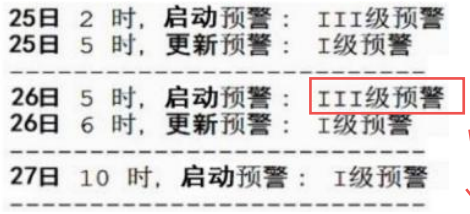
高三下信息算法综合练习 2

1.在某智能环境监测系统中，增加特定环境的温度监测功能，每小时采集一次温度数据。数据格式记为[tm,tp]，其中tm，tp分表为时刻与温度，tm的数据范围为0～23，tp的数据范围为5～45。例如[12,36]表示12点时，检测到的温度为36℃。编写程序，获取当前时刻温度，参照预警标准，进行**启动预警**;若温度达到更高一级标准或同等级则进行**更新预警**。



各级预警标准如下：
I级预警：24小时内当气温升至40°C及以上；
II级预警：24小时内当气温升至37°C及以上；
III级预警：24小时内当气温升至35°C及以上。
请回答下列问题：

（1）题图中预警的各时刻数据如下:[2,35],[5,41],[5,x],[6,40],[3,29],[19,41]，则 x 的值可能是__B__（单选，填字母）。A.30　　B.36　　C.38　　D.41

（2）实现上述功能的部分Python程序如下，请在划线处填入合适的代码。

```
limit = [40,37,35]
pre = -1      # 存储前一次预警的数据，初始化状态为-1
while True :
    # 采集一次数据，存入列表d,d[0]存储时刻，d[1]存储温度，代码略
    if d[0] == 0: # 每日0时初始化
        ① pre=-1
    for i in range(3):
        if d[1] >= limit[i]:
            if ② pre<limit[i] :
                if pre == -1:
                    # 启动预警信息，代码略
                else:
                    # 更新预警信息，代码略
                pre = d[1]
            elif i>0 and pre<limit[i-1] or i==0: #同级别时
                # 更新预警信息，代码略
                ③pre=d[1]
            break
    # 延时1小时，代码略
```

当前温度超过三个预警等级中的任何一个

此时需要查看此前的预警情况，如果之前大于当前的，不需要启动或更新预警

pre>=limit[i]时，看pre是否大于limit[i-1],如果是，说明pre温度更高，不需要更新，如果是同级别更新

2.小明设计了一个蔬菜大棚自动浇水系统。该系统利用传感器每天早、中、晚三次进行土壤湿度检测，若连续三次湿度值均低于阈值 s，则打开阀门浇水。浇水时间根据土壤平均湿度值 ave 决定：ave<100 为重度干旱，浇水 10 分钟；100≤ave<200 为中度干旱，浇水 8 分钟；200≤ave<300 为轻度干旱，浇水 5 分钟。请回答下列问题。

（1）若土壤湿度传感器早上采集的数据是 190，中午采集的数据是 180，晚上采集的数据是 160，则应浇水的时间是：__8__。

（2）实现上述功能的部分 Python 程序如下,请在划线处填入合适的代码。

```
cnt=0;ave=0
while True:
    hum=float(request.args.get("hum1")) #获取湿度传感器的数值
    #将土壤湿度值存入数据库，并从数据库中读取湿度阈值存入变量 s 中，代码略
    if hum<s:
        cnt+=1
        ① ave+=hum
    else:
        cnt=0
        ave=0
```

hum>=s时,湿度低于阈值的情况被打断

```
if cnt==3:
    cnt=0
    sta=1 #打开浇水系统
    ave=ave/3
    if ave<100:
        t=10 #t 为浇水时间，单位是分钟
    elif ②  100<=ave<200          :
        t=8
    elif 200<=ave<300:
        t=5
else:
    ③ sta=0
#延时 8 小时,代码略
```

3.有 n 个作业要在由两台机器 M1 和 M2 组成的流水线上完成加工。每个作业加工的顺序都必须先在 M1 上加工完成后，才能在 M2 上加工。要求确定这 n 个作业的最优加工顺序，使得从第一个作业在机器 M1 上开始加工，到最后一个作业在机器 M2 上加工完成所需的最少时间。某批次作业数据如题表 a 所示：

