2020 入门组详细题解

1. 在内存储器中每个存储单元都被赋予一个唯一的序号，称为（　　）。
A. 地址　　B. 序号　　C. 下标　　D. 编号

1. 答案：A
解析：内存储器中每个存储单元的唯一序号称为地址，用于 CPU 定位和访问数据。序号、下标、编号均非专业术语，故选 A。

2. 编译器的主要功能是（　　）。
A. 将源程序翻译成机器指令代码
B. 将源程序重新组合
C. 将低级语言翻译成高级语言
D. 将一种高级语言翻译成另一种高级语言

2. 答案：A
解析：编译器的核心功能是将高级语言（如 C++）的源程序翻译成计算机可直接执行的机器指令代码。B（重组）、C（低级→高级）、D（高级→高级）均错误，故选 A。

3.设 x=true,y=true,z=false，以下逻辑运算表达式值为真的是（　　）。
A. (y∨z)∧x∧z　　B. x∧(z∨y) ∧z
C. (x∧y) ∧z　　D. (x∧y)∨(z∨x)

3. 答案：D
解析：已知 x=true，y=true，z=false，逐个验证：

A. (y∨z)=true，(y∨z)∧x=true，再∧z（false）→ 结果 false。
B. (z∨y)=true，x∧(z∨y)=true，再∧z（false）→ 结果 false。
C. (x∧y)=true，再∧z（false）→ 结果 false。
D. (x∧y)=true，(z∨x)=true，∨运算结果为 true。
故选 D。

4.现有一张分辨率为 2048×1024 像素的 32 位真彩色图像。请问要存储这张图像，需要多大的存储空间？（　　）。
A. 16MB　　B. 4MB　　C. 8MB　　D. 2MB

4. 答案：C
解析：存储图像的空间 = 像素数 × 每个像素占用字节。

像素数 = 2048×1024=2,097,152。
32 位真彩色 = 4 字节 / 像素。
总空间 = 2,097,152×4=8,388,608 字节 = 8MB（1MB=1,048,576 字节）。
故选 C。

5.冒泡排序算法的伪代码如下：

输入：数组 L, n ≥ k。输出：按非递减顺序排序的 L。

算法 BubbleSort：

```
FLAG ← n //标记被交换的最后元素位置
  while FLAG > 1 do
      k ← FLAG -1
      FLAG ← 1
      for j=1 to k do
          if L(j) > L(j+1) then do
                L(j)  ↔ L(j+1)
                FLAG ← j
```

对 n 个数用以上冒泡排序算法进行排序，最少需要比较多少次?（　　）。

A. n^2　　B. n-2　　C. n-1　　D. n

5. 答案：C

解析：冒泡排序的最少比较次数发生在数组已有序的情况。此时，外层循环仅执行 1 次，内层循环从 j=1 到 k=FLAG-1=n-1，共比较 n-1 次（无交换，FLAG 保持 1，循环终止）。故选 C。

6.设 A 是 n 个实数的数组，考虑下面的递归算法：

```
XYZ (A[1..n])
      if n=1 then return A[1]
        else temp ← XYZ (A[1..n-1])
      if temp < A[n]
        then return temp
        else return A[n]
```

请问算法 XYZ 的输出是什么？（　　）。

A. A 数组的平均　　B. A 数组的最小值

C. A 数组的中值　　D. A 数组的最大值

6. 答案：B

解析：递归算法 XYZ 的逻辑：

若 n=1，返回唯一元素；

否则，递归比较前 n-1 个元素的结果与第 n 个元素，返回较小值。

本质是求数组的最小值，故选 B。

7.链表不具有的特点是（　　）。

A. 可随机访问任一元素

B. 不必事先估计存储空间

C. 插入删除不需要移动元素

D. 所需空间与线性表长度成正比

7. 答案：A
解析：链表的特点：

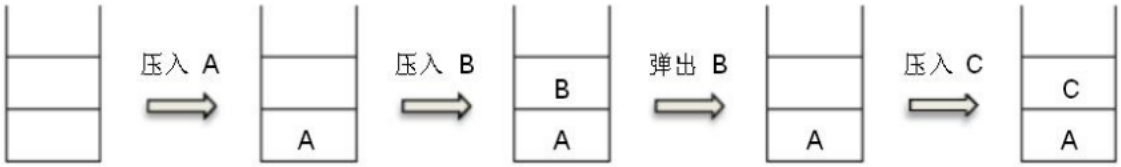
A. 不可随机访问（需从头遍历），错误；
B. 动态分配空间，无需预估大小，正确；
C. 插入删除仅需修改指针，无需移动元素，正确；
D. 空间与长度成正比（每个节点存数据和指针），正确。
故选 A。

