# 人工智能数学基础：作业 9 参考解答

## 2.1 判断 $\log(n!)$ 是否为 $\Theta(n\log n)$

**定理 1.** $\log(n!) = \Theta(n\log n)$.

证明. 注意到

$$\log(n!) = \sum_{k=1}^{n} \log k.$$

因为对所有 $1 \le k \le n$ 有 $\log k \le \log n$，故

$$\log(n!) = \sum_{k=1}^{n} \log k \le \sum_{k=1}^{n} \log n = n\log n.$$

当 $k \in \{\lceil n/2\rceil, \dots, n\}$ 时，有 $\log k \ge \log(n/2)$，因此

$$\log(n!) = \sum_{k=1}^{n} \log k \ge \sum_{k=\lceil n/2\rceil}^{n} \log(n/2) \ge \frac{n}{2}\log(n/2).$$

又 $\frac{n}{2}\log(n/2) = \frac{n}{2}(\log n - \log 2) = \Omega(n\log n)$。上下界合并即得 $\log(n!) = \Theta(n\log n)$。

**注记 1.** 也可以使用斯特令近似。

□

## 2.2 由 $f(x) = O(g(x))$ 能否推出 $2^{f(x)} = O(2^{g(x)})$?

不能。

反例. 取 $g(x) = x$，$f(x) = 2x$。显然 $f(x) = O(g(x))$（例如 $f(x) \le 2g(x)$ 对所有 $x$ 成立）。但

$$\frac{2^{f(x)}}{2^{g(x)}} = \frac{2^{2x}}{2^{x}} = 2^{x} \xrightarrow[x\to\infty]{} \infty,$$

因此不存在常数 $C$ 使得 $2^{f(x)} \le C \cdot 2^{g(x)}$ 对充分大的 $x$ 成立，即 $2^{f(x)} \notin O(2^{g(x)})$。 □

## 2.3 $f(n) = 2f(\sqrt{n}) + \log n$（$n$ 为完全平方数，$f(2) = 1$）

### (a) 求 $f(16)$

因为 16 为完全平方数，递推可连续展开：

$$f(16) = 2f(4) + \log 16, \qquad f(4) = 2f(2) + \log 4.$$

代入 $f(2) = 1$ 且 $\log 4 = 2,\ \log 16 = 4$ 得

$$f(4) = 2 \cdot 1 + 2 = 4, \qquad f(16) = 2 \cdot 4 + 4 = 12.$$

### (b) 求 $f(n)$ 的大 $O$ 估计

第 $j$ 层共有 $2^j$ 个子问题，每个子问题规模为 $n^{1/2^j}$，该层每个结点的非递归代价为

$$\log\left(n^{1/2^j}\right) = \frac{1}{2^j}\log n.$$

因此第 $j$ 层总代价为

$$2^j \cdot \frac{1}{2^j}\log n = \log n,$$

即 每一层总代价都是 $\Theta(\log n)$。

递归在规模降到 2 时终止：当 $n^{1/2^L} = 2$，有 $2^L = \log_2 n$，故 $L = \log_2 \log_2 n = \Theta(\log\log n)$。于是所有层的总代价为

$$\Theta(\log n) \cdot \Theta(\log\log n) = \Theta(\log n \log\log n).$$

叶子结点个数为 $2^L = \Theta(\log n)$，每个叶子代价为常数 $f(2) = 1$，叶子总贡献为 $\Theta(\log n)$，被 $\Theta(\log n \log\log n)$ 支配。

综上，

$$f(n) = \Theta(\log n \log\log n), \qquad \text{特别地} f(n) = O(\log n \log\log n).$$

