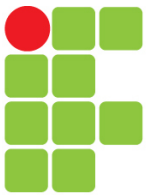
Ordenação por intercalação

Programação de computadores II

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

18 de junho de 2016



INSTITUTO FEDERAL
SANTA CATARINA

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

Tópicos da aula

1. Introdução
2. Ordenação por intercalação
3. Exercícios

Tópico

1 Introdução

2 Ordenação por intercalação

3 Exercícios

Objetivos

- Entender alguns fundamentos matemáticos relacionados com algoritmos de ordenação
- Aprender as ordenações por intercalação
- Conhecer o *Merge Sorting*
- Aplicar ordenação nas estruturas de dados conhecidas

Tópico

1 Introdução

2 Ordenação por intercalação

3 Exercícios

Ordenação por intercalação

Ordenação por intercalação

Compreende no algoritmo onde a ordenação é realizada por *intercalação* de elementos.

- Fusão de sequencias adjacentes

Algoritmo mais conhecido:

- *Merge Sorting*

Merge Sorting

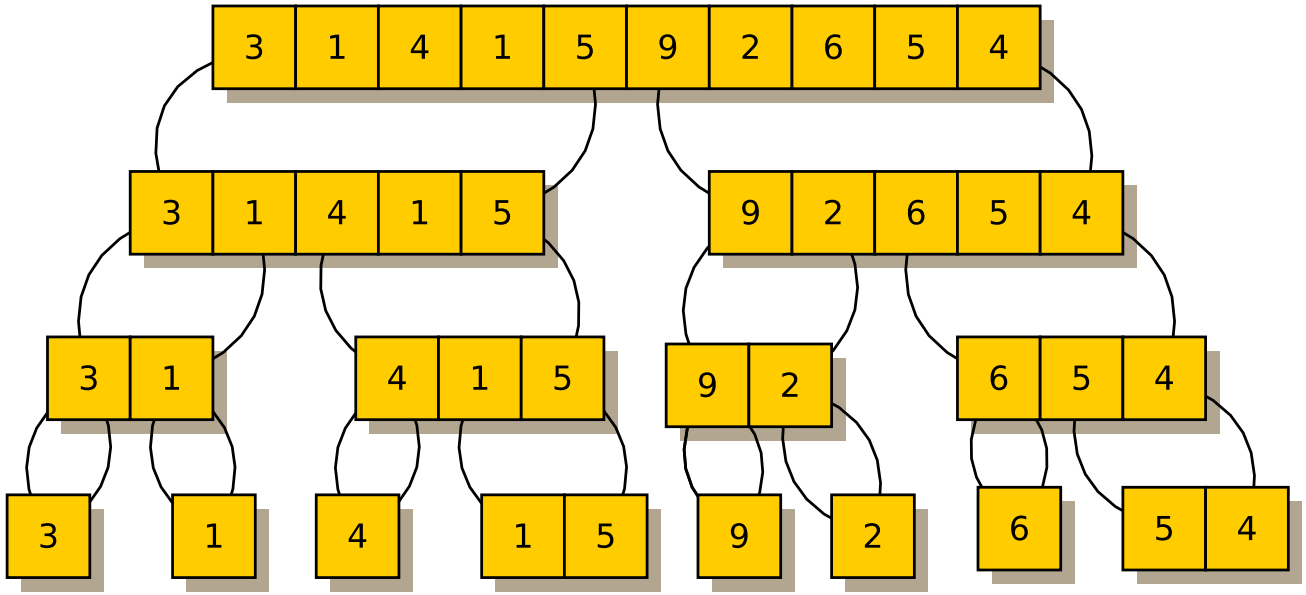
Merge Sorting

O *Merge Sorting* é um algoritmo no estilo "Dividir e conquistar". Este estilo de algoritmo resolve um problema dividindo-o em dois ou mais subproblemas, resolvendo-os cada um destes e combinando a solução no final.

