# 1. Quoit Design

**Time Limit : 10000/5000ms (Java/Other)   Memory Limit : 65536/32768K (Java/Other)**

**Total Submission(s) : 0   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

Have you ever played quoit in a playground? Quoit is a game in which flat rings are pitched at some toys, with all the toys encircled awarded.  
In the field of Cyberground, the position of each toy is fixed, and the ring is carefully designed so it can only encircle one toy at a time. On the other hand, to make the game look more attractive, the ring is designed to have the largest radius. Given a configuration of the field, you are supposed to find the radius of such a ring.  
  
Assume that all the toys are points on a plane. A point is encircled by the ring if the distance between the point and the center of the ring is strictly less than the radius of the ring. If two toys are placed at the same point, the radius of the ring is considered to be 0.

**Input**

The input consists of several test cases. For each case, the first line contains an integer N (2 <= N <= 100,000), the total number of toys in the field. Then N lines follow, each contains a pair of (x, y) which are the coordinates of a toy. The input is terminated by N = 0.

**Output**

For each test case, print in one line the radius of the ring required by the Cyberground manager, accurate up to 2 decimal places.

**Sample Input**

2

0 0

1 1

2

1 1

1 1

3

-1.5 0

0 0

0 1.5

0

**Sample Output**

0.71

0.00

0.75

# 2. Elevator

**Time Limit : 2000/1000ms (Java/Other)   Memory Limit : 65536/32768K (Java/Other)**

**Total Submission(s) : 0   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

The highest building in our city has only one elevator. A request list is made up with N positive numbers. The numbers denote at which floors the elevator will stop, in specified order. It costs 6 seconds to move the elevator up one floor, and 4 seconds to move down one floor. The elevator will stay for 5 seconds at each stop.  
  
For a given request list, you are to compute the total time spent to fulfill the requests on the list. The elevator is on the 0th floor at the beginning and does not have to return to the ground floor when the requests are fulfilled.

**Input**

There are multiple test cases. Each case contains a positive integer N, followed by N positive numbers. All the numbers in the input are less than 100. A test case with N = 0 denotes the end of input. This test case is not to be processed.

**Output**

Print the total time on a single line for each test case.

**Sample Input**

1 2

3 2 3 1

0

**Sample Output**

17

41

**3.Number Sequence**

**Time Limit : 2000/1000ms (Java/Other)   Memory Limit : 65536/32768K (Java/Other)**

**Total Submission(s) : 0   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

A number sequence is defined as follows:  
  
f(1) = 1, f(2) = 1, f(n) = (A \* f(n - 1) + B \* f(n - 2)) mod 7.  
  
Given A, B, and n, you are to calculate the value of f(n).

**Input**

The input consists of multiple test cases. Each test case contains 3 integers A, B and n on a single line (1 <= A, B <= 1000, 1 <= n <= 100,000,000). Three zeros signal the end of input and this test case is not to be processed.

**Output**

For each test case, print the value of f(n) on a single line.

**Sample Input**

1 1 3

1 2 10

0 0 0

**Sample Output**

2

5

**4.汉诺塔II**

**Time Limit : 2000/1000ms (Java/Other)   Memory Limit : 65536/32768K (Java/Other)**

**Total Submission(s) : 0   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

经典的汉诺塔问题经常作为一个递归的经典例题存在。可能有人并不知道汉诺塔问题的典故。汉诺塔来源于印度传说的一个故事，上帝创造世界时作了三根金刚石柱子，在一根柱子上从下往上按大小顺序摞着64片黄金圆盘。上帝命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一回只能移动一个圆盘。有预言说，这件事完成时宇宙会在一瞬间闪电式毁灭。也有人相信婆罗门至今仍在一刻不停地搬动着圆盘。恩，当然这个传说并不可信，如今汉诺塔更多的是作为一个玩具存在。Gardon就收到了一个汉诺塔玩具作为生日礼物。  
　　Gardon是个怕麻烦的人（恩，就是爱偷懒的人），很显然将64个圆盘逐一搬动直到所有的盘子都到达第三个柱子上很困难，所以Gardon决定作个小弊，他又找来了一根一模一样的柱子，通过这个柱子来更快的把所有的盘子移到第三个柱子上。下面的问题就是：当Gardon在一次游戏中使用了N个盘子时，他需要多少次移动才能把他们都移到第三个柱子上？很显然，在没有第四个柱子时，问题的解是2^N-1，但现在有了这个柱子的帮助，又该是多少呢？

**Input**

包含多组数据，每个数据一行，是盘子的数目N(1<=N<=64)。

**Output**

对于每组数据，输出一个数，到达目标需要的最少的移动数。

**Sample Input**

1

3

12

**Sample Output**

1

5

81

**5.Uniform Generator**

**Time Limit : 2000/1000ms (Java/Other)   Memory Limit : 65536/32768K (Java/Other)**

**Total Submission(s) : 0   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

Computer simulations often require random numbers. One way to generate pseudo-random numbers is via a function of the form  
  
seed(x+1) = [seed(x) + STEP] % MOD  
  
where '%' is the modulus operator.  
  
