The ACM North Central North America Regional Programming Contest

Sponsored by IBM

November 8, 2014

(Edited with corrections and clarifications November 21, 2014)

Max time is 10 seconds for A,B,D,E,F,G

Max time is 60 seconds for C,I

A. The Binary Search Efficiency Doubter

The binary search is a classic algorithm in computer science. For this problem we will use the following pseudocode to define our binary search:

```
Perform a binary search on the array a with values in a[0] through
a[n-1] to find, if it exists, the value x.  The values in the array a
may be assumed to be in strictly increasing order.

      Low = 0
      High = n - 1
      While Low <= High
            Mid = (Low + High) / 2 [We assume integer division
            truncates.]
            If a[Mid] = x then return FOUND
            If a[Mid] < x then Low = Mid + 1
            If a[Mid] > x then High = Mid - 1
      If we fall out of the while loop, return NOT_FOUND
```

Professors teach that this is an efficient algorithm with a worst case number of times through the loop of roughly log base 2 of n and an average case that is slightly better than that. A student who is not convinced decides to build lists of various sizes and search for every number in the list and keep track of how many times the loop is executed. In the following example, the number of times the loop is executed to find each value is indicated below the corresponding value.

the list	12	16	23	34	42	57	65
loop count	3	2	3	1	3	2	3

So, for this list, the total loop count is 17.

It should be clear that any list of length 7 will have a total loop count of 17 under the assumptions that the list is sorted and all the values are different. That is, the length of the list determines the total loop count.

The problem here is to determine the total loop count given the length of the list. You may assume that the answer for any test case in the input fits in a signed 64-bit integer.

Input:

There may be multiple cases. Input for each case will consist of a single positive integer, n, which gives the length of the list. You may assume that $2 < n < 10{,}000{,}000$ and that there are no more than 100 cases. Cases are delimited by arbitrary white space. Process until an end-of-file is detected.

Output:

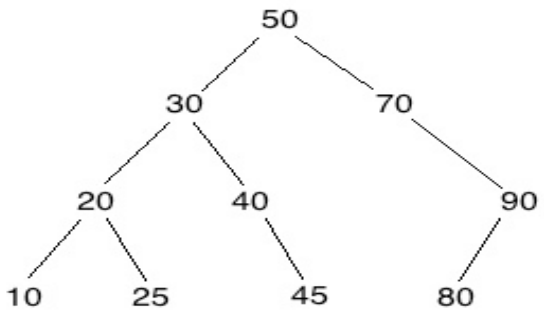
For each input case, print the total loop count to find all the values in a list of size n. Follow this format exactly: "`Case`", one space, the case number, a colon and one space, and the answer for that case with no trailing spaces. (see next page for sample)

Sample Input	Sample Output
3 7 321 124	Case 1: 5 Case 2: 17 Case 3: 2387 Case 4: 748

B. Preorder Traversals

A binary search tree is a common structure in computer science. A binary search tree is one in which all values in the left subtree of the root are smaller than the root value, all values in the right subtree of the root are bigger than the root value, and all subtrees in the tree are binary search trees. An empty tree is usually allowed as a binary tree, but we won't be dealing with empty trees in this problem. Also, any ordered data type can be allowed for contents of nodes, but we will just be interested in trees that contain positive integers that are less than 1,000,000,000 (one billion).

For example, the following is a binary search tree.



A preorder traversal is defined by the following recursive pseudocode:

```
preorder_traversal(root)
     print the value in root [in general, process the value, but
     we will just print it]
     if root has a left subtree
          preorder_traversal(left subtree of root)
     if root has a right subtree
          preorder_traversal(right subtree of root)
```

A preorder traversal of the above binary search tree would give 50, 30, 20, 10, 25, 40, 45, 70, 90, 80.

Note that 2, 3, 1 is not the preorder traversal of any binary search tree since 2 would have to be in the root as the first value printed and then either 3 would be on the left side or 1 would be on the right side since 1 comes after 3 in the preorder traversal.

The problem here is to read a list of numbers and determine if it is the preorder traversal of a binary search tree.

Input:

There may be multiple cases to process. Input for each case will consist of a list of positive integers followed by a negative integer that will signal end-of-list and should not be included in the list. Long lists may be given on more than one line. Do not assume a particular input format other than integers with whitespace separating them. You may assume the number of numbers in the list is a least 1 and at most 1,000. Process until the end-of-file is detected.