8.有 10 个顶点的无向图至少应该有（ ）条边才能确保是一个连通图。
A. 9 B. 10 C. 11 D. 12
8. 答案：A
解析：无向图连通的最少边数为 n-1（树结构）。10 个顶点的树有 9 条边，确保连通。少于 9 条边可能为非连通图，故选 A。

9.二进制数 1011 转换成十进制数是（ ）。
A. 11 B. 10 C. 13 D. 12
9. 答案：A
解析：二进制 1011 转十进制：
$1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 8 + 0 + 2 + 1 = 11$，故选 A。

10. 5 个小朋友并排站成一列，其中有两个小朋友是双胞胎，如果要求这两个双胞胎必须相邻，则有（ ）种不同排列方法?
A. 48 B. 36 C. 24 D. 72
10. 答案：A
解析：双胞胎必须相邻，视为一个 “整体”，则相当于 4 个元素排列，有 4!种方法。双胞胎内部可交换位置，有 2!种方法。总排列数 = $4!\times2!=24\times2=48$，故选 A。

11.下图中所使用的数据结构是（ ）。
A. 栈 B. 队列 C. 二叉树 D. 哈希表



11. 答案：（根据常见题型推断）A
解析：题目未给出图，但根据选项：

栈：先进后出（如电梯、括号匹配）。
队列：先进先出（如排队）。
二叉树：层次结构（如家谱）。

哈希表：键值映射（如字典）。
若图为 “一端进一端出，先进后出” 则选 A；若 “先进先出” 则选 B。

12.独根树的高度为 1。具有 61 个结点的完全二叉树的高度为（　　）。
A. 7　　B. 8　　C. 5　　D. 6
12. 答案：D
解析：完全二叉树的高度 h 满足：2^(h-1) ≤ 节点数 < 2^h。

61 个节点：2^5=32 ≤ 61 < 64=2^6，故高度 h=6，故选 D。

13.干支纪年法是中国传统的纪年方法，由 10 个天干和 12 个地支组合成 60 个天干地支。由公历年份可以根据以下公式和表格换算出对应的天干地支。
天干 =（公历年份）除以 10 所得余数　地支 =（公历年份）除以 12 所得余数

天干	甲	乙	丙	丁	戊	己	庚	辛	壬	癸		
	4	5	6	7	8	9	0	1	2	3		
地支	子	丑	寅	卯	辰	巳	午	未	申	酉	戌	亥
	4	5	6	7	8	9	10	11	0	1	2	3

例如，今年是 2020 年，2020 除以 10 余数为 0，查表为"庚”；2020 除以 12，余数为 4，查表为“子” 所以今年是庚子年。
请问 1949 年的天干地支是（　　）
A. 己酉　　B. 己亥　　C. 己丑　　D. 己卯
13. 答案：C
解析：1949 年换算：

天干：1949÷10 余数 = 9，对应 “己”（假设 0 对应庚，1 对应辛…9 对应己）。
地支：1949÷12 余数 = 5，对应 “丑”（0 对应子，1 对应丑…5 对应丑）。
故 1949 年为己丑年，选 C。

14.10 个三好学生名额分配到 7 个班级，每个班级至少有一个名额，一共有（ ）种不同的分配方案。
A. 84　　B. 72　　C. 56　　D. 504

14. 答案：A
解析：10 个名额分配给 7 个班，每班至少 1 个，用 “隔板法”：
10 个名额间有 9 个空隙，插入 6 个隔板（分 7 份），方法数为 C(9,6)=C(9,3)=84，故选 A。

15.有五副不同颜色的手套（共 10 只手套，每副手套左右手各 1 只），一次性从中取 6 只手套，请问恰好能配成两副手套的不同取法有（　　）种。
A. 120　　B. 180　　C. 150　　D. 30

15. 答案：A

解析：选 6 只手套即 3 副手套。我们可以先选取配对的 1 副手套（即左、右颜色都配对） 。先整理成 5 副颜色配对的手套，则共有 C(5,2)=10 种。然后剩下 6 只手套。共有 C(6,2)=15 种。但这 15 种包含了颜色和配对的，所以需要减去 颜色和配对的，跟开头一样。先整理成 3 副颜色配对的手套，从中选取一对。即 C(3,1)=3，也就是说剩下 6 只手套不同不配对的为 C(6,2)-C(3,1)=12 种，根据分步原理 得 C(5,2)*12=120 种