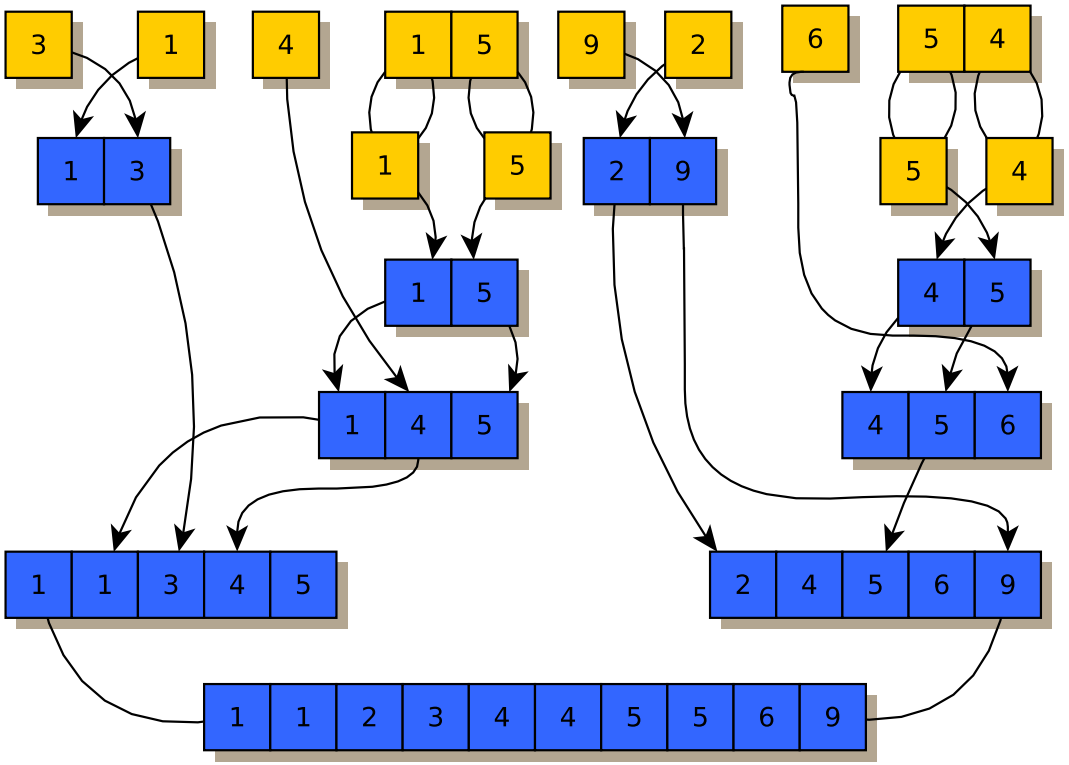
Para ordenar uma sequencia $S = \{s_1, s_2, ...s_3\}$, com $|S| > 1$, o Merge Sorting realiza os seguintes passos:

1. Divide a sequencia em duas de comprimento $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$.
2. Recursivamente ordena cada uma das duas subsequencias.
3. Funde as duas subsequencias para obter o resultado final.

