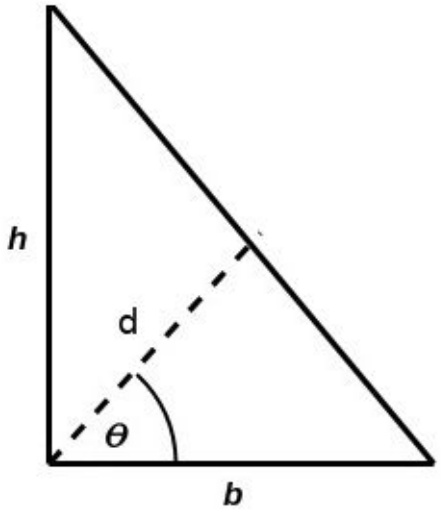
EDITORIAL of CODESMASH '18

RUET ANALYTICAL PROGRAMMING LAB

CSE, RUET

1) Hasinur Has two:
Setter: MD. MOSHIUR RAHMAN



Let, ***d*** be the length of the side that divide the right triangle. Then, the **area** of the triangles are, **½ * b * d * sin(*θ*)** and **½ * h * d * sin(90 - *θ*)**, where **d** is unknown.
The total area is **½ * b * h**.
So,
½ * b * d * sin(*θ*) + ½ * h * d * sin(90 - *θ*) = ½ * b * h.
By solving this equation, we can find **d**, hence area of the both triangle.

N.B: The trigonometric functions take parameter in radian.

2) Painful Giveaway:
Setter: Risal Shahriar Shefin, Shahwat Hasnaine

N can be **negative** too. Print **0** to **N** for **non-negative N**. Print **N** to **0** for **negative N**.
Don't print extra **space** after the last integer.

3) How Hasinur Met His Girlfriends
Setter: Nurul Islam Nobel

Constraints are flexible. So we can do bruteforce for this problem. We can generate all the substrings with a $\mathbf{n^2}$ loop, where **n** is the size of the string.

Every-time we generate a substring we insert it into a set. **Set** is a data-structure where every element comes only once and they are sorted. After inserting all the substrings, only **unique substrings** will remain in the set. So printing the **size of the set** is the answer for this problem.

4) Fabbyfied Triangle:
Setter: MD. HASINUR RAHMAN

It's clear that this problem could be solved easily if the value of **n** was small. You could run $\mathbf{n^2}$ loop and get accepted. But **n** could be $\mathbf{10^9}$. You can not get accepted doesn't mean you can not run a $\mathbf{n^2}$ bruteforce solution in your pc to observe things.

Run that bruteforce solution. You will get a **pattern**. It will help you to get a formula.

5) Hasinur Is Giving Treat
Setter: Tahsin Masrur, Nasif Mahbub

If the memory limit was larger, could you solve it with **cumulative sum** or **prefix sum**? It'd be easy then. But the memory limit is tighter, so what can we do now? Now, we can store the **prefix sum partially**, not fully. We can save the prefix sum after every **10 steps**. And for a query, we can iterate those extra **10 steps**. This is similar to *SQRT decomposition* where you save after every SQRT steps. This was judge solution approach.

Another approach was even easier. You can save all queries first for offline processing. For each query **[L, R]**, save **L - 1** and **R** in a set. Then, while iterating for prefix sum, save the answers of those positions in a **map**. Answering the queries become easy then.

Another trick in this problem was the modulo. Due to the modulo being **very large**, **normal multiplication can overflow**. To solve that, you can multiply in ***log(n)*** in the same way you do *modular exponentiation (Big Mod)*.

6) Secret Of Love
Setter: Nasif Mahbub, Pritom K. Paul

It is a simple ad-hoc problem. Two nested loops are required for both the input and output. But for the required output, the outer loop runs on columns instead of rows. Also, two conditions are required for the inner-loop.
For **odd** columns, start the inner-loop from the first row towards the last row and
For **even** column start the inner-loop from the last row towards the first row.

Time and Memory Complexity ***O(MxN)***.

7) Hasinur The Great
Setter: Tauhedul Islam Tanu

This is a modified version of **Sieve of Eratosthenes** (https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes).

You can precalculate answers for each **n**. In sieve, if we find a prime we mark all number those are divisible by the prime (except the prime) as composite. But in this modified version, if we find a prime we need to add +1 to all number that are divisible(including the prime) by that prime.

In original version of sieve, we iterate and check if a number is either prime or not upto ***sqrt(n)*** , because we know that a composite number **n**, must have a divisor which is less than or equal ***sqrt(n)*** . But in this problem, we have to iterate upto **n**. Because, we need every prime that will contribute to answer.

For example, if n = 10, and if we iterate to ***sqrt(n)***, i.e. **3**, the primes **5** and **7** won't contribute to answer, which is wrong. This is why we need to iterate upto n.

Complexity: Time complexity is ***O(n log(n))*** and space complexity is ***O(n)*** .

8) Pera of Prime
Setter: Pritom k. Paul

This is solvable by **sieve** and **cumulative sum**. Let, in this problem, **cumSum[i]** will store the count of prime from **1 to i (including i)**. If we are asked to find answer for a given range **[L, R]**, then the
`numerator = cumSum[R] - cumSum[L - 1] and`
`denominator = R - L + 1`
If numerator and denominator equals, then print **1**.
Else If numerator is **0**, then print **0**.
Else, numerator and denominator can be reduced. By dividing both with their **gcd**, i.e. ***gcd(numerator, denominator)***, we can ensure the fact that numerator and denominator are co-prime.

Complexity: Time complexity is ***O(n log(n))*** and space complexity is ***O(n)***.

9) Ajob toh!!
Setter: Dhruba Mitra

This is a problem of implementation. **Bruteforce** solution will pass easily.

10) Read The Problem Statement Carefully
Setter: Pritom k. Paul

Printf Problem.