2-15-3 对关键字序列(56，23，78，92，88，67，19，34)，进行增量为3的一趟希尔排序的结果为（ ）。

- A. (19，23，56，34，78，67，88，92)
- B. (23，56，78，66，88，92，19，34)
- C. (19，23，34，56，67，78，88，92)
- D. (19，23，67，56，34，78，92，88)

| 参考答案

答案	D

函数题 40 分

6-1 随机题目池 1（1 选 1） 每题 10 分

6-1-1 本题要求实现一个顺序表SeqList的方法，在顺序表SeqList第i个位置（从1开始）前插入元素x。插入成功返回True，否则返回False。

假设线性表长度为n，则插入位置i须满足$1 \leq i \leq n+1$

方法接口定义：

```
#在顺序表SeqList第i个位置（从1开始）前插入元素x。插入成功返回True，否则返回False。
def insert(self,i,x):
```

其中顺序表SeqList的类定义如下：

```
class SqList:
    #0.构造方法
    def __init__(self):
        self.initcapcity = 10             #初始容量
        self.capcity = self.initcapcity   #最大存储
        self.data = [None] * self.capcity#顺序表的数据，列表
        self.size = 0                     #顺序表的长度
    #0.顺序表的最大容量修改为n
    def __resize(self,n):
        assert n >= 0
        #备份原来的数据
        a = self.data
        self.data = [None] * n
        for i in range(self.size):
            self.data[i] = a[i]
    #1.创建顺序表,数据源是列表a
    def create(self,a):
        for i in range(len(a)):
            if self.size == self.capcity:#顺序表满了，2倍扩容
                self.__resize(self.capcity * 2)
            self.data[i] = a[i]
            self.size += 1
    #2.输出顺序表
    def print(self):
        print("the Length of SqList:",self.size)
        print("the Elements of SqList:",*self.data[:self.size])

    #3.你的代码将被嵌在这里，注意整个方法的代码都要缩进4个空格（类里面的方法）
```

裁判测试程序样例：

```
sq = SqList()   #创建顺序表
a = list(map(int,input().split())) #输入数据到列表a
i,x = map(int,input().split())    #输入插入位置i和元素x
sq.create(a)        #根据列表a整体创建顺序表sq
if sq.insert(i,x):   #位置i前插入元素x成功
    print("Insert Success")
    sq.print()
else:               #插入失败
    print("Insert Fail,Index is Error!")
```

输入样例1：

输入共有2行，第1行表示顺序表的元素，第2行表示插入的位置和元素

```
2 6 4
1 3
```

输出样例1：

插入成功，按样例格式输出。

```
Insert Success
the Length of SqList: 4
the Elements of SqList: 3 2 6 4
```

输入样例2：

输入共有2行，第1行表示顺序表的元素，第2行表示插入的位置和元素

```
2 6 4
0 8
```

输出样例2：

插入失败，按样例格式输出。

```
Insert Fail,Index is Error!
```


| 教师提交

代码　　编译器: PYTHON3

```
#3.在序号i前插入x
def insert(self,i,x):
    #异常情况
    if i <= 0 or i > self.size + 1:
        return False
    if self.size == self.capcity:
        self.__resize(2 * self.capcity)
    i -= 1
    #序号i开始依次依次后移，从后到前
    for j in range(self.size,i,-1):
        self.data[j] = self.data[j - 1]
    self.data[i] = x
    self.size += 1
    return True
```

| 参考答案

答案　　编译器: PYTHON3

```
#3.在序号i前插入x
def insert(self,i,x):
    #异常情况
    if i <= 0 or i > self.size + 1:
        return False
    if self.size == self.capcity:
        self.__resize(2 * self.capcity)
    i -= 1
    #序号i开始依次依次后移，从后到前
    for j in range(self.size,i,-1):
        self.data[j] = self.data[j - 1]
    self.data[i] = x
    self.size += 1
    return True
```

