Ordenação por seleção

Neste capítulo

- Você conhecerá arrays e listas encadeadas – dois tipos de estrutura básica. Eles estão por todo lugar. Você já teve acesso aos arrays no capítulo 1 e continuará se utilizando deles por praticamente todos os capítulos. Arrays são fundamentais, então preste atenção! Porém algumas vezes é melhor usar a lista encadeada em vez do array. Este capítulo explana os prós e os contras de ambas as estruturas para que possa decidir qual é a ideal para o seu algoritmo.
- Você aprenderá a fazer o seu primeiro algoritmo de ordenação. Muitos algoritmos só funcionam se os dados estiverem ordenados. Lembra-se da pesquisa binária? Você

só pode executá-la se os elementos de sua lista estiverem ordenados. Este capítulo lhe apresentará a ordenação por seleção. A maioria das linguagens de programação contém nativamente os algoritmos de seleção, então raramente terá de escrever a sua própria versão a partir do zero. No entanto a ordenação por seleção é um trampolim para o quicksort, que abordarei no próximo capítulo. O quicksort é um algoritmo importante e será compreendido mais facilmente se você já conhecer algum tipo de algoritmo de ordenação.

O que você precisa saber

Para entender a análise de desempenho deste capítulo, você precisa conhecer a notação Big O e logaritmos. Se não conhece, sugiro que leia o Capítulo 1. A notação Big O é utilizada em todo este livro.

Como funciona a memória

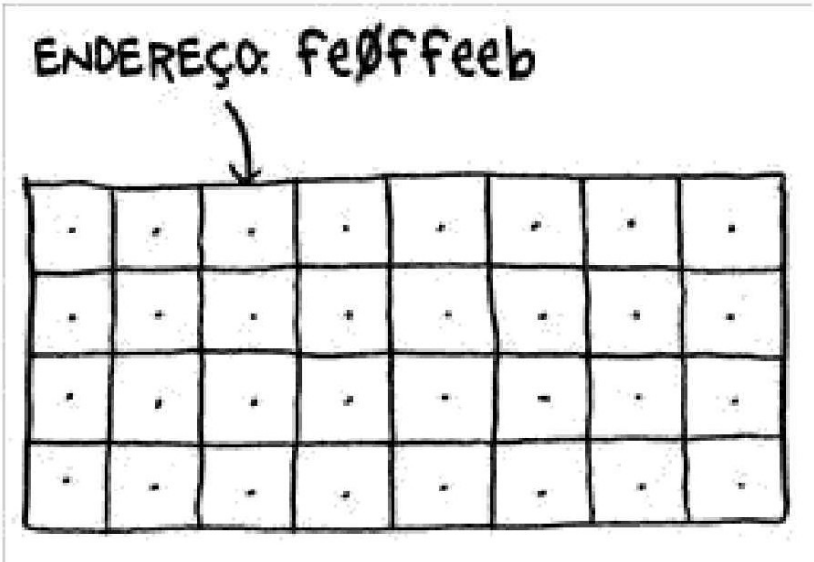
Imagine que você vai a um show e precisa guardar as suas coisas na chapelaria. Algumas gavetas estão disponíveis.

Cada gaveta pode guardar um elemento. Você deseja guardar duas

coisas, então pede duas gavetas.

Você guardou as suas duas coisas aqui.

Você está pronto para o show! É mais ou menos assim que a memória do seu computador funciona. O computador se parece com um grande conjunto de gavetas, e cada gaveta tem seu endereço.



feØffeeb é o endereço de um slot na memória.

Cada vez que quer armazenar um item na memória, você pede ao computador um pouco de espaço e ele te dá um endereço no qual você pode armazenar o seu item. Se quiser armazenar múltiplos itens, existem duas maneiras para fazer isso: arrays e listas. Falarei sobre arrays e listas depois, bem como sobre os prós e contras de cada um. Não existe apenas uma maneira correta para armazenar itens em cada um dos casos, então é importante saber as diferenças.

Arrays e listas encadeadas

Algumas vezes, você precisa armazenar uma lista de elementos na memória. Suponha que você esteja escrevendo um aplicativo para gerenciar os seus afazeres. É necessário armazenar os seus afazeres como uma lista na memória.

Você deve usar um array ou uma lista encadeada? Vamos armazenar

os afazeres primeiro em um array, pois assim a compreensão fica mais fácil. Usar um array significa que todas as suas tarefas estão armazenadas contiguamente (uma ao lado da outra) na memória.

Agora, suponha que você queira adicionar mais uma tarefa. No entanto a próxima gaveta está ocupada por coisas de outra pessoa!

