Master公式

一、什么是 Master 公式？

Master公式是**用来分析递归算法的时间复杂度**的一个方法。比如说很多“分治”算法，比如归并排序、快速排序、二分搜索等，它们的时间复杂度可以写成类似这样的形式：

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

你可以这样理解：

- 把一个规模为 n 的问题分成 **a个**子问题；
- 每个子问题的规模是原来的 $1/b$ ；
- 分解 & 合并这些子问题的**额外代价是 f(n)**。

Master公式就是帮你判断：整个算法的总耗时（T(n)）到底是哪一部分占主导——是递归的那部分？还是额外处理的那部分？

二、Master公式的三种情况（用通俗比喻讲）

我们看公式：

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

我们比较的是：

- 递归部分： $n^{\log_b a}$

- 和外部处理时间： $f(n)$

根据这两个量的大小关系，有三种情况：

情况一：递归工作量占主导

$$f(n) = O(n^{\log_b a - \epsilon}) \quad (\epsilon > 0)$$

也就是额外花费比递归部分少很多（递归是老大）。

👉 **结果**：

$$T(n) = \Theta(n^{\log_b a})$$

✅ **举例**：

$$T(n) = 8T(n/2) + n^2$$

- a = 8, b = 2, $\log_b a = \log_2 8 = 3$
- f(n) = n^2 ，它比 n^3 小很多

👉 所以符合情况一，答案就是：

$$T(n) = \Theta(n^3)$$

情况二：额外工作量和递归持平

$$f(n) = \Theta(n^{\log_b a})$$

也就是额外工作和递归工作一样多（五五开）。

👉 **结果**：

$$T(n) = \Theta(n^{\log_b a} \cdot \log n)$$

✅ **举例**：

$$T(n) = 2T(n/2) + n$$

- a = 2，b = 2， $\log_2 2 = 1$
- f(n) = n

👉 所以符合情况二，答案就是：

$$T(n) = \Theta(n \log n)$$

这就是归并排序的复杂度。

情况三：额外工作量占主导

$$f(n) = \Omega(n^{\log_b a + \epsilon}) \quad (\epsilon > 0)$$

并且满足技术条件：**regularity condition**，即：

$$a \cdot f(n/b) \le c \cdot f(n) \quad \text{for some } c < 1$$

也就是额外工作多得多，递归只是小头。

👉 **结果**：

$$T(n) = \Theta(f(n))$$

✅ **举例**：

$$T(n) = 2T(n/2) + n^2$$

- a = 2，b = 2， $\log_2 2 = 1$
- f(n) = n^2 ，它比 n^1 大很多

并且满足 regularity 条件，所以：

$$T(n) = \Theta(n^2)$$

三、总结口诀（便于记忆）

📌 Master公式的口诀可以记成：

> 比上小，服从主（情况一）
> 一样大，乘个log（情况二）
> 比上大，看条件，定主导（情况三）

四、补充：不能用 Master 公式的情况

Master 公式只能用在类似 $T(n) = aT(n/b) + f(n)$ 这种形式的递推式，不能用在：

- 子问题不是均等大小的（比如T(n) = T(n−1) + T(n−2)）
- 合并成本是奇怪函数（比如 f(n) = n / log n）