Code Jam 2011 - Round 2

Analysis: Expensive Dinner

This problem might look pretty intimidating at first:

- There are **N**! possible orders in which your friends could arrive.
- Not only is O(**N**!) too slow for this problem, even O(**N**) is too slow!
- The actual price your friends are paying will quickly exceed the bounds of a 64-bit integer.

When O(**N**) is too slow, it means you need to forget about coding for a while and think. You will need some insight to even get started.

Let's begin by fixing an ordering (x_1, x_2, ..., $x_{\mathbf{N}}$) of your friends. Also let y_i be the total price that your group is paying after the first i friends enter, the waiter has been called if necessary, and everyone has become happy.

Observation 1: $y_i = \text{LCM}(x_1, x_2, ..., x_i)$. This does not depend on the order your friends buy things after the waiter has been called.

Explanation: $\text{LCM}(x_1, x_2, ..., x_i)$ is, by definition, the smallest multiple of x_1, x_2, ..., x_i. Since y_i must be a multiple of all these numbers for your friends to be happy, it is certainly true that $y_i \geq \text{LCM}(x_1, x_2, ..., x_i)$. Conversely, a friend x_j would never buy something that skips over a multiple of x_j. In particular, none of the friends here would buy something that would skip over $\text{LCM}(x_1, x_2, ..., x_i)$. Since $y_{i-1} \leq \text{LCM}(x_1, x_2, ..., x_i)$, you will eventually reach this price and then stop.

Let **M** be the number of times the waiter is called over. Then, **M** is equal to the number of values i in {1, 2, ..., **N**} such that $y_i \mathrel{!}= y_{i-1}$. (We define $y_0 = 0$, because the waiter is always called over by the first friend who enters.) So the question becomes: how do we make **M** as big as possible according to this definition, and how do we make it as small as possible?

Least common multiples are closely related to prime numbers and factorizations, so let's define p_1, p_2, ..., $p_{\mathbf{P}}$ to be the primes less than or equal to **N**. Also let e_i be the largest integer such that p_i^{ei} is less than or equal to **N**.

Observation 2: Let **P'** be 1 + sum e_i (i.e., the number of prime powers less than or equal to **N**, including 1). Then the maximum value of **M** is exactly equal to **P'**.

Explanation: Suppose your friends arrive in the order 1, 2, ..., **N**. Then each friend with a prime power index will cause the price to increase, and so the maximum value of **M** is at least **P'**.

On the other hand, the price is always a least-common multiple by the first observation, and it always increases to a multiple of itself. In particular, the sum of the exponents in its prime factorization must always go up every time the waiter is called. After the first friend arrives, this sum is at least 0. At the very end, it is equal to **P'** - 1. Therefore, the waiter can be called at most **P'** - 1 times after the first person arrives. Combining that with the initial increase proves that **M** ≤ **P'**.

Observation 3: If **N** > 1, then the minimum value of **M** is equal to **P**.

Explanation: Suppose the first friends to arrive are numbered p_1^{e1}, p_2^{e2}, ..., $p_{\mathbf{P}}^{e\mathbf{P}}$. Then the price is already equal to LCM(p_1^{e1}, p_2^{e2}, ..., $p_{\mathbf{P}}^{e\mathbf{P}}$) = LCM(1, 2, ..., **N**) after these friends have arrived. This means no subsequent friend will change the total price, and hence **M** = **P** in this case.

On the other hand, notice there is no number less than **N** that is divisible by both $\mathbf{p}_i^{ei}$ and $\mathbf{p}_j^{ej}$. This is because $\mathbf{p}_i^{ei}$ and $\mathbf{p}_i^{ej}$ are relatively prime and larger than sqrt(**N**). (If one of them was at most sqrt(**N**), then its exponent could be increased, which is a contradiction.) Therefore, no single friend can make the total price divisible by two different entries out of {p_1^{e1}, p_2^{e2}, ..., $p_{\mathbf{P}}^{e\mathbf{P}}$}. And so the waiter must be called at least **P** times, as claimed.

To solve the small input, you can just calculate **P** and **P'** directly and go from there. The large input requires one last clever trick. The number of primes less than 10^{12} is quite large, and you probably cannot afford to just count them all. However, calculating **P** - **P'** is actually easier than this!

If you go back to the original definitions, you can see **P** - **P'** is precisely the number of integers less than or equal to **N** that can be written in the form p^e for e != 1. When e = 0, you just get 1. When e > 1, then p ≤ 10^6, and you can easily enumerate all primes of this size! One good way is to use the Sieve of Eratosthenes. As always, you can look at solutions from other contestants to see a full implementation.

Bonus Comment: The Sieve of Eratosthenes runs in O(**M** * log log **M**) time, so the simplest implementation of the method described here runs in O(sqrt(**N**) * **T** * log log **N**) time altogether. However, it's worth pointing out that you can do even better. If you pre-compute and sort all prime powers less than 10^{12} with exponent other than one, you can then use a binary search to very efficiently count how many are less than **N** for each test case. This is not necessary to solve the problem, but it lets you speed things up even more to a blazing O(sqrt(**N**) * log log **N** + **T** * log **N**). Proving this running time is a little tricky though!