É como se você estivesse indo ao cinema com os seus amigos e encontrasse um lugar para sentar, mas outro amigo se juntasse a vocês e não houvesse lugar para ele. Vocês todos precisariam se mover e encontrar um lugar onde todos coubessem. Neste caso, você precisaria solicitar ao computador uma área de memória em que coubessem todas as suas tarefas. Então você as moveria para lá.

Se outro amigo aparecesse, vocês ficariam sem lugar novamente – e todos precisariam se mover uma segunda vez! Que incômodo. Da mesma forma, adicionar novos itens a um array será muito lento. Uma maneira fácil de resolver isso é “reservando lugares”: mesmo

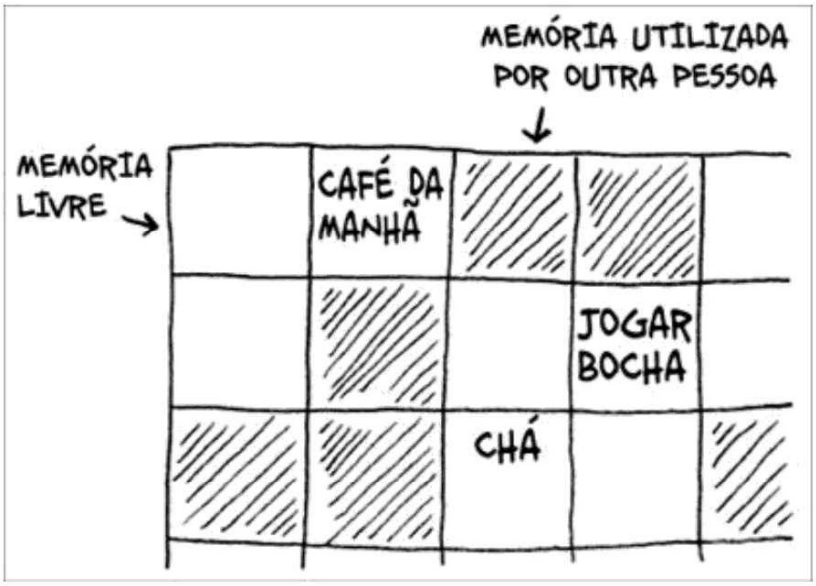
que você tenha três itens na sua lista de tarefas, você pode solicitar ao computador dez espaços, só por via das dúvidas. Então, você pode adicionar dez itens a sua lista sem precisar mover nada. Isto é uma boa maneira de contornar o problema, mas você precisa ficar atento às desvantagens:

- Você pode não precisar dos espaços extras que reservou; então a memória será desperdiçada. Você não está utilizando a memória, mas ninguém mais pode usá-la também.
- Você pode precisar adicionar mais de dez itens a sua lista de tarefas, então você terá de mover seus itens de qualquer maneira.

Embora seja uma boa forma de contornar o problema, não é uma solução perfeita. Listas encadeadas resolvem este problema de adição de itens.

Listas encadeadas

Com as listas encadeadas, seus itens podem estar em qualquer lugar da memória.



Cada item armazena o endereço do próximo item da lista. Um monte de endereços aleatórios de memória estão ligados.

Endereços de memória ligados.

É como uma caça ao tesouro. Você vai ao primeiro endereço e ele diz "o próximo item pode ser encontrado no endereço 123". Então vai ao endereço 123 e ele diz "O próximo item pode ser encontrado no endereço 847", e assim por diante. Adicionar um item a uma lista encadeada é fácil: você o coloca em qualquer lugar da memória e armazena o endereço do item anterior.

Com as listas encadeadas você nunca precisa mover os seus itens; também evita outro problema. Digamos que você vá a um cinema famoso com os seus amigos. Vocês seis estão tentando procurar um lugar para sentar, mas o cinema está cheio. Não há seis lugares juntos. Bem, algumas vezes isso acontece com arrays. Imagine que está tentando encontrar 10.000 slots para um array. Sua memória tem 10.000 slots, mas eles não estão juntos. Você não consegue arrumar um lugar para o seu array! Usar uma lista encadeada seria como dizer "vamos nos dividir e assistir ao filme". Se existir espaço na memória, você terá espaço para a sua lista encadeada.

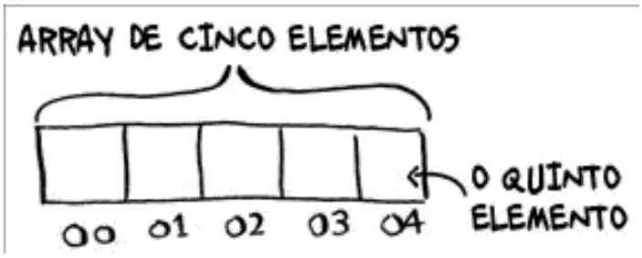
Se as listas encadeadas são muito melhores para inserções, para que servem os arrays?

