北京电子科技学院（BESTI）

实　　验　　报　　告

课程：算法分析与设计　班级：（）　姓名：（）　学号：（）

成绩：　　指导教师：赵绪营　　实验日期：

实验密级：　　预习程度：　　实验时间：

仪器组次：　　必修/选修：（）　　实验序号：（1/2/3）

实验名称：（必填项）

实验目的与要求：

（必填项）

实验仪器（软件/硬件环境）：

名称	型号	数量
Dev-C++	5.11	1
TDM-GCC	11.2.0	1

一、实验内容

第三组 题目 12（难度分数：5）木棍问题

问题描述: 在宇宙的某个角落，存在着地外文明，这些外星人都有 M 根串在一起的长棍。这些棍棒是一颗既有头也有尾宽度的木棍，而且这些宽度都是正整数。对于相邻的两个木棍，前一个木棍的尾宽度一定等于后一个木棍的头宽度。这样，两个木棍才能合成一个木棍，同时放出能量（类似核聚变）。如果前一根木棍的头宽度为 m，尾宽度为 r，后一根木棍的头宽度为 r，尾宽度为 n，则合二为一后释放的能量为 m*r*n，新产生的木棍的头宽度为 m，尾宽度为 n。遇到危险时候，外星人就聚合相邻的两个木棍，通过合二为一获得能量，直到木棍只剩下一根，显然，不一样的合木棍的顺序，得到的总能量是不同的，

例如：设 M=3，3 根木棍的头宽度与尾宽度依次为(2，6) (6，5) （5，2）。用记号！表示两根木棍的聚合，(j！k)表示第 j，k 两根木棍聚合后释放的能量。则第 3、1 两根木棍聚合后释放的能量为：

（注意：由于是串在一起的木棍，所以第一根是和最后一根木棍（第三根）相邻的，所以第

三根木棒才能和第一根木棒聚合!）
(3！1)=5*2*6=60。
所以说你可以有例如以下的合木棒的顺序：
(((1!2)!3))=2*6*5+2*5*2=80 剩下的木棒为（2，2）
((3!1)!2)=5*2*6+5*6*5=210 剩下的木棒为（5，5）
（2!(3!1)）=5*2*6+6*5*6=240 剩下的木棒为（6，6）
当然还可能有其他顺序，但是总有一个最大能量，请你求出来。
数据输入: 输入数据的第一行是一个正整数 M（4≤M≤60），表示串在一起的木棍个数。第二行是 M 个用空格隔开的正整数，所有的正整数均不超过 100。第 i 个数为第 i 根木棒的头宽（1≤i≤M），当 i<M 时，第 i 根木棒的尾宽应该等于第 i+1 根木棍的头宽。第 M 根木棍的尾宽应该等于第 1 根木棍的头宽。
结果输出: 输出只有一个数字，是一个正整数 E，为一个聚合的最大能量值!!!
输出格式(%d\n)

输入示例	输出示例
3 2 6 5	240
4 2 5 6 10	1000
4 2 10 5 6	1020
5 2 3 5 10 60	40260

第四组 题目 13（难度分数：4）：独立任务最优调度问题

问题描述:用两台处理机 A 和 B 处理 n 个作业。设第 i 个作业交给机器 A 处理时需要时间 ai，若由机器 B 来处理，则需要时间 bi。由于各作业的特点和机器的性能关系，可能对于某些 i，有 ai≥bi，而对于某些 j(j≠i)，有 aj<bj。既不能将一个作业分开由两台机器处理，也没有一台机器能同时处理 2 个作业。设计一个动态规划算法，使得这两台机器处理完这 n 个作业的时间最短（从任何一台机器开工到最后一台机器停工的总时间)。研究一个实例:(a1, a2, a3, a4, as, a6)=(2,5,7,10,5,2)，(b1, b2, b3, b4, b5,b6)=(3,8,4,11,3,4)。

