**The Great Computer Challenge, 2019**

***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 Diving**

# Dan is just learning diving. He wants to impress his friends with his diving prowess, but first he must finish his classes and get his SCUBA certification. The final exam is a deep water dive, where he must descend to a certain depth, spend as much time as possible there, and then safely ascend to the surface, leaving no air in his tank when he is done. As you may know, if a diver ascends too quickly, he becomes afflicted with a condition known as the bends, where nitrogen gas accumulates at his joints. To avoid this, Dan must spend a certain amount of time decompressing at a lesser depth on his way up. Dan is lousy at math, so he needs your help.

# Given a target depth that Dan must dive, implement a diving alarm that tells him when to start his ascent and decompression. For each 10 feet below 30 feet that Dan dives, he must spend one minute in decompression (this is simplified from real world diving tables, but it should work for this problem). Also, it takes Dan one minute to safely ascend or descend 10 feet. Dan’s SCUBA school has supplied him with a 60-minute air tank. Given these constraints, compute how long Dan can stay at his target depth.

# **Input:**

# Input text file “Dives.txt” will begin with a single, positive integer, n, indicating the number of dives that Dan must complete. On the next n lines will be the target depths of each dive, one positive integer per line. All depths will be multiples of ten.

# **Output:**

# Print the amount of time (in minutes) that Dan can spend at his target depth and still safely make it back to the surface. If Dan can’t make it to the target depth, then he can spend no time there.

# Follow the format of the Sample Output, leaving one blank line after the output for each dive.

# **Sample Input:**

# 3

# 90

# 20

# 1000

# **Sample Output:**

# 36 minute(s) at 90 feet

# 56 minute(s) at 20 feet

# 0 minute(s) at 1000 feet

# **Challenge 2 Pascale’s Triangle**

Pascale’s Triangle is a triangular array of binomial coefficients. It is also known by other names, having been used by other mathematicians, many of whom predate Pascale. It can be constructed by having each cell in the triangular array be equal to the sum of the two cells above-left and above-right, with a unique non-zero number entered at the top of the triangle to begin the calculation.

 0 1

 1 1 1

 2 1 2 1

 3 1 3 3 1

 4 1 4 6 4 1

 5 1 5 9 9 5 1 ….. and so on

In the case of coin flips, you can easily calculate the distribution of different possible combinations of heads or tails using Pascale’s Triangle.

For 1 coin flip, the possible combinations would be H or T, corresponding to row 1 (1,1).

For 2 coin flips, the combinations are HH, HT TH, TT, corresponding to row 2 (1,2,1)

For 3 coin flips, you have HHH, HHT HTH THH, TTH THT HTT, TTT, as in row 3(1,3,3,1)

For three coin flips, you can also observe the phenomena where the distribution of H and T mirrors the two coefficients being added, 2+1=3, and the predicted combinations are made up of 2 Hs and one T, or vice versa.

Write a program to calculate all possible combinations of Heads or Tails given a number of coin flips.

**Input**:

User input of a single integer 1 to 30, representing the number of coin flips.

**Output**:

A list of possible combinations, grouped by coefficient, i.e 3 flips:

 HHH

 HHT HTH THH

 TTH THT HTT

 TTT

# **Challenge 3: The Last Word**

On the game show *The Last Word*, the host begins a round by showing the contestant a string **S** of uppercase English letters. The contestant has a whiteboard which is initially blank. The host will then present the contestant with the letters of **S**, one by one, in the order in which they appear in **S**. When the host presents the first letter, the contestant writes it on the whiteboard; this counts as the first *word* in the game (even though it is only one letter long). After that, each time the host presents a letter, the contestant must write it at the beginning or the end of the word on the whiteboard before the host moves on to the next letter (or to the end of the game, if there are no more letters).

For example, for **S** = CAB, after writing the word C on the whiteboard, the contestant could make one of the following four sets of choices:

* put the A before C to form AC, then put the B before AC to form BAC
* put the A before C to form AC, then put the B after AC to form ACB
* put the A after C to form CA, then put the B before CA to form BCA
* put the A after C to form CA, then put the B after CA to form CAB

The word is called the *last word* when the contestant finishes writing all of the letters from **S**, under the given rules. The contestant wins the game if their last word is the last of an alphabetically sorted list of all of the possible last words that could have been produced. For the example above, the winning last word is CAB (which happens to be the same as the original word). For a game with **S** = JAM, the winning last word is MJA.

