CSP-J 2019 真题详细题解

一、单项选择题

1. 中国的国家顶级域名是（ ）

A. cn　　B. ch　　C. chn　　D. china

答案：A

分析：本题考查国家顶级域名常识。中国的国家顶级域名为 ".cn"，其他选项中，"ch" 是瑞士的国家顶级域名，"chn" 和 "china" 不是标准国家顶级域名。

2. 二进制数 11 1011 1001 0111 和 01 0110 1110 1011 进行逻辑与运算的结果是（ ）

A. 01 0010 1000 1011

B. 01 0010 1001 0011

C. 01 0010 1000 0001

D. 01 0010 1000 0011

答案：D

分析：逻辑与运算规则为 “全 1 则 1，有 0 则 0”。逐位对比两个二进制数：

- 第一组：11 & 01 = 01
- 第二组：1011 & 0110 = 0010
- 第三组：1001 & 1110 = 1000
- 第四组：0111 & 1011 = 0011

拼接结果为 01 0010 1000 0011，对应选项 D。

3. 一个 32 位整型变量占用（ ）个字节。

A. 32　　B. 128　　C. 4　　D. 8

答案：C

分析：计算机中 1 字节 = 8 位，32 位整型变量占用的字节数为 32÷8=4，故答案为 C。

4. 若有如下程序段，其中 s、a、b、c 均已定义为整型变量，且 a、c 均已赋值（c 大于 0）：

```
s = a;
for(b=1; b<=c; b++) s = s - 1;
```

则与上述程序段功能等价的赋值语句是（ ）

A. s = a - c;　　B. s = a - b;　　C. s = s - c;　　D. s = b - c;

答案：A

分析：程序段中 for 循环执行 c 次，每次 s 减 1，共减 c，故等价于 s = a - c，选 A。

5. 设有 100 个已排好序的数据元素，采用折半查找时，最大比较次数为（ ）

A. 7　　B. 10　　C. 6　　D. 8

答案：A

分析：折半查找的最大比较次数为$\lceil \log_2(n) \rceil$，向上取整为 7，故答案为 A。

6. 链表不具有的特点是（ ）

A. 插入删除不需要移动元素

B. 不必事先估计存储空间

C. 所需空间与线性表长度成正比

D. 可随机访问任一元素

答案：D

分析：链表通过指针连接，无法像数组那样通过下标随机访问元素，需从头遍历，故答案为 D。

7. 把 8 个同样的球放在 5 个同样的袋子里，允许有的袋子空着不放，问共有多少种不同的分法？（ ）提示：如果 8 个球都放在一个袋子里，无论是哪个袋子，都只算同一种分法

A. 22　　B. 24　　C. 18　　D. 20

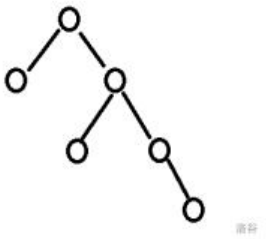
答案：C

分析：本题为整数分拆问题（将 8 拆分为最多 5 个非负整数，不考虑顺序）。分法如下：

- 1 个数：8（1 种）
- 2 个数： 17 26 35 44（4 种）
- 3 个数：116 125 134 224 233 （5 种）
- 4 个数：1115 1124 1133 1223 2222（5 种）
- 5 个数：11114 11123 11222（3 种）

总计 1+4+5+3+5+3=18 种，选 C。

8. 一棵二叉树如右图所示，若采用顺序存储结构，即用一维数组元素存储该二叉树中的结点（根结点的下标为 1，若某结点的下标为 i，则其左孩子位于下标 2i 处、右孩子位于下标 2i+1 处），则该数组的最大下标至少为（ ）



