CCF 编程能力等级认证
Grade Examination of Software Programming

C++ 五级

2023 年 12 月

1 单选题（每题 2 分，共 30 分）

题号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
答案	[redacted]														

第 1 题 下面C++代码用于求斐波那契数列，该数列第1、2项为1，以后各项均是前两项之和。下面有关说法错误的是( )。

```
int fiboA(int N)
{
    if (N == 1 || N == 2)
        return 1;
    return fiboA(N - 1) + fiboA(N - 2);
}
int fiboB(int N)
{
    if (N == 1 || N == 2)
        return 1;
    int last2 = 1, last1 = 1;
    int nowVal = 0;
    for (int i = 2; i < N; i++)
    {
        nowVal = last1 + last2;
        last2 = last1;
        last1 = nowVal;
    }
    return nowVal;
}
```

- ☐ **A.** `fiboA( )` 用递归方式，`fiboB()` 循环方式
- ☐ **B.** `fiboA( )` 更加符合斐波那契数列的数学定义，直观易于理解，而 `fiboB()` 需要将数学定义转换为计算机程序实现
- ☐ **C.** `fiboA( )` 不仅仅更加符合数学定义，直观易于理解，且因代码量较少执行效率更高
- ☐ **D.** `fiboB( )` 虽然代码量有所增加，但其执行效率更高

第 2 题 下面C++代码以递归方式实现合并排序，并假设 `merge (int T[], int R[], int s, int m, int t)` 函数将有序（同样排序规则）的T[s..m]和T[m+1..t]归并到R[s..t]中。横线处应填上代码是( )。

```
void mergeSort(int SList[], int TList[], int s, int t, int len)
{
    if (s == t){
        TList[s] = SList[s];
        return;
    }
    int *T2 = new int[len]; // 保存中间结果
    int m = (s + t) / 2;
    ____________________________________________;
    merge(T2, SList, s, m, t);
    delete T2;
    return ;
}
```

- ☐ **A.** `mergeSort(SList, T2, s, m,len), mergeSort(SList, T2, m,t,len)`

- [ ] **B.** `mergeSort(SList, T2, s, m-1,len), mergeSort(SList, T2, m+1,t,len)`

- [ ] **C.** `mergeSort(SList, T2, s, m,len), mergeSort(SList, T2, m+1,t,len)`

- [ ] **D.** `mergeSort(SList, T2, s, m-1,len), mergeSort(SList, T2, m-1,t,len)`

第 3 题 阅读下面的C++代码，执行后其输出是(　　)。

```
int stepCount = 0;
int fracA(int N)
{
    stepCount += 1;
    cout << stepCount << "->";
    int rtn = 1;
    for (int i = 1; i <= N; i++)
        rtn *= i;
    return rtn;
}
int fracB(int N)
{
    stepCount += 1;
    cout << stepCount << "->";
    if (N == 1)
        return 1;
    return N * fracB(N - 1);
}
int main()
{
    cout << fracA(5);
    cout << "<===>";
    cout << fracB(5);
    return 0;
}
```

- [ ] **A.** `1->120<===>2->120`

- [ ] **B.** `1->120<===>1->120`

- [ ] **C.** `1->120<===>1->2->3->4->5->120`

- [ ] **D.** `1->120<===>2->3->4->5->6->120`

第 4 题 下面的C++用于对 `lstA` 排序，使得偶数在前奇数在后，横线处应填入(　　)。

```
bool isEven(int N)
{
    return N % 2 == 0;
}

void swap(int &a, int &b)
{
    int t;
    t=a,a=b,b=t;
    return;
}

void sortA(int lstA[], int n)
{
    int i,j,t;
    for (i = n-1; i > 0; i--)
        for(j = 0; j < i; j++)
            if(_____________________________)
                swap(lstA[j], lstA[j+1]);

    return;
}
```

