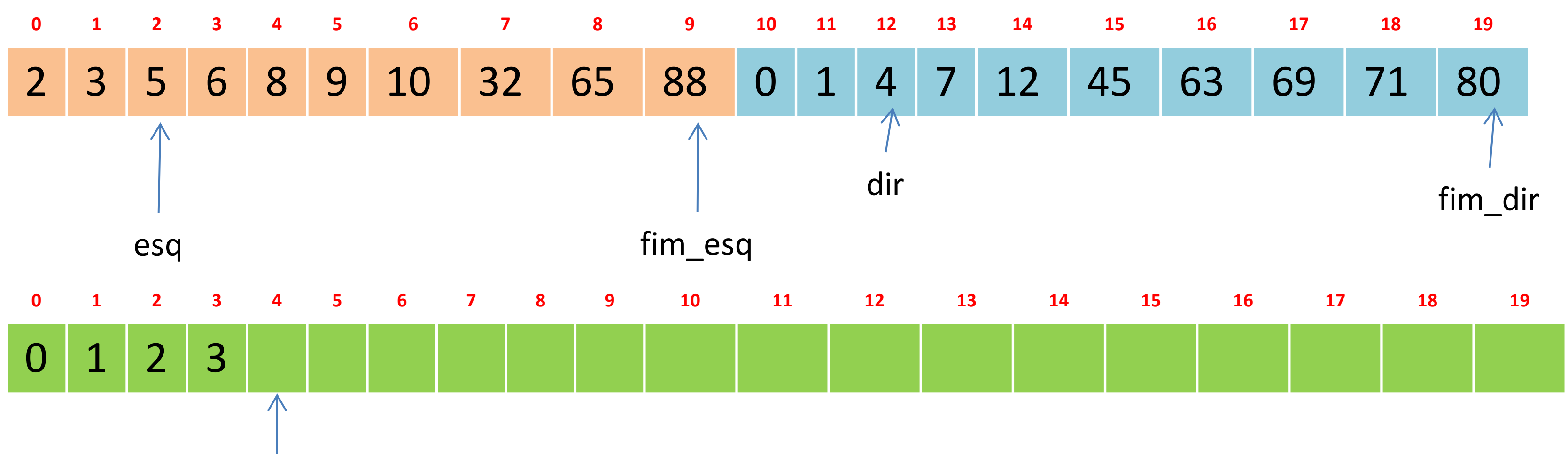
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



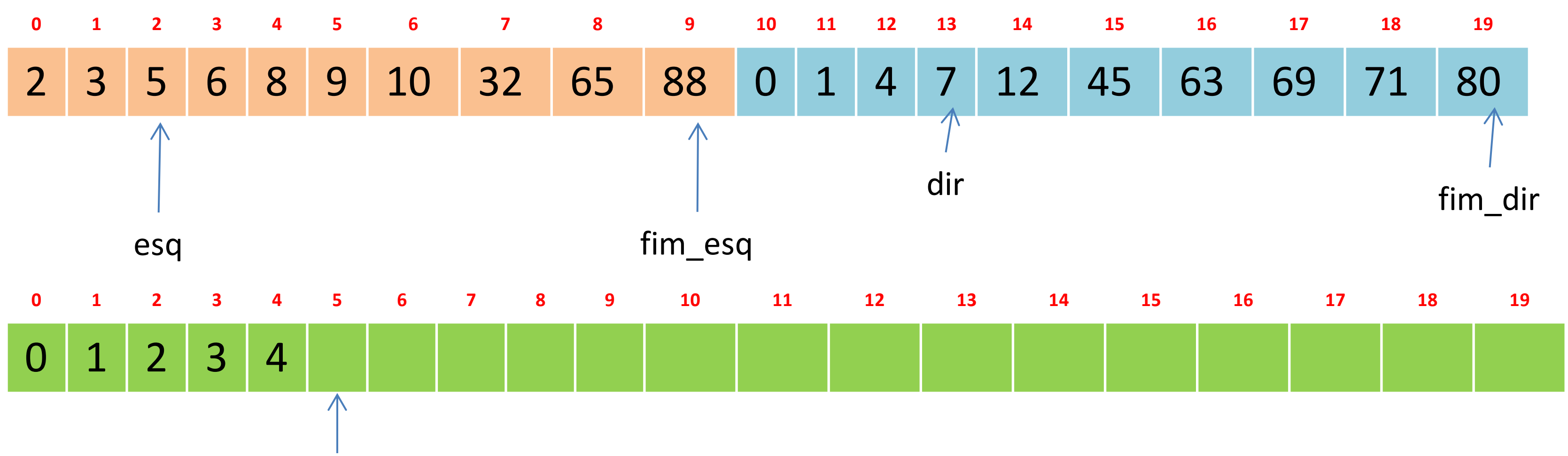
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



# Intercalação

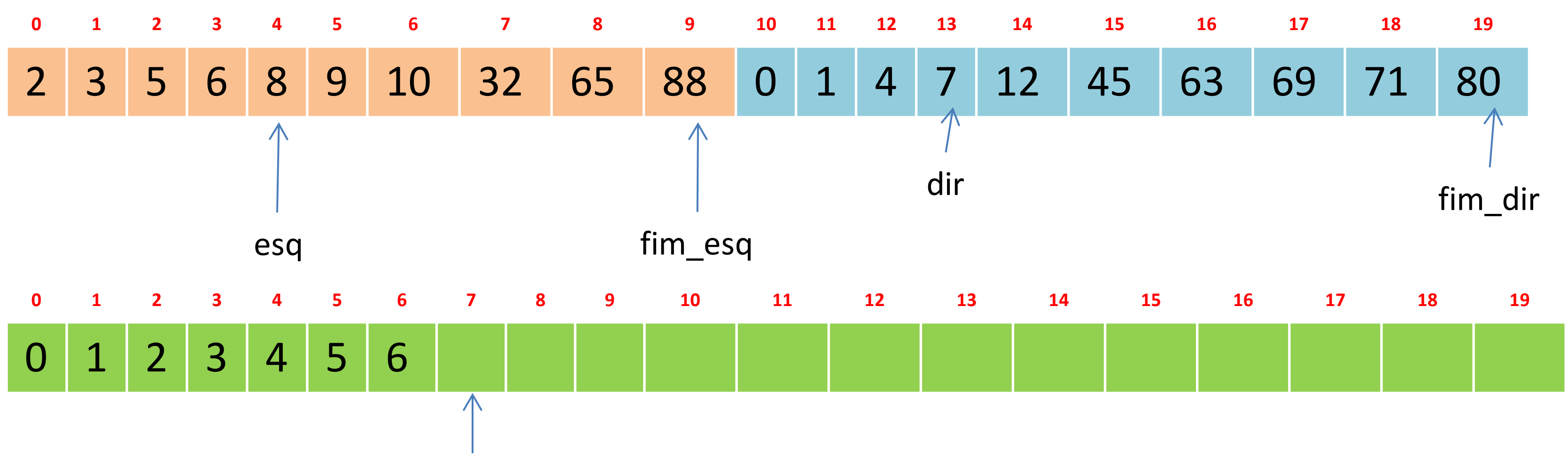
- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 6 | 8 | 9 | 10 | 32 | 65 | 88 | 0 | 1 | 4 | 7 | 12 | 45 | 63 | 69 | 71 | 80 |
| | | | esq | | | | | | fim_esq | | | | dir | | | | | | fim_dir |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | | | | | | | | | | | | | | |

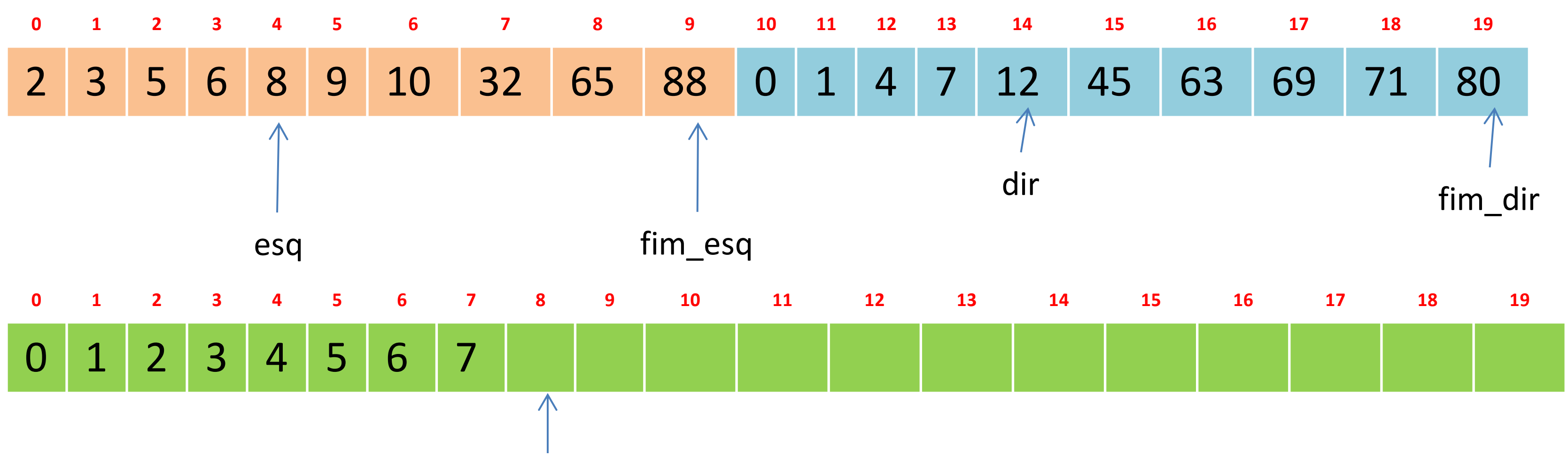
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



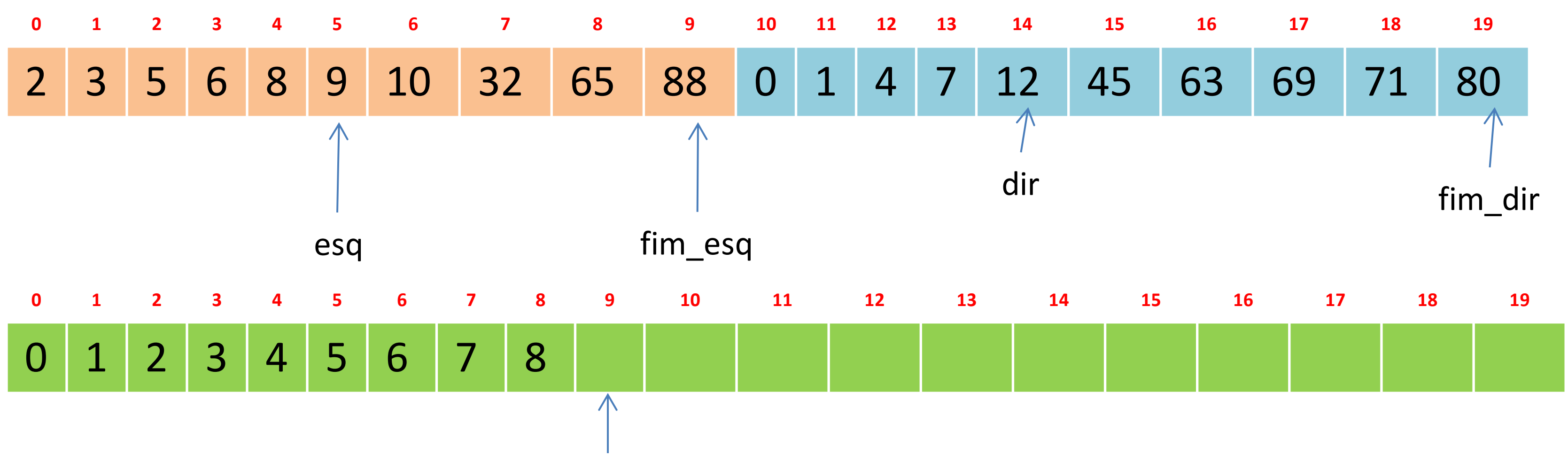
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



# Intercalação

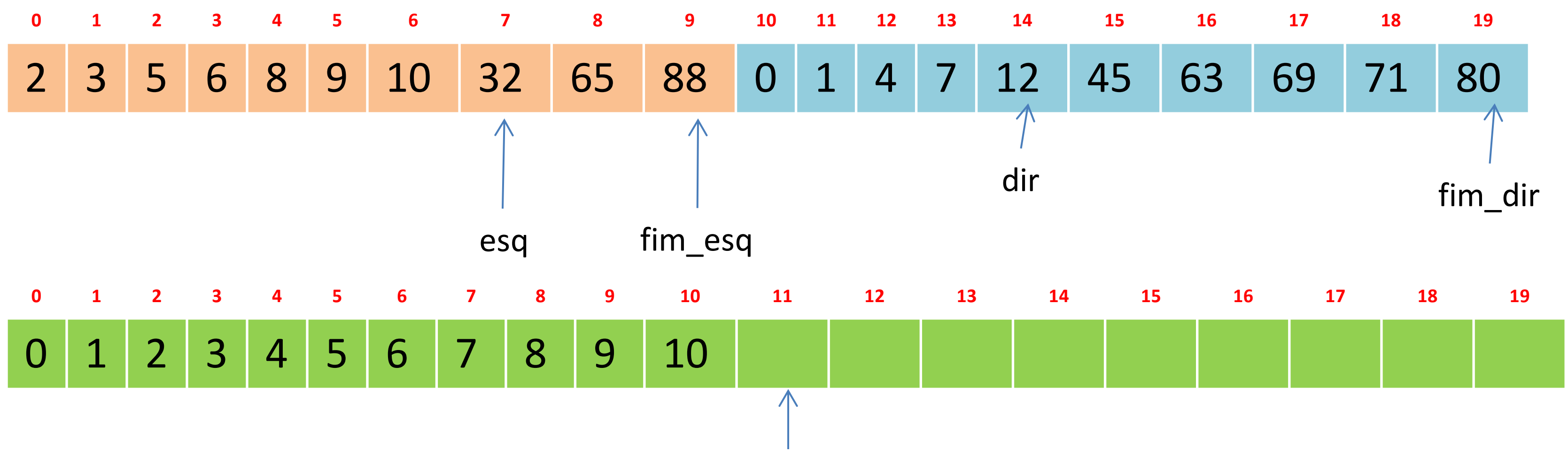
- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 6 | 8 | 9 | 10 | 32 | 65 | 88 | 0 | 1 | 4 | 7 | 12 | 45 | 63 | 69 | 71 | 80 |
| | | | | | | esq | | | fim_esq | | | | | dir | | | | | fim_dir |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | | | | | | | | | |

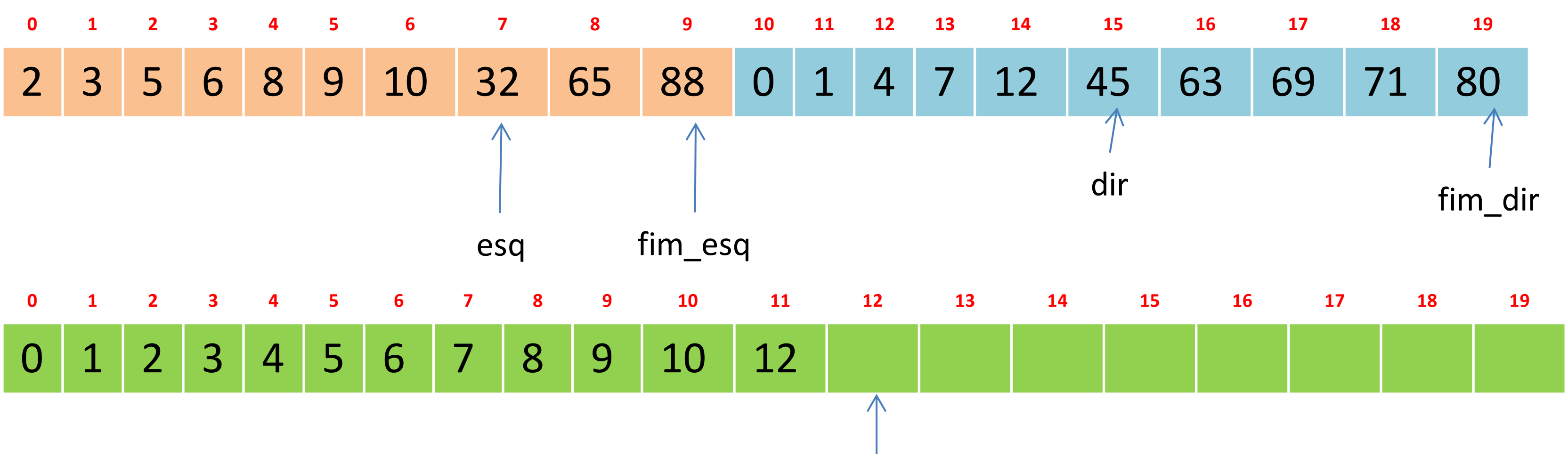
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



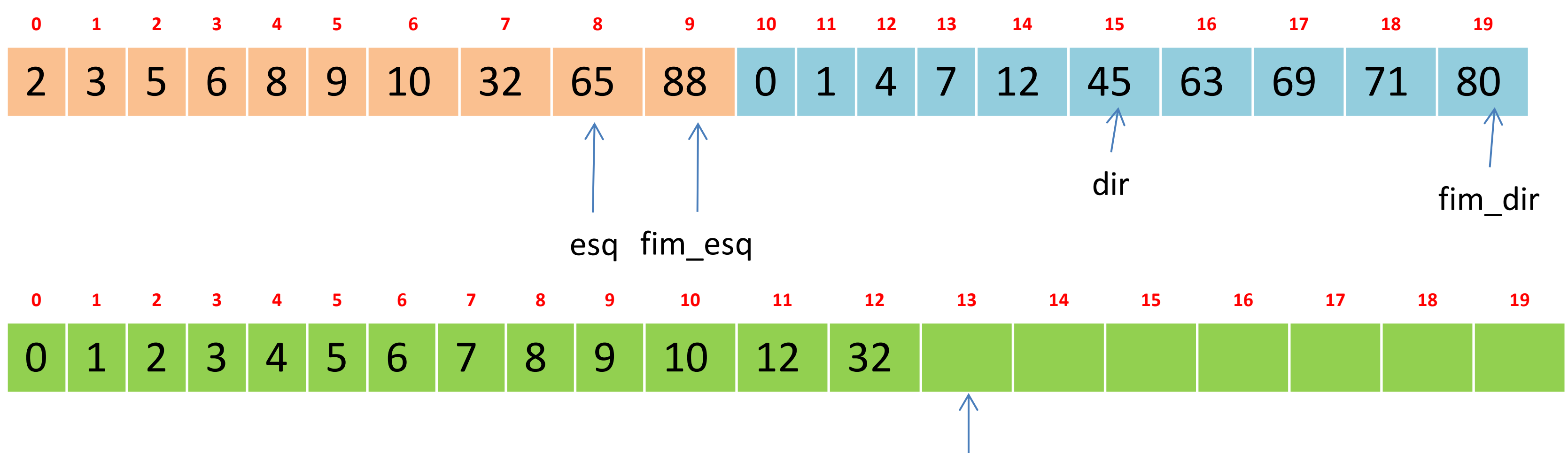
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



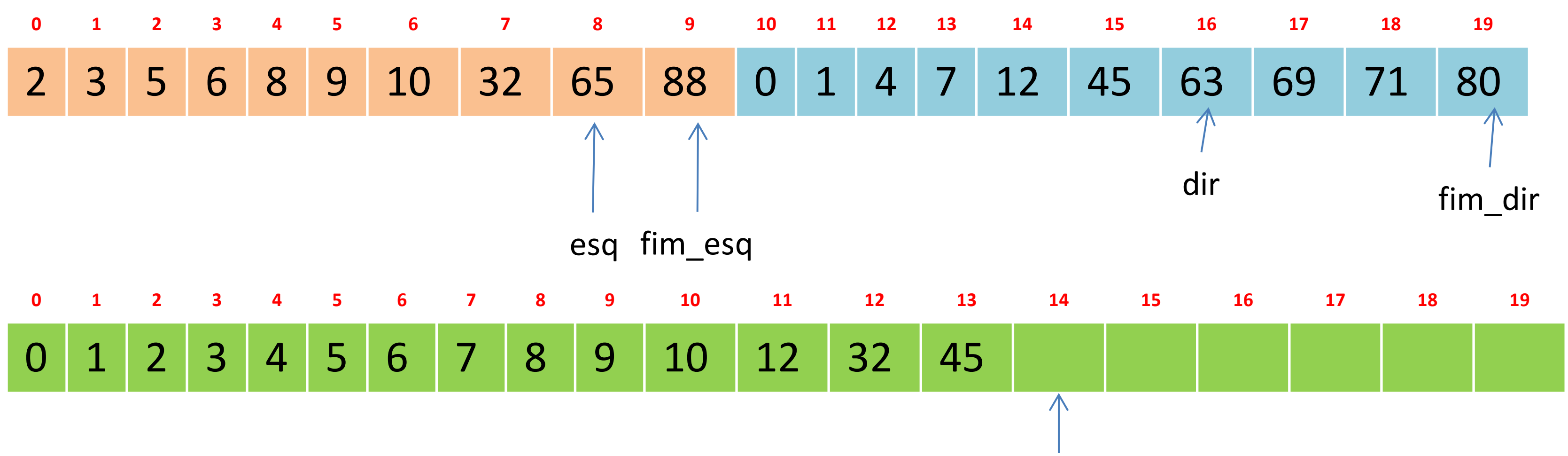
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



# Intercalação

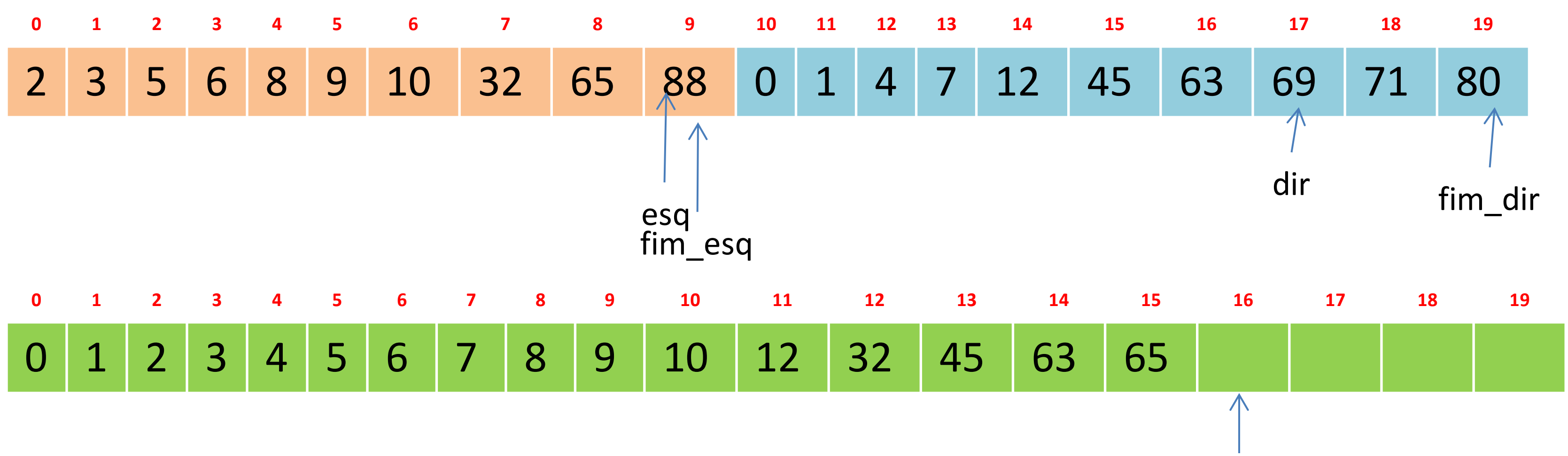
- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 6 | 8 | 9 | 10 | 32 | 65 | 88 | 0 | 1 | 4 | 7 | 12 | 45 | 63 | 69 | 71 | 80 |
| | | | | | | | | esq | fim_esq | | | | | | | | dir | | fim_dir |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 | 32 | 45 | 63 | | | | | |

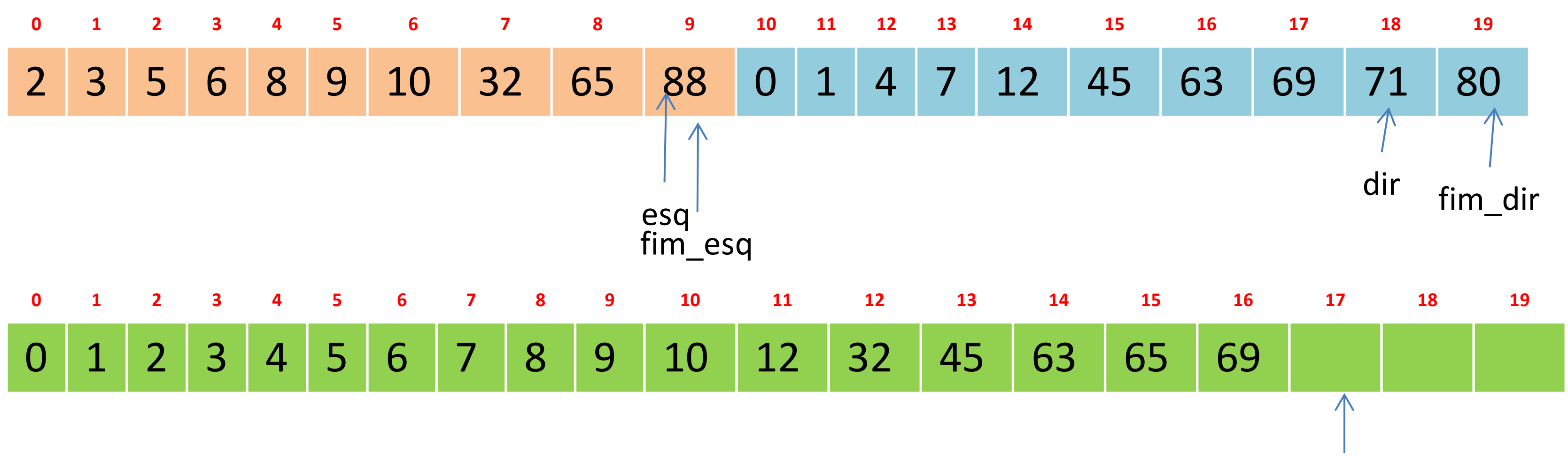
# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 6 | 8 | 9 | 10 | 32 | 65 | 88 | 0 | 1 | 4 | 7 | 12 | 45 | 63 | 69 | 71 | 80 |

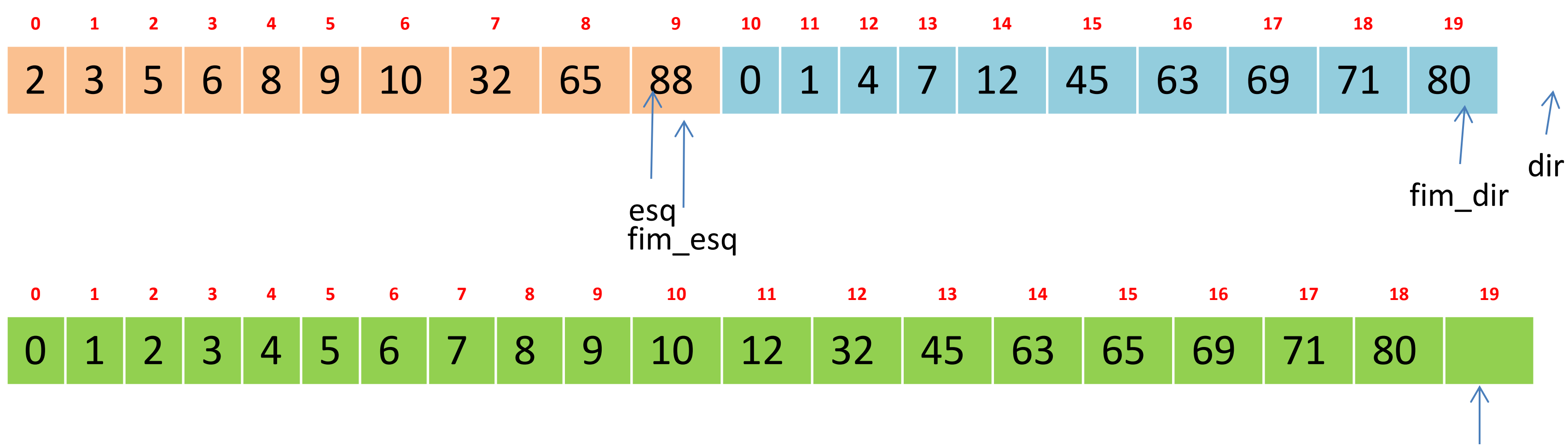
esq
fim_esq

dir
fim_dir

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 | 32 | 45 | 63 | 65 | 69 | 71 | | |

# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



# Intercalação

- Dado um vetor de inteiro de tamanho n, bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 6 | 8 | 9 | 10 | 32 | 65 | 88 | 0 | 1 | 4 | 7 | 12 | 45 | 63 | 69 | 71 | 80 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 | 32 | 45 | 63 | 65 | 69 | 71 | 80 | 88 |

# Merge Sort

- Segue a técnica de divisão e conquista. Intuitivamente opera da seguinte forma:
  - Divisão: Quebra a seqüência de $n$ elementos a serem ordenados em duas subseqüências de $n/2$ cada.
  - Conquista: Classifica-se ambas subseqüências recursivamente, utilizando a ordenação por intercalação.
  - Combinação: Intercala-se ambas subseqüências para formar a solução do problema original

- Prós e Contras:
  - Pró: Ligeiramente melhor que outros algoritmos de ordenação para entradas suficientemente grandes.
  - Contra: Requer no mínimo o dobro de memória em comparação com os demais algoritmos de ordenação.