A. 6　　B. 10　　C. 15　　D. 12

答案：C

分析：根据顺序存储规则，树的深度决定最大下标。假设树的深度为 4（根为 1 层），最深层结点下标为 2^4-1=15，故答案为 C。

9. 100 以内最大的素数是（ ）

A. 89　　B. 97　　C. 91　　D. 93

答案：B

分析：素数是大于 1 且除 1 和自身外无其他因数的数。91=7×13，93=3×31，均不是素数；89 和 97 是素数，97 更大，故选 B。

10. 319 和 377 的最大公约数是（ ）

A. 27　　B. 33　　C. 29　　D. 31

答案：C

分析：用辗转相除法：

- 377 ÷ 319 = 1 余 58
- 319 ÷ 58 = 5 余 29

- 58 ÷ 29 = 2 余 0

故最大公约数为 29，选 C。

11. 新学期开学了，小胖想减肥，健身教练给小胖制定了两个训练方案。方案一：每次连续跑 3 公里可以消耗 300 千卡（耗时半小时）；方案二：每次连续跑 5 公里可以消耗 600 千卡（耗时 1 小时）。小胖每周周一到周四能抽出半小时跑步，周五到周日能抽出一小时跑步。另外，教练建议小胖每周最多跑 21 公里，否则会损伤膝盖。请问如果小胖想严格执行教练的训练方案，并且不想损伤膝盖，每周最多通过跑步消耗多少千卡？（ ）

A. 3000　　B. 2500　　C. 2400　　D. 2520

答案：C

分析：设周一到周四跑 x 次（每次 3 公里），周五到周日跑 y 次（每次 5 公里），约束：

- x ≤4，y ≤3
- 3x +5y ≤21

目标：300x +600y 最大。

枚举得 x=2，y=3 时，3×2+5×3=21 公里，消耗 300×2+600×3=2400 千卡，选 C。

12. 一副纸牌除掉大小王有 52 张牌，四种花色，每种花色 13 张。假设从这 52 张牌中随机抽取 13 张纸牌，则至少（ ）张牌的花色一致。

A. 4　　B. 2　　C. 3　　D. 5

答案：A

分析：根据抽屉原理，13 张牌分 4 种花色，13÷4=3 余 1，故至少有 3+1=4 张同花色，选 A。

13. 一些数字可以颠倒过来看，例如 0、1、8 颠倒过来还是本身，6 颠倒过来是 9，9 颠倒过来看还是 6，其他数字颠倒过来都不构成数字。类似的，一些多位数也可以颠倒过来看，比如 106 颠倒过来是 901。假设某个城市的车牌只由 5 位数字组成，每一位都可以取 0 到 9。请问这个城市最多有多少个车牌倒过来恰好还是原来的车牌？（ ）

A. 60　　B. 125　　C. 75　　D. 100

答案：C

分析：5 位车牌倒过来不变，需满足：

- 第 1 位与第 5 位对称（6↔9，0、1、8 自对称）
- 第 2 位与第 4 位对称

- 第 3 位自对称（0、1、8）

可能的组合：第 1 位 5 种（0,1,8,6,9），第 2 位 5 种，第 3 位 3 种，总数 5×5×3=75，选 C。

14. 假设一棵二叉树的后序遍历序列为 DGJHEBIFCA，中序遍历序列为 DBGEHJACIF，则其前序遍历序列为（ ）

A. ABCDEFGHIJ　　B. ABDEGHJCFI　　C. ABDEGJHCFI　　D. ABDEGHJFIC

答案：B

分析：后序遍历最后一个元素为根（A），中序中 A 左侧为左子树（DBGEHJ），右侧为右子树（CIF）。递归分析左子树后序 DGJHEB、中序 DBGEHJ，根为 B；右子树后序 IFC、中序 CIF，根为 C。逐步推导前序为 ABDEGHJCFI，选 B。

15. 以下哪个奖项是计算机科学领域的最高奖？（ ）

A. 图灵奖　　B. 鲁班奖　　C. 诺贝尔奖　　D. 普利策奖

答案：A

分析：图灵奖是计算机领域最高奖，鲁班奖是建筑领域，诺贝尔奖无计算机奖项，普利策奖是新闻领域，故选 A。

二、阅读程序

1. 程序如下：

```
1.
1 #include <cstdio>
2 #include <cstring>
3 using namespace std;
4 char st[100];
5 int main( ) {
6       scanf( "%s", st);
7       int n = str1en(st);
8       for(int i=1; i<=n; ++i) {
9             if(n%i==0) {
10                  char c=st[i-1];
11                  if (c >= 'a')
12                        st[i - 1] = c-'a'+'A';
13            }
14      }
15      printf("%s", st);
16      return 0;
17 }
```