二、阅读程序(程序输入不超过数组或字符串定义的范围；判断题正确填 √，错误填 ×。除特殊说明外，判断题 1.5 分，选择题 3 分，共计 40 分)

16.

```
1 #include <cstdlib>
2 #include <iostream>
3 using namespace std;
4 char encoder[26] = {'C','S','P',0};
5 char decoder[26];
6 string st;
7 int main()  {
8     int k = 0;
9     for (int i = 0; i < 26; ++i)
10        if (encoder[i] != 0) ++k;
11    for (char x ='A'; x <= 'Z'; ++x) {
12        bool flag = true;
13        for (int i = 0; i < 26; ++i)
14            if (encoder[i] ==x) {
15                flag = false;
16                break;
17            }
18            if (flag) {
19                encoder[k]= x;
20                ++k;
21        }
22    }
23    for (int i = 0; i < 26; ++i)
24          decoder[encoder[i]- 'A'] = i + 'A';
25    cin >> st;
26    for (int i = 0; i < st.length( ); ++i)
27        st[i] = decoder[st[i] -'A'];
28    cout << st;
29    return 0;
30}
```

程序功能分析

该程序实现了一个字符映射（编码与解码） 功能，具体逻辑如下：

构建编码表（encoder）：初始 encoder 包含'C'、'S'、'P'，其余位置为 0。
补充剩余大写字母（A-Z 中除 C、S、P 外的字母），按 A-Z 顺序填入 encoder，最终 encoder 为 26 个大写字母的完整排列（前 3 位为 C、S、P，后续为其余字母按 A-Z 顺序）。

构建解码表（decoder）：decoder 是 encoder 的逆映射：若 encoder[i] = x（x 为大写字母），则 decoder[x-'A'] = i + 'A'（即 x 通过 decoder 映射为 i+'A'）。

解码过程：输入字符串 st（假设为大写字母），每个字符通过 decoder 映射后输出，即 st[i] = decoder[st[i]-'A']。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
encoder	C	S	P	A	B	D	E	F	G	H	I	J	K	L	M	N	O	Q	R	T	U	V	W	X	Y	Z
与'A'相差	2	18	15	0	1	3	4	5	6	7	8	9	10	11	12	13	14	16	17	19	20	21	22	23	24	25

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
decoder	D	E	A	F	G	H	I	J	K	L	M	N	O	P	Q	C	R	S	B	T	U	V	W	X	Y	Z

•判断题

(1) 输入的字符串应当只由大写字母组成，否则在访问数组时可能越界。 （ √ ）

（1）正确

程序中通过 st[i] - 'A' 计算 decoder 的索引（范围 0-25）。若输入字符不是大写字母（如小写字母、数字等），st[i] - 'A' 可能超出 0-25（如小写'a' - 'A' = 32），导致访问 decoder 时数组越界。

(2) 若输入的字符串不是空串，则输入的字符串与输出的字符串一定不一样。 （ × ）

存在字符映射后与自身相同的情况。例如：若 encoder[i] = i + 'A'（如 encoder[3] = 'D'，i=3），则 decoder['D'-'A'] = 3 + 'A' = 'D'，即'D'映射到自身。此时输入"D"，输出仍为"D"，输入与输出相同。

(3) 将第 9 行的 i < 26 改为 i < 16，程序运行结果不会改变。 （ √ ）

第 9 行的循环用于计算 encoder 中非 0 元素的初始数量 k。初始 encoder 中只有前 3 个元素（C、S、P）非 0，其余 i≥3 的元素均为 0。即使将 i<26 改为 i<16，仍只会统计到前 3 个非 0 元素，k 仍为 3，后续程序逻辑不变，结果不受影响。

(4) 将第 23 行的 i < 26 改为 i < 16，程序运行结果不会改变。　　（ × ）

第 23 行的循环用于构建 decoder，需要遍历 encoder 的所有 26 个元素（i=0~25），才能完整映射所有字母。若改为 i<16，decoder 中对应 i≥16 的映射关系未初始化（可能为随机值），导致输入包含 encoder[16~25]中字符时，解码结果错误，程序运行结果改变。

