MERGE SORT

Conceitos gerais, importância, custo e prática.

1. Entendendo DS

É a forma como as informações são organizadas.

➔ **Quantidade de Informações**

Essencial para que as máquinas possam fazer a leitura de uma grande quantidade de dados.

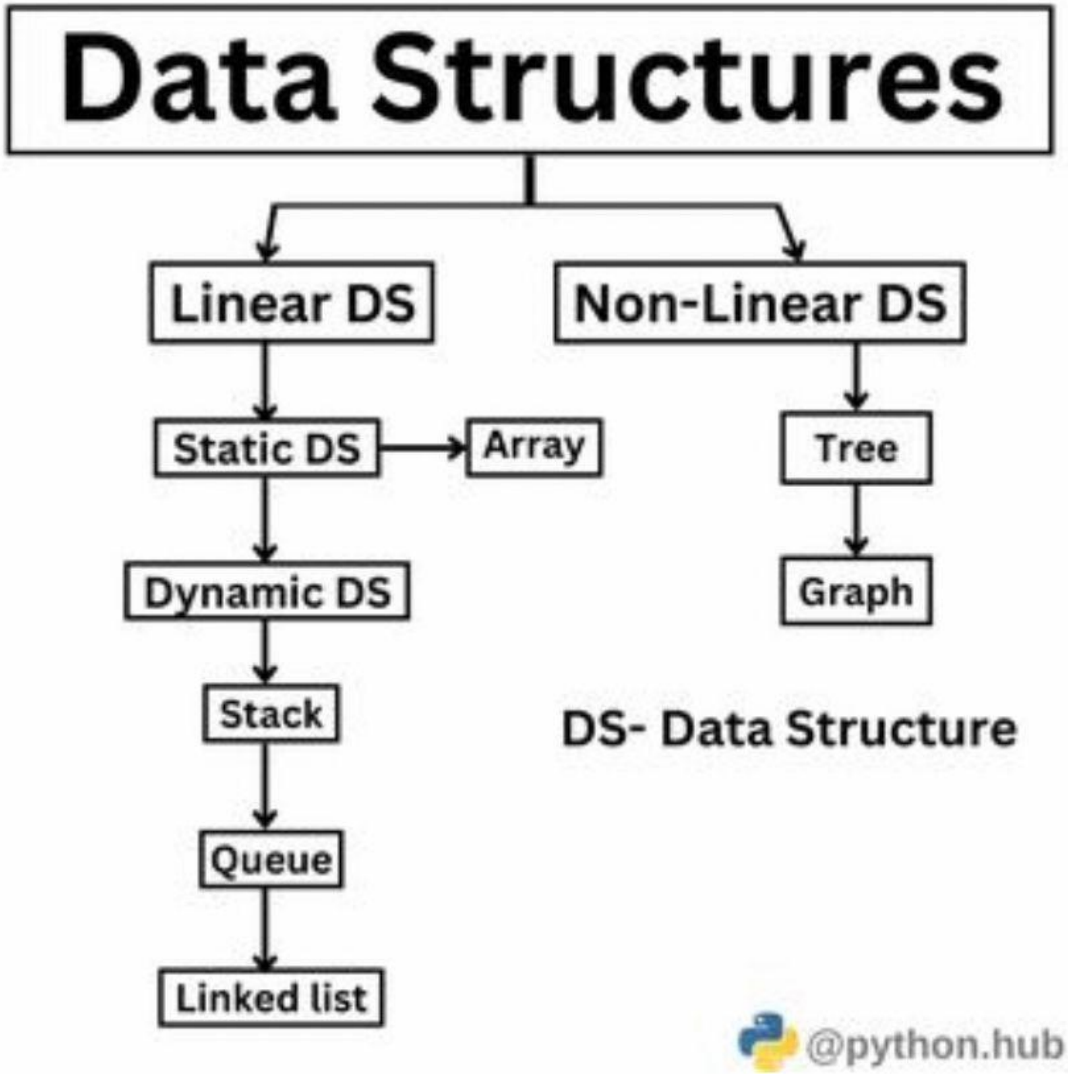
➔ **Eficiência**

Determinado dado deverá ser implementado de maneira eficiente.