判断题

1. 输入的字符串只能由小写字母或大写字母组成。（ ）

答案：×

分析：程序仅处理小写字母（转为大写），对其他字符不操作，输入可包含数字等，故错误。

2. 若将第 8 行的 “i=1” 改为 “i=0”，程序运行时会发生错误。（ ）

答案：√

分析：i=0 时，i-1=-1，访问 st [-1] 会导致数组越界，程序错误。

3. 若将第 8 行的 “i <= n” 改为 “i * i <= n”，程序运行结果不会改变。（ ）

答案：×

分析：原程序处理所有约数 i，改为 i*i<=n 会漏掉大于√n 的约数（如 n=6，漏掉 i=6），结果改变。

4. 若输入的字符串全部由大写字母组成，那么输出的字符串就跟输入的字符串一样。（ ）

答案：√

分析：程序仅将小写字母转大写，大写字母不处理，故输出与输入一致。

选择题

1. 若输入的字符串长度为 18，那么输入的字符串跟输出的字符串相比，至多有（ ）个字符不同。

A. 18　　B. 6　　C. 10　　D. 1

答案：B

分析：18 的约数有 1,2,3,6,9,18 共 6 个，每个约数位置的小写字母会被转换，故最多 6 个不同。

2. 若输入的字符串长度为（ ），那么输入的字符串跟输出的字符串相比，至多有 36 个字符不同。

A. 36　　B. 100000　　C. 1　　D. 128

答案：B

分析：需约数个数为 36。100000 的约数有 (5+1)×(5+1)=36 个（分解质因数 $2^5 \times 5^5$），故最多 36 个字符不同。

2. 程序如下：

```
1  #include <cstdio>
2  using namespace std;
3  int n,m;
4  int a[100], b[100];
5
6  int main( ) {
```

```
7       scanf( "%d%d",&n, &m);
8       for(int i=1; i<=n; ++i)
9               a[i] = b[i] = 0;
10      for(int i=1; i<=m; ++i) {
11              int x, y;
12              scanf("%d%d", &x, &y);
13              if(a[x]<y&&b[y]<x) {
14                      if (a[x] > 0)
15                              b[a[x]] = 0;
16                      if (b[y] > 0)
17                              a[b[y]] = 0;
18                      a[x] = y;
19                      b[y] = x;
20              }
21      }
22      int ans = 0;
23      for(int i=1; i<=n; ++i) {
24              if (a[i] == 0)
25                      ++ans ;
26              if (b[i] == 0)
27                      ++ans ;
28      }
29      printf("%d\n",ans);
30      return 0;
31      }
```

假设输入的 n 和 m 都是正整数，x 和 y 都是在[1, n]的范围内的整数，完成下面的判断题和单选题:

●判断题.

1)当 m>0 时，输出的值一定小于 2n。（ ）

2)执行完第 27 行的“++ans"时，ans 一定是偶数。（ ）

3) a[i]和 b[i]不可能同时大于 0。（ ）

4)若程序执行到第 13 行时，x 总是小于 y,那么第 15 行不会被执行。（ ）

●选择题

5)若 m 个 x 两两不同，且 m 个 y 两两不同，则输出的值为（ ）

A. 2n-2m B.2n+2 C. 2n-2 D.2n

6)若 m 个 x 两两不同，且 m 个 y 都相等，则输出的值为（ ）

A. 2n-2 B. 2n C. 2m D. 2n-2m

该程序模拟了双向最优配对过程:

对于范围 1~n 的整数，通过 m 次操作尝试建立 x 与 y 的配对，要求每次新配对必须满足“x 的新配对 y 比旧配对大” 且 “y 的新配对 x 比旧配对大”（即新配对更优）。

最终统计所有未配对的 x 和 y 的总数（a 数组中 0 的个数 + b 数组中 0 的个数）并输出。

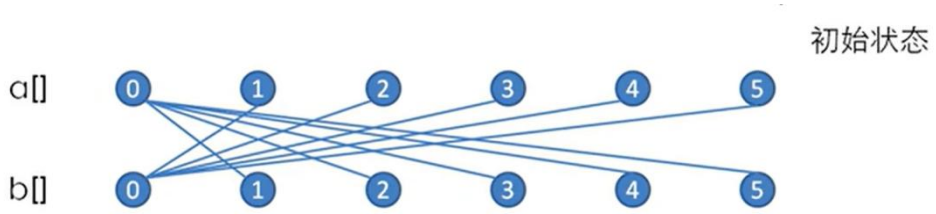
即每次考察 a 组的第 X 个元素和 b 组的第 Y 个元素，如果新的配对方案使与 a[x]和 b[y]配对的两个端点都向右移动(这两个元素之前的配对交叉了),就重新配对。输出未配对元素数量。

