**The Great Computer Challenge, 2018**

***Scientific Non-Business Programming, Level IV***

# **Background**

**Scientific computing** is a rapidly growing multidisciplinary field that uses advanced computing capabilities to understand and solve complex problems. It is an area of science which spans many disciplines, but at its core it involves the development of models and simulations to understand natural systems.

# **Guidelines & Requirements**

The guidelines and requirements are listed in each challenge.

# **Challenge 1 Lottery**

Fred likes to play the lotto. Whenever he does, he buys lots of tickets. Each ticket has 6 unique numbers in the range from 1 to 30, inclusive. Fred likes to “Cover all his bases." By that, he means that he likes for each set of lottery tickets to contain every number from 1 to 30, at least once, on some ticket. Write a program to help Fred see of his tickets “Cover all the bases."

**Input**

The input file is called “lottery.txt”. It consists of a number of test cases. Each case starts with an integer N (between 1 and 100, inclusive), indicating the number of tickets Fred has purchased. On the next N lines are the tickets, one per line. Each ticket will have exactly 6 integers, and all of them will be in the range from 1 to 30 inclusive. No ticket will have duplicate numbers, but the numbers on a ticket may appear in any order. The input ends with a line containing only a ‘0'.

**Output**

Print a list of responses for the input sets, one per line. Print the word ‘Yes' if every number from 1 to 30 inclusive appears in some lottery ticket in the set, and `No' otherwise. Print these words exactly as they are shown. Do not print any blank lines between outputs.

Sample Input (lottery.txt)

1

1 2 3 4 5 6

5

1 2 3 4 5 6

10 9 8 7 12 11

13 14 15 16 17 18

19 20 21 22 23 24

25 26 27 28 29 30

6

1 2 3 4 5 6

10 9 8 7 12 11

13 14 15 16 17 18

13 20 25 22 27 24

25 26 27 28 29 30

10 20 18 17 12 11

0

Sample Output

No

Yes

No

# **Challenge 2 Elegant Diamond**

The king has hired you to make him an elegant diamond. An elegant diamond is a two-dimensional object made out of digits that's symmetric about a horizontal and a vertical axis. For example, the following four shapes are elegant diamonds:



These three shapes are diamonds, but are not elegant:



These three shapes are not diamonds:



The king will start by giving you a diamond, which may not be elegant. Your job is to make it elegant by enhancing it, adding digits on to make a bigger diamond. Because you don't want to spend too much money, you want to do it with as little *cost* as possible.

### **Definitions**

A **diamond of size *k*** is 2k-1 lines of digits, 0-9, separated by single spaces, organized in the following way:

* Line i (1 ≤ i ≤ k) contains k-i spaces, then i digits separated by single spaces.
* Line i (k < i < 2k) contains i-k spaces, then 2k-i digits separated by single spaces.

An **elegant diamond of size *k*** is a diamond of size k with the following two symmetry properties:

* Horizontal symmetry: Let ci be the number of digits on line i. The jth digit on line i (where j=1 for the first digit) must be the same as the ci+1-jth digit.
* Vertical symmetry: The jth digit on line i (where i=1 for the first line) must be the same as the jth digit on line 2k-i.

A diamond of size k can be **enhanced** by adding digits to it. The result of enhancing a diamond of size k has the following properties:

* The result is a diamond of size ≥ k.
* The original diamond is part of the result. In other words, there exist some X and some Y such that, for all values of i and j such that the jth character of the ith line of the original is a digit (as opposed to a space), the j+Xth character on the i+Yth line of the result is also a digit and it's the same as the jth character on the ith line of the original.

The **cost** of enhancing a diamond is equal to the number of digits in the result of the enhancement, minus the number of digits in the original diamond.

### **Input**

The first line of the input gives the number of test cases, **T**.  **T** test cases follow. Each test case consists of a single integer **k** in a line on its own, followed by a diamond of size **k**.

### **Output**

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is the minimum cost required to enhance the given diamond into an elegant diamond. If the diamond is already elegant, y=0.

### **Limits**

1 ≤ **T** ≤ 100.

#### **Small dataset**

1 ≤ **k** ≤ 10.

#### **Large dataset**

1 ≤ **k** ≤ 51.

### **Sample**



|  |  |
| --- | --- |
|  |  |

### **Explanation**

There are four cases. The first two cases start as elegant diamonds of size 1 and 2, respectively, and don't need to be enhanced; so the cost is 0. The third case can be enhanced to look like:



There are several possible enhancements, but this is one with the lowest possible cost, which is 5. In the fourth case we can enhance the diamond into the following elegant diamond:



...for a cost of 7.

# **Challenge 3: Maximal Sequences**

In this problem the user is to provide a series of positive and negative integer values. Your code will then search through the sequence which has NO MAXIMUM LENGTH, but minimally would consist of 1 entry and find the largest sub-sequence of values. If the subsequence sum appears more than once then all maximal subsequences should be listed.