➔ **Organização**

Formas de organização de dados para diversos tipos de processamento

Tipos:



Data Structures
Linear DS
Non-Linear DS
Static DS
Array
Dynamic DS
Stack
Queue
Linked list
Tree
Graph
DS- Data Structure
@python.hub

2. Algoritmos de Comparação

- **Alguns tipos**
  Quick Sort, Merge Sort, Heap Sort, Selection Sort, Bubble Sort, Bucket Sort, Radix Sort…
- **Problema real**
  Imagine que você tem uma Base de dados com 9999999 nomes não ordenados, que precisam ser ordenados em forma alfabética até amanhã.

3. Custo

➔ **A análise de custo em um algoritmo**

- Tempo gasto para executar
- Espaço de memória ocupado na execução

➔ **Eficiência**

Quick Sort

➔ **Menor Custo**

Insertion Sort

➔ Se o custo de um algoritmo é igual ao menor custo possível, o algoritmo é ótimo para a medida de custo considerada.(a medida de custo depende do tamanho da entrada de dados.)

3.1. Custo

➔ **Função de Complexidade ou Função de Custo**

- Função de complexidade de tempo:

Classificada pela natureza da função T(n).

Por exemplo: Um algoritmo com

$T(n) = O(n)$ é chamado de **algoritmo de tempo linear**.

Um algoritmo com $T(n) = O(2^n)$ é chamado de **algoritmo de tempo exponencial.**

A seguir: Tabela de complexidade de tempo comum

Nome	Classe de Complexidade	Tempo de execução ($T(n)$)	Exemplos de tempo de execução	Exemplos de algoritmos
Tempo constante		$O(1)$	10	Determinando se o número é par ou ímpar
Função inversa de Ackermann		$O(\alpha(n))$		Tempo amortizado por operação usando um conjunto dijunto
Logaritmo Iterativo		$O(\text{log-star}(n))$		Algoritmo de Cole-Vishkin
log-logarítmico		$O(\log \log n)$		Tempo amortizado por operação usando uma fila de prioridades limitada [1]
Tempo logarítmico	DLOGTIME	$O(\log n)$	$\log n$, $\log(n^2)$	Busca binária
Tempo poli-logarítmico		$\text{poly}(\log n)$	$(\log n)^2$	
potência fracionária		$O(n^c)$ onde $0 < c < 1$	$n^{1/2}$, $n^{2/3}$	Procurando em uma kd-tree

Tempo linear		$O(n)$	n	Procurando o menor item em um array não ordenado
"n log star n"		$O(n\text{log-star}(n))$		Algoritmo de triangulação de polígonos de Seidel.
Tempo linearitmico		$O(n \log n)$	$n \log n$, $\log n!$	A ordenação por comparação mais rápida
Tempo quadrático		$O(n^2)$	n^2	Bubble sort; Insertion sort
Tempo cúbico		$O(n^3)$	n^3	Multiplicação ingênua de duas matrizes $n \times n$. Calculando correlação parcial.
Tempo polinomial	P (classe de complexidade)	$2^{O(\log n)} = \text{poly}(n)$	n, $n \log n$, n^{10}	Algoritmo de Karkamar para programação linear; Teste de primalidade de AKS
Tempo quasi-polinomial	QP	$2^{\text{poly}(\log n)}$	$n^{\log \log n}$, $n^{\log n}$	Melhor conhecido $O(\log^2 n)$-algoritmo de aproximação para o problema da árvore de Steiner dirigida.
Tempo sub-exponencial (primeira definição)	SUBEXP	$O(2^{n^{\varepsilon}})$ para todos $\varepsilon > 0$	$O(2^{\log n^{\log \log n}})$	Assumindo a complexidade do teorema de conjecturas, Polinômio probabilístico com limitação de erro está contido em um SUBEXP.[2]

3.2. Custo

➔ **Função de Complexidade ou Função de Custo**

- Função de complexidade de espaço:
- Espaço total ocupado pelo algoritmo em relação ao tamanho de entrada(input). A complexidade do espaço inclui o espaço auxiliar e o espaço usado pelo input.
- Espaço auxiliar: Espaço extra ou temporário usado por um algoritmo em sua execução.

