

# Ordenação de dados

Métodos de ordenação:

**Quicksort, Shellsort, Radixsort,  
Mergesort, Heapsort**

# Quicksort

- Algoritmo de ordenação mais rápido de propósito geral, aplica o princípio do "dividir para conquistar"

## Algoritmo

Seleciona-se um pivô (elemento escolhido aleatoriamente, ou o elemento mediano do vetor), com o qual serão comparados os demais elementos.

Os registros são então re-arranjados em duas “partições”:

- Os  $k$  valores menores que o pivô são colocados à esquerda dele.
- Os  $n-k$  valores maiores que o pivô são colocados à direita dele.
- O pivô é colocado na posição  $k$ .

Início

72	6	57	88	85	42	83	73	48	60
----	---	----	----	----	----	----	----	----	----

e

pivô

d

Passo 1

<u>72</u>	6	57	88	85	42	83	73	<u>48</u>	60
-----------	---	----	----	----	----	----	----	-----------	----

e

d

48	6	57	88	85	42	83	73	72	60
----	---	----	----	----	----	----	----	----	----

e

Passo 2

48	6	57	<u>88</u>	85	<u>42</u>	83	73	72	60
----	---	----	-----------	----	-----------	----	----	----	----

e

d

48	6	57	42	85	88	83	73	72	60
----	---	----	----	----	----	----	----	----	----

e

d

Passo 3

48	6	57	<u>42</u>	<u>85</u>	88	83	73	72	60
----	---	----	-----------	-----------	----	----	----	----	----

d

e

48	6	57	85	42	88	83	73	72	60
----	---	----	----	----	----	----	----	----	----

d

e

48	6	57	42		85	88	83	73	72	60
----	---	----	----	--	----	----	----	----	----	----

d

e

72	6	57	88	60	42	83	73	48	85
----	---	----	----	----	----	----	----	----	----

pivô=60

48	6	57	42	60	88	83	73	72	85
----	---	----	----	----	----	----	----	----	----

pivô=6

pivô=73

6	42	57	48
---	----	----	----

pivô=57

72	73	85	88	83
----	----	----	----	----

pivô=88

42	48	57
----	----	----

pivô=42

85	83	88
----	----	----

pivô=85

42	48
----	----

83	85
----	----

Final

6	42	48	57	60	72	73	83	85	88
---	----	----	----	----	----	----	----	----	----

# Quicksort

- complexidade:


  - caso médio:  $O(n \log n)$

  - pior caso:  $O(n^2)$

# Shellsort (incrementos decrescentes)


- 1º passo: classifica 8 sublistas de tamanho 2 e incremento 8:

**59** 20 **17** 13 **28** 14 **23** 83 **36** 98 **11** 70 **65** 41 **42** 15



- 2º passo: classifica 4 sublistas de tamanho 4 e incremento 4:


**36** 20 **11** 13 **28** 14 **23** 15 **59** 98 **17** 70 **65** 41 **42** 83



# Shellsort

- - 3° passo: classifica 2 sublistas de tamanho 8 e incremento 2:

28 14 11 13 36 20 17 15 59 41 23 70 65 98 42 83



- 4° passo: classifica 1 sublista de tamanho 16 e incremento 1 (usa inserção direta):

11 13 17 14 23 15 28 20 36 41 42 70 59 83 65 98

- final:

11 13 14 15 17 20 23 28 36 41 42 59 65 70 83 98



# Radixsort

- algoritmo de ordenação que **ordena inteiros** processando **dígitos individuais**
- como os inteiros podem representar strings compostas de caracteres (como nomes ou datas), radix sort não é limitado somente a inteiros
- Complexidade:  **$O(n * k)$**  , sendo k a quantidade de passos