Arrays

Os websites que apresentam listas "top 10" usam uma tática trapaceira para conseguir mais visualizações. Em vez de mostrarem a lista em uma única página, eles colocam um item em cada página e fazem você clicar em "próximo" para ler o item seguinte. Por exemplo, "Os 10 melhores vilões da TV" não estarão listados em uma única página, em vez disso, você começará pelo #10 (Newman) e seguirá clicando em "próximo" até chegar em #1 (Gustavo Fring). Esta técnica fornece aos sites dez páginas inteiras para incluir anúncios, mas fica chato ficar clicando em "próximo" nove vezes até chegar ao número 1. Seria muito melhor se a lista estivesse em uma única página e você pudesse clicar no nome de cada vilão para saber mais.

Listas encadeadas têm um problema similar. Suponha que você queira ler o último item de uma lista encadeada. Você não pode fazer isso porque não sabe o endereço dele. Em vez disso, precisa ir ao item #1 para pegar o endereço do item #2. Então, é necessário ir ao item #2 para encontrar o endereço do item #3, e assim por diante, até conseguir o endereço do último item. Listas encadeadas são ótimas se você quiser ler todos os itens, um de cada vez: você pode ler um item, seguir para o endereço do próximo item e fazer isso até o fim da lista. Mas se você quiser pular de um item para outro, as listas encadeadas são terríveis.

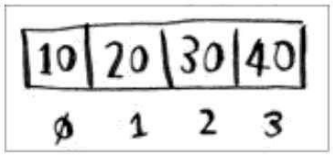
Com arrays é diferente. Você sabe o endereço de cada item. Por exemplo, suponha que seu array tenha cinco itens e que você saiba que o primeiro está no endereço 00. Qual é o endereço do item #5?



A matemática lhe dá a resposta: está no endereço 04. Arrays são ótimos se você deseja ler elementos aleatórios, pois pode encontrar qualquer elemento instantaneamente em um array. Na lista encadeada, os elementos não estão próximos uns dos outros, então você não pode calcular instantaneamente a posição de um elemento na memória – precisa ir ao primeiro elemento para encontrar o endereço do segundo, então ir ao segundo elemento para encontrar o endereço do terceiro e seguir fazendo isso até chegar ao elemento que deseja.

Terminologia

Os elementos em um array são numerados. Essa numeração começa no 0, não no 1. Neste array, por exemplo, o número 20 está na posição 1.



O número 10 está na posição 0. Isso geralmente confunde novos programadores. Começar no 0 simplifica todos os tipos de array na programação, logo, os programadores não podem fugir disso. Quase todas as linguagens de programação começarão os arrays numerando o primeiro elemento como 0. Logo você se acostuma!

A posição de um elemento é chamada de *índice*. Portanto, em vez de dizer "o número 20 está na *posição* 1", a terminologia correta seria dizer "o número 20 está no *índice* 1". Usarei índice para falar de *posição* neste livro.

Aqui está o tempo de execução para operações comuns de arrays e listas.

Pergunta: Por que é necessário tempo de execução O(n) para inserir um elemento em um array? Suponha que você queira inserir um elemento no começo de um array. Como faria isso? Quanto tempo levaria? Encontre as respostas no final desta seção!

EXERCÍCIOS

2.1 Suponha que você esteja criando um aplicativo para acompanhar as suas finanças.

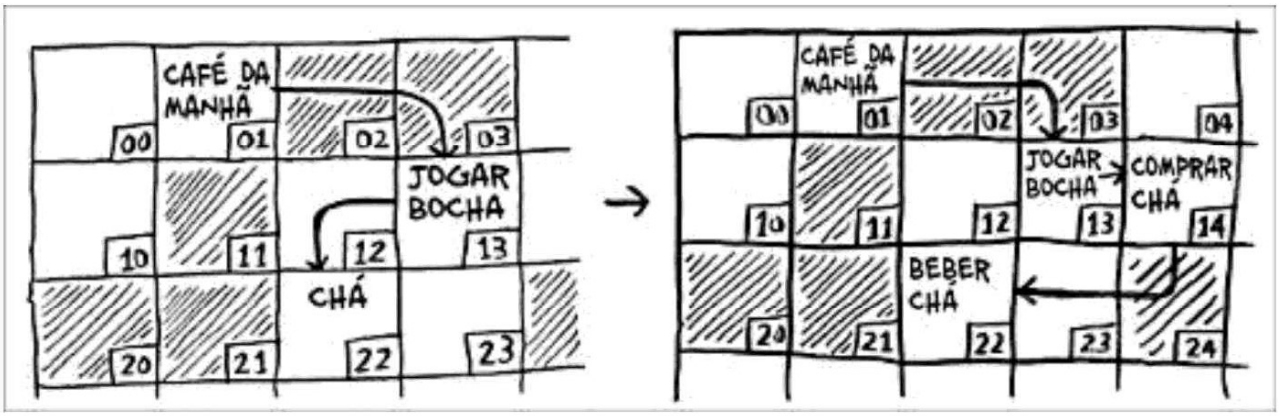
Todos os dias você anotará tudo o que gastou e onde gastou. No final do mês, você deverá revisar os seus gastos e resumir o quanto gastou. Logo, você terá um monte de inserções e poucas leituras. Você deverá usar um array ou uma lista para implementar este aplicativo?