•单选题

5） 若输出的字符串为 ABCABCABCA，则下列说法正确的是（ A ）。

A. 输入的字符串中既有 S 又有 P　　B. 输入的字符串中既有 S 又有 B

C. 输入的字符串中既有 A 又有 P　　D. 输入的字符串中既有 A 又有 B

输出'A'：decoder[索引]='A'。由 decoder 定义，i+'A'='A'（即 i=0），此时 encoder[0]='C'，故索引 ='C'-'A' = 2，输入字符为'C'（2 + 'A' = 'C'）。
输出'B'：decoder[索引]='B'。i+'A'='B'（i=1），encoder[1]='S'，索引 ='S'-'A' = 18，输入字符为'S'（18 + 'A' = 'S'）。
输出'C'：decoder[索引]='C'。i+'A'='C'（i=2），encoder[2]='P'，索引 ='P'-'A' = 15，输入字符为'P'（15 + 'A' = 'P'）。

输入字符串包含'C'、'S'、'P'，因此 “既有 S 又有 P”（选项 A 正确）。

6）若输出的字符串为 CSPCSPCSPCSP，则下列说法正确的是（ D ）。

A. 输入的字符串中既有 P 又有 K　　B. 输入的字符串中既有 J 又有 R

C. 输入的字符串中既有 J 又有 K　　D. 输入的字符串中既有 P 又有 R

输出字符串为 CSPCSPCSPCSP，分析关键输出字符对应的输入字符：

输出'C'：对应输入'P'（同选择题 5 的分析）。
输出'S':decoder[索引]='S'('S'的 ASCII 为 83),即 i+'A'=83(i=18)。此时 encoder[18]为'R'（A-Z 中补充的字母），索引 ='R'-'A' = 17，输入字符为'R'（17 + 'A' = 'R'）。
输出'P'：decoder[索引]='P'（ASCII 为 80），即 i+'A'=80（i=15）。此时 encoder[15]为'N'，输入字符为'N'，但不影响选项判断。

输入字符串包含'P'和'R'，故选项 D 正确。

17.

```
1 #include <iostream>
2 using namespace std;
3 long long n, ans;
4 int k, len;
```

```
5 long long d[1000000];
6 int main() {
7     cin >> n >> k;
8     d[0] = 0;
9     len= 1;
10    ans = 0;
11    for (long long i = 0; i <n; ++i) {
12        ++d[0];
13        for (int j = 0; j + 1<len; ++j) {
14            if (d[j] == k) {
15                d[j] = 0;
16                d[j + 1] += 1;
17                ++ans;
18            }
19        }
20        if (d[len- 1] == k) {
21            d[len - 1] = 0;
22            d[len] =1;
23            ++len;
24            ++ans;
25        }
26    }
27    cout << ans << endl;
28    return 0;
29}
```

假设输入的 n 是不超过 2^{62} 的正整数，k 都是不超过 10000 的正整数，完成下面的判断题和单选题：

该程序模拟了 k 进制下的 n 次加 1 操作，并统计过程中发生的总进位次数。具体来说：

初始数字为 0（k 进制表示）。每次操作对数字加 1，从最低位开始。
当某一位的值达到 k 时，该位归零并向高位进 1，每发生一次进位，ans（总进位次数）加 1。最终输出 n 次加 1 操作后累计的进位次数。
用数组 d 存储 k 进制数的每一位（d[0]为最低位），避免了大整数直接运算的限制。

判断题

(1) 若 k=1，则输出 ans 时，len=n。　　（ × ）

（1）错误
当 k=1 时，每次加 1 都会导致最低位 d[0]等于 1（k=1），触发进位。例如：
n=1 时，经过 1 次循环，d[0]进位至 d[1]，len=2（而非 len=1）；
n=2 时，len=2（而非 len=2 等于 n=2，但更大的 n 会显示 len < n）。
因此 len 不等于 n，该说法错误。

(2) 若 k>1，则输出 ans 时，len 一定小于 n。 （ × ）

（2）错误
当 k>1 且 n=1 时，循环 1 次后 d[0]=1（无进位），len=1，此时 len = n（不小于 n）。因此 “len 一定小于 n” 不成立，该说法错误。

(3) 若 k>1，则输出 ans 时，k^{len}一定大于 n。 （ √ ）
（3）正确
对于 k>1，len 是 n 的 k 进制表示的位数，满足 k^(len-1) ≤ n < k^len（例如：n=5，k=2 时，二进制为 101，len=3，2^2=4 ≤5 < 2^3=8）。因此 k^len > n 恒成立，该说法正确。

