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

C++ 五级

2024 年 09 月

1 单选题（每题 2 分，共 30 分）

第 1 题 下面关于链表和数组的描述，错误的是（ ）。

- A. 数组大小固定，链表大小可动态调整。
- B. 数组支持随机访问，链表只能顺序访问。
- C. 存储相同数目的整数，数组比链表所需的内存多。
- D. 数组插入和删除元素效率低，链表插入和删除元素效率高。

第 2 题 通过（ ）操作，能完成在双向循环链表结点 `p` 之后插入结点 `s` 的功能（其中 `next` 域为结点的直接后继，`prev` 域为结点的直接前驱）。

- A. `p->next->prev = s; s->prev = p; p->next = s; s->next = p->next;`
- B. `p->next->prev = s; p->next = s; s->prev = p; s->next = p->next;`
- C. `s->prev = p; s->next = p->next; p->next = s; p->next->prev = s;`
- D. `s->next = p->next; p->next->prev = s; s->prev = p; p->next = s;`

第 3 题 对下面两个函数，说法错误的是（ ）。

```
1  int sumA(int n) {
2      int res = 0;
3      for (int i = 1; i <= n; i++) {
4          res += i;
5      }
6      return res;
7  }
8
9  int sumB(int n) {
10     if (n == 1)
11         return 1;
12     int res = n + sumB(n - 1);
13     return res;
14 }
```

- A. sumA体现了迭代的思想。
- B. SumB采用的是递归方式。

- [ ] **C.** SumB函数比SumA的时间效率更高。
- [ ] **D.** 两个函数的实现的功能相同。

第 4 题 有如下函数 `fun`，则 `fun(20, 12)` 的返回值为（ ）。

```
int fun(int a, int b) {
    if (a % b == 0)
        return b;
    else
        return fun(b, a % b);
}
```

- [ ] **A.** 20
- [ ] **B.** 12
- [ ] **C.** 4
- [ ] **D.** 2

第 5 题 下述代码实现素数表的埃拉托斯特尼筛法，筛选出所有小于等于 `n` 的素数，则横线上应填的最佳代码是( )。

```
void sieve_Eratosthenes(int n) {
    vector<bool> is_prime(n + 1, true);
    vector<int> primes;

    for (int i = 2; i * i <= n; i++) {
        if (is_prime[i]) {
            primes.push_back(i);
            ____________________________ {     // 在此处填入代码
                is_prime[j] = false;
            }
        }
    }

    for (int i = sqrt(n) + 1; i <= n; i++) {
        if (is_prime[i]) {
            primes.push_back(i);
        }
    }

    return primes;
}
```

- [ ] **A.** `for (int j = i; j <= n; j++)`
- [ ] **B.** `for (int j = i * i; j <= n; j++)`
- [ ] **C.** `for (int j = i * i; j <= n; j += i)`
- [ ] **D.** `for (int j = i; j <= n; j += i)`

第 6 题 下述代码实现素数表的线性筛法，筛选出所有小于等于 `n` 的素数，则横线上应填的代码是( )。

```
vector<int> sieve_linear(int n) {
    vector<bool> is_prime(n + 1, true);
```

```
    vector<int> primes;

    for (int i = 2; i <= n / 2; i++) {
        if (is_prime[i])
            primes.push_back(i);
        _____________________________ {     // 在此处填入代码
            is_prime[i * primes[j]] = 0;
            if (i % primes[j] == 0)
                break;
        }
    }

    for (int i = n / 2 + 1; i <= n; i++) {
        if (is_prime[i])
            primes.push_back(i);
    }

    return primes;
}
```

- [ ] **A.** `for (int j = 0; j < primes.size() && i * primes[j] <= n; j++)`
- [ ] **B.** `for (int j = 1; j < primes.size() && i * j <= n; j++)`
- [ ] **C.** `for (int j = 2; j < primes.size() && i * primes[j] <= n; j++)`
- [ ] **D.** 以上都不对

第 7 题 下面函数可以将 `n` 的所有质因数找出来，其时间复杂度是（ ）。

```
#include <iostream>
#include <vector>

vector<int> get_prime_factors(int n) {
    vector<int> factors;

    while (n % 2 == 0) {
        factors.push_back(2);
        n /= 2;
    }

    for (int i = 3; i * i <= n; i += 2) {
        while (n % i == 0) {
            factors.push_back(i);
            n /= i;
        }
    }

    if (n > 2) {
        factors.push_back(n);
    }

    return factors;
}
```