Inserindo algo no meio da lista

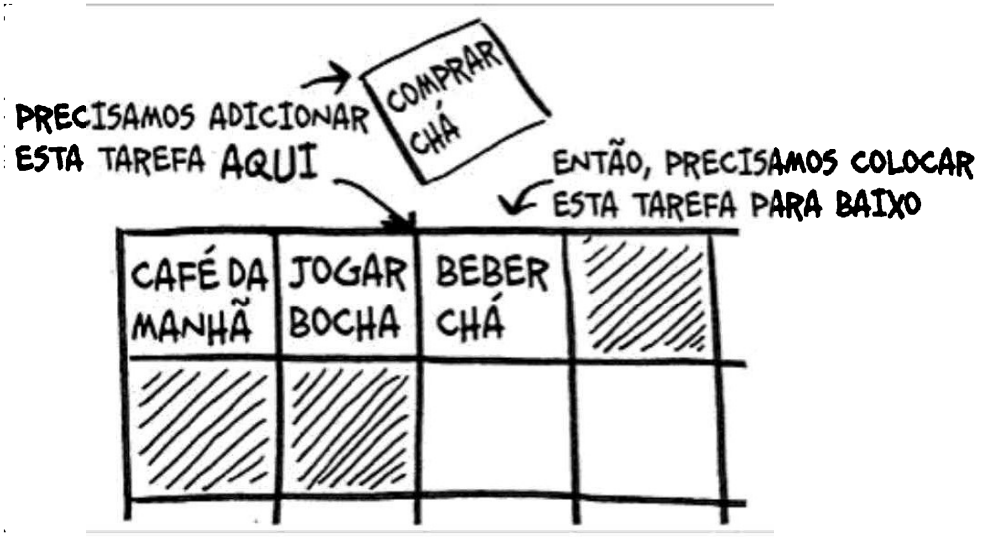
Imagine que você queira que a sua lista de tarefas se pareça mais com um calendário. Antes, você adicionava os itens ao final da lista. Agora, quer adicionar suas tarefas na ordem em que elas devem ser realizadas.

Lista desordenada.

O que seria melhor se você quisesse inserir elementos no meio de uma lista: arrays ou listas encadeadas? Usando listas encadeadas, basta mudar o endereço para o qual o elemento anterior está apontando.



Já para arrays, você deve mover todos os itens que estão abaixo do endereço de inserção.



Se não houver espaço, pode ser necessário mover tudo para um novo local! Por isso, listas encadeadas são melhores caso você queira inserir um elemento no meio de uma lista.

Deleções

E se você quiser deletar um elemento? Novamente, é mais fácil fazer isso usando listas encadeadas, pois é necessário mudar apenas o endereço para o qual o elemento anterior está apontando. Com arrays, tudo precisa ser movido quando um elemento é eliminado.

Ao contrário do que ocorre com as inserções, a eliminação de elementos sempre funcionará. A inserção poderá falhar quando não houver espaço suficiente na memória.

Aqui estão os tempos de execução para as operações mais comuns em arrays e listas encadeadas.

	ARRAYS	LISTAS
LEITURA	O(1)	O(n)
INSERÇÃO	O(n)	O(1)
ELIMINAÇÃO	O(n)	O(1)

Vale a pena mencionar que inserções e eliminações terão tempo de execução O(1) somente se você puder acessar instantaneamente o elemento a ser deletado. É uma prática comum acompanhar o primeiro e o último item de uma lista encadeada para que o tempo de execução para deletá-los seja O(1).

O que é mais usado: arrays ou listas? Obviamente, isso depende do caso em que se aplicam. Entretanto os arrays são mais comuns porque permitem acesso aleatório. Existem dois tipos de acesso: o *aleatório* e o *sequencial*. O sequencial significa ler os elementos, um por um, começando pelo primeiro. Listas encadeadas *só* podem lidar com acesso sequencial. Se você quiser ler o décimo elemento de uma lista encadeada, primeiro precisará ler os nove elementos anteriores

para chegar ao endereço do décimo elemento. O aleatório permite que você pule direto para o décimo elemento. Muitos casos requerem o acesso aleatório, o que faz os arrays serem mais utilizados. Arrays e listas são usados para implementar outras estruturas de dados (isso será explicado mais adiante).