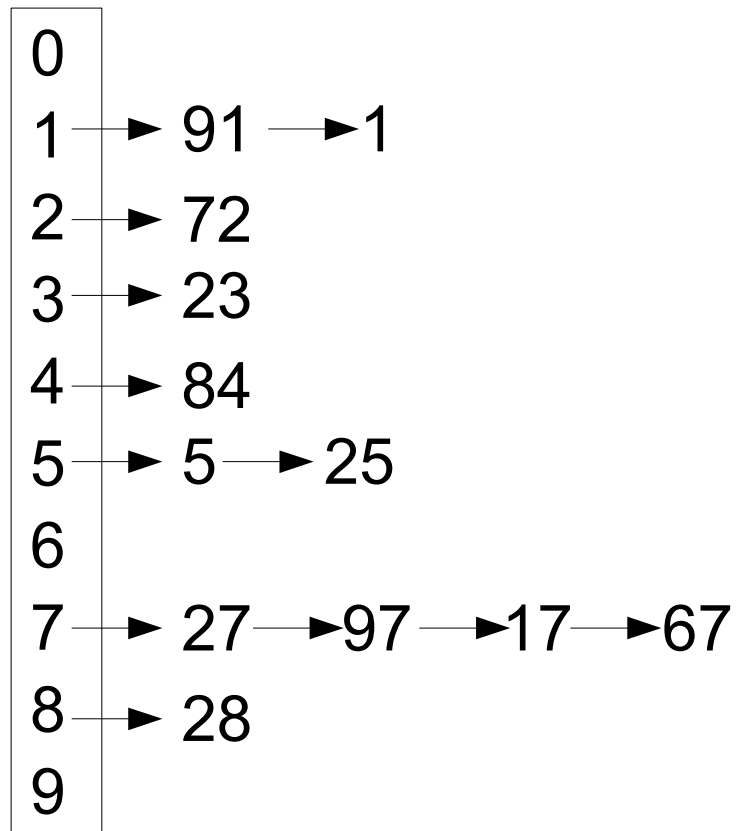
**Exemplo: ordenar a sequência**

27 91 1 97 17 23 84 28 72 5 67 25

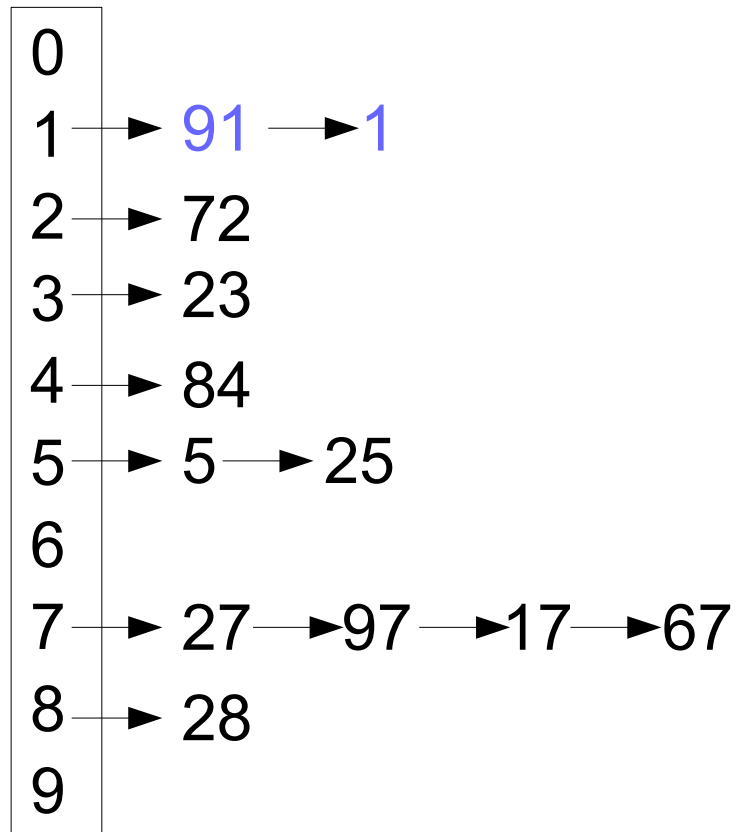
# Radixsort

27 91 1 97 17 23 84 28 72 5 67 25

1º passo: desenfileirar, e inserir pela unidade:



# Radixsort



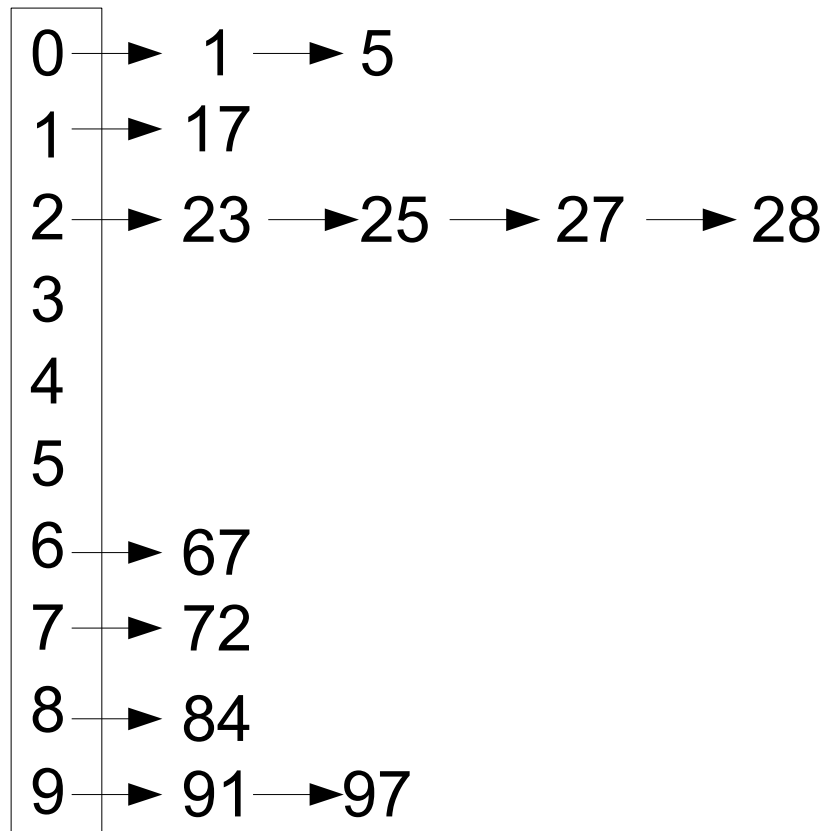
Enfileirar novamente, na ordem dos números:

91 1 72 23 84 5 25 27 97 17 67 28

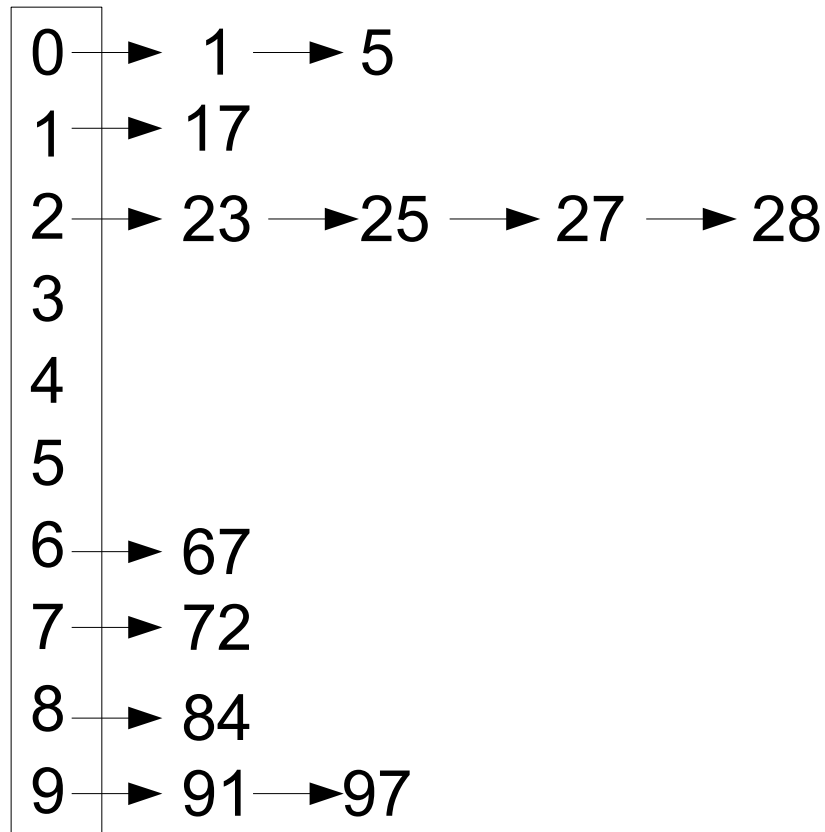
# Radixsort

91 1 72 23 84 5 25 27 97 17 67 28

2º passo: desenfileirar, e inserir pela dezena:



# Radixsort



Enfileirar novamente, na ordem dos números:

1 5 17 23 25 27 28 67 72 84 91 97

# Mergesort

- baseado no princípio de **dividir os dados em subconjuntos** e obter um vetor ordenado resultante pela ***intercalação ordenada*** dos valores dos dois subconjuntos
- Complexidade:  $O(n \log n)$

