INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CURSO TÉCNICO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: ESTRUTURA DE DADOS

TIMSORT

Componentes:

Ian Pedro

Mardone Silva Pereira

Rodrigo Cardoso de Farias

INTRODUÇÃO

O Timsort desenvolvido por Tim Peters em 2002 para ser usado na linguagem Python. O Timsort é eficiente para ordenar dados reais, pois aproveita-se de padrões já existentes nos dados (como sequências ordenadas) para melhorar o desempenho.

MERGE SORT

INSERTION SORT

INSERTION SORT

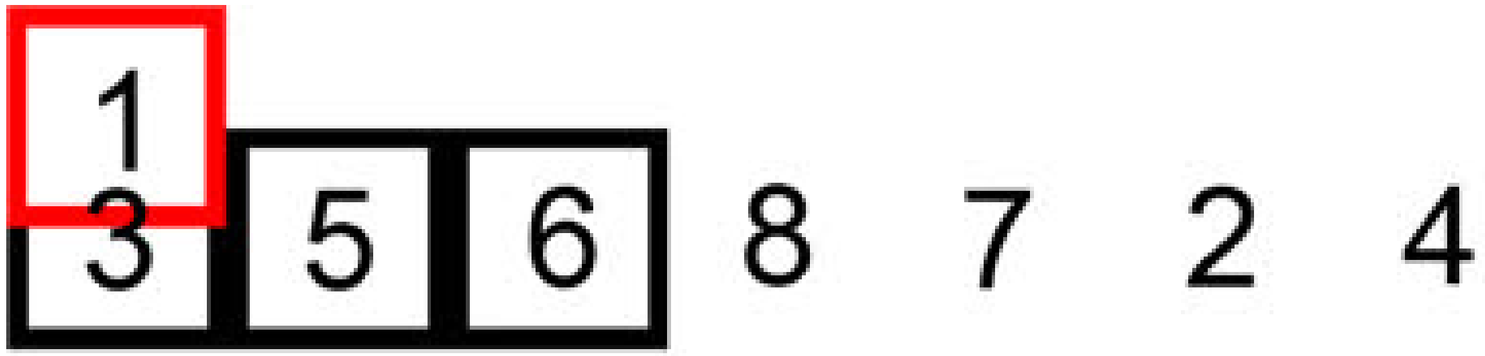
O Insertion Sort é um algoritmo de ordenação que constrói a sequência ordenada elemento por elemento , inserindo cada novo elemento na posição correta dentro da parte já ordenada do array.

Suponha que o primeiro elemento já esteja ordenado

Para cada elemento seguinte, ele é comparado com os anteriores e inserido na posição correta.

Os elementos à direita do inserido são deslocados para abrir espaço.

EXEMPLO PRÁTICO - INSERTION SORT



MERGE SORT

O Merge Sort é um algoritmo de ordenação eficiente que segue o paradigma dividir para conquistar.

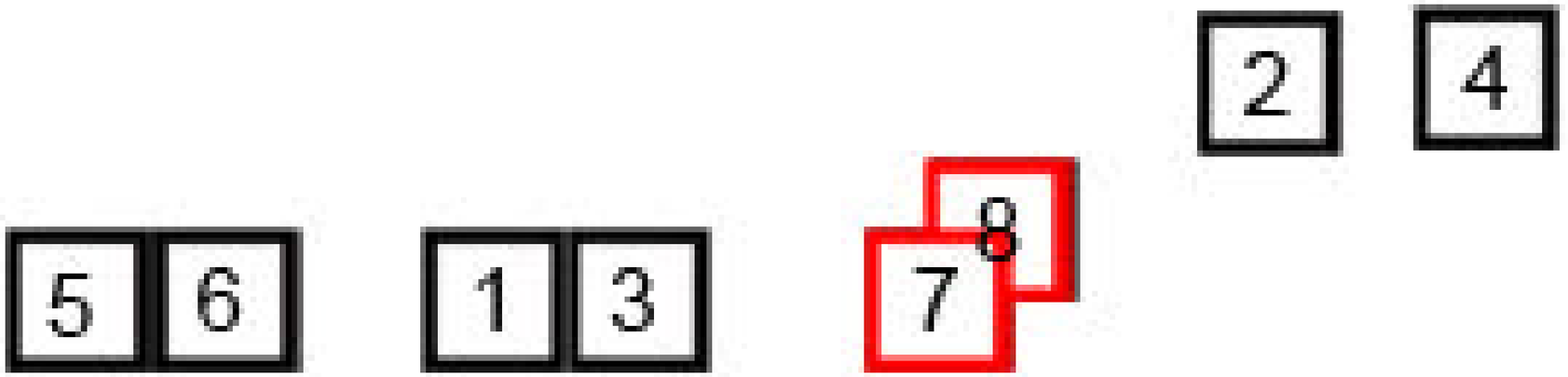
- *Estável*
- *Uso de memória elevado*

A lista é dividida ao meio repetidamente até que cada sublista contenha apenas um elemento (o que significa que já está ordenada).