Se criarmos um array de tamanho x, isso exigirá espaço

O (x). Se criarmos um array bidimensional de tamanho x*x, isso exigirá espaço

O (x^2).

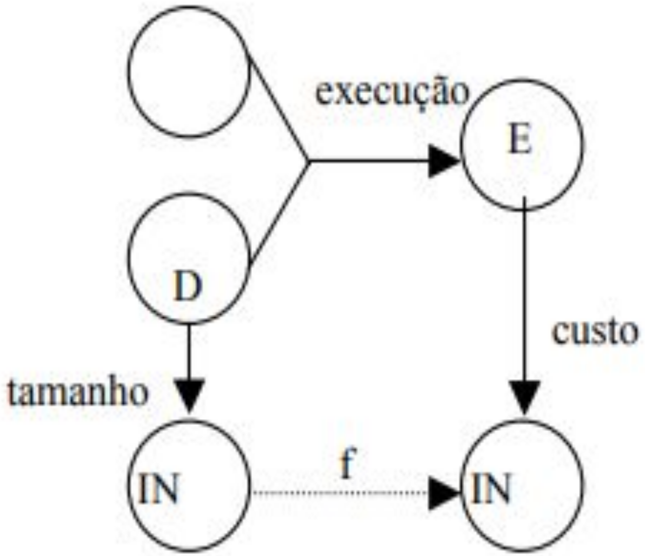


Figura 1 – Diagrama de Complexidade

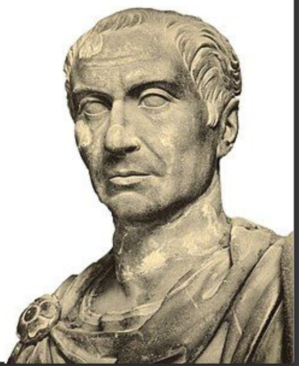
onde:

- <u>execução</u>: x D $\rightarrow$ E; *execução*(a, d) := seqüência de execuções de operações fundamentais efetuadas na execução do algoritmo a, com entrada d.
- <u>custo:</u> E $\rightarrow$ IN; *custo*(s) := comprimento da seqüência s, definido conforme o peso estabelecido para as operações fundamentais.
- <u>tamanho:</u> D $\rightarrow$ IN; *tamanho*(d) := tamanho da entrada d.