算法设计:对于给定的两台处理机 A 和 B 处理 n 个作业，找出一个最优调度方案，使 2 台机器处理完这 n 个作业的时间最短。

数据输入:第 1 行是 1 个正整数 n，表示要处理 n 个作业。在接下来的 2 行中，每行有 n 个正整数，分别表示处理机 A 和 B 处理第 i 个作业需要的处理时间。

结果输出:输出计算出的最短处理时间。

输入示例	输出示例
6 2 5 7 10 5 2 3 8 4 11 3 4	15
8 1 6 8 9 3 4 7 6	18

12 6 2 3 4 10 9 7	
7 5 8 3 9 10 9 3 7 10 9 4 6 8 10 6	19
4 9 5 21 3 6 3 9 10	12

二、实验分析

第三组 题目 12（难度分数：5）木棍问题

我们可以定义一个状态 dp[i][j]，表示前 i 个作业中有 j 个作业分配给机器 A 时的最短处理时间。通过这个状态，我们可以设计出递推公式来求解问题。

定义状态：

dp[i][j]：表示前 i 个作业中有 j 个作业分配给机器 A 时的最短处理时间。

初始条件：

dp[0][0] = 0：即没有作业时，两台机器的处理时间为 0。

其余 dp[0][j]（j>0）和 dp[i][0]（i>0）初始化为无穷大，因为这些状态不可能达到。（没有作业时，不可能分配超过 0 个作业给机器 A，因此这些状态初始化为无穷大。）（前 i 个作业不可能都不分配给机器 A，因为这会导致机器 B 独自承担所有作业，而机器 A 完全空闲，这不是最优的分配方式。）

状态转移方程：

如果第 i 个作业分配给机器 A，那么 dp[i][j] = min(dp[i-1][j-1] + ai, dp[i][j])。

如果我们决定将第 i 个作业分配给机器 A，那么在分配完前 i-1 个作业后，机器 A 已经处理了 j-1 个作业，当前处理时间是 dp[i-1][j-1]。因此，分配第 i 个作业给机器 A 的处理时间就是 dp[i-1][j-1] + ai。我们需要取这个值和当前 dp[i][j]的最小值，确保 dp[i][j]始终是最小值。

如果第 i 个作业分配给机器 B，那么 dp[i][j] = min(dp[i-1][j] + bi, dp[i][j])。

如果我们决定将第 i 个作业分配给机器 B，那么在分配完前 i-1 个作业后，机器 A 已经处理了 j 个作业，当前处理时间是 dp[i-1][j]。因此，分配第 i 个作业给机器 B 的处理时间就是 dp[i-1][j] + bi。我们需要取这个值和当前 dp[i][j]的最小值，确保 dp[i][j]始终是最小值。

结果：

最终我们需要找出 dp[n][j]（$0 \le j \le n$）中的最小值，作为最终的最短处理时间。

（在处理完所有 n 个作业后，我们需要找到在 j 个作业分配给机器 A 的情况下，整个系统的最短处理时间。因为 j 可以从 0 到 n，所以我们需要遍历所有 dp[n][j]，找到其中最小的那个值。）