# Merge Sort

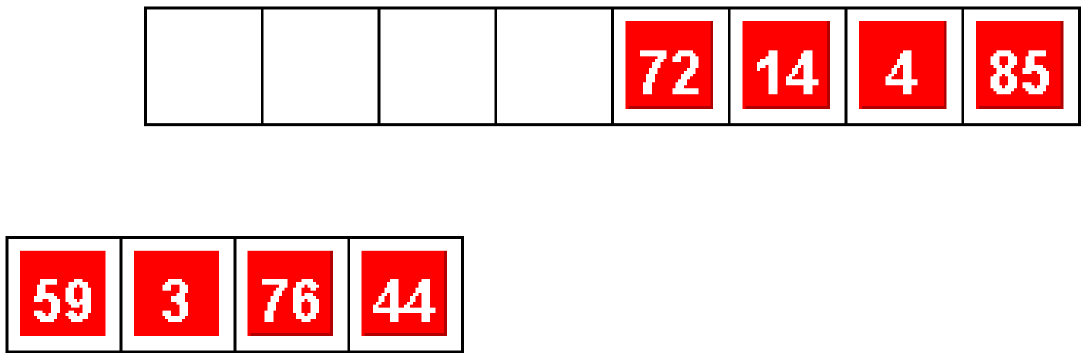
Idéia:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 59 | 3 | 76 | 44 | 72 | 14 | 4 | 85 |

Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



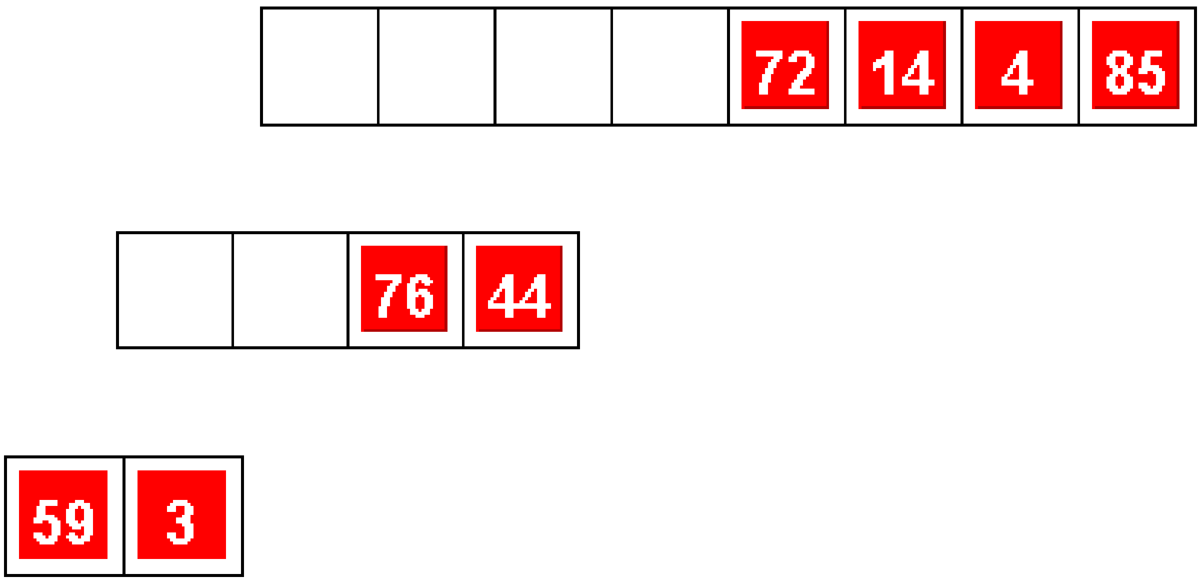

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

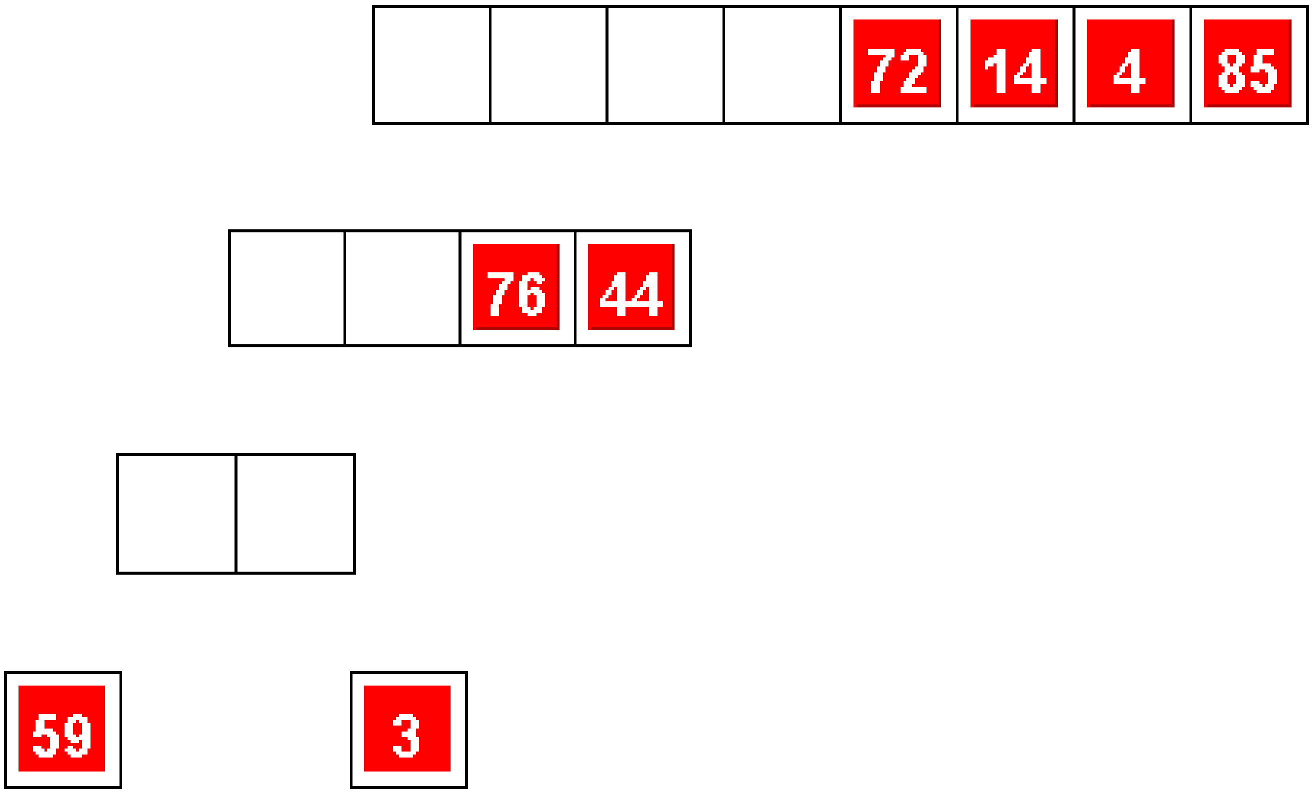
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



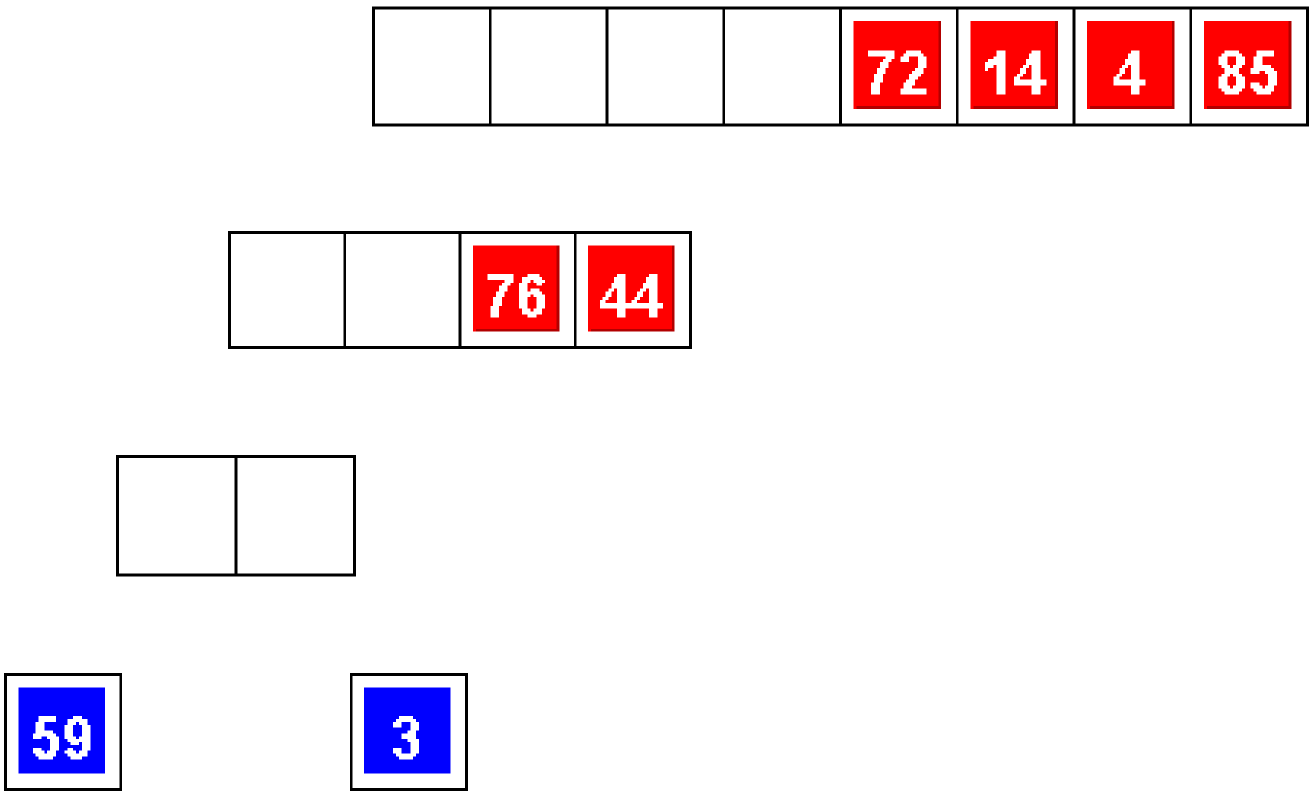

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

Idéia:



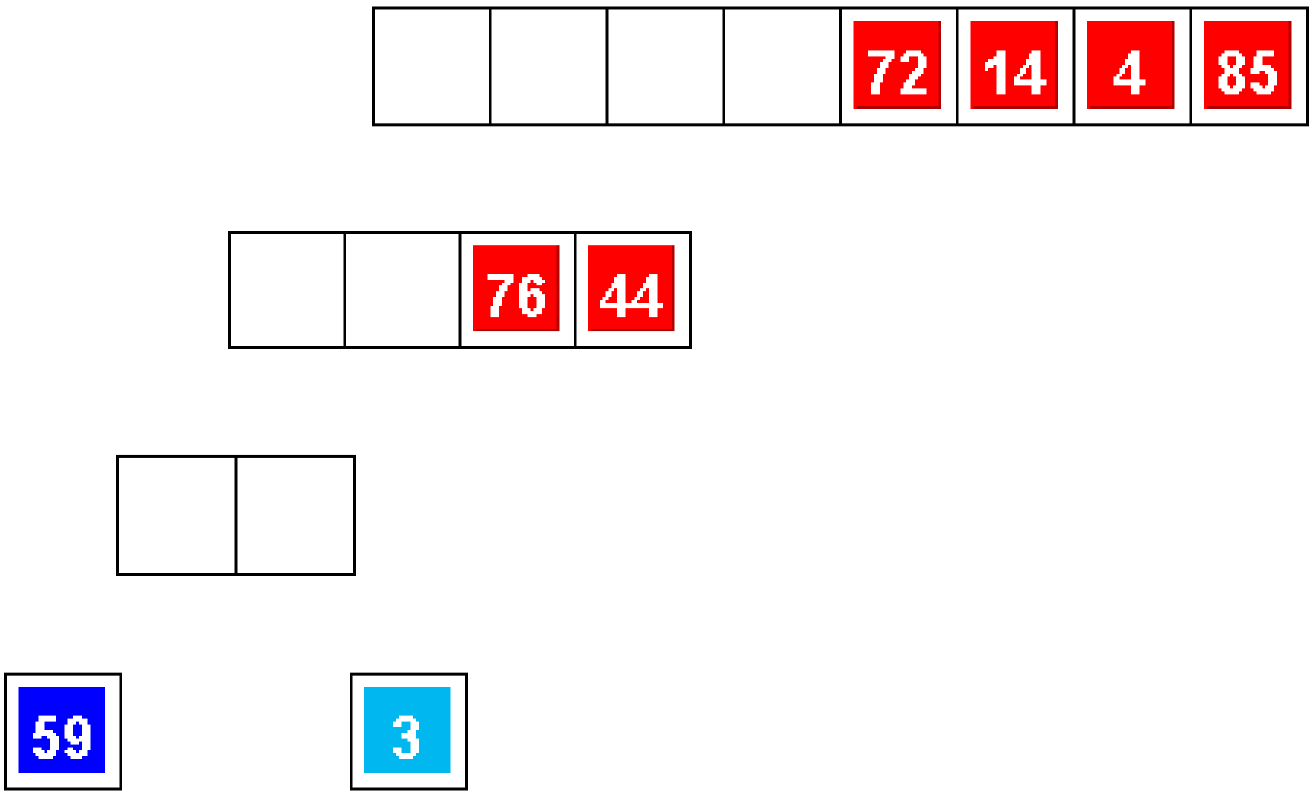

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

Idéia:



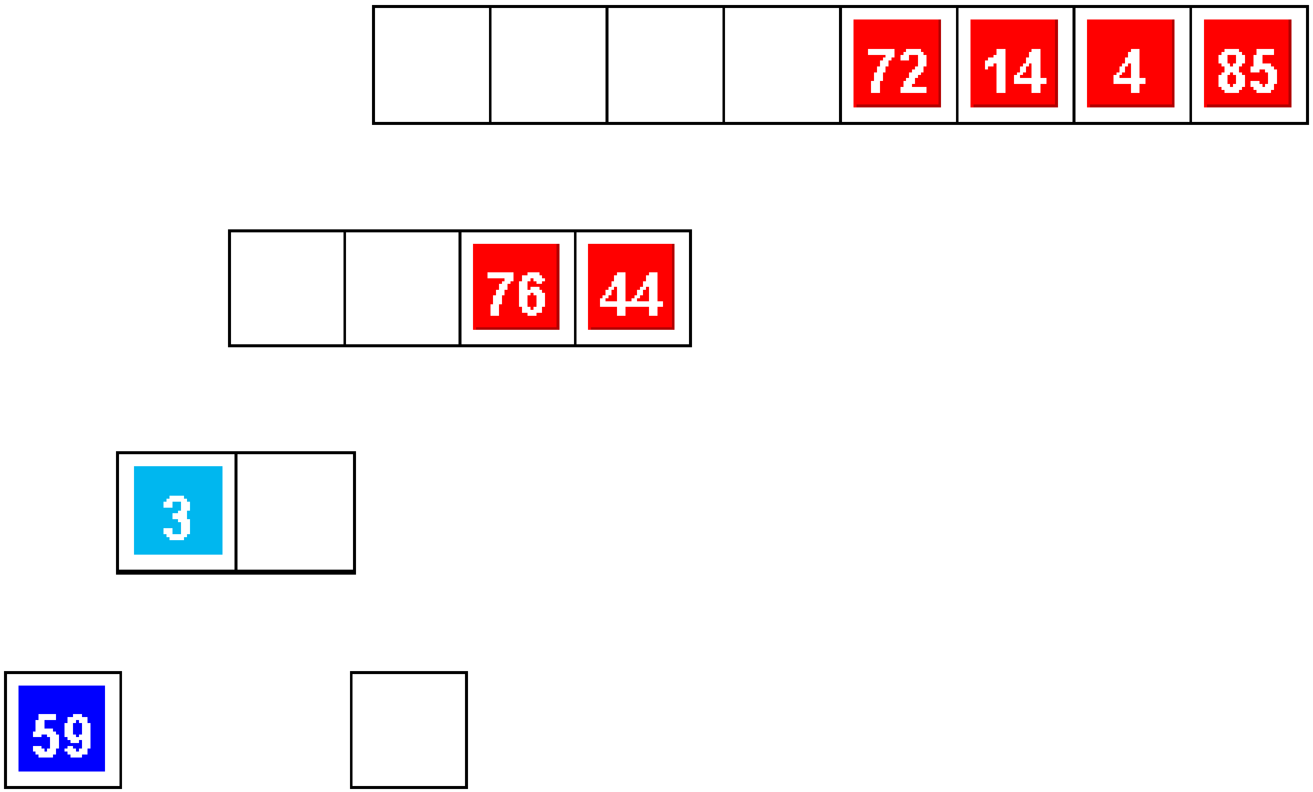

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

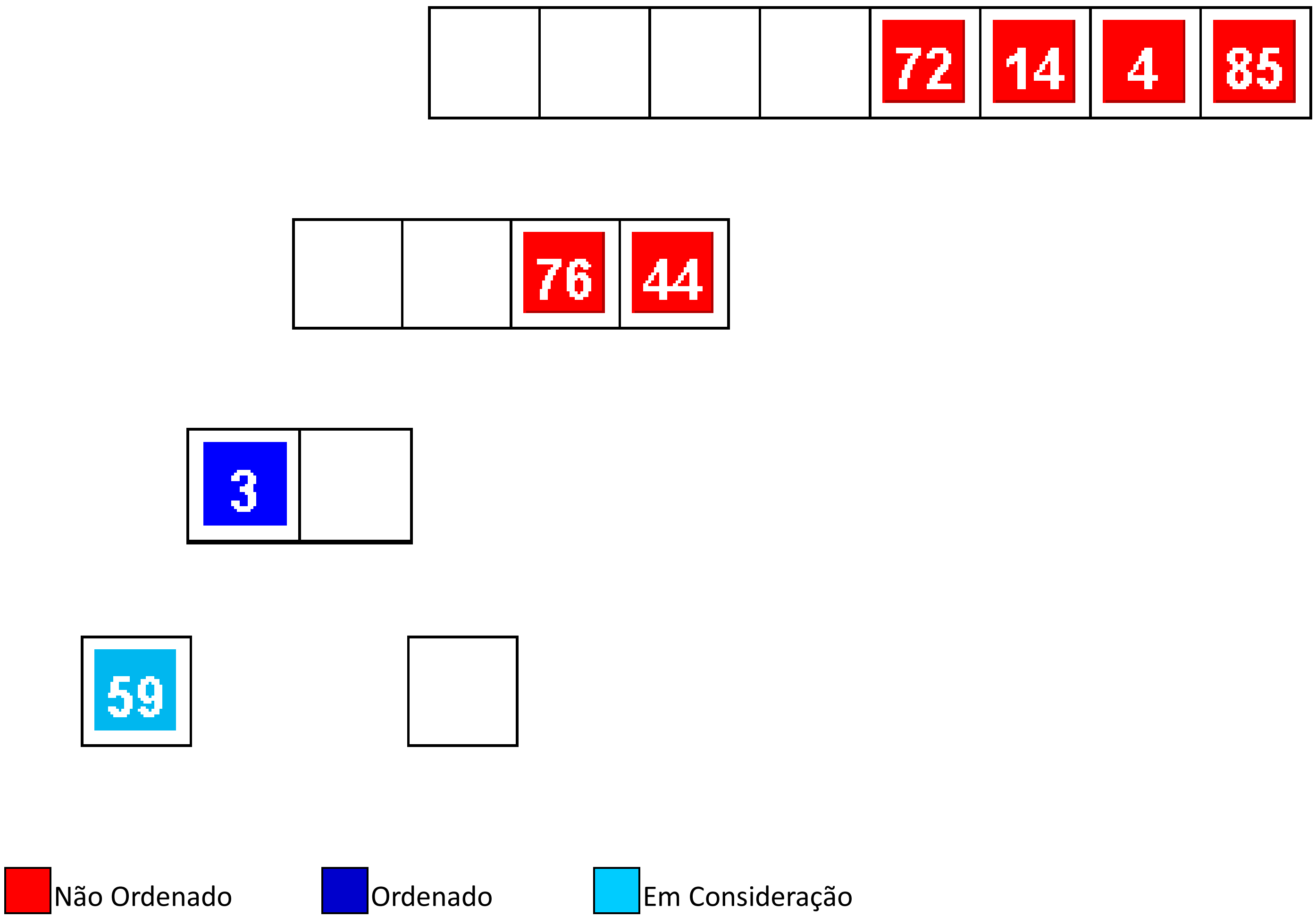
Idéia:




Não Ordenado Ordenado Em Consideração

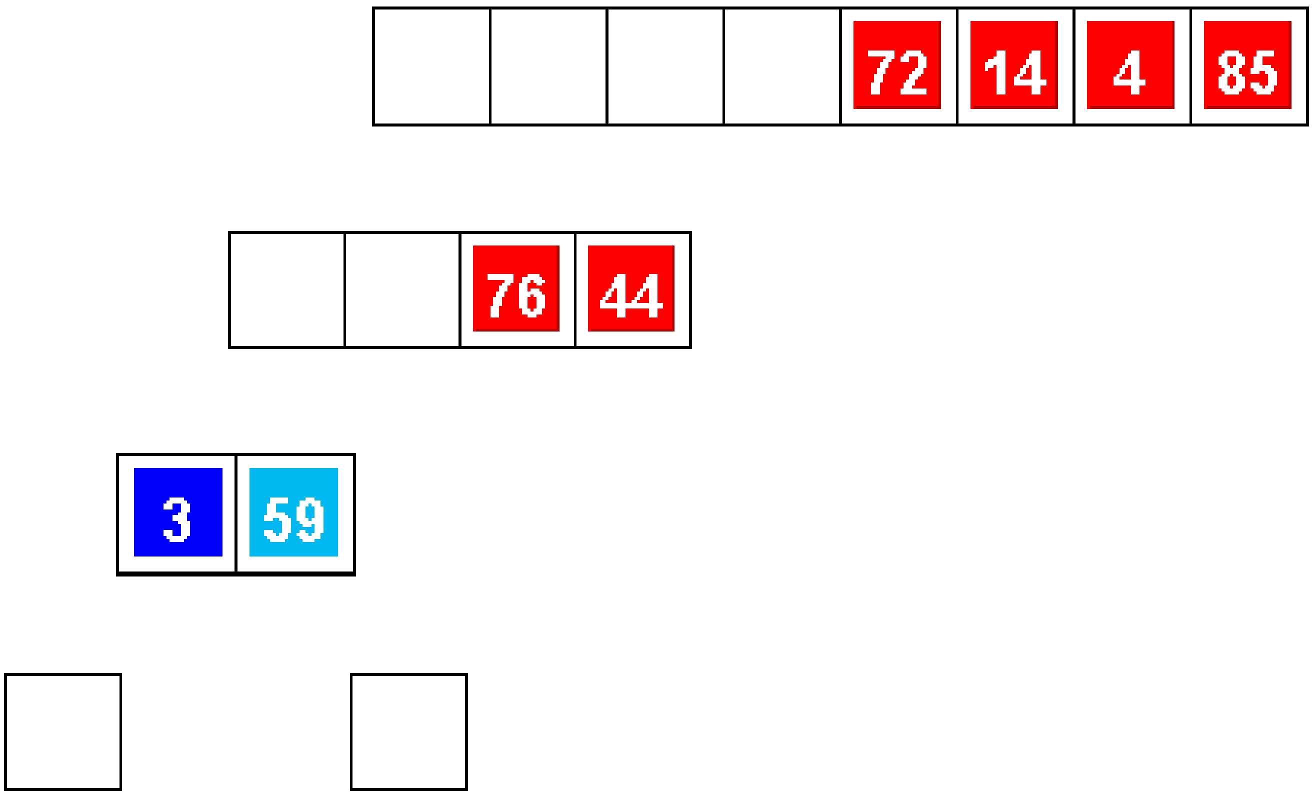
# Merge Sort

Idéia:



# Merge Sort

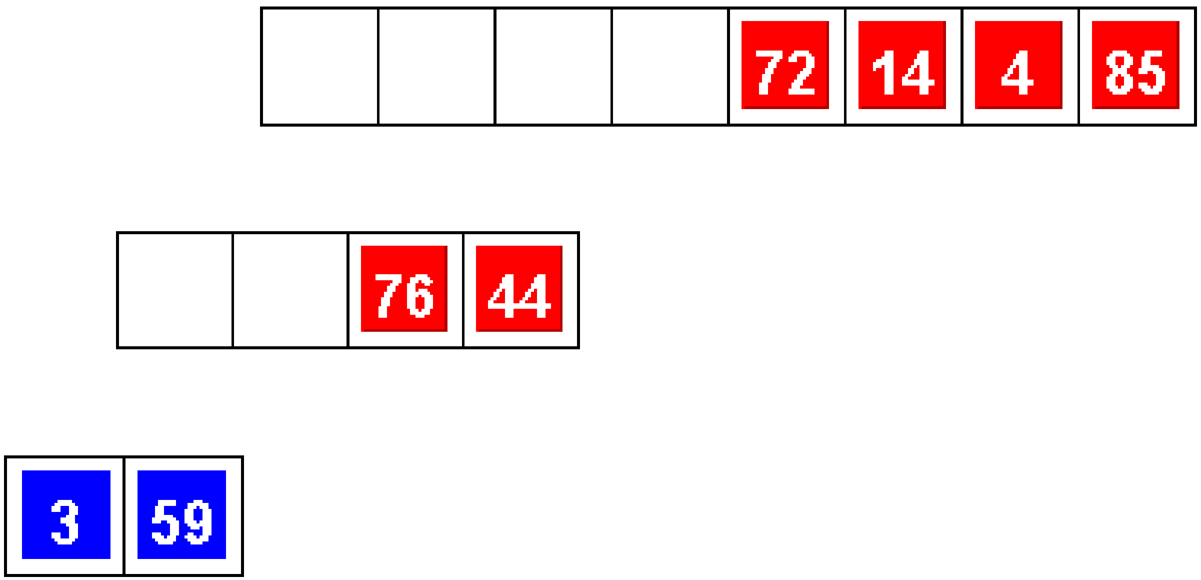
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



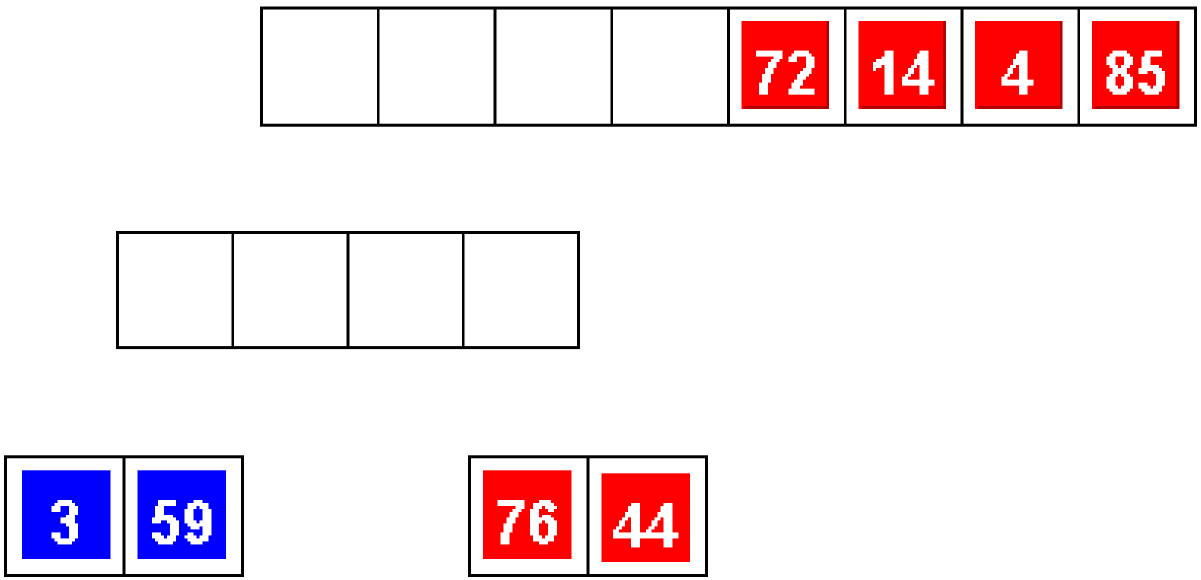

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

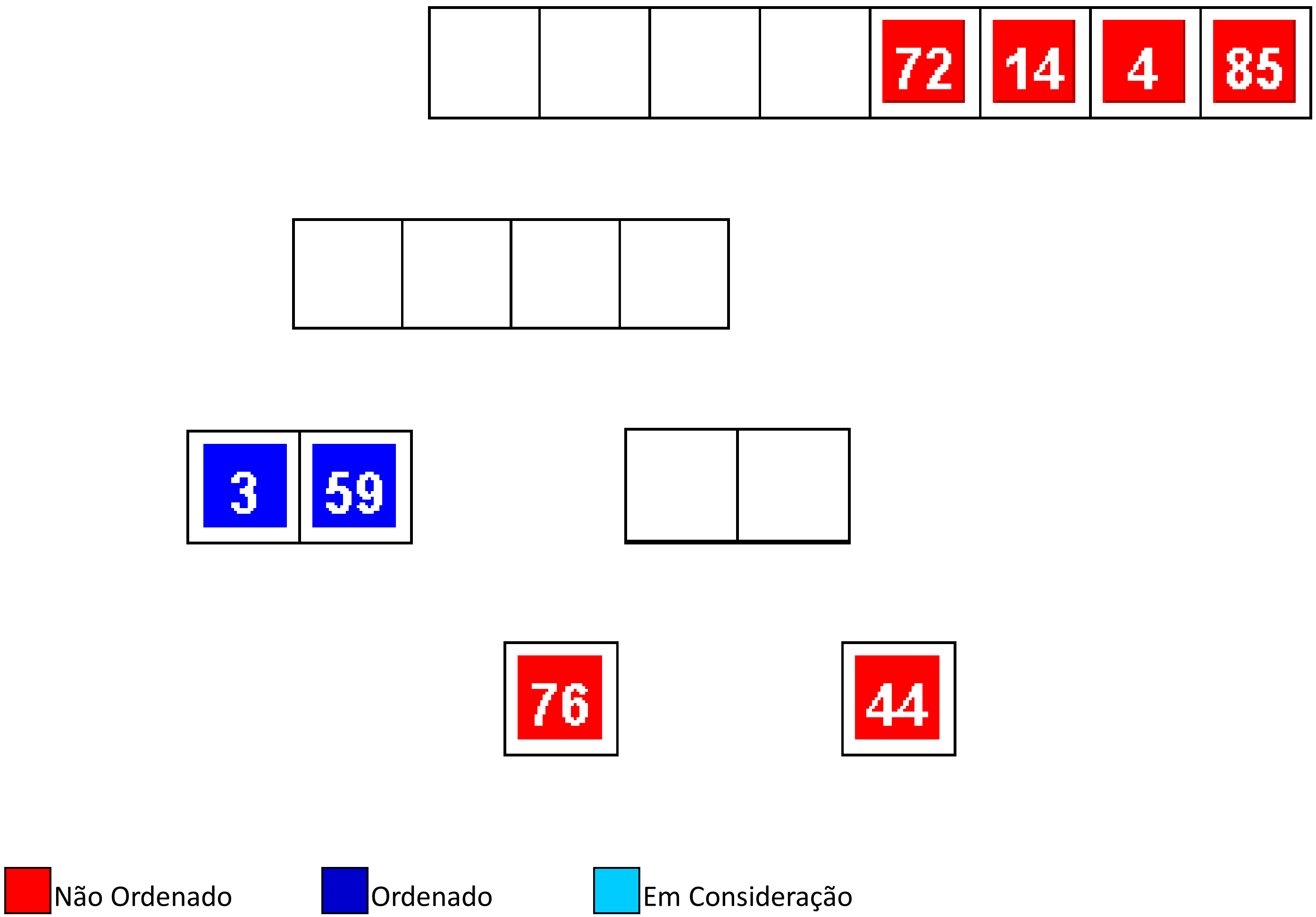
Idéia:




Não Ordenado Ordenado Em Consideração

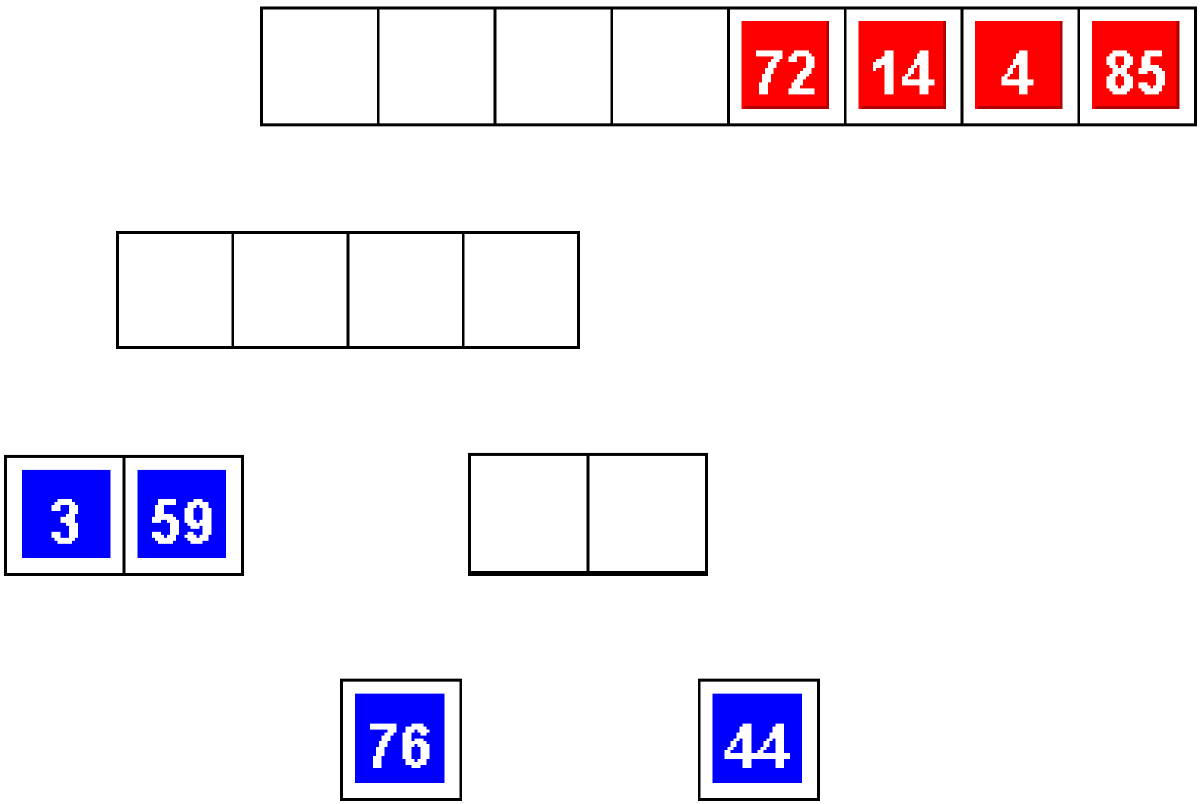
# Merge Sort

Idéia:



# Merge Sort

Idéia:



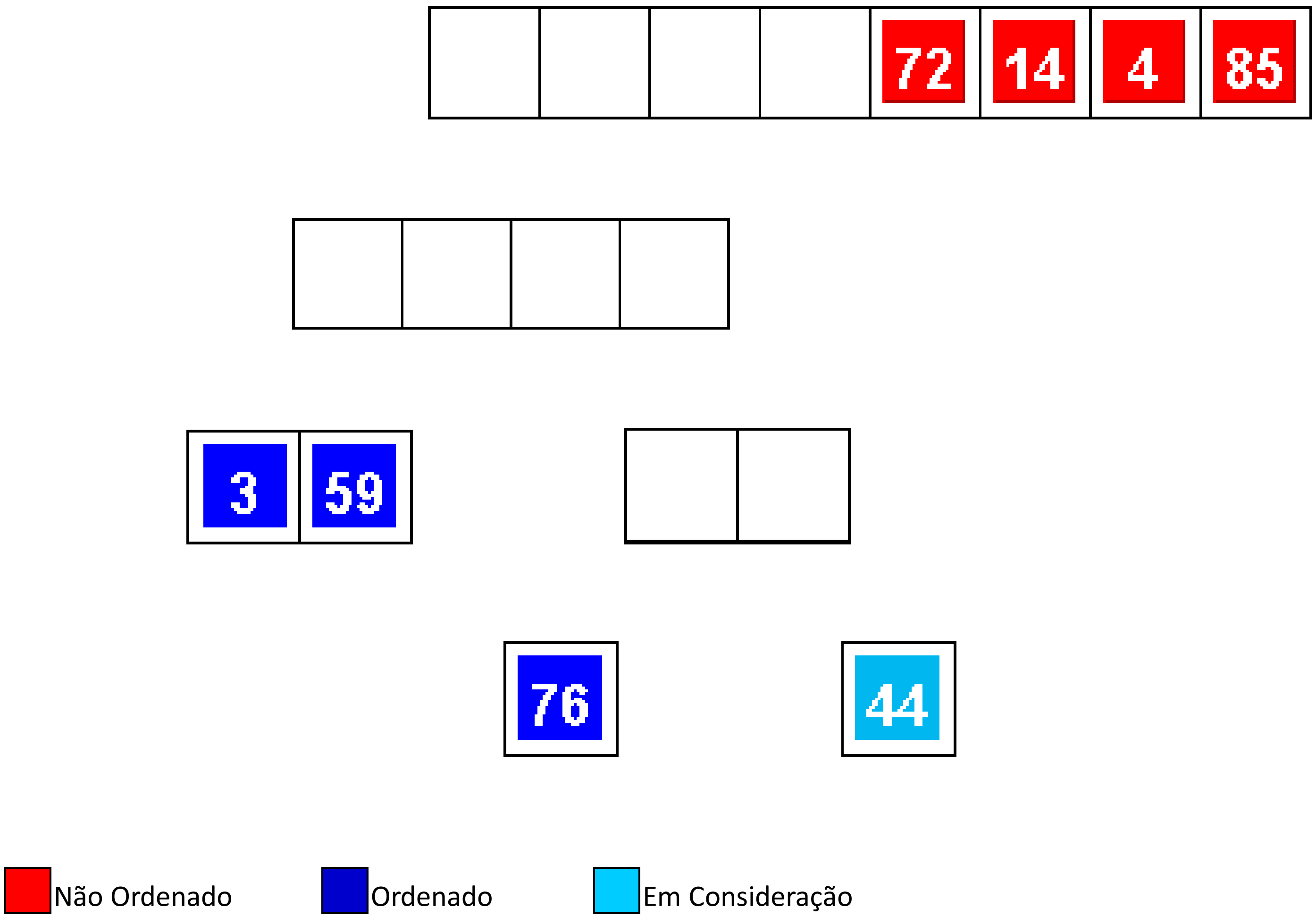

Não Ordenado

Ordenado

Em Consideração

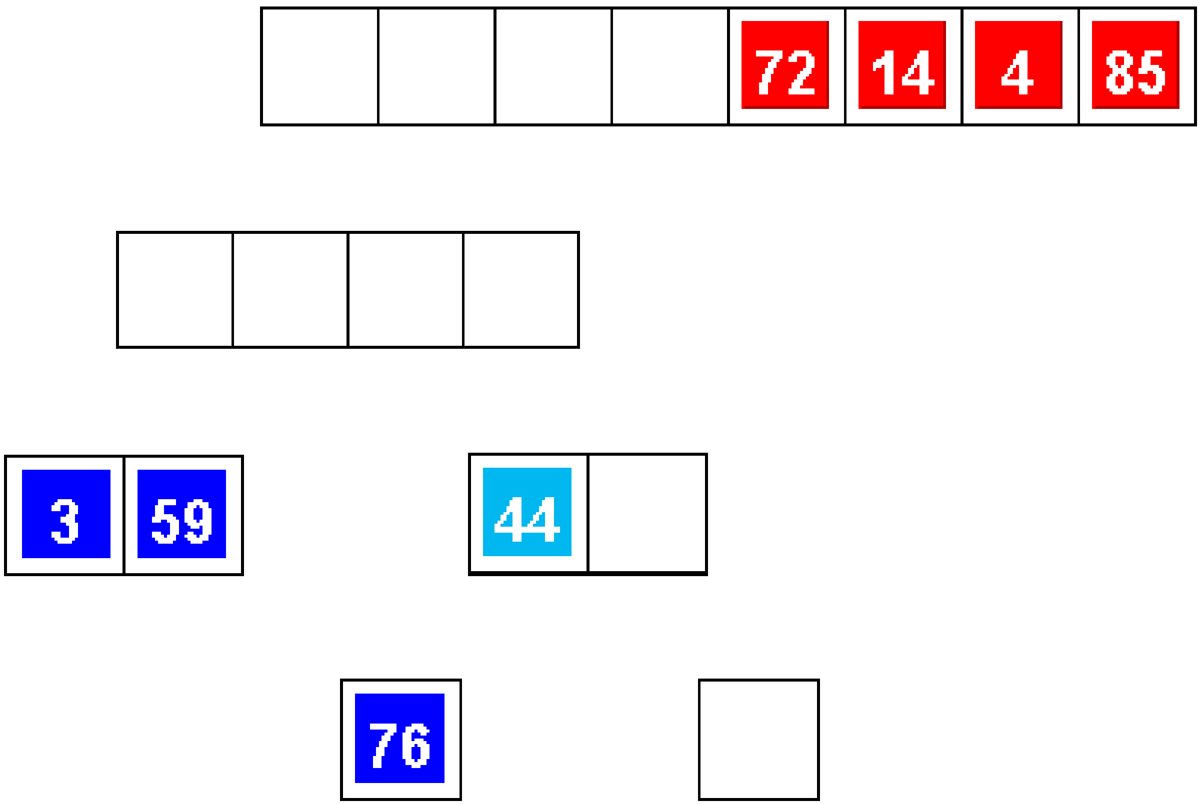
# Merge Sort

Idéia:



# Merge Sort

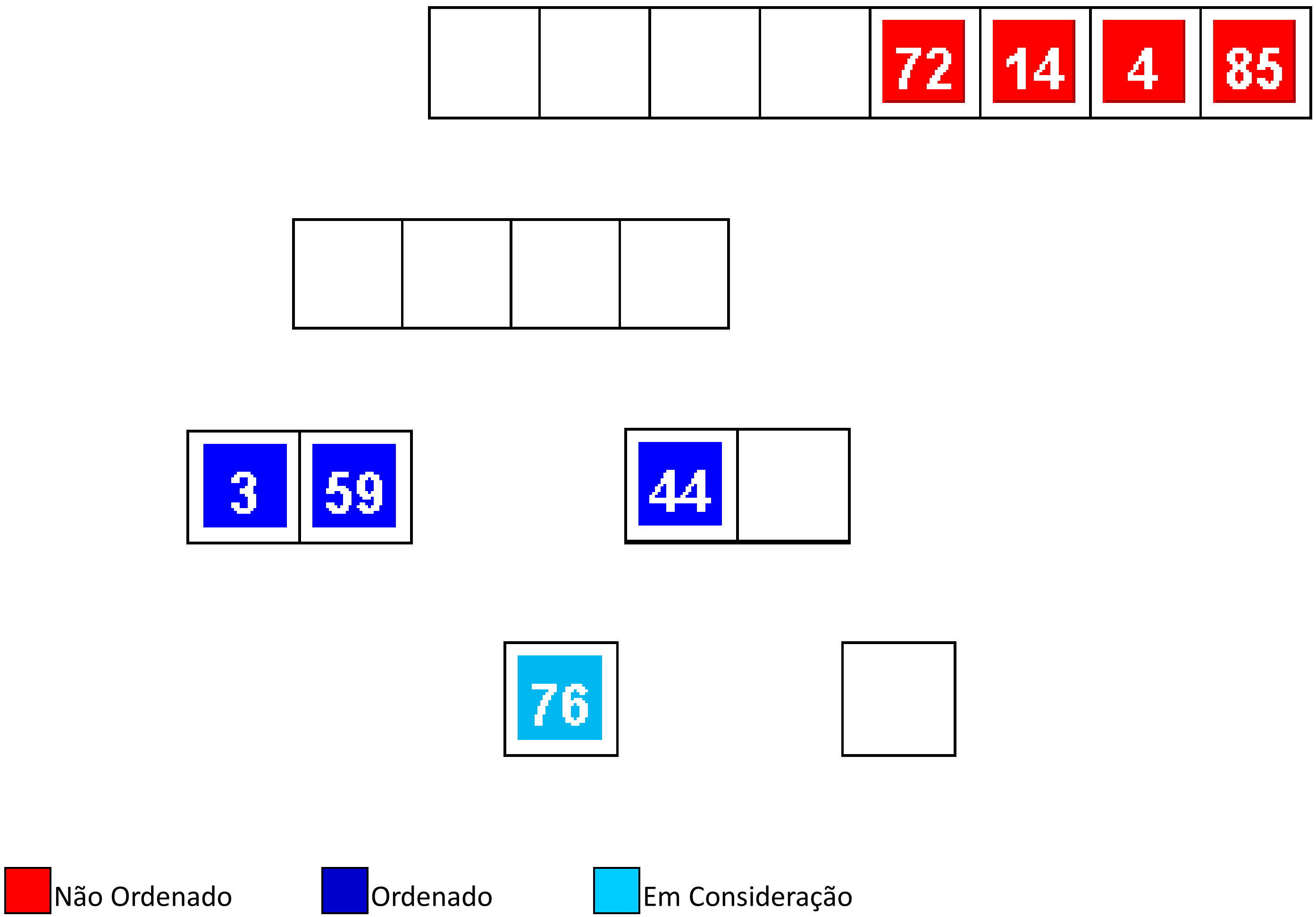
Idéia:




Não Ordenado Ordenado Em Consideração

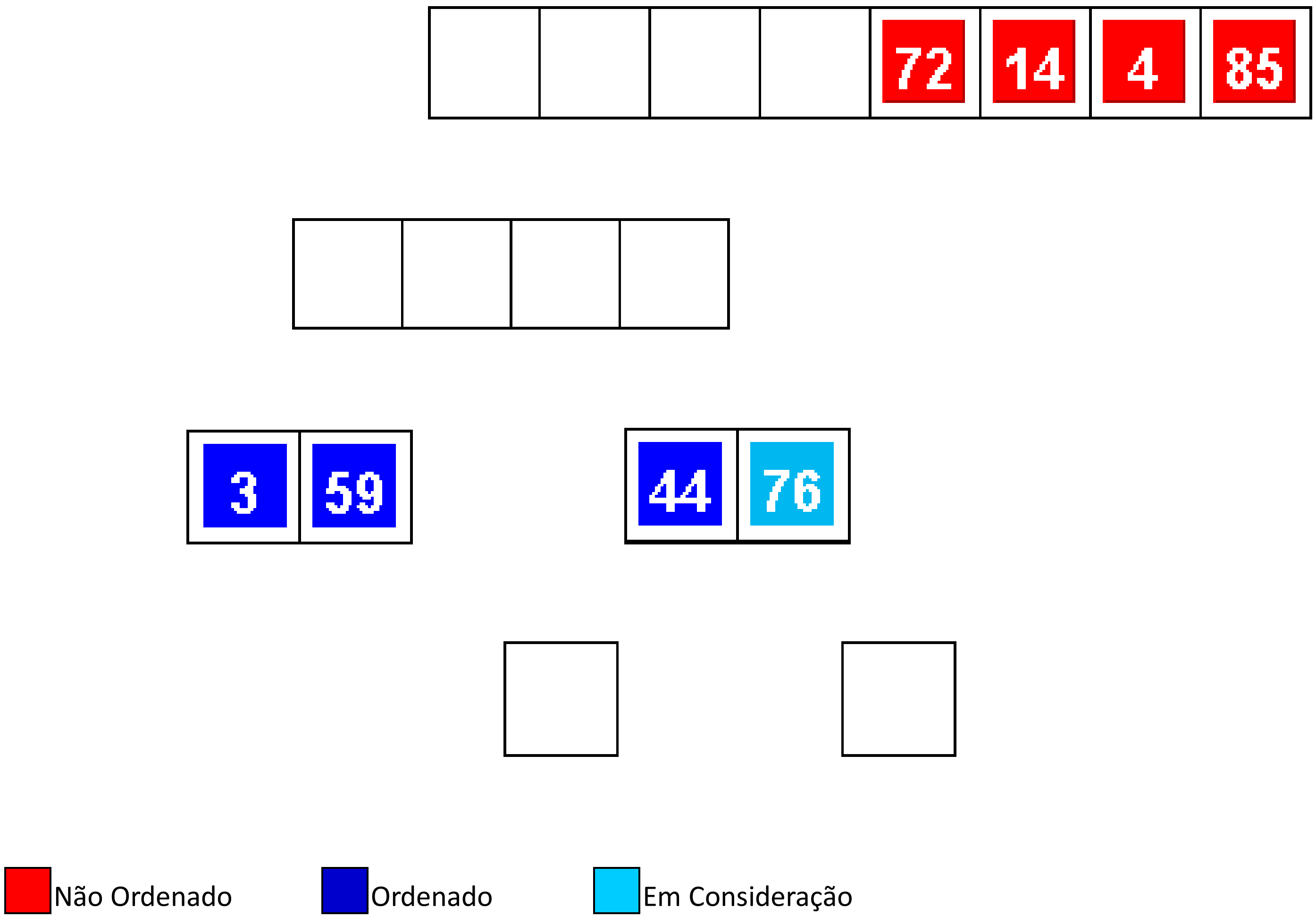
# Merge Sort

Idéia:



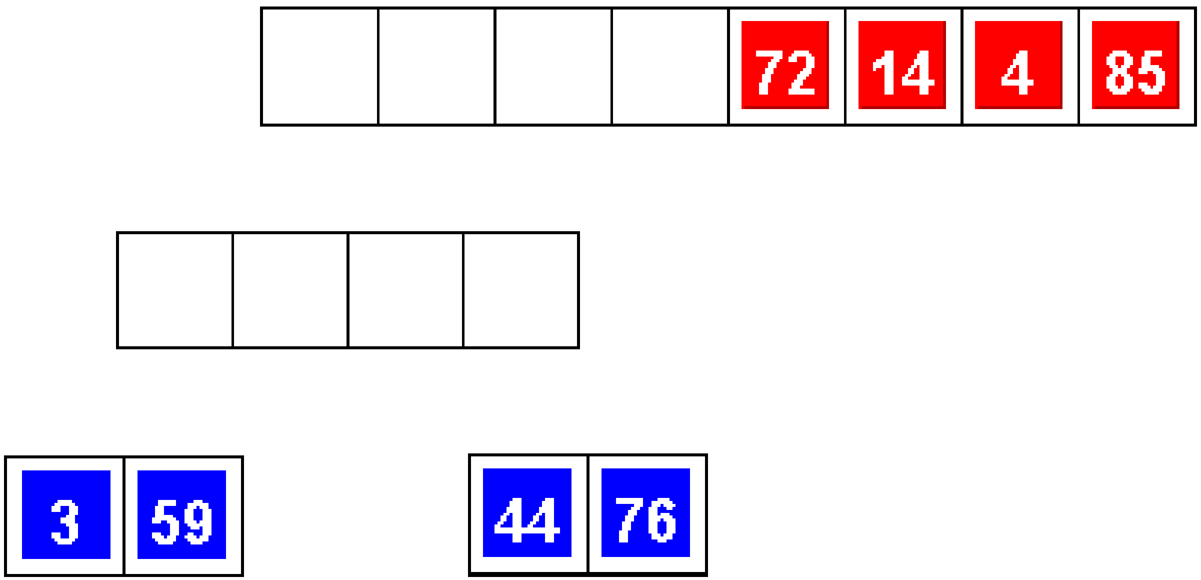
# Merge Sort

Idéia:



# Merge Sort

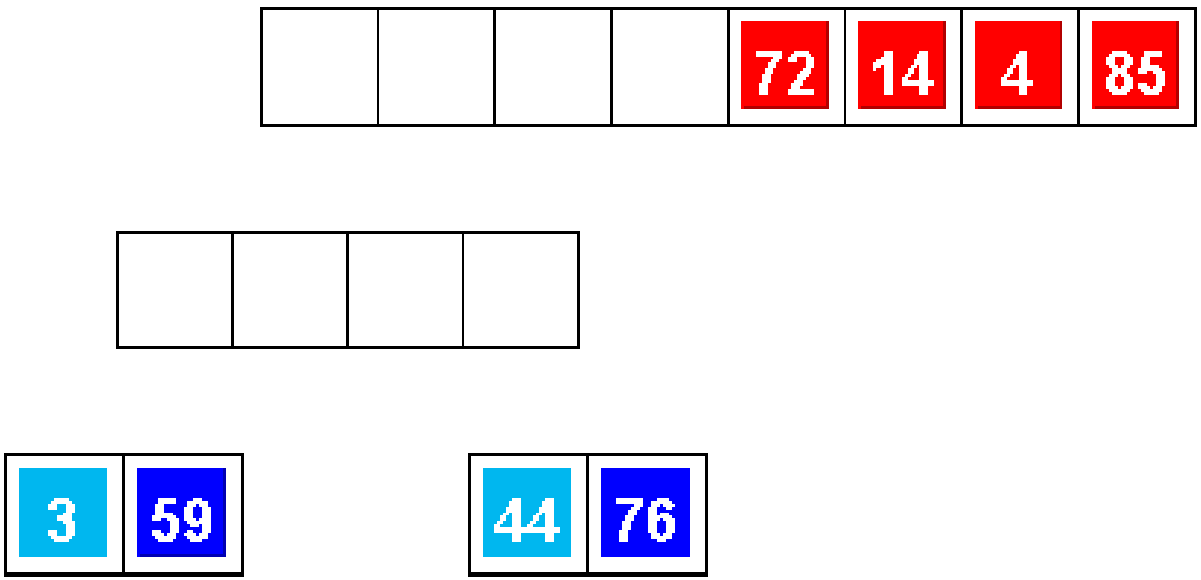
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

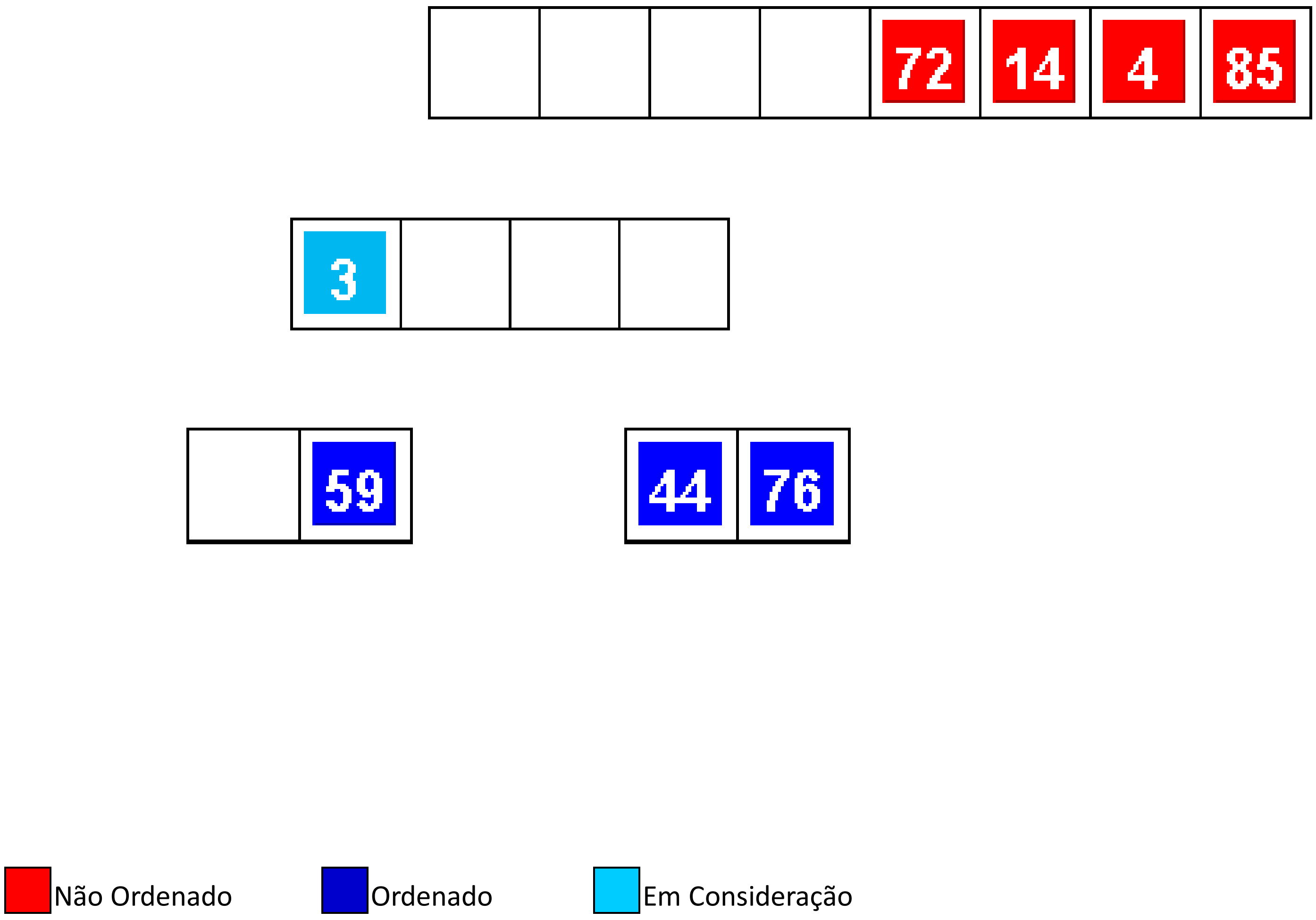
Idéia:




Não Ordenado    Ordenado    Em Consideração

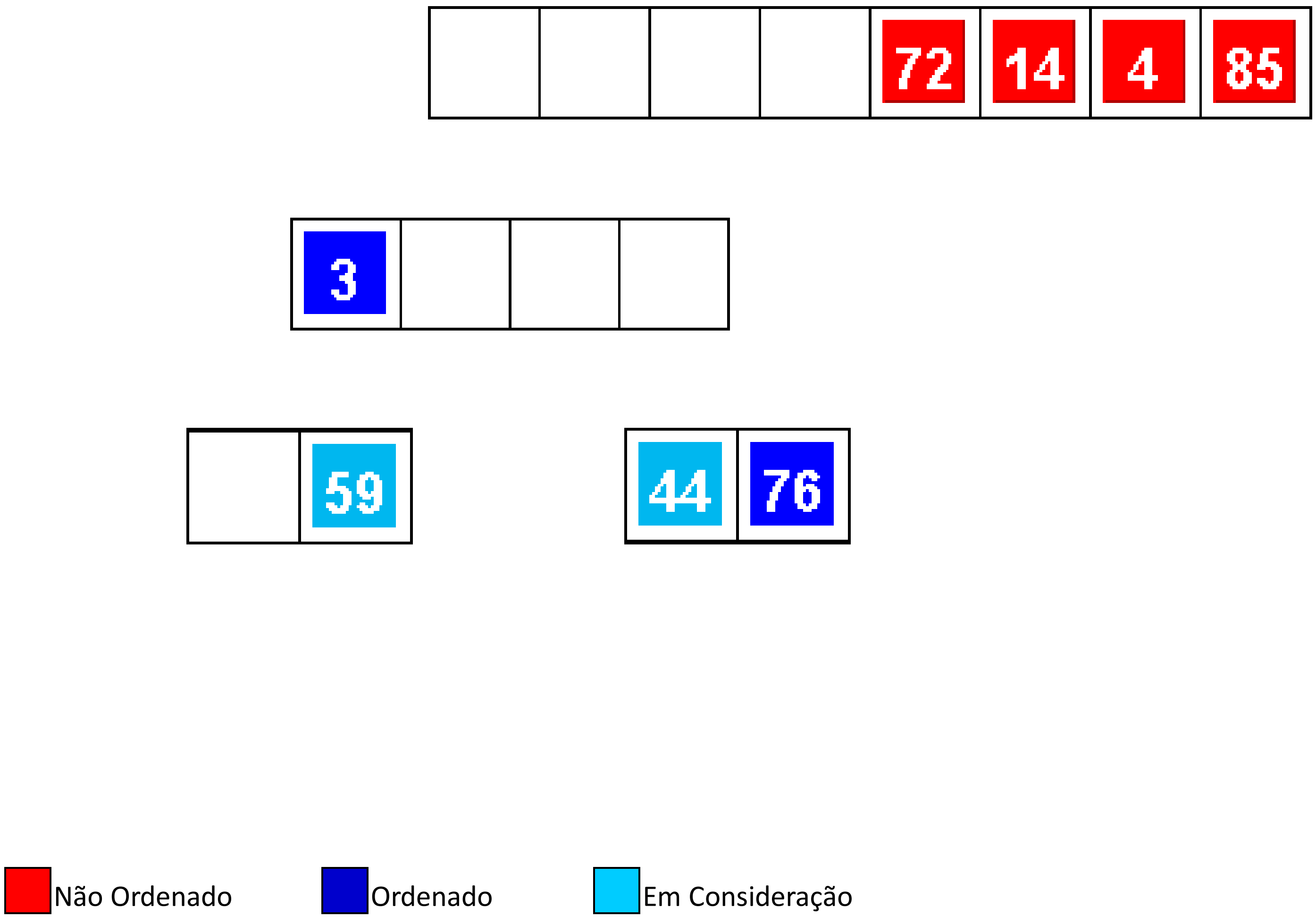
# Merge Sort

Idéia:



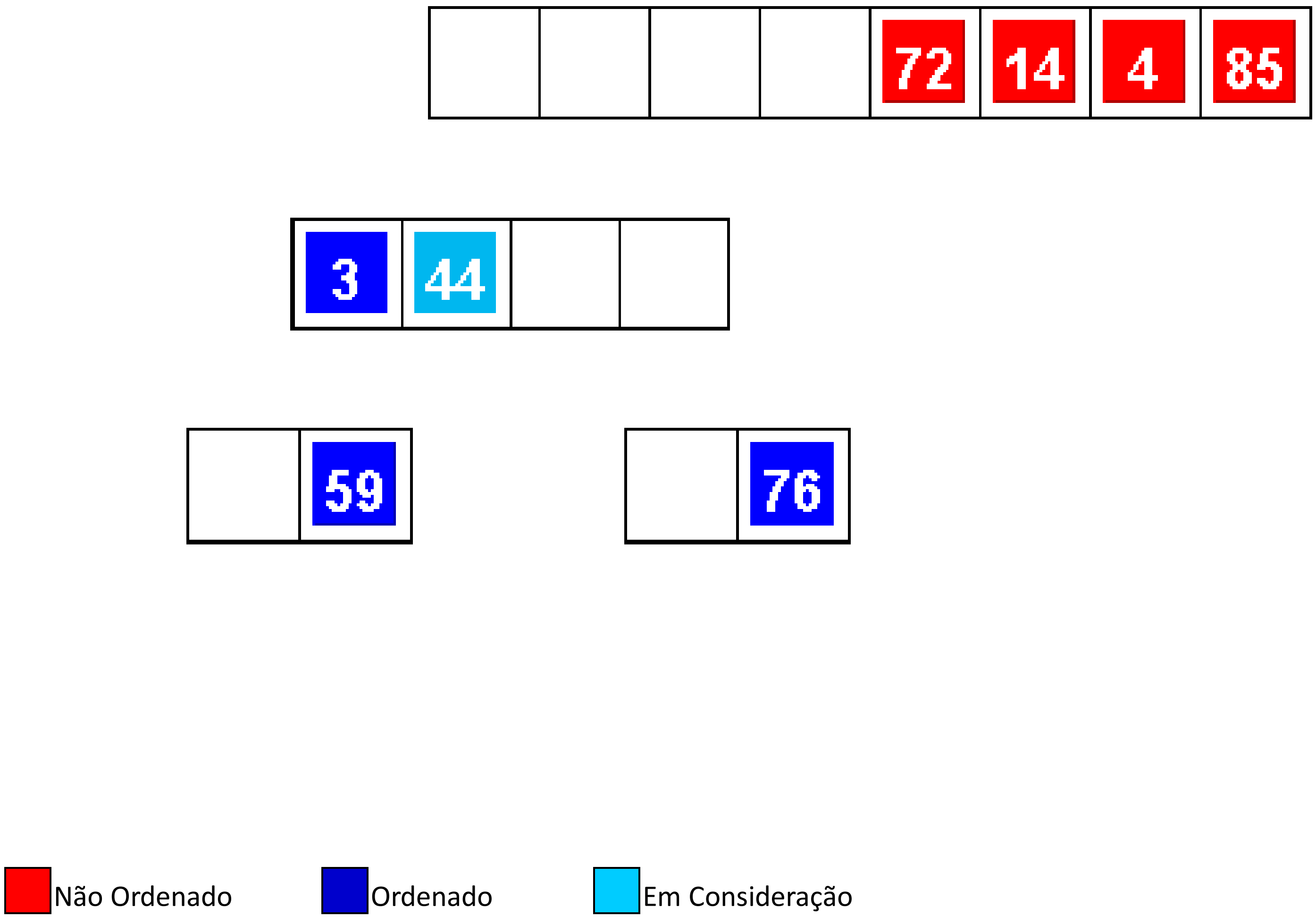
# Merge Sort

Idéia:



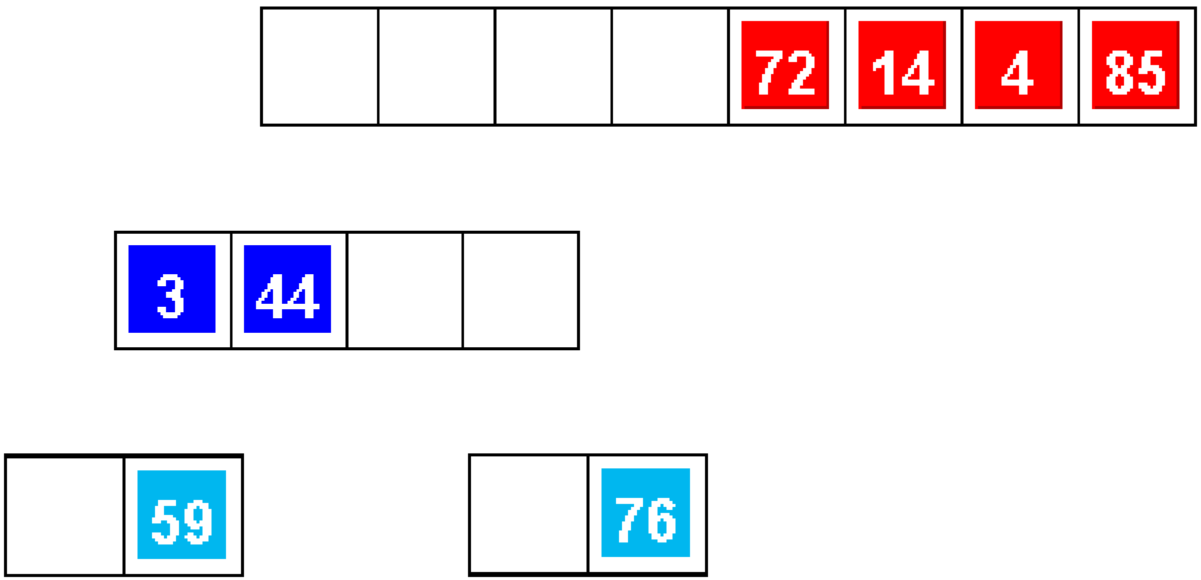
# Merge Sort

Idéia:



# Merge Sort

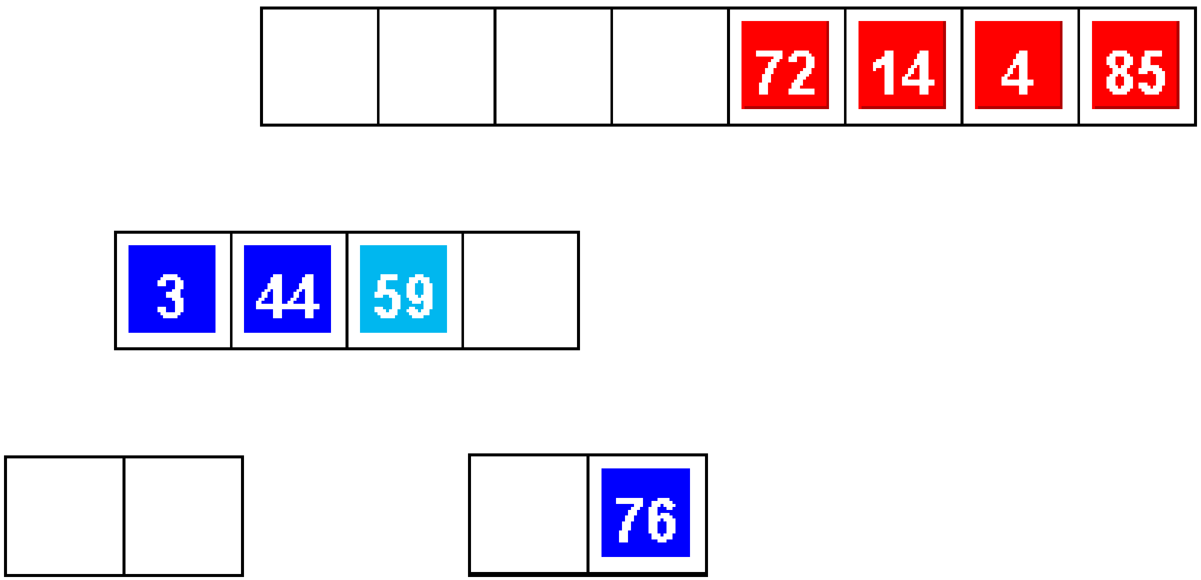
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

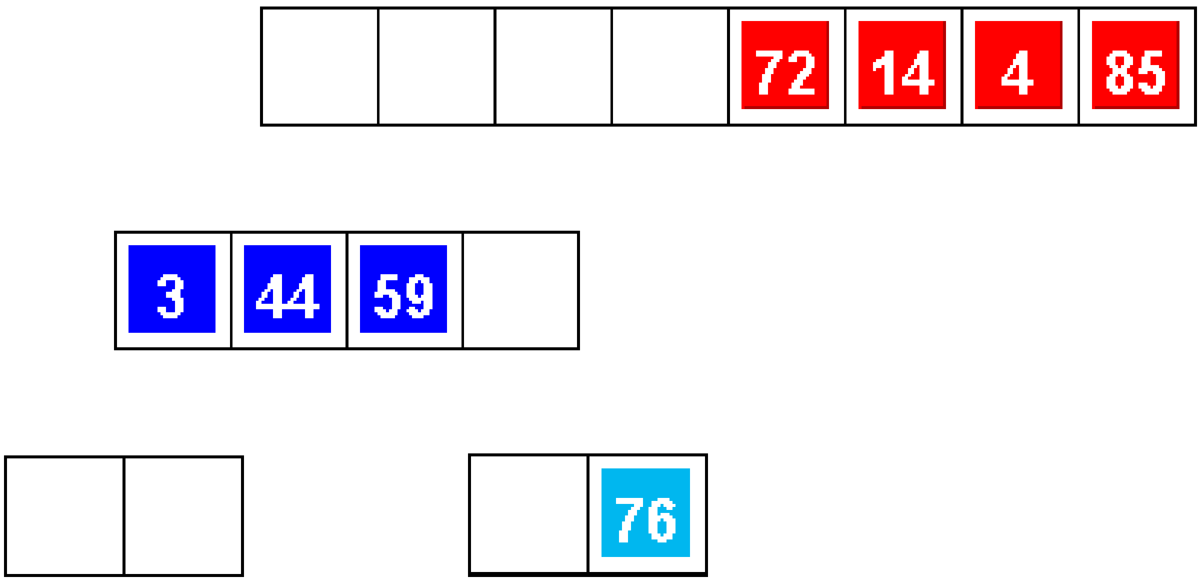
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

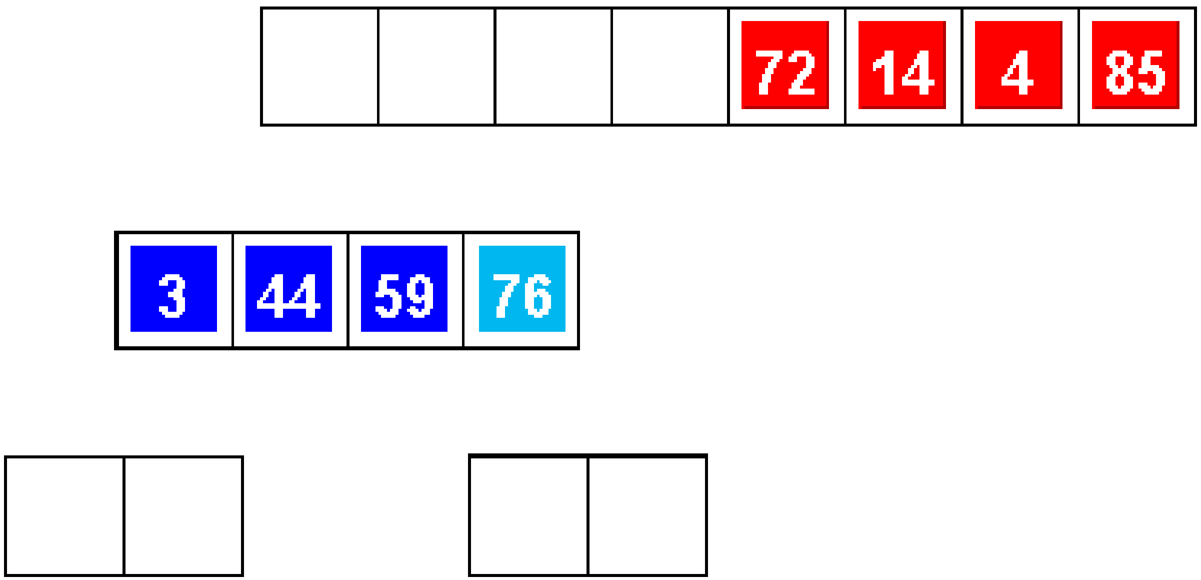
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



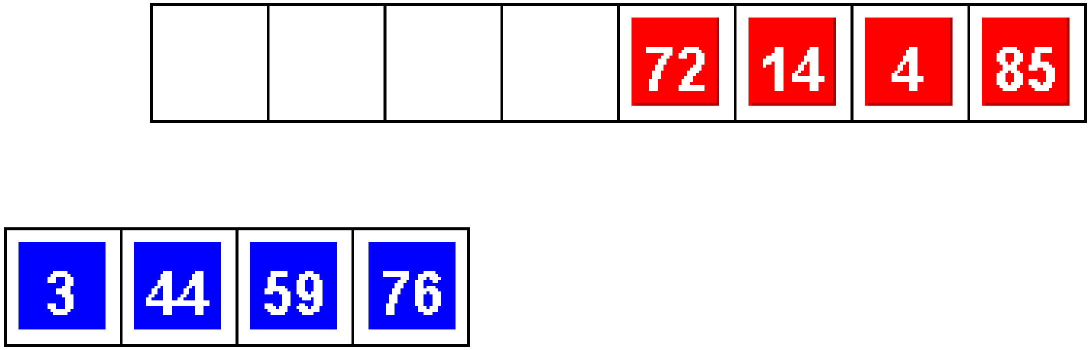

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

Idéia:



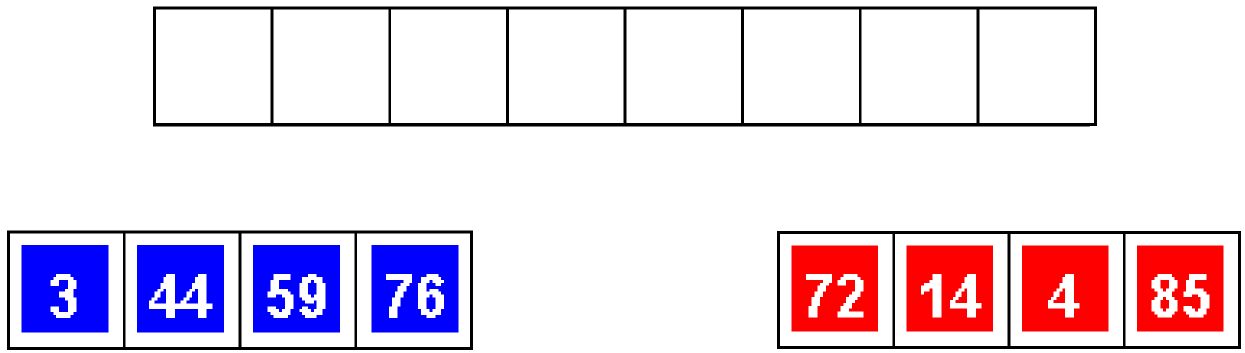

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

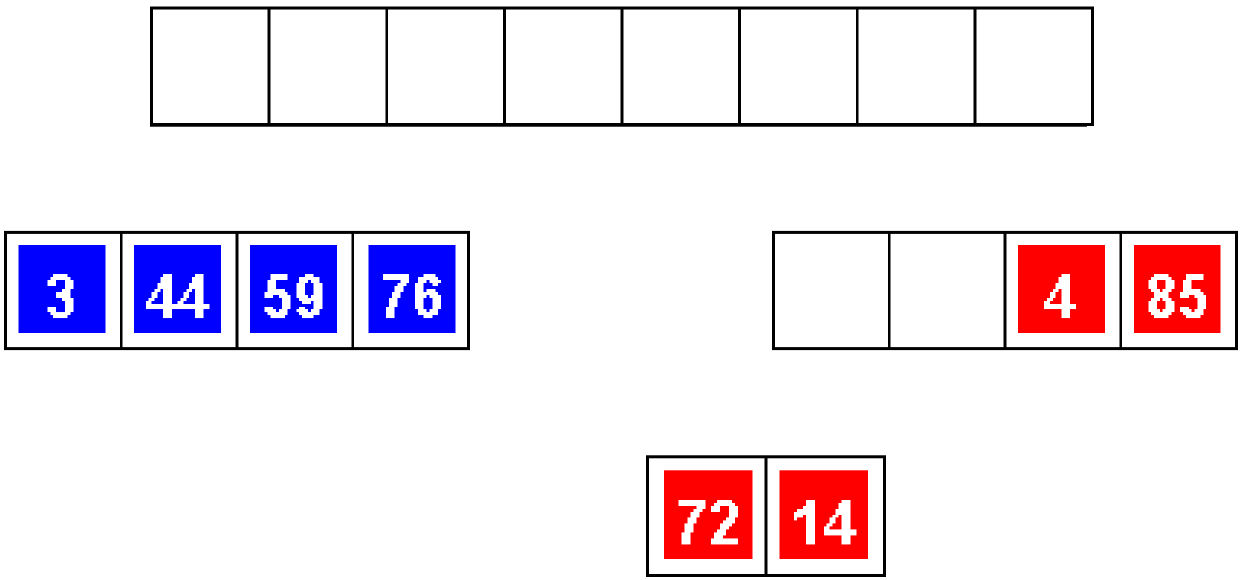
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



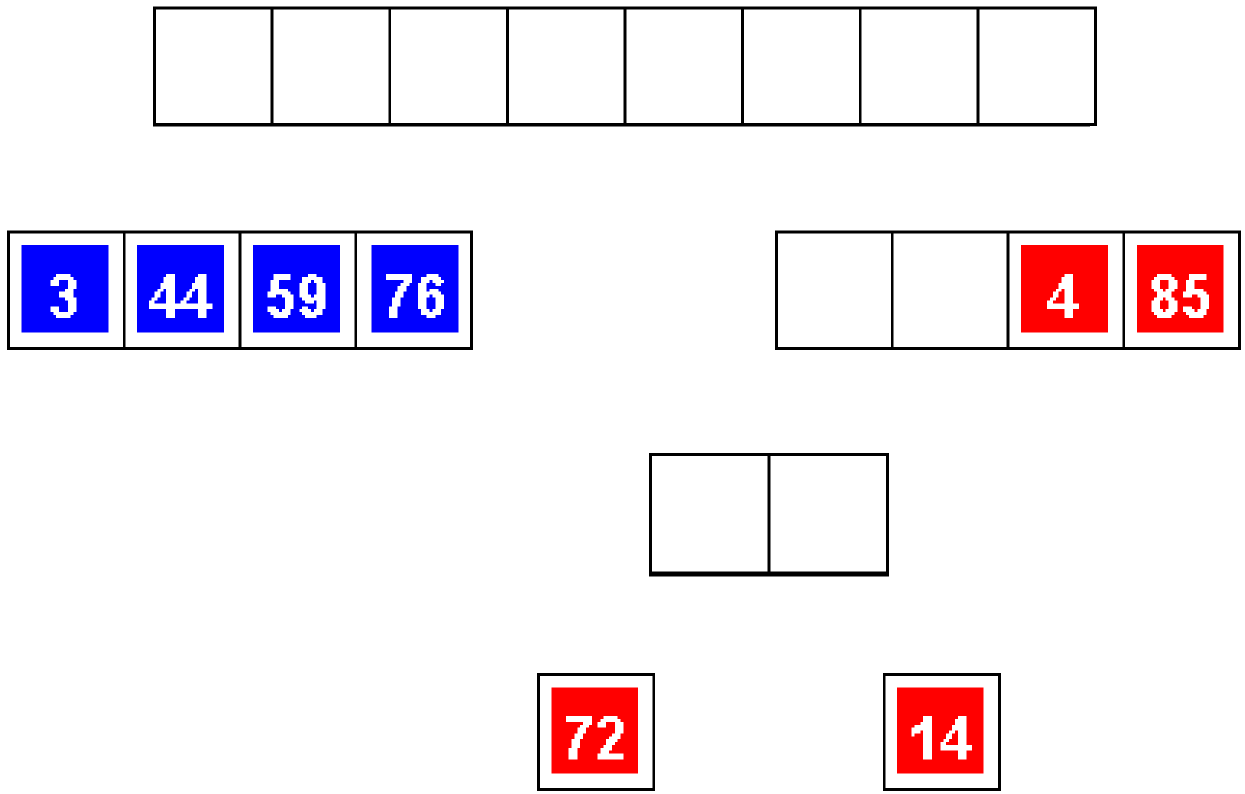

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

Idéia:



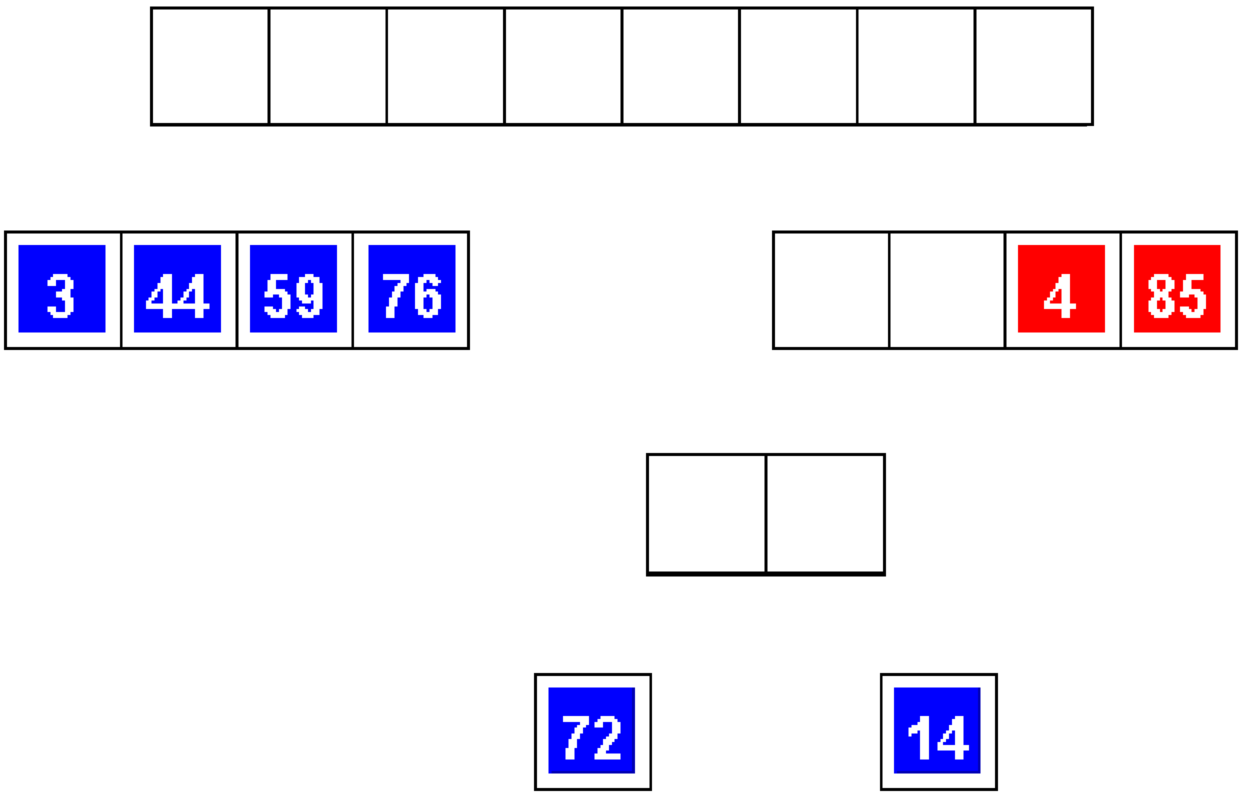

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

Idéia:



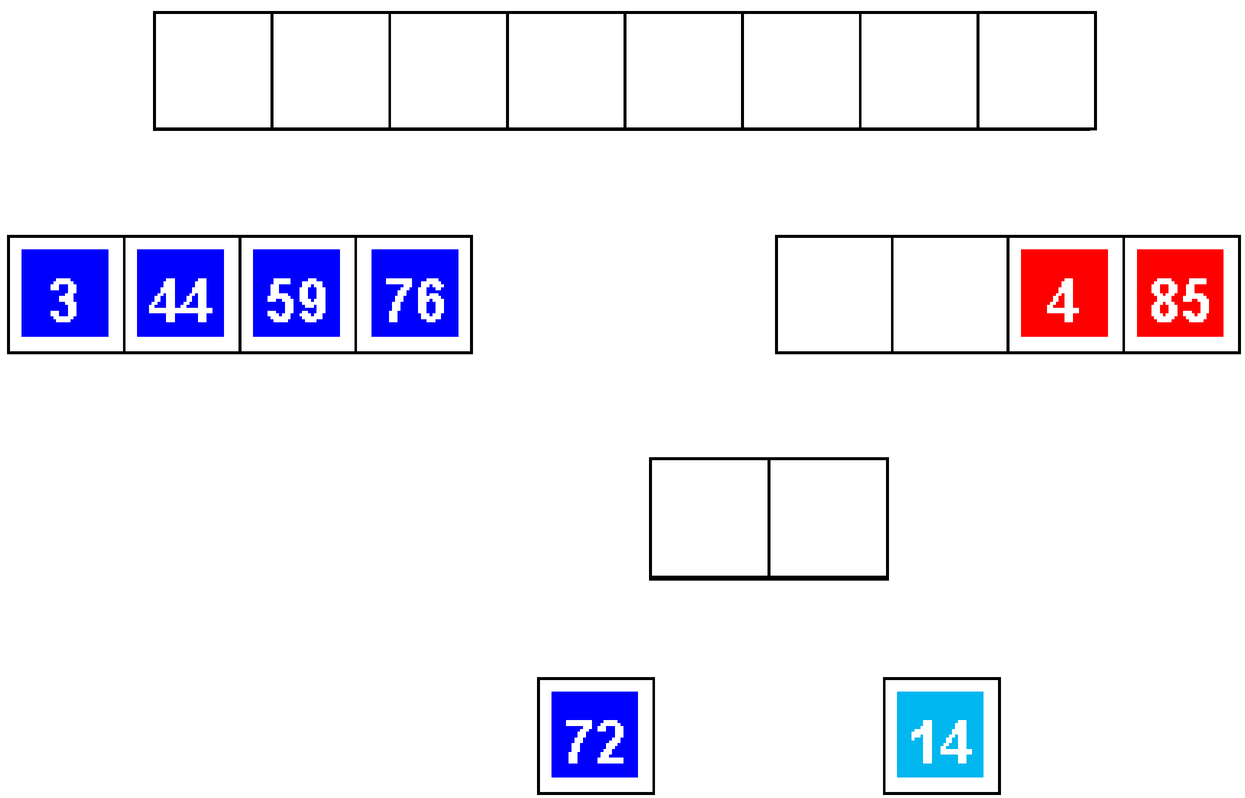

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

Idéia:



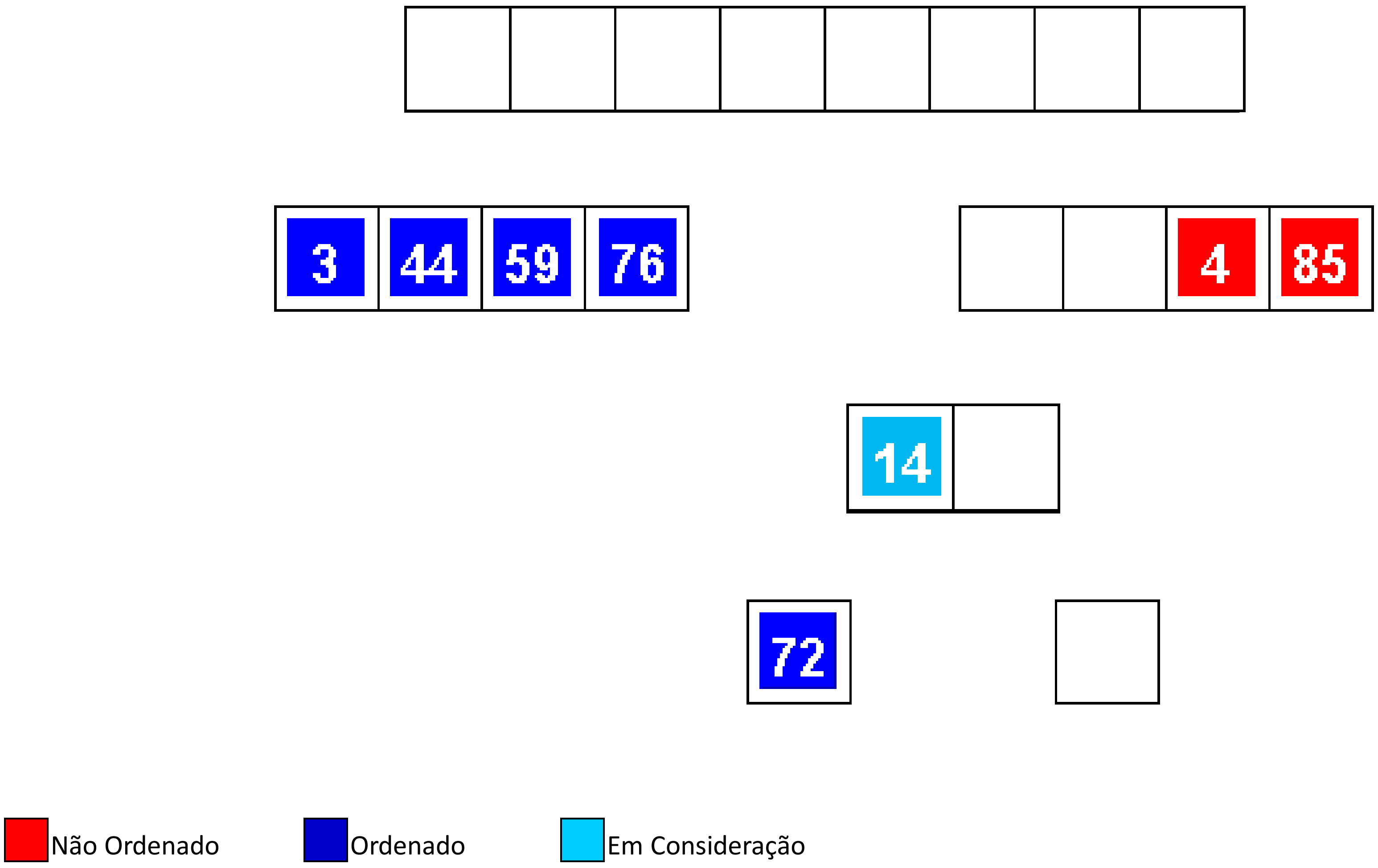

Não Ordenado

Ordenado

Em Consideração

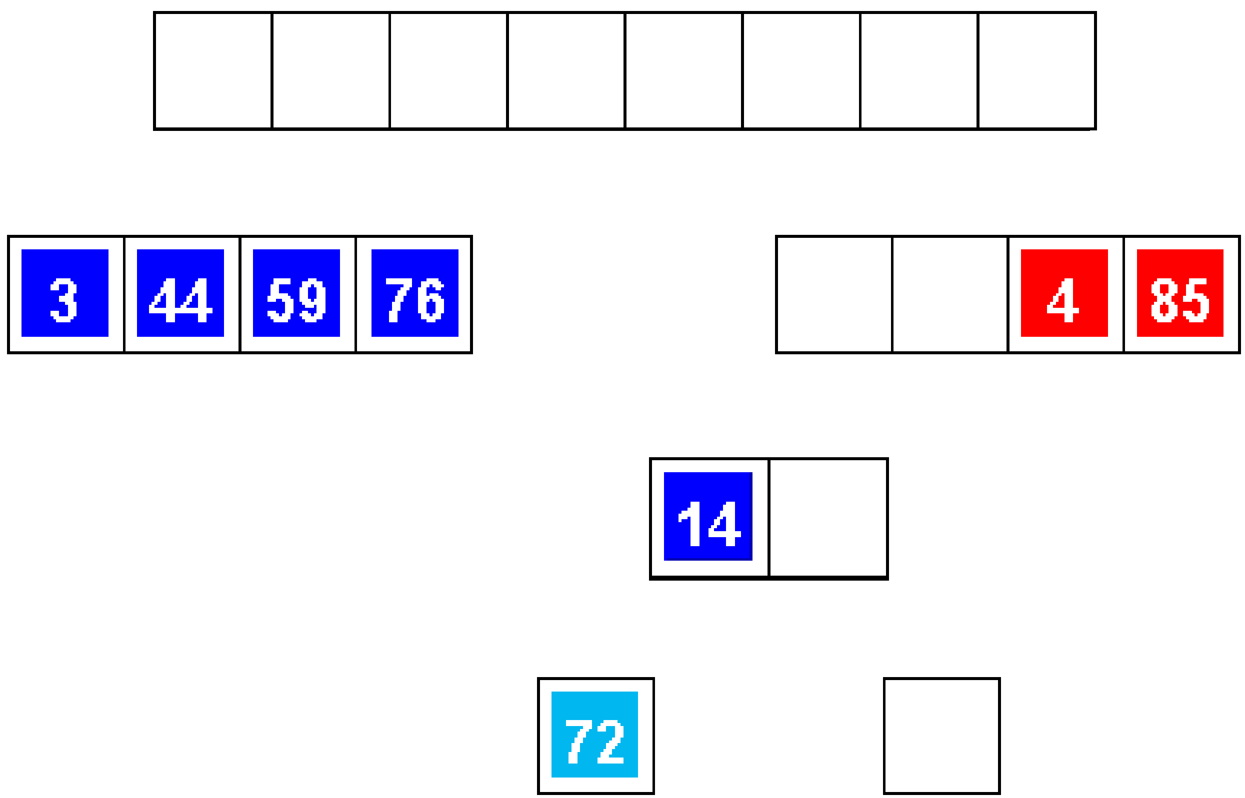
# Merge Sort

Idéia:



# Merge Sort

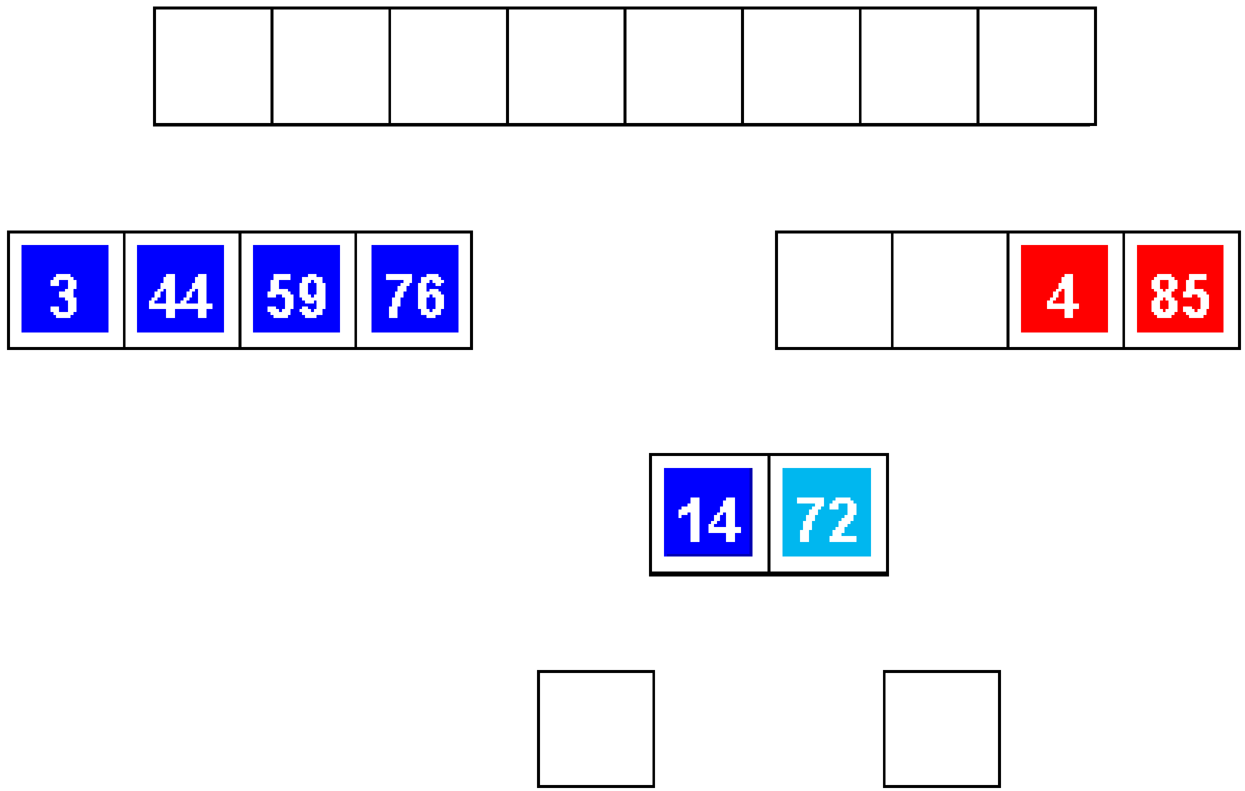
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



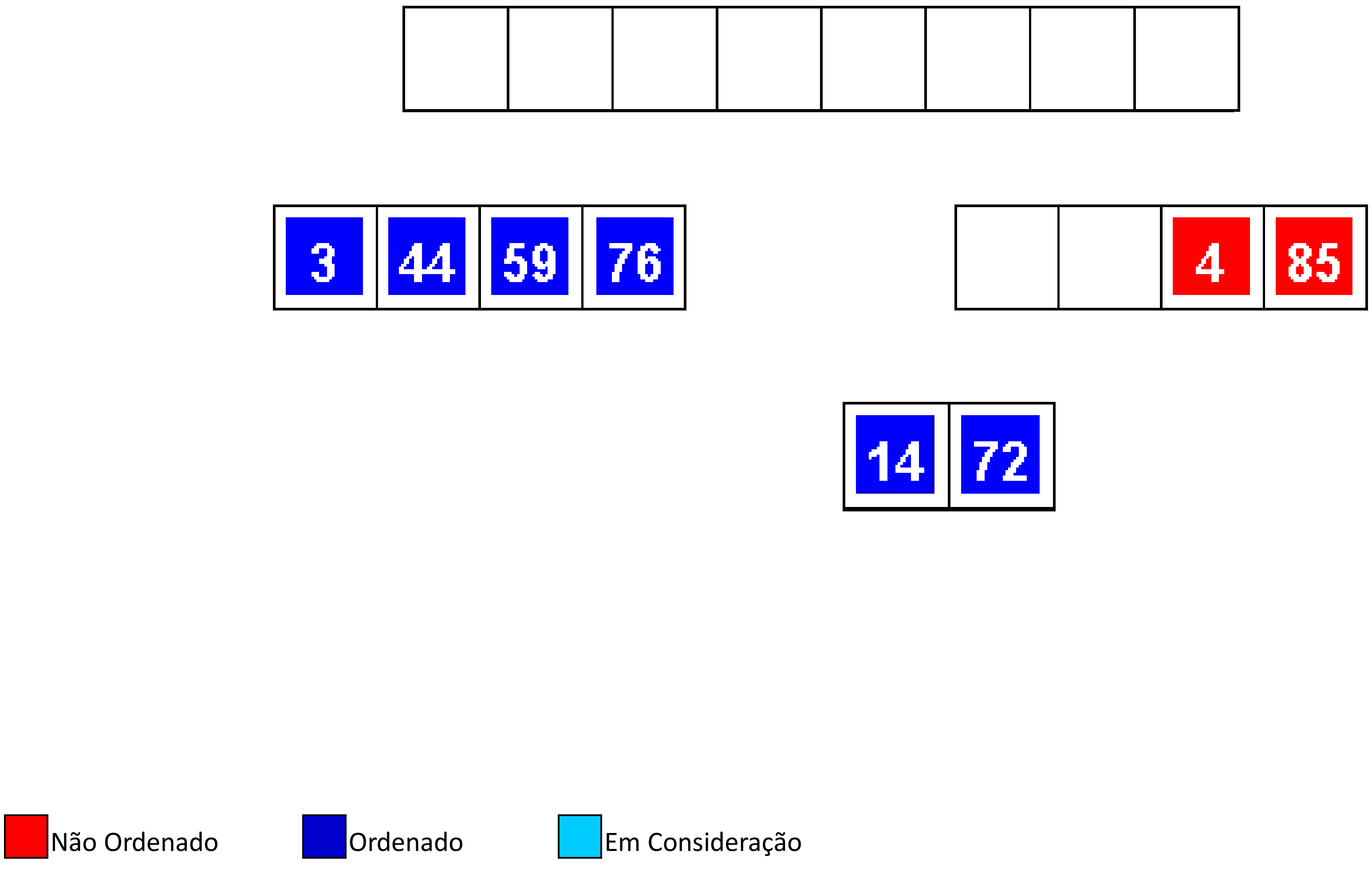

Não Ordenado

Ordenado

Em Consideração

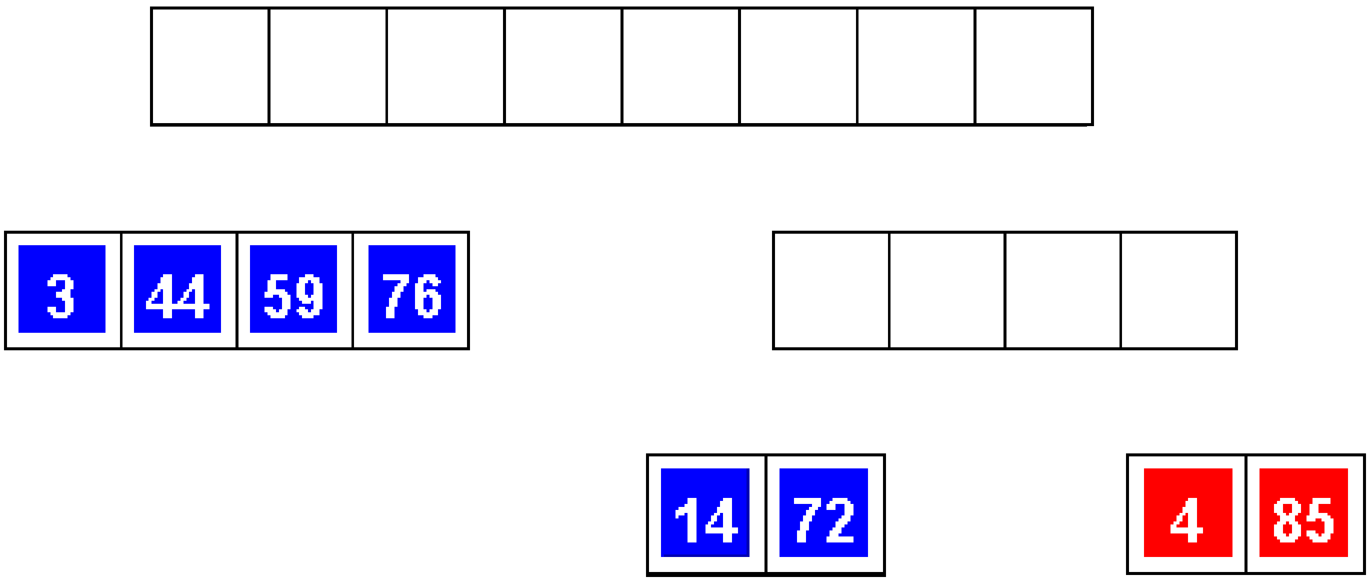
# Merge Sort

Idéia:



# Merge Sort

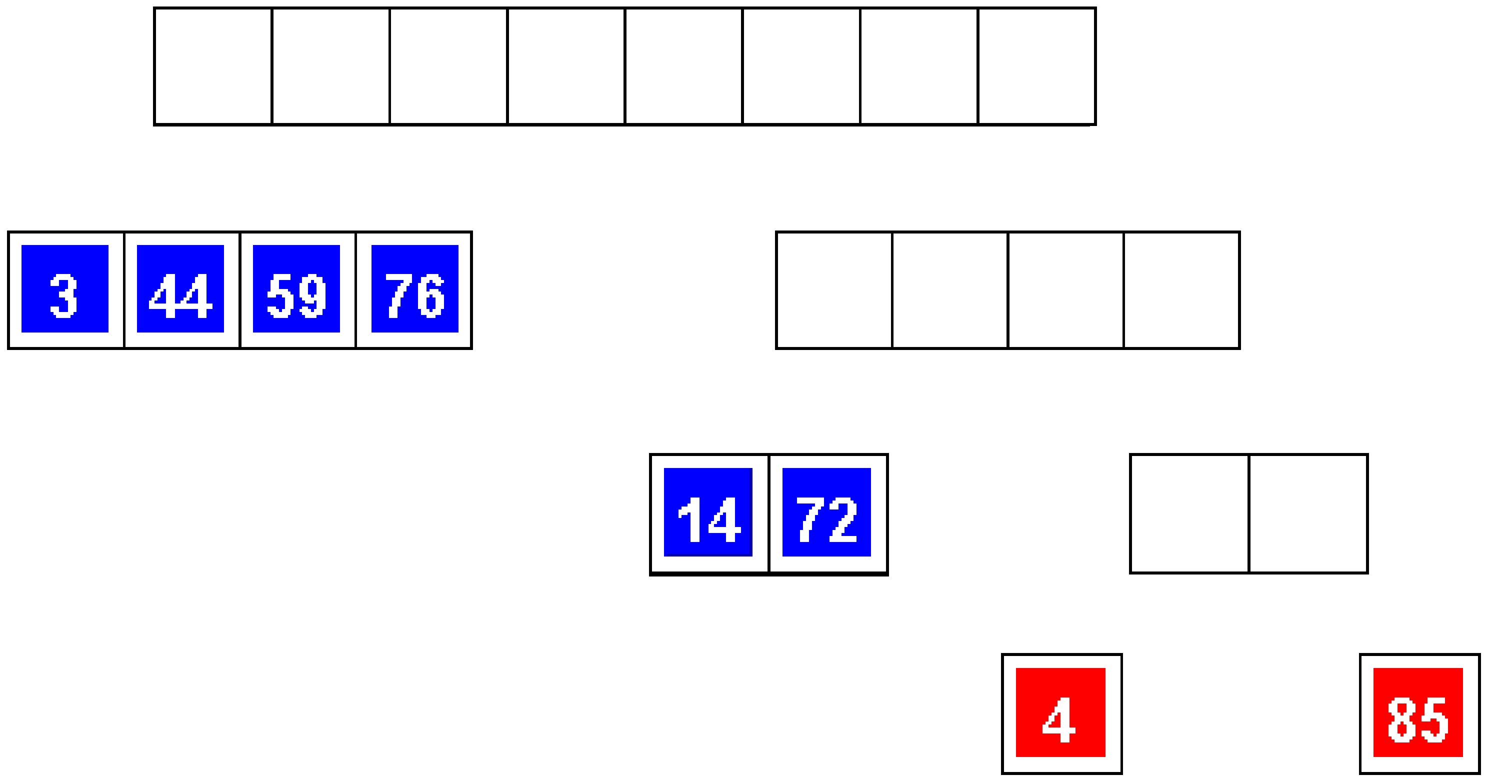
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

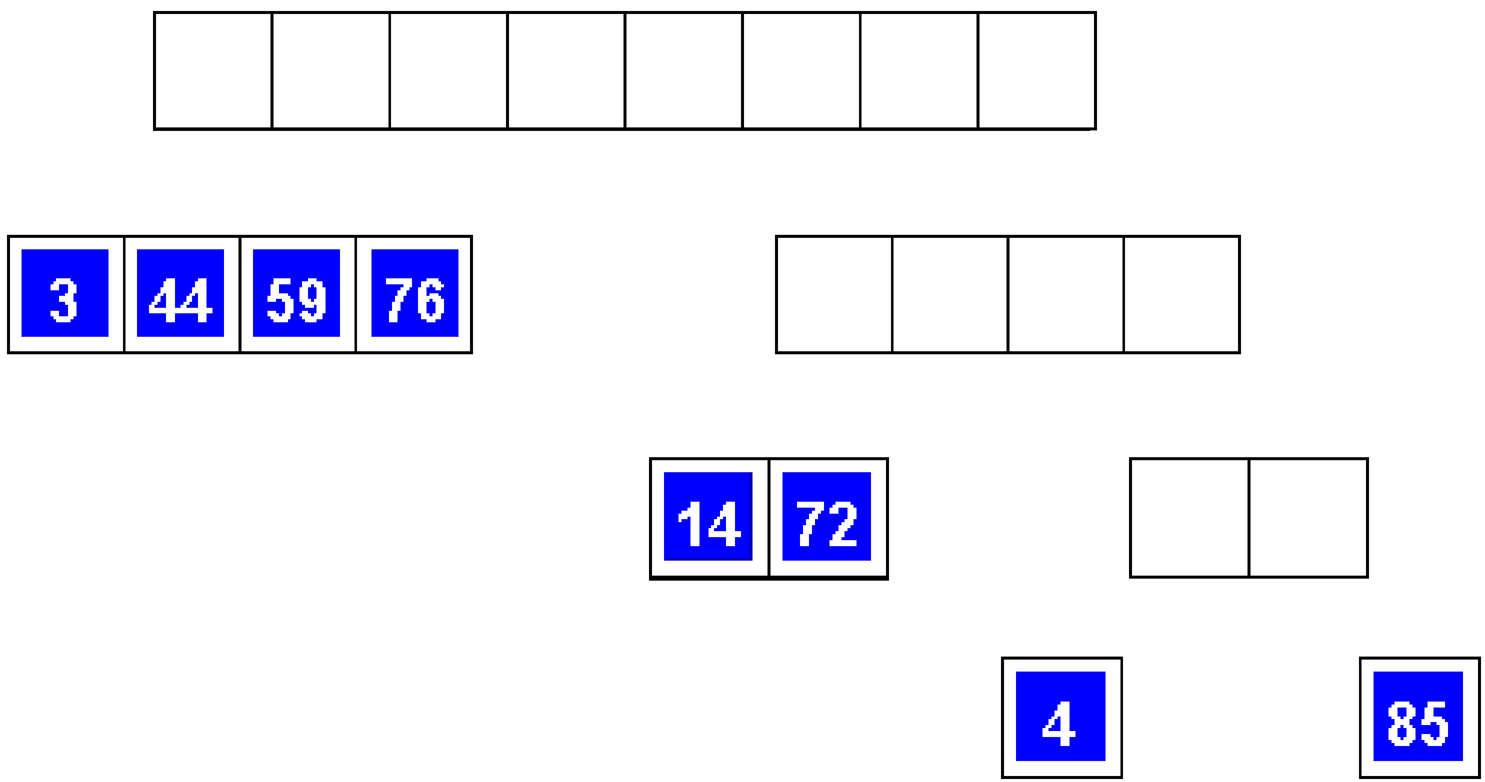
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



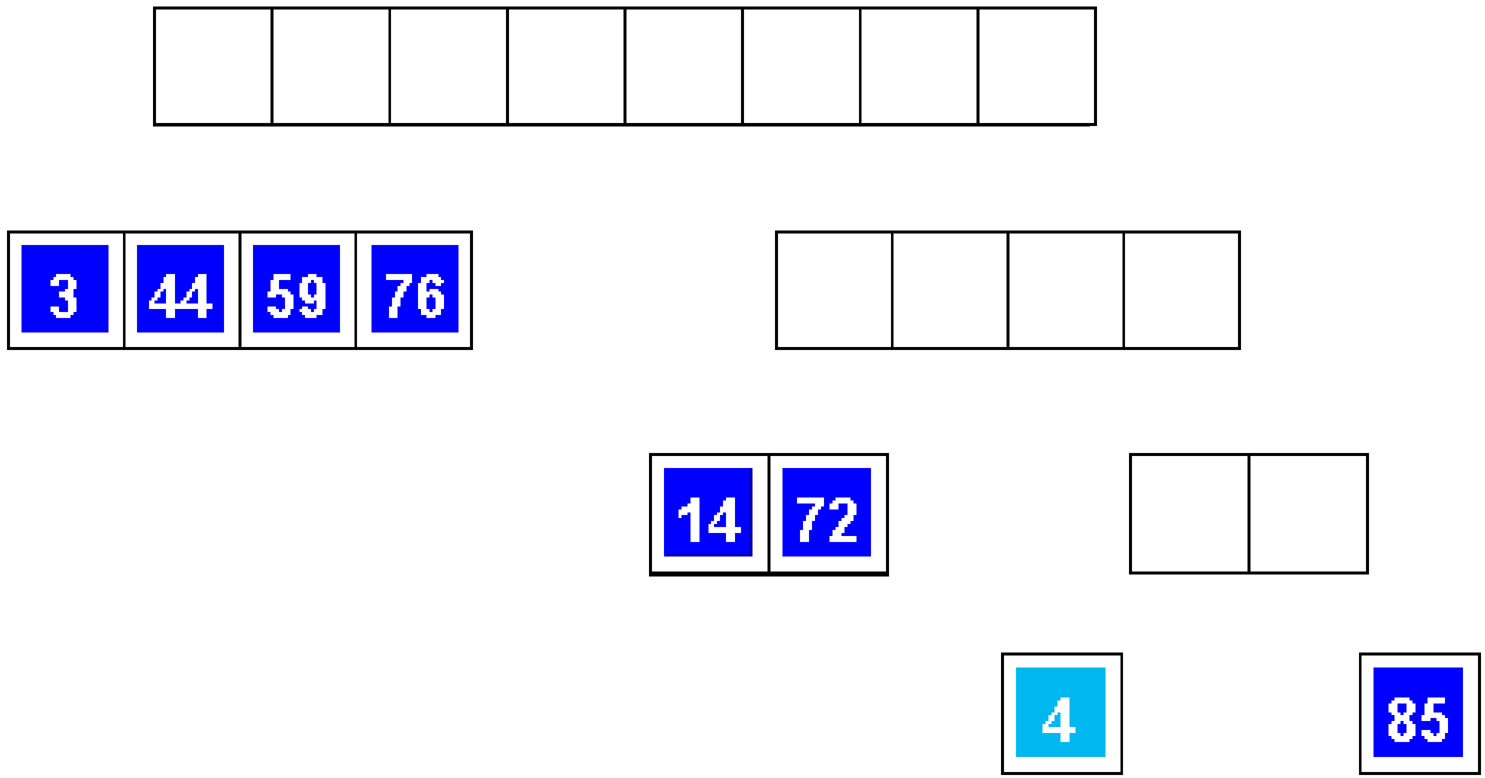

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

Idéia:



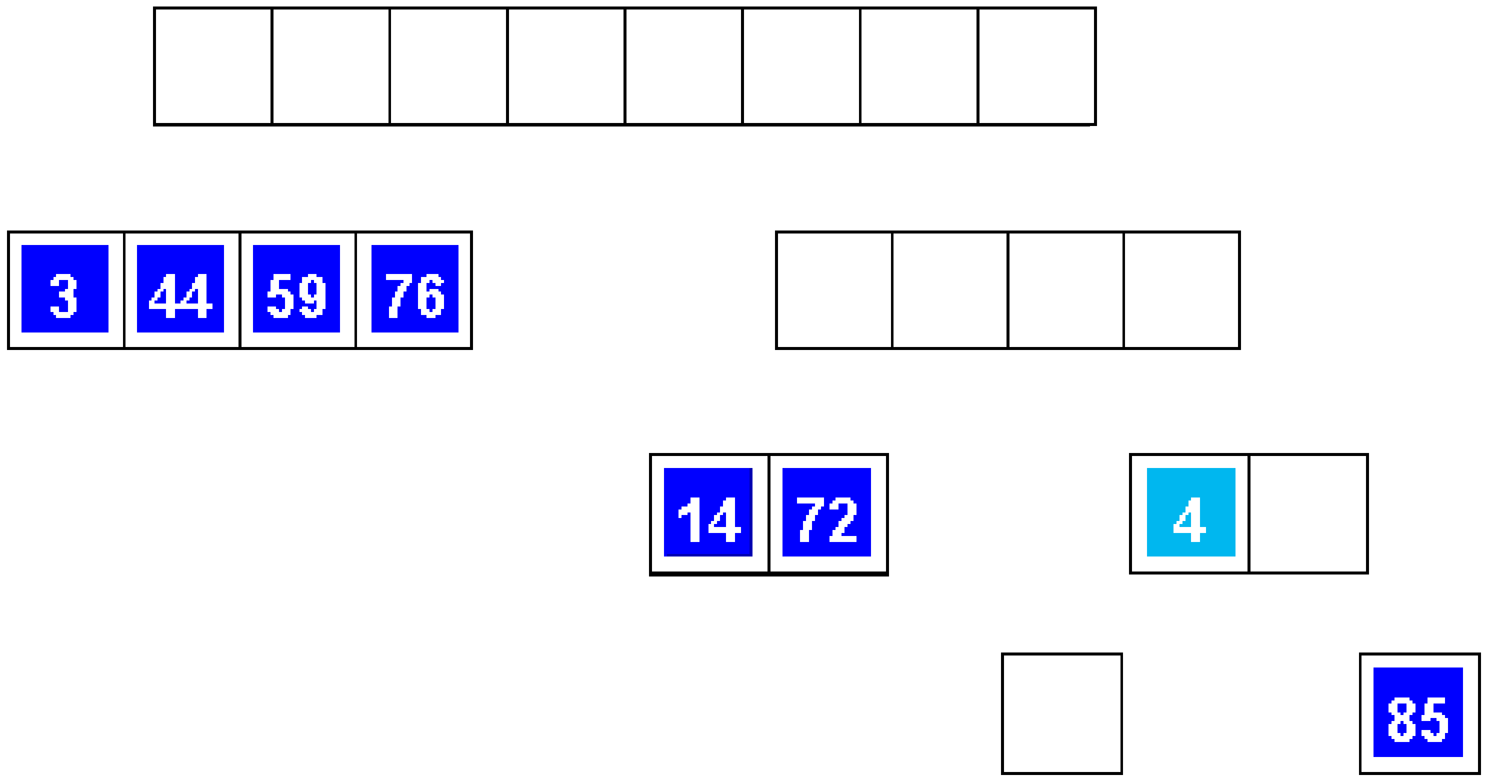

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

Idéia:



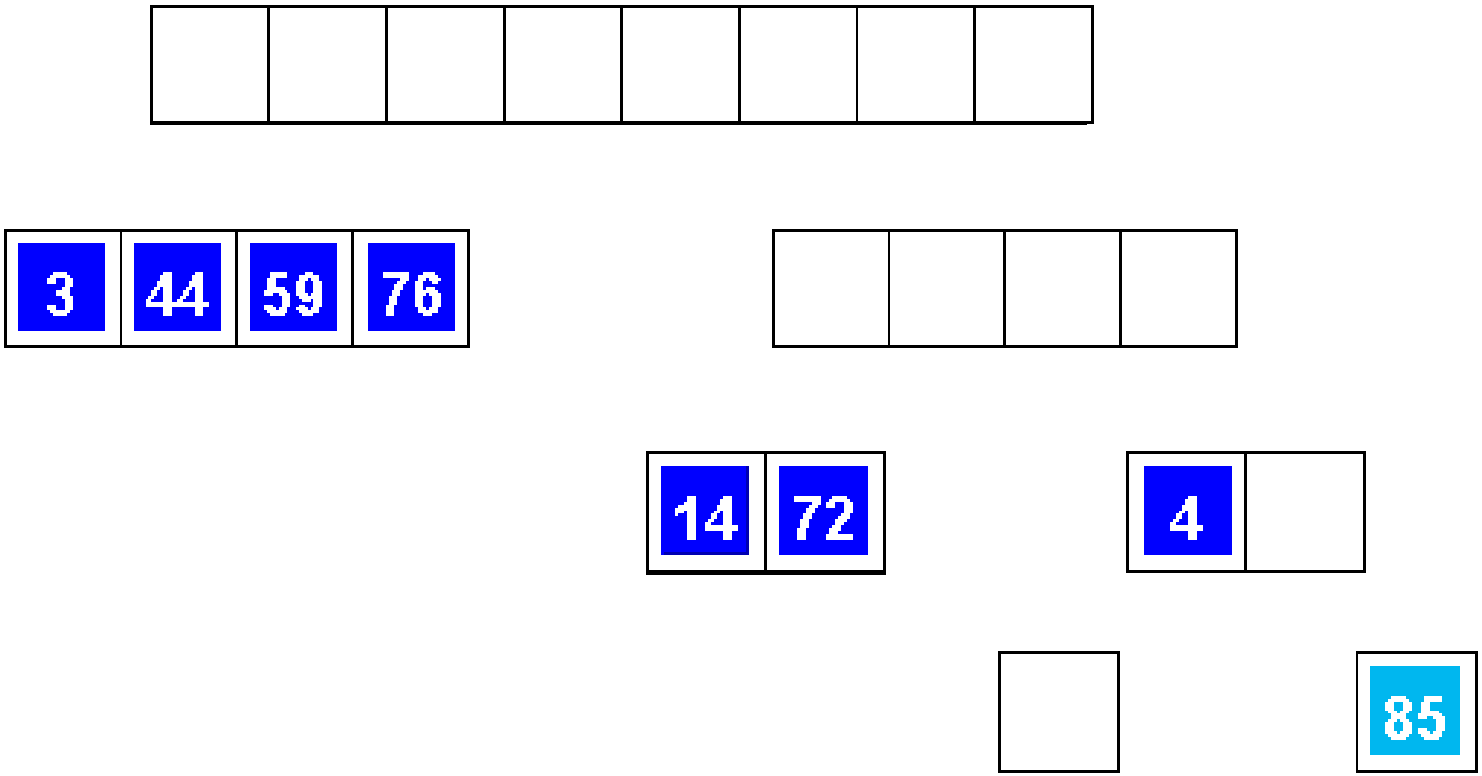

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

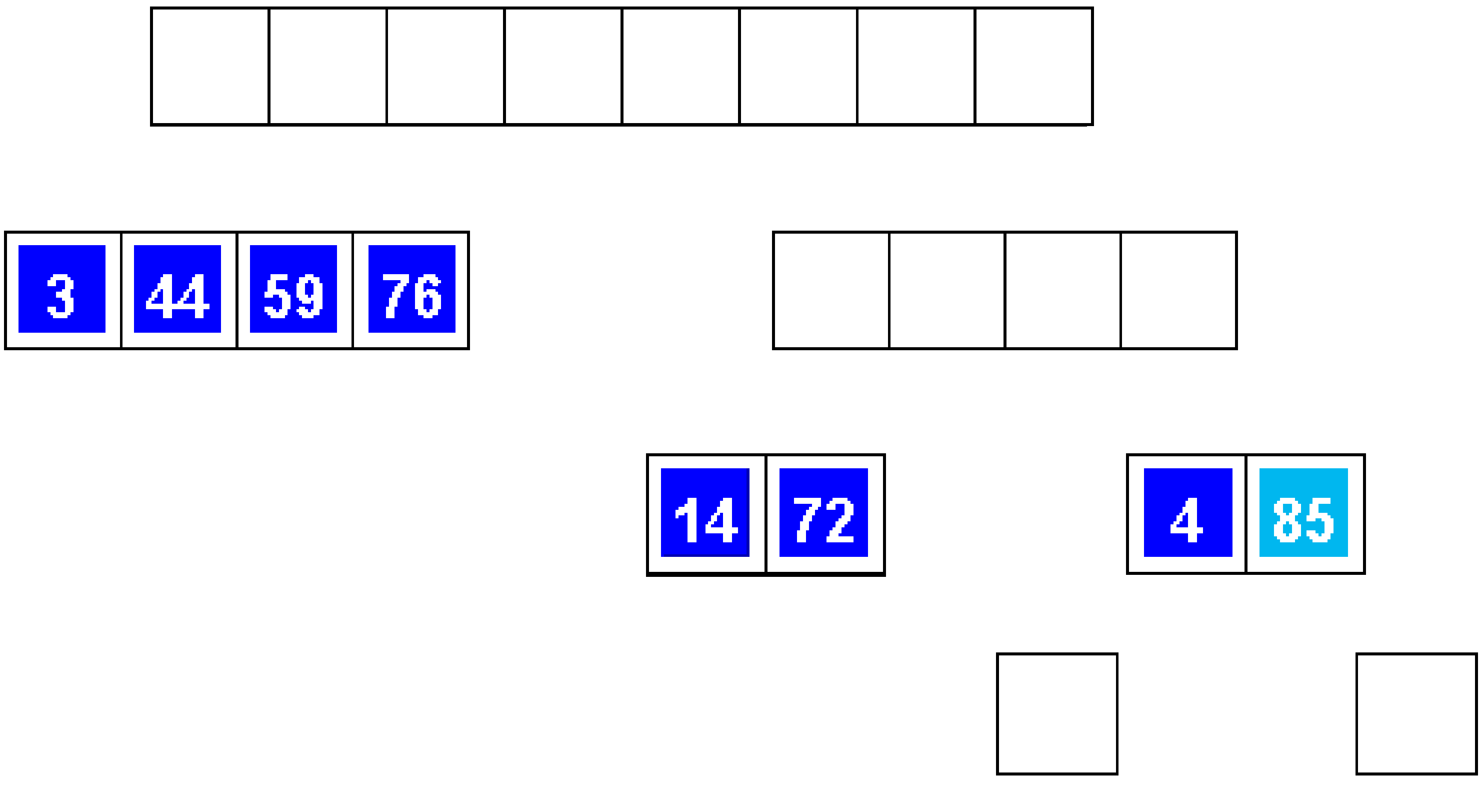
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

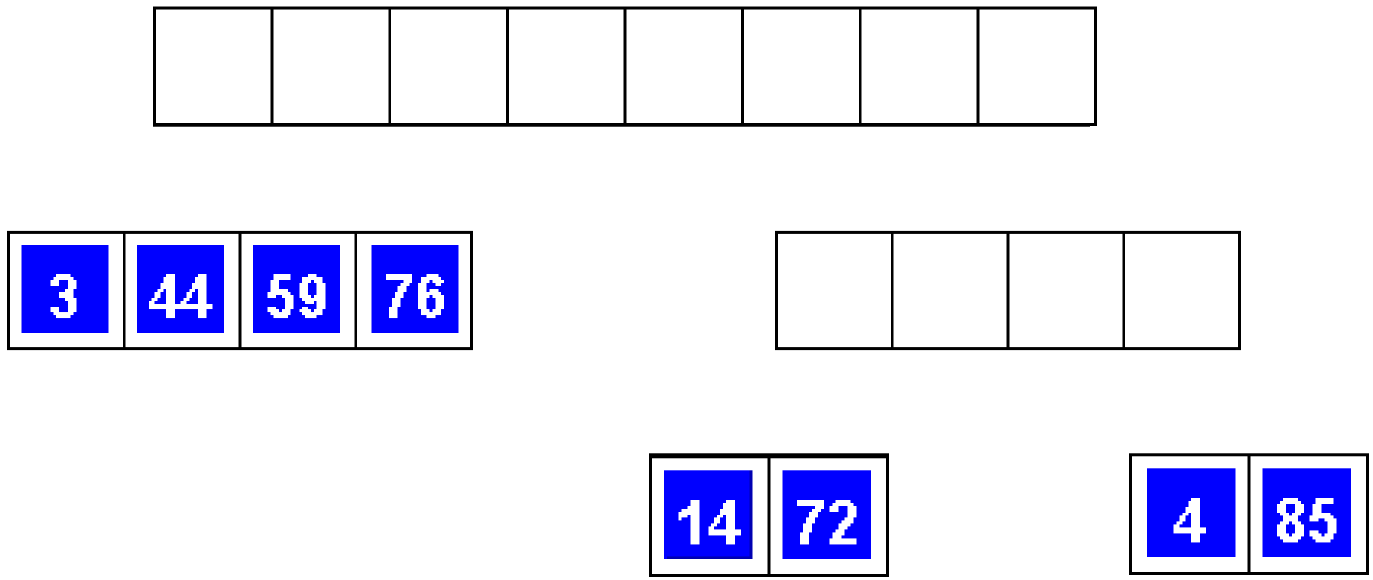
Idéia:




Não Ordenado    Ordenado    Em Consideração

# Merge Sort

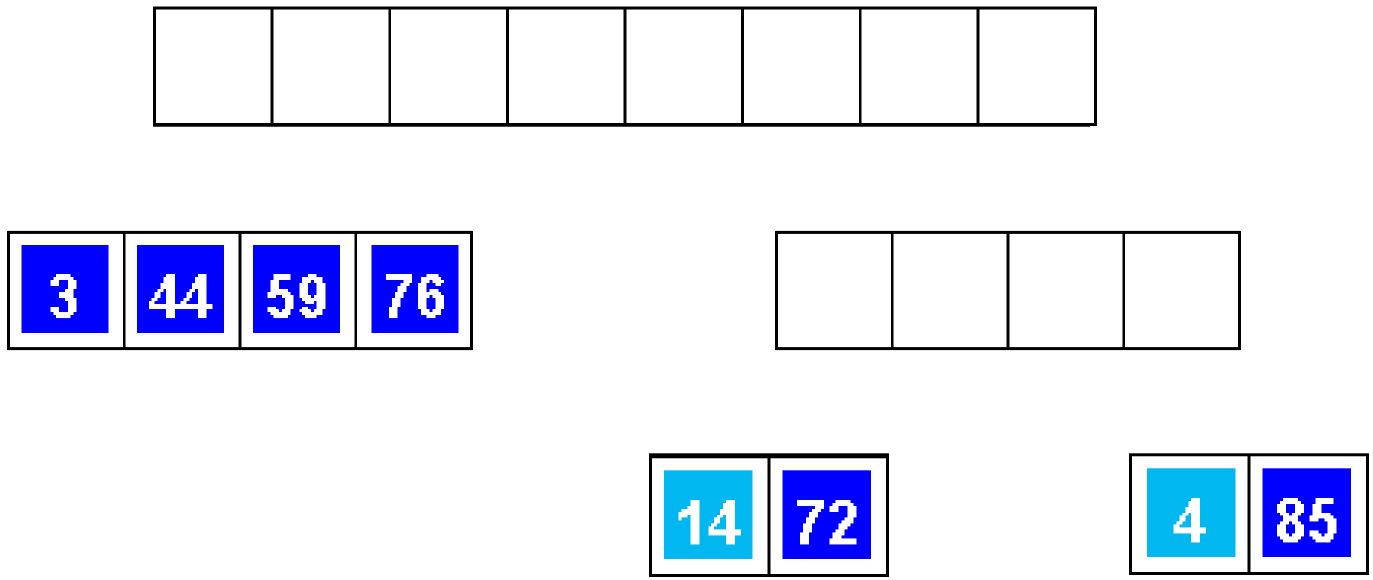
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



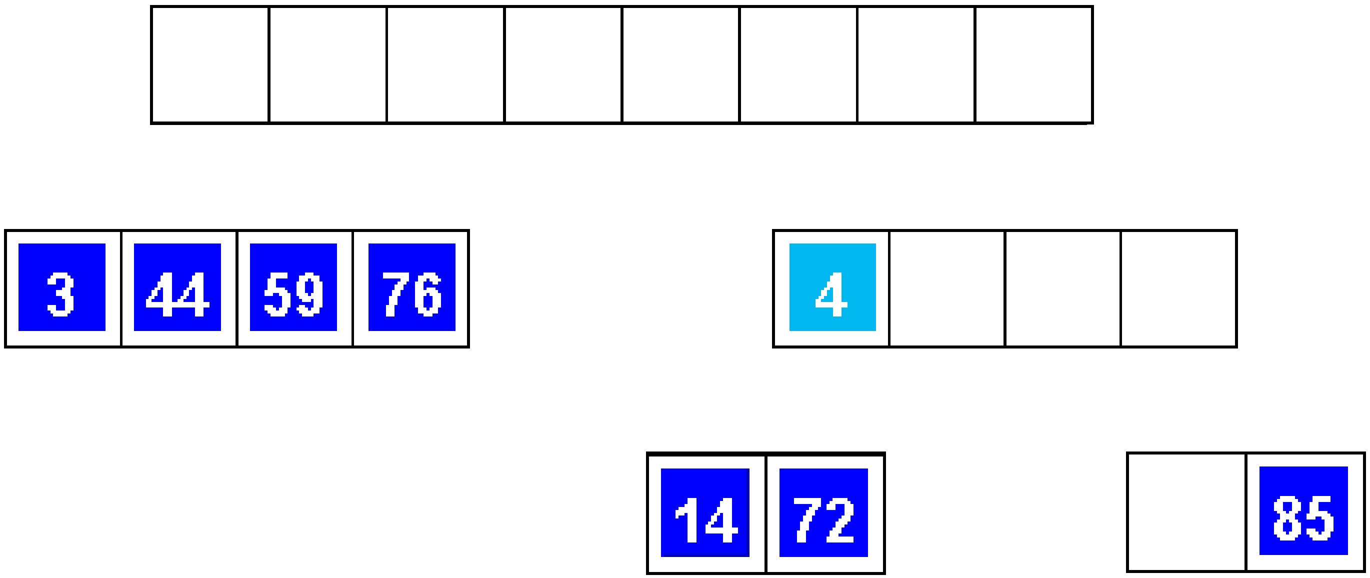

Não Ordenado

Ordenado

Em Consideração

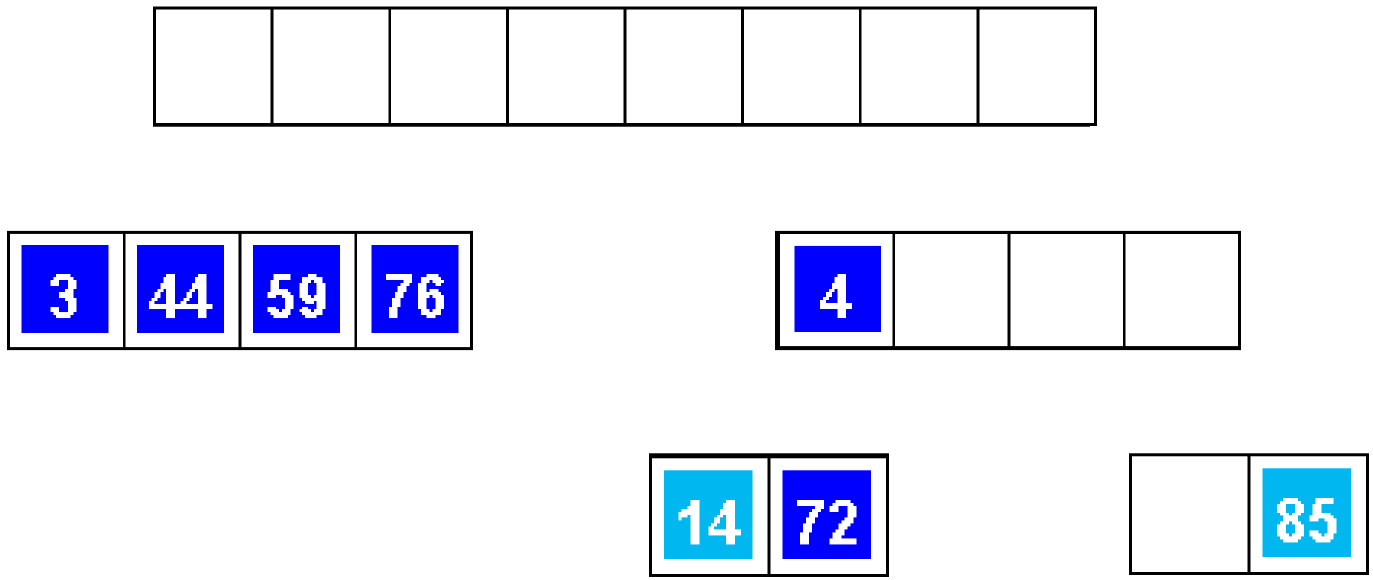
# Merge Sort

Idéia:




# Merge Sort

Idéia:



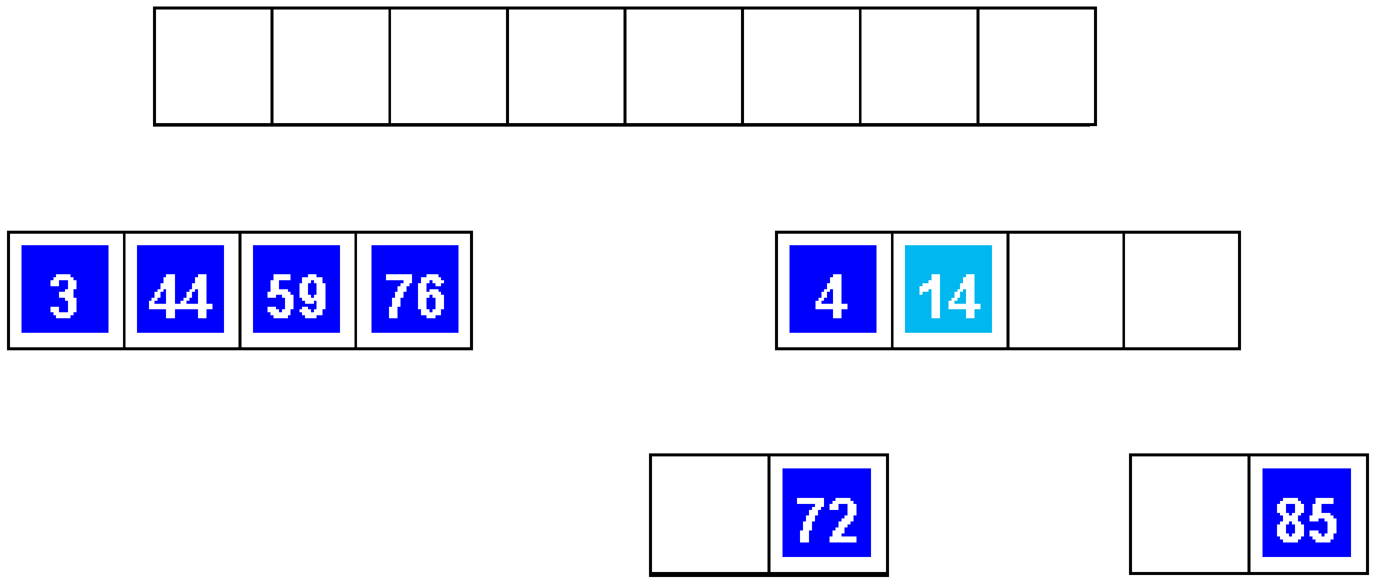

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

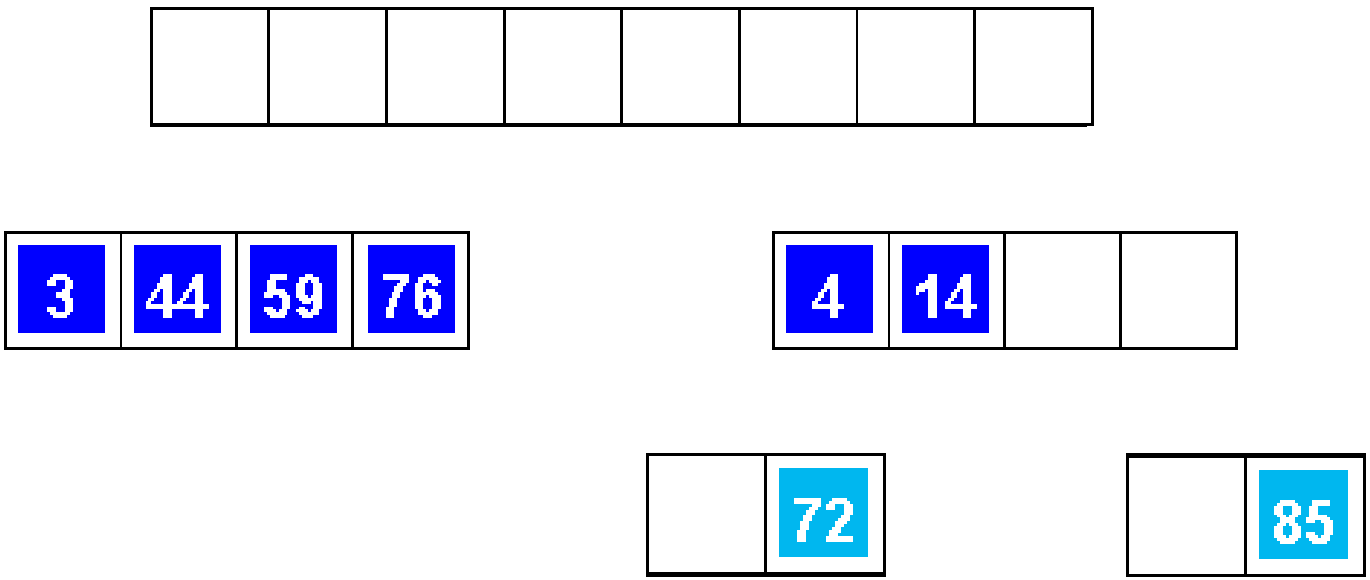
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

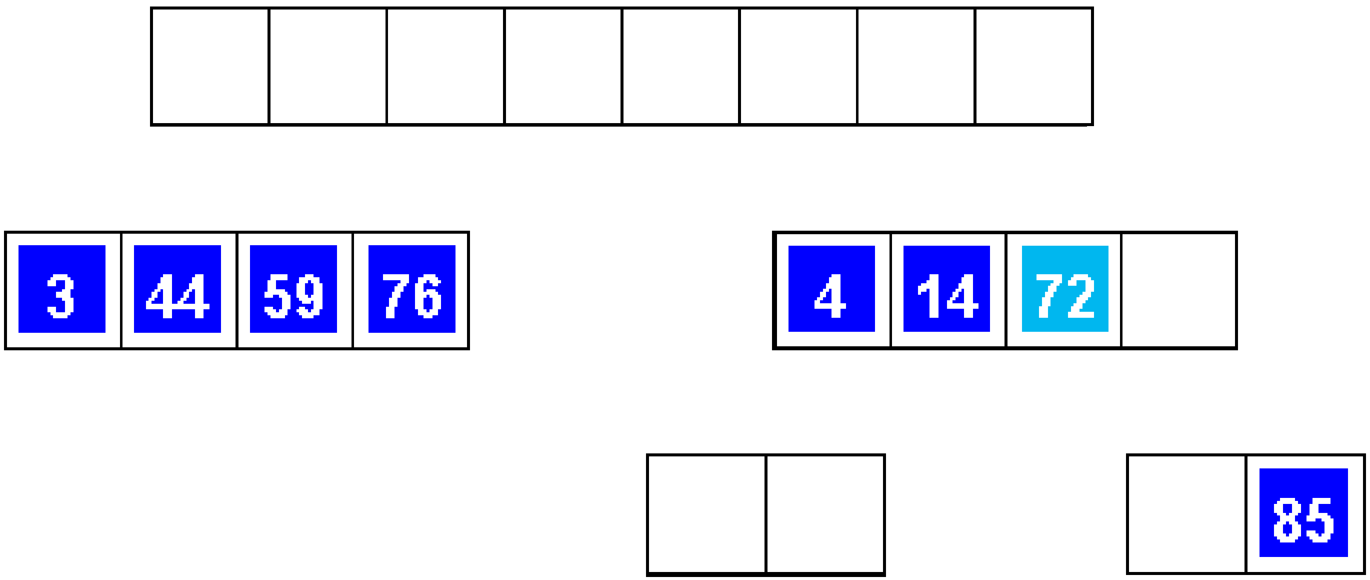
Idéia:




Não Ordenado
Ordenado
Em Consideração

# Merge Sort

Idéia:



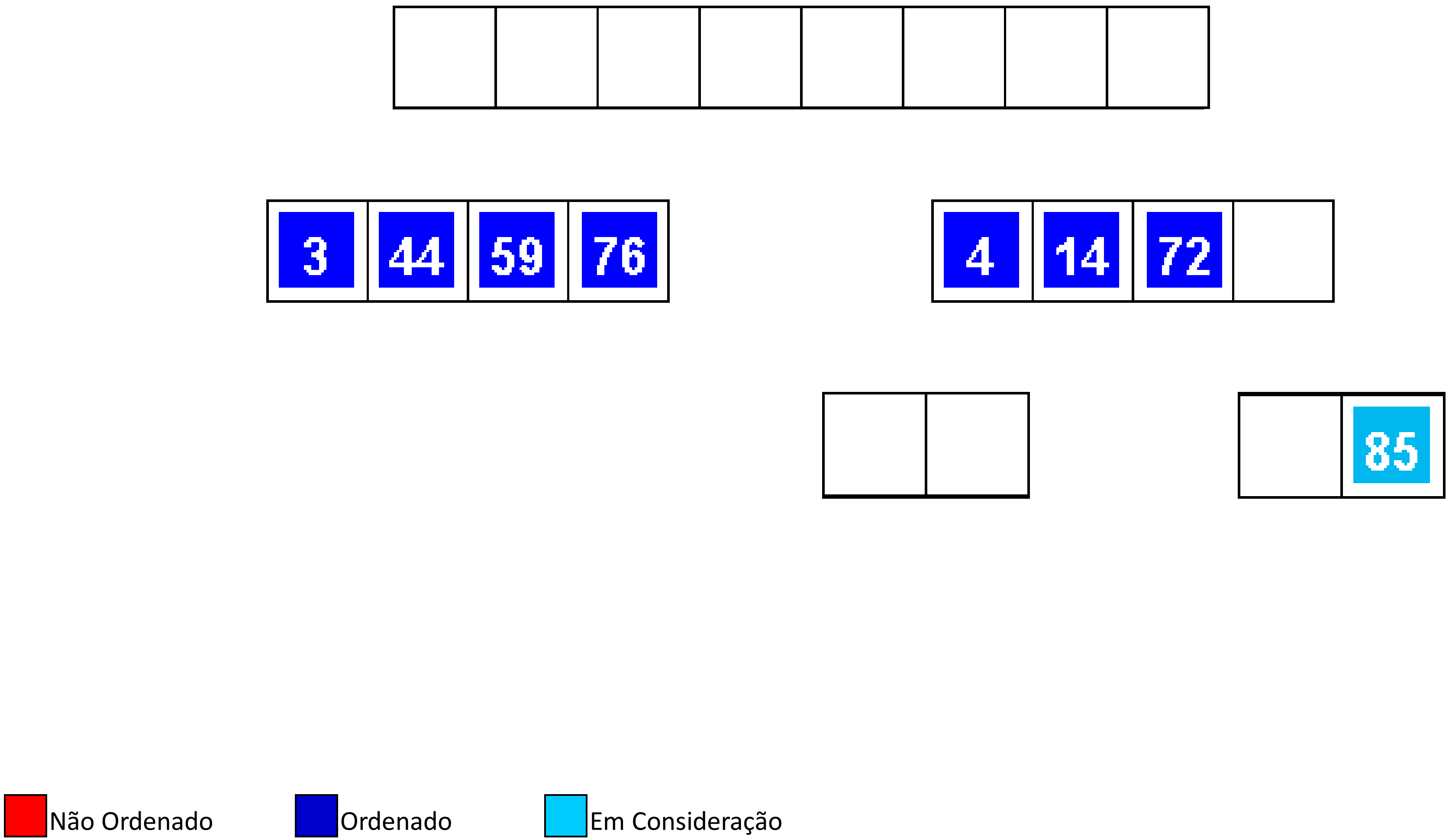

Não Ordenado

Ordenado

Em Consideração

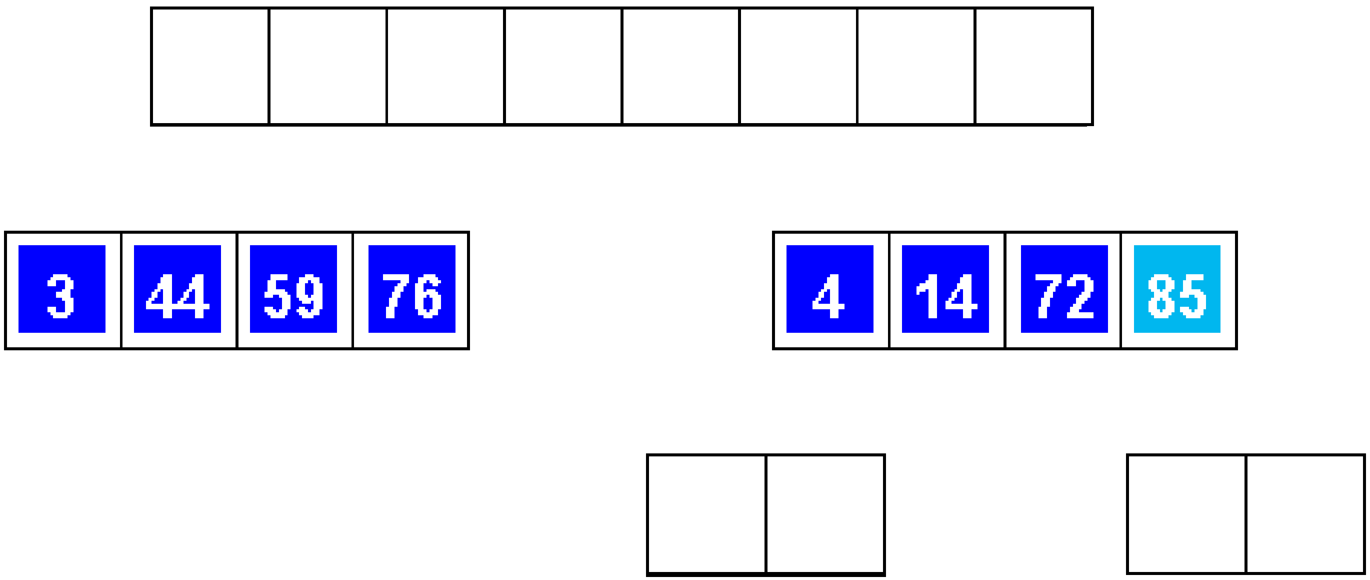
# Merge Sort

Idéia:



# Merge Sort

Idéia:



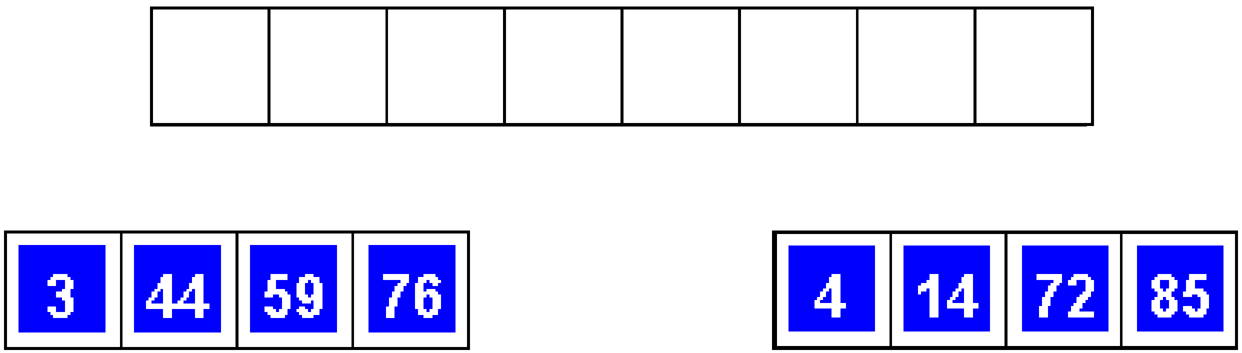

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

Idéia:



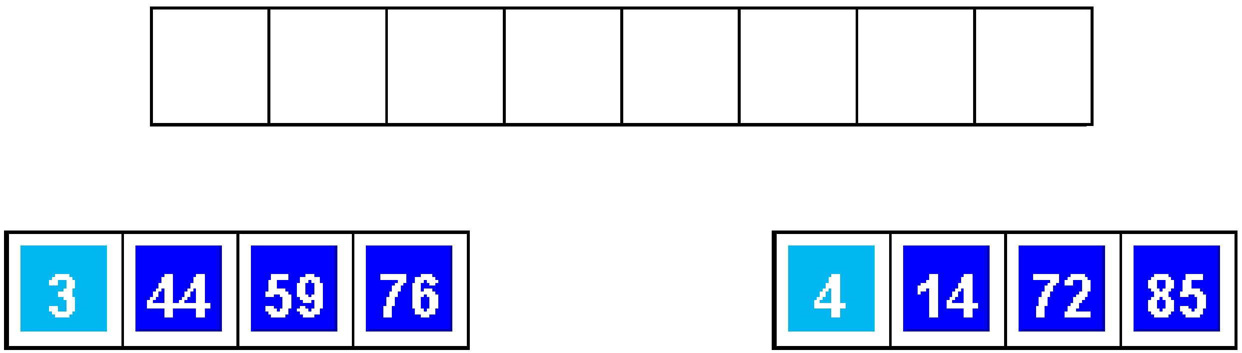

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

Idéia:



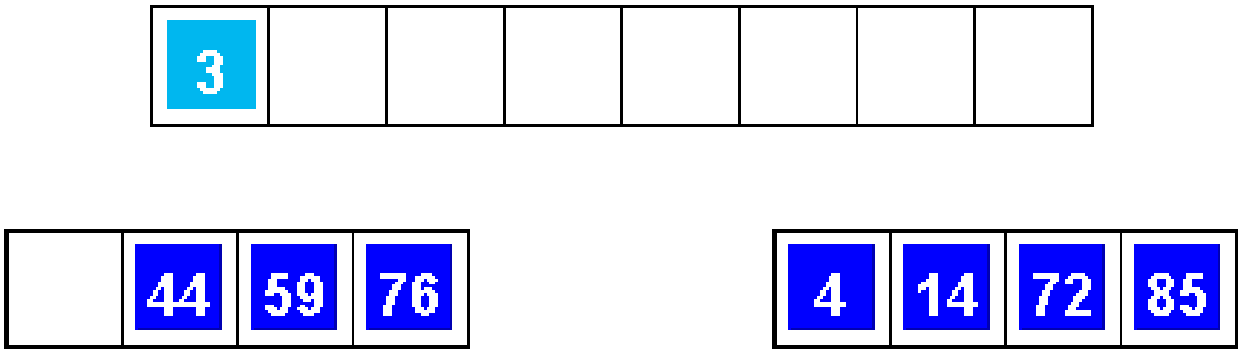

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

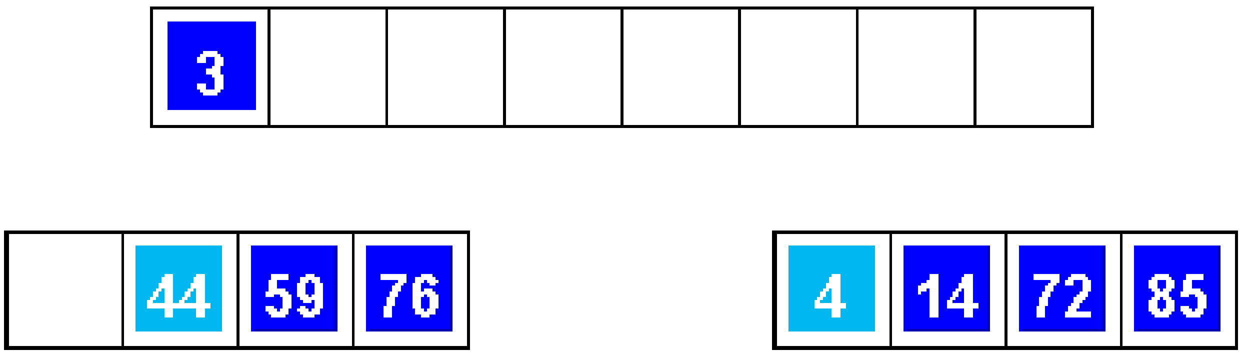
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



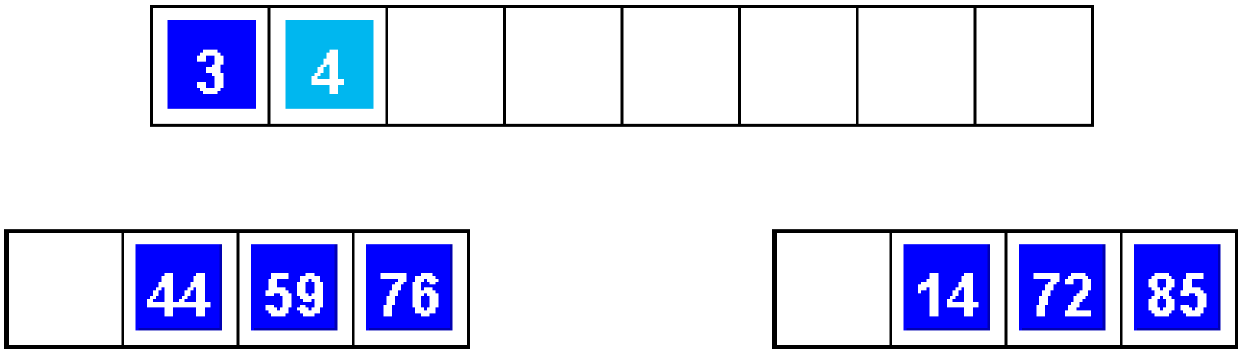

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

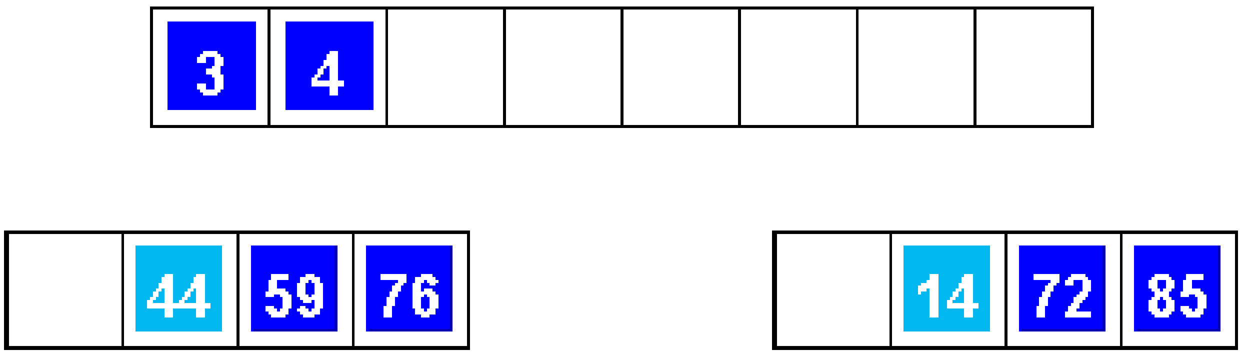
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

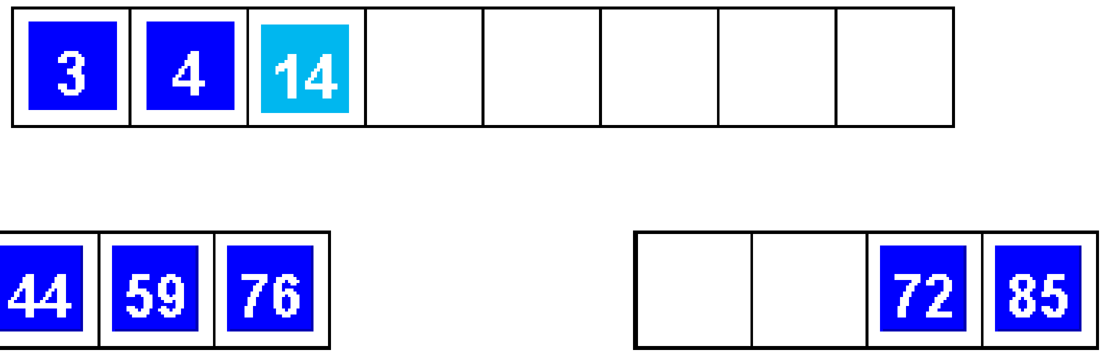
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



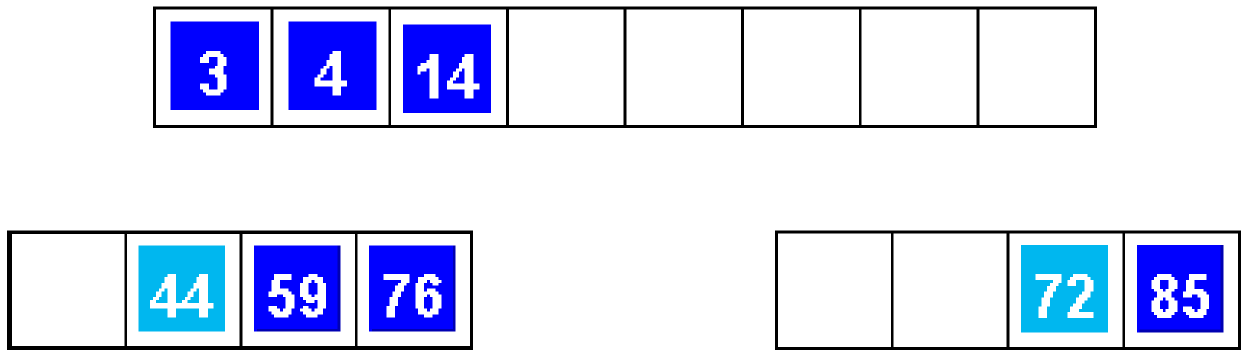

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

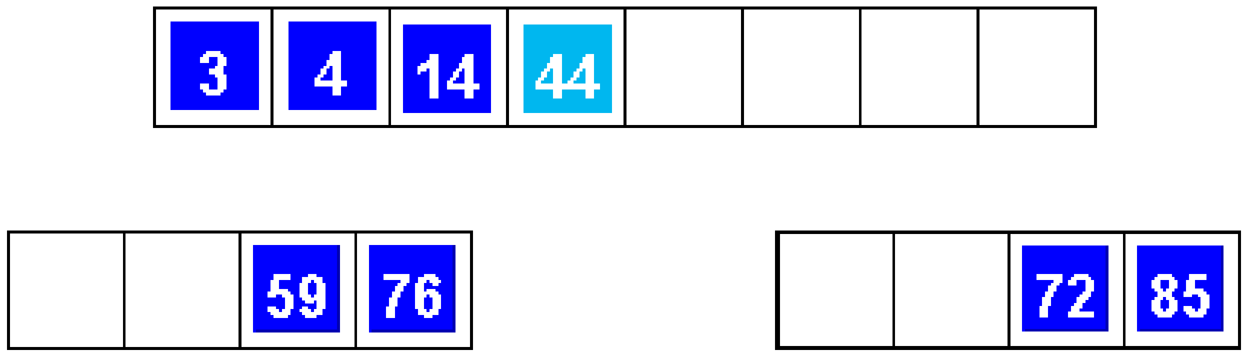
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



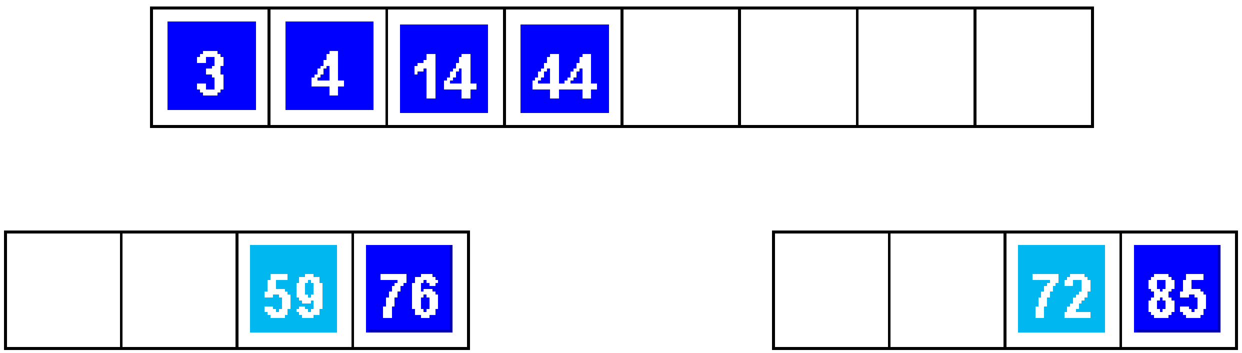

Não Ordenado

Ordenado

Em Consideração

# Merge Sort

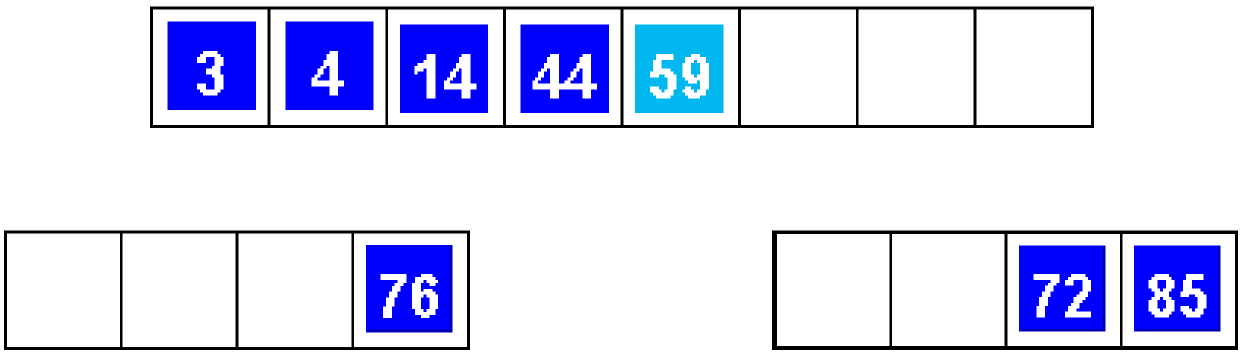
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

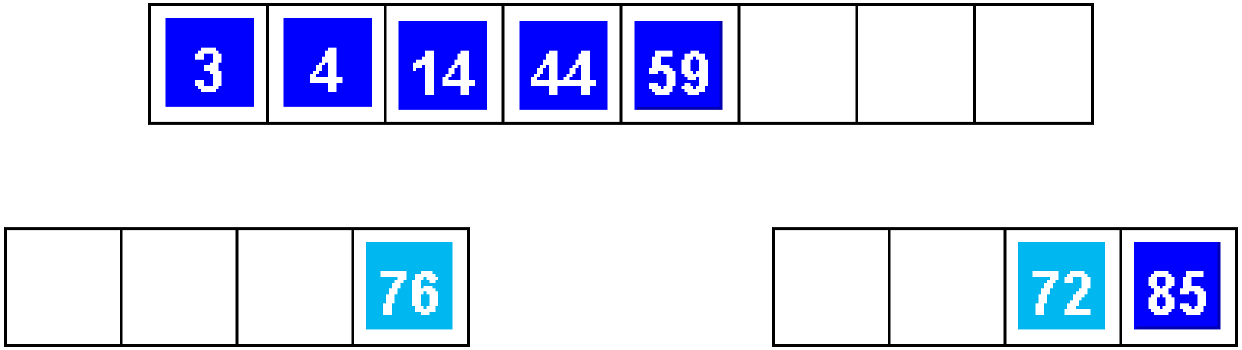
Idéia:




Não Ordenado    Ordenado    Em Consideração

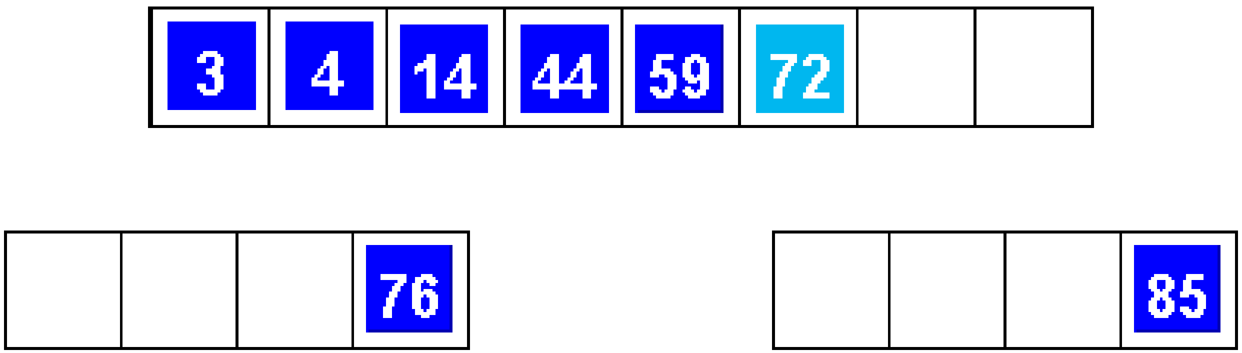
# Merge Sort

Idéia:




# Merge Sort

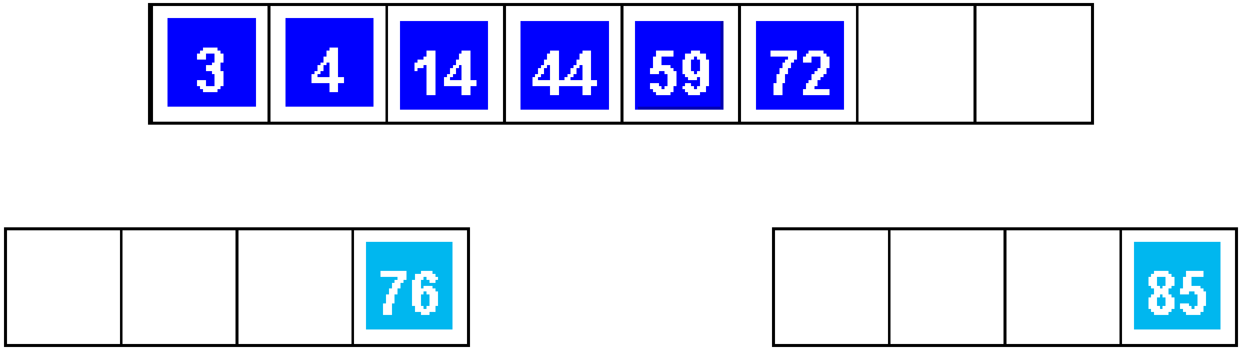
Idéia:




Não Ordenado Ordenado Em Consideração

# Merge Sort

Idéia:



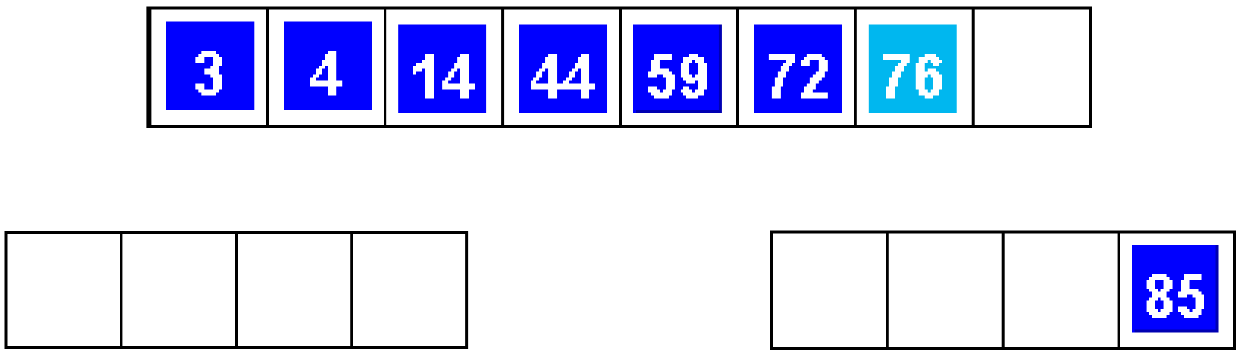

Não Ordenado

Ordenado

Em Consideração

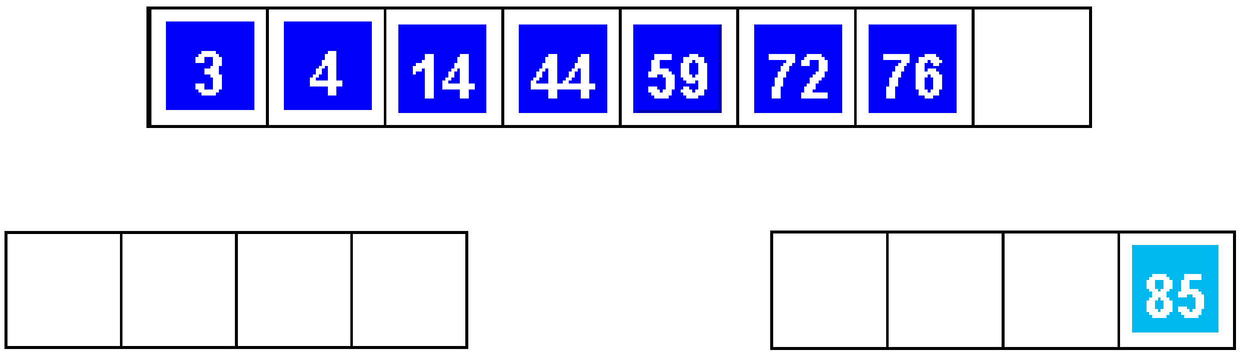
# Merge Sort

Idéia:




# Merge Sort

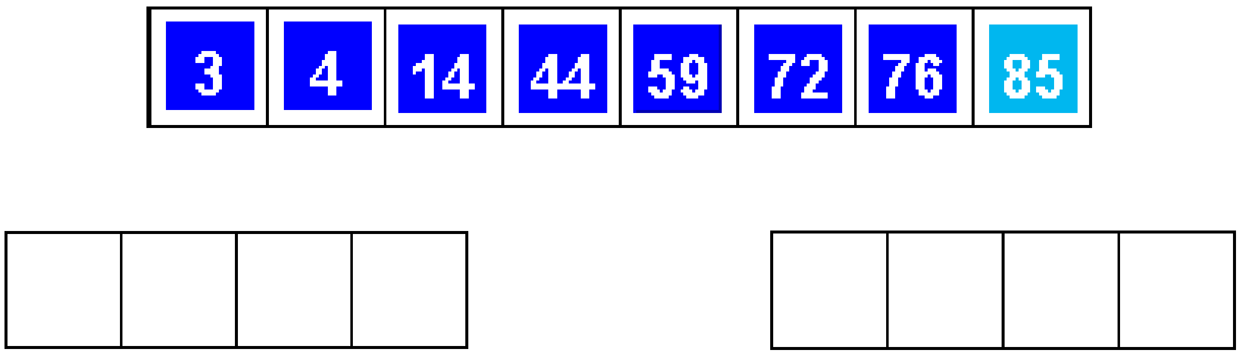
Idéia:




Não Ordenado
Ordenado
Em Consideração

# Merge Sort

Idéia:




Não Ordenado    Ordenado    Em Consideração

# Merge Sort

Idéia:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 14 | 44 | 59 | 72 | 76 | 85 |

Não Ordenado Ordenado Em Consideração

# Merge Sort

E
t
a
p
a

d
a

D
i
v
i
s
ã
o

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **20** | **43** | **23** | **54** | **87** | **12** | **03** | **45** | **01** | **98** | **05** | **52** | **69** | **07** | **21** | **04** |

# Merge Sort

**Etapa da Divisão**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 | 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 |

| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|
| 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

# Merge Sort

Etapa da Divisão

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 | 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 |

| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|
| 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 20 | 43 | 23 | 54 |

| 4 | 5 | 6 | 7 |
|---|---|---|---|
| 87 | 12 | 03 | 45 |

| 8 | 9 | 10 | 11 |
|---|---|---|---|
| 01 | 98 | 05 | 52 |

| 12 | 13 | 14 | 15 |
|---|---|---|---|
| 69 | 07 | 21 | 04 |

# Merge Sort

Etapa da Divisão

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **20** | **43** | **23** | **54** | **87** | **12** | **03** | **45** | **01** | **98** | **05** | **52** | **69** | **07** | **21** | **04** |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| **20** | **43** | **23** | **54** | **87** | **12** | **03** | **45** |

| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|
| **01** | **98** | **05** | **52** | **69** | **07** | **21** | **04** |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| **20** | **43** | **23** | **54** |

| 4 | 5 | 6 | 7 |
|---|---|---|---|
| **87** | **12** | **03** | **45** |

| 8 | 9 | 10 | 11 |
|---|---|---|---|
| **01** | **98** | **05** | **52** |

| 12 | 13 | 14 | 15 |
|---|---|---|---|
| **69** | **07** | **21** | **04** |

| 0 | 1 |
|---|---|
| **20** | **43** |

| 2 | 3 |
|---|---|
| **23** | **54** |

| 4 | 5 |
|---|---|
| **87** | **12** |

| 6 | 7 |
|---|---|
| **03** | **45** |

| 8 | 9 |
|---|---|
| **01** | **98** |

| 10 | 11 |
|---|---|
| **05** | **52** |

| 12 | 13 |
|---|---|
| **69** | **07** |

| 14 | 15 |
|---|---|
| **21** | **04** |

# Merge Sort

**Etapa da Divisão**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 | 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 |

| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|
| 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 20 | 43 | 23 | 54 |

| 4 | 5 | 6 | 7 |
|---|---|---|---|
| 87 | 12 | 03 | 45 |

| 8 | 9 | 10 | 11 |
|---|---|---|---|
| 01 | 98 | 05 | 52 |

| 12 | 13 | 14 | 15 |
|---|---|---|---|
| 69 | 07 | 21 | 04 |

| 0 | 1 |
|---|---|
| 20 | 43 |

| 2 | 3 |
|---|---|
| 23 | 54 |

| 4 | 5 |
|---|---|
| 87 | 12 |

| 6 | 7 |
|---|---|
| 03 | 45 |

| 8 | 9 |
|---|---|
| 01 | 98 |

| 10 | 11 |
|---|---|
| 05 | 52 |

| 12 | 13 |
|---|---|
| 69 | 07 |

| 14 | 15 |
|---|---|
| 21 | 04 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 | 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