- [ ] **A.** `!isEven(lstA[j]) && isEven(lstA[j+1])`

- [ ] **B.** `isEven(lstA[j]) && !isEven(lstA[j+1])`

- [ ] **C.** `lstA[j] > lstA[j+1]`

- [ ] **D.** `lstA[j] < lstA[j+1]`

第 5 题 下面的C++代码用于将字符串保存到带头节点的双向链表中，并对重复的串计数，然后将最新访问的串的节点放在链头便于查找。横线处应填入代码是（ ）。

```
typedef struct Node{
    string str;
    int ref;
    struct Node *next, *prev;
}Node;
Node * Insert(Node *pHead, string s)
{
    Node *p = pHead->next;
    Node *q;
    while(p){
        if(p->str == s){
            p->ref++;
            p->next->prev = p->prev;
            p->prev->next = p->next;
            break;
        }
        p=p->next;
    }
    if(!p) {
        p = new Node;
        p->str = s;
        p->ref=0;
        p->next = p->prev = NULL;
    }
    ____________________________________________
    pHead->next = p, p->prev = pHead;
    return pHead;
}
```

- [ ] **A.** `if(pHead) {p->next = pHead->next, pHead->next->prev = p;}`
- [ ] **B.** `if(pHead->next) {p->next = pHead->next, pHead->next->prev = p;}`
- [ ] **C.** `p->next = pHead->next, pHead->next->prev = p;`
- [ ] **D.** 触发异常，不能对空指针进行操作。

第 6 题 有关下面C++代码说法正确的是（ ）。

```
int rc;
int foo(int x, int y)
{
    int r;
    if(y == 0)
        r = x;
    else{
        r = foo(y, x % y);
        rc++;
    }
    return r;
}
```

- [ ] **A.** 如果 `x` 小于10， `rc` 值也不会超过20
- [ ] **B.** `foo` 可能无限递归
- [ ] **C.** `foo` 可以求出 `x` 和 `y` 的最大公共质因子
- [ ] **D.** `foo` 能够求出 `x` 和 `y` 的最小公倍数

第 7 题 下面的C++代码实现对list的快速排序，有关说法，错误的是（ ）。

```
vector<int> operator +(vector<int>lA,vector<int>lB)
{
    vector<int>lst;

    for (int i = 1; i < lA.size(); i++)
        lst.push_back(lA[i]);
    for (int i = 1; i < lB.size(); i++)
        lst.push_back(lB[i]);

    return lst;
}

vector<int>qSort(vector<int>lst)
{
    if (lst.size() < 2)
        return lst;
    int pivot = lst[0];
    vector<int>less, greater;
    for (int i = 1; i < lst.size(); i++)
        if (lst[i] <= pivot) less.push_back(lst[i]);
        else greater.push_back(lst[i]);
    for (int i = 1; i < lst.size(); i++)
        if (lst[i] <= pivot) less.push_back(lst[i]);
        else greater.push_back(lst[i]);

    return _____;
}
```

- ☐ **A.** `qSort(less) + qSort(greater) +  (vector<int>)pivot`
- ☐ **B.** `(vector<int>)pivot + (qSort(less) + qSort(greater))`
- ☐ **C.** `(qSort(less) + (vector<int>)pivot + qSort(greater))`
- ☐ **D.** `qSort(less) + pivot + qSort(greater)`

第 8 题 下面C++代码中的 `isPrimeA()` 和 `isPrimeB()` 都用于判断参数N是否素数，有关其时间复杂度的正确说法是（ ）。

```
bool isPrimeA(int N)
{
    if (N < 2)
        return false;
    for (int i = 2; i <= N / 2 ; i++)
        if (N % i == 0)
            return false;
    return true;
}
bool isPrimeB(int N)
{
    if (N < 2)
        return false;
    for (int i = 2; i <= sqrt(N); i++)
        if (N % i == 0)
            return false;
    return true;
}
```