Cada par de sublistas ordenadas é combinado (mesclado) em uma única lista ordenada.

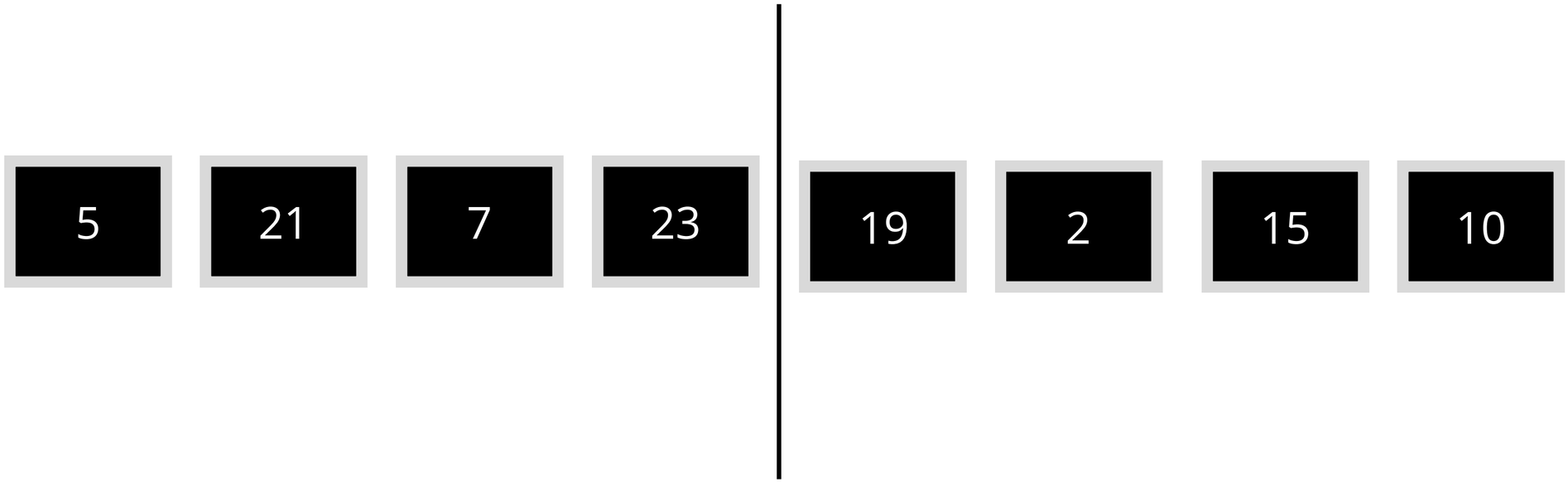
A fusão das sublistas ocorre de maneira ordenada, garantindo que a lista final também esteja ordenada.

