**Brief Explanation:**

Expected solution is O(MAX\_N \* log(log MAX\_A)) using Sieve of Eratosthenes. Though a bit optimised brute force of O(MAX\_N \* root(MAX\_A)) will also pass.(Due to some limitations of OJ).

If you have used O(MAX\_N \* root(MAX\_A)) then we request you to once implement using Sieve of Eratosthenes or at least just go through optimal solution provided by us.

If you have used Sieve of Eratosthenes then also we request you to check solution provided by us. It is more optimized implementation than most of the others you would have seen. (Go through comments and will get it.)

**Detailed Explanation:**

O(MAX\_N \* MAX\_A) is trivial. For each element in a we iterate over 2 to a[i] - 1 and see if any of them divides a[i] or not.

Now let’s talk about O(MAX\_N \* root(MAX\_A)). If any positive number x is not a prime(and not 1) then it can be written as x = a \* b where a > 1 and b > 1. Now we should note that both of them can not be > root(x) because if a > root(x) and b > root(x) then a \* b > root(x) \* root(x) hence a \* b > x. Also when number if a square like 16 then it can be written as root(16) \* root(16). So, if a number is not a prime then for any representation of x = a \* b (like 30 = 6 \* 5) or x = a \* b \* c (like 30 = 2 \* 3 \* 5) etc. At least one factor will be <= root(x). Now we can say that we need not to check over 2 to a[i] - 1 but iterating over 2 to root(a[i]) will be enough.

Now still we can improve. We can write any composite number x as multiplication of prime numbers. Like 60 = 2 \* 2 \* 3 \* 5 = 2 \* 30. So what we can do is that when we find any prime number we visit all the multiple of it and mark them as non prime. We start from 2 as base case and mark 4, 6, 8, 10, … as visited means they are multiples of 2 hence they are composite. Now 3 comes and 3 is not visited hence we do same thing mark 6, 9, 12, 15 ...as visited means they are multiple of 3 hence they are composite. Now when 4 comes we can do the same thing mark 4, 8, 12, 16… as visited but compare these positions with visited positions by 2! They are same because 4 is a multiple of 2 so any multiple of 4 has already been visited by 2. So we need not to visit multiple of 4! Now follow same method and try to find primes till 30 and you will get more clear idea.

Now still there can be some optimizations. When we find some prime number like 11 then we visit 22, 33, 44, 55… but see that 22, 33, 44, 55 are already visited because 55 = 5 \*11 and 5 < 11. So instead of x + x we can start visiting from x \* x! Now this solution will be O(MAX\_N \* log(log MAX\_A)).

**Now should we say that O(MAX\_N \* log(log MAX\_A)) is better than O(MAX\_N \* root(MAX\_A))? NO! It depends on situation. In terms of time complexity of course it is better, but we should also consider space complexity! Solution with time complexity O(MAX\_N \* root(MAX\_A)) requires O(1) extra space but other needs O(MAX\_A) extra space. So when ever space is more imp than time then we should opt the slower one!**

Now when we are given a single integer and we are told to find if it is a prime or not then should we use Sieve of Eratosthenes? No!! If there is only one number to check then we should use O(root(MAX\_A)) where we iterate over 2 to root(MAX\_A) and check for divisibility. We use Sieve of Eratosthenes only when there are multiple numbers! Think!

Now have a look at the code provided by us.