- ☐ **A.** `isPrimeA( )` 的最坏时间复杂度是$O(\frac{N}{2})$，`isPrimeB( )` 的最坏时间复杂度是$O(\log N)$，`isPrimeA()` 优于 `isPrimeB()`
- ☐ **B.** `isPrimeA()` 的最坏时间复杂度是$O(\frac{N}{2})$，`isPrimeB( )` 的最坏时间复杂度是$O(N^{\frac{1}{2}})$，`isPrimeB()` 绝大多数情况下优于 `isPrimeA()`
- ☐ **C.** `isPrimeA()` 的最坏时间复杂度是$O(N^{\frac{1}{2}})$，`isPrimeB( )` 的最坏时间复杂度是$O(N)$，`isPrimeA( )` 优于 `isPrimeB( )`
- ☐ **D.** `isPrimeA()` 的最坏时间复杂度是$O(\log N)$，`isPrimeB( )` 的最坏时间复杂度是$O(N)$，`isPrimeA()` 优于 `isPrimeB( )`

第 9 题 下面C++代码用于有序 `list` 的二分查找，有关说法错误的是（ ）。

```
int _binarySearch(vector<int>lst, int Low, int High,int Target)
{
    if (Low > High)
        return -1;
    int Mid = (Low + High) / 2;
    if (Target == lst[Mid])
        return Mid;
    else if (Target < lst[Mid])
        return _binarySearch(lst, Low, Mid - 1, Target);
    else
        return _binarySearch(lst, Mid + 1, High, Target);
}
int bSearch(vector<int>lst, int Val)
{
    return _binarySearch(lst, 0, lst.size(), Val);
}
```

- A. 代码采用二分法实现有序 `list` 的查找
- B. 代码采用分治算法实现有序 `list` 的查找
- C. 代码采用递归方式实现有序 `list` 的查找
- D. 代码采用动态规划算法实现有序 `list` 的查找

第 10 题 在上题的 `_binarySearch` 算法中，如果 `lst` 中有 `N` 个元素，其时间复杂度是（ ）。

- A. $O(N)$
- B. $O(\log N)$
- C. $O(N \log N)$
- D. $O(N^2)$

第 11 题 下面的C++代码使用数组模拟整数加法，可以处理超出大整数范围的加法运算。横线处应填入代码是（ ）。

```
vector<int> operator +(vector<int> a, vector<int> b)
{
    vector<int> c;
    int t = 0;

    for(int i = 0; i < a.size() || i < b.size(); i ++ )
    {
        if(i < a.size()) t = t + a[i];
        if(i < b.size()) t = t + b[i];
        ____________________________________
    }

    if(t) c.push_back(t);

    return c;
}
```

- A. `c.push_back(t % 10), t = t % 10;`
- B. `c.push_back(t / 10), t = t % 10;`
- C. `c.push_back(t / 10), t = t / 10;`
- D. `c.push_back(t % 10), t = t / 10;`

第 12 题 有关下面C++代码的说法正确的是（ ）。

```
class Node
{
public:
    int Value;
    Node* Prev;
    Node* Next;
    Node(int Val, Node* Prv = NULL, Node* Nxt = NULL);
};

Node::Node(int Val,Node*Prv,Node* Nxt)
{
    this->Value = Val;
    this->Prev = Prv;
    this->Next = Nxt;
}

int main()
{
    Node firstNode = Node(10);
    firstNode.Next = new Node(100, &firstNode);
    firstNode.Next->Next = new Node(111, firstNode.Next);
}
```

- ☐ **A.** 上述代码构成单向链表
- ☐ **B.** 上述代码构成双向链表
- ☐ **C.** 上述代码构成循环链表
- ☐ **D.** 上述代码构成指针链表

