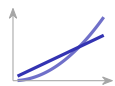
Algorithms

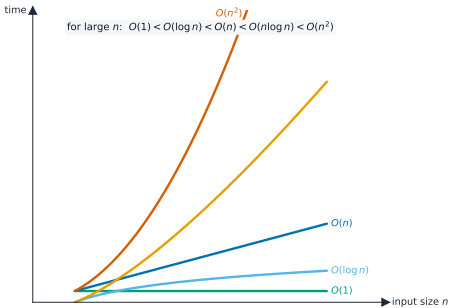
A-Level Computer Science ▪ Topic 19

A-Level Computer Science



What we will cover

1. Searching
2. Sorting
3. ADTs in algorithms
4. Comparing algorithms (Big-O)
5. Recursion
6. Recap



Searching algorithms

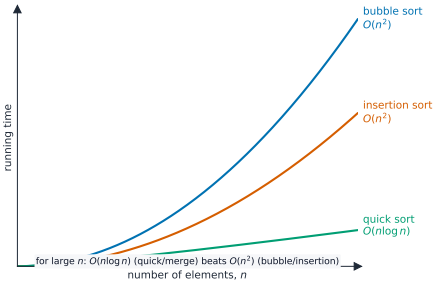
step 1: A B C D E F G H I J K L **M** N O P Q R S T U V W X Y Z — low = A, mid = M, high = Z — M < W, go right

step 2: N O P Q R S **T** U V W X Y Z — low = N, mid = T, high = Z — T < W, go right

step 3: U V **W** X Y Z — low = U, mid = W, high = Z — W = W, **found**

phone book

Sorting algorithms



- **bubble** 冒泡排序 / **insertion** 插入排序: simple, $O(n^2)$
- **merge sort** 归并排序: divide and conquer, $O(n \log n)$

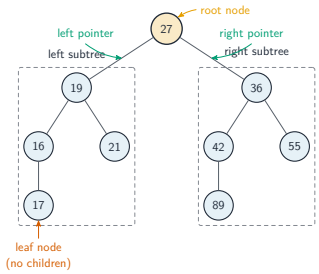
> The $O(n^2)$ sorts are fine for small lists; $O(n \log n)$ wins as n grows large.

ADTs in algorithms

Abstract data types 抽象数据类型
package data with its operations:

- stack (LIFO) – undo, recursion, RPN
- queue (FIFO) – buffers, scheduling
- tree – searching, hierarchy

> Choosing the right ADT often makes the algorithm obvious – a stack makes depth-first traversal trivial.



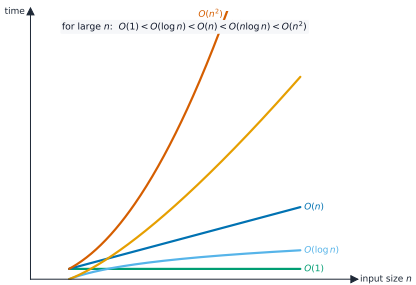
Comparing algorithms

Big-O 大 O 记号 states how the work grows with the input size n:

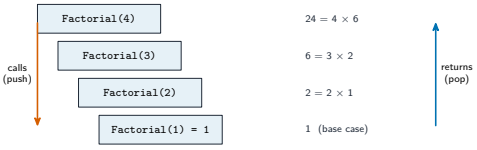
$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2)$$

- it captures the dominant term, ignoring constants
- time and space both matter

> Judge an algorithm by how it scales, not by how it runs on a small test.



Recursion



A **recursive** 递归 routine calls itself on a smaller problem:

- needs a base case to stop
- each call is saved on the call stack

> Elegant, but too deep a recursion overflows the stack – an iterative version uses constant space.

Topic 19 – key takeaways

1 Search

linear $O(n)$; binary $O(\log n)$, sorted

2 Sort

bubble/insertion $O(n^2)$; merge $O(n \log n)$

3 ADTs & Big-O

stack/queue/tree; judge by scaling

4 Recursion

base case + call stack

Check yourself

1. What must a list be for binary search?
2. Big-O of merge sort?
3. Which ADT is LIFO?
4. What stops a recursion?

Answers **1.** sorted **2.** $O(n \log n)$ **3.** stack **4.** the base case