**Algoritmo de Ordenação**

Os algoritmos de ordenação baseiam-se em um conjunto de instruções, pegam uma matriz ou lista como entrada e organizam os itens em uma ordem específica.  
 **Qual a importância de Algoritmos de Ordenação?**

A ordenação muitas vezes pode reduzir significativamente a complexidade de um problema, trata-se de um algoritmo muito importante na Ciência da Computação. Esses algoritmos têm aplicações diretas em algoritmos de busca, algoritmos de banco de dados, métodos de divisão e conquista, algoritmos de estrutura de dados entre muitos outros.

**Alguns Algoritmos de Ordenação:**

* Selection Sort
* Bubble Sort
* Insertion Sort
* Merge Sort
* Quick Sort
* Heap Sort
* Counting Sort
* Radix Sort
* Bucket Sort

**Classificação de um Algoritmo de Ordenação:**

Os algoritmos de classificação podem ser categorizados com base nos seguintes parâmetros:

1 - Com base no número de trocas ou inversão. Este é o número de vezes que o algoritmo troca elementos para classificar a entrada. Selection Sort requer o número mínimo de trocas.

2 - Com base no número de comparações. Este é o número de vezes que o algoritmo compara elementos para classificar a entrada.

Usando a notação Big-O (A notação Big O é uma forma de descrever a velocidade ou complexidade de um determinado algoritmo), os exemplos de algoritmos de classificação listados acima requerem pelo menos O(nlogn) comparações no melhor caso e O(n^2) comparações no pior caso para a maioria das saídas.

3- Baseado em recursão ou não recursão. Alguns algoritmos de ordenação, como Quick Sort, usam técnicas recursivas para classificar a entrada. Outros algoritmos de ordenação, como Selection Sort ou Insertion Sort, usam técnicas não recursivas. Finalmente, algum algoritmo de ordenação, como Merge Sort, faz uso de técnicas recursivas e não recursivas para classificar a entrada.

4 - Os algoritmos de classificação de estabilidade são considerados como stable se o algoritmo mantivesse a ordem relativa dos elementos com chaves iguais. Em outras palavras, dois elementos equivalentes permanecem na mesma ordem na saída classificada como estavam na entrada.

5 - Insertion sort, Merge Sort, e Bubble Sort são estáveis.

6 - Heap Sort e Quick Sort não são estáveis.

7 - Com base no requisito de espaço extra, os algoritmos de classificação são considerados como in place se eles precisassem de um O (1) espaço extra constante para classificação.

8 - Insertion sort e Quick-sort são in place classificados conforme movemos os elementos em torno do pivô e não usam uma matriz separada que NÃO é o caso na classificação por mesclagem, onde o tamanho da entrada deve ser alocado de antemão para armazenar a saída durante a classificação.

9 - Merge Sort é um exemplo de out place classificação, pois requer espaço de memória extra para suas operações.

**Selection Sort:**

O Selection Sort uma rotina bem simples e direta: encontrar o menor elemento e coloca-lo na primeira posição. A ordenação nada mais é do que aplicar essa rotina repetidas vezes para o restante de array.  
Pontos importantes:

- O Selection Sort é in-place e O(n 2).

- A implementação clássica do Selection Sort não é estável.

- Embora sejam da mesma classe de complexidade, o Selection Sort, na prática, é mais lento do que o Insertion Sort.

- O Selection Sort não é considerado um algoritmo eficiente para grandes entradas. Há alternativas O(n∗logn), como Quick Sort e Merge Sort, além de alternativas lineares como o Counting Sort.­

**Bubble Sort**

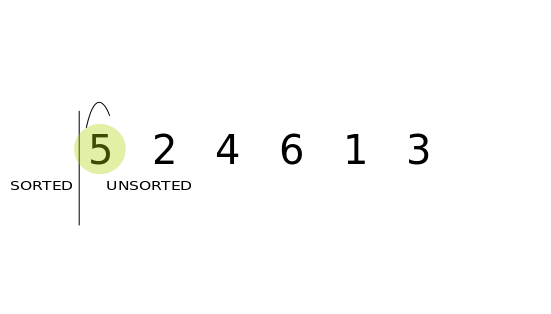
O Bubble Sort é um algoritmo de ordenação que pode ser aplicado em Arrays e Listas dinâmicas.

Se o objetivo é ordenar os valores em forma decrescente, então, a posição atual é comparada com a próxima posição e, se a posição atual for maior que a posição posterior, é realizada a troca dos valores nessa posição. Caso contrário, não é realizada a troca, apenas passa-se para o próximo par de comparações.

Se o objetivo é ordenar os valores em forma crescente, então, a posição atual é comparada com a próxima posição e, se a posição atual for menor que a posição posterior, é realizada a troca. Caso contrário, a troca não é feita e passa-se para o próximo par de comparação.

**Insertion Sort**

O Insert Sort tem como objetivo ordenar os itens inserindo-os na posição correspondente da lista. O Insertion Sort aplica várias vezes a inserção ordenada para ordenar uma sequência.



- O Insertion Sort nada mais é do que a execução do algoritmo de inserção ordenada repetidas vezes.

- O Insertion Sort é in-place, estável e O(n2)O(n2).