第四组 题目 13（难度分数：4）：独立任务最优调度问题

我们可以先确定一个基础的处理机（即所有作业都用这个处理机处理），然后通过动态规划的方法，得出最优的作业分配方案。

定义状态：

dp[j]：表示前 i 个作业中 A 机器花 j 分钟的时候 B 机器所花时间。

初始条件：

dp[0] = 0：即没有作业时，两台机器的处理时间为 0。

状态转移方程：

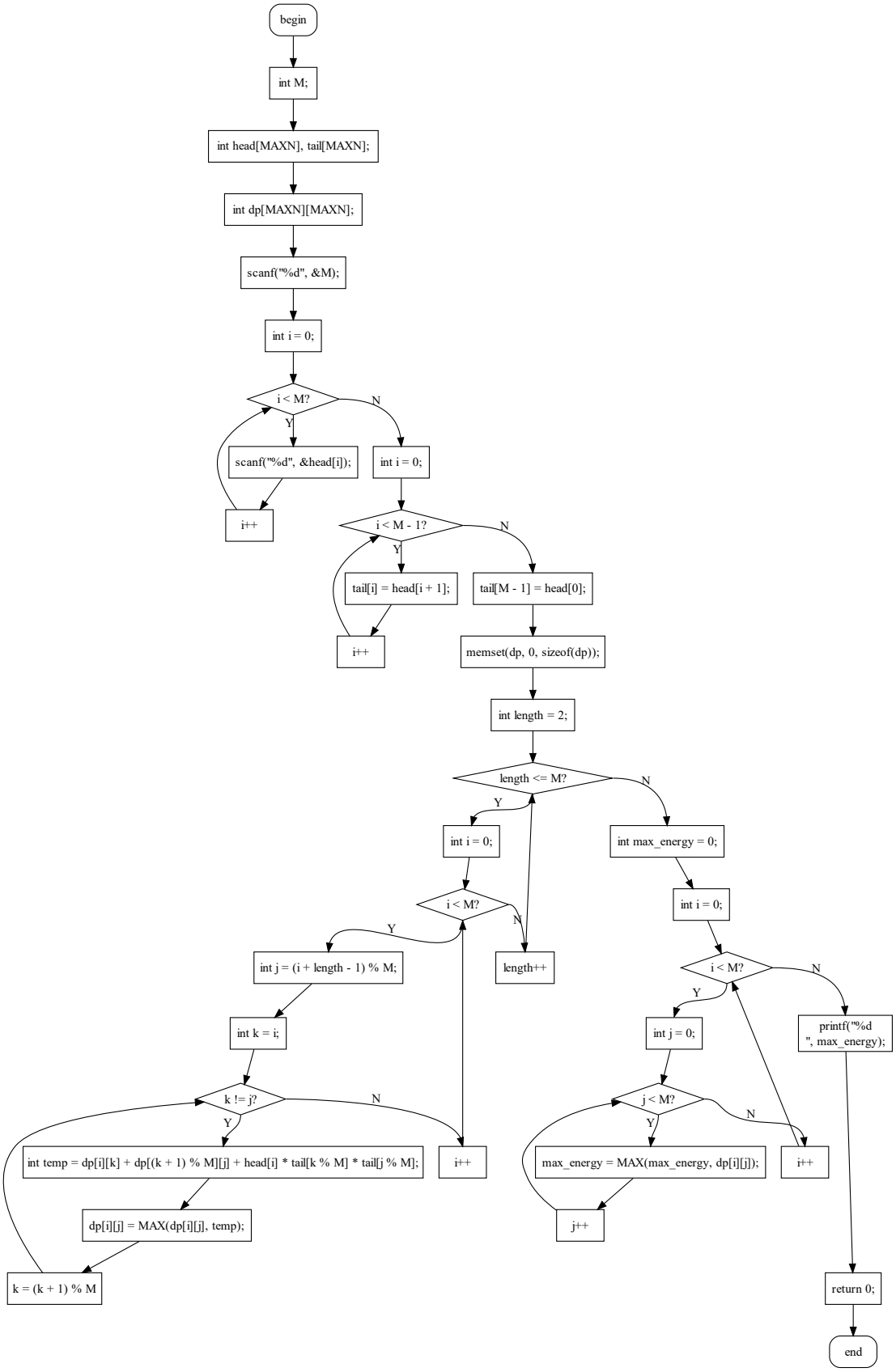
如果第 i 个作业分配给机器 A，那么：dp[i][j] = dp[i-1][j] + b[i];

其中，dp[i-1][j]表示前 i-1 个作业中 A 机器花 j 分钟的时候 B 机器所花时间，加上当前作业 i 由 B 处理机处理的时间 b[i]。
如果第 i 个作业分配给机器 A 且 j >= a[i]，那么：dp[i][j] = min(dp[i-1][j] + b[i], dp[i-1][j-a[i]]);
其中，dp[i-1][j-a[i]]表示前 i-1 个作业中 A 机器花 j-a[i]分钟的时候 B 机器所花时间。
具体步骤：
初始化：将所有作业都交由处理机 A 处理，计算总处理时间 sum。
使用动态规划数组 dp 来记录在不同时间段内处理所有作业的最优处理时间。
遍历所有作业，对每个作业，更新 dp 数组，考虑将该作业交由处理机 A 或处理机 B 处理的最优选择。
最后遍历 dp 数组，找出处理时间的最小值。

