

Devido ao grande volume de dados e à demanda por respostas rápidas, é evidente a necessidade de algoritmos mais eficientes no mundo atual. Entender a complexidade dos algoritmos é um princípio fundamental no desenvolvimento de software. É crucial compreender o funcionamento do código, mesmo que haja linguagens de programação modernas gerenciando a execução dos algoritmos. A criação de uma função de busca de contatos em uma agenda telefônica, onde os contatos já estão ordenados alfabeticamente, é um exemplo prático.

Para medir a complexidade, podemos usar métricas como o tempo de execução, o número de operações e a taxa de crescimento. Cada uma delas analisa diferentes aspectos do comportamento dos algoritmos à medida que o tamanho da entrada muda.

As definições de Complexidade de Tempo e Complexidade de Espaço são:

Notação Big O e a sua importância: De um modo mais simples, a notação Big O descreve utilizando termos algébricos a complexidade do código utilizado. Dando uma olhada em um exemplo típico, $O(n^2)$ a letra “n” representa o tamanho da entrada enquanto a função nos dá a ideia da complexidade do algoritmo

Complexidade de Tempo: Diz respeito ao tempo que um algoritmo leva para executar, dependendo do tamanho da entrada. Normalmente usamos a notação assintótica. Se um algoritmo tem complexidade de tempo $O(n)$, seu tempo de execução cresce linearmente com o tamanho da entrada. Complexidade de tempo é estimada pela contagem do número de operações elementares realizadas pelo algoritmo, onde cada operação leva um tempo fixo para ser executada. O tempo total e o número de operações diferem no máximo por um fator constante.

Complexidade de Espaço: Diz respeito à quantidade de memória (espaço) que um algoritmo utiliza durante sua execução. A complexidade de tempo é expressa usando notação assintótica, da mesma forma.

Classificação dos Algoritmos de Ordenação (bubblesort, selection sort, merge sort e quick sort):

Bubblesort:

Complexidade de tempo no pior caso: $O(n^2)$

Bubblesort é um algoritmo de ordenação simples, percorrendo repetidamente a lista, comparando os elementos adjacentes e trocando eles na ordem correta. O processo é repetido até que esteja completamente ordenada a lista.

Selection Sort:

Complexidade de tempo no pior caso: $O(n^2)$

Selection Sort é um algoritmo que divide a lista em duas partes, uma lista ordenada e a outra não ordenada. Procurando o menor elemento na lista que não está ordenada e o coloca no final da lista já ordenada, esse processo é repetido até que esteja completamente ordenada a lista

Merge Sort:

Complexidade de tempo no pior caso: $O(n \log n)$

Merge Sort é um algoritmo de ordenação baseado em dividir para conquistar. Ele divide em partes menores a lista, ordenando cada parte separadamente e após as ordenações, combina as partes para obter a lista finalizada e ordenada. Merge Sort é eficiente para listas grandes, mas requer espaço adicional para armazenar as partes divididas no processo.

Quick Sort:

Complexidade de tempo no pior caso: $O(n^2)$

Quick Sort também é um algoritmo baseado em dividir para conquistar. Ele seleciona um elemento como pivô e assim reorganiza a lista de modo que todos os elementos menores que o elemento selecionado estejam antes dele e todos os maiores após o elemento selecionado. Quick Sort tem um desempenho médio muito bom, porém o seu pior caso ocorre quando a lista já está ordenada ou então quase ordenada por inteiro.

Referências:

<https://www.alura.com.br/artigos/analise-complexidade-algoritmos-qual-importancia>

<https://www.escoladnc.com.br/blog/a-importancia-da-complexidade-algoritmica-na-ciencia-da-computacao/>

<https://www.freecodecamp.org/portuguese/news/o-que-e-a-notacao-big-o-complexidade-de-tempo-e-de-espaco/>

https://pt.wikipedia.org/wiki/Complexidade_de_tempo

<https://medium.com/@dev.gilberth/uma-visão-geral-sobre-os-principais-algoritmos-de-ordenação-4b930c917183>

ASCENCIO, Ana Fernanda Gomes; ARAÚJO, Graziela Santos de. Estruturas de dados: algoritmos, análise da complexidade e implementações em Java e C/C++. São Paulo: Pearson Prentice Hall, 2010. Disponível em: https://mega.nz/file/mE50nJgY#iqrhU7JAJB9xGW17CbP0slhvzuCRWnxf8u_GRkEFhU.

BORIN, Vinícius Pozzobon. Estruturas de Dados. Curitiba: Contentus, 2020. Disponível em: https://mega.nz/file/uJlmFLaS#2yTR40CwXGHLNs8Kqkg-w74_0adNSw-RtSinpbzLboY.

PIVA JUNIOR, Dilermando et al. Estruturas de dados e técnicas de programação. Rio de Janeiro: Elsevier, 2014. Disponível em: https://mega.nz/file/vVQTTT7K#By0gTZNXohaZvaqmr53yTEI_H9FgB51ZC8eObjD1kGs.

SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. Estruturas de dados e seus algoritmos. 3. ed. Rio de Janeiro: LTC, 2015. Disponível em: <https://mega.nz/file/6E4CjYIL#IUJCqiu02wbXidhsMC9VbqG2JhMdOV6r6VRUtQOCeWk>.