A.GoodEnough?

A positive integer number is "Special" if it is both a square (eg. **1, 4, 9, 16, 64 ...**) and a cube (eg. **1, 8, 27, 64 ...**). The smallest special number is 1.Now your job is to write a program that finds whether a number less than 100000000 is special or not. It may be noted that there are only 21 such numbers within this range and these are 1**, 64, 729, 4096, 15625, 46656, 117649, 262144, 531441, 1000000,**
1771561, 2985984, 4826809, 7529536, 11390625, 16777216, 24137569, 34012224, 47045881, 64000000 and **85766121**. A very childish but legitimate C/C++ solution, which would work for positive numbers not exceeding 15624, is shown below.

```
int num;
 while(scanf("%d",&num) &&
 num)
{
        if(num==1 || num==64 || num==729 ||
            num==4096) printf("Special\n");
        else
            printf("Ordinary\n"
            );
}
```

A C/C++ code that will work for positive numbers not exceeding 15624

Input

The input file contains at most 1001 lines of input. Each line contains a positive integer less than **100000000**.Input is terminated by a line containing a zero.

Output

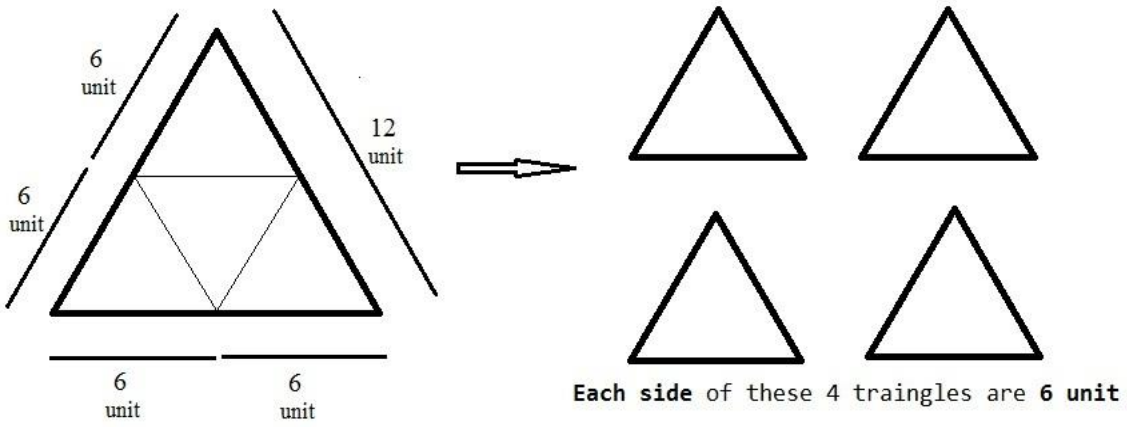
For each line of input except the last one produce one line of output. This line contains a string (without the quotes) "**Special"** if the number is special and "**Ordinary"** if the number is not special. Look at the output for the sample input for details.

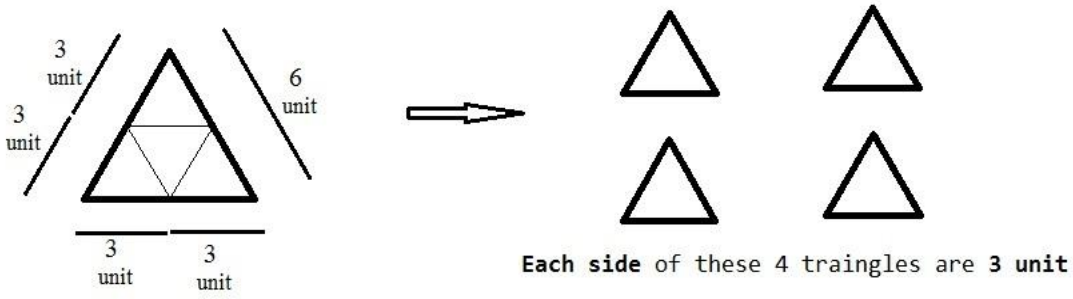
Sample Input	**Output for Sample Input**
1 **2** **64** **100** **15625** **0**	**Special** **Ordinary** **Special** **Ordinary** **Special**

B. EUILATERAL TRIANGLE

Time Limit: 1s

Students of East West University don't like inequality. They always want to spread anything equally to others. They also don't like fraction. They prefer integer to fraction. They have an equilateral triangle. They want to divide it into smaller equilateral triangle in such a way that each side of a triangle will have integer unit. In the following triangle, each side is 12 unit. It can be divided into 4 smaller pieces, each side of 6 unit. Then each of them can be again divided into 4 equal pieces, each side of 3 unit. Then it should be stopped, otherwise each side will have fraction unit.





Input:

First line of input will contain an integer **T<=100000** denoting the number of test cases. Then next **T** lines will contain an integer **0<=N<=2^{31}** denoting the length of initial triangle.