Output:

For each input case, print "`yes`" if the list is a valid preorder traversal of a binary search tree and "`no`" if not. Follow this format exactly: "`Case`", one space, the case number, a colon and one space, and the answer for that case with no trailing spaces.

Sample Input	Sample Output

`2 3 1 -7`
`50 30 20 10 25 40 45`
`70 90 80 -1`
	`Case 1: no`
	`Case 2: yes`

C. Rank Order

Your team has been retained by the director of a competition who supervises a panel of judges. The competition asks the judges to assign integer scores to competitors – the higher the score, the better. Although the event has standards for what score values mean, each judge is likely to interpret those standards differently. A score of 100, say, may mean different things to different judges.

The director's main objective is to determine which competitors should receive prizes for the top positions. Although absolute scores may differ from judge to judge, the director realizes that relative rankings provide the needed information – if two judges rank the same competitors first, second, third, ... then they agree on who should receive the prizes.

Your team is to write a program to assist the director by comparing the scores of pairs of judges. The program is to read two lists of integer scores in competitor order and determine the highest ranking place (first place being highest) at which the judges disagree.

Input:

Input to your program will be a series of score list pairs. Each pair begins with a single integer giving the number of competitors N, $1 < N < 1{,}000{,}000$. The next N integers are the scores from the first judge in competitor order. These are followed by the second judge's scores – N more integers, also in competitor order. Scores are in the range 0 to 100,000,000 inclusive. Judges are not allowed to give ties, so each judge's scores will be unique. Values are separated from each other by one or more spaces and/or newlines. The last score list pair is followed by the end-of-file indicator.

Output:

For each score pair, print a line with the integer representing the highest-ranking place at which the judges do not agree. If the judges agree on every place, print a line containing only the word 'agree'. Use the format below: "`Case`", one space, the case number, a colon and one space, and the answer for that case with no trailing spaces.

Sample Input	Sample Output
4 3 8 6 2 15 37 17 3 8 80 60 40 20 10 30 50 70 160 100 120 80 20 60 90 135	Case 1: agree Case 2: 3

D. Server

You are in charge of a server that needs to run some submitted tasks in a first-come, first-served basis. Each day, you can dedicate the server to run these tasks for at most T minutes. Given the time each task takes, you want to know how many of them will be finished today.

Consider the following example: Assume $T = 180$ and the tasks take $45, 30, 55, 20, 80,$ and 20 minutes (in order of submission). Then, only four tasks can be completed. The first four tasks can be completed because they take 150 minutes, but not the first five, because they take 230 minutes which is greater than 180. Notice that although after completion of the fourth task, there is enough time to perform the sixth task (which takes 20 minutes), you cannot do that because the fifth task is not done yet.

Input:

The input may consist of a multiple test cases. For each case, the first line contains two integers n and T where $1 \le n \le 50$ is the number of tasks and $1 \le T \le 500$. The next line contains n positive integers (each no more than 100) that indicate how long each task takes in order of when they are submitted. Process until an end-of-file is detected.

Output:

Display the number of tasks that can be completed in T minutes in a first-come, first-served basis. Follow this format exactly: "`Case`", one space, the case number, a colon and one space, and the answer for that case with no trailing spaces.

Sample Input	Sample Output

`6 180`
`45 30 55 20 80 20`
`10 60`
`20 7 10 8 10 27 2 3 10 5`
	`Case 1: 4`
	`Case 2: 5`

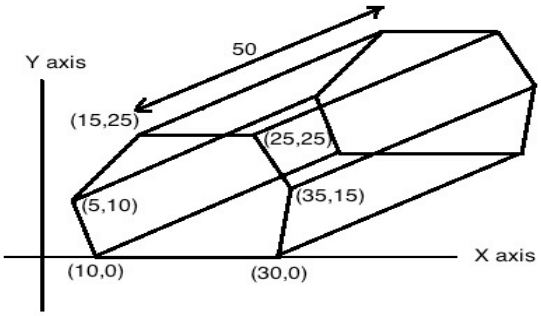
E. Aquarium Tank

You just bought an "artistic" aquarium tank that has an interesting shape, and you poured L liters of water into the tank. How high is the water in the tank?

When you look at this tank from one side, it has the shape of a convex polygon. This polygon is the cross section for any slice parallel to this side. This polygon has exactly two vertices on the table (y-coordinates are 0), and all other vertices have positive y-coordinates. There are also exactly two vertices with maximum y-coordinates, and water is poured into the opening between these two vertices. This aquarium tank has a breadth of B centimeters. The tank is glued to the table, so no matter what shape it has, it keeps its position and does not tip over.