6-2 随机题目池 2（1 选 1）　　每题 10 分

6-2-1 本题要求实现一个链表类LinkList的方法，求链表（含有带头结点）的长度。

方法接口定义：

```
#求链表的长度，空表返回0
def length(self):
```

其中链表结点类LinkNode和链表类LinkList的定义如下：

```
class LinkNode:
    def __init__(self,x = None):
        self.data = x        #数据域
        self.next = None     #指针域初始化为空
#单链表类
class LinkList:
    #0.构造方法
    def __init__(self):
        self.head = LinkNode()  #创建一个带头结点
    #1.输出单链表
    def print(self):
        p = self.head.next  #p指向链表的第一个有效结点
        if p is None:
            print("空链表")
        else:
            while p.next is not None:
                print(p.data,"->",sep = "",end = "")
                p = p.next
            print(p.data)
    #2.创建单链表-尾插法
    def createTail(self,a):
        tail = self.head   #p指向链表的带头结点
        #遍历列表a
        for x in a:
            #根据x创建新结点
            p = LinkNode(x)
            #新结点p链接到链表的尾部
            tail.next = p
            tail = p
    #3.创建单链表-头插法
```

```
    def createHead(self,a):
        for x in a:
            p = LinkNode(x)
            p.next = self.head.next
            self.head.next = p
    #4.你的代码将被嵌在这里，注意整个方法的代码都要缩进4个空格（类里面的方法）
```

裁判测试程序样例:

```
h = LinkList()   #创建链表
a = list(map(int,input().split())) #输入数据到列表a
h.createTail(a)     #尾插法创建单链表
print(h.length())   #输出链表的长度
```

输入样例:

输入共有1行，数据之间用空格分隔

```
2 0 2 3 0 5 0 4
```

输出样例:

输出链表的长度

```
8
```

| 教师提交

代码　　编译器: PYTHON3

```
    def length(self):
        p = self.head
        n = 0
        while p.next is not None:
            n += 1
            p = p.next
        return n
```

| 参考答案

答案　　编译器: PYTHON3

```
    #3.你的代码将被嵌在这里，注意整个方法的代码都要缩进4个空格（类里面的方法）
    def length(self):
        p = self.head
        n = 0
        while p.next is not None:
            n += 1
            p = p.next
        return n
```


6-3 随机题目池 3（1 选 1）　　每题 10 分

6-3-1 本题要求计算二叉树中有多少片树叶，输出格式见样例。

函数接口定义:

```
#树叶统计
def leafCount(T)
```

其中二叉树类的定义如下：

```
#二叉树的存储-二叉链表
class BinaryTree:
    #1.构造方法
    def __init__(self,newValue):
        self.key = newValue      #树根
        self.left = None         #左子树初始化为空
        self.right = None        #右子树初始化为空
    #2.访问左子树
    def getLeft(self):
        return self.left
    #3.访问右子树
    def getRight(self):
        return self.right
    #4.修改树根的值
    def setRoot(self,newValue):
        self.key = newValue
    #5.访问树根的值
    def getRoot(self):
        return self.key
```

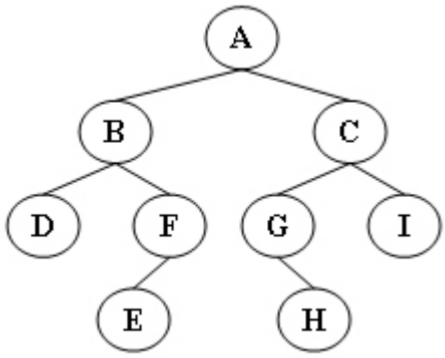
裁判测试程序样例：

```
T = createBT()   #创建二叉树，实现细节不表
print(leafCount(T))

#你的代码将被嵌在这里
```

输入样例：

```
ABDFECGHI
DBEFAGHCI
```

输出样例（对于图中给出的树）：




```
4
```

| 教师提交

代码	编译器: PYTHON3

```
def leafCount(T):
    if not T:
        return 0
    x = 0
    if not T.getLeft() and not T.getRight():
        x += 1
    return x + leafCount(T.getLeft())+ leafCount(T.getRight())
```

| 参考答案

答案　　编译器: PYTHON3

```
def leafCount(T):
    if not T:
        return 0
    x = 0
    if not T.getLeft() and not T.getRight():
        x += 1
    return x + leafCount(T.getLeft())+ leafCount(T.getRight())
```

