2nd Annual

Brookfield Computer Programming Challenge

2018

Problem Set

Problems:

1. Everybody Do the Flop (Flop)
2. Funny Pairs (Funny)
3. Goat Simulator (Goat)*
4. Codeforces Handle (Handle)
5. Tax Evasion (Tax)
6. Two (Two)*

*All problems have a 5 second time limit. Inefficient solutions on these problems will result in a Time Limit Exceeded verdict. Modern computers can do about 10^8 commands per second in Java. [Python is slower, but all problems are solvable in Python.]

**If you use packages or multiple Scanner objects, you will get all the problems wrong. Use the default package and a single Scanner object for all input.

Everybody Do the Flop (Flop)

The Do-the-Flop Guy was a viral sensation; everybody did the flop across the whole entire nation! But then one day, things started going wrong...

The Do-the-Flop craze caused so many head injuries that the British Cranial Protection Center has issued a statement asking Do-the-Flop Guy to replace his backup-dancers with robots! In an effort to minimize the concussional impact of his dance moves, Do-the-Flop Guy is doing his best to comply with the BCPC's regulations. However, with flopping as dangerous as it is these days, these robots will break permanently upon flopping. Therefore, Do-the-Flop Guy needs your help to program them to listen for the lyrics "everybody to the flop" before flopping.

Input:

- The input will contain several lines representing the lyrics to Do-the-Flop Guy's next song. The last line will be "everybody do the flop". You are guaranteed that no lines except the last will be "everybody do the flop". All lines will contain only lowercase characters, spaces, and apostrophes. Lines will not start or end with whitespace or be empty.

Output:

- For each line of song lyrics that is not exactly "everybody do the flop" output "Wait for it...". At the end of the song, output "FLOP!".

Example:

Input	Output
everybody do the stand up straight everybody do the don't fall down everybody do the flop	Wait for it... Wait for it... FLOP!
everybody do the flop tomorrow everybody do the flop	Wait for it... FLOP!

Funny Pairs (Funny)

Mr. Haha is an honors Precalculus teacher who loves funny pairs. And pears. Especially the funny ones. Mr. Haha considers a pair of numbers (x,y) funny if and only if x*x = y+y. And while Mr. Haha knows of quite a few funny pears, he can't ever seem to recall any funny bananas.

Mr. Haha is pealing particularly funny today and would like to know if there exists a corresponding integer y for a given x that would make the pair (x,y) funny. And unfortunately, as Mr. Haha is busy babysitting his tangerines (usually he gets the older fruit to babysit, but today the banana split), he needs your help. In fact, he finds funny pairs so ap-*peal*-ing that he wants you to find him a whole -*bunch*-. Just be careful not to -*slip*- up and start telling knock-knock jokes in the middle of your output![1]

Input:

- The first line will contain a single integer **n**, the number of test cases
- N space-separated integers follow, each containing a single integer $\mathbf{x_i}$, the first number of the ith pair.

$1 \leq \mathbf{n} \leq 10^4$
$0 \leq \mathbf{x_i} \leq 10^4$

Output:

- Output n lines each containing either:
  - a single integer $\mathbf{y_i}$, the matching number to $\mathbf{x_i}$ to make $[\mathbf{x_i}, \mathbf{y_i}]$ a funny pair
  - Or the sentence "Can he do it? Cosecant!" (without quotes) if there exists no integer $\mathbf{y_i}$ for which the pair would be funny.

Example:

Input	Output
3 4 5 0	8 Can he do it? Cosecant! 0

The pairs (4, 8) and (0, 0) are both funny. There exists no pair (5, y) that is funny.

[1] Knock Knock. ("Who's there," you ask?) Orange. ("Orange who?") Orange you glad this wasn't another banana pun? (Get it? That was the -fruit punch-line!)

Goat Simulator [Goat]

Some elementary school teachers tell their students that they can be anything they want when they grow up if they work hard enough. Clearly these teachers have not had Gary. Gary wants to be a goat.

Gary works so hard at learning to grow up to be a good goat that he has purchased a Goat Simulator video game. This game is played in a linear world (represented by the x-axis) and contains **n** obstacles the player may need to traverse in order to get from their spawn point to the food (the goal). When the game starts, players choose the location at which their player and the food spawn. Gary needs your help to determine the number of obstacles he would have to traverse if he chose two given positions. Gary tried writing a program to iterate through all obstacles for every one of his queries, but this was much too slow for large inputs, so he needs you to write a more efficient algorithm to do this.[2]

Input:

- The first line will contain a single integer **n**, the number of obstacles
- The second line will contain **n** unique integers $\mathbf{p_i}$, each representing the position of an obstacle, not necessarily in ascending order.
- The third line will contain an integer **Q**, the number of queries.
- The following **Q** lines will contain two integers $\mathbf{s_i}$ and $\mathbf{f_i}$, representing the spawn location and the food location for the ith query. These points are guaranteed to not contain any obstacles, and $\mathbf{s_i}$ will be to the left of $\mathbf{f_i}$.