三、流程图/源代码

第三组 题目 12（难度分数：5）木棍问题

流程图：

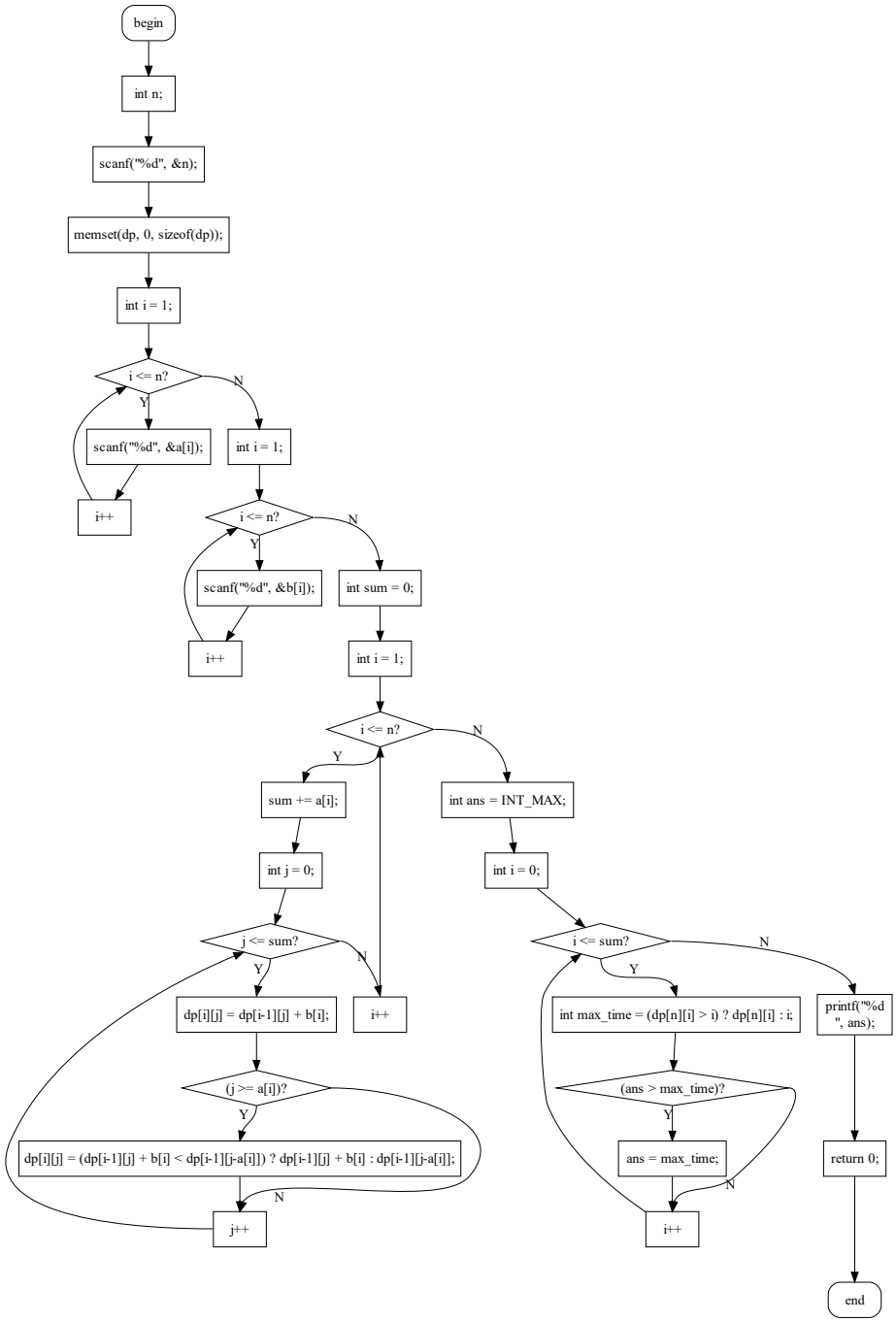


源代码：

```
#include <stdio.h>
#include <string.h>
#define MAXN 60
#define MAX(a, b) ((a) > (b) ? (a) : (b))

int main() {
    int M;
    int head[MAXN], tail[MAXN];
    int dp[MAXN][MAXN];

    // 读取输入
    scanf("%d", &M);
    for (int i = 0; i < M; i++) {
        scanf("%d", &head[i]);
    }
    for (int i = 0; i < M - 1; i++) {
        tail[i] = head[i + 1];
    }
    tail[M - 1] = head[0]; // 环形结构

    // 初始化 DP 数组
    memset(dp, 0, sizeof(dp));

    // 动态规划计算最大能量
    for (int length = 2; length <= M; length++) {  // 子区间长度
        for (int i = 0; i < M; i++) {  // 子区间起点