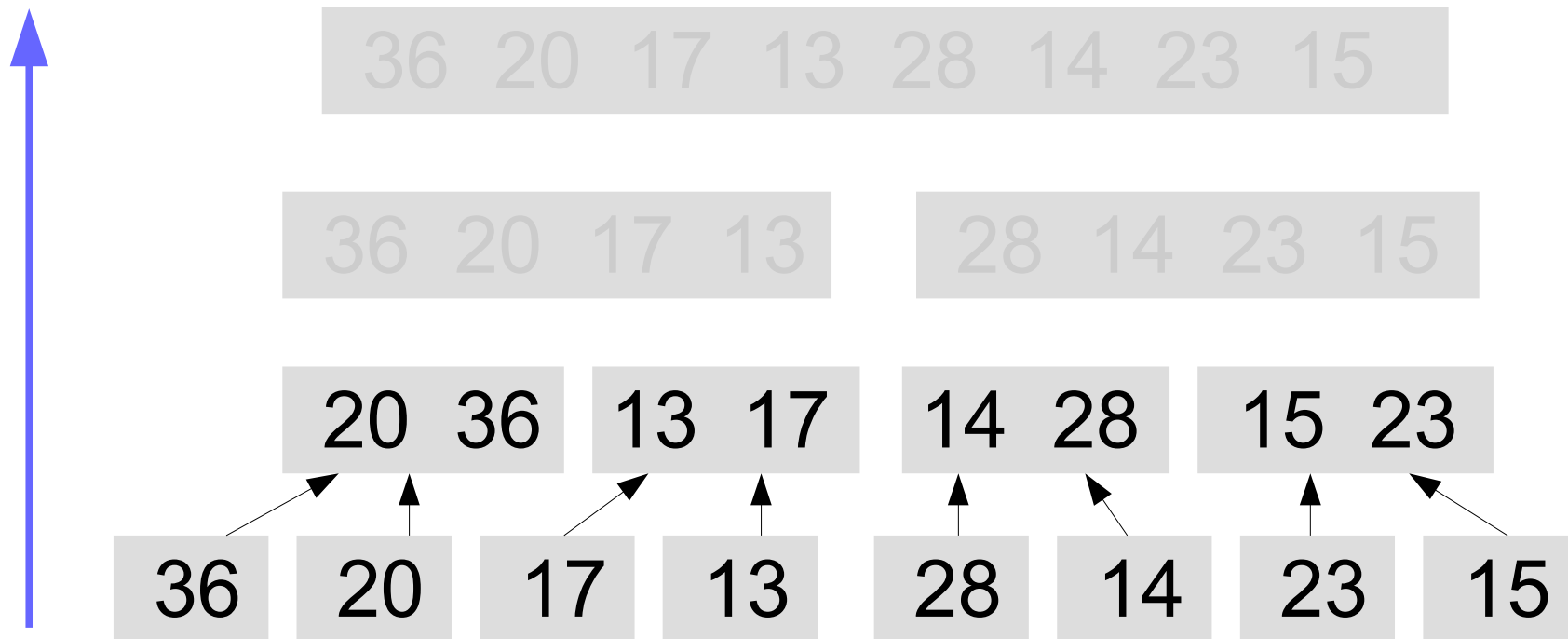
36 20 17 13 28 14 23 15

# Mergesort



Dividir recursivamente cada lista em 2 sublistas

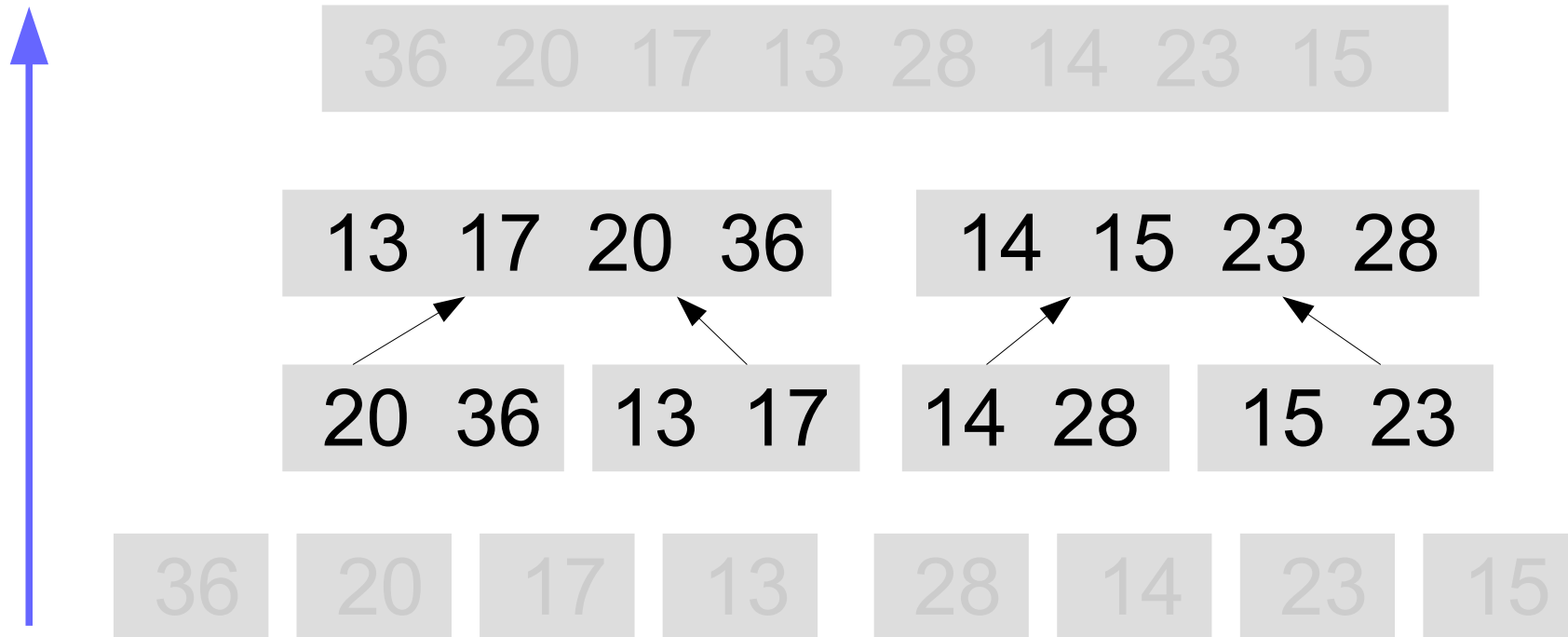
# Mergesort



Recombinar cada 2 sublistas, ordenando-as

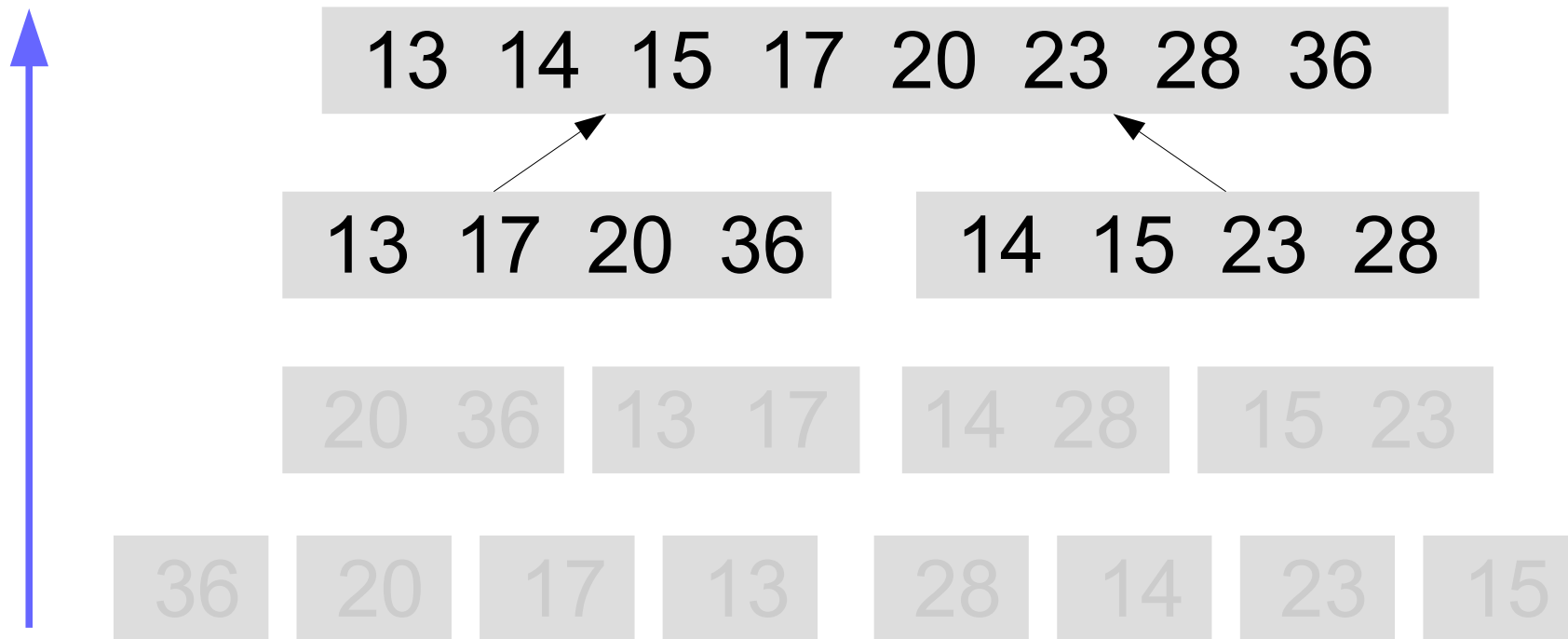