Etapa da Conquista

# Merge Sort

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **20** | **43** | **23** | **54** | **87** | **12** | **03** | **45** | **01** | **98** | **05** | **52** | **69** | **07** | **21** | **04** |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| **20** | **43** | **23** | **54** | **87** | **12** | **03** | **45** |

| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|
| **01** | **98** | **05** | **52** | **69** | **07** | **21** | **04** |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| **20** | **43** | **23** | **54** |

| 4 | 5 | 6 | 7 |
|---|---|---|---|
| **87** | **12** | **03** | **45** |

| 8 | 9 | 10 | 11 |
|---|---|---|---|
| **01** | **98** | **05** | **52** |

| 12 | 13 | 14 | 15 |
|---|---|---|---|
| **69** | **07** | **21** | **04** |

| 0 | 1 |
|---|---|
| **20** | **43** |

| 2 | 3 |
|---|---|
| **23** | **54** |

| 4 | 5 |
|---|---|
| **87** | **12** |

| 6 | 7 |
|---|---|
| **03** | **45** |

| 8 | 9 |
|---|---|
| **01** | **98** |

| 10 | 11 |
|---|---|
| **05** | **52** |

| 12 | 13 |
|---|---|
| **69** | **07** |

| 14 | 15 |
|---|---|
| **21** | **04** |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **20** | **43** | **23** | **54** | **87** | **12** | **03** | **45** | **01** | **98** | **05** | **52** | **69** | **07** | **21** | **04** |

Etapa da Combinação

# Merge Sort

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 | 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 | | 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

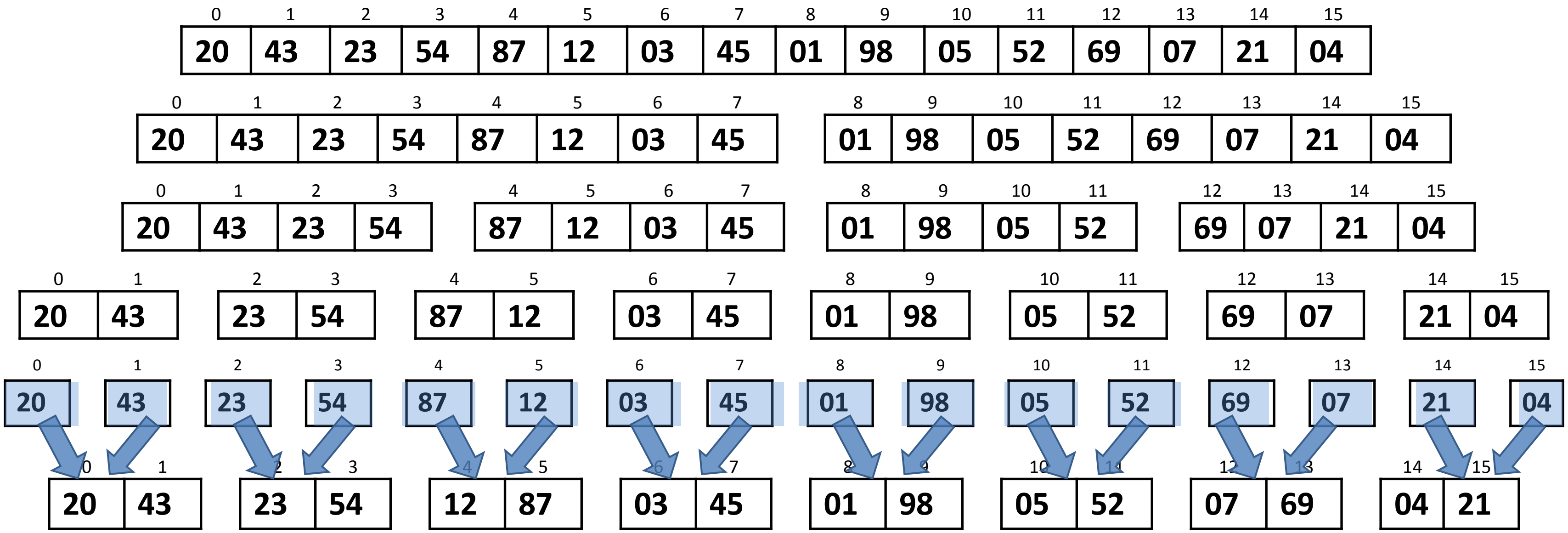
| 0 | 1 | 2 | 3 | | 4 | 5 | 6 | 7 | | 8 | 9 | 10 | 11 | | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | | 87 | 12 | 03 | 45 | | 01 | 98 | 05 | 52 | | 69 | 07 | 21 | 04 |

| 0 | 1 | | 2 | 3 | | 4 | 5 | | 6 | 7 | | 8 | 9 | | 10 | 11 | | 12 | 13 | | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | | 23 | 54 | | 87 | 12 | | 03 | 45 | | 01 | 98 | | 05 | 52 | | 69 | 07 | | 21 | 04 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 | 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

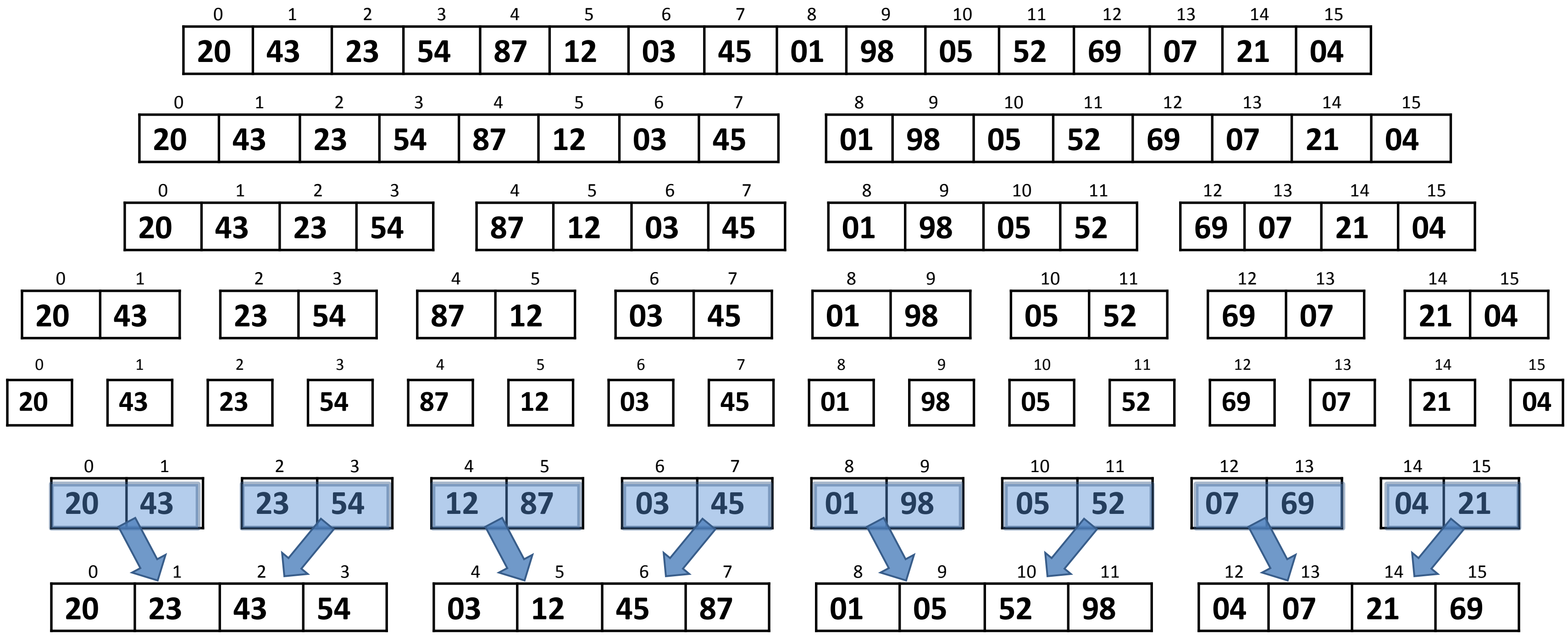
# Merge Sort

Etapa da Combinação



Etapa da Combinação

# Merge Sort



Etapa da Combinação

# Merge Sort

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 | 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 |

| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|
| 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 20 | 43 | 23 | 54 |

| 4 | 5 | 6 | 7 |
|---|---|---|---|
| 87 | 12 | 03 | 45 |

| 8 | 9 | 10 | 11 |
|---|---|---|---|
| 01 | 98 | 05 | 52 |

| 12 | 13 | 14 | 15 |
|---|---|---|---|
| 69 | 07 | 21 | 04 |

| 0 | 1 | | 2 | 3 | | 4 | 5 | | 6 | 7 | | 8 | 9 | | 10 | 11 | | 12 | 13 | | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | | 23 | 54 | | 87 | 12 | | 03 | 45 | | 01 | 98 | | 05 | 52 | | 69 | 07 | | 21 | 04 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 | 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 | | 2 | 3 | | 4 | 5 | | 6 | 7 | | 8 | 9 | | 10 | 11 | | 12 | 13 | | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | | 23 | 54 | | 12 | 87 | | 03 | 45 | | 01 | 98 | | 05 | 52 | | 07 | 69 | | 04 | 21 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 20 | 23 | 43 | 54 |

| 4 | 5 | 6 | 7 |
|---|---|---|---|
| 03 | 12 | 45 | 87 |

| 8 | 9 | 10 | 11 |
|---|---|---|---|
| 01 | 05 | 52 | 98 |

| 12 | 13 | 14 | 15 |
|---|---|---|---|
| 04 | 07 | 21 | 69 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 03 | 12 | 20 | 23 | 43 | 45 | 54 | 87 |

| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|
| 01 | 04 | 05 | 07 | 21 | 52 | 69 | 98 |

# Merge Sort

Etapa da Combinação

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 | 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 |

| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|
| 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 20 | 43 | 23 | 54 |

| 4 | 5 | 6 | 7 |
|---|---|---|---|
| 87 | 12 | 03 | 45 |

| 8 | 9 | 10 | 11 |
|---|---|---|---|
| 01 | 98 | 05 | 52 |

| 12 | 13 | 14 | 15 |
|---|---|---|---|
| 69 | 07 | 21 | 04 |

| 0 | 1 |
|---|---|
| 20 | 43 |

| 2 | 3 |
|---|---|
| 23 | 54 |

| 4 | 5 |
|---|---|
| 87 | 12 |

| 6 | 7 |
|---|---|
| 03 | 45 |

| 8 | 9 |
|---|---|
| 01 | 98 |

| 10 | 11 |
|---|---|
| 05 | 52 |

| 12 | 13 |
|---|---|
| 69 | 07 |

| 14 | 15 |
|---|---|
| 21 | 04 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 43 | 23 | 54 | 87 | 12 | 03 | 45 | 01 | 98 | 05 | 52 | 69 | 07 | 21 | 04 |

| 0 | 1 |
|---|---|
| 20 | 43 |

| 2 | 3 |
|---|---|
| 23 | 54 |

| 4 | 5 |
|---|---|
| 12 | 87 |

| 6 | 7 |
|---|---|
| 03 | 45 |

| 8 | 9 |
|---|---|
| 01 | 98 |

| 10 | 11 |
|---|---|
| 05 | 52 |

| 12 | 13 |
|---|---|
| 07 | 69 |

| 14 | 15 |
|---|---|
| 04 | 21 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 20 | 23 | 43 | 54 |

| 4 | 5 | 6 | 7 |
|---|---|---|---|
| 03 | 12 | 45 | 87 |

| 8 | 9 | 10 | 11 |
|---|---|---|---|
| 01 | 05 | 52 | 98 |

| 12 | 13 | 14 | 15 |
|---|---|---|---|
| 04 | 07 | 21 | 69 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 03 | 12 | 20 | 23 | 43 | 45 | 54 | 87 |

| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|
| 01 | 04 | 05 | 07 | 21 | 52 | 69 | 98 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 01 | 03 | 04 | 05 | 07 | 12 | 20 | 21 | 23 | 43 | 45 | 52 | 54 | 69 | 87 | 98 |

# Intercala

- O algoritmo de intercalação do Merge Sort usa como vetor subvetores do vetor original. Por este motivo, ele é ligeiramente diferente da implementação usando um vetor bipartido.
  - Pois agora o vetor será n-partido, então deve-se preocupar com o tamanho do subvetor.
  - Deve-se ter cuidado ao transferir o resultado do vetor temporário para o original, pois os índices do original devem ser mantidos.

# Merge Sort

```
void mergeSort(int *v, int e, int d)
{
    int meio;
    if(e < d)
    {
        meio = (d+e)/2;
        mergeSort(v, e, meio);
        mergeSort(v, meio+1, d);
        intercala(v, e, meio+1, d);
    }
}
```

# Análise de Algoritmo

```
void mergeSort(int *v, int e, int d)
{
    int meio;
    if(e < d)
    {
        meio = (d+e)/2;
        mergeSort(v, e, meio);
        mergeSort(v, meio+1, d);
        intercala(v, e, meio+1, d);
    }
}
```

$T(n/2)$

# Análise de Algoritmo

```
    void mergeSort(int *v, int e, int d)
    {
01.     int meio;
02.     if(e < d)
        {
03.         meio = (d+e)/2;
04.         mergeSort(v, e, meio);      → T(n/2)
05.         mergeSort(v, meio+1, d);    → T(n/2)
06.         intercala(v, e, meio+1, d);
        }
    }
```

# Análise de Algoritmo

```
void mergeSort(int *v, int e, int d)
{
    int meio;
    if(e < d)
    {
        meio = (d+e)/2;
        mergeSort(v, e, meio);            T(n/2)
        mergeSort(v, meio+1, d);          T(n/2)
        intercala(v, e, meio+1, d);       O(n)
    }
}
```

# Análise de Algoritmo

```
void mergeSort(int *v, int e, int d)
{
    int meio;
    if(e < d)
    {
        meio = (d+e)/2;
        mergeSort(v, e, meio);          T(n/2)
        mergeSort(v, meio+1, d);          T(n/2)
        intercala(v, e, meio+1, d);          O(n)
    }
}
```

$$T(n) = \begin{cases} 1 & \text{, para n = 1} \\ 2 \cdot T\left(\frac{n}{2}\right) + n & \text{, para n > 1} \end{cases}$$

# Análise de Algoritmo

$$T(n) = \begin{cases} 1 \\ 2 \cdot T\left(\dfrac{n}{2}\right) + n \end{cases}$$

Supondo que a entrada sempre será potências de 2. Ou seja, n = $2^k$

# Análise de Algoritmo

$$T(n) = \begin{cases} 1 \\ 2 \cdot T\left(\dfrac{n}{2}\right) + n \end{cases}$$

Supondo que a entrada sempre será potências de 2. Ou seja, n = $2^k$

$$T(2^k) = \begin{cases} 1 \\ 2 \cdot T\left(\dfrac{2^k}{2}\right) + 2^k \end{cases}$$

# Análise de Algoritmo

$$T(n) = \begin{cases} 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + n \end{cases}$$

Supondo que a entrada sempre será potências de 2. Ou seja, n = $2^k$

$$T(2^k) = \begin{cases} 1 \\ 2 \cdot T\left(\frac{2^k}{2}\right) + 2^k \end{cases}$$

Usando o método da expansão telescópica (método da iteração) para encontrar a fórmula fechada da recorrência

# Análise de Algoritmo

Usando o método da expansão telescópica (método da iteração) para encontrar a fórmula fechada da recorrência

$$T(2^k) = 2 \cdot T(2^{k-1}) + 2^k$$

# Análise de Algoritmo

Usando o método da expansão telescópica (método da iteração) para encontrar a fórmula fechada da recorrência

$$T(2^k) = 2 \cdot T(2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot (2^{k-1}) + 2^k$$

# Análise de Algoritmo

Usando o método da expansão telescópica (método da iteração) para encontrar a fórmula fechada da recorrência

$$T(2^k) = 2 \cdot T(2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot (2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot 2^k$$

# Análise de Algoritmo

Usando o método da expansão telescópica (método da iteração) para encontrar a fórmula fechada da recorrência

$$T(2^k) = 2 \cdot T(2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot (2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot 2^k$$

$$T(2^k) = 2^3 \cdot T(2^{k-3}) + 2^2 \cdot (2^{k-2}) + 2 \cdot 2^k$$

# Análise de Algoritmo

Usando o método da expansão telescópica (método da iteração) para encontrar a fórmula fechada da recorrência

$$T(2^k) = 2 \cdot T(2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot (2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot 2^k$$

$$T(2^k) = 2^3 \cdot T(2^{k-3}) + 2^2 \cdot (2^{k-2}) + 2 \cdot 2^k$$

$$T(2^k) = 2^3 \cdot T(2^{k-3}) + 3 \cdot 2^k$$

# Análise de Algoritmo

Usando o método da expansão telescópica (método da iteração) para encontrar a fórmula fechada da recorrência

$$T(2^k) = 2 \cdot T(2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot (2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot 2^k$$

$$T(2^k) = 2^3 \cdot T(2^{k-3}) + 2^2 \cdot (2^{k-2}) + 2 \cdot 2^k$$

$$T(2^k) = 2^3 \cdot T(2^{k-3}) + 3 \cdot 2^k$$

$$T(2^k) = 2^4 \cdot T(2^{k-4}) + 2^3 \cdot (2^{k-3}) + 3 \cdot 2^k$$

# Análise de Algoritmo

Usando o método da expansão telescópica (método da iteração) para encontrar a fórmula fechada da recorrência

$$T(2^k) = 2 \cdot T(2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot (2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot 2^k$$

$$T(2^k) = 2^3 \cdot T(2^{k-3}) + 2^2 \cdot (2^{k-2}) + 2 \cdot 2^k$$

$$T(2^k) = 2^3 \cdot T(2^{k-3}) + 3 \cdot 2^k$$

$$T(2^k) = 2^4 \cdot T(2^{k-4}) + 2^3 \cdot (2^{k-3}) + 3 \cdot 2^k$$

$$T(2^k) = 2^4 \cdot T(2^{k-4}) + 4 \cdot 2^k$$

# Análise de Algoritmo

Usando o método da expansão telescópica (método da iteração) para encontrar a fórmula fechada da recorrência

$$T(2^k) = 2 \cdot T(2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot (2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot 2^k$$

$$T(2^k) = 2^3 \cdot T(2^{k-3}) + 2^2 \cdot (2^{k-2}) + 2 \cdot 2^k$$

$$T(2^k) = 2^3 \cdot T(2^{k-3}) + 3 \cdot 2^k$$

$$T(2^k) = 2^4 \cdot T(2^{k-4}) + 2^3 \cdot (2^{k-3}) + 3 \cdot 2^k$$

$$T(2^k) = 2^4 \cdot T(2^{k-4}) + 4 \cdot 2^k$$

Continuando até a i-ésima iteração:

# Análise de Algoritmo

Usando o método da expansão telescópica (método da iteração) para encontrar a fórmula fechada da recorrência

$$T(2^k) = 2 \cdot T(2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot (2^{k-1}) + 2^k$$

$$T(2^k) = 2^2 \cdot T(2^{k-2}) + 2 \cdot 2^k$$

$$T(2^k) = 2^3 \cdot T(2^{k-3}) + 2^2 \cdot (2^{k-2}) + 2 \cdot 2^k$$

$$T(2^k) = 2^3 \cdot T(2^{k-3}) + 3 \cdot 2^k$$

$$T(2^k) = 2^4 \cdot T(2^{k-4}) + 2^3 \cdot (2^{k-3}) + 3 \cdot 2^k$$

$$T(2^k) = 2^4 \cdot T(2^{k-4}) + 4 \cdot 2^k$$

Continuando até a i-ésima iteração:

$$T(2^k) = 2^i \cdot T(2^{k-i}) + i \cdot 2^k$$

# Análise de Algoritmo

Continuando até a i-ésima iteração:

$$T(2^k) = 2^i \cdot T(2^{k-i}) + i \cdot 2^k$$

Eliminando a recorrência invocando o caso base, ou seja $2^{k-i}$=1 ou i=k:

# Análise de Algoritmo

Continuando até a i-ésima iteração:

$$T(2^k) = 2^i \cdot T(2^{k-i}) + i \cdot 2^k$$

Eliminando a recorrência invocando o caso base, ou seja $2^{k-i}=1$ ou i=k:

$$T(2^k) = 2^k \cdot T(2^{k-k}) + k \cdot 2^k$$

# Análise de Algoritmo

Continuando até a i-ésima iteração:

$$T(2^k) = 2^i \cdot T(2^{k-i}) + i \cdot 2^k$$

Eliminando a recorrência invocando o caso base, ou seja $2^{k-i}=1$ ou i=k:

$$T(2^k) = 2^k \cdot T(2^{k-k}) + k \cdot 2^k$$

$$T(2^k) = 2^k \cdot T(2^0) + k \cdot 2^k$$

# Análise de Algoritmo

Continuando até a i-ésima iteração:

$$T(2^k) = 2^i \cdot T(2^{k-i}) + i \cdot 2^k$$

Eliminando a recorrência invocando o caso base, ou seja $2^{k-i}=1$ ou $i=k$:

$$T(2^k) = 2^k \cdot T(2^{k-k}) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(2^0) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(1) + k \cdot 2^k$$

# Análise de Algoritmo

Continuando até a i-ésima iteração:

$$T(2^k) = 2^i \cdot T(2^{k-i}) + i \cdot 2^k$$

Eliminando a recorrência invocando o caso base, ou seja $2^{k-i}=1$ ou $i=k$:

$$T(2^k) = 2^k \cdot T(2^{k-k}) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(2^0) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(1) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot 1 + k \cdot 2^k$$

# Análise de Algoritmo

Continuando até a i-ésima iteração:

$$T(2^k) = 2^i \cdot T(2^{k-i}) + i \cdot 2^k$$

Eliminando a recorrência invocando o caso base, ou seja $2^{k-i}=1$ ou $i=k$:

$$T(2^k) = 2^k \cdot T(2^{k-k}) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(2^0) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(1) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot 1 + k \cdot 2^k$$

Retornando a suposição inicial: $n=2^k$.

# Análise de Algoritmo

Continuando até a i-ésima iteração:

$$T(2^k) = 2^i \cdot T(2^{k-i}) + i \cdot 2^k$$

Eliminando a recorrência invocando o caso base, ou seja $2^{k-i}$=1 ou i=k:

$$T(2^k) = 2^k \cdot T(2^{k-k}) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(2^0) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(1) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot 1 + k \cdot 2^k$$

Retornando a suposição inicial: n=$2^k$.

$$T(n) = n + k \cdot n$$

# Análise de Algoritmo

Continuando até a i-ésima iteração:

$$T(2^k) = 2^i \cdot T(2^{k-i}) + i \cdot 2^k$$

Eliminando a recorrência invocando o caso base, ou seja $2^{k-i}=1$ ou i=k:

$$T(2^k) = 2^k \cdot T(2^{k-k}) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(2^0) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(1) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot 1 + k \cdot 2^k$$

Retornando a suposição inicial: $n=2^k$.

$$T(n) = n + k \cdot n$$

$$k = \log_2 n$$

# Análise de Algoritmo

Continuando até a i-ésima iteração:

$$T(2^k) = 2^i \cdot T(2^{k-i}) + i \cdot 2^k$$

Eliminando a recorrência invocando o caso base, ou seja $2^{k-i}=1$ ou $i=k$:

$$T(2^k) = 2^k \cdot T(2^{k-k}) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(2^0) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot T(1) + k \cdot 2^k$$
$$T(2^k) = 2^k \cdot 1 + k \cdot 2^k$$

Retornando a suposição inicial: $n=2^k$.

$$T(n) = n + k \cdot n$$

$$k = \log_2 n$$

$$T(n) = n + \log_2 n \cdot n$$

# Análise de Algoritmo

Verificando a fórmula fechada através do método da substituição:

$$T(n) = \begin{cases} 1 & \text{, para n = 1} \\ 2 \cdot T\left(\dfrac{n}{2}\right) + n & \text{, para n > 1} \end{cases}$$

$$T(n) = n \cdot \log_2 n + n$$

Testando para a base:

# Análise de Algoritmo

Verificando a fórmula fechada através do método da substituição:

$$T(n) = \begin{cases} 1 & \text{, para n = 1} \\ 2 \cdot T\left(\dfrac{n}{2}\right) + n & \text{, para n > 1} \end{cases}$$

$$T(n) = n \cdot \log_2 n + n$$

Testando para a base:

$$T(n) = n \cdot \log_2 n + n$$

$$T(1) = 1 \cdot \log_2 1 + 1$$

# Análise de Algoritmo

Verificando a fórmula fechada através do método da substituição:

$$T(n) = \begin{cases} 1 & \text{, para n = 1} \\ 2 \cdot T\left(\frac{n}{2}\right) + n & \text{, para n > 1} \end{cases}$$

$$T(n) = n \cdot \log_2 n + n$$

Testando para a base:

$$T(n) = n \cdot \log_2 n + n$$

$$T(1) = 1 \cdot \log_2 1 + 1$$

$$T(1) = 1$$

# Análise de Algoritmo

Verificando a fórmula fechada através do método da substituição:

$$T(n) = \begin{cases} 1 & \text{, para n = 1} \\ 2 \cdot T\left(\frac{n}{2}\right) + n & \text{, para n > 1} \end{cases}$$

$$T(n) = n \cdot \log_2 n + n$$

Testando para o caso recursivo:

# Análise de Algoritmo

Verificando a fórmula fechada através do método da substituição:

$$T(n) = \begin{cases} 1 & \text{, para n = 1} \\ 2 \cdot T\left(\frac{n}{2}\right) + n & \text{, para n > 1} \end{cases}$$

$$T(n) = n \cdot \log_2 n + n$$

Testando para o caso recursivo:

$$T(n) = n \cdot \log_2 n + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot \log_2 \frac{n}{2} + \frac{n}{2} \right) + n$$

# Análise de Algoritmo

Verificando a fórmula fechada através do método da substituição:

$$T(n) = \begin{cases} 1 & \text{, para n = 1} \\ 2 \cdot T\left(\dfrac{n}{2}\right) + n & \text{, para n > 1} \end{cases}$$

$$T(n) = n \cdot \log_2 n + n$$

Testando para o caso recursivo:

$$T(n) = n \cdot \log_2 n + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot \log_2 \frac{n}{2} + \frac{n}{2} \right) + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot (\log_2 n - \log_2 2) + \frac{n}{2} \right) + n$$

# Análise de Algoritmo

Verificando a fórmula fechada através do método da substituição:

$$T(n) = \begin{cases} 1 & \text{, para n = 1} \\ 2 \cdot T\left(\dfrac{n}{2}\right) + n & \text{, para n > 1} \end{cases}$$

$$T(n) = n \cdot \log_2 n + n$$

Testando para o caso recursivo:

$$T(n) = n \cdot \log_2 n + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot \log_2 \frac{n}{2} + \frac{n}{2} \right) + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot (\log_2 n - \log_2 2) + \frac{n}{2} \right) + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot (\log_2 n - 1) + \frac{n}{2} \right) + n$$

# Análise de Algoritmo

Verificando a fórmula fechada através do método da substituição:

$$T(n) = \begin{cases} 1 & \text{, para n = 1} \\ 2 \cdot T\left(\dfrac{n}{2}\right) + n & \text{, para n > 1} \end{cases}$$

$$T(n) = n \cdot \log_2 n + n$$

Testando para o caso recursivo:

$$T(n) = n \cdot \log_2 n + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot \log_2 \frac{n}{2} + \frac{n}{2} \right) + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot (\log_2 n - \log_2 2) + \frac{n}{2} \right) + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot (\log_2 n - 1) + \frac{n}{2} \right) + n$$

$$T(n) = n \cdot (\log_2 n - 1) + n + n$$

# Análise de Algoritmo

Verificando a fórmula fechada através do método da substituição:

$$T(n) = \begin{cases} 1 & \text{, para n = 1} \\ 2 \cdot T\left(\frac{n}{2}\right) + n & \text{, para n > 1} \end{cases}$$

$$T(n) = n \cdot \log_2 n + n$$

Testando para o caso recursivo:

$$T(n) = n \cdot \log_2 n + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot \log_2 \frac{n}{2} + \frac{n}{2} \right) + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot (\log_2 n - \log_2 2) + \frac{n}{2} \right) + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot (\log_2 n - 1) + \frac{n}{2} \right) + n$$

$$T(n) = n \cdot (\log_2 n - 1) + n + n$$

$$T(n) = n \cdot \log_2 n - n + n + n$$

# Análise de Algoritmo

Verificando a fórmula fechada através do método da substituição:

$$T(n) = \begin{cases} 1 & \text{, para n = 1} \\ 2 \cdot T\left(\frac{n}{2}\right) + n & \text{, para n > 1} \end{cases}$$

$$T(n) = n \cdot \log_2 n + n$$

Testando para o caso recursivo:

$$T(n) = n \cdot \log_2 n + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot \log_2 \frac{n}{2} + \frac{n}{2} \right) + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot (\log_2 n - \log_2 2) + \frac{n}{2} \right) + n$$

$$T(n) = 2 \cdot \left( \frac{n}{2} \cdot (\log_2 n - 1) + \frac{n}{2} \right) + n$$

$$T(n) = n \cdot (\log_2 n - 1) + n + n$$

$$T(n) = n \cdot \log_2 n - n + n + n$$

$$T(n) = n \cdot \log_2 n + n$$

# Análise de Algoritmo

Portanto, o Merge Sort ordena um arranjo com n elementos, consumindo tempo proporcional a:

$$n \cdot \log_2 n + n$$

Deste modo, sua complexidade é:

$$O(n \cdot \log_2 n)$$

E de acordo com a análise de Lower Bound do Problema da Ordenação, o Merge Sort é um Algoritmo Assintóticamente Ótimo.