EXERCÍCIOS

2.2 Suponha que você esteja criando um aplicativo para anotar os pedidos dos clientes em um restaurante. Seu aplicativo precisa de uma lista de pedidos. Os garçons adicionam os pedidos a essa lista e os chefes retiram os pedidos da lista. Funciona como uma fila. Os garçons colocam os pedidos no final da fila e os chefes retiram os pedidos do começo dela para cozinhá-los.

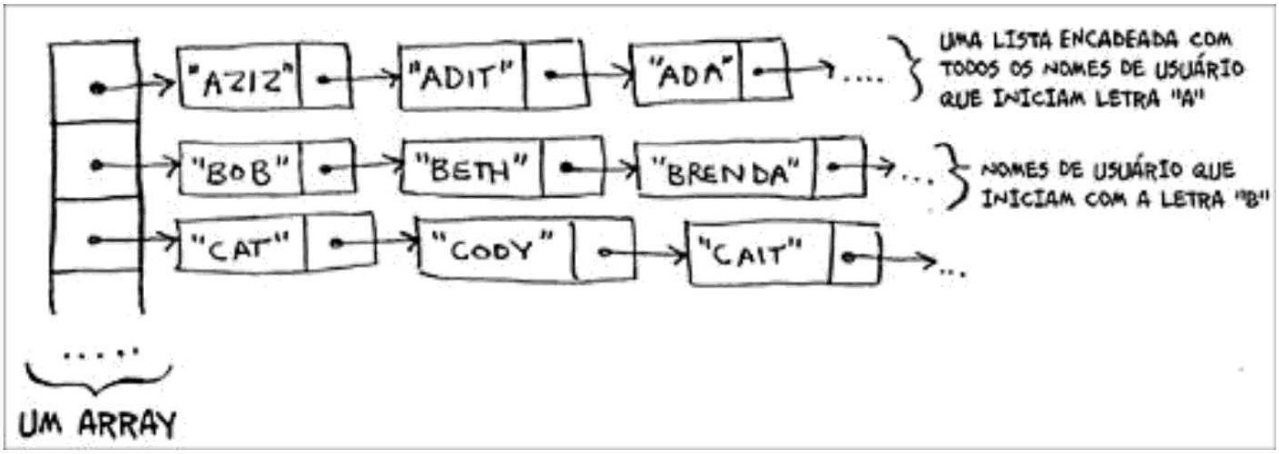
Você usaria um array ou uma lista encadeada para implementar essa lista? (Dica: listas encadeadas são boas para inserções/eliminações e arrays são bons para acesso aleatório. O que fazer neste caso?)

2.3 Vamos analisar um experimento. Imagine que o Facebook guarda uma lista de usuários. Quando alguém tenta acessar o Facebook, uma busca é feita pelo nome de usuário. Se o nome da pessoa está na lista, ela pode continuar o acesso. As pessoas acessam o Facebook com muita frequência, então existem muitas buscas nessa lista. Presuma que o Facebook usa a pesquisa binária para procurar um nome na lista. A pesquisa binária requer acesso aleatório – você precisa ser capaz de acessar o meio da lista de

nomes instantaneamente. Sabendo disso, você implementaria essa lista como um array ou uma lista encadeada?

2.4 As pessoas se inscrevem no Facebook com muita frequência também. Suponha que você decida usar um array para armazenar a lista de usuários. Quais as desvantagens de um array em relação às inserções? Em particular, imagine que você está usando a pesquisa binária para buscar os logins. O que acontece quando você adiciona novos usuários em um array?

2.5 Na verdade, o Facebook não usa nem arrays nem listas encadeadas para armazenar informações. Vamos considerar uma estrutura de dados híbrida: um array de listas encadeadas. Você tem um array com 26 slots. Cada slot aponta para uma lista encadeada. Por exemplo, o primeiro slot do array aponta para uma lista encadeada que contém os usuários que começam com a letra A. O segundo slot aponta para a lista encadeada que contém os usuários que começam com a letra B, e assim por diante.



Suponha que o Adit B se inscreva no Facebook e você queira adicioná-lo à lista. Você vai ao slot 1 do array, a seguir para a lista encadeada do slot 1, e adiciona Adit B no final. Agora, suponha que você queira procurar o Zakhir H. Você vai ao slot 26, que aponta para a lista encadeada de todos os nomes começados em Z. Então, procura pela lista até encontrar o Zakhir H.

Compare esta estrutura híbrida com arrays e listas encadeadas. É mais lento ou mais rápido fazer inserções e eliminações nesse caso? Você não precisa responder dando o tempo de execução

Big(O), apenas diga se a nova estrutura de dados é mais rápida ou mais lenta do que os arrays e as listas encadeadas.

