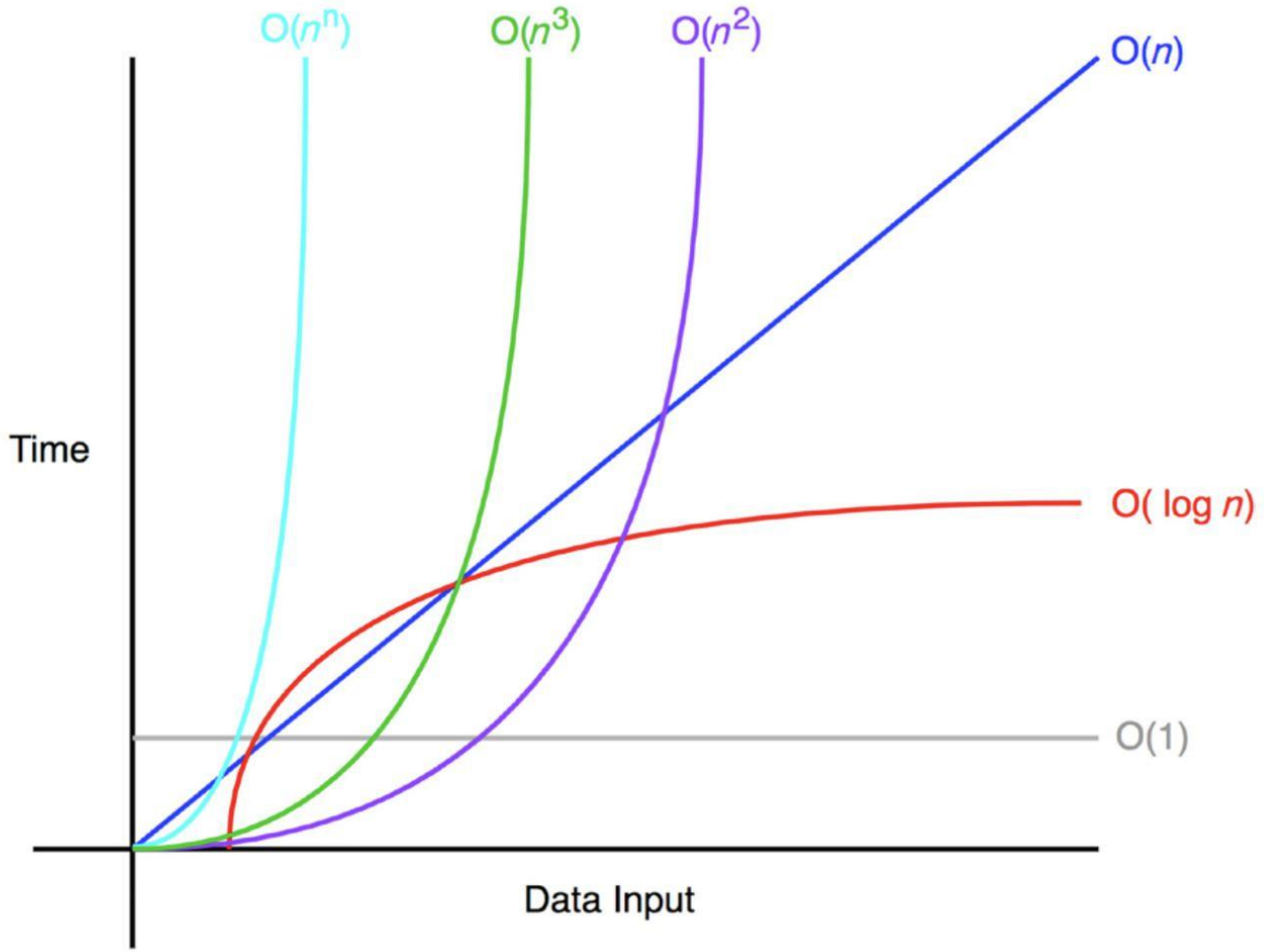
ESTRUTURA DE DADOS

Aula 11 – Ordenação II

Prof. Rodrigo Maciel

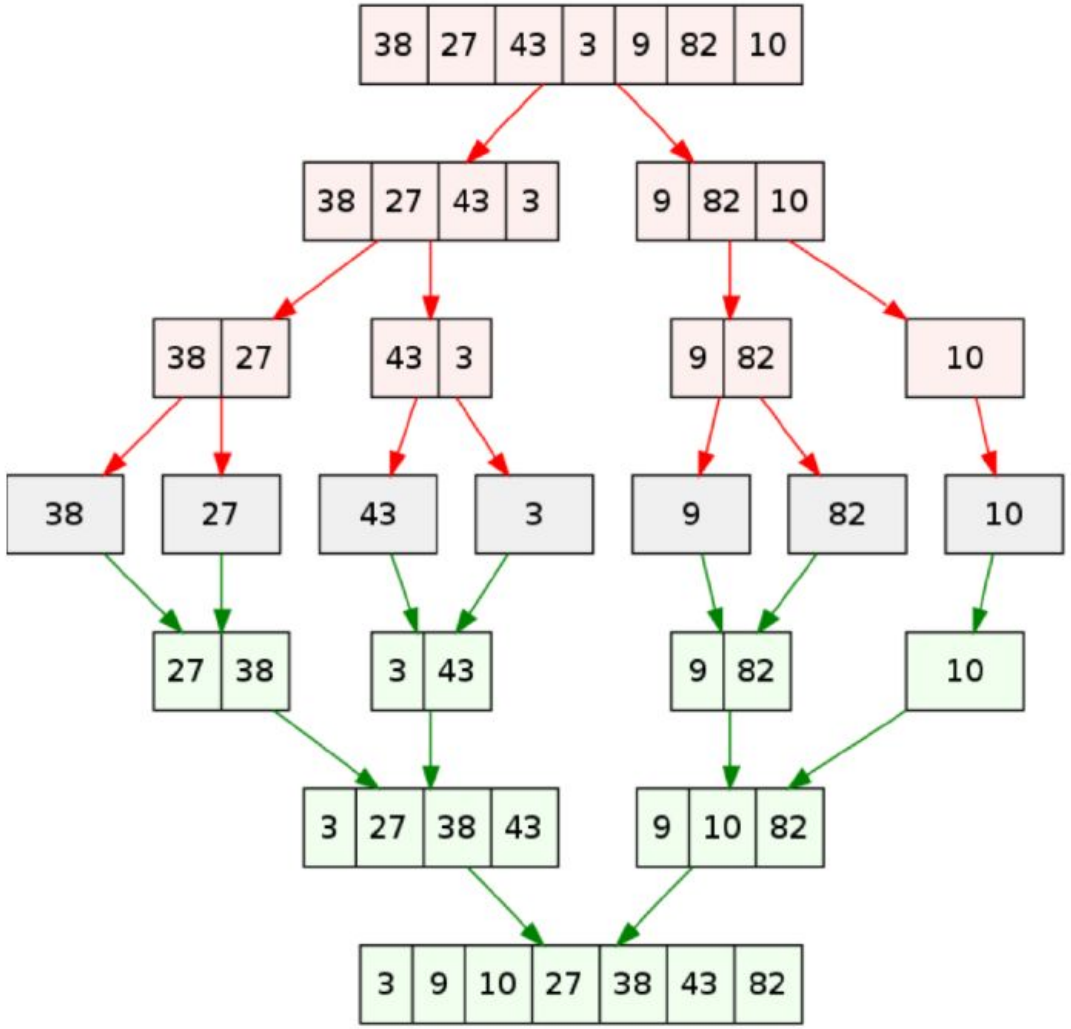
CURIOSIDADE: Notação Big-O



MERGE SORT

- Um dos algoritmos de ordenação mais populares
- Funcionamento:
  - Divisão do problema em subproblemas (dividir e conquistar)
  - Divide o vetor continuamente pela metade, ordena e combina (merge)
- Mesmo desempenho para melhor caso (randômico) e pior caso (ordenação inversa);
- Visualização on-line: https://visualgo.net/en/sorting

MERGE SORT


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

MERGE SORT_PARTE 1

```
def merge_sort(array):
# Divisão do Array
    se tamanho(array) > 1 então:
        divisão <- metade(array)
        esquerda <- cópia(array[0 até divisão])
        direita <- cópia(array[divisão até final])

        merge_sort(esquerda)
        merge_sort(direita)

        i = j = k = 0
```

MERGE SORT_PARTE 2

```
# Ordena esquerda e direita
enquanto i < tamanho(esquerda) E j < tamanho(direita) então:
    se esquerda[ i ] < direita[ j ] então:
        array[ k ] <- esquerda[ i ]
        i += 1
    senão:
        array[ k ] <- direita[ j ]
        j += 1
    k += 1
```

MERGE SORT_PARTE 3

```