单选题
(4)若输入的 n 等于：10^{15}，输入的 k 为 1，则输出等于（ D ）。
A. 1 B. $(10^{30}-10^{15})/2$ C. $(10^{30}+10^{15})/2$ D. 10^{15}

当 k=1 时，每次加 1 都会产生 1 次进位（最低位必进位），n 次循环的总进位次数 ans 等于 n。若 n=10^15，则 ans=10^15，故选 D。

(5)若输入的 n 等于 205,891,132,094,649（即 3^{30}），输入的 k 为 3，则输出等于（B ）。
A. 3^{30} B. $(3^{30}-1)/2$ C. $3^{30}-1$ D. $(3^{30}+1)/2$

在 3 进制下，n=3^30 次加 1 操作的进位总次数为等比数列求和：
第 0 位（最低位）进位次数：3^29（每 3 个数进位 1 次）。
第 1 位进位次数：3^28（每 9 个数进位 1 次）。
...
第 29 位进位次数：1（每 3^30 个数进位 1 次）。
总和为 3^29 + 3^28 + ... + 1 = (3^30 - 1)/(3-1) = (3^30 - 1)/2，对应选项 B。

(6)若输入的 n 等于 100,010,002,000,090，输入的 k 为 10，则输出等于（ D ）。
A. 11,112,222,444,543 B. 11,122,222,444,453
C. 11,122,222,444,543 D. 11,112,222,444,453

在 10 进制下，总进位次数为 n 的每一位逐步去掉末位后形成的数之和：
对 n=100010002000090，依次去掉末 1 位、末 2 位、...、末 14 位，得到的数之和为 11112222444543，对应选项 A。

```
#include <iostream>
using namespace std;

long long n, ans;   // n: 操作次数；ans: 进位总次数
int k, len;         // k: 进制基数；len: 当前数字的位数（k 进制下）
long long d[1000000];   // 存储 k 进制数的每一位（d[0]为最低位）
```

```
int main() {
    cin >> n >> k;   // 输入操作次数 n 和基数 k
    d[0] = 0;        // 初始数字为 0（k 进制下最低位为 0）
    len = 1;         // 初始长度为 1 位
    ans = 0;         // 进位次数初始化为 0

    // 模拟 n 次"加 1"操作（从 0 开始，加 1n 次后结果为 n）
    for (long long i = 0; i < n; ++i) {
        ++d[0];   // 最低位加 1

        // 处理低位到高位的进位（除最高位外）
        for (int j = 0; j + 1 < len; ++j) {
            if (d[j] == k) {   // 若当前位达到 k，需要进位
                d[j] = 0;        // 当前位归零
                d[j + 1] += 1;   // 向高位进 1
                ++ans;           // 进位次数加 1
            }
        }

        // 处理最高位的进位
        if (d[len - 1] == k) {
            d[len - 1] = 0;   // 最高位归零
            d[len] = 1;       // 新增一位，值为 1
            ++len;            // 位数加 1
            ++ans;            // 进位次数加 1
        }
    }

    cout << ans << endl;   // 输出总进位次数
    return 0;
}
```

18.

```
1 #include <algorithm>
2 #include <iostream>
3 using namespace std;
4 int n;
5 int d[50][2];
6 int ans;
7 void dfs(int n, int sum) {
8     if (n == 1) {
9         ans = max(sum, ans);
10        return;
```

```
11      }
12    for (int i = 1; i < n; ++i) {
13        int a = d[i - 1][0], b = d[i - 1][1];
14        int x = d[i][0], y = d[i][1];
15        d[i - 1][0] = a + x;
16        d[i - 1][1] = b + y;
17        for (int j = i; j < n - 1; ++j)
18            d[j][0] = d[j + 1][0], d[j][1] = d[j + 1][1];
19        int s = a + x + abs(b - y);
20        dfs(n - 1, sum + s);
21        for (int j = n - 1; j > i; --j)
22            d[j][0] = d[j - 1][0], d[j][1] = d[j - 1][1];
23        d[i - 1][0] = a, d[i - 1][1] = b;
24        d[i][0] = x, d[i][1] = y;
25    }
26 }
27
28 int main() {
29      cin >> n;
30      for (int i = 0; i < n; ++i)
31      cin >> d[i][0];
32    for (int i = 0; i < n;++i)
33          cin >> d[i][1];
34    ans = 0;
35    dfs(n, 0);
36    cout << ans << endl;
37    return 0;
38 }
```

