

第一题

结论: $\log n!$ 是 $\Theta(n \log n)$ 的

证明:

(1) 证明 $\log n!$ 是 $O(n \log n)$ 的 (找上界):

$$\because \log n! = \sum_{i=1}^n \log i \leq \sum_{i=1}^n \log n = n \log n$$

$\therefore \log n!$ 是 $O(n \log n)$ 的

(2) 证明 $\log n!$ 是 $\Omega(n \log n)$ 的 (找下界):

$$\because \log n! = \sum_{i=1}^n \log i \geq \sum_{i=\lceil \frac{n}{2} \rceil}^n \log i \geq \sum_{i=\lceil \frac{n}{2} \rceil}^n \log \frac{n}{2} = \frac{n}{2} \log \frac{n}{2}$$

$$\text{又 } \because \frac{n}{2} \log \frac{n}{2} = \frac{n}{2}(\log n - 1) \geq \frac{1}{2}n \cdot \frac{1}{2} \log n = \frac{n}{4} \log n (n \geq 4)$$

$\therefore \log n!$ 是 $\Omega(n \log n)$ 的

综上所述, $\log n!$ 是 $\Theta(n \log n)$ 的

第二题

结论: 无法推出 $2^{f(x)}$ 是 $O(2^{g(x)})$ 的

$\because f(x)$ 是 $O(g(x))$ 的

\therefore 存在 $c > 0$ 和 x_0 , 当 $x \geq x_0$ 时, 有 $f(x) \leq cg(x)$

同理若 $2^{f(x)}$ 是 $O(2^{g(x)})$ 的, 则存在 $k > 0$ 和 x_1 , 当 $x \geq x_1$ 时, 有 $2^{f(x)} \leq k2^{g(x)}$

可以观察到指数级别的增长远快于常数级别的增长

若取 $f(x) = 2x, g(x) = x$, 则 $f(x)$ 是 $O(g(x))$ 的

但 $2^{f(x)} = 2^{2x} = (2^2)^x = 4^x$, 而 $2^{g(x)} = 2^x$, $\lim_{x \rightarrow \infty} \frac{4^x}{2^x} = \lim_{x \rightarrow \infty} 2^x = \infty$, 显然 4^x 不是 $O(2^x)$ 的

综上所述, 无法推出 $2^{f(x)}$ 是 $O(2^{g(x)})$ 的

第三题

(1) $\because f(2) = 1$

$$\therefore f(4) = 2f(2) + \log 4 = 4$$

$$\text{therefore } f(16) = 2f(4) + \log 16 = 12$$

(2) 首先拆开几项观察规律: $f(n) = 2f(n^{\frac{1}{2}}) + \log n = 2^2 f(n^{\frac{1}{2^2}}) + 2 \log n = 2^m f(n^{\frac{1}{2^m}}) + m \log n$

因此令 $n^{\frac{1}{2^m}} = 2$, 即 $n = 2^{2^m}$, 则 $m = \log_2 \log_2 n$

代入上式得: $f(n) = 2^{\log_2 \log_2 n} f(2) + \log_2 \log_2 n \cdot \log n = \log_2 n + \log_2 \log_2 n \cdot \log n = \log n (\log \log n + 1)$

$\therefore \exists c > 0, n_0 > 0$, 当 $n \geq n_0$, $f(n) \leq c \log n \cdot \log \log n$

$\therefore f(n)$ 是 $O(\log n \cdot \log \log n)$ 的

第四题

分治算法思想：取序列中索引为 $\lfloor \frac{n}{2} \rfloor$ 的元素，比较 $a_{\lfloor \frac{n}{2}-1 \rfloor}$ 、 $a_{\lfloor \frac{n}{2} \rfloor}$ 和 $a_{\lfloor \frac{n}{2}+1 \rfloor}$ 的大小。若 $a_{\lfloor \frac{n}{2}-1 \rfloor} > a_{\lfloor \frac{n}{2} \rfloor} > a_{\lfloor \frac{n}{2}+1 \rfloor}$ ，则 a_m 在前半部分，进一步从 1 至 $\lfloor \frac{n}{2} \rfloor$ 中进行递归处理；若 $a_{\lfloor \frac{n}{2}-1 \rfloor} > a_{\lfloor \frac{n}{2} \rfloor} > a_{\lfloor \frac{n}{2}+1 \rfloor}$ ，则 a_m 在前半部分，进一步从 1 至 $\lfloor \frac{n}{2} \rfloor$ 中进行递归处理；若 $a_{\lfloor \frac{n}{2}-1 \rfloor} < a_{\lfloor \frac{n}{2} \rfloor} < a_{\lfloor \frac{n}{2}+1 \rfloor}$ ，则 a_m 在后半部分，进一步从 $\lfloor \frac{n}{2} \rfloor$ 至 n 中进行递归处理；若 $a_{\lfloor \frac{n}{2}-1 \rfloor} < a_{\lfloor \frac{n}{2} \rfloor}$ 且 $a_{\lfloor \frac{n}{2} \rfloor} > a_{\lfloor \frac{n}{2}+1 \rfloor}$ ，则 $a_{\lfloor \frac{n}{2} \rfloor}$ 即为 a_m ，返回索引 m

