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

Python　六级（样题）

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	D	D	C	B	D	D	B	B	B	B	C	C	A	D

第 1 题 近年来，线上授课变得普遍，很多有助于改善教学效果的设备也逐渐流行，其中包括比较常用的手写板，那么它属于哪类设备？（ ）。

- ☐ **A.** 输入
- ☐ **B.** 输出
- ☐ **C.** 控制
- ☐ **D.** 记录

第 2 题 以下关于Python语言的描述，错误的是（ ）。

- ☐ **A.** Python提供了常用的数据结构，并支持面向对象编程
- ☐ **B.** Python是解释型语言
- ☐ **C.** Python是一种高级程序设计语言
- ☐ **D.** Python程序在运行前需要预先编译

第 3 题 以下不属于面向对象程序设计语言的是（ ）。

- ☐ **A.** C++
- ☐ **B.** Python
- ☐ **C.** Java
- ☐ **D.** C

第 4 题 下面有关Python类定义的说法，错误的是( )

- ☐ **A.** Python类实例化时，先执行**new**()和**init**()
- ☐ **B.** Python内置函数bool()对于自定义类有效，必须在新定义类中定义**bool**()函数
- ☐ **C.** Python自定义类不能适用于for-in循环
- ☐ **D.** Python自定义类可用**getitem**()魔术方法定义方括号运算符

第 5 题 有关下面Python代码的说法，错误的是 （ ）。

```
class strReverse:
    def __init__(self, strData = ""):
        self.Data = strData
    def __repr__(self):
        return self.Data[::-1]

print(strReverse("ABC"))
```

- ☐ **A.** 最后一行代码将输出CBA
- ☐ **B.** 最后一行代码将不能输出CBA，因为没有定义print()函数
- ☐ **C.** 第3行代码的Data是strReverse类的数据属性
- ☐ **D.** 最后一行代码将自动执行**init**()函数

第 6 题 有关下面Python代码的说法，正确的是 ( )。

```
class Num:
    def __init__(self,val):
        self.Value = val
    def __add__(self,other):
        return self.Value +
other.Value
    def add(self,other):
        return self.Value +
other.Value

a = Num(10)
print(a + Num(20),
Num(20).__add__(a), a.add(Num(20)))
print(a)
```

- ☐ **A.** 在倒数第2行代码中， `a + Num(20)` 将执行正确，而 `Num(20).__add__(a)` 将导致错误
- ☐ **B.** 由于类Num中没有定义加号运算符，所以倒数第2行代码中的 `a + Num(20)` 被执行时将导致错误
- ☐ **C.** 如果将倒数第2行代码中的 `a.add(num(20))` 修改为 `Num(20).add(a)` 将导致错误，因为 `Num(20)` 不是一个对象，而a是类Num的对象
- ☐ **D.** 倒数第1行代码 `print(a)` 将被正确执行，虽然没有定义相关成员函数，或者称之为方法

第 7 题 有关下面Python代码的说法，正确的是 ( )。

```
class manyData:
    def __init__(self, lstData):
        self.__data = lstData
    def push(self, val):
        self.__data.append(val)
        return self
    def pop(self):
        popVal = self.__data[-1]
        self.__data.pop()
        return popVal
    def __len__(self):
        return len(self.__data)

myData = manyData([1,2,3])
myData.push(100)
print(len(myData))
print(myData.peek())
print(myData.pop())
```

- ☐ **A.** manyData类可用于构造队列(queue)数据结构
- ☐ **B.** 在上面代码环境，代码 `myData.__data.append(10)` 可以增加10到 `myData.__data` 之中
- ☐ **C.** len()是Python内置函数，不适用于上面代码环境中的manyData
- ☐ **D.** 异常处理可以用于自定义类，因此manyData类的pop()函数执行可可以增加异常处理代码，否则可能导致异常