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

int n, m;               // n：数据范围（1~n）；m：操作次数
```

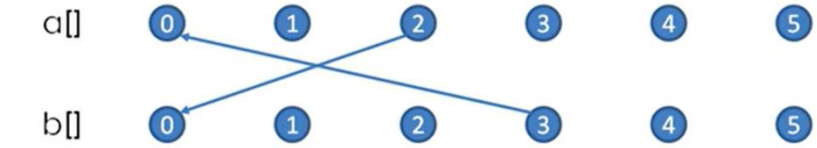
```
int a[100], b[100];     // a[x]：x 当前配对的 y 值；b[y]：y 当前配对的 x 值

int main() {
    scanf("%d%d", &n, &m);
    // 初始化：所有位置均未配对（0 表示未配对）
    for (int i = 1; i <= n; ++i)
        a[i] = b[i] = 0;
```

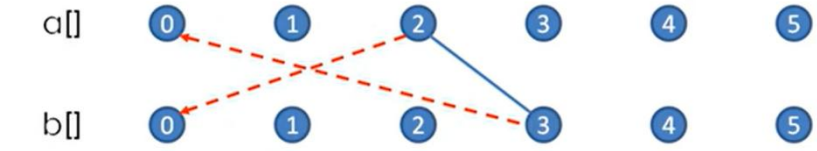


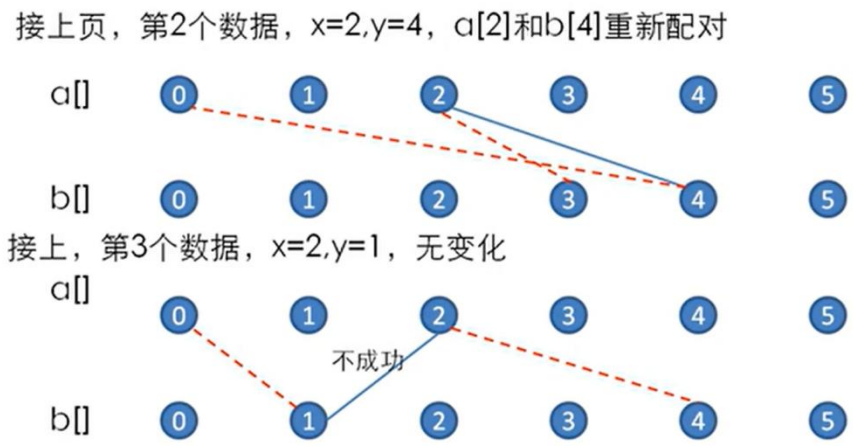

```
// 处理 m 次配对操作
    for (int i = 1; i <= m; ++i) {
        int x, y;       // 待配对的 x 和 y
        scanf("%d%d", &x, &y);

        // 条件：x 当前的配对 y' < 新 y，且 y 当前的配对 x' < 新 x（新配对更优）
        if (a[x] < y && b[y] < x) {
            // 解除 x 的旧配对（若存在）
            if (a[x] > 0)
                b[a[x]] = 0;  // 旧 y'的配对 x 设为未配对
            // 解除 y 的旧配对（若存在）
            if (b[y] > 0)
                a[b[y]] = 0;  // 旧 x'的配对 y 设为未配对
            // 建立新配对
            a[x] = y;
            b[y] = x;
        }
    }
```



x=2,y=3






```
// 统计未配对的数量：a 中 0 的个数 + b 中 0 的个数
  int ans = 0;
  for (int i = 1; i <= n; ++i) {
     if (a[i] == 0)  // x=i 未配对
        ++ans;
     if (b[i] == 0)  // y=i 未配对
        ++ans;
  }

