Searching, sorting & efficiency

Python Reference

Linear & binary search

A search 查找 finds where a value is. Linear search 线性查找 checks each item in turn, so it works on any list.

```
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
```

Binary search 二分查找 is much faster but needs a **sorted** list. It halves the range each step.

```
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
```

Sorting (bubble & insertion)

To sort 排序 is to put items in order. Bubble sort 冒泡排序 repeatedly swaps 交换 neighbours that are in the wrong order.

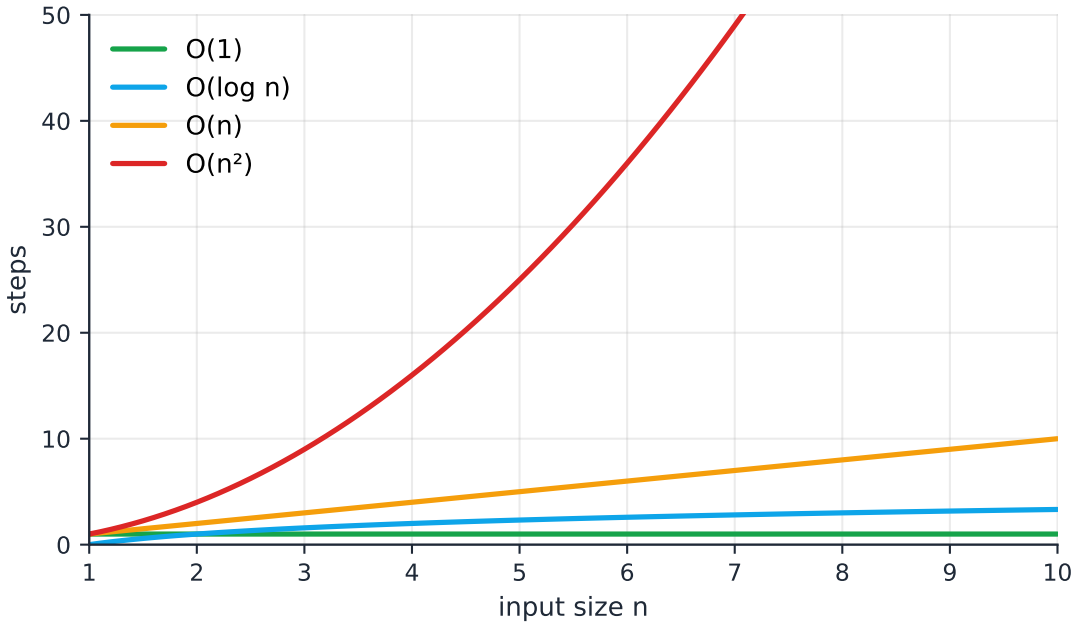
```
def bubble_sort(a):
    a = a[:]                          # work on a copy
    for i in range(len(a)):
        for j in range(len(a) - 1 - i):
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
    return a

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

- Insertion sort 插入排序 builds a sorted part one item at a time.
- In real code, use Python's built-in `sorted()`:

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

Algorithmic efficiency

Efficiency 效率 asks how the work grows as the input grows. We describe it with Big-O 大 O 记号.



How the number of steps grows with input size for common complexities

Big-O	Name	Example
$O(1)$	constant	look up a dict key
$O(\log n)$	logarithmic	binary search
$O(n)$	linear	linear search
$O(n^2)$	quadratic	bubble sort

```
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)
```

Randomness & simulation

The `random` module makes random numbers. Use a seed 种子 to make results repeatable. A simulation 模拟 runs many random trials to estimate an answer.

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