6-4 随机题目池 4（1 选 1）　　每题 10 分

6-4-1 本题要求实现一个函数，试实现邻接矩阵存储图的广度优先遍历。

函数接口定义：

```
#顶点v(编号)出发对图G进行广度优先遍历
def bfs(G,v)
```

其中图G的定义如下：

```
class adjMatrixGraph:
    # 构造方法，n个顶点m条边
    def __init__(self,n,m):
        self.verNum = n     #顶点数
        self.edgeNum = m    #边数
        self.vertex = [0] * n       #顶点列表
        self.edge = [[0 for i in range(self.verNum)] \
                       for j in range(self.verNum)]   #邻接矩阵二维列表
        self.vis = [False] * n     #顶点的访问列表，默认没访问过
    def addVertex(self,ls): #添加顶点列表
        self.vertex = ls
    def addEdge(self,fr,to):#添加边(fr,to)
        ifr = self.vertex.index(fr)     #起点下标
        ito = self.vertex.index(to)     #终点下标
        self.edge[ifr][ito] = self.edge[ito][ifr] = 1 #邻接矩阵

#邻接矩阵建图
def createGraph():
    n,m = map(int,input().split())  #输入n个顶点和m条边
    g = adjMatrixGraph(n,m)         #创建无向图G
    g.addVertex(list(input().split()))  #输入顶点列表
    for i in range(m):              #输入m条边
        fr,to = input().split()
        g.addEdge(fr,to)
    return g                        #返回无向图g

#定义抽象类型队列Queue，FIFO (First In,First Out)
class Queue:
```

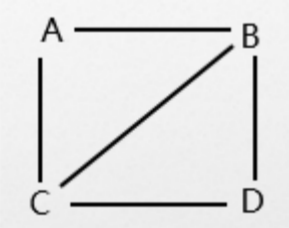
```
    #1.构造方法,定义一个空的列表
    def __init__(self):
        self.items = []
    #2.入队,队尾（列表尾部）入队
    def push(self,item):
        self.items.append(item)
    #3.出队，队首（列表头部）出队
    def pop(self):
        return self.items.pop(0)
    #4.判断队列是否为空
    def isEmpty(self):
        return self.items == []
    #5.取队首
    def getFront(self):
        return self.items[0]
    #6.求队列大小
    def getSize(self):
        return len(self.items)

#你的代码将被嵌在这里
```

裁判测试程序样例:

```
g = createGraph()   #创建无向图g
v = int(input())    #输入出发顶点的编号
print("BFS from " + g.vertex[v] + " :",end = "")
bfs(g,v)            #顶点v(编号)出发对图G进行广度优先遍历
```

输入样例:

例如无向图



第一行给出图的顶点数n和边数m。第二行给出n个顶点字符串（中间用1个空格分隔），表示n个顶点的数据元素的值。后面是m行，给出每一条边的两个顶点（中间用1个空格分隔）。最后一行输入出发顶点的编号。

```
4 5
A B C D
A B
A C
B C
B D
C D
0
```

输出样例:

对于给定的无向图，输出顶点v(编号)出发进行广度优先遍历的顶点（每个顶点前都有1个空格），格式参照样例。

```
BFS from A : A B C D
```

| 教师提交

代码　　编译器: PYTHON3

```
#顶点v(编号)出发对图g进行广度优先遍历
def bfs(g,v):
    q = Queue()    #广搜用队列
    #1.出发点入队标记为已访问
    q.push(v)
    g.vis[v] = True
    #2.队列循环
    while not q.isEmpty():
        #2.1出队并输出
        x = q.pop()
        print(f" {g.vertex[x]}",end = "")
        #2.2搜索相邻的顶点（有边并且没访问过）
        for i in range(g.verNum):
            if g.edge[x][i] == 1 and not g.vis[i]:
                #2.3入队并且标记为已访问
                q.push(i)
                g.vis[i] = True
```

| 参考答案

答案　　编译器: PYTHON3

```
# 广度优先遍历，从顶点st（下标）出发搜到的连通分量
def bfs(G, st):
    #0.初始化，定义队列，顶点的访问列表
    q = Queue()
    vis = [False] * G.verNum
    #1.出发点入队并标记访问过
    q.push(st)
    vis[st] = True
    #2.只要队列非空，进入队列循环
    while not q.isEmpty():
        #2.1出队并输出当前顶点vt
        vt = q.pop()
        print(" " + str(G.vertex[vt]),end = "")
        #2.2遍历邻接矩阵vt行的每个邻接点i
        for i in range(G.verNum):
            #vt的邻接点i只要没访问过就入队
            if G.edge[vt][i] == 1 and not vis[i]:
                q.push(i)
                vis[i] = True
```