# Mergesort



Recombinar cada 2 sublistas, ordenando-as

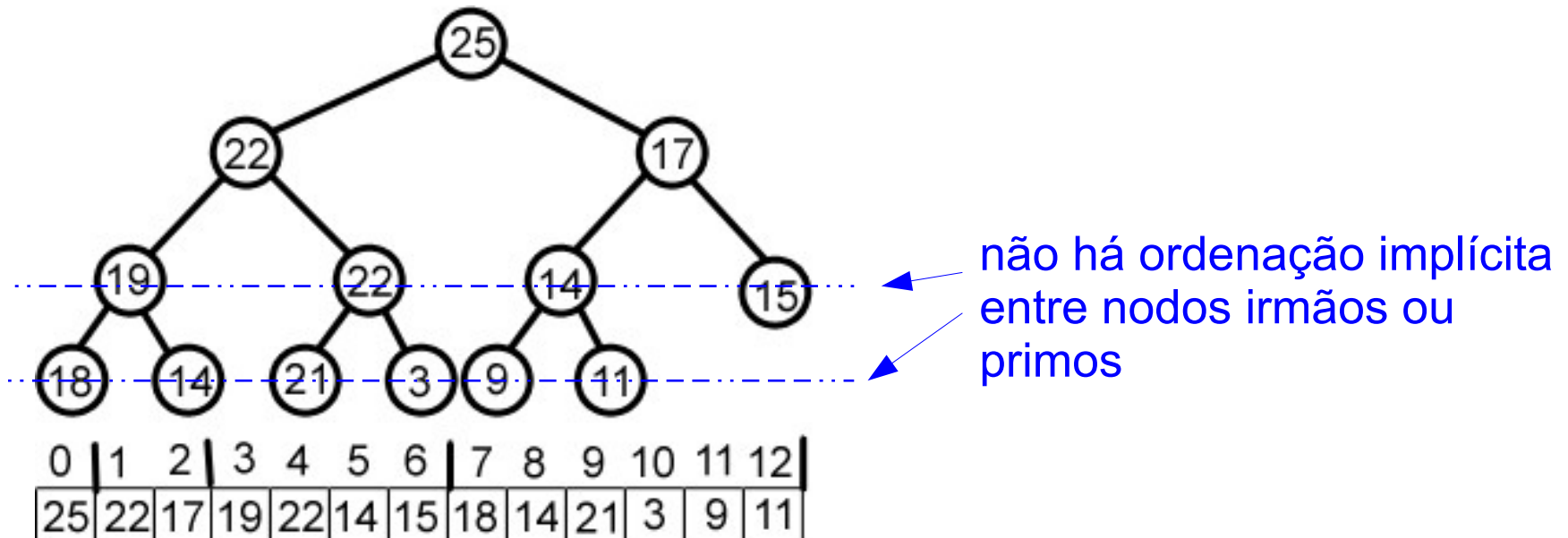
# Mergesort



Recombinar cada 2 sublistas, ordenando-as

# Heapsort

- **Heap** é uma estrutura de dados, baseada em árvore binária, implementada em um vetor
- satisfaz a seguinte propriedade : se P é um nó pai de C, então a chave (o valor) de P é maior que ou igual a (em uma *heap máxima*) ou menor que ou igual a (em uma *heap mínima*) chave de C



# Heapsort

**Example:-** The fig. shows steps of heap-sort for list (2 3 7 1 8 5 6)

