1.

a. 证明：∵ $t(n)\in O(g(n))$ ∴ $t(n)$的增长速率 ≤ $g(n)$的增长速率

由互对称性也可知 $t(n)\in O(g(n)) \Leftrightarrow g(n)\in \Omega(t(n))$ 成立.

b. 不成立. 理由：∵ $\Theta(\alpha g(n)) = \Theta(g(n))$ $(\alpha>0)$.

∴~~原假设即 $\Theta(g(n))=O(g(n))$~~

而 $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n)) \neq O(g(n))$

举例：~~$\Theta(\alpha g(n))$ $3n^2 \in \Theta(\alpha g(n))$~~

~~$3n \in \Theta(g(n))$~~

$3n \in O(n^2)$ $3n \notin \Theta(n^2)$

∴原式 $\Theta(\alpha g(n))=O(g(n))$ $(\alpha>0)$不成立

c. 证明∵ $t(n)\in\Theta(g(n))$ 表示 $t(n)$增长速率 $= g(n)$.

而 $t(n)\in O(g(n))\cap\Omega(g(n))$表示 $t(n)$增长速率既 $\le g(n)$ 而又 $\ge g(n)$ 即 $t(n)$增长速率 $=g(n)$

∴ $\Theta(g(n)) = O(g(n))\cap\Omega(g(n))$

d. 证明：对于任意 $t(n)\ge 0$，$g(n)\ge 0$ 则 $t(n)$可能 $\ge g(n)$ 即 $t(n)\in\Omega(g(n))$

$t(n)$可能 $\le g(n)$ 即 $t(n)\in O(g(n))$

$t(n)$可能 $\in\Theta(g(n))$ 即 $t(n)\in O(g(n))\cap\Omega(g(n))$ 成立.

2.

a. 循环的条件为 $(i+1)^2\le n$ 即 $i\le\sqrt{n}-1$

每次循环 $i+1$，则循环判断次数为 $\sqrt{n}+1$ 即时间复杂度为 $\Theta(\sqrt{n})$

b. 三层循环，最外层运行n次，第二层运行i次，且中i是外循环当前值，第三层运行j次.

因此当第二~~这~~层为i次时，第三层即x共增加 $\frac{i(i+1)}{2}$ 次

∴该算法x增加总次数为 $\sum_{i=1}^{n}\frac{i(i+1)}{2} = \frac{1}{2}\left(\sum_{i=1}^{n}i^2+\sum_{i=1}^{n}i\right) = \frac{1}{2}\left(\frac{n(n+1)(2n+1)}{6}+\frac{n(n+1)}{2}\right) = \frac{n^3+3n^2+2n}{6}$

∴总的时间复杂度为 $\Theta(n^3)$

3.

a. 算法在每次递归调用过程中将n/2 且调用次数只受n影响. 即 $T_{avg}=T_{worst}=T_{best}=\log_2 n$

因此算法的时间复杂度为 $O(\log_2 n)$，在最坏、最好、平均情况下~~时间复杂度均为 $O(\log_2 n)$~~

b. 该算法的时间复杂度取决于分区函数的比较与交换次数和递归调用层数.

由于该算法每次分区后均要进行 $O(n)$ 次比较与交换，故算法时间复杂度取决于递归深度

在最坏情况下分区函数选择最小或最大数作为pivot，此时分区会得到长度为0与n-1的分区，递归深度为 $O(n)$ 总时间复杂度达到 $O(n^2)$

而在最佳与平均情况下分区使得递归深度变为 $O(\log_2 n)$，此时总时间复杂度为 $O(n\log_2 n)$

∴综上最坏情况为 $O(n^2)$，最佳和平均情况为 $O(n\log n)$

4.

a. $T(n) = T(n-1) + n$

$= T(n-2) + (n-1) + n$

$= T(n-3) + (n-2) + (n-1) + n$

$\cdots$

$= T(0) + 1 + \cdots + n$.

$= T(0) + \frac{n(n+1)}{2} = 1 + \frac{n(n+1)}{2}$

b. $T(n) = 4T(n-1)$

$= 4 \times 4\, T(n-2)$

$= 4^3\, T(n-3)$

$\cdots = 4^i\, T(n-i)$.

$\cancel{= 4^{n-1}\, T(n-n+1)}$

$= 4^{n-1}\, T[n-(n-1)]$

$= 5 \times 4^{n-1}$

c. 令 $n = 3^K$.

则 $T(n) = T(3^k) = T(3^{k-1}) + 3^K$

$= T(3^{k-2}) + 3^{k-1} + 3^K$

$\cdots = T(3^{k-i}) + 3^{k-i+1} + \cdots + 3^K$

$\cancel{= T(3^{k-k}) + 3^2 + 3^3 + \cdots + 3^K}$

$= T(3^{k-k}) + 3^1 + \cdots + 3^K$

$= T(1) + 3^1 + \cdots + 3^K$

$= 1 + 3^1 + 3^2 + \cdots + 3^K$

$= \frac{3^{k+1}-1}{2} = \frac{3n-1}{2}$

Assignment One-Programming

Q1

编程思路

考虑需要爬到第 m 级台阶时的方法只有两种，第一种是从 m-1 级台阶爬到第 m 级，第二种是从 m-2 级台阶爬到 m 级，由此可以想到以递归的方式解决该问题，设计一个函数解决可用 n 卡路里到达第 m 级台阶的总方法数

运行结果

```
Microsoft Visual Studio 调试控制台
6 6
1

E:\VS\算法设计与分析\Assignment1\Debug\Problem1.exe (进程 25048)已退出，代码为 0。
按任意键关闭此窗口. . .
```

```
Microsoft Visual Studio 调试控制台
3 6
3

E:\VS\算法设计与分析\Assignment1\Debug\Problem1.exe (进程 37468)已退出，代码为 0。
按任意键关闭此窗口. . .
```

```
Microsoft Visual Studio 调试控制台
-5 7
0

E:\VS\算法设计与分析\Assignment1\Debug\Problem1.exe (进程 31876)已退出，代码为 0。
按任意键关闭此窗口. . .
```

Q2

编程思路

考虑到需要消耗最多卡路里，则可以先用递归算法求出所用方案剩余的卡路里最少的量 n_MinLeft，再用一次递归算法求出符合该剩余量的所有方案数

运行结果