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

Python 五级

2026 年 06 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	A	C	A	B	A	B	C	C	C	B	B	B	A	C	A

第 1 题 假设 `head` 不为空，下面是实现单向循环链表在头节点后插入新节点的代码，横线处应填入（ ）。

```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None

def insert_after_head(head, x):
    new_node = Node(x)
    --------------------
```

- [ ] **A.**

```
new_node.next = head.next
head.next = new_node
```

- [ ] **B.**

```
new_node.next = head
head.next = new_node
```

- [ ] **C.**

```
head.next = new_node
new_node.next = head
```

- [ ] **D.**

```
new_node.next = head.next;
head = new_node
```

第 2 题 下面代码遍历并输出一个循环单链表，其中 `head` 指向链表的第一个节点，横线处应填入的是（ ）。

```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None

def printList(head):
    if head is None:
        return
    p = head
    ----------------------
    print()
```

- [ ] **A.**

```
while p is not None:
    print(p.val, end=" ")
    p = p.next
```

- [ ] **B.**

```
while p.next is not None:
    print(p.val, end=" ")
    p = p.next
```

- [ ] **C.**

```
while True:
    print(p.val, end=" ")
    p = p.next
    if p == head:
        break
```

- [ ] **D.**

```
while p:
    print(p.val, end=" ")
    p = p.next
```

第 3 题 双链表结点定义如下，若要删除双链表中的中间结点（非首尾节点） `p`，写法正确的是（ ）。

```
class Node:
    def __init__(self, val):
        self.val = val
        self.prev = None
        self.next = None
```

- [ ] **A.**

```
p.prev.next = p.next
p.next.prev = p.prev
del p
```

- [ ] **B.**

```
p.next.prev = p.next
p.prev.next = p.prev
del p
```

- [ ] **C.**

```
p.prev = p.next
p.next = p.prev
del p
```

- [ ] **D.**

```
p.next.next = p.prev
p.prev.prev = p.next
del p
```

第 4 题 使用如下欧几里得算法求 `gcd(105, 45)` 时，函数 `gcd(a, b)` 的递归调用序列正确的是（ ）。

```
def gcd(a, b):
    return a if b == 0 else gcd(b, a % b)
```

- [ ] **A.** `gcd(105, 45) -> gcd(45, 60) -> gcd(60, 15) -> gcd(15, 0)`
- [ ] **B.** `gcd(105, 45) -> gcd(45, 15) -> gcd(15, 0)`
- [ ] **C.** `gcd(105, 45) -> gcd(60, 45) -> gcd(15, 45)`
- [ ] **D.** `gcd(105, 45) -> gcd(15, 45) -> gcd(15, 0)`

第 5 题 下面代码实现线性筛（欧拉筛），以筛选出 `n` 以内的所有素数。横线处的代码应为（ ）。

```python
def sieve(n):
    is_prime = [True] * (n + 1)
    primes = []

    if n >= 0:
        is_prime[0] = False
    if n >= 1:
        is_prime[1] = False

    for i in range(2, n + 1):
        if is_prime[i]:
            primes.append(i)
        j = 0
        while j < len(primes) and i * primes[j] <= n:
            is_prime[i * primes[j]] = False
            if _________________:  # 在此处填入代码
                break
            j += 1
    return primes
```

- [ ] **A.** `i % primes[j] == 0`
- [ ] **B.** `primes[j] % i == 0`
- [ ] **C.** `i % primes[j] != 0`
- [ ] **D.** `i == primes[j]`

第 6 题 下面关于埃氏筛法的说法正确的是（ ）。

- [ ] **A.** 每个合数只会被筛掉一次
- [ ] **B.** 从每个素数出发，把它的倍数标记为合数
- [ ] **C.** 只能判断一个数是不是偶数
- [ ] **D.** 不能求出素数表

第 7 题 下面代码实现了计算 x^n 的快速幂算法，该算法体现的编程思想是（ ）。

```python
def power(x, n):
    if n == 0:
        return 1
    res = power(x, n // 2)
    if n % 2 == 0:
        return res * res
    else:
        return res * res * x
```

- [ ] **A.** 枚举
- [ ] **B.** 贪心
- [ ] **C.** 分治
- [ ] **D.** 模拟

第 8 题 下面代码用于统计 `n` 中因子 `2` 出现了多少次。若 `n = 40`，输出是（ ）。

```
n = 40
cnt = 0
while n % 2 == 0:
    cnt += 1
    n = n // 2
print(cnt)
```

- A. `1`
- B. `2`
- C. `3`
- D. `4`

第 9 题 在一个有序数组中查找第一个大于或等于 `x` 的元素位置，横线处应填写（ ）。

```
def lowerBound(a, x):
    l = 0
    r = len(a)
    while l < r:
        mid = l + (r - l) // 2
        if a[mid] >= x:
            _____________  # 在此处填入代码
        else:
            l = mid + 1
    return l
```