编程题　　20 分

7-1 随机题目池 1（1 选 1）　　每题 10 分

7-1-1 给定一串字符，不超过100个字符，可能包括括号、数字、字母、标点符号、空格，编程检查这一串字符中的( ) ,[ ],{ }是否匹配。

输入格式:

输入在一行中给出一行字符串，不超过100个字符，可能包括括号、数字、字母、标点符号、空格。

输出格式:

如果括号配对，输出yes，否则输出no。

输入样例1:

```
sin(10+20)
```

输出样例1:

```
yes
```

输入样例2:

```
{[}]
```

输出样例2:

```
no
```

| 教师提交

代码　　编译器: PYTHON3

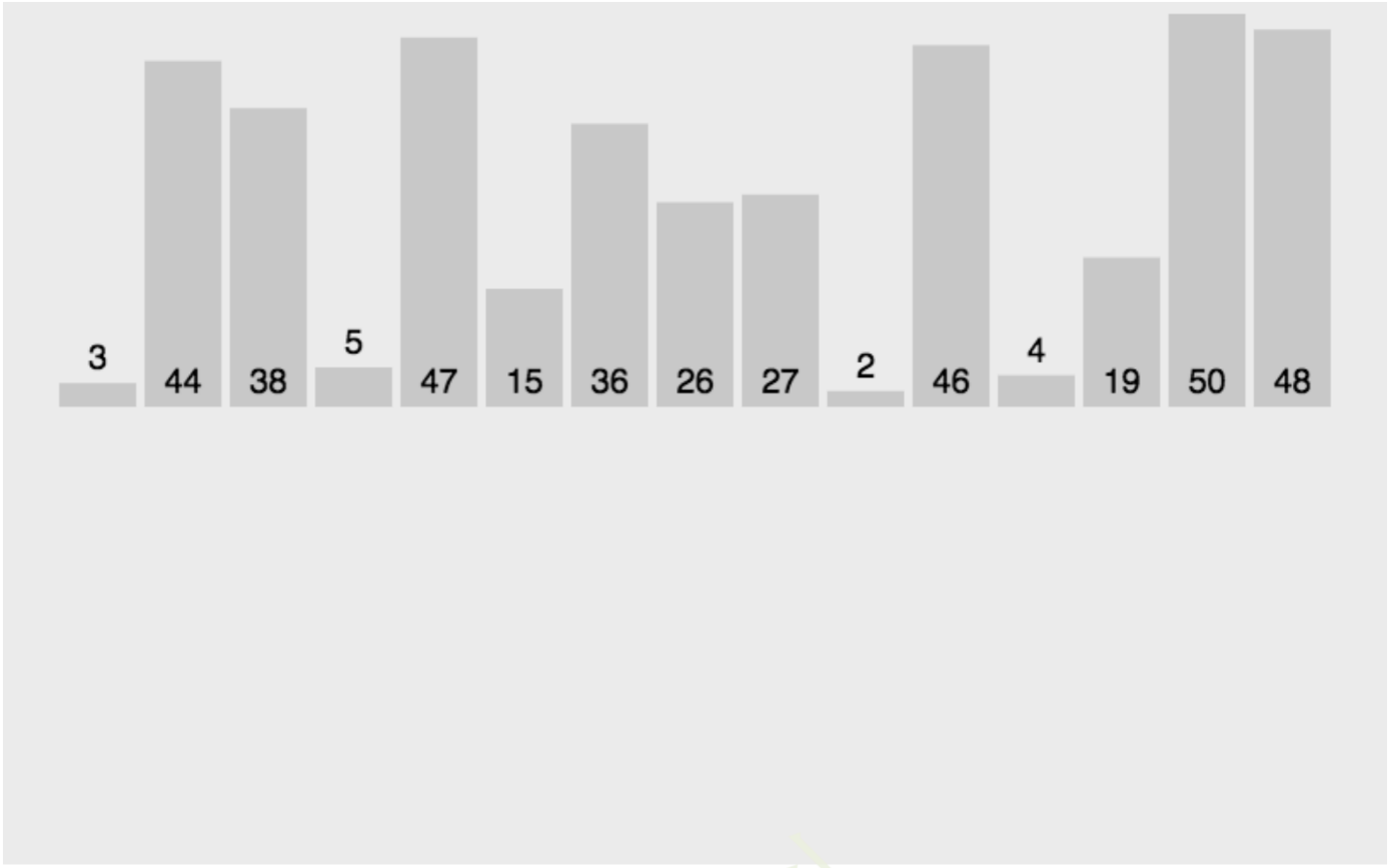
```
#括号匹配
from collections import deque
def isValid(s):
    dic = {"(":")","[":"]","{":"}"}
    st = deque()
    for c in s:
        if c in "([{":
            st.append(c)
        elif c in ")]}":
            if len(st) == 0
                return False
            t = st.pop()
            if dic[t] != c:    #错配
                return False
    return len(st) == 0
s = input()
print("yes" if isValid(s) else "no")
```