**注记 2.** 也可以使用变量代换或者主定理。

## 2.4 单峰序列的分治算法

给定严格单峰序列 $a_1, \ldots, a_n$，存在唯一 $m$ 使得

$$a_1 < a_2 < \cdots < a_m > \cdots > a_n.$$

令 $l = 1, r = n$。当 $l < r$ 时：

1. 取 $mid = \lfloor (l + r)/2 \rfloor$；

2. 若 $a_{mid} < a_{mid+1}$，则峰值在右侧，令 $l = mid + 1$；

3. 否则（$a_{mid} > a_{mid+1}$），则峰值在左侧或就是 $mid$，令 $r = mid$。

最终返回 $m = l(= r)$。

不变式：循环开始时峰值下标 $m$ 总在区间 $[l, r]$ 内。

当比较 $a_{mid}$ 与 $a_{mid+1}$ 时：

- 若 $a_{mid} < a_{mid+1}$，说明序列在 $mid$ 处仍处于严格上升段，因此峰值必在 $mid + 1, \ldots, r$ 中，即 $m \in [mid + 1, r]$，令 $l = mid + 1$ 保持不变式；
- 若 $a_{mid} > a_{mid+1}$，说明在 $mid$ 处已经到达下降段（或峰值在更左），因此 $m \in [l, mid]$，令 $r = mid$ 保持不变式。

区间长度每轮至少减半，最终 $l = r$，由不变式得 $l = r = m$。

每轮 $O(1)$ 比较，轮数 $O(\log n)$，总时间 $O(\log n)$，空间 $O(1)$。

## 2.5 程序排序：最小化最大延迟

每个程序 $i$ 有运行时间 $t_i$、截止时间 $d_i$，完成时间 $f_i$。延迟取

$$T_i = \max\{0, f_i - d_i\}, \qquad \text{目标是最小化} \quad T_{\max} = \max_i T_i.$$

### (a) 四个程序的最小最大延迟

题面图示给出四个程序（按出现顺序记为 $1, 2, 3, 4$）：

$$(t_1, d_1) = (2, 6), \quad (t_2, d_2) = (3, 4), \quad (t_3, d_3) = (4, 5), \quad (t_4, d_4) = (5, 12).$$

按截止时间从小到大排序（Earliest Due Date, EDD）得到顺序：

$$2\ (d = 4)\ \rightarrow\ 3\ (d = 5)\ \rightarrow\ 1\ (d = 6)\ \rightarrow\ 4\ (d = 12).$$

计算完成时间与延迟：

| 顺序位置 | 程序 | $t_i$ | $d_i$ | 完成时间 $f_i$ | 延迟 $T_i = \max(0, f_i - d_i)$ |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 3 | 0 |
| 2 | 3 | 4 | 5 | 7 | 2 |
| 3 | 1 | 2 | 6 | 9 | 3 |
| 4 | 4 | 5 | 12 | 14 | 2 |

故该顺序下 $T_{\max} = 3$。由 (b) 的最优性证明（EDD 最优）可知最小可能的最大延迟为 3。

## (b) 贪婪算法与最优性证明

**贪婪算法.** 将所有程序按截止时间 $d_i$ 非降序排序，然后按该顺序依次运行。

**最优性证明.** 考虑任意一个调度（序列）。若其中存在一对相邻程序 $i$、$j$ 满足 $d_i > d_j$，则称其为 “逆序对”。设这对程序之前的总运行时间为 $S$。

原顺序 $i$ 在前、$j$ 在后时，两者完成时间为

$$f_i = S + t_i, \qquad f_j = S + t_i + t_j.$$

交换后 $j$ 在前、$i$ 在后时，

$$f'_j = S + t_j, \qquad f'_i = S + t_j + t_i.$$

我们比较交换前后，这两个程序导致的最大延迟是否变大。注意到 $T_x = \max(0, f_x - d_x)$ 随 $f_x$ 单调不减。由于 $d_j < d_i$，有

$$f'_j - d_j = S + t_j - d_j \le S + t_i + t_j - d_j = f_j - d_j,$$

即程序 $j$ 的（可能为负的）“lateness” 以及延迟 $T_j$ 不会因交换而变大。

再看程序 $i$：交换后其完成时间由 $f_i = S + t_i$ 变为 $f'_i = S + t_j + t_i = f_j$。因此

$$f'_i - d_i = (S + t_i + t_j) - d_i \le (S + t_i + t_j) - d_j = f_j - d_j,$$

其中不等式使用了 $d_i > d_j$。因此交换后程序 $i$ 的 lateness 也不超过交换前程序 $j$ 的 lateness。

综上，交换后这两个程序的最大 lateness（从而最大延迟 $T_{\max}$）不超过交换前的最大 lateness。因此，我们可以不断对任何调度进行相邻逆序对交换而不增大目标函数，最终得到一个不存在逆序对的调度，即按 $d_i$ 非降序排列的 EDD 调度。于是 EDD 调度达到最小的最大延迟，故贪婪算法最优。