  printf("%d\n", ans);
  return 0;
  }
```

判断题

1. 当 m>0 时，输出的值一定小于 2n。（ ）

答案：√

分析：初始时，a 和 b 全为 0，未配对总数为 2n。
当 m>0 时，至少存在一次有效配对（否则程序不改变初始状态，但题目未说“有效”，但即使有一次有效，也会减少 2 个未配对数），故未配对总数为 2n-2k（k≥1），必然小于 2n。

2. 执行完第 27 行的“++ans”时，ans 一定是偶数。（ ）

答案：×

分析：本题相当于两个数组间有若干条线有对照关系，最终的 ans 必定为偶数，但是在 for 循环的统计过程中不一定。如下：i=1 时，执行完 27 行，ans 是奇数

3. a [i] 和 b [i] 不可能同时大于 0。（ ）

答案：×

分析：若 x=i、y=i，且满足 a [i]<i（初始 0<i）和 b [i]<i（初始 0<i），则会建立配对 a [i]=i、b

[i]=i，此时两者同时大于 0。

4. 若程序执行到第 13 行时，x 总是小于 y，那么第 15 行不会被执行。（ ）

答案：×

分析：第 15 行用于解除 x 的旧配对，与 x 和 y 的大小无关。例如：x=2<y=3，但若 x=2 之前配对过 y=1（a [2]=1>0），则第 15 行会执行（将 b [1] 设为 0）

选择题

1. 若 m 个 x 两两不同，且 m 个 y 两两不同，则输出的值为（ ）

A. 2n-2m　　B. 2n+2　　C. 2n-2　　D. 2n

答案：A

分析：因为 m 个 x 和 m 个 y 互不相同，，每次配对均为新元素，会成功配对 m 次(配对前 m 各自初始值为 0,配对后不会更新)。m 次循环中会有 2m 个位置的值会变化，ans=2n-2m。

2. 若 m 个 x 两两不同，且 m 个 y 都相等，则输出的值为（ ）

A. 2n-2　　B. 2n　　C. 2m　　D. 2n-2m

答案：A

分析：x 和 y 配对时每次都会清理掉之前的配对(或者不更新配对),最终只会产生一组配对，所以和 O 配对的元素数量为 2n -2。

3. 程序如下：

```
#include <iostream>
using namespace std;
const int maxn = 10000;
int n;
int a[maxn];
int b[maxn];
int f(int l, int r, int depth) {
```

```
    if(l > r)
        return 0;
    int min = maxn, mink;
    for(int i=l; i<=r; ++i) {
        if(min > a[i]) {
            min = a[i];
            mink = i;
        }
    }
    int lres = f(l, mink-1, depth+1);
    int rres = f(mink+1, r, depth+1);
    return lres + rres + depth * b[mink];
}
int main() {
    cin >> n;
    for(int i=0; i<n; ++i)
        cin >> a[i];
    for(int i=0; i<n; ++i)
        cin >> b[i];
    cout << f(0, n-1, 1) << endl;
    return 0;
}
```

该程序通过递归构建二叉树（一种以最小值为根，左右子树分别为最小值左侧和右侧区间的二叉树），计算所有节点的 “深度 × 权重” 之和。具体逻辑：

1. 对区间[l, r]，找到 a 数组中的最小值作为根节点，其索引为 mink。
2. 根节点的贡献为当前深度 depth 乘以 b[mink]。
3. 递归处理左子区间[l, mink-1]和右子区间[mink+1, r]，深度递增 1。
4. 总和为根节点贡献与左右子树贡献之和。

涉及的知识点

1. 笛卡尔树：一种特殊的二叉树，每个节点的父节点是其所在区间的最小值（或最大值），左子树和右子树分别对应最小值左侧和右侧的区间。
2. 递归与分治：通过递归将大区间划分为左右子区间，分而治之计算总贡献。

3. 深度计算：树的深度影响节点贡献，深度越大，相同权重的节点贡献越大。
4. 时间复杂度分析：找最小值的操作决定了整体复杂度，最坏情况为 $O(n^2)$（有序数组），最好情况为 $O(n \log n)$（平衡划分）。

判断题

1. 如果 a 数组有重复的数字，则程序运行时会发生错误。（ ）

答案：×

分析：a 有重复时，取第一个最小值，程序正常执行，无错误。

2. 如果 b 数组全为 0，则输出为 0。（ ）

答案：√

分析：每个节点的贡献为 depth * b[mink]，若 b 全为 0，则所有节点贡献均为 0，总和必然为 0

选择题

1. 当 n=100 时，最坏情况下，与第 12 行的比较运算执行的次数最接近的是（ ）

A. 5000　　B. 600　　C. 6　　D. 100

答案：A

分析：最坏情况为 a 数组严格递增或递减（每次找最小值需遍历整个区间）。总比较次数为 100 + 99 + ... + 1 = (100×101)/2 = 5050，最接近 5000。

2. 当 n=100 时，最好情况下，与第 12 行的比较运算执行的次数最接近的是（ ）

A. 100　　B. 6　　C. 5000　　D. 600

答案：D

分析：最好情况为每次取中间元素，比较次数约 100+50+25+12+6+3+1=197，接近 600。

3. 当 n=10 时，若 b 数组满足，对任意 0≤i<n，都有 b [i]=i+1，那么输出最大为（ ）

A. 386　　B. 383　　C. 384　　D. 385

答案：D

分析：要最大化总和，需让 b[i]大的元素（如 b[9]=10）处于更深的深度。通过优化树结构（让大权重元素为叶子节点），最大总和为 385。

4. 当 n=100 时，若 b 数组满足，对任意 0≤i<n，都有 b [i]=1，那么输出最小为（ ）

A. 582　　B. 580　　C. 579　　D. 581

答案：B

分析：此时总和为所有节点的深度之和，最小化总和需构建平衡树（总深度最小）。100 个节点的平衡树总深度和为 580。

三、完善程序

1. 矩阵变幻