Output:

For each input of **N** you have to print "**Case X: A L**" where **X** denotes the case number (starting from 1), **A** denotes the number of smaller triangles and **L** denotes the length of these smaller triangles.

Sample Input:	**Sample Output:**
4 1 2 12 100000	Case 1: 1 1 Case 2: 4 1 Case 3: 16 3 Case 4: 1024 3125

C. PSHYCO_PRINTF

Time Limit: 1s

Read the following code:

```
#include<stdio.h>
long long int function1(long long int n)
{
   if(n==0 || n==1)
   {
      return n;
   }
   return function1(n+1-n)+function1(n-1);
}
long long int function2(long long int n)
{
   if(n==0 || n==1)
   {
      return 1;
   }
return function2(n-1)*function2(n-2);
}
int main()
{
   long long int n;
   scanf("%lld", &n);
   printf("Output: %lld\n", function1(n)/function2(n));
   return 0;
}
```

Input:

There is only one line of input.

Output:

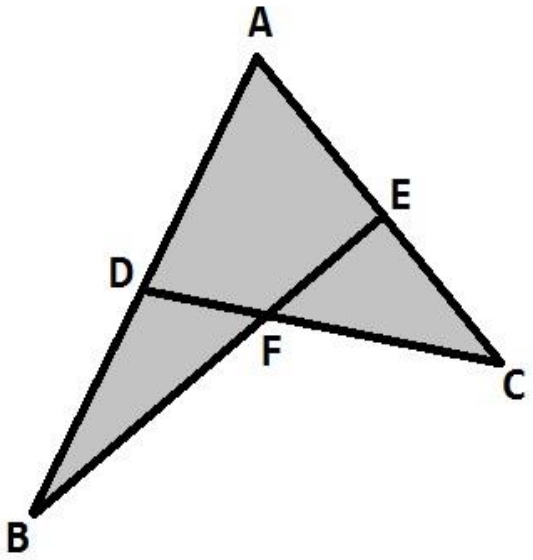
Print only one line of output. Follow the sample output.

Sample Output:
Output: 123

D. TYPICAL TRIANGLE

Time Limit: 1s

Students of East West University have made a design for their celebration like a triangle shape to decorate the room. But there is not enough space in the wall. So, they have become very curious to measure the area which will be needed by that shape.



Here, **D** is the midpoint of **AB** and **E** is the midpoint of **AC**, **CD** and **BE** intersect at point **F**.

Input:

Input contains several test cases. Each test case will contain six floating point numbers denoting the co-ordinates of **A, B, C** respectively where first two numbers denote co-ordinate of point **A(x, y)** and so on. Input will be terminated by '0 0 0 0 0 0' without quotes. You shouldn't print anything for this.

Output:

For each test case you have to print "**Case P: S**" where **P** denotes the case number (starting from 1), **S** denotes the area of **ABC** means **shaded portion** with 6 decimal places.

Sample Input:
2.00 3.01 -2.00 -3.01 2.00 -3.01 4.87 8.41 -4.87 -8.41 -4.87 8.41 14092.60 26620.36 -14092.60 -26620.36 -14092.60 26620.36 6003.44 11339.81 -6003.44 -11339.81 6003.44 -11339.81 0 0 0 0 0 0
Sample Output:
Case 1: 8.026667 Case 2: 54.608933 Case 3: 500200113.781333 Case 4: 90770491.928533

E. BINARY FRIEND

Time Limit: 1s

Students of East West University like to play with binary digits means **1** and **0**. One day they discovered some behaviors of binary numbers. Some students were confused about one behavior. They want to count the number from **0** to $\mathbf{2^n-1}$ which have **exactly one 0** in their binary representation.

Number	Binary
0	***0***
1	1
2	***10***
3	11
4	100
5	***101***
6	***110***
7	111
8	1000
9	1001
10	1010
11	***1011***
12	1100

13	***1101***
14	***1110***
15	1111

The example is given here for **n=4**.

When n=4 you have total 16 numbers from 0 to 2^n-1 or simply 0 to 15. When we find the binary form of 0 to 15, we can see total 7 numbers have **exactly one 0** in their binary form (Which are bold in Italic font). These numbers are 0, 2, 5, 6, 11, 13 and 14.

Now it's your turn to help them by counting the total numbers from **0** to **2^n-1** which has **exactly one 0** in their binary representation.

Input:

First line of input will contain an integer **K<=100000** denoting the number of test cases. Then next **T** lines will contain an integer **0<=n<=10^9**.

Output:

For each input of **n** you have to print "**Case C: D**" where **C** denotes the case number (starting from 1), **D** denotes the total numbers from **0** to **2^n-1** which has **exactly one 0** in their binary representation.

Sample Input:	**Sample Output:**
2 4 9999	Case 1: 7 Case 2: 49985002