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

C++ 五级

2026 年 03 月

1 单选题（每题 2 分，共 30 分）

第 1 题 关于单链表、双链表和循环链表，下列说法正确的是（ ）。

- A. 在单链表中，若已知任意结点的指针，则可以在 $O(1)$ 时间内删除该结点。
- B. 循环链表中一定不存在空指针。
- C. 在循环双链表中，尾结点的 `next` 指针一定为 `nullptr`。
- D. 在带头结点的循环单链表中，判定链表是否为空只需判断头结点的 `next` 是否指向自身。

第 2 题 双向循环链表中要在结点 `p` 之前插入新结点 `s`（均非空），以下指针操作正确的是（ ）。

A.

```
s -> next = p;
p -> prev = s;
p -> next = s;
s -> prev = p;
```

B.

```
s -> prev = p;
s -> next = p -> next;
p -> next -> prev = s;
p -> next = s;
```

C.

```
s -> next = p;
s -> prev = p->prev;
p -> prev -> next = s;
p -> prev = s;
```

D.

```
s -> next = p;
s -> prev = nullptr;
p -> prev = s;
```

第 3 题 下面函数用“哑结点”统一处理删除单向链表中的头结点与中间结点。横线处应填（ ）。

```cpp
struct Node{
    int val;
    Node* next;
    Node(int v):val(v),next(nullptr){}
};

Node* eraseAll(Node* head, int x){
    Node dummy(0);
    dummy.next = head;
    Node* cur = &dummy;
    while(cur->next){
        if(cur->next->val == x){
            Node* del = cur->next;
            ----------------------
            delete del;
        }else cur = cur->next;
    }
    return dummy.next;
}
```

- [ ] **A.**

```cpp
cur = cur->next;
```

- [ ] **B.**

```cpp
cur->next = del->next;
```

- [ ] **C.**

```cpp
del->next = cur->next;
```

- [ ] **D.**

```cpp
cur->next = nullptr;
```

第 4 题 对如下代码实现的欧几里得算法（辗转相除法），执行 `gcd(48, 18)` 得到的调用序列为（ ）。

```cpp
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}
```

- [ ] **A.**

```
gcd(48,18) -> gcd(18,12) -> gcd(12,6) -> gcd(6,0)
```

- [ ] **B.**

```
gcd(48,18) -> gcd(30,18) -> gcd(12,18)
```

- [ ] **C.**

```
gcd(48,18) -> gcd(18,30) -> gcd(30,6)
```

- [ ] **D.**

```
gcd(48,18) -> gcd(12,18) -> gcd(6,12)
```

第 5 题 下面代码实现了欧拉（线性）筛，横线处应填写（ ）。

```cpp
vector<int> euler_sieve(int n) {
    vector<bool> is_composite(n + 1, false);
    vector<int>  primes;

    for (int i = 2; i <= n; i++) {
        if (!is_composite[i])
            primes.push_back(i);

        for (int j = 0; ____________________________ && (long long)i * primes[j] <= n; j++) {
            is_composite[i * primes[j]] = true;

            if (i % primes[j] == 0)
                break;
        }
    }
    return primes;
}
```

- [ ] **A.**

```cpp
j <= n
```

- [ ] **B.**

```cpp
j < sqrt(n)
```

- [ ] **C.**

```cpp
j < primes.size()
```

- [ ] **D.**

```cpp
j < i
```

第 6 题 埃氏筛中将内层循环从 `j = i*i` 开始而不是 `j = 2*i` 的主要原因是（ ）。

```cpp
vector<int> eratosthenes_sieve(int n) {
    vector<bool> is_composite(n + 1, false);
    vector<int>  primes;

    for (int i = 2; i <= n; i++) {
        if (is_composite[i]) continue;

        primes.push_back(i);

        for (long long j = (long long)i * i; j <= n; j += i)
            is_composite[j] = true;
    }
    return primes;
}
```

- [ ] **A.** 因为 `2*i` 一定不是合数
- [ ] **B.** `i*i` 一定是质数
- [ ] **C.** 小于 `i*i` 的 `i` 的倍数已被更小质因子筛过
- [ ] **D.** 这样可以把时间复杂度降为 $O(n)$

