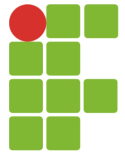
Ordenação: Mergesort

Estrutura de Dados e Algoritmos – Ciência da Computação

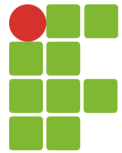
Prof. Daniel Saad Nogueira Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

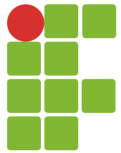
1 Mergesort



Mergesort

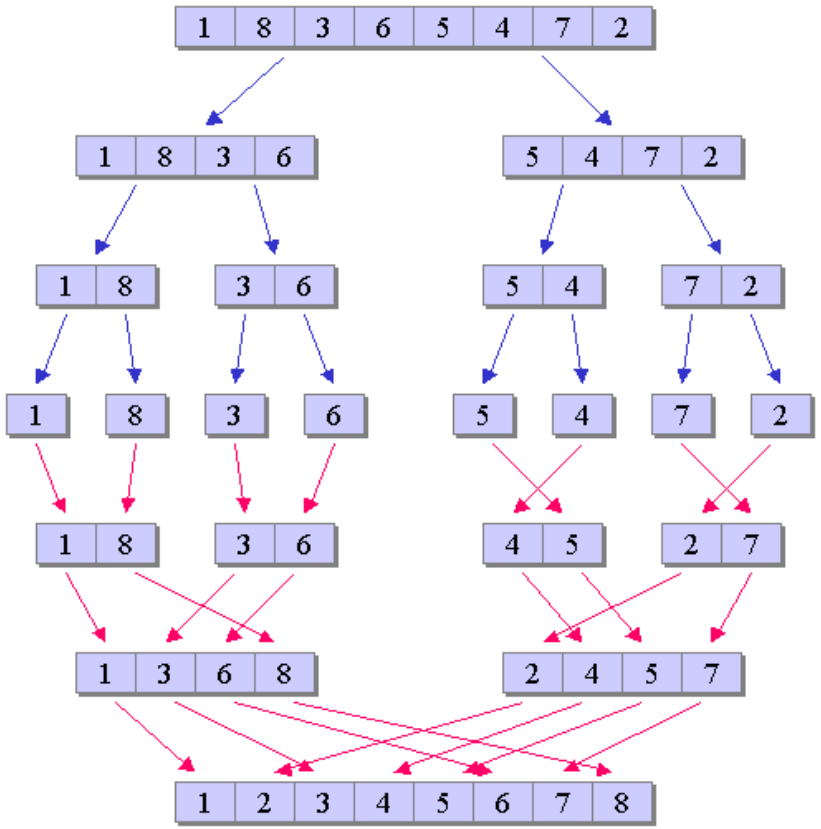
Mergesort

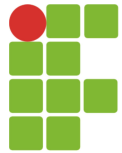
- O Mergesort se baseia no conceito de Merge (junção) de duas sequências ordenadas. Primeiramente ele subdivide a sequência original na metade e ordena recursivamente essas sequências.
- Caso base: sequência unitária ou vazia, pois essas já são ordenadas.
- Por fim, faz a junção das duas sequências ordenadas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



Mergesort

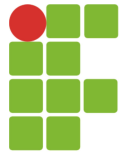
Exemplo





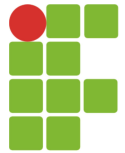
Mergesort

```
40 void merge_sort(int *v, size_t size) {
41     size_t mid;
42     if (size > 1) {
43         mid = size / 2;
44         /* aloca espaço para os subvetores */
45         int *v1 = mallocx(sizeof(int) * mid);
46         int *v2 = mallocx(sizeof(int) * (size - mid));
47         /* Copia os elementos de v para os subvetores */
48         int i;
49         for (i = 0; i < mid; i++) {
50             v1[i] = v[i];
51         }
```



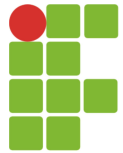
Mergesort

```
52          for (i = mid; i < size; i++) {
53              v2[i - mid] = v[i];
54          }
55          /* Ordena recursivamente a primeira metade */
56          merge_sort(v1, mid);
57          /* Ordena recursivamente a segunda metade */
58          merge_sort(v2, size - mid);
59          /* Faz a junção das duas metades */
60          merge(v, v1, v2, size);
61          /* Libera o espaço alocado */
62          free(v1);
63          free(v2);
64      }
65  }
```



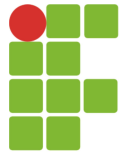
Mergesort: Merge

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



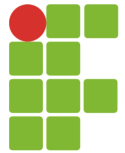
Mergesort: Merge

```
11      /** Enquanto não chegar ao fim da primeira
12       *  e da segunda metade **/
13      for (i = 0; j < size_v1 && k < size_v2; i++) {
14          /* Se o elemento da primeira metade
15           * é menor ou igual ao da segunda metade,
16           * insira-o no vetor resultado
17           */
18          if (v1[j] <= v2[k]) {
19              v[i] = v1[j++];
20          }
21          /* Caso contrário, insira o elemento da
22           * segunda metade no vetor resultado */
23          else {
24              v[i] = v2[k++];
25          }
26      }
```



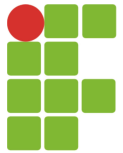
Mergesort: Merge

```
28      /** Se ainda  restam elementos na primeira partição **/
29      while (j < size_v1) {
30          /* Copiamos os elementos para o vetor resultado */
31          v[i++] = v1[j++];
32      }
33      /** Se ainda  restam elementos na segunda partição **/
34      while (k < size_v2) {
35          /* Copiamos os elementos para o vetor resultado */
36          v[i++] = v2[k++];
37      }
38  }
```



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!