- [ ] **A.** $O(n^2)$

- [ ] **B.** $O(n \log n)$

- [ ] **C.** $O(\sqrt{n} \log n)$

- [ ] **D.** $O(n)$

第 8 题 现在用如下代码来计算x^n（n个x相乘），其时间复杂度为（ ）。

```
double quick_power(double x, unsigned n) {
    if (n == 0) return 1;
    if (n == 1) return x;
    return quick_power(x, n / 2) * quick_power(x, n / 2) * ((n & 1) ? x : 1);
}
```

- [ ] **A.** $O(n)$

- [ ] **B.** $O(n^2)$

- [ ] **C.** $O(\log n)$

- [ ] **D.** $O(n \log n)$

第 9 题 假设快速排序算法的输入是一个长度为n的已排序数组，且该快速排序算法在分治过程总是选择第一个元素作为基准元素。下面选项（ ）描述的是在这种情况下的快速排序行为。

- [ ] **A.** 快速排序对于此类输入的表现最好，因为数组已经排序。

- [ ] **B.** 快速排序对于此类输入的时间复杂度是$O(n \log n)$。

- [ ] **C.** 快速排序对于此类输入的时间复杂度是$O(n^2)$。

- [ ] **D.** 快速排序无法对此类数组进行排序，因为数组已经排序。

第 10 题 考虑以下C++代码实现的归并排序算法：

```
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
```

```
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void merge_sort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        merge_sort(arr, left, mid);
        merge_sort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}
```

对长度为 `n` 的数组 `arr`，挑用函数 `merge_sort(a, 0, n-1)`，在排序过程中 `merge` 函数的递归调用次数大约是（ ）。

- [ ] **A.** $O(1)$
- [ ] **B.** $O(n)$
- [ ] **C.** $O(\log n)$
- [ ] **D.** $O(n \log n)$

第 11 题 现在有 `n` 个人要过河，每只船最多载2人，船的承重为100kg。下列代码中，数组 `weight` 中保存有 `n` 个人的体重（单位为kg），已经按从小到大排好序，代码输出过河所需要的船的数目，采用的思想为（ ）。

```
int i, j;
int count = 0;
for (i = 0, j = n - 1; i < j; j--) {
    if (weight[i] + weight[j] <= 100) {
        i++;
    }
    count++;
}
printf("过河的船数：%d\n", count);
```

- [ ] **A.** 枚举算法
- [ ] **B.** 贪心算法
- [ ] **C.** 迭代算法

- [ ] **D.** 递归算法

第 12 题 关于分治算法，以下哪个说法正确?

- [ ] **A.** 分治算法将问题分成子问题，然后分别解决子问题，最后合并结果。
- [ ] **B.** 归并排序不是分治算法的应用。
- [ ] **C.** 分治算法通常用于解决小规模问题。
- [ ] **D.** 分治算法的时间复杂度总是优于 $O(n\log(n))$。

第 13 题 根据下述二分查找法，在排好序的数组 `1, 3, 6, 9, 17, 31, 39, 52, 61, 79` 中查找数值 `31` ，循环 `while (left <= right)` 执行的次数为（ ）。

```
int binary_search(vector<int>& nums, int target) {
    int left = 0;
    int right = nums.size() - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (nums[mid] == target) {
            return mid;
        }
        else if (nums[mid] < target) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }

    return -1; // 如果找不到目标元素，返回-1
}
```

- [ ] **A.** 1
- [ ] **B.** 2
- [ ] **C.** 3
- [ ] **D.** 4

第 14 题 以下关于高精度运算的说法错误的是( )。

- [ ] **A.** 高精度计算主要是用来处理大整数或需要保留多位小数的运算。
- [ ] **B.** 大整数除以小整数的处理的步骤可以是，将被除数和除数对齐，从左到右逐位尝试将除数乘以某个数，通过减法得到新的被除数，并累加商。
- [ ] **C.** 高精度乘法的运算时间只与参与运算的两个整数中长度较长者的位数有关。
- [ ] **D.** 高精度加法运算的关键在于逐位相加并处理进位。

第 15 题 当$n=7$时，下面函数的返回值为（ ）。

```cpp
int fun(int n) {
    if (n == 1) return 1;
    else if (n >= 5) return n * fun(n - 2);
    else return n * fun(n - 1);
}
```