第 8 题 有关下面Python代码的说法，错误的是 ( )。

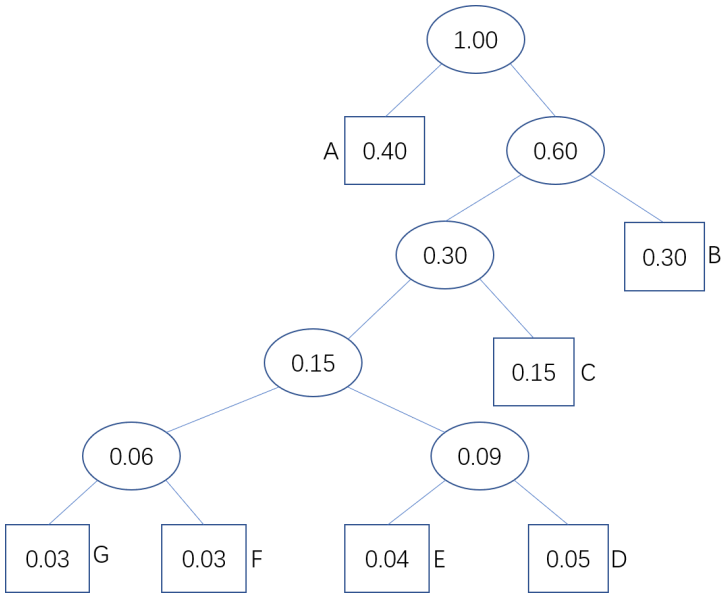
```
class moreData:
    def __init__(self, lstData):
        self.__data = lstData
    def push(self, val):
        self.__data.append(val)
        return self
    def pop(self):
        popVal = self.__data[0]
        self.__data.pop(0)
        return popVal
    def __len__(self):
        return len(self.__data)

myData = moreData([1,2,3])
myData.push(11).push(12).push(13)
print(myData.pop())
```

- ☐ **A.** moreData类可用于构造队列(queue)数据结构
- ☐ **B.** 代码倒数第2行连续push()用法将导致错误

☐ **C.** moreData可以认为是list类型的适配器，裁剪了list功能

☐ **D.** __data可以认为是moreData类的私有成员，只能在类内访问

第 9 题 某内容仅会出现ABCDEFG，其对应的出现概率为0.40、0.30、0.15、0.05、0.04、0.03、0.03，如下图所示。按照哈夫曼编码规则，假设B的编码为11，则D的编码为( )。



☐ **A.** 10010

☐ **B.** 10011

☐ **C.** 10111

☐ **D.** 10001

第 10 题 下面有关格雷码的说法，错误的是 ( )。

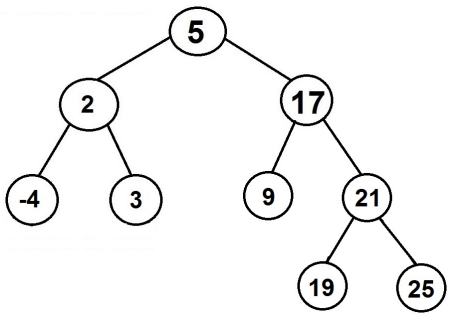
☐ **A.** 在格雷码中，任意两个相邻的代码只有一位二进制数不同

☐ **B.** 格雷码是一种唯一性编码

☐ **C.** 在格雷码中，最大数和最小数只有一位二进制数不同

☐ **D.** 格雷码是一种可靠性编码

第 11 题 有关下图的二叉树，说法正确的是（ ）。



☐ **A.** 既是完全二叉树也是满二叉树

☐ **B.** 既是二叉搜索树也是平衡二叉树

- [ ] **C.** 非平衡二叉树
- [ ] **D.** 以上说法都不正确

第 12 题 N个节点的二叉搜索树，其查找的平均时间复杂度为（ ）。

- [ ] **A.** $O(1)$
- [ ] **B.** $O(N)$
- [ ] **C.** $O(\log N)$
- [ ] **D.** $O(N^2)$

第 13 题 青蛙每次能调1或2步。下面是青蛙跳到第N步台阶Python实现代码。该段代码采用的算法是（ ）。

```
def jumpFrog(N):
    if N <= 3:
        return N
    else:
        return jumpFrog(N - 1) +
jumpFrog(N - 2)
print(jumpFrog(4))
```