            int j = (i + length - 1) % M;  // 子区间终点
            for (int k = i; k != j; k = (k + 1) % M) {  // 划分点
                int temp = dp[i][k] + dp[(k + 1) % M][j] + head[i] * tail[k % M] * tail[j % M];
                dp[i][j] = MAX(dp[i][j], temp);
            }
        }
    }

    // 找到最大能量
    int max_energy = 0;
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < M; j++) {
            max_energy = MAX(max_energy, dp[i][j]);
        }
    }
```

```
42.
43.     // 输出结果
44.     printf("%d\n", max_energy);
45.
46.     return 0;
47. }
```

流程图：



源代码：

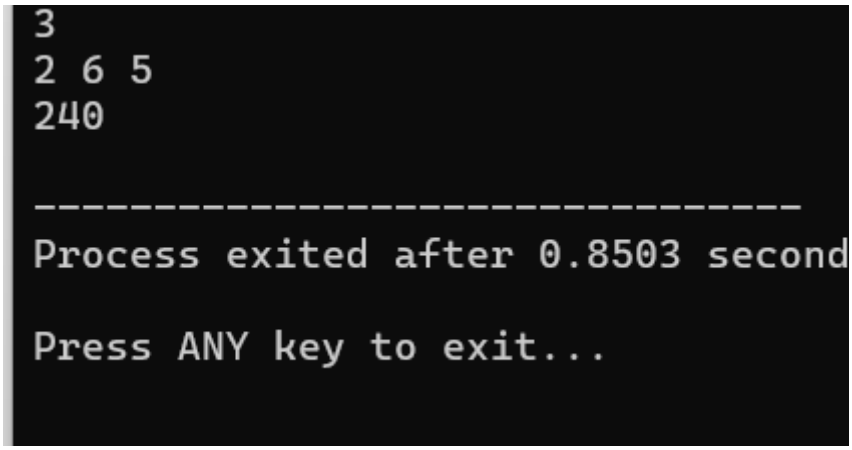
```
1.  #include <stdio.h>
2.  #include <string.h>
```

