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

Python　五级

2025 年 03 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	A	A	B	D	B	A	D	A	D	C	A	A	D	B

第 1 题 链表不具备的特点是( )。

- [ ] **A.** 可随机访问任何一个元素
- [ ] **B.** 插入、删除操作不需要移动元素
- [ ] **C.** 无需事先估计存储空间大小
- [ ] **D.** 所需存储空间与存储元素个数成正比

第 2 题 双向链表中每个结点有两个指针域 `prev` 和 `next`，分别指向该结点的前驱及后继结点。设 `p` 指向链表中的一个结点，它的前驱结点和后继结点均非空。现要求删除结点 `p`，则下述语句中正确的是（ ）。

- [ ] **A.**

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

if p.next:
    p.next.prev = p.prev
if p.prev:
    p.prev.next = p.next
p = None
```

- [ ] **B.**

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

if p.next:
    p.next.next = p.prev
if p.prev:
    p.prev.next = p.next
p = None
```

C.

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

if p.next:
    p.next.prev = p.prev
if p.prev:
    p.prev.next = p.prev
p = None
```

D.

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

if p.next:
    p.next.prev = p.next
if p.prev:
    p.prev.next = p.next
p = None
```

第 3 题 假设双向循环链表包含头尾哨兵结点(不存储实际内容)，分别为 `head` 和 `tail` ，链表中每个结点有两个指针域 `prev` 和 `next` ，分别指向该结点的前驱及后继结点。下面代码实现了一个空的双向循环链表，横线上应填的最佳代码是( )。

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

class LinkedList:
    def __init__(self):
        self.head = ListNode()
        self.tail = ListNode()
```

```
        ____________________
        ____________________


def init_linked_list():
    return LinkedList()

```

- [ ] **A.**

```
    self.head.next = self.tail
    self.tail.prev = self.head
```

- [ ] **B.**

```
    self.head.next = self.head
    self.tail.prev = self.tail
```

- [ ] **C.**

```
    self.head.next = self.tail
    self.tail.next = self.head
```

- [ ] **D.**

```
    self.head.prev = self.tail
    self.tail.next = self.head
```

第 4 题 用以下辗转相除法（欧几里得算法）求gcd(84, 60)的步骤中，第二次调用gcd()函数计算的数是（ ）。

```
def gcd(a, b):
    big = max(a, b)
    small = min(a, b)
    if big % small == 0:
        return small
    return gcd(small, big % small)
```

- [ ] **A.** 84和60
- [ ] **B.** 60和24
- [ ] **C.** 24和12
- [ ] **D.** 12和0

第 5 题 根据唯一分解定理，下面整数的唯一分解是正确的（ ）。

- [ ] **A.** $18 = 3 \times 6$
- [ ] **B.** $28 = 4 \times 7$
- [ ] **C.** $36 = 2 \times 3 \times 6$
- [ ] **D.** $30 = 2 \times 3 \times 5$

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，横线上应填的最佳代码是( )。

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

    if n < 2:
        return primes  #

    is_prime[0] = is_prime[1] = False

    for i in range(2, n // 2 + 1):
        if is_prime[i]:
            primes.append(i)


        j = 0
        ____________________________________
            is_prime[i * primes[j]] = False
            if i % primes[j] == 0:
                break
            j += 1


    for i in range(n // 2 + 1, n + 1):
        if is_prime[i]:
            primes.append(i)

    return primes
```

- A. `while j < len(primes) and j * primes[j] <= n:`
- B. `while j < len(primes) and i * primes[j] <= n:`
- C. `while j < len(primes) and j * primes[i] <= n:`
- D. `while i < len(primes) and i * primes[j] < n:`

第 7 题 在程序运行过程中，如果递归调用的层数过多，会因为（ ）引发错误。

- A. 系统分配的栈空间溢出
- B. 系统分配的堆空间溢出
- C. 系统分配的队列空间溢出
- D. 系统分配的链表空间溢出

第 8 题 对下面两个函数，说法错误的是（ ）。

```python
def factorialA(n):
    if n <= 1:
        return 1
    return n * factorialA(n - 1)

def factorialB(n):
    if n <= 1:
        return 1
    res = 1
    for i in range(2, n + 1):
        res *= i
    return res
```

- [ ] **A.** 两个函数的实现的功能相同。
- [ ] **B.** 两个函数的时间复杂度均为$O(n)$。
- [ ] **C.** factorialA采用递归方式。
- [ ] **D.** factorialB采用递归方式。