第 7 题 下面程序的运行结果为（ ）。

```cpp
bool check(int n, int a[], int k, int dist) {
    int cnt = 1;
    int last = a[0];

    for (int i = 1; i < n; i++) {
        if (a[i] - last >= dist) {
            cnt++;
            last = a[i];
        }
    }

    return cnt >= k;
}

int solve(int n, int a[], int k) {
    std::sort(a, a + n);

    int l = 0;
    int r = a[n - 1] - a[0];

    while (l < r) {
        int mid = (l + r + 1) / 2;

        if (check(n, a, k, mid))
            l = mid;
        else
            r = mid - 1;
    }

    return l;
}

int main() {
    int a[] = {1, 2, 8, 4, 9};
    int n = 5;
    int k = 3;

    std::cout << solve(n, a, k) << std::endl;

    return 0;
}
```

- [ ] **A.** 2
- [ ] **B.** 3
- [ ] **C.** 4
- [ ] **D.** 5

第 8 题 在升序数组中查找第一个大于等于 `x` 的位置，下面循环中横线应填（ ）。

```cpp
int lowerBound(const vector<int>& a, int x){
    int l=0, r=a.size();
    while(l<r){
        int mid = l + (r - l)/2;
        if(a[mid] >= x)  _____________;
        else l = mid + 1;
    }
    return l;
}
```

☐ **A.**

```
r = mid;
```

☐ **B.**

```
r = mid - 1;
```

☐ **C.**

```
l = mid;
```

☐ **D.**

```
l = mid + 1;
```

第 9 题 关于递归函数调用，下列说法错误的是（ ）。

☐ **A.** 递归调用层次过深时，可能会耗尽栈空间导致栈溢出

☐ **B.** 尾递归函数可以通过编译器优化来避免栈溢出

☐ **C.** 所有递归函数都可以通过循环结构来改写，从而避免栈溢出

☐ **D.** 栈溢出发生时，程序会抛出异常并可以继续执行后续代码

第 10 题 给定 `n` 根木头，第 `i` 根长度为 `a[i]`。要切成不少于 `m` 段等长木段，求最大可能长度，则横线上应填写（ ）。

```cpp
const int MAXN = 100005;
long long a[MAXN];
int n, m;

bool check(long long x){
    long long cnt = 0;
    for(int i = 1; i <= n; i++){
        if(x == 0) return true;
        cnt += a[i] / x;
        if(cnt >= m) return true;
    }
    return false;
}

int main(){
    cin >> n >> m;
    long long mx = 0;
    for(int i = 1; i <= n; i++){
        cin >> a[i];
        mx = max(mx, a[i]);
    }

    long long l = 1, r = mx;
    long long ans = 0;

    while(l <= r){
        long long mid = l + (r - l) / 2;

        if(check(mid)){
            ans = mid;
            ----------------------
        }else{
            ----------------------
        }
    }

    cout << ans << endl;
    return 0;
}
```

- [ ] **A.**

```cpp
l = mid + 1;
r = mid - 1;
```

- [ ] **B.**

```cpp
l = mid - 1;
r = mid + 1;
```

- [ ] **C.**

```cpp
l = mid + 1;
r = mid;
```

- [ ] **D.**

```cpp
l = mid;
r = mid + 1;
```

第 11 题 下面代码用分治求“最大连续子段和”，其时间复杂度为（ ）。

```cpp
int solve(vector<int>& a, int l, int r){
    if(l == r) return a[l];

    int mid = l + (r - l) / 2;

    int left = solve(a, l, mid);
    int right = solve(a, mid + 1, r);

    int sum = 0, lmax = INT_MIN;
    for(int i = mid; i >= l; i--){
        sum += a[i];
        lmax = max(lmax, sum);
    }

    sum = 0;
    int rmax = INT_MIN;
    for(int i = mid + 1; i <= r; i++){
        sum += a[i];
        rmax = max(rmax, sum);
    }

    return max({left, right, lmax + rmax});
}
```

- ☐ **A.** $O(n^2)$
- ☐ **B.** $O(n \log n)$
- ☐ **C.** $O(\log n)$
- ☐ **D.** $O(n)$