# Ordenação final
enquanto i < tamanho(esquerda) então:
   array[ k ] <- esquerda[ i ]
    i += 1
    k += 1

enquanto j < tamanho(direita) então:
   array[ k ] <- direita[ j ]
    j += 1
    k += 1

 return array
```

QUICK SORT

- Algoritmo rápido e eficiente criado em 1960 para traduzir um dicionário de inglês para russo
- Funcionamento:
  - O vetor é dividido em sub vetores que são chamados recursivamente para ordenar os elementos;
  - Estratégia da divisão e conquista
- Pior caso: **O(n2)** – quando o elemento pivot é o maior ou menor elemento;
- Melhor caso: **O(n*log n)**;
- Visualização on-line: https://visualgo.net/en/sorting

QUICK SORT_PARTE 1

```
def particao(array, inicio, final):
  pivo <- vetor[ final ]
  i <- inicio - 1

  para cada j no range(inicio, final) então:
    se array[ j ] <= pivo então:
      i <- i + 1
      array[ i ], array[ j ] <- array[ j ], array[ i ]
  array[i + 1], array[final] <- array[final], array[i + 1]
  return i + 1
```

QUICK SORT_PARTE 2

```
def quick_sort(vetor, inicio, final):
  se inicio < final então:
    posicao <- particao(vetor, inicio, final)

    # Esquerda
    quick_sort(vetor, inicio, posicao - 1)

    # Direito
    quick_sort(vetor, posicao + 1, final)
  return vetor
```