| 参考答案

答案　　编译器: NO_COMPILER

7-2 随机题目池 2（1 选 1）　　每题 10 分

7-2-1 插入排序是一种简单直观的排序算法。它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。



输入格式:

输入在第1行中给出N（1<N≤100），在第2行中给出N个待排序的整数，数字间用1个空格分隔。

输出格式:

输出插入排序每一遍后的中间结果序列，数字间用1个空格分隔，但末尾不得有多余空格。

注意，输出时已排序的数据用一对方括号包围，具体格式参见输出样例。

输入样例:

```
7
4 5 7 6 3 2 1
```

输出样例:

```
[4 5] 7 6 3 2 1
[4 5 7] 6 3 2 1
[4 5 6 7] 3 2 1
[3 4 5 6 7] 2 1
[2 3 4 5 6 7] 1
[1 2 3 4 5 6 7]
```

| 教师提交

代码　　编译器: PYTHON3

```
#插入排序，对列表a进行递增的直接插入排序
def insertSort(a):
    n = len(a)      #排序的数据
    if n == 1:
        print("[" + str(a[0]) + "]")
```

```
    #1.从a[1]到a[n-1]进行n-1趟插入排序
    for i in range(1,n):
        t = a[i]    #插入的数a[i]暂存
        #从后往前找到t的插入位置j[0,i]
        j = i - 1
        while j >= 0 and t < a[j]:
            a[j + 1] = a[j]    #a[j]后移
            j -= 1
        a[j + 1] = t    #位置j的后面插入a[i]
        #输出本趟插入排序的中间结果
        print("[",end ="")
        for j in range(i):
            print(a[j],end = " ")
        print(str(a[i]) + "]",*a[i + 1:])
#主程序
n = int(input())
a = list(map(int,input().split()))
#a = [49,38,97,76,65,13,27,50]                          #测试数据1，课堂演示案例
#a = [3,44,38,5,47,15,36,26,27,2,46,4,19,50,48] #测试数据2，插入排序动画
insertSort(a)
```

参考答案

答案　　编译器: PYTHON3

```
#插入排序，对列表a进行递增的直接插入排序
def insertSort(a):
    n = len(a)      #排序的数据
    if n == 1:
        print("[" + str(a[0]) + "]")
    #1.从a[1]到a[n-1]进行n-1趟插入排序
    for i in range(1,n):
        t = a[i]    #插入的数a[i]暂存
        #从后往前找到t的插入位置j[0,i]
        j = i - 1
        while j >= 0 and t < a[j]:
            a[j + 1] = a[j]    #a[j]后移
            j -= 1
        a[j + 1] = t    #位置j的后面插入a[i]
        #输出本趟插入排序的中间结果
        print("[",end ="")
        for j in range(i):
            print(a[j],end = " ")
        print(str(a[i]) + "]",*a[i + 1:])
#主程序
n = int(input())
a = list(map(int,input().split()))
#a = [49,38,97,76,65,13,27,50]                          #测试数据1，课堂演示案例
#a = [3,44,38,5,47,15,36,26,27,2,46,4,19,50,48] #测试数据2，插入排序动画
insertSort(a)
```