UVa Online Judge

10402 Triangle Covering

Triangle covering problem is a very interesting. Some people have done interesting research on this topic. In this problem you are asked to solve some elementary covering problems. The basic idea of covering problem is to find out the smallest size of a particular shape that can entirely cover another particular shape. Covering problems have many practical applications such as finding smallest possible square sized covers for a round table. In some ways covering problems are similar to packing problems but they have significant differences also. In this problem you are asked to find out the maximum possible size of an equilateral triangle which can be covered by 2, 3, 4 or 6 squares of equal size. The following pictures show how these coverings can be done.

You can assume that:

a) When a picture looks exactly symmetric along a certain axis they are actually symmetric;

b) When three or more lines appear coincident they are actually coincident; and

c) When a part of a figure looks exactly similar to another part they are actually similar.

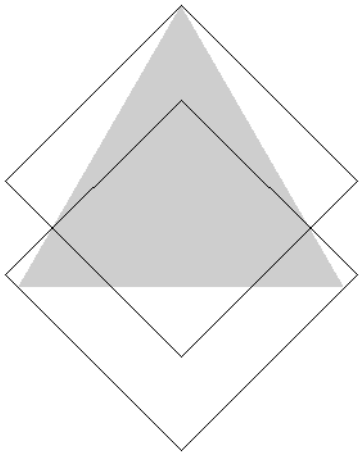

Fig 1: Covering with two squares

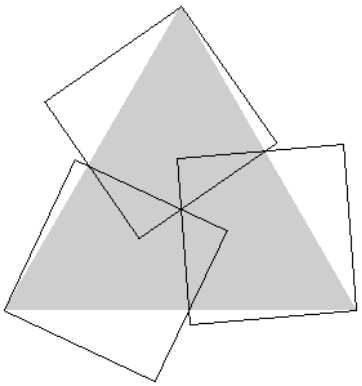

Fig 2: Covering with three squares

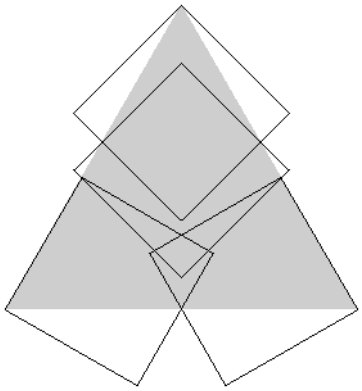

Fig 3: Covering with four squares

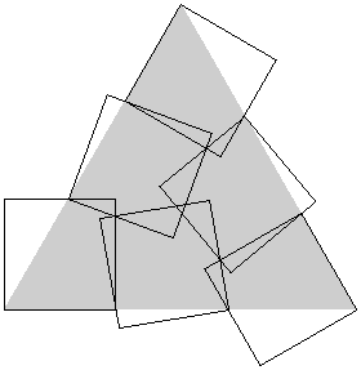

Fig 4: Covering with six squares

Input

First line of the input file contains a single integer N ($N \leq 5000$) which indicates how many sets of input are there in the input file. Each of the next N lines makes a set of input.

Each line contains a single floating point number S ($0 \leq S \leq 10000$) which indicates the side of the squares that will cover the equilateral triangle.

Output

For each set of input produce one line of output. So the output file contains N lines of output.

Each line contains four floating point numbers T_2, T_3, T_4 and T_6. Each floating point number has ten digits after the decimal point. Here T_x means the side of the largest triangle that can be covered by x squares of side S. You don't need to worry about small precision errors as small precision errors will be ignored (max(1e-7, 0.00001%)). All the floating point numbers in the output should have ten digits after the decimal point.

Sample Input

```
5
0.000000001
0.000000002
0.000000003
0.000000004
0.000000005
```

Sample Output

```
0.0000000013 0.0000000021 0.0000000023 0.0000000032
0.0000000026 0.0000000042 0.0000000046 0.0000000063
0.0000000039 0.0000000063 0.0000000069 0.0000000095
0.0000000053 0.0000000084 0.0000000092 0.0000000127
0.0000000066 0.0000000105 0.0000000115 0.0000000158
```