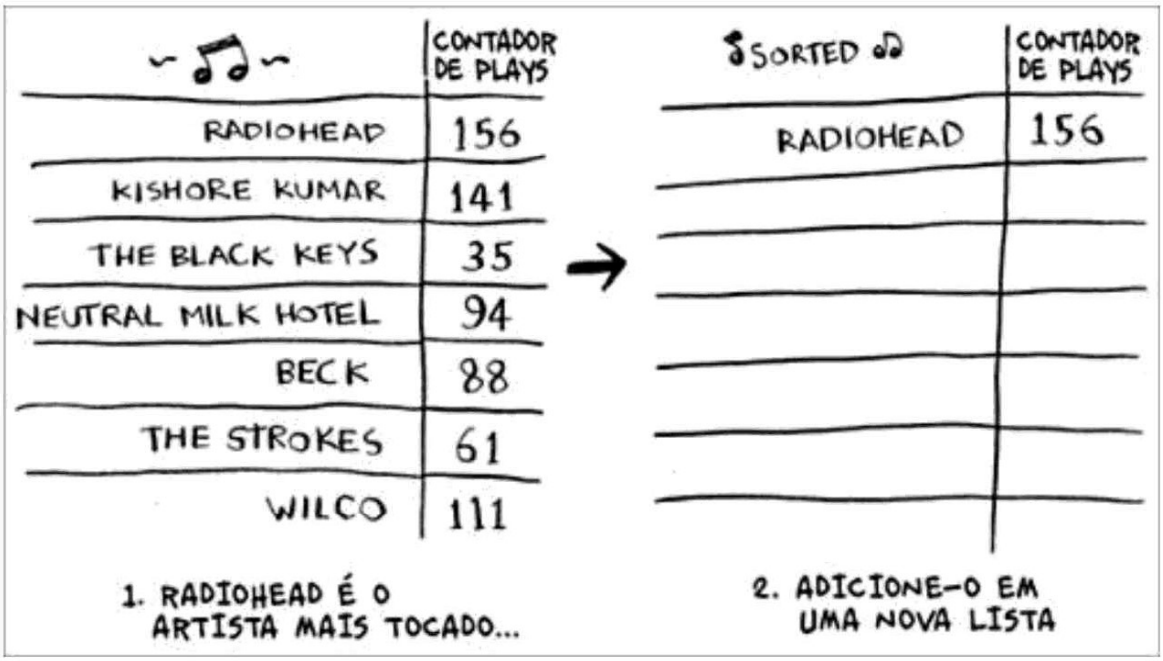
Ordenação por seleção

Vamos juntar tudo aprendido até aqui para você conhecer o seu segundo algoritmo: a ordenação por seleção. Para seguir nesta seção, você precisa ter compreendido arrays e listas, bem com a notação Big O.

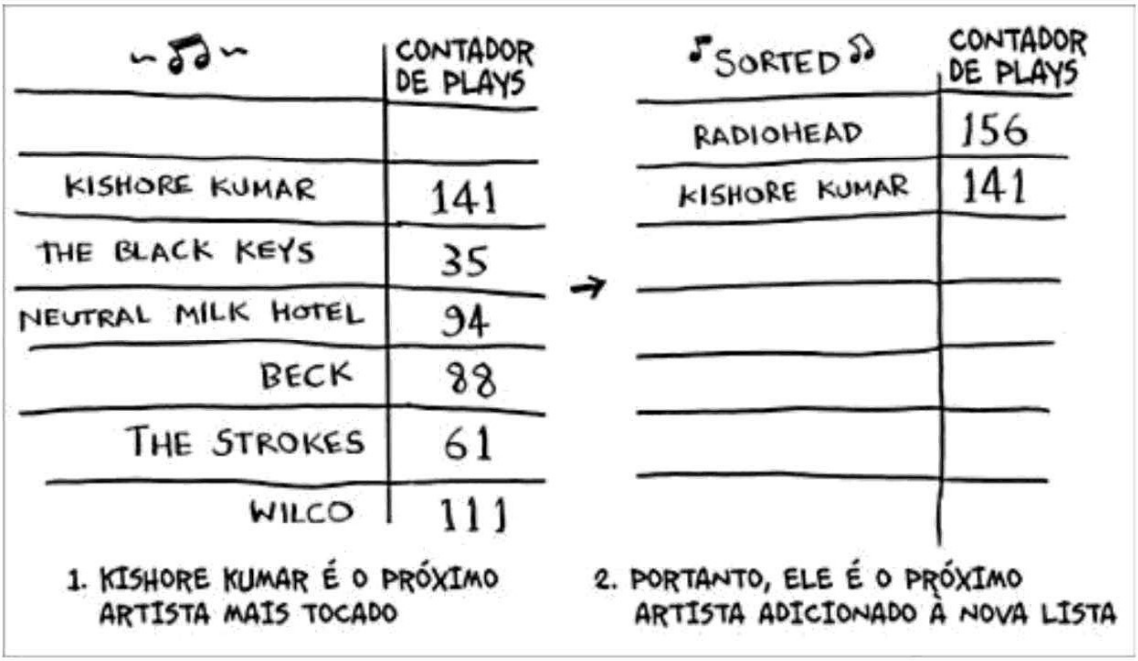
Suponha que você tenha um monte de músicas no seu computador. Para cada artista, você tem um contador de plays.