- [ ] **A.** 递推算法
- [ ] **B.** 贪心算法
- [ ] **C.** 动态规划算法
- [ ] **D.** 分治算法

第 14 题 Python字典值查找的时间复杂度是（ ）。

- [ ] **A.** $O(1)$
- [ ] **B.** $O(N)$
- [ ] **C.** $O(\log N)$
- [ ] **D.** $O(N^2)$

第 15 题 下面有关Python的in运算符说法错误的是（ ）。

- [ ] **A.** 对于不同的数据类型，in运算符的时间复杂度不同
- [ ] **B.** 对于set和dict类型，in运算符的时间复杂度是$O(1)$
- [ ] **C.** 对于list和tuple类型，in运算符的时间复杂度是$O(N)$
- [ ] **D.** 对于Python的in运算符，其时间复杂度相同

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 TCP/IP的传输层的两个不同的协议分别是UDP和TCP。

第 2 题 5G网络中，5G中的G表示Gigabytes/s，其中 1 GB = 1024 MB。

第 3 题 在面向对象中，类是对象的实例。

第 4 题 在Python类的定义中，可以有类属性和实例属性，类属性值被该类的对象共享。

第 5 题 在Python类的定义中，可以用魔术方法定义初始化函数或运算符函数等。

第 6 题 DFS是深度优先搜索算法的英文简写。

第 7 题 哈夫曼编码是一种有损压缩算法。

第 8 题 Python本身并不支持指针和引用语法，因此有关链表等算法或数据结构在Python中不能实现。

第 9 题 如果节点数为N，广度搜索算法的最差时间复杂度为$O(N)$。

第 10 题 二叉搜索树的左右子树也是二叉搜索树。

3　编程题（每题 25 分，共 50 分）

3.1　编程题 1

- **试题名称**：小杨买饮料
- **时间限制**：1.0 s
- **内存限制**：128.0 MB

3.1.1　问题描述

小杨来到了一家商店，打算购买一些饮料。这家商店总共出售 N 种饮料，编号从 0 至 $N-1$，其中编号为 i 的饮料售价 c_i 元，容量 l_i 毫升。

小杨的需求有如下几点：

1. 小杨想要尽可能尝试不同种类的饮料，因此他希望每种饮料至多购买 1 瓶；
2. 小杨很渴，所以他想要购买总容量不低于 L 的饮料；
3. 小杨勤俭节约，所以在 1 和 2 的前提下，他希望使用尽可能少的费用。

方便起见，你只需要输出最少花费的费用即可。特别地，如果不能满足小杨的要求，则输出 `no solution` 。

3.1.2　输入描述

第一行两个整数 N, L。

接下来 N 行，依次描述第 $i = 0, 1, \ldots, N-1$ 种饮料：每行两个整数 c_i, l_i。

3.1.3 输出描述

输出一行一个整数，表示最少需要花费多少钱，才能满足小杨的要求。特别地，如果不能满足要求，则输出 `no solution`。

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

```
5 100
100 2000
2 50
4 40
5 30
3 20
```

3.1.6 样例输出 1

```
9
```

3.1.7 样例解释 1

小杨可以购买 $1, 2, 4$ 号饮料，总计获得 $50 + 40 + 20 = 110$ 毫升饮料，花费 $2 + 4 + 3 = 9$ 元。

如果只考虑前两项需求，小杨也可以购买 $1, 3, 4$ 号饮料，它们的容量总和为 $50 + 30 + 20 = 100$ 毫升，恰好可以满足需求。但遗憾的是，这个方案需要花费 $2 + 5 + 3 = 10$ 元。