第 9 题 下算法中，（ ）是不稳定的排序。

- [ ] **A.** 选择排序
- [ ] **B.** 插入排序
- [ ] **C.** 归并排序
- [ ] **D.** 冒泡排序

第 10 题 考虑以下python代码实现的快速排序算法，将数据从小到大排序，则横线上应填的最佳代码是( )。

```python
def partition(arr, low, high):
    pivot = arr[high]
    i = low - 1

    for j in range(low, high):
    ____________________________

    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

def quick_sort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quick_sort(arr, low, pi - 1)
        quick_sort(arr, pi + 1, high)
```

- [ ] **A.**

```python
    if arr[i] < pivot:
        i += 1
        arr[i], arr[j] = arr[j], arr[i]
```

- [ ] **B.**

```
    if arr[j] < pivot:
            j += 1
```

C.

```
    if arr[i] < pivot:
            j += 1
            arr[i], arr[i] = arr[j], arr[i]
```

D.

```
     if arr[j] < pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
```

第 11 题 若用二分法在[1, 100]内猜数，最多需要猜（ ）次。

A. 100

B. 10

C. 7

D. 5

第 12 题 下面的python代码实现了二分查找算法，在数组 `arr` 找到目标元素 `target` 的位置，则横线上能填写的最佳代码是（ ）。

```
def binary_search(arr, left, right, target):
    while left <= right:
        ____________________

        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1

```

A.

```
mid = left + (right - left) // 2
```

B.

```
mid = left;
```

C.

```
mid = (left + right) // 2 + 1;
```

```
mid = right;
```

第 13 题 贪心算法的核心特征是（ ）。

- [ ] **A.** 总是选择当前最优解
- [ ] **B.** 回溯尝试所有可能
- [ ] **C.** 分阶段解决子问题
- [ ] **D.** 总能找到最优解

第 14 题 函数 `def find_max(arr, low, high):` 计算数组中最大元素，其中数组 `arr` 从索引 `low` 到 `high`，（ ）正确实现了分治逻辑。

- [ ] **A.**

```python
def find_max(arr, low, high):
    if low = high:
        return arr[low]
    mid = low + (high - low) // 2
    left_max = find_max(arr, low, mid)
    right_max = find_max(arr, mid, high)
    return left_max if left_max > right_max else right_max
```

- [ ] **B.**

```python
def find_max(arr, low, high):
    if low == high:
        return arr[low]
    mid = low + (high - low) // 2
    left_max = find_max(arr, low, mid)
    right_max = find_max(arr, mid, high)
    return left_max if left_max > right_max else right_max
```

- [ ] **C.**

```python
def find_max(arr, low, high):
    if low == high:
        return arr[low]
    mid = low + (high - low) // 2
    left_max = find_max(arr, low, mid)
    right_max = find_max(arr, mid - 1, high)
    return left_max if left_max > right_max else right_max
```

- [ ] **D.**

```python
def find_max(arr, low, high):
    if low == high:
        return arr[low]
    mid = low + (high - low) // 2
    left_max = find_max(arr, low, mid)
    right_max = find_max(arr, mid + 1, high)
    return left_max if left_max > right_max else right_max
```

第 15 题 小杨编写了一个如下的高精度乘法函数，则横线上应填写的代码为（ ）。

```
def multiply(a, b):
    m, n = len(a), len(b)
    c = [0] * (m + n)
    for i in range(m):
        for j in range(n):
            c[i + j] += a[i] * b[j]
    carry = 0
    for k in range(len(c)):
        ____________________
        c[k] = temp % 10
        carry = temp // 10

    while len(c) > 1 and c[-1] == 0:
        c.pop()

    return c
```

A.

```
temp = c[k]
```

B.

```
temp = c[k] + carry
```

C.

```
temp = c[k] - carry
```

D.

```
temp = c[k] * carry
```

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 单链表中删除某个结点 `p` (非尾结点)，但不知道头结点，可行的操作是将 `p` 的值设为 `p.next` 的值，然后删除 `p.next` 。