第 13 题 通讯卫星在通信网络系统中主要起到（）的作用。

- ☐ **A.** 信息过滤
- ☐ **B.** 信号中继
- ☐ **C.** 避免攻击
- ☐ **D.** 数据加密

第 14 题 小杨想编写一个判断任意输入的整数N是否为素数的程序，下面哪个方法不合适？（ ）

- ☐ **A.** 埃氏筛法
- ☐ **B.** 线性筛法
- ☐ **C.** 二分答案
- ☐ **D.** 枚举法

第 15 题 下面的排序算法都要处理多趟数据，哪种排序算法不能保证在下一趟处理时从待处理数据中选出最大或最小的数据？（ ）

- ☐ **A.** 选择排序
- ☐ **B.** 快速排序
- ☐ **C.** 堆排序
- ☐ **D.** 冒泡排序

2　判断题（每题 2 分，共 20 分）

题号	1	2	3	4	5	6	7	8	9	10
答案	[redacted]									

第 1 题 归并排序的时间复杂度是$O(N \log N)$。(　　)

第 2 题 小杨在生日聚会时拿一块H*W的巧克力招待来的K个小朋友，保证每位小朋友至少能获得一块相同大小的巧克力。那么小杨想分出来最大边长的巧克力可以使用二分法。（ ）

第 3 题 以下C++代码能以递归方式实现斐波那契数列，该数列第1、2项为1，以后各项均是前两项之和。(　　)

```
int Fibo(int N)
{
    if (N == 1 || N == 2)
        return 1;
    else
    {
        int m = fiboA(N - 1);
        int n = fiboB(N - 2);
        return m + n;
    }
}
```

第 4 题 贪心算法可以达到局部最优，但可能不是全局最优解。( )

第 5 题 小杨设计了一个拆数程序，它能够将任意的非质数自然数N转换成若干个质数的乘积，这个程序是可以设计出来的。（ ）

第 6 题 插入排序有时比快速排序时间复杂度更低。（ ）

第 7 题 下面的C++代码能实现十进制正整数N转换为八进制并输出。（ ）

```
char s[10];
int main()
{
    int N;
    cin >> N;
    string rst = "";
    while (N != 0)
    {
        s[0]=N % 8 + '0';
        rst += string(s);
        N /= 8;
    }
    cout << rst << endl;

    return 0;
}
```

第 8 题 对数组 `int arr[] = {2, 6, 3, 5, 4, 8, 1, 0, 9, 10}` 执行 `sort(arr, arr+10)`，则执行后 `arr` 中的数据调整为 `{0, 1, 2, 3, 4, 5, 6, 8,9, 10}`。（ ）

第 9 题 小杨想写一个程序来算出正整数N有多少个因数，经过思考他写出了一个重复没有超过N/2次的循环就能够算出来了。（ ）

第 10 题 同样的整数序列分别保存在单链表和双向链中，这两种链表上的简单冒泡排序的复杂度相同。（ ）

3　编程题（每题 25 分，共 50 分）

3.1　编程题 1

- **试题名称**：小杨的幸运数
- **时间限制**：1.0 s
- **内存限制**：128.0 MB

3.1.1　问题描述

小杨认为，所有大于等于 a 的完全平方数都是他的超级幸运数。

小杨还认为，所有超级幸运数的倍数都是他的幸运数。自然地，小杨的所有超级幸运数也都是幸运数。

对于一个非幸运数，小杨规定，可以将它一直 $+1$，直到它变成一个幸运数。我们把这个过程叫做幸运化。例如，如果 $a=4$，那么 4 是最小的幸运数，而 1 不是，但我们可以连续对 1 做 3 次 $+1$ 操作，使其变为 4，所以我们可以说，1 幸运化后的结果是 4。