算法 Python 代码：

```
function findPeak(a, low, high):
    if low == high:
        return low
    mid = floor((low + high) / 2) # 取中间索引
    if mid == 1:      # 处理左侧边界
        if a[1] > a[2]:
            return 1
        else:
            return findPeak(a, 2, high)
    if mid == n:      # 处理右侧边界
        if a[n-1] < a[n]:
            return n
        else:
            return findPeak(a, low, n-1)
    # 函数核心
    if a[mid-1] < a[mid] and a[mid] > a[mid+1]:
        return mid
    elif a[mid-1] > a[mid]:  # 在递减部分，峰值在左
        return findPeak(a, low, mid-1)
    else:  # a[mid] < a[mid+1]，在递增部分，峰值在右
        return findPeak(a, mid+1, high)
```

时间复杂度分析：每次递归将问题规模减半，递归树的高度为 $O(\log n)$ ，每层递归的工作量为 $O(1)$ ，因此总时间复杂度为 $O(\log n)$ ，满足分治算法的效率要求

第五题

(1) 按 Deadline 时间升序的规则排列所有程序来运行，可以最小化所有程序的最大延迟。计算按此种规则运行时这四个程序的延迟：

- 2: $f_2 = 0 + t_2 = 3$, $d_2 = 4 \rightarrow L = f_2 - d_2 = 3 - 4 = -1$
- 3: $f_3 = f_2 + t_3 = 7$, $d_3 = 5 \rightarrow L = f_3 - d_3 = 7 - 5 = 2$
- 1: $f_1 = f_3 + t_1 = 9$, $d_1 = 6 \rightarrow L = f_1 - d_1 = 9 - 6 = 3$
- 4: $f_4 = f_1 + t_4 = 14$, $d_4 = 12 \rightarrow L = f_4 - d_4 = 14 - 12 = 2$

因此最大延迟为 3

(2) 贪心算法框架：

```
Sort procedures by Deadline in ascending order
so that d1 <= d2 <= ... <= dn
currentTime = 0
L_max = 0
for i from 1 to n:
```

```

运行程序 Pi
currentTime = currentTime + ti
fi = currentTime
Li = fi - di
L_max = max(L_max, Li)
return L_max

```

假设我们需要针对已有的调度方式进行优化，使最大延迟不断减小至最小值

那么这种情境中一定存在相邻的程序 P_i 和 P_j ($i < j$)，在该调度方式下 $d_i > d_j$

该调度方式下，假设 $f_{i-1} = t_0$

则程序 P_i 和 P_j 的完成时间为 $f_i = t_0 + t_i$ 和 $f_j = t_0 + t_i + t_j$

此时，程序 P_i 和 P_j 的延迟分别为 $L_i = f_i - d_i = t_0 + t_i - d_i$ 和 $L_j = f_j - d_j = t_0 + t_i + t_j - d_j$

若交换程序 P_i 和 P_j 的调度顺序，则新的完成时间为 $f'_j = t_0 + t_j$ 和 $f'_i = t_0 + t_j + t_i$

此时，程序 P_i 和 P_j 的延迟分别为 $L'_j = f'_j - d_j = t_0 + t_j - d_j$ 和 $L'_i = f'_i - d_i = t_0 + t_j + t_i - d_i$

由于 $d_i > d_j$ ，因此 $t_0 + t_i + t_j - d_i < t_0 + t_i + t_j - d_j$ ，即 $L'_i < L_j$

又因为 $L_i = t_0 + t_i - d_i < t_0 + t_j - d_j = L'_j$

因此交换后顺序执行的相应位置上延迟均不大于交换前的延迟

通过不断交换相邻程序的位置可以使该调度方式的最大延迟不断减小，直至不存在相邻程序 P_i 和 P_j ($i < j$) 使 $d_i > d_j$ ，即按 *Deadline* 升序排列，从而证明了贪心算法的最优性