	CONTADOR DE PLAYS
RADIOHEAD	156
KISHORE KUMAR	141
THE BLACK KEYS	35
NEUTRAL MILK HOTEL	94
BECK	88
THE STROKES	61
WILCO	111

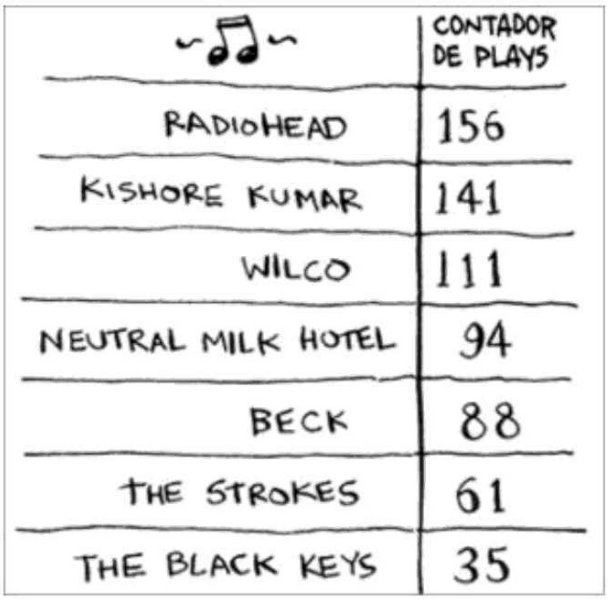
Você quer ordenar uma lista de artistas, do artista mais tocado para o menos tocado, para que possa categorizar os seus artistas favoritos. Como pode fazer isso? Uma maneira seria pegar o artista mais tocado da lista de músicas e adicioná-lo a uma nova lista.



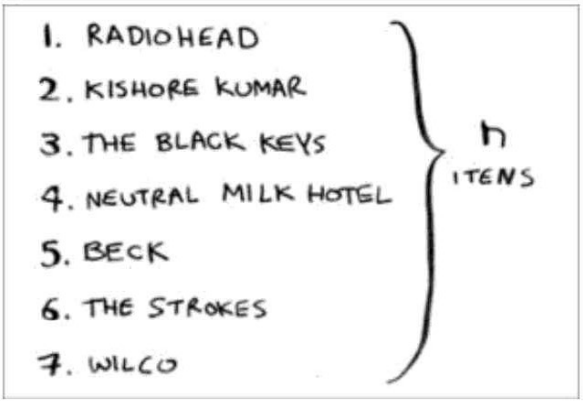
Faça isso de novo para encontrar o próximo artista mais tocado.



Continue fazendo isso e então você terminará com uma lista ordenada.



Vamos pensar como engenheiros da computação e avaliar quanto tempo isso demoraria a ser executado. Lembre-se de que o tempo de execução O(n) significa que você precisa passar por todos os elementos da lista uma vez. Por exemplo, executar uma pesquisa simples na lista de artistas significa olhar para cada artista uma vez.



Para encontrar o artista com o maior número de plays você precisa verificar cada item da lista. Isso tem tempo de execução O(n), como você acabou de ver. Então você tem uma operação com tempo de execução O(n) e precisa repetir essa operação n vezes:

Isso tem tempo de execução $O(n \times n)$ ou $O(n^2)$.

Algoritmos de ordenação são muito úteis. Agora você pode ordenar:

- nomes em uma agenda telefônica
- datas de viagem
- emails (do mais novo ao mais antigo)

Verificando menos elementos a cada vez

Talvez você esteja pensando: conforme passa pelas operações, o número de elementos que precisa analisar diminui. Eventualmente, você acaba tendo de checar apenas um elemento. Então como o tempo de execução permanece sendo $O(n^2)$? Isto é uma boa pergunta, e a resposta tem a ver com a notação Big O. Falarei mais sobre isso no capítulo 4, mas aqui vai o ponto principal.

Você estava certo sobre não precisar verificar n elementos a cada vez.

Você verifica n elementos, então $n - 1$, $n - 2$ … 2, 1. Na média, você verifica uma lista que tem $\frac{1}{2} \times n$ elementos. O tempo de execução é $O(n \times \frac{1}{2} \times n)$. Mas constantes como $\frac{1}{2}$ são ignoradas na notação Big O (novamente, leia o Capítulo 4 para ter acesso à discussão completa), então você escreve apenas $O(n \times n)$ ou $O(n^2)$.

