Algorithm HW5

1. (1) $\Theta(n^2)$。

   (2) 分治算法伪代码如下：

```
MaxSub(l, r):
    if l == r:
        return max(0, a[l])
    mid = (l + r) / 2
    leftMax  = MaxSub(l, mid)
    rightMax = MaxSub(mid+1, r)

    sum = 0; leftBorder = 0
    for i = mid downto l:
        sum += a[i]
        leftBorder = max(leftBorder, sum)

    sum = 0; rightBorder = 0
    for j = mid+1 to r:
        sum += a[j]
        rightBorder = max(rightBorder, sum)

    crossMax = leftBorder + rightBorder
    return max(leftMax, rightMax, crossMax)
```

   时间复杂度为 $\Theta(n \log n)$。

   (3) 设

$$b(j) = \max_{1 \le i \le j} \sum_{k=i}^{j} a_k,$$

   即“必须以第 j 个元素结尾的最大子段和”。若最优子段以 j 结尾，则要么它只包含 a_j，要么是在以 $j-1$ 结尾的最优子段后再加上 a_j。因此状态转移为

$$b(j) = \max\{a_j, b(j-1) + a_j\}.$$

全局最优解为

$$\max_{1\le j\le n} b(j),$$

该转移显式使用了最优子结构：以 j 结尾的最优解只依赖于以 $j-1$ 结尾的最优解。动态规划算法伪代码如下：

```
MaxSubSum(a[1..n]):
    b = 0
    ans = 0
    for j = 1 to n:
        b = max(a[j], b + a[j])
        ans = max(ans, b)
    return ans
```

该算法时间复杂度为 $\Theta(n)$。

2. 如果已知分配给 A 的作业集合，其总时间 T_A 确定，那么

$$T_B = \sum_{i=1}^{n} b_i - \sum_{i\in A} b_i,$$

所以问题本质是在所有作业中选择一个子集给 A，使

$$\max\left(T_A, \sum b_i - \sum_{i\in A} b_i\right)$$

最小。

最优子结构说明：考虑前 i 个作业。若在最优调度中，第 i 个作业分配给机器 A，则前 $i-1$ 个作业必须已经以"最优方式"分配，使得当前 A 的总时间达到该状态下的最优值；若第 i 个作业分配给 B，同理。因此，前 (i) 个作业的最优决策可以由前 (i-1) 个作业的最优决策递推得到，满足最优子结构性质，适合动态规划。

设

$f[i][t] =$ 前 i 个作业中，分配给 A 的总时间为 t 时，B 的最小总时间

这是一个典型的二维 DP，初始条件：

$$f[0][0] = 0, \quad \text{其余状态为} +\infty$$

对第 i 个作业，有两种选择：

1）作业 i 分给 A

$$f[i][t] = \min\left(f[i][t], f[i-1][t-a_i]\right)$$

2）作业 i 分给 B

$$f[i][t] = \min\left(f[i][t], f[i-1][t]+b_i\right)$$

所有作业完成后，对所有可能的 t，计算

$$\min_t \max\{t, f[n][t]\}$$

即为最小总完成时间。

伪代码

```
sumA = sum of all a[i]
initialize f[0..n][0..sumA] = +∞
f[0][0] = 0

for i = 1 to n:
    for t = 0 to sumA:
        f[i][t] = +∞
        if t >= a[i]:
            f[i][t] = min(f[i][t], f[i-1][t-a[i]])
        f[i][t] = min(f[i][t], f[i-1][t] + b[i])

ans = +∞
for t = 0 to sumA:
    ans = min(ans, max(t, f[n][t]))

return ans
```

例子计算：题目给定

$$n = 6, \quad a = (2, 5, 7, 10, 5, 2), \quad b = (3, 8, 4, 11, 3, 4)$$

通过 DP 计算所有可行分配后，最优解为：

$$T_A = 16, \quad T_B = 17$$

因此最小总完成时间为 17。

3. 这是一个**每个物品最多可选 2 次的背包问题**。

   最优子结构性质说明：考虑前 i 个物品、剩余容量为 w 的最优解。在最优解中，第 i 个物品只可能取三种状态之一：取 0 个、1 个或 2 个。若最优解中第 i 个物品取 $k \in 0, 1, 2$ 个，则剩余问题必然是："在容量 $w - ka_i$ 下，由前 $i-1$ 个物品构成的最优解"。否则，若子问题不是最优，则可替换得到更优整体解，与最优性矛盾。因此该问题满足最优子结构，适合动态规划。

