資料結構第六章習題參考解答

課本： Data Structures Using C and C++

by Y. Langsam, M. J. Augenstein and A. M.Tenenbaum1.

1. Design an algorithm that prints all sets of six positive integers a_1, a_2, a_3, a_4, a_5 and a_6 such that $a_1 \le a_2 \le a_3 \le n$ and $a_1 < a_4 \le a_5 \le a_6 \le n$, and the sum of the squares of a_1, a_2 and a_3 equals the sum of the squares of a_4, a_5 and a_6. Note that your algorithm should be of $O(n^3 \log n)$ time.

Solution:

滿足 $a_1 \le a_2 \le a_3 \le n$ 的所有情形 (a_1, a_2, a_3) 與滿足 $a_4 \le a_5 \le a_6 \le n$ 的所有情形 (a_4, a_5, a_6) 相同，找出滿足 $a_p \le a_q \le a_r \le n$ 形式的所有情形，計算 $S_k = a_{k,p}^2 + a_{k,q}^2 + a_{k,r}^2$ 之值，與 $(a_{k,p}, a_{k,q}, a_{k,r})$ 一併記錄於陣列中，總個數介於 n^3 與 C_3^n，此步驟時間複雜度為 $O(n^3)$。

接著排序陣列，主要依據 S_k 的大小，若 S_k 之值相等，次要則依 $a_{k,p}$ 的大小，較小者於前頭。因元素數量約為 (略小於) n^3，所需的時間複雜度為 $O(n^3 \log n^3) = O(n^3 \cdot 3 \log n) = O(n^3 \log n)$。

排序後，總和相等者，將會排在隔鄰（有可能連續 h 組的總和相等）。因此，依序檢查陣列中的元素，檢查總和 S_k 是否與前 h 組相等。若相等，則印出這 $h+1$ 組解 $(a_{k-i,p}, a_{k-i,q}, a_{k-i,r}, a_{k,p}, a_{k,q}, a_{k,r})$，其中 i 為小於等於 h 的正整數。假定最大可能的 h 為常數，此步驟時間複雜度亦為 $O(n^3)$。

如下圖為 $n = 9$ 時，此演算法的部份示意圖：

$S_k, a_{k,p}, a_{k,q}, a_{k,r}$ $(a_1, a_2, a_3, a_4, a_5, a_6)$ ...

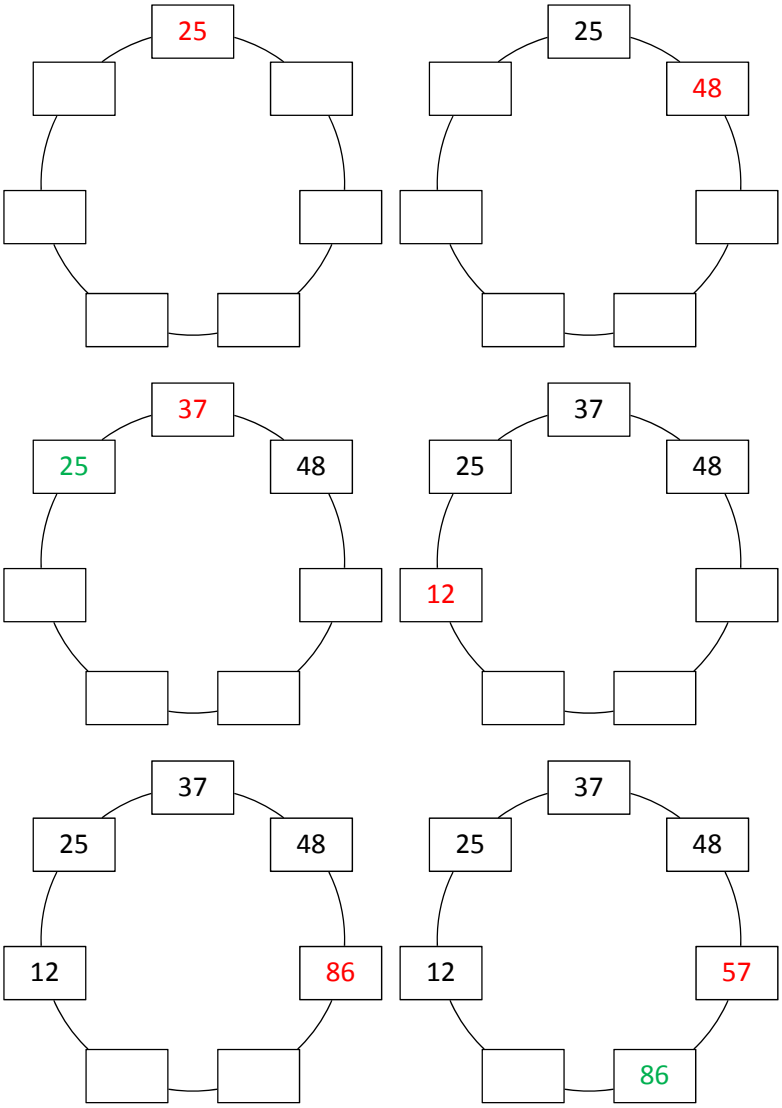
```
3,1,1,1
...
82,3,3,8
83,1,1,9
83,3,5,7 (1,1,9,3,5,7)
84,2,4,8
86,1,2,9
86,1,6,7 (1,2,9,1,6,7)
86,5,5,6 (1,2,9,5,5,6)(1,6,7,5,5,6)
88,4,6,6
89,2,2,9
89,2,6,7 (2,2,9,2,6,7)
89,3,4,8 (2,2,9,3,4,8)(2,6,7,3,4,8)
90,1,5,8
90,4,5,7 (1,5,8,4,5,7)
91,1,3,9
93,2,5,8
...
243,9,9,9
```