All coordinates in this problem are given in centimeters. It should be noted that a cubic meter is equivalent to 1,000 liters.

An illustration showing the configuration of the tank of the first sample input is given below:



Input:

The input may consist of multiple test cases. The first line for each case contains an integer N ($4 \le N \le 100$) giving the number of vertices in the polygon. The next line contains two numbers B and L, where $1 \le B \le 1{,}000$ giving the breadth of the aquarium tank and $0 \le L \le 2{,}000$ the number of liters of water to pour into the tank. The next N lines each contain two integers, giving the (x, y) coordinates of the vertices of the convex polygon in counterclockwise order. The absolute values of x and y are at most 1,000. You may assume that the tank has a positive capacity, and of course you never pour more water than the tank can hold. Process until an end-of-file is detected.

Output:

Print the height of water (in centimeters) in the aquarium tank on a line rounded to 4 decimal places. Input will be constructed so that rounding will not cause problems for values that are sufficiently close to correct. Follow this format exactly: "`Case`", one space, the case number, a colon and one space, and the answer for that case with no trailing spaces.

Sample Input	Sample Output
6 50 28.175 25 25 15 25 5 10 10 0 30 0 35 10 9 30 70 110 70 100 80 80 80 -10 60 -40 30 -40 25 20 0 100 0 120 10	Case 1: 25.0000 Case 2: 19.7375

Note: The figure has a coordinate (35,15) which is different than the corresponding coordinate (35,10) in the first Sample Input. The Sample Output works with the Sample Input coordinate.

F. Restaurant Ratings

A famous travel web site has designed a new restaurant rating system. Each restaurant is rated by n ($1 \le n \le 15$) critics, each giving the restaurant a nonnegative numeric rating (higher score means better). The restaurants in each city are ranked as follows. First, sum up the ratings given by all the critics for a restaurant. A restaurant with a higher total sum is always better than one with a lower total sum. For restaurants with the same total sum, we rank them based on the ratings given by just critic 1. If there is still a tie, it is broken by comparing the ratings by critic 2, etc.

A restaurant owner received the ratings for his restaurant, and is curious about how it ranks in the city. He can easily find the sum of his own ratings, but he does not know the ratings or sums for all the other restaurants in the city. He decides to estimate his ranking by comparing his ratings to the number of possible unique sets of ratings that is no better than his own. You are asked to write a program to calculate this estimate (whether or not you think this will be very accurate).

Input:

The input may consist of a number of cases. Each case is specified on one line. On each line, the first integer is n, followed by n integers containing the ratings given by the n critics (in order). You may assume that the total sum of ratings for each restaurant is at most 30. The input is terminated by a line containing a 0 (zero). All restaurants in one city are assumed to be rated by the same number of critics, n.

Output:

For each input, print the number of different sets of ratings that is no better than the given set of ratings. You may assume that the output fits in a 64-bit signed integer. Follow this format exactly: "`Case`", one space, the case number, a colon and one space, and the answer for that case with no trailing spaces.

Sample Input	Sample Output

`1 3`
`2 4 3`
`5 4 3 2 1 4`
`0`
	`Case 1: 4`
	`Case 2: 33`
	`Case 3: 10810`

G. Locked Treasure

A group of n ($1 \le n \le 30$) bandits hid their stolen treasure in a room. The treasure needs to be locked away until there is a need to retrieve it. Since the bandits do not trust each other, they wanted to ensure that at least k ($1 \le k \le n$) of the bandits must agree in order to retrieve the treasure.

They have decided to place multiple locks on the door such that the door can be opened if and only if all the locks are opened. Each lock may have up to n keys, distributed to a subset of the bandits. A group of bandits can open a particular lock if and only if someone in the group has a key to that lock.

Given n and k, how many locks are needed such that if the keys to the locks are distributed to the bandits properly, then every group of bandits of size at least k can open all the locks, and no smaller group of bandits can open all the locks?

For example, if $n = 3$ and $k = 2$, only 3 locks are needed — keys to lock 1 can be given to bandits 1 and 2, keys to lock 2 can be given to bandits 1 and 3, and keys to lock 3 can be given to bandits 2 and 3. No single bandit can open all the locks, but any group of 2 bandits can open all the locks.