- A. `r = mid + 1`
- B. `r = mid - 1`
- C. `r = mid`
- D. `l = mid`

第 10 题 有若干根木头，长度存于 `wood`。每切一刀可以把一段木头分成两段。函数 `check(wood, K, x)` 返回：用不超过 `K` 刀，能否使所有木段长度都不超过 `x`。下面代码使用二分答案查找最小可行的 `x`，横线处应填（ ）。

```
def binary_cut(wood, K):
    l = 1
    r = 0
    for length in wood:
        r = max(r, length)
    while l < r:
        mid = l + (r - l) // 2
        if check(wood, K, mid):
            _____________  # 在此处填入代码
        else:
            l = mid + 1
    return l
```

- A. `r = mid + 1`
- B. `r = mid`
- C. `l = mid`
- D. `r = mid - 1`

第 11 题 下面代码段实现了快速排序的划分操作（以首元素为基准），横线处代码应填入（ ）。

```python
def partition(arr, low, high):
    pivot = arr[low]
    i = low
    j = high
    while i < j:
        while i < j and arr[j] >= pivot:
            j -= 1
        while i < j and arr[i] <= pivot:
            i += 1
        if i < j:
            arr[i], arr[j] = arr[j], arr[i]
    _________________  # 在此处填入代码
    return i
```

- [ ] **A.** `arr[low], arr[high] = arr[high], arr[low]`
- [ ] **B.** `arr[low], arr[i] = arr[i], arr[low]`
- [ ] **C.** `arr[i], arr[high] = arr[high], arr[i]`
- [ ] **D.** `arr[i] = pivot`

第 12 题 下面哪句话最符合归并排序的思想？（ ）

- [ ] **A.** 每次选择最小元素放到前面
- [ ] **B.** 将数组分成两半分别排序，再合并两个有序部分
- [ ] **C.** 相邻元素两两交换
- [ ] **D.** 从左到右把元素插入有序区

第 13 题 在对长度为 $n(n \geq 1)$ 的数组进行归并排序的过程中，`mergeArray` 函数（合并两个有序子数组的操作）被调用的次数是（ ）。

```python
MAXN = 100005
a = [0] * MAXN
tempArr = [0] * MAXN

def mergeArray(left, mid, right):
    i = left
    j = mid + 1
    k = left
    while i <= mid and j <= right:
        if a[i] <= a[j]:
            tempArr[k] = a[i]
            i += 1
        else:
            tempArr[k] = a[j]
            j += 1
        k += 1
    while i <= mid:
        tempArr[k] = a[i]
        i += 1
        k += 1
    while j <= right:
        tempArr[k] = a[j]
        j += 1
        k += 1
    for p in range(left, right + 1):
        a[p] = tempArr[p]

def mergeSort(left, right):
    if left >= right:
        return
    mid = left + (right - left) // 2
    mergeSort(left, mid)
    mergeSort(mid + 1, right)
    mergeArray(left, mid, right)
```

- ☐ **A.** $n-1$
- ☐ **B.** $\log n$
- ☐ **C.** $n \log n$
- ☐ **D.** $2n$

第 14 题 小杨在学校义卖会上负责打包“零食盲盒”。每个盲盒重量不同，快递盒最多承重 `limit` 克，每个快递盒最多装两个盲盒。为了尽量少用快递盒，他采用如下策略：

（1）每次把最轻的盲盒和最重的盲盒尝试放在一起；

（2）如果两者重量之和不超过 `limit`，就一起装；

（3）否则，只能让最重的盲盒单独装一盒。

下面代码用于计算最少需要多少个快递盒，则横线处应填入的是（ ）。

```python
def minBoxes(w, limit):
    w.sort()
    l = 0
    r = len(w) - 1
    boxes = 0
    while l <= r:
        if w[l] + w[r] <= limit:
            __________  # 在此处填入代码
        else:
            r -= 1
        boxes += 1
    return boxes
```

- [ ] **A.** `l += 1`
- [ ] **B.** `r -= 1`
- [ ] **C.**

```
l += 1
r -= 1
```

- [ ] **D.** `boxes -= 1`

第 15 题 高精度减法中，假设两个高精度数按低位在前存储，且已经保证被减数不小于减数。下面处理借位逻辑代码中横线处应填入（ ）。

```
if a[i] < b[i]:
    a[i + 1] -= 1
    ----------------
t = a[i] - b[i]
```

- [ ] **A.** `a[i] += 10`
- [ ] **B.** `a[i] -= 10`
- [ ] **C.** `b[i] += 10`
- [ ] **D.** `a[i] = a[i + 1] + 10`

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	√	×	√	√	√	×	×	×	√	×

第 1 题 数组的存储空间在物理上通常是连续的，而链表的结点可以存储在不连续的内存空间中。

第 2 题 带哨兵头尾节点的双向循环链表，在表头插入节点 `p`，以下四步操作无论什么顺序执行结果都正确。

```
# 1
p.next = head.next
# 2
p.prev = head
# 3
head.next.prev = p
# 4
head.next = p
```

第 3 题 对任意正整数 `a`、`b`，以下两种写法的 `gcd` 函数返回值完全相同。

```
def gcd1(a, b):
    return gcd1(b, a % b) if b else a

def gcd2(a, b):
    while b:
        t = b
        b = a % b
        a = t
    return a
```

