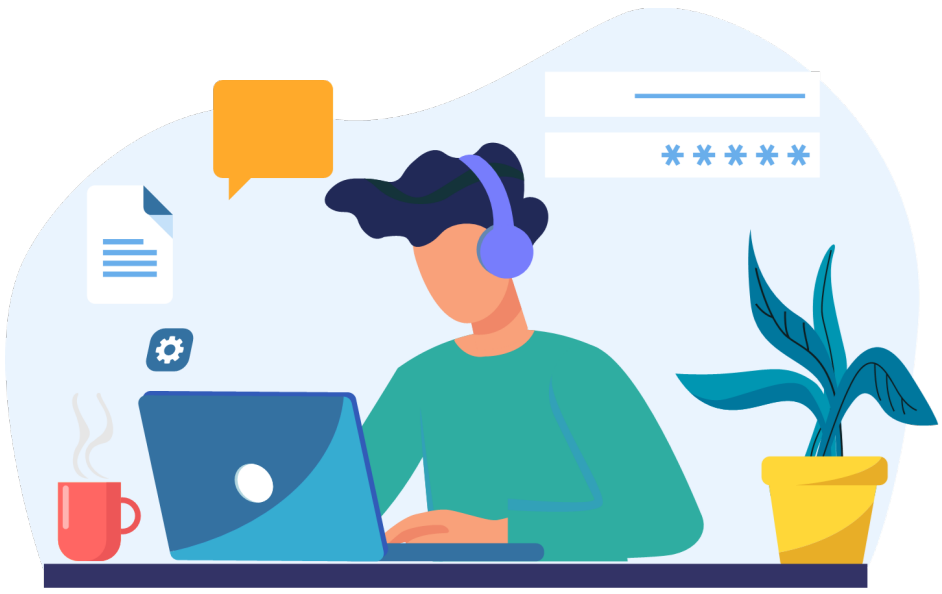
Algoritmos de ordenação

Você sabia que seu material didático é interativo e multimídia? Isso significa que você pode interagir com o conteúdo de diversas formas, a qualquer hora e lugar. Na versão impressa, porém, alguns conteúdos interativos ficam desabilitados. Por essa razão, fique atento: sempre que possível, opte pela versão digital. Bons estudos!

Os algoritmos de ordenação determinam uma forma de organizar os dados em uma determinada sequência. Em geral, pode-se ordenar os elementos nas ordens numérica ou alfabética. Existem diversas aplicações do dia a dia; por exemplo, os objetos podem ser ordenados por tamanho, altura, peso, cor, nomes em ordem alfabética, entre outras opções.

Como uma aplicação do cotidiano, imagine que você vai utilizar algum sistema de compra on-line e deseja listar os itens por ordem de preço (do mais barato para o mais caro). Essa é uma simples aplicação de ordenação utilizada na rotina diária.



Fonte: Shutterstock.

Nesta webaula, vamos ver como ocorre o funcionamento dos algoritmos de ordenação por meio de animações.

Classificação dos algoritmos

Antes de exploramos o funcionamento dos algoritmos de ordenação, é importante conhecer um pouco sobre a classificação deles. Os algoritmos de ordenação podem ser classificados quanto à complexidade da seguinte forma:

Complexidade O(N2)

Nesse grupo estão os algoritmos *selection sort*, *bubble sort* e *insertion sort*. Esses algoritmos são lentos para ordenação de grandes listas, porém são mais intuitivos de entender e possuem codificação mais simples.

Complexidade O(N log N)