Such a function will generate pseudo-random numbers (seed) between 0 and MOD-1. One problem with functions of this form is that they will always generate the same pattern over and over. In order to minimize this effect, selecting the STEP and MOD values carefully can result in a uniform distribution of all values between (and including) 0 and MOD-1.  
  
For example, if STEP = 3 and MOD = 5, the function will generate the series of pseudo-random numbers 0, 3, 1, 4, 2 in a repeating cycle. In this example, all of the numbers between and including 0 and MOD-1 will be generated every MOD iterations of the function. Note that by the nature of the function to generate the same seed(x+1) every time seed(x) occurs means that if a function will generate all the numbers between 0 and MOD-1, it will generate pseudo-random numbers uniformly with every MOD iterations.  
  
If STEP = 15 and MOD = 20, the function generates the series 0, 15, 10, 5 (or any other repeating series if the initial seed is other than 0). This is a poor selection of STEP and MOD because no initial seed will generate all of the numbers from 0 and MOD-1.  
  
Your program will determine if choices of STEP and MOD will generate a uniform distribution of pseudo-random numbers.

**Input**

Each line of input will contain a pair of integers for STEP and MOD in that order (1 <= STEP, MOD <= 100000).

**Output**

For each line of input, your program should print the STEP value right- justified in columns 1 through 10, the MOD value right-justified in columns 11 through 20 and either "Good Choice" or "Bad Choice" left-justified starting in column 25. The "Good Choice" message should be printed when the selection of STEP and MOD will generate all the numbers between and including 0 and MOD-1 when MOD numbers are generated. Otherwise, your program should print the message "Bad Choice". After each output test set, your program should print exactly one blank line.

**Sample Input**

3 5

15 20

63923 99999

**Sample Output**

3 5 Good Choice

15 20 Bad Choice

63923 99999 Good Choice

**6.Max Sum**

**Time Limit : 2000/1000ms (Java/Other)   Memory Limit : 65536/32768K (Java/Other)**

**Total Submission(s) : 0   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

Given a sequence a[1],a[2],a[3]......a[n], your job is to calculate the max sum of a sub-sequence. For example, given (6,-1,5,4,-7), the max sum in this sequence is 6 + (-1) + 5 + 4 = 14.

**Input**

The first line of the input contains an integer T(1<=T<=20) which means the number of test cases. Then T lines follow, each line starts with a number N(1<=N<=100000), then N integers followed(all the integers are between -1000 and 1000).

**Output**

For each test case, you should output two lines. The first line is "Case #:", # means the number of the test case. The second line contains three integers, the Max Sum in the sequence, the start position of the sub-sequence, the end position of the sub-sequence. If there are more than one result, output the first one. Output a blank line between two cases.

**Sample Input**

2

5 6 -1 5 4 -7

7 0 6 -1 1 -6 7 -5

**Sample Output**

Case 1:

14 1 4

Case 2:

7 1 6

**7.Eddy's digital Roots**

**Time Limit : 2000/1000ms (Java/Other)   Memory Limit : 65536/32768K (Java/Other)**

**Total Submission(s) : 0   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

The digital root of a positive integer is found by summing the digits of the integer. If the resulting value is a single digit then that digit is the digital root. If the resulting value contains two or more digits, those digits are summed and the process is repeated. This is continued as long as necessary to obtain a single digit.  
  
For example, consider the positive integer 24. Adding the 2 and the 4 yields a value of 6. Since 6 is a single digit, 6 is the digital root of 24. Now consider the positive integer 39. Adding the 3 and the 9 yields 12. Since 12 is not a single digit, the process must be repeated. Adding the 1 and the 2 yeilds 3, a single digit and also the digital root of 39.  
  
The Eddy's easy problem is that : give you the n,want you to find the n^n's digital Roots.

**Input**

The input file will contain a list of positive integers n, one per line. The end of the input will be indicated by an integer value of zero. Notice:For each integer in the input n(n<10000).

**Output**

Output n^n's digital root on a separate line of the output.

