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 $g(n)=n^2$， $\alpha=1$， $t(n)=n\in O(g(n))$ 但 $t(n)\notin\Theta(g(n))$。故 $\Theta(\alpha g(n))\neq O(g(n))$

c.Prove:

设 $t(n)\in\Theta(g(n))$，则 $\exists n_0,c_1,c_2$，使得 $n\geq n_0$ 时有 $c_2g(n)\leq t(n)\leq c_1g(n)$ ①

设 $t(n)\in O(g(n))$，则 $\exists n_0,c_1$，使得 $n\geq n_0$ 时有 $t(n)\leq c_1g(n)$ ②

设 $t(n)\in\Omega(g(n))$，则 $\exists n_0,c_2$，使得 $n\geq n_0$ 时有 $t(n)\geq c_2g(n)$ ③

由 ② ③ 得 $t(n)\in O(g(n))\cap\Omega(g(n))$ 时，$\exists n_0,c_1,c_2$，使得 $n\geq n_0$ 时有 $c_2g(n)\leq t(n)\leq c_1g(n)$

由①得 $\Theta(g(n))=O(g(n))\cap\Omega(g(n))$

d.Disprove:

设 $t(n)=\sin n+1$，$g(n)=\cos n+1$，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 $(i+1)(i+1)\leq n$ 是否满足

Check : 无论 n 有多大，循环都会在 $(i+1)(i+1)>n$，即 $i>\sqrt{n}-1$ 时停止，故无需区分最差、平均和最好情况

Time efficiency : 经过了约 $\sqrt{n}-1$ 次循环，故 $C(n)=\sqrt{n}-1\in\Theta(\sqrt{n})$

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：

$$C(n)=\sum_{i=1}^{n}\sum_{k=1}^{i}\sum_{j=1}^{k}1=\sum_{i=1}^{n}\sum_{k=1}^{i}k=\sum_{i=1}^{n}\frac{(i+1)i}{2}=\frac{1}{2}(\frac{n(n+1)(2n+1)}{6}+\frac{(n+1)n}{2})=\frac{1}{6}n^3+\frac{1}{2}n^2+\frac{1}{3}n$$

$\in\Theta(n^3)$

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 $C(0)=0$

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：$C(n)=C(n/2)+1$ 且 $C(1)=1$

设 $n=2^k$，则 $C(n)=C(2^{k-1})+1=C(2^{k-2})+1=...=C(1)+k=k+1=\log_2 n+1$

$\in\Theta(\log_2 n)$

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 $C(1)=0$

Average cases : 对于 Partition 操作来说，比较次数 $P(n)=high-low=n$

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 $C(n)=P(n)+2C(n/2)=n+2C(n/2)$

设 $n=2^k$，则有：$C(n)=n+2C(2^{k-1})=n+2(\frac{n}{2}+2C(2^{k-2}))=2n+4C(2^{k-2})$

$$=2n+4(\frac{n}{4}+2C(2^{k-3}))=3n+8C(2^{k-3})=\ldots$$
$$=kn+2^kC(1)=n\log_2 n$$
$$\in\Theta(n\log_2 n)$$

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：

$$C(n)=\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}\in\Theta(n^2)$$

4.Solve the following recurrence relations.

a.

$$T(n)=T(n-1)+n=T(n-2)+n+(n-1)=...$$
$$=T(0)+\sum_{i=1}^{n}i=1+\frac{(n+1)n}{2}$$

b.

$$T(n)=4T(n-1)=4^2T(n-2)=...$$
$$=4^{n-1}T(1)=5*4^{n-1}$$

c.

设$n=3^k$，则有：

$$T(n)=T(3^{k-1})+n=T(3^{k-2})+n+\frac{n}{3}=...$$
$$=T(1)+\sum_{i=1}^{k}\frac{n}{3^{i-1}}=1+\frac{3}{2}n-\frac{3}{2}=\frac{3}{2}n-\frac{1}{2}$$

Programming

Q1.

思路:

设一步走次数为k_1，两步走次数为k_2，由台阶数和卡路里的关系可得：

$$\begin{cases}k_1+2k_2=m\\k_1+3k_2\le n\end{cases}$$

两式相减得$k_2\le n-m$，故k_2从$n-m$开始循环，重复下面步骤：

①按当前k_2的值，计算出满足方程组的k_1

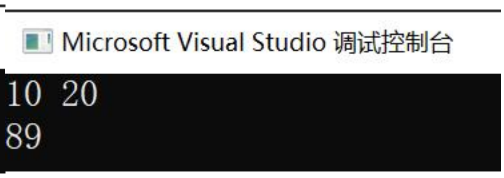
②若k_1非负，则说明找到一种情况，需要k_1次一步走和k_2次两步走

③计算出②中情况的方法数$C_{k_1+k_2}^{k_2}$（由方程组，为$C_{m-k_2}^{k_2}$）

④将③中的结果累加到总方法数上

⑤若$k_2=0$则循环结束，得出总方法数；否则k_2自减，重复①

运行结果：



Q2.

思路：

卡路里消耗速率上，$v_{1步走}=\frac{1}{1}=1(卡/阶)$，$v_{2步走}=\frac{3}{2}=1.5(卡/阶)$，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 k_2 最大的那组解，对应的方法数就是总方法数。

依然是 k_2 从 $n-m$ 开始循环，重复下面步骤：

①按当前 k_2 的值，计算出满足方程组的 k_1

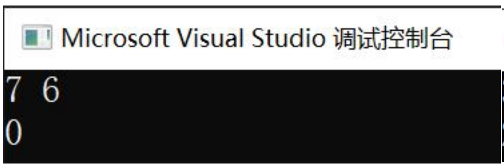
②若 k_1 非负，则说明所需 k_2 最大的情况找到，此时需要 k_1 次一步走和 k_2 次两步走，执行③④

③计算出②中情况的方法数 $C_{k_1+k_2}^{k_2}$（由方程组，为 $C_{m-k_2}^{k_2}$）

④将③中的结果输出，结束

⑤若 k_1 为负，则 k_2 自减，重复①

运行结果：




Microsoft Visual Studio 调试控制台
10 20
1