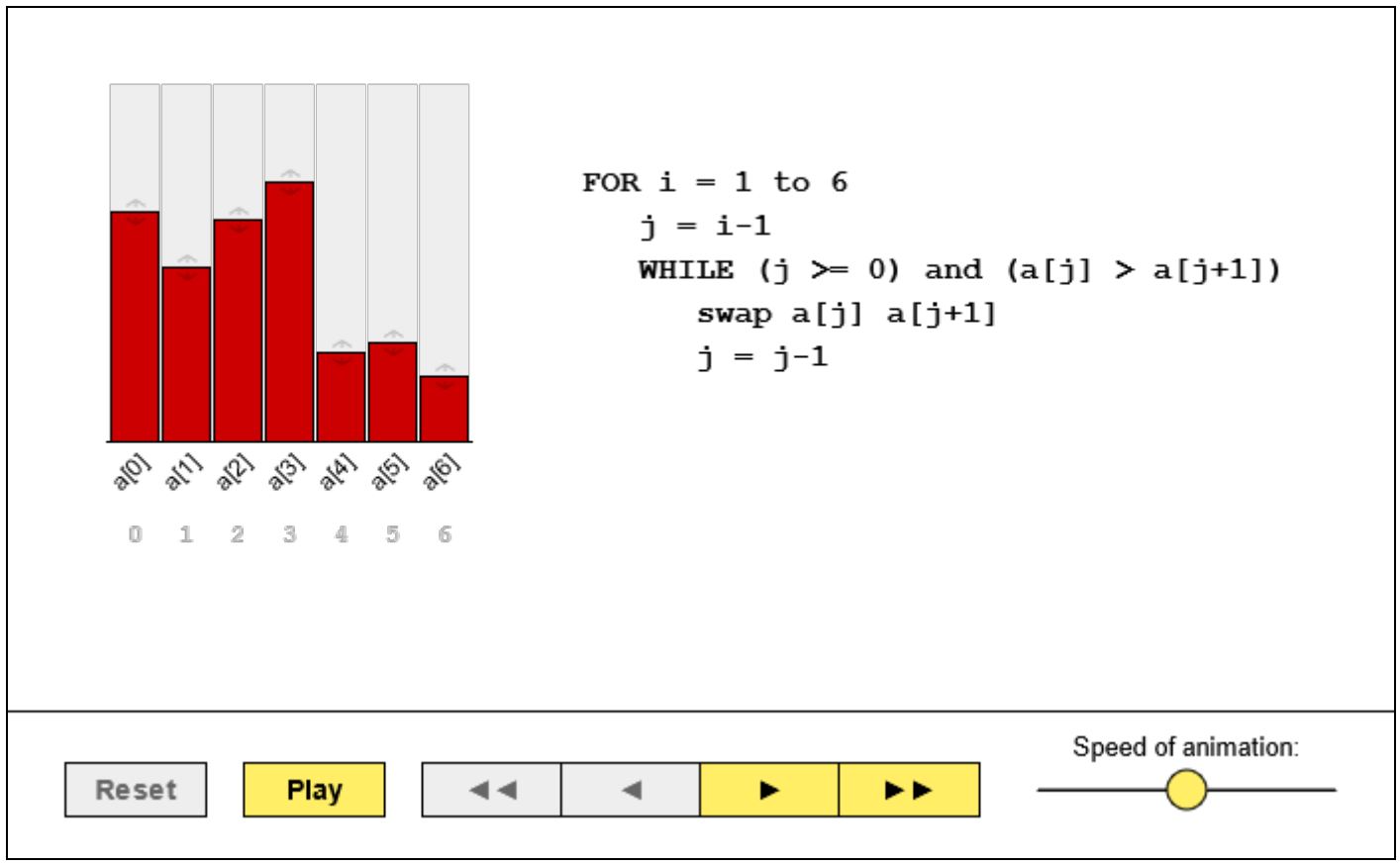
Deste grupo, vamos estudar os algoritmos *merge sort* e *quick sort*. Tais algoritmos possuem performance superior, porém são um pouco mais complexos de serem implementados.

Algoritmo de Ordenação por inserção (*Insertion Sort*)

O algoritmo de ordenação por inserção é iniciado a partir do segundo valor no vetor, pois o primeiro elemento será uma referência para a ordenação. Ou seja, o segundo elemento do vetor será comparado com o primeiro.

O vetor é percorrido da esquerda para a direita. Nesse caminho, compara-se sempre o elemento da direita com os elementos à esquerda de modo que os elementos mais à esquerda sejam organizados e ordenados. O algoritmo de ordenação por inserção tem funcionamento similar ao das pessoas que ordenam cartas em um jogo de baralho.

A animação a seguir mostra o funcionamento do algoritmo de ordenação por inserção.