3.1.8 样例输入 2

```
5 141
100 2000
2 50
4 40
5 30
3 20
```

3.1.9 样例输出 2

```
100
```

3.1.10 样例解释 2

$1, 2, 3, 4$ 号饮料总计 140 毫升，如每种饮料至多购买 1 瓶，则恰好无法满足需求，因此只能花费 100 元购买 0 号饮料。

3.1.11 样例输入 3

```
4 141
2 50
4 40
5 30
3 20
```

3.1.12 样例输出 3

```
no solution
```

3.1.13 数据规模

对于 40% 的测试点，保证 $N \leq 20$；$1 \leq L \leq 100$；$l_i \leq 100$。

对于 70% 的测试点，保证 $l_i \leq 100$。

对于所有测试点，保证 $1 \leq N \leq 500$；$1 \leq L \leq 2000$；$1 \leq c_i, l_i \leq 10^6$。

3.1.14 参考程序

```python
n, L = map(int, input().split())
inf = 10**10
dp = [inf for i in range(L)]
dp[0] = 0
ans = inf
for i in range(n):
    c, l = map(int, input().split())
    for j in range(max(L - l, 0), L):
        ans = min(ans, dp[j] + c)
    for j in range(L - 1, l - 1, -1):
        dp[j] = min(dp[j], dp[j - l] + c)

if ans == inf:
    print("no solution")
else:
    print(ans)
```

3.2 编程题 2

- **试题名称**：小杨的握手问题
- **时间限制**：1.0 s
- **内存限制**：128.0 MB

3.2.1 问题描述

小杨的班级里共有 N 名同学，学号从 0 至 $N-1$。

某节课上，老师安排全班同学进行一次握手游戏，具体规则如下：老师安排了一个顺序，让全班 N 名同学依次进入教室。每位同学进入教室时，需要和**已经在教室内**且**学号小于自己**的同学握手。

现在，小杨想知道，整个班级总共会进行多少次握手。

提示：可以考虑使用归并排序进行降序排序，并在此过程中求解。

3.2.2 输入描述

输入包含 2 行。第一行一个整数 N，表示同学的个数；第二行 N 个用单个空格隔开的整数，依次描述同学们进入教室的顺序，每个整数在 $0 \sim N-1$ 之间，表示该同学的学号。

保证每位同学会且只会进入教室一次。

3.2.3 输出描述

输出一行一个整数，表示全班握手的总次数。

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

```
4
2 1 3 0
```

3.2.6 样例输出 1

```
2
```

3.2.7 样例解释 1

2 号同学进入教室，此时教室里没有其他同学。

1 号同学进入教室，此时教室里有 2 号同学。1 号同学的学号小于 2 号同学，因此他们之间不需要握手。

3 号同学进入教室，此时教室里有 $1, 2$ 号同学。3 号同学的学号比他们都大，因此 3 号同学需要分别和另外两位同学握手。

0 号同学进入教室，此时教室里有 $1, 2, 3$ 号同学。0 号同学的学号比他们都小，因此 0 号同学不需要与其他同学握手。

综上所述全班一共握手 $0 + 0 + 2 + 0 = 2$ 次。

3.2.8 样例输入 2

```
6
0 1 2 3 4 5
```

3.2.9 样例输出 2

```
15
```

3.2.10 样例解释 2

全班所有同学之间都会进行握手，因为每位同学来到教室时，都会发现他的学号是当前教室里最大的，所以他需要和教室里的每位其他同学进行握手。

3.2.11 数据规模

对于 30% 的测试点，保证 $N \leq 100$。

对于所有测试点，保证 $2 \leq N \leq 3 \times 10^5$。

3.2.12 参考程序

```python
def merge(arr, l, r):
    if l + 1 == r:
        return 0

    mid = (l + r) >> 1
    ans = merge(arr, l, mid) + merge(arr, mid, r)

    _arr = []
    i, j = l, mid

    for _ in range(l, r):
        if j == r or (i < mid and arr[i] > arr[j]):
            _arr.append(arr[i])
            i += 1
        else:
            _arr.append(arr[j])
            j += 1
            ans += mid - i
    for idx, item in enumerate(_arr):
        arr[l+idx] = item
    #print(arr)
    return ans

n = int(input())
arr = list(map(int, input().split(' ')))
ans = merge(arr, 0, n)
print(ans)
```