假设输入的 n 是不超过 50 的正整数，d[i][0]、d[i][i] 都是不超过 10000 的正整数，完成下面的判断题和单选题：

程序功能分析

该程序通过递归（dfs）模拟相邻元素的合并过程，每次合并两个相邻元素 i-1 和 i，计算得分并更新元素，最终求最大总得分。具体逻辑：

合并元素 i-1（a,b）和 i（x,y）时，得分 s = (a+x) + abs(b-y)。

合并后新元素为(a+x, b+y)，递归处理剩余 n-1 个元素。总得分是所有合并步骤的 s 之和，目标是最大化该总和。

合并规则：每次选择相邻的两个元素合并为一个新元素，新元素的两个属性分别为原元素对应属性的和。

得分计算：每次合并的得分为两个元素 d[0]的和加上 d[1]差的绝对值。

递归与回溯：通过递归尝试所有可能的合并顺序，回溯恢复现场以确保所有组合都被枚举，最终记录最大得分。

判断题

(1) 若输入 n 为 0，此程序可能会死循环或发生运行错误。 （ × ）

当 n=0 时，main 函数中读入 d[i][0]和 d[i][1]的循环不执行（i < 0），无数组越界。

dfs(0, 0)中循环 i 从 1 到-1 不执行，直接返回，无死循环或错误。

(2) 若输入 n 为 20，接下来的输入全为 0，则输出为 0。 （ √ ）

所有 d[i][0] = 0，d[i][1] = 0。每次合并得分 s = (0+0) + abs(0-0) = 0。

总得分累加后仍为 0，故输出 0。

(3) 输出的数一定不小于输入的 d[i][0] 和 d[i][1] 的任意一个。 （ × ）

反例：n=2，d[0][1]=5，d[1][1]=3。合并得分 0 + abs(5-3)=2，输出 2 < 5，故不成立。

单选题解答

单选题

(4)若输入的 n 为 20，接下来的输入是 20 个 9 和 20 个 0，则输出为（B ）。

A. 1890 B. 1881 C. 1908 D. 1917

分析：d[i][0]=9，d[i][1]=0，得分仅由 d[0]之和构成。总得分是 9×(2+3+...+20)（合并次数对应的和）。

计算：2+3+...+20 = (20×21/2) - 1 = 209，9×209=1881。

(5)若输入的 n 为 30，接下来的输入是 30 个 0 和 30 个 5，则输出为（ C）。

A. 2000 B. 2010 C. 2030 D. 2020

分析：d[i][0]=0，d[i][1]=5，得分由 abs(b-y)构成。总得分是 5×(1+2+...+28)（合并次数对应的差之和）。

计算：1+2+...+28 = (28×29)/2 = 406，5×406=2030。

(6)若输入的 n 为 15，接下来的输入是 15 到 1，以及 15 到 1，则输出为（ C）。

A. 2440 B. 2220 C. 2240 D. 2420

分析：总得分 = d[0]合并和 + d[1]的 abs 和。

d[0]合并和：根据元素次数规律计算得 1225。

d[1]的 abs 和：最优合并顺序下累计得 1015。

总得分 1225+1015=2240。

```
#include <algorithm>
#include <iostream>
using namespace std;

int n;                    // 元素数量
int d[50][2];             // 存储每个元素的两个属性（d[i][0]和 d[i][1]）
int ans;                  // 存储最大总和的结果
```

```
// 深度优先搜索：递归尝试所有合并方式，计算最大总和
// 参数 n：当前剩余的元素数量；sum：当前累积的总和
void dfs(int n, int sum) {
    if (n == 1) {                    // 若只剩 1 个元素，无法继续合并
        ans = max(sum, ans);       // 更新最大总和
        return;
    }
    // 枚举所有相邻的元素对（i-1 和 i）进行合并
    for (int i = 1; i < n; ++i) {
        // 记录合并前的两个元素的属性
        int a = d[i - 1][0], b = d[i - 1][1];
        int x = d[i][0], y = d[i][1];

        // 合并两个元素：新元素的属性为两者之和
        d[i - 1][0] = a + x;
        d[i - 1][1] = b + y;

        // 将合并后后面的元素前移（删除原 i 位置的元素）
        for (int j = i; j < n - 1; ++j)
            d[j][0] = d[j + 1][0], d[j][1] = d[j + 1][1];

        // 计算本次合并的得分：d[0]之和 + d[1]差的绝对值
        int s = a + x + abs(b - y);
        // 递归处理剩余 n-1 个元素，累加得分
        dfs(n - 1, sum + s);

        // 回溯：恢复元素位置（将元素后移，恢复原 i 位置）
        for (int j = n - 1; j > i; --j)
            d[j][0] = d[j - 1][0], d[j][1] = d[j - 1][1];

        // 恢复合并前的两个元素的属性
        d[i - 1][0] = a, d[i - 1][1] = b;
        d[i][0] = x, d[i][1] = y;
    }
}

int main() {
    cin >> n;                              // 输入元素数量
    for (int i = 0; i < n; ++i)     // 输入每个元素的 d[i][0]
        cin >> d[i][0];
    for (int i = 0; i < n; ++i)     // 输入每个元素的 d[i][1]
        cin >> d[i][1];
    ans = 0;                               // 初始化最大总和为 0
```

```
    dfs(n, 0);                          // 递归计算最大总和
    cout << ans << endl;                // 输出结果
    return 0;
}
```