Fonte: https://anim.ide.sk/sorting_algorithms_1.php

A seguir podemos verificar e testar o algoritmo no emulador implementado em Python.

Algoritmo de ordenação por seleção (*selection sort*)

Seleção pelo valor mínimo

O princípio de funcionamento deste algoritmo é transferir o menor valor do vetor para a primeira posição e, em seguida, o segundo menor valor para a segunda posição, e assim sucessivamente, para os n-1 elementos até os últimos dois elementos.

A animação a seguir mostra o funcionamento do algoritmo de ordenação por seleção.

Fonte: http://anim.ide.sk/sorting_algorithms_1.php

Agora podemos verificar e testar o algoritmo em Python no emulador. O pseudocódigo apresentado na animação foi implementado em Python considerando um vetor de n-1 elementos.

Algoritmo de ordenação por bolha (*bubble sort*)

O algoritmo de ordenação por bolha (ou flutuação) consiste naquele baseado em comparações, em que se percorre vetor ou lista múltiplas vezes. A cada passagem, os pares de elementos adjacentes do vetor são comparados e, depois disso, são trocados se não estiverem ordenados. Esse processo segue para todos os elementos não ordenados da lista.

É importante destacar que esse tipo de algoritmo não é indicado para uma grande quantidade de dados.

A animação a seguir mostra o funcionamento do algoritmo de ordenação por bolha (flutuação).

Fonte: https://anim.ide.sk/sorting_algorithms_1.php

Agora podemos verificar e testar o algoritmo em Python no emulador.

Algoritmo de ordenação *merge sort*

O algoritmo de ordenação *merge sort* é baseado no princípio dividir para conquistar. Basicamente, o vetor é dividido em duas metades e, em seguida, novamente é dividido novamente em outras duas partes – ao repetir a divisão do vetor, em um determinado momento teremos vários vetores com apenas um elemento.

Cada metade é ordenada recursivamente, bem como o são as partes com um elemento. Assim, realizamos a junção (*merge*) dos subvetores ordenados, iniciando a junção de dois a dois de todos os vetores de um elemento e gerando vetores de tamanho dois. Depois disso, o processo é repetido para vetores de tamanho três, quatro, e assim sucessivamente.

A animação a seguir mostra o funcionamento deste algoritmo.

Fonte: https://anim.ide.sk/sorting_algorithms_1.php

Agora você pode testar o algoritmo no emulador.

Algoritmo de ordenação *quick sort*

No algoritmos *quick sort*, primeiramente escolhemos o pivô, que corresponde ao elemento do meio do vetor. Depois, nós trocamos os elementos no início do vetor com os elementos do final – até no início do vetor serão apenas esses elementos, os quais são menores ou iguais ao pivô; e no final do vetor, este permanecerá apenas com esses elementos, que são menores ou iguais ao pivô. Por fim, se no início ou no fim há mais que um elemento, repetimos todo o processo para essas partes (elementos).

Veja como fica o funcionamento deste algoritmo.

Fonte: http://anim.ide.sk/sorting_algorithms_2.php

A seguir podemos verificar e testar o algoritmo no emulador implementado em Python.

Nesta webaula você conheceu os algoritmos de ordenação, os quais já pode testar na prática. Além de ser útil trabalhar com dados ordenados, estudar essa classe de algoritmos permite explorar várias técnicas de programação, tais como a recursão e o problema de dividir para conquistar. Com esse conhecimento, podemos arriscar dizer que você, desenvolvedor, passa de um nível de conhecimento mais "usuário" para um nível mais técnico, mais profundo, o que lhe capacita a entender vários mecanismos que estão por trás de funções prontas que encontramos no dia a dia. Continue seus estudos e boa sorte.

Pesquise mais

No Capítulo 5 (Projeto e análise de algoritmos) do seguinte livro, você encontrará uma seção que fala sobre algoritmos de divisão e conquista. Faça a leitura das páginas 160 a 168.

VIEIRA, L. V. **Complexidade de algoritmos**: análise, projeto e métodos. 3. ed. Porto Alegre: Bookman, 2012.

Fonte: Shutterstock.

Para visualizar o vídeo, acesse seu material digital.