- O pior caso da execução deste algoritmo manifesta-se quando a entrada está ordenada em ordem decrescente.

* + No melhor caso o Insertion Sort é O(n)O(n). Isso ocorre quando o array já está ordenado.

- Na teoria, Insertion Sort, Selection Sort e Bubble Sort estão na mesma classe de complexidade, qual seja O(n2) O(n2). Na prática, o Insertion Sort apresenta o melhor desempenho entre esses 3 algoritmos.

**Merge Sort**

Merge Sort é um algoritmo eficiente de ordenação por divisão e conquista. Se nossa missão é ordenar um array comparando seus elementos, do ponto de vista assintótico, n∗logn é o nosso limite inferior. Ou seja, nenhum algoritmo de ordenação por comparação é mais veloz do que n∗logn. Formalmente, todos são Ω(n∗logn).

No caso do Merge Sort, uma característica importante é que sua eficiência é n∗logn para o melhor, pior e para o caso médio. Ou seja, ele não é somente Ω(n∗logn), mas é Θ(n∗logn). Isso nos dá uma garantia de que, independentemente da disposição dos dados em um array, a ordenação será eficiente.

O funcionamento do Merge Sort baseia-se em uma rotina fundamental cujo nome é merge. Primeiro vamos entender como ele funciona e depois vamos ver como sucessivas execuções de merge ordena um array.

- Merge Sort é um algoritmo eficiente de ordenação.

- Independente do caso (melhor, pior ou médio) o Merge Sort sempre será n∗logn. Isso ocorre porque a divisão do problema sempre gera dois subproblemas com a metade do tamanho do problema original (2∗T(n/2)).

- O algoritmo baseia a ordenação em sucessivas execuções de merge, uma rotina que une duas partes ordenadas de um array em uma outra também ordenada.

- O algoritmo de Merge é O(n).

- Apesar de estar na mesma classe de complexidade do Quick Sort, o Merge Sorte tende a ser, na prática, um pouco menos eficiente do que o Quick Sort, pois suas constantes são maiores. Contudo, a seu favor, o Merge Sort garante n∗logn para qualquer caso, enquanto o Quick Sort pode ter ordenação n2 no pior caso, embora raro.

- O Merge Sort não é in-place.

- O Merge Sort é estável.

**Quick Sort**

Quick Sort é um algoritmo eficiente de ordenação por divisão e conquista. Apesar de ser da mesma classe de complexidade do Merge Sort e do  Heap Sort, o Quick Sort é na prática o mais veloz deles, pois suas constantes são menores. Contudo, é importante destacar de antemão que, em seu pior caso, o Quick Sort é O(n2)O(n2), enquanto que o Merge Sort e o Heap Sort garantem n∗lognn∗log⁡n para todos os casos. A boa notícia é que há estratégias simples com as quais podemos minimizar as chances de ocorrência do pior caso.

O funcionamento do Quick Sort baseia-se em uma rotina fundamental cujo nome é ***particionamento***. Particionar significa escolher um número qualquer presente no array, chamado de ***pivot***, e colocá-lo em uma posição tal que todos os elementos à esquerda são menores ou iguais e todos os elementos à direita são maiores.

- Quick Sort é um algoritmo eficiente de ordenação.

- O algoritmo baseia a ordenação em sucessivas execuções de particionamento, uma rotina que escolhe um pivot e o posiciona no array de uma maneira em que os elementos menores ou iguais ao pivot estão à sua esquerda e os maiores estão à sua direita.

- O algoritmo de particionamento é O(n).

- Há dois algoritmos populares de particionamento: o de Lomuto e o de Hoare.

- O particionamento Hoare, embora mais complexo, é na prática mais eficiente que o de Lomuto.

- No caso médio e no melhor caso, o Quick Sort é O(n∗logn).

- No pior caso, o Quick Sort é O(n2).

- O pior caso do Quick Sort é raro e é causado por sucessivas péssimas escolhas de pivot quando o array já está ordenado. Para remediar este problema, há estratégias para escolher melhor o elemento que será o pivot do particionamento, entre elas:

Escolher o pivot aleatoriamente.

Escolher o pivot como sendo a mediana entre o primeiro, o elemento central e o último elemento do array.

- O caso médio é muito mais provável do que o pior e o melhor caso.

- Apesar de estar na mesma classe de complexidade do Merge Sort e do Heap Sort, há experimentos que demonstram que o Quick Sort em seu melhor caso e caso médio é por volta de 3x mais eficiente que o Merge Sort, porque ele contém constantes menores.

- O Quick Sort não é estável.

- O Quick Sort é in-place.

**Counting Sort**

A classificação por contagem funciona iterando através da entrada, contando o número de vezes que cada item ocorre e usando essas contagens para calcular o índice de um item na matriz final classificada .

**Bibliografia:**   
<https://www.freecodecamp.org/news/sorting-algorithms-explained-with-examples-in-python-java-and-c/>

<https://joaoarthurbm.github.io/eda/posts/selection-sort/>

<https://www.embarcados.com.br/algoritmos-de-ordenacao-bubble-sort/>

https://joaoarthurbm.github.io/eda/posts/insertion-sort/

**https://www.interviewcake.com/concept/java/counting-sort**