Gabriela de Melo Freitas PWBE

**Merge Sort**

O algoritmo Merge Sort é um método eficiente de ordenação baseado no conceito de divisão e conquista. O método divide a lista em elementos e listas menores, criando assim sub-listas e ordenando-as para depois obter uma lista ordenada como resultado final. Esse processo acontece até que cada sub-lista tenha apenas um elemento.

Esse método é vantajoso porque divide um problema principal em sub-problemas menores e independentes, facilitando dessa forma o entendimento do código. O Merge Sort é comumente utilizado em aplicações que utilizam organização de dados.

**Insertion Sort**

O algoritmo Insertion Sort é um dos métodos mais simples e funcionais para ordenar os elementos de uma lista. Esse método funciona comparando o elemento atual com seu elemento anterior e os organizando de forma correta na lista. O Insertion Sort funciona como a organização de cartas em um baralho, que, quando recebemos uma carta nova, a comparamos com as anteriores e a inserimos em seu devido lugar de acordo com a sua ordem.

O Insertion Sort tem alta eficiência quando utilizado em lista pequenas ou ordenadas, mas quando as listas são muito grandes ou não apresentam ordenação, outros algoritmos de ordenação como o Merge Sort ou o Quick Sort podem ser mais eficientes.

**Bubble Sort**

O Bubble Sort é um algoritmo de ordenação que em uma lista, compara os elementos da esquerda e da direita e troca as posições desses elementos caso eles estejam em uma ordem errada. Esse método repete esse processo até que todos os elementos da lista estejam na ordem correta.

**Busca Linear**

A Busca Linear funciona verificando cada item e se algum deles atender uma certa condição de correspondência estabelecida, esse item é retornado. Caso o um item que corresponda à condição não seja encontrado, a pesquisa continua até o final dos itens. Dependendo da posição do item desejado, a Busca Linear pode não ser tão eficaz. Por exemplo, em uma lista de cinco posições em que o item desejado está na posição 3, o algoritmo fará quatro comparações antes de encontrar o item correspondente. Assim, é uma solução de grande eficiência para listas pequenas, mas acaba se tornando menos viável e eficiente em conjuntos de itens maiores.

O tempo da Busca Linear é dado com base no número de elementos da lista, então quanto maior o número de elementos, maior o tempo de execução. A Busca Linear é simples de ser entendida e sua implementação também é fácil. Além disso, a Busca Linear funciona para listas não ordenadas, mas em listas ordenadas, outros algoritmos de busca podem ser mais viáveis.

**Busca Binária**

A Busca Binária é um algoritmo de buscas eficiente usado para encontrar um elemento em uma lista ordenada. Ela funciona seguindo o princípio de dividir pela metade, dividindo a lista pela metade em cada iteração e verificando em qual das metades o elemento correspondente está localizado. Assim, o algoritmo descarta metade da lista a cada iteração e reduz o tempo de busca, apresentando mais rapidez quando comparada com outros algoritmos.

**Complexidade de Algoritmos**

A complexidade de um algoritmo é analisada com base em seu tempo e espaço consumidos quando o tamanho da entrada aumenta. O desempenho de um algoritmo pode variar com base em seu processador, disco, memória e outros parâmetros de seu hardware. A complexidade é utilizada para medir a velocidade de um algoritmo.

A Notação Big O é uma forma padrão de representar a complexidade algorítmica, principalmente a taxa de crescimento. Ela expressa o comportamento assintótico dos algoritmos conforme o tamanho da entrada aumenta. O(1), O(log n), O(n), O(n log n), O(n^2) e O(2^n) são exemplos de Notações Big O, cada exemplo representa uma taxa de crescimento e complexidade algorítmica específica.

**Merge Sort:** O(n log n); Apresenta maior eficiência.

**Insertion Sort:** Pensando em um cenário em que a lista já está ordenada, a complexidade do algoritmo é de O(n), realizando apenas uma passagem pela lista, não necessitando de trocas. No cenário de uma lista desordenada, o algoritmo compara cada elemento com todos os outros anteriores, levando a complexidade para O(n²).

**Bubble Sort:** Pensando em um cenário em que a lista já está ordenada, a complexidade do algoritmo é de O(n) se houver uma otimização que detecta se a lista está ordenada, realizando assim apenas uma passagem pela lista e não necessitando de trocas. No cenário de uma lista desordenada, o algoritmo compara cada elemento com todos os outros anteriores, levando a complexidade para O(n²).

**Busca Linear:** Apresenta complexidade de O(n) no melhor e no pior cenário.

**Busca Binária:** Apresenta complexidade de O(log n).

**Referências**

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

https://awari.com.br/busca-binaria-em-python-aprenda-a-encontrar-elementos-de-forma-eficiente/

https://www.dio.me/articles/analise-de-algoritmos-tipos-de-busca-busca-linear

https://medium.com/@josianebarros/bubble-sort-defini%C3%A7%C3%A3o-e-exemplo-em-python-d7f2939d5b2e

https://awari.com.br/aprenda-a-implementar-o-algoritmo-de-ordenacao-insertion-sort-em-python/

https://awari.com.br/aprenda-a-implementar-o-algoritmo-de-merge-sort-em-python/