You are the next contestant on this show, and the host has just showed you the string **S**. What's the winning last word that you should produce?

**Input**

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with a string **S**.

**Output**

For each test case, output one line containing Case #x: y, where x is the test case number (starting from 1) and y is the winning last word, as described in the statement.

**Sample**



# **Challenge 4: Rail Fence Cipher**

A Rail Fence Cipher is a type of transposition cipher. When given a message “I had the craziest dream last night” and a key representing the number of rails, the letters of the message are transposed onto the rails in a zig-zag pattern. Let’s assume a key of ‘3’:

 I . . . T . . . R . . . E . . . R . . . L . . . N . . . T

 . H . D . H . C . A . I . S . D . E . M . A . T . I . H .

 . . A . . . E . . . Z . . . T . . . A . . . S . . . G . .

As you can see, the message goes down and up the rails, excluding spaces. Once the message is on the rails, it is read off rail by rail to get the following encoded message.

ITRERLNTHDHCAISDEMATIHATEZTASG

In order to decrypt the message, notice the distribution of letters on each rail. For a key of 3, the cycle is actually 4 letters long, as it goes down the rails and then back up, with the cycle beginning again at the first rail. This would indicate that for a key , *length(cycle) = 2K-2*

Given the length of the encrypted message, you can calculate the number of letters on each rail, beginning with the first rail, and redistribute them. In this case, there are 29 letters. Divided by the cycle length of 4, you get 7 cycles, with 1 letter remaining. For each of the 7 cycles there is 1 letter on the first rail. The 1 remaining is the first letter in a new uncompleted cycle. This means that 8 letters fall on the first rail. For each cycle there are two letters on the second rail, and 1 letter on the third. This means there are 14 total letters on the second rail, and 7 total on the third for this message. Once the letters are separated onto their rails, the original message can be recovered by cycling over the rails again. The resulting decrypted message will not have spaces.

Write a program that can encrypt and decrypt a message given the appropriate key.

**Input**:

For the encryption process, an unencrypted message in a text file “message.txt” and a User inputted key 3-10 inclusive.

For the decryption process, input is the resulting file from the encryption process and a User inputted key 3-10 inclusive.

**Output**:

For the encryption process, the encrypted message is output to a text file “encrypted.txt”.

For the decryption process, the decrypted message is printed to the screen.

Use the example to verify your solution, but be sure to test on other short messages and keys.

# **Challenge 5: BFFs**

You are a teacher at the brand new Little Coders kindergarten. You have **N** kids in your class, and each one has a different student ID number from 1 through **N**. Every kid in your class has a single best friend forever (BFF), and you know who that BFF is for each kid. BFFs are not necessarily reciprocal -- that is, B being A's BFF does not imply that A is B's BFF.

Your lesson plan for tomorrow includes an activity in which the participants must sit in a circle. You want to make the activity as successful as possible by building the largest possible circle of kids such that each kid in the circle is sitting directly next to their BFF, either to the left or to the right. Any kids not in the circle will watch the activity without participating.

What is the greatest number of kids that can be in the circle?

**Input**

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of two lines. The first line of a test case contains a single integer **N**, the total number of kids in the class. The second line of a test case contains **N** integers **F1**, **F2**, ..., **FN**, where **Fi** is the student ID number of the BFF of the kid with student ID i.

**Output**

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the maximum number of kids in the group that can be arranged in a circle such that each kid in the circle is sitting next to his or her BFF.

**Sample**



In sample case #4, the largest possible circle seats the following kids in the following order: 7 9 3 10 4 1. (Any reflection or rotation of this circle would also work.) Note that the kid with student ID 1 is next to the kid with student ID 7, as required, because the list represents a circle.

***Have fun and thanks for participating in the
Great Computer Challenge, 2019!***

# **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.