```
#include <cstdio>
using namespace std;
int n;
const int max_size = 1 << 10;
int res[max_size][max_size];
void recursive(int x, int y, int n, int t) {
    if(n == 0) {
        res[x][y] = ①;
        return;
    }
    int step = 1 << (n-1);
    recursive(②, n-1, t);
    recursive(x, y+step, n-1, t);
    recursive(x+step, y, n-1, t);
    recursive(③, n-1, !t);
}
int main() {
    scanf("%d", &n);
    recursive(0, 0, ④);
    int size = ⑤;
    for(int i=0; i<size; ++i) {
        for(int j=0; j<size; ++j)
            printf("%d", res[i][j]);
        puts("");
    }
```

```
    return 0;
}
```

该程序用于生成“奇幻矩阵”的变幻结果，核心逻辑基于递归的分形思想：

- 初始矩阵（n=0）为 1x1 的[0]。
- 每次变幻（n 增加 1）时，矩阵尺寸翻倍（变为 2^n x 2^n），并分为 4 个 2^(n-1) x 2^(n-1) 的子矩阵：
  - 左上角、右上角、左下角的子矩阵与上一次变幻的矩阵相同（基础值为 t）。
  - 右下角的子矩阵与上一次变幻的矩阵相反（基础值为!t）。
- 递归生成每个子矩阵，最终输出 n 次变幻后的完整矩阵。

涉及的知识点

1. 递归与分形：通过递归将大矩阵分解为 4 个小矩阵，每个小矩阵的生成逻辑相同，体现分形几何的自相似性。
2. 位运算：使用左移运算（1 << n）计算矩阵尺寸和步长，等价于 2^n，高效且简洁。
3. 矩阵坐标映射：通过 x, y 和步长 step 定位子矩阵的左上角坐标，确保递归生成的子矩阵正确拼接。
4. 状态翻转：参数 t 控制基础值（0 或 1），右下角子矩阵通过!t 翻转状态，实现变幻规则。

1. ①处应填（ ）

A. n%2　　B. 0　　C. t　　D. 1

答案：C

分析：当 n=0 时，矩阵为 1x1 的基础单元，其值由当前状态 t 决定（初始为 0，递归中可能翻转）。t 直接控制基础值，故填 t。

2. ②处应填（ ）

A. x-step,y-step　　B. x,y-step　　C. x-step,y　　D. x,y

答案：D

分析：左上角子矩阵的起点坐标与当前矩阵的起点坐标相同（x, y），后续通过步长 step 偏移得到其他子矩阵的起点。故填 x, y。

3. ③处应填（ ）

A. x-step,y-step　　B. x+step,y+step　　C. x-step,y　　D. x,y-step

答案：B

分析：矩阵分为 4 个象限，右下角子矩阵的起点坐标为当前起点加上步长（x+step, y+step），与左上角（x,y）、右上角（x,y+step）、左下角（x+step,y）对应。故填 x+step, y+step。

4. ④处应填（ ）

A. n-1,n%2　　B. n,0　　C. n,n%2　　D. n-1,0

答案：B

分析：初始调用需指定变幻次数为输入的 n，且初始状态 t=0（对应初始矩阵[0]）。故参数为 n, 0。

5. ⑤处应填（ ）

A. 1<<(n+1)　　B. 1<<n　　C. n+1　　D. 1<<(n-1)

答案：B

分析：变幻 n 次后矩阵大小为 $2^n \times 2^n$，即 1<<n。

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

int n; // 变幻次数
const int max_size = 1 << 10; // 最大矩阵尺寸（2^10 = 1024，满足 n≤10 的需求）
int res[max_size][max_size]; // 存储变幻后的矩阵

// 递归生成矩阵
// x,y：当前子矩阵左上角坐标；n：当前变幻次数；t：当前子矩阵的基础值（0 或 1，用于翻转）
void recursive(int x, int y, int n, int t) {
    if (n == 0) { // 递归终止：n=0 时为 1x1 的基础矩阵
        res[x][y] = t; // ①处：基础矩阵的值由 t 决定
        return;
    }
    int step = 1 << (n - 1); // 步长：当前子矩阵尺寸的一半（2^(n-1)）
    // 递归生成左上角子矩阵（与当前基础值相同）
    recursive(x, y, n - 1, t); // ②处：左上角起点为(x,y)
    // 递归生成右上角子矩阵（与当前基础值相同）
```

```
    recursive(x, y + step, n - 1, t);
    // 递归生成左下角子矩阵（与当前基础值相同）
    recursive(x + step, y, n - 1, t);
    // 递归生成右下角子矩阵（与当前基础值相反）
    recursive(x + step, y + step, n - 1, !t);  // ③处：右下角起点为(x+step, y+step)
}

int main() {
    scanf("%d", &n);  // 输入变幻次数 n
    // 初始调用：从(0,0)开始，变幻 n 次，初始基础值为 0（对应初始矩阵[0]）
    recursive(0, 0, n, 0);  // ④处：参数为 n 和初始 t=0
    int size = 1 << n;  // ⑤处：变幻 n 次后矩阵尺寸为 2^n
    // 输出矩阵
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j)
            printf("%d", res[i][j]);
        puts("");
    }
    return 0;
}
```