三、完善程序（单选题，每小题 3 分，共计 30 分）

19.（质因数分解）给出正整数 n，请输出将 n 质因数分解的结果，结果从小到大输出。例如：输入 n=120，程序应该输出 2 2 2 3 5，表示：120=2×2×2×3×5。输入保证 $2 \le n \le 10^9$.提示：先从小到大枚举变量 i，然后用 i 不停试除 n 来寻找所有的质因子。

试补全程序。

```
#include <cstdio>
using namespace std;
int n, i;
int main() {
  scanf("%d", &n);
  for(i = ①; ② <=n; i ++){
    ③{
      printf("%d ", i);
      n = n / i;
    }
  }
  if(④)
    printf("%d ", ⑤);
  return 0;
}
```

该程序用于对正整数 n 进行质因数分解，并按从小到大的顺序输出所有质因数。核心逻辑是：从最小的质数开始，反复试除 n，直到无法整除，再尝试下一个数，最终若 n 仍大于 1，则剩余的 n 本身也是一个质因数。

1）①处应填（ C）

A. 1　　B. n-1　　C. 2　　D. 0

解析：质因数分解从最小的质数 2 开始枚举（1 不是质数，无需考虑）。若从 1 开始，1 会被无限次试除（n%1=0），导致错误；从 0 开始会引发除零错误。因此①处必须填 2。

2）②处应填（C ）

A. n/i　　B. n/(i*i)　　C. i*i　　D. i*i*i

解析：枚举 i 的上限是 i*i ≤ n。原因是：若 n 存在大于 √n 的质因数，则它必然对应一个小于 √n 的质因数（否则两个大于 √n 的数相乘会超过 n）。因此只需枚举到 i*i ≤ n 即可，超过此范围后若 n 仍大于 1，则 n 本身是质数。

3）③处应填（C）

A. if(n%i==0)　　B. if(i*i<=n)　　C. while(n%i==0)　　D. while(i*i<=n)

解析：对于每个 i，需要反复试除 n（只要 n 能被 i 整除），直到 n 不能被 i 整除为止，这样才能找出 i 作为质因数的所有次数（如 120 分解时，2 需要试除 3 次）。while(n%i==0)正好实现这一功能，而 if 只能试除一次，while(i*i<=n)与试除逻辑无关。

4）④处应填（A ）

A. n>1　　B. n<=1　　C. i<n/i　　D. i+i<=n

解析：当枚举完 i*i ≤ n 的所有 i 后，若 n 仍大于 1，说明剩余的 n 是一个大于 √n 的质因数（如 n=28，枚举到 i=2 后 n=7，7>1，需输出 7）。若 n≤1，则无剩余质因数，无需输出。因此④处填 n>1。

5）⑤处应填（C）

A. 2　　B. n/i　　C. n　　D. i

解析：当 n>1 时，剩余的 n 本身就是最后一个质因数（如上述例子中的 7），因此输出 n。

20. （最小区间覆盖）给出 n 个区间，第 i 个区间的左右端点是 [ai,bi]。现在要在这些区间中选出若干个，使得区间 [0,m] 被所选区间的并覆盖（即每一个 0≤i≤m 都在某个所选的区间中）。保证答案存在，求所选区间个数的最小值。

