查找、排序与效率

Python Reference

线性查找与二分查找

查找 (search) 用来找到某个值在哪里。线性查找 (linear search) 逐个检查每个元素，所以它对任何列表都有效。

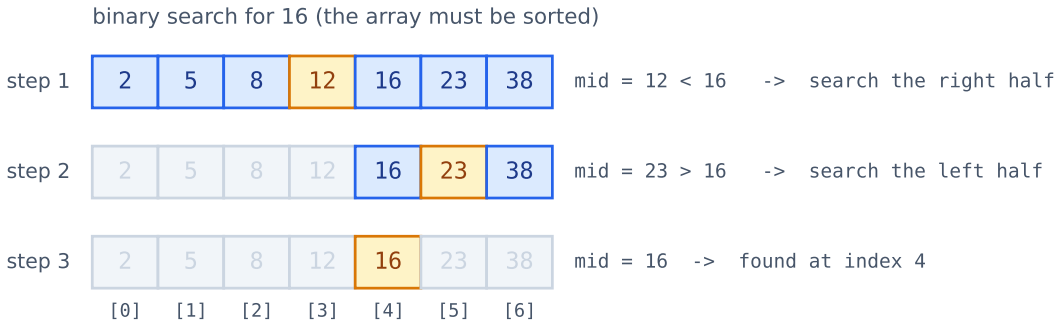
```
def linear_search(items, target):
    for i in range(len(items)):
        if items[i] == target:
            return i
    return -1      # not found

print(linear_search([4, 8, 2, 9], 2))   # 2
```

二分查找 (binary search) 快得多，但需要一个**已排序**的列表。它每一步把范围折半。

```
def binary_search(items, target):
    lo, hi = 0, len(items) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if items[mid] == target:
            return mid
        elif items[mid] < target:
            lo = mid + 1
        else:
            hi = mid - 1
    return -1

print(binary_search([1, 3, 5, 7, 9], 7))   # 3
```



binary search: halve the range each step · O(log n)

二分查找每一步把范围减半 —— 已排序列表上是 $O(\log n)$

排序（冒泡与插入）

排序 (sort) 就是把元素按顺序排好。冒泡排序 (bubble sort) 反复交换 (swap) 顺序不对的相邻元素。

```
def bubble_sort(a):
    a = a[:]                            # work on a copy
    for i in range(len(a)):
        for j in range(len(a) - 1 - i):
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
    return a

print(bubble_sort([5, 2, 4, 1]))      # [1, 2, 4, 5]
```

插入排序 (insertion sort) 一次把一个元素放进已排好的部分, 把每个新元素往回滑到属于它的位置:

```
def insertion_sort(a):
    a = a[:]                                # 在副本上操作
    for i in range(1, len(a)):
        key = a[i]
        j = i - 1
        while j >= 0 and a[j] > key:    # 把较大的值右移
            a[j + 1] = a[j]
            j -= 1
        a[j + 1] = key                  # 把 key 放进空位
    return a

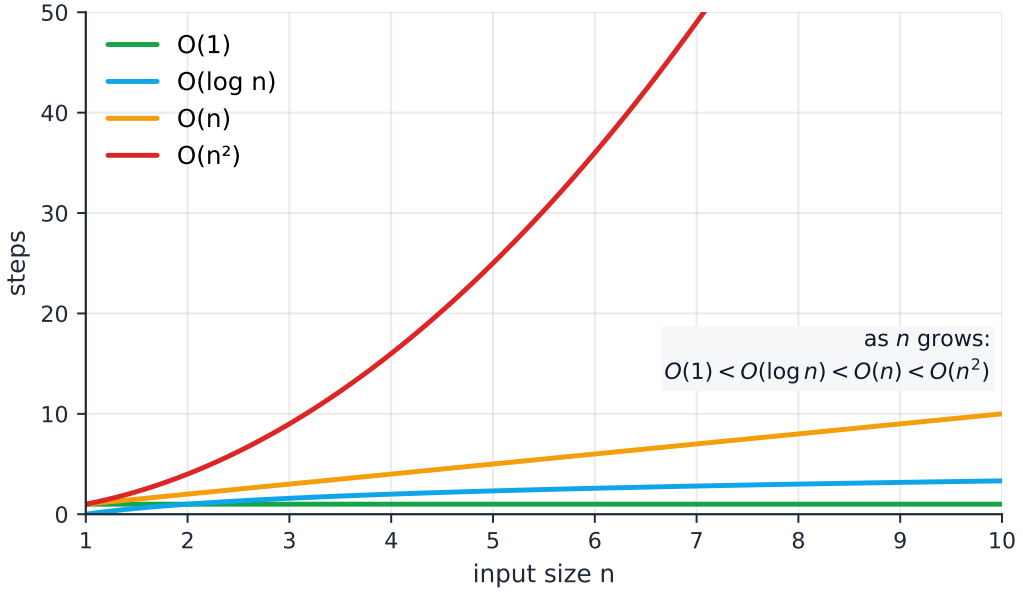
print(insertion_sort([5, 2, 4, 1]))   # [1, 2, 4, 5]
```

- 在真实代码里, 用 Python 内置的 `sorted()`:

```
print(sorted([5, 2, 4, 1]))           # [1, 2, 4, 5]
```

算法效率

效率 (efficiency) 问的是: 当输入变大时, 工作量怎样增长。我们用大 O 记号 (Big-O) 来描述它。



常见复杂度下, 步数随输入规模增长的情况

Big-O	名称	例子
$O(1)$	常数	查找字典的键
$O(\log n)$	对数	二分查找
$O(n)$	线性	线性查找
$O(n^2)$	平方	冒泡排序

```
def steps(n):           # how many steps a linear scan takes
    count = 0
    for i in range(n):
        count = count + 1
    return count

print(steps(100))       # 100  -> O(n)
```

随机性与模拟

`random` 模块产生随机数。用一个种子 (seed) 让结果可重复。模拟 (simulation) 运行很多次随机试验来估计一个答案。

```
import random
random.seed(0)
rolls = [random.randint(1, 6) for _ in range(1000)]
print(rolls.count(6))   # about 1/6 of 1000
```

常见错误

- 二分查找 (binary search) 只对**已排序**的列表有效。
- 大 O(Big-O) 说的是时间如何**增长**, 不是确切时间;O(n^2) 的方法只在极小输入时才可能快过 O(n)。
- 冒泡排序 (bubble sort) 是 O(n^2)— 学习可以, 但对大列表很慢。