作业编号	M1 上加工的时间 t1	M2 上加工的时间 t2
1	5	6
2	12	2
3	4	14
4	8	7

表 a

作业编号	M1 上加工的时间 t1	M2 上加工的时间 t2
2	12	2
3	4	14
1	5	6
4	8	7

表 b

加工方法为：每个作业取 t1 和 t2 的较小值，按此最小值进行升序排序，得到如题表 b 所示的数据。

若 t1 <= t2，则在 M1 上优先加工，且优先加工 t1 时间短的作业，得出前 2 个加工的作业为作业 3 和作业 1；若 t1 > t2，则在 M1 上优先加工 t2 时间长的作业，得出后 2 个加工的作业为作业 4 和作业 2。最终所有作业的加工顺序为 3、1、4、2。 M1:4+5+8+12 M2:4+14+6+7+2

编写 Python 程序实现：输入某批次作业数据，按上述方法，计算加工完成所有作业的最优顺序和最少时间,请回答下列问题：

（1）如题表 a 所示的作业数据，加工完成所有作业的最少时间为 33 。

（2）定义如下 bubble_sort(lst)函数，参数 lst 的每个元素由作业编号、M1 上加工的时间 t1、M2 上加工的时间 t2 三个数据项组成。函数的功能是根据 t1 和 t2 的最小值，对 lst 进行升序排序。

```
def bubble_sort(lst):
    n = len(lst)
    for i in range(n-1):
        for j in range(n-1,i,-1):
            a = min(lst[j][1],lst[j][2])
            b = min(lst[j-1][1],lst[j-1][2])
            if a < b:
                lst[j],lst[j-1] = lst[j-1],lst[j]
    return
```

调用该函数，若 lst 为[[1,5,6],[2,12,2],[3,4,14],[4,8,7]]，则加框处的代码执行的次数 5 2 4 7 求交换次数，逆序法 2 。

（3）实现该功能的部分 Python 程序如下，请在划线处填入合适的代码。

```
def sequence(lst):
    n = len(lst)
    order = [0] * n
    i = 0; j = n-1
```

```
    for k in range(n):
        if ① lst[k][1] <= lst[k][2] :      t1<=t2时
            order[i] = lst[k][0]
            i += 1
        else:                    t1>t2时，t2越小越迟处理，即越放后面，j从尾部开始
            order[j] = lst[k][0]
            ②  j-=1
    return order
def process(task):
    bubble_sort(task)
    order = sequence(task)
    n = len(task)
    t = 0 #流水线 M1 需要的总时间
    total = 0 #完成所有任务需要的总时间
    for bh in order:
        for k in range(n):
            if task[k][0] == bh:
                t += task[k][1]t是在M1机器上完成的时间
                x = max(t,total)   total是前一个任务在M2机器上完工时间
                ③total=x+task[k][2]
    return total
'''读取某批次作业 1～n 的信息，依次存入列表 data 的 data[0]至 data[n-1]中。
data[i]包含 3 个数据项，data[i][0],data[i][1],data[i][2]分别存放作业编号、M1 上加工的时间 t1、M2
上加工的时间 t2，代码略'''
ans = process(data)
#输出加工完成所有作业的最优顺序和最少时间，代码略
```

4. 某图书馆需要存储一组按升序排列的图书编号。为提高查找、插入与删除等操作的效率，决定采用“跳跃表”结构进行存储。跳跃表是一种多层链表，其中每一层都是下一层的子集。借助高层链表直接跳过部分底层节点，实现对数据的快速访问。

■ 如给定以下初始数据（已按编号升序存入列表 books=[101, 103, 105, 107, 109, 112, 115, 118, 120, 122]），跳跃表的结构设计如下，假设层 i 的跳跃步长为 2^i：