第 12 题 游戏大赛决赛，两组选手分别按得分从小到大排好队，现在要把他们合并成一个有序排行榜。

A组：`A = {12, 35, 67, 89}`，B组：`B = {20, 45, 55, 78}`，下面是归并合并函数的核心循环，横线处应填入（ ）。

```cpp
int i = 0, j = 0;
vector<int> result;

while (i < A.size() && j < B.size()) {
    if (____________________) {
        result.push_back(A[i++]);
    } else {
        result.push_back(B[j++]);
    }
}

while (i < A.size()) {
    result.push_back(A[i++]);
}

while (j < B.size()) {
    result.push_back(B[j++]);
}
```

- ☐ **A.** `A[i] >= B[j]`
- ☐ **B.** `A[i] <= B[j]`
- ☐ **C.** `i >= j`
- ☐ **D.** `i <= j`

第 13 题 有 n 位同学的成绩已经从小到大排好序，现在对它执行下面这段以第一个元素为 `pivot` 的快速排序，请问此次排序的时间复杂度是（ ）。

```cpp
void quicksort(vector<int>& a, int l, int r) {
    if (l >= r) return;
    int pivot = a[l];
    int i = l, j = r;
    while (i < j) {
        while (i < j && a[j] >= pivot) j--;
        while (i < j && a[i] <= pivot) i++;
        if (i < j) swap(a[i], a[j]);
    }
    swap(a[l], a[i]);
    quicksort(a, l, i - 1);
    quicksort(a, i + 1, r);
}
```

- ☐ **A.** $O(n)$
- ☐ **B.** $O(n \log n)$
- ☐ **C.** $O(n^2)$
- ☐ **D.** $O(\log n)$

第 14 题 下面关于排序算法的描述中，不正确的是( )。

- ☐ **A.** 冒泡排序和插入排序都是稳定的排序算法
- ☐ **B.** 快速排序和归并排序都是不稳定的排序算法
- ☐ **C.** 冒泡排序和插入排序最好时间复杂度均为 $O(n)$
- ☐ **D.** 归并排序在最好、最坏和平均三种情况的时间复杂度均为 $O(n \log n)$

第 15 题 下面代码实现两个整数除法，其中被除数为一个“大整数”，用字符串表示，除数是一个小整数，用 `int` 表示，则横线处应该填写（ ）。

```cpp
int main(){
    string s;
    int b;
    cin >> s >> b;

    vector<int> a;
    for(char c : s){
        a.push_back(c - '0');
    }

    vector<int> c;
    long long rem = 0;

    for(int i = 0; i < a.size(); i++){
        rem = rem * 10 + a[i];
        int q = rem / b;
        c.push_back(q);
        ----------------------
    }

    int pos = 0;
    while(pos < c.size() - 1 && c[pos] == 0) pos++;

    for(int i = pos; i < c.size(); i++){
        cout << c[i];
    }

    cout << endl;
    cout << rem << endl;
    return 0;
}
```

- [ ] **A.**

```cpp
rem /= b;
```

- [ ] **B.**

```cpp
rem %= b;
```

- [ ] **C.**

```cpp
rem = b;
```

- [ ] **D.**

```cpp
rem = q;
```

2　判断题（每题 2 分，共 20 分）

第 1 题 有一个存储了n个整数的线性表，分别用数组和单链表两种方式实现。在已知下标（或结点指针）的前提下，数组的随机访问是 $O(1)$，而在链表中已知某结点的指针时，在该结点之后插入一个新结点的操作也是 $O(1)$。

第 2 题 若数组 `a` 已按升序排列，则下面代码可以正确实现“在 a 中查找第一个大于等于 x 的元素的位置”。

```cpp
int lowerBound(vector<int>& a,int x){
    int l=0, r=a.size();
    while(l < r) {
        int mid = (l + r) / 2;
        if( a[mid] >= x) r = mid;
        else l = mid + 1;
    }
    return l;
}
```