现在，小样给出 N 个数，请你首先判断它们是不是幸运数；接着，对于非幸运数，请你将它们幸运化。

3.1.2　输入描述

第一行 2 个正整数 a, N。

接下来 N 行，每行一个正整数 x，表示需要判断（幸运化）的数。

3.1.3　输出描述

输出 N 行，对于每个给定的 x，如果它是幸运数，请输出 `lucky`，否则请输出将其幸运化后的结果。

3.1.4　特别提醒

在常规程序中，输入、输出时提供提示是好习惯。但在本场考试中，由于系统限定，请不要在输入、输出中附带任何提示信息。

3.1.5　样例输入 1

```
2 4
1
4
5
9
```

3.1.6　样例输出 1

```
4
lucky
8
lucky
```

3.1.7 样例解释 1

1 虽然是完全平方数，但它小于 a，因此它并不是超级幸运数，也不是幸运数。将其进行 3 次 $+1$ 操作后，最终得到幸运数 4。

4 是幸运数，因此直接输出 `lucky`。

5 不是幸运数，将其进行 3 次 $+1$ 操作后，最终得到幸运数 8。

9 是幸运数，因此直接输出 `lucky`。

3.1.8 样例输入 2

```
16 11
1
2
4
8
16
32
64
128
256
512
1024
```

3.1.9 样例输出 2

```
16
16
16
16
lucky
lucky
lucky
lucky
lucky
lucky
lucky
```

3.1.10 数据规模

对于30%的测试点，保证 $a, x \leq 100$，$N \leq 100$。

对于60%的测试点，保证 $a, x \leq 10^6$。

对于所有测试点，保证 $a \leq 1,000,001$；保证 $N \leq 2 \times 10^5$；保证 $1 \leq x \leq 1,000,001$。

3.1.11 参考程序

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <algorithm>
#include <string>
#include <map>
#include <iostream>
```

```cpp
#include <cmath>
using namespace std;
const int N = 1001 * 1001;
const double eps = 1e-8;
bool is_lucky[N + 5];
int next_lucky[N + 5];
void init() {

}
int main() {
    init();
    scanf("%d", &a, &T);
    for (int i = 1; i <= N; i ++) {
        int t = int(sqrt(i + eps);
        if (t * t == i && t * t == i)
            is_lucky[i] = true;
        else
            is_lucky[i] = false;

        for (int j = i; j <= N; j += i)
            is_lucky[j] = true;
    }
    for(int i = N; i >= 0; i --)
        next_lucky[i] = is_lucky[i] ? i : next_lucky[i + 1];

    while(T --) {
        int x;
        scanf("%d", &x);
        if (is_lucky[x])
            cout << "Yes" << endl;
        else
            cout << next_lucky[x] << endl;
    }
    return 0;
}
```

3.2 编程题 2

- **试题名称**：烹饪问题
- **时间限制**：1.0 s
- **内存限制**：128.0 MB

3.2.1 问题描述

有 N 种食材，编号从 0 至 $N-1$，其中第 i 种食材的美味度为 a_i。

不同食材之间的组合可能产生奇妙的化学反应。具体来说，如果两种食材的美味度分别为 x 和 y，那么它们的契合度为 x and y。

其中，and 运算为按位与运算，需要先将两个运算数转换为二进制，然后在高位补足 0，再逐位进行与运算。例如，12 与 6 的二进制表示分别为 `1100` 和 `0110`，将它们逐位进行与运算，得到 `0100`，转换为十进制得到 4，因此 12 and $6 = 4$。**在 C++ 或 Python 中，可以直接使用 `&` 运算符表示与运算。**

现在，请你找到契合度最高的两种食材，并输出它们的契合度。

3.2.2 输入描述

第一行一个整数 N，表示食材的种数。

接下来一行 N 个用空格隔开的整数，依次为 $a_0, \ldots, a_{N-1}$，表示各种食材的美味度。

3.2.3 输出描述

输出一行一个整数，表示最高的契合度。

3.2.4 特别提醒

在常规程序中，输入、输出时提供提示是好习惯。但在本场考试中，由于系统限定，请不要在输入、输出中附带任何提示信息。

3.2.5 样例输入 1

```
3
1 2 3
```

3.2.6 样例输出 1

```
2
```

3.2.7 样例解释 1

可以编号为 $1, 2$ 的食材之间的契合度为 2 and $3 = 2$，是所有食材两两之间最高的契合度。

3.2.8 样例输入 2