第 4 题 在归并排序的合并操作中，如下代码片段可以正确地将两个已排序的子数组 `L` 和 `R` 合并回原数组 `arr` 中。

```python
def merge(arr, left, mid, right):
    n1 = mid - left + 1
    n2 = right - mid
    L = [0] * n1
    R = [0] * n2

    for i in range(n1):
        L[i] = arr[left + i]
    for j in range(n2):
        R[j] = arr[mid + 1 + j]

    i = 0
    j = 0
    k = left

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1
```

第 5 题 分治法通常将一个规模较大的问题拆分为若干个规模较小、结构相似的子问题，分别求解后再合并子问题的结果。

第 6 题 贪心算法只要每一步选择当前最优解，就一定能得到全局最优解。

第 7 题 二分查找不仅可以应用于有序数组，也可以在不增加时间复杂度的情况下应用于有序的单链表，因为链表也支持 $O(1)$ 时间内的随机访问。

第 8 题 以下函数 `f1` 的时间复杂度比函数 `f2` 的更高。

```python
def f1(n):
    i = 1
    while i < n:
        i *= 2

def f2(n):
    if n <= 1:
        return
    f2(n - 1)
    f2(n - 1)
```

第 9 题 唯一分解定理表明，任何一个大于 1 的自然数都可以唯一地分解为若干个质数的乘积，如果不考虑质因数的顺序，这种分解方式是唯一的。

第 10 题 归并排序和快速排序在平均情况下的时间复杂度均为 $O(n \log n)$。但在稳定性方面，归并排序通常是不稳定的，而快速排序是稳定的。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

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

3.1.1 题目描述

老师正在和小朋友们分糖果。
小朋友们先在自己的手上写一个数字，然后坐成一排。
老师分发糖果的规则是：每个小朋友获得自己以及左侧所有小朋友的手上数字之和个糖果。

现在小朋友们都已经在自己手上写上了数字。
请帮小朋友们安排合适的座位顺序，使得小朋友们分到的糖果总量最大，输出这个最大值。

3.1.2 输入格式

输入 2 行，
第一行为一个正整数 n，表示小朋友的个数；
第二行为 n 个正整数 $a_1, a_2, \cdots, a_n$，表示小朋友们手上的数字，整数之间以空格分隔。

3.1.3 输出格式

输出一个整数，表示小朋友们可能分到的最大糖果总数量。

3.1.4 样例

3.1.5 输入样例 1

```
5
7 5 8 9 3
```

3.1.6 输出样例 1

```
111
```

3.1.7 样例解释

小朋友安排座位后从左向右每人手上数字依次是：$9, 8, 7, 5, 3$。

这时可以得到最多的糖果：$(9) + (9+8) + (9+8+7) + (9+8+7+5) + (9+8+7+5+3) = 111$。

3.1.8 数据范围

$1 \leq n \leq 1000$，$1 \leq a_i \leq 1000$。

3.1.9 参考程序

```python
n = int(input())
# 将糖果的数目合并在一个数组里面，进行降序排序
a = sorted(map(int, input().split()), reverse=True)
# 总答案
ans = 0
# 当前这个小孩能拿到多少糖果
sum = 0
# 遍历每一个小孩
for x in a:
    # 计算这个小孩能拿多少糖果，sum就是左侧所有小孩的累加
    sum += x
    # 最后的答案，加上当前小孩能拿多少糖果
    ans += sum
# 展示最后的答案
print(ans)
```

3.2 编程题 2

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

3.2.1 题目描述

小明去参加晚宴。晚宴中有 n 个菜肴，每个菜肴都有一个美味度，第 i 个菜肴的美味度为 v_i。

晚宴规定小明只能**恰好**选取两道菜肴，并且这两道菜肴的美味度必须要互质（即最大公约数为 1）。

请帮助小明选取两道菜肴，使得两道菜肴美味度之和最大。

3.2.2 输入格式

输入 2 行，
第一行为一个正整数 n，表示菜肴的个数；
第二行为 n 个整数 $v_1, v_2, \cdots, v_n$ 表示菜肴的美味度，整数之间以空格分隔。

3.2.3 输出格式

输出一个整数，表示两道互质菜肴美味度之和的最大值。

3.2.4 样例

3.2.5 输入样例 1

```
5
3 5 7 35 105
```

3.2.6 输出样例 1

```
38
```

3.2.7 样例解释 1

最优选择是 3 和 35。
注意到，105 与其他任意菜肴的最大公约数都大于 1，因此无法参与合法选择。

3.2.8 数据范围

$2 \leq n \leq 1000, 1 \leq v_i \leq 1000000$。

数据保证不存在相同美味度的菜肴。

数据保证至少存在一种选取两道菜肴的方案。

3.2.9 参考程序

```python
# 辗转相除法
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)

# 菜品数量为n
n = int(input())
# 每一道菜品的美味度
arr = list(map(int, input().split()))
ans = 0
for i in range(n):
    for j in range(i + 1, n):
        if gcd(arr[i], arr[j]) == 1:
            ans = max(ans, arr[i] + arr[j])
assert ans != 0
print(ans)
```