第二章：选择排序

- 基本数据结构：数组和链表
- 学习一种排序算法（二分查找只能用于有序元素列表）

2.1 数组和链表

数组意味着所有待储存元素在内存中是相连的，如果没有空间，就得移动到内存其他地方，因此添加新元素的速度会很慢。

数组可以直接知道每个元素的地址，数组的读取速度很快。

数组的元素带编号，编号从 0 开始。元素的位置称为**索引。**

链表中的元素可以存储在内存的任何地方，链表的每个元素都储存了下一个元素的地址，从而使一系列随机的内存地址串在一起。

链表的问题在于：在需要读取链表的最后一个元素的时候，不能直接读取，因为不知道它所处的地址，需要先访问元素#1，获取#2 的地址，以此类推。

数组和列表的操作运行时间：

	数组	链表
读取	O(1) (常量时间)	O(n)
插入	O(n) (线性时间)	O(1)
删除	O(n)	O(1)

✓ 在中间插入：

使用链表时，插入元素更简单，只需要修改它前面那个元素指向的地址。（better choice）

使用数组时，需要将后面的元素都向后移。如果没有足够的空间，可能需要将整个数组复制到其他地方。

✓ 删除元素：

删除元素总能成功。

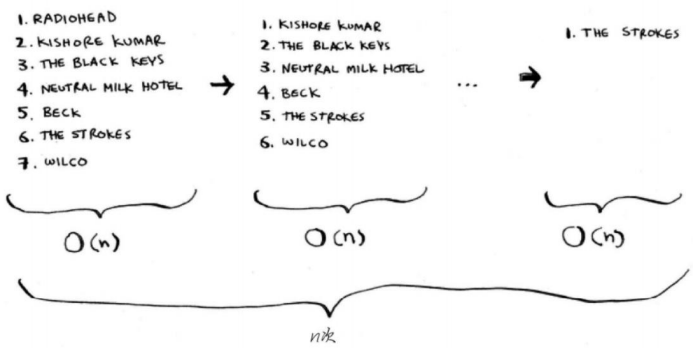
✓ 数组使用相对较多，因为它支持随机访问。

访问方式：随机访问和顺序访问。

链表只能顺序访问。

2.2 选择排序：

要找出播放次数最多的乐队，必须检查列表中的每个元素。正如你刚才看到的，这需要的时间为$O(n)$。因此对于这种时间为$O(n)$的操作，你需要执行n次。



需要的总时间为 $O(n \times n)$，即$O(n^2)$。

需要检查的元素数越来越少

随着排序的进行，每次需要检查的元素数在逐渐减少，最后一次需要检查的元素都只有一个。既然如此，运行时间怎么还是$O(n^2)$呢？这个问题问得好，这与大O表示法中的常数相关。第4章将详细解释，这里只简单地说一说。

你说得没错，并非每次都需要检查n个元素。第一次需要检查n个元素，但随后检查的元素数依次为$n-1, n-2, \cdots, 2$和1。平均每次检查的元素数为$1/2 \times n$，因此运行时间为$O(n \times 1/2 \times n)$。但大O表示法省略诸如1/2这样的常数（有关这方面的完整讨论，请参阅第4章），因此简单地写作$O(n \times n)$或$O(n^2)$。

前面没有列出对乐队进行排序的代码，但下述代码提供了类似的功能：将数组元素按从小到大的顺序排列。先编写一个用于找出数组中最小元素的函数。

```
def findSmallest(arr):
  smallest = arr[0]        <-- 存储最小的值
  smallest_index = 0       <-- 存储最小元素的索引
  for i in range(1, len(arr)):
    if arr[i] < smallest:
      smallest = arr[i]
      smallest_index = i
  return smallest_index
```

现在可以使用这个函数来编写选择排序算法了。

```
def selectionSort(arr):    <-- 对数组进行排序
  newArr = []
  for i in range(len(arr)):
      smallest = findSmallest(arr)   <-- 找出数组中最小的元素，并将其加入到新数组中
      newArr.append(arr.pop(smallest))
return newArr

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