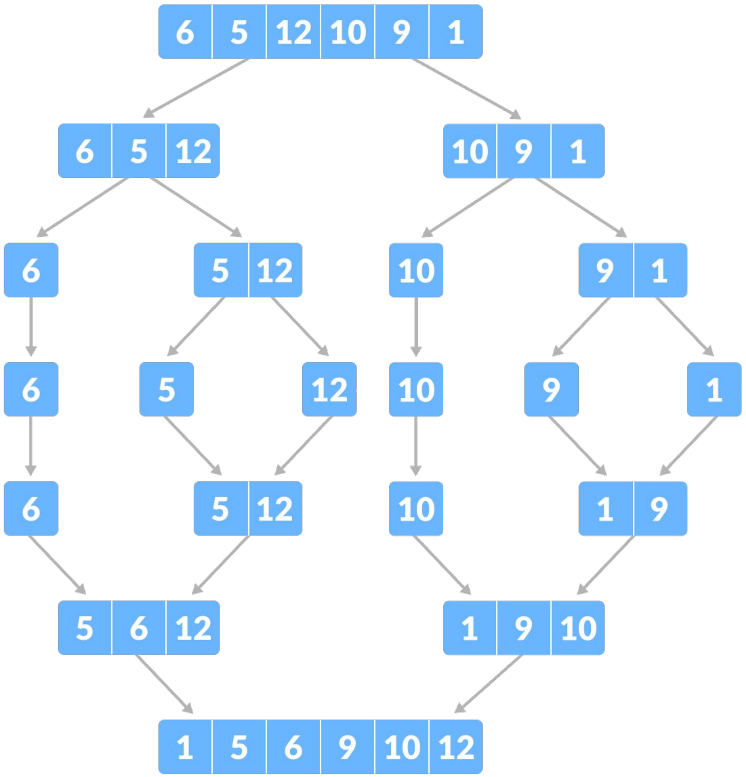
4. Merge Sort

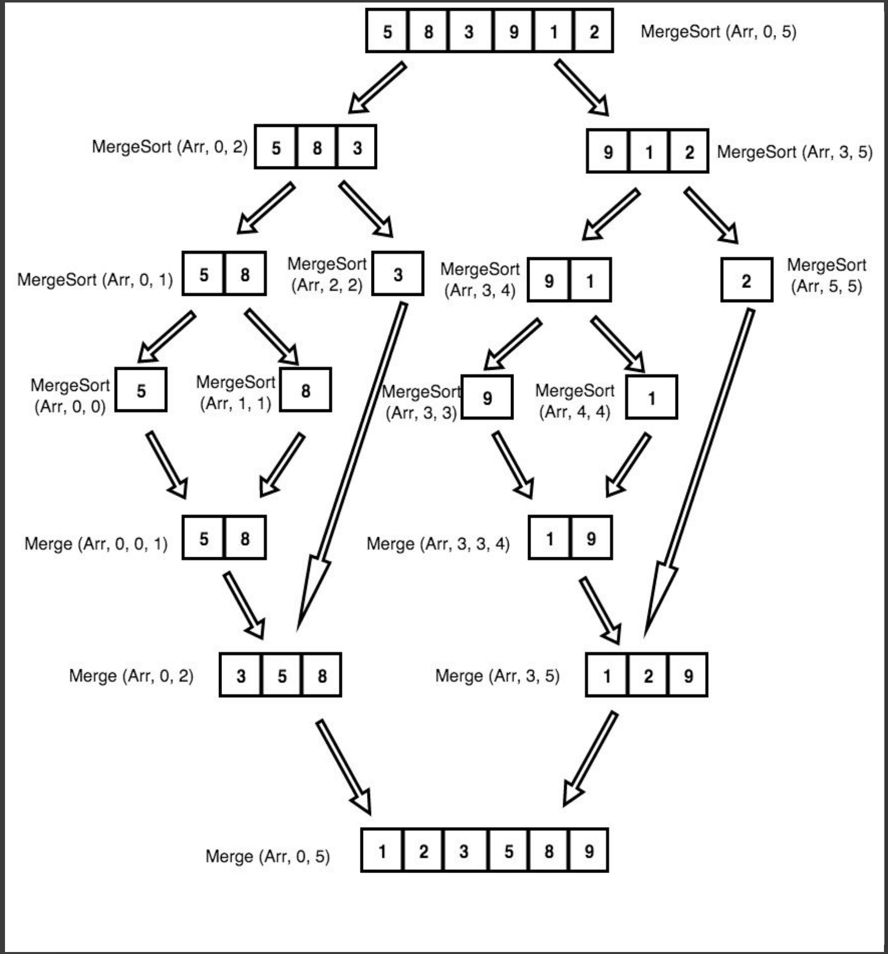
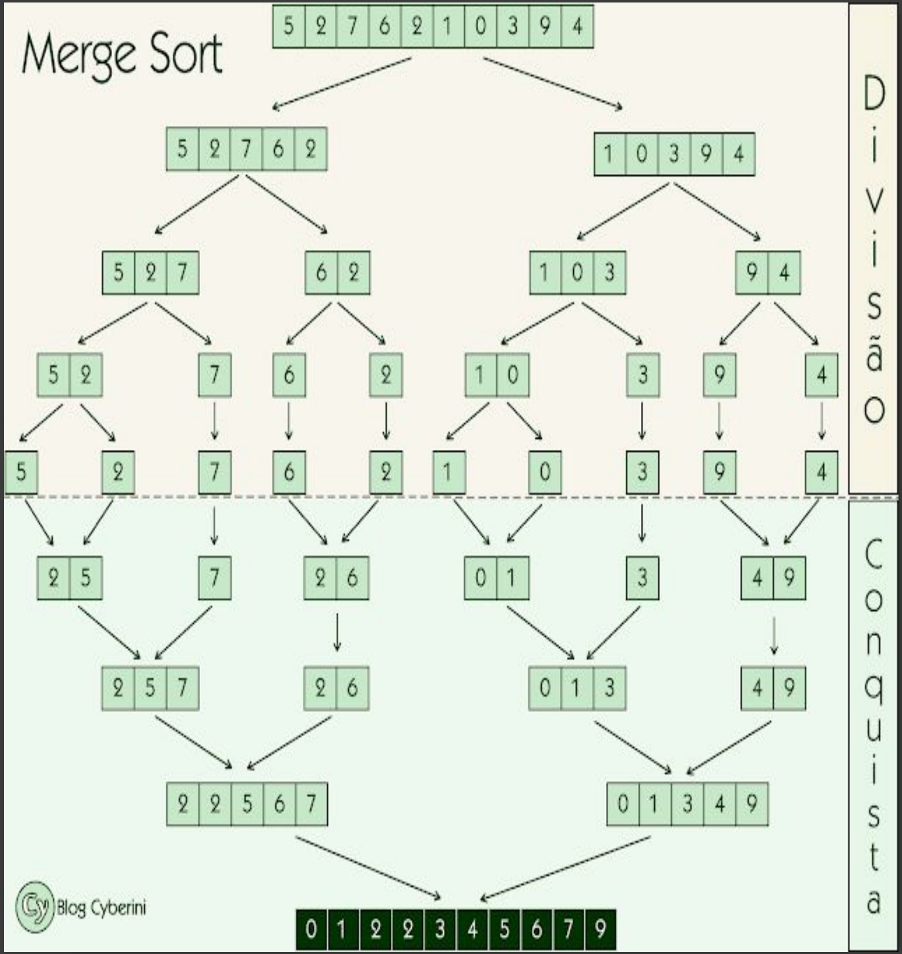
➔ ***Divide Et Impera***