层 2: 101 → 109 → 120
↓ ↓ ↓
层 1: 101 → 105 → 109 → 115 → 120
↓ ↓ ↓ ↓ ↓
层 0: 101 → 103 → 105 → 107 → 109 → 112 → 115 → 118 → 120 → 122

■ 跳跃表的查找规则为：从最高层开始，在当前层向右逐个访问节点，直至遇到下一个节点大于目标值时，下降至下一层继续查找；重复此过程，直至找到目标节点或确认其不存在。以查找 118 为例：
层 2：从 101 开始，向右跳至 109（因下一个节点 120 大于 118），故下降至层 1；
层 1：从 109 开始，向右跳至 115（因下一个节点 120 大于 118），故下降至层 0；
层 0：从 115 开始，向右跳至 118，成功找到目标节点 118。
在此过程中，节点的跳跃顺序为 101 → 109 → 115 → 118，充分展现了跳跃表的高效查找能力。

（1）若需在跳跃表中查找为 107 的图书，从最高层（即第 2 层）开始查找。则查找过程中节点的跳跃顺序为：101 → 105 → 107（填写节点编号，用"→"连接）

（2）定义函数 build_skip(books, step_list) ，其功能是根据步长列表 step_list 构建跳跃表。函数返回一个多层链表结构 skip，其中 skip[i]代表第 i 层链表的节点列表，每个节点用列表[编号,下层索引,本层下一索引]表示，其中：

- 编号：节点的图书编号
- 下层索引：该节点在下一层链表中的索引，-1 表示无下层

• 本层下一索引：该节点在本层链表中下一个节点的索引，-1 表示无下一节点请在以下划线处填入恰当的代码：

```
def build_skip(books, step_list):
    n = len(books)
    layers = len(step_list)
    skip = [[] for i in range(layers)]
    for i in range(layers):# 构建每一层链表
        skip[i].append([books[0], -1, -1])  # 添加头节点
        step = step_list[i]
        t = 0
        for j in range(step, n, step):
            node = [books[j], -1, -1]
            skip[i].append(node)
            skip[i][t][2] = len(skip[i]) - 1
            t = ①_____
        skip[i][t][2] = -1
    for i in range(layers - 1, 0, -1):# 建立层与层之间的链接
        p1 = 0 #p1 为下层索引
        p2 = 0
        while p2 !=-1:
            if skip[i][p2][0]==skip[i-1][p1][0]:
                skip[i][p2][1] =p1
                p2=②_____
            p1=skip[i-1][p1][2]
    return skip
```

t = len(skip[i]) - 1 或 t += 1

p2=skip[i][p2][2]

完成两层链表之间的链接建立
下一层链表p2节点往后遍历一个，为下一次做准备

（3）定义函数 search_skip(skip, target)，其功能是在跳跃表 skip 中查找编号为 target 的图书。若找到该图书，则返回 True；否则，返回 False。请在划线处填入合适的代码：

```
def search_skip(skip, target):
    p = 0
    for i in range(len(skip) - 1, -1, -1):
        q =skip[i][p][2]
        while ①_____:
            p = q
            q = skip[i][p][2]
        if skip[i][p][0] == target:
            return True
        ②_____
    return False
'''
```

q != -1 and skip[i][q][0] <= target

p和target比对，所以上面条件中要有等号

p = skip[i][p][1]

将图书编号信息升序排列存入 books 列表，例如 [101,103,105,107,109,112,115 ...]将每一层的跳跃步长信息存入 step_list 列表，例如 [1, 2, 4,...]
表示层 0 步长为 1（每节点都保留），层 1 步长为 2（每 2 个节点保留 1 个），层 2 步长为4（每 4 个节点保留 1 个），层 i 的跳跃步长为 2^i...，代码略
'''

```
skip = build_skip(books, step_list)
#输出跳跃表结构，代码略
target = int(input("查找编号为："))
result = search_skip(skip, target)
print("查找结果为：",result)
#根据查询结果，结合实际图书数据，进一步进行插入、删除等维护操作，代码略
```