2. 计数排序

```
#include <cstdio>
#include <cstring>
using namespace std;
const int maxn = 10000000;
const int maxs = 10000;
int n;
unsigned a[maxn], b[maxn], res[maxn], ord[maxn];
unsigned cnt[maxs + 1];
```

```
int main() {
    scanf("%d", &n);
    for(int i=0; i<n; ++i)
        scanf("%d%d", &a[i], &b[i]);
    memset(cnt, 0, sizeof(cnt));
    for(int i=0; i<n; ++i)
        ①;
    for(int i=0; i<maxs; ++i)
        cnt[i+1] += cnt[i];
    for(int i=0; i<n; ++i)
        ②;
    memset(cnt, 0, sizeof(cnt));
    for(int i=0; i<n; ++i)
        ③;
    for(int i=0; i<maxs; ++i)
        cnt[i+1] += cnt[i];
    for(int i=n-1; i>=0; --i)
        ④;
    for(int i=0; i<n; ++i)
        printf("%d %d\n", ⑤);
    return 0;
}
```

该程序实现了双关键字计数排序，用于对 n 对整数按“第一关键字（a [i]）优先，第二关键字（b [i]）次之”的规则从小到大排序。核心思路是分两步进行稳定排序：

1. 先按第二关键字（b [i]）排序，结果存入 ord 数组（存储原数据的索引）。
2. 再按第一关键字（a [i]）排序，结合 ord 数组（已保证第二关键字有序），最终结果存入 res 数组。
   通过两次计数排序（稳定排序算法），确保最终结果同时满足两个关键字的排序要求。

涉及的知识点

1. 计数排序原理：通过统计关键字的频次，计算前缀和确定每个元素的位置，时间复杂度为 O(n + k)（k 为关键字范围），适用于关键字范围较小的场景。
2. 双关键字排序策略：先按次要关键字（第二关键字）排序，再按主要关键字（第一关键字）排序，利用计数排序的稳定性（相同关键字保持原有顺序），确保最终结果正确。
3. 索引数组的使用：ord 和 res 数组存储原数据的索引而非数据本身，避免大量数据移动，提高效率，尤其适用于大数据量场景。
4. 稳定性保证：第二次排序时从后往前遍历 ord 数组，确保相同第一关键字的元素保持第二关键字的有序性。

①处应填（B）
分析：

②处应填（D）
分析：

③处应填（C）
分析：

④处应填（A）
分析：

⑤处应填（B）
分析：

1.