- [ ] **A.** 105
- [ ] **B.** 840
- [ ] **C.** 210
- [ ] **D.** 420

2　判断题（每题 2 分，共 20 分）

第 1 题 在操作系统中，需要对一组进程进行循环。每个进程被赋予一个时间片，当时间片用完时，CPU将切换到下一个进程。这种循环操作可以通过环形链表来实现。

第 2 题 找出自然数 `n` 以内的所有质数，常用算法有埃拉托斯特尼（埃氏）筛法和线性筛法，其中线性筛法效率更高。

第 3 题 唯一分解定理表明任何一个大于 `1` 的整数都可以唯一地分解为素数之和。

第 4 题 贪心算法通过每一步选择局部最优解，从而一定能获得最优解。

第 5 题 快速排序和归并排序的平均时间复杂度均为$O(n \log n)$，且都是稳定排序。

第 6 题 插入排序的时间复杂度总是比快速排序低。

第 7 题 引入分治策略往往可以提升算法效率。一方面，分治策略减少了操作数量；另一方面，分治后有利于系统的并行优化。

第 8 题 二分查找要求被搜索的序列是有序的，否则无法保证正确性。

第 9 题 在C++语言中，递归的实现方式通常会占用更多的栈空间，可能导致栈溢出。

第 10 题 对于已经定义好的标准数学函数 `sin(x)` ，应用程序中的语句 `y=sin(sin(x));` 是一种递归调用。

3　编程题（每题 25 分，共 50 分）

3.1　编程题 1

- **试题名称**：小杨的武器
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题面描述

小杨有 n 种不同的武器，他对第 i 种武器的初始熟练度为 c_i。

小杨会依次参加 m 场战斗，每场战斗小杨只能且必须选择一种武器使用，假设小杨使用了第 i 种武器参加了第 j 场战斗，战斗前该武器的熟练度为 c_i'，则战斗后小杨对该武器的熟练度会变为 $c_i'+a_j$。需要注意的是，a_j 可能是正数，0 或负数，这意味着小杨参加战斗后对武器的熟练度可能会提高，也可能会不变，还有可能降低。

小杨想请你编写程序帮他计算出如何选择武器才能使得 m 场战斗后，自己对 n 种武器的熟练度的**最大值尽可能大**。

3.1.2 输入格式

第一行包含两个正整数 n,m，含义如题面所示。

第二行包含 n 个正整数 $c_1,c_2,\ldots,c_n$，代表小杨对武器的初始熟练度。

第三行包含 m 个正整数 $a_1,a_2,\ldots,a_m$，代表每场战斗后武器熟练度的变化值。

3.1.3 输出格式

输出一个整数，代表 m 场战斗后小杨对 n 种武器的熟练度的最大值最大是多少。

3.1.4 样例1

```
2 2
9 9
1 -1
```

```
10
```

一种最优的选择方案为，第一场战斗小杨选择第一种武器，第二场战斗小杨选择第二种武器。

子任务编号	数据点占比	n	m
1	20%	$=1$	$\le 10^5$
2	20%	$\le 10^5$	$=2$
3	60%	$\le 10^5$	$\le 10^5$

对于全部数据，保证有 $1\le n,m\le 10^5$，$-10^4\le c_i,a_i\le 10^4$。

3.2 编程题 2

- **试题名称**：挑战怪物
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题面描述

小杨正在和一个怪物战斗，怪物的血量为 h，只有当怪物的血量**恰好**为 0 时小杨才能够成功击败怪物。

小杨有两种攻击怪物的方式：

- 物理攻击。假设当前为小杨第 i 次使用物理攻击，则会对怪物造成 2^{i-1} 点伤害。
- 魔法攻击。小杨选择任意一个质数 x（x 不能超过怪物当前血量），对怪物造成 x 点伤害。由于小杨并不擅长魔法，他只能使用**至多一次**魔法攻击。

小杨想知道自己能否击败怪物，如果能，小杨想知道自己最少需要多少次攻击。

3.2.2 输入格式

第一行包含一个正整数 t，代表测试用例组数。

接下来是 t 组测试用例。对于每组测试用例，第一行包含一个正整数 h，代表怪物血量。

3.2.3 输出格式

对于每组测试用例，如果小杨能够击败怪物，输出一个整数，代表小杨需要的最少攻击次数，如果不能击败怪物，输出 -1。

3.2.4 样例1

```
3
6
188
9999
```

```
2
4
-1
```

对于第一组测试用例，一种可能的最优方案为，小杨先对怪物使用魔法攻击，选择质数 5 造成 5 点伤害，之后对怪物使用第 1 次物理攻击，造成 $2^{1-1}=1$ 点伤害，怪物血量恰好为 0，小杨成功击败怪物。

子任务编号	数据点占比	t	h
1	20%	≤ 5	≤ 10
2	20%	≤ 10	≤ 100
3	60%	≤ 10	$\leq 10^5$

对于全部数据，保证有 $1 \leq t \leq 10, 1 \leq h \leq 10^5$。