**Sample Input**

2

4

0

**Sample Output**

4

4

**8.Island of Logic**

**Time Limit : 2000/1000ms (Java/Other)   Memory Limit : 65536/32768K (Java/Other)**

**Total Submission(s) : 0   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

The Island of Logic has three kinds of inhabitants: divine beings that always tell the truth, evil beings that always lie, and human beings that are truthful during the day and lie at night. Every inhabitant recognizes the type of every other inhabitant.  
A social scientist wants to visit the island. Because he is not able to distinguish the three kinds of beings only from their looks, he asks you to provide a communication analyzer that deduces facts from conversations among inhabitants. The interesting facts are whether it is day or night and what kind of beings the speakers are.

**Input**

The input contains several descriptions of conversations. Each description starts with an integer n, the number of statements in the conversation. The following n lines each contain one statement by an inhabitant. Every statement line begins with the speaker's name, one of the capital letters A, B, C, D, E, followed by a colon `:'. Next is one of the following kinds of statements:  
  
I am [not] ( divine | human | evil | lying ).  
  
X is [not] ( divine | human | evil | lying ).  
  
It is ( day | night ).  
  
Square brackets [] mean that the word in the brackets may or may not appear, round brackets () mean that exactly one of the alternatives separated by | must appear. X stands for some name from A, B, C, D, E. There will be no two consecutive spaces in any statement line, and at most 50 statements in a conversation.  
  
The input is terminated by a test case starting with n = 0.

**Output**

For each conversation, first output the number of the conversation in the format shown in the sample output. Then print "This is impossible.'', if the conversation cannot happen according to the rules or "No facts are deducible.'', if no facts can be deduced. Otherwise print all the facts that can be deduced. Deduced facts should be printed using the following formats:  
  
X is ( divine | human | evil ).  
  
It is ( day | night ).  
  
X is to be replaced by a capital letter speaker name. Facts about inhabitants must be given first (in alphabetical order), then it may be stated whether it is day or night.  
  
The output for each conversation must be followed by a single blank line.

**Sample Input**

1

A: I am divine.

1

A: I am lying.

1

A: I am evil.

3

A: B is human.

B: A is evil.

A: B is evil.

0

**Sample Output**

Conversation #1

No facts are deducible.

Conversation #2

This is impossible.

Conversation #3

A is human.

It is night.

Conversation #4

A is evil.

B is divine.

Reasoning made easy

To make things clearer, we will show the reasoning behind the third input example, where A says ``I am evil.''.

What can be deduced from this?

Obviously A cannot be divine, since she would be lying, similarly A cannot be evil, since she would tell the truth.

Therefore, A must be human, moreover, since she is lying, it must be night. So the correct output is as shown.

In the fourth input example, it is obvious that A is lying since her two statements are contradictory.

So, B can be neither human nor evil, and consequently must be divine. B always tells the truth, thus A must be evil. Voil‘a!

**9.Picture**

**Time Limit : 6000/2000ms (Java/Other)   Memory Limit : 32768/32768K (Java/Other)**

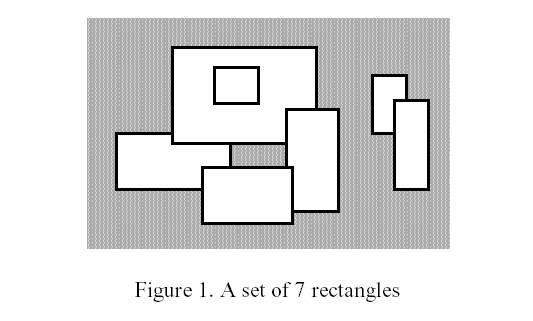
**Total Submission(s) : 0   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

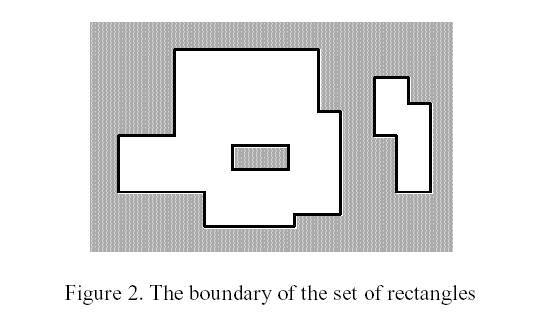
Font Size: ← →

**Problem Description**

A number of rectangular posters, photographs and other pictures of the same shape are pasted on a wall. Their sides are all vertical or horizontal. Each rectangle can be partially or totally covered by the others. The length of the boundary of the union of all rectangles is called the perimeter.  
  
Write a program to calculate the perimeter. An example with 7 rectangles is shown in Figure 1.



The corresponding boundary is the whole set of line segments drawn in Figure 2.



The vertices of all rectangles have integer coordinates.

