**1.Prove (by using the definitions of the notations involved) or disprove (by giving a specific counterexample)the following assertions.**

**a.Prove:**

t(n)∈O(g(n))，则t(n)<=g(n)，故g(n)∈Ω(t(n))

**b.Disprove:**

设，，但。故

**c.Prove:**

设，则①

设，则②

设，则③

由②③得时，

由①得

**d.Disprove:**

设，，则两个函数不属于任何一种情况

-----------------------------------------------------------------------------------------------------------------

**2.Calculate the time complexity of the following algorithms respectively**

**a.**

**Input size :** n的大小

**Basic operation :** 判断是否满足

**Check :** 无论n有多大，循环都会在，即时停止，故无需区分最差、平均和最好情况

**Time efficiency :** 经过了约次循环，故

**b.**

**Input size :** n的大小

**Basic operation :** 最内层循环内x的自增

**Check :** 无论n有多大，循环都会在最外层循环结束后停止，故无需区分最差、平均和最好情况

**Time efficiency :** 执行次数C(n)求法为：



-----------------------------------------------------------------------------------------------------------------

**3.Calculate the time complexity of the following recursive algorithms respectively(If it may, the worst, average, and best cases must be investigated separately.)**

**a.**

**Input size :** n的大小

**Basic operation :** 判断n % 2 == 0是否成立（或计算n % 2）

**Check :** 当n=0时，执行0次判断操作，为最好情况；其余情况都为平均情况；无最坏情况。

**Time efficiency :**

**Best case :** 由Check得

**Average cases :** 输入n执行次数为输入n/2的次数加1，将1代入n得执行次数为1，C(n)满足以下递推公式：且

设，则



**b.**

**Input size :** A[]的元素个数n

**Basic operation :** 两个数组元素之间的大小比较

**Check :** （算法目的是对数组进行排序，设实参除了A[]非空, low = 0, high = n外不会有其他情况）当n=1时，执行0次比较操作，为最好情况；当A[]已经有序时，每次需要花费很多时间才能找到基准元素的存放位置，且每一个位置都要检查一次，为最坏情况；其余情况都为平均情况。

**Time efficiency :**

**Best case :** 由Check得

**Average cases :** 对于Partition操作来说，比较次数

每次完成Partition的操作，都有一个元素正确定位，且生成两个子序列。再进行递归时，两边的high-low之和刚好等于n，与两子序列元素相同的比较次数是一致的，可以看做每次都划分为等长的两个子序列，即

设，则有：



**Worst cases :** 若A[]已经有序，则每次划分只能得到比上次少一个元素的子序列，因此第1趟需要n-1次比较才能得到基准（第1个元素）的存放位置，第2趟需n-2次比较......以此类推，比较次数C(n)将会是：



-----------------------------------------------------------------------------------------------------------------

**4.Solve the following recurrence relations.**

**a.**



**b.**



**c.**

设，则有：



-----------------------------------------------------------------------------------------------------------------

**Programming**

**Q1.**

**思路：**

设一步走次数为，两步走次数为，由台阶数和卡路里的关系可得：



两式相减得，故从开始循环，重复下面步骤：

①按当前的值，计算出满足方程组的

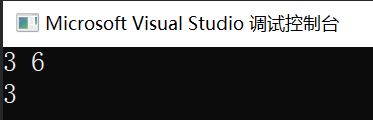
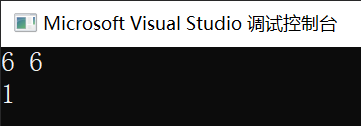
②若非负，则说明找到一种情况，需要次一步走和次两步走

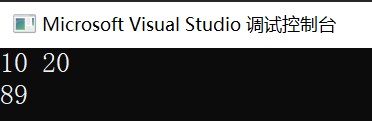
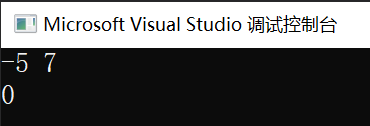
③计算出②中情况的方法数（由方程组，为）

④将③中的结果累加到总方法数上

⑤若则循环结束，得出总方法数；否则自减，重复①

**运行结果：**





**Q2.**

**思路：**

卡路里消耗速率上，，，故想消耗尽可能多的卡路里，需要尽可能多的2步走，只需在Q1算法基础上，找出最大的那组解，对应的方法数就是总方法数。

依然是从开始循环，重复下面步骤：

①按当前的值，计算出满足方程组的

②若非负，则说明所需最大的情况找到，此时需要次一步走和次两步走，执行③④

③计算出②中情况的方法数（由方程组，为）

④将③中的结果输出，结束

⑤若为负，则自减，重复①

**运行结果：**