*Runtime Samples:*

*>how many integer values in this sequence?*

*>8*

*>please enter the values now*

*>6 7 4 -4 2 -15 2 9*

*>maximal sequence: 6 7 4 at positions 0, 1, 2*

*>how many integer values in this sequence?*

*>10*

*>please enter the values now*

*>-1 0 5 0 5 -3 2 -11 10 -2*

*>maximal sequence 5 0 5 at positions 2, 3, 4*

*>maximal sequence 10 at position 8*

*>how many integer values in this sequence*

*>1*

*>please enter the values now*

*>-1*

*>maximal sequence is -1 at position 0*

*>how many integer values in this sequence*

*>0*

*>sorry not allowed*

*>how many integer values in this sequence*

*>5*

*>enter the values now*

*> 2 4 5 0 2*

*>maximal sequence is 2 4 5 0 2 at positions 0, 1, 2 , 3, 4*

# **Challenge 4: Tic-Tac-Toe Tomek**

Tic-Tac-Toe-Tomek is a game played on a 4 x 4 square board. The board starts empty, except that a single 'T' symbol may appear in one of the 16 squares. There are two players: X and O. They take turns to make moves, with X starting. In each move a player puts her symbol in one of the empty squares. Player X's symbol is 'X', and player O's symbol is 'O'.

After a player's move, if there is a row, column or a diagonal containing 4 of that player's symbols, or containing 3 of her symbols and the 'T' symbol, she wins and the game ends. Otherwise the game continues with the other player's move. If all of the fields are filled with symbols and nobody won, the game ends in a draw. See the sample input for examples of various winning positions.

Given a 4 x 4 board description containing 'X', 'O', 'T' and '.' characters (where '.' represents an empty square), describing the current state of a game, determine the status of the Tic-Tac-Toe-Tomek game going on. The statuses to choose from are:

* "X won" (the game is over, and X won)
* "O won" (the game is over, and O won)
* "Draw" (the game is over, and it ended in a draw)
* "Game has not completed" (the game is not over yet)

If there are empty cells, and the game is not over, you should output "Game has not completed", even if the outcome of the game is inevitable.

### **Input**

The first line of the input gives the number of test cases, **T**.  **T** test cases follow. Each test case consists of 4 lines with 4 characters each, with each character being 'X', 'O', '.' or 'T' (quotes for clarity only). Each test case is followed by an empty line.

### **Output**

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is one of the statuses given above. Make sure to get the statuses exactly right. When you run your code on the sample input, it should create the sample output exactly, including the "Case #1: ", the capital letter "O" rather than the number "0", and so on.

### **Limits**

The game board provided will represent a valid state that was reached through play of the game Tic-Tac-Toe-Tomek as described above.

#### **Small dataset**

1 ≤ **T** ≤ 10.

#### **Large dataset**

1 ≤ **T** ≤ 1000.

### **Sample**

|  |  |
| --- | --- |
| Input   | Output   |
| 6XXXT....OO......XOXTXXOOOXOXXXOOXOX.OX..........OOXXOXXXOX.TO..OXXXO..O..O..T...OXXXXO....O....O | Case #1: X wonCase #2: DrawCase #3: Game has not completedCase #4: O wonCase #5: O wonCase #6: O won |

# **Challenge 5: Dealer’s Difficulty**

It is possible for card experts to do a perfect shuffle. That is they divide the deck into 2 equal parts then fold them together.

You are to write a code that simulates this behavior with integers. Your code will accept from the user the values N (number of integers) and S (number of shuffles) after each shuffle print the result to the screen. For example, for user input of 7 and 3 the result would be. (n.b. please pay close attention to the placement of even and odd sized sub arrays,)

*Initial sequence:*

*1 2 3 4 5 6 7*

*Shuffle #1:*

* *1 2 3 and 4 5 6 7*

*4 1 5 2 6 3 7*

*Shuffle #2*

* *4 1 5 and 2 6 3 7*

*2 4 6 1 3 5 7*

*Shuffle #3*

* *2 4 6 and 1 3 5 7*

*1 2 3 4 5 6 7*

There are no constrains to the size of N or the number of shuffles S.

# **SOL Correlation**

C/T 9-12.11

Analyze, synthesize, and evaluate information based on source validity and the appropriateness to specific tasks.

A. Analyze and draw conclusions about the comprehensive nature and bias of electronic information sources.

• Follow best practice guidelines for analyzing information from particular Web sites.

• Evaluate information in the original context.

B. Evaluate the relevance of electronic information sources to a given situation.

• Determine appropriate types of information sources for various situations.

• Choose only relevant information when citing resources.

C. Use various digital tools to organize, analyze, and synthesize data for learning tasks.

• Use digital tools, such as graphic organizers, spreadsheets, and databases.

C/T 9-12.12

Practice reasoning skills when gathering and evaluating data.

A. Employ technology in developing strategies for solving problems.

• Regularly use technology tools to assist in authentic problem-solving activities.

• Investigate and apply expert systems and intelligent agents in real-world situations.

B. Select resources that extend one’s own capability to solve problems and make informed decisions.

• Choose resources that extend one’s own capabilities when solving problems.

C/T 9-12.14

Use models and simulations to understand complex systems and processes.

A. Use simulations to understand complex concepts.

• Enhance understanding of concepts and skills by using simulations.

B. Use various digital resources to produce graphical representations of data.

• Complete assignments involving data by using data graphing or imaging tools.

Have fun and thanks for participating in the Great Computer Challenge, 2018