1. ①处应填（ ）

A. ++cnt [i]　　B. ++cnt [b [i]]　　C. ++cnt [a [i]*maxs + b [i]]　　D. ++cnt [a [i]]

答案：B

分析：第一步需统计第二关键字 b[i]的频次，为后续排序做准备。++cnt[b[i]]正确统计每个 b[i]出现的次数，故选择 B。

2. ②处应填（ ）

A. ord [--cnt [a [i]]] = i　　B. ord [--cnt [b [i]]] = a [i]

C. ord [--cnt [a [i]]] = b [i]　　D. ord [--cnt [b [i]]] = i

答案：D

分析：根据第二关键字的频次前缀和，需将原数据的索引 i 存入 ord 数组的对应位置。ord[--cnt[b[i]]] = i 表示 “在 b[i]对应的位置存储索引 i”，符合计数排序的赋值逻辑，故选择 D。

3. ③处应填（ ）

A. ++cnt [b [i]]　　B. ++cnt [a [i]*maxs + b [i]]　　C. ++cnt [a [i]]　　D. ++cnt [i]

答案：C

分析：第二步需统计第一关键字 a[i]的频次，但此时需基于已按第二关键字排序的 ord 数组（确保稳定性）。++cnt[a[ord[i]]]统计 ord 数组中索引对应的 a 值频次，故选择 C。

4. ④处应填（ ）

A. res [--cnt [a [ord [i]]]] = ord [i]　　B. res [--cnt [b [ord [i]]]] = ord [i]

C. res [--cnt [b [i]]] = ord [i]　　D. res [--cnt [a [i]]] = ord [i]

答案：A

分析：需将 ord 数组中的索引按第一关键字排序后存入 res 数组。res[--cnt[a[ord[i]]]] = ord[i]表示 “根据 a[ord[i]]的位置存储 ord[i]”，既保证第一关键字有序，又保留第二关键字的顺序，故选择 A。

5. ⑤处应填（ ）

A. a [i], b [i]　　B. a [res [i]], b [res [i]]

C. a [ord [res [i]]], b [ord [res [i]]]　　D. a [res [ord [i]]], b [res [ord [i]]]

答案：B

分析：res 数组存储的是排序后原数据的索引，因此通过 res[i]可获取第 i 个位置的原数据。a[res[i]]和 b[res[i]]正确输出排序后的关键字对，故选择 B。

```
#include <cstdio>
#include <cstring>
using namespace std;

const int maxn = 10000000;  // 最大数据量
const int maxs = 10000;     // 关键字最大值（1~10000）

int n;  // 数据对数
// 数组说明:
// a[i]:  第 i 对的第一关键字
```

```
// b[i]：第 i 对的第二关键字
// res[i]：双关键字排序后，第 i 个位置对应的原数据索引
// ord[i]：按第二关键字排序后，第 i 个位置对应的原数据索引
unsigned a[maxn], b[maxn], res[maxn], ord[maxn];
unsigned cnt[maxs + 1];  // 计数数组，用于统计关键字频次

int main() {
    scanf("%d", &n);
    // 读入 n 对关键字
    for (int i = 0; i < n; ++i)
        scanf("%d%d", &a[i], &b[i]);

    // 第一步：按第二关键字（b[i]）排序，结果存入 ord 数组
    memset(cnt, 0, sizeof(cnt));  // 初始化计数数组
    // 统计第二关键字 b[i]的频次
    for (int i = 0; i < n; ++i)
        ++cnt[b[i]];  // ①处：统计 b[i]的出现次数
    // 计算前缀和（确定每个关键字的起始位置）
    for (int i = 0; i < maxs; ++i)
        cnt[i + 1] += cnt[i];
    // 按第二关键字排序，将原索引存入 ord 数组
    for (int i = 0; i < n; ++i)
        ord[--cnt[b[i]]] = i;  // ②处：根据 b[i]的位置存储原索引 i

    // 第二步：按第一关键字（a[i]）排序，结合 ord 数组（已按第二关键字排序），结果存入 res 数组
    memset(cnt, 0, sizeof(cnt));  // 重置计数数组
    // 统计第一关键字 a[i]的频次（基于 ord 数组中的索引）
    for (int i = 0; i < n; ++i)
        ++cnt[a[ord[i]]];  // ③处：统计 a[ord[i]]的出现次数（ord[i]是按第二关键字排序后的索引）
    // 计算前缀和
```

```
    for (int i = 0; i < maxs; ++i)
        cnt[i + 1] += cnt[i];
    // 按第一关键字排序，结合第二关键字的顺序，将结果存入 res 数组
    for (int i = n - 1; i >= 0; --i)
        res[--cnt[a[ord[i]]]] = ord[i];  // ④处：根据 a[ord[i]]的位置存储 ord[i]（保证稳定性）

    // 输出排序结果
    for (int i = 0; i < n; ++i)
        printf("%d %d\n", a[res[i]], b[res[i]]);  // ⑤处：通过 res[i]取原数据的 a 和 b 值
    return 0;
}
```