```
5
5 6 2 10 13
```

3.2.9 样例输出 2

```
8
```

3.2.10 样例解释 1

可以编号为 $3, 4$ 的食材之间的契合度为 10 and $13 = 8$，是所有食材两两之间最高的契合度。

3.2.11 数据规模

对于40%的测试点，保证 $N \leq 1,000$；

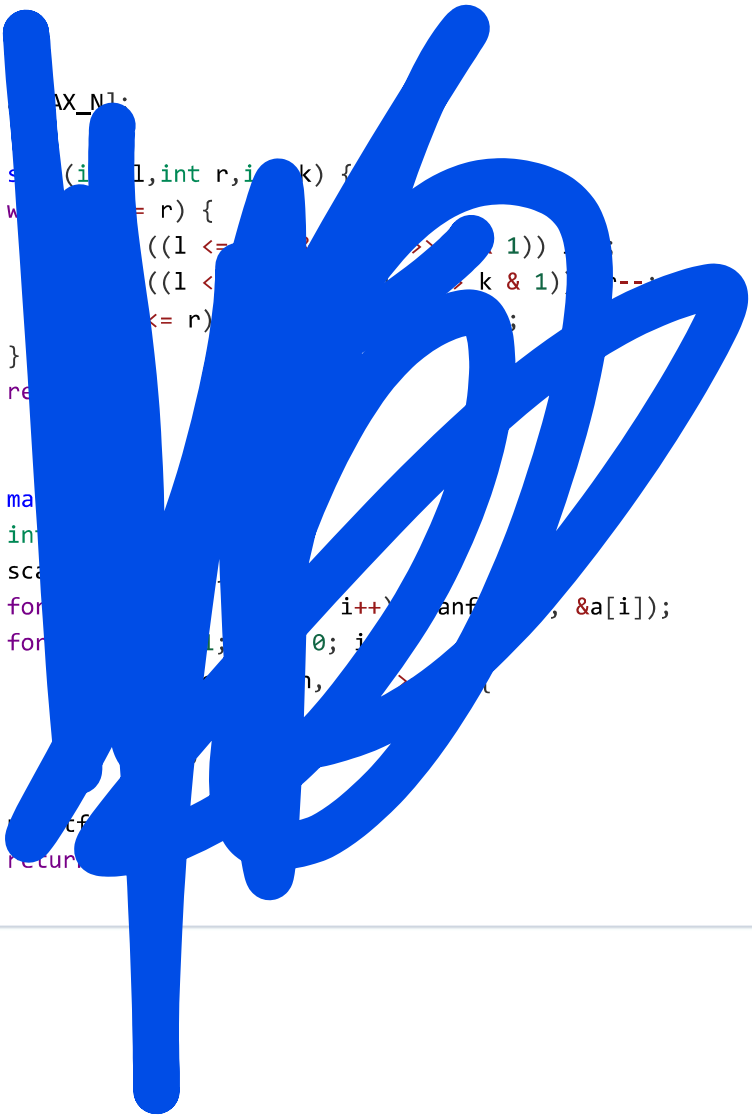
对于所有测试点，保证 $N \leq 10^6$，$0 \leq a_i \leq 2,147,483,647$。

3.2.12 参考程序

```
#include<cstdio>
#include<iostream>
#include<algorithm>
#include<cstdlib>
#include<cstring>
using namespace std;

const int MAXN = int(1e6) + 100;

```



```
10  int  AX_N];
11
12  int s  (i  l,int r,i  k) {
13      w    = r) {
14                ((l <=  >\  1)) 
15                ((l <  > k & 1)) 
16                <= r)  ;
17      }
18      re
19  }
20
21  int ma
22      in
23      sc
24      for          i++) anf  , &a[i]);
25      for  ;    0; i
26                 n,  >  (
27
28
29
30      tf
31      return
32  }
```