**Input**

Your program is to read from standard input. The first line contains the number of rectangles pasted on the wall. In each of the subsequent lines, one can find the integer coordinates of the lower left vertex and the upper right vertex of each rectangle. The values of those coordinates are given as ordered pairs consisting of an x-coordinate followed by a y-coordinate.  
  
0 <= number of rectangles < 5000  
All coordinates are in the range [-10000,10000] and any existing rectangle has a positive area.  
  
Please process to the end of file.

**Output**

Your program is to write to standard output. The output must contain a single line with a non-negative integer which corresponds to the perimeter for the input rectangles.

**Sample Input**

7

-15 0 5 10

-5 8 20 25

15 -4 24 14

0 -6 16 4

2 15 10 22

30 10 36 20

34 0 40 16

**Sample Output**

228

**10.汉诺塔III**

**Time Limit : 1000/1000ms (Java/Other)   Memory Limit : 32768/32768K (Java/Other)**

**Total Submission(s) : 0   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

约19世纪末，在欧州的商店中出售一种智力玩具，在一块铜板上有三根杆，最左边的杆上自上而下、由小到大顺序串着由64个圆盘构成的塔。目的是将最左边杆上的盘全部移到右边的杆上，条件是一次只能移动一个盘，且不允许大盘放在小盘的上面。  
现在我们改变游戏的玩法，不允许直接从最左(右)边移到最右(左)边(每次移动一定是移到中间杆或从中间移出)，也不允许大盘放到下盘的上面。  
Daisy已经做过原来的汉诺塔问题和汉诺塔II，但碰到这个问题时，她想了很久都不能解决，现在请你帮助她。现在有N个圆盘，她至少多少次移动才能把这些圆盘从最左边移到最右边？

**Input**

包含多组数据，每次输入一个N值(1<=N=35)。

**Output**

对于每组数据，输出移动最小的次数。

**Sample Input**

1

3

12

**Sample Output**

2

26

531440

**u Calculate e**

**Time Limit : 2000/1000ms (Java/Other)   Memory Limit : 65536/32768K (Java/Other)**

**Total Submission(s) : 2   Accepted Submission(s) : 2**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

A simple mathematical formula for e is  
  
  
  
where n is allowed to go to infinity. This can actually yield very accurate approximations of e using relatively small values of n.

**Output**

Output the approximations of e generated by the above formula for the values of n from 0 to 9. The beginning of your output should appear similar to that shown below.

**Sample Output**

n e

- -----------

0 1

1 2

2 2.5

3 2.666666667

4 2.708333333

**As Easy As A+B**

**Time Limit : 2000/1000ms (Java/Other)   Memory Limit : 65536/32768K (Java/Other)**

**Total Submission(s) : 0   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

These days, I am thinking about a question, how can I get a problem as easy as A+B? It is fairly difficulty to do such a thing. Of course, I got it after many waking nights.  
Give you some integers, your task is to sort these number ascending (升序).  
You should know how easy the problem is now!  
Good luck!

**Input**

Input contains multiple test cases. The first line of the input is a single integer T which is the number of test cases. T test cases follow. Each test case contains an integer N (1<=N<=1000 the number of integers to be sorted) and then N integers follow in the same line.  
It is guarantied that all integers are in the range of 32-int.

**Output**

For each case, print the sorting result, and one line one case.

**Sample Input**

2

3 2 1 3

9 1 4 7 2 5 8 3 6 9

**Sample Output**

1 2 3

1 2 3 4 5 6 7 8 9

**Financial Management**

**Time Limit : 400/200ms (Java/Other)   Memory Limit : 65536/32768K (Java/Other)**

**Total Submission(s) : 2   Accepted Submission(s) : 0**

Font: Times New Roman | Verdana | Georgia

Font Size: ← →

**Problem Description**

Larry graduated this year and finally has a job. He’s making a lot of money, but somehow never seems to have enough. Larry has decided that he needs to grab hold of his financial portfolio and solve his financing problems. The first step is to figure out what’s been going on with his money. Larry has his bank account statements and wants to see how much money he has. Help Larry by writing a program to take his closing balance from each of the past twelve months and calculate his average account balance.

**Input**

The input will be twelve lines. Each line will contain the closing balance of his bank account for a particular month. Each number will be positive and displayed to the penny. No dollar sign will be included.

**Output**

The output will be a single number, the average (mean) of the closing balances for the twelve months. It will be rounded to the nearest penny, preceded immediately by a dollar sign, and followed by the end-of-line. There will be no other spaces or characters in the output.

**Sample Input**

100.00

489.12

12454.12

1234.10

823.05

109.20

5.27

1542.25

839.18

83.99

1295.01

1.75

**Sample Output**

$1581.42