第 2 题 链表存储线性表时要求内存中可用存储单元地址是连续的。

第 3 题 线性筛相对于埃拉托斯特尼筛法，每个合数只会被它的最小质因数筛去一次，因此效率更高。

第 4 题 贪心算法通过每一步选择当前最优解，从而一定能获得全局最优解。

第 5 题 递归函数必须具有一个终止条件，以防止无限递归。

第 6 题 快速排序算法的时间复杂度与输入是否有序无关，始终稳定为$O(n \log n)$。

第 7 题 归并排序算法的时间复杂度与输入是否有序无关，始终稳定为$O(n \log n)$。

第 8 题 二分查找适用于对无序数组和有序数组的查找。

第 9 题 小杨有10元去超市买东西，每个商品有各自的价格，每种商品只能买1个，小杨的目标是买到最多数量的商品。小杨采用的策略是每次挑价格最低的商品买，这体现了分治思想。

第 10 题 归并排序算法体现了分治算法，每次将大的待排序数组分成大小大致相等的两个小数组，然后分别对两个小数组进行排序，最后对排好序的两个小数组合并成有序数组。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **时间限制**：3.0 s
- **内存限制**：512.0 MB

3.1.1 平均分配

3.1.2 题目描述

小 A 有 $2n$ 件物品，小 B 和小 C 想从小 A 手上买走这些物品。对于第 i 件物品，小 B 会以 b_i 的价格购买，而小 C 会以 c_i 的价格购买。为了平均分配这 $2n$ 件物品，小 A 决定小 B 和小 C 各自只能买走恰好 n 件物品。你能帮小 A 求出他卖出这 $2n$ 件物品所能获得的最大收入吗?

3.1.3 输入格式

第一行，一个正整数 n。

第二行，$2n$ 个整数 $b_1, b_2, \ldots, b_{2n}$。

第三行，$2n$ 个整数 $c_1, c_2, \ldots, c_{2n}$。

3.1.4 输出格式

一行，一个整数，表示答案。

3.1.5 样例

3.1.5.1 输入样例 1

```
3
1 3 5 6 8 10
2 4 6 7 9 11
```

3.1.5.2 输出样例 1

```
36
```

3.1.5.3 输入样例 2

```
2
6 7 9 9
1 2 10 12
```

3.1.5.4 输出样例 2

```
35
```

3.1.6 数据范围

对于 20% 的测试点，保证 $1 \leq n \leq 8$。

对于另外 20% 的测试点，保证 $0 \leq b_i \leq 1$，$0 \leq c_i \leq 1$。

对于所有测试点，保证 $1 \leq n \leq 10^5$，$0 \leq b_i \leq 10^9$，$0 \leq c_i \leq 10^9$。

3.1.7 参考程序

```python
n = int(input())
b = list(map(int, input().split()))
c = list(map(int, input().split()))
d = [c[i] - b[i] for i in range(2 * n)]
print(sum(b) + sum(sorted(d)[n:]))
```

3.2 编程题 2

- **时间限制**：3.0 s
- **内存限制**：512.0 MB

3.2.8 原根判断

3.2.9 题目描述

小 A 知道，对于质数 p 而言，p 的原根 g 是满足以下条件的正整数：

- $1 < g < p$；
- $g^{p-1} \bmod p = 1$；
- 对于任意 $1 \leq i < p-1$ 均有 $g^i \bmod p \neq 1$。

其中 $a \bmod p$ 表示 a 除以 p 的余数。

小 A 现在有一个整数 a，请你帮他判断 a 是不是 p 的原根。

3.2.10 输入格式

第一行，一个正整数 T，表示测试数据组数。

每组测试数据包含一行，两个正整数 a, p。

3.2.11 输出格式

对于每组测试数据，输出一行，如果 a 是 p 的原根则输出 `Yes`，否则输出 `No`。

3.2.12 样例

3.2.12.5 输入样例 1

```
3
3 998244353
5 998244353
7 998244353
```

3.2.12.6 输出样例 1

```
Yes
Yes
No
```

3.2.13 数据范围

对于 40% 的测试点，保证 $3 \leq p \leq 10^3$。

对于所有测试点，保证 $1 \leq T \leq 20$，$3 \leq p \leq 10^9$，$1 < a < p$，p 为质数。

3.2.14 参考程序

```python
def fpw(b, e, p):
    if e == 0:
        return 1
    r = fpw(b, e >> 1, p)
    return r * r * (b if e & 1 else 1) % p

def solve():
    a, p = map(int, input().split())
    phi, v = p - 1, p - 1
    for i in range(2, int(phi ** 0.5) + 2):
        if v % i == 0:
            while v % i == 0:
                v //= i
            if fpw(a, phi // i, p) == 1:
                print("No")
                return
    if v > 1:
        if fpw(a, phi // v, p) == 1:
            print("No")
            return
    print("Yes")

[solve() for _ in range(int(input()))]
```