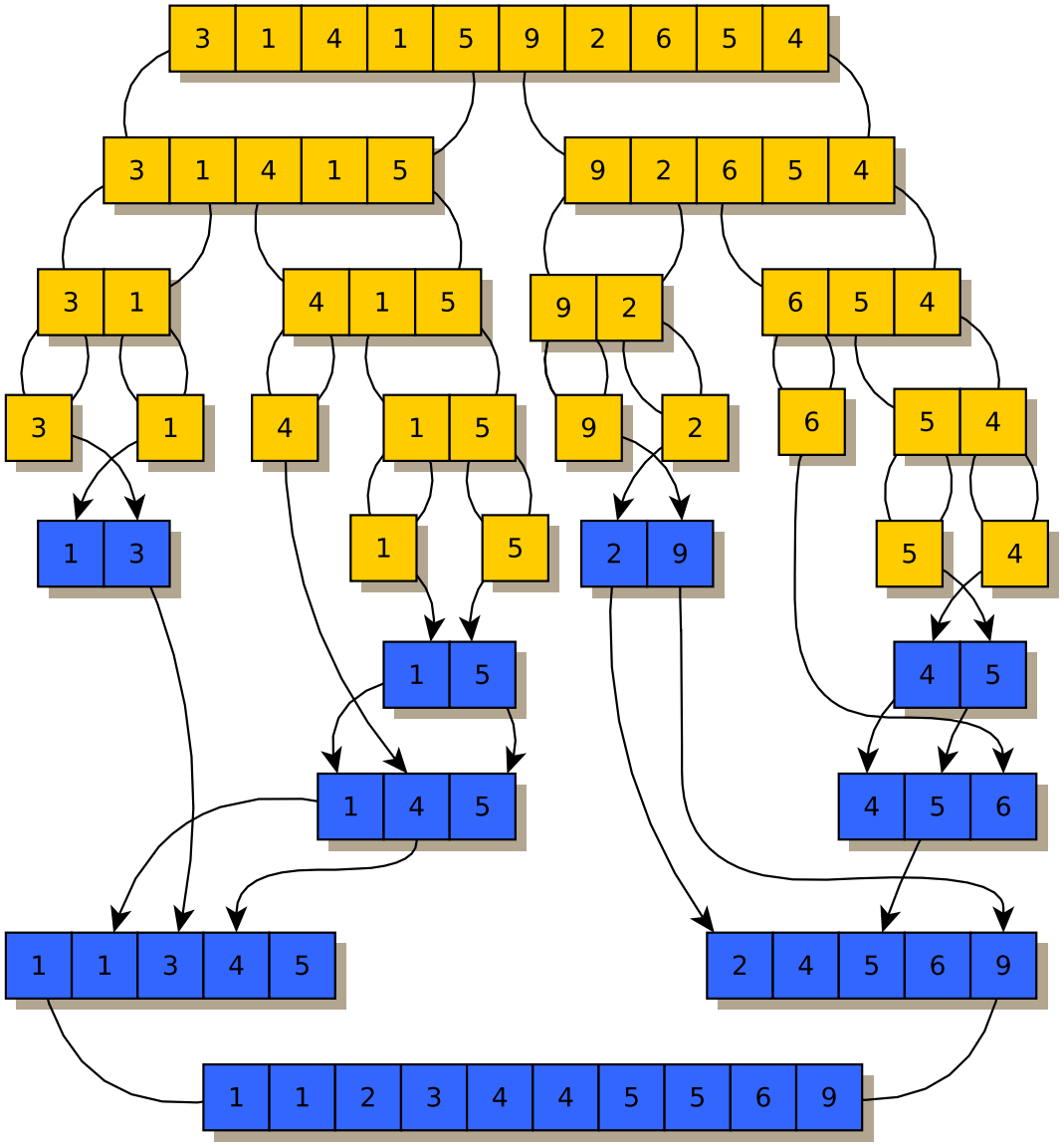
Merge Sorting



Merge Sorting



Merge Sorting



Complexidade

$$T(n) = \begin{cases} O(1), & n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) & n > 1 \end{cases}$$

$$T(n) = \begin{cases} O(1), & n = 1 \\ 2T(n/2) + O(n) & n > 1 \end{cases}$$

Resolvendo som substituições:

$$T(n) = O(n \log n)$$

Algoritmo – mergesort

```
1  mergesort(int *array, int esq, int dir)
2  {
3    if (esq < dir) {
4
5      meio = (esq + dir) / 2;
6
7      mergesort(array, esq, meio);
8      mergesort(array, meio + 1, dir);
9
10     funde(array, esq, meio, dir);
11
12   }
13 }
```

- Para manter $O(n \log n)$, quanto deve ser a complexidade de *funde*?

Algoritmo – funde

```
1  funde (int *array, int esq, int meio, int dir)
2  {
3    i = esq;
4    j = esq;
5    k = meio + 1;
6
7    while (j <= meio && k <= dir) {
8
9      if (array[j] < array[k])
10       temparray[i++] = array[j++]
11     else
12       temparray[i++] = array[k++]
13   }
14
15   while (j <= meio)
16     temparray[i++] = array[j++];
17
18   for (i = esq; i < k; i++)
19     array[i] = temparray[i];
20
21 }
```

Tópico

1 Introdução

2 Ordenação por intercalação

3 Exercícios

Exercícios

- Implemente o *Merge sort* para um vetor de inteiros.
  - Teste seu algoritmo para um vetor de 100.000 de elementos alocados dinamicamente.
  - Inicialize-o com números aleatórios.
  - Meça o tempo de execução para 20 execuções do *Merge sort*.
  - Calcule o tempo de execução médio.
  - **OBS: números aleatórios gerados por:** srand() alimentando a semente com srand (getpid() ^time(NULL));
- Estenda a implementação da lista duplamente encadeada com uma função de ordenação por *Merge sort*.
- Compare com o tempo de execução do *BubbleSort*
- Compare com o tempo de execução do *QuickSort*
- Compare com o tempo de execução do *InsertSort – binário*