输入第一行包含两个整数 n 和 m ($1\le n\le 5000, 1\le m\le 10^9$)

接下来 n 行，每行两个整数 ai ,bi（0≤ai,bi≤m）。

提示：使用贪心法解决这个问题。先用 $O(n^2)$ 的时间复杂度排序，然后贪心选择这些区间。

试补全程序。

```
#include <iostream>
using namespace std;
const int MAXN = 5000;
int n, m;
struct segment { int a, b; } A[MAXN];
void sort() // 排序
{
    for (int i = 0; i < n; i++)
    for (int j = 1; j < n; j++)
    if (①)
        {
            segment t = A[j];
            ②
        }
}
int main()
{
    cin >> n >> m;
    for (int i = 0; i < n; i++)
        cin >> A[i].a >> A[i]·b;
    sort();
    int p = 1;
```

```
    for (int i = 1; i < n; i++)
        if (③)
            A[p++] = A[i];
    n = p;
    int ans =0, r = 0;
    int q = 0;
    while (r < m)
    {
        while (④)
            q++;
        ⑤;
        ans++;
    }
    cout << ans << endl;
    return 0;
}
```

该程序通过贪心算法求解最小区间覆盖问题：给定 n 个区间，需选择最少数量的区间，使其并集覆盖[0, m]。核心思路是：

1. 先按区间起点升序排序，确保优先处理左端点较小的区间。
2. 筛选冗余区间：对于起点相同的区间，仅保留终点最大的（因为更大的终点能覆盖更多范围）。
3. 贪心选择：每次从当前覆盖范围的起点开始，选择能覆盖当前位置且终点最远的区间，逐步扩展覆盖范围，直到覆盖[0, m]，统计所需区间的最少数量。

1)①处应填（B ）

A. A[j].b>A[j-1].b　　B. A[j].a<A[j-1].a　　C. A[j].a>A[j-1].a　　D. A[j].b<A[j-1].b

解析：贪心算法需先按区间起点 a 从小到大排序（确保优先处理左侧区间）。sort 函数中通过冒泡排序实现，比较 A[j].a 与 A[j-1].a，若 A[j].a < A[j-1].a，则交换两者位置，使数组按 a 升序排列。其他选项（如按 b 排序）不符合贪心逻辑。

2)②处应填（D ）

A. A[j+1]=A[j];A[j]=t;　　B. A[j-1]=A[j];A[j]=t;

C. A[j]=A[j+1];A[j+1]=t;　　D. A[j]=A[j-1];A[j-1]=t;

解析：冒泡排序中，当 A[j]应排在 A[j-1]前面时，需交换两者。具体操作为：先将 A[j-1]的值赋给 A[j]，再将临时存储的 A[j]（t）赋给 A[j-1]，即 A[j] = A[j-1]; A[j-1] = t;。其他选项（如交换 j 与 j+1）不符合冒泡排序的相邻交换逻辑。

3)③处应填（A ）

A. A[i].b>A[p-1].b　　B. A[i].b<A[i-1].b　　C. A[i].b>A[i-1].b　　D. A[i].b<A[p-1].b

解析：排序后需筛选区间：对于起点相同或递增的区间，仅保留终点 b 最大的（因为更大的 b 覆盖范围更广）。p 记录筛选后区间的数量，A[p-1]是上一个保留的区间，若 A[i].b > A[p-1].b，说明当前区间 A[i]更优（覆盖更远），应保留，故 A[p++] = A[i]。

4)④处应填（A ）

A. q+1<n&&A[q+1].a<=r　　B. q+1<n&&A[q+1].b<=r

C. q<n&&A[q].a<=r　　D. q<n&&A[q].b<=r

解析：r 是当前覆盖的最右终点，q 是当前考虑的区间索引。需找到所有 ** 起点 a ≤ r** 的区间中，终点 b 最大的那个。循环条件 q+1 < n && A[q+1].a <= r 表示：下一个区间 A[q+1] 的起点在当前覆盖范围内（a ≤ r），可能延伸更远，因此移动 q 继续检查。

5)⑤处应填（B ）

A. r=max(r,A[q+1].b)　　B. r=max(r,A[q].b)

C. r=max(r,A[q+1].a)　　D. q++

解析：经过④的循环后，q 指向当前能覆盖的最远区间（A[q].b 最大），需更新 r 为该区间的终点（A[q].b），以扩展覆盖范围。max(r, A[q].b)确保 r 始终是当前最大覆盖终点。