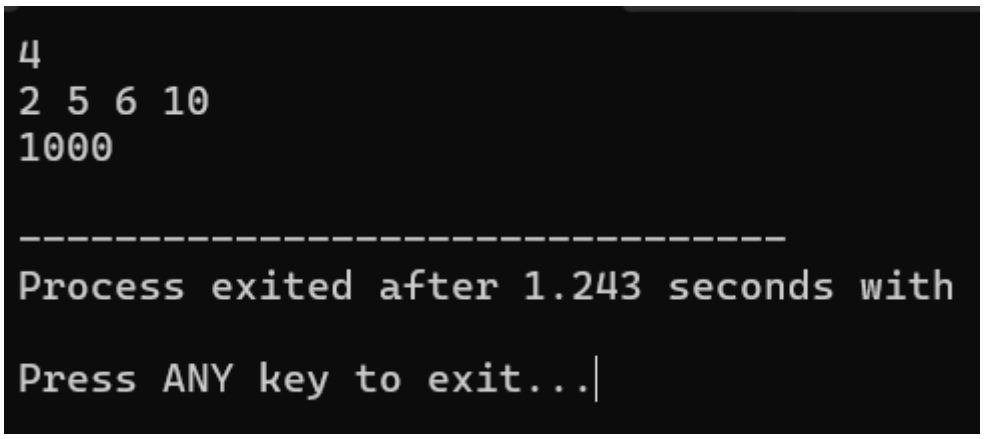
```
#include <limits.h>

int a[201], b[201];
int dp[202][10000]; // dp[i][j] 表示前 i 个作业中 A 机器花 j 分钟的时候 B 机器所花时间

int main() {
    int n;
    scanf("%d", &n);

    memset(dp, 0, sizeof(dp));

    for(int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
    }

    for(int i = 1; i <= n; i++) {
        scanf("%d", &b[i]);
    }

    int sum = 0;

    for(int i = 1; i <= n; i++) {
        sum += a[i];
        for(int j = 0; j <= sum; j++) {
            dp[i][j] = dp[i-1][j] + b[i];
            if(j >= a[i]) {
                dp[i][j] = (dp[i-1][j] + b[i] < dp[i-1][j-a[i]]) ? dp[i-1][j] + b[i] : dp[i-1][j-a[i]];
            }
        }
    }

    int ans = INT_MAX;

    // max(dp[n][i], i) 表示完成前 n 个作业 A 机器花 i 分钟 B 机器花 dp[n][i]分钟情况下，最迟完工时间
    for(int i = 0; i <= sum; i++) {
        int max_time = (dp[n][i] > i) ? dp[n][i] : i;
        if(ans > max_time) {
            ans = max_time;
        }
    }

```

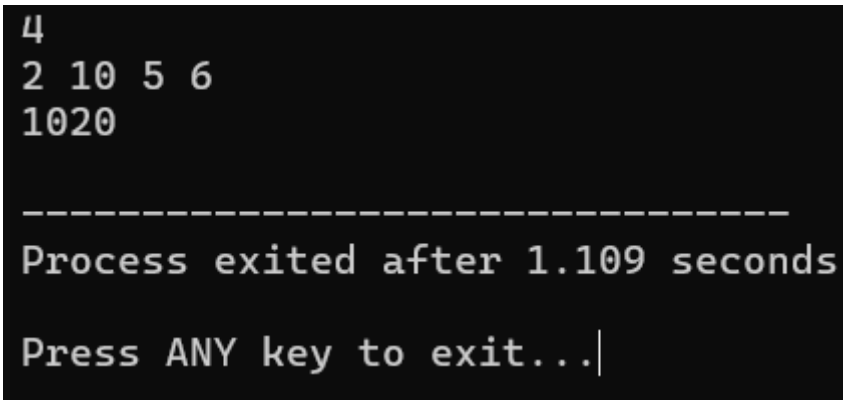
```
    printf("%d\n", ans);

    return 0;
}
```

四、实验结果

第三组 题目 12（难度分数：5）木棍问题







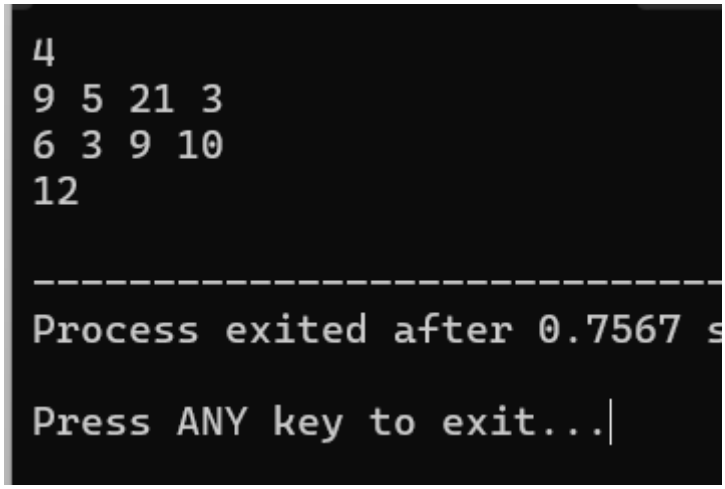
```
5
2 3 5 10 60
40260

--------------------------------
Process exited after 0.8482 seconds wit

Press ANY key to exit...
```