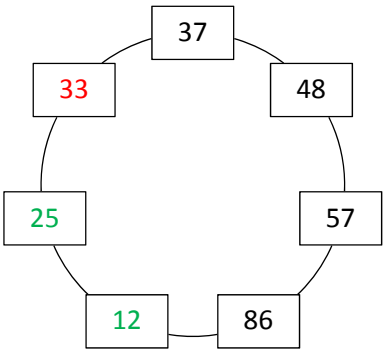
<附件， C++ 程式實作>

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class record {
public:
    int p,q,r,sum;

    record(int p,int q,int r) : p(p),q(q),r(r),sum(p*p+q*q+r*r) {}

    bool operator <(const record &r) const {
        return (sum!=r.sum)?(sum<r.sum):(p<r.p);
    }

    friend ostream &operator <<(ostream &os,const record &r) {
        os<<r.p<<","<<r.q<<","<<r.r;
    }
};

int main() {
    int n;
    cin>>n;

    vector<record> vec;
    for (int p=1;p<=n;p++)
        for (int q=p;q<=n;q++)
            for (int r=q;r<=n;r++)
                vec.push_back(record(p,q,r)); // O(n^3)

    sort(vec.begin(),vec.end()); // O(n^3 log n)
    int eq=0;
    cout<<vec[0].sum<<","<<vec[0]<<endl;
    for (int i=1;i<vec.size();i++) {
        cout<<vec[i].sum<<","<<vec[i];
        if (vec[i].sum==vec[i-1].sum) {
            eq++;
            for (int j=1;j<=eq;j++) {
                cout<<" ("<<vec[i-j]<<","<<vec[i]<<")";
                            // O(h n^3)
            }
        } else
            eq=0;
        cout<<endl;
    }
    return 0;
}
```

2. The *two-way insertion sort* is a modification of the simple *insertion sort* as follows: A separate output array of size n is set aside. This output array acts as a circular structure. (The right position of x[i] is x[i+1] if $0 \le i \le n-2$, and the right position of x[n-1] is x[0].) x[0] is placed into the middle element of the array. Once a contiguous group of elements are in the array, room for a new element is made by shifting all smaller elements one step to the left or all larger element one step to the right. The choice of which shift to perform depends on which would cause the smallest amount of shifting. Use the following 7 elements to explain how this algorithm works: 25, 48, 37, 12, 86, 57, 33.

紅色：inserted；綠色：shifted。




37
33
48
25
57
12
86