EXEMPLO PRÁTICO - MERGE SORT



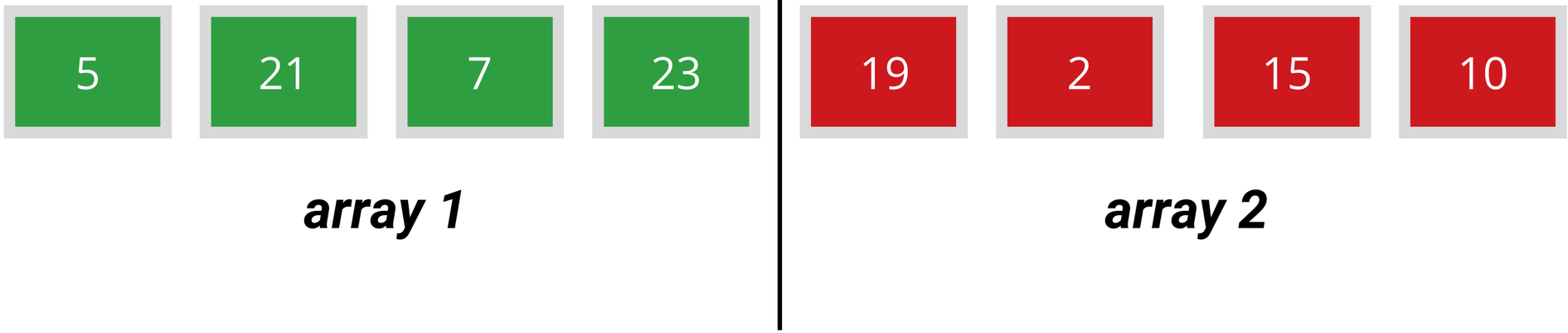
TIMSORT - FUNCIONAMENTO

Dividir o array em arrays menores



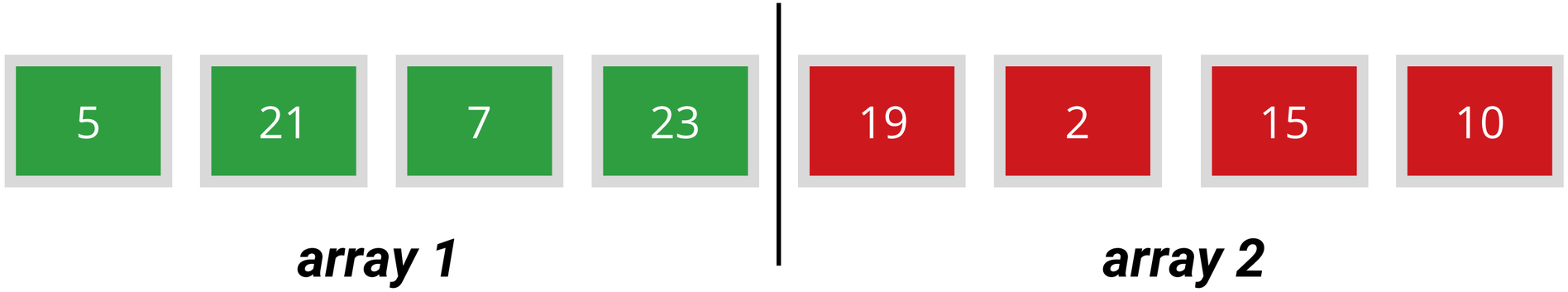
Tamanho dos arrays menores: 4

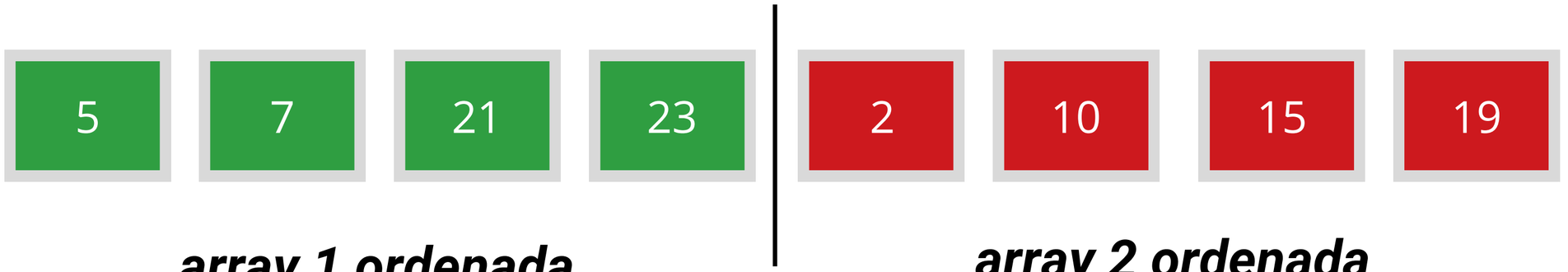
TIMSORT - FUNCIONAMENTO



Ordenar o array 1 e array 2 com algortimo insertion sort.

TIMSORT - FUNCIONAMENTO





TIMSORT - FUNCIONAMENTO

Juntando os array usando o Merge Sort

TIMSORT - FUNCIONAMENTO

2	5	7	10	15	19	21	23

VANTAGENS

Eficiência em dados parcialmente ordenados

O Timsort é altamente eficiente quando o array já está parcialmente ordenado ou contém subsequências ordenadas (runs). Ele detecta esses padrões e os aproveita para reduzir o número de operações necessárias.

Desempenho consistente

O Timsort foi projetado para ter um bom desempenho em uma ampla variedade de cenários, incluindo o pior caso.

DESVANTAGENS

Complexidade de implementação

O Timsort é significativamente mais complexo de implementar do que os demais algoritmos de ordenação.

Uso de memória adicional

O Timsort requer memória adicional para realizar as operações de merge, o que pode ser um problema em sistemas com restrições de memória.

EFICIÊNCIA

MAIOR

- *Dados Parcialmente Ordenados*
- *Arrays de Tamanho Moderado a Grande*

MENOR

- *Dados Completamente desordenados*
- *Arrays Muito Pequenos*
- *Restrições de Memória*