

# Ordenação: Mergesort

Estrutura de Dados e Algoritmos – Ciência da Computação



**INSTITUTO  
FEDERAL**  
Brasília

Prof. Daniel Saad Nogueira  
Nunes

IFB – Instituto Federal de Brasília,  
Campus Taguatinga



# Sumário

---

## 1 Mergesort



# Mergesort

---

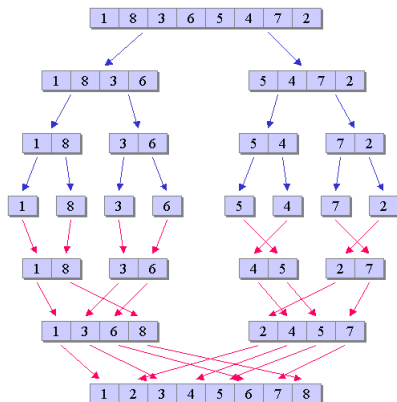
## Mergesort

- O Mergesort se baseia no conceito de Merge (junção) de duas sequências ordenadas. Primeiramente ele subdivide a sequência original na metade e ordena recursivamente essas sequências.
- Caso base: sequência unitária ou vazia, pois essas já são ordenadas.
- Por fim, faz a junção das duas sequências ordenadas para compor uma sequência maior ordenada.
- $(1, 3, 5, 7, 9) + (0, 2, 4, 6, 8) \xrightarrow{\text{merge}} (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$



# Mergesort

## Exemplo





# Mergesort

```
38 void merge_sort(int *v, size_t size) {
39     if (size > 1) {
40         size_t mid = size / 2;
41         /* aloca espaço para os subvetores */
42         int *v1 = malloc(sizeof(int) * mid);
43         int *v2 = malloc(sizeof(int) * size - mid);
44         /* Copia os elementos de v para os subvetores */
45         for (int i = 0; i < mid; i++) {
46             v1[i] = v[i];
47         }
48         for (int i = mid; i < size; i++) {
49             v2[i - mid] = v[i];
50         }

```



# Mergesort

---

```
51     /* Ordena recursivamente a primeira metade */
52     merge_sort(v1, mid);
53     /* Ordena recursivamente a segunda metade */
54     merge_sort(v2, size - mid);
55     /* Faz a junção das duas metades */
56     merge(v, v1, v2, size);
57     /* Libera o espaço alocado */
58     free(v1);
59     free(v2);
60 }
61 }
```



## Mergesort: Merge

---

```
1  static void merge(int *v, int *v1, int *v2, size_t size) {  
2  
3      size_t size_v1 = size / 2;  
4      size_t size_v2 = size - size_v1;  
5      size_t i = 0;  
6      size_t j = 0;  
7      size_t k = 0;  
8
```



## Mergesort: Merge

```
9      /** Enquanto não chegar ao fim da primeira
10      * e da segunda metade **/
11     for (i = 0; j < size_v1 && k < size_v2; i++) {
12         /* Se o elemento da primeira metade
13          * é menor ou igual ao da segunda metade,
14          * insira-o no vetor resultado
15         */
16         if (v1[j] <= v2[k]) {
17             v[i] = v1[j++];
18         }
19         /* Caso contrário, insira o elemento da
20          * segunda metade no vetor resultado */
21         else {
22             v[i] = v2[k++];
23         }
24     }
```





## Mergesort: Merge

---

```
25
26     /** Se ainda restam elementos na primeira partição **/
27     while (j < size_v1) {
28         /* Copiamos os elementos para o vetor resultado */
29         v[i++] = v1[j++];
30     }
31     /** Se ainda restam elementos na segunda partição **/
32     while (k < size_v2) {
33         /* Copiamos os elementos para o vetor resultado */
34         v[i++] = v2[k++];
35     }
36 }
```



# Sumário

---

## 2 Análise



# Mergesort

---

## Análise

A relação de recorrência do Mergesort corresponde à:

$$T(n) = 2 \cdot T(n/2) + O(n) \in \Theta(n \lg n)$$

In-place	Estável
✗	✓

## Observação

- Requer uma quantidade de memória superior a  $O(1)$  (vetores auxiliares).
- Recursivo!