- Passo-a-passo:
- **1:** Dividir a matriz em duas metades
- **2:** Classificar cada metade
- **3:** Juntar a matriz novamente
- **4:** Repetir até ordenar



- Tio Julinho que disse isso




5 8 3 9 1 2
MergeSort (Arr, 0, 5)
MergeSort (Arr, 0, 2)
5 8 3
9 1 2
MergeSort (Arr, 3, 5)
MergeSort (Arr, 0, 1)
5 8
MergeSort (Arr, 2, 2)
3
MergeSort (Arr, 3, 4)
9 1
2
MergeSort (Arr, 5, 5)
MergeSort (Arr, 0, 0)
5
MergeSort (Arr, 1, 1)
8
MergeSort (Arr, 3, 3)
9
MergeSort (Arr, 4, 4)
1
Merge (Arr, 0, 0, 1)
5 8
Merge (Arr, 3, 3, 4)
1 9
Merge (Arr, 0, 2)
3 5 8
Merge (Arr, 3, 5)
1 2 9
Merge (Arr, 0, 5)
1 2 3 5 8 9

Merge Sort
5 2 7 6 2 1 0 3 9 4
5 2 7 6 2
1 0 3 9 4
5 2 7
6 2
1 0 3
9 4
5 2
7
6
2
1 0
3
9
4
5
2
7
6
2
1
0
3
9
4
2 5
7
2 6
0 1
3
4 9
2 5 7
2 6
0 1 3
4 9
2 2 5 6 7
0 1 3 4 9
0 1 2 2 3 4 5 6 7 9
Blog Cyberini
Divisão
Conquista

4.1. Merge Sort

➔ **Por que merge sort?**

- Ele tem uma complexidade de tempo de O(n*log n) - Classifica arrays com rapidez.
- Classificação estável: ordem dos elementos com valores iguais é preservada durante a classificação.

O Merge sort *é definido como um algoritmo de classificação que funciona dividindo um array em subarrays menores, classificando cada subarray e, em seguida, mesclando os subarrays classificados novamente para formar o array classificado final.*

VAMOS PRATICAR?