第四组 题目 13（难度分数：4）：独立任务最优调度问题

```
6
2 5 7 10 5 2
3 8 4 11 3 4
15

--------------------------------
Process exited after 11.2 secon

Press ANY key to exit...
```

```
8
1 6 8 9 3 4 7 6
12 6 2 3 4 10 9 7
18
```

```
7
5 8 3 9 10 9 3 7
10 9 4 6 8 10 6
19

--------------------------------
Process exited after 0.7291 second

Press ANY key to exit...
```



五、实验体会

这次算法分析与设计实验给我带来了深刻的启发和丰富的学习体验。通过解决木棍问题和独立任务最优调度问题，我不仅掌握了动态规划的核心思想，还领悟到了将复杂问题分解为子问题的艺术。

在木棍问题中，最初面对这个看似复杂的能量计算题目时，我感到有些无从下手。但当我开始尝试将问题分解，并定义状态 dp[i][j]来表示子区间[i, j]内的最大能量时，问题的结构逐渐清晰起来。通过仔细分析，我发现可以通过遍历不同的划分点 k 来更新状态，这种方法既优雅又高效。在实现过程中，处理环形结构是一个有趣的挑战，它要求我们在数组索引操作时格外小心，使用模运算来模拟循环结构。这个细节让我意识到，在算法设计中，不仅要考虑主要逻辑，还要注意边界情况和特殊结构的处理。

独立任务最优调度问题则让我体会到了动态规划在实际应用中的强大威力。起初，我尝试用贪心策略来解决这个问题，认为总是将任务分配给处理时间较短的机器就能得到最优解。然而，通过仔细分析和验证，我发现这种直观的方法并不总是正确的。转而使用动态规划后，我定义了状态 dp[i][j]来表示前 i 个作业中 A 机器花 j 分钟时 B 机器所花的时间。这种定义方式允许我们全面考虑所有可能的任务分配方案，从而找出真正的最优解。在编程实现时，我注意到 dp 数组的大小需要足够大以容纳所有可能的时间值，这个细节反映了理论分析和实际编程之间的微妙平衡。

在解决这两个问题的过程中，我深刻体会到了算法设计的艺术性。它不仅需要扎实的理论基础，还需要创造性思维和对问题本质的洞察。例如，在木棍问题中，如何巧妙地利用环形结构的特性；在调度问题中，如何设计状态转移方程以捕捉问题的核心特征。这些都需要反复思考和尝试。

实验过程中，我也遇到了一些挑战。比如在调试木棍问题的代码时，因为一个小小的数组越界错误而困扰了很久。这让我意识到，在算法实现中，不仅要关注宏观的设计思路，还要注意微观的编程细节。同时，独立任务调度问题的优化过程让我明白，有时候看似微小的改进可能会带来显著的性能提升。

通过这次实验，我不仅提升了算法设计和编程能力，更重要的是培养了解决复杂问题的思维方式。我学会了如何将大问题分解为小问题，如何在繁杂的细节中把握问题的本质，以及如何在理论分析和实际编码之间找到平衡。这些技能和思维方式无疑将在未来的学习和工作中发挥重要作用，帮助我更好地应对各种挑战。

这次实验经历让我对算法设计产生了更浓厚的兴趣。我意识到，算法不仅仅是解决特定问题的工具，更是一种思考和解决问题的方法论。在未来，我希望能够进一步深入学习更多

的算法设计技巧，并尝试将它们应用到更广泛的领域中，解决实际生活中的复杂问题。