$0 \leq \mathbf{n}, \mathbf{p_i}, \mathbf{Q}, \mathbf{s_i}, \mathbf{f_i} \leq 10^6$

Output:

- Output a line containing a single integer: the sum of the number of obstacles crossed across all queries.

Example:

Input	Output
4 0 10 4 5 3 1 6 2 12 11 1000000	5

There are 2 obstacles from 1 to 6, 3 from 2 to 12, and 0 from 11 to 1000000.

[2] Note that there is a strictly-enforced 5 second time limit. Your program must terminate and give a correct answer even with 10^6 obstacles and 10^6 queries.

Codeforces Handle (Handle)

Peter wants to create a Codeforces account and needs your help coming up with a handle-a name by which he will be referenced on the site. Peter wants his handle to reflect his name as accurately as possible, but knows that since all handles on the site have to be unique and must contain only lowercase letters, he may not get his first choice. He decides that instead of trying things like 'xxxpeterxxx' or 'legendarygrandmasterpeter', he will try versions of his name with combinations of vowels removed.

After not getting 'peter', 'ptr', or 'pter', Peter lucked out with the handle of 'petr', which certainly wasn't a bad consolation prize. Because he likes to give back to the community, Peter wants you to make a program to tell people how many unique versions of a name exist which have some combination of vowels removed. To make things easier, Peter will only run your program on names that do not contain two adjacent vowels that are different (so he will not run your program on 'Joan', but might on 'David', for example).

Note: y is not considered a vowel for the purposes of this problem.

Input:

- The first line will contain an integer **n**, the number of testcases.
- **n** lines follow, each containing a single string $\mathbf{s}_i$, the name of the $\mathbf{i}^{th}$ person converted to all lowercase letters.

$1 \leq \mathbf{n} \leq 50$
$1 \leq$ length of $\mathbf{s}_i \leq 100$

Output:

- Output **n** lines, each containing a single integer representing the number of unique handles for the respective name that exist following the above rule.

Example:

Input	Output
4 peter john steeve elizabeth	4 2 6 16

The only possible handles for john are 'john' and 'jhn'.

Tax Evasion (Tax)

Bill is a good, law-abiding citizen, entirely unwilling to steal from a grocery store. However, he also happens to be an incredibly thrifty shopper. Because he is running short on change and is itching for the adrenaline rush of being criminally neglecting to pay the full amount of sales taxes, Bill plans to buy his **n** groceries in such a way as to pay the lowest possible price.

Bill's state has a sales tax rate of 33.333...%, meaning that to check out with a total price of **P**, Bill has to pay 1.333...***P** because of his coercive, overspending, always-up-to-no-good state government. However, because the smallest currency available is the penny, this amount is *rounded up or down* to the nearest cent (just like in Wisconsin). Bill is okay with checking out several times as long as he can scam his state out of some of their undue taxation.

Also, Bill ironically only pays for his groceries in pennies and thought it would be easiest if he gave you all of the prices as integers. Can you help him determine the smallest amount he has to pay to buy all of the groceries he wants?

Input:

- The first line will contain an integer **n**, the number of groceries bill has.
- The second line contains **n** space-separated integers $\mathbf{p}_i$, the price of the i^{th} good in pennies.

$\mathbf{1} \leq \mathbf{n} \leq 10^5$

$1 \leq \mathbf{p}_i \leq 10^5$

Output:

- Output a single integer representing the number of pennies Bill will have to pay if he buys his groceries optimally.

Example:

Input	Output
4 5 9 5 10	38
5 1 1 1 1 1	5

In the first example, Bill could buy his products in the sets of (5 5) (9 10) and pay only 9 pennies in tax. Note that if he bought all of his items together (or separately), he would have to pay 10 pennies in tax. In the second example, Bill could avoid paying sales tax altogether by buying each item separately.

Two (Two)

Hooray! It's time for the obligatory problem concerning the random mathematical properties of an arbitrary number! And since today is the 2nd Brookfield Computer Programming Challenge, this year's number is 2.

Two is a pretty cool number. It's a fibonacci number, the only number whose square equals the sum of it and itself, a Catalan number, and its own factorial. Perhaps most interestingly, though, is that it is the only number that is both prime and anti-prime. Of course, since we were interested in primes last year, this year we want to know all about anti-primes!

An anti-prime (or highly composite number) is a natural number that has more unique factors than any natural number less than it. Some anti-primes include 1 (1 factor), 2 (2 factors), 4 (3 factors), 6 (4 factors), and 12 (6 factors). Given an natural number **i**, find the smallest anti-prime greater than or equal to **i**.

Input:

- The first line will contain an integer **n**, the number of test cases
- **n** lines follow, each containing a single integer, **i**.

$1 \leq \mathbf{n} \leq 10^4$
$1 \leq \mathbf{i} \leq 10^7$

Output:

- For each test case, output a single integer: the smallest anti-prime greater than or equal to **i**.[3]

Input	Output
4 1 12 13 10000000	1 12 24 10810800

[3] Note: There is a strictly enforced 5-second runtime limit. Be sure to test your program on cases with lots of large inputs to make sure it is fast enough.