第 3 题 快速排序只要每次都选取中间元素作为枢轴，就一定是稳定排序。

第 4 题 若某算法满足递推式：$T(n) = 2T(n/2) + O(n)$，则其时间复杂度为 $O(n \log n)$。

第 5 题 在一个数组中，如果两个元素 `a[i]` 和 `a[j]` 满足 `i < j` 且 `a[i] > a[j]`，则 `a[i]` 和 `a[j]` 是一个逆序对。
下面代码可以正确统计数组 `a` 区间 `[l,r]` 内的逆序对总数。

```cpp
long long cnt=0;
void merge_count(vector<int>& a, int l, int m, int r){
    int i = l, j = m + 1;
    while(i <= m && j <= r) {
        if(a[i] <= a[j]) i++;
        else {
            cnt += (m -  i+ 1);
            j++;
        }
    }
}
```

第 6 题 根据唯一分解定理，如果大于1的整数不能被任何不超其平方根的质数整除，那么n 必定是质数。

第 7 题 假设数组 a 的值域范围是 D，以下程序的时间复杂度是 $O(n \log n + n \log D)$。

```cpp
bool check(int n, int a[], int k, int dist) {
    int cnt = 1;
    int last = a[0];

    for (int i = 1; i < n; i++) {
        if (a[i] - last >= dist) {
            cnt++;
            last = a[i];
        }
    }

    return cnt >= k;
}

int solve(int n, int a[], int k) {
    std::sort(a, a + n);

    int l = 0;
    int r = a[n - 1] - a[0];

    while (l < r) {
        int mid = (l + r + 1) / 2;

        if (check(n, a, k, mid))
            l = mid;
        else
            r = mid - 1;
    }

    return l;
}

int main() {
    int a[] = {1, 2, 8, 4, 9};
    int n = 5;
    int k = 3;

    std::cout << solve(n, a, k) << std::endl;

    return 0;
}
```

第 8 题 若一个问题满足最优子结构性质，则一定可以用贪心算法得到最优解。

第 9 题 线性筛相比埃氏筛的核心改进在于：埃氏筛中一个合数可能被多个质数重复标记，线性筛通过"每个合数只被其最大质因子筛去"的策略，保证每个合数恰好被标记一次，从而实现 $O(n)$ 的时间复杂度。

第 10 题 任何递归程序都可以改写为等价的非递归程序，但改写后的非递归程序一定需要显式地使用栈来模拟递归调用过程。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：有限不循环小数
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

若 $\frac{1}{a}$ 可化为一个有限的，不循环的小数，则称 a 为**终止数**。

请你求出在 L 到 R 中终止数的数量。

3.1.2 输入格式

输入一行，包含两个整数 L, R。

3.1.3 输出格式

输出一行，包含一个整数，表示 L 到 R 中终止数的数量。

3.1.4 样例

3.1.4.1 输入样例

```
2 11
```

3.1.4.2 输出样例

```
5
```

3.1.5 样例解释

在 $[2, 11]$ 终止数有 2、4、5、8、10。

3.1.6 数据范围

保证 $1 \leq L \leq R \leq 10^6$。

3.2 编程题 2

- **试题名称**：找数
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

给定一个包含 n 个互不相同的正整数的数组 A 与一个包含 m 个互不相同的正整数的数组 B，请你帮忙计算有多少数在数组 A 与 数组 B 中均出现。

3.2.2 输入格式

第一行包含两个整数 n, m。

第二行包含 n 个正整数 $a_1, a_2, \cdots, a_n$ 表示数组 A。

第二行包含 m 个正整数 $b_1, b_2, \cdots, b_m$ 表示数组 B。

3.2.3 输出格式

输出一个整数，表示在数组 A 与 数组 B 中均出现的数的个数。

3.2.4 样例

3.2.4.1 输入样例

```
3 5
4 2 3
3 1 5 4 6
```

3.2.4.2 输出样例

```
2
```

3.2.5 样例解释

样例 1 中，4、3 在数组 A 与 B 中均出现。

3.2.6 数据范围

对于 40% 的数据，保证 $1 \leq n, m \leq 1000$。

对于 100% 的数据，保证 $1 \leq n, m \leq 10^5$，$1 \leq a_i, b_i \leq 10^9$。