Input:

The first line of input contains a positive integer indicating the number of cases to follow. Each case is specified by the two integers n and k on one line.

Output:

For each line of input, print on one line the minimum number of locks needed. Follow this format exactly: "`Case`", one space, the case number, a colon and one space, and the answer for that case with no trailing spaces.

Sample Input	Sample Output
2 3 2 5 3	Case 1: 3 Case 2: 10

H. Continued Fractions

The (simple) continued fraction representation of a real number r is an expression obtained by an iterative process of representing r as the sum of its integer part and the reciprocal of another number, then writing this other number as the sum of its integer part and another reciprocal, and so on. In other words, a continued fraction representation of r is of the form

$$r = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \dots}}}$$

where $a_0, a_1, a_2, \ldots$ are integers and $a_1, a_2, \ldots > 0$. We call the a_i-values *partial quotients*. For example, in the continued fraction representation of 5.4, the partial quotients are $a_0 = 5$, $a_1 = 2$, $a_2 = 2$. This representation of a real number has several applications in theory and practice. If r is a rational number, the partial quotients are eventually all zero, so we only need to consider a finite number of partial quotients.

Given two rational numbers in continued fraction representation, your task is to perform the four elementary arithmetic operations on these numbers and display the result in continued fraction representation.

Input:

There may be multiple cases to consider. Process until an end-of-file is detected. Each test case consists of three lines. The first line contains two integers n_1 and n_2, where $1 \le n_i \le 9$ is the number of partial quotients of rational number r_i for $1 \le i \le 2$. The second line contains the partial quotients of r_1 and the third line contains the partial quotients of r_2. The absolute values of the quotients are not more than 10 and you may assume that $r_1 > r_2 > 0$.

Output:

Display the partial quotients of the continued fraction representations of $r_1 + r_2$, $r_1 - r_2$, $r_1 \times r_2$, and r_1 / r_2, in order, each in a line. Do not print any trailing zero partial quotients. Follow this format exactly: "`Case`", one space, the case number, a colon and no trailing space on the first line for a case, and the answers on separate lines, again with no trailing spaces.

Sample Input	Sample Output
4 3 5 1 1 2 5 2 2	Case 1: 11 0 5 30 4 6 1 27

I. Anagram Pyramids

Back in the 20th century, anagram puzzles were often found in the back pages of newspapers. Although they never reached the level of popularity of Sudoku puzzles, they were still a reliable means for killing a few minutes. One variant was the *anagram pyramid*, which consisted of a sequence of words stacked on top of each other. The word at the base of the pyramid had *N* letters, the second word had *N-1* letters, the third word had *N-2* letters, and so on. Each word (other than the word at the base) was formed by removing one letter from the word below and shuffling the remaining letters. Here is one example of an anagram pyramid:

```
PIN
SNIP
PAINS
PIANOS
```

Mr. Zino Ponzi, a retired financier, was reminiscing about the good old times one day when he got the idea of creating a few anagram puzzles to share with his friends at the Zigurat Retirement Home. Although he loved constructing anagram pyramids by hand, often he would get stuck, unable to think of a suitable word in the middle of the pyramid. Finally, he decided to hire a computer science student to write a program to make things easier. He didn't want to kill all the fun of doing it by hand, so he only wanted the program to tell him whether an anagram pyramid was possible for a given pair of top and bottom words and a dictionary.

Input:

There may be multiple cases to consider. Process until an end-of-file marker is detected. Words in this problem will be strings of from one to thirty letters. Treat upper and lower case versions of the same letter as equal. The input for each case will consist of a dictionary of *N* words ($N < 1{,}000{,}000$) to be used to form pyramids, followed by *M* ($M < 100$) pairs of top/bottom words from the dictionary with the top word shorter than the bottom in the following format (Strict pairing of words per line in second part is NOT to be implied by this sample):

```
N
word1
…
wordN
M
top1 bottom1
…
topM bottomM
```

Output:
The answers for each case should consist of M lines, one for each pair of top/bottom words, saying either "`yes`" or "`no`" depending on whether an anagram pyramid was possible given the input dictionary. Follow this format exactly: "`Case`", one space, the case number, a colon and no trailing space on the first line for a case, and the answers on separate lines, again with no trailing spaces.

(see example on next page)

Sample Input **Sample Output**

8 PEN PIN SNIP PINE PAINS SPAIN PIANOS SNIPER 2 PIN PIANOS PEN SNIPER	Case 1: yes no