A ordenação por seleção é um algoritmo bom, mas não é muito rápido. O Quicksort é um algoritmo de ordenação mais rápido, que tem tempo de execução de apenas $O(n \log n)$. Falarei dele no capítulo 4!

EXEMPLO DE CÓDIGO

Nós não lhe mostramos o código para ordenar a lista de músicas, mas a seguir estão alguns códigos que farão algo bem similar: ordenar um array do menor para o maior. Vamos escrever uma função para encontrar o menor elemento em um array:

```
def buscaMenor(arr):
    menor = arr[0] ❶
    menor_indice = 0 ❷
    for i in range(1, len(arr)):
        if arr[i] < menor:
            menor = arr[i]
            menor_indice = i
    return menor_indice
```

❶ Armazena o menor valor.

❷ Armazena o índice do menor valor.

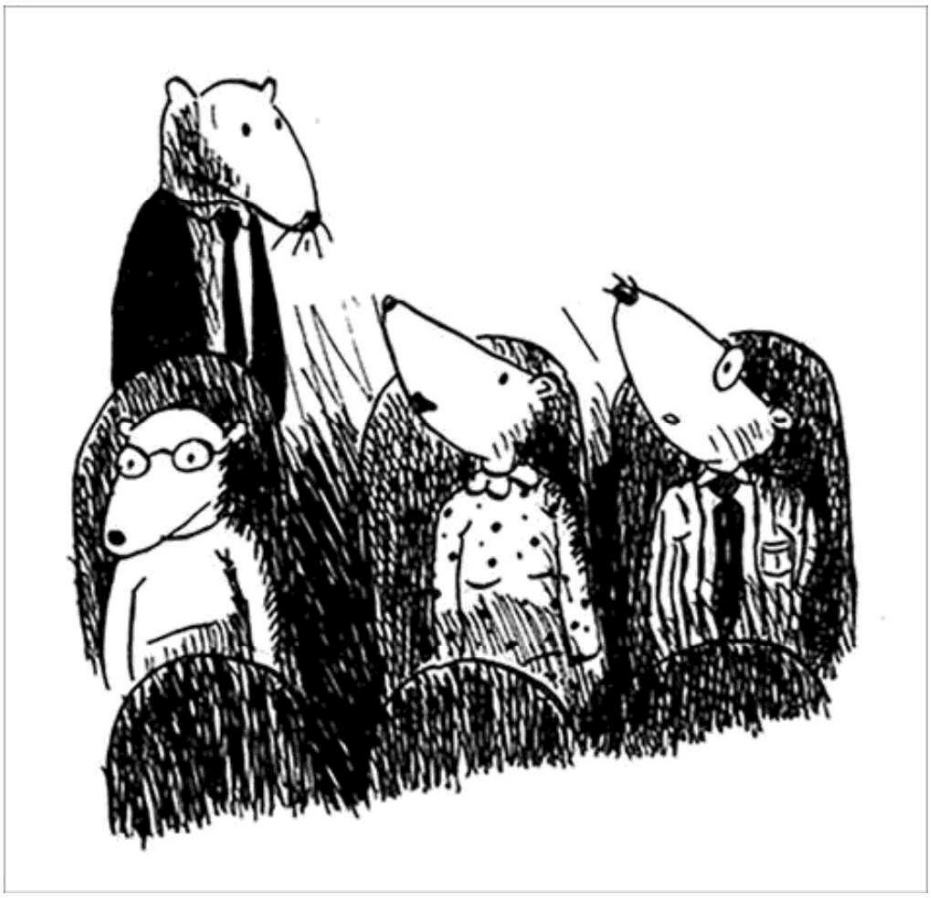
Agora, você pode usar esta função para escrever a ordenação por seleção:

```
def ordenacaoporSelecao(arr): ❶
    novoArr = []
    for i in range(len(arr)):
        menor = buscaMenor(arr) ❷
        novoArr.append(arr.pop(menor))
    return novoArr

print ordenacaoporSelecao([5, 3, 6, 2, 10])
```

❶ Ordenações em um array.

❷ Encontra o menor elemento do array e adiciona ao novo array.



Recapitulando

- A memória do seu computador é como um conjunto gigante de gavetas.
- Quando se quer armazenar múltiplos elementos, usa-se um array ou uma lista.
- No array, todos os elementos são armazenados um ao lado do outro.
- Na lista, os elementos estão espalhados e um elemento armazena o endereço do próximo elemento.
- Arrays permitem leituras rápidas.
- Listas encadeadas permitem rápidas inserções e eliminações.
- Todos os elementos de um array devem ser do mesmo tipo (todos ints, todos doubles, e assim por diante).