   设

   $$dp[i][w] = \text{使用前 } i \text{ 个物品，在容量为 } w \text{ 时可获得的最大价值}$$

   其中：$i = 0, 1, \ldots, n$，$w = 0, 1, \ldots, b$。

   初始条件：

   $$dp[0][w] = 0, \quad \forall w$$

   对第 i 个物品，有三种选择：

   - 不选：$dp[i-1][w]$
   - 选 1 个（若 $w \geq a_i$）：$dp[i-1][w - a_i] + c_i$
   - 选 2 个（若 $w \geq 2a_i$）：$dp[i-1][w - 2a_i] + 2c_i$

   因此转移为：

   $$dp[i][w] = \max \begin{cases} dp[i-1][w] \\ dp[i-1][w - a_i] + c_i & w \geq a_i \\ dp[i-1][w - 2a_i] + 2c_i & w \geq 2a_i \end{cases}$$

   算法伪代码

```
initialize dp[0..n][0..b] = 0

for i = 1 to n:
    for w = 0 to b:
        dp[i][w] = dp[i-1][w]
        if w >= a[i]:
```

```
            dp[i][w] = max(dp[i][w],
                           dp[i-1][w-a[i]] + c[i])
        if w >= 2*a[i]:
            dp[i][w] = max(dp[i][w],
                           dp[i-1][w-2*a[i]] + 2*c
[i])

return dp[n][b]
```

4. 系统可靠性为

$$R=\prod_{i=1}^{n} g_i(m_i), \qquad \sum_{i=1}^{n} c_i m_i \le C, \quad m_i \in \mathbb{Z}_{\ge 1}.$$

其中第 i 级并联 m_i 台相同设备，单台成本 c_i，该级可靠性为 $g_i(m_i)$（随 m 单调递增）。

为了把"乘积最大化"变成可加的形式，取对数：

$$\max \prod_{i=1}^{n} g_i(m_i) \iff \max \sum_{i=1}^{n} \ln g_i(m_i).$$

记

$$v_i(m)=\ln g_i(m).$$

最优子结构性质：设前 i 级在预算为 B 的最优对数可靠性为 $F(i,B)$。若最优解在第 i 级选了 $m_i=m$，则它贡献 $v_i(m)$，同时前 $i-1$ 级必须在剩余预算 $B-mc_i$ 下达到最优 $F(i-1,B-mc_i)$。否则若前缀不是最优，用更优前缀替换即可提升总目标，和"整体最优"矛盾。因此该问题满足最优子结构，可用动态规划。

定义状态

$$dp[i][B] = \text{在总预算恰为 } B \text{ 时，前 } i \text{ 级能达到的最大 } \sum_{k=1}^{i} \ln g_k(m_k).$$

初始化

$$dp[0][0]=0, \quad dp[0][B\neq 0]=-\infty.$$

状态转移：第 i 级选择并联台数 $m \ge 1$，花费 mc_i，要求 $mc_i \le B$：

$$dp[i][B] = \max_{m \ge 1, mc_i \le B} \Big(dp[i-1][B - mc_i] + \ln g_i(m) \Big).$$

取

$$\max_{0 \le B \le C} dp[n][B]$$

并将其指数化得到最大可靠性

$$R^* = \exp\Big(\max_{0 \le B \le C} dp[n][B] \Big).$$

若还要输出每级 m_i，在转移时记录取得最大值的 m 作为决策即可回溯。

伪代码如下：

```
ReliabilityDesign(n, C, c[1..n], g[1..n]):
    for B = 0..C:
        dp[0][B] = -INF
    dp[0][0] = 0

    for i = 1..n:
        for B = 0..C:
            dp[i][B] = -INF
            choice[i][B] = 0
            for m = 1..floor(B / c[i]):
                val = dp[i-1][B - m*c[i]] + ln(g[i](m))
                if val > dp[i][B]:
                    dp[i][B] = val
                    choice[i][B] = m

    bestB = 0
    best = -INF
    for B = 0..C:
        if dp[n][B] > best:
            best = dp[n][B]
            bestB = B
```

```
    Rstar = exp(best)

    // recover m_i
    B = bestB
    for i = n..1:
        m[i] = choice[i][B]
        B = B - m[i]*c[i]

    return (Rstar, m[1..n])
```